THE UNIVERSITY OF AUCKLAND
LIBRARIES AND LEARNING SERVICES

http://researchspace.auckland.ac.nz

ResearchSpace@Auckland

Copyright Statement

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from their thesis.

To request permissions please use the Feedback form on our webpage.
http://researchspace.auckland.ac.nz/feedback

General copyright and disclaimer

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the Library Thesis Consent Form and Deposit Licence.

Presentation Methods for Open Data

Jimmy Oh

A thesis submitted in fulfilment of the requirements for the degree of PhD in Statistics, the University of Auckland, 2015.

Abstract

This thesis examines ways to make Open Data more open, more accessible and thus more useful to a wider audience. It begins with a Literature Review of Open Data in New Zealand to understand what data is out there and to identify what barriers currently exist that hinder the usage of the data. To address a major problem identified in the Literature Review, it then presents TableToLongForm, an R package that automatically converts hierarchical tables intended for a human reader into a simple longform dataframe that is machine readable, making it easier to access and use the data for analysis. Once data is in a useful format, one of the most effective ways to communicate it to a wider audience is through visualisations, so the thesis continues with a Literature Review of Graphical Tools examining what is currently available for creating web-based interactive graphics. It then presents WeBIPP a web-based interactive tool for building statistical graphics from scratch without writing any code. The graphical user interface can be used to build graphics from scratch, but as the user does this, they are not simply creating a graphic, they are writing code in the background. For those who have no coding knowledge, this fact can be ignored, making WeBIPP accessible to this audience. But for those who have coding knowledge, they can utilise it to examine the code, to tweak it and to reuse it. This makes WeBIPP much more powerful than other tools of similar nature. The thesis concludes with summarising remarks and possibilities for future work.

Acknowledgments

I want to begin by thanking the people from the many State Sector Organisations covered in the Literature Review of Open Data in New Zealand, without their help the Literature Review would be far less informative. Thanks to Richard Penny and Peter Crosland from Statistics New Zealand, who felt the perspective of a PhD student was valuable to Statistics New Zealand and Official Statistics; the opportunity to present my work to more people led to interesting discussions that informed my work. Thanks to Keith Ng, who pointed out other interesting data visualisations tools in development and gave helpful comments on WeBIPP's interface. And of course I must thank my supervisor and co-supervisor, Paul Murrell and Ross Ihaka, their wealth of knowledge supported everything I did, and guided me not only to the completion of this thesis, but also my growth as a programmer and an academic.

Contents

Contents ii

List of Figures iv

List of Tables xii

1 Introduction 1

2 Open Data in New Zealand 5

2.1 Introduction . . . 6
2.2 What is Open Data? . . . 7
2.3 Overview of Open Data in New Zealand . . . 19
2.4 Conclusion . . . 40

3 TableToLongForm 41

3.1 Introduction . . . 42
3.2 How to Use . . . 48
3.3 Vocabulary . . . 51
3.4 Implementation Details . . . 53
3.5 Identification . . . 53
3.6 Discern Parentage . . . 60
3.7 Reconstruction . . . 68
3.8 Summary . . . 70

4 Graphical Tools 73

4.1 Introduction . . . 74
4.2 GUI Tools . . . 76
4.3 High-level Languages . . . 80
4.4 Low-level Languages . . . 85
4.5 Other Relevant Topics . . . 95
4.6 Conclusion . . . 101

5 WeBIPP **103**
5.1 Introduction . . . 105
5.2 What is WeBIPP . . . 108
5.3 How to Use . . . 113
5.4 Creating a Scatterplot . . . 115
5.5 Creating a Population Pyramid . . . 125
5.6 How WeBIPP Works . . . 161
5.7 Creating an Object Addon . . . 168
5.8 Creating a Value Interface . . . 183
5.9 Core's Set Attribute . . . 188
5.10 Discussion and Limitations . . . 194
5.11 Conclusion and Future Work . . . 197

6 Conclusion **199**

Bibliography **201**

List of Figures

2.1 An example of an XLS-Longform. The data is *active-customers-by-entity-type-2001-to-2011.xls* from *Inland Revenue*. 12

2.2 An example of an XLS-Table. While this may be quite easy for the human brain to decipher, it is a nightmare for a computer. Consider for instance trying to obtain all the information for the first 'piece' of data. First we need to get the Rank from B8, understand this number is a 'Rank' by getting the column label from A5 (noting this is a different column to where the actual Rank value is stored). Second we need the name from D8, and the label for this from C6. Third we need to get the count of babies from E8 with label from E6. Finally we need the year from C5. The data is *Top100BabyNamesNewZealand2011.xls* from the *Department of Internal Affairs*. 12

2.3 An example of an XLS-Table. Once again, this may be relatively easy for a human brain to decipher, but a nightmare for a computer. Note that in this instance in-cell indentation and the Excel Grouping feature are used to communicate the hierarchical relationships in the data rows. Such information will typically be lost when converting the XLS file to another format, effectively forcing the user to a) Use Excel and b) Conduct any analysis manually. As Excel itself understands some of this information, it may be possible to write code in VBA (*Visual Basic for Applications*, a slightly modified version of *Visual Basic* integrated with Microsoft Office applications) to extract this information, but a cursory look at the code suggests this might be quite a challenge. The data is *Oil.xls* from the *Ministry of Economic Development* (now a part of the *Ministry of Business, Innovation & Employment*). 13

2.4 An example of an XLS-PivotTable. The Pivot Table is an interactive feature of Excel that allows dynamic creation and modification of tables via a graphical user interface. It greatly enhances data exploration that can be done within Excel, though at first glance appears to be a significant barrier to analysis outside Excel. However, this is not the case, as it is possible to extract the underlying data using *Show Details* from the right-click menu. The data is *Pivot-Table-Student-Numbers-by-Age.xls* from the *Ministry of Education*. 14

2.5 An example of the XLS-PivotTable from Figure 2.4 after clicking *Show Details* from the right-click menu. Note that the data is presented in long-form and could now be converted to a CSV for analysis in different software. 14

2.6 An example of an XLS-ReportTable. Much like an XLS-Table, these are relatively easy for a human brain to decipher. However, we classify an XLS file as ReportTable if there are substantial barriers to machine reading. Generally any data released as an XLS-ReportTable can only be used manually and presents significant barriers to any re-use, even by manual means. The data is *jtei-july-08.xls* from the *Department of Labour*. 15

2.7 An example of an HTML-Table. The data is *Income bands for salaries and wages, 2002 to 2011* found under the *Wage/salary distributions for individual customers* category from *Inland Revenue*. 16

2.8 An example of an HTML-Table that does not use HTML Table code (instead utilising `div` and `span` for visual formatting). This means the data is not structured making it extremely difficult to extract the data. Luckily in this case, a link is provided (not shown) to download the data as an XLS-Table. The data is a *Results by Subject and Standard* report from the *New Zealand Qualifications Authority*. 17

3.1 An example of a hierarchical Table. The Table is of the Labour Force Status data (Statistics New Zealand, 2013) and in total spans 240 columns. The Table is too large to be immediately useful for humans, and yet cannot be manipulated easily with a computer. 44

3.2 An example of a LongForm dataframe. This is the Table in Figure 3.1 after automatic conversion with TableToLongForm and in total spans 660 rows. While it is still not immediately useful for humans, all related information can be found in the same row or column, making the data much easier to manipulate with a computer. 45

3.3 Row Labels excerpted from NZQA Scholarships data displaying both Empty Right (red) and Empty Below (green) patterns. 47

3.4 Using Empty Right to segment the labels into the children of the Subject headings. 47

3.5 Using Empty Below to segment the labels into the children of the Ethnic headings. 47
3.6 An example Table that will be used to define the vocabulary. 52
3.7 A breakdown of the Identification step of the workflow. The arguments IdentPrimary and IdentAuxiliary specify the respective algorithms to use. 54
3.8 This is a Table of New Zealand GDP Data (Statistics New Zealand, 2013) and is an example of a (comparatively) good Table as it is consistent in format with no weird features. However, the row labels are numbers, posing some complications as the main Identification algorithm looks for numbers to identify the data. In cases such as this, TableToLongForm uses pattern recognition (e.g. sequences of numbers such as 1972, 1973, 1974... are more likely to be labels than data) to attempt to identify such numbers to be labels, but this process is far from perfect. 55
3.9 This is a Table of NZQA Scholarship Data (New Zealand Qualifications Authority, 2012) and is the original motivating dataset that resulted in the creation of TableToLongForm. The Table contains a title and metadata in a form typical of Tables, but also displays some clear dividing rows and columns. This seems like an attractive way to attempt Identification of the features of this Table, but closer inspection reveals that these dividers are not very informative. Consider for instance the dividing columns 6 and 11. Though appearing to be the same, column 6 is only a (meaningless) sub-divider, while column 11 is a divider that splits the Decile groups. This problem can be overcome with pattern recognition, but as dividing rows and columns are an uncommon feature in Tables, such an algorithm is of limited value. The current method of Identification searches for blocks of numbers, and this is mostly successful for this Table. There is a slight problem with column headings due to misaligned column label, e.g. `Decile 1-3` in (3, 4) should be in (3, 3) to be aligned with `# of Entries`. This problem is resolved not in Identification, but in the next stage, Discern Parentage. 56
3.10 This is a Table of UK NEET Data (Department for Education (UK), 2013), displaying another case of number labels. In this case the current pattern recognition fails to correctly recognise the number labels as labels and manual input is required for correct conversion. The manual specification would be: 57
3.11 This is a Table of Top 100 Baby Girls' Names in New Zealand (Department of Internal Affairs (NZ), 2012), displaying a peculiar mismatch of column labels to the corresponding data, in addition to number labels. The number labels here are correctly identified with pattern recognition. The correction of the mismatched column labels is resolved not in Identification, but in the next stage, Discern Parentage. However, during Identification the columns identified as 'data' must include all the data and column labels (in this case `cols = list(label = 2, data = 3:26)`). 58

3.12 A truncated example of the `colplist` for the Labour Force Status data used in Figure 3.1. It represents the hierarchical relationships of the column labels. We can see that it has correctly identified `Male` as a top-level parent with the ethnic categories, such as `European Only`, nested inside. The ethnic categories are in turn parents to the lowest-level categories, such as `Employment Rate`. 61

3.13 A truncated example of the `colplist` from Figure 3.12, printed with the default list printing method. This is much less useful, but does give better insight into the internal storage structure for users familiar with R lists. 61

3.14 A breakdown of the Discern Parentage step of the workflow. `IdentResult` from the Identification step is used obtain subsets of matFull that correspond to the labels and the data. These subsets are then processed with the ParePreRow and ParePreCol algorithms as specified by the respective arguments. These adjusted subsets are then used by the Main Parentage algorithm to discern the parentage of the Table. 62

3.15 An example of a `rowplist` and the reconstructed version (both truncated). These must then be combined with the reconstructed `colplist` and the data for the final dataframe. Broadly speaking, the Reconstruction algorithms simply iterate down a plist recursively, extracting the information and pasting them together appropriately to create the dataframe. 68

3.16 A breakdown of the Reconstruction step of the workflow. ReconsRowLabels only makes Row Labels. ReconsColLabels takes the Row Labels, may make additional Row Labels (for Col Parents), then combines it with the Col Labels, grabbing the associated columns of data as it does so. Thus the output of ReconsColLabels is the final dataframe output. 69

3.17 While there is no data for the total number of downloads across all CRAN mirrors, there is data from the RStudio mirror. The different colours indicate different versions of TableToLongForm. The trend has held remarkably steady across time, to the point I suspect the reliability of this data (the steady downloads may be from other mirrors syncing, or other such automated downloads, and not indicative of 'real' users), but it is unfortunately one of the only sources of data on package downloads. 71

4.1 The Many Eyes 'interface'. This is a screenshot of one of the webpages used to create a new plot. Many Eyes is entirely web-based and requires no separate client software. 77

4.2 The Tableau Public interface. In this example, we are looking at a 'dashboard', allowing us to place multiple graphics on the same page. On the right, Tableau Public suggests some appropriate graph types for the selected data subset (inappropriate graph types are greyed out). 79
4.3 A screenshot from `https://developers.google.com/chart/interactive/docs/examples` demonstrating interactivity between a *Table* and a *BarChart*, sorting by Name by clicking on the appropriate column in the *Table*. 81
4.4 A continuation of Figure 4.3, where the *Table* has been resorted by Salary, with the *BarChart* following suit. 81
4.5 A screenshot from `http://www.highcharts.com/demo/bar-basic` demonstrating a basic bar chart. 83
4.6 A continuation of Figure 4.5, where clicking on **Year 2008** in the legend has filtered this data and the bar chart (including axis) has updated accordingly. . . 83
4.7 The **Test Bargraph** that will be used to explore each Low-level Language. I will attempt to recreate this same bargraph as closely as possible with every Low-level Language covered. 85
4.8 The **Test Bargraph** showing a 'ghost bar' which displays the potential height of the updated bar if the user clicks. Note that this screenshot does not capture the mouse pointer. 86
4.9 Upon clicking in Figure 4.8, the bar height is updated. 86
4.10 A screenshot of the webpage containing the HTML Table from which we will obtain our data. **Source:** Inland Revenue and licensed by Inland Revenue for re-use under the Creative Commons Attribution 3.0 New Zealand Licence. . . . 96
4.11 An example raster graphic of a circle. The original image is a 150 by 150 pixel image saved as a png. 99
4.12 An example vector graphic of a circle. The circle is drawn in LaTeX using the `tikzpicture` package. 99

5.1 The layout of WeBIPP's Graphical User Interface. Free Form Windows, as their name implies, are not static elements of the interface, and can be moved about and closed. The Free Form Window shown here is the WeBIPP Code interface, an embedded and fully functional code editor. 114
5.2 This scatterplot was drawn using R (R Core Team, 2014). The data is one of the datasets included with R. Using WeBIPP, we wish to recreate a graphic similar to this. The R command used to produce this image is: 115
5.3 Read in the data to plot. 116
5.4 Select the Cartesian Frame from the menu and click anywhere on the Graph Region to place a Cartesian Frame object. 117

5.5 Assign `speed` from our data to the x-axis. . . . 117
5.6 Click the attribute name (`x` currently) to open a list of attributes for the Cartesian Frame. . . . 118
5.7 Select the attribute `y` and repeat the process from before to assign `dist` to the y-axis. We are done with the Cartesian Frame for now and we can unselect the object by clicking it again in the Object Menu. . . . 118
5.8 By a similar process as before, a Circle may be placed inside the Cartesian Frame by selecting the Circle from the menu, then clicking anywhere inside the Frame. 119
5.9 The Quick Menu is an alternative to selecting from the Object Menu and using the Attribute Interface. It is accessed by right-clicking directly on the Object in the Graph Region. . . . 119
5.10 Assign `speed` from our data to `cx` of the Circle. The Figure shows the data variable `speed` highlighted, but not yet clicked. Once clicked, as the data contains multiple values, the Circle Object will replicate itself to become multiple circles, with each data value being assigned to a different circle. See Figure 5.11 for how the interface is changed after the click. . . . 120
5.11 Assign `dist` from our data to `cy` of the Circle. The Figure shows the data variable `dist` highlighted, but not yet clicked. See Figure 5.12 for how the interface is changed after the click. . . . 120
5.12 A basic scatterplot is complete, the remaining steps add polish to make it look better. The first two steps of polish involve giving the axes more informative names by changing `x-name` and `y-name` of the Cartesian Frame. . . . 121
5.13 There are some overlapping points, so it would be better if the circles were not filled to make it easier to identify the individual points. This is done by changing the `fill` attribute of the Circle to `"none"` (quotation marks are necessary). . . . 121
5.14 WeBIPP's `rgb` interface. . . . 122
5.15 Making the circles smaller will reduce overlap and help distinguish each individual point. This can be achieved by assigning a smaller value (in this case 5) to the `r` attribute of the Circle. Unfortunately this won't help with the two overlapping points at `(13, 34)`, as they are exactly on top of each other. . . . 123
5.16 Save the final graphic. . . . 123
5.17 This Population Pyramid was drawn using R. The data is of the New Zealand Population data (Statistics New Zealand, 2011). Using WeBIPP, we wish to recreate a graphic as close to this as possible. The steps that follow are more for demonstration purposes, and less to be precise and clear steps to recreate the graph. See the code at the end, or the tutorial on the website, for concise instructions. . . . 125
5.18 Setting up the Cartesian Frame. . . . 126

5.19 Adjust the x-axis domain. . . . 127
5.20 Flip the x-axis to the correct orientation for the left-side of the pyramid. . . . 127
5.21 For similar reasons, adjust the y axis range to make full use of the height. . . . 128
5.22 We only want the x axis labels, so we set `axes` to `[1]` (WeBIPP uses the same system as R for referencing axes). We also want exactly 4 tick marks on the x axis, so we set `axes-opts` to `{"1":{"ticks":4}}`. Unfortunately neither of these attributes have nice interfaces for assigning these values, so for now, the user must simply know what to assign directly using a default interface, if they wish to tweak these settings. . . . 128
5.23 The resize interface for WeBIPP. . . . 129
5.24 The dimensions have been adjusted to roughly the correct value, using the interface. The WeBIPP Code interface is open showing the code. . . . 130
5.25 The dimensions have been computed exactly via manual adjustment of the code. 130
5.26 Shift the Cartesian Frame into position by applying a translation of `[20, 80]`. . 131
5.27 The line segment button. . . . 132
5.28 A line segment using the default values. . . . 132
5.29 Set the x values for the line segment. . . . 133
5.30 Have the line segment start from the top. . . . 133
5.31 Have the line segment end at the bottom. . . . 134
5.32 Make the line segment dashed. . . . 134
5.33 Rectangle with x centred at 0 pixels. . . . 135
5.34 Rectangle with x centred at 0 according to the x axis scale. . . . 136
5.35 Rectangle with its right-edge at 0 according to the x axis scale. . . . 136
5.36 Assign `Male` to the widths of the bars. . . . 137
5.37 Set the rectangle `height` to 1. . . . 138
5.38 With `height-useScale` set to `true`, there are now no gaps between the bars. . 138
5.39 As this is Male data, assign an appropriate blue colour to the bars using the interface. . . . 139
5.40 We could also assign a specific colour by name, in this case `"lightblue"` (quotation marks necessary). This more closely matches the graph we wish to replicate. 139
5.41 The text button. . . . 140
5.42 A text object is used to draw a label. . . . 140
5.43 Position the text on the left-edge of the graph. . . . 141
5.44 Position the text above the graph. . . . 141
5.45 Anchor the text correctly for correct alignment. . . . 142
5.46 Adjust the size to something appropriate for a label. . . . 142
5.47 Formalised reuse of code using the Functionise interface. . . . 145

5.48 Check the highlighted text carefully to ensure only what we want is turned into arguments. . . . 146
5.49 Sometimes a more cumbersome match may be required to get a unique match. . . 146
5.50 In other cases multiple matches may be desirable. As the right-side will be using Female data, all cases of Male being updated to Female is exactly what we want. 147
5.51 With all the arguments set, we finish creating the function. . . . 147
5.52 Clicking on the newly created button will open an interface to help use the function. 148
5.53 The available arguments. . . . 149
5.54 Setting the arguments. . . . 149
5.55 We can also compute argument values, as before. . . . 150
5.56 Be sure to match the text that has been turned into an argument. . . . 150
5.57 Finish off the call. . . . 151
5.58 The right-side of the pyramid complete. . . . 151
5.59 Placing the shared y axis label. . . . 152
5.60 Assign `AgeGroup` to y for positioning. . . . 153
5.61 Assign `AgeGroup` to text for the axis labels. . . . 153
5.62 Placing an overall Title for the graph. . . . 154
5.63 To make the Title more prominent, we can bold it. . . . 155
5.64 We can also change the font of the title. . . . 155
5.65 Placing an overall x axis label. . . . 156
5.66 The finished Population Pyramid, which is fully scalable. . . . 157
5.67 The interface resized to 1600 by 720. . . . 157
5.68 The resize interface with the default size of 920 by 720. . . . 158
5.69 The interface resized to 640 by 480. Notice that our chosen font sizes may not be appropriate at other interface sizes. . . . 158
5.70 The SVG containing the WeBIPP GUI. The first five groups (`<g>` with IDs `gGraph`, `gMenuTab`, `gMenuPri`, `gMenuSec` and `gMenuTer`) form the GUI, while the next two groups (with IDs `wbip-code-high` and `wbip-code-low`) form the Free Form Windows that contain ForeignObjects that hold the CodeMirror instances. 162
5.71 An example of Frames and nesting. gELs 1, 2 and 6 are children of the Graph Region itself. gELs 3 and 4 and children of gEL 2. gEL 6 is a child of gEL 4. . . 163
5.72 The icons that will be used for the Structure diagrams. . . . 164
5.73 Structure of the Initialisation process. Not mentioned are the D3 External Library and the WeBIPP's Utils Internal Library. Almost everything in WeBIPP makes use of the Utils library and all drawing on the SVG is done through D3. Some utility functions in the D3 library are also used. . . . 165
5.74 Simplified structure of how a Circle is added. Can be generalised to most other objects, with some variances in complexity. . . . 166

5.75 Simplified structure of how a Circle's attribute is adjusted. Can be generalised to most other objects, with some variances in complexity. 167
5.76 Two examples of the Numeric Value Interface. Left: Assigning a specific value of 150. Right: Modifying the `OV` (Original Value), the mouse is over the `OV` button, highlighting it and displaying the tooltip (if defined). 187
5.77 How the Circle looks before (left) and after (right). Note that for the right graph, many of the `cx` values are the same, resulting in overlapping circles and the appearance of a smaller number of circles than the 50 that actually exist. . . 192

List of Tables

2.1 Example CSV and CSV-like data using (from left-to-right) a comma, a single space and a tab as the delimitters. The simplicity of data structure makes it very easy for this data format to be read and manipulated via a computer, but the lack of nice alignment of cells makes it unsuited for the naked eye. 11

Chapter 1

Introduction

> *It's like that basic rule in nutrition: Food that is not eaten has no nutritional value. Data which is not understood has no value.*
> — *Hans Rosling*

In recent times there has been a movement toward *Open Data*, particularly for government data[1,2] (which also falls under the purview of *Open Government*). The reasons cited for this movement are typically intangible[3,4,5], but it is easy to appreciate how such benefits can arise. As Cole (2012) notes however, there are degrees of openness. For instance, data may only be available on request, or the released data may only be a subset or an aggregated result of the original data. Additionally, even if the data itself is 'open', the format or content of the data may require additional resources, such as expert knowledge or significant computing power, to process or understand the data. This presents barriers around accessibility and usability, limiting the potential benefits of making the data open.

The purpose of this thesis is to examine ways to make *Open Data* more open, more accessible and thus more useful to a wider audience. It begins with a Literature Review of Open Data in New Zealand (Chapter 2) to understand what data is out there and to identify what barriers currently exist that hinder the usage of the data. It then presents TableToLongForm (Chapter 3) a tool that attempts to address a major problem identified

[1] "In many countries across the world, discussions, policies and developments are actively emerging around open access to government data." Davies and Bawa (2012a)

[2] "Over 100 OGD [Open Government Data] initiatives are active across the globe, ranging from community-led OGD projects in urban India, to a World Bank sponsored OGD programme in Kenya, government-led developments in Brazil, civil-society initiated work in Russia, and a World Wide Web Foundation supported programme in Ghana." Davies and Bawa (2012b)

[3] "We promote open knowledge because of its potential to deliver far-reaching societal benefits which include... better governance... culture... research... economy" Open Knowledge Foundation (2012)

[4] "One of the pillars of open and transparent government is open government data and information." New Zealand Government ICT (2012a)

[5] "The online publication of structured datasets by governments is seen as playing an important role in driving the transparency and accountability of states, enabling new forms of civic participation and action, and stimulating economic growth and development." Davies and Bawa (2012b)

in the Literature Review, that of data being released in non-machine-readable hierarchical tables. Once data is in a useful format, one of the most effective ways to communicate it to a wider audience is through visualisations, so the thesis continues with a Literature Review of Graphical Tools (Chapter 4) examining what is currently available for creating web-based interactive graphics. It then presents WeBIPP (Chapter 5) an innovative tool for prototyping web-based statistical graphics, which works in a way that no other graphical tool does. The thesis concludes (Chapter 6) with summarising remarks and possibilities for future work.

Literature Review: Open Data in New Zealand

To examine ways to make Open Data more useful, it is first necessary to understand what Open Data is and how it is released, accessed and used. This Literature Review begins by defining Open Data, including key desirable properties and terminology relevant to the discussion. It then provides an overview of New Zealand State Sector sources of Open Data.

TableToLongForm

TableToLongForm is an R package that automatically converts hierarchical Tables intended for a human reader into a simple LongForm dataframe that is machine readable, making it easier to access and use the data for analysis. It does this by recognising positional cues present in the hierarchical Table (which would normally be interpreted visually by the human brain) to decompose, then reconstruct the data into a LongForm dataframe. The chapter motivates the benefit of such a conversion with an example Table, followed by the core concepts that drive TableToLongForm. It continues with examples of how the package may be used, including how it may be extended with external algorithms. Finally it goes into the implementation details that covers in more depth how TableToLongForm accomplishes its task, but also its limitations and how it can fail.

Literature Review: Graphical Tools

This Literature Review examines free Graphical Tools that produce interactive output ideal for web-based viewing, ranging from *Tableau Public* to *Google Chart Tools* and *D3.js*. To compare between the complex tools (defined as *Low-level Languages*) like *Processing* and *Raphaël*, the same graphic is implemented under each *Low-level Language*, which serves both as a method of directly comparing between the tools, and also as an example of how to use the tool. A brief primer on some graphical terms is also provided, such as the difference between *raster* and *vector* graphics, or how the *SVG* image file format works, to make the review more accessible to audiences unfamiliar with such details.

WeBIPP

WeBIPP is a web-based interactive tool for building statistical graphics from scratch without writing any code. It has a Graphical User Interface that can be used to place the building blocks of graphics (rectangles, circles, lines, etc.) and assign these building blocks data. By assigning data to the x and y of Circles, a scatterplot can be made. By assigning data to the widths of Rectangles and by having these rectangles start from 0, a bargraph can be made. In such ways, simple statistical plots can be built from scratch in minutes. This is but a part of what WeBIPP can do.

Not only can these building blocks be used to build more complex graphics, WeBIPP has extensive addon support, and the interface can handle not just the basic building blocks, but complex objects that can be used to easily build more complex graphics. As the interface is used to build the graphics, the user is not simply creating a graphic, they are writing code in the background. For those who have no coding knowledge, this fact can be ignored, making WeBIPP accessible to this audience. But for those who have coding knowledge, they can utilise it to examine the code, to tweak it and to reuse it.

Unlike many other tools for creating statistical graphics where the user is limited to what the tool itself allows, WeBIPP does not limit the user. If the interface currently lacks support for something, the user can simply write their own code to accomplish it. In this way, WeBIPP becomes much more powerful than other tools of similar nature.

In addition to an introduction that covers what WeBIPP is, what it seeks to achieve and how it differs from similar tools, the chapter also has extensive sections demonstrating how both GUI and code can be used to produce graphics from scratch. This is followed by some details on how WeBIPP works and how it may be extended with addons. It concludes with a discussion on its limitations and possible future work.

Chapter 2

Open Data in New Zealand
A Literature Review

2.1 Introduction 6
2.2 What is Open Data? 7
2.2.1 A Definition 7
2.2.2 Users of Open Data 8
2.2.3 Good and Bad Open Data 9
2.2.4 Data Formats 10
2.2.5 Other Relevant Topics 18
2.3 Overview of Open Data in New Zealand 19
2.3.1 Overview of the Overview 19
2.3.2 Statistics New Zealand 22
2.3.3 data.govt.nz 25
2.3.4 The Template 26
2.3.5 Major Sources 27
2.3.6 Minor Sources 31
2.3.7 Not Sources 37
2.3.8 Outside Scope 39
2.4 Conclusion 40

2.1 Introduction

To examine ways to make Open Data more useful, it is first necessary to understand what Open Data is and how it is released, accessed and used. This Literature Review aims to provide an Overview of Open Data in New Zealand giving particular focus to New Zealand State Sector sources. Of course, Open Data includes more than just government data, such as data arising from academic research (which also falls under the purview of *Open Research*), but the vast amount of data openly available from governments around the world make it an ideal low-hanging fruit. The scope is limited to New Zealand to attain depth, in exchange for breadth.

The intent of this review is not to evaluate the performance of the New Zealand government and its organisations' release of data, but instead to understand the current situation, to understand what barriers currently exist that hinder the utilisation of Open Data. In the process it becomes necessary to deal with some policy issues and some thoughts are given on what would be ideal from a Technical Users' perspective; however such comments are intended primarily to aid in the understanding of the current situation, rather than be evaluative remarks on current practice.

Thus this Literature Review will:

1. Define Open Data (Section 2.2), including key desirable properties and terminology relevant to the discussion. In doing so some policy and procedure issues are considered, as they are relevant to understanding what makes for *Good* Open Data.

2. Provide an Overview of Open Data in New Zealand State Sector sources (Section 2.3). The State Sector organisations covered are those listed as *Public Service departments* on the State Services Commission's website (State Services Commission, 2012), which includes many of the organisations the general public will be familiar with, such as the Ministry of Health or Inland Revenue.

This Literature Review was conducted in 2012, and due to the nature of the subject many details become out-dated at an alarming rate. Many of the points made in the various summaries should remain valid for some time, but specific details, particularly pertaining to specific organisations as in Section 2.3, may no longer apply at the time of reading.

2.2 What is Open Data?

2.2.1 A Definition

We all vaguely understand what is meant by Open Data, but let us be more formal with our definition. The Open Knowledge Foundation (2012) defines *Open Knowledge* to be *"content that people are free to use, re-use and distribute without legal, technological or social restrictions."* Davies and Bawa (2012b) observe that:

> "Open data" is just one of a number of high-profile labels with the prefix "open". Open government, open access, open innovation, open education and open knowledge are some of the other initiatives and movements in this area. Many of these draw from the emergence of "open source" as the inspiration for their development.

Cole (2012) notes that there are degrees of openness. For instance, data may only be available on request, or the released data may only be a subset or an aggregated result of the original data. Additionally, even if the data itself is 'open', the format or content of the data may require additional resources, such as expert knowledge or significant computing power, to process or understand the data. This presents barriers around accessibility and usability, limiting the potential benefits of making the data open.

For the purposes of this Literature Review, the key features of Open Data are:

Free to use and re-use At a minimum this means there are no legal restrictions on using the data for any purpose. Ideally, it also means there are no non-legal restrictions, such as the data format being obfuscated, either intentionally or unintentionally, such that it is unsuitable for use other than for its 'intended' purpose.

Free to distribute Ideally this means we are free to distribute the data itself, as well as any derived output of the data. Sometimes the case may be that the data is freely available and we are free to distribute derived output (so long as it is sufficiently aggregated or processed to no longer be considered raw data), but we are restricted from re-distributing the data itself[1]. Such cases, while not ideal, will still be considered Open Data.

Freely available Meaning the data is readily accessible, ideally on the internet, with little to no requirements for authentication or approval. Sometimes the case may be that

[1]This may arise for example, when the raw unaggregated data is subject to privacy restrictions that cease to be a problem with sufficient aggregation. Due to New Zealand's privacy laws (see Section 2.2.5) we must remain wary of potential privacy breaches even if we are using data that has been released 'openly' by another party. N.B. When dealing with data from another state/country, it would be wise to consult the relevant privacy laws for that state/country.

registration and login is required to access the data. If this process is easy and incurs no monetary cost, while not ideal, the data is still considered Open. Other times the data may be 'freely available on request', but with some manner of lengthy approval and response process (often requiring human involvement on the provider side). Such requirements are considered to be a significant barrier for the wider public, and hence such data is not sufficiently Open to be considered Open Data.

2.2.2 Users of Open Data

Users of Open Data will be classified into the following categories:

Casual Users The 'ordinary person'. Generally these users will only use data *as is* with little to no independent effort to further process the data. They will appreciate data being presented in a human-friendly fashion, such as hierarchical tables that make it easy to locate a specific value, and will value plots of the data, as they are unlikely to make such themselves.

Semi-Technical Users These are 'ordinary' people with some technical knowledge, or they are in a profession which requires some technical skill with data, such as data journalists or financial professionals. They may carry out some data manipulation, usually manually and often with the aid of a GUI tool such as Excel, to draw conclusions from the data. As manipulation is often manual, they generally prefer human-friendly data formats; machine-readability is a concern to the extent that the data is easily selected for copying/referencing.

Technical Users These are users who use script-based approaches to data manipulation to handle very large volumes of data, possibly spread across multiple files. They typically desire data in as raw and simple a format as possible so that it is machine-readable; they can then use this to produce any other format they may desire, to conduct more sophisticated statistical analyses, and to create summaries and visualisations of the data. Their outputs can often be of value to all three types of users.

Desirable Open Data will depend on the type of user, ideally Open Data should be released in multiple formats to cater to the different users. Where that is impractical the format should be one that caters to as many users as possible, ideally encompassing *Technical Users* who are capable of taking this data and re-using it in a way that benefits the other users. Conversely, though *Casual Users* form the largest user base, as they are usually not capable of re-using the data to further benefit other users, catering only to *Casual Users* will severely limit the benefits of Open Data.

The focus of this Literature Review will typically be from the perspective of a *Technical User*, desiring machine-readable data as inputs, with the aim of producing tools and output that are suitable for use by all three types of users.

2.2.3 Good and Bad Open Data

Within Open Data, it is helpful to have some examples of what might be ideal and desirable qualities to see in Open Data, and conversely, what might be undesirable qualities. These are from the perspective of a *Technical User* (as defined in Subsection 2.2.2).

Open Data is considered *Good* if it possesses many desirable qualities and few undesirable qualities. Conversely, Open Data possessing many undesirable qualities (while still classified as Open Data) will be defined as *Bad* Open Data. Desirable qualities are:

- Any restrictions (or lack thereof) are made clear, e.g. a known copyright licence is clearly attached (see Section 2.2.5).
- Data is accessible in a simple, easy way, e.g. the data can be obtained by simply clicking on a link.
- Data is structured in a simple, easy to use format (see Subsection 2.2.4).
- Any supplementary information is clearly connected to the data. These may include:
    1. Details on how the data was collected and processed, including any steps taken to identify and correct errors in the data (such as mistakes during data entry), or any modifications made to ensure confidentiality and privacy protection.
    2. A clear definition of the headings, groupings or other notation used in the data.
    3. Any relevant information that may affect analysis and interpretation, e.g. if there was a change in how data was collected, processed or classified over time.

The lack of desirable qualities (such as not having a copyright licence) is undesirable. However in some cases, there may be specific features that are undesirable, such as:

- Data is only accessible via an inconvenient or slow user interface, such as a restrictive online form.
- Data is structured for presentation and human reading. This could be through excessive aggregation of the data before release, or a layout that is chosen for aesthetic appeal rather than ease of data processing. In particular, this makes it very difficult to use the data in any other way than originally envisaged by the data provider (which may be intended in some cases), stifling innovation and potential re-use of the data.

2.2.4 Data Formats

The data format can significantly influence how the data can be used. This section discusses some common data formats used by State Sector Open Data Sources and how these formats can enable or prevent usage of the data.

Executive Summary

- Releases in CSV are the simplest, and hence enable the greatest data re-use but requires some experience with data manipulation making it less accessible to non-technical users.
- Releases in XLS can have significantly different implications for data use and re-use depending on the specific internal format. Of these internal formats the Pivot Table is most recommended as it caters to the widest range of users, enabling some data manipulation by users within Excel, while also making it possible to convert and export the data to a simpler, machine-readable format. Ideally though, the data should also be released in a simple multi-platform format, such as a CSV, rather than exclusively as an XLS file.
- Releases in HTML Tables are easier to share via the web while still being structured to some extent, making it possible to salvage and convert the data to a more convenient format. However, where possible the data should also released in a different format (such as CSV) so such conversions are not necessary.
- Releases in PDF Tables should never be done as the PDF is purely a presentation format and not suitable for data transfer. While it is acceptable to include tables of data in PDF documents for *presentation* purposes, the data itself should also be released in a better format, and this should be clearly indicated and referred to in the PDF document.

CSV

Perhaps the most basic data format is the *CSV*, or *Comma Separated Values*. Despite the name, CSV-like files that use other delimiters, such as a blank space or a tab, are sometimes called CSV. In CSV data is stored in *longform*, that is each row represents an individual observation and contains all the information relating to that individual in that single row, with each piece of information separated (into 'columns') by the delimiter. An analogy would be if we stored information on people, then each row would be a person and the columns would be attributes like height, weight, gender and age. We do not require information from any other person (row) in order to obtain information about any given

person. This simplicity of structure makes CSV files very easy to read and manipulate via a computer as there is no need to decipher relationships in the data, but this simplicity can make the format cumbersome to work with for users unfamiliar with data manipulation of such a format (see Table 2.1).

`Col1,Col2,Col3`
`"A",1,19`
`"B",2,28`
`"C",3,37`
`"Longer text",4,46`
	`Col1 Col2 Col3`
	`"A" 1 16`
	`"B" 2 25`
	`"C" 3 34`
	`"Longer text" 4 46`
		`Col1  Col2  Col3`
		`"A"  1  16`
		`"B"  2  25`
		`"C"  3  34`
		`"Longer text"  4  46`

Table 2.1: Example CSV and CSV-like data using (from left-to-right) a comma, a single space and a tab as the delimitters. The simplicity of data structure makes it very easy for this data format to be read and manipulated via a computer, but the lack of nice alignment of cells makes it unsuited for the naked eye.

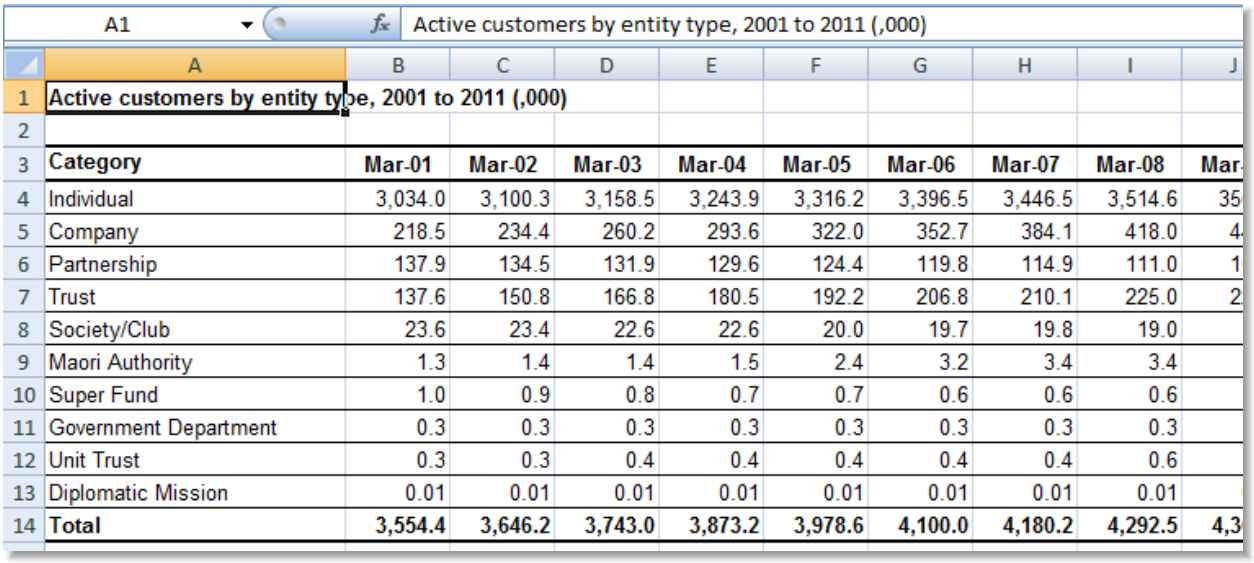
XLS

Another common format is the *XLS* (*Excel Spreadsheet*), and this appears to be the preferred data format for the majority of State Sector Open Data Sources. We will further classify XLS files as follows:

XLS-Longform - When the data in the spreadsheet are presented in longform. These are very similar to a CSV, but can add hurdles to users who may wish to work in a software other than Excel as they must convert the data first. There is little benefit in releasing data in XLS-Longform rather than CSV, as Excel is perfectly capable of reading both, while XLS-Longform can only be read by Excel or similar XLS-compatible software. See Figure 2.1.

XLS-Table - When the data is presented in some form of table that relies on non-trivial relationships. Unlike a longform where all the information for a piece of data is essentially contained in a single row, an XLS-Table requires collecting information across multiple rows to form the full picture. Such tables are relatively easy for a human brain to decipher and may even be easier to read for a human compared to a longform, however the non-trivial relationships makes it extremely difficult to machine-read the data. See Figure 2.2 and Figure 2.3.

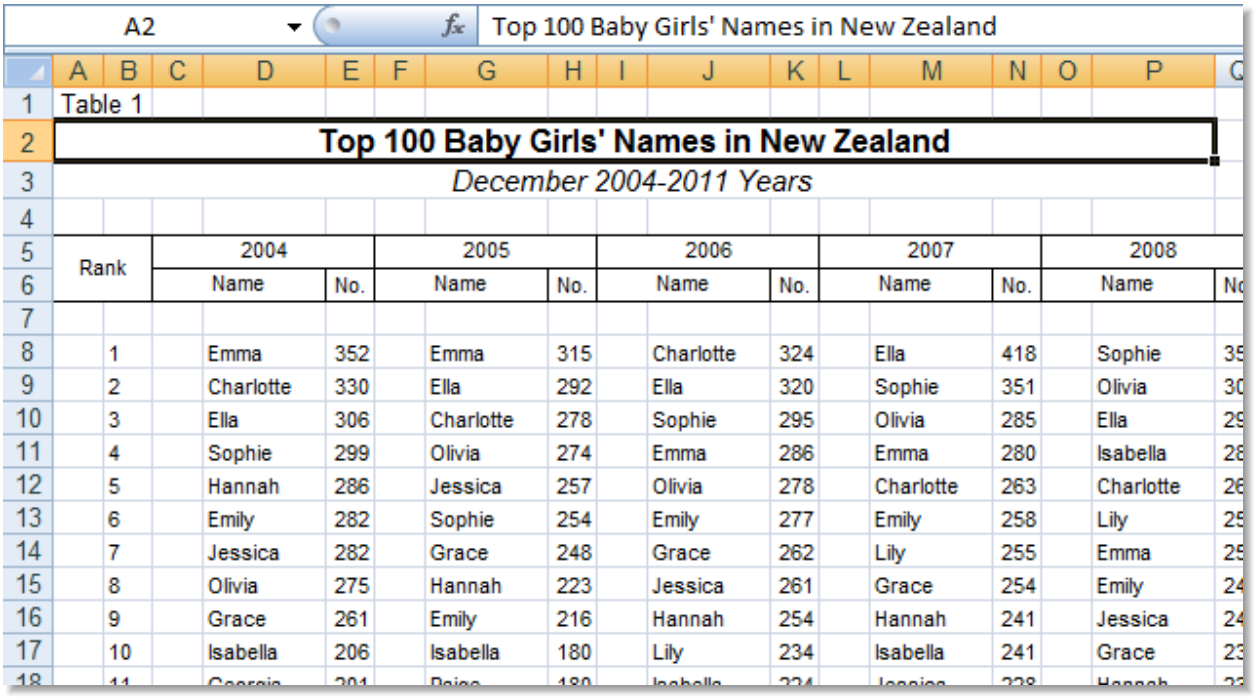
XLS-PivotTable - When the data in the spreadsheet is presented using the Pivot Table feature of Excel. The Pivot Table is an interactive feature of Excel that allows dynamic



A1 | Active customers by entity type, 2001 to 2011 (,000)

	A	B	C	D	E	F	G	H	I	J
1	Active customers by entity type, 2001 to 2011 (,000)									
2										
3	Category	Mar-01	Mar-02	Mar-03	Mar-04	Mar-05	Mar-06	Mar-07	Mar-08	Mar
4	Individual	3,034.0	3,100.3	3,158.5	3,243.9	3,316.2	3,396.5	3,446.5	3,514.6	35
5	Company	218.5	234.4	260.2	293.6	322.0	352.7	384.1	418.0	4
6	Partnership	137.9	134.5	131.9	129.6	124.4	119.8	114.9	111.0	1
7	Trust	137.6	150.8	166.8	180.5	192.2	206.8	210.1	225.0	2
8	Society/Club	23.6	23.4	22.6	22.6	20.0	19.7	19.8	19.0	
9	Maori Authority	1.3	1.4	1.4	1.5	2.4	3.2	3.4	3.4	
10	Super Fund	1.0	0.9	0.8	0.7	0.7	0.6	0.6	0.6	
11	Government Department	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	
12	Unit Trust	0.3	0.3	0.4	0.4	0.4	0.4	0.4	0.6	
13	Diplomatic Mission	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	
14	Total	3,554.4	3,646.2	3,743.0	3,873.2	3,978.6	4,100.0	4,180.2	4,292.5	4,3

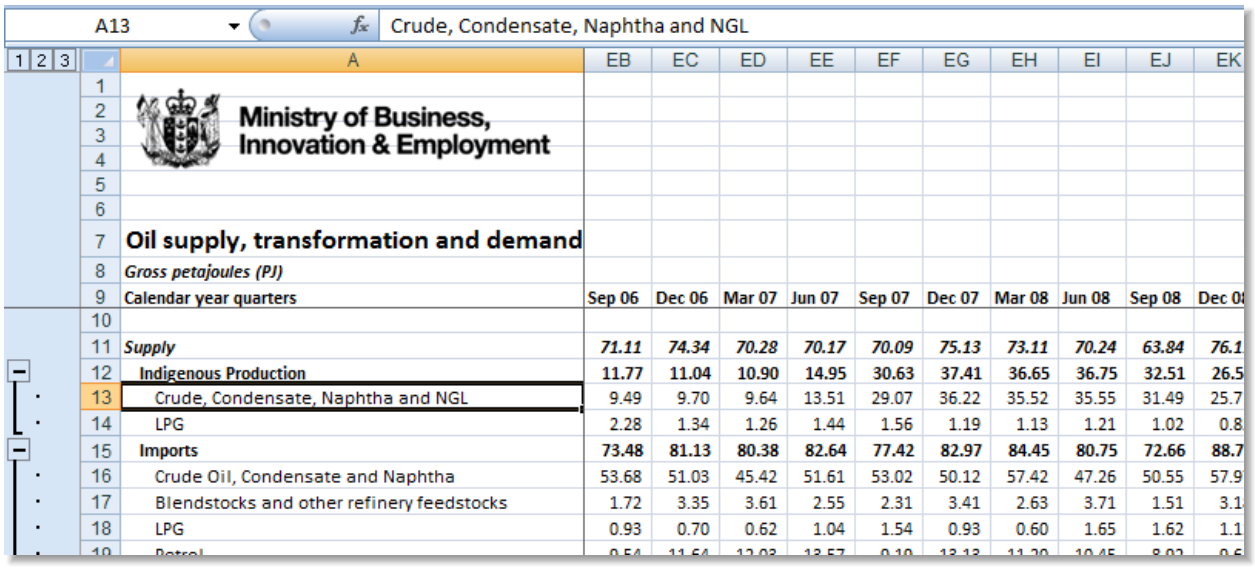
Figure 2.1: An example of an XLS-Longform. The data is *active-customers-by-entity-type-2001-to-2011.xls* from *Inland Revenue*.



A2 | Top 100 Baby Girls' Names in New Zealand

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Table 1																
2	Top 100 Baby Girls' Names in New Zealand																
3	December 2004-2011 Years																
4																	
5	Rank	2004			2005			2006			2007			2008			
6		Name		No.	Name		No.	Name		No.	Name		No.	Name		No	
7																	
8	1	Emma		352	Emma		315	Charlotte		324	Ella		418	Sophie		35	
9	2	Charlotte		330	Ella		292	Ella		320	Sophie		351	Olivia		30	
10	3	Ella		306	Charlotte		278	Sophie		295	Olivia		285	Ella		29	
11	4	Sophie		299	Olivia		274	Emma		286	Emma		280	Isabella		28	
12	5	Hannah		286	Jessica		257	Olivia		278	Charlotte		263	Charlotte		26	
13	6	Emily		282	Sophie		254	Emily		277	Emily		258	Lily		25	
14	7	Jessica		282	Grace		248	Grace		262	Lily		255	Emma		25	
15	8	Olivia		275	Hannah		223	Jessica		261	Grace		254	Emily		24	
16	9	Grace		261	Emily		216	Hannah		254	Hannah		241	Jessica		24	
17	10	Isabella		206	Isabella		180	Lily		234	Isabella		241	Grace		23	
18	11	Georgia		201	Paige		180	Isabella		224	Jessica		228	Hannah		23	

Figure 2.2: An example of an XLS-Table. While this may be quite easy for the human brain to decipher, it is a nightmare for a computer. Consider for instance trying to obtain all the information for the first 'piece' of data. First we need to get the Rank from B8, understand this number is a 'Rank' by getting the column label from A5 (noting this is a different column to where the actual Rank value is stored). Second we need the name from D8, and the label for this from C6. Third we need to get the count of babies from E8 with label from E6. Finally we need the year from C5. The data is *Top100BabyNamesNewZealand2011.xls* from the *Department of Internal Affairs*.



A13 | Crude, Condensate, Naphtha and NGL

Ministry of Business, Innovation & Employment

Oil supply, transformation and demand

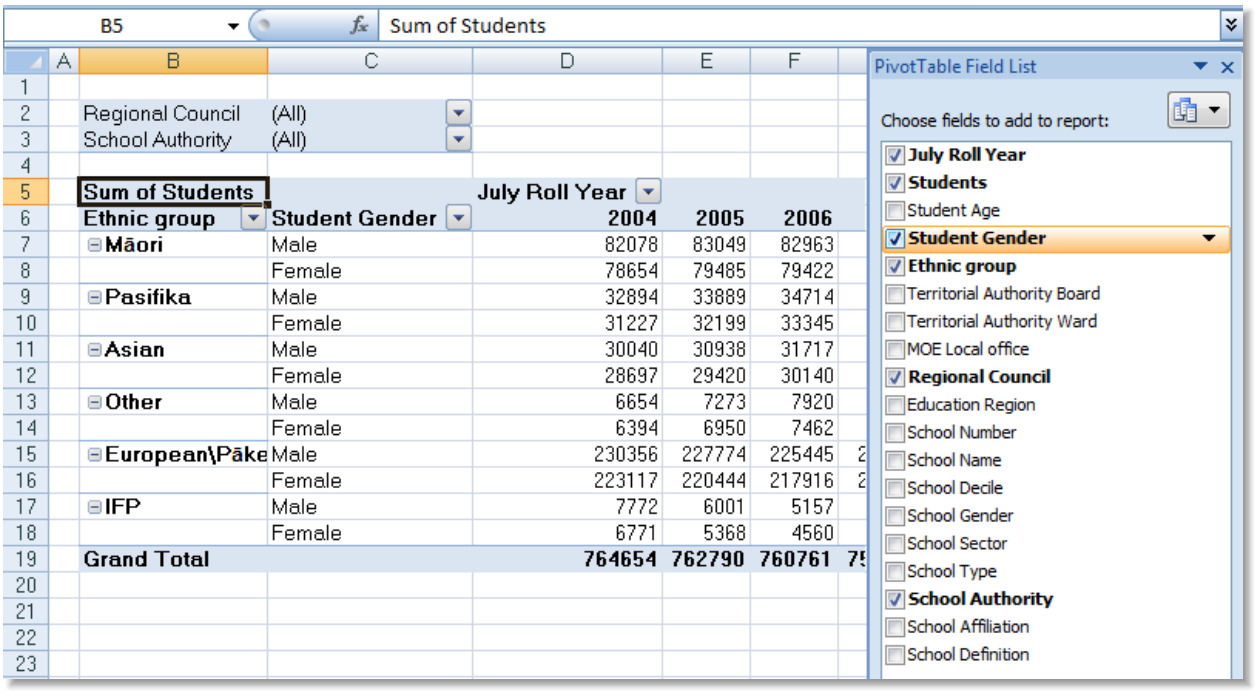
Gross petajoules (PJ)

	Calendar year quarters	Sep 06	Dec 06	Mar 07	Jun 07	Sep 07	Dec 07	Mar 08	Jun 08	Sep 08	Dec 0
11	*Supply*	*71.11*	*74.34*	*70.28*	*70.17*	*70.09*	*75.13*	*73.11*	*70.24*	*63.84*	*76.1*
12	Indigenous Production	11.77	11.04	10.90	14.95	30.63	37.41	36.65	36.75	32.51	26.5
13	Crude, Condensate, Naphtha and NGL	9.49	9.70	9.64	13.51	29.07	36.22	35.52	35.55	31.49	25.7
14	LPG	2.28	1.34	1.26	1.44	1.56	1.19	1.13	1.21	1.02	0.8
15	Imports	73.48	81.13	80.38	82.64	77.42	82.97	84.45	80.75	72.66	88.7
16	Crude Oil, Condensate and Naphtha	53.68	51.03	45.42	51.61	53.02	50.12	57.42	47.26	50.55	57.9
17	Blendstocks and other refinery feedstocks	1.72	3.35	3.61	2.55	2.31	3.41	2.63	3.71	1.51	3.1
18	LPG	0.93	0.70	0.62	1.04	1.54	0.93	0.60	1.65	1.62	1.1

Figure 2.3: An example of an XLS-Table. Once again, this may be relatively easy for a human brain to decipher, but a nightmare for a computer. Note that in this instance in-cell indentation and the Excel Grouping feature are used to communicate the hierarchical relationships in the data rows. Such information will typically be lost when converting the XLS file to another format, effectively forcing the user to a) Use Excel and b) Conduct any analysis manually. As Excel itself understands some of this information, it may be possible to write code in VBA (*Visual Basic for Applications*, a slightly modified version of *Visual Basic* integrated with Microsoft Office applications) to extract this information, but a cursory look at the code suggests this might be quite a challenge. The data is *Oil.xls* from the *Ministry of Economic Development* (now a part of the *Ministry of Business, Innovation & Employment*).

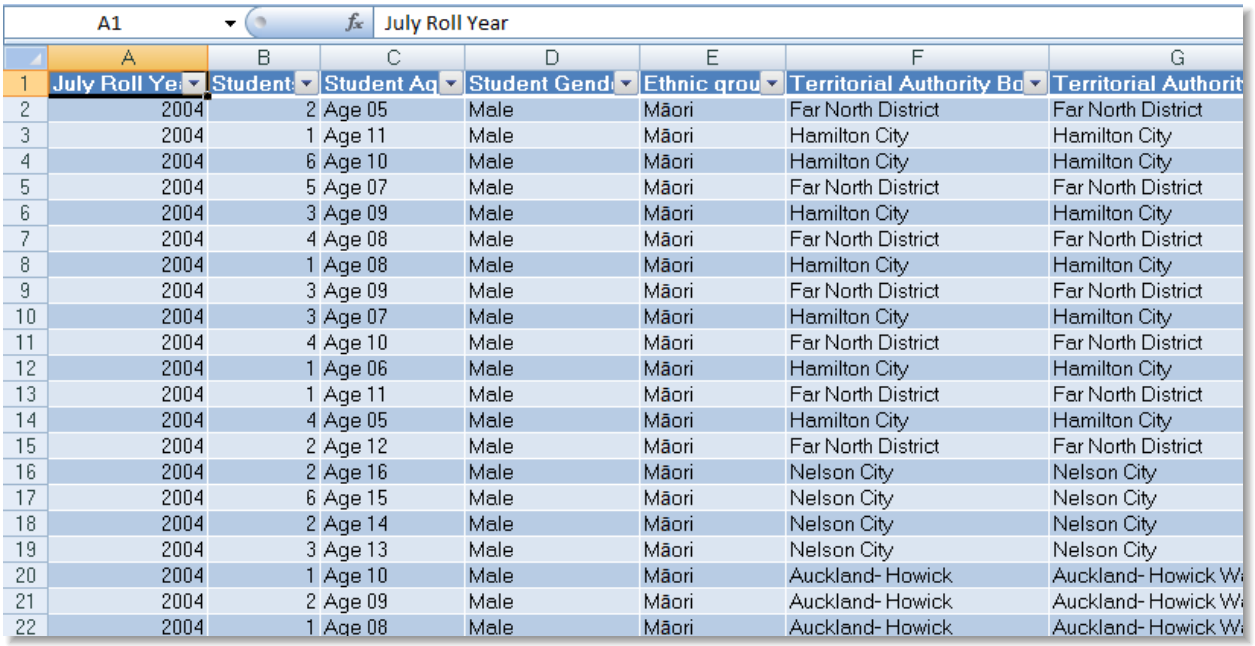
creation and modification of tables via a graphical user interface. It greatly enhances data exploration that can be done within Excel, though at first glance appears to be a significant barrier to analysis outside Excel. However, this is not the case, as it is possible to extract the underlying data using *Show Details* from the right-click menu. This makes XLS-PivotTables much better than XLS-Tables as it caters to many different types of users. It does however require the user to have software capable of using the Pivot Table. See Figure 2.4 and Figure 2.5.

XLS-ReportTable - When the data is presented as some form of report or table that is highly tuned to human-reading. We classify an XLS file as ReportTable if there are substantial barriers to machine reading. Generally any data released as an XLS-ReportTable can only be used manually and presents significant barriers to any re-use, even by manual means. This form of release is only appropriate if the underlying data is also released in a simpler form. See Figure 2.6.



Regional Council	(All)					
School Authority	(All)					
Sum of Students		July Roll Year				
Ethnic group	Student Gender	2004	2005	2006		
Māori	Male	82078	83049	82963		
	Female	78654	79485	79422		
Pasifika	Male	32894	33889	34714		
	Female	31227	32199	33345		
Asian	Male	30040	30938	31717		
	Female	28697	29420	30140		
Other	Male	6654	7273	7920		
	Female	6394	6950	7462		
European\Pāke	Male	230356	227774	225445		
	Female	223117	220444	217916		
IFP	Male	7772	6001	5157		
	Female	6771	5368	4560		
Grand Total		764654	762790	760761		

Figure 2.4: An example of an XLS-PivotTable. The Pivot Table is an interactive feature of Excel that allows dynamic creation and modification of tables via a graphical user interface. It greatly enhances data exploration that can be done within Excel, though at first glance appears to be a significant barrier to analysis outside Excel. However, this is not the case, as it is possible to extract the underlying data using *Show Details* from the right-click menu. The data is *Pivot-Table-Student-Numbers-by-Age.xls* from the *Ministry of Education*.



July Roll Ye	Student	Student Ag	Student Gend	Ethnic grou	Territorial Authority Bo	Territorial Authorit
2004	2	Age 05	Male	Māori	Far North District	Far North District
2004	1	Age 11	Male	Māori	Hamilton City	Hamilton City
2004	6	Age 10	Male	Māori	Hamilton City	Hamilton City
2004	5	Age 07	Male	Māori	Far North District	Far North District
2004	3	Age 09	Male	Māori	Hamilton City	Hamilton City
2004	4	Age 08	Male	Māori	Far North District	Far North District
2004	1	Age 08	Male	Māori	Hamilton City	Hamilton City
2004	3	Age 09	Male	Māori	Far North District	Far North District
2004	3	Age 07	Male	Māori	Hamilton City	Hamilton City
2004	4	Age 10	Male	Māori	Far North District	Far North District
2004	1	Age 06	Male	Māori	Hamilton City	Hamilton City
2004	1	Age 11	Male	Māori	Far North District	Far North District
2004	4	Age 05	Male	Māori	Hamilton City	Hamilton City
2004	2	Age 12	Male	Māori	Far North District	Far North District
2004	2	Age 16	Male	Māori	Nelson City	Nelson City
2004	6	Age 15	Male	Māori	Nelson City	Nelson City
2004	2	Age 14	Male	Māori	Nelson City	Nelson City
2004	3	Age 13	Male	Māori	Nelson City	Nelson City
2004	1	Age 10	Male	Māori	Auckland- Howick	Auckland- Howick W
2004	2	Age 09	Male	Māori	Auckland- Howick	Auckland- Howick W
2004	1	Age 08	Male	Māori	Auckland- Howick	Auckland- Howick W

Figure 2.5: An example of the XLS-PivotTable from Figure 2.4 after clicking *Show Details* from the right-click menu. Note that the data is presented in long-form and could now be converted to a CSV for analysis in different software.

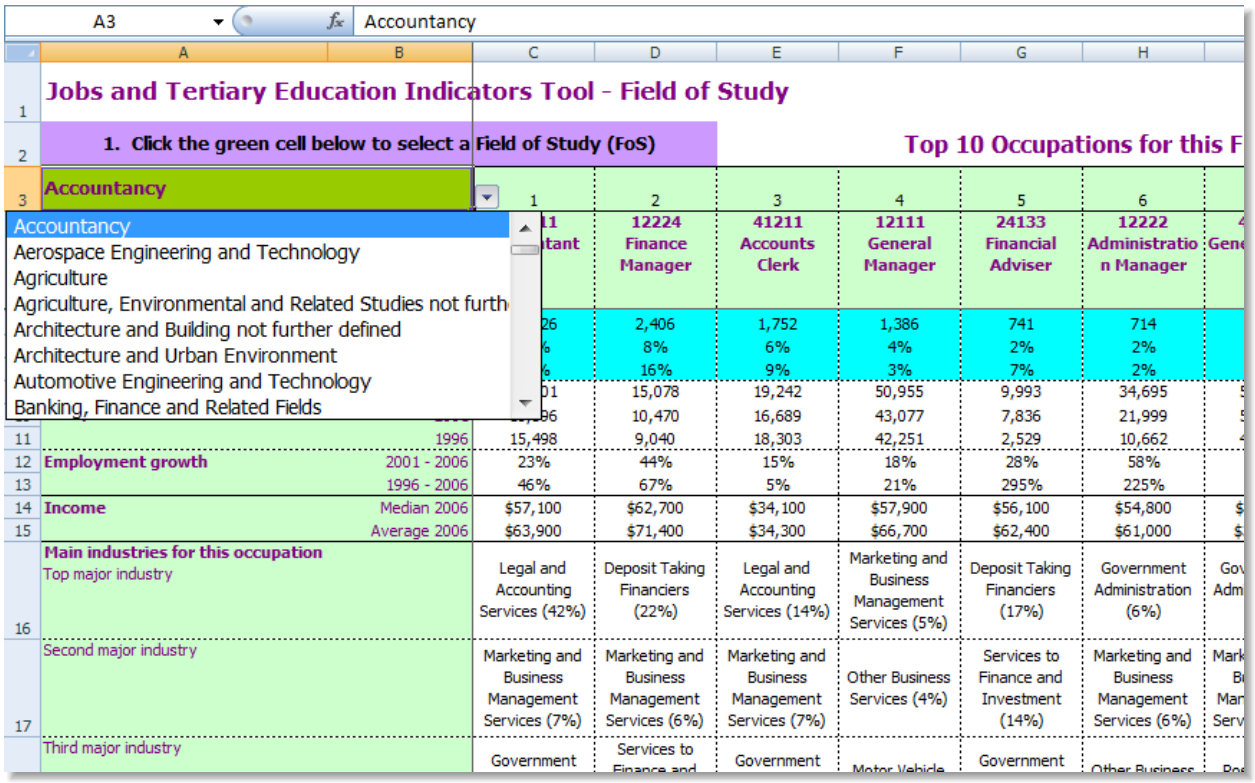


Figure 2.6: An example of an XLS-ReportTable. Much like an XLS-Table, these are relatively easy for a human brain to decipher. However, we classify an XLS file as ReportTable if there are substantial barriers to machine reading. Generally any data released as an XLS-ReportTable can only be used manually and presents significant barriers to any re-use, even by manual means. The data is *jtei-july-08.xls* from the *Department of Labour*.

HTML Table

Another format is to release the data as an *HTML Table* element. Like XLS files, HTML Tables could also be released in simple longform or in more complex structures, though practically when HTML Tables are used, they tend toward the former as it is not very suited for more complex structures. As the data is still stored in a structured format, there are several tools to convert HTML Tables to a different format, such as a CSV, for analysis in a variety of software tools. Though easy to share on a website, as HTML Tables do incur a conversion cost when re-used, the underlying data should also be released in a different downloadable format such as CSV.

