Lab 2 - Working with Data Frames

Rafael Campos-Gottardo*

February 6 & 7, 2025

Learning Objectives

1. Load dataframes that are not in the csv format.
2. Summarize data using base R functions.
3. Load the `tidyverse` and use `pipes`.
4. Clean data using the `dplyr` and `stringr` packages.
5. Summarize data using `dplyr`.

Before Lab

1. Read this lab guide.
2. Review downloading packages in R.
3. Read section "3.4 the pipe" of the R for data science textbook. We will be using the pipe (`%>%`) from the `magrittr` package in the `tidyverse`. However, this pipe uses the same logic as the base R pipe (`|>`) discussed in the textbook and can be used interchangeably.
4. Review the `stringr` package documentation and regular expressions.

Working with "Raw Data"

Now that we have learned the basics of R we can start cleaning dataframes and generating descriptive statistics. Let us start by importing the class dataset we created last week. We can download "class_survey_poli311.xlsx" from *mycourses*.

This dataset has 6 variables that are based on the following questions:

Variable	Question
fav_col	What is your favourite colour?
location	Where are you from?
age	How old are you?
interest	How interested are you in politics?
Gender	Whats your gender?
social_use	Approximately how many hours a week do you spend on social media? (please only input numbers)

You will notice that Google forms exports Excel spreadsheets instead of .csv (comma-separated values) files. Most data you will work with in R is not in the .csv format and requires an additional package to import our data into R.

*This lab guide builds on those created by Aaron Erlich and Colin Scott. I would also like to acknowledge Brendan Szendro, Nicholas Vaxelaire, Guila Cohen, and Natalie Delmonte for their help developing this lab guide.

Installing packages

In order to, import Excel files into R we need to install the `readxl` package. To install a package we use the `install.packages()` function with the name of the package in quotation marks `install.packages("readxl")`. This function allows us to download packages from CRAN (The Comprehensive R Archive Network) into the libraries folder on your computer. CRAN includes all officially approved R packages. However, some user created packages have not been officially approved by the R team and these packages can be installed from github. Github packages usually have more "niche" uses and contain less documentation.

To install the `readxl` package you can run the following lines of code.

```
# The install.packages function installs the package on your computer
# install.packages("readxl")

# Use the library function to load packages
library(readxl)

# If you are curious where R install packages you can run the libParths function
.libPaths()
```

```
## [1] "/Users/rafaelc-g/Library/R/arm64/4.4/library"
## [2] "/Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library"
```

When installing packages you only have to run the `install.packages()` function once. It will typically cause an error if you try to run it more than once. Therefore, after you run it you can delete it or comment it out. **However**, you have to run the `library()` function every time you restart R or change R projects. If you get the error `Error in read_xlsx() : could not find function "read_xlsx"` it means that you have not loaded the package yet for this R session.

There are a number of other packages that can be used to load data into R. The `haven` package lets you load SPSS (.sav) files with the `read_sav()` function and STATA (.dta) files with the `read_dta()` function.

Practice Problem 1: Install and load the `haven` package in R.

Loading Data

Now we can load the class dataset using the `read_xlsx()` function.

```
class_df <- read_xlsx("class_survey_poli311.xlsx")
```

Now we can start working with our new dataset. We can use the `head()` function to look at the first 5 rows of our dataframe and the `str()` function to look at the variables.

```
head(class_df)
```

```
## # A tibble: 6 x 6
##   fav_col location                    age interest Gender social_use
##   <chr>   <chr>                     <dbl>    <dbl> <chr>  <chr>
## 1 purple  The United States            19        4 Woman  21
## 2 Red     The United States            19        5 Man    19
## 3 Pink    Another Canadian Province    21        3 Woman  14
## 4 Purple  Another Canadian Province    21        4 Woman  8
## 5 Purple  Another Canadian Province    19        5 Woman  10
## 6 <NA>    Another Canadian Province    20        5 Woman  10
```

```
str(class_df)
```

```
## tibble [45 x 6] (S3: tbl_df/tbl/data.frame)
##  $ fav_col   : chr [1:45] "purple" "Red" "Pink" "Purple" ...
```

```
##  $ location  : chr [1:45] "The United States" "The United States" "Another Canadian Province" "Anothe
##  $ age       : num [1:45] 19 19 21 21 19 20 20 20 20 20 ...
##  $ interest  : num [1:45] 4 5 3 4 5 5 5 5 5 5 ...
##  $ Gender    : chr [1:45] "Woman" "Man" "Woman" "Woman" ...
##  $ social_use: chr [1:45] "21" "19" "14" "8" ...
```

