Minilab 3c Worksheet

Data Scraping and Data Cleaning

In a previous minilab, we have seen that it is very easy to read a dataset from a "comma-separated values" or CSV file using `read_csv()` from the R tidyverse package "readr" (see https://readr.tidyverse.org/). You can open a CSV file in Notepad as see that it is just a text file where each line is just entries (strings or numbers) separated by commas. Microsoft Excel will also read and write CSV files.

There are other very useful functions in readr that help to read data from nicely formatted files. You can read a dataset from a Microsoft Excel file (XLS or XLSX) using functions from the "readxl" package (see https://readxl.tidyverse.org/). We can also read data from a database (using the "dbplyr" package).

In this minilab, we will look at another useful way of reading data into R, which is to *scrape* it from a webpage. However, we get data into R we often have to do some *cleaning* up of data into form that is useful for further analysis.

1. Data Scraping

Recall the Data Mining (or Data Science) lifecycle.

Image from https://towardsdatascience.com/5-steps-of-a-data-science-project-lifecycle-26c50372b492

You may be wondering where data comes from and why it might need scrubbing (or cleaning). Often data comes from files (CSV, XLS) or databases in a nice tidy form ready for querying and analysis. But sometimes we see a table of data on a webpage and wish to import it into R somehow.

We wish to avoid having to retype or copy-and-paste, i.e., we wish to be able to automate and scale this process.

Since no two websites are the same, scraping requires you to identify and exploit patterns in the code that renders websites. Each website is rendered by your browser from HTML, and the goal of scraping is to parse the HTML that is sent to your browser into usable data.

The R tidyverse package "rvest" (think harvest) provides functions for parsing webpages. We need to tell it which web page to load, where to "look" on the page (particular CSS tags) and how to extract the data in a usable format.

(1) Take a look at the webpage http://www.espn.com/nfl/superbowl/history/winners. The page includes a table of all the results of the "Super Bowl" in NFL history (the annual final of American Football).

(2) Now we will import this webpage (as html) into R. If you try `View(page)` you will see that the page is a structured set of html tags (nodes).

```
library(tidyverse)
library(rvest)
library(lubridate)
url = "http://espn.go.com/nfl/superbowl/history/winners"
page = read_html(url)
page
```

(3) We can look for the html tag "table" on the page to extract the table from the page.

```
sb_table = html_nodes(page, 'table')
sb = html_table(sb_table)[[1]]
sb
```

(4) The top two rows are headings, so we can remove these from the table and set the column labels with meaningful names. Finally we can bring it into the tidyverse by making it a tibble.

```
sb = sb[c(-1,-2),]
```

```
names(sb) = c("number", "date", "site", "result")
sb = as_tibble(sb)
sb
```

Exercise. Below is some code for scraping data from IMDb (the Internet Movie Database). The structure of the webpage is not a clean html table, so the approach is slightly different.

```
library(tidyverse)
library(rvest)
url="https://www.imdb.com/search/title/?title_type=feature&genres=sci-
fi&sort=num_votes,desc&count=250&explore=genres&view=advanced"
webpage = read_html(url)
title = html_text(html_nodes(webpage, '.lister-item-header a'))
rating = html_text(html_nodes(webpage, '.ratings-imdb-rating strong'))
table = tibble(title=title, rating=rating)
View(table)
```

(a) The *runtime* of each film can be extracted using the tag '.text-muted .runtime'. Try adding this to the R code above.

(b) What do you notice about the runtime of each film?

2. Data Cleaning

Image from: https://www.youtube.com/watch?v=7zc55KYLN28

We have managed to *capture* the table from the webpage into R. It is still a bit of a mess.

- The first column is a string (notice the <chr> when we look at the first few rows of the tibble) which give Roman numerals.
- The second column is a string which gives the date.
- The third column gives both the name of the stadium and the city (two data values in one cell of the table).
- The final column gives the winning team, winning score, losing team and losing score (four data values in one cell of the table).

We can use functions from the R packages *dplyr*, *lubridate* and *tidyr* to clean up these common issues.

(1) We can replace the roman numerals with integers, just by noting that the superbowl number is just the same as the row number.

```
mutate(sb,number=1:nrow(sb))
```

(2) We can use the mdy() function (month/day/year format of dates) from the R tidyverse package "lubridate" to translate the string format dates to an actual date type. This then allows for calculations involving dates, e.g., how many days elapsed. The functions in lubridate allow for dates and times involving timezones, leap days, etc.

```
mutate(sb,date=mdy(date))
```

(3) Separating the site column into the stadium and the city is by specifying the characters used to make the split, i.e., '(' and ')'. This uses the separate() function from the R tidyverse package "tidyr" and the separator is actually a *regular expression* (regex). This is whole new level of attention to detail (blood, sweat and tears) to get correct.

```
separate(sb,site,c("stadium","city",NA),sep='[()]')
```

(4) We can use separate again to split the winning team/score from the losing team/ score, because there is a convenient comma to use as the separator.

```
separate(sb,result,c("winner","loser"),sep=', ')
```

(5) It is a bit more fiddly to separate the winning team from the winning score. The regular expression in the variable *pattern* below is a space, followed by one or more digits, followed by the end of the string ($).

```
pattern = ' \\d+$'
separate(sb,result,c("winner","loser"),sep=', ') %>%
  mutate(winner_score=as.numeric(str_extract(winner,pattern))) %>%
  mutate(winner=gsub(pattern,"",winner))
```

Similarly, we can separate the losing team from the losing score (see exercise below).

