<script type="text/javascript">

// toggle visibility of R source blocks in R Markdown output function toggle$_R()varx = document.getElementsByClassName('r'); if(x.length == 0)return; functiontoggle_vis(o)vard =$

for (i = 0; i < x.length; i++) var y = x[i]; if (y.tagName.toLowerCase() === 'pre') toggle$_vis(y)$;

var elem = document.getElementById("myButton1"); if (elem.value === "Hide Global") elem.value = "Show Global"; else elem.value = "Hide Global";

document.write('<input onclick="toggle$_R(); "type = "button"value = "HideGlobal"id = "myButton1"style = "position : absolute; top : 10$

</script>

# Introduction to R for health data analysis

Ehsan Karim, An Hoang and Yang Qu

2021-07-25

# Contents

# Preface

This is a R tutorial for those who are not familiar with data wrangling. For providing some practical introduction to data wrangling, NHANES datasets will be used as examples in this tutorial.

## Main references

- Overall reference **?**

## Version history

This tutorial was initially created by a team supported by worklearn program in 2021 May-August (during Covid-19 pandemic). Initial team members included An Hoang and Yang Qu, working under the supervision of Ehsan Karim.

## Contributor list

- An Hoang (forestry, UBC)
- Yang Qu (statistics, UBC)

### Prerequisites

None.

### Comments

For any comments regarding this document, reach out to me.

# Chapter 1

# R and RStudio set up

## 1.1 INSTRUCTIONS

This tutorial is aiming to introduce you to R and RStudio. It will guide you to download and install R and RStudio and walk you through the main components in RStudio. Follow this tutorial step-by-step and finish setting up R and RStudio before the next tutorial.

Accompanying this tutorial is **a short Google quiz** for your own self-assessment. The instructions of this tutorial will clearly indicate when you should answer which question.

## 1.2 LEARNING OBJECTIVES

- Understand the difference between R and RStudio
- Download and install R
- Download and install RStudio
- Be familiar with the main components in RStudio

## 1.3 1. What is R

R is a language and enviroment for statistical computing and graphics. It is commonly used in borh academia and industry.

It is:

- Free and open source
- Easy to learn and use

- Good compatibility - can be used in Windows, macOS, and Linux

For more information about R, check out this website.

**DO QUESTION 1 OF THE QUIZ NOW** > True of False: R is a programming language

## 1.4 2. What is RStudio

RStudio is not a language - is an IDE (Integrated Development Environment) for R. It hae two versions available: RStudio desktop and RStudio Server. We will use RStudio desktop in this tutorial.

There are lots of available IDEs for R. The reason why we chooes RStudio is that it has a fancy GUI and required features that makes working with R much easier and more efficient.

**DO QUESTION 2 OF THE QUIZ NOW** > True of False: RStudio is a programming language which is similar to R

## 1.5 3. Download and install R

- Go to https://www.r-project.org/ and click on **download R**
- Choose CRAN location based on your geological location
- Download R based on your operating system and choose the latest release version (it is Python 3.9.5 for now)
- Open the downloaded package and follow the instruction there to finish the installation

## 1.6 4. Download and install RStudio

**Make sure you downloaded and installed R before doing the following steps**

- Go to https://www.rstudio.com/products/rstudio/download/ and download the RStudio Desktop
- Open the downloaded file and follow the instruction there to finish the installation

## 1.7 5. RStudio basics

RStudio have 4 main components:

- Script (top left)

https://i.imgur.com/TKTocvu.png

Figure 1.1: image

  - the Script is where you write the R code
  - you can save the script as a `.R` file
- Console (bottom left)
  - the console is where the R code being executed
  - Output (except graphs and plots) will be shown after code executed
- Workspace (top right)
  - all objects in the current working environments including variables, data, and functions are listed here with a brief display of their corresponding values.
  - you can import other workspaces, save the current workspace, and clean up the current workspace
  - R workspace file ends with `.RData`
- Files, Plots, Packages, Help (bottom right)
  - Files is the place to view the Files and to set Working Directory
  - Plots gives a preview of plot - it is the place where graphical output will be displayed
  - Packages is the place to install/view/update packages
  - Help is the place to get help about R

**DO QUESTION 3 OF THE QUIZ NOW** > In RStudio, where do you write your R code if you don't want it be saved?

**DO QUESTION 4 OF THE QUIZ NOW** > In RStudio, where do you write your R code if you do want it be saved?

**DO QUESTION 5 OF THE QUIZ NOW** > What type is the saved R code file?

**DO QUESTION 6 OF THE QUIZ NOW** > In RStudio, where is your R code executed?

**DO QUESTION 7 OF THE QUIZ NOW** > In RStudio, where is numerical output displayed?

**DO QUESTION 8 OF THE QUIZ NOW** > In RStudio, where is graphical output displayed?

**DO QUESTION 9 OF THE QUIZ NOW** > In RStudio, where are variables, data, functions stored?

**DO QUESTION 10 OF THE QUIZ NOW** > What type is the workspace file?

**DO QUESTION 11 OF THE QUIZ NOW** > Is there a way to find the details about functions and packages in RStudio?

## 1.8 TAKEAWAYS

By the end of this tutorial, you should be able to set up R and RStudio successfully. Please feel free to reach out if you have any issues with the set ups.

Before we proceed to the next tutorial, make sure that you're familiar with the RStudio GUI and features.

# Chapter 2

# Introduction to R

## 2.1 INSTRUCTIONS

This tutorial will introduce you to the basics of the language of R. We will cover how to set up our working environment, mathematical and logical operators, the most common data types in R, explore a simple dataset, write an R function as well as how to seek for help within and outside of R.

Accompanying this tutorial is **a short Google quiz** for your own self-assessment. The instructions of this tutorial will clearly indicate when you should answer which question.

## 2.2 LEARNING OBJECTIVES

- Be familiar with the basic procedures for setting up an R session with functions such as `getwd()`, `setwd()`, `dir()`, `install.packages()`, and `library()`.
- Understand the very basic of how and when to use arithmetic and logical operators in R.
- Be familiar with the most common types of data in R including string, vector, data frame, and list.
- Explore a dataset using basic Base R functions.
- Know how to write a new function in R.
- Be comfortable with and know how to seek for help within and outside of R.

## 2.3 1. SET UP BASICS

### 2.3.1 Working Directory

One of the most important function in R is `getwd()`, or "get working directory". The output of this code is the pathway of your current R file. Interestingly, `getwd()` does not have any argument. In other words, you do not have to type anything in the `()`.

It is highly recommended that all of your files (the R file, any data files, images, etc) be in the same directory. This will make your project much more organized and your life a lot easier when we get into more complicated data analysis that involves more data files.

By default, your working directory is whatever folder your current R file is in.

```
getwd()
```

```
## [1] "C:/Users/ehsan/Documents/GitHub/intro2R"
```

If at any point, you want to change your working directory to another folder, you can use `setwd()`. Different from `getwd()`, `setwd()` requires an argument within its brackets. To set a new working directory, you need to copy and paste the pathway within the brackets and in quotation marks ("").

This code is helpful when you need to pull files outside of the default working directory. However, you should be mindful when using this function because it gets very confusing very quickly.

```
#setwd("")
```

Another important function is `dir()`. This function lets you check all of the files that exist in your working directory.

This function is a good option if you want to check if there are any extra or missing files from your working directory.

```
dir()
```

```
##  [1] "_book"
##  [2] "_bookdown.yml"
##  [3] "_bookdown_files"
##  [4] "_build.sh"
##  [5] "_deploy.sh"
##  [6] "_output.yml"
##  [7] "0-r-and-rstudio-set-up.Rmd"
##  [8] "1-introduction-to-r.Rmd"
##  [9] "2-importing-data-into-r-with-readr.Rmd"
## [10] "3-introduction-to-nhanes.Rmd"
## [11] "4-data-analysis-with-dplyr.Rmd"
## [12] "5-data-visualization-with-ggplot.Rmd"
```

```
## [13] "6-date-time-data-with-lubridate.Rmd"
## [14] "7-data-summary-with-tableone.Rmd"
## [15] "8-Exercise-Solutions.Rmd"
## [16] "9-references.Rmd"
## [17] "book.bib"
## [18] "data"
## [19] "DESCRIPTION"
## [20] "Dockerfile"
## [21] "docs"
## [22] "header.html"
## [23] "images"
## [24] "index.Rmd"
## [25] "intro2R.Rmd"
## [26] "intro2R_cache"
## [27] "intro2R_files"
## [28] "LICENSE"
## [29] "now.json"
## [30] "packages.bib"
## [31] "preamble.tex"
## [32] "R.Rproj"
## [33] "README.md"
## [34] "style.css"
## [35] "toc.css"
```

You will see that there are 2 CSV files in our working directory: last_15_bpx.csv and last_15_demo.csv. Do not worry about what they are right now (we will cover this in later tutorials). All you have to know for now is that these two files are currently residing in our input/tutorial-demo folder, AKA our working directory.

The example above is only to demonstrate how we would change our working directory. But since we want to remain in our default working directory for the rest of this tutorial, we will set our working directory back to the original directory.

```
# setwd("..") # this ".." argument allows us to move back 1 folder
# setwd("..")
# setwd("..") # so doing this three times means that we are moving back 3 folders


# setwd("./data/")
# now we're back in the large folder that houses both input/tutorial-demo and kaggle/working
```

After setting a new working directory, it is in our best interest to check the working directory again to see if we are in the right place.

```
# check to see if we're back to our original directory
getwd()
```

```
## [1] "C:/Users/ehsan/Documents/GitHub/intro2R"
```

#### 2.3.1.1 Functions Debunked

Throughout our tutorials, you will see a recurring section named **Functions Debunked**. These sections aim to break down the function that you were just introduced to. Each of these section will include a link for you to find more information about the function, the different arguments that can be nested within each function, and an example.

For this first section, we will debunk `#`. When you see a `#` in a code chunk, this means that the following information is a note or comment. In other words, it doesn't code for anything - it is just notes explaining what we are doing. You can try adding a `#` in front of our `getwd()` code above to see what happens!

If this doesn't make any sense to you right now, do not worry! It will make more sense as we move along the tutorials.

### 2.3.2 Installing and Attaching Packages

Now that we understand what working directories are, we can move onto installing and attaching packages.

There are a lot of packages on R, each has its own set of functions for different purposes. To access each set of function, we need to install the respective package. To install a package, we use `install.packages()`. Within the brackets, the only argument you need is the name of the package in quotation marks ("").

The most basic package on R is Base R. We do not actually need to install this package as it should be built into R by default. Therefore, by default, we should already have access to a range of basic R functions without having to install any packages. In this tutorial, we will only be using functions in this Base R package. However, in future tutorials, we will need to install pacakages such as dplyr and ggplot, which will give you access to even more and more advanced functions.

For the sake of demonstration, the ggplot2 package is installed below, but note that we will not be using any ggplot functions in this tutorial.

```
# install.packages("ggplot2")
```

A related function is `library()`. This function is used to attach installed packages to your R session. Unlike `install.packages()` where you only need to use once, `library()` needs to be run every R session. In other words, you need to attach whatever package you need everytime to you and then reopen R.

In future tutorials, if the `library()` function does not work for you, it is most likely because you have not installed the package, and therefore need to use `install.packages()` first before `library()`.

```
# another difference from install.packages() is that we do not need "" in library()
library(ggplot2)
```

To make sure the package is successfully attached, we can try running a function in that package. After running the code below, you should only see a blank square. This is correct! We will go over why this is in tutorial 5.

```
ggplot()
```

intro2R_files/figure-latex/unnamed-chunk-8-1.pdf

#### 2.3.2.1 DO QUESTIONS 1-3 OF THE GOOGLE QUIZ NOW

> What is a "Working Directory"?
>
> What is the main difference between `setwd()` and `getwd()`?
>
> We need to install the packages first before we can load them using `library()`. (True or False)

## 2.4 2. ARITHMETIC OPERATORS

As expected from a data analysis software, you can use R like a calcultor using arithmetic operators! Here is a list of a few basic and common arithmetic operators in R:

https://i.imgur.com/1ww25MM.png

Figure 2.1: Figure 1. Arithmetic Operators in R

These operators will prove themselves to be more useful in data analysis when we get to later tutorials, especially our Tutorial 4 on the dplyr package.

### 2.4.1 1.1 Try it yourself

a. Can you replicate and solve these problems in R?

- 2ˆ2
- $2 \times 2$
- $2 + 5 \times (5 \div 4)$ˆ6
- what is the remainder of $52 \div 5$
- what is the whole number solution to $82 \div 8$

b. Can you solve for x using R?

a <- 9 + 3 * 6

x <- a $\div$ 2

#### 2.4.1.1 DO QUESTION 4 OF THE QUIZ NOW

The output of `10 %% 2` is equal to which of the following?

## 2.5 3. LOGICAL OPERATORS

In addition, logical operators are also available on R. The main difference between arithmetic operators and logical operators is that logical operators will yield a logical (or TRUE/FALSE) output. This is a list of some common logical operators that can be used on R:

https://i.imgur.com/xG4F5kA.png

Figure 2.2: Figure 2. Logical Operators in R

Similarly, these operators will be more useful when we learn about filtering data in future tutorials and you will also be provided with more examples then.

Here are a few example codes that you can try running:

```
a <- 5 > 4
    # the <- indicates that the information 5 > 4 is stored in the variable a - we wil
```

```
b <- 8
b != 8
```

```
## [1] FALSE
```

```
a == b
```

```
## [1] FALSE
```

```
9 + 10 * 15 - 8 <= 103
```

```
## [1] FALSE
```

```
11+ 3^9 == 19694
```

```
## [1] TRUE
```

```
    # note that in R, "equal to" is coded by ==, = has another meaning that you will see in secti
```

### 2.5.1 1.2 Try it yourself

Translate the following into R and find the output: * 8 times 3 is greater than 8? * eleven divided by seven is not equal to 2? * 9 is less than or equal to 18?

#### 2.5.1.1 DO QUESTION 5 OF THE QUIZ NOW

Which of the following operators code for "equal to"?

## 2.6 4. MOST COMMON DATA TYPES IN R

### 2.6.1 Strings

Strings are either single character or a collection of characters. Note that all strings are in `""`. For example:

```
"hello, my name is Alex"
```

```
## [1] "hello, my name is Alex"
```

```
"Where are you?"
```

```
## [1] "Where are you?"
```

```
"I like to eat 6 apples"
```

```
## [1] "I like to eat 6 apples"
```

### 2.6.2 Vectors

Vector is the simplest data type in R. It is basically a list of components stored in the same place (or variable). To write a vector, starts with `c()` and input the appropriate components within the brackets. For example:

```
c(1, 2, 3) # numeric
```

```
## [1] 1 2 3
```

```
c("hello", "bonjour", "ciao") # character - note that text needs to be in ""
```

```
## [1] "hello"   "bonjour" "ciao"
```

```
c(TRUE, FALSE, TRUE) # logical - we will cover this in more detail later
```

```
## [1]  TRUE FALSE  TRUE
```

```
c(1, "hello", TRUE) # mixed
```

```
## [1] "1"     "hello" "TRUE"
```

We can also store these vectors using the symbol `<-`.

**Note:** If you are using RStudio, they will be stored in our environment located in the top right window.

```
numeric <- c(1, 2, 3, 10:12) # 10:12 means 10, 11, 12!
```

```
character <- c("hello", "bonjour", "ciao")
```

```
logical <- c(TRUE, FALSE, TRUE)
```

What we just did was assigning values to variables where numeric, character, and logical are all variables!

### 2.6.3 1.3 Try it yourself

Can you try storing a string? Assign the string "hello world, I am here" to the variable named start.

