COMP 120: Lab 8 Worksheet

As discussed in the lectures, many sources of data that you need to work with are not provided in rectangular format. In this lab, we will introduce the concept of Web scraping (also covered in lecture 17), where information has been provided within Web pages, and the structure of those pages (HTML + CSS) is used to identify and extract the relevant parts. Rather than work directly with the source code of the Web pages, we will use commands from the rvest package to do the bulk of the work for us.

Remember that you should copy and paste anything in a code block into the RStudio’s script window so that you can see the desired result for yourself.

Remember to set your working directory and library path (if working on the Student Desktop) at the start of your session. See the Lab 01 for advice on how to do this.

**This lab makes use of several Web sources, all residing on remote servers. Therefore, the work that you do in this lab may involve some waiting. Also, to perform Web scraping properly, you need a reasonable understanding of cascading style sheets (CSS) and hypertext mark-up language (HTML): we do not have time to explore HTML and CSS in detail in this lab[[1]](#footnote-1), and will only introduce the bare minimum to get by. Having said that, this lab does not assume any prior knowledge on HTML/CSS.**

# Setup

To scrape the Web, we need two things:

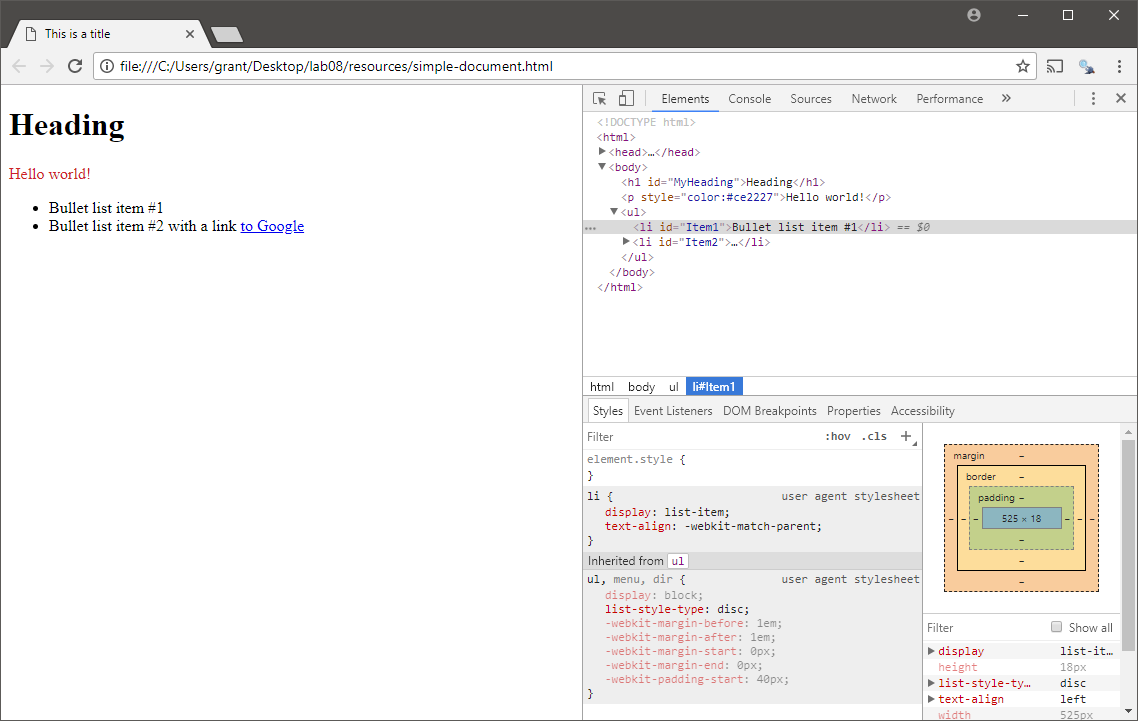
1. a package containing functions to ease the retrieval and manipulation of Web documents; and
2. browser tools to make it easier to navigate documents and identify clues and tags to extract document elements.

