Cutting out the Copy

Author: Marieth Coetzer - Mentor: Professor Michael Tsiang

June 10th, 2022

1. Introduction

It is no secret that we live in a data-driven society. With everything from our phones to our cars to our household appliances gathering massive amounts of data, the world of data analyzing often feels intimidating for a regular person. In my early education, the only way I knew how to analyze data was with Excel, which often included using tedious copy-and-paste techniques. However, once I started university, I realized that there were much more efficient ways to work with data using R and Python.

Despite the power of R to manage and sort huge sets of data, one thing was lacking - control of what was imported. Usually, entire data sets are imported into R as data frames so that they can be analyzed. However, these data sets often contain irrelevant information, meaning that when the user loads all the data into R, they are using more R resources than what they need. This is negligible when working with small data sets, but when sets become large, this can lead to a slow program. Additionally, some users do not have the processing power needed to import entire data sets into R.

This leads to the first part of my project, which was to write a program for a user to specify which rows and columns of an Excel file they want to import into R. Once the data is in R, it can be more easily explored, analyzed, and used to create images. I did not explore how to do this with .csv files, but was still able to apply the program to these files after saving them as Excel files. The second part of my project consisted of exploring web scraping and how a data frame in R can be used to convert data imported as a character type into a numeric type.

2. Documentation

In order to start my project, I explored several libraries that could be used to import Excel files into R. XLConnect contained the most features that I was interested in, but when I tried running it I ran into the error, “JAVA_HOME cannot be determined from the Registry.” I did some research and found that it required an extension of Java (6). Because I wanted to keep my program as simple as possible, I focused on other packages.

In the end I chose to go with readxl because it is compatible with tidyverse and ggplot2 (8), which I thought would be useful if people wanted to import Excel data in R to create graphs. I also found detailed documentation on how to implement readxl (7) and a very useful cheat sheet from Github (10).

Compared to libraries that import Excel files, I did not find many options or clear documentation on what libraries to use for web scraping. In the end, I found a YouTube video that showed how to import a table of data from Wikipedia, which was very similar to what I wanted to do (9). The video introduced me to the library dplyr (2) and rvest (11). The library dplyr exists to manage data similar to a data frame while rvest was created to scrap data online.

3.1 Findings from Excel

Once I had chosen readxl as the library I wanted to use to import data from Excel, I experimented to see what would happen to a data set when I imported it. As can be seen from the code below, it is quite easy to import an entire Excel sheet and put it into a data frame using *read_excel*. One shortcoming of this function is that it only has

a column names argument, meaning it does not import row names. However, once the Excel sheet is in R, it is possible to take the first column to make it the row names of the data frame.

```
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.1.3
```

```
file_loc <- "C:\\Users\\schoo\\Desktop\\UCLA 3\\Stats 199\\Week 10\\Small File.xlsx"

data <- read_excel(file_loc,  col_names = TRUE)
```

```
## New names:
## * `` -> `...1`
```

```
data_frame <- data.frame(data)

print(data_frame)
```

```
##      ...1 Bob.Store Ann.Store David.store
## 1  Apples        23        32          21
## 2 Oranges        43        19          34
## 3  Lemons        35        27          40
```

```
print("Sum of vegetables in every store: ")
```

```
## [1] "Sum of vegetables in every store: "
```

```
print(lapply(data_frame[,2:4], sum))
```

```
## $Bob.Store
## [1] 101
##
## $Ann.Store
## [1] 78
##
## $David.store
## [1] 95
```