**Note how the string is in "" but the variable name is not. Why do you think this is?**

After assigning values to our variables, we can tell R to retrieve them by typing any of these variable names. R will give us the components of data that we assigned to each variable as the output.

```
numeric
```

```
## [1]  1  2  3 10 11 12
```

```
character
```

```
## [1] "hello"   "bonjour" "ciao"
```

```
logical
```

```
## [1]  TRUE FALSE  TRUE
```

If you want to extract a particular component from a variable, you can use `[]`. For example:

```
numeric[1:4] # the first 4 components
```

```
## [1]  1  2  3 10
```

```
character[2] # only the 2nd component
```

```
## [1] "bonjour"
```

### 2.6.4 1.4 Try it yourself

It is important to note that **R is case-sensitive**. This means that it distinguishes capitalized from non-capitalized characters, so logical and Logical are read as two separate things by R!

Try typing Logical with a capitalized "L". How does R respond to this?

We can also replace `<-` with `=` when assigning values to variables. But `=` has other uses as well - you will be introduced to their slight differences in future tutorials. Also, note that in R, `=` does not mean "equal to". As you have see in the previous section, "equal to" is coded by `==`.

### 2.6.5 Lists

A list is a collection of possibly unrelated components. It allows you to gather different types of data into one place. In the code below, we have numbers, characters, and data frame all in one place.

```
list <- list(numeric = 1, character = c("bonjour", "hello"), "I like to eat 6 apples")
```

Similarly, you can also extract specific information from this list using `[]`. Note that in the code below, the output is "bonjour" AND "hello", this is because the second component of our list is a vector that houses both of these words.

```
list[2]
```

```
## $character
## [1] "bonjour" "hello"
```

#### 2.6.5.1 Functions Debunked

The arguments for list() are as follows:

list( > **VARIABLE NAME OF ANY DATA TYPE** = **ANY DATA STORED WITHIN THAT VARIABLE** OR **ANY STRING**

)

**For example:** `list(numeric = 1, character = c("bonjour", "hello"), dataframe = logical)`

### 2.6.6 Dataframe

Dataframe, you guessed it, stores your data in the form of a dataframe or a table! Dataframe allows you to store multiple vectors into one single table.

```
dataframe <- data.frame(numeric, character, logical)
```

```
dataframe
```

```
##   numeric character logical
## 1       1     hello    TRUE
## 2       2   bonjour   FALSE
## 3       3      ciao    TRUE
## 4      10     hello    TRUE
## 5      11   bonjour   FALSE
## 6      12      ciao    TRUE
```

#### 2.6.6.1 Functions debunked

The arguments for data.frame() are as follows:

data.frame( > **VECTOR 1**

**VECTOR 2**

**VECTOR n**

)

**For example:** `dataframe <- data.frame(numeric, character, logical)`

If you want to change the column names of your data frame, you can use the function `names()`.

```
names(dataframe) <- c("Number", "Text", "T/F")
```

Now if we check our data frame again, the new column names should appear.

```
dataframe
```

```
##   Number    Text   T/F
## 1      1   hello  TRUE
## 2      2 bonjour FALSE
## 3      3    ciao  TRUE
## 4     10   hello  TRUE
## 5     11 bonjour FALSE
## 6     12    ciao  TRUE
```

#### 2.6.6.2 Functions Debunked

The arguments for names() are as follows:

names( > **NAME OF DATASET**) <-

**A VECTOR OF COMPONENTS WHOSE LENGTH MATCHES WITH THAT OF THE DATASET** (we will go over length in section 5 of this tutorial)

**For example:** `names(dataframe) <- c("Number", "Text", "T/F")`

### 2.6.7 1.5 Try it yourself

Why do you think numeric, character, and logical are not in "" but Number, Text, and T/F are?

Similar to how we extracted information from vectors and lists, we can also use `[]` to extract certain rows, columns, or cells in a data frame.

```
dataframe[1, ] # only fhe first row
```

```
##   Number  Text  T/F
## 1      1 hello TRUE
```

```
dataframe[, 2] # only the second column
```

```
## [1] "hello"   "bonjour" "ciao"    "hello"   "bonjour" "ciao"
```

```
dataframe[3, 2] # only cell (3,2) - the third row and second column
```

```
## [1] "ciao"
```

#### 2.6.7.1 DO QUESTIONS 6 & 7 OF THE QUIZ NOW

> Which of the following codes would extract only rows 1, 3, 6 and only column 1 from our data frame?
>
> What is the value of the cell in the first row and third column of our data frame?

We can also add a new column to our data frame using the function `cbind()` like below. Note how the column name is in `""`. It is also important that the new column has the same number of values as the rest of the columns. If the new column contains less values than the other columns, you can use `NA`, or "not available" values, to fill up the rest of the places!

```
new_column <- cbind(dataframe, "new column" = c(2, 3, 4, 5, 1, NA))
    # we will learn more about NA values in tutorial 4
```

Similarly, the function to add a new row is `rbind()`. The two functions work almost identical, but `rbind()` does not require a row name.

```
(new_row <- rbind(new_column, c(13, "hello", FALSE, NA)))
```

```
##   Number    Text   T/F new column
## 1      1   hello  TRUE          2
## 2      2 bonjour FALSE          3
## 3      3    ciao  TRUE          4
## 4     10   hello  TRUE          5
## 5     11 bonjour FALSE          1
```

```
## 6      12    ciao  TRUE       <NA>
## 7      13   hello FALSE       <NA>
```

You may have noticed that the code above has an extra `()` that encompasses the whole code. This `()` is another way for us to print the output of our function - it is equivalent to if we just run the name of data frame new_row. Try removing the extra `()` and see what happens!

#### 2.6.7.2 Functions Debunked

**cbind()** is used to create new columns in a data frame. The arguments are as follows: cbind( > **THE CURRENT DATAFRAME**

> "**NAME OF THE NEW COLUMN**" = **VALUE(S) IN THE NEW COLUMN**

)

**rbind** is used to create new rows in a data frame. The arguments are as follows: rbind( > **THE CURRENT DATAFRAME**

> **VALUES IN THE NEW ROW**

)

There exists many other types of data types on R, you are free to explore them on your own time. But what we have been introduced to are the most basic ones.

## 2.7 5. EXPLORING A DATASET

Now that you are familiar with the different data types and operators of R, we can move on to the fun parts of this tutorial: exploring a dataset!

To introduce you to the concept of exploring data on R, we will be using a dataset already available on R - in other words, we will not be importing data into R yet, we will cover this in another tutorial. Conveniently, R has a set of built-in datasets that we can use to practice using basic R functions. In this tutorial, we will use the dataset named "faithful" which contains information on the Old Faithful Geyser in Yellowstone National Park. Run the codes below to explore the dataset.

```
# information on the data
# ?faithful
```

```
# the actual data
faithful
```

```
##     eruptions waiting
## 1       3.600      79
## 2       1.800      54
```

```
## 3        3.333      74
## 4        2.283      62
## 5        4.533      85
## 6        2.883      55
## 7        4.700      88
## 8        3.600      85
## 9        1.950      51
## 10       4.350      85
## 11       1.833      54
## 12       3.917      84
## 13       4.200      78
## 14       1.750      47
## 15       4.700      83
## 16       2.167      52
## 17       1.750      62
## 18       4.800      84
## 19       1.600      52
## 20       4.250      79
## 21       1.800      51
## 22       1.750      47
## 23       3.450      78
## 24       3.067      69
## 25       4.533      74
## 26       3.600      83
## 27       1.967      55
## 28       4.083      76
## 29       3.850      78
## 30       4.433      79
## 31       4.300      73
## 32       4.467      77
## 33       3.367      66
## 34       4.033      80
## 35       3.833      74
## 36       2.017      52
## 37       1.867      48
## 38       4.833      80
## 39       1.833      59
## 40       4.783      90
## 41       4.350      80
## 42       1.883      58
## 43       4.567      84
## 44       1.750      58
## 45       4.533      73
## 46       3.317      83
## 47       3.833      64
## 48       2.100      53
```

```
## 49       4.633      82
## 50       2.000      59
## 51       4.800      75
## 52       4.716      90
## 53       1.833      54
## 54       4.833      80
## 55       1.733      54
## 56       4.883      83
## 57       3.717      71
## 58       1.667      64
## 59       4.567      77
## 60       4.317      81
## 61       2.233      59
## 62       4.500      84
## 63       1.750      48
## 64       4.800      82
## 65       1.817      60
## 66       4.400      92
## 67       4.167      78
## 68       4.700      78
## 69       2.067      65
## 70       4.700      73
## 71       4.033      82
## 72       1.967      56
## 73       4.500      79
## 74       4.000      71
## 75       1.983      62
## 76       5.067      76
## 77       2.017      60
## 78       4.567      78
## 79       3.883      76
## 80       3.600      83
## 81       4.133      75
## 82       4.333      82
## 83       4.100      70
## 84       2.633      65
## 85       4.067      73
## 86       4.933      88
## 87       3.950      76
## 88       4.517      80
## 89       2.167      48
## 90       4.000      86
## 91       2.200      60
## 92       4.333      90
## 93       1.867      50
## 94       4.817      78
```

```
## 95       1.833      63
## 96       4.300      72
## 97       4.667      84
## 98       3.750      75
## 99       1.867      51
## 100      4.900      82
## 101      2.483      62
## 102      4.367      88
## 103      2.100      49
## 104      4.500      83
## 105      4.050      81
## 106      1.867      47
## 107      4.700      84
## 108      1.783      52
## 109      4.850      86
## 110      3.683      81
## 111      4.733      75
## 112      2.300      59
## 113      4.900      89
## 114      4.417      79
## 115      1.700      59
## 116      4.633      81
## 117      2.317      50
## 118      4.600      85
## 119      1.817      59
## 120      4.417      87
## 121      2.617      53
## 122      4.067      69
## 123      4.250      77
## 124      1.967      56
## 125      4.600      88
## 126      3.767      81
## 127      1.917      45
## 128      4.500      82
## 129      2.267      55
## 130      4.650      90
## 131      1.867      45
## 132      4.167      83
## 133      2.800      56
## 134      4.333      89
## 135      1.833      46
## 136      4.383      82
## 137      1.883      51
## 138      4.933      86
## 139      2.033      53
## 140      3.733      79
```

```
## 141      4.233      81
## 142      2.233      60
## 143      4.533      82
## 144      4.817      77
## 145      4.333      76
## 146      1.983      59
## 147      4.633      80
## 148      2.017      49
## 149      5.100      96
## 150      1.800      53
## 151      5.033      77
## 152      4.000      77
## 153      2.400      65
## 154      4.600      81
## 155      3.567      71
## 156      4.000      70
## 157      4.500      81
## 158      4.083      93
## 159      1.800      53
## 160      3.967      89
## 161      2.200      45
## 162      4.150      86
## 163      2.000      58
## 164      3.833      78
## 165      3.500      66
## 166      4.583      76
## 167      2.367      63
## 168      5.000      88
## 169      1.933      52
## 170      4.617      93
## 171      1.917      49
## 172      2.083      57
## 173      4.583      77
## 174      3.333      68
## 175      4.167      81
## 176      4.333      81
## 177      4.500      73
## 178      2.417      50
## 179      4.000      85
## 180      4.167      74
## 181      1.883      55
## 182      4.583      77
## 183      4.250      83
## 184      3.767      83
## 185      2.033      51
## 186      4.433      78
```

```
## 187     4.083      84
## 188     1.833      46
## 189     4.417      83
## 190     2.183      55
## 191     4.800      81
## 192     1.833      57
## 193     4.800      76
## 194     4.100      84
## 195     3.966      77
## 196     4.233      81
## 197     3.500      87
## 198     4.366      77
## 199     2.250      51
## 200     4.667      78
## 201     2.100      60
## 202     4.350      82
## 203     4.133      91
## 204     1.867      53
## 205     4.600      78
## 206     1.783      46
## 207     4.367      77
## 208     3.850      84
## 209     1.933      49
## 210     4.500      83
## 211     2.383      71
## 212     4.700      80
## 213     1.867      49
## 214     3.833      75
## 215     3.417      64
## 216     4.233      76
## 217     2.400      53
## 218     4.800      94
## 219     2.000      55
## 220     4.150      76
## 221     1.867      50
## 222     4.267      82
## 223     1.750      54
## 224     4.483      75
## 225     4.000      78
## 226     4.117      79
## 227     4.083      78
## 228     4.267      78
## 229     3.917      70
## 230     4.550      79
## 231     4.083      70
## 232     2.417      54
```

```
## 233     4.183      86
## 234     2.217      50
## 235     4.450      90
## 236     1.883      54
## 237     1.850      54
## 238     4.283      77
## 239     3.950      79
## 240     2.333      64
## 241     4.150      75
## 242     2.350      47
## 243     4.933      86
## 244     2.900      63
## 245     4.583      85
## 246     3.833      82
## 247     2.083      57
## 248     4.367      82
## 249     2.133      67
## 250     4.350      74
## 251     2.200      54
## 252     4.450      83
## 253     3.567      73
## 254     4.500      73
## 255     4.150      88
## 256     3.817      80
## 257     3.917      71
## 258     4.450      83
## 259     2.000      56
## 260     4.283      79
## 261     4.767      78
## 262     4.533      84
## 263     1.850      58
## 264     4.250      83
## 265     1.983      43
## 266     2.250      60
## 267     4.750      75
## 268     4.117      81
## 269     2.150      46
## 270     4.417      90
## 271     1.817      46
## 272     4.467      74
```

```
print(faithful)
```

```
##     eruptions waiting
## 1       3.600      79
## 2       1.800      54
```

```
## 3       3.333     74
## 4       2.283     62
## 5       4.533     85
## 6       2.883     55
## 7       4.700     88
## 8       3.600     85
## 9       1.950     51
## 10      4.350     85
## 11      1.833     54
## 12      3.917     84
## 13      4.200     78
## 14      1.750     47
## 15      4.700     83
## 16      2.167     52
## 17      1.750     62
## 18      4.800     84
## 19      1.600     52
## 20      4.250     79
## 21      1.800     51
## 22      1.750     47
## 23      3.450     78
## 24      3.067     69
## 25      4.533     74
## 26      3.600     83
## 27      1.967     55
## 28      4.083     76
## 29      3.850     78
## 30      4.433     79
## 31      4.300     73
## 32      4.467     77
## 33      3.367     66
## 34      4.033     80
## 35      3.833     74
## 36      2.017     52
## 37      1.867     48
## 38      4.833     80
## 39      1.833     59
## 40      4.783     90
## 41      4.350     80
## 42      1.883     58
## 43      4.567     84
## 44      1.750     58
## 45      4.533     73
## 46      3.317     83
## 47      3.833     64
## 48      2.100     53
```

```
## 49        4.633      82
## 50        2.000      59
## 51        4.800      75
## 52        4.716      90
## 53        1.833      54
## 54        4.833      80
## 55        1.733      54
## 56        4.883      83
## 57        3.717      71
## 58        1.667      64
## 59        4.567      77
## 60        4.317      81
## 61        2.233      59
## 62        4.500      84
## 63        1.750      48
## 64        4.800      82
## 65        1.817      60
## 66        4.400      92
## 67        4.167      78
## 68        4.700      78
## 69        2.067      65
## 70        4.700      73
## 71        4.033      82
## 72        1.967      56
## 73        4.500      79
## 74        4.000      71
## 75        1.983      62
## 76        5.067      76
## 77        2.017      60
## 78        4.567      78
## 79        3.883      76
## 80        3.600      83
## 81        4.133      75
## 82        4.333      82
## 83        4.100      70
## 84        2.633      65
## 85        4.067      73
## 86        4.933      88
## 87        3.950      76
## 88        4.517      80
## 89        2.167      48
## 90        4.000      86
## 91        2.200      60
## 92        4.333      90
## 93        1.867      50
## 94        4.817      78
```

```
## 95      1.833     63
## 96      4.300     72
## 97      4.667     84
## 98      3.750     75
## 99      1.867     51
## 100     4.900     82
## 101     2.483     62
## 102     4.367     88
## 103     2.100     49
## 104     4.500     83
## 105     4.050     81
## 106     1.867     47
## 107     4.700     84
## 108     1.783     52
## 109     4.850     86
## 110     3.683     81
## 111     4.733     75
## 112     2.300     59
## 113     4.900     89
## 114     4.417     79
## 115     1.700     59
## 116     4.633     81
## 117     2.317     50
## 118     4.600     85
## 119     1.817     59
## 120     4.417     87
## 121     2.617     53
## 122     4.067     69
## 123     4.250     77
## 124     1.967     56
## 125     4.600     88
## 126     3.767     81
## 127     1.917     45
## 128     4.500     82
## 129     2.267     55
## 130     4.650     90
## 131     1.867     45
## 132     4.167     83
## 133     2.800     56
## 134     4.333     89
## 135     1.833     46
## 136     4.383     82
## 137     1.883     51
## 138     4.933     86
## 139     2.033     53
## 140     3.733     79
```

```
## 141      4.233      81
## 142      2.233      60
## 143      4.533      82
## 144      4.817      77
## 145      4.333      76
## 146      1.983      59
## 147      4.633      80
## 148      2.017      49
## 149      5.100      96
## 150      1.800      53
## 151      5.033      77
## 152      4.000      77
## 153      2.400      65
## 154      4.600      81
## 155      3.567      71
## 156      4.000      70
## 157      4.500      81
## 158      4.083      93
## 159      1.800      53
## 160      3.967      89
## 161      2.200      45
## 162      4.150      86
## 163      2.000      58
## 164      3.833      78
## 165      3.500      66
## 166      4.583      76
## 167      2.367      63
## 168      5.000      88
## 169      1.933      52
## 170      4.617      93
## 171      1.917      49
## 172      2.083      57
## 173      4.583      77
## 174      3.333      68
## 175      4.167      81
## 176      4.333      81
## 177      4.500      73
## 178      2.417      50
## 179      4.000      85
## 180      4.167      74
## 181      1.883      55
## 182      4.583      77
## 183      4.250      83
## 184      3.767      83
## 185      2.033      51
## 186      4.433      78
```

```
## 187     4.083      84
## 188     1.833      46
## 189     4.417      83
## 190     2.183      55
## 191     4.800      81
## 192     1.833      57
## 193     4.800      76
## 194     4.100      84
## 195     3.966      77
## 196     4.233      81
## 197     3.500      87
## 198     4.366      77
## 199     2.250      51
## 200     4.667      78
## 201     2.100      60
## 202     4.350      82
## 203     4.133      91
## 204     1.867      53
## 205     4.600      78
## 206     1.783      46
## 207     4.367      77
## 208     3.850      84
## 209     1.933      49
## 210     4.500      83
## 211     2.383      71
## 212     4.700      80
## 213     1.867      49
## 214     3.833      75
## 215     3.417      64
## 216     4.233      76
## 217     2.400      53
## 218     4.800      94
## 219     2.000      55
## 220     4.150      76
## 221     1.867      50
## 222     4.267      82
## 223     1.750      54
## 224     4.483      75
## 225     4.000      78
## 226     4.117      79
## 227     4.083      78
## 228     4.267      78
## 229     3.917      70
## 230     4.550      79
## 231     4.083      70
## 232     2.417      54
```

```
## 233      4.183      86
## 234      2.217      50
## 235      4.450      90
## 236      1.883      54
## 237      1.850      54
## 238      4.283      77
## 239      3.950      79
## 240      2.333      64
## 241      4.150      75
## 242      2.350      47
## 243      4.933      86
## 244      2.900      63
## 245      4.583      85
## 246      3.833      82
## 247      2.083      57
## 248      4.367      82
## 249      2.133      67
## 250      4.350      74
## 251      2.200      54
## 252      4.450      83
## 253      3.567      73
## 254      4.500      73
## 255      4.150      88
## 256      3.817      80
## 257      3.917      71
## 258      4.450      83
## 259      2.000      56
## 260      4.283      79
## 261      4.767      78
## 262      4.533      84
## 263      1.850      58
## 264      4.250      83
## 265      1.983      43
## 266      2.250      60
## 267      4.750      75
## 268      4.117      81
## 269      2.150      46
## 270      4.417      90
## 271      1.817      46
## 272      4.467      74
```