Getting the first in place is easy, we simply issue an install.packages("rvest") command in the R console (do this now!). For the second requirement, **we strongly recommend the use of the Chrome browser**, as it provides excellent “Developer Tools” for navigating the structure of Web pages. We will be using Chrome in all the examples in this lab.

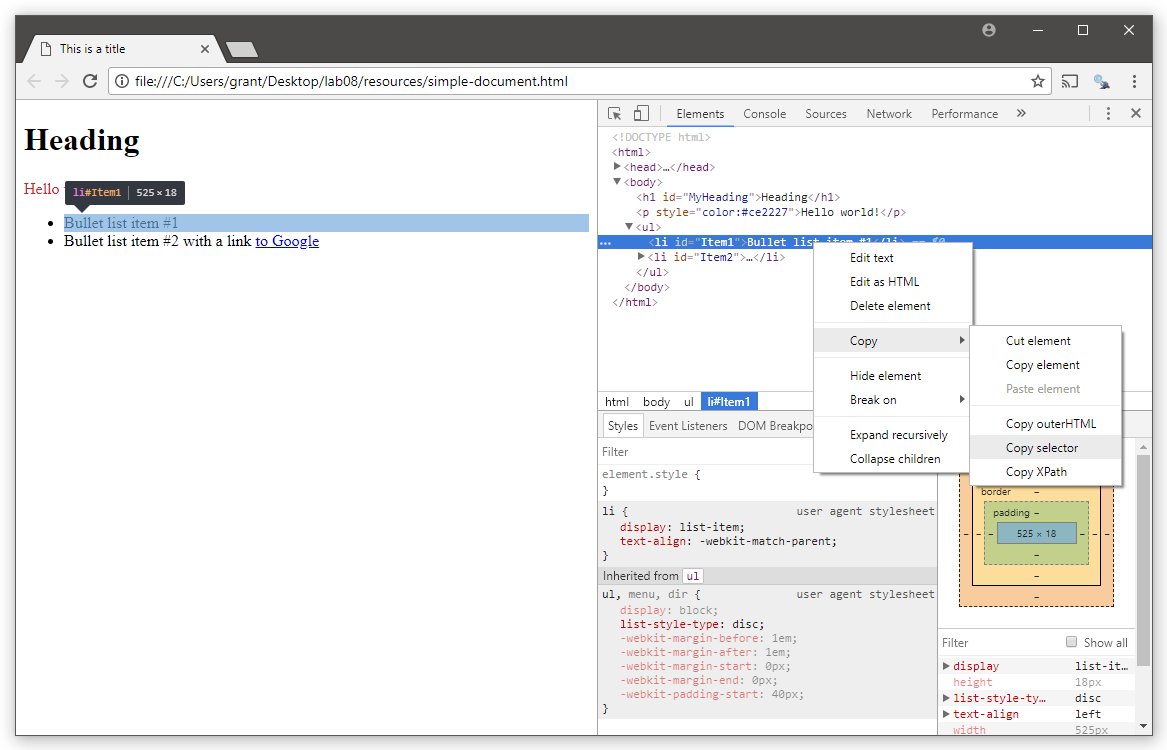
# Precursor to scraping - understanding the document structure

Prior to performing the actual scraping, we need to identify the pages we retrieve data from, how to obtain those pages, and then what parts of those pages contain data that are of interest. Essentially, Web scraping is nothing more than writing code to recreate the steps that you would use to visit the required pages manually and copy and paste the relevant data into your workspace. Therefore, an important step in scraping is writing down a good description of the required workflow.

Let’s start with a very simple scrape involving a single Web page that you can download from Blackboard. Download the file *simple-document.html* from Blackboard and open it in Chrome. You will be presented with a very simple Web page containing only a few text elements. Now, right-click on the first bullet point item in the page and choose the ‘Inspect’ option at the bottom of the menu. This opens Chrome’s developer mode and highlights the relevant part of the document’s source code that produces the bullet point item, and should resemble the following example:



The top-right panel shows the source code for the page: if you hover over the elements in this document, you should see that the corresponding outputs in the left panel are highlighted. We can identify any of the relevant parts of this document by right-clicking on the corresponding piece of source code and choosing the ‘Copy selector’ option from the Copy submenu, as shown below:



Here, we have requested the selector for the first bullet point item, which should provide the following: #Item1. If we request the selector for the Heading, then we should get: #MyHeading. As documents get more complex and structured, these selectors become more complex.