Descriptive Statistics

We are also interested in generating descriptive statistics. One way we can do so is using the `summary()` function. If we use the `summary` function on the whole dataset it will generate the mean, median and quartiles for all the numeric variables in our dataframe.

```
summary(class_df)
```

```
##    fav_col            location             age           interest
##  Length:45          Length:45          Min.   :18.00   Min.   :2.000
##  Class :character   Class :character   1st Qu.:19.00   1st Qu.:4.000
##  Mode  :character   Mode  :character   Median :20.00   Median :5.000
##                                        Mean   :19.98   Mean   :4.545
##                                        3rd Qu.:20.00   3rd Qu.:5.000
##                                        Max.   :26.00   Max.   :5.000
##                                        NA's   :1       NA's   :1
##     Gender           social_use
##  Length:45          Length:45
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
##
```

We can also generate the mean, median, and standard deviation using individual functions.

We can start by looking at the age variable. Remember we can use the `$` operator to extract specific variables from a data frame.

```
mean(class_df$age)
```

```
## [1] NA
```

```
median(class_df$age)
```

```
## [1] NA
```

```
sd(class_df$age)
```

```
## [1] NA
```

Why do we get NA for the mean, median, and standard deviation?

We can use the unique function to explore our variables and see what is causing this error. The `unique()` function displays all the unique values of a variable.

```
unique(class_df$age)
```

```
## [1] 19 21 20 18 22 23 26 NA
```

Practice Problem 2: Please only try this problem if you want more practice with more advanced features in R. Try to run the unique function on each variable using a loop.

As we can see there are NA values in the age variable. Therefore, we need to add an additional argument to the `mean()`, `median()`, and `sd()` functions to indicate that we want the function to ignore the `NA` values. By default the `na.rm` argument equals `FALSE`.[1] We need to specify `na.rm = TRUE`. Now the `mean()`, `median()`, and `sd()` functions will generate the correct values.

```
mean(class_df$age)
```

```
## [1] NA
```

```
median(class_df$age)
```

```
## [1] NA
```

```
sd(class_df$age)
```

```
## [1] NA
```

The Tidyverse

Now that we have generated some basic descriptive statistics we can clean our dataset more using the tidyverse suite of packages. These packages provide very useful tools for data science in R and share a common "grammer." We can load the `tidyverse` the same way as the `readxl` package.

```
#install.packages("tidyverse")
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

One of the most useful features of the `tidyverse` is the pipe (`%>%`) from the `magrittr` package. The pipe lets you insert the previous line into the first argument of the function on next line of code. The most common usage of the pipe is outlined below. One of the most useful features of the pipe is that it improves the readability of your code and lets you easily use multiple functions at once.

```
# Without the pipe

other_fuction(data, function(arguement2))

# With the pipe

data %>%
  function() %>%
  other_function(arguement2)
```

If you are still unsure about the pipe we will discuss the pipe further as we discuss the `tidyverse`.

[1] If we look at the help file for the mean function (`?mean`), we can see the default options for this function `## Default S3 method: mean(x, trim = 0, na.rm = FALSE, ...)`

Cleaning Data

Now that we have imported our data we can clean it using functions from the dplyr package in the tidyverse. Dplyr uses the grammar of the tidyverse and contains a number of functions that can be used for data manipulation. We will go over the main dplyr functions one by one to clean our data set.

Mutate

The mutate() function adds new variables to a data frame that are a function of existing variables. The new variable will either be added as a new row in your data frame or replace an existing variable if you use the same name. Let us use the mutate function to fix the social media usage variable. You will notice that some respondents included words in their response to the social media usage question.

We want to make this variable numeric so we can find the average number of hours 210 students spend on social media. In order to, extract just the numbers from the responses we use the mutate() function and str_extract() function from the stringr package. This function allows us to extract information from string (character) variables that match a certain pattern. The pattern we want to match is "\\d+" which is the first occurrence of a digit. The \\d is the regex (regular expression) short hand for digits and the + indicates that you want R to match one or more of the preceding character.