#### 2.7.0.1 Functions Debunked

**print()** is another option for you to use if you want to see a variable, dataset, or any other type of output.

print( > **ANY OBJECT**

)

**For example:** `print(faithful)`, `print(faithful$eruptions)`, `print(1:12)`

### 2.7.1 1.6 Try it yourself

1. What are 2 ways that we can print rows 1 to 5 of the data frame faithful?
2. What is the value of the cell in the fourth row and second column of the data frame faithful?

### 2.7.2 Dimensions

Usually, the first thing we want to do when exploring a dataset is to check its dimensions. To do this, we use the function `dim()` with the dataset name between the `()`.

```
dim(faithful)
```

```
## [1] 272   2
```

You should see two numbers as the output: 272 and 2. This tells us that the dataset faithful has 272 rows (AKA observations) and 2 columns (AKA variables). Checking the dimensions of our datasets may be helpful when we want to check how large our dataset is after a certain data manipulation method. This may also be helpful to check if our manipulated or original data is abnormally large or small.

### 2.7.3 Structure

We can also check the structure of our data using the function `str()` with the dataset name between the `()`.

```
str(faithful)
```

```
## 'data.frame':    272 obs. of  2 variables:
##  $ eruptions: num  3.6 1.8 3.33 2.28 4.53 ...
##  $ waiting  : num  79 54 74 62 85 55 88 85 51 85 ...
```

In this case, we are provided with several pieces of information: 1. the dataset faithful is a data frame 2. there are two columns, or variables, in this dataset: eruptions and waiting 3. both eruptions and waiting contain numerical data

As you can see, `str()` can be very helpful if we want to check what kind of data we are working with and how large the data is.

### 2.7.4 Class

The class of our data can also be checked using the function `class()`. Similarly, the dataset name or the variable name can go between the `()`.

```
class(faithful)
```

```
## [1] "data.frame"
```

```
class(faithful$eruptions)
```

```
## [1] "numeric"
```

Note how if we are checking the class of the variable eruption, we need to have the dataset name followed by a $ first before we can write the variable name.

### 2.7.5 1.7 Try it yourself

Write a code to find the structure of the variable waiting in the faithful dataset.

### 2.7.6 Length

We can also check the length of our dataset or variable using `length()`.

```
length(faithful)
```

```
## [1] 2
```

```
    ## the output should be 2 - the number of variables in our dataset!
```

```
length(faithful$eruption)
```

```
## [1] 272
```

```
    ## the output should be 272 - the number of observations in this variable!
```

### 2.7.7 1.8 Try it yourself

Remember those variables that we created earlier in the tutorial? Try finding the lengths of data frame and numeric.

**Challenge:** Psst! There are actually 2 ways for you to find the length of numeric.

### 2.7.8 Head and Tail

So we are now somewhat familiar with the basic functions to explore the general information about our dataset, YAY! If you want to check the actual dataset (AKA see the actual table), but do not want to see the whole data frame with 272 rows, `head()` and `tail()` are good options.

`head()` shows you the first few rows of your dataset.

```
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

`tail()` shows you the last few rows of your dataset.

```
tail(faithful)
```

```
##     eruptions waiting
## 267     4.750      75
## 268     4.117      81
## 269     2.150      46
## 270     4.417      90
## 271     1.817      46
## 272     4.467      74
```

You can also choose how many rows you want to see

```
head(faithful, 10)
```

```
##    eruptions waiting
## 1      3.600      79
## 2      1.800      54
## 3      3.333      74
## 4      2.283      62
## 5      4.533      85
## 6      2.883      55
## 7      4.700      88
## 8      3.600      85
## 9      1.950      51
## 10     4.350      85
```

```
tail(faithful, 2)
```

```
##     eruptions waiting
## 271     1.817      46
## 272     4.467      74
```

As you can see, `head()` and `tail()` allow you to check just a portion of the dataset. This is especially useful when you're working with large datasets and you only want to see part of it to make sure everything is okay!

#### 2.7.8.1 DO QUESTION 8 OF THE QUIZ NOW

Which of the following codes is best to find how large our dataset is?

### 2.7.9 Mathematical Functions in R

R has a range of mathemtical functions for us to use. Below are only a few basic ones, we will cover much more as we move through our tutorials. Note that these functions only work if the data class is numeric.

We can find the **mean** of waiting:

```
mean(faithful$waiting)
```

```
## [1] 70.89706
```

We can also find the **maximum** and **minimum** values of waiting:

```
max(faithful$waiting)
```

```
## [1] 96
```

```
min(faithful$waiting)
```

```
## [1] 43
```

And the **1st, 2nd, 3rd, and 4th quantile** of waiting:

```
quantile(faithful$waiting, 0.25)
```

```
## 25%
##  58
```

```
quantile(faithful$waiting, 0.5)
```

```
## 50%
##  76
```

```
quantile(faithful$waiting, 0.75)
```

```
## 75%
##  82
```

```
quantile(faithful$waiting, 1)
```

```
## 100%
##   96
```

As well as the **median** of waiting:

```
median(faithful$waiting)
```

```
## [1] 76
```

Another powerful function in R is `summary()`. It literally summarizes everything that we have just covered in this subsection in one single table. Not only that, if you place the dataset name within the `()`, it actually runs all of the functions above for all variables in the dataset.

```
summary(faithful)
```

```
##    eruptions        waiting
##  Min.   :1.600   Min.   :43.0
##  1st Qu.:2.163   1st Qu.:58.0
##  Median :4.000   Median :76.0
##  Mean   :3.488   Mean   :70.9
##  3rd Qu.:4.454   3rd Qu.:82.0
##  Max.   :5.100   Max.   :96.0
```

### 2.7.10 1.9 Try it yourself

Recall that in order for us to refer to a variable in a dataset, we need to first type the dataset name following by a $ before we can type the variable name.

A way to avoid repeating `faithful$` everytime is to attach the dataset using `attach(faithful)`.

Try attaching the dataset faithful then find the mean of the variable eruptions without using $!

#### 2.7.10.1 DO QUESTION 9 OF THE QUIZ NOW

> What information does the function `summary()` provide us with? (select all that apply)

## 2.8 6. WRITING A NEW FUNCTION IN R

While R and its existing packages has a lot to offer, there may be times when the function that we want to use does not really exist. In situations like this, we may want to just write our own function! A function in R is a script and it aims to help you to write reproducible code.

For example, we want to write a function that helps us convert temperature from Celsius to Farenheit. To do this, we would need this shell first:

```
F_to_C <- function(F) {}
```

F_to_C is our function name, and `function(F)` tells R that we want to write a new function. The actual function that we write will be placed between the brackets `{}`.

Let's think of the math really quickly. To convert °F to °C, we would need to subtract 32 then times that by 5/9. So that should look like this: `(F - 32) * 5 / 9`. And we want the function to print the results, so our final function should look like this:

```
F_to_C <- function(F) {
    print((F - 32) * 5 / 9)
}
```

Each function has its own name and it is meant to be different from any other function names. To run it, you need to call the function name.

Now let's see if your function works. We can test it using a known value. Since we know that 32°F is equal to 0°C:

```
F_to_C(32)
```

```
## [1] 0
```

Awesome! It works!

Here are a few other ways that we can create new functions! Note that we can also nest existing functions inside new functions to make our own functions! For example, knowing that `sprintf()` is an existing function, we can write the following:

```
hello <- function(name){
    sprintf("Hello %s", name)
}
```

```
hello("Tim")
```

```
## [1] "Hello Tim"
```

```
hello("Elisa")
```

```
## [1] "Hello Elisa"
```

We can also write new functions that will give us a return statement. A return statement means you want some output or result after running the script to be returned. For example, the following function will always plus 1 to whatever number we nest between the `()`.

```
plus_1 <- function(x){
    return(x+1)
}
```

```
plus_1(10)
```

```
## [1] 11
```

```
plus_1(20.93)
```

```
## [1] 21.93
```

Note also that there can be more than one argument in a function! For example, the following function requires the arguments x and y, of which x will be multiplied by 10 and y will be added to the new x.

```
math_work <- function(x, y){
    x = x * 10
    y = y + x
    result <- list(x, y)
    return(result)
}
```

```
math_work(3, 5)
```

```
## [[1]]
## [1] 30
##
## [[2]]
## [1] 35
```

Once we have written a new function, it will saved in your Global Environment and we can continue to use it as long as the Environment is not wiped. Note that the new functions will not work if you open a new R session that does not store them as functions. But we can also run the code chunks above again to remind R of the new functions.

## 2.9 7. FOR LOOPS

In R, we can also write for loops that iterate a particular action/code that we want. For example, we can write a loop that adds 1 to every number of a vector. To do this, we first need to create a vector and assign it to a variable. Let's say we want the vector 1:10 to be assigned to k.

```
k <- 1:10
```

After that, the for loop is written like so:

```
for (i in k){
  print(k[i] + 1)
}
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
```

Another way for us to yield the same output is to define the vector 1:10 within the for loop directly, like so:

```
for (i in 1:10){
  print(i + 1)
}
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
```

We can write more complicated for loops by adding more functions, arithmetic operators, or even vectors! For example, if we want to create a different for loop that calculates for the `k^2`, we should first create a new vector, k.sq, with the the same length as vector k.

```
k.sq <- 1:10
```

After that, we can write our for loop like so:

```
for (i in k){
  k.sq[i] <- k[i]^2
  print(k.sq[i])
}
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
## [1] 64
## [1] 81
## [1] 100
```

Now, everytime we call for `k.sq`, the new vector should be the output!

```
k.sq
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

We can also write for loops for data frames. Let's take a look at our data frame, dataframe, again.

```
dataframe
```

```
##   Number    Text   T/F
## 1      1   hello  TRUE
## 2      2 bonjour FALSE
## 3      3    ciao  TRUE
## 4     10   hello  TRUE
## 5     11 bonjour FALSE
## 6     12    ciao  TRUE
```

Now, let's say we want to loop through the entire data frame to find only the values under the "Number" column. To do that, we first need to know the number of rows our data frame has - we can use `nrow()` for this. But as we have learned before, we can just define this value directly in our for loop instead of having to do an extra step of finding the number of rows outside of the for loop.

In other words, our code would look like so:

```
for (i in 1:nrow(dataframe)){
  print(dataframe[i,"Number"])
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 10
## [1] 11
## [1] 12
```

We can also add conditions to our for loops by using `if`. For instance, we only want to print values under the "T/F" column where `i > 3`. To do that, we would need to write the following codes:

```
for (i in 1:nrow(dataframe)){
  if (i > 3) print(dataframe[i,"T/F"])
}
```

```
## [1] TRUE
## [1] FALSE
## [1] TRUE
```

## 2.10 8. HELP WITHIN AND OUTSIDE OF R

It gets a while to get used to the language of R and no tutorials can fully explain the complexities and nuances of this language. Therefore, it is important to know how to and where to get help about R! There are so many ways for you to search for help on R, here are only a few methods:

### 2.10.1 Within R

If you have questions about a function or dataset in R, the easiest way to get help is to type ? before that particular function or dataset - as we have previously seen. Try running the codes below, what do you see?

```
# ?mean
```

```
# ?faithful
```

If you want to know whether there is a function for a particular action, you can use ?? before the action. If your action is more than one word long (e.g. geometric functions), you can put the entire phrase in quotation marks ("").

```
# ??"geometic functions"
```

After running the code above, you should see a list of functions, the package that it belongs to, as well as what it does. For example, the function `phil()` in the R package named BAS can be usedd to compound confluent hypergeometric function of two variables.

If you are using RStudio, you can also use the search box in the help window at the bottom right corner of our screen.

https://i.imgur.com/M9KAqcj.png

Figure 2.3: Figure 3. Help window in RStudio

### 2.10.2 Outside of R

Outside of R, there are also plenty of resources for you to tap into R Documentation is a good starting point.

Another resource is Stack Overflow, an online forum where you can ask and answer questions relating to coding!