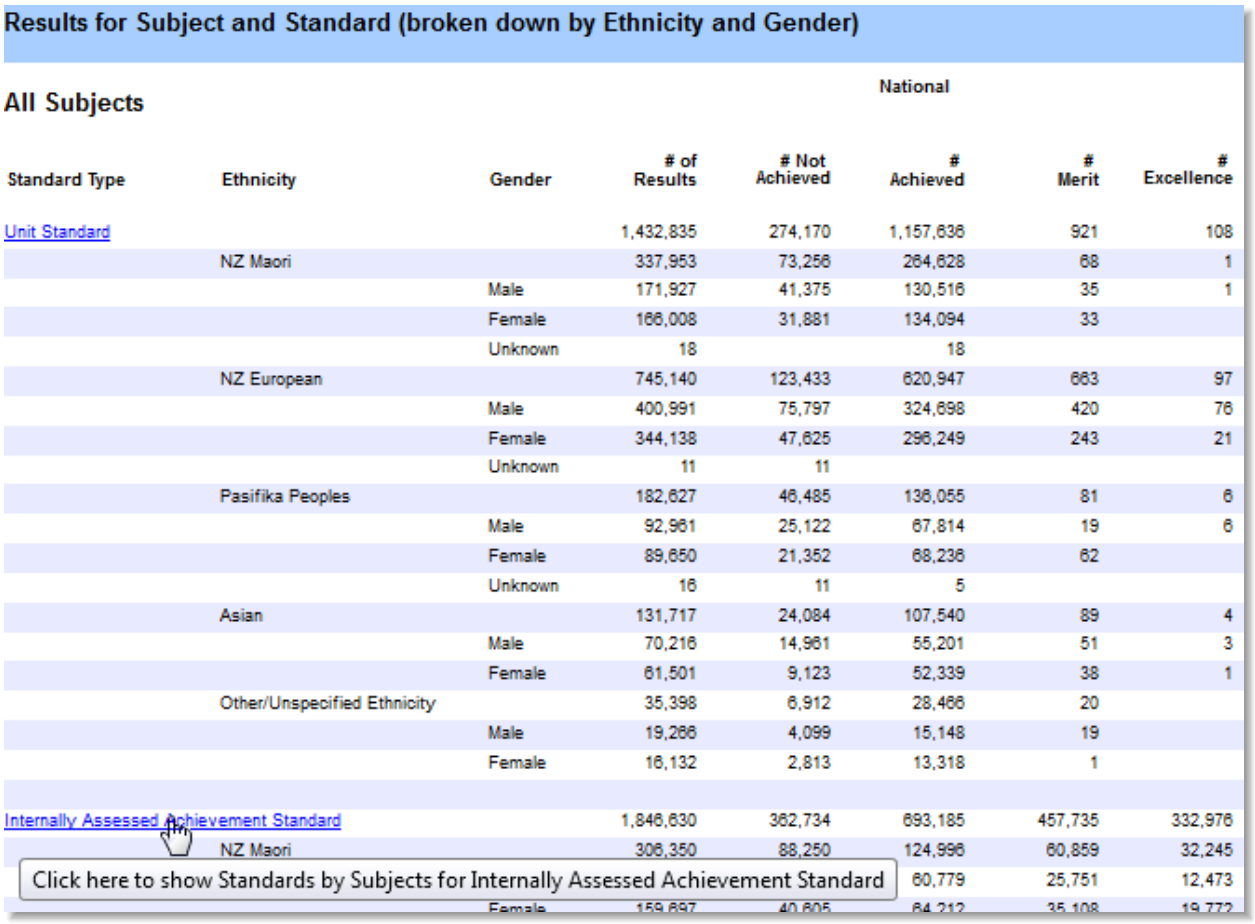
Income bands for salaries and wages, 2002 to 2011

2002		
Wage/Salary Income Band	Number of People	Wage/Salary Income ($000,000)
$1 - $1,000	153,900	65.1
$1,001 - $2,000	90,320	133.7
$2,001 - $3,000	73,640	183.9

Figure 2.7: An example of an HTML-Table. The data is *Income bands for salaries and wages, 2002 to 2011* found under the *Wage/salary distributions for individual customers* category from *Inland Revenue*.

PDF Table

Finally, the last format we cover is a table embedded inside a *PDF* document. These can barely be considered data as PDF documents do not retain any data structure, being purely a presentation format. Generally, any data released as a PDF can only be used manually and any re-use will be costly in time and effort. Data should never be released exclusively in a PDF format, any data included in a PDF document should be released separately in a proper data format.



Results for Subject and Standard (broken down by Ethnicity and Gender)

All Subjects

National

Standard Type	Ethnicity	Gender	# of Results	# Not Achieved	# Achieved	# Merit	# Excellence
Unit Standard			1,432,835	274,170	1,157,636	921	108
	NZ Maori		337,953	73,266	264,628	68	1
		Male	171,927	41,375	130,516	35	1
		Female	166,008	31,881	134,094	33	
		Unknown	18		18		
	NZ European		745,140	123,433	620,947	663	97
		Male	400,991	75,797	324,698	420	76
		Female	344,138	47,625	296,249	243	21
		Unknown	11	11			
	Pasifika Peoples		182,627	46,485	136,055	81	6
		Male	92,961	25,122	67,814	19	6
		Female	89,650	21,352	68,236	62	
		Unknown	16	11	5		
	Asian		131,717	24,084	107,540	89	4
		Male	70,216	14,961	55,201	51	3
		Female	61,501	9,123	52,339	38	1
	Other/Unspecified Ethnicity		35,398	6,912	28,466	20	
		Male	19,266	4,099	15,148	19	
		Female	16,132	2,813	13,318	1	
Internally Assessed Achievement Standard			1,846,630	362,734	693,185	457,735	332,976
	NZ Maori		306,350	88,250	124,996	60,859	32,245
					60,779	25,751	12,473
			159,697	40,605	64,212	35,108	19,772

Click here to show Standards by Subjects for Internally Assessed Achievement Standard

Figure 2.8: An example of an HTML-Table that does not use HTML Table code (instead utilising `div` and `span` for visual formatting). This means the data is not structured making it extremely difficult to extract the data. Luckily in this case, a link is provided (not shown) to download the data as an XLS-Table. The data is a *Results by Subject and Standard* report from the *New Zealand Qualifications Authority*.

2.2.5 Other Relevant Topics

Licensing

Without going into technical details, a standardised *Copyright-Licence*, such as one of the *Creative Commons* licences or the *GNU General Public License*, can specify a certain set of copyright conditions or waivers. By attaching such a licence to Open Data, the legalities of copyright become clearer, allowing a user to know exactly where they stand with regards to what they may do with this data.

The *Creative Commons* licences appears to be the most common licence used by New Zealand State Sector Open Data Sources, so a brief overview is provided. For more information visit the *Creative Commons* website[2]. The default CC licence is the 'baseline' licence. It in effect allows the user to freely copy, distribute or make use of the work, but still assigns ownership of the copyright to the original holder. A CC licence may attach additional conditions, such as:

by Attribution - must reference the original author in the manner specified.

nc Noncommercial - can only be used for noncommercial purposes.

nd No Derivative Works - a derivative work that includes major elements of the original cannot be made. Thus you can still copy and pass on the original to others, but you cannot make any changes, alterations or additions.

sa Share-alike - derivative works may be made, but they must have a licence identical to the original.

Thus, a **CC by** licence would allow free use, modification, distribution, etc, as long as the correct attribution is given. An alternative to using a Licence is to simply make it Public Domain, which waives any and all copyright.

Privacy

The relevant legislation in New Zealand is the Privacy Act 1993, which restricts anyone's collection, storage and use of data about an identifiable private individual. In essence, data on an identifiable individual can only be collected for a specific purpose, requires the person's permission, should (where possible) be taken directly from the person, should be stored securely, and should not be released to anyone else. This data should only be used for the purpose it was originally collected for (which was communicated to the person) and once it serves its purpose, it should be securely disposed (Penk and Tobin 2010).

[2] http://creativecommons.org/

These restrictions apply not only to the data collector, but extend to anyone who subsequently gains possession of the private data. This creates a problem: if Open Data is released that is in fact breaching privacy, any user of this 'Open' data becomes liable for any privacy breaches themselves; blame cannot be shifted to the original data provider. Fortunately, the New Zealand Privacy Commission takes a corrective role rather than a punitive one, thus in most cases the 'liability' will be minor (e.g. required to apologise to affected individuals), especially if the breach was unintentional.

Obtaining permission directly from any affected individuals, to effectively waive their privacy rights relating to the particular data collected, is allowed and can be used to avoid privacy issues. However in the case of Open Data, often the users are separate and distinct from the data collector and provider, thus obtaining permission is rarely possible.

N.B. It may be of interest to know that private information collected purely for 'domestic' use (for personal affairs) is not bound to the restrictions of the Privacy Act 1993.

2.3 Overview of Open Data in New Zealand

2.3.1 Overview of the Overview

Motivation

We give an overview of Open Data in New Zealand for two reasons:

1. It is useful to understand what Open Data is actually out there in practice, by who and in what form. It is also helpful to know how this data might be acquired.

2. Restricting to New Zealand limits the scope to something that is manageable.

The focus of the overview will be on the State Sector, as this is the most abundant source of Open Data. In particular, the New Zealand Government is making a push for an Open and Transparent Government and we can expect to see even more data coming from the State Sector. Related initiatives include:

Declaration on Open and Transparent Government (New Zealand Government ICT, 2011) - "Building on New Zealand's democratic tradition, the government commits to actively releasing high value public data [non-personal and unclassified data.]... To support this declaration, the government asserts that the data and information it holds on behalf of the public must be open, trusted and authoritative, well managed, readily available, without charge where possible, and reusable, both legally and technically."

NZGOAL (New Zealand Government Open Access and Licensing)(New Zealand Government ICT, 2010) - "The Government wants to encourage... individuals and organisa-

> tions to be able to leverage State Services agencies' data stores for their own, agencies' and others' benefit... In essence, NZGOAL... sets out a series of open licensing and open access principles"

The government has also released the **2012 Report On Agency Adoption of the New Zealand Declaration on Open and Transparent Government** (New Zealand Government ICT, 2012b). Of particular interest is paragraph 23 which highlights what the various departments and agencies consider to be barriers to releasing 'high value public data'.

- Restrictive licensing terms imposed by third parties
- Issues with data quality and inconsistent data
- A lack of data standards, which causes confusion with data format
- A lack of resources to address the above barriers
- A shift in culture is required
- Considerable time is required to analyse the risks of releasing data for re-use
- A perceived lack of data to release
- A lack of analysis of what information is of interest to consumers

As supplement to the report, the government also released information providing details of Current and Future Data Releases. However, during the Literature Review data releases were encountered that were not found in either of these lists, so they are not a comprehensive source of all data releases.

The Method

To gain the overview I desire, I examined the organisations listed as *Public Service departments* on the State Services Commission's website (State Services Commission, 2012). For each organisation the website was examined to gain some understanding of how they release data. A small sample of the discovered data was then examined in greater depth to gain an understanding of any difficulties that might be encountered in obtaining and trying to make active use the data. The organisations were then contacted to confirm my findings and to better understand the rationale behind how they release data.

I classified the organisations based on the amount of data they release, as it is reasonable to expect that organisations that release more data will have better processes in place. Additionally some were classified as being Outside the Scope of this Literature Review.

Major Sources Those identified as releasing significant amounts of data regularly and often have an extensive history of data releases.

Minor Sources Those identified as releasing data regularly, or have otherwise released a non-trivial amount of data, but not enough to be considered *Major*.

Not Sources No significant datasets found.

Outside Scope Some organisations only release geospatial data, and these are classified as Outside the Scope of this Literature Review. The complexity and specialised nature of geospatial data means it is a topic deserving of its own thesis.

Situation Report

As we might expect, those sources that release large amounts of data tended to have better data releases than those that release smaller amounts. In particular, those classified as *Major* generally have a mandate to release data, often being *Tier 1 Statistics* (see Section 2.3.2), and thus have good reason to focus on releasing data. In contrast, the other organisations must focus first on their primary objectives, with releases of data being a secondary, even tertiary objective, one with no additional funding attached.

However, a few organisations took the recent government initiatives as an opportunity to focus on releasing data, and some activity has been seen in attempting to overcome the lack of resources with a cross-agency solution. LINZ led a working group to develop a business case for a shared Open Data Service (ODS) (not to be confused with the LINZ Data Service, a geospatial data service). The ODS Agency Demand Survey (Land Information New Zealand, 2012b) noted:

> At present agencies provide their released information through a variety of mechanisms ranging from website content though to custom designed database applications. This variability means that the data can be difficult for users to find, access and use, thereby limiting its potential reuse and subsequent benefit to the New Zealand economy.
>
> The goal of the ODS is to reduce duplication and get the best return for investment in the production of systems that enable government agencies to release high value, non-personal data, and to provide a portal or portals where users can seamlessly access information from a number of different agencies.

As noted in the above quote, such a shared effort will standardise how the data is released across many state organisations, making it much easier to find and obtain data. Unfortunately, the initiative never made it past the consultation stage. Land Information New Zealand (2012a) noting:

> A survey of local, regional and central government agencies found that there was a strong level of interest in using an ODS if it were available. The same survey also revealed that agencies were unwilling to prioritise investment in the development of the service in a constrained fiscal environment.
>
> The working group concluded that while there was strong interest in the concept of an ODS, the inability to obtain funding to carry out the development meant that it was not possible to produce a viable business case for an ODS.

Unfortunately this means the current situation, where there is a lack of standardisation across the different agencies, continues. Each agency has its own website layout that can make it difficult to find the data, with some agencies even lacking a listing of all data releases. A data aggregator, like *data.govt.nz*, alleviates this problem and is currently one of the best methods for finding data (see Subsection 2.3.3 for details). Once found, it is often the case that the data is not in a format ideal for re-use (see Subsection 2.2.4 for more on data formats), requiring extra work to make use of the data.

Though change and improvements are on the horizon, the current situation poses many challenges in making use of Open Data released by state organisations.

2.3.2 Statistics New Zealand

http://www.stats.govt.nz/

Statistics New Zealand Tatauranga Aotearoa is a government department and New Zealand's national statistical office. We're New Zealand's major source of official statistics and leader of the Official Statistics System.

Official Statistics in New Zealand

Statisphere (2012a), "New Zealand's official statistics portal" managed by Statistics New Zealand (StatsNZ) "on behalf of all government departments" provides a good explanation of Official Statistics in New Zealand.

> Official statistics are defined in section 2 of the Statistics Act 1975. They are statistics derived by government departments from:
>
> - statistical surveys
> - administrative and registration records and other forms and papers that are published regularly, or planned to be published regularly, or could be published regularly.
>
> Statistical survey is defined in the Statistics Act as: "a survey of undertakings, or of the public of New Zealand, whereby information is collected from all

> persons in a field of inquiry or from a sample, by a Government Department with the authority of this Act or any other Act, for the purpose of processing and summarising by appropriate statistical procedures and publishing the results of the survey in some statistical form".
>
> Statistics New Zealand is New Zealand's national statistical office. Statistics New Zealand is the leader of the Official Statistics System and is the major producer of official statistics in New Zealand.
>
> The Government Statistician, who is also the Chief Executive of Statistics New Zealand... [provides] direction to the Official Statistics System and engaging other government departments to build shared ownership, minimise duplication, and maximise reuse of data.

Thus Statistics New Zealand is very important on matters relating to policy, as well as being an important source of data.

Tier 1 Statistics

In 2003 a review conducted by StatsNZ identified a set of key official statistics; these are known as Tier 1 statistics (Statisphere, 2012c) which...

- are essential to critical decision-making
- are of high public interest
- meet expectations of impartiality and statistical quality, in accordance with the Tier 1 principles and protocols
- require long-term data continuity
- allow international comparability
- meet international statistical obligations.

A principle and protocol of particular interest is **Protocol 5: Release Practices**. The following are excerpts from the *Principles and Protocols for Producers of Tier 1 Statistics* (Statisphere, 2012b):

- Tier 1 statistics producers will ensure equality of access. Statistics are presented in an understandable manner and are widely disseminated.
- Release of Tier 1 statistics is by the Chief Executive of the producing agency, according to a calendar of release dates published at least six months in advance.

- Information dissemination practices are responsive to the needs of users.
- Statistics are released in a variety of formats that meet the needs of users.
- Tier 1 statistics producers endeavour to integrate and harmonise their publications and products with users' needs and give them easy access to related statistics through common gateways or interlinked websites.
- Tier 1 statistics producers respond to changing expectations about access to outputs. Formats, media, content and support materials are regularly reviewed and are modified to meet users' current and future needs.
- The cost of accessing Tier 1 statistics is minimal.
- Catalogues and directories are readily available so that potential users know where and what statistics are available. A list of official statistics is available at www.statisphere.govt.nz
- Tier 1 statistics producers provide facilities (by electronic and/or print media) to ensure easy, user-friendly access to statistics for everyone, including regular, professional users as well as casual users and the interested public.
- Regularly recurring statistical releases are delivered in consistent formats. The format is sufficiently flexible to allow explanations of the data as they vary between periods.

Unfortunately, but understandably, practical application of these protocols appears to be very difficult, most likely due to limited resources, and these protocols seem more a goal than a reality.

Statistics New Zealand as a Data Source

The majority of data is released through one of the online tools, of which there are currently three: Infoshare, Table Builder and NZ.Stat.

NZ.Stat is a new tool released near the end of 2012 that replaces Table Builder. Table Builder is scheduled to be decommissioned at the end of February 2013. NZ.Stat is also intended to replace Infoshare in 2014.

The general idea behind all the online tools are the same. They are graphical user interfaces that are used to:

1. Narrow down user query to a specific category.
2. Provide a breakdown of variables related to the chosen category.
3. Generate a table based on the selection.

4. The output is a table tailored more for human consumption and is not presented in Longform.

At this stage, it would appear there is no API support for any of the online tools, though NZ.Stat may have this capability in the future "We will also be providing more functionality, such as the ability to create your own custom queries that can be shared with colleagues, and machine-to-machine data transfer." (Statistics New Zealand, 2012)[3]

Some data is released via direct download, usually in an XLS format. The internal format of these files are variable, but they appear to be intended for direct consumption by human audiences (i.e. Table, not Longform). In some cases, the files mention that more data can be found via one of the tools, though output generated from Infoshare appeared to be considerably more aggregated than the data available via direct download.

2.3.3 data.govt.nz

data.govt.nz is a catalogue of Open Data released by New Zealand government agencies. It is administered by the Department of Internal Affairs and has official support through the New Zealand Declaration on Open and Transparent Government initiative.

Advantages:

- good coverage of data released by government agencies
- good search engine which includes some metadata (mainly in the form of tags and categories)
- generally much easier to search for a particular agency in *data.govt.nz* than it is to look through the agency's official website
- Has official backing from the New Zealand Declaration on Open and Transparent Government initiative.

Disadvantages:

- Not all open government data can be found on *data.govt.nz*
- A lot of metadata is incomplete or left unfilled, e.g. the RBNZ datasets are missing information for many of the categories, such as Contact Person, Email, Phone, Date of creation and Frequency of update.
- *data.govt.nz* does not hold the data themselves and often does not link directly to the data, leading instead to a webpage on the agency's official website, where it may or may not be obvious how the sought data should be obtained.

[3] As at Februrary 2015, an API still does not exist for NZ.Stat.

- Lacks a documented API to facilitate computerised calls to search and/or access metadata (though DIA plans for a blog post related to this at http://webtoolkit.govt.nz/).

It is worth noting that *data.govt.nz* is not the only data catalogue. Some others are:

Statisphere http://www.statisphere.govt.nz/ Serves as a listing of Tier 1 Statistics, though this feature is currently unavailable "pending a review and update of the content".

DigitalNZ http://www.digitalnz.org/ A catalogue of any *digital content*.

Open Data Catalogue http://cat.open.org.nz/ "an open, independent catalogue of Government and Local Body datasets.".

2.3.4 The Template

The data sources are presented using a standard format. This information is recorded in *NZDataSources.xml* and is later parsed by script to this Literature Review. As it is a structured document there are other possible ways of parsing or using this information. The template is as follows:

Name of Organisation

`URL to Organisation's homepage`
Organisation's Purpose, quoted from their homepage.

Comments An overview of the organisation.

Data Grab Pattern Step-by-step instructions for finding the data from main page URL, generally referring to the names of the links to follow.

Data Format The type of data formats found (see Subsection 2.2.4).

Related Information Where related information (see Subsection 2.2.3) can be found. The two types found were In-file (where the information is found in the same files as the data) and webpage (where the information is found on a webpage located near where the data is found).

Found on data.govt.nz Whether the data is also found on *data.govt.nz*. Can be TRUE, FALSE or Partial.

Copyright Licence The copyright licence attached to the data.

2.3.5 Major Sources

Ministry of Economic Development

http://www.med.govt.nz/

The Ministry of Economic Development's purpose is to create the conditions for businesses to succeed and New Zealanders to prosper.

Comments The Ministry of Economic Development (MED) is one of the organisations that have been merged under the Ministry of Business, Innovation and Employment (MBIE). Internally this merge is more or less complete though the merging of website content has been a low priority. Thus MBIE still makes MED-related releases through the former MED website.

The MED does not yet have a centralised process for data releases and instead provides data along two primary groups: Tourism and Energy. These two groups have different teams involved and hence have different processes.

The Tourism group classes users into three categories based on their data needs: 1) Requiring high level aggregate stats, 2) Requiring flexible cross tabulation (similar to Statistics NZ's Table Builder or Infoshare), 3) Microdata (eg respondent-level survey data complete with weights and replicate weights from standard error estimation). They see the greatest demand from category 2. Data under category 3 are provided to "bona fide researchers who asks and we are sure is not going to hurt themselves with it."

Some data is released via a third-party software called Infoview, which allows for some interactive visualisations and data exploration.

They face minimal privacy or confidentiality issues, thus the biggest constraints on releasing data are time and resources to communicate required information to users, including background necessary to understand the data.

The Energy group generally release all data unless there is a good reason not to. They currently have a number of informal procedures in place to identify potential risks of data release, and to check data quality.

All Energy data is released via Excel, in some cases using some non-trivial Excel features to present the data, such as in-cell indentation and Excel's grouping feature.

Data Grab Pattern Sectors & industries tab ->Energy data and modelling (or) Tourism research and data.

Some data not found this way, best found via data.govt.nz.

Data Format CSV, XLS-Report-Table, XLS-Table, HTML-Table

Found on data.govt.nz TRUE

Copyright Licence Crown Copyright, restrictions may apply, refer to http://www.med.govt.nz/help-support/about-this-site/legal-notices/copyright

Ministry of Education

http://www.minedu.govt.nz/

Building a world-leading education system that equips all New Zealanders with the knowledge, skills, and values to be successful citizens in the 21st century.

Comments The Ministry of Education releases its data through a separate website, Education Counts. Note that the New Zealand Qualifcations Authority also releases some data relating to Secondary Schools, but this data is collected as part of NZQA's operations and is often different from the data collected by the Ministry. The Ministry assesses secondary schooling achievement using NZQA data and non-NQF (National Qualifications Framework) data collected directly from schools.

The data is typically collected for administrative reasons, rather than for research purposes, though the latter does happen. Generally, the Ministry will publish all such data, using Education Counts for routine releases.

They carry out several data quality checks, including looking for results and trends out of the ordinary, checking with schools if necessary to confirm correctness of their data.

National Standards which have been introduced recently with a lot of surrounding controversy (in particular relating to potential misuse of the data by unfairly ranking schools) are currently catering to the needs of each individual school, and thus standardised national data collection is not possible. However, it is anticipated that this will eventually be possible.

The Ministry prefers to release data in tables with accompanying explanatory notes, rather than simple text. They also make use of XLS-PivotTables to allow end-user exploration of the data. Where applicable they use standardised terms and definitions, usually defined by Statistics New Zealand. They regularly receive data requests for specific breakdowns of routine data releases, which are answered where possible, subject to privacy and data quality issues.

Data Grab Pattern NZ Education ->Researchers ->Education Counts

Links to http://www.educationcounts.govt.nz/

Data Format XLS-Table, XLS-PivotTable, HTML-Table

Found on data.govt.nz TRUE

Copyright Licence Crown Copyright, but also "CC By", refer to http://www.educationcounts.govt.nz/help/privacy

Inland Revenue

http://www.ird.govt.nz/

We collect most of the revenue that government needs to fund its programmes. We also administer a number of social support programmes.

Comments The Inland Revenue Department (IRD), more commonly just "Inland Revenue", makes regular releases of tax statistics via an easy to use listing of data on their website. The data release pages themselves provide simple plots along with links to the data.

They also release some other data that are not on the tax statistics listing (as they are not tax statistics), such as the quarterly Customer Satisfaction and Perceptions Survey results or the Industry Benchmarks. These are however not found via data.govt.nz.

The Industry Benchmarks are created by Statistics New Zealand from data provided by the IRD, the results of which are published on the IRD website.

IRD has a formal 15 point process for releasing data. Broad steps involve quality control, peer-review, production of tables and graphs and an assessment of confidentiality and other disclosure procedures (including accompanying notes).

To encourage re-use of the data the IRD clearly releases all data under a Creative Commons licence (CC BY), and releases all data in both HTML and Excel formats.

Data Grab Pattern About us ->Research and tax statistics

(The sources below are not linked to in data.govt.nz)

Business & employers ->Industry benchmarks ->Find Your Industry Benchmarks

Data Format XLS-Table, HTML-Table

Found on data.govt.nz Partial (Only lists Tax Statistics)

Copyright Licence CC By

Ministry of Transport

http://www.transport.govt.nz/

The Ministry of Transport is the government's principal transport adviser, and the bulk of our work is in providing policy advice and support to Ministers.

Comments The Ministry of Transport releases a large volume of data in a reasonably easy to use format, primarily XLS or CSV. Though some data is only available as part of a PDF report, in personal correspondence these have been identified as mostly a legacy feature, and the Ministry intends to release such data in XLS or CSV in the future.

The Ministry considers the main cost to be in collecting data. Once collected, they do not consider the release process to be a major burden. They also possess more detailed data (unit-record level, with potential privacy issues) that may be available to researchers on request.

Data Grab Pattern Research

Data Format XLS-Table, HTML-Table, PDF-Report-Table

Found on data.govt.nz TRUE

Copyright Licence Crown Copyright, restrictions may apply, refer to http://www.transport.govt.nz/copyright-and-disclaimer/

Reserve Bank of New Zealand

http://www.rbnz.govt.nz/
The Reserve Bank of New Zealand is New Zealand's central bank. We promote a sound and dynamic monetary and financial system.

Comments The Reserve Bank of New Zealand (RBNZ) releases a significant amount of data often going back decades. They release data in two formats, HTML-Tables for recent data and XLS-Tables for historical data (which includes recent data).

They also release Key Graphs on topics of high public interest, such as Inflation, GDP or Unemployment. These graphs also have accompanying commentary and a link to the data (XLS).

The RBNZ maintains an Advance Release Calendar which provides details on when data is updated, or the update frequency (e.g. Daily).

Data Grab Pattern Statistics

Data Format XLS-Table, HTML-Table

Found on data.govt.nz TRUE

Copyright Licence Slightly conflicting. The Statistics page states: "Material published in our website Statistical tables may be used without restriction, but acknowledgement

would be appreciated." However, the copyright page states some restrictions: http://www.rbnz.govt.nz/0161308.html Most likely, the statement on the Statistics page is given precendent.

2.3.6 Minor Sources

Department of Building and Housing

http://www.dbh.govt.nz/

Our vision is for a building and housing market that delivers good quality homes and buildings for New Zealanders that contribute to strong communities and a prosperous economy.

Comments The Department of Building and Housing (DBH) is one of the organisations that have been merged under the Ministry of Business, Innovation and Employment (MBIE). Internally this merge is more or less complete though the merging of website content has been a low priority. Thus MBIE still makes Building and Housing related releases through the former DBH website.

As things stand, the website lacks a comprehensive listing of all data releases, and not all data releases are listed on data.govt.nz, presenting a significant barrier in finding the data.

The Department releases reports with accompanying commentary, graphs and data, but this data is not available separately making re-use difficult. They have however indicated that more data may be available upon request, e.g. they regularly provide raw rental bond data to those who request it.

They have indicated a desire to release their Key Indicator Reports (currently released as PDF reports) as 'live' spreadsheets, "so that interested people can access the raw data behind the graphs", though the integration into MBIE is likely to delay this development.

They have recently made open some data behind their Key Indicator Reports. Currently this only consists of some market rent data, but the data is in a very simple, machine-friendly format.

Quality assurance procedures exist for any data releases, including data quality checks and peer review of analysis.

Data Grab Pattern No Standard Pattern.

Data Format HTML-Table, PDF-Report-Table, XLS-Report-Table

Found on data.govt.nz Partial

Copyright Licence Crown Copyright, restrictions may apply, refer to http://www.dbh.govt.nz/disclaimer-and-copyright

Department of Conservation

http://www.doc.govt.nz/

This Department of Conservation (DOC) site has information about the protection of New Zealand's natural and historic heritage, how and where you can enjoy public conservation places and how to get involved in conservation.

Comments The Department of Conservation (DOC) has a history of releasing data even prior to official government policy on data releases, however there were no formal procedures or processes in place. While care was given when releasing a dataset, including considerations for using common formats for releases rather than what may be most convenient to use internally, these have been ad hoc. Their website management model has also meant that it was easy for data releases to be duplicated, as evidenced by the website currently having two independent but very similar listings of data (see DataGrabPattern).

Despite this, the data releases are in fairly good condition. Much of the data released are geospatial released through a third-party provider, Koordinates Limited. Other releases are provided as HTML-Tables, though a few appear to only be available as part of PDF reports.

DOC had actively participated in the Open Data Service initiative (see Section 2.3.1), which they had intended to use for future data releases.

They are now investing in a federated infrastructure for a biodiversity data exchange. This is intended to enable sharing across agencies (both nationally and internationally) and also enable access by the public.

Data Grab Pattern About DOC ->Role ->Visitor statistics & research

Publications ->About DOC ->Role ->Maps & statistics

The two pages above have some duplicates between them, but one may have something the other doesn't.

Data Format HTML-Table, PDF-Report-Table

Found on data.govt.nz Partial

Copyright Licence Crown Copyright, restrictions may apply, refer to http://www.doc.govt.nz/footer-links/copyright/

Department of Corrections

http://www.corrections.govt.nz/
The Department of Corrections enforces the sentences and orders of the criminal courts and parole board. Corrections improves public safety by ensuring sentence compliance and works to reduce re-offending by providing offenders with rehabilitation programmes, education and job training.

Comments The Department of Corrections does not release a lot of data, but they do make regular quarterly releases for "community sentences and orders statistics" and "prison statistics". These releases are as HTML-Tables and come with some simple graphs.

They are currently in discussions with the Department of Internal Affairs on how best to support the Open Data initiative, and are likely to adopt what DIA prescribe.

Data Grab Pattern About Us ->Facts and statistics

Data Format HTML-Table

Found on data.govt.nz FALSE

Copyright Licence Crown Copyright, restrictions may apply, refer to http://www.corrections.govt.nz/utility-navigation/disclaimer-and-copyright.html

Ministry of Health

http://www.health.govt.nz/
The Government's principal advisor on health and disability: improving, promoting and protecting the health of all New Zealanders

Comments The Ministry of Health releases data primarily to satisfy various requirements placed upon the Ministry. This data may be classified under Tier 1 Statistics, or are released for other accountability purposes, and hence have various requirements on how they are released. Typically these data releases are for direct human consumption and are often not ideal for machine-reading.

Internal discussions are underway to adopt the Government's Open Data Initiative, including working towards more machine readable formats, but as no additional resources were provided it may take a while (potentially several years) to incorporate the Open Data processes into existing projects.

Data Grab Pattern Health statistics

Data Format XLS-Report-Table

Found on data.govt.nz TRUE

Copyright Licence Crown Copyright, restrictions may apply, refer to http://www.health.govt.nz/about-site/copyright

Department of Internal Affairs

http://www.dia.govt.nz/
The Department of Internal Affairs serves and connects people, communities and government to build a safe, prosperous and respected nation.

Comments The Department of Internal Affairs (DIA) does not release significant amounts of data but is notable for being the department that administers data.govt.nz, webtoolkit.govt.nz and ict.govt.nz, which provides the platform, tools and policies that will no doubt affect the open data releases of other government organisations.

The DIA does not maintain its own independent listing of data, instead opting to utilise data.govt.nz's ATOM feed to generate a list of their data releases, which is then presented to the user. This is good to see as it reduces duplication of effort (DIA does not have to spend additional resources to update its own independent listing) and makes it more likely that their listing on data.govt.nz is more comprehensive and up-to-date.

Unfortunately this method of using data.govt.nz to generate a listing automatically has not been suggested to the other agencies, though our contact with DIA considered it to be a good idea. It would be good to see the method adopted by other agencies in the future.

Being the agency that administers ict.govt.nz, DIA also has extensive documentation on data release processes which are available publically at ict.govt.nz:

http://ict.govt.nz/library/3191296DA%20-%20Process%20for%20Prioritisation%20and%20Release%20of%20High%20Value%20Public%20Data%20for%20Reuse_0.pdf

http://ict.govt.nz/library/3191296DA-Open-Data-Identification-Prioritisation-and-Planning-Template-Worksheet_0.xls

Data Grab Pattern Data and Statistics

Data Format XLS-Report-Table, XLS-Table, CSV

Found on data.govt.nz TRUE

Copyright Licence Licence generally denoted in the data.govt.nz listing (usually "CC By"). Where it's not specified, it is Crown Copyright, restrictions may apply, refer to http://www.dia.govt.nz/Legal-Copyright-Index

Department of Labour

http://www.dol.govt.nz/
To grow New Zealand's economy and improve the quality of lives through a high-performing labour market and immigration system.

Comments The Department of Labour is one of the organisations that have been merged under the Ministry of Business, Innovation and Employment (MBIE). Internally this merge is more or less complete though the merging of website content has been a low priority. Thus MBIE still makes Labour related releases through the former Department of Labour's website.

The website links to a Statistics page which leads to a Health and Safety Statistics page that contains Workplace fatalities, serious harm and prosecutions data. These datasets cannot be found via data.govt.nz, though data.govt.nz links to several hidden pieces of data (often contained in reports) that were not found by following the Statistics link from the website.

No formal policies existed for the release of data. Such work was underway when the Department was merged into MBIE and has now ceased, presumably to be replaced by a unified policy from MBIE.

Data Grab Pattern Statistics

Data Format XLS-Report-Table, HTML-Table

Found on data.govt.nz Partial

Copyright Licence Crown Copyright, restrictions may apply, refer to http://www.dol.govt.nz/common/copyright.asp

Ministry for Primary Industries

http://www.mpi.govt.nz/
Our vision is to grow and protect New Zealand. We do this by: maximising export opportunities for the primary industries; improving sector productivity; increasing sustainable resource use; and protecting New Zealand from biological risk.

Comments The Ministry for Primary Industries (MPI) releases data collected from their own activities, and also data sourced from Statistics New Zealand. Some of these are available only as a part of a report making re-use difficult. For some reports (e.g. Farm Monitoring Reports) the data is available separately, but only on request.

The Ministry used to release data in both Excel and CSV format, but have since stopped as they found the process to be too cumbersome. CSV format is still available, but only on request.

Most of MPI's data releases are Tier 1 Statistics and follow the guidelines specified for Tier 1 Statistics (see Section 2.3.2). The Ministry intends to incorporate the NZGOAL framework (see Section 2.3.1) for future releases.

Data Grab Pattern News & Resources ->Statistics & Forecasting

Data Format XLS-Table

Found on data.govt.nz Partial

Copyright Licence Unusual copyright, restrictions may apply, refer to http://www.mpi.govt.nz/Copyright.aspx

Ministry of Social Development

http://www.msd.govt.nz/
Our Ministry is all about helping to build successful individuals, and in turn building strong, healthy families and communities.

Comments While the Ministry collects large amounts of data as part of their core purpose (of Social Development), much of the data collected is sensitive information and collected to fulfil the Ministry's primary objectives. Any release of such data to the wider public is incidental to their core purpose, which contributes to the manner in which they release data. The Ministry of Social Development mostly releases data as reports rather than in formats more conducive to re-use.

The Ministry is aware of some of the limitations in how they currently release the data but their focus is on fulfilling their core purpose, not on the public release of collected data. They receive hundreds of Official Information Act requests every year, in addition to requests from parliament, media and researchers, which provides some idea of what sort of data is in demand, but the potential for misuse of data limits what they can release as Open Data.

The Ministry has very detailed data quality procedures which was audited by PricewaterhouseCoopers in 2006 and is periodically audited by Audit NZ.

Their listing on data.govt.nz reveals statistics released by StudyLink (which is one of the services provided by the Ministry), and links to their Annual Statistical Reports, but does not link to their primary listing of Statistics found via the *Data Grab Pattern* steps below.

Data Grab Pattern Publications & Resources ->Statistics

Data Format HTML-Table, PDF-Report-Table, DOC-Report-Table

Found on data.govt.nz Partial

Copyright Licence Crown Copyright, restrictions may apply, refer to http://www.msd.govt.nz/about-msd-and-our-work/tools/copyright-statement.html

2.3.7 Not Sources

Ministry of Business, Innovation and Employment

http://www.mbie.govt.nz/

A new 'parent' Ministry for: the Department of Building and Housing, the Ministry of Economic Development, the Department of Labour and the Ministry of Science and Innovation.

Formed on 1 July 2012, internally the merge is more or less complete but the merging of website content has been a low priority. MBIE still communicates mainly via the old websites and because of this, we cover the websites of the above agencies separately.

Crown Law Office

http://www.crownlaw.govt.nz/

No Comments.

Ministry for Culture and Heritage

http://www.mch.govt.nz/

No Comments.

Ministry of Defence

http://www.defence.govt.nz/

No Comments.

Education Review Office

http://www.ero.govt.nz/
ERO is not a true data provider (instead see Ministry of Education for education data), but the School Reports http://www.ero.govt.nz/Early-Childhood-School-Reports contain some broad "About the School" information presented as an HTML-Table. Contains information like: Decile, School roll, Gender composition, Ethnic composition.

Ministry of Foreign Affairs and Trade

http://mfat.govt.nz/
The Ministry appears to be policy focused and do not release trade-related data.

Government Communications Security Bureau

http://www.gcsb.govt.nz/
No Comments.

Ministry of Justice

http://www.justice.govt.nz/
Some very old data might be found via data.govt.nz, recent data appears to be held by StatsNZ.

Ministry for Maori Development

http://www.tpk.govt.nz/en/
Only 'data' is the Iwi Directory. This is grouped by region but these region names are in Maori with no English translation provided, reducing accessibility.

However, a simple but effective interactive map is also provided, which can also be used to navigate to the region sub-pages (still only has Maori names, but can at least go off the geography).

Beyond the directory, there is also information on each Iwi's 'population' (found in the individual Iwi's sub-page).

New Zealand Customs Service

http://www.customs.govt.nz/
Somewhat disappointing that no data is apparently available to the public. Might have been interesting to see data such as a table of goods with annual tariff revenue. Note StatsNZ does have some data on imports and exports, such as goods by value and volume by country, but does not appear to have customs specific data (e.g. tariffs, goods seized, etc.).

Ministry of Pacific Island Affairs

http://www.mpia.govt.nz/
Some simple statistics available on their website, but no real data. Statistics are likely derived from StatsNZ data.

Department of the Prime Minister and Cabinet

http://www.dpmc.govt.nz/
No Comments.

Serious Fraud Office

http://www.sfo.govt.nz/about
No Comments.

State Services Commission

http://www.ssc.govt.nz/
List of state organisations http://www.ssc.govt.nz/state_sector_organisations

Ministry of Women's Affairs

http://www.mwa.govt.nz/
Only 'data' found is Students' Occupational Choice Study, Dunedin, Auckland, which is mostly a report but includes some HTML-Tables of data.

It also has a Statistics page http://www.mwa.govt.nz/women-in-nz/stats that links to some other agencies (Ministry of Social Development, StatsNZ, Department of Labour).

2.3.8 Outside Scope

Ministry for the Environment

http://www.mfe.govt.nz/index.html
Only release geospatial data and all data appears to be released via a third-party provider, Koordinates Limited (also used by the Department of Conservation).

Land Information New Zealand

http://www.linz.govt.nz/
Geospatial data only, see http://www.geodata.govt.nz/.

2.4 Conclusion

The value of Open Data can be significant but realising this value is dependent on how easy the Open Data is to find, access and use. This Literature Review has examined features necessary for *Good* Open Data - clear copyright terms, easy to find, easy to access, good data structure and accompanying documentation. It is also clear from Protocol 5 of *Principles and Protocols for Producers of Tier 1 Statistics* that the government agrees these, and other considerations, are important. Unfortunately, resource constraints mean few government agencies are able to fully meet these requirements.

While a lot of Open Data is out there, and while improvements are on the horizon, there are currently many challenges in making use of this Open Data. This also means there are many opportunities to create tools that will make this data more useful, and hence help unlock the benefits of this Open Data.

Chapter 3

TableToLongForm

3.1 Introduction . . . 42
3.1.1 The Problem with Tables . . . 43
3.1.2 Core Concepts . . . 44
3.2 How to Use . . . 48
3.2.1 Obtaining TableToLongForm . . . 48
3.2.2 Loading the Data . . . 48
3.2.3 Calling TableToLongForm . . . 48
3.2.4 Manual Conversion . . . 48
3.2.5 Diagnostics . . . 49
3.2.6 Extending TableToLongForm . . . 50
3.3 Vocabulary . . . 51
3.4 Implementation Details . . . 53
3.5 Identification . . . 53
3.5.1 Purpose . . . 53
3.5.2 Key Challenges . . . 55
3.5.3 Limitations and Future Work . . . 59
3.6 Discern Parentage . . . 60
3.6.1 Purpose . . . 60
3.6.2 Key Challenges . . . 63
3.6.3 Limitations and Future Work . . . 67
3.7 Reconstruction . . . 68
3.8 Summary . . . 70

3.1 Introduction

This chapter deals with concepts that, while widely understood, are poorly defined. This opening section attempts to more clearly define and distinguish the relevant concepts, to aid in the discussion that follows. I begin by defining the *Basic Format*. It is basic both for its properties and the fact that it is a format commonly used by technical users (such as data analysts). It can be defined as follows:

- A matrix with `i` rows and `j` columns.
- Each column, `j`, represents a variable and is often named.
- Each row, `i`, represents a single observation.

Users of R will be familiar with this format as the `data.frame`:

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Above is a `data.frame` in R, with variables `Sepal.Length`, `Sepal.Width`, etc. and where each row represents a single observation. The Basic Format is well understood by data-processing software and there exist many tools to make it even easier to manipulate and deal with this data. The R package `tidyr` (Wickham, 2014) (which calls the Basic Format *tidy* data) can be used to reformat the data, for instance `gather` can be used to combine multiple variables/columns into fewer columns (also known as *Long-Form data*), while `spread` can do the opposite, expanding a single variable/column into multiple columns (also known as *Wide-Form data*). The R package `dplyr` (Wickham and Francois, 2015) can be used to manipulate the data, examples being to take a subset based on some conditions using `filter` and to group and take summaries (such as means) with `group_by` and `summarise`.

A requirement for these packages however, is that the data must first be in the Basic Format. Once in this format, the data is much easier to handle, and thus much more useful. Unfortunately, while conducting the Literature Review in the previous chapter it was discovered that a significant proportion of Open Data released by the New Zealand State Sector are only released as *hierarchical tables*[1], and not in the Basic Format. Tables are a

[1] Henceforth be referred to as a *Table* with a capital T.

method of presenting data for human consumption, with the data grouped in a hierarchical manner using one or more grouping variables. This grouping information is typically represented with visual clues, which are sufficient for humans but present significant barriers to using the data with computer software. The tools mentioned above, such as `dplyr`, are only useful once the data is in the Basic Format, meaning conversion of Table data is necessary to make it more useful. As this conversion task is often time-consuming, it seems like a useful task to create a tool that can automatically convert these Tables into the Basic Format. Naming such a tool poses some challenges, as there is no agreed upon naming convention for concepts like the Basic Format or a Table. I have chosen the terms **Table** and **LongForm** Dataframe, as they carry a balance of recognition (more so than terms such as 'basic', or 'tidy') while being close enough proxies of the underlying concepts.

3.1.1 The Problem with Tables

An example of a Table is shown in Figure 3.1. For such a Table, the computer will be unable to easily read in the data due to the difficulty in finding all information related to a piece of data. Take the number `876` in cell (5, 9) for instance; to collect all the information linked to that number we must look at cell (5, 1) for the time period (`2007Q4`), cell (4, 9) for the data heading (`Total Labour Force`), cell (3, 2) for the ethnic category (`European Only`) and cell (2, 2) for the gender category (`Male`). Note that, aside from the time period and the data heading, the other information related to cell (5, 9) were neither in the same row nor the same column as the data value. The human brain can interpret the positional cues to understand the hierarchy fairly easily, the computer requires a lot more work.

Preparing such data for use would normally require a non-trivial time investment to restructure the data in a manner that can be machine read and used. If such preparatory work was done manually, such work will have to be repeated multiple times as the data is updated. In some cases the data will be spread across multiple files, which means that much more preparatory work. Even if the work is scripted, small changes in the format can easily throw a wrench into the works and break it. All of this quickly adds up to a significant time cost to make use of data released in Tables.

This trend of releasing data in a human-friendly, but machine-unfriendly Table can be seen in many organisations, both in private and public sector, in New Zealand and overseas. TableToLongForm is a tool that automatically converts a family of Tables to a machine-readable form, once so converted the user is free to use their favourite tool to make full use of the data. Figure 3.2 shows the result of automatic conversion with TableToLongForm.

	1	2	3	4	5	6	7	8	9	10	11
1	Labour Force Status by Sex by Sing/Comb Ethnic Group (Qrtly−Mar/Jun/Sep/Dec)										
2		Male									
3		European Only								Maori Only	
4		Persons Em	Persons Un	Not in Labo	Working Ag	Labour Forc	Unemploym	Employmen	Total Labou	Persons Em	Persons Un
5	2007Q4	856	20	280	1,156	76	2	74	876	71	6
6	2008Q1	863	25	284	1,172	76	3	74	888	69	8
7	2008Q2	850	26	281	1,157	76	3	74	876	67	6
8	2008Q3	840	30	286	1,155	75	3	73	869	72	9
9	2008Q4	855	30	275	1,159	76	3	74	884	76	8
10	2009Q1	845	35	279	1,160	76	4	73	880	75	8
11	2009Q2	832	35	280	1,146	76	4	73	866	74	10
12	2009Q3	813	42	290	1,146	75	5	71	856	71	11
13	2009Q4	831	40	277	1,148	76	5	72	871	72	14
14	2010Q1	822	36	283	1,142	75	4	72	859	72	11
15	2010Q2	825	40	290	1,155	75	5	71	865	72	14
16	2010Q3	837	31	287	1,155	75	4	72	868	70	14
17	2010Q4	838	40	277	1,155	76	4	73	878	71	14
18	2011Q1	830	37	281	1,148	76	4	72	866	71	14
19	2011Q2	839	41	279	1,159	76	5	72	880	67	10
20	2011Q3	830	35	280	1,145	76	4	72	865	70	13
21	2011Q4	842	35	278	1,154	76	4	73	877	69	13
22	2012Q1	843	43	283	1,169	76	5	72	886	72	11
23	2012Q2	837	38	296	1,172	75	4	71	875	66	11
24	2012Q3	833	38	298	1,169	74	4	71	871	67	13
25	2012Q4	833	41	298	1,173	75	5	71	874	63	12
26	2013Q1	832	36	295	1,162	75	4	72	868	70	12
27	Table information:										
28	Units:										
29	Persons Employed in Labour Force: Number, Magnitude = Thousands										
30	Persons Unemployed in Labour Force: Number, Magnitude = Thousands										
31	Not in Labour Force: Number, Magnitude = Thousands										
32	Working Age Population: Number, Magnitude = Thousands										
33	Labour Force Participation Rate: Percent, Magnitude = Units										
34	Unemployment Rate: Percent, Magnitude = Units										
35	Employment Rate: Percent, Magnitude = Units										
36	Total Labour Force: Number, Magnitude = Thousands										
37	Footnotes:										
38											
39	Symbols:										
40	.. figure not available										

Figure 3.1: An example of a hierarchical Table. The Table is of the Labour Force Status data (Statistics New Zealand, 2013) and in total spans 240 columns. The Table is too large to be immediately useful for humans, and yet cannot be manipulated easily with a computer.

3.1.2 Core Concepts

Despite the many variations of Tables that data is released in, very few have instructions on how to understand the Table. It is assumed that a human user will be able to read it without instruction, and this assumption often proves to be true. So then the question arises, how is it that a human is able to decipher the hierarchical relationships encoded into the Tables?

A human reader can make use of a variety of features to gather information, such as:

Prior knowledge - If the Table contains two cells named *Auckland* and *New Zealand*, a human may deduce that since Auckland is a city of New Zealand, Auckland is

	1	2	3	4	5	6	7	8	9	10	11
1				Persons En	Persons Un	Not in Labo	Working Ag	Labour For	Unemploym	Employmen	Total Labou
2	Male	European C	2007Q4	856	20	280	1,156	76	2	74	876
3	Male	European C	2008Q1	863	25	284	1,172	76	3	74	888
4	Male	European C	2008Q2	850	26	281	1,157	76	3	74	876
5	Male	European C	2008Q3	840	30	286	1,155	75	3	73	869
6	Male	European C	2008Q4	855	30	275	1,159	76	3	74	884
7	Male	European C	2009Q1	845	35	279	1,160	76	4	73	880
8	Male	European C	2009Q2	832	35	280	1,146	76	4	73	866
9	Male	European C	2009Q3	813	42	290	1,146	75	5	71	856
10	Male	European C	2009Q4	831	40	277	1,148	76	5	72	871
11	Male	European C	2010Q1	822	36	283	1,142	75	4	72	859
12	Male	European C	2010Q2	825	40	290	1,155	75	5	71	865
13	Male	European C	2010Q3	837	31	287	1,155	75	4	72	868
14	Male	European C	2010Q4	838	40	277	1,155	76	4	73	878
15	Male	European C	2011Q1	830	37	281	1,148	76	4	72	866
16	Male	European C	2011Q2	839	41	279	1,159	76	5	72	880
17	Male	European C	2011Q3	830	35	280	1,145	76	4	72	865
18	Male	European C	2011Q4	842	35	278	1,154	76	4	73	877
19	Male	European C	2012Q1	843	43	283	1,169	76	5	72	886
20	Male	European C	2012Q2	837	38	296	1,172	75	4	71	875
21	Male	European C	2012Q3	833	38	298	1,169	74	4	71	871
22	Male	European C	2012Q4	833	41	298	1,173	75	5	71	874
23	Male	European C	2013Q1	832	36	295	1,162	75	4	72	868
24	Male	Maori Only	2007Q4	71	6	28	105	73	8	68	77
25	Male	Maori Only	2008Q1	69	8	31	108	71	10	64	77
26	Male	Maori Only	2008Q2	67	6	27	100	73	8	67	73
27	Male	Maori Only	2008Q3	72	9	31	111	72	11	64	80
28	Male	Maori Only	2008Q4	76	8	28	113	75	10	67	84
29	Male	Maori Only	2009Q1	75	8	36	120	70	10	63	84
30	Male	Maori Only	2009Q2	74	10	33	117	72	12	63	84
31	Male	Maori Only	2009Q3	71	11	36	118	70	13	60	82
32	Male	Maori Only	2009Q4	72	14	33	118	72	16	60	85
33	Male	Maori Only	2010Q1	72	11	35	118	70	14	61	83

Figure 3.2: An example of a LongForm dataframe. This is the Table in Figure 3.1 after automatic conversion with TableToLongForm and in total spans 660 rows. While it is still not immediately useful for humans, all related information can be found in the same row or column, making the data much easier to manipulate with a computer.

probably a Child of New Zealand. Likewise if that same Table also contained a cell named *Australia*, they may deduce that New Zealand and Australia are on the same level of the hierarchy.

Formatting - Bold, emphasis, bigger size, etc. all hint towards a higher place in the hierarchy.

Layout - Parents tend to be positioned to the left and/or above their Children. This positioning could be achieved by using physically different cells (as in Figure 3.1), or by indentation (e.g. by having white space leading the cell contents).

Many of these features communicate the same hierarchical information, introducing redundancy and aiding in the understanding of the Table even if some features are not understood.

Understanding all of these features may require writing some form of an Artificial Intelligence, but some of these can be tapped for information using relatively simple algorithms.

TableToLongForm makes use of positional cues, which would be a subset of the layout-method of communication. In particular, TableToLongForm uses encoding with physically different cells, which is information that can be accessed quite easily through an algorithmic process in R.

Consider the Table fragment in Figure 3.3, showing row labels from NZQA Scholarships data (New Zealand Qualifications Authority, 2012). A human reader can quickly understand that the hierarchy of the labels goes thus:

1. Subject (Accounting, Biology)
2. Ethnicity (NZ Maori, NZ European)
3. Gender (Male, Female)

Notice that to the **right** of the Subject headings, we have empty cells. We also have empty cells **below** the Ethnicity headings. The core algorithms that form the backbone of TableToLongForm's hierarchy deciphering process are the *By Empty Right* and *By Empty Below* algorithms, both of which make use of positional cues encoded through physically different cells, specifically, empty cells to the right of, or below, a label cell.

As seen in Figure 3.4, by identifying the cells with empty cells to the right, we can sub-divide the row labels into smaller chunks that are the children of the empty-right cells. Within these chunks, we can identify the cells with empty cells below, and sub-divide again, as in Figure 3.5. This process of iteratively dividing and conquering the hierarchy is core to the deciphering process of TableToLongForm.

How is this is accomplished with algorithms? Here is the simplified process:

ByEmptyRight Check if the most upper-left cell has empty cells to its right.

> **TRUE** Check for any other cells that also have empty cells to the right. For all cells with empty cells to the right, grab the contents and sub-divide. For each sub-divided Table, call **ByEmptyRight**.
>
> **FALSE** Call **ByEmptyBelow**.

ByEmptyBelow Find all empty cells in the left-most column. Collect the non-empty cells and grab the cell contents[2]. Use the empty cells below to sub-divide the Table. For each sub-divided Table, call **ByEmptyRight**.

[2]Noting that in a ByEmptyBelow pattern, it is possible for parents to have a single child, in which case they will not have any empty cells below. Thus we must collect all non-empty cells, not just the cells that are empty below.

	1	2
1	Accounting	
2	NZ Maori	Male
3		Female
4	NZ European	Male
5		Female
6	Biology	
7	NZ Maori	Male
8		Female
9	NZ European	Male
10		Female

	1	2
1	Accounting	
2	NZ Maori	Male
3		Female
4	NZ European	Male
5		Female
6	Biology	
7	NZ Maori	Male
8		Female
9	NZ European	Male
10		Female

Figure 3.3: Row Labels excerpted from NZQA Scholarships data displaying both Empty Right (red) and Empty Below (green) patterns.

	1	2
1	Accounting	
2	NZ Maori	Male
3		Female
4	NZ European	Male
5		Female
6	Biology	
7	NZ Maori	Male
8		Female
9	NZ European	Male
10		Female

	1	2
1	Accounting	
2	NZ Maori	Male
3		Female
4	NZ European	Male
5		Female
6	Biology	
7	NZ Maori	Male
8		Female
9	NZ European	Male
10		Female

Figure 3.4: Using Empty Right to segment the labels into the children of the Subject headings.

	1	2
1	Accounting	
2	NZ Maori	Male
3		Female
4	NZ European	Male
5		Female
6	Biology	
7	NZ Maori	Male
8		Female
9	NZ European	Male
10		Female

	1	2
1	Accounting	
2	NZ Maori	Male
3		Female
4	NZ European	Male
5		Female
6	Biology	
7	NZ Maori	Male
8		Female
9	NZ European	Male
10		Female

Figure 3.5: Using Empty Below to segment the labels into the children of the Ethnic headings.

3.2 How to Use

3.2.1 Obtaining TableToLongForm

TableToLongForm is implemented in R (R Core Team, 2014), and has been released as a Package. It can be obtained via CRAN[3], or from the official website[4]. As with all other R Packages, once installed it must be loaded with the command `library(TableToLongForm)`.

3.2.2 Loading the Data

TableToLongForm's preferred argument is a `matrix` of mode `character`. If a `data.frame` is supplied instead, it is coerced to a `matrix` with a warning. Empty cells should be classed as `NA` for correct operation of the algorithms. Currently TableToLongForm does not distinguish between missing values and empty space, both are treated as `NA` values.

As the Labour Force Status data used in Figure 3.1 classifies missing values as `".."`, we must ensure R correctly reads these, in addition to empty cells, as `NA` values.

```
1 LabourForce = as.matrix(
2   read.csv("StatsNZLabourForce.csv",
3            header = FALSE, na.strings = c("", "..")))
```

3.2.3 Calling TableToLongForm

If the Table can be recognised by TableToLongForm a simple call to `TableToLongForm` with just a single argument is all that is needed. `TableToLongForm` has additional optional arguments used primarily for diagnostic purposes.

```
1 LabourForce.converted = TableToLongForm(LabourForce)
```

3.2.4 Manual Conversion

For comparison the code for manual conversion of the table is provided below. We note after careful observation of the data that:

- There are 3 gender categories: `Male`, `Female` and `Total Both Sexes`, each 80 columns in width.
- There are 10 ethnic categories, each a consistent 8 columns in width.
- The data are found in rows 5 to 26.

[3] http://cran.at.r-project.org/
[4] https://www.stat.auckland.ac.nz/ joh024/Research/TableToLongForm/

Armed with this knowledge, we can write the above code that, with some trial and error and cross-checking of results, will successfully convert the Table to a LongForm. This code is fairly compact and efficiency-wise will beat TableToLongForm (which by necessity is more complicated, and thus computationally expensive). However, it took a non-trivial investment of time to code and test the results (Approx. 30 minutes), is mostly useless for any other Table, and if any of the many strong assumptions it makes are violated (e.g. a new row of data is added), it breaks and requires fixing, which means even more time consumed. All this work and hassle to just *read in the data* in a useful format.

```
LFout = NULL
chYear = LabourForce[5:26, 1]
for(Gender in 0:2)
  for(Ethni in 0:9){
    chGender = LabourForce[2, 2 + Gender * 80]
    chEthni = LabourForce[3, 2 + Ethni * 8]
    LFout = rbind(LFout,
      cbind(chGender, chEthni, chYear,
      LabourForce[5:26,
        2 + Gender * 80 + (Ethni * 8):((Ethni + 1) * 8 - 1)])
    )
  }
colnames(LFout) = c("Gender", "Ethnicity", "Time.Period",
   LabourForce[4, 2:9])
```

3.2.5 Diagnostics

The primary limitation of TableToLongForm is that the function will be a black box to most users. After running the function on a Table, the user will either be given back a `data.frame` with no easy way of verifying if the result is correct, or be confronted with an error with little idea of what went wrong. Based on ad hoc tests conducted so far, TableToLongForm will either succeed, or fail catastrophically in a manner that is easily recognised as utter failure. However methods for verifying correct operation (or to understand failures) would be desirable. TableToLongForm currently does quite poorly in providing useful diagnostics output, but some methods are available.

The simplest method currently available is to examine the additional output returned when TableToLongForm is called with the optional argument `fulloutput = TRUE`. This will return the 'final product' of TableToLongForm's algorithms in the form of `IdentResult`, `rowplist` and `colplist` (see Section 3.3). Unfortunately this output has two key limitations. First, it is not obvious from this output what went wrong (or if nothing went wrong),

requiring some detective work to piece together the evidence. Second, if anything did go wrong, the user still does not know *why*.

The option with the potential to provide the most information is calling `TableToLongForm` with the optional argument `diagnostics = TRUE`, which will write diagnostic output to a file, printing key variables at each major stage of the conversion process. This output can thus be used to track TableToLongForm's progress as it works to convert the Table, enabling fairly accurate assessment of where exactly it went wrong. Unfortunately understanding this output requires familiarity with the workings of the code and is unlikely to be of much use to anyone other than the author.

3.2.6 Extending TableToLongForm

It is possible to extend TableToLongForm by writing custom algorithms and registering them with TableToLongForm. Custom algorithms can be added to the Identification (Section 3.5) and the Discern Parentage (Section 3.6) stages of the conversion process. Refer to the respective sections for the necessary requirements of such algorithms.

Registration of new algorithms are accomplished via a call to `TTLFaliasAdd`, which has the following arguments:

Type e.g. IdentPrimary

Fname the name of the Function/Algorithm

Falias the alias for the Function/Algorithm, which is used for the call to `TableToLongForm`

Author (optional) name of the author of the algorithm

Description (optional) a short description of the purpose of the algorithm

The `.R` file that contains the function(s) should also contain this registration line. Then during an R session, we can load the TableToLongForm package, then source the custom algorithm's `.R` file to register the new algorithm(s).

We can check if this is successful by then calling `TTLFaliasList`. The output with no additional modules is as follows:

```
1 > TTLFaliasList()
2 ==Type: IdentPrimary==
3 Name: IdentbyMostCommonBoundary
4 Alias: combound
5 Author: Base Algorithm
6 Description: Default IdentPrimary algorithm
7
```

```
==Type: IdentAuxiliary==
Name: IdentbySequence
Alias: sequence
Author: Base Algorithm
Description: Search for fully numeric row labels (e.g. Years)
    that were misidentified as data

==Type: ParePreCol==
Name: ParePreColMismatch
Alias: mismatch
Author: Base Algorithm
Description: Correct for column labels not matched correctly
   over data (label in a different column to data)

Name: ParePreColMisaligned
Alias: misalign
Author: Base Algorithm
Description: Correct for column labels not aligned correctly
   over data (parents not positioned on the far-left,
   relative to their children in the row below)

Name: ParePreColMultirow
Alias: multirow
Author: Base Algorithm
Description: Merge long column labels that were physically
   split over multiple rows back into a single label
```

If your new algorithms were successfully registered, they should appear on this list, and the aliases for the new algorithms can be used during a call to `TableToLongForm`. The default arguments for TableToLongForm are:

```
TableToLongForm(..., IdentPrimary = "combound",
  IdentAuxiliary = "sequence", ParePreRow = NULL,
  ParePreCol = c("mismatch", "misalign", "multirow"))
```

3.3 Vocabulary

We must define some vocabulary necessary to better understand TableToLongForm.

	1	2	3	4	5	6
1			Column 1	Column 2	Column 3	Column 4
2	Row Parent1	Row Child1	10	20	30	40
3		Row Child2	11	21	31	41
4	Row Parent2	Row Child1	12	22	32	42
5		Row Child2	13	23	33	43

Figure 3.6: An example Table that will be used to define the vocabulary.

matFull The entire Table.

matRowLabel Blue region.

matColLabel Green region.

matData Red region.

IdentResult A list containing these two elements:

label - a vector of the rows or columns where the labels are found.

data - a vector of the rows or columns where the data are found.

rowplist/colplist The row and column parentage lists. These are nested `list` objects that represent all the hierarchical relationships in the Table. TableToLongForm also provides a custom `print` function for the class `plist`, which results in much more informative printed output.

For this Table:

```
IdentResult = list(rows = list(label = 1,   data = 2:5),
                   cols = list(label = 1:2, data = 3:6))
```

```
> rowplist
+ Row Parent1 (1, 1)
- + Row Child1 (1, 2)
- + Row Child2 (2, 2)
+ Row Parent2 (3, 1)
- + Row Child1 (3, 2)
- + Row Child2 (4, 2)
```

```
> colplist
+ Column 1 (1, 1)
+ Column 2 (2, 1)
+ Column 3 (3, 1)
+ Column 4 (4, 1)
```

Note that the cell positions given in the `plist` refer to positions within `matRowLabel` and `matColLabel`.

3.4 Implementation Details

The complete conversion process from a Table to a LongForm Dataframe requires more than just the core algorithms described in Core Concepts (Subsection 3.1.2). There are 3 stages in the conversion process:

Identification - Identify where in the Table the data is found and where the accompanying labels are, while ignoring any extraneous information we do not want.

Discern Parentage - Understand the hierarchical structure (the *parentage*) of the row and column labels.

Reconstruction - Use the information from the first two stages to reconstruct the Table as a Dataframe.

In the following sections these stages will be explained in greater depth. Details of the actual code is sparse - for such details refer to the Literate Document available on the website[5].

3.5 Identification

3.5.1 Purpose

The purpose of Identification is to identify the rows and columns in which the labels and data values can be found, while ignoring any extraneous information we do not want. This task can be surprisingly difficult, requiring many fringe-case checks and exception handling. The current core identification algorithm searches for blocks (rectangular regions) of numbers in the supplied Table. This region is assumed to contain the data and from it the algorithm infers the locations of the corresponding labels. The result, after some extra work to handle fringe-cases and the like, is the `IdentResult`, a list which specifies the rows and columns in which the labels and the data can be found.

If TableToLongForm fails to correctly identify the correct rows and columns, it is possible to manually specify the `IdentResult` as an argument. Even for cases where the `IdentResult` must be manually specified, the work required for the conversion with TableToLongForm will be strictly less than for a manual conversion as we would need the same information, and more, to convert manually.

Identification is comprised of two classes of algorithms. Custom algorithms can be used to extend functionality (see Subsection 3.2.6).