```
# Let's pipe in our dataset and assign back into our dataset to save it
class_df <- class_df %>%
# Let's use a new name to not override the old variable
  mutate(social_use_num = str_extract(social_use, "\\d+"),
# Now we need to transform it into a numeric variable
# Let's override the previous version
         social_use_num = as.numeric(social_use_num))
```

Summarize

Now we can use the summarize function in dplyr to create a new data frame of summary information. We want the min, max, mean, median, and standard deviation.

```
class_df %>%
  summarize(Mean = mean(social_use_num, na.rm = TRUE),
            Median = median(social_use_num, na.rm = TRUE),
            `Standard Deviation` = sd(social_use_num, na.rm = TRUE),
            Min = min(social_use_num, na.rm = TRUE),
            Max = max(social_use_num, na.rm = TRUE))
```

```
## # A tibble: 1 x 5
##    Mean Median `Standard Deviation`   Min   Max
##   <dbl>  <dbl>                <dbl> <dbl> <dbl>
## 1  9.91    8.5                 6.16     2    35
```

Now we have produced a nice summary table using the summarize() function. We can also breakdown our descriptive statistics by gender using the group_by() function. We follow the same syntax but add a separate line using the pipe that includes the group_by() function.

```
class_df %>%
  group_by(Gender) %>%
  summarize(Mean = mean(social_use_num, na.rm = TRUE),
            Median = median(social_use_num, na.rm = TRUE),
            `Standard Deviation` = sd(social_use_num, na.rm = TRUE),
            Min = min(social_use_num, na.rm = TRUE),
            Max = max(social_use_num, na.rm = TRUE))
```

```
## # A tibble: 3 x 6
```

```
##   Gender  Mean Median `Standard Deviation`   Min   Max
##   <chr>  <dbl>  <dbl>                <dbl> <dbl> <dbl>
## 1 Man    12.9      10                 9.77     3    35
## 2 Woman   9.34      8                 4.39     3    21
## 3 <NA>    4         4                 2.83     2     6
```

In future weeks we will discuss how to turn these tables into markdown or latex tables that can be used in your assignments and research papers.

Select

The next important `dplyr` function is the `select()` function. This function picks out variables from your data frame using their names. For example if we want to create a new data frame that does not include the "favourite colour" variable we can do so using the `select()` function.

```
class_df_no_col <- class_df %>%
  select(-fav_col) # the - indicates that we want to exclude that variable

head(class_df_no_col)
```

```
## # A tibble: 6 x 6
##   location                    age interest Gender social_use social_use_num
##   <chr>                     <dbl>    <dbl> <chr>  <chr>               <dbl>
## 1 The United States            19        4 Woman  21                     21
## 2 The United States            19        5 Man    19                     19
## 3 Another Canadian Province    21        3 Woman  14                     14
## 4 Another Canadian Province    21        4 Woman  8                       8
## 5 Another Canadian Province    19        5 Woman  10                     10
## 6 Another Canadian Province    20        5 Woman  10                     10
```

```
# This code only creates a data frame that only includes the favourite colour variable
class_df_col <- class_df %>%
  select(fav_col)

head(class_df_col)
```

```
## # A tibble: 6 x 1
##   fav_col
##   <chr>
## 1 purple
## 2 Red
## 3 Pink
## 4 Purple
## 5 Purple
## 6 <NA>
```

Practice Problem 3: What could have caused this error: `Error in select()! Can't select columns that don't exist. Column fav_col doesn't exist.`?

Filter

Finally, we want to use the `filter()` function to subset the data. This function selects rows that match a certain condition. For example, if we want a subset of our data frame that only includes the individuals in our class who are the most interested in politics (5) we can use the following code.

```
most_interested_df <- class_df %>%
  filter(interest == 5)
```

Notice that we use == instead of = to test for equality (whether one value equals another). We use two equal signs because one equals sign is another assignment operator that works the same as the arrow we usually use.

Back to pipes

Now is a good time to show you the usefulness of the pipe. We can perform all the cleaning code we just did in one go with and without the pipe.