While the code above does work with small data sets, the goal of this project was to import only certain rows or columns of data. As a result, I used the *range* argument of the *read_excel* function to specify which cells to import then used them to create a data frame. Doing this taught me about tibble data frames, which use the class tbl_df (1). This restricted how rows and columns could be added to a data frame, so I worked on removing this attribute by creating the *excel_import* function. Doing this made it possible to add vectors to a data frame, which I used to create the bulk of my result, as is shown below. Please note that some of the inputs in my program have been commented out for easy readibility in R Markdown.

```
# Data set used for this example is from: https://worldhappiness.report/ed/2021/

library(writexl)
```

```
## Warning: package 'writexl' was built under R version 4.1.3
```

```
excel_import <- function(file_loc, coord) {

  data <- read_excel(file_loc, range = coord, col_names = FALSE)
  attributes(data) <- NULL
  data_tidy <- unlist(data[1:length(data)])
  return (data_tidy)

  }

# print("You are about to start the process of creating a data frame")
# file_loc <- readline(prompt = "Please enter the location of the Excel file using double back s
lashes (\\): ")
file_loc <- "C:\\Users\\schoo\\Desktop\\UCLA 3\\Stats 199\\Week 10\\DataPanelWHR2021C2.xls"

# print("Would you like to import a row or a column?")
# row_col <- readline(prompt = "'r' for row, 'c' for column: ")
row_col <- "c"

# print("Do you want the names in the first row or column of your import to be the names of the
 dataframe?")
# print("Note: Duplicate row and column names are not allowed. If the error occurs, your data wi
ll be")
# print("put into the data frame, but the duplicate name will not be saved and the program will
 end.")
# names_yn <- readline(prompt = "'y' for yes, 'n' for no: ")
names_yn <- "n"

# print("What row or column would you like to import?")
# coord1 <- readline(prompt = "Please enter in the following format 'A1:A3': ")
coord1 <- "B1838:B1852"

data <- excel_import(file_loc, coord1)
```

```
## New names:
## * `` -> `...1`
```

```
if (row_col == "c") {

  if (names_yn == "y") {
    data_frame = data.frame(cbind(data[2:length(data)]))
    colnames(data_frame)[1] <- data[1]
  } else {
    data_frame = data.frame(cbind(data[1:length(data)]))
    }
  }

if (row_col == "r") {
  if (names_yn == "y") {
    data_frame = data.frame(rbind(data[2:length(data)]))
    rownames(data_frame)[1] <- data[1]
  } else {
    data_frame = data.frame(rbind(data[1:length(data)]))
    }
}

# print("Here is your data frame")
# print(data_frame)

# input_yn <- readline(prompt = "Would you like to add more rows or columns to your dataframe?
 'y' for yes 'n' for no: ")
input_yn <- "y"
# input_df <- readline(prompt = "Would you like to see the dataframe as columns or rows are adde
d? 'y' for yes 'n' for no: " )
input_df <- "n"

while (input_yn == "y") {

  # cat("Your original entry was", coord1)
  # coord2 <- readline(prompt = "Please enter the location of the cells you want to import in th
e format 'A1:A3': ")
  coord2 <- "C1838:C1852"
  data2 <- excel_import(file_loc, coord2)

  if (row_col == "c") {

    if (names_yn == "n") {
      data_frame[,ncol(data_frame) + 1] <- data2
    }

    if (names_yn == "y") {
      data_frame[,ncol(data_frame) + 1] <- data2[2:length(data2)]
      colnames(data_frame)[ncol(data_frame)] <- data2[1]
    }
  }


  if (row_col == "r") {
```

```
    if (names_yn == "n") {
      data_frame[nrow(data_frame) + 1, ] <- data2
    }

    if (names_yn == "y") {
      data_frame[nrow(data_frame) + 1, ] <- data2[2:length(data2)]
      rownames(data_frame)[nrow(data_frame)] <- data2[1]
    }
  }


  if (input_df == "y") {
    print(data_frame)
  }

  # input_yn <- readline(prompt = "Would you like to add more columns or rows? 'y' for yes 'n' f
or no: ")
    input_yn <- "n"
  }
```

```
## New names:
## * `` -> `...1`
```

```
print("Here is your final dataframe: ")
```

```
## [1] "Here is your final dataframe: "
```

```
print(data_frame)
```