We will use this process of opening pages, inspecting, and copying selectors as the bedrock for the rest of this lab.

# Your First Scrape

It’s time to scrape the elements of the document that we worked with in the previous section. We start by introducing the rvest library into R:

library(rvest)

This package contains several useful functions for interrogating Web pages:

* read\_html(url) reads the contents of Web pages for you and loads the document into R
* html\_node(doc, selector) extracts a specific element from the document doc that matches the CSS selector selector. If more than one element matches, then the first one encountered in the document is returned.
* html\_nodes(doc, selector) extracts all the elements from the document doc that matches the CSS selector selector.
* html\_text(element, trim) extracts the text from the provided HTML elements, and optionally trims any leading and trailing whitespace from the result (when trim is set to TRUE).
* html\_table(node, fill) parses the html table data available in the *node* into a data frame. The fill argument when set to TRUE, fills the table with column data (with NAs for unavailable values).

There are other functions in rvest[[2]](#footnote-2) which are out of the scope for this lab (and the course).

Let’s start by loading our document into R:

doc <- read\_html("simple-document.html")

From here, we can extract a specific element from the document (e.g., the first bullet list):

e <- html\_node(doc, "#MyHeading")

and then finally, we can extract the text from the selected element using the html\_text() function:

html\_text(e, trim=TRUE)

[1] "Heading"

Like the other packages that we’ve seen in previous labs, rvest is designed to be used with the pipe operator. Therefore, you can perform the above in a single step using pipes as:

read\_html("simple-document.html") %>%

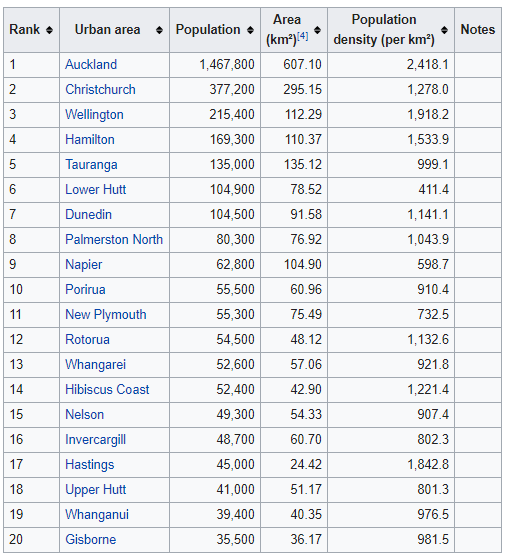
html\_node("#MyHeading") %>% html\_text(trim=TRUE)

[1] "Heading"

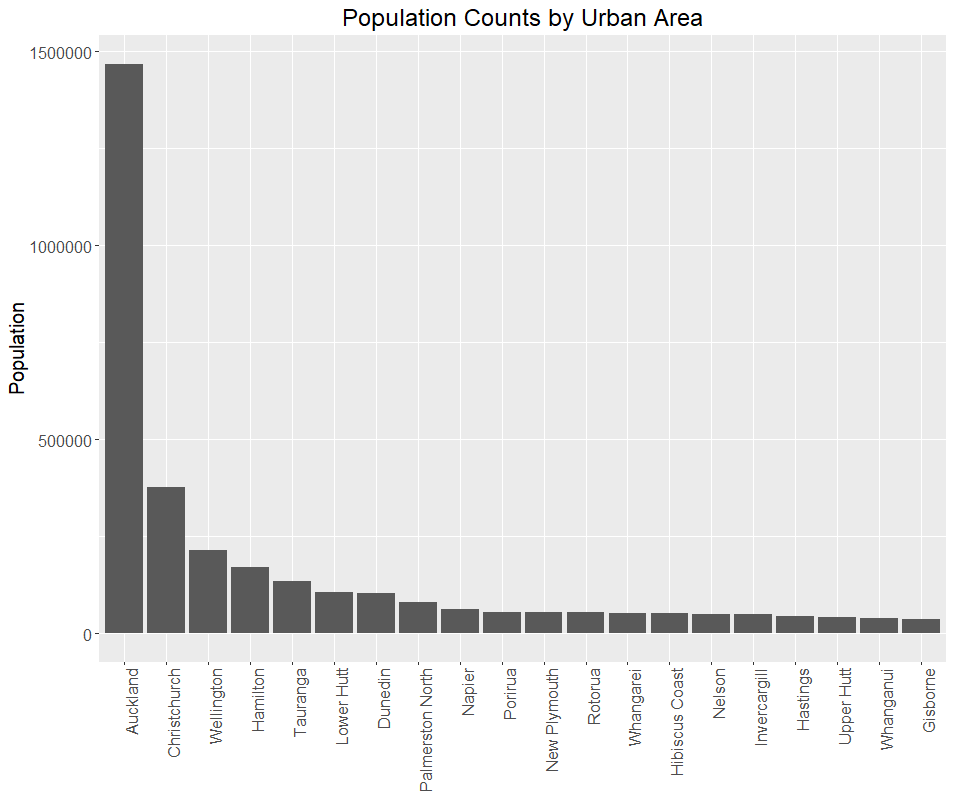
The pipes option is more compact and readable. So, we would recommend using pipes!