```
# With the pipe
class_df %>%
  mutate(social_use_num = str_extract(social_use, "\\d+"),
         social_use_num = as.numeric(social_use_num)) %>%
    select(-fav_col) %>%
  filter(interest == 5) %>%
    group_by(Gender) %>%
  summarize(Mean = mean(social_use_num, na.rm = TRUE),
            Median = median(social_use_num, na.rm = TRUE),
            `Standard Deviation` = sd(social_use_num, na.rm = TRUE),
            Min = min(social_use_num, na.rm = TRUE),
            Max = max(social_use_num, na.rm = TRUE))
```

```
## # A tibble: 3 x 6
##   Gender  Mean Median `Standard Deviation`   Min   Max
##   <chr>  <dbl>  <dbl>                <dbl> <dbl> <dbl>
## 1 Man    13.1      10                10.2      3    35
## 2 Woman   8.16      8                 3.96     3    21
## 3 <NA>    6         6                NA        6     6
```

```
# Without the pipe
  summarize(
    group_by(
    filter(
    select(
      mutate(
        class_df,
        social_use_num = str_extract(social_use, "\\d+"),
         social_use_num = as.numeric(social_use_num)),
      -fav_col),
    interest == 5),
    Gender),
            Mean = mean(social_use_num, na.rm = TRUE),
            Median = median(social_use_num, na.rm = TRUE),
            `Standard Deviation` = sd(social_use_num, na.rm = TRUE),
            Min = min(social_use_num, na.rm = TRUE),
            Max = max(social_use_num, na.rm = TRUE))
```

```
## # A tibble: 3 x 6
##   Gender  Mean Median `Standard Deviation`   Min   Max
##   <chr>  <dbl>  <dbl>                <dbl> <dbl> <dbl>
## 1 Man    13.1      10                10.2      3    35
## 2 Woman   8.16      8                 3.96     3    21
## 3 <NA>    6         6                NA        6     6
```

Notice how the code that does not use the pipe is more confusing than the code that uses the pipe (even with the hard returns). The code without the pipe is also harder to write because it is less intuitive. Without the pipe you have to put the last function first instead of the first function first. The pipe is also useful because

it allows you to separate each function with a line and place the functions in order.

Practice Problem Answers

Practice Problem 1:

```
# install.packages("haven") #Uncomment this line to load the package

library(haven)
```

Practice Problem 2:

```
variables <- names(class_df)

for(i in 1:length(variables)){
  print(unique(class_df[, variables[i]]))
}
```

```
## # A tibble: 20 x 1
##    fav_col
##    <chr>
##  1 purple
##  2 Red
##  3 Pink
##  4 Purple
##  5 <NA>
##  6 Green
##  7 pink
##  8 navy blue
##  9 Blue
## 10 Navy Blue
## 11 Navy blue
## 12 blue
## 13 Yellow
## 14 yellow
## 15 Burgundy
## 16 green
## 17 Camel, black
## 18 Cyan
## 19 Forest Green
## 20 Hot Pink
## # A tibble: 6 x 1
##   location
##   <chr>
## 1 The United States
## 2 Another Canadian Province
## 3 Quebec
## 4 Europe
## 5 Australia
## 6 <NA>
## # A tibble: 8 x 1
##     age
##   <dbl>
## 1    19
## 2    21
## 3    20
```

```
## 4    18
## 5    22
## 6    23
## 7    26
## 8    NA
## # A tibble: 5 x 1
##   interest
##      <dbl>
## 1        4
## 2        5
## 3        3
## 4        2
## 5       NA
## # A tibble: 3 x 1
##   Gender
##   <chr>
## 1 Woman
## 2 Man
## 3 <NA>
## # A tibble: 21 x 1
##    social_use
##    <chr>
##  1 21
##  2 19
##  3 14
##  4 8
##  5 10
##  6 3
##  7 35
##  8 5
##  9 7
## 10 20
## # i 11 more rows
## # A tibble: 17 x 1
##    social_use_num
##             <dbl>
##  1             21
##  2             19
##  3             14
##  4              8
##  5             10
##  6              3
##  7             35
##  8              5
##  9              7
## 10             20
## 11             16
## 12              6
## 13             15
## 14              4
## 15              9
## 16             NA
## 17              2
```

Practice Problem 3:

This error occurs when you remove a variable from a dataset and then try to select that variable. For example:

```
class_df_no_col <- class_df %>%
  select(-fav_col) # the - indicates that we want to exclude that variable

head(class_df_no_col)

# This code only creates a data frame that only includes the favourite colour variable
 class_df_no_col %>%
  select(fav_col)
```

This code will cause that error to occur.