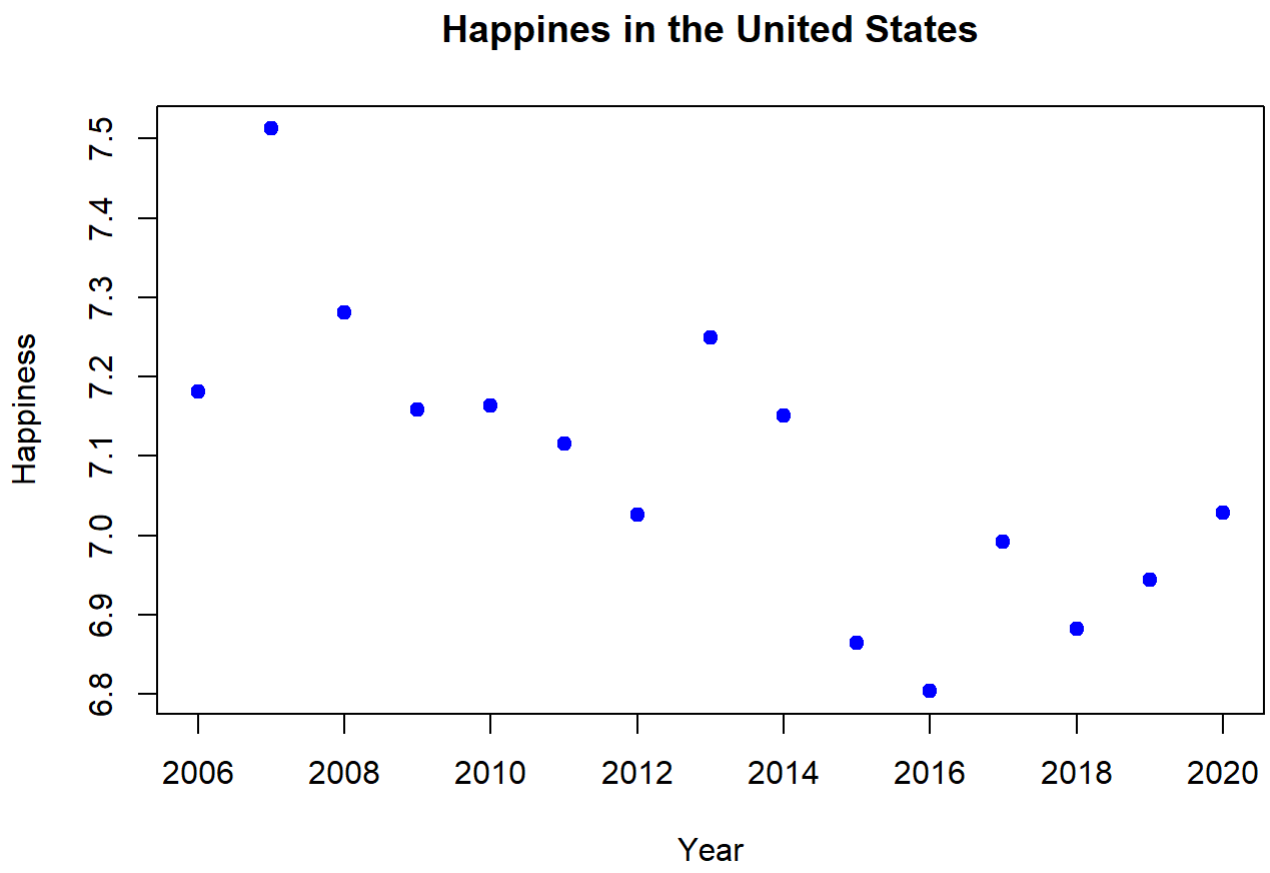
```
##    cbind.data.1.length.data...       V2
## 1                         2006 7.181794
## 2                         2007 7.512688
## 3                         2008 7.280386
## 4                         2009 7.158032
## 5                         2010 7.163616
## 6                         2011 7.115139
## 7                         2012 7.026227
## 8                         2013 7.249285
## 9                         2014 7.151114
## 10                        2015 6.863947
## 11                        2016 6.803600
## 12                        2017 6.991759
## 13                        2018 6.882685
## 14                        2019 6.943701
## 15                        2020 7.028088
```

```
# input_yn2 <- readline(prompt = "Would you like to export your data frame into and Excel file?
 'y' for yes 'n' for no: ")
 input_yn2 <- "n"

if (input_yn2 == "y") {
  print("Please enter the location to save the the Excel file using double back slashes (\\).")
  path <- readline(prompt = "Remember to give your file a different name from the original file:
")
  write_xlsx(data_frame, path)
}
```