#### 2.10.2.1 DO QUESTION 10 OF THE QUIZ NOW

> What are some of the ways that we can find help about R? (select all that apply)

## 2.11 9. SUMMARY AND TAKEWAYS

In this tutorial, we learned a few basic steps of using R as a data analysis language. Completing this tutorial will prepare you more advanced tutorials in the future. Learning R is like learning another language, so the biggest tip is to practice practice practice!

After this tutorial, you should be familiar with setting up for an R session using basic working directory functions. Additionally, you should also be comfortable with using arithmetic and logical operators and a few common data types of R. You are also introduced with a few basic functions for exploring a dataset as well as several common methods on how to seek help within and outside of R.

# Chapter 3

# Importing Data into R with readr

## 3.1 INSTRUCTIONS

This tutorial will teach you the basics of importing data from your hard drive into R. We will cover how to import a Comma-Separated Values (csv) file into R using the `read_csv()` function in the readr package. We will also be covering the different data types that R can recognize from a csv file, how to manipulate how data show up on R, as well as how to export the manipulated file back into a csv file.

Accompanying this tutorial is **a short Google quiz** for your own self-assessment. The instructions of this tutorial will clearly indicate when you should answer which question.

## 3.2 LEARNING OBJECTIVES

- Know how to import a Comma-Separated Values (csv) file from a hard drive into R.
- Understand the basics of `read_csv()` including how to use it to import data and how to manipulate the presentation of the data on R.
- Be familiar with the different data types that R can recognize.
- Know how to import other data files such as txt, xlsx, xpt, and sas into R.
- Know how to export a csv file from R into a hard drive.

## 3.3 1. SET UP

In this tutorial, we will be using the readr package, so we will need to install and attach this package onto our R and R session. The readr package is part of a larger **tidyverse core**. This tidyverse core contains many R packages that give us access to functions that mainly work to organize data. In this tutorial series, we will be covering three packages from tidyverse: readr (tutorial 2), dplyr (tutorial 4), and ggplot (tutorial 5).

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

For this tutorial, we will be using the `demo_csv.csv` file. This data is a subset of the National Health and Nutrition Examination Survey (NHANES) conducted by the National Center for Health Statistics (NCHS). Our `demo_csv.csv`, in particular, contains a portion of the information about the demographic of the survey's participants in the years 2013-2014. We will cover NHANES in more detail in tutorial 3. For now, you can explore NHANES in general by visiting this website.

After attaching the readr package, one other thing that we need to complete this tutorial is a csv file. Note that the csv file should be in your working directory - this just makes out lives much easier when we want to import data from a hard drive onto R. Recall that we can use the function `dir()` to check if all of the files we need are in our working directory.

```
dir()
```

```
##  [1] "_book"
##  [2] "_bookdown.yml"
##  [3] "_bookdown_files"
##  [4] "_build.sh"
##  [5] "_deploy.sh"
##  [6] "_output.yml"
##  [7] "0-r-and-rstudio-set-up.Rmd"
##  [8] "1-introduction-to-r.Rmd"
##  [9] "2-importing-data-into-r-with-readr.Rmd"
## [10] "3-introduction-to-nhanes.Rmd"
## [11] "4-data-analysis-with-dplyr.Rmd"
## [12] "5-data-visualization-with-ggplot.Rmd"
## [13] "6-date-time-data-with-lubridate.Rmd"
## [14] "7-data-summary-with-tableone.Rmd"
## [15] "8-Exercise-Solutions.Rmd"
## [16] "9-references.Rmd"
## [17] "book.bib"
## [18] "data"
## [19] "DESCRIPTION"
## [20] "Dockerfile"
```

```
## [21] "docs"
## [22] "header.html"
## [23] "images"
## [24] "index.Rmd"
## [25] "intro2R.Rmd"
## [26] "intro2R_cache"
## [27] "intro2R_files"
## [28] "LICENSE"
## [29] "now.json"
## [30] "packages.bib"
## [31] "preamble.tex"
## [32] "R.Rproj"
## [33] "README.md"
## [34] "style.css"
## [35] "toc.css"
```

You may be a bit confused about the output of the previous code. This is because we are working on Kaggle and the csv file that we will be using is not located in our working directory. More specifically, the csv file is in the *input* folder, whereas our working directory is the *output* folder.

```
getwd()
```

```
## [1] "C:/Users/ehsan/Documents/GitHub/intro2R"
```

There are two ways that we can approach this problem. We will go over how to do both in the next section of this tutorial.

#### 3.3.0.1 DO QUESTION 1 OF THE QUIZ NOW

> **REVIEW:** Which of the following functions lets us set a new working directory?

## 3.4 2. BASICS OF IMPORTING A CSV FILE INTO R

### 3.4.1 Method 1: Setting a Different Working Directory

As we have sort of alluded to in tutorial 1, we can set our working directory as the location of the csv file to import it into R. After successfully importing the file, we can then set the working directory back to our original folder.

```
# setwd("../")
```

Now that we have set our working directory as the *input* folder, we can check if the **demo_csv.csv** file is actually there by using `dir()` again.

```
dir()
```

```
##  [1] "_book"
##  [2] "_bookdown.yml"
##  [3] "_bookdown_files"
##  [4] "_build.sh"
##  [5] "_deploy.sh"
##  [6] "_output.yml"
##  [7] "0-r-and-rstudio-set-up.Rmd"
##  [8] "1-introduction-to-r.Rmd"
##  [9] "2-importing-data-into-r-with-readr.Rmd"
## [10] "3-introduction-to-nhanes.Rmd"
## [11] "4-data-analysis-with-dplyr.Rmd"
## [12] "5-data-visualization-with-ggplot.Rmd"
## [13] "6-date-time-data-with-lubridate.Rmd"
## [14] "7-data-summary-with-tableone.Rmd"
## [15] "8-Exercise-Solutions.Rmd"
## [16] "9-references.Rmd"
## [17] "book.bib"
## [18] "data"
## [19] "DESCRIPTION"
## [20] "Dockerfile"
## [21] "docs"
## [22] "header.html"
## [23] "images"
## [24] "index.Rmd"
## [25] "intro2R.Rmd"
## [26] "intro2R_cache"
## [27] "intro2R_files"
## [28] "LICENSE"
## [29] "now.json"
## [30] "packages.bib"
## [31] "preamble.tex"
## [32] "R.Rproj"
## [33] "README.md"
## [34] "style.css"
## [35] "toc.css"
```

Perfect! Now that we have confirmed the csv file is in our working directory, we can now import it using `read_csv()`. This function is relatively easy to use. All we need to do is add the name of our csv file along with the `.csv` extension in `""` within the brackets, and we are good to go!

```
read_csv("data/demo_csv.csv")
```

```
## Rows: 15 Columns: 5
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): gender, race, edu
## dbl (2): id, age

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 15 x 5
##       id gender   age race                           edu
##    <dbl> <chr>  <dbl> <chr>                          <chr>
##  1 83717 Female    80 Mexican American               Less than 9th grade
##  2 83718 Female    60 Non-Hispanic Black             High school graduate/GED or~
##  3 83719 Male       3 Mexican American               <NA>
##  4 83720 Male      36 Non-Hispanic Black             Some college or AA degree
##  5 83721 Male      52 Non-Hispanic White             College graduate or above
##  6 83722 Male       0 Other Race - Including Multi~ <NA>
##  7 83723 Male      61 Mexican American               9-11th grade (Includes 12th~
##  8 83724 Male      80 Non-Hispanic White             High school graduate/GED or~
##  9 83725 Male       7 Mexican American               <NA>
## 10 83726 Male      40 Mexican American               Less than 9th grade
## 11 83727 Male      26 Other Hispanic                 College graduate or above
## 12 83728 Female     2 Mexican American               <NA>
## 13 83729 Female    42 Non-Hispanic Black             College graduate or above
## 14 83730 Male       7 Other Hispanic                 <NA>
## 15 83731 Male      11 Non-Hispanic Asian             <NA>
```

You should see a list of "Column specification" and the `demo_csv.csv` file imported into a data frame in R after running the codes above. We will go over what "Column specification" means later in this tutorial.

Now that we have successfully imported our csv file into R, it is time for us to set our working directory back to our original directory.

```
#setwd("..")
#setwd("..")
#setwd("..")


#setwd("./data/")
```

```
getwd()
```

```
## [1] "C:/Users/ehsan/Documents/GitHub/intro2R"
```

### 3.4.2 Method 2: Copying the Exact Pathway of the File

Another way for us to import the csv file into R is to copy and paste the exact pathway of the file into `read_csv()`. You should see the exact same "Column specification" and data frame as before!

```
read_csv("data/demo_csv.csv")
```

```
## Rows: 15 Columns: 5

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): gender, race, edu
## dbl (2): id, age

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 15 x 5
##       id gender   age race                          edu
##    <dbl> <chr>  <dbl> <chr>                         <chr>
##  1 83717 Female    80 Mexican American              Less than 9th grade
##  2 83718 Female    60 Non-Hispanic Black            High school graduate/GED or~
##  3 83719 Male       3 Mexican American              <NA>
##  4 83720 Male      36 Non-Hispanic Black            Some college or AA degree
##  5 83721 Male      52 Non-Hispanic White            College graduate or above
##  6 83722 Male       0 Other Race - Including Multi~ <NA>
##  7 83723 Male      61 Mexican American              9-11th grade (Includes 12th~
##  8 83724 Male      80 Non-Hispanic White            High school graduate/GED or~
##  9 83725 Male       7 Mexican American              <NA>
## 10 83726 Male      40 Mexican American              Less than 9th grade
## 11 83727 Male      26 Other Hispanic                College graduate or above
## 12 83728 Female     2 Mexican American              <NA>
## 13 83729 Female    42 Non-Hispanic Black            College graduate or above
## 14 83730 Male       7 Other Hispanic                <NA>
## 15 83731 Male      11 Non-Hispanic Asian            <NA>
```

Note that you can also store this imported data into an object using `<-`.

```
DEMO <- read_csv("data/demo_csv.csv", show_col_types = FALSE)
```

Now, we can just type DEMO to see the data frame.

```
DEMO
```

```
## # A tibble: 15 x 5
##       id gender   age race                          edu
##    <dbl> <chr>  <dbl> <chr>                         <chr>
##  1 83717 Female    80 Mexican American              Less than 9th grade
```

```
##  2 83718 Female    60 Non-Hispanic Black           High school graduate/GED or~
##  3 83719 Male       3 Mexican American             <NA>
##  4 83720 Male      36 Non-Hispanic Black           Some college or AA degree
##  5 83721 Male      52 Non-Hispanic White           College graduate or above
##  6 83722 Male       0 Other Race - Including Multi~ <NA>
##  7 83723 Male      61 Mexican American             9-11th grade (Includes 12th~
##  8 83724 Male      80 Non-Hispanic White           High school graduate/GED or~
##  9 83725 Male       7 Mexican American             <NA>
## 10 83726 Male      40 Mexican American             Less than 9th grade
## 11 83727 Male      26 Other Hispanic               College graduate or above
## 12 83728 Female     2 Mexican American             <NA>
## 13 83729 Female    42 Non-Hispanic Black           College graduate or above
## 14 83730 Male       7 Other Hispanic               <NA>
## 15 83731 Male      11 Non-Hispanic Asian           <NA>
```

#### 3.4.2.1 DO QUESTIONS 2-4 OF THE QUIZ NOW

> **REVIEW:** Which of the following codes will print the entire DEMO data frame?
>
> read_csv can also be used to import Excel and txt files. (True or False)
>
> Which R package does the function `read_csv()` belong to?

### 3.4.3 2.1 Try it yourself

Can you try importing the `bpx.csv` file into R using the function `read_csv()`?

### 3.4.4 Key Notes About Importing Data into R

There are a few key things that we should note when using `read_csv()`: 1. The file name or pathway to the file needs to be in `""`, 2. The file extension, `.csv`, needs to be present, and 3. The name of the file needs to be **exact**.

The third point is related to one of the most common mistakes. When importing any data from your hard drive onto R, you need to make sure that the file name that you write in R is **exactly** what it displays on your hard drive. For instance, take note of spaces, capital letters, spelling of words, as well as the correct extensions. In other words, `demo_csv-1.csv` or `Demo_csv.csv` is much different than `demo_csv.csv`.

Another point to note is that `read_csv()` automatically assumes that the first row of your csv file is the header. We will learn how to tell R this assumption is not correct in section 3 of this tutorial.

#### 3.4.4.1 DO QUESTION 5 OF THE QUIZ NOW

> **REVIEW:** R is case sensitive. (True or False)

### 3.4.5 2.2 Try it yourself

Can you identify the mistakes of the following codes?

```
# a.
# read_csv(../input/import/demo_csv.csv)

# b.
# read_csv("data/DEMO_csv.csv")

# c.
# Read_csv("data/demo_csv.csv")

# d.
# read_csv(data/"demo_csv.csv")
```

#### 3.4.5.1 DO QUESTION 6 OF THE QUIZ NOW

> Which of the following statements about `read_csv()` are correct? (select all that apply)

### 3.4.6 Column Specification

You may notice that when you import a data into R by running `read_csv()`, a "Column specification" list appears. This list tells us two things: 1. The **names** of our columns and 2. The **type of data** that each column contains.

```
read_csv("data/demo_csv.csv")
```

```
## Rows: 15 Columns: 5

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): gender, race, edu
## dbl (2): id, age

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 15 x 5
##       id gender   age race                         edu
##    <dbl> <chr>  <dbl> <chr>                        <chr>
##  1 83717 Female    80 Mexican American             Less than 9th grade
##  2 83718 Female    60 Non-Hispanic Black           High school graduate/GED or~
##  3 83719 Male       3 Mexican American             <NA>
##  4 83720 Male      36 Non-Hispanic Black           Some college or AA degree
##  5 83721 Male      52 Non-Hispanic White           College graduate or above
##  6 83722 Male       0 Other Race - Including Multi~ <NA>
```

```
##  7 83723 Male      61 Mexican American            9-11th grade (Includes 12th~
##  8 83724 Male      80 Non-Hispanic White          High school graduate/GED or~
##  9 83725 Male       7 Mexican American            <NA>
## 10 83726 Male      40 Mexican American            Less than 9th grade
## 11 83727 Male      26 Other Hispanic              College graduate or above
## 12 83728 Female     2 Mexican American            <NA>
## 13 83729 Female    42 Non-Hispanic Black          College graduate or above
## 14 83730 Male       7 Other Hispanic              <NA>
## 15 83731 Male      11 Non-Hispanic Asian          <NA>
```

As we can see after running the code above, there are five columns in our data frame: id (the participant's unique ID number), gender, age, race, and edu (highest level of education).

We can also see that there are two types of data in this data frame `col_double()` and `col_character()`.

#### 3.4.6.1 DO QUESTION 7 OF THE QUIZ NOW

> Which of the following is the best an example of a data that would be classified as `col_double()`?

### 3.4.7 2.3 Try it yourself

Just by looking at the actual data frame, can you guess what type of data `col_double()` and `col_character()` are?

(**HINT**: doubles? integers? logical? character?)