[5] https://www.stat.auckland.ac.nz/~joh024/Research/TableToLongForm/

Ident Primary contain Primary Identification algorithms, of which one is chosen when calling `TableToLongForm`. They should take a single argument, `matFull`. They should return an `IdentResult`.

Default: `IdentPrimary = "combound"`

Ident Auxiliary contain Auxiliary Ident algorithms, of which any combination, in any order, can be chosen when calling `TableToLongForm`. They are called after the Primary algorithm, to refine the `IdentResult`. They should take two arguments, `matFull` and `IdentResult`. They should return an `IdentResult`.

Default: `IdentAuxiliary = "sequence"`

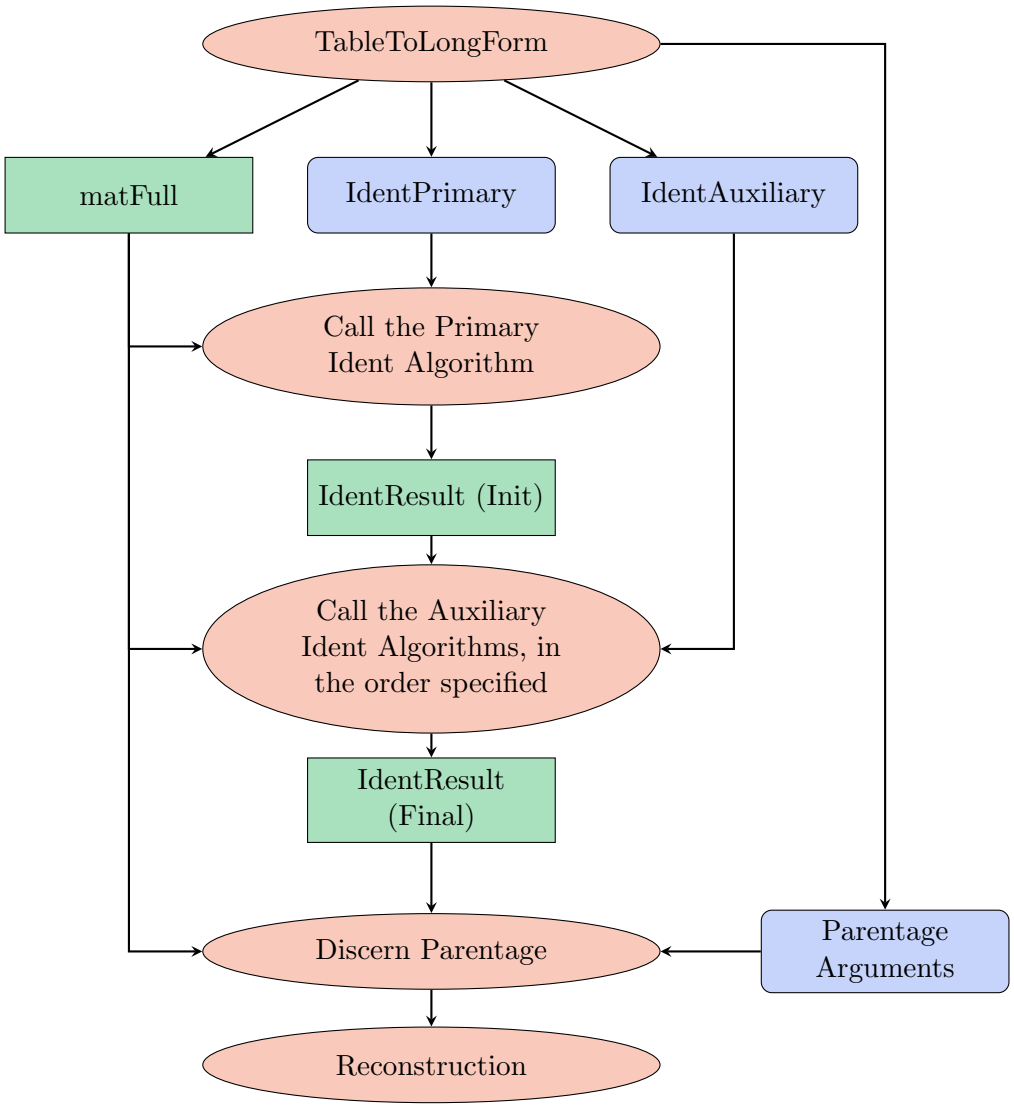


Figure 3.7: A breakdown of the Identification step of the workflow. The arguments IdentPrimary and IdentAuxiliary specify the respective algorithms to use.

3.5.2 Key Challenges

A common feature of Tables is that they do not just contain the data, but extra information as well (e.g. titles and metadata). For instance in Figure 3.1, we see that row 1 is a title, while from rows 27 onwards we have metadata. This is useful information for the human user, but must be ignored for the purposes of automatic conversion. Thus when TableToLongForm is identifying where the required information is in the Table, it must pick out the right features while ignoring the rest. The following Figures are examples of actual datasets released as Tables. The challenges they present are described in the Figure captions.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Series, GDP(P), Nominal, Actual, Sector of ownership (Annual−Mar)												
2		Gross Domestic Product − product measure											
3		Market				Non−Market				Total Market and Non−Market			
4		Private	Central Gov	Local Gove	All Sectors	Private	Central Gov	Local Gove	All Sectors	Private	Central Gov	Local Gove	All Sectors
5	1972	5,200	568	155	5,923	58	660	100	818	5,258	1,227	255	6,740
6	1973	5,993	649	169	6,811	68	738	118	924	6,062	1,387	287	7,735
7	1974	7,013	723	183	7,919	78	880	136	1,093	7,091	1,604	318	9,013
8	1975	7,636	750	202	8,588	90	1,049	164	1,303	7,727	1,798	366	9,891
9	1976	8,526	840	231	9,597	108	1,264	202	1,574	8,634	2,104	433	11,171
10	1977	10,450	1,179	295	11,924	130	1,431	213	1,774	10,580	2,611	508	13,698
11	1978	11,469	1,398	362	13,230	150	1,713	247	2,111	11,620	3,111	610	15,341
12	1979	12,729	1,685	394	14,808	180	2,105	306	2,591	12,909	3,789	700	17,398
13	1980	14,707	2,050	501	17,258	216	2,434	368	3,018	14,923	4,484	869	20,275
14	1981	16,947	2,358	559	19,864	261	3,042	439	3,742	17,207	5,401	998	23,606
15	1982	20,751	2,770	655	24,175	316	3,619	542	4,477	21,067	6,388	1,197	28,652
16	1983	23,525	3,184	759	27,468	332	3,975	606	4,913	23,858	7,158	1,365	32,381
17	1984	26,577	3,644	800	31,021	327	4,081	653	5,061	26,903	7,725	1,453	36,081
18	1985	30,530	3,924	850	35,304	361	4,270	720	5,351	30,891	8,194	1,570	40,655
19	1986	35,104	4,741	1,008	40,853	418	5,012	815	6,245	35,522	9,752	1,823	47,098
20	1987	40,167	6,189	1,160	47,515	501	6,235	896	7,631	40,667	12,423	2,055	55,146
21	1988	43,613	6,957	1,192	51,762	582	6,885	961	8,427	44,195	13,842	2,153	60,190
22	1989	47,816	7,052	1,364	56,231	663	7,410	1,076	9,149	48,479	14,461	2,440	65,380
23	1990	51,170	6,267	1,364	58,800	722	7,710	1,153	9,585	51,892	13,977	2,517	68,385
24	1991	53,613	4,970	1,412	59,995	801	7,912	1,228	9,941	54,415	12,882	2,640	69,936
40	2007	127,531	4,857	2,735	135,123	3,131	16,257	2,105	21,493	130,662	21,114	4,840	156,616
41	2008	138,127	5,554	2,962	146,644	3,317	17,569	2,298	23,184	141,444	23,124	5,260	169,828
42	2009	137,240	5,783	3,195	146,219	3,742	19,067	2,470	25,279	140,982	24,850	5,666	171,498
43	2010	139,525	6,090	3,252	148,866	3,942	20,007	2,605	26,553	143,466	26,097	5,856	175,419
44	Table information:												
45	Units:												
46	$, Magnitude = Millions												
47	Footnotes:												
48													
49	Symbols:												
50	.. figure not available												
51	C: Confidential												
52	E: Early Estimate												
53	P: Provisional												
54	R: Revised												
55	S: Suppressed												
56	Status flags are not displayed												

Figure 3.8: This is a Table of New Zealand GDP Data (Statistics New Zealand, 2013) and is an example of a (comparatively) good Table as it is consistent in format with no weird features. However, the row labels are numbers, posing some complications as the main Identification algorithm looks for numbers to identify the data. In cases such as this, TableToLongForm uses pattern recognition (e.g. sequences of numbers such as 1972, 1973, 1974... are more likely to be labels than data) to attempt to identify such numbers to be labels, but this process is far from perfect.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Scholarship Entries and Results by Gender and Ethnicity (Broken down by Decile)												
2													
3				Decile 1–3									Decile 4–7
4			# of	#	#		#	# Not	#	#		# of	#
5	Results		Entries	Absent	SNA		Assessed	Achieved	Scholarship	Outstanding		Entries	Absent
6													
7	All Subjects		714	148	13		553	462	81	10		6,482	1,772
8													
9	Accounting		22	4	0		18	16	2	0		156	41
10	NZ Maori	Male	2	1	0		1	1	0	0		2	1
11		Female	0	0	0		0	0	0	0		7	2
12	NZ Europea	Male	2	0	0		2	1	1	0		51	13
13		Female	3	0	0		3	2	1	0		44	12
14		Unknown	0	0	0		0	0	0	0		0	0
15	Pasifika Pe	Male	2	0	0		2	2	0	0		3	0
16		Female	6	2	0		4	4	0	0		4	2
17	Asian	Male	5	0	0		5	5	0	0		29	4
18		Female	2	1	0		1	1	0	0		15	7
19	Other/Unsp	Male	0	0	0		0	0	0	0		0	0
20		Female	0	0	0		0	0	0	0		1	0
21													
22	Agricultural		0	0	0		0	0	0	0		15	3
23	NZ Maori	Male	0	0	0		0	0	0	0		0	0
24		Female	0	0	0		0	0	0	0		0	0
25	NZ Europea	Male	0	0	0		0	0	0	0		10	1
26		Female	0	0	0		0	0	0	0		5	2
27		Unknown	0	0	0		0	0	0	0		0	0
28	Pasifika Pe	Male	0	0	0		0	0	0	0		0	0
29		Female	0	0	0		0	0	0	0		0	0
30	Asian	Male	0	0	0		0	0	0	0		0	0
31		Female	0	0	0		0	0	0	0		0	0
32	Other/Unsp	Male	0	0	0		0	0	0	0		0	0
33		Female	0	0	0		0	0	0	0		0	0
34													
451	Technology		7	2	0		5	5	0	0		99	45
452	NZ Maori	Male	1	0	0		1	1	0	0		2	1
453		Female	0	0	0		0	0	0	0		5	0
454	NZ Europea	Male	2	1	0		1	1	0	0		26	11
455		Female	0	0	0		0	0	0	0		34	15
456		Unknown	0	0	0		0	0	0	0		0	0
457	Pasifika Pe	Male	0	0	0		0	0	0	0		2	1
458		Female	1	1	0		0	0	0	0		2	1
459	Asian	Male	2	0	0		2	2	0	0		16	10
460		Female	0	0	0		0	0	0	0		9	5
461	Other/Unsp	Male	1	0	0		1	1	0	0		1	0
462		Female	0	0	0		0	0	0	0		2	1
463													
464													
465													
466	Report Parameters												
467	Show gend	yes											
468	Show ethni	yes											
469	Show decile	yes											
470	Scholarship	All Subjects											

Figure 3.9: This is a Table of NZQA Scholarship Data (New Zealand Qualifications Authority, 2012) and is the original motivating dataset that resulted in the creation of TableToLongForm. The Table contains a title and metadata in a form typical of Tables, but also displays some clear dividing rows and columns. This seems like an attractive way to attempt Identification of the features of this Table, but closer inspection reveals that these dividers are not very informative. Consider for instance the dividing columns 6 and 11. Though appearing to be the same, column 6 is only a (meaningless) sub-divider, while column 11 is a divider that splits the Decile groups. This problem can be overcome with pattern recognition, but as dividing rows and columns are an uncommon feature in Tables, such an algorithm is of limited value. The current method of Identification searches for blocks of numbers, and this is mostly successful for this Table. There is a slight problem with column headings due to misaligned column label, e.g. `Decile 1-3` in (3, 4) should be in (3, 3) to be aligned with `# of Entries`. This problem is resolved not in Identification, but in the next stage, Discern Parentage.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	TABLE 1: (a) Number of 16−24 year olds Not in Education, Employment or Training (NEET) and (b) associated Confidence Intervals by Region												
2													
3				(a) Number of 16−24 year olds NEET									
4		Quarterly LFS series		England	North East	North West	Yorks & Hu	East Midlan	West Midla	East of Eng	London	South East	South West
5		Q2	2000	652,000	61,000	92,000	72,000	60,000	75,000	53,000	127,000	69,000	42,000
6		Q3	2000	750,000	57,000	113,000	87,000	69,000	89,000	64,000	131,000	91,000	49,000
7		Q4	2000	629,000	48,000	97,000	72,000	57,000	84,000	55,000	93,000	74,000	48,000
8		Q1	2001	667,000	53,000	114,000	77,000	58,000	82,000	61,000	100,000	77,000	45,000
9		Q2	2001	650,000	42,000	112,000	77,000	53,000	75,000	62,000	111,000	68,000	49,000
10		Q3	2001	774,000	50,000	132,000	84,000	60,000	79,000	75,000	140,000	97,000	58,000
11		Q4	2001	660,000	46,000	110,000	75,000	50,000	79,000	60,000	116,000	83,000	42,000
12		Q1	2002	699,000	51,000	114,000	83,000	59,000	88,000	61,000	111,000	83,000	48,000
13		Q2	2002	703,000	45,000	117,000	84,000	55,000	85,000	61,000	123,000	90,000	42,000
14		Q3	2002	795,000	49,000	115,000	111,000	58,000	96,000	71,000	143,000	96,000	56,000
15		Q4	2002	660,000	49,000	100,000	74,000	55,000	77,000	69,000	113,000	75,000	49,000
16		Q1	2003	730,000	51,000	99,000	95,000	54,000	90,000	78,000	122,000	89,000	52,000
17		Q2	2003	709,000	51,000	107,000	87,000	59,000	88,000	66,000	112,000	94,000	46,000
18		Q3	2003	813,000	59,000	134,000	96,000	64,000	97,000	71,000	136,000	102,000	54,000
19		Q4	2003	666,000	46,000	104,000	78,000	51,000	80,000	50,000	122,000	96,000	39,000
20		Q1	2004	680,000	51,000	95,000	76,000	53,000	83,000	52,000	136,000	97,000	38,000
21		Q2	2004	700,000	48,000	110,000	72,000	60,000	85,000	62,000	133,000	84,000	46,000
22		Q3	2004	837,000	57,000	124,000	101,000	76,000	98,000	68,000	166,000	91,000	57,000
23		Q4	2004	738,000	54,000	113,000	81,000	69,000	84,000	59,000	142,000	79,000	58,000
24		Q1	2005	735,000	52,000	109,000	89,000	59,000	79,000	58,000	133,000	92,000	63,000
25		Q2	2005	770,000	59,000	117,000	102,000	59,000	84,000	63,000	139,000	85,000	61,000
26		Q3	2005	880,000	68,000	122,000	110,000	65,000	102,000	69,000	159,000	109,000	76,000
27		Q4	2005	827,000	58,000	112,000	108,000	69,000	90,000	75,000	146,000	107,000	62,000
28		Q1	2006	803,000	50,000	115,000	95,000	76,000	91,000	73,000	137,000	103,000	62,000
29		Q2	2006	852,000	51,000	125,000	107,000	73,000	100,000	87,000	143,000	105,000	61,000
30		Q3	2006	969,000	51,000	138,000	117,000	78,000	113,000	91,000	191,000	114,000	75,000
31		Q4	2006	806,000	44,000	120,000	94,000	65,000	95,000	76,000	149,000	93,000	69,000
32		Q1	2007	823,000	47,000	124,000	97,000	71,000	113,000	75,000	121,000	104,000	71,000
33		Q2	2007	825,000	48,000	130,000	92,000	65,000	103,000	83,000	132,000	108,000	63,000
34		Q3	2007	906,000	51,000	145,000	106,000	81,000	110,000	91,000	137,000	120,000	67,000
35		Q4	2007	772,000	45,000	120,000	82,000	70,000	93,000	72,000	123,000	113,000	56,000
36		Q1	2008	799,000	52,000	134,000	74,000	68,000	98,000	83,000	128,000	100,000	62,000
37		Q2	2008	839,000	54,000	135,000	102,000	72,000	103,000	74,000	134,000	108,000	56,000
38		Q3	2008	986,000	65,000	156,000	121,000	83,000	125,000	82,000	152,000	130,000	72,000
39		Q4	2008	850,000	54,000	137,000	94,000	67,000	106,000	80,000	126,000	123,000	63,000
40		Q1	2009	924,000	55,000	151,000	111,000	70,000	125,000	83,000	123,000	129,000	76,000
41		Q2	2009	950,000	66,000	154,000	123,000	76,000	130,000	81,000	113,000	140,000	68,000
42		Q3	2009	1,064,000	73,000	181,000	129,000	92,000	123,000	90,000	145,000	144,000	87,000
43		Q4	2009	888,000	66,000	142,000	112,000	72,000	104,000	86,000	119,000	115,000	73,000
44		Q1	2010	921,000	61,000	149,000	124,000	80,000	121,000	85,000	109,000	114,000	79,000
45		Q2	2010	868,000	57,000	133,000	116,000	71,000	104,000	82,000	116,000	116,000	72,000
46		Q3	2010	1,023,000	69,000	164,000	124,000	83,000	116,000	102,000	138,000	142,000	86,000
47		Q4	2010	934,000	62,000	140,000	112,000	66,000	125,000	94,000	128,000	127,000	81,000
48		Q1	2011	927,000	63,000	137,000	113,000	73,000	131,000	98,000	104,000	119,000	88,000
49		Q2	2011	991,000	66,000	169,000	130,000	81,000	115,000	101,000	134,000	122,000	74,000
50		Q3	2011	1,181,000	71,000	202,000	164,000	84,000	137,000	121,000	158,000	158,000	86,000
51		Q4	2011	969,000	57,000	155,000	138,000	82,000	120,000	89,000	126,000	126,000	76,000
52		Q1	2012	960,000	62,000	160,000	139,000	82,000	116,000	86,000	121,000	110,000	86,000
53		Q2	2012	986,000	72,000	150,000	142,000	74,000	128,000	82,000	130,000	118,000	90,000
54		Q3	2012	1,038,000	68,000	147,000	141,000	83,000	136,000	88,000	152,000	135,000	87,000
55		Q4	2012	890,000	56,000	118,000	111,000	85,000	113,000	80,000	132,000	122,000	73,000
56		Q1	2013	909,000	65,000	121,000	113,000	77,000	116,000	88,000	123,000	124,000	84,000
57		Q2	2013	935,000	57,000	119,000	123,000	96,000	127,000	81,000	119,000	119,000	94,000
58													
59		Notes:											
60		1) Age refers to academic age, which is the respondent's age at the preceeding 31 August.											
61		2) All estimates should be viewed in conjunction with their Confidence Intervals. Confidence Intervals indicate how accurate an estimate is.											
62		For example, a 95% CI of +/− 1,000 means that the true value is between 1,000 above the estimate and 1,000 below the estimate, for 95% of e											
63		3) All estimates are taken from the Labour Force Survey.											
64		4) All estimates refer to calendar quarters.											
65		5) Numbers are rounded to the nearest 1,000.											
66		6) The Labour Force Survey has not been reweighted to reflect the Census 2011 population estimates.											

Figure 3.10: This is a Table of UK NEET Data (Department for Education (UK), 2013), displaying another case of number labels. In this case the current pattern recognition fails to correctly recognise the number labels as labels and manual input is required for correct conversion. The manual specification would be:

```
IdentResult = list(rows = list(label = 3:4, data = 5:57),
                   cols = list(label = 2:3, data = 4:24))
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Table 1												
2	Top 100 Baby Girls' Names in New Zealand												
3	December 2004–2011 Years												
4													
5	Rank		2,004			2,005			2,006			2,007	
6			Name		No.	Name		No.	Name		No.	Name	
7													
8		1		Emma	352		Emma	315		Charlotte	324		Ella
9		2		Charlotte	330		Ella	292		Ella	320		Sophie
10		3		Ella	306		Charlotte	278		Sophie	295		Olivia
11		4		Sophie	299		Olivia	274		Emma	286		Emma
12		5		Hannah	286		Jessica	257		Olivia	278		Charlotte
13		6		Emily	282		Sophie	254		Emily	277		Emily
14		7		Jessica	282		Grace	248		Grace	262		Lily
15		8		Olivia	275		Hannah	223		Jessica	261		Grace
16		9		Grace	261		Emily	216		Hannah	254		Hannah
17		10		Isabella	206		Isabella	180		Lily	234		Isabella
18		11		Georgia	201		Paige	180		Isabella	224		Jessica
19		12		Samantha	196		Ruby	180		Lucy	194		Ruby
20		13		Brooke	192		Lucy	174		Chloe	190		Amelia
21		14		Lucy	190		Lily	169		Ruby	174		Lucy
22		15		Paige	187		Maia	168		Georgia	168		Madison
23		16		Lily	181		Brooke	162		Paige	167		Chloe
24		17		Sarah	161		Georgia	162		Amelia	164		Brooke
25		18		Holly	160		Holly	160		Maia	161		Ava
26		19		Chloe	154		Chloe	150		Zoe	161		Mia
27		20		Ruby	143		Amelia	146		Madison	157		Paige
28		21		Madison	142		Samantha	141		Brooke	154		Zoe
29		22		Amelia	140		Jade	137		Holly	150		Holly
30		23		Zoe	132		Sarah	135		Samantha	150		Kate
31		24		Mia	131		Kate	134		Sarah	149		Caitlin
32		25		Caitlin	124		Caitlin	130		Mia	143		Maia
33		26		Kate	123		Zoe	122		Ava	142		Georgia
34		27		Jade	118		Madison	121		Jasmine	132		Samantha
35		28		Maia	118		Amy	119		Kate	123		Sophia
36		29		Amy	116		Mia	119		Hayley	122		Sienna
37		30		Jasmine	114		Jasmine	118		Caitlin	121		Jade
38		31		Amber	106		Hayley	114		Jade	114		Amber
39		32		Hayley	102		Amber	99		Sophia	105		Maddison
40		33		Molly	102		Anna	95		Amber	103		Sarah
41		34		Sophia	101		Katie	92		Eva	102		Hayley
42		35		Paris	100		Sophia	91		Molly	100		Amy
43		36		Danielle	99		Molly	89		Amy	96		Summer
44		37		Jorja	94		Summer	88		Katie	92		Jasmine
45		38		Mackenzie	90		Elizabeth	87		Keira	91		Nevaeh
46		39		Tayla	90		Stella	79		Anna	89		Abigail
47		40		Anna	87		Danielle	77		Sienna	89		Eden
48		41		Elizabeth	87		Rebecca	76		Summer	88		Eva
49		42		Kayla	86		Kayla	75		Zara	87		Katie
50		43		Alexandra	84		Alexandra	73		Maddison	86		Keira
51		44		Ashley	83		Jorja	72		Kayla	84		Mikayla
52		45		Phoebe	83		Abigail	70		Elizabeth	83		Isabelle
53		46		Katie	82		Maddison	69		Abigail	82		Anna
54		47		Summer	82		Trinity	69		Isla	82		Bella
55		48		Rebecca	81		Tayla	68		Brianna	80		Leah
56		49		Abigail	80		Ashley	67		Danielle	78		Poppy
57		50		Ava	80		Paris	66		Rebecca	77		Isla
58		51		Maddison	75		Ava	62		Alyssa	76		Kayla
59		52		Brianna	73		Mikayla	62		Jorja	74		Elizabeth
60		53		Laura	73		Mya	62		Leah	72		Molly
61		54		Alice	71		Aimee	60		Aaliyah	70		Tayla
62		55		Alyssa	71		Alyssa	60		Alexandra	68		Alyssa
63		56		Leah	70		Eva	60		Tayla	68		Danielle
64		57		Mikayla	69		Victoria	60		Ashley	67		Stella
65		58		Stella	68		Aaliyah	58		MacKenzie	67		Zara
66		59		Trinity	65		Alice	57		Phoebe	67		Mackenzie

Figure 3.11: This is a Table of Top 100 Baby Girls' Names in New Zealand (Department of Internal Affairs (NZ), 2012), displaying a peculiar mismatch of column labels to the corresponding data, in addition to number labels. The number labels here are correctly identified with pattern recognition. The correction of the mismatched column labels is resolved not in Identification, but in the next stage, Discern Parentage. However, during Identification the columns identified as 'data' must include all the data and column labels (in this case `cols = list(label = 2, data = 3:26)`).

3.5.3 Limitations and Future Work

Due to the many variations encountered in how Tables are presented, Identification proves to be surprisingly difficult to automate. It is perhaps an area where a more AI-orientated approach (as opposed to a purely algorithmic one) will be more successful as the variations are often minor. For the same reason however, a human user can very easily identify the necessary information and TableToLongForm makes it possible for manual input of `IdentResult`, enabling use of TableToLongForm with only minor manual input on a much greater variety of Tables.

Short of implementing some form of AI-orientated approach, the most promising future development for Identification would be better diagnostics output to aid the end user in knowing if and when TableToLongForm has gone wrong, and to make it easier to manually input the `IdentResult` at that stage if that is the problem.

Implementation of multiple Identification algorithms would also enable a consensus-based approach to Identification by checking if multiple algorithms agree on the same `IdentResult`. However creating algorithms that work beyond a very narrow class of Tables proves to be surprisingly difficult, and multiple algorithms that can handle a wide range of Tables would be required for a consensus approach to be practically useful.

While seeming at a glance to be one of the easier tasks, Identification instead proves to be the greatest hurdle in making TableToLongForm work with a greater variety of Tables.

3.6 Discern Parentage

3.6.1 Purpose

The purpose of Discern Parentage is to understand the hierarchical structure (the *parentage*) of the row and column labels. The output will be the `rowplist` and `colplist`, the row and column parentage lists. The parentage lists are nested `list` objects that represent all the hierarchical relationships in the Table. For easier reading they are assigned the `plist` class which has a custom `print` method. An example of a `colplist` is shown in Figure 3.12.

The process of Discerning Parentage is comprised of three classes of algorithms. Custom algorithms can be used to extend functionality for two of the three classes (see Subsection 3.2.6).

Pare Pre Row Pre-requisite algorithms that tidy up the Row Labels for correct operation of the Main Parentage algorithm. Any combination of these algorithms, in any order, can be chosen. The current implementation of TableToLongForm has no Pre Row algorithms. They should take two arguments, `matData` and `matRowLabel`. They should return a named list containing two elements, `matData` and `matRowLabel`.

Default: `ParePreRow = NULL`

Pare Pre Col Pre-requisite algorithms that tidy up the Column Labels for correct operation of the Main Parentage algorithm. Any combination of these algorithms, in any order, can be chosen. They should take two arguments, `matData` and `matColLabel`. They should return a named list containing two elements, `matData` and `matColLabel`.

Default: `ParePreCol = c("mismatch", "misalign", "multirow")`

Pare Main contains the Main algorithm that will recursively call itself until all parentage is discerned. Unlike the Pre-requisite algorithms, the Main algorithm is not an argument that can be changed.

```
> TableToLongForm(LabourForce, fulloutput = TRUE)[["colplist"
  ]]
+ Male (1, 2)
- + European Only (1, 3)
- - + Persons Employed in Labour Force (1, 4)
- - + Persons Unemployed in Labour Force (2, 4)
- - + Not in Labour Force (3, 4)
- - + Working Age Population (4, 4)
- - + Labour Force Participation Rate (5, 4)
- - + Unemployment Rate (6, 4)
- - + Employment Rate (7, 4)
- - + Total Labour Force (8, 4)
- + Maori Only (9, 3)
- - + Persons Employed in Labour Force (9, 4)
## Output truncated
```

Figure 3.12: A truncated example of the `colplist` for the Labour Force Status data used in Figure 3.1. It represents the hierarchical relationships of the column labels. We can see that it has correctly identified `Male` as a top-level parent with the ethnic categories, such as `European Only`, nested inside. The ethnic categories are in turn parents to the lowest-level categories, such as `Employment Rate`.

```
> colplistNoClass
$Male
$Male$`European Only`
  Persons Employed in Labour Force Persons Unemployed in Labour Force
                                 1                                   2
              Not in Labour Force             Working Age Population
                                 3                                   4
   Labour Force Participation Rate                 Unemployment Rate
                                 5                                   6
                  Employment Rate                Total Labour Force
                                 7                                   8
attr(,"Loc")
     cols
[1,]    4

$Male$`Maori Only`
  Persons Employed in Labour Force Persons Unemployed in Labour Force
                                 9                                  10
## Output truncated
```

Figure 3.13: A truncated example of the `colplist` from Figure 3.12, printed with the default list printing method. This is much less useful, but does give better insight into the internal storage structure for users familiar with R lists.

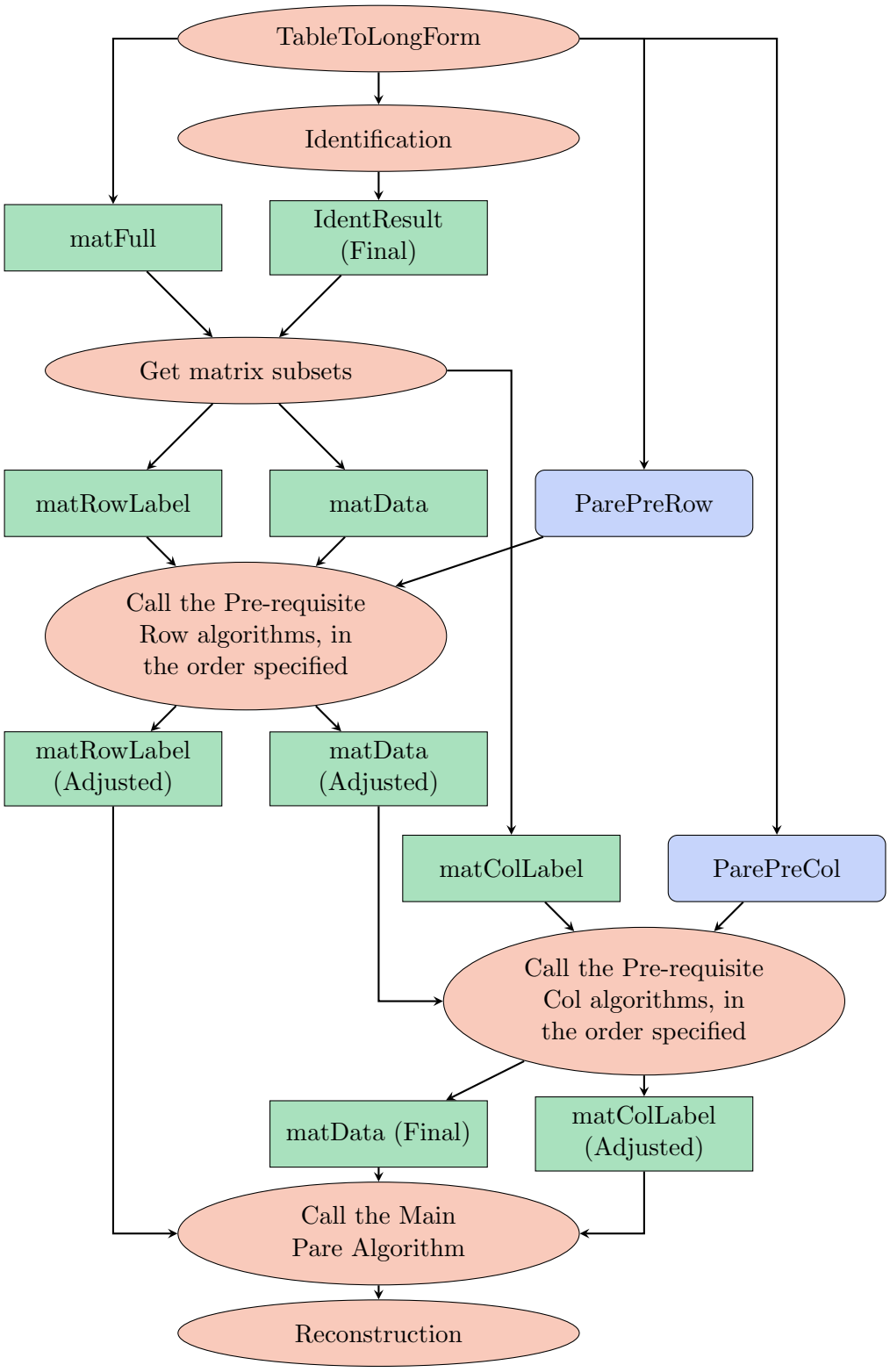


Figure 3.14: A breakdown of the Discern Parentage step of the workflow. `IdentResult` from the Identification step is used obtain subsets of matFull that correspond to the labels and the data. These subsets are then processed with the ParePreRow and ParePreCol algorithms as specified by the respective arguments. These adjusted subsets are then used by the Main Parentage algorithm to discern the parentage of the Table.

3.6.2 Key Challenges

The Main Parentage algorithm consists primarily of the process covered in Core Concepts (Subsection 3.1.2), and this fairly simple algorithm proves to be surprisingly robust to many Tables. Unfortunately, even slightly unusual presentation can lead to catastrophic breakdown. The Pre-requisite algorithms are designed to correct certain known cases of breakdown, such that the adjusted Table subsets conform to a format the Main algorithm can handle.

The types of Tables that the Pre-requisite and Main Parentage algorithms can handle are called the *recognised patterns*. Any Table that consists of some combination of the recognised patterns can be successfully handled with TableToLongForm. It is not strictly necessary for a user to know what the patterns are, as they can simply try calling TableToLongForm on the Table to see if it converts. All the recognised patterns are listed here primarily for reference purposes.

For each pattern an example table is first shown using toy data, that displays the pattern, followed by a short description of the pattern, and ending with the example table converted with TableToLongForm.

Many of the recognised patterns apply only for row labels. Column labels are recognised by noticing that the transpose of column labels can often be processed as row labels, though there are several fringe cases that must be corrected for.

Empty Below

	1	2	3	4	5	6
1			Column 1	Column 2	Column 3	Column 4
2	Row Parent1	Row Child1	10	20	30	40
3		Row Child2	11	21	31	41
4	Row Parent2	Row Child1	12	22	32	42
5		Row Child2	13	23	33	43

Above, we have an example of the Empty Below pattern, the most simple type of parentage. Here the *parent* and *children* are in different columns and we can see which of the children belong to which parent through the use of empty space below each parent. The Table after conversion to a LongForm follows.

	1	2	3	4	5	6
1			Column 1	Column 2	Column 3	Column 4
2	Row Parent1	Row Child1	10	20	30	40
3	Row Parent1	Row Child2	11	21	31	41
4	Row Parent2	Row Child1	12	22	32	42
5	Row Parent2	Row Child2	13	23	33	43

Empty Right 1

	1	2	3	4	5	6	7
1				Column 1	Column 2	Column 3	Column 4
2	Row Parent1			10	20	30	40
3	Row Child1	Row Child−Child1		11	21	31	41
4	Row Child2	Row Child−Child2		12	22	32	42
5	Row Parent2			13	23	33	43
6	Row Child1	Row Child−Child1		14	24	34	44
7		Row Child−Child2		15	25	35	45

Above, we have an example of the most basic form of the Empty Right pattern. In this situation we have children in the same column as their parent. We can still recognise these as children if the children have children (*Child-Child*) in a different column, while the parent does not (and hence the parent is Empty Right). Note the values pertaining to the Parent (if any) are discarded. This is because they are assumed to simply represent the sum of their children's values. The Table after conversion to a LongForm follows.

	1	2	3	4	5	6	7
1				Column 1	Column 2	Column 3	Column 4
2	Row Parent1	Row Child1	Row Child−Ch	11	21	31	41
3	Row Parent1	Row Child2	Row Child−Ch	12	22	32	42
4	Row Parent2	Row Child1	Row Child−Ch	14	24	34	44
5	Row Parent2	Row Child1	Row Child−Ch	15	25	35	45

Empty Right 2

	1	2	3	4	5	6
1			Column 1	Column 2	Column 3	Column 4
2	Row Parent1		10	20	30	40
3		Row Child1	11	21	31	41
4		Row Child2	12	22	32	42
5	Row Parent2		13	23	33	43
6		Row Child1	14	24	34	44
7		Row Child2	15	25	35	45

Above, we have an example of both Empty Below and Empty Right. Either algorithm can handle this situation, but simply due to the ordering of the algorithms such situations are handled as Empty Right. The Table after conversion to a LongForm follows.

	1	2	3	4	5	6
1			Column 1	Column 2	Column 3	Column 4
2	Row Parent1	Row Child1	11	21	31	41
3	Row Parent1	Row Child2	12	22	32	42
4	Row Parent2	Row Child1	14	24	34	44
5	Row Parent2	Row Child2	15	25	35	45

Empty Right 3

	1	2	3	4	5	6	7	8
1				Column 1	Column 2	Column 3	Column 4	
2	Row Super−Parent1			10	20	30	40	
3	Row Parent1			11	21	31	41	
4	Row Child1	Row Child−Child1		12	22	32	42	
5	Row Parent2			13	23	33	43	
6	Row Child1	Row Child−Child1		14	24	34	44	
7	Row Super−Parent2			15	25	35	45	
8	Row Parent1			16	26	36	46	
9	Row Child1	Row Child−Child1		17	27	37	47	
10	Row Parent2			18	28	38	48	
11	Row Child1	Row Child−Child1		19	29	39	49	

Above, we have an example of a complex version of the Empty Right pattern. The "parent-child in the same column" situation has been extended further and we now have parents (*Super-Parent*) who have children (*Parent*), who each further have children (*Child*), all in the same column. Such situations can still be recognised if the lowest-level children in the column (*Child*) have children in a different column (*Child-Child*), while its direct parents (*Parent*) each have children in the same column (*Child*) but not in a different column (is Empty Right), and the top-most parents (*Super-Parents*) also have no children in a different column (is also Empty Right). The algorithm cannot currently handle super-super-parents. The Table after conversion to a LongForm follows.

	1	2	3	4	5	6	7	8
1					Column 1	Column 2	Column 3	Column 4
2	Row Super−P	Row Parent1	Row Child1	Row Child−Ch	12	22	32	42
3	Row Super−P	Row Parent2	Row Child1	Row Child−Ch	14	24	34	44
4	Row Super−P	Row Parent1	Row Child1	Row Child−Ch	17	27	37	47
5	Row Super−P	Row Parent2	Row Child1	Row Child−Ch	19	29	39	49

Multi-row Column Label

	1	2	3	4	5	6
1		Column	Column	Column	Column	
2		Child1	Child2	Child3	Child4	
3	Row 1	10	20	30	40	
4	Row 2	11	21	31	41	
5	Row 3	12	22	32	42	
6	Row 4	13	23	33	43	

Above, we have an example of Multi-row Column Labels. Often column labels are physically split over multiple rows rather than making use of line breaks in the same cell. In such occurrences, any row not identified as a parent are collapsed into a single row of labels. The Table after conversion to a LongForm follows.

	1	2	3	4	5	6
1		Column Child1	Column Child2	Column Child3	Column Child4	
2	Row 1	10	20	30	40	
3	Row 2	11	21	31	41	
4	Row 3	12	22	32	42	
5	Row 4	13	23	33	43	

Mismatched Column Label

	1	2	3	4	5	6	7	8	9
1		Col Child1		Col Child2		Col Child3		Col Child4	
2	Row 1		10		20		30		40
3	Row 2		11		21		31		41
4	Row 3		12		22		32		42
5	Row 4		13		23		33		43

Above, we have an example of Mismatched Column Labels. Sometimes the column labels are in a different column to the data, usually due to a misguided attempt at visual alignment of labels to the data. As long as the correct rows and columns were identified for the data and the labels, and if there are the same number of data columns as label columns, these mismatched column labels will be paired with the data columns. The Table after conversion to a LongForm follows.

	1	2	3	4	5	6	7	8	9
1		Col Child1	Col Child2	Col Child3	Col Child4				
2	Row 1	10	20	30	40				
3	Row 2	11	21	31	41				
4	Row 3	12	22	32	42				
5	Row 4	13	23	33	43				

Misaligned Column Label

	1	2	3	4	5	6	7	8	9
1			Col Parent1				Col Parent2		
2		Col Child1	Col Child2	Col Child3	Col Child4	Col Child1	Col Child2	Col Child3	Col Child4
3	Row 1	10	20	30	40	50	60	70	80
4	Row 2	11	21	31	41	51	61	71	81
5	Row 3	12	22	32	42	52	62	72	82
6	Row 4	13	23	33	43	53	63	73	83

Above, we have an example of Misaligned Column Labels. Often column parents are physically centred over their children (N.B. where a spreadsheet's cell-merge feature is used to do the centering, the actual value is usually stored in the top-left cell and hence causes no problems). TableToLongForm makes use of pattern recognition to identify repeating patterns in the labels, or in empty cells surrounding the labels, to correct for the misalignment. For the *Column Parents* row, we find (starting from column 2, the first data column) a pattern of Empty-NonEmpty-Empty-Empty, with the pattern occurring twice. In the *Col Child* row, we also find a pattern of length 4 occurring twice. This can be used to correctly align the *Column Parents* to its children. The Table after conversion to a LongForm follows.

	1	2	3	4	5	6	7	8	9
1			Col Child1	Col Child2	Col Child3	Col Child4			
2	Col Parent1	Row 1	10	20	30	40			
3	Col Parent1	Row 2	11	21	31	41			
4	Col Parent1	Row 3	12	22	32	42			
5	Col Parent1	Row 4	13	23	33	43			
6	Col Parent2	Row 1	50	60	70	80			
7	Col Parent2	Row 2	51	61	71	81			
8	Col Parent2	Row 3	52	62	72	82			
9	Col Parent2	Row 4	53	63	73	83			

Misaligned Column Label 2

	1	2	3	4	5	6	7	8	9
1				Col Super−Parent					
2			Col Parent1				Col Parent2		
3		Col Child1	Col Child2	Col Child3	Col Child4	Col Child1	Col Child2	Col Child3	Col Child4
4	Row 1	10	20	30	40	50	60	70	80
5	Row 2	11	21	31	41	51	61	71	81
6	Row 3	12	22	32	42	52	62	72	82
7	Row 4	13	23	33	43	53	63	73	83

Above, we have a generalised example of Misaligned Column Labels. We now have *Column Super-Parent* which is misaligned to both its direct children, the *Column Parents*, and to the lowest-level children. The Table after conversion to a LongForm follows.

	1	2	3	4	5	6	7	8	9
1				Col Child1	Col Child2	Col Child3	Col Child4		
2	Col Super−Pa	Col Parent1	Row 1	10	20	30	40		
3	Col Super−Pa	Col Parent1	Row 2	11	21	31	41		
4	Col Super−Pa	Col Parent1	Row 3	12	22	32	42		
5	Col Super−Pa	Col Parent1	Row 4	13	23	33	43		
6	Col Super−Pa	Col Parent2	Row 1	50	60	70	80		
7	Col Super−Pa	Col Parent2	Row 2	51	61	71	81		
8	Col Super−Pa	Col Parent2	Row 3	52	62	72	82		
9	Col Super−Pa	Col Parent2	Row 4	53	63	73	83		

3.6.3 Limitations and Future Work

The Main Parentage algorithms can be considered 'complete' as they have been largely unchanged since inception, the pre-requisite algorithms are the problem. In many ways, the Pre algorithms share the same problems as Identification, as small changes in the Table, ones easily understood by humans, can break the algorithms, again suggesting that an AI-orientated approach might be more helpful. As with Identification, better diagnostics will be of greatest aid at this stage of development, with manual correction of problematic features enabling usage of TableToLongForm with a wider variety of Tables for less work than it would take to undertake the full conversion manually.

Another set of problems are other methods of encoding hierarchical information that are difficult to access through R, e.g. in-cell indentation in Excel. As by definition these are difficult to access through R, they are also difficult to solve within R. Possible solutions include writing in a language more appropriate for the problem, e.g. a VBA script for Excel-based problems. These scripts can then take care of some of the pre-requisite cleaning, before the Table is loaded into R for the conversion.

3.7 Reconstruction

The purpose of Reconstruction is to take the information from Identification and Parentage and use it to Reconstruct a dataframe. In other words, there should be nothing beyond simple grunt-work at this stage, and the current Reconstruction code does what it is supposed to. If problems do arise, they need to be addressed in the previous stages.

```
> rowplist
+ Row Super-Parent (1, 1)
- + Row Parent1 (2, 1)
- - + Row Child1 (3, 1)
- - - + Row Child-Child1 (3, 2)
- - - + Row Child-Child2 (4, 2)
- - + Row Child2 (5, 1)
- - - + Row Child-Child1 (5, 2)
- - - + Row Child-Child2 (6, 2)
- + Row Parent2 (7, 1)
- - + Row Child1 (8, 1)
- - - + Row Child-Child1 (8, 2)
- - - + Row Child-Child2 (9, 2)
- - + Row Child2 (10, 1)
- - - + Row Child-Child2 (10, 2)

> rowvecs
 [,1]                [,2]           [,3]          [,4]
 "Row Super-Parent" "Row Parent1" "Row Child1" "Row Child-
    Child1"
 "Row Super-Parent" "Row Parent1" "Row Child1" "Row Child-
    Child2"
 "Row Super-Parent" "Row Parent1" "Row Child2" "Row Child-
    Child1"
 "Row Super-Parent" "Row Parent1" "Row Child2" "Row Child-
    Child2"
 "Row Super-Parent" "Row Parent2" "Row Child1" "Row Child-
    Child1"
 "Row Super-Parent" "Row Parent2" "Row Child1" "Row Child-
    Child2"
```

Figure 3.15: An example of a `rowplist` and the reconstructed version (both truncated). These must then be combined with the reconstructed `colplist` and the data for the final dataframe. Broadly speaking, the Reconstruction algorithms simply iterate down a plist recursively, extracting the information and pasting them together appropriately to create the dataframe.

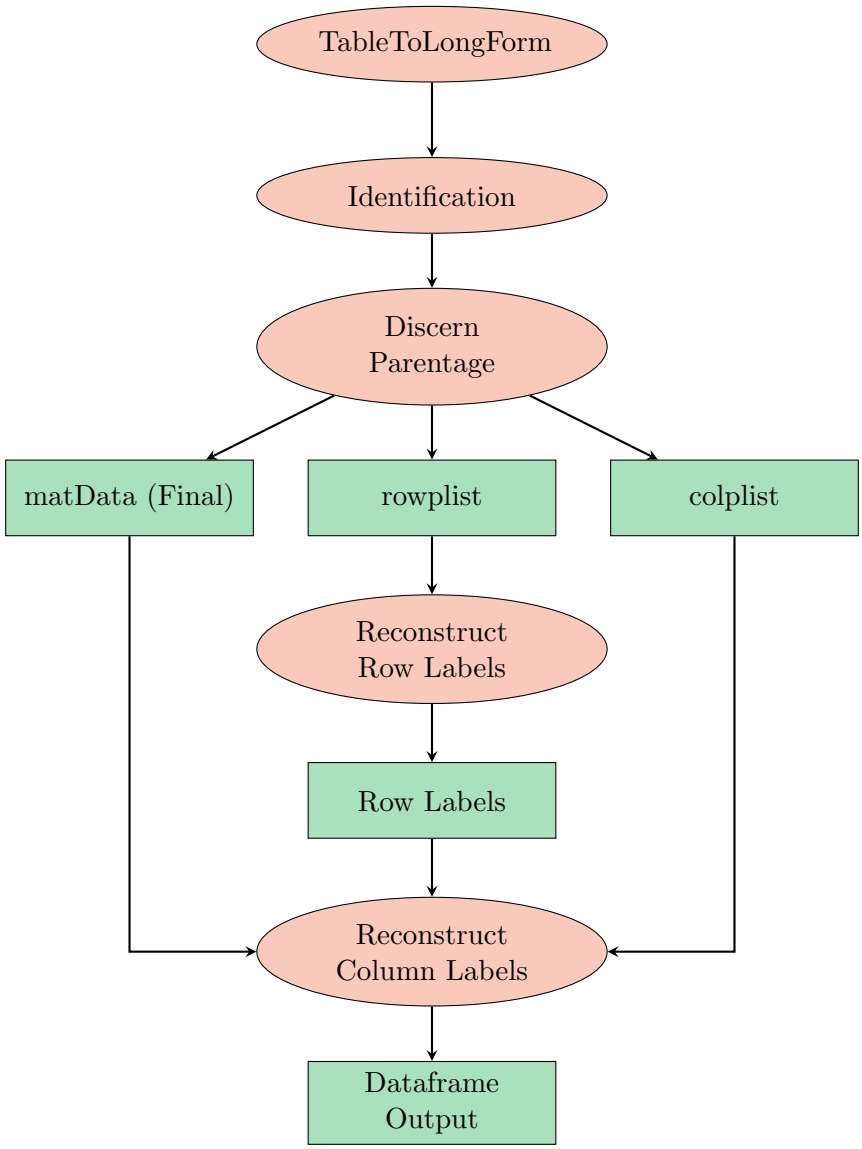


Figure 3.16: A breakdown of the Reconstruction step of the workflow. ReconsRowLabels only makes Row Labels. ReconsColLabels takes the Row Labels, may make additional Row Labels (for Col Parents), then combines it with the Col Labels, grabbing the associated columns of data as it does so. Thus the output of ReconsColLabels is the final dataframe output.

3.8 Summary

TableToLongForm is a response to a problem identified while conducting the Literature Review on Open Data in New Zealand - data is being released in Tables that are unfit for machine consumption. It then seemed logical that an automatic tool for converting such Tables into more machine-friendly LongForm Dataframes would be useful, and it seems many others agree. The package has been available on CRAN since September 2013 and has seen a steady number of downloads since then (see Figure 3.17). More recently, an article on TableToLongForm has been published in the RJournal (Oh, 2014), and a reviewer for the article commented:

> This is a really important contribution to R. These file types are everywhere and they waste a huge amount of time to read for virtually every analyst... I'm unaware of any alternative approaches in R that are anywhere near this easy to use.

Despite the positive feedback, TableToLongForm still has many limitations. Ultimately, TableToLongForm uses relatively simple algorithms to detect a known set of recognised patterns and any Table that deviates from these patterns will break TableToLongForm. Better diagnostics output and error messages are needed to ensure that when TableToLongForm does work, it can be verified that it worked correctly, and by the same token, if it failed, the user needs to be able to understand it failed and why, so that they may attempt to fix the problem (either by adjusting the Table so that TableToLongForm can handle it, or by converting manually).

Overall, reception of TableToLongForm seems to be divided into two main groups. Those in the first group are very positive of the idea of an automatic way to handle data released in Tables, because they have painful first-hand experience in trying to deal with Tables manually. Those in the second group have been fortunate enough to have never needed to work with data released in Tables, and their reaction is lukewarm, conceding only that it may be useful for a *limited* set of problems - a set of problems that has little to do with them. These two reactions are quite indicative of the value of TableToLongForm - for those seeking to make use of data released in Tables that TableToLongForm can handle, the package will prove to be very valuable. However not only is there data that TableToLongForm cannot handle, but there is data that is already released in a nice machine-readable format. If more organisations release their data well, then tools like TableToLongForm will become obsolete. While it would be sad to see something I've worked on become useless, I will nonetheless welcome the day when all data is released in a good format from the start.

Figure 3.17: While there is no data for the total number of downloads across all CRAN mirrors, there is data from the RStudio mirror. The different colours indicate different versions of TableToLongForm. The trend has held remarkably steady across time, to the point I suspect the reliability of this data (the steady downloads may be from other mirrors syncing, or other such automated downloads, and not indicative of 'real' users), but it is unfortunately one of the only sources of data on package downloads.

Chapter 4

Graphical Tools
A Literature Review

4.1 Introduction . . . 74
4.1.1 Free . . . 74
4.1.2 Interactive Web-based Output . . . 74
4.1.3 Tool Classification . . . 75
4.2 GUI Tools . . . 76
4.2.1 Many Eyes . . . 76
4.2.2 Tableau Public . . . 78
4.2.3 Other GUI Tools . . . 80
4.3 High-level Languages . . . 80
4.3.1 Google Chart Tools . . . 80
4.3.2 Highcharts . . . 82
4.4 Low-level Languages . . . 85
4.4.1 Processing.js . . . 87
4.4.2 Paper.js . . . 89
4.4.3 Raphaël.js . . . 91
4.4.4 D3.js (Data-Driven Documents) . . . 92
4.4.5 Other Low-level Languages . . . 94
4.5 Other Relevant Topics . . . 95
4.5.1 R . . . 95
4.5.2 Raster graphics . . . 99
4.5.3 Vector graphics . . . 99
4.5.4 HTML5 Canvas Element . . . 100
4.5.5 SVG . . . 100
4.6 Conclusion . . . 101

4.1 Introduction

This Literature Review examines **free** Graphical Tools that produce **interactive output** ideal for **web-based viewing**. Three different **Classifications** of tools are examined: *GUI* Tools, *High-level* languages and *Low-level*[1] languages.

Section 4.5 covers Other Relevant Topics, including how `R` might be used to aid in preparing and exporting data for use with one of the tools, and brief explanations on the difference between *raster* and *vector* graphics, or how the *SVG* image file format works, to make the review more accessible to audiences unfamiliar with such details.

N.B. This Literature Review was conducted in 2012, and many details may now be out-of-date.

4.1.1 Free

In keeping with the spirit of Open Data, I am interested in free tools, so that a casual user interested in exploring some Open Data can use such a tool at no cost.

Ideally, not only is the tool free, it is also *Open Source*. The distinction is not very important for the casual user, but for me as I look to develop new tools, existing tools that are *Open Source* provide opportunities to learn and/or extend functionality.

Additionally, if the output is also in an *Open Format* (such as SVG), this opens up opportunities to utilise multiple tools (that share this Open Format) in conjunction, for greater effect.

4.1.2 Interactive Web-based Output

I focus on interactive output rather than more traditional static images as these are more interesting, and are a better candidate for future work. Being web-based makes the output substantially easier to share, and also passes off much of the rendering burden to web browser developers, allowing me to focus more on visualisation methods. I classify interactivity into two categories:

Simple Interactivity This includes cases where the user can mouseover a bar on a bar-graph, or a slice of a piechart, and obtain more information about that bar or slice. It is technically interactive and not static, but typically adds very little value.

End-user Data Exploration This is where the final output can be used by the *end-user*[2] to *explore* the data in some way. At the very basic level, this may involve merely

[1] By more conventional definition, these should be *Very High* and *High*, but for better distinguishing power more extreme words are used.

[2] **End-User:** The user of the final output. Distinguished from the user of one of the graphical tools, who produce the output.

resorting the data. At a more advanced level, this can effectively turn the output into a simple *GUI* tool of its own, where the *end-user* can change the variables being examined, the chart type, and so on.

4.1.3 Tool Classification

I classify the tools broadly into three categories: *GUI* (Graphical user interface), *High-Level* language and *Low-Level* language.

Note that the definition of *High-* and *Low-* level programming languages is largely relative. A quick google search suggests a conventional definition for a *Low-level* language might be something like an assembly language. For the purposes of this review the following definitions will be used:

GUI A typical example would be a point and click interface, requiring absolutely no writing of code to accomplish the desired task. Without doubt, to a casual user the *GUI* is the most accessible and desired tool.

High-level A language that can achieve the desired result with a few lines of very simple code, usually involving specific *High-level* functions that perform several tasks. An example might be doing a simple linear regression model in R. The task can be accomplished with a single call to `lm` to fit the model. We may then need a few additional calls to check the assumptions are met and print the output. Excel formulae may also fall under this definition, though in that case, the formulae can be used via a GUI rather than direct 'coding'. Under this rather restrictive definition of *High-level* language, it is feasible that a casual user might be willing to learn and use a *High-level* language tool, though they would be resistant to such a notion.

Low-level A language that requires several lines of code to achieve the desired result, including the use of *primitive* or *Low-level* functions that only accomplish a very narrow, simple and specific task (such as drawing a single rectangle). Under this definition, a language like *JavaScript* would be considered *Low-level*, as it would be difficult to learn and use for the casual user. While out of reach for the casual user, these *Low-level* language tools will be valuable in the creation of new tools more appropriate for the casual user.

I am most interested in the *Low-level* language tools, as these are likely to be used to develop new tools and methods for presenting Open Data. However, a brief overview of existing *GUI* tools and *High-level* language tools is necessary to understand what already exists, what they do well, and what gaps exists where a new tool might be valuable.

4.2 GUI Tools

4.2.1 Many Eyes

http://www-958.ibm.com/software/data/cognos/manyeyes/

Key Points

Learning Difficulty Very Easy.

Has a library of standard plots Yes.

Output Java or Flash object.

Interactivity Wide variety depending on plot type chosen.

Legal Free to use, but any output using Many Eyes will be published online via the website, making the visual and underlying data public. Data has a size limit of 5 MB.

Main Features

- Tool is entirely web-based requiring no download.
- But output is Java and Flash based, meaning the end-user must download and install the platform.
- Has good community support, such as rating of visualisations and datasets, and commenting by users.
- An 'experiment' by IBM, future of the tool is unknown.
- Purely a GUI with no command-line support (Terms of Use expressly forbid automation: "You agree not to send automated queries of any sort to the Services").
- All data uploaded, and any visualisation created, becomes publicly available on the website. However, data and visuals can be deleted at a later date (This method should not be considered a 'secure' way of ensuring limited publicity).
- Very strict about accepted data format. A lot of effort is often required to get the data in just the right format for the desired plot.

Comments

The primary advantage of Many Eyes is that it is a Visualisation GUI Tool that is entirely online, requiring no download or install beyond the software platforms it requires (Java and Flash). Any data uploaded and any visualisation created is automatically published

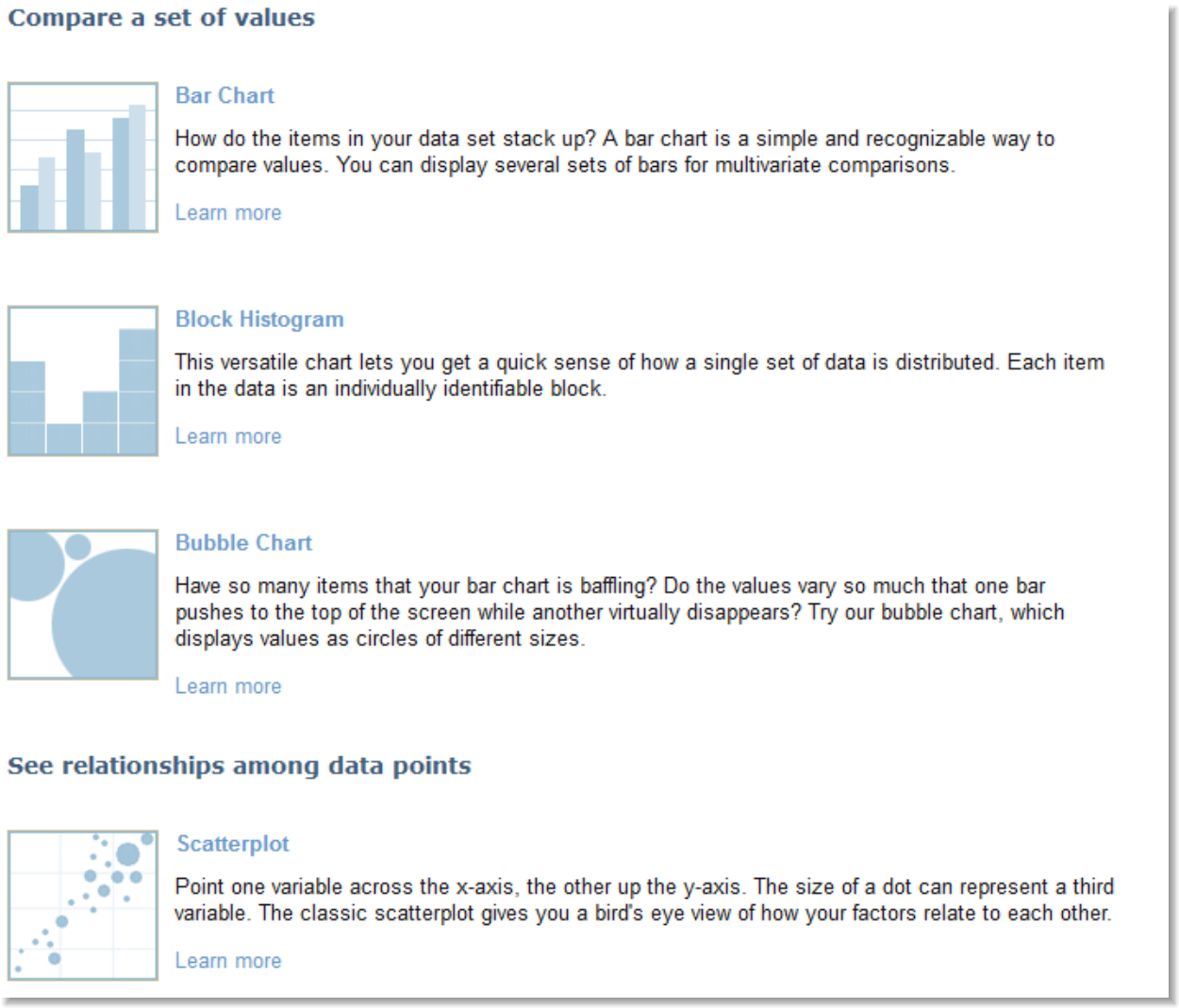


Figure 4.1: The Many Eyes 'interface'. This is a screenshot of one of the webpages used to create a new plot. Many Eyes is entirely web-based and requires no separate client software.

publicly online, making it extremely easy to share (conversely it also means that it cannot be made private. Less of an issue when dealing with Open Data, but has problems for Closed Data).

The number of datasets being uploaded and visualisations being created would indicate Many Eyes has an active user base. Many Eyes first began in 2007 and is still available, though the last update appears to have been in March 2011. Though it is being developed by a well-known major corporation (IBM), its current classification as an 'experiment' does cast a slight shadow of doubt on its future availability.

For example output, visit the public gallery: http://www-958.ibm.com/software/data/cognos/manyeyes/visualizations

4.2.2 Tableau Public

http://www.tableausoftware.com/products/public

Key Points

Learning Difficulty Very Easy.

Has a library of standard plots Yes.

Output Proprietary Format.

Interactivity Extensive potential, including end-user exploration of the data via interaction with the output, but still limited to what the tool provides.

Legal Free to use, but any output using Tableau Public must be published online via their website, making the visual and underlying data public. There is also a data size limitation of 100,000 rows. Paid versions of Tableau are available which relaxes these constraints.

Main Features

- Requires a download and install of a client software, that requires an active internet connection to work.
- Quite powerful relative to ease of use.
- Attempts to automatically detect data type, such as whether it is a 'Dimension' (Qualitative) or a 'Measure' (Quantitative), or identification of names, such as the States of the USA (which greatly simplifies the creation of map-based graphics).
- Suggests appropriate chart types based on data variables selected.
- Powerful interactivity features, including a 'dashboard' feature that can enable data exploration by the end-user. That is, once a graphic output is made and published, any future user of that graphic may have the capability to rearrange categories, change graph type, etc.
- Purely a GUI with no command-line support.
- All data uploaded, and any visualisation created, becomes publicly available on the website.
- The free Public version has various restrictions.

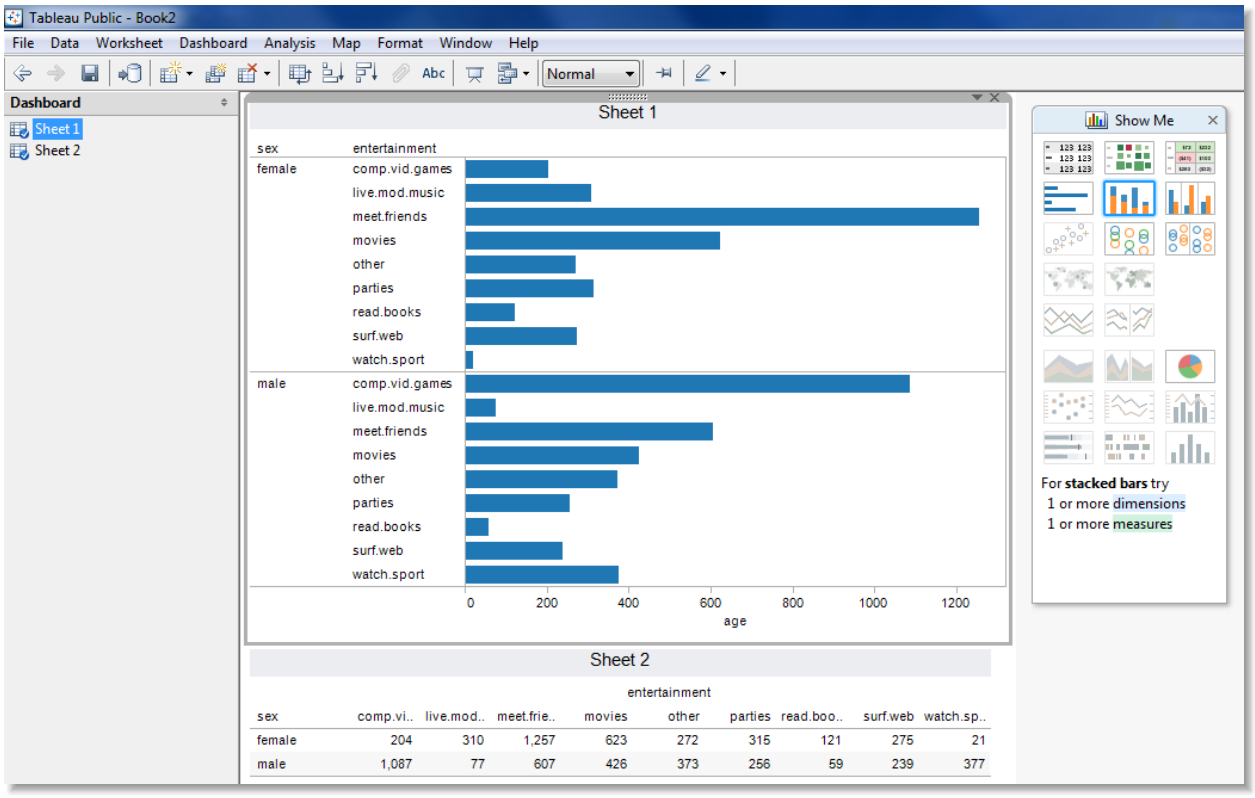


Figure 4.2: The Tableau Public interface. In this example, we are looking at a 'dashboard', allowing us to place multiple graphics on the same page. On the right, Tableau Public suggests some appropriate graph types for the selected data subset (inappropriate graph types are greyed out).

Comments

Starting out in 2003 as an output of a PhD project called Polaris, "an interface for the exploration of multidimensional databases that extends the Pivot Table interface to directly generate a rich, expressive set of graphical displays" (Stolte et al. 2008), it became commercialised as Tableau later that year. In 2010, the free version, Tableau Public was released.

Tableau Public requires a client to be downloaded and installed, and also requires an internet connection to function. Rather than accepting a specific data structure for a specific plot type, Tableau accepts an entire database, and allows the user to explore the variables in the data via a variety of potential plots.

For example output, visit the public gallery: `http://www.tableausoftware.com/public/gallery`

4.2.3 Other GUI Tools

This review is concerned with *GUI Tools* that produce **Interactive Web-based Output**, but it is worth noting that there are many offline *GUI Tools* that can be used for visualising and exploring data interactively. For instance *Improvise* (http://www.cs.ou.edu/~weaver/improvise/) or *The InfoVis Toolkit* (http://ivtk.sourceforge.net/), which may better serve the needs of the reader if they do not require easy facilities for publishing the output online.

4.3 High-level Languages

4.3.1 Google Chart Tools

https://developers.google.com/chart/

Key Points

Learning Difficulty Easy (Very Easy using something like the *Google Vis* R Package, which greatly reduces the work).

Has a library of standard plots Yes.

Output SVG, with support for VML for compatibility with older versions of Internet Explorer. Some of the older graphs (e.g. the Annotated Time Line or Hans Rosling's Gapminder Motion Chart) use Flash.

Interactivity Varies by plot type, most only have simple interactivity, though some graphs offer more interesting interactivity, such as the Annotated Time Line (similar to the one used for the Google Finance graphs) or Hans Rosling's Gapminder Motion Chart (which was acquired by Google in 2007).

Legal Free to use, some fine print in the Terms of Service.

Main Features

- Very easy to use, especially when using the *Google Vis* R Package (or other similar packages).
- A significant variety of plot types to choose from, including community contributed plots.
- In active development by Google.

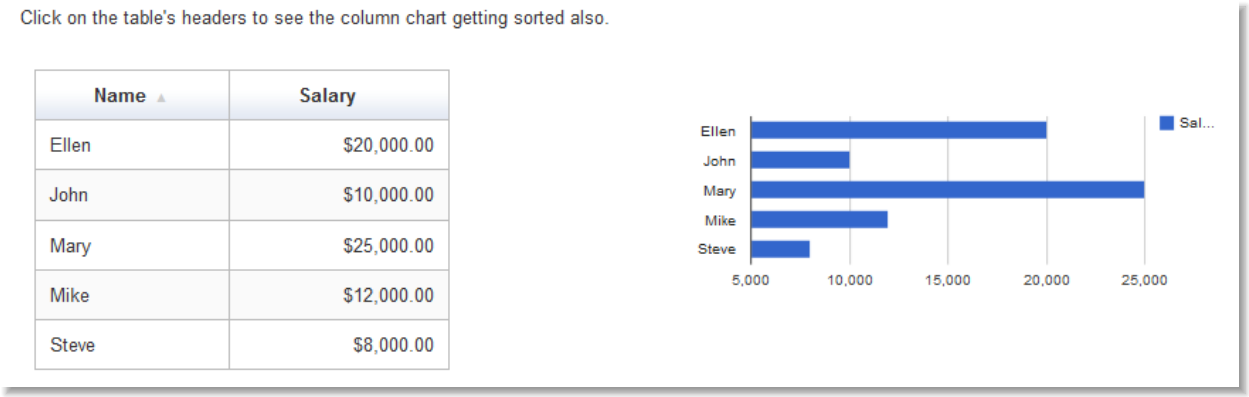


Figure 4.3: A screenshot from `https://developers.google.com/chart/interactive/docs/examples` demonstrating interactivity between a *Table* and a *BarChart*, sorting by Name by clicking on the appropriate column in the *Table*.

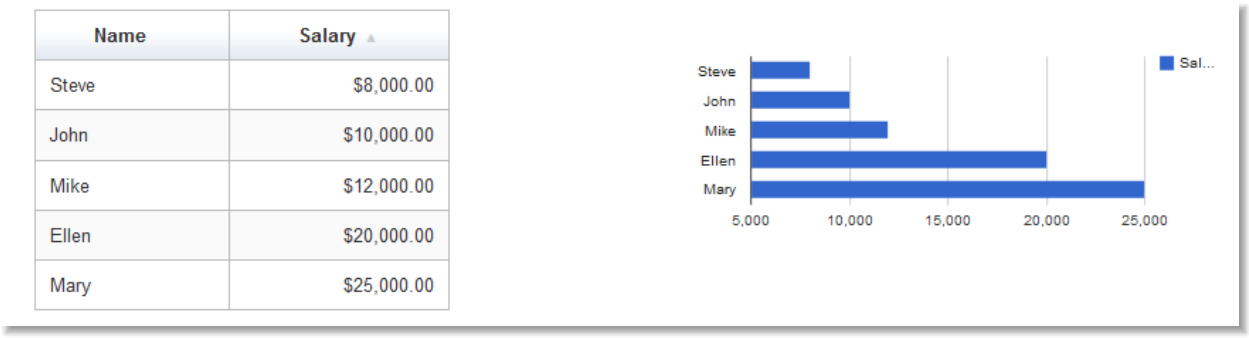


Figure 4.4: A continuation of Figure 4.3, where the *Table* has been resorted by Salary, with the *BarChart* following suit.

Comments

For users already familiar with R, *Google Chart Tools* is the easiest to use of the two *High-level* language tools covered, because of the fantastic *Google Vis* R Package. The package allows everything to be done within R, removing the need to learn and fiddle with HTML and JavaScript code, generally reducing the coding required and enabling easy access to the many data processing and manipulation tools of R. This makes *Google Chart Tools* as easy to use as a *GUI Tool* (for those already familiar with R) if not easier due to the ease of scripting.

One possible problem with *Google Chart Tools* is that it is still in active development, and may not be suited towards use that requires long-term reliability and stability.

A bargraph in Google Chart Tools

Examples are available on the official website for directly writing the HTML and JavaScript code. Here is how to do it in R, which is considerably shorter than writing directly.