An example of what this program can be used for is creating graphs. R is able to make a huge variety of easily customizable graphs, while in my experience, creating and customizing graphs in Excel is tedious. Using the data frame from the code above, a user could create a scatter plot of Happiness in the United States from the World Happiness Report 2021 data set. This is shown below.

```
colnames(data_frame) <- c("Year","Happiness")
plot(data_frame, pch = 19, col = "blue", main = "Happines in the United States")
```



The biggest limitation of this result is that the way I chose to eliminate the tibble attribute means only one row or column can be imported at a time.

3.2 Findings from Web Scraping

After learning about the limitations of readxl, I realized that it was unlikely that I could import specific rows and columns using rvest. Thus, I decided to simply focus on web scraping a data set into R as a data frame. I was able to follow the instructions of the video mentioned in **Documentation** to import the data, but I ran into an error that said, “Error in .[[]] : subscript out of bounds.” This lead me to read more about rvest and html_nodes (12), where I learned that html_nodes has its own R documentation (4) and that it acts similar to indexing, hence the error. I was able to find the right index number using trial and error but I am still learning how the line of code *URL_html %>% html_nodes(“table.data”) %>%. [[2]] %>% html_table* works. An example of how to use the web scraping I learned is shown below.

```
# Dataset used for this example is from: https://www.thearda.com/internationalData/countries/Cou
ntry_41_2.asp

library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.1.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(rvest)
```

```
## Warning: package 'rvest' was built under R version 4.1.3
```

```
URL <- "https://www.thearda.com/internationalData/countries/Country_41_2.asp"

URL_html <- read_html(URL)
data <- URL_html %>% html_nodes("table.data") %>%. [[2]] %>% html_table

print(head(data[,1:3], 11))
```

```
## # A tibble: 11 x 3
##    X1                            X2                          X3
##    <chr>                         <chr>                       <chr>
##  1 "Religious Adherents, (2015)2" Religious Adherents, (2015)2 Canada
##  2 "Christian"                   Christian                   57.17%
##  3 ""                            Catholic                    34.71%
##  4 ""                            Protestant                  18.94%
##  5 ""                            Orthodox                    2.25%
##  6 "Not Religious"               Not Religious               25.8%
##  7 ""                            Unspecified Not Religious   22.87%
##  8 ""                            Atheist                     2.93%
##  9 "Muslim"                      Muslim                      3.56%
## 10 ""                            Sunni                       3.19%
## 11 ""                            Shiite                      0.36%
```

```
new_data <- data[-1,]
names <- unlist(data[1,])
data_frame <- data.frame(new_data)

for (i in 1:length(names)) {
  colnames(data_frame)[i] <- names[i]
}

print(head(data_frame[,1:3], 10))
```

```
##    Religious Adherents, (2015)2 Religious Adherents, (2015)2.1 Canada
## 1                     Christian                       Christian 57.17%
## 2                                                      Catholic 34.71%
## 3                                                    Protestant 18.94%
## 4                                                      Orthodox  2.25%
## 5                 Not Religious                   Not Religious  25.8%
## 6                                    Unspecified Not Religious 22.87%
## 7                                                       Atheist  2.93%
## 8                       Muslim                          Muslim  3.56%
## 9                                                         Sunni  3.19%
## 10                                                        Shiite  0.36%
```

```
numeric_convert <- function(x) {
  y = as.numeric(strsplit(x, "%"))
  return (y)
}

data_frame[3] <- lapply(data_frame[3], numeric_convert)

print(head(data_frame[,1:3],10))
```

```
##    Religious Adherents, (2015)2 Religious Adherents, (2015)2.1 Canada
## 1                     Christian                       Christian  57.17
## 2                                                     Catholic  34.71
## 3                                                   Protestant  18.94
## 4                                                     Orthodox   2.25
## 5                 Not Religious                 Not Religious  25.80
## 6                                   Unspecified Not Religious  22.87
## 7                                                     Atheist   2.93
## 8                       Muslim                        Muslim   3.56
## 9                                                      Sunni   3.19
## 10                                                     Shiite   0.36
```

```
print(sum(data_frame[(data_frame[,1] == "Christian") | (data_frame[,1] == "Muslim"),][,3]))
```

```
## [1] 60.73
```