## 3.5 3. MORE ARGUMENTS OF READ_CSV

### 3.5.1 Skip

There are a range of other arguments that we can use with `read_csv()`. Firstly, we can nest `skip` inside the `()` of `read_csv()` to tell R to skip (AKA not import) a certain number of rows when importing our data.

```
Skip_2 <- read_csv("data/demo_csv.csv", skip = 2)
```

```
## Rows: 13 Columns: 5
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Female, Non-Hispanic Black, High school graduate/GED or equi
## dbl (2): 83718, 60
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Skip_2
```

```
## # A tibble: 13 x 5
##    `83718` Female  `60` `Non-Hispanic Black`        `High school graduate/GED o~
##      <dbl> <chr>  <dbl> <chr>                       <chr>
##  1   83719 Male       3 Mexican American            <NA>
##  2   83720 Male      36 Non-Hispanic Black          Some college or AA degree
##  3   83721 Male      52 Non-Hispanic White          College graduate or above
##  4   83722 Male       0 Other Race - Including Mul~ <NA>
##  5   83723 Male      61 Mexican American            9-11th grade (Includes 12th~
##  6   83724 Male      80 Non-Hispanic White          High school graduate/GED or~
##  7   83725 Male       7 Mexican American            <NA>
##  8   83726 Male      40 Mexican American            Less than 9th grade
##  9   83727 Male      26 Other Hispanic              College graduate or above
## 10   83728 Female     2 Mexican American            <NA>
## 11   83729 Female    42 Non-Hispanic Black          College graduate or above
## 12   83730 Male       7 Other Hispanic              <NA>
## 13   83731 Male      11 Non-Hispanic Asian          <NA>
```

```
DEMO
```

```
## # A tibble: 15 x 5
##       id gender   age race                          edu
##    <dbl> <chr>  <dbl> <chr>                         <chr>
##  1 83717 Female    80 Mexican American              Less than 9th grade
##  2 83718 Female    60 Non-Hispanic Black            High school graduate/GED or~
##  3 83719 Male       3 Mexican American              <NA>
##  4 83720 Male      36 Non-Hispanic Black            Some college or AA degree
##  5 83721 Male      52 Non-Hispanic White            College graduate or above
##  6 83722 Male       0 Other Race - Including Multi~ <NA>
##  7 83723 Male      61 Mexican American              9-11th grade (Includes 12th~
##  8 83724 Male      80 Non-Hispanic White            High school graduate/GED or~
##  9 83725 Male       7 Mexican American              <NA>
## 10 83726 Male      40 Mexican American              Less than 9th grade
## 11 83727 Male      26 Other Hispanic                College graduate or above
## 12 83728 Female     2 Mexican American              <NA>
## 13 83729 Female    42 Non-Hispanic Black            College graduate or above
## 14 83730 Male       7 Other Hispanic                <NA>
## 15 83731 Male      11 Non-Hispanic Asian            <NA>
```

When comparing the *Skip_2* table with our original *DEMO* table, we can see that *Skip_2* has two less rows. This is because the argument `skip = 2` has told R to not import the first two rows of our `demo.csv`.

#### 3.5.1.1 DO QUESTION 8 OF THE QUIZ NOW

Which of the following statements is true about the argument `skip`?

### 3.5.2 2.4 Try it yourself

You may also notice that the header of *Skip_2* is incorrect. This is because R recognizes the header of our data as the first row, thus omiting it when importing `demo.csv` into R.

Let's say this is not what we really want. What we actually want to do is to remove the first two rows of actual data while keeping the header. What do you think we have to do to achieve this?

(**HINT**: Recall what we learn about extracting rows in tutorial 1)

### 3.5.3 Remove Header & Header Names

Recall how `read_csv()` assumes that the first row of our data is the header. If this is not true, we can use `col_names = FALSE` to tell R that the first row of our data do not contain headers and that R should add headers for our data.

```
No_header <- read_csv("data/demo_csv.csv",
                      col_names = FALSE)
```

```
## Rows: 16 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (5): X1, X2, X3, X4, X5
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
No_header
```

```
## # A tibble: 16 x 5
##    X1    X2     X3    X4                            X5
##    <chr> <chr>  <chr> <chr>                         <chr>
##  1 id    gender age   race                          edu
##  2 83717 Female 80    Mexican American              Less than 9th grade
##  3 83718 Female 60    Non-Hispanic Black            High school graduate/GED or~
##  4 83719 Male   3     Mexican American              <NA>
##  5 83720 Male   36    Non-Hispanic Black            Some college or AA degree
##  6 83721 Male   52    Non-Hispanic White            College graduate or above
##  7 83722 Male   0     Other Race - Including Multi~ <NA>
##  8 83723 Male   61    Mexican American              9-11th grade (Includes 12th~
##  9 83724 Male   80    Non-Hispanic White            High school graduate/GED or~
## 10 83725 Male   7     Mexican American              <NA>
## 11 83726 Male   40    Mexican American              Less than 9th grade
## 12 83727 Male   26    Other Hispanic                College graduate or above
## 13 83728 Female 2     Mexican American              <NA>
## 14 83729 Female 42    Non-Hispanic Black            College graduate or above
```

```
## 15 83730 Male   7     Other Hispanic                 <NA>
## 16 83731 Male   11    Non-Hispanic Asian             <NA>
```

We can also change the names of our headers by using `col_names =` following by a vector of names. For example:

```
Header_names <- read_csv("data/demo_csv.csv",
                      col_names = c("ID", "Gender", "Age", "Race", "Education"))
```

```
## Rows: 16 Columns: 5
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (5): ID, Gender, Age, Race, Education
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Header_names
```

```
## # A tibble: 16 x 5
##    ID    Gender Age   Race                           Education
##    <chr> <chr>  <chr> <chr>                          <chr>
##  1 id    gender age   race                           edu
##  2 83717 Female 80    Mexican American               Less than 9th grade
##  3 83718 Female 60    Non-Hispanic Black             High school graduate/GED or~
##  4 83719 Male   3     Mexican American               <NA>
##  5 83720 Male   36    Non-Hispanic Black             Some college or AA degree
##  6 83721 Male   52    Non-Hispanic White             College graduate or above
##  7 83722 Male   0     Other Race - Including Multi~ <NA>
##  8 83723 Male   61    Mexican American               9-11th grade (Includes 12th~
##  9 83724 Male   80    Non-Hispanic White             High school graduate/GED or~
## 10 83725 Male   7     Mexican American               <NA>
## 11 83726 Male   40    Mexican American               Less than 9th grade
## 12 83727 Male   26    Other Hispanic                 College graduate or above
## 13 83728 Female 2     Mexican American               <NA>
## 14 83729 Female 42    Non-Hispanic Black             College graduate or above
## 15 83730 Male   7     Other Hispanic                 <NA>
## 16 83731 Male   11    Non-Hispanic Asian             <NA>
```

#### 3.5.3.1 DO QUESTION 9 OF THE QUIZ NOW

> In which of the following scenarios do you think we would **NEED** to use `col_names = FALSE`? (select all that apply)

With the added`col_names`, you may notice that the column specification for our data is not incorrect (everything is recognized as `col_character()`!

This is because R now reads "id", "gender", "age", "race", and "edu" as a content row, and since all of these are texts, R recognizes the entire column as `col_character()`. This is something worthy to note when you are importing data into R.

We can solve this problem with this solution:

```
(Skip_and_Header_Names <- read_csv("data/demo_csv.csv",
                                  skip = 1,
                                  col_names = c("ID", "Gender", "Age", "Race", "Education")))
```

```
## Rows: 15 Columns: 5

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Gender, Race, Education
## dbl (2): ID, Age

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 15 x 5
##       ID Gender   Age Race                          Education
##    <dbl> <chr>  <dbl> <chr>                         <chr>
##  1 83717 Female    80 Mexican American              Less than 9th grade
##  2 83718 Female    60 Non-Hispanic Black            High school graduate/GED or~
##  3 83719 Male       3 Mexican American              <NA>
##  4 83720 Male      36 Non-Hispanic Black            Some college or AA degree
##  5 83721 Male      52 Non-Hispanic White            College graduate or above
##  6 83722 Male       0 Other Race - Including Multi~ <NA>
##  7 83723 Male      61 Mexican American              9-11th grade (Includes 12th~
##  8 83724 Male      80 Non-Hispanic White            High school graduate/GED or~
##  9 83725 Male       7 Mexican American              <NA>
## 10 83726 Male      40 Mexican American              Less than 9th grade
## 11 83727 Male      26 Other Hispanic                College graduate or above
## 12 83728 Female     2 Mexican American              <NA>
## 13 83729 Female    42 Non-Hispanic Black            College graduate or above
## 14 83730 Male       7 Other Hispanic                <NA>
## 15 83731 Male      11 Non-Hispanic Asian            <NA>
```

### 3.5.4 Missing Values

We can also define missing values by using `na =`. For example, if we want to assign "Some college or AA degree" under the edu column as NA, we can use the following code:

```
Missing_values <- read_csv("data/demo_csv.csv",
                           na = "Some college or AA degree")
```

```
## Rows: 15 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): gender, race, edu
## dbl (2): id, age
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Missing_values
```

```
## # A tibble: 15 x 5
##       id gender   age race                           edu
##    <dbl> <chr>  <dbl> <chr>                          <chr>
##  1 83717 Female    80 Mexican American               Less than 9th grade
##  2 83718 Female    60 Non-Hispanic Black             High school graduate/GED or~
##  3 83719 Male       3 Mexican American               NA
##  4 83720 Male      36 Non-Hispanic Black             <NA>
##  5 83721 Male      52 Non-Hispanic White             College graduate or above
##  6 83722 Male       0 Other Race - Including Multi~ NA
##  7 83723 Male      61 Mexican American               9-11th grade (Includes 12th~
##  8 83724 Male      80 Non-Hispanic White             High school graduate/GED or~
##  9 83725 Male       7 Mexican American               NA
## 10 83726 Male      40 Mexican American               Less than 9th grade
## 11 83727 Male      26 Other Hispanic                 College graduate or above
## 12 83728 Female     2 Mexican American               NA
## 13 83729 Female    42 Non-Hispanic Black             College graduate or above
## 14 83730 Male       7 Other Hispanic                 NA
## 15 83731 Male      11 Non-Hispanic Asian             NA
```

#### 3.5.4.1 DO QUESTION 10 OF THE QUIZ NOW

> Only characters can be assigned a value of NA, there is a different missing-value designation for numeric values. (True or False)

## 3.6 4. IMPORTING OTHER FILE TYPES INTO R

While csv is the most common file type to import into R, we can also import other types of data file into R using different functions. In this section, you will be introduced to the very basics of how to import txt, xlsx, xpt, and sas files into R.

### 3.6.1 Text file (txt)

The simplest function that we can use to import a txt file is `read.table()`. This function belongs to the default Base R package, so we do not need to install or attach any packages before using it!

The first argument of this function is the file path. What do you think `header = TRUE` mean?

```
read.table("data/demo_txt.txt", header = TRUE)
```

```
##       id gender age
## 1  83717 Female  80
## 2  83718 Female  60
## 3  83719   Male   3
## 4  83720   Male  36
## 5  83721   Male  52
## 6  83722   Male   0
## 7  83723   Male  61
## 8  83724   Male  80
## 9  83725   Male   7
## 10 83726   Male  40
## 11 83727   Male  26
## 12 83728 Female   2
## 13 83729 Female  42
## 14 83730   Male   7
## 15 83731   Male  11
```

### 3.6.2 Excel file (xlsx)

To import an xlsx file into R, we use `read_excel()`. But before we can use this function, we need to install and attach the readxl package. Similarly to `read.table()`, this function requires a file path.

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

read_excel("data/demo_xlsx.xlsx")
```

```
## # A tibble: 15 x 5
##       id gender   age race                         edu
##    <dbl> <chr>  <dbl> <chr>                        <chr>
##  1 83717 Female    80 Mexican American             Less than 9th grade
##  2 83718 Female    60 Non-Hispanic Black           High school graduate/GED or~
##  3 83719 Male       3 Mexican American             NA
##  4 83720 Male      36 Non-Hispanic Black           Some college or AA degree
##  5 83721 Male      52 Non-Hispanic White           College graduate or above
##  6 83722 Male       0 Other Race - Including Multi~ NA
```

```
##  7 83723 Male      61 Mexican American            9-11th grade (Includes 12th~
##  8 83724 Male      80 Non-Hispanic White          High school graduate/GED or~
##  9 83725 Male       7 Mexican American            NA
## 10 83726 Male      40 Mexican American            Less than 9th grade
## 11 83727 Male      26 Other Hispanic              College graduate or above
## 12 83728 Female     2 Mexican American            NA
## 13 83729 Female    42 Non-Hispanic Black          College graduate or above
## 14 83730 Male       7 Other Hispanic              NA
## 15 83731 Male      11 Non-Hispanic Asian          NA
```

### 3.6.3 2.5 Try it yourself

Import the `bpx.xlsx` into R using the `read_excel()` function.

### 3.6.4 XPT File Extension

Another file type that you may need to import into R is xpt. To do this, we need the function `read.xport()` that belongs to the SASxport package.

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

read.xport("data/demo_xpt.xpt")
```

```
##       ID GENDER                             RACE
## 1  83717 Female                 Mexican American
## 2  83718 Female               Non-Hispanic Black
## 3  83719   Male                 Mexican American
## 4  83720   Male               Non-Hispanic Black
## 5  83721   Male               Non-Hispanic White
## 6  83722   Male Other Race - Including Multi-Rac
## 7  83723   Male                 Mexican American
## 8  83724   Male               Non-Hispanic White
## 9  83725   Male                 Mexican American
## 10 83726   Male                 Mexican American
## 11 83727   Male                   Other Hispanic
## 12 83728 Female                 Mexican American
## 13 83729 Female               Non-Hispanic Black
## 14 83730   Male                   Other Hispanic
## 15 83731   Male Other Race - Including Multi-Rac
```

### 3.6.5 Statistical Analysis Software (SAS)

We can use the function `read_sas()` to import sas files into R. But before we do this, we need to install and attach the haven package.

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

read_sas("data/demo_sas.sas")
```

```
## # A tibble: 15 x 3
##        id gender  race
##     <dbl>  <dbl> <dbl>
##  1 83717      2     1
##  2 83718      2     4
##  3 83719      1     1
##  4 83720      1     4
##  5 83721      1     3
##  6 83722      1     5
##  7 83723      1     1
##  8 83724      1     3
##  9 83725      1     1
## 10 83726      1     1
## 11 83727      1     2
## 12 83728      2     1
## 13 83729      2     4
## 14 83730      1     2
## 15 83731      1     5
```

## 3.7 5. EXPORTING THE DATA FRAME FROM R

After changing and manipulating our data in R, we can also export it back into a csv file to share it. To do this, we can use the function `write_csv()`. For example, let's say we want to export our "Missing_values" data frame.

```
write_csv(Missing_values, "data/Missing Values.csv")
```

We can also export it to an Excel file by using `write_excel_csv()`.

```
write_excel_csv(Missing_values, "data/Missing Values.xlsx")
```

Now if we check our working directory, there should be 2 new files, "Missing Values.csv" and "Missing Values.xlsx"!