```
1 library("googleVis")
2 ## Generate some data
3 dat = data.frame(val = runif(10, 0, 100), label = LETTERS
    [1:10])
4
5 ## Make the plot
6 googBar = gvisBarChart(data = dat, xvar = "label", yvar = "
    val", options = list(width = 1280, height = 720))
7
8 ## Save the plot as an html file
9 ## Package takes care of all the required html code including
     exporting of the dataset into an appropriate JSON format.
10 print(googBar, file = "googBar.html")
```

There is an extensive list of additional arguments that can be passed to `options` to fine tune the chart to your liking.

4.3.2 Highcharts

http://www.highcharts.com/

Key Points

Learning Difficulty Easy.

Has a library of standard plots Yes.

Output SVG, with support for VML for compatibility with older versions of Internet Explorer.

Interactivity Varies by plot type, most only have simple interactivity, but API allows for more interesting interactivity.

Legal Open Source (CC By-NC). Has provisions for commercial use (can buy a commercial licence).

Main Features

- Download comes with an impressive set of examples in the form of complete `.html` files, making it very easy to tweak for actual use.
- Export capability built-in to the graphs, allowing for export of the charts to several formats, including static PNG and JPG images for offline use.

- Commercial side likely to provide incentives for the active development and continuing support of the tool.

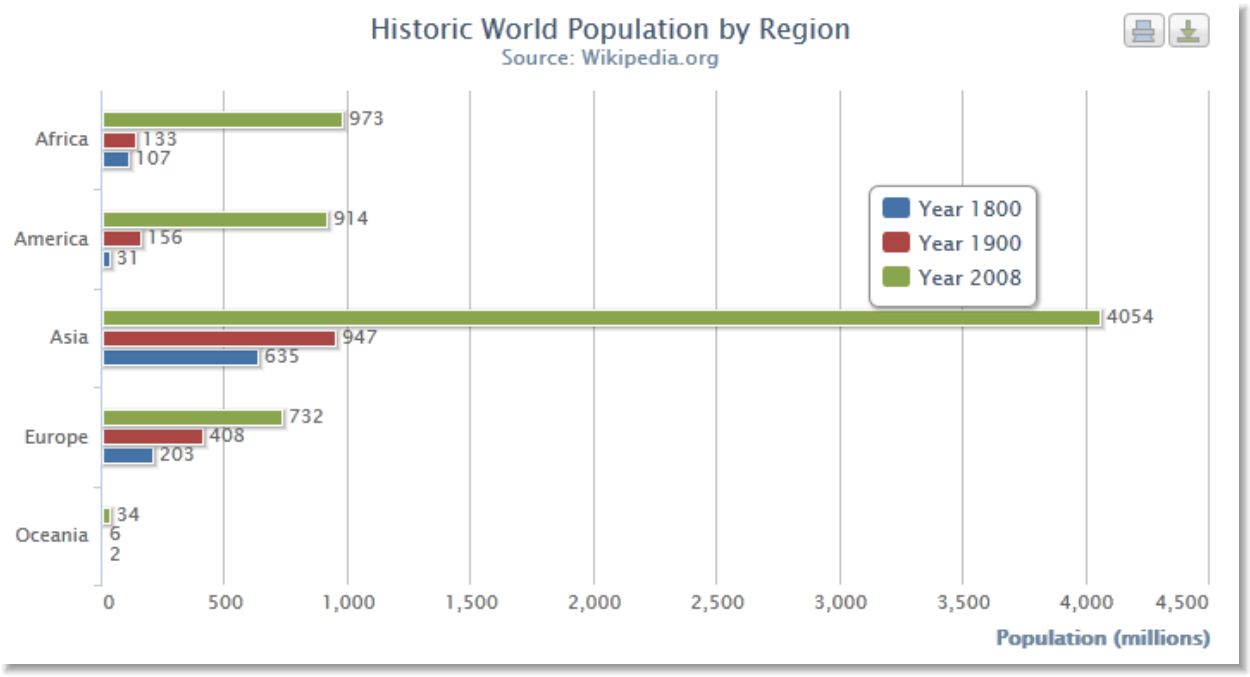


Figure 4.5: A screenshot from `http://www.highcharts.com/demo/bar-basic` demonstrating a basic bar chart.

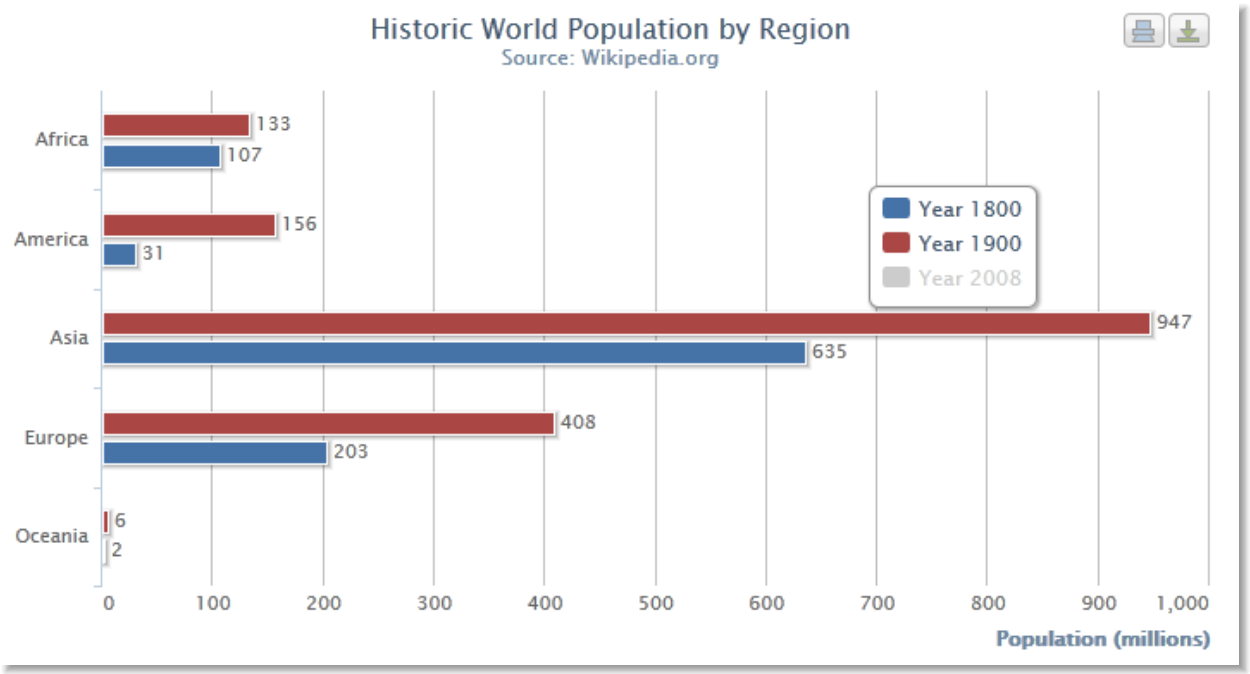


Figure 4.6: A continuation of Figure 4.5, where clicking on **Year 2008** in the legend has filtered this data and the bar chart (including axis) has updated accordingly.

Comments

Though harder to use than *Google Chart Tools* (even if you're not familiar with R), *Highcharts* has a bit more flexibility and is technically Open Source (as long as you have no commercial interest).

A potentially large barrier to use is getting the data into the right format. For instance, the official *How To Use* (`http://www.highcharts.com/documentation/how-to-use`) goes into some detail on loading in data, including writing your own parser for CSV files. I would instead recommend using R to prepare and export the data to JSON (see Section 4.5.1), which makes the process substantially less work.

A bargraph in Highcharts

Refer to the examples that come with the download.

For reference and comparison purposes, the `bar-basic` example JavaScript code (excluding HTML code, data and blank spaces) is 60 lines and 738 characters.

4.4 Low-level Languages

As Low-level Languages can be used to create custom visualisations from scratch, I will use each language to create roughly the same visualisation. I will call this the **Test Bargraph**.

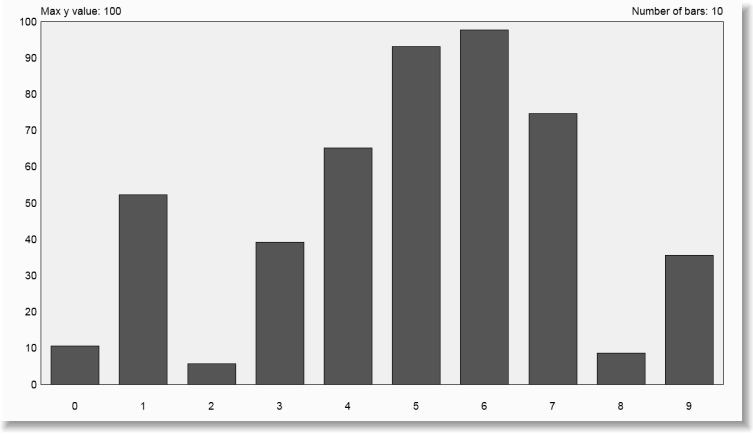


Figure 4.7: The **Test Bargraph** that will be used to explore each Low-level Language. I will attempt to recreate this same bargraph as closely as possible with every Low-level Language covered.

The Test Bargraph must have the following basic features:

Size Any reasonable arbitrary size of the Canvas or SVG image can be specified, including aspect ratio.

Number of Bars Can handle any positive integer.

Max Value I will be generating a random number between 0 and an arbitrary maximum value. The graph, including the y axis label, should be able to handle this.

Margins The width of the margins around the central plot can be specified. This is also where the axis labels will go, so there is a minimum constraint. For simplicity all my examples will simply use margin sizes that are proportionate to the total Size, but it should be possible to specify a fixed number (say to perfectly fit the axis labels).

To test how difficult it is to add interactivity, the Test Bargraph will also enable the user to alter the heights of the bars. This will involve the following elements:

Ghost Bar A 'ghost bar' tracks the position of the mouse and displays the potential height of the updated bar.

Bar Updates Upon a mouse click, the height of the bar is updated to the new value based on the position of the mouse.

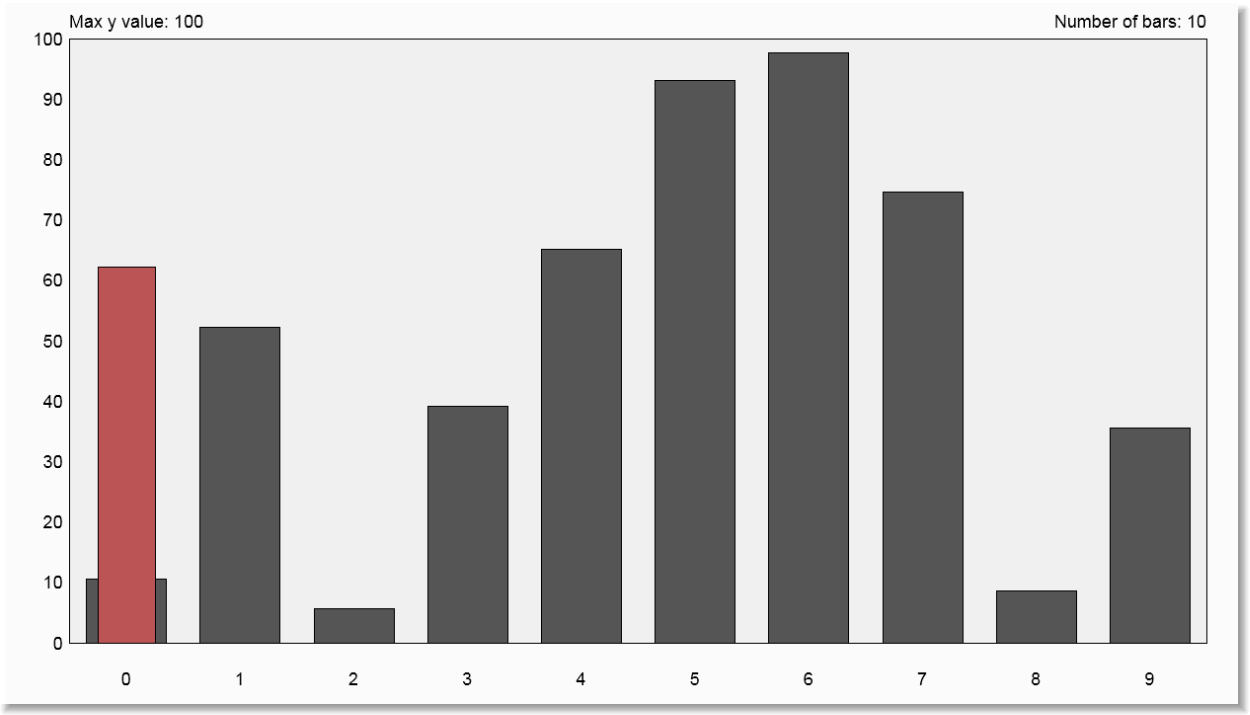


Figure 4.8: The **Test Bargraph** showing a 'ghost bar' which displays the potential height of the updated bar if the user clicks. Note that this screenshot does not capture the mouse pointer.

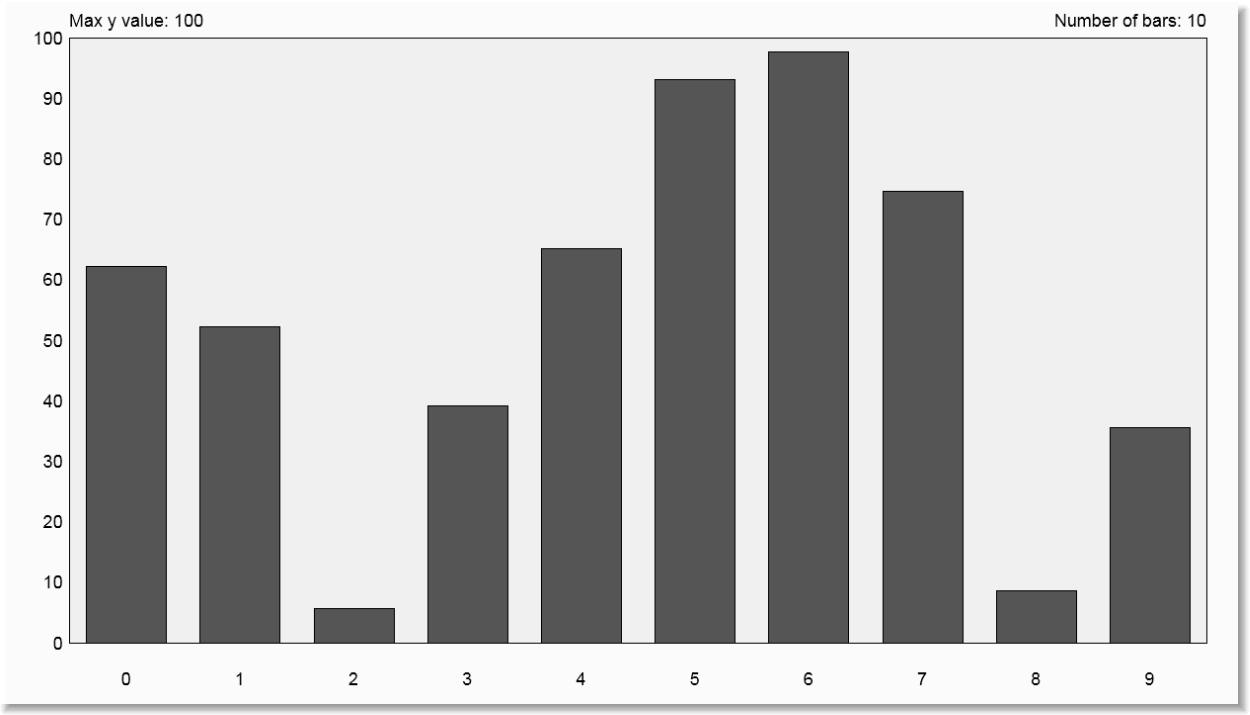


Figure 4.9: Upon clicking in Figure 4.8, the bar height is updated.

4.4.1 Processing.js

http://processing.org/
http://processingjs.org/

Key Points

Learning Difficulty Hard (Moderate if experienced in Java, JavaScript or Java-like languages).

Has a library of standard plots No.

Output HTML5 Canvas.

Interactivity Broad potential, limited only by what the user can code. No native object support.

Legal Processing: Open Source (GPL, version unknown).
Processing.js: Open Source via Seneca's *Centre for Development of Open Technology.*

Main Features

- Despite being based on Java, can generally be considered a language on its own, meaning almost everything needed to learn and write Processing code can be found in one place (the Processing API Reference).
- Extending functionality to include interactivity easy.
- Have to build graphics from scratch using low-level functions.
- Can be used in two ways, either writing *Processing* code directly, or using *Processing.js* as a JavaScript library.

Comments

Processing is a *Low-level* language for visualisations in general, though often of an interactive nature. Because of this, even simple plots like a bargraph requires a fair level of coding to build from scratch using low-level functions.

No generalised plotting library was found, but it is quite possible to write a library of low-level plotting aid functions that would make creation of new graphs almost as easy as R's default graphics system.

The *Processing* language is based on Java, and *Processing.js* is a JavaScript implementation that essentially compiles *Processing* code into JavaScript to run. Due to the way *Processing.js* works, a user has two choices when coding. One can either write *Processing*

code, or one can choose to use *Processing.js* as a JavaScript library, in effect gaining access to the *Processing* API while writing JavaScript.

Choosing to write in *Processing* code leaves open the possibility of compiling this using the original *Processing* software, which may lead to performance gains. However, *Processing.js* is generally easier to use than Processing, as the required file is small (403 KB) and requires no installation beyond a modern web browser, which should already be installed on any modern operating system.

The use of the HTML5 Canvas element as the output attracts with it certain properties (see Subsection 4.5.4), perhaps the biggest being that there is no native support for objects. This can make certain kinds of interactivity more difficult than it would be if objects were recognised.

The redraw framerate (which also affects how quickly interactive elements respond) can be specified using the `frameRate()` function, and the live framerate can be checked via the `frameRate` variable (e.g. by inserting `println(frameRate);` somewhere in the `draw()` function).

A bargraph in Processing

A working example of the **Test Bargraph** in *Processing* can be found at `http://www.stat.auckland.ac.nz/~joh024/Research/Processingjs/`. The HTML document is named `test_bargraph.html` (which contains the required HTML code), while the Processing code (with comments) is named `test_bargraph_fullcomments.pde`.

Getting the desired interactivity to work requires matching the coordinates to the 'objects'. We must first check if the mouse is inside the graph region (and not in the margins), then match the `x` coordinate of the mouse to the appropriate bar. Once this is accomplished, it is trivial to then draw the *Ghost Bar* based on the mouse position, and also to have the matching bar update on mouse click.

For reference and comparison purposes, the *Processing* code (excluding data and blank spaces) is 84 lines and 2071 characters. Due to the excessive features (low-level plotting aid functions for generality, demonstrative interactivity), this is not directly comparable to the *High-level* languages, but can be loosely compared to other *Low-level* languages.

As I chose to write *Processing* code rather than using *Processing.js* as a JavaScript library, the inclusion of this code is slightly different to the usual procedure for JavaScript libraries. As usual, I must link to the JavaScript library (in this case, *processing.js*), but then I must use include a `canvas` element specifying the `data-processing-sources` attribute to be the *Processing* file.

```
<html>
  <head>
```

```
    <meta lang="en" charset="utf-8">
    <script src="processing.js"></script>
    <canvas data-processing-sources="processingjs_bar.pde"></canvas>
  </head>
</html>
```

Alternatively, one could use *Processing* (rather than *Processing.js*) to compile the `pde`. Several outputs are possible, including an HTML output or a stand-alone executable (which requires Java). A caveat, *Processing* is slightly stricter in grammar than *Processing.js*. For instance, in the example code there are cases where a `float` is implicitly coerced into an `int`. This is fine in *Processing.js*, but will cause errors in *Processing* (the coercion must be explicit using `int()`).

4.4.2 Paper.js

http://paperjs.org/

Key Points

Learning Difficulty Hard (Moderate if experienced in JavaScript).

Has a library of standard plots No.

Output HTML5 Canvas.

Interactivity Some potential, but not all object properties will cause the object to update in real-time restricting possibilities. Some internal object support.

Legal Open Source (MIT License).

Main Features

- JavaScript library for visualisations in general. Thus will require at least passing familiarity with JavaScript.
- Extending functionality to include interactivity cumbersome due to a lack of documentation on which specific properties can be altered dynamically. Thus potential is limited.
- Internal tracking of individual objects, allowing for object-based interactivity, despite using Canvas.
- Have to build graphics from scratch using low-level functions.

Comments

Paper.js is a JavaScript library for visualisations in general. Because of this, even simple plots like a bargraph requires a fair level of coding to build from scratch using low-level functions.

No generalised plotting library was found, but it is quite possible to write a library of low-level plotting aid functions that would make creation of new graphs almost as easy as R's default graphics system.

It can use a so-called *PaperScript*, this is merely JavaScript extended slightly. Thus using *Paper.js* does require some familiarity with JavaScript.

Though *Paper.js* outputs to a HTML5 Canvas element it internally tracks objects, allowing for object based interaction. However, this object system is not quite as robust as something like SVG.

A bargraph in Paper

A partially working example of the **Test Bargraph** in *Paper.js* can be found at `http://www.stat.auckland.ac.nz/~joh024/Research/Paperjs/test_bargraph.html`, all the code (including the JavaScript code) is embedded inside the HTML file. Of all the Low-level Languages covered, *Paper.js* performed the worst.

- I found no way to control vertical justification of text. If it does exist (and I assume some method exists) it is very difficult to find.

- Though the internally tracked objects have several properties, such as `width`, `height`, `left`, `topLeft`, etc. most of these do not work dynamically. That is, one cannot adjust an object's size or position by changing these properties, and only a certain few properties work (no documentation was found clearly stating which ones work). This frustrating limitation has meant the numerous attempts to make the **Test Bargraph** interactivity work has failed. Based on the properties I know work dynamically, the only possible way I can think of to make the interactivity work requires scaling of the bars by computing the relative size of the new bar height to the previous height. This was considered to be too cumbersome to be bothered with, thus the example is only partially working, lacking the interactive features.

For reference and comparison purposes, the *Paper.js* code (excluding data and blank spaces) is 71 lines and 2277 characters. Due to the excessive features (low-level plotting aid functions for generality, but lacking interactivity), this is not directly comparable to the *High-level* languages, and is also not very comparable to other *Low-level* languages due to the incomplete nature of the example.

4.4.3 Raphaël.js

http://raphaeljs.com/

Key Points

Learning Difficulty Very Hard (Hard if experienced in JavaScript, Moderate if also familiar with the SVG specification).

Has a library of standard plots Yes but limited, see http://g.raphaeljs.com/.

Output SVG, with support for VML for compatibility with older versions of Internet Explorer.

Interactivity Broad object-based interactivity.

Legal Open Source (MIT License).

Main Features

- JavaScript library for visualisations in general. Thus will require at least passing familiarity with JavaScript.
- Extending functionality to include interactivity easy.
- Have to build graphics from scratch using low-level functions.
- As the output is SVG, specifying stylistic features, such as an object's colour, line width, etc., requires knowledge of the relevant SVG object specification.
- Does not give finer control over the specification of the SVG.

Comments

Raphaël is a JavaScript library for visualisations in general. Because of this, even simple plots like a bargraph requires a fair level of coding to build from scratch using low-level functions. As it is a JavaScript library, some familiarity with JavaScript is required.

Raphaël essentially treats SVG as another type of drawing canvas, handling many of the underlying SVG specifications via wrapper functions like `Paper.rect`. Thus, an in-depth understanding of the SVG specifications is not required, however, if changes to specific attributes are desired (e.g. colour, stroke width, etc.), then one must still refer to the relevant SVG specification. As *Raphaël* treats SVG as another type of canvas, this does limit the degree of control over the exact SVG specification. For instance, SVG's grouping feature (`g`) cannot be used with *Raphaël*, and certain optional attributes may not be available.

No generalised plotting library was found, but there is a very basic plotting library called *gRaphaël*. Usage is similar to a *High-level* language, but functionality is limited. It does

however provide a basis that can be extended in more interesting directions. Additionally, it is quite possible to write a library of low-level plotting aid functions that would make creation of new graphs almost as easy as R's default graphics system.

A bargraph in Raphaël

A working example of the **Test Bargraph** in *Raphaël* can be found at http://www.stat.auckland.ac.nz/~joh024/Research/Raphaeljs/test_bargraph.html, all the code (including the JavaScript code) is embedded inside the HTML file.

Due to the interactivity required for the Test Bargraph, the object-based interactivity does not help us. In fact, it poses a barrier as the various bar objects will interfere with capturing mouse movement. One workaround is to draw a transparent rectangle over the entire graph region and working off this surface. The rest works similarly to a raster graphic (see *Processing*).

For reference and comparison purposes, the *Raphaël* code (excluding data and blank spaces) is 78 lines and 2382 characters. Due to the excessive features (low-level plotting aid functions for generality, demonstrative interactivity), this is not directly comparable to the *High-level* languages, but can be loosely compared to other *Low-level* languages.

4.4.4 D3.js (Data-Driven Documents)

http://d3js.org/

Key Points

Learning Difficulty Extremely Hard (Very Hard if experienced in JavaScript, Hard if also familiar with the SVG specification).

Has a library of standard plots Examples available, but not as high-level functions.

Output SVG, but the nature of D3 allows for any document-based output (e.g. a simple HTML document containing only text).

Interactivity Broad object-based interactivity, generally limited only by what the user can code.

Legal Open Source (BSD 3-Clause License).

Main Features

- JavaScript library for "manipulating documents based on data" to create data-based visualisations. It requires passing familiarity with JavaScript, HTML and

anything else desired in that document, which in the case of using it for graphics will mean SVG and CSS.

- The requirement of having to understand several almost independent standards (HTML, JavaScript, SVG and CSS), then bring them together by using the D3 API, results in a very high barrier to entry.
- Extending functionality to include interactivity easy, assuming user familiarity with all the required components.
- Have to build graphics from scratch using low-level functions.
- As the output is SVG, specifying stylistic features, such as an object's colour, line width, etc., requires knowledge of the relevant SVG object specification.
- Gives a high level of control and flexibility over the specification of the document, and in turn the SVG image(s).

Comments

D3.js is a JavaScript library intended to be *Transformative* rather than *Representative*. That is, unlike the other Low-level Languages where the user uses the library's API to draw a rectangle here, or a circle there, *D3.js* is instead used to create a structured document, and in the case of the SVG that document describes an image. So the user might define a *circle* element with certain attributes, or a *rect* element, etc. This 'document' can then be rendered into an image with a modern browser.

In practice what this means is the user must have some understanding of all the components involved, which with visualisation would include HTML, JavaScript, *D3.js*, SVG and CSS. Learning all these components, even to the basic level to begin to use *D3.js*, poses a significant challenge, and that is the biggest disadvantage in attempting to use *D3.js*. However, once these components are learnt *D3.js* gives incredible control over the output, as the user is in effect writing the HTML and SVG from scratch, but with the help of the *D3.js* API to make things easier.

Convenience features *D3.js* add include:

Selections A shorter and easy way to select objects (including elements in a structured document like XML or SVG), based on the W3C Selectors API (the same API used by CSS).

Attaching Data Allows data to be attached to a group of elements, for instance attaching 10 numbers to 10 `rect` elements to be used as heights for a bargraph.

Scales Easily create conversion functions to convert to and back from a particular coordinate space, such as in a graph region. Also provides an automatic way to generate

ticks appropriate for the scale (including consideration of numbers appropriate for human-reading).

Transitions Automatically makes the necessary calculations and adjustments to display an animated transition from one set of attributes to another. This can be used, for instance, to smoothly transition (over a specified period of time) the height of a bar from one value to another.

No generalised plotting library was found though there are examples available that draw some standard statistical plots, which can help in writing new code. Additionally, it is quite possible to write a library of low-level plotting aid functions that would make creation of new graphs almost as easy as R's default graphics system.

A bargraph in D3

A working example of the **Test Bargraph** in *D3.js* can be found at `http://www.stat.auckland.ac.nz/~joh024/Research/D3js/test_bargraph.html`, all the code (including the JavaScript code) is embedded inside the HTML file.

Due to the interactivity required for the Test Bargraph, the object-based interactivity does not help us. In fact, it poses a barrier as the various bar objects will interfere with capturing mouse movement. Due to the level of control *D3.js* grants us, it is possible to utilise the `pointer-events` property of SVG elements to effectively ignore the bars from registering on mouse events. The rest works similarly to a raster graphic (see *Processing*).

For reference and comparison purposes, the *D3.js* code (excluding data and blank spaces) is 94 lines and 2925 characters for the main code plus the CSS with 11 elements and 492 characters. Due to the excessive features (low-level plotting aid functions for generality, demonstrative interactivity), this is not directly comparable to the *High-level* languages, but can be loosely compared to other *Low-level* languages.

4.4.5 Other Low-level Languages

The tools covered Section 4.4 are by no means comprehensive. They were chosen as they were found to be popular and comparatively easy to use and implement (as they are simply JavaScript libraries), compared to tools which might use Java, which often requires the Java platform to be installed by both the developer (jdk - Java Developer Kit) and the user (jre - Java Runtime Environment), making implementation and distribution of output considerably more cumbersome. Here I briefly cover *some* other tools I came across but did not cover in depth.

Prefuse `http://prefuse.org/` Uses Java to produce Java-based graphics.

Flare http://flare.prefuse.org/ By the same developers as Prefuse. Uses ActionScript to produce SWF graphics. This has been shown to have some performance benefits compared to a tool like *D3.js* (Bostock et al. 2011) which may be worth the added hassle of using ActionScript and SWF.

Piccolo2D http://www.piccolo2d.org/ Uses Java and C# to produce Java-based or .NET-based graphics.

Protovis http://mbostock.github.com/protovis/ JavaScript Library for producing SVG output. Can be considered obsolete, developers moved on to *D3.js*.

4.5 Other Relevant Topics

4.5.1 R

http://www.r-project.org/

R is a free software environment for statistical computing and graphics. (R Core Team, 2014)

Using R to Prepare and Export the Data

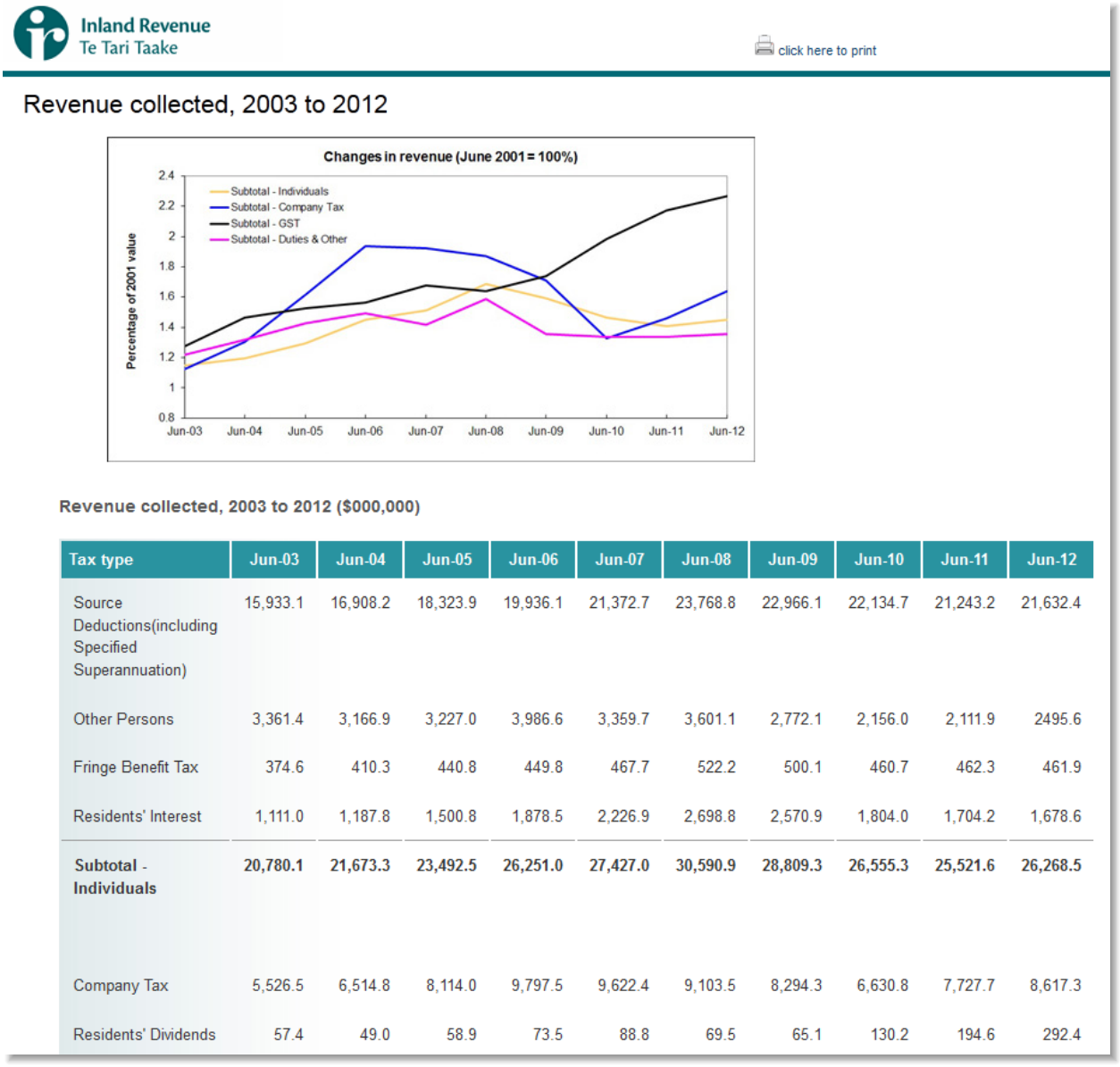
All the *High-* and *Low-level* language tools covered are JavaScript libraries. This means a data format that's very easy to use with these tools is the JSON (JavaScript Object Notation). Though JavaScript functions can be found to read some other data formats (such as CSV), I have found these to be comparatively cumbersome, and if any data cleaning or other preparation is required, working with JavaScript is not recommended.

One much better alternative is to use `R`. The base installation gives capabilities to easily read many simple data formats, including CSV (and variants thereof) and fixed width. More valuably, many packages extend functionality, such as the `XLConnect` (Mirai Solutions GmbH 2012) package which enables the reading of Excel files, even from Linux or Mac machines, or the `XML` (Lang 2012b) package which enables extraction of HTML Tables, potentially straight from websites (with the `RCurl` (Lang 2012a) package which allows `R` to be used as a text-web-browser).

Once the data has been read in, `R` provides extensive capabilities to manipulate the data, and if any statistical analysis is desired, this can also be conducted in `R`.

When the data is ready to be exported for use in a visualisation tool, a package such as `rjson` (Couture-Beil 2012) makes it very easy to convert any of `R`'s accepted data formats to JSON. This result can then be saved to a `.json` format and read in, or alternatively it can be copied directly into your JavaScript as a vector or array.

As a practical example of using `R`, the following code demonstrates how to download a HTML Table directly from the IRD website (Inland Revenue, 2012), convert to JSON, then write to a `.js` file for easy access by any JavaScript code.



Revenue collected, 2003 to 2012 ($000,000)

Tax type	Jun-03	Jun-04	Jun-05	Jun-06	Jun-07	Jun-08	Jun-09	Jun-10	Jun-11	Jun-12
Source Deductions(including Specified Superannuation)	15,933.1	16,908.2	18,323.9	19,936.1	21,372.7	23,768.8	22,966.1	22,134.7	21,243.2	21,632.4
Other Persons	3,361.4	3,166.9	3,227.0	3,986.6	3,359.7	3,601.1	2,772.1	2,156.0	2,111.9	2495.6
Fringe Benefit Tax	374.6	410.3	440.8	449.8	467.7	522.2	500.1	460.7	462.3	461.9
Residents' Interest	1,111.0	1,187.8	1,500.8	1,878.5	2,226.9	2,698.8	2,570.9	1,804.0	1,704.2	1,678.6
Subtotal - Individuals	**20,780.1**	**21,673.3**	**23,492.5**	**26,251.0**	**27,427.0**	**30,590.9**	**28,809.3**	**26,555.3**	**25,521.6**	**26,268.5**
Company Tax	5,526.5	6,514.8	8,114.0	9,797.5	9,622.4	9,103.5	8,294.3	6,630.8	7,727.7	8,617.3
Residents' Dividends	57.4	49.0	58.9	73.5	88.8	69.5	65.1	130.2	194.6	292.4

Figure 4.10: A screenshot of the webpage containing the HTML Table from which we will obtain our data. **Source:** Inland Revenue and licensed by Inland Revenue for re-use under the Creative Commons Attribution 3.0 New Zealand Licence.

```
1 ## Load required libraries
2 library(RCurl)
3 library(XML)
4 library(rjson)
5
```

```
## Load webpage and get the data table
URL = getURL("http://www.ird.govt.nz/aboutir/external-stats/revenue-refunds/tax-revenue/printable/")
htmlpage = htmlParse(URL, asText = TRUE)
datatable = readHTMLTable(htmlpage, stringsAsFactors = FALSE)[[1]]
```

```
> datatable[1:9,1:5]
                          V1       V2       V3       V4       V5
1                   Tax type   Jun-03   Jun-04   Jun-05   Jun-06
2         Source Deductions 15,933.1 16,908.2 18,323.9 19,936.1
3             Other Persons  3,361.4  3,166.9  3,227.0  3,986.6
4         Fringe Benefit Tax    374.6    410.3    440.8    449.8
5       Residents' Interest  1,111.0  1,187.8  1,500.8  1,878.5
6    Subtotal - Individuals 20,780.1 21,673.3 23,492.5 26,251.0
7
8               Company Tax  5,526.5  6,514.8  8,114.0  9,797.5
9      Residents' Dividends     57.4     49.0     58.9     73.5
```

```
## Clean data table
colnames(datatable) = datatable[1,]
datatable = datatable[-c(1, 6:7, 12:13, 15:16, 21:23),]
```

```
> datatable[1:9,1:5]
                   Tax type   Jun-03   Jun-04   Jun-05   Jun-06
2         Source Deductions 15,933.1 16,908.2 18,323.9 19,936.1
3             Other Persons  3,361.4  3,166.9  3,227.0  3,986.6
4         Fringe Benefit Tax    374.6    410.3    440.8    449.8
5       Residents' Interest  1,111.0  1,187.8  1,500.8  1,878.5
8               Company Tax  5,526.5  6,514.8  8,114.0  9,797.5
9      Residents' Dividends     57.4     49.0     58.9     73.5
10 Foreign Source Dividends    153.6    138.6    188.4    160.1
11    Non-residents' Income    731.7    799.9    926.7  1,095.6
14                  IRD GST  7,393.9  8,467.9  8,838.6  9,054.4
```

```
## Convert to JSON and write to a .js that can be sourced directly for use as a JavaScript array
writeLines(paste("var IRDTable =", toJSON(datatable)), "IRDTable.js")
```

A few notes:

- The row name for Source Deductions has been trimmed for space.
- In general, the numbers should be processed in R to remove any commas before exporting. This has been skipped above for clarity. For completeness the code required for this particular example is:

```
cbind(datatable[1], apply(datatable[,-1], 2, function(x)
  as.numeric(gsub("-$", "0", gsub(",| ", "", x)))))
```

  This handles commas in numbers, erroneous spaces and 0 being represented as `"-"` (problems not shown in the screenshot).

Visualisations with R

`R` has extensive visualisation capabilities mostly focused on traditional offline static graphics, however certain packages extend functionality in ways worth mentioning.

RGL http://rgl.neoscientists.org/about.shtml Has the potential to produces 3D output using OpenGL, which utilises the GPU for a very high level of performance.

RGGobi http://www.ggobi.org/rggobi/ Access GGobi from `R`, a powerful tool enabling interactive dynamic graphics, though only offline and not so easily shared. Learning curve for the tool is moderate to high.

iPlots http://www.rosuda.org/iplots/ Interactive Java-based graphics, though only offline and not so easily shared.

playwith https://code.google.com/p/playwith/ Adds a GUI for editing and interacting with `R` plots.

Of greater relevance however is the `gridSVG` (Murrell and Potter 2015) package. This enables graphics drawn in `grid` (R Core Team 2014) (which include plots drawn by `lattice` (Sarkar 2008a) and `ggplot2` (Wickham 2009)) to be exported to SVG, while still retaining much of the structured information. This output can then be manipulated from within `R` through the `XML` package, or even extended using a tool such as `D3.js`, opening the possibility of extending existing `R` graphics to become interactive web-based (SVG) output. Of course, doing this would require familiarity with all the tools involved, including `R`, `grid`, `gridSVG`, the `grid`-based plotting library (such as `lattice`), `SVG` and `D3.js`.

4.5.2 Raster graphics

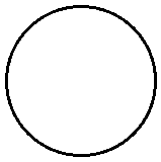


Figure 4.11: An example raster graphic of a circle. The original image is a 150 by 150 pixel image saved as a png.

The most familiar type of graphic, files like jpeg, png, gif and bmp, are *Raster graphics*. The image is defined by specifying each pixel. An easy way to tell is to zoom in on the image: the individual pixels will be revealed and any curves will become jagged.

Raster graphics has no concept of objects. There is simply the picture, as defined by what each pixel is. Thus in terms of computing power required to render the image, what is most important is how many pixels there are (the *resolution* or *dimensions* of the image).

A simple demonstration of this can be found at `http://www.stat.auckland.ac.nz/~joh024/Research/Processingjs/stress_testing.html`. Every second 1000 circles are drawn on a canvas with dimensions 1280 by 720, however the computing resources required to render the image does not increase. Zooming in or out of the image also poses little computational burden, though zooming in too far will reveal the individual pixels.

4.5.3 Vector graphics

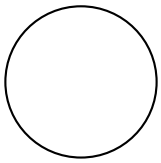


Figure 4.12: An example vector graphic of a circle. The circle is drawn in LaTeX using the `tikzpicture` package.

Perhaps more familiar than people may realise, text on a pdf file can be considered to be *Vector graphics*. The 'image' (in this case the letters of the text) is defined by code, and is drawn by the rendering software (in this case your pdf viewer). As you zoom in or out, the rendering software redraws, meaning the image always looks good. However this also means the drawn image can look subtly different depending on the rendering software used.

As *Vector graphics* are defined by code, it is possible to identify each individual object. However, this also means that each individual object must be rendered separately, which can become a computational burden when the number of individual objects becomes very large (e.g. a million).

A simple demonstration of this can be found at http://www.stat.auckland.ac.nz/~joh024/Research/D3js/stress_testing.html. Every second 1000 circles are drawn on a SVG image with dimensions 1280 by 720, resulting in an increase in CPU and RAM usage. Running the graphic for an extended period of time will cause severe performance loss including affecting the responsiveness of the renderer (web browser) itself. Zooming in or out of the image also results in a high computational burden as each individual circle must be redrawn, though this means the circles always look good at any zoom level.

4.5.4 HTML5 Canvas Element

Without going into too much technical detail, the HTML5 *Canvas Element* can be understood to be a drawing surface for *Raster graphics*, where the drawing can be specified by scripts. As the *Canvas* is a *Raster graphic*, objects cannot be separately identified or modified once drawn; dynamic graphics are achieved by redrawing the *Canvas* from scratch. However, web browsers are amazingly good at drawing and redrawing the *Canvas* quickly and high framerates are easily achieved.

This also means object-based interactivity is no easier than any other interactivity, and is accomplished via *coordinate matching* - that is, the coordinate of the mouse is checked to see if it 'hits' any 'object'. This hit detection is trivial for rectangular objects but becomes more difficult for increasingly complex shapes.

Also because of the *Raster* property the image will scale poorly (much like zooming too closely in a jpeg image). This can however be overcome by forcing a redraw of the image dynamically to match the zoom level, or to dynamically resize the Canvas resolution and redraw the image to match this resolution (allowing for a greater level of zoom before defects can be found).

See *Processing* (Subsection 4.4.1) or *Paper.js* (Subsection 4.4.2) for tools that use Canvas.

4.5.5 SVG

http://www.w3.org/Graphics/SVG/

The *Scalable Vector Graphics* (*SVG*) is an image that is specified via a XML file (but with extension `.svg`). As the name implies, it is a *Vector graphic*, that can be rendered by modern web browsers (like Firefox, Internet Explorer, etc.).

The standards are extensive. The pdf version of SVG 1.1 Second Edition[3] is 826 A4 pages. These standards can be considered to be a 'low-level graphics language' that can be used to draw (via a renderer like a web browser) anything from a rectangle to professionally typeset text. The following code specifies a SVG image of 150 by 150, drawing a rectangle

[3] http://www.w3.org/TR/SVG/

with upper-left corner at x = 25 and y = 25, with width and height of 100. The units are pixels at 100% zoom, but of course being a *Vector graphic* it can be scaled at will.