(6) All the data cleaning carried out above has been one operation at a time and no cumulative changes have been made. Typically we would experiment to get each stage of the cleaning "just right" and put it all together in a pipe. The pipe below gives everything we have done so far.

```
# Pipe
pattern = ' \\d+$'
sb %>%
  mutate(number=1:nrow(sb)) %>%
  mutate(date=mdy(date)) %>%
  separate(site,c("stadium","city",NA),sep='[()]') %>%
  separate(result,c("winner","loser"),sep=', ') %>%
  mutate(winner_score=as.numeric(str_extract(winner,pattern))) %>%
  mutate(winner=gsub(pattern,"",winner))
```

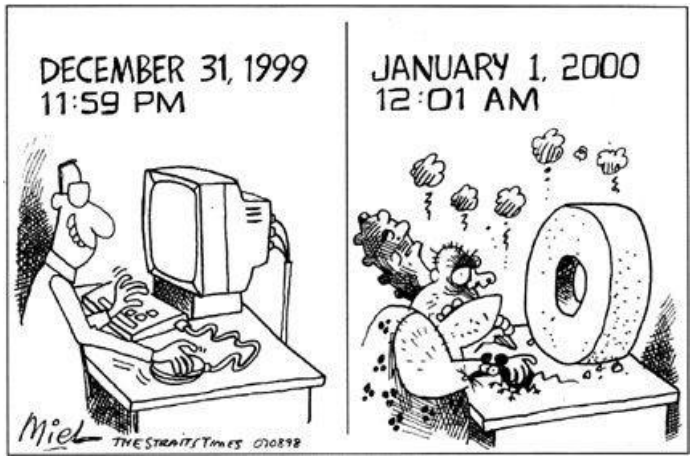
Exercise.

(a) Complete the pipe above to separate the losing team from the losing score.

(b) Use the result part (a) to plot a scatterplot of date vs margin of win (winner score minus loser score).

3. More about Dates

What was the issue with the "Millennium Bug" (also known as "Y2K")?



https://www.reddit.com/r/agedlikemilk/comments/egl5et/y2k_comic/

The R package "lubridate" provides tools that make it easy to enter and manipulate dates. It takes account of leap years and time zones.

(1) Have a look through the documentation for lubridate at: https://lubridate.tidyverse.org/reference/lubridate-package.html

(2) Suppose we wish to know which day of the week it is.

```
library(lubridate)
wday(today(),label=TRUE)
```

(3) The functions dmy(), myd(), ymd(), ydm(), dym(), mdy() and ymd_hms() all provide various ways to enter dates.

```
dmy(01032020)
```

[1] "2020-03-01" *which is 01 March 2020.*

```
dmy(28022020)
```

[1] "2020-02-28" *which is 28 March 2020.*

```
dmy(01032020) - dmy(28022020)
```

Time difference of 2 days *Remember that 2020 is a leap year.*

There is a lot more information at: https://r4ds.had.co.nz/dates-and-times.html

Exercise.

(a) Tyson Fury (born 12/08/1988) and Deontay Wilder (born 22/10/1985) boxed for the WBC Heavyweight title. Use lubridate to calculate how many days older Wilder is than Fury. *Hint:*

```
tyson_fury = dmy("12/08/1988")
```

(b) At the very end of 31 December 2016 there was an additional "leap second" inserted into the world clock, i.e., the final minute of 2016 was 61 seconds long. Check if lubridate can calculate this.

Challenge (optional). Foo and Bar are twins. Foo is older than Bar by 5 minutes. This year, Bar (the younger twin) will celebrate his birthday and then two days later Foo (the older twin) will celebrate his birthday. *Explain.* As an additional challenge, investigate how to record the twin's birthdates using lubridate (you will need to investigate timezones).

4. (Optional) Identifying CSS tags

The difficult part with scraping data from a website is identifying the CSS tags that show you which part of a html page comprise the data you are after. Fortunately, there is a clever tool called "SelectorGadget".

Open up the Google Chrome browser, and go to
https://rvest.tidyverse.org/articles/selectorgadget.html

Make sure your Bookmarks bar is showing (ctrl-shift-B) and then drag the Selector Gadget to the bookmarks bar.

Read the "vignette" webpage and see how SelectorGadget works.

Try scraping data from another website.

Summary

In this minilab, we have seen how it is possible scrape a dataset from a static webpage (using the R package "rvest").

We then followed a process to scrub/clean the data into a suitable form for further analysis. The key idea is that **each cell in the data table must consist of just one piece of information** (value). This is "First Normal Form (1NF)" in the world of relational databases.

We have also seen some indication of the usefulness of the R package "lubridate" for storing dates and times and performing calculations on dates and times.