```
dir()
```

```
##  [1] "_book"
##  [2] "_bookdown.yml"
##  [3] "_bookdown_files"
##  [4] "_build.sh"
##  [5] "_deploy.sh"
```

```
##  [6] "_output.yml"
##  [7] "0-r-and-rstudio-set-up.Rmd"
##  [8] "1-introduction-to-r.Rmd"
##  [9] "2-importing-data-into-r-with-readr.Rmd"
## [10] "3-introduction-to-nhanes.Rmd"
## [11] "4-data-analysis-with-dplyr.Rmd"
## [12] "5-data-visualization-with-ggplot.Rmd"
## [13] "6-date-time-data-with-lubridate.Rmd"
## [14] "7-data-summary-with-tableone.Rmd"
## [15] "8-Exercise-Solutions.Rmd"
## [16] "9-references.Rmd"
## [17] "book.bib"
## [18] "data"
## [19] "DESCRIPTION"
## [20] "Dockerfile"
## [21] "docs"
## [22] "header.html"
## [23] "images"
## [24] "index.Rmd"
## [25] "intro2R.Rmd"
## [26] "intro2R_cache"
## [27] "intro2R_files"
## [28] "LICENSE"
## [29] "now.json"
## [30] "packages.bib"
## [31] "preamble.tex"
## [32] "R.Rproj"
## [33] "README.md"
## [34] "style.css"
## [35] "toc.css"
```

Congratulations! We have now succeeded in exporting a dataset from R to an external file! This will make our work much easier to share and access!

## 3.8 6. SUMMARY AND TAKEWAYS

In this tutorial, we have learned how to import csv files from our hard drive into R using `read_csv()`. This is an important first step in data analysis or manipulation since we need to be able to have the data in R in order to process it!

# Chapter 4

# Introduction to NHANES

## 4.1 INSTRUCTIONS

This tutorial is aiming to provide an introduction to NHANES dataset and a guide on how to access the NHANES. It will guide you to retrieve the dataset in two days: CDC website and nhanesA package. At the end of this tutorial, it will cover some basic functions in the nhanesA package.

Accompanying this tutorial is **a short Google quiz** for your own self-assessment. The instructions of this tutorial will clearly indicate when you should answer which question.

## 4.2 LEARNING OBJECTIVES

- Be familiar with the survey data in NHANES dataset
- Be able to import NHANES dataset from CDC website
- Be able to set up the nhanesA package and import NHANES dataset from the nhanesA package
- Be familiar with the basic functions in the nhanesA package
- Be able to understand the difference between NHANES dataset and nhanesA package

## 4.3 1. Introduction to NHANES

The National Health and Nutrition Examination Survey (NHANES) is a program with a series of studies aimed at determining the health and nutritional status of Americans, including adults and children. The NHANES program started in the early 1960s and was transformed into a countinuous program in 1999.

Started in the early 1960s, the NHANES programme has consisted of a seriess of surveys concentrating on various population groups or health themes. The survey was transformed into a continuous programme in 1999, with a shifting focus on a variety of health and nutrition measurements. For continuous NHANES,the survey is conducted in two-year cycles, i.e, 1999-2000,2001-2002,etc.

There are 5 types of continues NHANES survey data available to the public:

- Demographics Data
- Dietary Data
- Examination Data
- Laboratory Data
- Questionnaire Data

We will focus on the continuou NHANES dataset in this series of tutorials. On this CDC website, you will see all the continous NHANES ordered by year.

`https://i.imgur.com/8uGOUmc.png`

Figure 4.1: image

We will use NHANES 2013-2014 just for demostration in this tutorial. If you click on NHANES 2013-2014, you will be directed to the page where you can access the data collected between 2013 and 2014:

`https://i.imgur.com/I94dBUa.png`

Figure 4.2: image

We will introduce how to access the data in the following section.

**DO QUESTION 1 OF THE QUIZ NOW** > How many categories (available to public) are there in NHANES dataset?

## 4.4 2. Importing NHANES dataset from website

Now we will learn how to download the NHANES dataset from CDC website and import it to R. In the last section, we're on this page and we'll continue from there.

For example, we want to use the Demographics Data for further analysis. The first step is to download the dataset - click on "Demographics Data":

https://i.imgur.com/wUReKXp.png

Figure 4.3: image

Then, you will see the following page:

https://i.imgur.com/MhlrkEn.png

Figure 4.4: image

To download the dataset, click on *DEMO_H Data [XPT - 3.7 MB]* under the *Data File*.

To find the meanings of the variables, click on *NHANES 2013-2014 Demographics Variable List*.

Last tutorial, we learned how to import `.csv` file into R. However, the file we downloaded from CDC website is not a `.csv` file - it is a `.xpt` file. Instead of using `read_csv()`, we need to use `read.xport()` function housed in `SASxport` package.

First, we install (if needed) and load the`SASxport` package:

```
# install.packages("SASxport")
```

```
library(SASxport)
```

Then, we are ready to load the datatset into R:

```
demo <- read.xport("data/DEMO_H.XPT")
```

We can use the `head()` function to quickly browse the dataset:

```
head(demo,5)
```

```
##    SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1 RIDRETH3 RIDEXMON
## 1 73557        8        2        1       69       NA        4        4        1
## 2 73558        8        2        1       54       NA        3        3        1
## 3 73559        8        2        1       72       NA        3        3        2
## 4 73560        8        2        1        9       NA        3        3        1
## 5 73561        8        2        2       73       NA        3        3        1
##   RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN DMDYRSUS DMDEDUC3 DMDEDUC2
## 1       NA        1       1        1        1       NA       NA        3
## 2       NA        2      NA        1        1       NA       NA        3
## 3       NA        1       1        1        1       NA       NA        4
## 4      119       NA      NA        1        1       NA        3       NA
## 5       NA        2      NA        1        1       NA       NA        5
##   DMDMARTL RIDEXPRG SIALANG SIAPROXY SIAINTRP FIALANG FIAPROXY FIAINTRP MIALANG
## 1        4       NA       1        2        2       1        2        2       1
## 2        1       NA       1        2        2       1        2        2       1
## 3        1       NA       1        2        2       1        2        2       1
## 4       NA       NA       1        1        2       1        2        2       1
## 5        1       NA       1        2        2       1        2        2       1
##   MIAPROXY MIAINTRP AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA DMDHHSZB DMDHHSZE
## 1        2        2        1        3        3        0        0        2
## 2        2        2        1        4        4        0        2        0
## 3        2        2       NA        2        2        0        0        2
## 4        2        2        1        4        4        0        2        0
## 5        2        2       NA        2        2        0        0        2
##   DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU WTINT2YR WTMEC2YR
## 1        1       69        1        3        4       NA 13281.24 13481.04
## 2        1       54        1        3        1        1 23682.06 24471.77
## 3        1       72        1        4        1        3 57214.80 57193.29
## 4        1       33        1        3        1        4 55201.18 55766.51
## 5        1       78        1        5        1        5 63709.67 65541.87
##   SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR
## 1       1      112        4        4     0.84
## 2       1      108        7        7     1.78
## 3       1      109       10       10     4.51
## 4       2      109        9        9     2.52
## 5       2      116       15       15     5.00
```

Now that we've successfully imported the dataset!

#### 4.4.0.1 Functions debunked

**read.xport()** is the function we use to read and load SAS XPORT file in R - it is housed in the SASxport package. The arguments are as follows: read.xport( > FILE PATH

)

**For example:** `read.xport("../input/demo-h/DEMO_H.XPT")`

**DO QUESTION 2 OF THE QUIZ NOW** > Which package is read.xport() in?

## 4.5 3. Importing NHANES dataset from R package: nhanesA

Another way to access the NHANES dataset is to import it from R packages. One popular R package developed for retrieving NHANES dataset is the **nhanesA** package.

As introduced before, we need to first install the **`nhanesA`** package from CRAN:

```
# install.packages("nhanesA")
```

Second, we need to load the **`nhanesA`** package:

```
library(nhanesA)
```

Recall that we have 5 data categories available to the public in the NHANES dataset. How do we access the data from nhanesA package?

There is a useful function called - `nhanesTables()` - list all the data files in each data category in each survey cycle as a table. For example, if we want to see all the Demographics Data in survey cycle 2013-2014:

```
nhanesTables('DEMO', 2013)
```

```
## Warning: `xml_nodes()` was deprecated in rvest 1.0.0.
## Please use `html_elements()` instead.
```

```
##   Data.File.Name                    Data.File.Description
## 1         DEMO_H Demographic Variables and Sample Weights
```

To see all the Examination Data in survey cycle 2015-2016:

```
nhanesTables('EXAM', 2015)
```

```
##    Data.File.Name                        Data.File.Description
## 1           BPX_I                               Blood Pressure
## 2           BMX_I                                Body Measures
## 3        OHXDEN_I                      Oral Health - Dentition
```

```
## 4        OHXREF_I          Oral Health - Recommendation of Care
## 5        FLXCLN_I                           Fluorosis - Clinical
## 6           AUX_I                                     Audiometry
## 7           DXX_I Dual-Energy X-ray Absorptiometry - Whole Body
## 8          AUXAR_I                   Audiometry - Acoustic Reflex
## 9         AUXTYM_I                     Audiometry - Tympanometry
## 10        AUXWBR_I             Audiometry - Wideband Reflectance
```

To see all the Dietary Data in survey cycle 2014-2015:

```
nhanesTables('DIETARY', 2014)
```

```
##    Data.File.Name
## 1        DR1TOT_H
## 2        DR2TOT_H
## 3        DR1IFF_H
## 4        DR2IFF_H
## 5        DRXFCD_H
## 6        DS1IDS_H
## 7        DSQIDS_H
## 8        DS2IDS_H
## 9        DS1TOT_H
## 10       DS2TOT_H
## 11       DSQTOT_H
##                                                    Data.File.Description
## 1                        Dietary Interview - Total Nutrient Intakes, First Day
## 2                       Dietary Interview - Total Nutrient Intakes, Second Day
## 3                              Dietary Interview - Individual Foods, First Day
## 4                             Dietary Interview - Individual Foods, Second Day
## 5                        Dietary Interview Technical Support File - Food Codes
## 6   Dietary Supplement Use 24-Hour - Individual Dietary Supplements, First Day
## 7                  Dietary Supplement Use 30-Day - Individual Dietary Supplements
## 8  Dietary Supplement Use 24-Hour - Individual Dietary Supplements, Second Day
## 9        Dietary Supplement Use 24-Hour - Total Dietary Supplements, First Day
## 10      Dietary Supplement Use 24-Hour - Total Dietary Supplements, Second Day
## 11                     Dietary Supplement Use 30-Day - Total Dietary Supplements
```

#### 4.5.0.1 Functions debunked

**nhanesTables()** is the function we use to display the data in a table format - it is housed in the nhanesA package. The arguments are as follows:

nhanesTables( > 'DATA CATEGORY',

YEAR

)

**Note**:Abbreviation for the data category in the first argument is listed below:

Demographics Data = DEMO

Dietary Data = DIETARY

Examination Data = EXAM

Labortary Data = LAB

Questionnaire Data = Q

**For example:** `nhanesTables('DEMO', 2013)`

For demostration purpose, we will focus on Demographics Data in survey cycle 2013-2014 in the rest of the tutorial.

Now that we need to access and import the dataset from **`nhanesA`** package. The **`nhanes()`** function (exactly the same as the package name) is used for importing NHANES datasets:

```
demo <- nhanes('DEMO_H')
```

```
## Processing SAS dataset DEMO_H      ..
```

Browse the top 5 rows in the demo dataframe:

```
head(demo,5)
```

```
##    SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1 RIDRETH3 RIDEXMON
## 1 73557        8        2        1       69       NA        4        4        1
## 2 73558        8        2        1       54       NA        3        3        1
## 3 73559        8        2        1       72       NA        3        3        2
## 4 73560        8        2        1        9       NA        3        3        1
## 5 73561        8        2        2       73       NA        3        3        1
##   RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN DMDYRSUS DMDEDUC3 DMDEDUC2
## 1       NA        1       1        1        1       NA       NA        3
## 2       NA        2      NA        1        1       NA       NA        3
## 3       NA        1       1        1        1       NA       NA        4
## 4      119       NA      NA        1        1       NA        3       NA
## 5       NA        2      NA        1        1       NA       NA        5
##   DMDMARTL RIDEXPRG SIALANG SIAPROXY SIAINTRP FIALANG FIAPROXY FIAINTRP MIALANG
## 1        4       NA       1        2        2       1        2        2       1
## 2        1       NA       1        2        2       1        2        2       1
## 3        1       NA       1        2        2       1        2        2       1
## 4       NA       NA       1        1        2       1        2        2       1
## 5        1       NA       1        2        2       1        2        2       1
##   MIAPROXY MIAINTRP AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA DMDHHSZB DMDHHSZE
## 1        2        2        1        3        3        0        0        2
## 2        2        2        1        4        4        0        2        0
## 3        2        2       NA        2        2        0        0        2
## 4        2        2        1        4        4        0        2        0
## 5        2        2       NA        2        2        0        0        2
```

```
##   DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU WTINT2YR WTMEC2YR
## 1        1       69        1        3        4       NA 13281.24 13481.04
## 2        1       54        1        3        1        1 23682.06 24471.77
## 3        1       72        1        4        1        3 57214.80 57193.29
## 4        1       33        1        3        1        4 55201.18 55766.51
## 5        1       78        1        5        1        5 63709.67 65541.87
##   SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR
## 1       1      112        4        4     0.84
## 2       1      108        7        7     1.78
## 3       1      109       10       10     4.51
## 4       2      109        9        9     2.52
## 5       2      116       15       15     5.00
```

You may get confused why we put `DEMO_H` instead of `DEMO` in the argument - recall that *DEMO* is the abbreviation for *Demographcis data*. But we also want to tell the function which survey cycle we are particularly interested in.

Go back to the output from *nhanesTables('DEMO', 2013)* above, now we have the data file name - `DEMO_H` and `H` specifies the survey cycle 2013-2014.

As you are getting familiar with the dataset, you may notice that different letter represents different survey cycle year. For example, `H` represents survey cycle 2013-2014 and `I` represents survey cycle 2015-2016.

#### 4.5.0.2 Functions debunked

**nhanes()** is the function we use to retrieve the dataset and return a dataframe - it is housed in the nhanesA package. The arguments are as follows:

nhanesTranslate( > NAME OF TABLE

)

**For example:** `nhanes('DEMO_H')`

If you run demo alone, you will see that *RIAGENDR* (gender) is coded as 1 and 2. For ease of future use, we want to translate this 1 and 2 into male and female.

To translate the categorical variables in NHANES, use nhanesTranslate():

```
demo_translate <- nhanesTranslate('DEMO_H',
                                  c('SEQN', # Respondent sequence number
                                    'RIAGENDR'),
                                 data = demo)
```

```
## Warning in FUN(X[[i]], ...): No translation table is available for SEQN
```

```
## Translated columns: RIAGENDR
```

```
head(demo_translate,5)
```

```
##    SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1 RIDRETH3 RIDEXMON
```

```
## 1 73557         8         2     Male       69       NA        4        4        1
## 2 73558         8         2     Male       54       NA        3        3        1
## 3 73559         8         2     Male       72       NA        3        3        2
## 4 73560         8         2     Male        9       NA        3        3        1
## 5 73561         8         2   Female       73       NA        3        3        1
##   RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN DMDYRSUS DMDEDUC3 DMDEDUC2
## 1       NA        1       1        1        1       NA       NA        3
## 2       NA        2      NA        1        1       NA       NA        3
## 3       NA        1       1        1        1       NA       NA        4
## 4      119       NA      NA        1        1       NA        3       NA
## 5       NA        2      NA        1        1       NA       NA        5
##   DMDMARTL RIDEXPRG SIALANG SIAPROXY SIAINTRP FIALANG FIAPROXY FIAINTRP MIALANG
## 1        4       NA       1        2        2       1        2        2       1
## 2        1       NA       1        2        2       1        2        2       1
## 3        1       NA       1        2        2       1        2        2       1
## 4       NA       NA       1        1        2       1        2        2       1
## 5        1       NA       1        2        2       1        2        2       1
##   MIAPROXY MIAINTRP AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA DMDHHSZB DMDHHSZE
## 1        2        2        1        3        3        0        0        2
## 2        2        2        1        4        4        0        2        0
## 3        2        2       NA        2        2        0        0        2
## 4        2        2        1        4        4        0        2        0
## 5        2        2       NA        2        2        0        0        2
##   DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU WTINT2YR WTMEC2YR
## 1        1       69        1        3        4       NA 13281.24 13481.04
## 2        1       54        1        3        1        1 23682.06 24471.77
## 3        1       72        1        4        1        3 57214.80 57193.29
## 4        1       33        1        3        1        4 55201.18 55766.51
## 5        1       78        1        5        1        5 63709.67 65541.87
##   SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR
## 1       1      112        4        4     0.84
## 2       1      108        7        7     1.78
## 3       1      109       10       10     4.51
## 4       2      109        9        9     2.52
## 5       2      116       15       15     5.00
```

#### 4.5.0.3 Functions debunked

**nhanesTranslate()** is the function we use to translate variables in a dataset - it is housed in the nhanesA package. The arguments are as follows:

nhanesTranslate( > 'NAME OF DATASET',

> COLUMNS YOU WANT TO BE TRANSLATED, (can be written as a vector)
>
> data = SOURCE DATAFRAME

)

**For example:** `nhanesTranslate('DEMO_H', RIAGENDR, data = demo)`

#### 4.5.0.4 Try it yourself 3.1

Find all the Examination Data in survey cycle 2013-2014

#### 4.5.0.5 Try it yourself 3.2

Import the blood pressure dataset in the Examination Data in survey cycle 2013-2014

#### 4.5.0.6 Try it yourself 3.3

Translate the following variables in the BPX dataset

BPXPULS - Pulse regular or irregular?

BPAARM - Arm selected

**DO QUESTION 3 OF THE QUIZ NOW**

> Fill in the blank to answer 'Try it yourself 3.1':
>
> nhanesTables(`___`,`___`)

**DO QUESTION 4 OF THE QUIZ NOW** > Fill in the blank to answer 'Try it yourself 3.2':

> bpx <- nhanes(`___`)

**DO QUESTION 5 OF THE QUIZ NOW**

> Fill in the blank to answer 'Try it yourself 3.3':
>
> bpx_translate <- nhanesTranslate(`___`,

```
c('BPXPULS', # Pulse regular or irregular?
'BPAARM'), # Arm selected
data = `___`)
```

### 4.5.1 Other packages in R

There are other packages that are developed as a tool to retrieve and analyze the NHANES dataset, such as `RNHANES` package. You are encouraged to explore packages beside **`nhanesA`** and play with the data. However, we will be using **`nhanesA`** for this series of tutorials, so it is important that you're familiar with it before we move on.

### 4.5.2 Alternative ways to download NHANES

If you are wondering how the `nhanes()` function works, here is a glimpse at what it looks like backstage. First of all, we can create our own function that does the same action as `nhanes()`. The function we are creating together is not exactly how the `nhanes()` function works, but the principles are similar. **The general gist is that we need a function that can download the .XPT file on the NHANES website and then import it into R as a data frame.**

The first step that we need to create a new function is to have a function name and the function… `function()`! Let's name our new function `downloadnhanes()`.