```
<svg width = "150" height = "150">
  <rect x = "25" y = "25" width = "100" height = "100"></rect>
</svg>
```

Being a *Vector graphic* it also understands objects. For instance that rectangle can be assigned a specific `id`, and further manipulation (via something like JavaScript) can be conducted on an object basis. Thus object-based interactivity (like clicking a specific object) is very easy. However, any interactivity that encompasses multiple objects may still require coordinate matching to work.

See *Google Chart Tools* (Subsection 4.3.1), *Highcharts* (Subsection 4.3.2), *Raphaël* (Subsection 4.4.3) or *D3.js* (Subsection 4.4.4) for tools that use SVG.

4.6 Conclusion

One great way to make sense of data is by way of visualisations; by using web-based output these visualisations become very easy to share and capable of some interesting and advanced interactivity. To aid in the creation of such web-based interactive visualisations there are a variety of Graphical Tools already available.

GUI tools and *High-level* languages are easy to use and enable the use of predefined visualisations. Though these tools offer some level of customisation, these are minor and never fundamentally add or remove features from the visual. To gain the freedom to extend existing visualisations to include new features (such as new interactive or dynamic features) or to create a wholly new type of visualisation, requires the use of serious graphical tools, broad graphical tools not limited to just statistical plots but capable of any visualisation - provided the user can code it in. Indeed, that is the major barrier to these *Low-level* languages: they require the learning and use of programming languages (typically JavaScript plus the API of the graphical tool plus any other applicable web standards, such as HTML, SVG and CSS), a barrier that can be considered insurmountable to a significant portion of the general public.

Thus there are opportunities to fill the gap between the very easy and the very difficult, a tool that can enable users who may have creative and innovative ideas for new visualisation methods, but do not have the programming expertise to actually make it.

Chapter 5

WeBIPP

5.1 Introduction . . . 105
5.1.1 Motivation . . . 105
5.1.2 Background . . . 105
5.1.3 Conceptual framework . . . 107
5.2 What is WeBIPP . . . 108
5.2.1 The Cookie Analogy . . . 109
5.2.2 Objectives . . . 109
5.2.3 Similar Tools . . . 111
5.3 How to Use . . . 113
5.4 Creating a Scatterplot . . . 115
5.4.1 The Model . . . 115
5.4.2 Read in the Data . . . 116
5.4.3 Set up the Axes . . . 117
5.4.4 Draw the Dots . . . 119
5.4.5 Adding Polish . . . 121
5.4.6 The Final Save Data . . . 124
5.5 Creating a Population Pyramid . . . 125
5.5.1 The Model . . . 125
5.5.2 Set up the Axes . . . 126
5.5.3 Resize the Frame . . . 129
5.5.4 Reposition the Frame . . . 131
5.5.5 Draw the Dotted Lines . . . 132
5.5.6 Draw the Bars . . . 135
5.5.7 Draw the Gender Label . . . 140
5.5.8 Re-using the Left-Side . . . 143
5.5.9 Functionise the Left-Side . . . 145
5.5.10 Use the new Function . . . 148
5.5.11 Draw the y-axis Labels . . . 152

5.5.12 Draw the Title . . . 154
5.5.13 Draw the x-axis Label . . . 156
5.5.14 Resizing the Finished Pyramid . . . 157
5.5.15 The Final Save Data . . . 159
5.6 How WeBIPP Works . . . 161
5.6.1 Frames . . . 162
5.6.2 IDvar . . . 164
5.6.3 Structure Diagrams . . . 164
5.7 Creating an Object Addon . . . 168
5.7.1 Define a Sub-Namespace . . . 168
5.7.2 Define the Icon . . . 169
5.7.3 Define the Attributes . . . 171
5.7.4 Define how to Set Attribute Values . . . 173
5.7.5 Define how to Get Attribute Values . . . 174
5.7.6 Define Interactions with the Graph Region . . . 174
5.7.7 Any other code, as necessary . . . 178
5.7.8 Summary . . . 178
5.7.9 Stylesheets . . . 178
5.7.10 Creating Complex Objects . . . 179
5.8 Creating a Value Interface . . . 183
5.8.1 Assign the Value Interface . . . 183
5.8.2 The Logical Interface . . . 184
5.8.3 The Numeric Interface . . . 184
5.9 Core's Set Attribute . . . 188
5.9.1 Handle useScale type . . . 188
5.9.2 Process value and discern valtype . . . 189
5.9.3 Handle auto useScale . . . 190
5.9.4 Apply Scale if needed . . . 190
5.9.5 Adjust low-level code . . . 191
5.10 Discussion and Limitations . . . 194
5.11 Conclusion and Future Work . . . 197

5.1 Introduction

5.1.1 Motivation

To restate the conclusions of the prior chapter's Literature Review, tools that are currently available for creating web-based graphics can be classified into two broad categories:

GUI - Which have a comparatively easy-to-use user interface but has limited customisation options.

Code - Which can have tremendous breadth and depth in terms of visualisations that can be created, but require coding knowledge and is not accessible to everyone.

The question arises then, if it is possible to bridge this divide, if it is possible to create a tool that has elements of being accessible and easy-to-use, while having the same level of power provided by *Code* tools. To create a Web-Based Interactive Plot Prototyping tool, also known as WeBIPP, that is entirely web-based, requiring no installation of software (beyond a modern web browser), while being capable of easily creating a broad variety of graphics from scratch, the way you can with a Low-level tool.

Before introducing such a tool, other background which has not been covered explicitly in this Thesis, such as past and current developments in statistical graphics, will be touched on briefly to better establish context for the work.

5.1.2 Background

The history of statistical graphics can go back a long time but as a starting point **Elements of Graphing Data** (Cleveland, 1985) is a good resource. In a chapter titled *Principles of Graph Construction*, Cleveland makes a number of recommendations, such as:

- Make the data stand out. Avoid superfluity.
- Use visually prominent graphical elements to show the data.
- Use a pair of scale lines for each variable. Make the data region the interior of the rectangle formed by the scale lines. Put tick marks outside the data region.
- Do not clutter the data region.

In sum, he gives guidelines on how the final graphic should look for it to be effective in communicating data. R's (R Core Team, 2014) base graphics is based on many of these principles, and while it has since been extended in many ways via packages, the base graphics has proven itself to be a good graphics system for producing a variety of statistical graphics.

The Cleveland approach to graphing data can be described as *top-down*, where the starting point is the 'top' (what the final graphic should look like), and one works their way down to the details that are required to produce this final graphic. The specific implementation details are not as important as long as the resulting picture looks the same as (or close to) what was originally desired. This contrasts with another popular approach to statistical graphics, the Grammar of Graphics (Wilkinson, 1999) approach, implemented in R via the packages `ggplot2` (Wickham, 2009) and `ggvis` (Chang and Wickham, 2015).

Wilkinson describes the Grammar of Graphics approach as being a three stage process: specification, assembly and display. This then would be a *bottom-up* approach, where one starts with the details of what to graph (the specification), and works their way to a graphic that displays these details (assembly and display). With a bottom-up approach, the appearance of the final graphic is of less importance and will vary by implementation. What is important is that the final graphic captures the particulars of the specification, and thus fulfils its purpose of graphing data.

In addition to the two approaches for crafting new graphics, there is also a history of tools for creating interactive statistical graphics. *Interactive* however can mean very different things depending on the context. In their paper for `SVGAnnotation`, Nolan and Lang (2012) note:

> The interactivity and animation described here are very different from what is available in more commonly used statistical graphics systems such as GGobi, iPlots, or Mondrian. Each of these provide an interactive environment for the purpose of exploratory visualization and are intended primarily for use in the data analysis phase. While one could in theory build similar systems in SVG, this would be quite unnatural. Instead, SVG is mostly used for creating presentation graphics that typically come at the end of the data analysis stage. Rather than providing general interactive features for data exploration, we use SVG to create application-specific interactivity, including graphical interfaces for a display.

WeBIPP is also a tool for creating presentation graphics, and thus deals with a different kind of interactivity to most existing interactive statistical graphics. Nonetheless, to provide broader context, some of the prior work that has been done in interactive statistical graphics will be covered.

A tool like `tinkerplots` (Konold, 2007) initially appears to tackle the same question as WeBIPP (though `tinkerplots` was designed foremost to aid in education), providing a GUI interface with basic operators to transform data into a graphic. However, these operators, while seeming general, are of a narrow scope, and the resulting graphics, though impressive in variety, consist of a limited, predefined set. This then makes it more similar to the GUI tools of the previous literature review, being polished, but limited in scope, compared to

the intent of WeBIPP, which is to create a tool capable of low-level control to enable the construction of wholly new graphics from scratch, as you can with a low-level tool.

Though `rggobi` (Swayne et al., 2004) has a GUI, it is better thought of as a High-level tool, and along with `cranvas` (Xie et al., 2013) and `iplots` (Urbanek and Theus, 2003), provide high-level functions to create predefined, interactive plots for data analysis. They additionally rely on other software being installed, all requiring R, `iplots` requiring the R package `rJava` (which additionally requires a separate install of the Java Development Kit), and `cranvas` requiring `qt` installed. As noted above however, WeBIPP is not a tool for data analysis, aims to provide lower-level building functions (not predefined plots) and is entirely web-based with minimal dependencies (in theory, just a modern web browser, which most computers should already have). Plotly (2014) is another High-level tool, though differing little from the High-level tools covered in the literature review of the previous chapter. It was created after the Literature Review was conducted, and aside from providing many language-specific APIs to make it more accessible, shares the same general characteristics, being designed for creating a limited set of predefined plots.

Tools have also been created for exporting R graphics into SVG, after which it becomes possible to use a web browser and add interactivity in this way. Already mentioned is `SVGAnnotation`, which works by saving an R graphic using the `svg` function and post-processing the file to identify the different components. Interactivity, animation, etc. can then be added using JavaScript with the help of identifiers. Though it contains several functions to make this easier, these are limited, meaning JavaScript knowledge is required for more interesting interactivity. Likewise, `animint` (Hocking et al., 2015) provides a way to export `ggplot2` graphics to SVG, but has limited options in adding interactivity to the result without writing JavaScript. Finally, `gridSVG` (Murrell and Potter, 2015) provide a way to export `grid` graphics to SVG, but again provides little to help with adding interactivity to the SVG. Thus these tools are export tools of limited scope, taking R graphics and exporting them to SVG, which differs greatly from what WeBIPP is intended to be, an entirely web-based tool for creating graphics from scratch.

5.1.3 Conceptual framework

Given this background, in addition to the background from the prior chapter's Literature Review, we can establish the conceptual framework for WeBIPP.

Anyone who has tried to create unconventional graphics in both systems will know that it is by far easier to do so in a top-down system, as to do so in a bottom-up requires you to convert your idea, almost certainly a picture of the final graphic, into a specification understood by the bottom-up system, and often details are lost in translation, meaning the final output may not even be precisely what you desired. On the other hand having a picture

and attempting to reproduce it is exactly what a top-down system is designed to do. For this reason, WeBIPP will use the top-down approach, as this is the more natural approach for creating new kinds of graphics, which is what WeBIPP is intended to facilitate.

To enable access to the full extent of the output format's capabilities, WeBIPP will use D3 to create an SVG image. Unlike other Low-level tools that provide an intermediary language that blocks direct access to the underlying format, D3 enables the complete access desired. As R's base graphics has proven itself to be a good graphics system for producing a variety of statistical graphics, and captures many of the important components of Cleveland's advice, using it as the model when developing WeBIPP should prove to be a good way to create a useful proof of concept. Thus WeBIPP will, at its core, possess elements of R base graphics, D3 and SVG.

As WeBIPP is designed for creating presentation graphics, frameworks for interactive graphics for data analysis, such as the MVC architecture or the reactive framework (Xie et al., 2014) are not appropriate for WeBIPP. Such models are of value when the output graphic is well-defined, and the challenge lies with managing the data pipeline that connects the data to the plot as it is manipulated. For WeBIPP however, the final graphic is far from well-defined, as it could potentially be anything. The tricky detail instead lies with bridging the divide between GUI and Code. WeBIPP addresses this by layering GUI, High-level and Low-level aspects. The GUI layer will provide the basic functionality to enable users to do things quickly and easily. Actions in the GUI will be translated and recorded as a set of High-level function calls, which can be manually adjusted for a greater degree of control that the GUI cannot provide. The High-level calls also serve as a record of the steps taken to create a graphic, and can be generalised and shared. The Low-level layer translates the High-level function calls into low-level D3 and JavaScript code that achieves the desired result, and can be adjusted for the complete level of control equalling the power provided by D3 on its own, ensuring that a using WeBIPP will never prevent the user from using the full extent and power of a lower-level tool. As the output is a presentation graphic, management of a data pipeline is not a major concern, though the layered framework means that an informal data pipeline is created in the form of discrete High-level function calls evaluated in a linear sequence. This could be exploited to create a more formal data pipeline for WeBIPP, if such a need arises in future development.

5.2 What is WeBIPP

What WeBIPP attempts to achieve can be hard to grasp, while at a conference a fellow PhD candidate said as such, then demanded an analogy involving cookies.

Cookies? Wait... Cookies!

5.2.1 The Cookie Analogy

Currently, there are two fundamental ways you can acquire cookies:

1. Go to a supermarket, there's an aisle of cookies of all sorts, brands and sizes. You pick one and buy it. You have now acquired a cookie easily but your choice was limited to what was available. If the cookie you wanted was not available, too bad. If the cookie you wanted is a little different, you *might* be able to do something, but your options are limited.

2. Bake the cookie yourself. Gather all the ingredients you want, prepare them the way you want, then bake it the way you want it. This obviously requires a certain level of cooking and baking skills, in addition to a lot of effort and time, but the cookie you get is the one you wanted.

Now, imagine you have an aisle in front of you, but instead of pre-packaged cookies, you have an assortment of features of a cookie. It's size, shape, dough, whether it has chocolate chips, etc. You then pick and choose what you want, and the cookie is assembled before your very eyes. But there's more! At the same time, the entire baking process is also available in front of you. The cookie is not magically appearing, what's really happening is that as you choose the features you want, all the necessary ingredients are created, then are prepared in the right way, and at the end of it (magically instantly, without actually consuming the ingredients), the cookie with the features you chose is made.

This baking process is not just there for show, you can go in and change it as you wish. Add some milk here, take away some sugar there, add some chocolate here. As you change it, the cookie is magically remade, incorporating any changes you have made. Now, if you have no cooking skills, you can ignore the cooking process, and it's similar to going to the supermarket, except you need to choose a few more things, and you get a slightly more customised cookie. But, if you have any cooking skill at all, you can make full use of it to fine-tune and add polish to the baking process. You are in no way limited by the choices initially presented, indeed you can think of it as an easy way to magically set up all the time-consuming and tedious tasks, then use your cooking skills to its full potential. This may be limited to simply adjusting the level of sugar to suit your needs, or it may be to add a masterful touch that makes the cookie truly special.

Unfortunately, such a cookie preparing process is impossible in reality. But an analogous tool may be possible for interactive graphics.

5.2.2 Objectives

WeBIPP aims to satisfy the following objectives:

1. Possess an interactive, graphical user interface that can produce visualisations quickly.
2. Does not require coding knowledge to use.
3. Is easy to utilise data in creating and modifying the visualisations.
4. Is easy to deploy, by being a purely web-based tool that requires nothing more than a modern browser.
5. Is extensible and easy for developers to create new addons that extend capability.
6. Keeps a record of actions taken in creating the visualisations such that steps (not just results) are fully reproducible, and thus reusable.
7. Can reuse the record of actions to reduce repetitive actions, in essence *functionising user interactions*.
8. Is easy to further manipulate the output with other tools, such that using WeBIPP does not limit the user.

WeBIPP can be thought of as a set of bridges that connect the GUI and the Code. To better understand this analogy consider *machine code*, commonly spoken about as *"the 1's and 0's computers use to do things"*. Machine code can be taken to be the lowest-level programming language, but is very difficult to work with and practically, higher-level languages are used instead, such as *Fortran* or *C*.

JavaScript, *HTML*, *CSS* and *SVG* are all examples of quite high-level "programming languages", or fairly human-friendly instructions that can be used to tell the computer what to do. The JavaScript library *D3* is even higher-level and can be used to bring these four languages together to produce web-based interactive visualisations. To put it another way, D3 provides an easier interface to bridge these languages together.

However there is some loss with each step towards easiness. There is generally a trade-off between control and convenience. For instance a lower-level language like Fortran gives control over memory allocation, while JavaScript handles it automatically making it more convenient. Likewise D3 handles many things automatically, hence making it more convenient to do common tasks, but you lose some control. Luckily as a JavaScript library, the programmer can always fall back on the lower-level languages (JavaScript, HTML, CSS and SVG) to retain that control where required, while using D3 for other tasks for improved coding efficiency.

WeBIPP makes use of D3 and also provides further functions at about the same level, to make it easier to produce web-based visualisations. It then further provides an even higher-level set of possible instructions, which in essence are convenient shorthands for writing

D3-level code. WeBIPP finally provides a GUI that makes it possible to write these higher-level instructions, via user interaction with the interface. Through this layering, WeBIPP makes it possible to provide multiple levels of convenience to cater to different users who require a different control-convenience trade-off, from those who wish to only use the GUI and not touch any code, to those who would sometimes prefer to write directly in JavaScript or similar level code.

Additionally, as the WeBIPP interface writes high-level WeBIPP code, these instructions act as a record of actions taken when creating the visualisation, a record that is reproducible and thus reusable.

5.2.3 Similar Tools

While there exist tools that possess an easy-to-use GUI front-end to produce a limited set of graphics (e.g. Tableau[1]), and tools that provide more flexibility and power but require coding (e.g. D3), the combination of both ease and flexibility is rare.

In early 2013 Bret Victor proposed *Drawing Dynamic Visualizations*[2,3], and introduced a tool that enabled the creation of graphics from scratch via a GUI and without the use of code. Later in early 2014, the alpha version of a tool called *Lyra* was released[4], which shared many similarities to the idea proposed by Bret Victor.

Conceptually these tools are similar to the idea behind WeBIPP, though there are some fundamental differences in motives resulting in significant differences in practice. Unlike the above tools where a key motivator appears to be to avoid coding, WeBIPP embraces coding. Rather than the GUI being an interactive way to create graphics from scratch, the WeBIPP GUI is an interactive way to write code; this code is written in the background and can be ignored, but the fact that it is code gives WeBIPP many advantages over not having code. Code can be extended, tweaked and reused. Consider:

- The user wishes to do something the interface does not allow for. With the other tools, the user is blocked from doing this (a difficult workaround may be possible by exporting the graphic into a different tool). With WeBIPP the user can simply adjust the code.

- The user finds the interface more cumbersome to do something they could do easier and faster by writing code. With the other tools, in order to enjoy the benefits of using any part of the tool, the user must use the tool for everything. With WeBIPP the

[1] http://www.tableausoftware.com/products/public
[2] http://vimeo.com/66085662
[3] http://worrydream.com/DrawingDynamicVisualizationsTalkAddendum/
[4] http://idl.cs.washington.edu/projects/lyra/

user is free to switch between using the interface and writing code manually, choosing whichever is faster and more convenient.

- The user has created something and wishes to do something very similar. With the other tools, the user usually must start again from scratch. With WeBIPP the user can take the code generated previously and tweak only the parts necessary to make the change. More, this code could then be generalised to create a template that can be shared with other users.

- A different user has created a nice graphic and another user wishes to know how. With the other tools, they only have the final graphic and must guess (or ask) how it was created. With WeBIPP as each action is writing code in the background, the entire process can be tracked closely. If the original creator exclusively used the interface, then the code is a complete record of the steps taken. Otherwise the code still gives a very good idea of the steps required to create the graphic.

In these ways WeBIPP is founded upon unique principles that make it a different beast from other tools that currently exist. However, these principles are not without their drawbacks (see Section 5.10).

5.3 How to Use

WeBIPP includes an integrated tutorial that covers the basics of its use. There are also tutorials available from the official website[5] that covers the steps to reproduce some example graphics. Like the tutorials on the website, this section will cover the steps to reproduce some example graphics. But unlike the tutorials on the website, the focus will be on exploring WeBIPP's interface and demonstrating some of the current interface's features. In some cases, demonstration will take precedence over clear instructions. While WeBIPP is intended to work in all modern browsers, it is developed on*Mozilla Firefox*[6], and is thus only guaranteed to be fully compatible with Firefox.

Figure 5.1 covers key elements of the Graphical User Interface, which are referenced in the tutorials that follow. It is also worth noting that WeBIPP draws to an SVG image. The base coordinate system for SVG images uses pixels and the origin `(0, 0)` of the coordinate space is the upper-left corner of the image. This is consistent with most other graphical systems, but may be confusing to those familiar with statistical graphics systems which commonly have the origin in the lower-left corner, with positive y values associated with 'up', rather than 'down'. WeBIPP implements arbitrary coordinate spaces via the use of Frames (see Subsection 5.6.1), and the default coordinate system for Frames follow the statistical graphics convention of the origin being on the lower-left corner. Objects added as children to Frames can obey both the SVG coordinate system, and the Frame's coordinate system. Which it obeys is determined by the `useScale` attribute, with a value of `true` indicating it will obey the Frame's arbitrary coordinate space, as will be demonstrated in the tutorials that follow.

The tutorials are presented primarily as a sequence of Figures with accompanying captions, rather than the usual format of Figures being accompaniments to the text. Longer captions are formatted as regular text for ease of reading, but still function as captions.

[5] https://www.stat.auckland.ac.nz/~joh024/Research/WeBIPP/
[6] https://mozilla.org/firefox

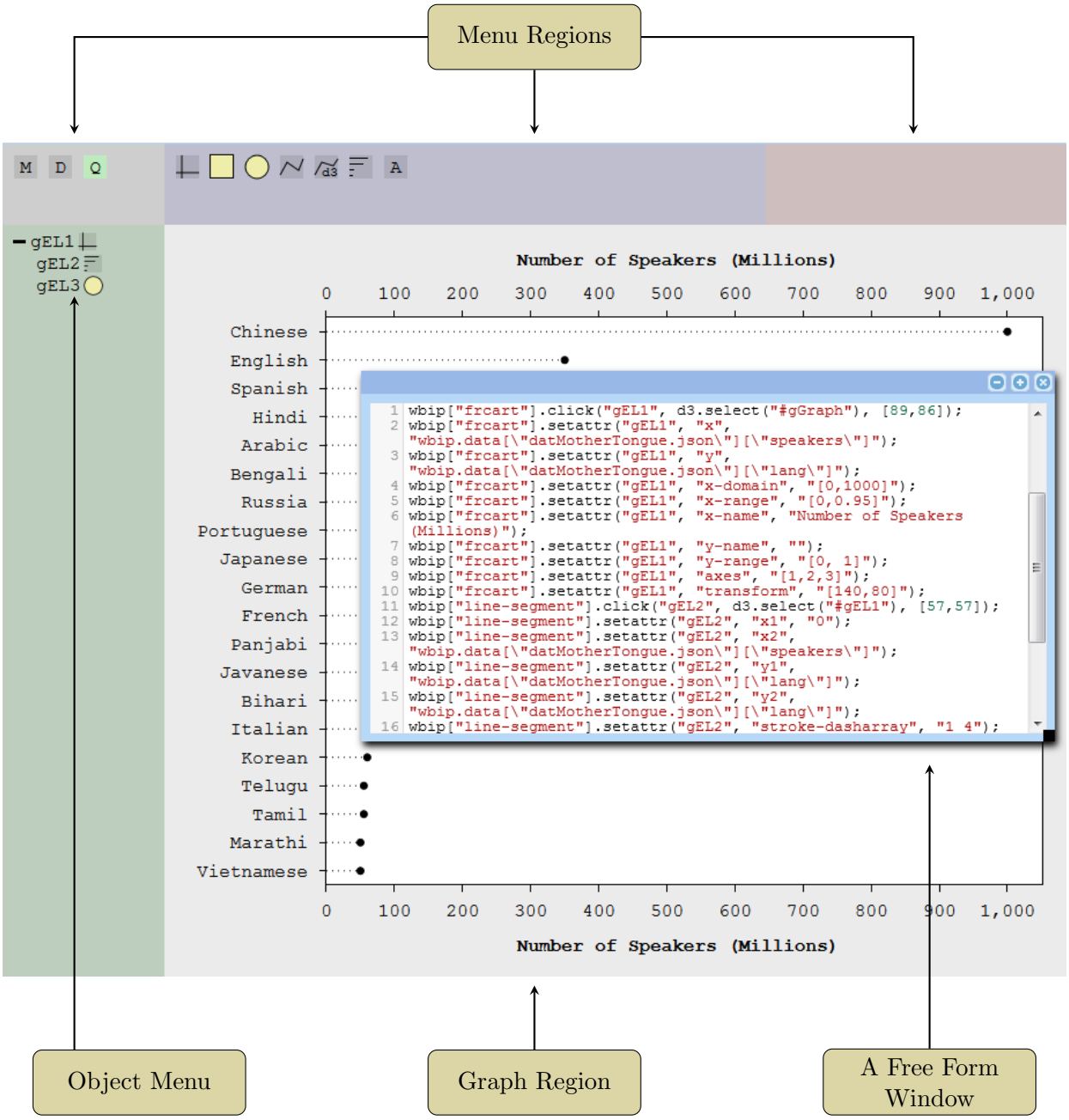


Figure 5.1: The layout of WeBIPP's Graphical User Interface. Free Form Windows, as their name implies, are not static elements of the interface, and can be moved about and closed. The Free Form Window shown here is the WeBIPP Code interface, an embedded and fully functional code editor.

5.4 Creating a Scatterplot

5.4.1 The Model

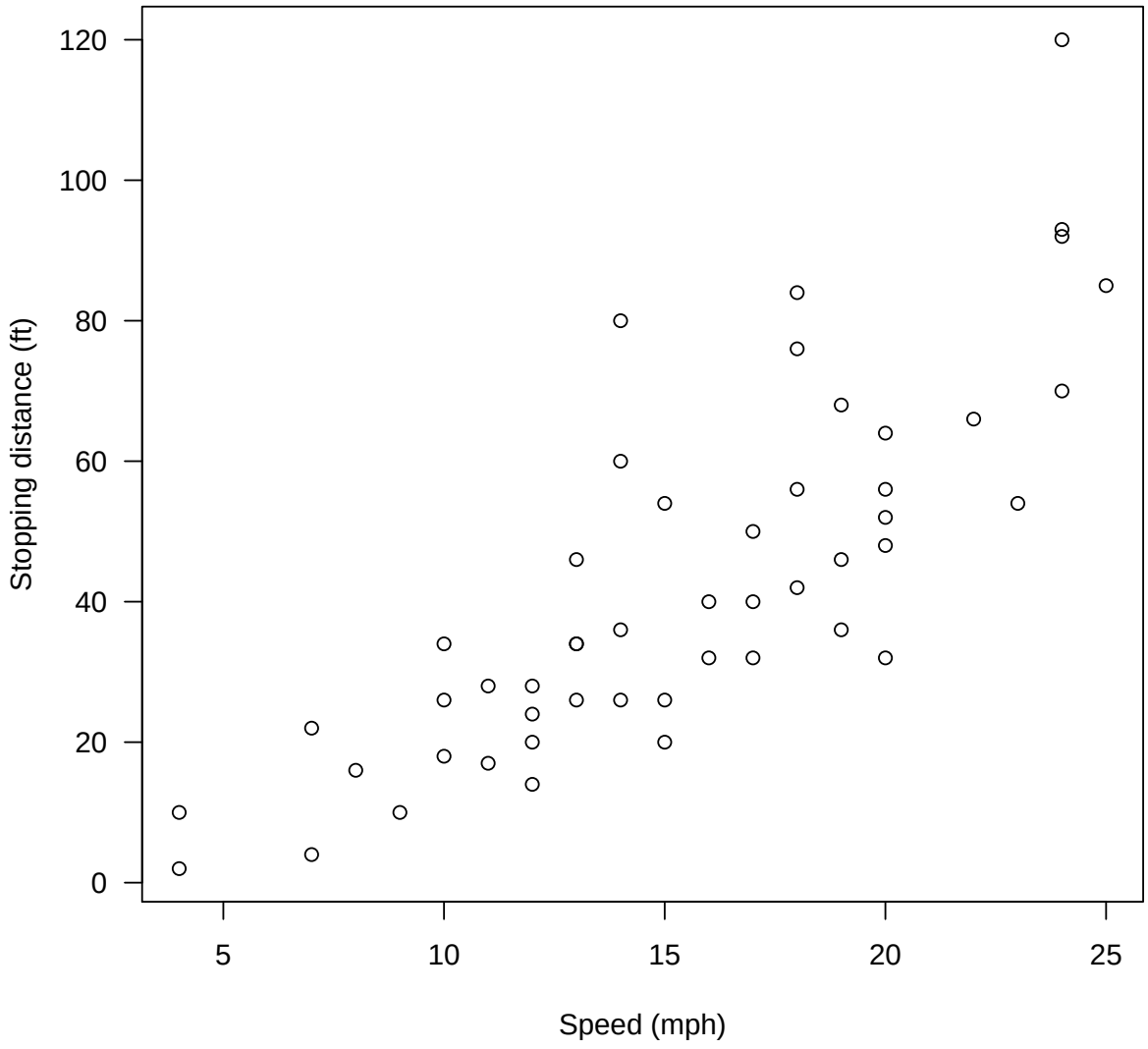


Figure 5.2: This scatterplot was drawn using R (R Core Team, 2014). The data is one of the datasets included with R. Using WeBIPP, we wish to recreate a graphic similar to this. The R command used to produce this image is:

```
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)", las = 1)
```

5.4.2 Read in the Data

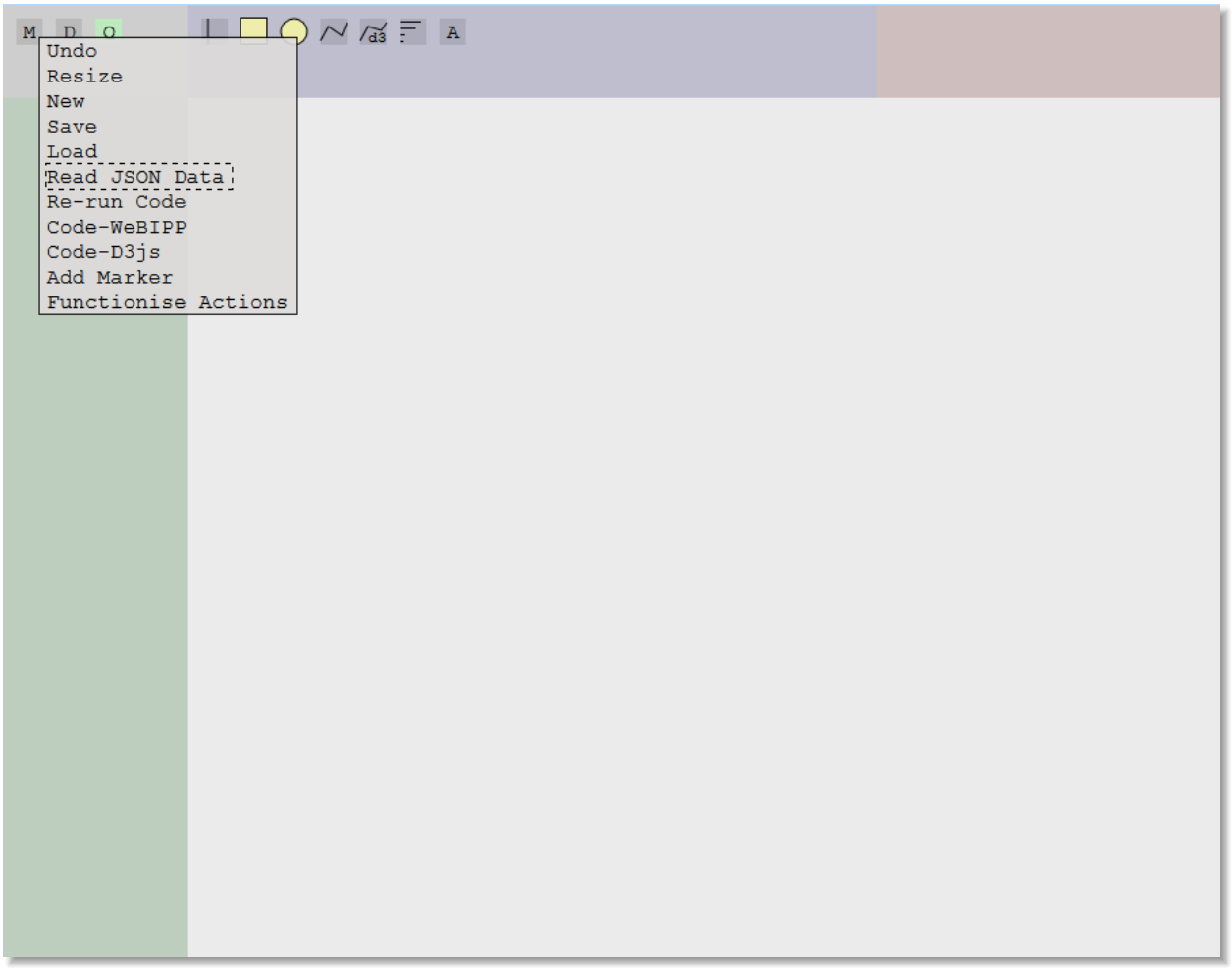


Figure 5.3: Read in the data to plot.

Figure 5.3: Load the data to plot by going to the main menu (click on the `M` button on the top-left) and clicking on *Read JSON Data*. Use the file navigation interface provided by your web browser to select the data file[7]. Once the data is loaded, the available data variables can be checked by opening the data viewer (click on the `D` button on the top-left).

WeBIPP currently only supports data in the JSON format as this is the easiest data format to work with in JavaScript. However requiring the user to convert their data to JSON is not ideal, hence support for other common data formats (such as csv) is one of the priorities for coming updates.

[7]The data for this example can be downloaded here:
https://www.stat.auckland.ac.nz/~joh024/Research/WeBIPP/Examples/datCars.json

5.4.3 Set up the Axes

Figure 5.4: Select the Cartesian Frame from the menu and click anywhere on the Graph Region to place a Cartesian Frame object.

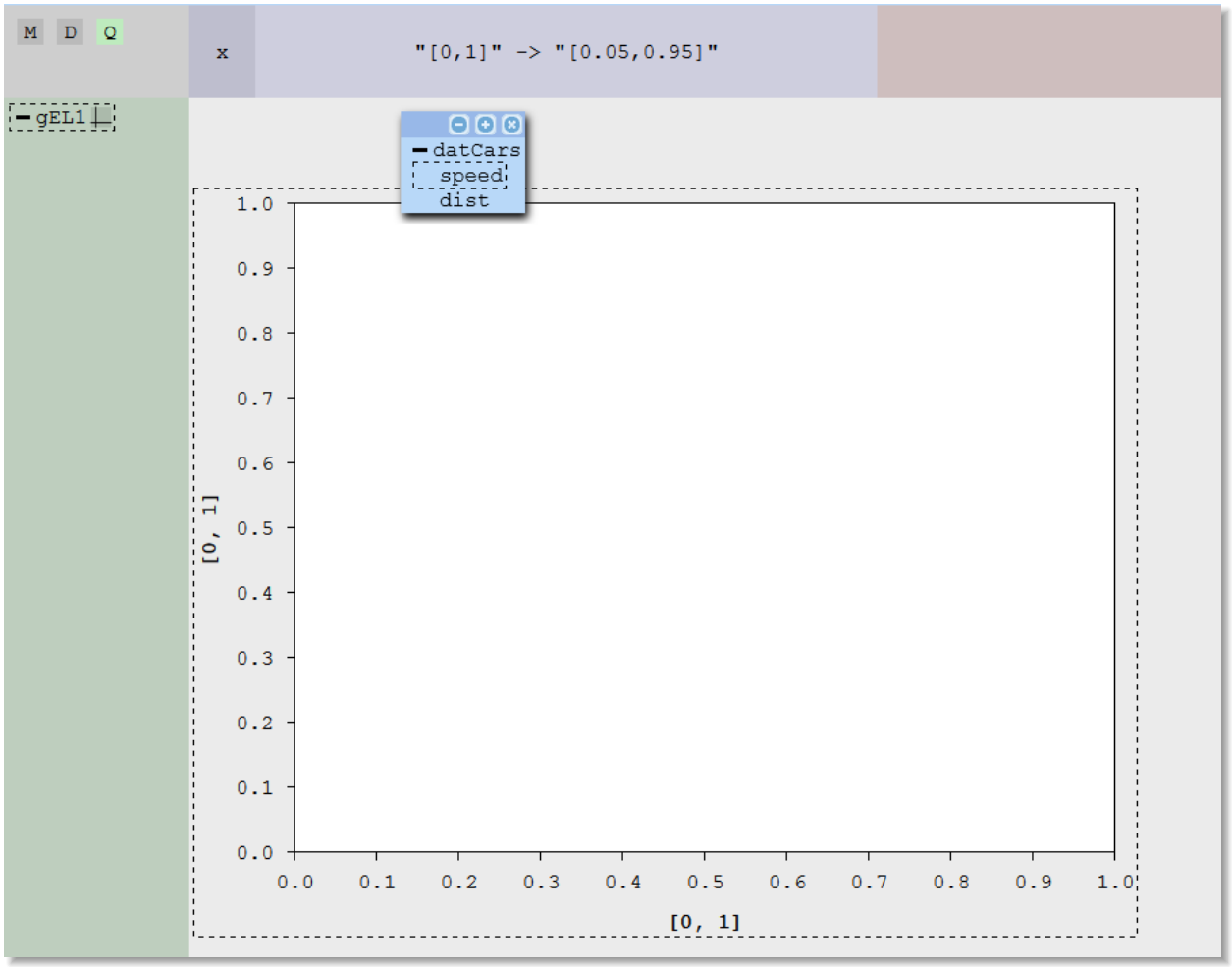


Figure 5.5: Assign `speed` from our data to the x-axis.

Figure 5.5: Select the Cartesian Frame from the Object Menu on the left. This will bring up the Attribute Interface on the top where the attribute `x` will be selected and displaying the current value of the attribute. The Data Viewer interface can be used to assign data directly to an attribute by clicking on one of the data variables. The Figure shows the data variable `speed` highlighted, but not yet clicked. See Figure 5.6 for how the interface is changed after the click.

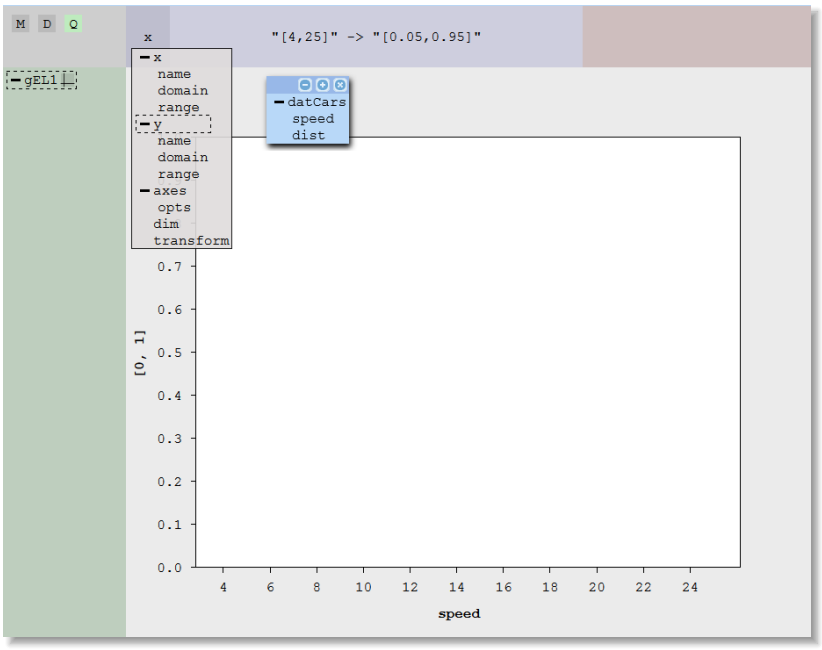


Figure 5.6: Click the attribute name (`x` currently) to open a list of attributes for the Cartesian Frame.

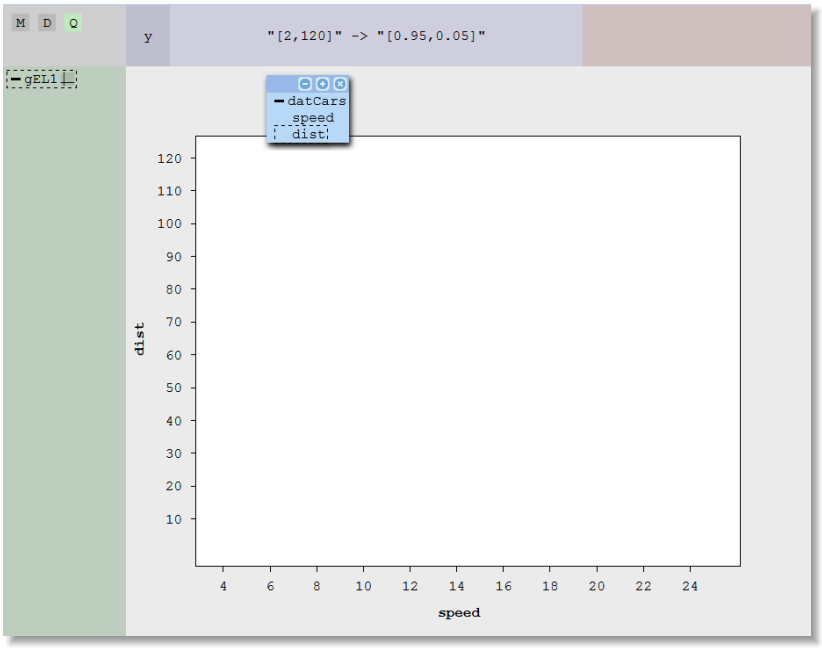


Figure 5.7: Select the attribute `y` and repeat the process from before to assign `dist` to the y-axis. We are done with the Cartesian Frame for now and we can unselect the object by clicking it again in the Object Menu.

5.4.4 Draw the Dots

Figure 5.8: By a similar process as before, a Circle may be placed inside the Cartesian Frame by selecting the Circle from the menu, then clicking anywhere inside the Frame.

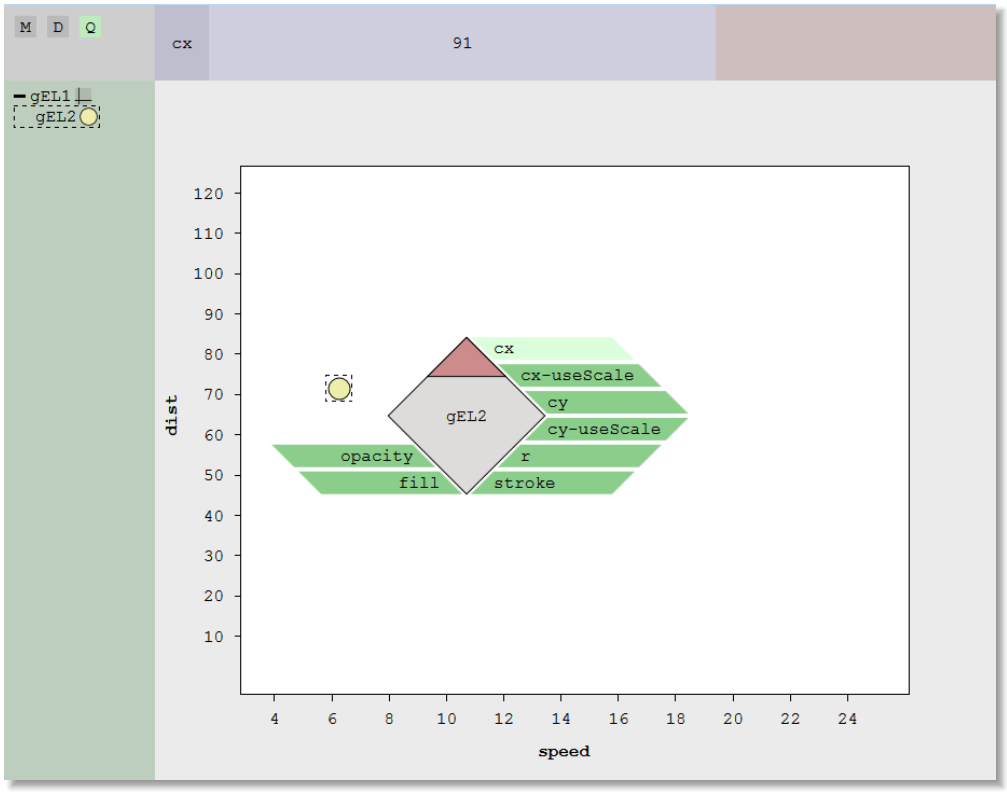


Figure 5.9: The Quick Menu is an alternative to selecting from the Object Menu and using the Attribute Interface. It is accessed by right-clicking directly on the Object in the Graph Region.

Figure 5.9: An alternative to the steps taken for Figure 5.5 is to use the Quick Menu. While the Quick Menu is enabled[8], right-clicking directly on any Object in the Graph Region will open a Quick Menu for that object. The grey part of the central diamond can be used to drag the Quick Menu into a more convenient position. The options on the sides may be selected to reveal further options (see Figure 5.10 and Figure 5.11). The red triangle at the top of the diamond will reverse the selection and ultimately close the Quick Menu if there is no selection to reverse.

[8]The `Q` button on the top-left can be clicked to enable or disable the Quick Menu. When enabled, it is highlighted in green. It may be disabled to make use of the browser's context menu, e.g. to inspect the SVG.

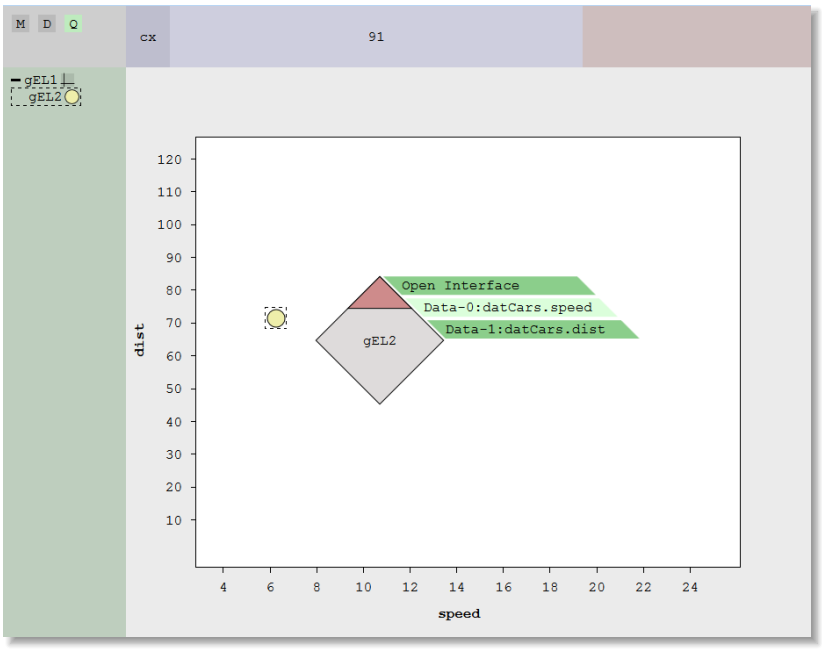


Figure 5.10: Assign `speed` from our data to `cx` of the Circle. The Figure shows the data variable `speed` highlighted, but not yet clicked. Once clicked, as the data contains multiple values, the Circle Object will replicate itself to become multiple circles, with each data value being assigned to a different circle. See Figure 5.11 for how the interface is changed after the click.

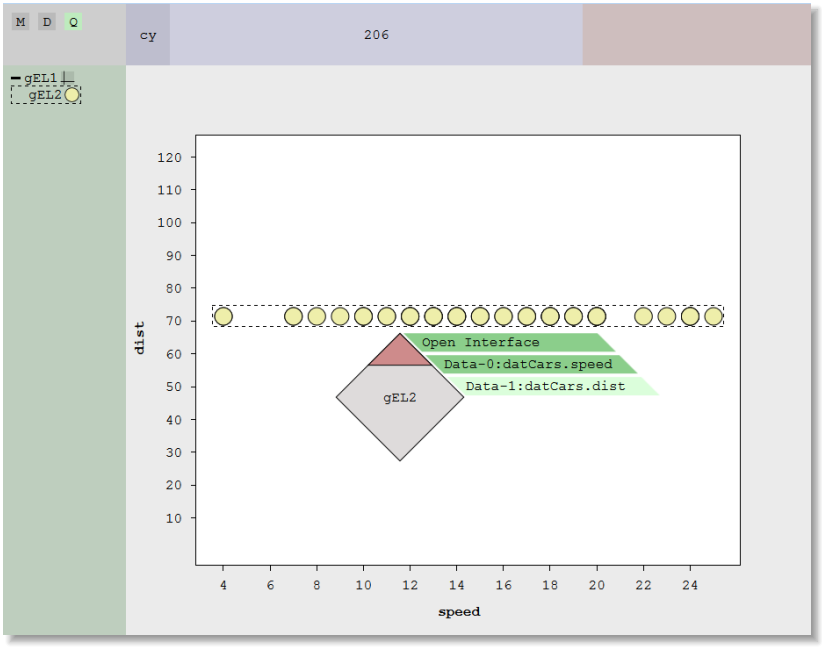


Figure 5.11: Assign `dist` from our data to `cy` of the Circle. The Figure shows the data variable `dist` highlighted, but not yet clicked. See Figure 5.12 for how the interface is changed after the click.

5.4.5 Adding Polish

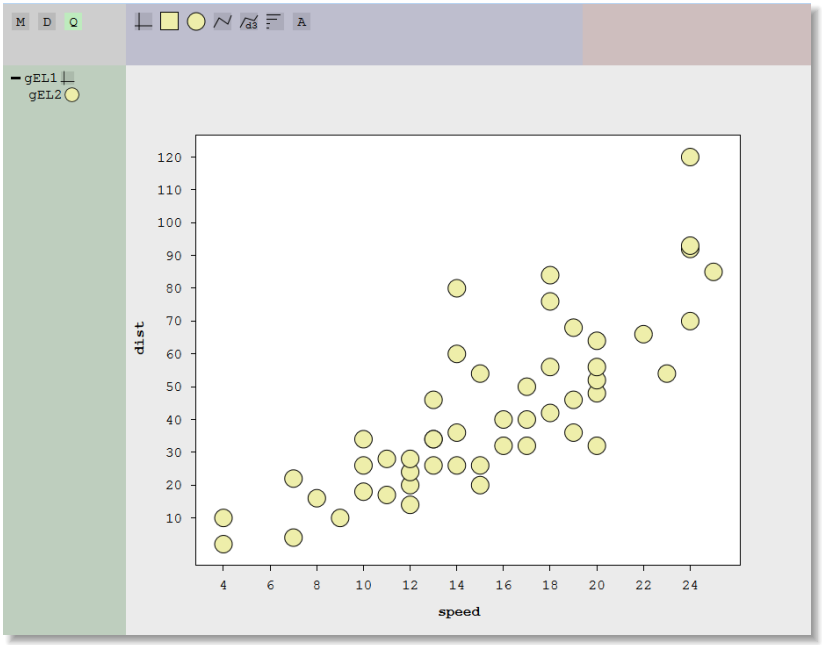


Figure 5.12: A basic scatterplot is complete, the remaining steps add polish to make it look better. The first two steps of polish involve giving the axes more informative names by changing `x-name` and `y-name` of the Cartesian Frame.

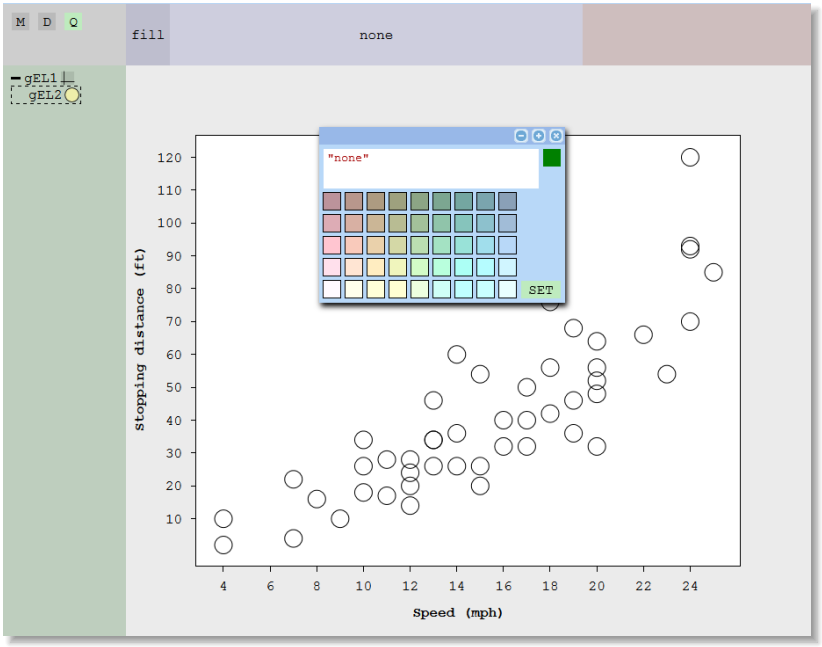


Figure 5.13: There are some overlapping points, so it would be better if the circles were not filled to make it easier to identify the individual points. This is done by changing the `fill` attribute of the Circle to `"none"` (quotation marks are necessary).

Aside: Assigning Colour

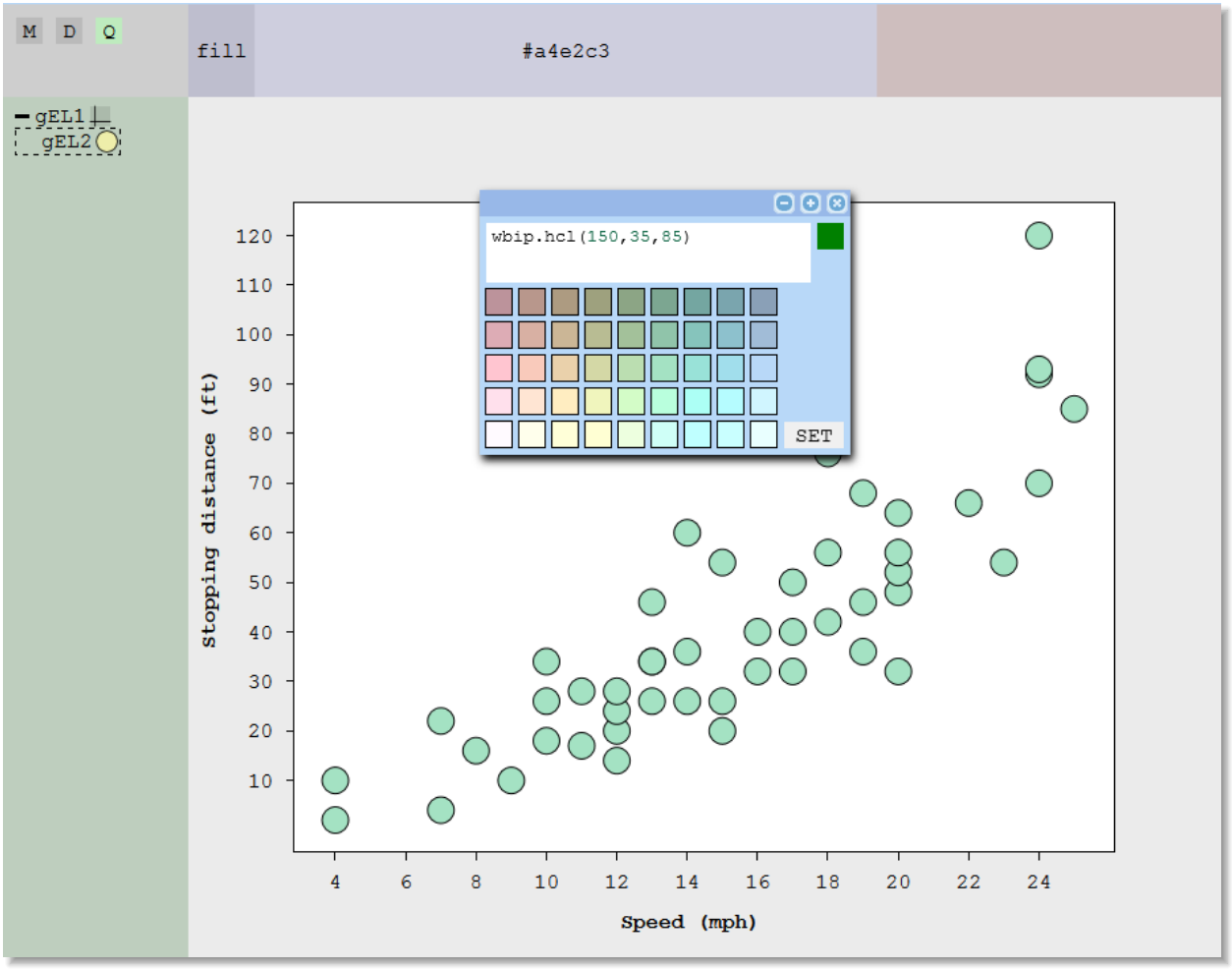


Figure 5.14: WeBIPP's `rgb` interface.

Figure 5.14: One may notice that WeBIPP's `rgb` interface (used, among other attributes, for `fill`) provides an arrangement of colours to choose from. While we did not want any colour in Figure 5.13, we demonstrate what clicking one of these would look like. WeBIPP implements the `hcl` function for R (Ihaka, 2003) in JavaScript, enabling colour assignment in the CIELUV colour space.

Going across from left-to-right on the interface will alter the hue in even steps, while chroma and luminance remain fixed. Going down from top-to-bottom will alter the chroma and luminance in even steps, while hue remains fixed. In addition to providing a nice selection of colours to choose from, this layout is useful for assigning harmonious, perceptually uniform colours and ensuring no colour-induced bias. Additionally, the values for hue, chroma and luminance can be assigned manually by adjusting the text region at the top of the interface. In a future update, it will become possible to use data variables to assign harmonious colour values based on the data.

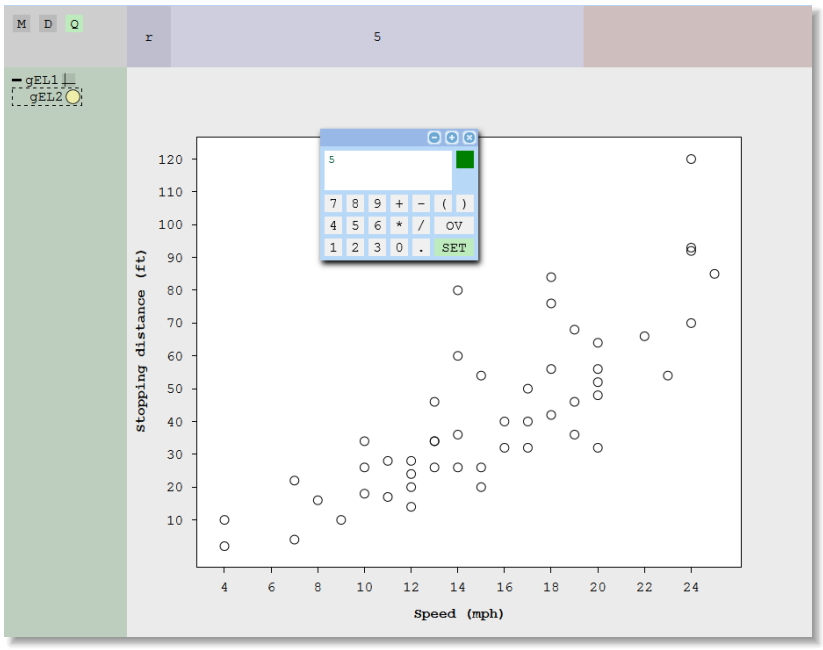


Figure 5.15: Making the circles smaller will reduce overlap and help distinguish each individual point. This can be achieved by assigning a smaller value (in this case 5) to the `r` attribute of the Circle. Unfortunately this won't help with the two overlapping points at `(13, 34)`, as they are exactly on top of each other.

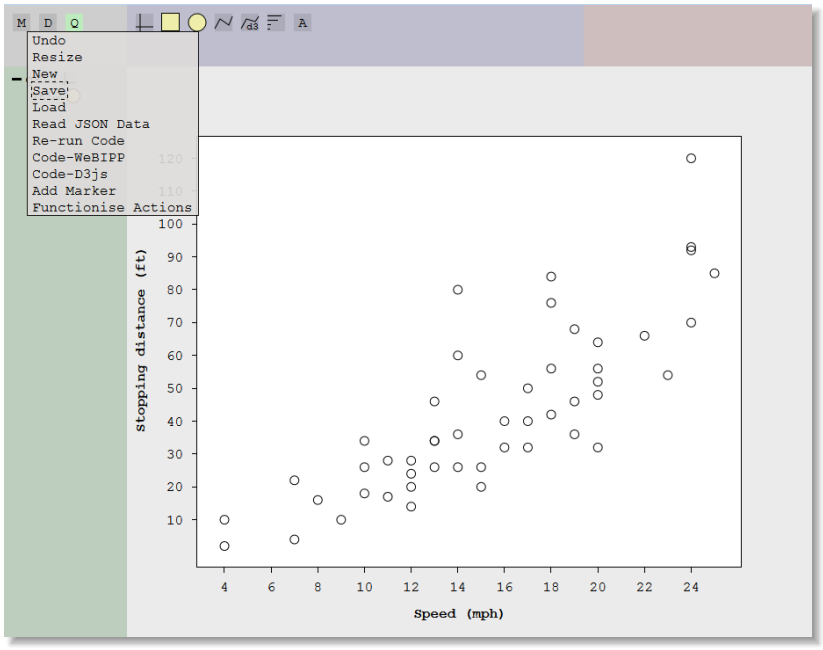


Figure 5.16: Save the final graphic.

5.4.6 The Final Save Data

Figure 5.16: We can save the final graphic by selecting `Save` from the main menu. This will create a new text file containing the save information. The exact handling will depend on the user's current browser options, but most likely will result in a new window or tab being opened containing the save data. The user may then save this data to their hard-drive, e.g. by using the browser's save interface. The save data for the final graphic is copied below (with some adjustments to save space and improve clarity):

```
/*WBIP SAVE HEADER{
  "version": "0.10.4",
  "addonList": {
    // List of Addons removed for space
  },
  "dim": [
    920,
    720
  ],
  "ELIndex": 3
}END*/
wbip.data["datCars"] = JSON.parse(/*Data removed for space*/);
wbip["frcart"].click("gEL1", d3.select("#gGraph"), [43,42]);
wbip["frcart"].setattr("gEL1", "x", "wbip.data[\"datCars\"][\"speed\"]");
wbip["frcart"].setattr("gEL1", "y", "wbip.data[\"datCars\"][\"dist\"]");
wbip["circle"].click("gEL2", d3.select("#gEL1"), [91,206]);
wbip["circle"].setattr("gEL2", "cx", "wbip.data[\"datCars\"][\"speed\"]");
wbip["circle"].setattr("gEL2", "cy", "wbip.data[\"datCars\"][\"dist\"]");
wbip["frcart"].setattr("gEL1", "x-name", "Speed (mph)");
wbip["frcart"].setattr("gEL1", "y-name", "Stopping distance (ft)");
wbip["circle"].setattr("gEL2", "fill", "none");
wbip["circle"].setattr("gEL2", "r", "5");
```

The text that is the save data is a complete record of all the steps taken in creating the graphic, and this record can be used to reproduce the exact same graphic from scratch (e.g. by loading this save data). The save data additionally contains some extra information as a header. The save data is not only a record of all the steps taken, it is also human-legible code, and like any other piece of code, it can be altered manually if desired. This has many advantages as discussed in Subsection 5.2.3.

The scatterplot is one of the easiest graphs to create in WeBIPP, and the graph produced in Section 5.4 can be recreated in minutes by a user familiar with WeBIPP. Next we will cover a more complex graphic that will require a bit more work, a Population Pyramid.

5.5 Creating a Population Pyramid

5.5.1 The Model

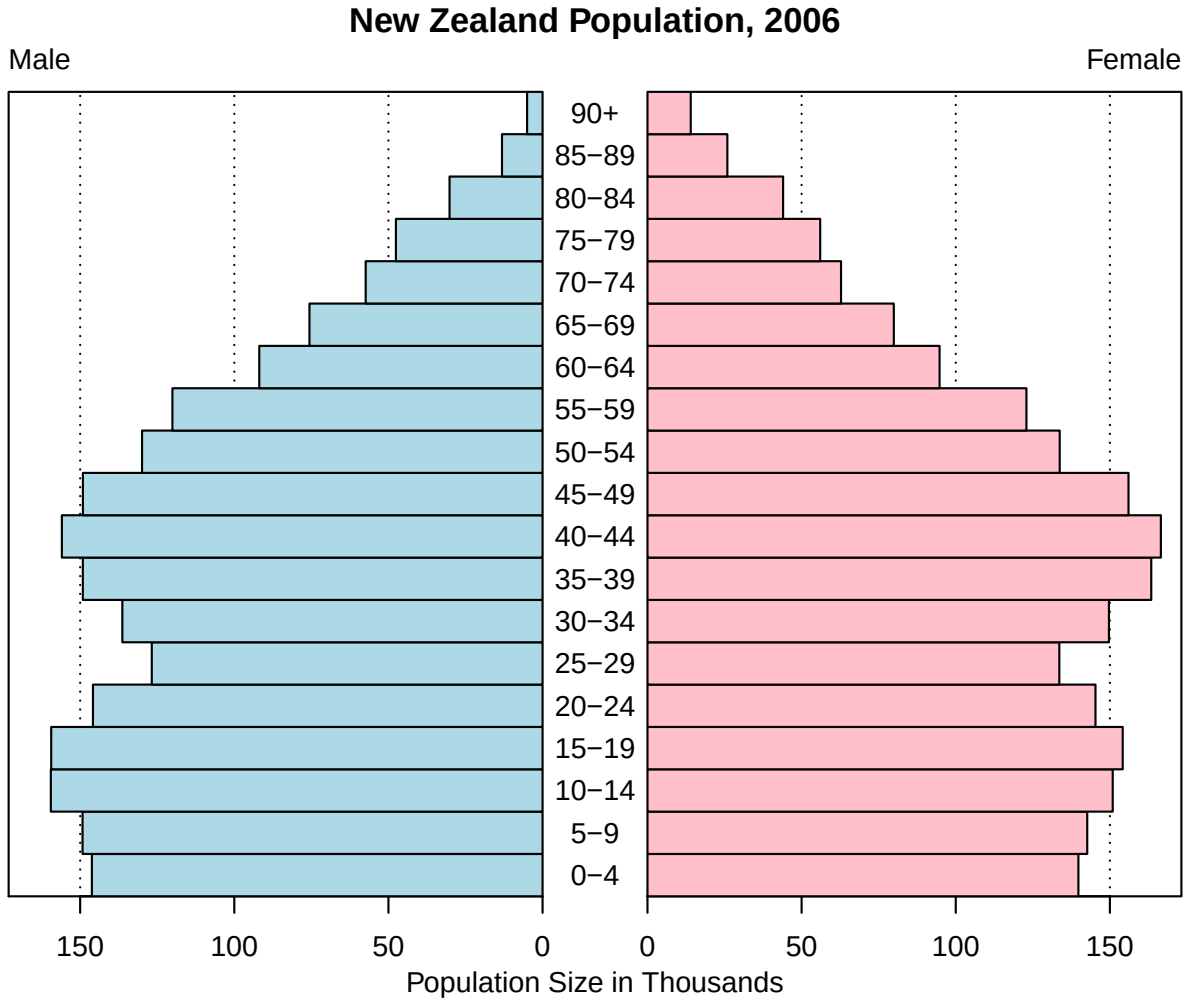


Figure 5.17: This Population Pyramid was drawn using R. The data is of the New Zealand Population data (Statistics New Zealand, 2011). Using WeBIPP, we wish to recreate a graphic as close to this as possible. The steps that follow are more for demonstration purposes, and less to be precise and clear steps to recreate the graph. See the code at the end, or the tutorial on the website, for concise instructions.

5.5.2 Set up the Axes

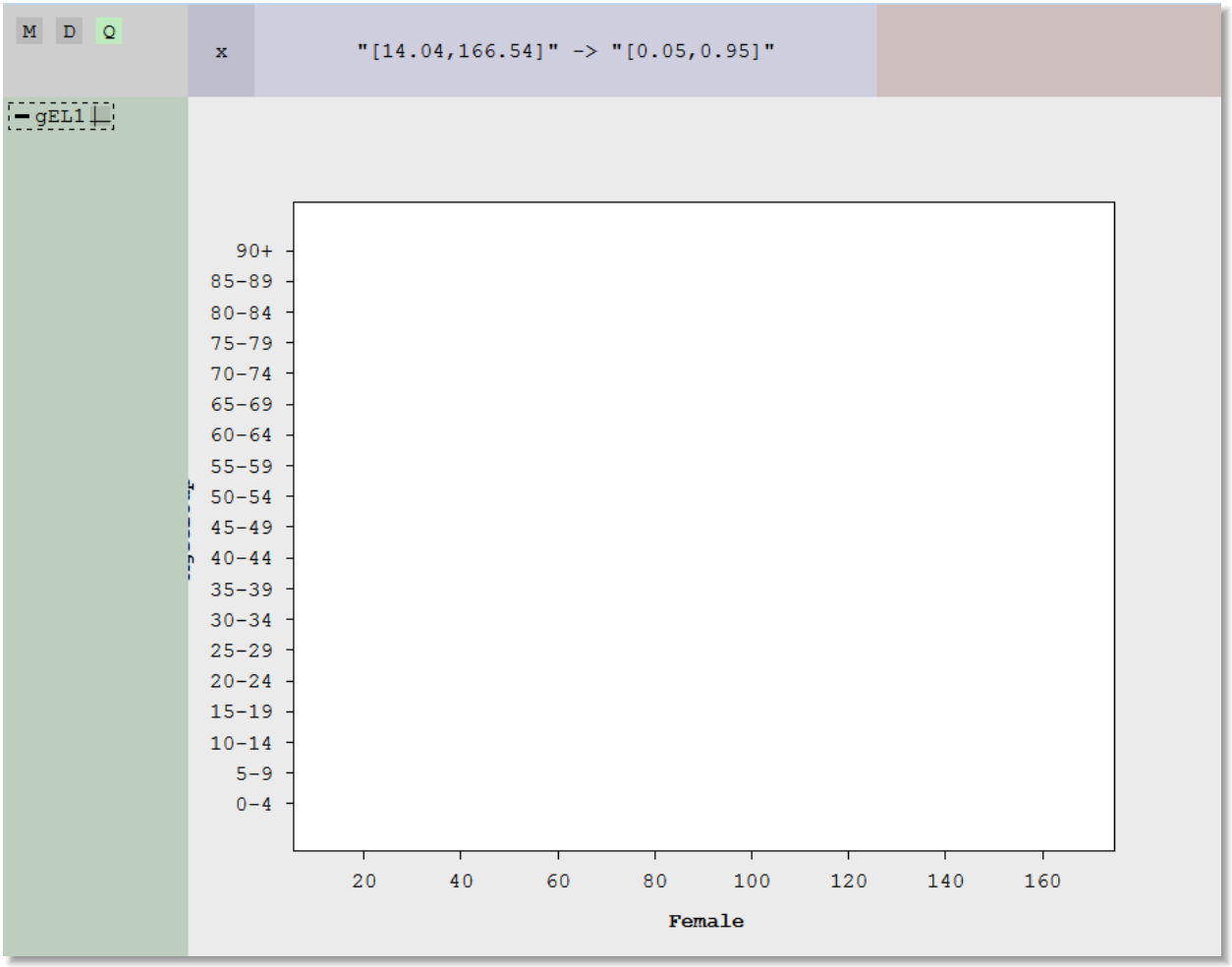


Figure 5.18: Setting up the Cartesian Frame.

Figure 5.18: The first few steps are as before, load the data[9], drop a Cartesian Frame and assign the data variables `Female` to `x` and `AgeGroup` to `y`. Unlike before, the default handling of the axes is not suitable for what we want and must be adjusted. Notice in the Figure that our x-axis is mapping the domain `[14.04, 166.54]` (the minimum and maximum values of the `Female` data) to the range `[0.05, 0.95]` (representing 5% and 95% of the width of the Cartesian Frame). For our Population Pyramid, the x-axis must start exactly at 0, and end at some value above the maximum. We also want a shared x-label (Population Size in Thousands) and not individual labels for each side of the pyramid. To do this, we set `x-name` to `""` and `x-domain` to `[0, 170]`. These changes can be seen in Figure 5.19. We will also do the left-side of the pyramid first, hence we change `x-range` to `[1, 0.01]`. This will map the respective domain values to 100% and 1% of the width of the Cartesian Frame (Figure 5.20).

[9]The data for this example can be downloaded here:
https://www.stat.auckland.ac.nz/~joh024/Research/WeBIPP/Examples/datnzpop_2006.json

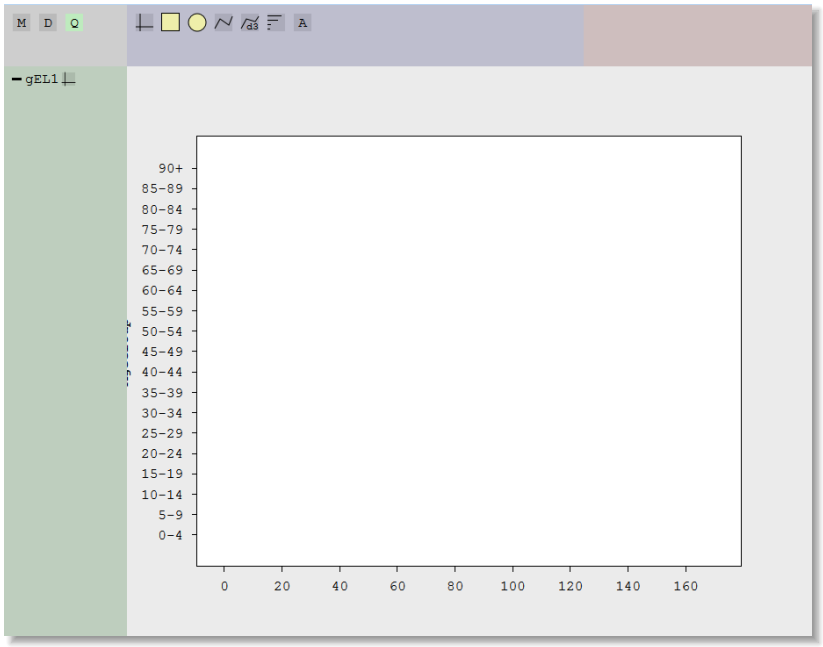


Figure 5.19: Adjust the x-axis domain.

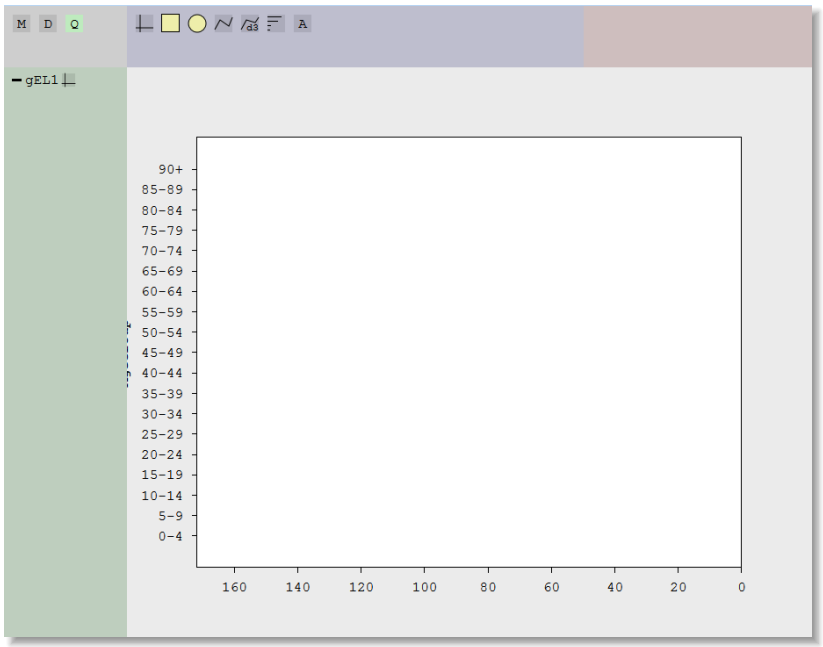


Figure 5.20: Flip the x-axis to the correct orientation for the left-side of the pyramid.

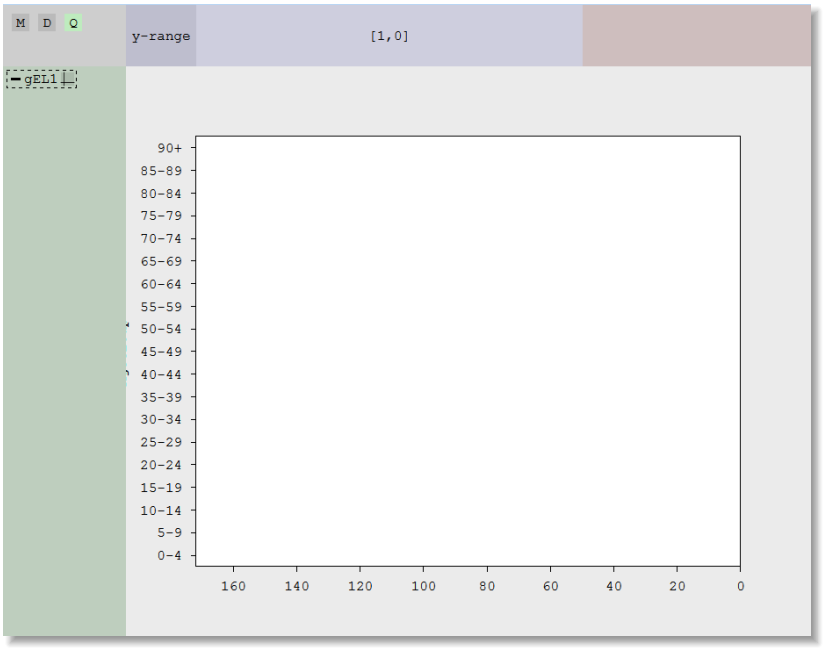


Figure 5.21: For similar reasons, adjust the y axis range to make full use of the height.

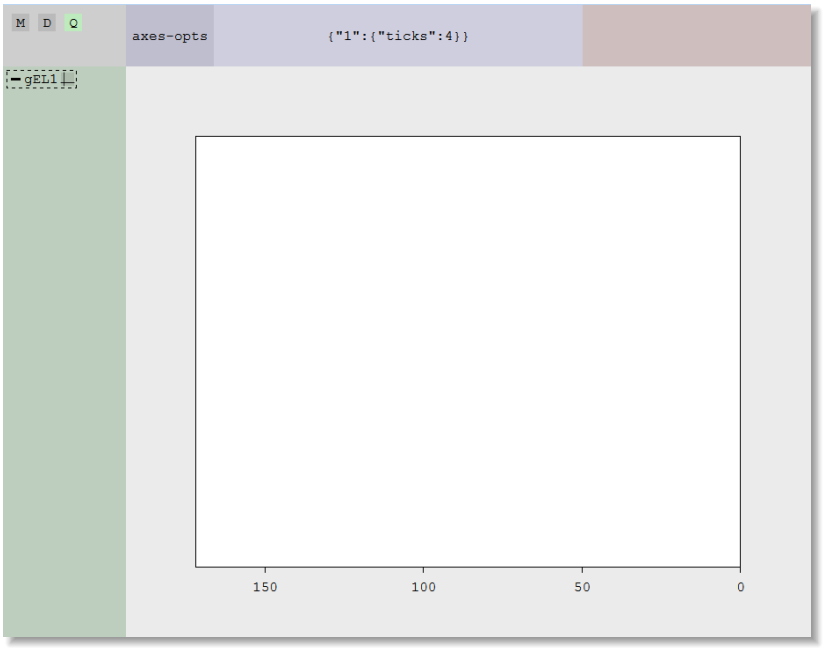


Figure 5.22: We only want the x axis labels, so we set `axes` to `[1]` (WeBIPP uses the same system as R for referencing axes). We also want exactly 4 tick marks on the x axis, so we set `axes-opts` to `{"1":{"ticks":4}}`. Unfortunately neither of these attributes have nice interfaces for assigning these values, so for now, the user must simply know what to assign directly using a default interface, if they wish to tweak these settings.

5.5.3 Resize the Frame

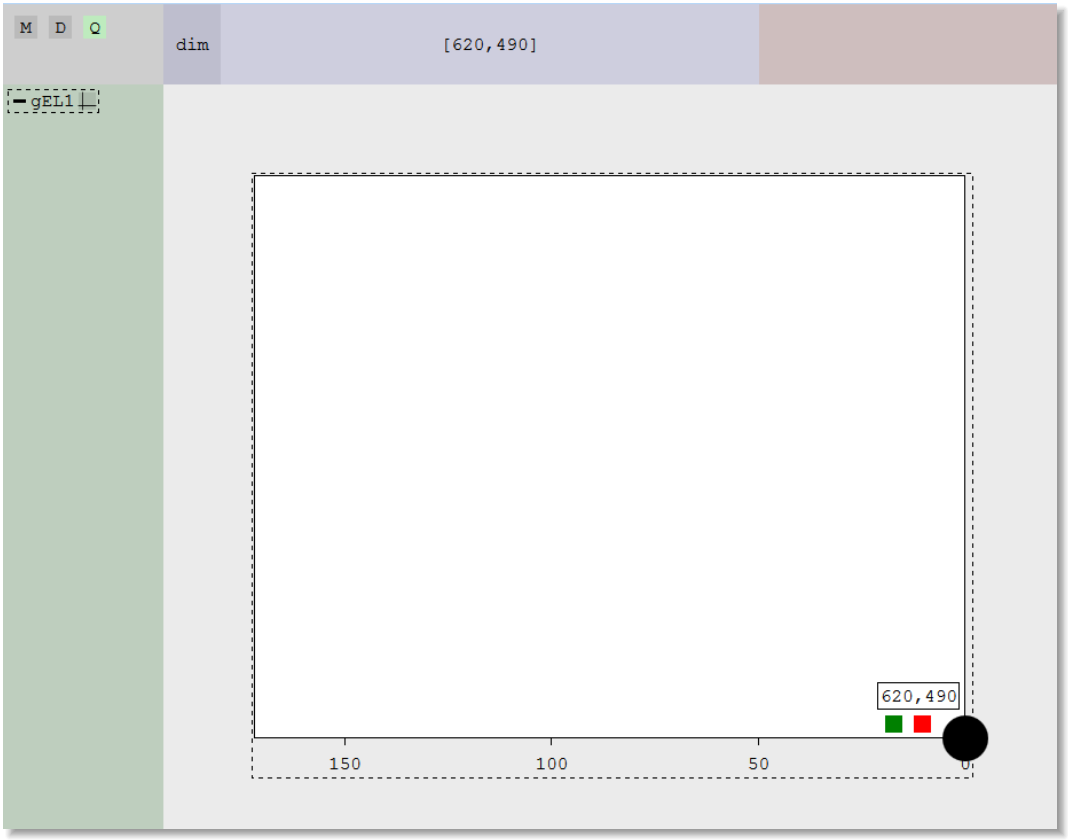


Figure 5.23: The resize interface for WeBIPP.

Figure 5.23: With the Cartesian Frame axes set up to our satisfaction, we must resize and position the frame to something appropriate for the left-side of the pyramid. WeBIPP currently lacks a nice way to lay out juxtaposed graphics so the resizing and positioning must be done by modifying the `dim` and `transform` attributes. Resizing the dimensions can be accomplished by eye using the interface shown, but if we wish to be more exact we can go to the code and compute the dimensions.

In Figure 5.24, we have set a roughly correct dimension using the interface, and the Code-WeBIPP interface (accessed via the main menu) is open showing the code. We will adjust this specific number to one that is computed instead:

```
1 [(wbip.getdim("gGraph")[0] - 100)/2,
2  wbip.getdim("gGraph")[1] - 80 * 2]
```

The width is computed by taking the width of the Graph Region (`wbip.getdim("gGraph")[0]`), leaving 20 pixels for margins on either side, and 60 pixels for the labels in the middle, then diving by two to get the size for half of the pyramid. The height is computed in a similar manner, leaving 80 pixels on both sides for the margins and labels.

The graph after these adjustments is seen in Figure 5.25. It looks remarkably similar, and in fact the width only differs by 14 pixels and the height is exactly the same. Using the interface can quickly get a roughly right result, while computation can be used for perfection and scaling (see Figure 5.67 and Figure 5.69).

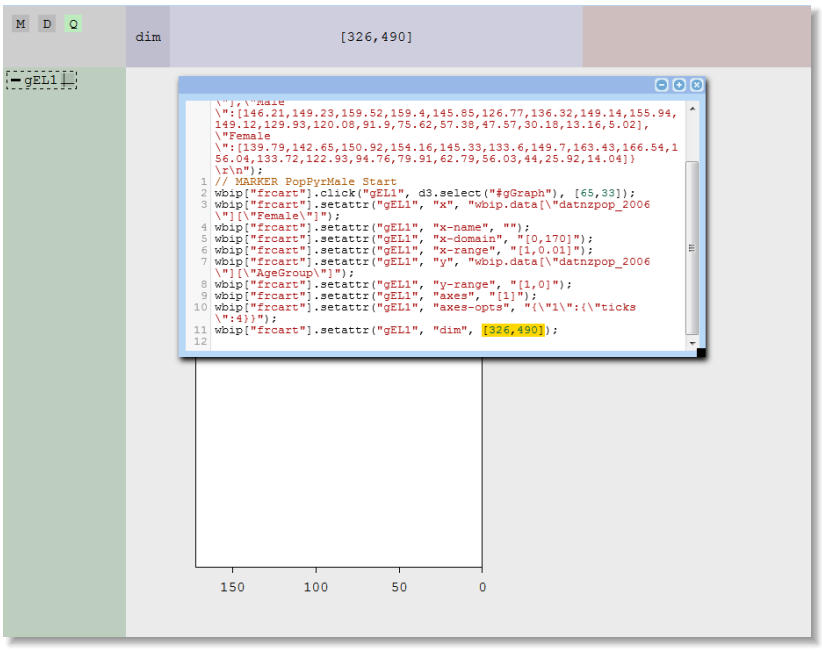


Figure 5.24: The dimensions have been adjusted to roughly the correct value, using the interface. The WeBIPP Code interface is open showing the code.

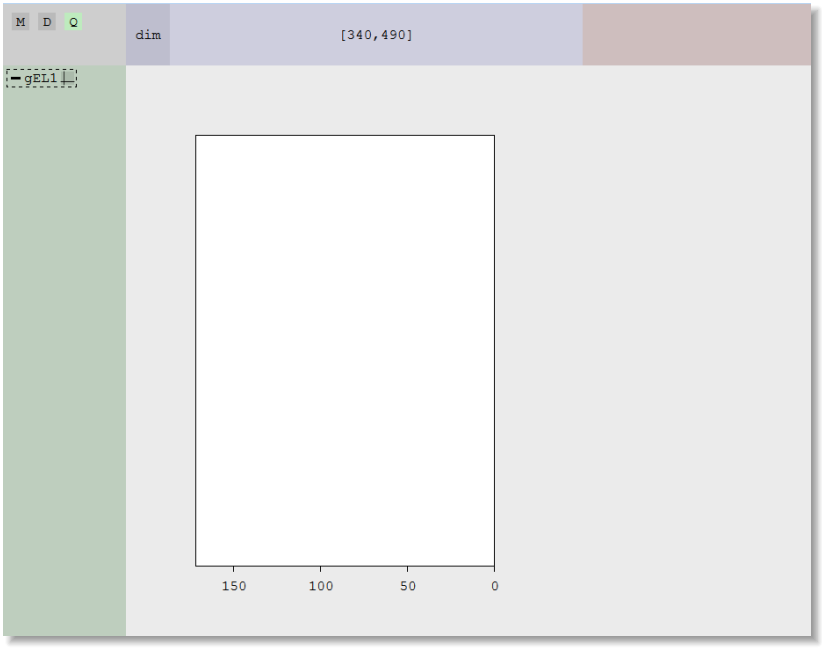


Figure 5.25: The dimensions have been computed exactly via manual adjustment of the code.

5.5.4 Reposition the Frame

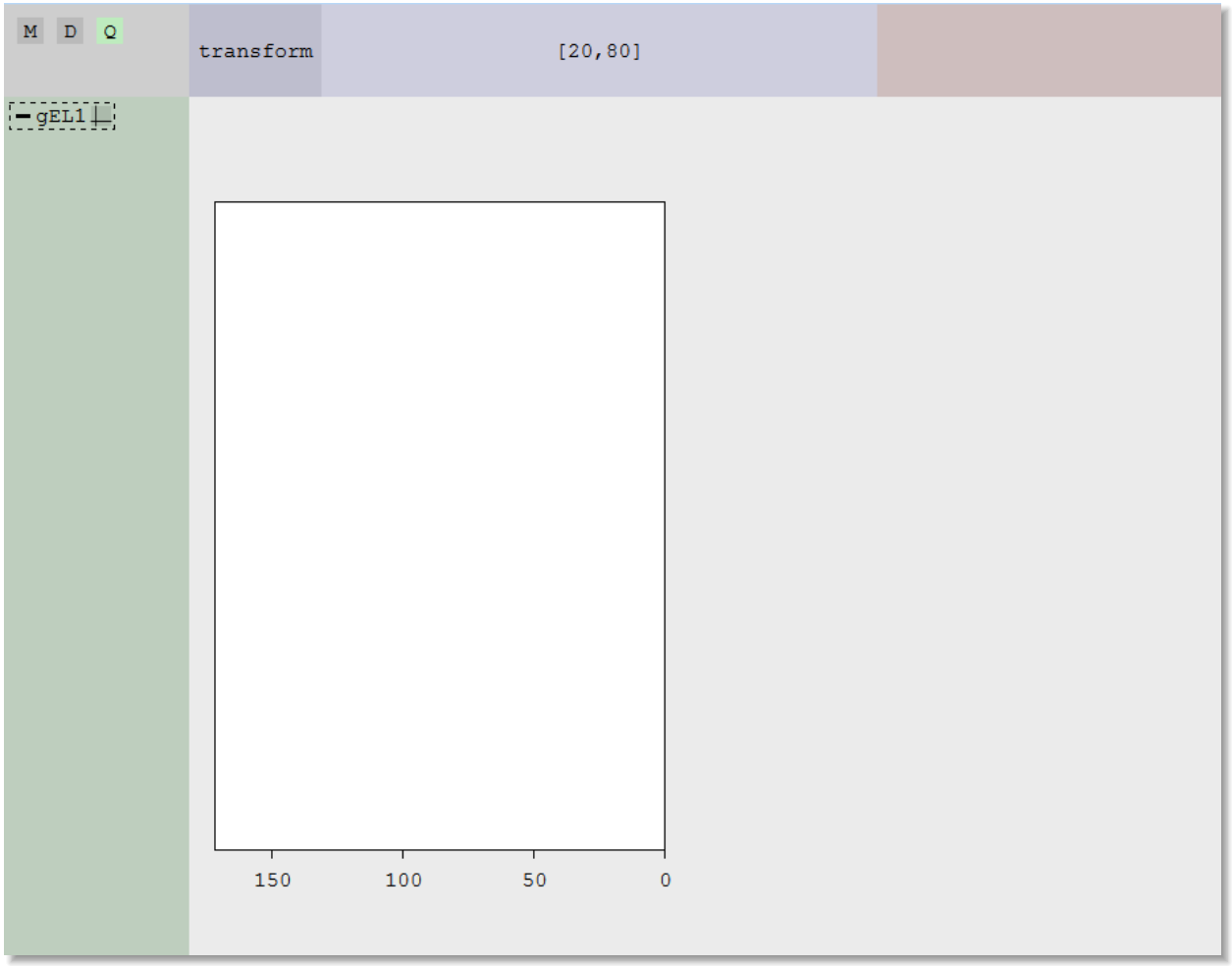


Figure 5.26: Shift the Cartesian Frame into position by applying a translation of `[20, 80]`.

Figure 5.26: Per the calculation in Figure 5.23, we want exactly 20 pixels for the margin on the left, and 80 pixels for the margin on the top. This can be accomplished by applying an SVG transform. Currently however, no nice transform interface exists, and despite being named `transform`, the attribute only handles `translate`. This is sufficient for our purposes, but eventually a proper transform interface will be added, and this attribute will properly support all the transforms.

5.5.5 Draw the Dotted Lines

Figure 5.27: The line segment button.

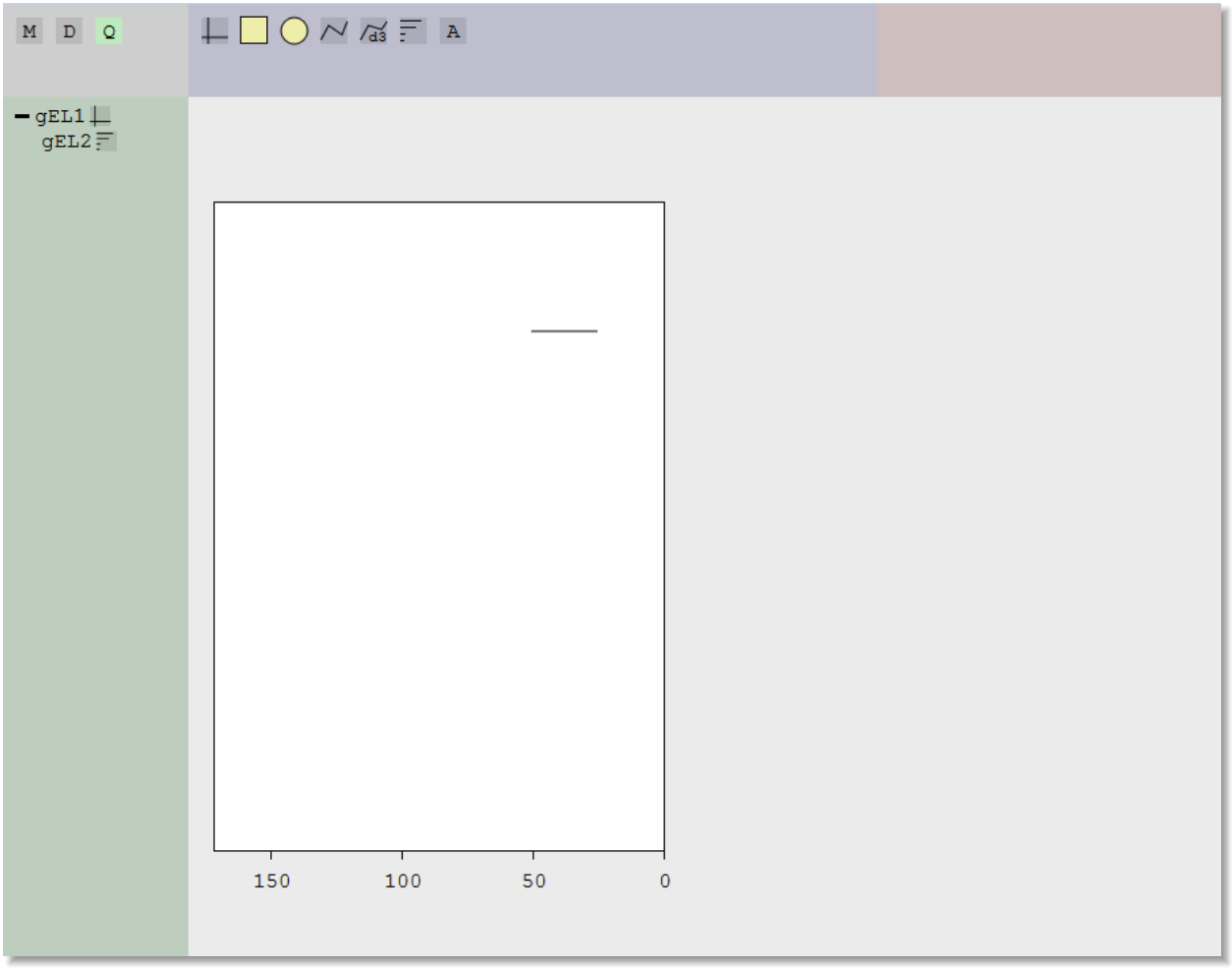


Figure 5.28: A line segment using the default values.

Figure 5.28: To draw the dotted lines in the background, we use the Line Segment Object. Like the `segments` function in R, these take four attributes: `x1`, `x2`, `y1` and `y2`. A line segment will be drawn from each (x1, y1) pair to each (x2, y2) pair. Thus to draw our dotted lines we need to set `x1` to `[50, 100, 150]` (Figure 5.29), and `x2` to the same values (resulting in vertical lines). Then we set `y1` to `0` (Figure 5.30) so the lines start from the top, and set `y2` to `490` (Figure 5.31) so the lines go to the bottom. Better than setting `490` to `y2` is to compute the value. This can be achieved by setting it to `expr:curDim[1]`, though we will also need to specify that `y2-useScale` must be `false`. Finally for the dotted lines, we can set `stroke-dasharray` to `1 4` (1 pixel drawn, 4 pixels blank) (Figure 5.32).

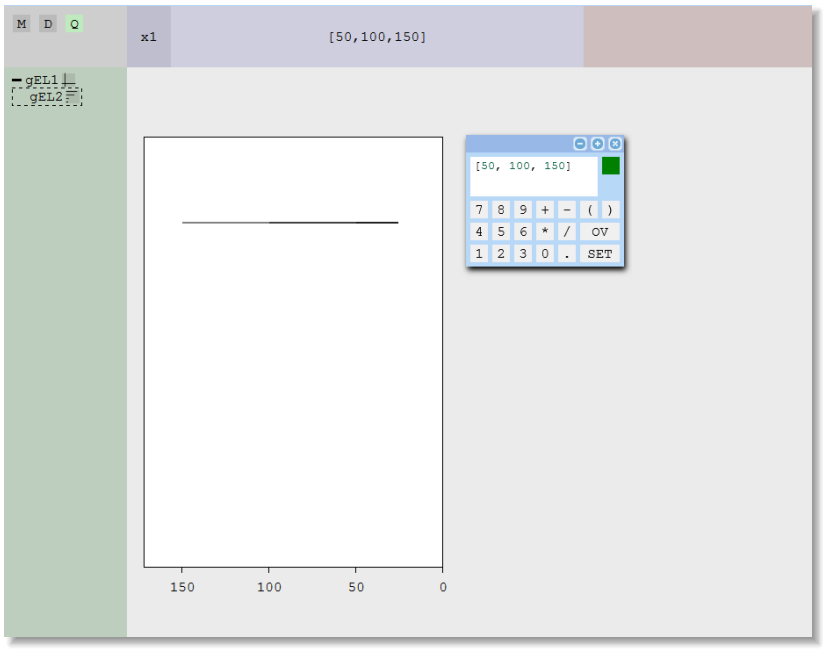


Figure 5.29: Set the x values for the line segment.

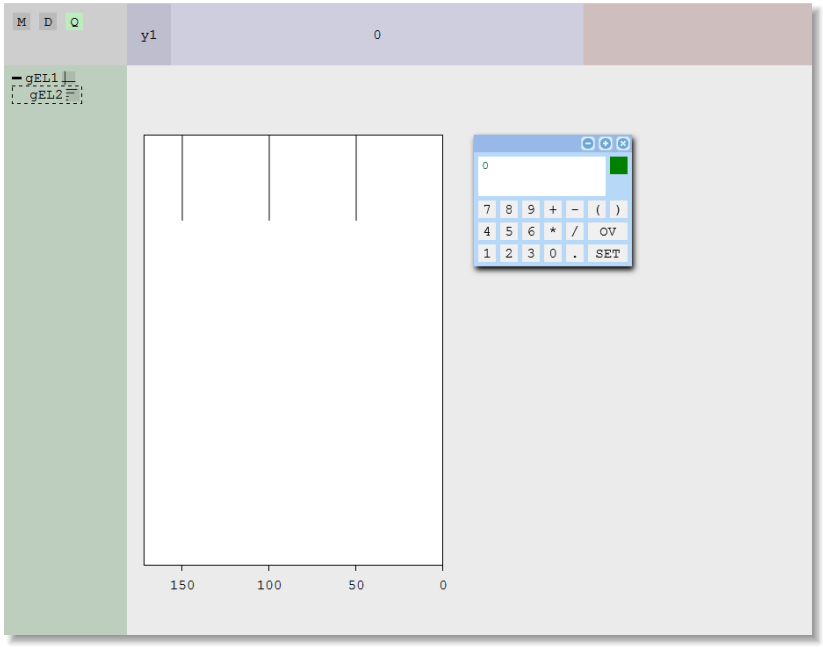


Figure 5.30: Have the line segment start from the top.

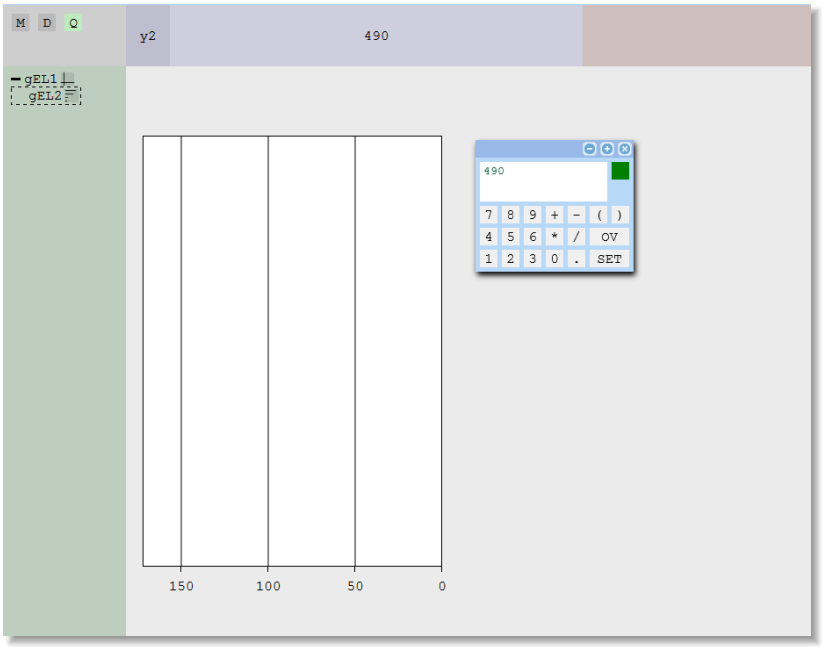


Figure 5.31: Have the line segment end at the bottom.

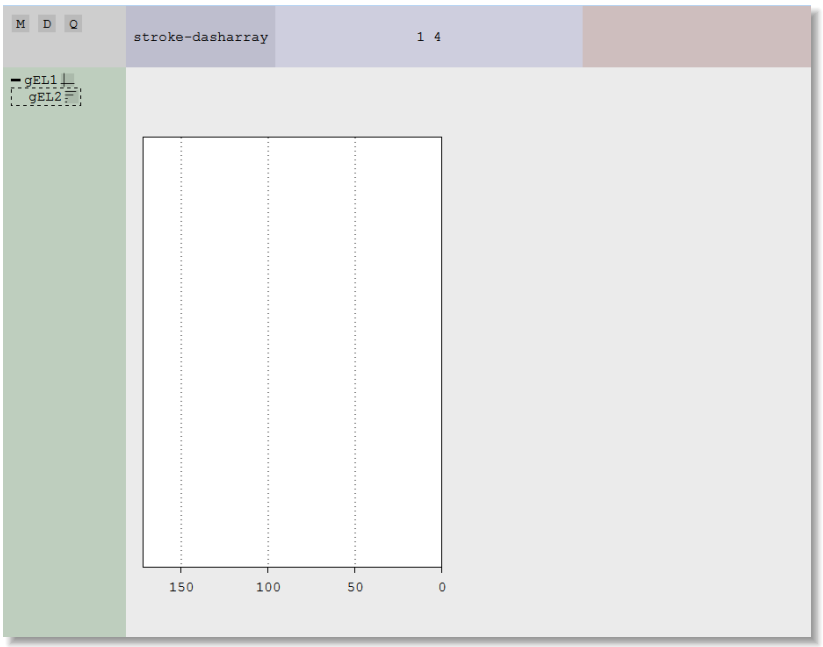


Figure 5.32: Make the line segment dashed.

5.5.6 Draw the Bars

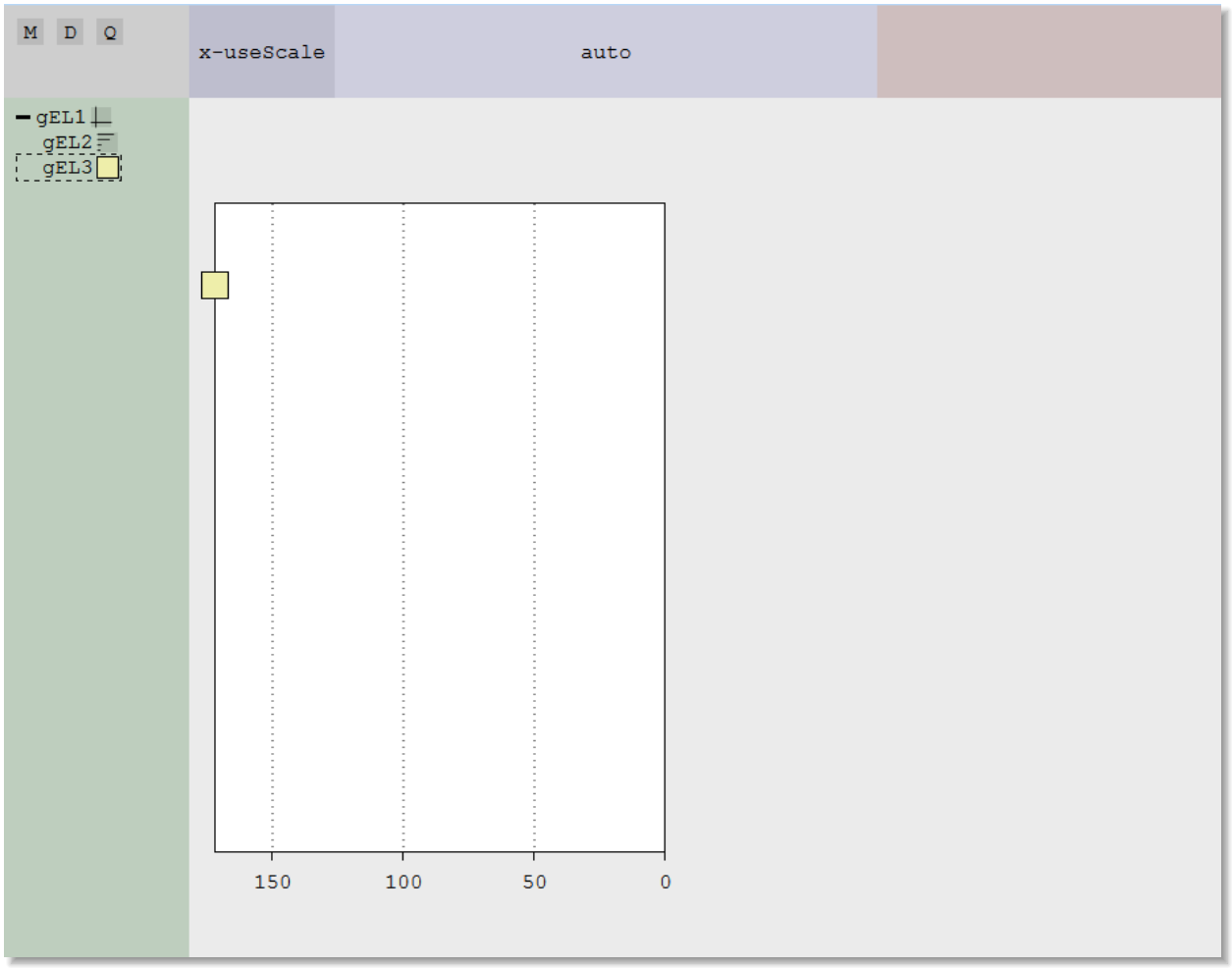


Figure 5.33: Rectangle with x centred at 0 pixels.

Figure 5.33: To draw the histogram rectangles, the rectangles must start from 0, and go across to the data values. The first part can be accomplished by making the following changes:

x = `0`

x-useScale = `true` - In the current figure, `x` has been set, but we can see that the scale is not taking effect as the rectangle is centred on pixel 0, not 0 according to the scale. So we force it to use the scale by setting this to `true` in Figure 5.34.

x-adj = `1` - This works like in R. A value of `0` means the x value corresponds to the left-edge of the rectangle. A value of `0.5` (the default) means the x value represents the centre. A value of `1` means the x defines the right-edge of the rectangle, which is what we want as we see in Figure 5.35.

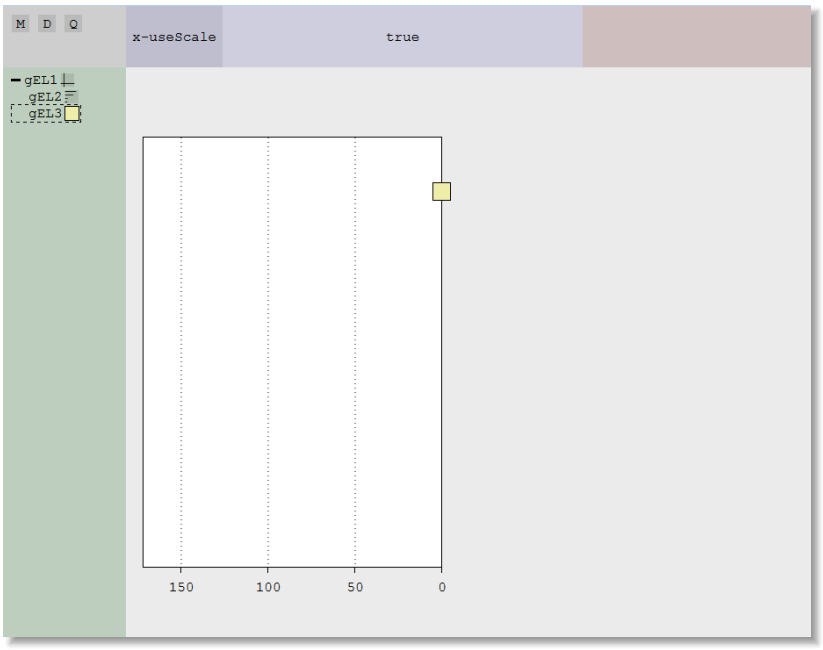


Figure 5.34: Rectangle with x centred at 0 according to the x axis scale.

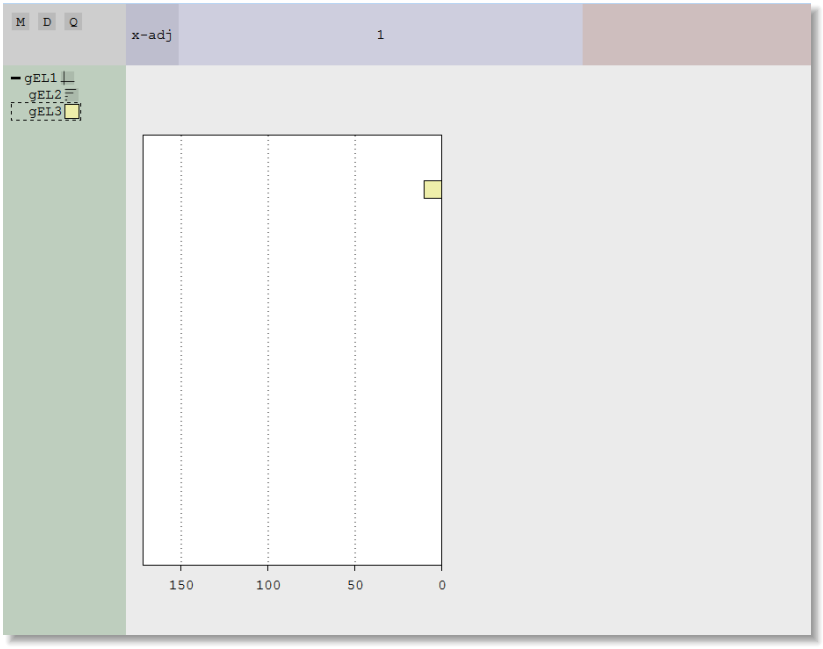


Figure 5.35: Rectangle with its right-edge at 0 according to the x axis scale.

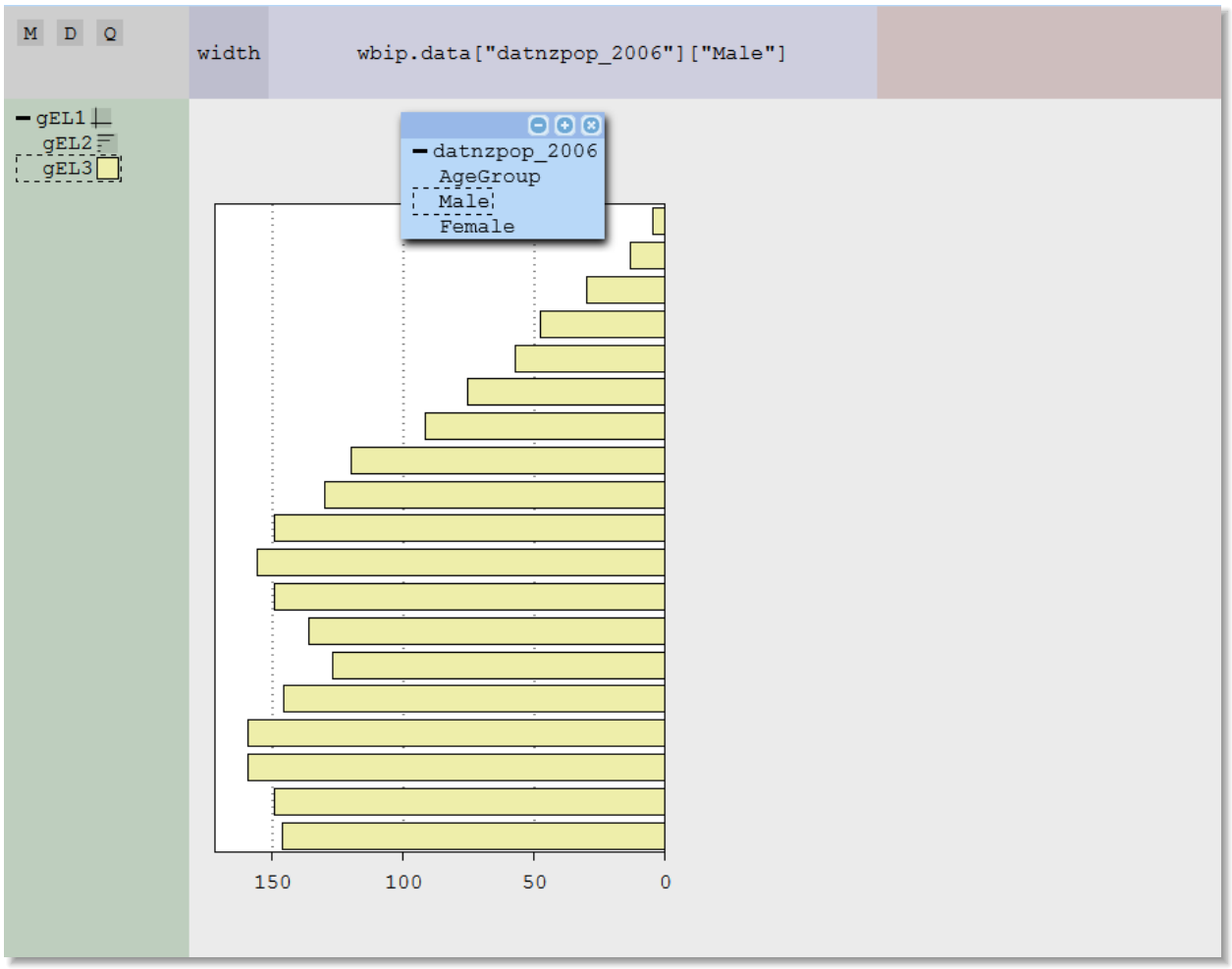


Figure 5.36: Assign `Male` to the widths of the bars.

Figure 5.36: With all the rectangles starting from 0, we can now assign the data variable `Male` to the `width` of the rectangle, resulting in something that looks like a bargraph. As the Population Pyramid should be histograms, we do not want any gaps between the bars. As y is categorical, we can achieve the correct `height` by assigning `1` (Figure 5.37) and setting `height-useScale` to `true` (Figure 5.38).

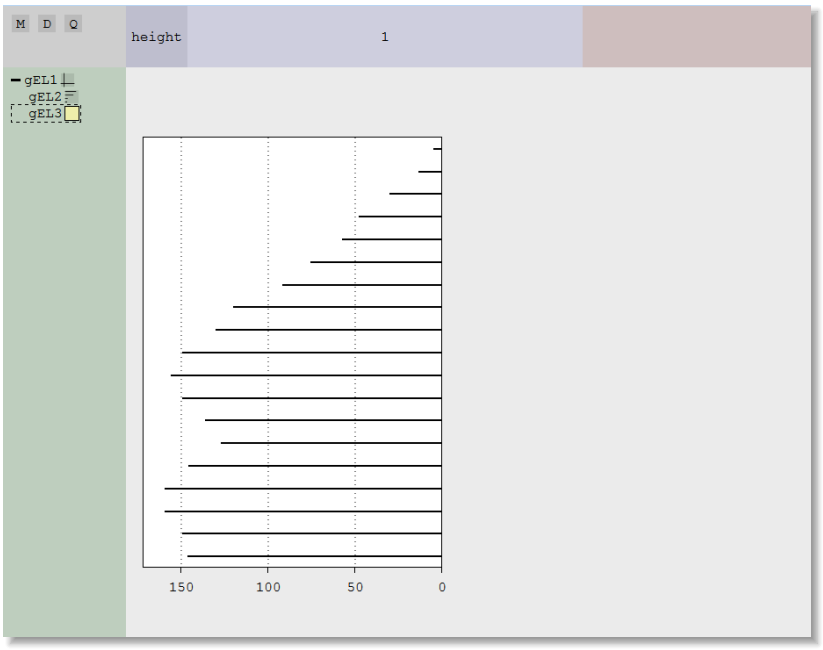


Figure 5.37: Set the rectangle `height` to 1.

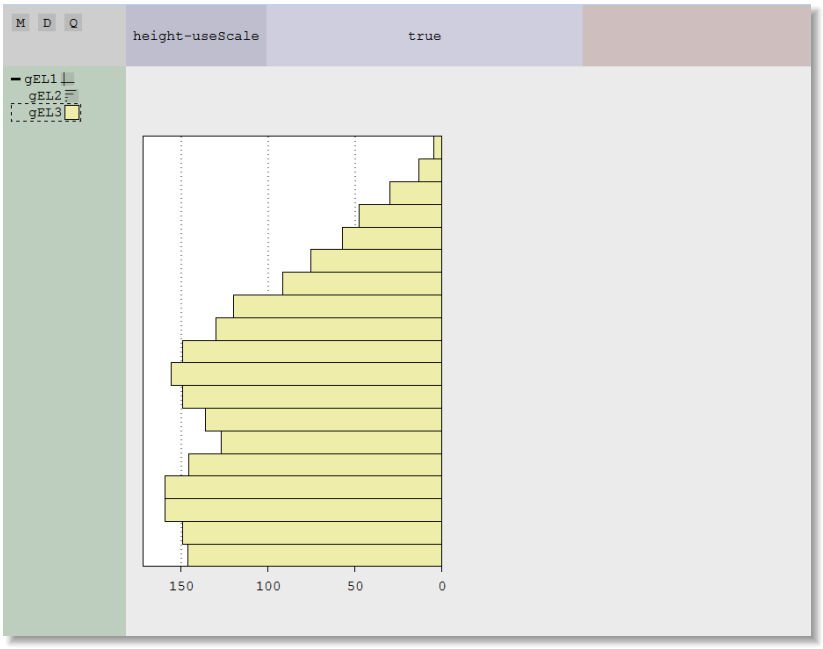


Figure 5.38: With `height-useScale` set to `true`, there are now no gaps between the bars.

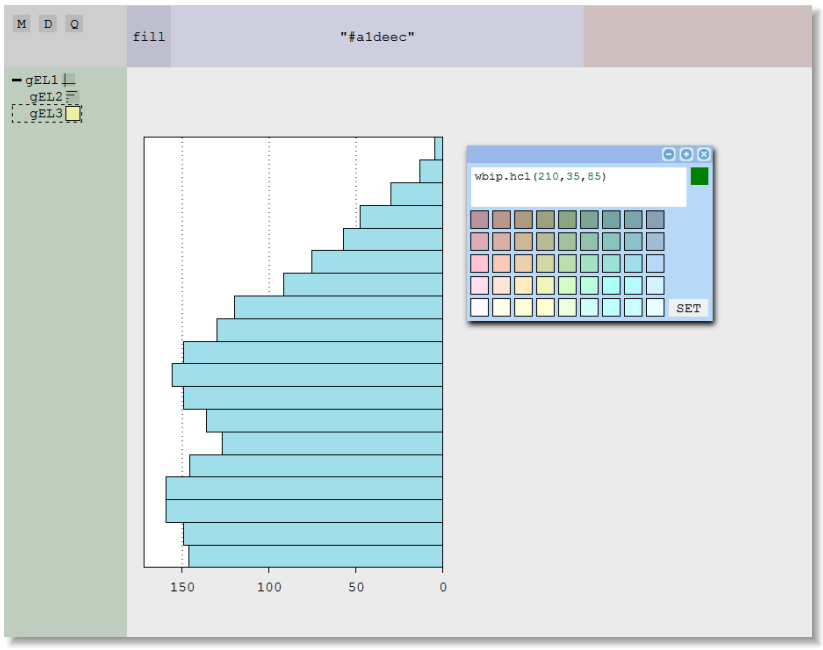


Figure 5.39: As this is Male data, assign an appropriate blue colour to the bars using the interface.

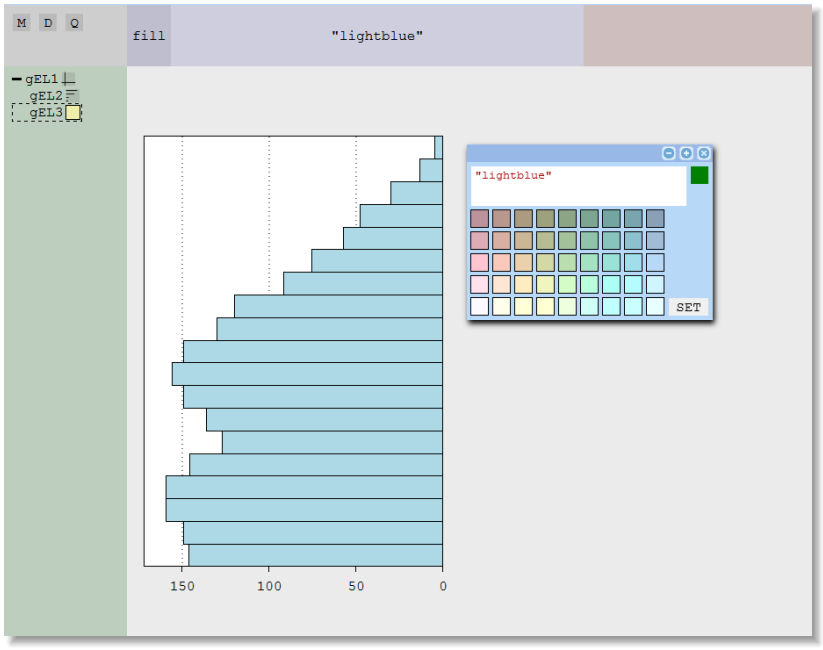


Figure 5.40: We could also assign a specific colour by name, in this case `"lightblue"` (quotation marks necessary). This more closely matches the graph we wish to replicate.

5.5.7 Draw the Gender Label

Figure 5.41: The text button.

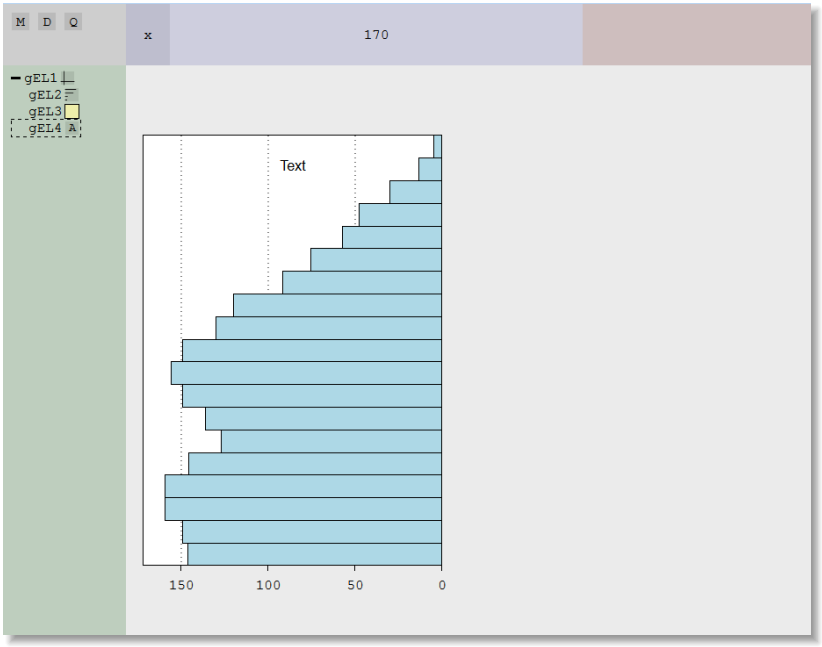


Figure 5.42: A text object is used to draw a label.

Figure 5.42: The last touch needed to finish this side of the pyramid is a label indicating which data this side represents. This is done by placing a Text Object anywhere on the frame, and then adjusting its position:

x = `170` - Remembering that the x domain is `[0, 170]`.

x-useScale = `true` - This will position the text on the far left-edge (Figure 5.43).

y = `-20` - In pixels (not using the scale), to position the label above the graph (Figure 5.44).

Finally, by setting `text-anchor` to `start` we align the text correctly on its x-position (Figure 5.45), and we increase the font-size to something that is appropriate for a label (Figure 5.46).

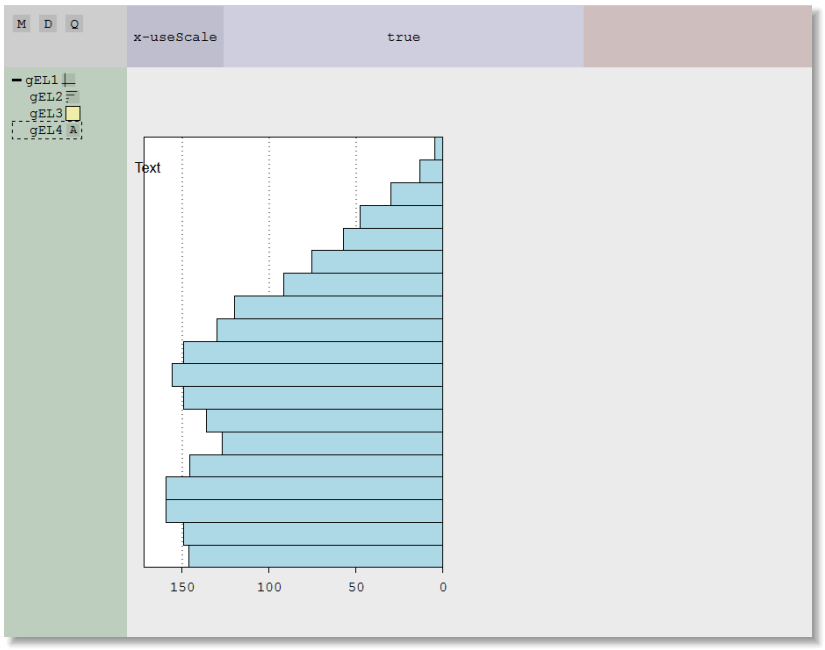


Figure 5.43: Position the text on the left-edge of the graph.

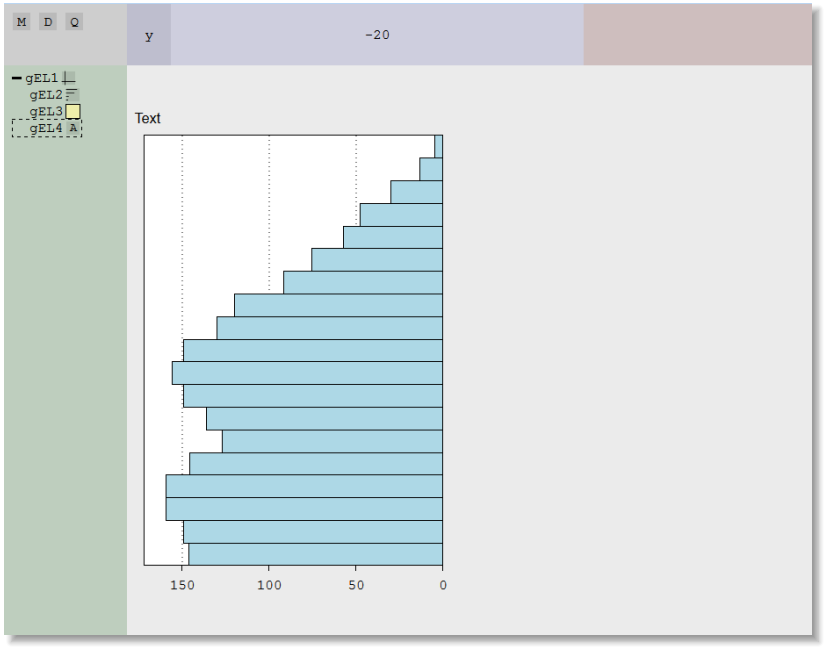


Figure 5.44: Position the text above the graph.

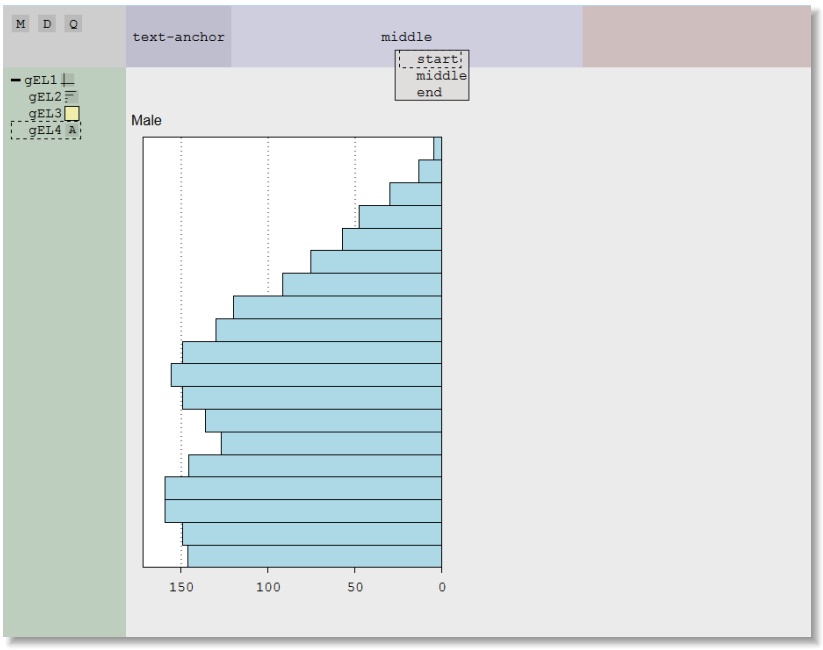


Figure 5.45: Anchor the text correctly for correct alignment.

Figure 5.46: Adjust the size to something appropriate for a label.

5.5.8 Re-using the Left-Side

We have now completed one side of the pyramid. To create the other side, we can repeat the steps with minor adjustments. In other interface-based tools, this would be the only option, but as WeBIPP writes code in the background, we have code-based approaches available too. Instead of repeating the steps, we can take the code generated and tweak the arguments slightly. The simplest way to accomplish this is to copy the relevant code, paste it, and then manually adjust as needed. The code that generates the left-side is copied below:

```
wbip["frcart"].click("gEL1", d3.select("#gGraph"), [65,33]);
wbip["frcart"].setattr("gEL1", "x", "wbip.data[\"datnzpop_2006\"][\"Female\"]");
wbip["frcart"].setattr("gEL1", "x-name", "");
wbip["frcart"].setattr("gEL1", "x-domain", "[0,170]");
wbip["frcart"].setattr("gEL1", "x-range", "[1,0.01]");
wbip["frcart"].setattr("gEL1", "y", "wbip.data[\"datnzpop_2006\"][\"AgeGroup\"]");
wbip["frcart"].setattr("gEL1", "y-range", "[1,0]");
wbip["frcart"].setattr("gEL1", "axes", "[1]");
wbip["frcart"].setattr("gEL1", "axes-opts", "{\"1\":{\"ticks\":4}}");
wbip["frcart"].setattr("gEL1", "dim", [(wbip.getdim("gGraph")[0] - 100)/2, wbip.getdim("gGraph")[1] - 80 * 2]);
wbip["frcart"].setattr("gEL1", "transform", "[20,80]");
wbip["line-segment"].click("gEL2", d3.select("#gEL1"), [239,97]);
wbip["line-segment"].setattr("gEL2", "x1", "[50, 100, 150]");
wbip["line-segment"].setattr("gEL2", "x2", "[50, 100, 150]");
wbip["line-segment"].setattr("gEL2", "y1", "0");
wbip["line-segment"].setattr("gEL2", "y2-useScale", "false");
wbip["line-segment"].setattr("gEL2", "y2", "expr:curDim[1]");
wbip["line-segment"].setattr("gEL2", "stroke-dasharray", "1 4");
wbip["rect"].click("gEL3", d3.select("#gEL1"), [118,62]);
wbip["rect"].setattr("gEL3", "x", "0");
wbip["rect"].setattr("gEL3", "x-useScale", "true");
wbip["rect"].setattr("gEL3", "x-adj", "1");
wbip["rect"].setattr("gEL3", "y", "wbip.data[\"datnzpop_2006\"][\"AgeGroup\"]");
wbip["rect"].setattr("gEL3", "width", "wbip.data[\"datnzpop_2006\"][\"Male\"]");
wbip["rect"].setattr("gEL3", "height", "1");
wbip["rect"].setattr("gEL3", "height-useScale", "true");
wbip["rect"].setattr("gEL3", "fill", "lightblue");
wbip["text"].click("gEL4", d3.select("#gEL1"), [181,34]);
wbip["text"].setattr("gEL4", "x", "170");
wbip["text"].setattr("gEL4", "x-useScale", "true");
```

```
31 wbip["text"].setattr("gEL4", "y", "-20");
32 wbip["text"].setattr("gEL4", "text", "Male");
33 wbip["text"].setattr("gEL4", "text-anchor", "start");
34 wbip["text"].setattr("gEL4", "font-size", "20");
```

For reuse of the code, we need to:

- Flip the x-axis around to left-to-right

```
5 wbip["frcart"].setattr("gEL1", "x-range", "[0,0.99]");
```

- Have the bars start on the left and go to the right

```
22 wbip["rect"].setattr("gEL3", "x-adj", "0");
```

- Shift the frame to its new position

```
11 wbip["frcart"].setattr("gEL1", "transform", <calculation
   ommitted as it will be explained later>);
```

- Adjust the data

```
24 wbip["rect"].setattr("gEL3", "width", "wbip.data[\"
   datnzpop_2006\"][\"Female\"]");
```

- Adjust the colour