# A More Advanced Scrape

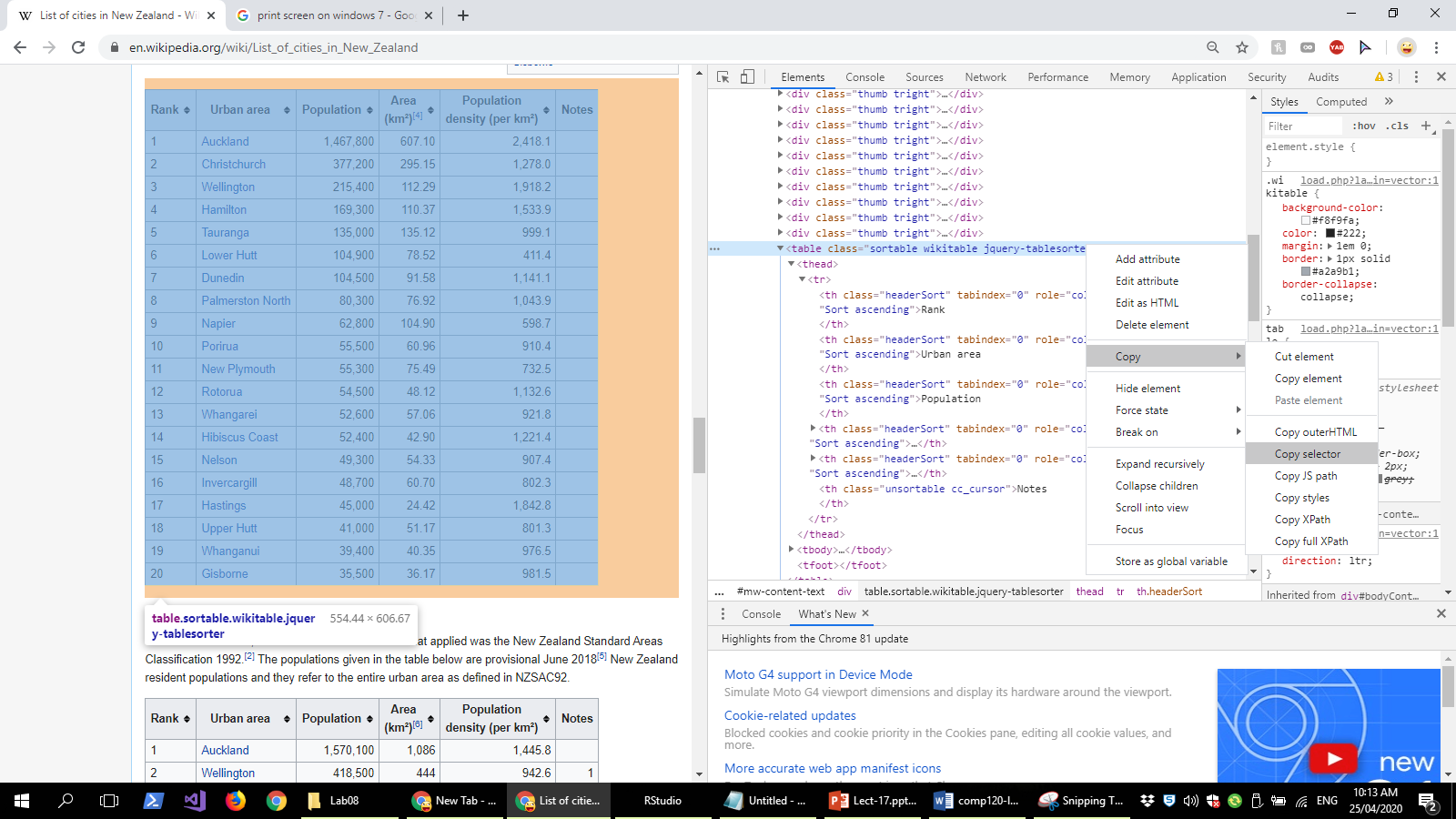
One of the problems with the Web is that data is presented in a way that the page author deemed appropriate (e.g., a table), but sometimes you need it in another format (e.g., a graph). Take, for example, the Wikipedia page for New Zealand population centres[[3]](#footnote-3): this page presents the population, area and density for the major NZ urban centres as a series of tables such as the one given below:



Our goal here is to scrape this information from Wikipedia, clean it up as required, and then produce the plot shown in the next page.



As before, the first thing that we need to do is to understand the required Web pages (in this case, just the one page), and the plan required to extract the required data (in this case, just the single table). Let’s start by opening the required page in Chrome and identifying the required selector for this page: in Chrome, inspect the table in the ‘Urban areas by population’ section under ‘population based on 2018 standard’:



If you copy the selector for this table you should obtain: #mw-content-text > div > table:nth-child(33). We now have everything that we need to extract this table from the page:

wikipedia.page <-

"https://en.wikipedia.org/wiki/List\_of\_cities\_in\_New\_Zealand"

page <- read\_html(wikipedia.page)

cities <- page %>%

html\_node("#mw-content-text > div.mw-parser-output > table:nth-child(35)") %>%

html\_table(fill=TRUE)

As it stands, the cities table is a bit messy, and needs some cleaning. Let us inspect the result of our scraping by using the following line of code.

cities

You may observe that there are a few issues with the results. First, the column names can be cleaned (named appropriately). We’ll use the colnames function to set the names of the columns in the cities data frame. Second, let us convert the *Name* column into a factor which will be useful in creating graphs. Third, the commas in the *population* and *area* columns should be removed. Also, the data in these columns are characters and they need to be converted to numbers. Only then we can perform some computation with these columns. Finally, let us create a new column called *Density* (i.e., Population/Area).

Let’s perform these operations on the cities data frame obtained above, using the following code:

colnames(cities) <- c("Rank", "Name", "Population", "Area",

"Density", "Notes")

cities <- cities %>% mutate(

Name=factor(Name, levels=Name),

Population=as.numeric(str\_replace\_all(Population, ",", "")),

Area=as.numeric(str\_replace\_all(Area, ",", "")),

Density=Population / Area

) %>% select(Name, Population, Density)

Name Population Density

1 Auckland 1467800 2417.7236

2 Christchurch 377200 1277.9942

3 Wellington 215400 1918.2474

4 Hamilton 169300 1533.9313

5 Tauranga 135000 999.1119

6 Lower Hutt 104900 1335.9654

7 Dunedin 104500 1141.0788

8 Palmerston North 80300 1043.9418

9 Napier 62800 598.6654

10 Porirua 55500 910.4331

11 New Plymouth 55300 732.5474

12 Rotorua 54500 1132.5852

13 Whangarei 52600 921.8367

14 Hibiscus Coast 52400 1221.4452

15 Nelson 49300 907.4176

16 Invercargill 48700 802.3064

17 Hastings 45000 1842.7518

18 Upper Hutt 41000 801.2507

19 Whanganui 39400 976.4560

20 Gisborne 35500 981.4764

Note that the mutate function in the third line in the snippet above is used to modify the values within columns and to add the new column (*Density*).