As can be seen from the code above, one of the most powerful aspects of web scraping data sets using R is that it's possible to change the type of the data. In the original version of the data frame, the third column gives the percentage of religious populations in Canada using the % sign. This makes the type of the data 'character', meaning the values cannot be added together. When I tried to remove the % symbol in Excel, it wanted me to replace the symbol with something else instead of just deleting it (although it was still able to add numbers using the % symbol).

In R, I was able to run the whole column through the small *numeric_convert* function, which removed the % sign using *strsplit*, and saved the values as 'numeric'. From there on, I could use logical sub setting to find the total percentage of the Christian and Muslim population in Canada.

4. Conclusions

Overall, this project was a success. While it didn't turn out exactly as I envisioned it, I was still able to import only the data I needed from a data set. The biggest way that my focus has shifted since beginning this project is realizing that R isn't made to manipulate data in an Excel sheet. It works best when the data is imported into R and put into a data frame.

I was able to put the early stages of my project into practice at UCLA's 2022 Data Fest. At Data Fest, teams are given a huge data set to analyze. This year's data set consisted of 1048576 rows and 'EB' number of columns. My idea was to import only the columns I needed. However, when I tried to use the argument *range = B1:B1048576* in the function *read_excel*, R outputted, "Error: cannot allocate vector of size 4.6 Gb." I tried a smaller argument, *range = B1:B10*, and received the same error. This leads me to believe that the error doesn't have anything to do with the size of the row or column *read_excel* is trying to import, but rather with the size of the whole Excel file. In the end, I was only able to import the some of the data when the file was split into quarters.

This leads me to evaluate the strengths of my code in **Results 3.1**, which are the following:

- Even though it still has limits, my code can make a program more efficient by only importing rows and columns of interest. In some cases this may cause a program to run faster, even though the databases I used to test the code were not big enough to measure the difference.
- This code can be modified to add in columns from different data sets into one big data set. As it stands right now, the program does not account for any user errors and the rows or columns must be the same size, so the user would have to carefully check the dimensions of different data sets.

The weaknesses of my code in **Results 3.1** are the following:

- If the data set is too big for R to evaluate, then this code still cannot import the data. It does not matter how big or small the subset of interest is.
- With the way the code is written right now, only one row or column can be imported at a time. This is very inefficient if a user wants to import a block of code or several rows and columns close to each other.

Despite these weaknesses, I will still use this code in the future. Specifically, I plan on applying it as a skeleton code next academic year in my study of the United Nations World Happiness Report. Using the code I can import columns and rows from different databases and put them together as one. This will be done using country names and while some adjustments will need to be made for rows and columns of different lengths, I now have a bases with which to start that process.

5. Reflection

When I started this project in week 1 of the quarter, I was very excited to immediately start jumping into R Studio and type out as much code as I possibly could. However, as I started researching all the different libraries there were to import Excel files into R, I quickly realized that this was not going to be the case in this project. In fact, I spent most of my time reading documentation, watching YouTube videos on R, and practicing what I was learning. This helped give me a deeper perspective on learning R as a language.