```
27 wbip["rect"].setattr("gEL3", "fill", "pink");
```

- Adjust the label

```
32 wbip["text"].setattr("gEL4", "text", "Female");
33 wbip["text"].setattr("gEL4", "text-anchor", "end");
```

We additionally need to adjust the element names correctly (e.g. `gEL1` would need to be renamed to, say, `gEL1b`).

While it is certainly possible to re-use the code with this copy+paste+tweak approach, it would be nice to have a formalised way of doing so that is both more convenient and more robust. It is convenient then, that WeBIPP has a feature called Functionise, which is a formalised way of re-using code. The current Functionise interface is accessed through the main menu. Though primitive, it is in working condition, and will be demonstrated in the next few figures.

5.5.9 Functionise the Left-Side

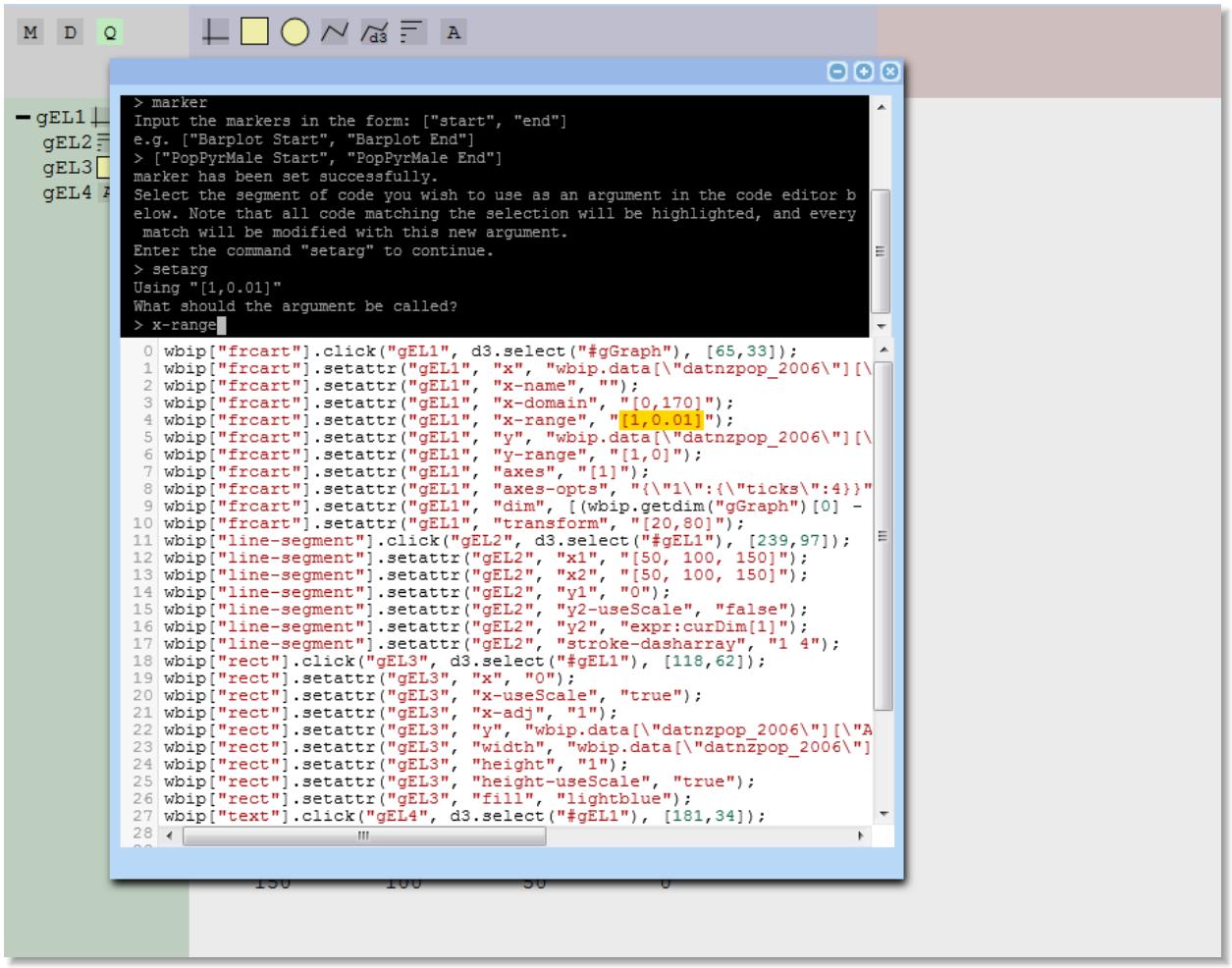


Figure 5.47: Formalised reuse of code using the Functionise interface.

Figure 5.47: To begin, the code that is to be functionised must be identified. This can be done by line number, or to be more robust, by marker. In the above Figure, the marker lines `// MARKER PopPyrMale Start` and `// MARKER PopPyrMale End` were added around the code that is used to create the left-side of the pyramid. When given the line numbers or the markers, the interface will expand to show the identified code. By following the instructions in the interface, we identify segments of code that need to be changed by highlighting them in the interface. The code is identified not by their position, but by matching the highlighted text. All highlighted text will be turned into an *argument*, so the user must take care to only match the desired code. For instance to turn `x-adj` into an argument, a simple highlight of `"1"` would also match `height` (Figure 5.48), which is not desired. A more unique text pattern, `"x-adj", "1"` is used instead (Figure 5.49). In the case of `Male` however, a double-match is desirable as it will change both the data used and the label attached to the graph, in a single argument (Figure 5.50). When we are done setting the arguments, we finish off and give our new 'function' a name (Figure 5.51).

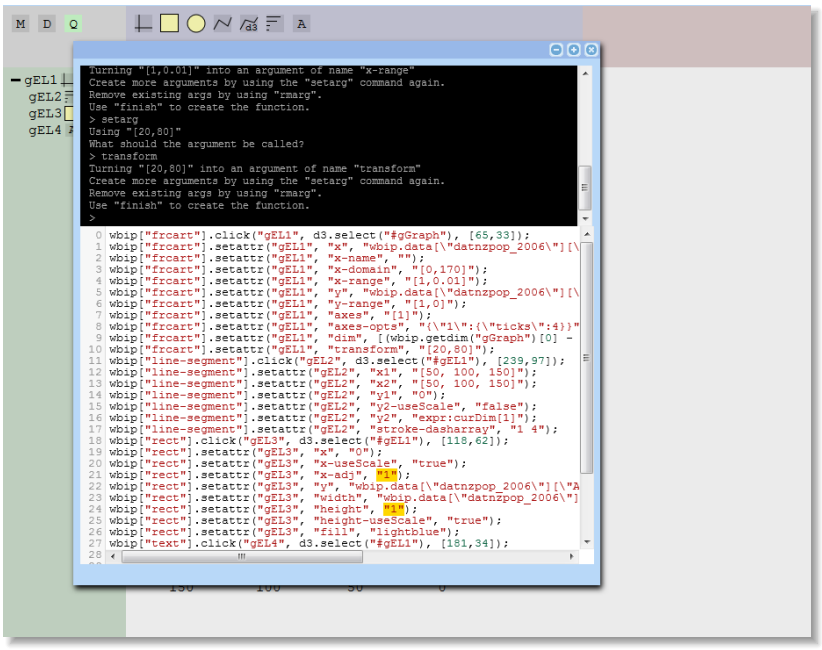


Figure 5.48: Check the highlighted text carefully to ensure only what we want is turned into arguments.

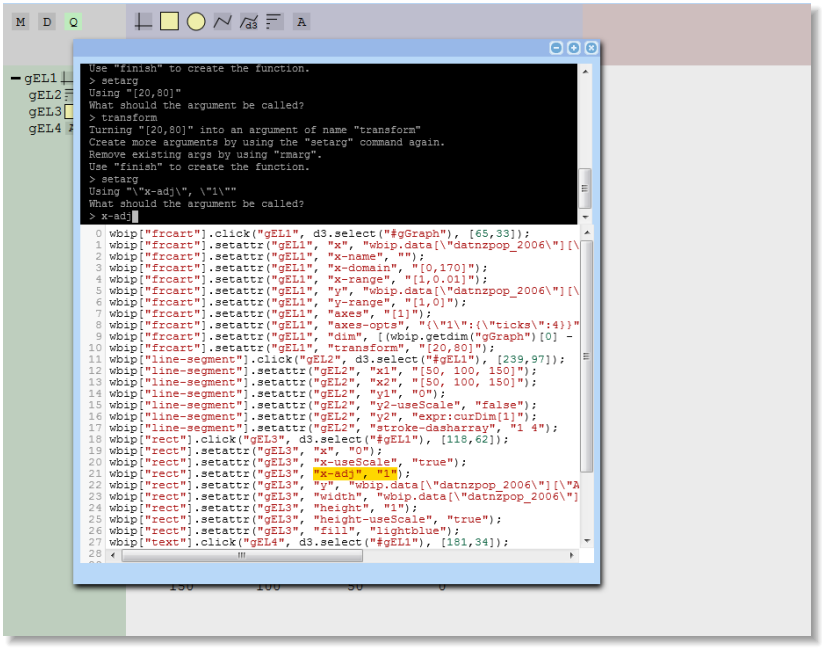


Figure 5.49: Sometimes a more cumbersome match may be required to get a unique match.

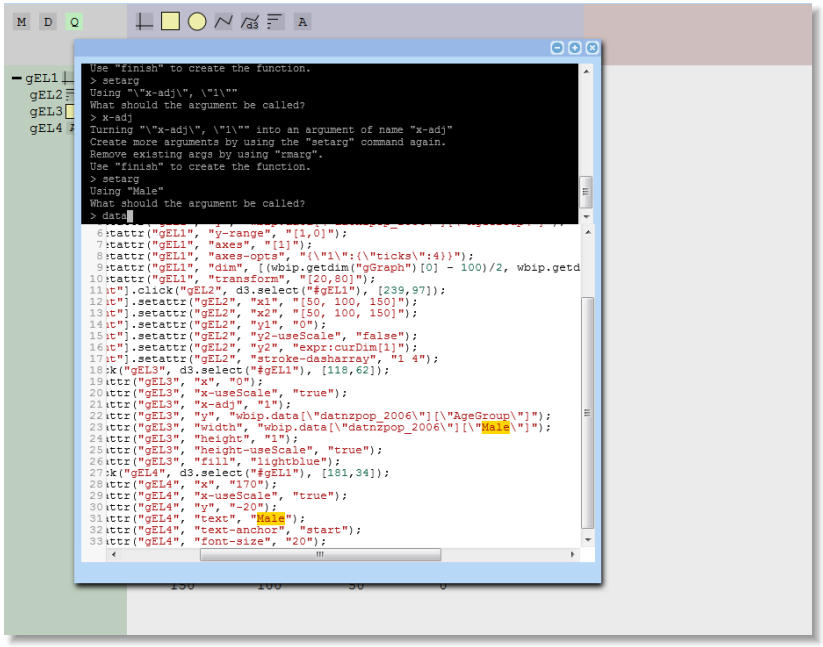


Figure 5.50: In other cases multiple matches may be desirable. As the right-side will be using Female data, all cases of Male being updated to Female is exactly what we want.

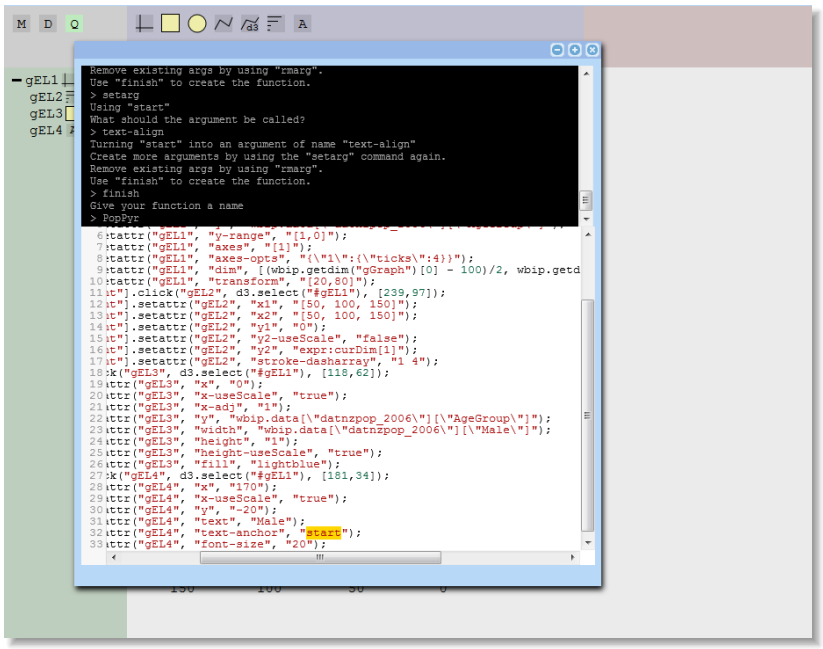


Figure 5.51: With all the arguments set, we finish creating the function.

5.5.10 Use the new Function

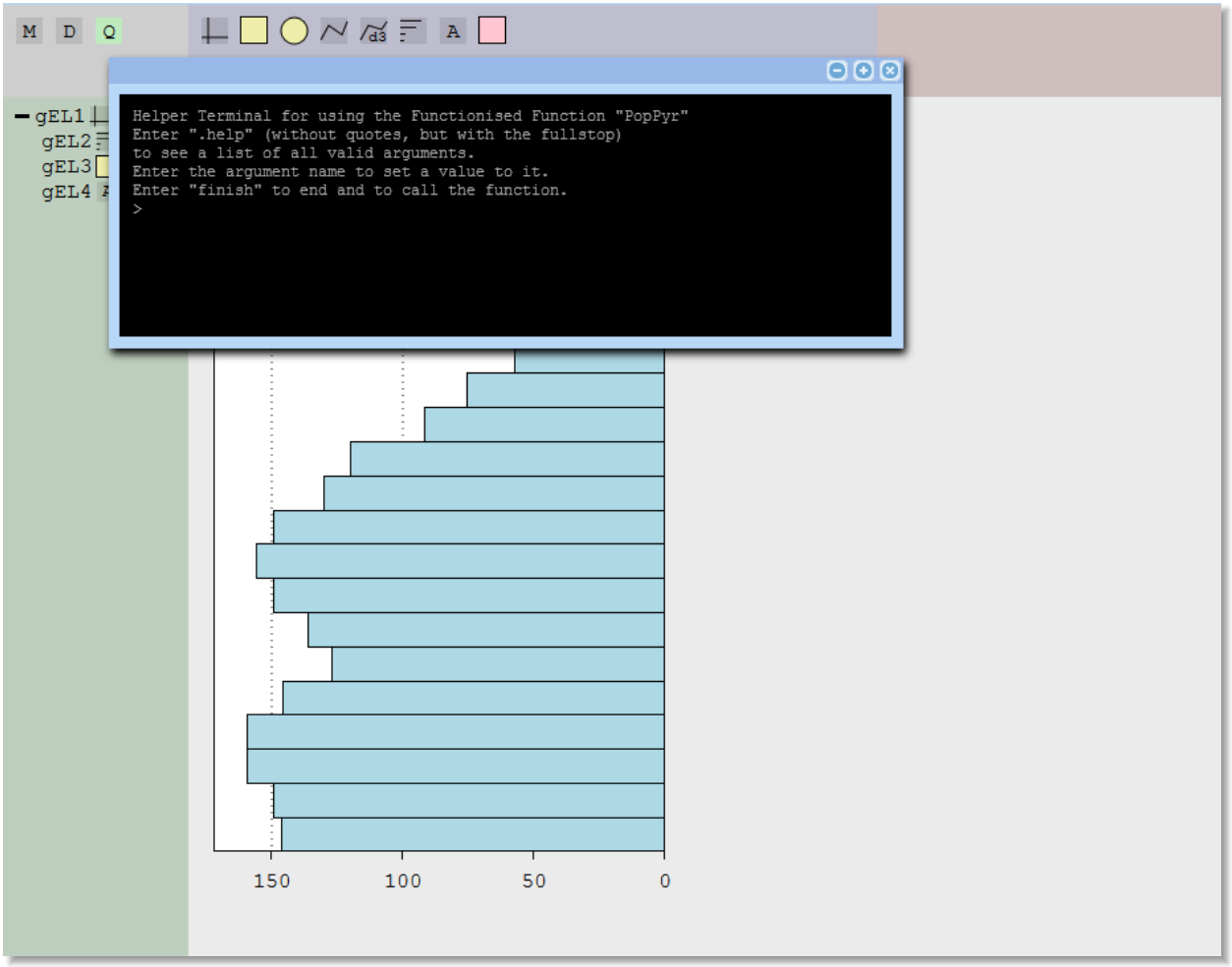


Figure 5.52: Clicking on the newly created button will open an interface to help use the function.

Figure 5.52: When code has been Functionised, the newly created function becomes available as a button in the menu at the top. Clicking this button will option a new interface for using the function. Through this interface, we can see what the available arguments are (Figure 5.53), and begin assigning new values to these arguments (Figure 5.54). In the same vein as computing the correct dimensions as described in Figure 5.23, we can assign a computed transform to position the right-side of the pyramid correctly. That is, 20 pixels margin + the width of the left-side pyramid + 60 pixels for the labels in-between the two sides (Figure 5.55).

```
1 [(wbip.getdim(\"gGraph\")[0] - 100)/2 + 80, 80]
```

When specifying the argument values, the interface displays the text that the argument is being matched to (and hence being changed), so this should be taken into account when supplying the new values, e.g. for the match `"x-adj", "1"`, the new argument value supplied should be `"x-adj", "0"`, and not just `0` on its own (Figure 5.56). Once all the arguments have been supplied, we finish (Figure 5.57), and WeBIPP will run the function, creating the right-side of the pyramid (Figure 5.58).

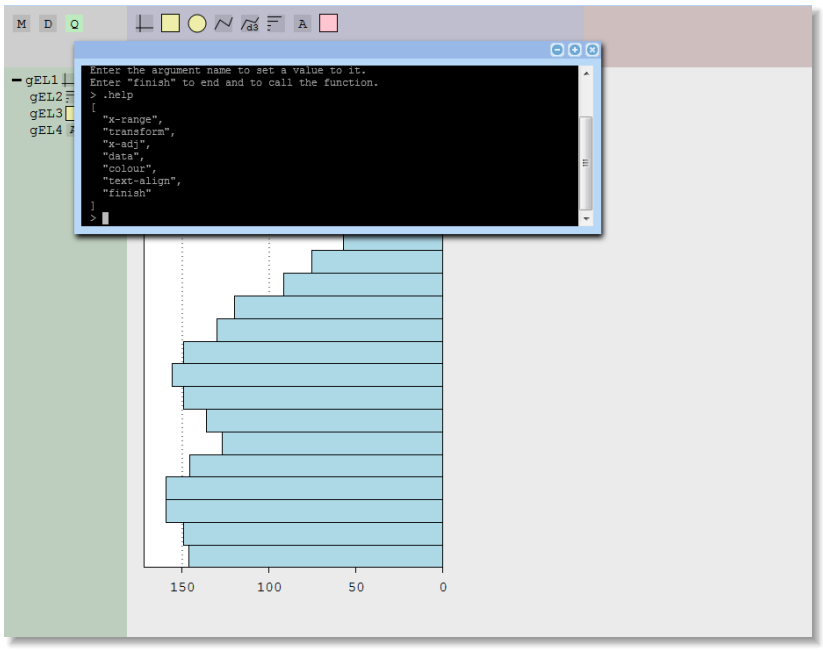


Figure 5.53: The available arguments.

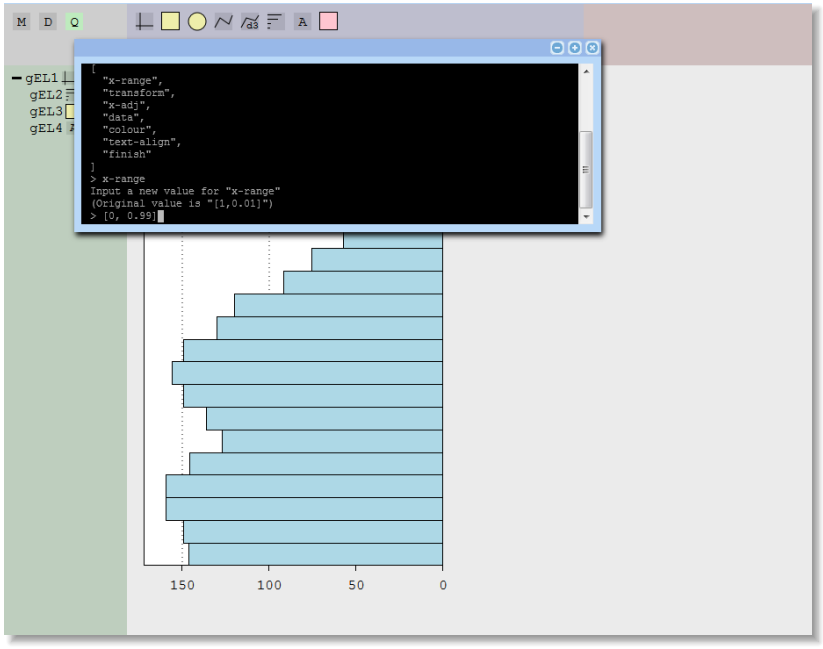


Figure 5.54: Setting the arguments.

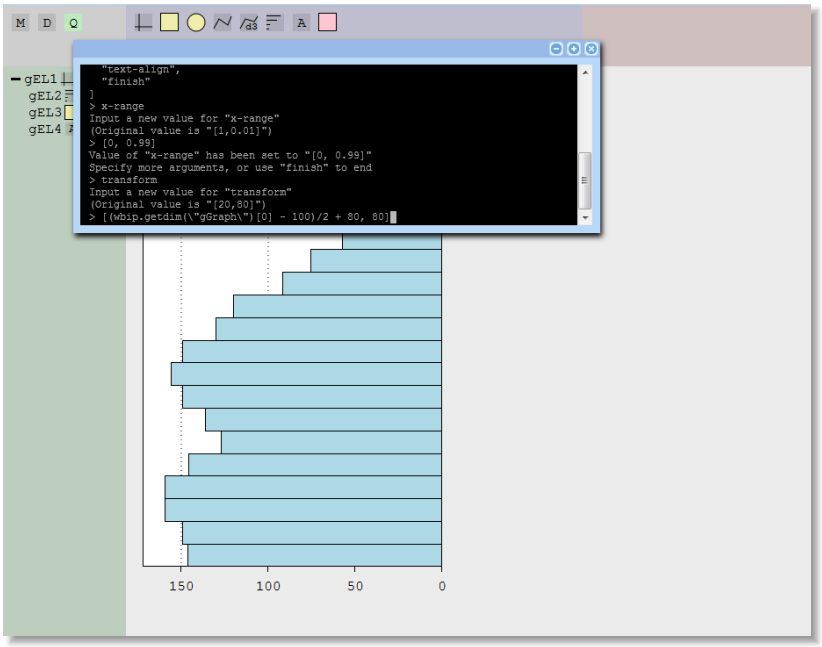


Figure 5.55: We can also compute argument values, as before.

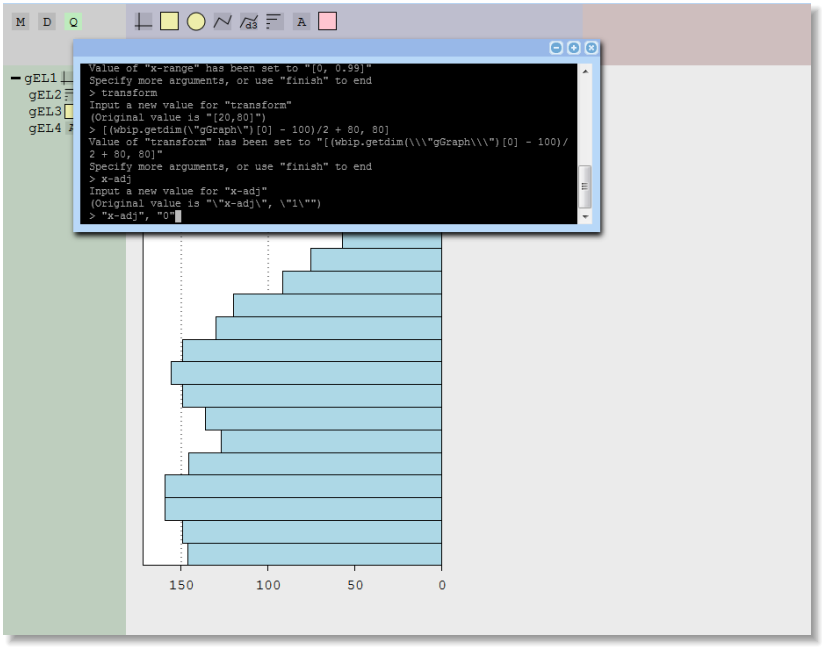


Figure 5.56: Be sure to match the text that has been turned into an argument.

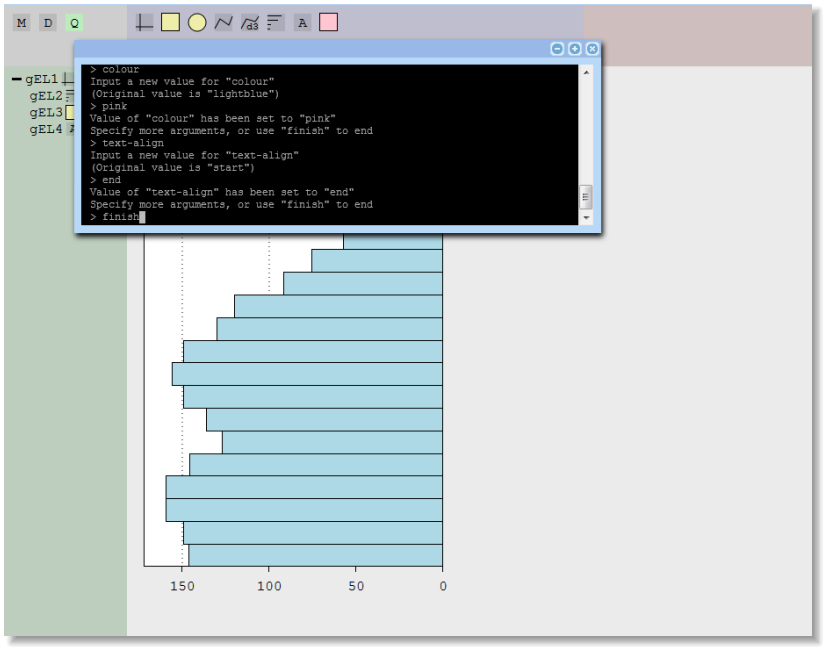


Figure 5.57: Finish off the call.

Figure 5.58: The right-side of the pyramid complete.

5.5.11 Draw the y-axis Labels

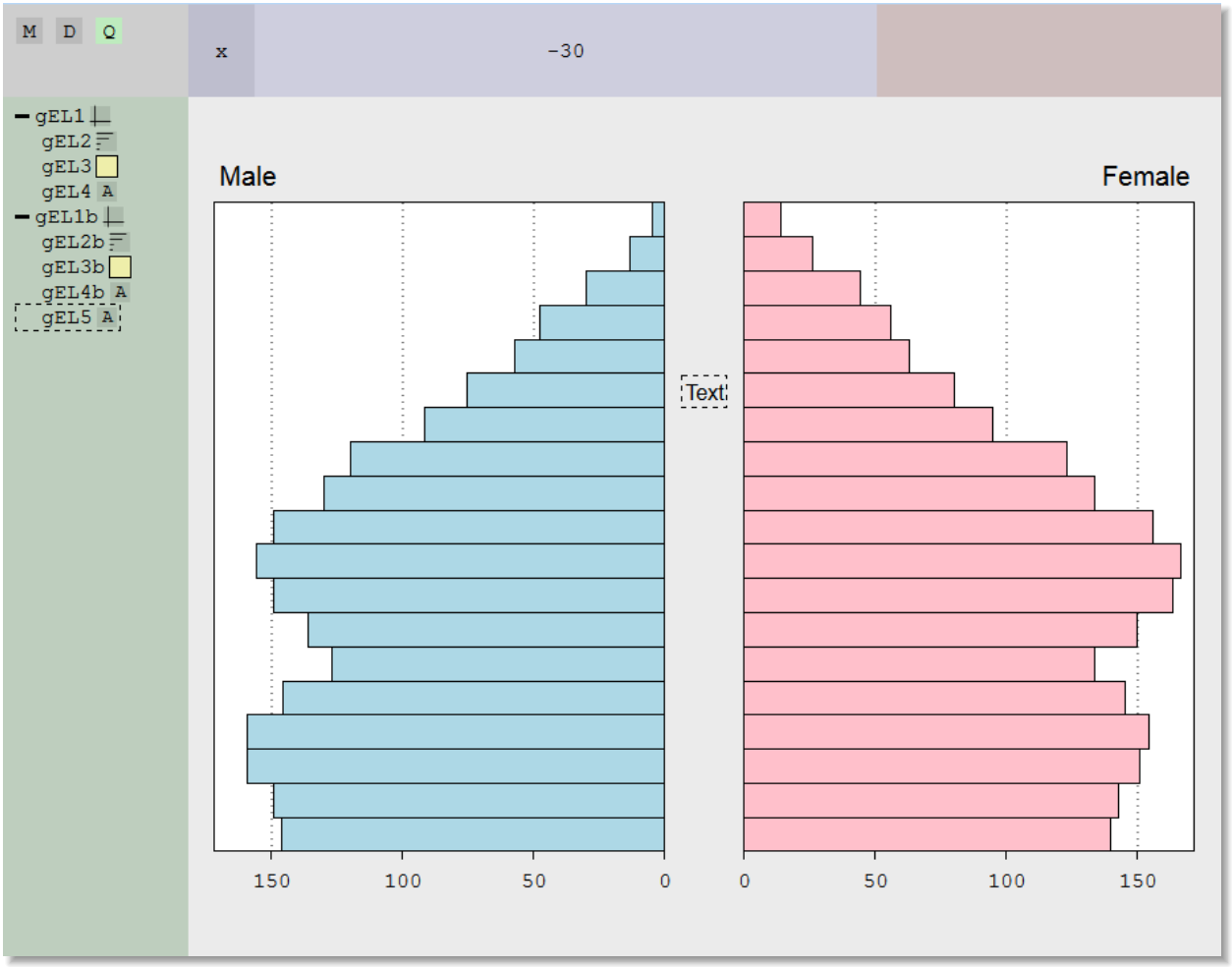


Figure 5.59: Placing the shared y axis label.

Figure 5.59: With the two-sides done, we finish off by placing the shared labels. First is the label for the y axis. To ensure it is centred correctly, we place a Text Object into the right-side of the pyramid and set `x` to `-30` (as we computed a space of 60 pixels for the label). This is easier than placing it elsewhere and having to do a more complex calculation involving dimensions. We assign the data variable `AgeGroup` to both `y` (Figure 5.60) and `text` (Figure 5.61), which finishes the label.

Figure 5.60: Assign `AgeGroup` to y for positioning.

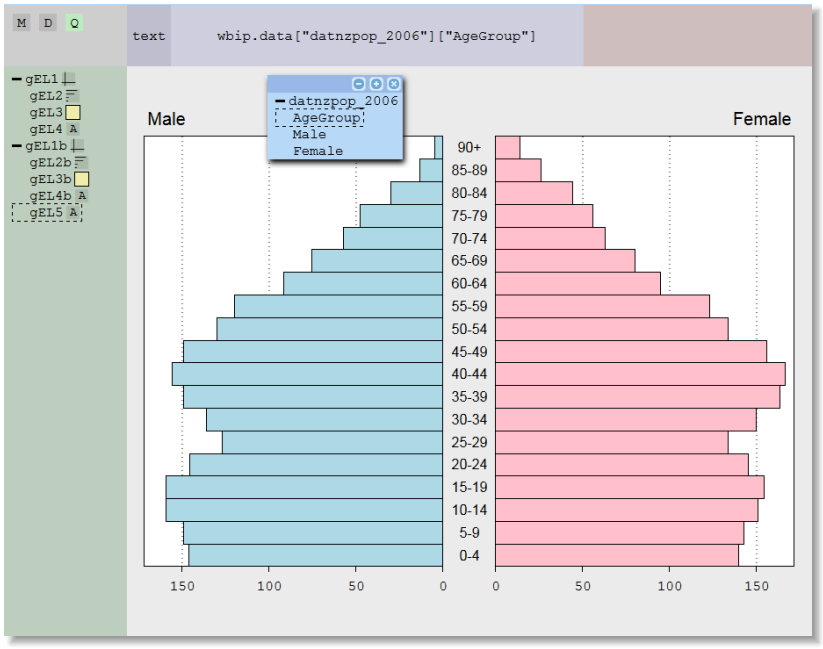


Figure 5.61: Assign `AgeGroup` to text for the axis labels.

5.5.12 Draw the Title

Figure 5.62: Placing an overall Title for the graph.

Figure 5.62: For the title at the top, we again place a Text Object in the right-side of the pyramid and set `x` to `-30` for correct centring. Adjusting `y` to `-45` positions the label nicely above the graph. After setting the `text` and a suitable `font-size` (30), we can also bold the Title by setting `font-weight` (Figure 5.63), and change the font style by changing `font-family` (Figure 5.64).

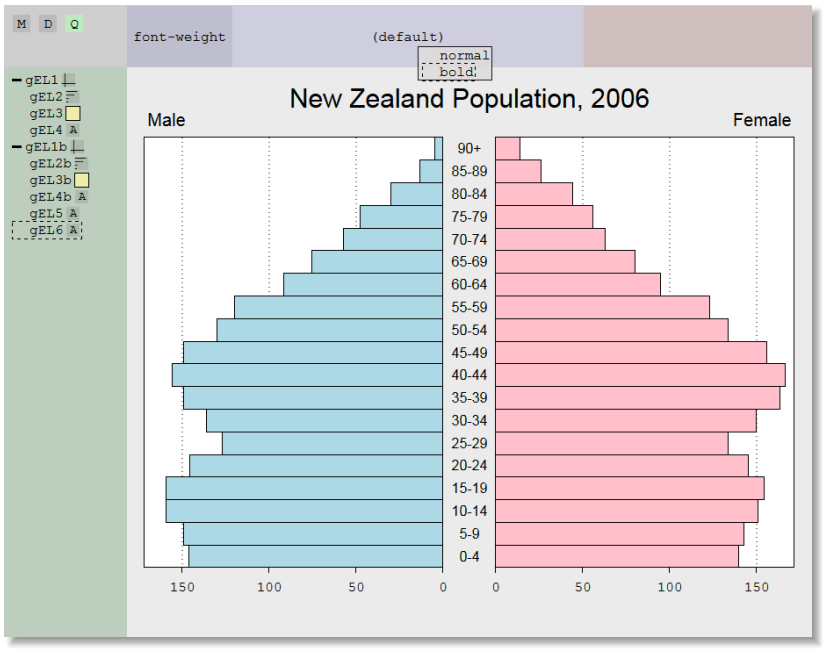


Figure 5.63: To make the Title more prominent, we can bold it.

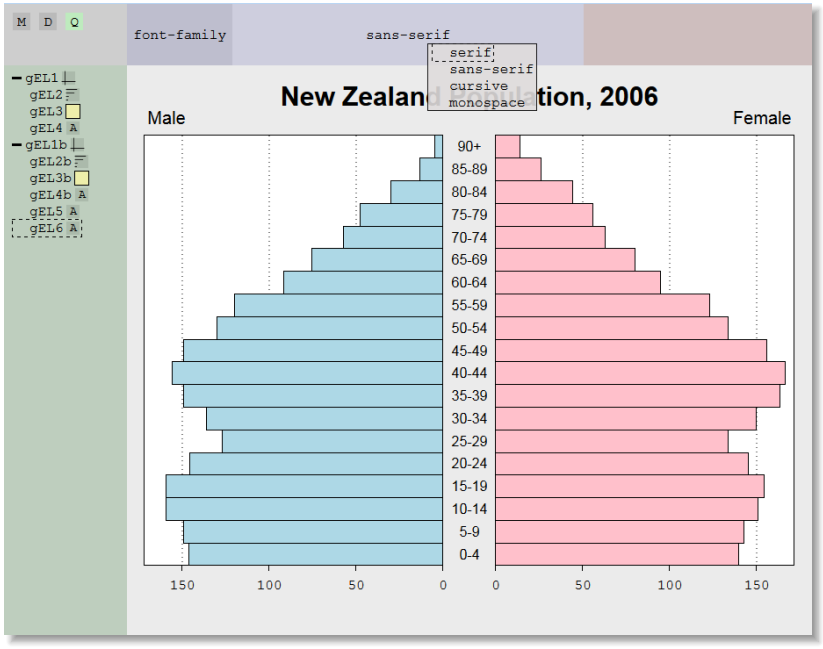


Figure 5.64: We can also change the font of the title.

5.5.13 Draw the x-axis Label

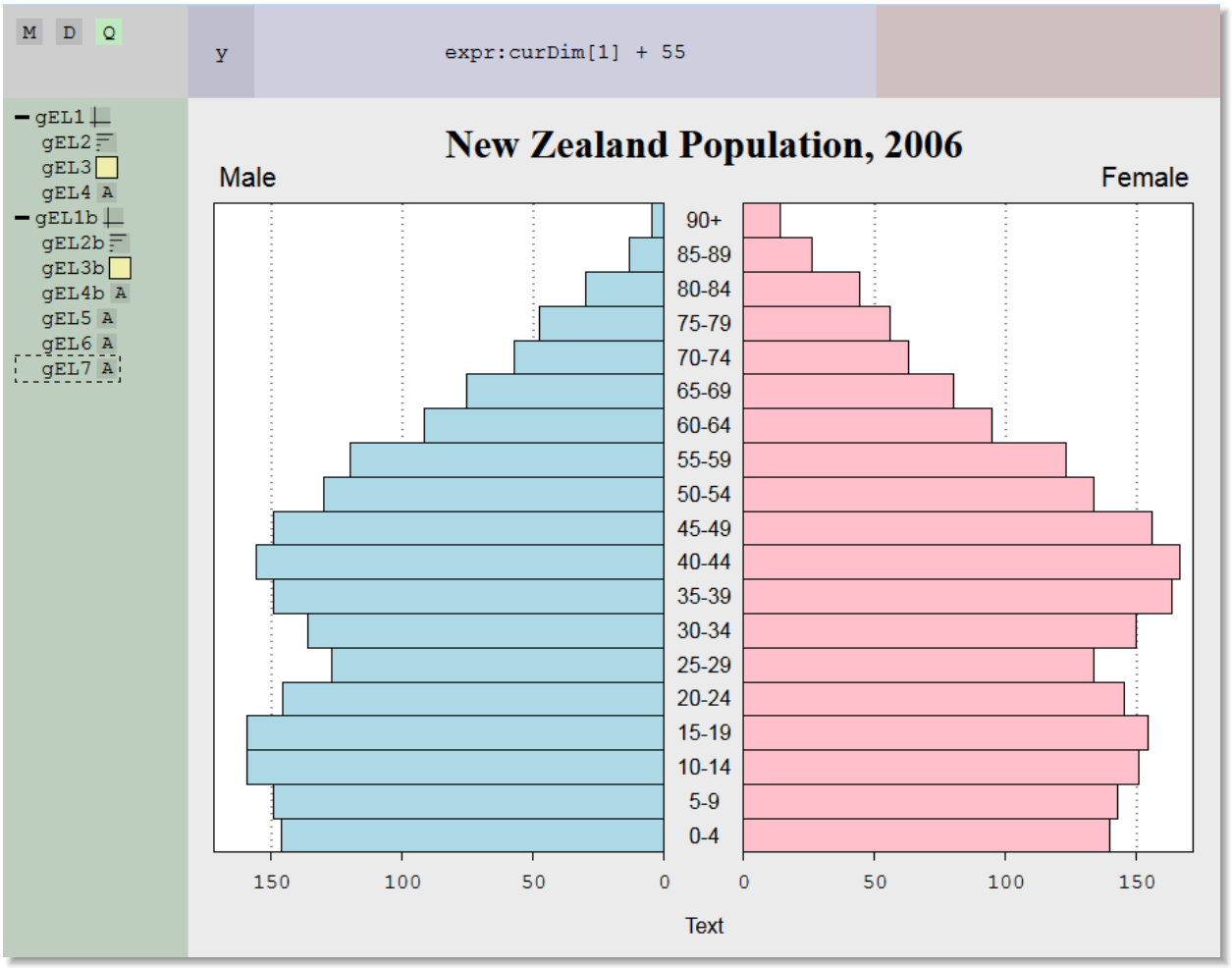


Figure 5.65: Placing an overall x axis label.

Figure 5.65: There is unfortunately no easy way to position the label at the bottom. We must compute its vertical position using a similar method for the dotted lines in the background. We set `y-useScale` to `false`, and set `y` to `expr:curDim[1] + 55`. Other than that, it is a simple process to set the `text` and `font-size`.

5.5.14 Resizing the Finished Pyramid

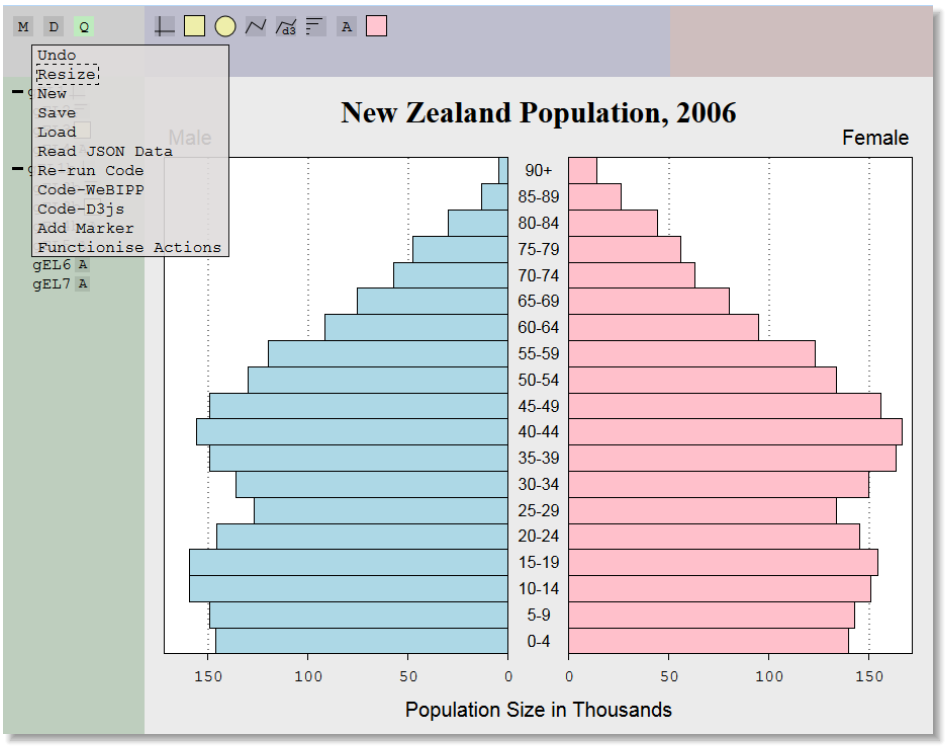


Figure 5.66: The finished Population Pyramid, which is fully scalable.

Figure 5.66: The Population Pyramid is complete. One of the major advantages of computing the various layout details, rather than doing it by eye, is that the graphic is now fully scalable. We can resize the WeBIPP interface by using `Resize` from the main menu, which will open a similar interface to `dim`. Upon resize, WeBIPP will redraw the graphic.

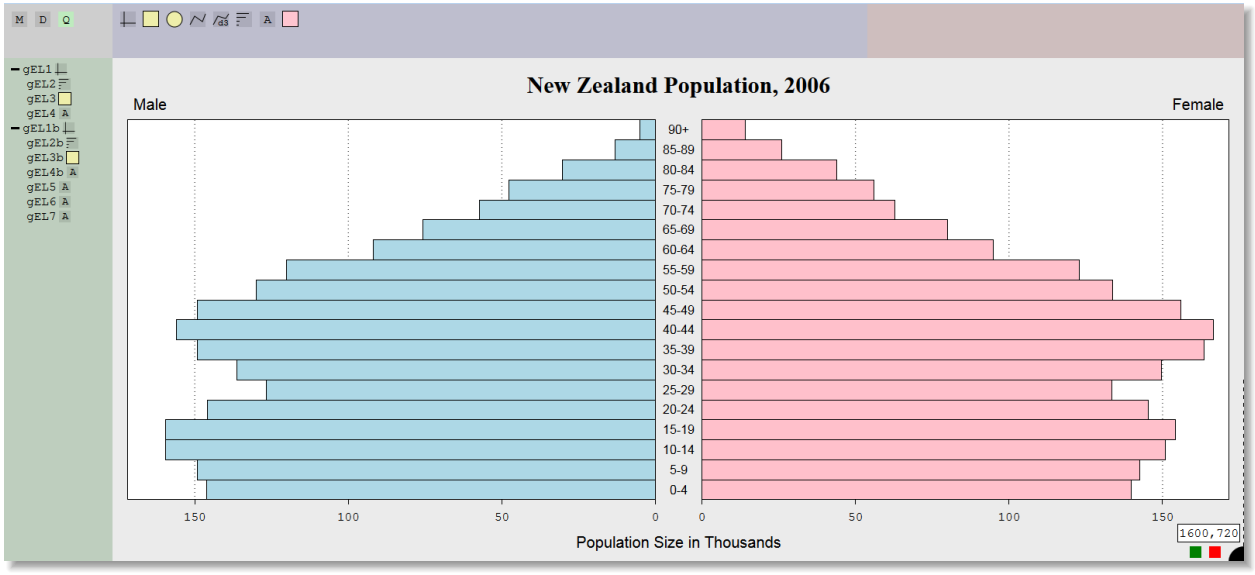


Figure 5.67: The interface resized to 1600 by 720.

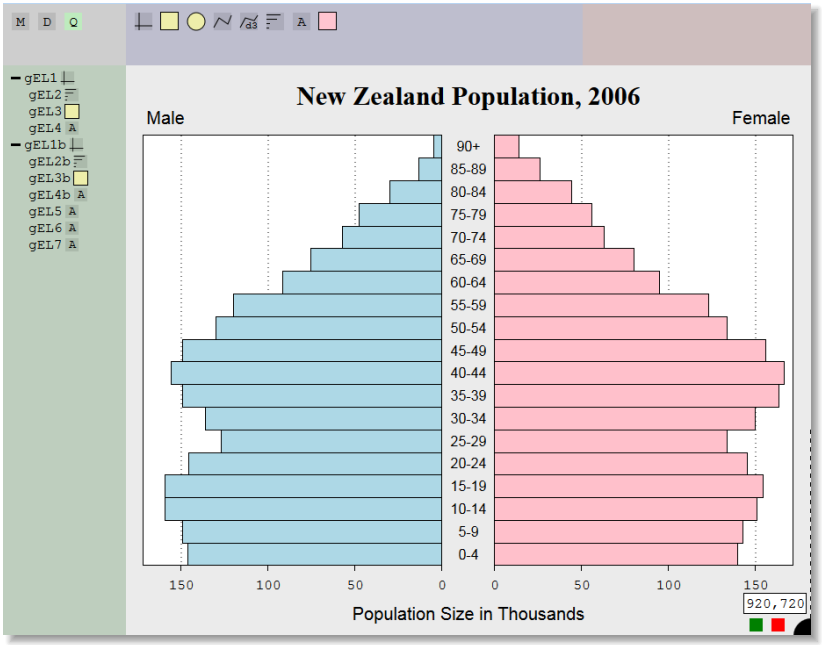


Figure 5.68: The resize interface with the default size of 920 by 720.

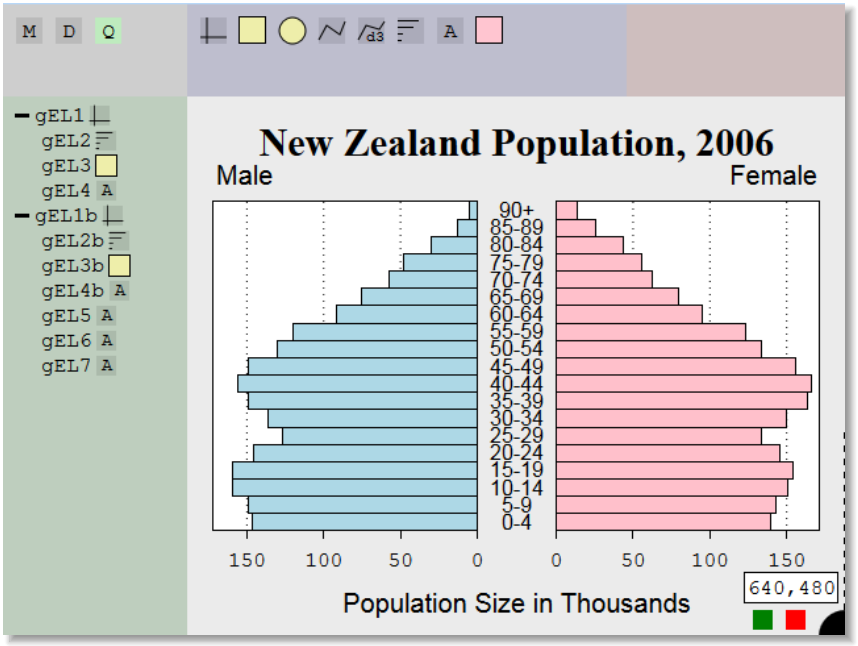


Figure 5.69: The interface resized to 640 by 480. Notice that our chosen font sizes may not be appropriate at other interface sizes.

5.5.15 The Final Save Data

The save data for the Population Pyramid, less the save header and the data, is copied below. Notice the identifying MARKER lines (lines 2 and 37) and the Functionise lines that re-uses this code (lines 38 and 39). Even with some interfaces missing, the Population Pyramid can be reproduced in 10 – 20 minutes using WeBIPP, though some knowledge of SVG graphics is required. Writing the code in R to draw Figure 5.17 takes quite a bit longer, and would also require familiarity with base R graphics.

As the Functionise interface is developed further, once a graphic like this is produced, it should be possible to Functionise the whole thing and convert it into an Addon. This would enable creative users who lack programming expertise, to create a new and innovative graphic in WeBIPP, and then share this for others to use, not just as a concept picture for others to reproduce themselves, but as an easy-to-use Addon.

```
1 wbip.data["datnzpop_2006"] = JSON.parse(/*Data removed for
    space*/);
2 // MARKER PopPyrMale Start
3 wbip["frcart"].click("gEL1", d3.select("#gGraph"), [65,33]);
4 wbip["frcart"].setattr("gEL1", "x", "wbip.data[\"datnzpop_
    2006\"][\"Female\"]");
5 wbip["frcart"].setattr("gEL1", "x-name", "");
6 wbip["frcart"].setattr("gEL1", "x-domain", "[0,170]");
7 wbip["frcart"].setattr("gEL1", "x-range", "[1,0.01]");
8 wbip["frcart"].setattr("gEL1", "y", "wbip.data[\"datnzpop_
    2006\"][\"AgeGroup\"]");
9 wbip["frcart"].setattr("gEL1", "y-range", "[1,0]");
10 wbip["frcart"].setattr("gEL1", "axes", "[1]");
11 wbip["frcart"].setattr("gEL1", "axes-opts", "{\"1\":{\"ticks\
    ":4}}");
12 wbip["frcart"].setattr("gEL1", "dim", [(wbip.getdim("gGraph")
    [0] - 100)/2, wbip.getdim("gGraph")[1] - 80 * 2]);
13 wbip["frcart"].setattr("gEL1", "transform", "[20,80]");
14 wbip["line-segment"].click("gEL2", d3.select("#gEL1"),
    [239,97]);
15 wbip["line-segment"].setattr("gEL2", "x1", "[50, 100, 150]");
16 wbip["line-segment"].setattr("gEL2", "x2", "[50, 100, 150]");
17 wbip["line-segment"].setattr("gEL2", "y1", "0");
18 wbip["line-segment"].setattr("gEL2", "y2-useScale", "false");
19 wbip["line-segment"].setattr("gEL2", "y2", "expr:curDim[1]");
20 wbip["line-segment"].setattr("gEL2", "stroke-dasharray", "1 4
    ");
21 wbip["rect"].click("gEL3", d3.select("#gEL1"), [118,62]);
22 wbip["rect"].setattr("gEL3", "x", "0");
23 wbip["rect"].setattr("gEL3", "x-useScale", "true");
24 wbip["rect"].setattr("gEL3", "x-adj", "1");
25 wbip["rect"].setattr("gEL3", "y", "wbip.data[\"datnzpop_2006\
    "][\"AgeGroup\"]");
26 wbip["rect"].setattr("gEL3", "width", "wbip.data[\"datnzpop_
```

```
   2006\"][\"Male\"]");
wbip["rect"].setattr("gEL3", "height", "1");
wbip["rect"].setattr("gEL3", "height-useScale", "true");
wbip["rect"].setattr("gEL3", "fill", "lightblue");
wbip["text"].click("gEL4", d3.select("#gEL1"), [181,34]);
wbip["text"].setattr("gEL4", "x", "170");
wbip["text"].setattr("gEL4", "x-useScale", "true");
wbip["text"].setattr("gEL4", "y", "-20");
wbip["text"].setattr("gEL4", "text", "Male");
wbip["text"].setattr("gEL4", "text-anchor", "start");
wbip["text"].setattr("gEL4", "font-size", "20");
// MARKER PopPyrMale End
wbip.fise.new("PopPyr", {"marker":["PopPyrMale Start","
   PopPyrMale End"]}, [{"match":"[1,0.01]","name":"x-range","
   type":"none"},{"match":"[20,80]","name":"transform","type"
   :"none"},{"match":"\"x-adj\", \"1\"","name":"x-adj","type"
   :"none"},{"match":"Male","name":"data","type":"none"},{"
   match":"lightblue","name":"colour","type":"none"},{"match"
   :"start","name":"text-align","type":"none"}]);
wbip.vars.fise["PopPyr"]({"x-range":"[0, 0.99]","transform":"
   [(wbip.getdim(\\\"gGraph\\\")[0] - 100)/2 + 80, 80]","x-
   adj":"\"x-adj\", \"0\"","data":"Female","colour":"pink","
   text-align":"end"});
wbip["text"].click("gEL5", d3.select("#gEL1b"), [44.5,143]);
wbip["text"].setattr("gEL5", "x", "-30");
wbip["text"].setattr("gEL5", "y", "wbip.data[\"datnzpop_2006\
   "][\"AgeGroup\"]");
wbip["text"].setattr("gEL5", "text", "wbip.data[\"datnzpop_
   2006\"][\"AgeGroup\"]");
wbip["text"].click("gEL6", d3.select("#gEL1b"), [44.5,68]);
wbip["text"].setattr("gEL6", "x", "-30");
wbip["text"].setattr("gEL6", "y", "-45");
wbip["text"].setattr("gEL6", "text", "New Zealand Population,
    2006");
wbip["text"].setattr("gEL6", "font-size", "30");
wbip["text"].setattr("gEL6", "font-weight", "bold");
wbip["text"].setattr("gEL6", "font-family", "serif");
wbip["text"].click("gEL7", d3.select("#gEL1b"), [84.5,446]);
wbip["text"].setattr("gEL7", "x", "-30");
wbip["text"].setattr("gEL7", "y-useScale", "false");
wbip["text"].setattr("gEL7", "y", "expr:curDim[1] + 55");
wbip["text"].setattr("gEL7", "text", "Population Size in
   Thousands");
wbip["text"].setattr("gEL7", "font-size", "20");
```

5.6 How WeBIPP Works

As mentioned in the Introduction (see Subsection 5.2.2), WeBIPP can be thought of as a set of bridges connecting the GUI and the Code. Broadly it consists of three parts:

GUI - The Graphical User Interface, which includes the visualisation being created. User interaction with interface elements and the graphic in production will lead to the generation of *High-level Code*.

High-level Code - Simple instructions that can map to interactions with the GUI. The High-level Code is used to generate *Low-level Code*, and is also a record of all meaningful user interactions with the GUI. The High-level Code alone is a complete record of the interactions and can completely replicate the output graphic.

Low-level Code - Mainly D3 code, but with other low-level JavaScript code, including those provided by WeBIPP. This low-level code is used to manipulate the SVG that is both the GUI and the visualisation that is being created.

Both the High-level and Low-level Code are stored within CodeMirror instances that are embedded (as ForeignObjects) into the same SVG that the GUI resides. Thus most of the information is stored within the XML document that is the SVG 'image' (see Figure 5.70), though some information is stored as JavaScript objects (in particular, see Subsection 5.6.2). WeBIPP is designed to be highly modular built with the philosophy that *everything is an addon*. Thus even core elements of WeBIPP are written as separate 'addons', meaning that WeBIPP as a whole was developed to be very addon friendly. These addons can be grouped into two categories, those that simply provide functions that can be called by other addons, and those that are evaluated during the WeBIPP initialisation process, usually to add a button to the menu for use by the user. We will define both categories to be *Internal Libraries*, but the latter to be *Addons* that are loaded during the initialisation process. In addition to Internal Libraries, WeBIPP also makes use of several *External Libraries*, such as D3 and CodeMirror. Bringing these libraries together is the true core of WeBIPP appropriate named `WeBIPP_Core` (hereafter referred to as WeBIPP Core or `Core`). WeBIPP Core handles the initialisation process by making the calls to set up the GUI and to load the Addons (see Figure 5.73).

In keeping with the philosophy that everything is an addon, `Core` does not need to understand how each of the loaded Addons work, and the Addons in turn also do not need to understand how `Core` works. The two parts stick to a kind of formula, and in so doing can make use of each other without understanding the details. For instance, when a menu object is selected and added to the Graph Region (see Figure 5.74), `Core` simply generates a high level click call for that object, and `Core` does not need to know what this

```
<svg id="main-svg" width="920" height="720"> ev
  <defs></defs>
  <g id="gGraph" class="wbip-frame" transform="translate(140,70)"></g>
  <g id="gMenuTab" transform="translate(0,70)"></g>
  <g id="gMenuPri" class="wbip-menu" transform="translate(140,0)"></g>
  <g id="gMenuSec" class="wbip-menu" transform="translate(660,0)"></g>
  <g id="gMenuTer" class="wbip-menu" transform="translate(0,0)"></g>
  <g id="wbip-code-high" class="wbip-ffw" transform="translate(199,100
  <g id="wbip-code-low" class="wbip-ffw" transform="translate(250,200)
    <rect class="wbip-ffw-bar" width="600" height="20"></rect>
    <g class="wbip-ffw-inner" transform="translate(0,20)"> ev
      <rect class="wbip-ffw-inner-bg" width="600" height="400"></rect>
      <foreignobject width="600" height="400">
        <body>
          <div id="wbip-code-low-cm">
            <div class="CodeMirror cm-s-default CodeMirror-wrap" style=
          </div>
        </body>
      </foreignobject>
    </g>
    <g class="wbip-ffw-bar-icons" transform="translate(540,0)"></g>
  </g>
</svg>
```

Figure 5.70: The SVG containing the WeBIPP GUI. The first five groups (`<g>` with IDs `gGraph`, `gMenuTab`, `gMenuPri`, `gMenuSec` and `gMenuTer`) form the GUI, while the next two groups (with IDs `wbip-code-high` and `wbip-code-low`) form the Free Form Windows that contain ForeignObjects that hold the CodeMirror instances.

click call does. Likewise when an object's attributes need to be changed (see Figure 5.75), `Core` simply generates a set attribute call for the selected object. The Object library can run its own code to do something special, and `Core` has no need to understand how this works. However in the case of changing attributes there is a powerful set attribute function provided by `Core` that the Addon can use (see Section 5.9), which makes it more convenient when creating new Object Addons.

5.6.1 Frames

Most good statistical graphics begin with well-defined axes. More generally, creating accurate and useful statistical graphics often requires the use of arbitrary coordinate spaces so that the positioning and size of graphical objects on this coordinate space can encode data in a meaningful way. WeBIPP accomplishes this by the use of *Frames*. Frames are intended to be a general framework under which arbitrary coordinate spaces can be assigned, though

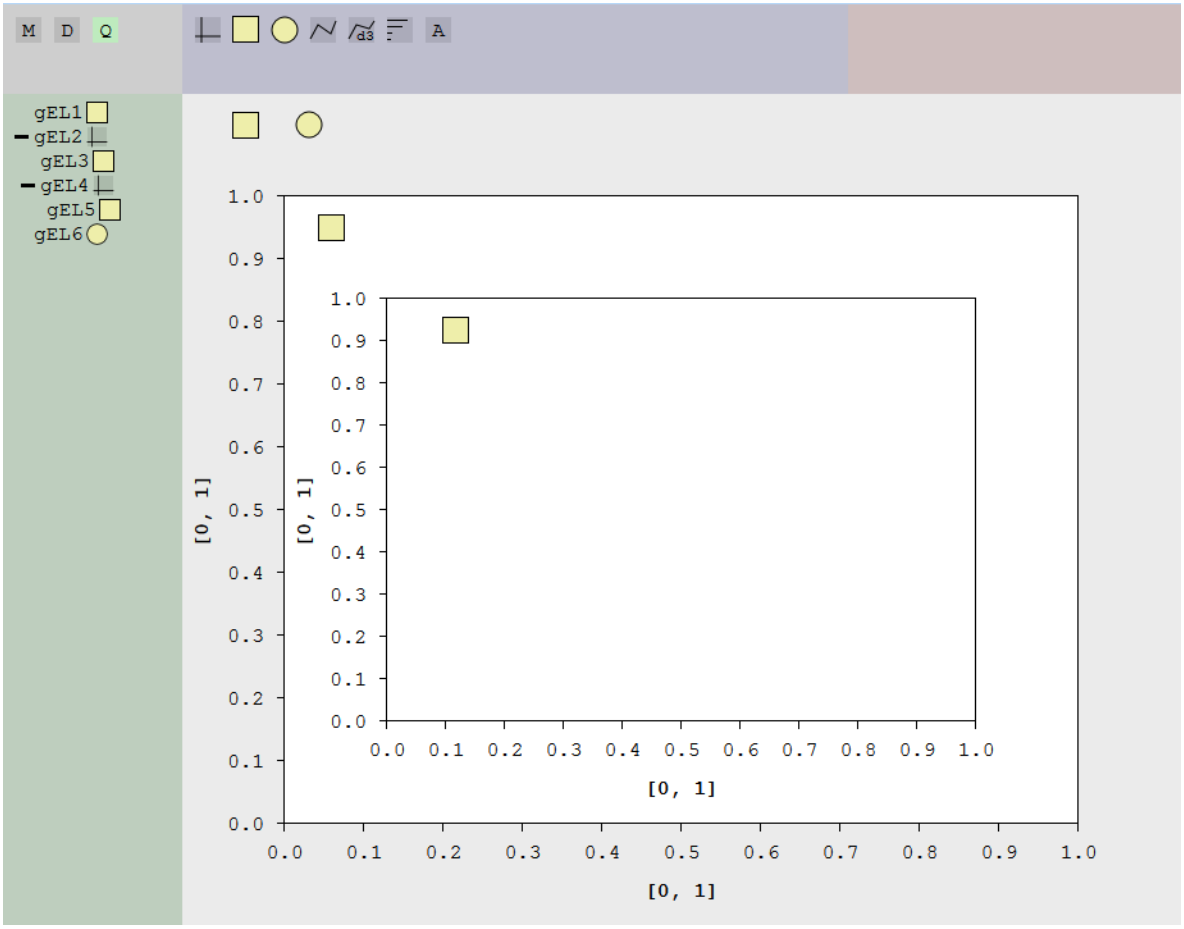


Figure 5.71: An example of Frames and nesting. gELs 1, 2 and 6 are children of the Graph Region itself. gELs 3 and 4 and children of gEL 2. gEL 6 is a child of gEL 4.

currently the only type of frame that exists is a Cartesian Frame. Hence the current framework most suits Cartesian Frames. It is expected the framework will evolve considerably as other types of frames are added. A frame has the following properties:

- Has the class `wbip-frame`.
- Has a well-defined dimension (`dim`).
- Provides Scales, which define the arbitrary coordinate spaces.

When an object is added to the Graph Region, it is attached as a child of the foremost frame corresponding to the mouse event which triggered the process (see Figure 5.71). Objects may use their parent frame's Scales to make use of the frame's coordinate space. Consistent with this, the Graph Region itself is also a frame, allowing objects to be attached directly to the Graph Region, though its Scales correspond directly to pixels and are generally not useful.

5.6.2 IDvar

When creating more complicated objects, it is often necessary to store additional information related to that object. It is possible, but inconvenient, to store this additional information in the SVG itself. WeBIPP instead stores the information in JavaScript using a system called *IDvar*. An IDvar is simply a JavaScript Object that is associated with the given ID. It should be created using the accessor function `wbip.setIDvar` and accessed with the accessor function `wbip.getIDvar`. For example:

```
1 // Create an example JavaScript Object
2 var ExObj = {aaa: "hello", bbb: "bye"};
3 // Save this Object to an IDvar
4 wbip.setIDvar("someID", ExObj);
5 // Retrieve the IDvar for use
6 wbip.getIDvar("someID").aaa;
7 // The value is the contents of aaa, "hello"
```

5.6.3 Structure Diagrams

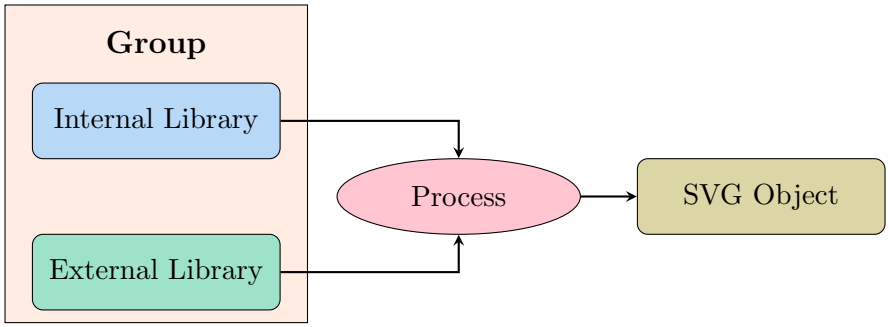


Figure 5.72: The icons that will be used for the Structure diagrams.

Internal Library e.g. `WeBIPP_Core.js`

External Library e.g. `d3.js`

Process e.g. Do something

SVG Object An object inside the SVG, which could be a grouping representing some area

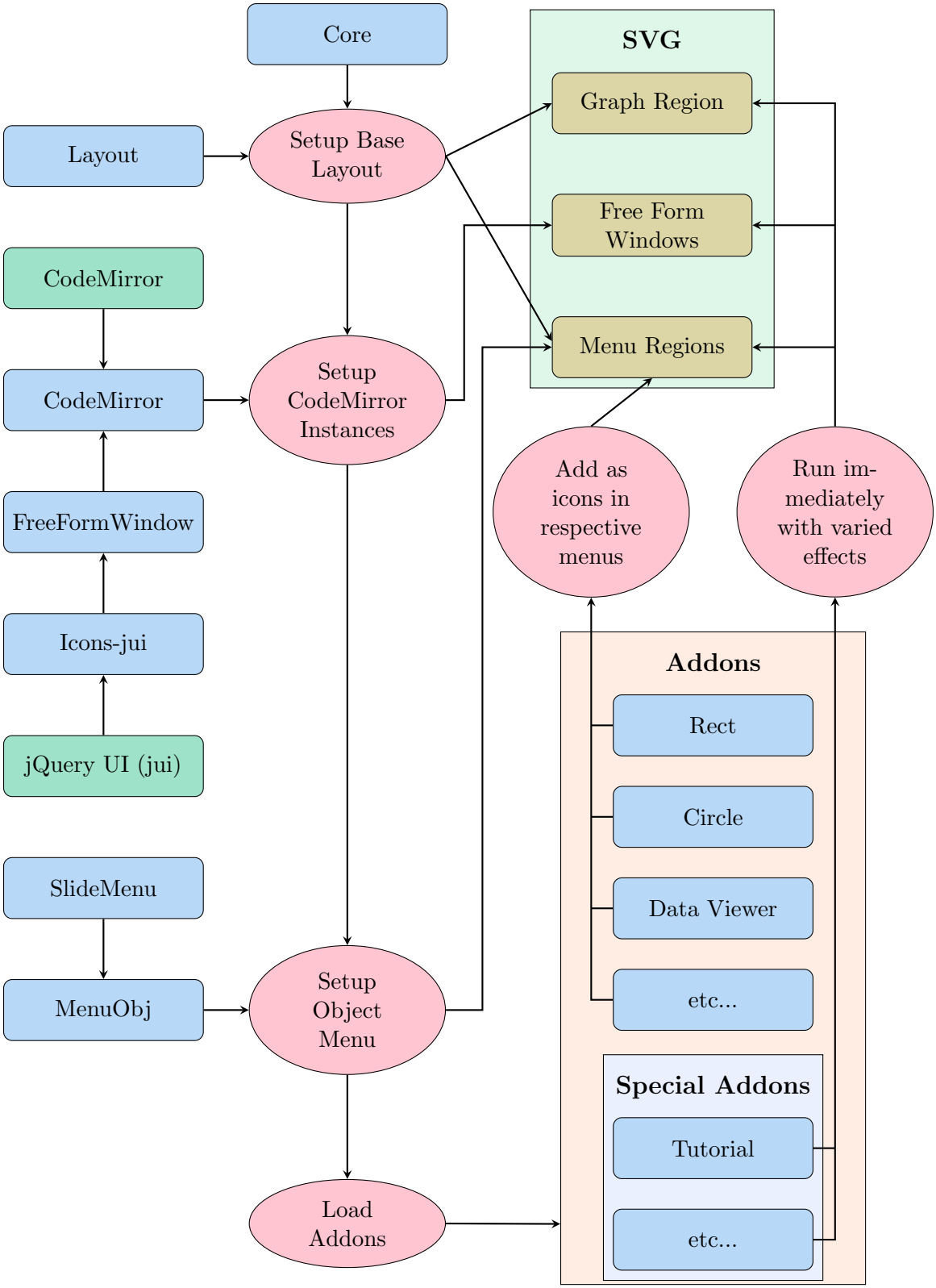


Figure 5.73: Structure of the Initialisation process. Not mentioned are the D3 External Library and the WeBIPP's Utils Internal Library. Almost everything in WeBIPP makes use of the Utils library and all drawing on the SVG is done through D3. Some utility functions in the D3 library are also used.

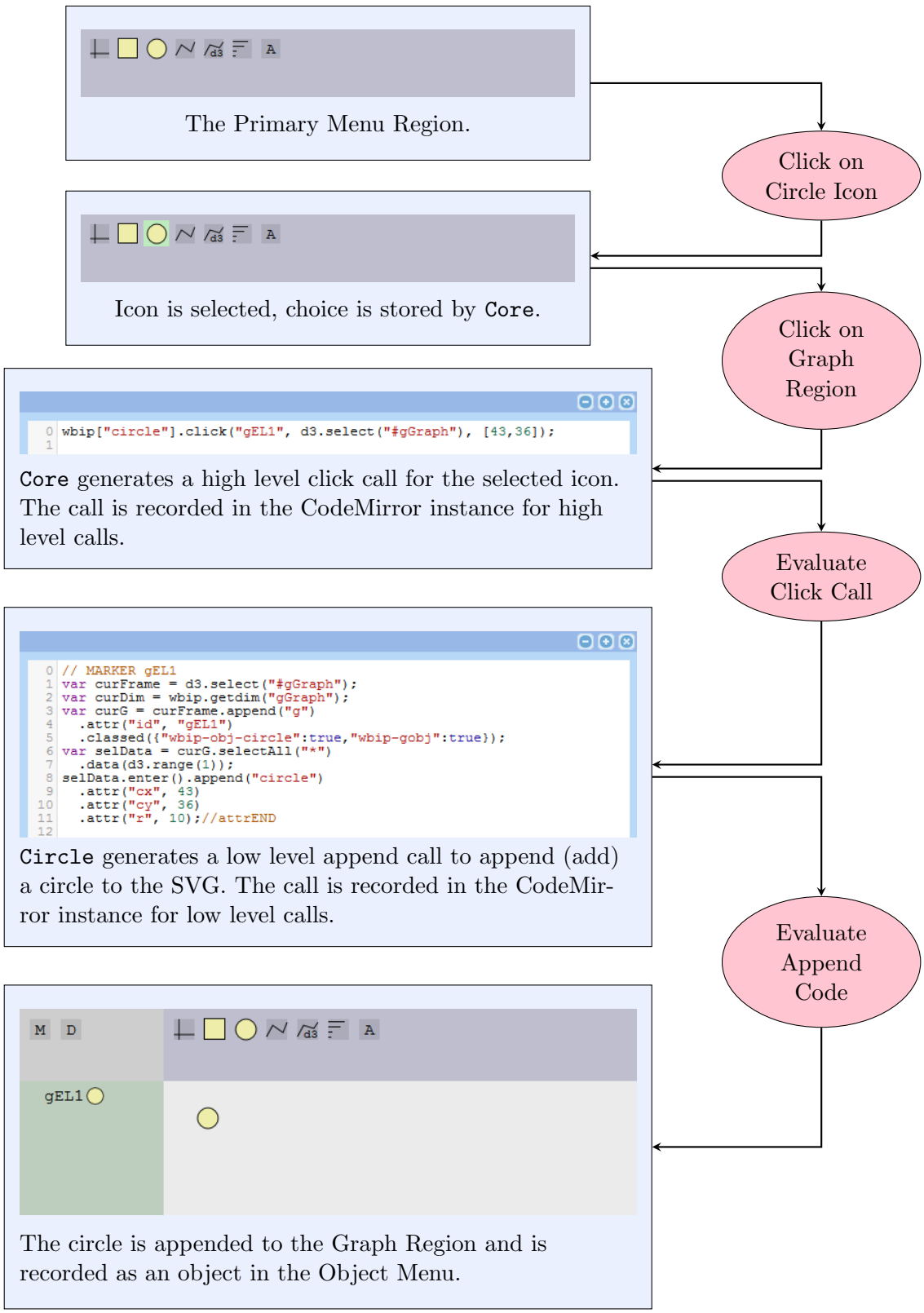


Figure 5.74: Simplified structure of how a Circle is added. Can be generalised to most other objects, with some variances in complexity.

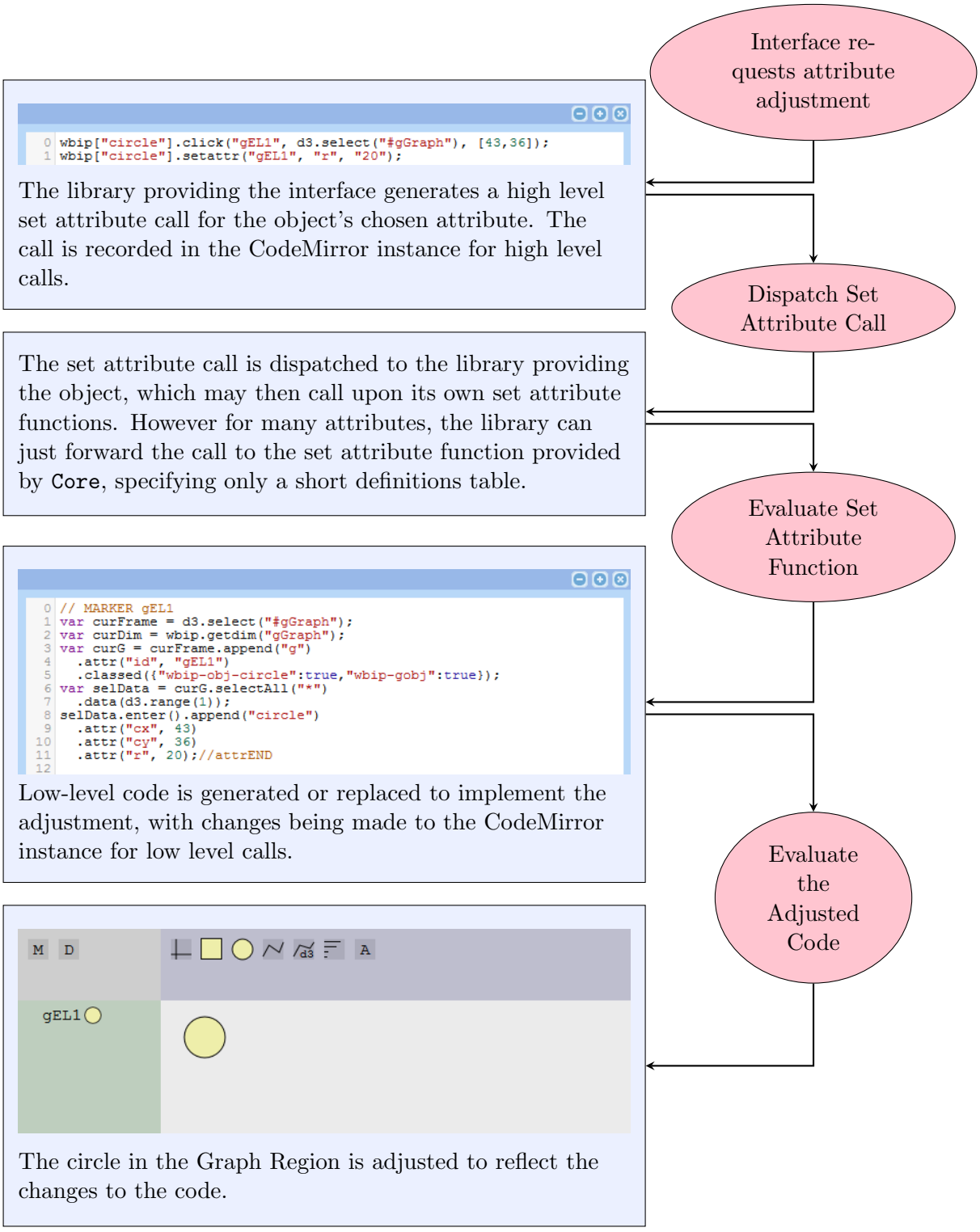


Figure 5.75: Simplified structure of how a Circle's attribute is adjusted. Can be generalised to most other objects, with some variances in complexity.

In the next sections we will discuss how to add to WeBIPP.

In keeping with the philosophy that everything is an addon the possibilities are very broad, anywhere from changing the basic layout of the GUI to adding templates for basic graphics. However it is expected that the main areas for possible user-contributed Addons to be in the following areas:

- Creating an Object Addon
- Creating a Value Interface

5.7 Creating an Object Addon

The objects that can be added via the main WeBIPP interface are provided by Object Addons. Even the most basic shapes like rectangles and circles are provided via Addons. Such objects can also be a complex collection of objects that are tied together into a single 'Object' for convenience, for instance a Bargraph could be provided as a single 'Object', with its attributes being the data to plot.

As described in Section 5.6, there is a kind of formula that dictates what an Object Addon should provide. For example the Object Addon must provide a *get attribute* function specifying how values of the attributes are recalled. `Core` does not need to understand how the Addon does this, giving the Addon creator freedom in what the Addon can do. Likewise, the Addon does not need to understand what `Core` does, as long as it provides the things dictated by the 'formula'. Refer to the Summary (Subsection 5.7.8) for a full list of the things the Object Addon must provide.

The purpose of this section is to explain what is required by the 'formula', with example code from the Circle and Rectangle Addons. As they follow the same 'formula', it becomes easy to notice what code is standardised. When creating a new Object Addon, such code can simply be reused from existing Addons. It is often not necessary to understand all of the code to create a new Addon, simply understanding what needs to be changed is enough.

5.7.1 Define a Sub-Namespace

The process for all Addons begins by creating a *sub-library* for WeBIPP by defining a namespace within the `wbip` namespace. In JavaScript, this is achieved by adding a JavaScript object as a property to the `wbip` object, e.g. for the Circle Addon, we create a 'property' of `wbip` called `circle`, that is, `wbip.circle`.