We now have a data frame that is ready for plotting:

cities %>% ggplot(aes(x=Name, y=Population)) +

geom\_bar(stat="identity") +

ggtitle("Population Counts by Urban Area") +

theme(axis.text.x = element\_text(angle=90, hjust=1),

axis.title.x = element\_blank(),

plot.title = element\_text(hjust = 0.5))

Note: within the theme() function, the code *axis.text.x=element\_text(angle=90, hjust=1)* is used to rotate the labels on the x-axis by 90 degrees so that they are easier to read. The code *axis.title.x=element\_blank()* sets a blank name for the x-axis title since a title is not required in this case. The code *plot.title = element\_text(hjust = 0.5)* sets the plot title to the center of the plot. So far, the plots we have been producing had the titles starting at the top left-hand corner.

Now, the task for you to do is to scrape the data given in the *third table* in the same web page on populations of present-day city councils (<https://en.wikipedia.org/wiki/List_of_cities_in_New_Zealand>). The extracted data should look like the following. After extracting the data, create the visualisation shown below.

Rank Council Population First\_proclaimed

1 1 Auckland 1618400 1871

2 2 Christchurch 380200 1868

3 3 Wellington 209000 1870

4 4 Hamilton 165900 1936

5 5 Tauranga 140800 1963

6 6 Dunedin 130500 1865

7 7 Lower Hutt 107600 1941

8 8 Palmerston North 87300 1930

9 9 Napier 64100 1950

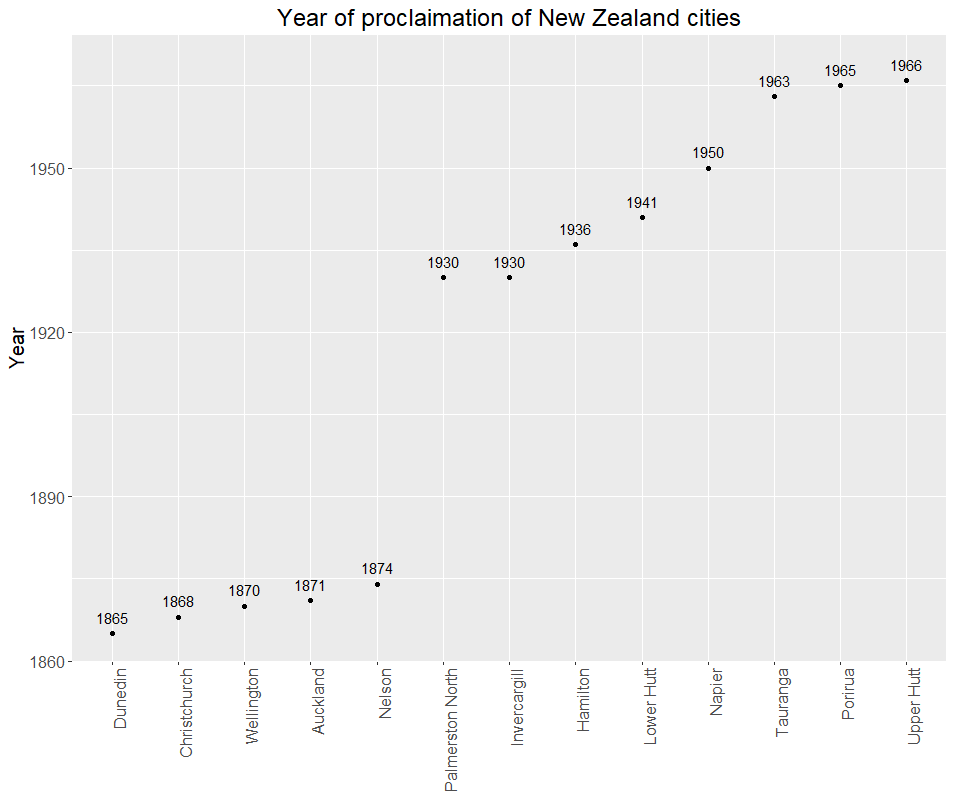
10 10 Porirua 58200 1965

11 11 Invercargill 55800 1930

12 12 Nelson 52400 1874

13 13 Upper Hutt 45300 1966

Use the data above to create the visualisation shown below.