After having taken Stats 20 in the Winter 2022 quarter, I recognized much of the structure and the format of the new functions I learned for this project. However, when I started realizing how many libraries there are for R, I realized just how little I actually knew. Learning a programming language is like learning a spoken language - there are vocabulary words, grammar rules, and conventions for communication. On top of that, there is also a whole world of literature that defines a language and adds to the understanding of it. This was my experience with this project. Even if I can “speak” R at a basic level, there is still a world of background literature and knowledge that I need to understand.

6. References

1. *Build a data frame — tibble.* (n.d.). Tibble.tidyverse.org. https://tibble.tidyverse.org/reference/tibble.html (https://tibble.tidyverse.org/reference/tibble.html)
2. *dplyr package | R Documentation.* (2018, November 19). Rdocumentation.org. https://www.rdocumentation.org/packages/dplyr/versions/0.7.8 (https://www.rdocumentation.org/packages/dplyr/versions/0.7.8)
3. *How to center the title in R Markdown.* (n.d.). Stack Overflow. https://stackoverflow.com/questions/19697402/how-to-center-the-title-in-r-markdown (https://stackoverflow.com/questions/19697402/how-to-center-the-title-in-r-markdown)
4. *html_nodes function - RDocumentation.* (n.d.). Www.rdocumentation.org. https://www.rdocumentation.org/packages/rvest/versions/0.3.6/topics/html_nodes (https://www.rdocumentation.org/packages/rvest/versions/0.3.6/topics/html_nodes) (researched May 26)
5. *MyBib Contributors.* (2019, May 26). APA Citation Generator – FREE & Fast – (6th Edition, 2019). MyBib. https://www.mybib.com/tools/apa-citation-generator (https://www.mybib.com/tools/apa-citation-generator) (How to cite in APA 7)
6. *Problem with loading XLConnect.* (2018, August 28). RStudio Community. https://community.rstudio.com/t/problem-with-loading-xlconnect/13322 (https://community.rstudio.com/t/problem-with-loading-xlconnect/13322)
7. *Read Excel Files.* (n.d.). Readxl.tidyverse.org. Retrieved June 9, 2022, from https://readxl.tidyverse.org/index.html (https://readxl.tidyverse.org/index.html)
8. *Read Excel File in R (6 Examples) | xlsx, xls, read_excel, readxl, openxlsx.* (n.d.). Statistics Globe. https://statisticsglobe.com/r-read-excel-file-xlsx-xls (https://statisticsglobe.com/r-read-excel-file-xlsx-xls)
9. *Reading Data in R From a Website (#R, #RStudio #DataScience #Website #Parse).* (2020, April 3). Www.youtube.com. https://www.youtube.com/watch?v=MMLeKZHNuwA&list=PLUxTge5ZmeHxeBENsNBox6BzT3w-VsltP&index=75&ab_channel=iAnalyticsGeek (https://www.youtube.com/watch?v=MMLeKZHNuwA&list=PLUxTge5ZmeHxeBENsNBox6BzT3w-VsltP&index=75&ab_channel=iAnalyticsGeek)
10. *rstudio/cheatsheets.* (2022, February 19). GitHub. https://github.com/rstudio/cheatsheets/blob/main/data-import.pdf (https://github.com/rstudio/cheatsheets/blob/main/data-import.pdf)
11. *rvest: easy web scraping with R.* (2014, November 24). Www.rstudio.com. https://www.rstudio.com/blog/rvest-easy-web-scraping-with-r/ (https://www.rstudio.com/blog/rvest-easy-web-scraping-with-r/)
12. *rvest package - RDocumentation.* (2021, October 16). Www.rdocumentation.org. https://www.rdocumentation.org/packages/rvest/versions/1.0.2 (https://www.rdocumentation.org/packages/rvest/versions/1.0.2)