```
1 wbip.circle = function(){
2   var obj = {tags: "subobj", type: "primary"};
3
```

```
4   <Define the Icon>
5   <Define the Attributes>
6   <Define how to Set Attribute Values>
7   <Define how to Get Attribute Values>
8   <Define Interactions with the Graph Region>
9   <Any other code, as necessary>
10
11  return obj;
12 }();
13 wbip.addon("circle");
```

Note that the object to be added is created via a self-evaluating function (that is, the function is evaluated immediately). `wbip.circle` is not a function, but what is returned by the self-evaluating function, `obj`.

When `obj` is created, the properties `tags` and `type` are assigned. `tags` is currently unused and exists for potential future use. `type` classifies the Object to be added, which mainly affects where the Icon for the Object will appear in the interface.

The interior of the self-evaluating function creates a local scope that can be used to evaluate code or to provide utility functions that are not exported, but available to functions of this sub-library. In general, Addon creators are encouraged to export all functions (by assigning it as a property of the sub-library object, `obj`), but hiding things in the local scope can sometimes be useful.

In the case of Object Addons, a call to `wbip.addon` is also made at the end, to register the Addon.

5.7.2 Define the Icon

The first required function is `obj.icon` that should take two arguments:

gMenu - A d3 selection for the menu region to which the icon belongs.

transXY - The translation required for proper positioning of the icon in the menu region.

The values for these arguments are supplied by WeBIPP Core as part of the Addon registration process and depends on the `type` set for `obj`, e.g. for `type: "primary"`, the `gMenu` will be for the Primary Menu Region. The function's purpose is to:

- Store as a variable reference (`obj.gMenu`) the `gMenu`, for later use.
- Define the appearance of the icon, used in the menu and other places where such icons are useful (e.g. the Object Menu).
- Append the icon to the appropriate Menu Region (as specified by `gMenu`).

- Define what occurs on interaction with the menu icon.

```
obj.icon =
  function(gMenu, transXY){
    obj.gMenu = gMenu;
    <Define Appearance>
    <Append to Menu Region>
    <Define Interaction>
  };
```

Define Appearance

Defining the appearance begins with a call to `wbip.icon.def`, passing the `gMenu` and an appropriate string for an ID, to establish a SVG symbol. The icon is then drawn on this symbol.

```
// wbip.circle
var curS = wbip.icon.def(gMenu, "wbip-icon-circle");
curS.append("circle")
  .attr("cx", 10)
  .attr("cy", 10)
  .attr("r", 10);
```

For comparison, the code for Rectangle:

```
// wbip.rect
var curS = wbip.icon.def(gMenu, "wbip-icon-rect");
curS.append("rect")
  .attr("width", 20)
  .attr("height", 20);
```

Append to Menu Region

The icon is then added to the Menu Region using the predefined symbol, with appropriate translation and a class assigned.

```
// wbip.circle
var curG = gMenu.append("g")
  .attr("transform", wbip.utils.translate(transXY))
  .classed("wbip-icon-circle", true);
wbip.icon.use(curG, "wbip-icon-circle");
```

For comparison, the code for Rectangle:

```
// wbip.rect
var curG = gMenu.append("g")
  .attr("transform", wbip.utils.translate(transXY))
  .classed("wbip-icon-rect", true);
wbip.icon.use(curG, "wbip-icon-rect");
```

Define Interaction

Finally the interaction(s) available for the menu icon are defined, typically only `click` is used.

```
// wbip.circle
curG.on("click", function(){
  wbip.gGraph.iconclick("circle", gMenu, transXY);
});
```

For comparison, the code for Rectangle:

```
// wbip.rect
curG.on("click", function(){
  wbip.gGraph.iconclick("rect", gMenu, transXY);
});
```

In both cases, the interaction is essentially the same and understanding what this does is generally not necessary. It suffices to understand that this is sufficient for WeBIPP Core to do its thing.

But for those unsatisfied with such an explanation, `wbip.gGraph.iconclick` results in registration of the icon's selection, triggering an interface change (the icon appears 'highlighted') and means any subsequent interaction with the Graph Region with the icon selected will trigger the processes specified in Subsection 5.7.6. See also Figure 5.74.

5.7.3 Define the Attributes

After the icon is defined, the attributes for the object must be defined. The JavaScript object `obj.def` acts as a named array, with each attribute definition stored as a sub-array with the following syntax:

```
[Type of Attribute, Category of Attribute, (optionally) Sub-attributes]
```

Type of Attribute - `attr` or `style` for direct mapping to SVG object attributes, `useScale` variants for WeBIPP supported scaling, or `special` for completely arbitrary attributes.

Category of Attribute - Defines what interface is used for assigning these attributes, matching an appropriate interface will make things easier for the end-user. If no interface exists for the category provided, WeBIPP will use a default fail-safe. Some possible categories are `logical`, `numeric`, `proportion`, `rgb`.

Sub-Attributes - Some attributes have sub-attributes, which are defined as the optional third item of the array. These have the same format as the top-most attribute definition, that is they are JavaScript objects containing named arrays that define the sub-attributes. There is no strict nesting cap, though most interfaces will only handle a single nesting depth.

```
// wbip.circle
obj.defs = {
  cx: ["attr", "numeric", {
    useScale: ["useScale-x", "logical"]
  }],
  cy: ["attr", "numeric", {
    useScale: ["useScale-y", "logical"]
  }],
  r: ["attr", "numeric"],
  stroke: ["style", "rgb"],
  fill: ["style", "rgb"],
  opacity: ["style", "proportion"]
};
```

For comparison, the code for Rectangle:

```
// wbip.rect
obj.defs = {
  x: ["attr", "numeric", {
    adj: ["special", "proportion"],
    useScale: ["useScale-x", "logical"]
  }],
  y: ["attr", "numeric", {
    adj: ["special", "proportion"],
    useScale: ["useScale-y", "logical"]
  }],
  width: ["attr", "numeric", {
    useScale: ["useScale-x-dist", "logical"]
  }],
```

```
14   height: ["attr", "numeric", {
15     useScale: ["useScale-y-dist", "logical"]
16   }],
17   stroke: ["style", "rgb"],
18   fill: ["style", "rgb"],
19   opacity: ["style", "proportion"]
20 };
```

5.7.4 Define how to Set Attribute Values

With the attributes defined, the Addon must now specify how to set values to these attributes with the `obj.setattr` function. This function should take three arguments:

curID - The ID of the object that is having an attribute set.

name - The name of the attribute.

val - The value to set the attribute.

In general, the values for these arguments will be generated as part of the end-users' interaction with the WeBIPP interface.

If there are no attributes of type `special`, this is very easy as the Addon can simply forward the call to the `setattr` function provided by WeBIPP Core. Attributes of type `special` are used to create any attribute that does not fit into one of the standard types that WeBIPP Core recognises.

In the Circle example below, there are no `special` attributes and thus the function is a simple call to `wbip.setattr` passing the three arguments, and additionally the attribute definitions (processed through `wbip.getrealdef`), which WeBIPP Core needs to understand what kind of attributes it is dealing with.

```
1 // wbip.circle
2 obj.setattr =
3   function(curID, name, val){
4     wbip.setattr(curID, wbip.getrealdef(obj.defs, name), name
          , val);
5   };
```

For comparison, the code for Rectangle must handle a `special` attribute in `x-adj` and `y-adj`. These are handled via a custom function (`setxyadj`), and a `switch` is used to divert to either the custom function, or to forward the call to WeBIPP Core.

```
1 // wbip.rect
```

```
obj.setattr =
  function(curID, name, val){
    var setxyadj = function(){
      <Code excised for space>
    };
    switch(name){
      case "x-adj":
      case "y-adj":
       setxyadj();
       break;
      default:
       wbip.setattr(curID, wbip.getrealdef(obj.defs, name),
          name, val);
    }
  };
```

5.7.5 Define how to Get Attribute Values

Similarly, the Addon must also specify how to get current attribute values with the `obj.getattr` function. This function should take two arguments:

curID - The ID of the object to get attributes values from.

name - The name of the attribute.

In general, the values for these arguments will be generated as part of the end-users' interaction with the WeBIPP interface.

Very often this call is simply forwarded to WeBIPP Core, even for attributes of type `special`, but custom functions could be implemented as required.

```
obj.getattr = wbip.getattr;
```

Both Circle and Rectangle simply forward the call. Or more technically, `obj.getattr` is made to refer to `wbip.getattr`, so that WeBIPP Core is called directly with no intermediary function.

5.7.6 Define Interactions with the Graph Region

Once an icon is selected and registered via WeBIPP Core (see Subsection 5.7.2), subsequent interactions with the Graph Region will result in WeBIPP checking to see if the selected Addon has defined a function for that interaction, e.g. if the user clicks on the Graph Region,

WeBIPP Core will check if there is a `obj.click` available. If it exists, it is called. The current implementation only supports click events, but support is easily extended to other interactions, including but not limited to mouse drag events and keyboard presses.

`obj.click` should accept three arguments:

curID - The ID to be assigned to the newly created object.

curFrame - A d3 selection for the frame the object is being nested under (see Subsection 5.6.1).

Coords - The coordinates where the event occurred, given relative to `curFrame` as an array `[x, y]`.

While any of these functions could be called on their own, in general they are called with arguments generated as part of the end-users' interaction with the WeBIPP interface.

The purpose of the function is to write the necessary low-level code (`outcode`) to create the object. Much like `obj.icon`, it is not necessary to completely understand all the code, sufficing to know only what needs to be adjusted for any new Object Addon.

```
obj.click =
  function(curID, curFrame, Coords){
    <Initialise outcode>
    <Set Starting Direct Attributes and Append>
    <Set Additional Starting Attributes and Save IDvar>
    <Any other code, as necessary>
    <Write and Evaluate outcode>
    <Add new Object to Object Menu listing>
  };
```

Initialise outcode

Writing the low-level code begins with a call to `wbip.outcode.init`, passing `curID`, `curFrame` and any classes, to initialise the outcode.

```
// wbip.circle
var outcode = wbip.outcode.init(curID, curFrame,
  {"wbip-obj-circle": true, "wbip-gobj": true});
```

For comparison, the code for Rectangle:

```
// wbip.rect
var outcode = wbip.outcode.init(curID, curFrame,
  {"wbip-obj-rect": true, "wbip-gobj": true});
```

Set Starting Direct Attributes and Append

Direct Attributes are those attributes that directly map to SVG object attributes, that is they are attributes of type `attr` as defined in `obj.defs`. The default values for Direct Attributes are set here, while the default values for any Styles (such as `fill`) are set in the `css` file corresponding to the Addon (see Subsection 5.7.9).

The starting attributes and their values are stored as a JavaScript Object (`curAttrs`) before being passed to `wbip.outcode.append` which will generate the appropriate low-level code.

```
1 // wbip.circle
2 var curAttrs = {cx: Coords[0], cy: Coords[1], r: 10};
3 outcode += wbip.outcode.append(1, "circle", curAttrs);
```

For comparison, the code for Rectangle:

```
1 // wbip.rect
2 var curAttrs = {x: Coords[0], y: Coords[1], width: 20, height
    : 20};
3 outcode += wbip.outcode.append(1, "rect", curAttrs);
```

Set Additional Starting Attributes and Save IDvar

Any other attributes (including styles) that require starting values set, but do not map directly to SVG object attributes, are added to the JavaScript Object containing the attributes (`curAttrs`) and then saved as an IDvar (see Subsection 5.6.2).

This code on its own does not result in any direct changes to the actual SVG object being created, it merely stores information for use later. Such future use may be to simply be a default value that appears when the user queries the attribute via an interface, but could also be made to interact with other code in the Addon in complex ways, e.g. the default value of a `useScale` variant attribute will alter how WeBIPP Core's `setattr` function handles its job (see Section 5.9).

```
1 // wbip.circle
2 curAttrs["cx-useScale"] = "auto";
3 curAttrs["cy-useScale"] = "auto";
4 var curVars = {attr: curAttrs, name: "circle"};
5 wbip.setIDvar(curID, curVars);
```

For comparison, the code for Rectangle:

```
1 // wbip.rect
2 curAttrs["x-adj"] = 0.5;
```

```
curAttrs["y-adj"] = 0.5;
curAttrs["x-useScale"] = "auto";
curAttrs["y-useScale"] = "auto";
curAttrs["width-useScale"] = "auto";
curAttrs["height-useScale"] = "auto";
var curVars = {attr: curAttrs, name: "rect"};
wbip.setIDvar(curID, curVars);
```

Any other code, as necessary

Any additional code necessary for the object can be appended here. If nothing else a line-break needs to be added.

```
// wbip.circle
outcode += "\n";
```

For comparison, the code for Rectangle:

```
// wbip.rect
outcode += 'wbip.rect.adjXY(selData, 0.5, 0.5);\n\n';
```

Note that Circle requires no other code, while Rectangle has code that calls another function defined in the Addon.

Write and Evaluate outcode

With the necessary low-level code generated, it must now be written to the CodeMirror instance and then evaluated to actuate the changes to the SVG. This code should be exactly the same for all Object Addons.

```
wbip.outcode.write(outcode, "low");
eval(outcode);
```

Add new Object to Object Menu listing

All newly created objects should be added to the Object Menu listing for easy reference and selection by the end-user. The required code is simple, yet listing on the Object Menu provides many major benefits for user interactivity.

```
// wbip.circle
wbip.slidemenu.append(d3.select("g.wbip-sm-" + curFrame.attr(
    "id")), [{text: curID, icon: "wbip-icon-circle"}]);
```

For comparison, the code for Rectangle:

```
1 // wbip.rect
2 wbip.slidemenu.append(d3.select("g.wbip-sm-" + curFrame.attr(
    "id")), [{text: curID, icon: "wbip-icon-rect"}]);
```

5.7.7 Any other code, as necessary

Additional code may be necessary for proper functioning of the object. The Circle does not require any extra code, but the Rectangle does, defining `obj.adjXY` to handle adjustment of `x` and `y`. Note that this is the function that gets called via `wbip.rect.adjXY` at the end of `outcode` above.

Such additional code may also be to create a Value Interface (see Section 5.8).

5.7.8 Summary

To summarise, the following exported properties exist:

obj.tags - Currently unused but defined for all base addons.

obj.type - For Addons that are registered via `wbip.addon`, the type determines what WeBIPP does with it. Primary, Secondary and Tertiary will add the icon to the corresponding menu regions. Special Addons do not have an icon, and instead of `obj.icon` have `obj.new`, which is evaluated as soon as the Addon is loaded.

obj.icon - A function that defines the icon's appearance, as well as creating the icon in the appropriate Menu Region.

obj.gMenu - A d3 selection for the Menu Region to which the Object's menu icon belongs.

obj.defs - A JavaScript Object acting as a named array, containing the definitions of all attributes of the object being created.

obj.setattr - A function specifying how values to the attributes are set.

obj.getattr - A function specifying how values of the attributes are recalled.

obj.click - A function to be run on a click interaction with the Graph Region while the icon is selected. Generally this adds a new instance of the object to the Graph Region.

And of course, any additional functions, exported or otherwise, that are defined for the Addon.

5.7.9 Stylesheets

In addition to the JavaScript code that defines the Object Addon, default style values for the Object should be set via an associated stylesheet (`css`). Examples of these are:

```
#wbip-icon-circle circle{
  stroke: black;
  fill: #EEEEAA;
}

g.wbip-obj-circle circle{
  stroke: black;
  fill: #EEEEAA;
}
```

```
#wbip-icon-rect rect{
  stroke: black;
  fill: #EEEEAA;
}

g.wbip-obj-rect rect{
  stroke: black;
  fill: #EEEEAA;
}
```

In both cases, the stylesheet defines a black border (`stroke`) and a yellow-ish interior (`fill`) for both the icon and any Objects created via the Addon.

5.7.10 Creating Complex Objects

These objects can be quite complex, as an example the Cartesian Frame object will be explored in brief detail. The same principles could be used to create an 'Object' that is a complete statistical plot. This is more an example of what can be done, but is likely to be too complicated for most addon makers to bother with.

Cartesian Frame

The Cartesian Frame is comprised of:

- A background rectangle.
- Up to 4 axes, which can be altered.
- The necessary attributes for a frame (see Subsection 5.6.1).

All attributes of the Cartesian Frame are of type `special`, meaning they must be handled by the Object Addon itself.

```
obj.defs = {
  x: ["special", "scale", {
    name: ["special", "string"],
    domain: ["special", "domain"],
    range: ["special", "range"]
  }],
  y: ["special", "scale", {
    name: ["special", "string"],
    domain: ["special", "domain"],
    range: ["special", "range"]
  }],
  axes: ["special", "axes", {
    opts: ["special", "axes-opts"]
  }],
  dim: ["special", "dim"],
  transform: ["special", "transform"]
};
```

The attributes `x` and `y` are particularly special, as they themselves do nothing. Their children however are very important in defining the arbitrary coordinate space the Frame defines. These scales are implemented by use of D3's linear and ordinal scales.

```
var updateScale =
  function(ScaleName){
    // If scale is x, range maps to width (dim[0])
    // else it's y, and range maps to height (dim[1])
    if(ScaleName === "x"){
      var rangemap = curVars.attr.dim[0];
    } else{
      var rangemap = curVars.attr.dim[1];
    }
    var valType = typeof curVars.attr[ScaleName + "-domain
       "][0];
    switch(valType){
      case "number":
        curVars.attr[ScaleName] = d3.scale.linear()
          .domain(curVars.attr[ScaleName + "-domain"])
          .range(curVars.attr[ScaleName + "-range"]
            .map(function(x) {return x * rangemap;}));
```

```
        break;
      case "string":
        curVars.attr[ScaleName] = d3.scale.ordinal()
          .domain(curVars.attr[ScaleName + "-domain"])
          .rangePoints(curVars.attr[ScaleName + "-range"]
            .map(function(x) {return x * rangemap;}), 1);
        break;
    }

    wbip.cm.loweval();
  };
```

`updateScale` handles the hard bits of changing the Scales, so the `setattr` code for these children attributes are simple:

```
case "x-name":
  curVars.attr[name] = val;
  updateScale("x");
  break;
case "y-name":
  curVars.attr[name] = val;
  updateScale("y");
  break;
case "x-domain":
case "x-range":
  val = eval(val);
  curVars.attr[name] = val;
  updateScale("x");
  break;
case "y-domain":
case "y-range":
  val = eval(val);
  curVars.attr[name] = val;
  updateScale("y");
  break;
```

To make it easier to set these children meaningful values, `x` and `y` can take Data inputs, which are used to populate their children.

```
case "x":
case "y":
```

```
3   // Only accepts wbip.data inputs
4   if(val.substr(0, 9) === "wbip.data"){
5     <Ugly Regular Expression that sets a smart name to -name>
6
7     val = eval(val);
8     var valType = typeof val[0];
9     switch(valType){
10      case "number":
11        curVars.attr[name + "-domain"] = d3.extent(val);
12        break;
13      case "string":
14        curVars.attr[name + "-domain"] = val;
15        break;
16    }
17    updateScale(name);
18  }
19  break;
```

Changing the `axes` requires some changes to low-level code, but here again another function is used to make life easier.

```
1 case "axes":
2   val = eval(val);
3   if(!wbip.utils.isArray(val)){val = [val];}
4   curVars.attr[name] = val;
5   var fromto = wbip.cm.findbyid(wbip.cm.low, curID);
6   var searchstr = 'wbip.frcart.axes(curG, ';
7   var newval = searchstr + JSON.stringify(val) + ");";
8   wbip.cm.replaceline(wbip.cm.low, fromto, searchstr, newval)
      ;
9   wbip.cm.loweval();
10  break;
```

The effect of this is simply to change this one line:

```
1 wbip.frcart.axes(curG, [1, 2]);
```

This line defines which of the 4 axes the Frame should display. It calls the function `wbip.frcart.axes`, which is too complex to go into detail here.

Summary on Complex Objects

By use of special attributes it is possible to tie together several different objects and make use of all sorts of code to create objects that are complex, yet easy to use. However creating these does require a significant understanding of the various libraries WeBIPP is built on, much more so than when creating a simple Object Addon.

5.8 Creating a Value Interface

When the user desires to change the value of an attribute for an Object, it often helps to have a nice interface to do so. These are provided via Value Interfaces. Which interface is called depends on the category assigned to the attributes in its definition (see Subsection 5.7.3), for instance if an attribute is of category `numeric`, WeBIPP will check to see if `wbip.valintf.numeric` is defined, if so it will call this interface. Otherwise it will fall back on the default interface, `wbip.valintf.none`.

It is possible to overwrite the Value Interfaces provided by WeBIPP, or to add new Value Interfaces, perhaps as part of a new Object Addon.

The process for creating a new Value Interface for WeBIPP is explained below. As the code can vary greatly, two example Value Interfaces are provided.

For more examples, see `Libs_Internal/WeBIPP_ValInterface.js`.

5.8.1 Assign the Value Interface

The process begins by assigning the value interface to the correct place. To create a Value Interface for category `numeric`, we would need to assign a function to `wbip.valintf.numeric`. If the code doing so is loaded as an Addon after the base Value Interfaces provided for WeBIPP are already loaded, then the new interface will overwrite any existing interface.

All Value Interface functions can take up to five arguments, of which only `returnfunc` is mandatory. The rest merely carry additional information to provide a more useful interface.

returnfunc - The function to call on the final return value. That is, once the user has chosen a new value via the interface, that value should be passed as a string to `returnfunc`.

initval - The initial value of the attribute being modified.

aiID - The ID of the Attribute Interface that is calling the Value Interface.

tarID - The ID of the Target Object being modified.

curAttr - The name of the attribute being modified.

For example, the `numeric` interface (which only makes use of the first three arguments):

```
wbip.valintf.numeric =
  function(returnfunc, initval, aiID){
    <Various Code>
  };
```

5.8.2 The Logical Interface

The simplest of the Value Interfaces provided with WeBIPP is for `logical` attributes. Despite being an 'interface', all it does is flip the value between `true` and `false`, the only two valid `logical` values.

```
function(returnfunc, initval){
  if(initval === true){initval = "true";}
  returnfunc(String(initval !== "true"));
};
```

If the `initval` given is `true` as a JavaScript logical (and not a string), it is first converted to a string. Then a logical check is made to flip between `true` and `false`, with the result being passed to `returnfunc` as a string. Because of the initial coercion, this means an initial value of either `true` or `"true"` will become `"false"`, while any other initial value will become `"true"`.

5.8.3 The Numeric Interface

The Value Interface for `numeric` attributes opens a Free Form Window that looks a bit like a calculator. This can then be used to input the number desired, and also to perform arithmetic operations on the current value.

The Numeric Interface makes use of one of the interface templates provided, the Button Interface. This provides an editable text area (using CodeMirror for syntax highlighting), an indicator for whether the expression in the text area is valid, along with any number of buttons as defined. The code required for the Numeric Interface is simply to specify the right arguments to the Button Interface function.

```
function(returnfunc, initval, aiID){
  <Define Cancel Func>
  <Define Eval Func>
  <Define Other Arguments>
  <Call Button Interface>
};
```

Define Cancel Func

If while messing with the interface the user decides to cancel out, WeBIPP must return everything to the way it was. The `cancelfunc` is what does this.

```
var cancelfunc =
  function(){
    wbip.outcode.freeze = true;
    returnfunc(initval);
    wbip.outcode.freeze = false;
  };
```

`wbip.outcode.freeze` can be used to freeze writing to the CodeMirror instances, allowing evaluation without permanent writing of code. We then simply send `initval` back to `returnfunc`, which restores the original value to the object.

Without freezing, the call to `returnfunc` would write a new `setattr` line to the CodeMirror instances, adding undesirable clutter.

Define Eval Func

The `initval` provided may not simply be a number. It could be an array of numbers (e.g. because it corresponds to data values) or even some kind of JavaScript expression (stored as a string prefixed with `expr:`, e.g. `expr:Math.pow(5, 2)`). The `evalfunc` correctly handles the `OV` when evaluating the expression created via the interface (notice similarities to discerning valtype in `wbip.setattr`, see Section 5.9)

```
var evalfunc =
  // Correctly handle the various cases of original value
  function(expr){
    // Only need to do complicated stuff if OV is used
    if(expr.match("OV") !== null){
      // if initval is an expr
      if(initval.substr(0, 5) === "expr:"){
        return "expr:" + expr.replace("OV", initval.substr(5)
            , "g");
      } else{
        // else try evaluating initval
        var valreal = eval(initval);
        // if valreal is just a number,
        // can return new number
        if(typeof valreal === "number"){
```

```
15            return eval(expr.replace("OV", initval, "g")).
                 toString();
16          }
17          // if valreal is an array, need to
18          // prefix "expr:"
19          // append "[d]" to valreal
20          if(wbip.utils.isArray(valreal)){
21            return "expr:" + expr.replace("OV", initval + "[d]"
                 , "g");
22          }
23        }
24      } else{
25        // Direct eval
26        return eval(expr).toString();
27      }
28    };
```

Understanding this code requires more detail than this chapter covers, suffice to say were it not for the ability to manipulate the initial value (original value, `OV`), `evalfunc` would simply be `eval(expr).toString()`, which would for instance convert `"1 + 2"` to `"3"`. However a more complicated function can be used to parse the expression generated via the interface, to accomplish more powerful end results.

Define Other Arguments

Now it is simply a matter of defining some arguments, most important being the layout of the buttons, and what the buttons contain.

layout - of the form `[ncols, nrows]`.

btndef - a JavaScript 'matrix', which is really a nested array. The contents can be `undefined` (no button drawn), a string (the button label), or a JavaScript object which can specify the `label`, a `tooltip` (which displays when the user hovers their mouse over the button) and `width` or `height` of the button (which are in relative terms, thus `{width: 2}` indicates a button roughly twice as wide as a normal button).

See how the arguments set below correspond to the resulting interface, Figure 5.76.

```
1 var curID = aiID + "-valintf";
2 var layout = [7, 3]; // 3x7 matrix
3 var btndef =
```

```
4 [[7, 8, 9, "+", "-", "(", ")"],
5  [4, 5, 6, "*", "/", {label: "OV", tooltip: "Represents the
     original value", width: 2}, undefined],
6  [1, 2, 3, 0, ".", {label: "SET", width: 2}, undefined]];
```

Figure 5.76: Two examples of the Numeric Value Interface. Left: Assigning a specific value of 150. Right: Modifying the `OV` (Original Value), the mouse is over the `OV` button, highlighting it and displaying the tooltip (if defined).

Call Button Interface

```
1 wbip.btnintf.new(wbip.vars.svg, curID, returnfunc,
2                  cancelfunc, evalfunc, layout, btndef);
```

Additionally, `wbip.btnintf.new` can accept three more (optional) arguments:

btndim - The dimension of the buttons (in pixels), default `[20, 20]`.

padding - The padding between the buttons (in pixels), default `5`.

transxy - The starting position of the Button Interface, default `[360, 160]`. The starting position is not that important as the interface is a Free Form Window and the user is free to move it to a more convenient place.

5.9 Core's Set Attribute

As mentioned, WeBIPP Core provides a default `setattr` function that Object Addons can rely on for convenience. This function is complex, handling multiple types of attributes and intelligently adjusting low-level code to implement the new attribute value. It is not necessary to understand how the function accomplishes this in order to make use of it, thus the code will not be examined in depth and only a brief overview is given.

```
wbip.setattr =
  function(curID, attrdef, name, val){
    // Below are the major processes in the function
    // It does not cover every process
    <Handle useScale type>
    <Process value and discern valtype>
    <Handle auto useScale>
    <Apply Scale if needed>
    <Adjust low-level code>
  };
```

5.9.1 Handle useScale type

When an Object Addon is created, it may have one or more attributes of type `useScale` (e.g. `x-useScale`), which determine if its parent attribute (e.g. `x`) makes use of a corresponding Scale provided by its parent frame (see Subsection 5.7.3). Valid values for `useScale` are `true`, `false` and `auto` (see Subsection 5.9.3 on how `auto` is handled).

When a `useScale` attribute is changed, this impacts how its parent attribute should be handled. In essence, a `setattr` for a `useScale` must call a subsequent `setattr` on its parent. For efficiency, a double-call to `setattr` is not made. Instead, it is handled by saving the `useScale` value, then `setattr` pretends it is setting the value of the parent attribute (by changing the values of the arguments). This results in the relevant parts of the low-level code being updated to ensure the parent attribute is using (or not using) the Scale without a double call to `setattr`.

```
if(attrdef[0].split("-")[0] === "useScale"){
  val = val === "true";
  curVars.attr[name] = val;
  // Pretend
  name = name.split("-")[0];
  attrdef = wbip[curVars.name].defs[name];
  val = curVars.attr[name].toString();
```

```
8 }
```

5.9.2 Process value and discern valtype

Different from the type of attribute we have, is the type of value we have. The type of value we have is called `valtype` and currently there are four types:

value - A single value, e.g. `5` or `"black"`

vector - An array of some sort, e.g. `[1, 2, 3]`

expr - A WeBIPP expression, e.g. `"expr:[1, 2, 3][d] * 10"` - WeBIPP expressions simply tell WeBIPP to take the value literally with no extra handling. This can give finer control over what low-level code is written.

(default) - Representing the null or undefined value, being the value of the (unknown)[10] default

The following code determines the `valtype` of the value provided, and may additionally tidy up the value itself (e.g. `"5"` would become `5`). This process allows `wbip.setattr` to handle different types of values relatively painlessly.

```
1 var valtype = "value";
2 if(val === "(default)"){
3   valtype = "(default)"
4 } else{
5   // if val starts with the "expr:" keyphrase
6   // set type to "expr" and adjust val to remove keyphrase
7   if(val.substr(0, 5) === "expr:"){
8     valtype = "expr";
9     val = val.substr(5);
10  } else{
11    // else try evaluating val
12    try{
13      var valreal = eval(val);
14      // if valreal is just a number,
15      // set val to that number (for cleanliness)
16      if(typeof valreal === "number"){
17        val = valreal;
18      }
```

[10] The default is unknown to WeBIPP Core but is known by the Object Addon itself.

```
19        // if valreal is an array, set type to be "vector"
20        if(wbip.utils.isArray(valreal)){
21          valtype = "vector";
22        }
23      } catch(e){
24        // assumes val is a normal string (e.g. "black")
25        val = JSON.stringify(val);
26      }
27    }
28 }
```

5.9.3 Handle auto useScale

```
1 if((valtype === "vector" || valtype === "expr") &&
2     curVars.attr[name + "-useScale"] === "auto"){
3   curVars.attr[name + "-useScale"] = true;
4 }
```

Attributes of type `useScale` can have the value `auto`, in addition to the simple logical values (`true` and `false`). The value of `auto`, and how it is handled, is a concession to convenience. To put it simply, `auto` attempts to guess what the user would want it to do. The explanation of how it works is thus: When an Object is first placed, its attributes are generally assigned a static number (e.g. `x = 10`). This static number (and any direct adjustments made to it) has meaning in terms of pixels and the SVG coordinate system, and does not have a meaningful correlation to any data-based Scale the frame has. Thus when `useScale` is set to `auto` and it is dealing with these static numbers, it treats itself to be `false` (but remains of value `auto`) and does not make use of the frame's Scale. However, when data is assigned to the Object, it is often in the context of some kind of data-based Scale, and not in terms of pixels. Thus when data is assigned to an attribute (e.g. `x`) which has a child `useScale` (e.g. `x-useScale`) of value `auto`, the value of this `useScale` will automatically be set to `true`. To summarise: a value of `auto` is the same as `false`, except that it can silently switch to `true` if data is assigned to its parent attribute. This behaviour may seem arbitrary, but is often very convenient in actual use of WeBIPP.

5.9.4 Apply Scale if needed

We check if the attribute has a `useScale` child attribute, and if this value is `true` the Scale must be applied. This is done by creating the function `scaleval` which adds the necessary code to the value to apply the Scale appropriately. If no Scale needs to be applied, `scaleval` simply returns the value unaltered.

The Scale can be applied in two different ways: directly or as a distance. Directly is a direct application of the scale, but for attributes such as `width`, a value of say `10` really represents the scaled distance from `0` to `10`, and not the position of `10` on a scale that may not start from `0`.

```
if(curVars.attr[name + "-useScale"] === true){
  var scaledefs = attrdef[2]["useScale"][0].split("-");
  var scalename = scaledefs[1];
  if(scaledefs[2] === "dist"){
    var scaleval = function(val){
      return "wbip.distScale(wbip.getScale(curFrame, " +
        JSON.stringify(scalename) + "), " + val + ")";
    };
  } else{
    var scaleval = function(val){
      return "wbip.getScale(curFrame, " + JSON.stringify(
         scalename) + ")(" + val + ")";
    };
  }
} else{
  var scaleval = function(val){return val;};
}
```

5.9.5 Adjust low-level code

This section is exceptionally complex and makes significant use of the CodeMirror API which is well beyond the scope of this chapter. What it does will instead be explained by way of example. Suppose the user has created a circle. The following low-level code would be generated.

```
// MARKER gEL2
var curFrame = d3.select("#gEL1");
var curDim = wbip.getdim("gEL1");
var curG = curFrame.append("g")
  .attr("id", "gEL2")
  .classed({"wbip-obj-circle":true,"wbip-gobj":true});
var selData = curG.selectAll("*")
  .data(d3.range(1));
selData.enter().append("circle")
```

```
10    .attr("cx", 91)
11    .attr("cy", 206)
12    .attr("r", 10);//attrEND
```

The user then assigns the data `speed` (from the dataset `datCars`) to `cx`. As this is not a `special` attribute, Circle's `setattr` forwards this call to `wbip.setattr` which intelligently finds the right bits of low-level code and adjusts them thus (changes are noted below):

```
1  // MARKER gEL2
2  var curFrame = d3.select("#gEL1");
3  var curDim = wbip.getdim("gEL1");
4  var curG = curFrame.append("g")
5    .attr("id", "gEL2")
6    .classed({"wbip-obj-circle":true,"wbip-gobj":true});
7  var selData = curG.selectAll("*")
8    .data(d3.range(50));
9  selData.enter().append("circle")
10   .attr("cx", function(d){return wbip.getScale(curFrame, "x")
       (wbip.data["datCars"]["speed"][d]);})
11   .attr("cy", 206)
12   .attr("r", 10);//attrEND
```

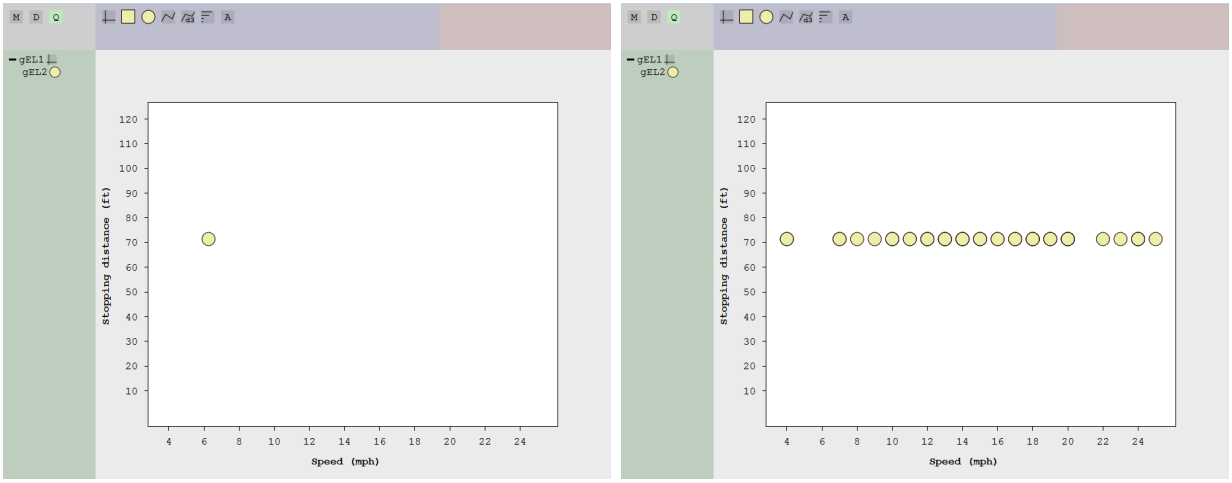


Figure 5.77: How the Circle looks before (left) and after (right). Note that for the right graph, many of the `cx` values are the same, resulting in overlapping circles and the appearance of a smaller number of circles than the 50 that actually exist.

The first change is to the `.data` line, which will mean there are now 50 Circles instead of 1.

```
1  // Before
2    .data(d3.range(1));
```

```
1 // After
2   .data(d3.range(50));
```

The second change is to the `cx` line, changing the static value to a function that will apply the 50 values of `speed` to the 50 different Circles.

Implicit in this code is that the default value of `cx-useScale` was `auto`. When a data vector like `speed` is assigned to `cx`, `cx-useScale` will have been set to `true` automatically behind the scenes (see Subsection 5.9.3). Which is why the After code uses `wbip.getScale` to apply the Scale.

```
1 // Before
2   .attr("cx", 116)
```

```
1 // After
2   .attr("cx", function(d){return wbip.getScale(curFrame, "x")
      (wbip.data["datCars"]["speed"][d]);})
```

With the low-level code adjusted, it is evaluated, updating the SVG image to the new values, and completing the setting of the attribute value.

5.10 Discussion and Limitations

The vision for WeBIPP was to create a tool that is both easy-to-use (like the GUI tools) and powerful (like the Code tools). To realise this vision, a number of objectives were set, with the key ones being summarised thus:

- Possess an interactive GUI that does not require coding knowledge to use, and hence is as easy-to-use as existing GUI tools.
- Capable of creating a diverse range of graphics, not just a limited predefined set.
- Easily extended with code, so that those with coding knowledge are not restricted by the interface, and hence is as powerful as Code tools.

The ideal was to have a GUI that makes accessible the basic tasks that can be done with Code, enabling non-coders to utilise the basic building blocks of graphics, while simultaneously not limiting coders with the interface. Other GUI tools tend to restrict the user to a limited set of predefined graphics, and those that do give access to more basic building blocks still restrict the user to only what the interface is capable of handling.

WeBIPP is different. Though it gives access to the basic building blocks through a GUI (see Section 5.3), this is really a front-end to writing code (see Section 5.6), hence anyone who desires to do more than what the GUI enables, can do so by directly writing extra code. This is a good thing, as WeBIPP still has quite a primitive GUI. Compared to other GUI tools, ones that specialise in making it easy to create one of the standard statistical plots, creating one through WeBIPP (e.g. see Section 5.4) requires quite a bit more steps using the basic building blocks currently provided, and in some cases it may be necessary to not rely on the WeBIPP GUI at all (e.g. see Subsection 5.5.3). However, the addon capabilities of WeBIPP (see Section 5.7) mean it should be possible to create Addons that provide the standard statistical plots, making it as easy to create such plots with WeBIPP as it is with other GUI tools, while *still* giving access to the basic building blocks and the capability to write and re-use code. This could be achieved in a similar way to the Cartesian Frame Addon (see Section 5.7.10) which consists of multiple sub-elements that are unified into a 'single' WeBIPP object, or by making use of the Functionise feature (see Subsection 5.5.9), by exporting the Functionised code as an Addon. A better Functionise interface is necessary to make the latter possible (and easy), and this is one of the eatures planned for the future.

Though the GUI is still fairly primitive, and though the Addons currently available are limited, the current version of WeBIPP works as a proof-of-concept, demonstrating that the objectives set out are feasible, and that it is possible to create a tool that is both easy-to-use and powerful, that it is possible to create a tool that is not so limited in scope, as the other tools are. Not only is it an enabling tool that enables non-coders to do things they could

not before. It is also an efficiency tool, leaving users who are happy with code free to switch between using helpful interface elements (where this is easier, faster or more convenient) and writing code directly (giving access to the full potential of code tools). However, in setting out to realise this vision, a number of choices had to be made, choices that brought with them limitations.

WeBIPP is web-based, making it much more accessible to a broader audience. The user does not have to install anything, they just need a modern browser. But being web-based carries with it limitations, as web-based scripts have necessary restrictions in place, e.g. to limit the effects of malicious websites. Another of WeBIPP's philosophies is that everything is an addon, but unlike an installed program, such as R, that (once installed) can easily install packages to extend functionality, it is difficult for a JavaScript tool, that is restricted in what it can do, to download and install addons on-the-fly. Loading addons typically requires rewriting the HTML code that launches WeBIPP, a task that is most cumbersome. Other JavaScript libraries that also have many addon capabilities tend to tackle this task by having a server that automatically compiles all the needed addons into a single JavaScript file, and something like this may be necessary for WeBIPP in the future, to make it easier to mix-and-match different addons.

Another choice was how to draw the graphic. The two main competitors for modern web-based graphics are SVG and HTML5 Canvas. The SVG format has many advantages, it is an open standard, has an object system that enables object-based interactions, uses vector-based drawing and is generally very powerful in what it can do. For these reasons it was chosen over Canvas, but Canvas has one major advantage over SVG: performance. Being raster-based it is much more conducive to hardware acceleration meaning Canvas can perform significantly better than SVG, particularly when dealing with a large number of objects being drawn (as Canvas has no native concept of objects and does not have to keep track of them and to re-render them). This limitation can be noticeable when trying to use WeBIPP with a large dataset. As this is most problematic when dealing with large datasets, one solution is to prototype first using a small subset of the data, and then redrawing a limited number of times with the full dataset, something that is reasonably simple to do with WeBIPP.

This performance problem is worsened due to how WeBIPP works. It essentially writes high-level code, that writes low-level code, that writes the XML document (the SVG), which is finally rendered by the browser. This layered process gives WeBIPP much of its power and flexibility, but comes at a definite performance cost. Worse, in its current state WeBIPP is poorly optimised in terms of what code it evaluates as the different levels of code are written and changed. It often evaluates more than is necessary to do the job, which again comes at a performance cost. Earlier versions of WeBIPP were in fact more optimised, but were less generalised, and when it came to extending WeBIPP via addons, this optimisation was often

a barrier. The current version trades this performance for increased generality and addon compatibility. However, some optimisation should be possible to improve performance.

This generalisation also means that the GUI is less tailored. Unlike some other software that can provide very specific GUI elements and shortcuts for modifying a limited number of well-defined elements, WeBIPP has to be very general to handle whatever may come its way via addons. While it is possible to create very specific GUI elements (see Section 5.8), and indeed Object Addon creators may create very specific Value Interfaces to go with their Objects, the base interfaces provided by WeBIPP will be, necessarily, very general, and not quite as easy to use as a more specific interface.

So the same choices that make WeBIPP what it is, and distinguishes it from other tools that seek similar ends, also limits WeBIPP, often in ways that can never be resolved as they are inherent to the choices made.

5.11 Conclusion and Future Work

Much of the work done on WeBIPP thus far has been to create a proof-of-concept, a working tool that demonstrates that the objectives I laid out at the start were feasible and could lead to a tool both powerful and easy-to-use. This work in creating a practically useful proof-of-concept is not yet over, and some key features WeBIPP could use are:

- More Objects, in particular different types of Frames and Templates for common plots.
- An easier way to layout Frames, to make it easier to produce juxtaposed plots.
- A better Functionise Interface, to make the process more intuitive.
- An ability to turn Functionise results into an Object Addon.
- Basic data manipulation capabilities, as it is often necessary to perform some transformations of the data to create interesting plots.
- A GUI way to add meaningful interactivity to the graphics.
- Batch processing capabilities, including the ability to export the resulting graphics.
- A way to collate and minify WeBIPP and its addons, to make it easier to mix-and-match addons.
- An ability to connect to other software, such as R, to make it easier to leverage more powerful statistical packages when using WeBIPP.

WeBIPP is still very much in early development, and has a lot of unexplored potential. Future uses for the tool go beyond creating interactive web-based graphics from scratch without writing code, it could also be used for teaching purposes, where the rapid construction of graphics from scratch can show students the component parts of graphics and how they come together to form a statistical graphic. I hope however that the current implementation works well enough as a proof-of-concept that it demonstrates it is indeed possible to create a tool both powerful yet easy-to-use.

Chapter 6

Conclusion

This thesis has introduced two tools that address some of the problems identified in the Literature Reviews. TableToLongForm seeks to make more Open Data useful by taking non-machine-readable hierarchical Tables and converting them into useful, machine-readable LongForm dataframes. WeBIPP attempts to make powerful graphics more accessible to a wider audience by bridging the gap between easy-to-use but limited GUI tools, and the powerful, but hard-to-use code-based tools.

The aim of this thesis was to make Open Data more *open*, and TableToLongForm does this by making more data available for manipulation with computer tools, enabling more technical users to make better use of the data, and these users can ultimately produce the kind of analyses, graphics, reports and summaries that will make the data consumable by a wider audience. WeBIPP also makes data more accessible by enabling users without coding knowledge to do now, what could previously only be done by writing code. Opening up the creation of new kinds of graphics to a wider audience, as well as making it easier to create new kinds of graphics, will help in making data more consumable.

Future Work

The future for TableToLongForm is limited. While it is certainly possible to increase its features with new modules, and while better diagnostics will be developed to increase usability, the fundamental structure of TableToLongForm restricts the kinds of things it can do. There is a lot of Open Data released in poor formats, and TableToLongForm can only handle some of it. A true expansion of the same philosophy, a tool that can handle far more formats, will likely require something quite different, something not as restrictive as a pure algorithmic approach. But there is another way in which TableToLongForm's future is limited, the problem that it tackles may cease to exist, hopefully in the near future. Consider for example Statistics New Zealand, which now releases a lot of its data

via NZ.Stat[1]. NZ.Stat has a feature to export the data into a CSV, and this CSV file is already in machine-readable longform. TableToLongForm is now obsolete with regards to converting data released by Statistics New Zealand, it is quite simply no longer necessary. While I like to think I had some part in this development, as I have been in contact with Statistics New Zealand regarding machine-readable data, nonetheless the problem no longer exists, at least for this organisation. It is my hope that eventually, all data is released in machine-readable formats, and tools like TableToLongForm become a thing of the past.

WeBIPP on the other hand has much more potential, venturing even beyond the original intent of the tool. In addition to improving the interface to make it better at doing what it does, the conceptual basis for WeBIPP has wider applications in teaching. WeBIPP itself can be used by a teacher to demonstrate to their students how the component parts of a graphic can come together to create a graph. Students could then get hands-on with the tool and create graphics themselves, a much more powerful way to learn than simply watching. But the same principles that make WeBIPP generalised, powerful and yet easy-to-use could be extended to other easy-to-deploy interactive web tools for other fields. Such tools could enable users to manipulate and see building before their eyes other systems, like mathematical equations or physical models, leading to a much more engaging way to teach these subjects as well. Currently these are only possibilities, but they are exciting possibilities, and I intend to explore them fully.

Final Remarks

A lot of Open Data is being released and already vast amounts of data are available. But simply being available does not make them valuable. This thesis has introduced concepts and tools that will hopefully make Open Data releases more useful, but making data more accessible and communicating its message to a wider audience is a massive task, one that goes far beyond what this thesis hopes to achieve. It is my hope that this makes a worthy contribution to the ongoing effort.

[1] http://nzdotstat.stats.govt.nz/wbos/Index.aspx

Bibliography

Bostock, M., Ogievetsky, V., Heer, J., 2011. D3: Data-driven documents. IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis) .
URL http://vis.stanford.edu/papers/d3

Chang, W., Wickham, H., 2015. ggvis: Interactive Grammar of Graphics. R package version 0.4.2.
URL http://CRAN.R-project.org/package=ggvis

Cleveland, W. S., 1985. The Elements of Graphing Data. Wadsworth Publ. Co., Belmont, CA, USA.

Cole, R., 2012. Some Observations on the Practice of Open Data As Opposed to Its Promise. The Journal of Community Informatics 8 (2).
URL http://ci-journal.net/index.php/ciej/article/view/920

Couture-Beil, A., 2012. rjson: JSON for R. R package version 0.2.10.
URL http://CRAN.R-project.org/package=rjson

Creative Commons, 2012.
URL http://creativecommons.org/

D3.js, 2012.
URL http://d3js.org/

Davies, T., Bawa, Z., 2012a. The Journal of Community Informatics 8 (2).
URL http://ci-journal.net/index.php/ciej/issue/view/41

Davies, T., Bawa, Z., 2012b. The Promises and Perils of Open Government Data (OGD). The Journal of Community Informatics 8 (2).
URL http://ci-journal.net/index.php/ciej/article/view/929

Department for Education (UK), 2013. NEET statistics quarterly brief: April to June.
URL https://www.gov.uk/government/publications/neet-statistics-quarterly-brief-april-to-june-2013

Department of Internal Affairs (NZ), 2012. Top 100 Baby Names.
URL http://www.dia.govt.nz/

Google Chart Tools, 2012.
URL https://developers.google.com/chart/

Highcharts, 2012.
URL http://www.highcharts.com/

Hocking, T. D., Sievert, C., Tsai, T., VanderPlas, S., 2015. Two new keywords for interactive, animated plot design: clickSelects and showSelected.
URL https://raw.githubusercontent.com/tdhock/animint-paper/master/HOCKING-animint.pdf

Ihaka, R., Mar. 2003. Colour for Presentation Graphics. In: Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003). Technische Universität Wien, Vienna, Austria.
URL http://www.r-project.org/conferences/DSC-2003/Proceedings/Ihaka.pdf

Inland Revenue, 2012. Tax statistics.
URL http://www.ird.govt.nz/aboutir/external-stats/

Konold, C., 2007. Designing a data analysis tool for learners. In: Thinking with data: The 33rd annual Carnegie Symposium on cognition. pp. 267–291.

Land Information New Zealand, 2012a. Open Data Service.
URL http://www.linz.govt.nz/open-data-service

Land Information New Zealand, 2012b. Open Data Service Agency Demand Survey.

Lang, D. T., 2012a. RCurl: General network (HTTP/FTP/...) client interface for R. R package version 1.91-1.1.
URL http://CRAN.R-project.org/package=RCurl

Lang, D. T., 2012b. XML: Tools for parsing and generating XML within R and S-Plus. R package version 3.9-4.1.
URL http://CRAN.R-project.org/package=XML

Many Eyes, 2012.
URL http://www-958.ibm.com/software/data/cognos/manyeyes/

Mirai Solutions GmbH, 2012. XLConnect: Excel Connector for R. R package version 0.2-0.
URL http://CRAN.R-project.org/package=XLConnect

Murrell, P., Potter, S., 2015. gridSVG: Export grid Graphics as SVG. R package version 1.4-3.
URL http://CRAN.R-project.org/package=gridSVG

New Zealand Government ICT, 2010. NZGOAL (New Zealand Government Open Access and Licensing) framework.
URL http://ict.govt.nz/guidance-and-resources/open-government/new-zealand-government-open-access-and-licensing-nzgoal-framework/

New Zealand Government ICT, 2011. Declaration on Open and Transparent Government.
URL http://ict.govt.nz/guidance-and-resources/open-government/declaration-open-and-transparent-government/

New Zealand Government ICT, 2012a.
URL http://ict.govt.nz/

New Zealand Government ICT, 2012b. 2012 report on adoption of the Declaration.
URL http://ict.govt.nz/guidance-and-resources/open-government/declaration-open-and-transparent-government/2012-report-adoption-declaration/

New Zealand Qualifications Authority, 2012. Secondary School Statistics.
URL http://www.nzqa.govt.nz/

Nolan, D., Lang, D. T., 2012. Interactive and animated scalable vector graphics and r data displays. Journal of Statistical Software 46 (1), 1–88.
URL http://www.jstatsoft.org/index.php/jss/article/view/v046i01

Oh, J., Dec. 2014. Automatic Conversion of Tables to LongForm Dataframes. The R Journal 6 (2), 16–26.
URL http://journal.r-project.org/archive/2014-2/oh.pdf

Open Knowledge Foundation, 2012.
URL http://okfn.org/

Open Knowledge Foundation, Jan 2013. "Carbon dioxide data is not on the worlds dashboard" says Hans Rosling.
URL http://blog.okfn.org/2013/01/21/carbon-dioxide-data-is-not-on-the-worlds-dashboard-says-hans-rosling/

Paper.js, 2012.
URL http://paperjs.org/

Penk, S., Tobin, R. (Eds.), March 2010. Privacy Law in New Zealand. Brookers Ltd.

Plotly, 2014.
URL https://plot.ly/

Processing, 2012.
URL http://processing.org/

Processing.js, 2012.
URL http://processingjs.org/

R Core Team, 2014. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.
URL http://www.R-project.org/

Raphaël.js, 2012.
URL http://raphaeljs.com/

Reas, C., Fry, B., 2006. Processing: programming for the media arts. AI & Society 20 (4), 526 – 538.

Robinson, D. G., Yu, H., Zeller, W. P., Felten, E. W., 2009. Government data and the invisible hand. Yale Journal of Law & Technology 11, 160.
URL http://ssrn.com/abstract=1138083

Sarkar, D., 2008a. Lattice: Multivariate Data Visualization with R. Springer, New York.
URL http://lmdvr.r-forge.r-project.org

Sarkar, D., 2008b. Lattice: Multivariate Data Visualization with R. Springer, New York, iSBN 978-0-387-75968-5.
URL http://lmdvr.r-forge.r-project.org

State Services Commission, 2012. New Zealand's State sector - the organisations.
URL http://www.ssc.govt.nz/state_sector_organisations

Statisphere, 2012a.
URL http://www.statisphere.govt.nz/

Statisphere, 2012b. Principles and protocols for producers of Tier 1 statistics.
URL http://www.statisphere.govt.nz/tier1-statistics/principles-protocols.aspx

Statisphere, 2012c. Tier 1 statistics.
URL http://www.statisphere.govt.nz/tier1-statistics.aspx

Statistics New Zealand, 2011. Age-Sex Population Pyramids.
URL http://www.stats.govt.nz/

Statistics New Zealand, 2012. About NZ.Stat.
URL http://www.stats.govt.nz/tools_and_services/nzdotstat.aspx

Statistics New Zealand, 2013. Infoshare.
URL http://www.stats.govt.nz/infoshare/

Stolte, C., Tang, D., Hanrahan, P., Nov. 2008. Polaris: a system for query, analysis, and visualization of multidimensional databases. Commun. ACM 51 (11), 75–84.
URL http://doi.acm.org/10.1145/1400214.1400234

Swayne, D. F., Buja, A., Temple Lang, D., 2004. Exploratory visual analysis of graphs in GGobi. In: Antoch, J. (Ed.), CompStat: Proceedings in Computational Statistics, 16th Symposium. Physica-Verlag.

Tableau, 2012.
URL http://www.tableausoftware.com/products/public

Urbanek, S., Theus, M., 2003. iplots: high interaction graphics for r. In: Proceedings of the 3rd International Workshop on Distributed Statistical Computing.

Wickham, H., 2009. ggplot2: elegant graphics for data analysis. Springer, New York.
URL http://had.co.nz/ggplot2/book

Wickham, H., 2014. tidyr: Easily Tidy Data with spread() and gather() Functions. R package version 0.2.0.
URL http://CRAN.R-project.org/package=tidyr

Wickham, H., Francois, R., 2015. dplyr: A Grammar of Data Manipulation. R package version 0.4.2.
URL http://CRAN.R-project.org/package=dplyr

Wilkinson, L., 1999. The Grammar of Graphics. Springer-Verlag New York, Inc.

World Wide Web Consortium, 2012.
URL http://www.w3.org/

Xie, Y., Hofmann, H., Cheng, X., et al., 2014. Reactive programming for interactive graphics. Statistical Science 29 (2), 201–213.

Xie, Y., Hofmann, H., Cook, D., Cheng, X., Schloerke, B., Vendettuoli, M., Yin, T., Wickham, H., Lawrence, M., 2013. Cranvas: Interactive statistical graphics based on qt. In: useR-2012.