If you were unable to create the plot, here is the code that can produce the result for you. Notice that the code assumes you have stored the data in a data frame called *city\_councils*.

ggplot(data = city\_councils, aes(x = reorder(Council, First\_proclaimed), y = First\_proclaimed)) +

geom\_point() +

geom\_text(aes(label = First\_proclaimed), nudge\_y = 3) +

ggtitle("Year of proclaimation of New Zealand cities") +

ylab("Year") +

theme(axis.text.x=element\_text(angle=90, hjust=1),

axis.title.x=element\_blank(),

plot.title = element\_text(hjust = 0.5))

A couple of places where you may have had trouble are: 1) the use of an appropriate function to create text labels for each data point and 2) the reordering of x-axis based on the ascending order of values in y-axis. Let us look at those two aspects in a bit more detail.

First, the code that creates label for the data points shown on the plot is given below.

geom\_text(aes(label = First\_proclaimed), nudge\_y = 3)

Note *First\_proclaimed* is the column that contains the values that get displayed above the data points (i.e., labels) in the plot. The argument *nudge\_y*, shifts the label’s location by certain units of y-axis. In this case, the label is shifted by 3 units (i.e., years). If this value had been -3, the labels would have been placed below the data point by 3 units. If you like to learn more about geom\_text function, see <https://ggplot2.tidyverse.org/reference/geom_text.html>.

Second, the reorder function is used to reorder the values on x-axis based on ascending order of the values in y-axis. The argument *x = reorder(Council, First\_proclaimed)* reorders the values of *Council* based on ascending order of the variable *First\_proclaimed*. Now, you should be ready to tackle the mastery tasks.

# Mastery Tasks

This set of mastery tasks requires you to create three plots based on data obtained using web scraping (covered in this lab).

**Task 1.** Extract the details of all lakes in New Zealand that have a surface area greater than 10 square kilometres given in this Wikipedia page (<https://en.wikipedia.org/wiki/List_of_lakes_of_New_Zealand>). There are a total of 33 such lakes whose details are given in the first table. You must remove the last row of data that does not contain details about a lake. Now, assuming the name of the column that stores the details about surface area of the lakes is *SurfaceArea*, and the data is stored in a tibble called *lakes*, the following code will clean the data and extract the first set of digits corresponding to surface area in square kilometres (e.g. 616 for the largest lake).

lakes$SurfaceArea <- str\_extract(lakes$SurfaceArea, "\\d\*")

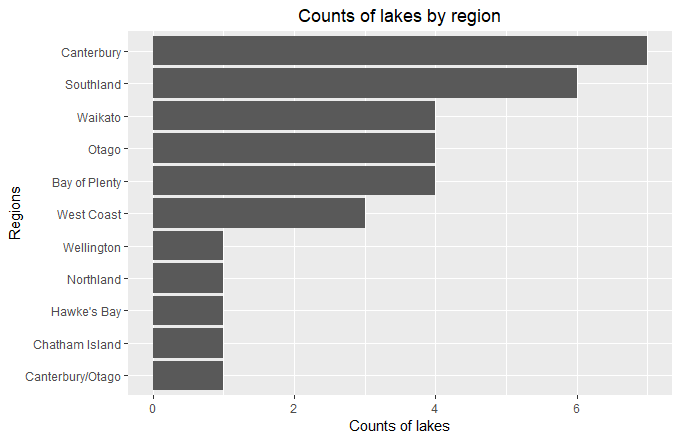
Note that you will need to process the data for the *SurfaceArea* column such that the column is a numeric data type column.

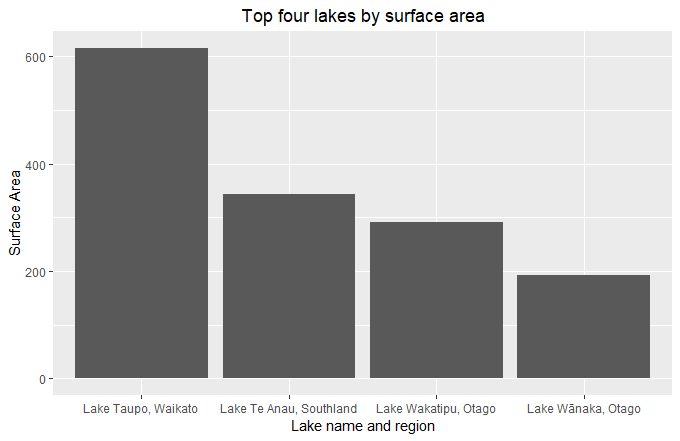
Now, write code to create the following two plots shown below. Use appropriate functions introduced in the previous labs to create the data for plots (e.g. data aggregation and reshaping functions).

1.1. Plot showing the counts of lakes per region

1.2. Plot showing the top four lakes based on surface area

Hint: the ggplot option that may come handy to create the first plot below is *coord\_flip()* that flips the coordinates.



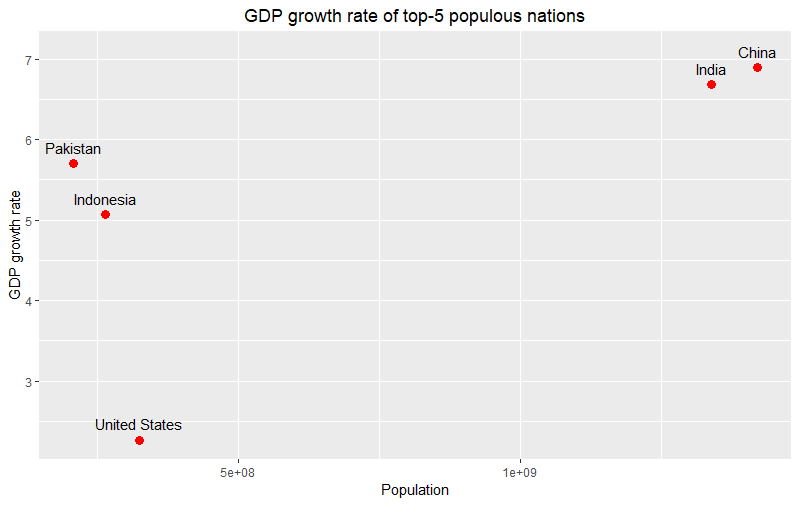


**Task 2.** For creating the next plot, you will be extracting Gross Domestic Product (GDP) data of different countries from this page (<https://www.worldometers.info/gdp/gdp-by-country/>) for the year 2017. There are 189 countries whose details are given in the table. We are only interested in data for five columns (the first three columns and columns five and six) out of a total of eight columns. Let us assume that the names of these five columns of interest have been changed to Rank, Country, GDP, GDP\_growth, Population. Now, assuming the data is stored in a tibble called GDPbyCountry the following code will clean the GDP column by removing comma (,), dollar sign ($), and the percent (%) symbols.

GDPbyCountry$GDP <- str\_replace\_all(GDPbyCountry$GDP, ",|\\$|%", "" )

Note that you should perform this step for the other columns of interest except for rank and country name columns and also you will need to convert these columns into numeric data type columns.

Now that you have cleaned the data, write code to create plot shown below. The plot shows the GDP growth rate of top-5 populated nations.



Put all the required code to scrape the data and to generate the three plots into a single script (mastery-08.R). Place a short comment before each block of code that you use to complete the tasks. When you have completed the tasks, **submit your work on Blackboard before 4pm on Thursday, 17th of September**. **Note: As usual, not writing good comments will attract 5% penalty for each plot**.

1. **But, there is a separate course on web development which covers these in detail – COMP 112 at Otago.** [↑](#footnote-ref-1)
2. <https://cran.r-project.org/web/packages/rvest/rvest.pdf> [↑](#footnote-ref-2)
3. List of cities in New Zealand: <https://en.wikipedia.org/wiki/List_of_cities_in_New_Zealand> [↑](#footnote-ref-3)