```
downloadnhanes <- function(){
}
```

Now, our function needs arguments. Let's give it 2 arguments: the years of the dataset (ex] 2013-2014 or 2014-2015) and the dataset name (ex] DEMO_H or BPX_H).

```
downloadnhanes <- function(years, prefix_suffix){
}
```

For the purpose of creating this function step-by-step, let's say that our years is 2013-2014 and our prefix_suffix is DEMO_H.

```
years <- '2013-2014'
prefix_suffix <- 'DEMO_H'
```

Next, our function needs to be able to download the .XPT file straight from the NHANES website. What this means is we need our function to 1. Know where on the web our .XPT file is and 2. Download that file

To do this, we need to create a variable that contains the .XPT's URL. If we look at the URL of an NHANES dataset's .XPT file, it should look something along this line: **https://wwwn.cdc.gov/nchs/nhanes/2013-2014/DEMO_H .XPT**. To lead R to this specific website first before we download the file, we need to use the `paste()` function like so. Note that "years" and "prefix_suffix" are written as variables because they are two arguments that we would need to define when we use this new function.

```
url <- paste('https://wwwn.cdc.gov/nchs/nhanes/', years,'/', prefix_suffix, '.XPT', sep = '')
```

### 4.5.3 Function debunked

**paste()** is a function we use to combine multiple elements from multiple vectors into one single element. In this case, we are combining hard-wired characters and open variables together into one single url. Some of the main arguments are as follows:

paste(

> **'TEXT STRING THAT WE WANT TO INCLUDE (note that quotation marks)'**,
>
> **A DEFINED VARIABLE**,
>
> sep = **' '** OR **'/'** OR **'_'** OR **'&'** etc (this argument tells R how you want each element to be separated, in our case, we don't want any separator, that's why there is nothing between the" quotation marks)

)

**For example:** `paste('https://wwwn.cdc.gov/nchs/nhanes/', years,'/', prefix_suffix, '.XPT', sep = '')`

After we have our URL, we are ready to download the file! To do this, we use `download.file()` like so:

```
download.file(url, tf <- tempfile(), mode = "wb")
```

There are a number of things that you can do with `download.file()`. We will not go over this function in detail, but here is a helpful website that explains the function really well.

In the function above, we first need the URL of the file that we are downloading - in our case it is just `url` because we already defined it in the previous `paste()` function. Next, we want a place to store our downloaded file. In this case, we want it to be `tf <- tempfile()` because we want to generate a temporary storage space for our downloaded file (you can find more about `tempfile()` here). After that, we need to use `mode = "wb"` because our .XPT file is binary. `wb` is also the most common mode type to use when we use `download.file()`.

Next, we want to import the temporary file `tf` that we create into R. To do this, we need to use `read.xport()`. We have already been introduced to the SASxport package, but an alternative to SASxport is foreign. This is another package that deals with importing different data types into R, not just .XPT.

```
outdf <- foreign::read.xport(tf)
```

Finally, we want to make sure that the file we are importing is a data frame, so we need to use `data.frame()` for this.

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

Now if we combine everything that we have talked about earlier, our function should look like this:

```
downloadnhanes <- function(prefix_suffix, years){
    url <- paste('https://wwwn.cdc.gov/nchs/nhanes/', years,'/', prefix_suffix, '.XPT'
    download.file(url, tf <- tempfile(), mode = "wb")
    outdf <- foreign::read.xport(tf)
    outdf <- data.frame(outdf)
```

```
  return(outdf)
}
```

Let's see if it works!

```
head(
    downloadnhanes('DEMO_H', '2013-2014')
    )
```

```
##    SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1 RIDRETH3 RIDEXMON
## 1 73557        8        2        1       69       NA        4        4        1
## 2 73558        8        2        1       54       NA        3        3        1
## 3 73559        8        2        1       72       NA        3        3        2
## 4 73560        8        2        1        9       NA        3        3        1
## 5 73561        8        2        2       73       NA        3        3        1
## 6 73562        8        2        1       56       NA        1        1        1
##   RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN DMDYRSUS DMDEDUC3 DMDEDUC2
## 1       NA        1       1        1        1       NA       NA        3
## 2       NA        2      NA        1        1       NA       NA        3
## 3       NA        1       1        1        1       NA       NA        4
## 4      119       NA      NA        1        1       NA        3       NA
## 5       NA        2      NA        1        1       NA       NA        5
## 6       NA        1       2        1        1       NA       NA        4
##   DMDMARTL RIDEXPRG SIALANG SIAPROXY SIAINTRP FIALANG FIAPROXY FIAINTRP MIALANG
## 1        4       NA       1        2        2       1        2        2       1
## 2        1       NA       1        2        2       1        2        2       1
## 3        1       NA       1        2        2       1        2        2       1
## 4       NA       NA       1        1        2       1        2        2       1
## 5        1       NA       1        2        2       1        2        2       1
## 6        3       NA       1        2        2       1        2        2       1
##   MIAPROXY MIAINTRP AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA DMDHHSZB DMDHHSZE
## 1        2        2        1        3        3        0        0        2
## 2        2        2        1        4        4        0        2        0
## 3        2        2       NA        2        2        0        0        2
## 4        2        2        1        4        4        0        2        0
## 5        2        2       NA        2        2        0        0        2
## 6        2        2        1        1        1        0        0        0
##   DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU WTINT2YR WTMEC2YR
## 1        1       69        1        3        4       NA 13281.24 13481.04
## 2        1       54        1        3        1        1 23682.06 24471.77
## 3        1       72        1        4        1        3 57214.80 57193.29
## 4        1       33        1        3        1        4 55201.18 55766.51
## 5        1       78        1        5        1        5 63709.67 65541.87
## 6        1       56        1        4        3       NA 24978.14 25344.99
##   SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR
## 1       1      112        4        4     0.84
```

```
## 2       1      108         7         7      1.78
## 3       1      109        10        10      4.51
## 4       2      109         9         9      2.52
## 5       2      116        15        15      5.00
## 6       1      111         9         9      4.79
```

It works! But does it match with the output of our usual `nhanes()` function? Let's test it out.

```
head(
    nhanes('DEMO_H')
    )
```

```
## Processing SAS dataset DEMO_H     ..
```

```
##    SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1 RIDRETH3 RIDEXMON
## 1 73557        8        2        1       69       NA        4        4        1
## 2 73558        8        2        1       54       NA        3        3        1
## 3 73559        8        2        1       72       NA        3        3        2
## 4 73560        8        2        1        9       NA        3        3        1
## 5 73561        8        2        2       73       NA        3        3        1
## 6 73562        8        2        1       56       NA        1        1        1
##   RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN DMDYRSUS DMDEDUC3 DMDEDUC2
## 1       NA        1       1        1        1       NA       NA        3
## 2       NA        2      NA        1        1       NA       NA        3
## 3       NA        1       1        1        1       NA       NA        4
## 4      119       NA      NA        1        1       NA        3       NA
## 5       NA        2      NA        1        1       NA       NA        5
## 6       NA        1       2        1        1       NA       NA        4
##   DMDMARTL RIDEXPRG SIALANG SIAPROXY SIAINTRP FIALANG FIAPROXY FIAINTRP MIALANG
## 1        4       NA       1        2        2       1        2        2       1
## 2        1       NA       1        2        2       1        2        2       1
## 3        1       NA       1        2        2       1        2        2       1
## 4       NA       NA       1        1        2       1        2        2       1
## 5        1       NA       1        2        2       1        2        2       1
## 6        3       NA       1        2        2       1        2        2       1
##   MIAPROXY MIAINTRP AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA DMDHHSZB DMDHHSZE
## 1        2        2        1        3        3        0        0        2
## 2        2        2        1        4        4        0        2        0
## 3        2        2       NA        2        2        0        0        2
## 4        2        2        1        4        4        0        2        0
## 5        2        2       NA        2        2        0        0        2
## 6        2        2        1        1        1        0        0        0
##   DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU WTINT2YR WTMEC2YR
## 1        1       69        1        3        4       NA 13281.24 13481.04
## 2        1       54        1        3        1        1 23682.06 24471.77
## 3        1       72        1        4        1        3 57214.80 57193.29
```

```
## 4        1       33        1        3        1         4 55201.18 55766.51
## 5        1       78        1        5        1         5 63709.67 65541.87
## 6        1       56        1        4        3        NA 24978.14 25344.99
##    SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR
## 1        1      112        4        4     0.84
## 2        1      108        7        7     1.78
## 3        1      109       10       10     4.51
## 4        2      109        9        9     2.52
## 5        2      116       15       15     5.00
## 6        1      111        9        9     4.79
```

The two imported datasets are identical! Fantastic, our new function works! And that's just a sneak peak of what goes behind our `nhanes()` function.

The function that we went over in this tutorial is originally written by **____. **For simplification and educational purposes, the function has been simplified. You can find the original function here** (need to insert link to document**).

## 4.6 TAKEAWAYS

By the end of this tutorial, you should be able to know what is NHANES and how to retrieve the NHANES dataset. You should also be familiar with the functions housed in nhanesA mentioned in this tutorial.

For the next two tutorials, we will introduce two new packages - dplyr for data analysis and ggplot for data visulization. We will continusely use NHANES dataset for illustration and you'll likely be using NHANES dataset for your own research in the future. Make sure you're familiar with NHANES dataset before we move on.

# Chapter 5

# Data Analysis with dplyr

## 5.1 INSTRUCTIONS

This tutorial is aiming to introduce you to how to manipulate data and transfer the raw data into ready-to-analysis form in R. It will guide you to learn and practice the basic and useful functions in the dplyr package. In this tutorial, follow the step-by-step instruction as well as the examples which demonstrating how the functions work.

Accompanying this tutorial is **a short Google quiz** for your own self-assessment. The instructions of this tutorial will clearly indicate when you should answer which question.

## 5.2 LEARNING OBJECTIVES

- Be able to import nhanesA and dplyr package. Be able to understand dataframe.
- Be able to use rename() and select() to rename and select variables (columns) in a dataframe.
- Be able to use filter() to subset a dataframe based on conditions.
- Be able to use arrange() to re-order the rows in a dataframe.
- Be able to use mutate() and transmute() to add new variables that are computed from existing variables in a dataframe.
- Be able to use summarize() to get summary statistics from a dataframe; Be able to perform grouping with summarize(), filter(), and mutate().
- Be able to use pipe to re-write mutilpe operations in a more readable way.
- Be able to check missing values existence and deal with missing values while using the above functions

## 5.3 1. Set up

### 5.3.1 Install and load packages

First, we need to install and load the dplyr package as well as bring in our National Health and Nutrition Examination Survey (NHANES) dataset. This particular tutorial uses data from 2013-2014.

For more information about NHANES, you can visit this website. It is recommended that you explore this website to familiar yourself with the data that we will be using throughout this tutorial.

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

```
##
## 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
```

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

### 5.3.2 Set working directory

We want to set the working directory

```
# setwd('/kaggle/working')
```

### 5.3.3 Import dataset

This particular tutorial uses the Demographics dataset and Blood pressure dataset from NHANES dataset (2013-2014).

The Demographics dataset contains a huge records of demographics information such as gender, race, annual income for each participant. The Blood pressure dataset is in the Examination data and contains a hugh records of measurements that related to blood pressure measurement.

More information on the imported dataset can be found here:

- Demographics data
- Examination data
- Complete variable dictionary

- DEMO_H Code book (Demographic Variables)
- BPX_H Code book (Blood Pressure)

```
demo <- nhanes('DEMO_H')
```

```
## Processing SAS dataset DEMO_H     ..
```

```
bpx <- nhanes('BPX_H')
```

```
## Processing SAS dataset BPX_H      ..
```

#### 5.3.3.1 Functions debunked

`nhanes()` is the function we use to import our NHANES data and save it in a dataframe. What is a dataframe? We'll introduce it below. The arguments are as follows:

nhanes( > 'NAME OF DATASET'

)

**For example:** `nhanes('DEMO_H')`

### 5.3.4 Explore our dataset

Recall the basic functions that we learned in tutorial 1, it is good practice for us to explore our datasets before doing any analysis.

We can check the dimension of the datasets:

```
dim(demo)
```

```
## [1] 10175    47
```

```
dim(bpx)
```

```
## [1] 9813   23
```

We can also check their first few rows:

```
head(demo, 3)
```

```
##    SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1 RIDRETH3 RIDEXMON
## 1 73557        8        2        1       69       NA        4        4        1
## 2 73558        8        2        1       54       NA        3        3        1
## 3 73559        8        2        1       72       NA        3        3        2
##   RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN DMDYRSUS DMDEDUC3 DMDEDUC2
## 1       NA        1       1        1        1       NA       NA        3
## 2       NA        2      NA        1        1       NA       NA        3
## 3       NA        1       1        1        1       NA       NA        4
##   DMDMARTL RIDEXPRG SIALANG SIAPROXY SIAINTRP FIALANG FIAPROXY FIAINTRP MIALANG
## 1        4       NA       1        2        2       1        2        2       1
```

```
## 2        1       NA       1        2        2       1        2        2       1
## 3        1       NA       1        2        2       1        2        2       1
##   MIAPROXY MIAINTRP AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA DMDHHSZB DMDHHSZE
## 1        2        2        1        3        3        0        0        2
## 2        2        2        1        4        4        0        2        0
## 3        2        2       NA        2        2        0        0        2
##   DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU WTINT2YR WTMEC2YR
## 1        1       69        1        3        4       NA 13281.24 13481.04
## 2        1       54        1        3        1        1 23682.06 24471.77
## 3        1       72        1        4        1        3 57214.80 57193.29
##   SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR
## 1       1      112        4        4     0.84
## 2       1      108        7        7     1.78
## 3       1      109       10       10     4.51
```

```
head(bpx, 3)
```

```
##    SEQN PEASCST1 PEASCTM1 PEASCCT1 BPXCHR BPAARM BPACSZ BPXPLS BPXPULS BPXPTY
## 1 73557        1      620       NA     NA      1      4     86       1      1
## 2 73558        1      766       NA     NA      1      4     74       1      1
## 3 73559        1      665       NA     NA      1      4     68       1      1
##   BPXML1 BPXSY1 BPXDI1 BPAEN1 BPXSY2 BPXDI2 BPAEN2 BPXSY3 BPXDI3 BPAEN3 BPXSY4
## 1    140    122     72      2    114     76      2    102     74      2     NA
## 2    170    156     62      2    160     80      2    156     42      2     NA
## 3    160    140     90      2    140     76      2    146     80      2     NA
##   BPXDI4 BPAEN4
## 1     NA     NA
## 2     NA     NA
## 3     NA     NA
```

It does not hurt to also check the last few rows of the datasets:

```
tail(demo, 3)
```

```
##        SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1 RIDRETH3
## 10173 83729        8        2        2       42       NA        4        4
## 10174 83730        8        2        1        7       NA        2        2
## 10175 83731        8        2        1       11       NA        5        6
##       RIDEXMON RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN DMDYRSUS DMDEDUC3
## 10173        2       NA        2      NA        2        1        6       NA
## 10174        1       84       NA      NA        1        1       NA        0
## 10175        1      140       NA      NA        1        1       NA        5
##       DMDEDUC2 DMDMARTL RIDEXPRG SIALANG SIAPROXY SIAINTRP FIALANG FIAPROXY
## 10173        5        3        2       1        2        2       1        2
## 10174       NA       NA       NA       1        1        2       1        2
## 10175       NA       NA       NA       1        1        2       1        2
##       FIAINTRP MIALANG MIAPROXY MIAINTRP AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA
## 10173        2      NA       NA       NA       NA        1        1        0
```

```
## 10174         2       NA        NA        NA        NA         4         4         1
## 10175         2        1         2         2        NA         4         4         0
##       DMDHHSZB DMDHHSZE DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU
## 10173        0        0        2       42        2        5        3       NA
## 10174        1        0        2       30        2        4        1        3
## 10175        2        0        1       43        2        5        1        5
##       WTINT2YR  WTMEC2YR SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR
## 10173 24122.25 26902.344       1      104        7        7     3.66
## 10174 25521.88 26686.026       2      109        6        6     1.05
## 10175  8930.18  9700.873       2      106       15       15     5.00
```

```
tail(bpx, 3)
```

```
##       SEQN PEASCST1 PEASCTM1 PEASCCT1 BPXCHR BPAARM BPACSZ BPXPLS BPXPULS
## 9811 83729        1      679       NA     NA      1      4     80       1
## 9812 83730        1      381       NA     72     NA     NA     NA       1
## 9813 83731        1      498       NA     NA      1      3     90       1
##      BPXPTY BPXML1 BPXSY1 BPXDI1 BPAEN1 BPXSY2 BPXDI2 BPAEN2 BPXSY3 BPXDI3
## 9811      1    150    136     82      2    130     82      2    138     80
## 9812     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
## 9813      1    120     94     68      2     94     56      2     90     62
##      BPAEN3 BPXSY4 BPXDI4 BPAEN4
## 9811      2     NA     NA     NA
## 9812     NA     NA     NA     NA
## 9813      2     NA     NA     NA
```

#### 5.3.4.1 is.na()

In real world, missing values are unavoidable. There are many reasons for missing values - may be a result from nonresponse or incorrect data collection and it happens all the time. It is always a good idea to check if there is any missing value before proceeding.

In a dataframe, you will see NAs if there're missing values and *NA* indicates missing values in R.

Note: Be careful when doing manipulation on real dataset!!!Missing values may be recorded in other ways, for example, *infinity* or other numbers. To deal with missing values in this case, one way is to convert these values into NAs and then treat them as regular NAs. Details will not be discussed in this tutorial but will likely be included in later tutorials.

To check if there is any missing values (NAs) in a dataframe, use `is.na()`:

```
head(is.na(demo),5)
```

```
##       SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1 RIDRETH3
## [1,] FALSE    FALSE    FALSE    FALSE    FALSE     TRUE    FALSE    FALSE
## [2,] FALSE    FALSE    FALSE    FALSE    FALSE     TRUE    FALSE    FALSE
```

```
## [3,] FALSE    FALSE    FALSE    FALSE    FALSE     TRUE    FALSE    FALSE
## [4,] FALSE    FALSE    FALSE    FALSE    FALSE     TRUE    FALSE    FALSE
## [5,] FALSE    FALSE    FALSE    FALSE    FALSE     TRUE    FALSE    FALSE
##      RIDEXMON RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN DMDYRSUS DMDEDUC3
## [1,]    FALSE     TRUE    FALSE   FALSE    FALSE    FALSE     TRUE     TRUE
## [2,]    FALSE     TRUE    FALSE    TRUE    FALSE    FALSE     TRUE     TRUE
## [3,]    FALSE     TRUE    FALSE   FALSE    FALSE    FALSE     TRUE     TRUE
## [4,]    FALSE    FALSE     TRUE    TRUE    FALSE    FALSE     TRUE    FALSE
## [5,]    FALSE     TRUE    FALSE    TRUE    FALSE    FALSE     TRUE     TRUE
##      DMDEDUC2 DMDMARTL RIDEXPRG SIALANG SIAPROXY SIAINTRP FIALANG FIAPROXY
## [1,]    FALSE    FALSE     TRUE   FALSE    FALSE    FALSE   FALSE    FALSE
## [2,]    FALSE    FALSE     TRUE   FALSE    FALSE    FALSE   FALSE    FALSE
## [3,]    FALSE    FALSE     TRUE   FALSE    FALSE    FALSE   FALSE    FALSE
## [4,]     TRUE     TRUE     TRUE   FALSE    FALSE    FALSE   FALSE    FALSE
## [5,]    FALSE    FALSE     TRUE   FALSE    FALSE    FALSE   FALSE    FALSE
##      FIAINTRP MIALANG MIAPROXY MIAINTRP AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA
## [1,]    FALSE   FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
## [2,]    FALSE   FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
## [3,]    FALSE   FALSE    FALSE    FALSE     TRUE    FALSE    FALSE    FALSE
## [4,]    FALSE   FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
## [5,]    FALSE   FALSE    FALSE    FALSE     TRUE    FALSE    FALSE    FALSE
##      DMDHHSZB DMDHHSZE DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU
## [1,]    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE     TRUE
## [2,]    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
## [3,]    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
## [4,]    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
## [5,]    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
##      WTINT2YR WTMEC2YR SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR
## [1,]    FALSE    FALSE   FALSE    FALSE    FALSE    FALSE    FALSE
## [2,]    FALSE    FALSE   FALSE    FALSE    FALSE    FALSE    FALSE
## [3,]    FALSE    FALSE   FALSE    FALSE    FALSE    FALSE    FALSE
## [4,]    FALSE    FALSE   FALSE    FALSE    FALSE    FALSE    FALSE
## [5,]    FALSE    FALSE   FALSE    FALSE    FALSE    FALSE    FALSE
```

To find the total number of missing values in a dataframe:

```
sum(is.na(demo))
```

```
## [1] 82092
```

To check if there is any missing values (NAs) in a column,for example, gender:

```
head(is.na(demo['RIAGENDR']),5)
```

```
##      RIAGENDR
## [1,]    FALSE
## [2,]    FALSE
## [3,]    FALSE
```

```
## [4,]    FALSE
## [5,]    FALSE
```

To find the total number of missing values in a column:

```
sum(is.na(demo['RIAGENDR']))
```

```
## [1] 0
```

To find the number of missing values for each column, use the summary() function:

```
summary(demo)
```

If there is a missing value in one cell, is.na() will return **TRUE**; if there is no missing value in one cell, it will return **FALSE**

For example, (1,1) (1st row 1st column) is FALSE - it means that (1,1) is not a missing value; (1,6) (1st row 6th column) is TRUE - it means that (1,6) is NA.

**is.na()** is the function we use to inspect any missing values in a dataframe - it is housed in the base package.

The arguments are as follows:

is.na( > NAME OF DATAFRAME

)

**For example:** `is.na(demo)`

**DO QUESTION 1 OF THE QUIZ NOW** > What package does the function `nhanes()` belong to?

**DO QUESTION 2 OF THE QUIZ NOW**

> Which code below is the correct one to print the last 10 rows of a dataframe called temp?

**DO QUESTION 3 OF THE QUIZ NOW** > Suppose a cell contains NA. What is the output after perform is.na() on it?

## 5.4 2.Dataset preparation

Now that we've successfully imported the two datasets and we're ready to do further data manipulations. But before we proceed, there are three issues in the datasets:

- The two datasets are **Huge**. Recall the dimensions of the two datasets in section 1, there are 10175 rows and 47 columns in the demo dataset and 9813 rows and 23 columns in the bpx dataset.
  - For demostration purpose, this tutorial will only focus on parts of the two datasets.

        * For the demo dataset, we want to keep the following variables as our primary interests only:
            · *SEQN* (Respondent sequence number)
            · *RIAGENDR* (Gender)
            · *RIDAGEYR* (Age in years at screening)
            · *RIDRETH3* (Race)
            · *DMDEDUC2* (Education level - Adults 20+)
        * For the bpx dataset, we want to keep the following variables as our primary interests only:
            · *SEQN* (Respondent sequence number)
            · *PEASCST1* (Blood Pressure Status)
            · *PEASCTM1* (Blood Pressure Time in Seconds)
            · *BPXSY1* (Systolic: Blood pres (1st rdg) mm Hg)
            · *BPXDI1* (Diastolic: Blood pres (1st rdg) mm Hg)
        * For both datasets, we want to keep the top 5 rows only.

- There is something odd about the demo dataset. For example,if you run demo alone, you will see that *RIAGENDR* (gender) is coded as 1 and 2. For ease of future use, we want to translate this 1 and 2 into male and female.
- The variable names are long are ambiguous. For ease of further use, we want to rename the variables so that they can be easily understand.
    - For the demo dataframe, we want to rename the variables in this way:
        * SEQN -> id
        * RIAGENDR -> gender
        * RIDAGEYR -> age
        * RIDRETH3 -> race
        * DMDEDUC2 -> race
    - For the bpx dataframe, we want to rename the variables in this way:
        * SEQN -> id
        * PEASCST1 -> bp_status
        * PEASCTM1 -> bpt_sec
        * BPXSY1 -> systolic
        * BPXDI1 -> diastolic
    - **Note**, there are many different ways of renaming as long as the new names are straight-forward. We will use the above rename strategy in this tutorial for the sake of consistency.

With the three issues listed above, here's an example of how to resolve them in the demo dataset:

First, we need to translate the way of variable encoding using the `nhanesTranslate()` function:

```
demo_translate <- nhanesTranslate('DEMO_H',
                                  c('SEQN', # Respondent sequence number
                                    'RIAGENDR', # Gender
                                    'RIDAGEYR', # Age in years at screening
                                    'RIDRETH3', # Race
                                    'DMDEDUC2'), # Education level - Adults 20+
                                 data = demo)
```

```
## Warning in FUN(X[[i]], ...): No translation table is available for SEQN
```

```
## Translated columns: RIAGENDR RIDRETH3 DMDEDUC2
```

```
head(demo_translate,5)
```

```
##    SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1
## 1 73557        8        2     Male       69       NA        4
## 2 73558        8        2     Male       54       NA        3
## 3 73559        8        2     Male       72       NA        3
## 4 73560        8        2     Male        9       NA        3
## 5 73561        8        2   Female       73       NA        3
##             RIDRETH3 RIDEXMON RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN
## 1 Non-Hispanic Black        1       NA        1       1        1        1
## 2 Non-Hispanic White        1       NA        2      NA        1        1
## 3 Non-Hispanic White        2       NA        1       1        1        1
## 4 Non-Hispanic White        1      119       NA      NA        1        1
## 5 Non-Hispanic White        1       NA        2      NA        1        1
##   DMDYRSUS DMDEDUC3                          DMDEDUC2 DMDMARTL RIDEXPRG SIALANG
## 1       NA       NA High school graduate/GED or equi        4       NA       1
## 2       NA       NA High school graduate/GED or equi        1       NA       1
## 3       NA       NA        Some college or AA degree        1       NA       1
## 4       NA        3                              <NA>       NA       NA       1
## 5       NA       NA        College graduate or above        1       NA       1
##   SIAPROXY SIAINTRP FIALANG FIAPROXY FIAINTRP MIALANG MIAPROXY MIAINTRP
## 1        2        2       1        2        2       1        2        2
## 2        2        2       1        2        2       1        2        2
## 3        2        2       1        2        2       1        2        2
## 4        1        2       1        2        2       1        2        2
## 5        2        2       1        2        2       1        2        2
##   AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA DMDHHSZB DMDHHSZE DMDHRGND DMDHRAGE
## 1        1        3        3        0        0        2        1       69
## 2        1        4        4        0        2        0        1       54
## 3       NA        2        2        0        0        2        1       72
## 4        1        4        4        0        2        0        1       33
## 5       NA        2        2        0        0        2        1       78
##   DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU WTINT2YR WTMEC2YR SDMVPSU SDMVSTRA
```

```
## 1        1        3        4       NA 13281.24 13481.04        1       112
## 2        1        3        1        1 23682.06 24471.77        1       108
## 3        1        4        1        3 57214.80 57193.29        1       109
## 4        1        3        1        4 55201.18 55766.51        2       109
## 5        1        5        1        5 63709.67 65541.87        2       116
##   INDHHIN2 INDFMIN2 INDFMPIR
## 1        4        4     0.84
## 2        7        7     1.78
## 3       10       10     4.51
## 4        9        9     2.52
## 5       15       15     5.00
```

Second, we want to keep the variabls which we're interested in only:

```
new_demo <- select(demo_translate,
                   c(SEQN,
                    RIAGENDR,
                    RIDAGEYR,
                    RIDRETH3,
                    DMDEDUC2))
head(new_demo,5)
```

```
##    SEQN RIAGENDR RIDAGEYR           RIDRETH3                         DMDEDUC2
## 1 73557     Male       69 Non-Hispanic Black High school graduate/GED or equi
## 2 73558     Male       54 Non-Hispanic White High school graduate/GED or equi
## 3 73559     Male       72 Non-Hispanic White        Some college or AA degree
## 4 73560     Male        9 Non-Hispanic White                             <NA>
## 5 73561   Female       73 Non-Hispanic White        College graduate or above
```

Third, we want to rename the variables using the `rename()` function:

```
new_demo <- rename(new_demo,
       id = SEQN, # Respondent sequence number
       gender = RIAGENDR, # Gender
       age = RIDAGEYR, # Age in years at screening
       race = RIDRETH3, # Race/Hispanic origin
       edu = DMDEDUC2, # Education level - Adults 20+
       )
head(new_demo,5)
```

```
##      id gender age               race                              edu
## 1 73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 2 73558   Male  54 Non-Hispanic White High school graduate/GED or equi
## 3 73559   Male  72 Non-Hispanic White        Some college or AA degree
## 4 73560   Male   9 Non-Hispanic White                             <NA>
## 5 73561 Female  73 Non-Hispanic White        College graduate or above
```

Last, we want to keep the top 10 rows using the `head()` function and save the dataframe as `final_demo`:

```
final_demo <- head(new_demo,10)
final_demo
```

```
##       id gender age               race                            edu
## 1  73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 2  73558   Male  54 Non-Hispanic White High school graduate/GED or equi
## 3  73559   Male  72 Non-Hispanic White        Some college or AA degree
## 4  73560   Male   9 Non-Hispanic White                             <NA>
## 5  73561 Female  73 Non-Hispanic White        College graduate or above
## 6  73562   Male  56   Mexican American        Some college or AA degree
## 7  73563   Male   0 Non-Hispanic White                             <NA>
## 8  73564 Female  61 Non-Hispanic White        College graduate or above
## 9  73565   Male  42     Other Hispanic High school graduate/GED or equi
## 10 73566 Female  56 Non-Hispanic White High school graduate/GED or equi
```

#### 5.4.0.1 Functions debunked

**nhanesTranslate** is the function we use to translate variables in a dataset - it is housed in the nhanesA package. The arguments are as follows:

nhanesTranslate( > 'NAME OF DATASET',

> COLUMNS YOU WANT TO BE TRANSLATED, (can be written as a vector)
>
> data = SOURCE DATAFRAME

)

**For example:** `nhanesTranslate('DEMO_H', RIAGENDR, data = demo)`

**rename()** is the function we use to rename the variables and return all variables - it is housed in the dplyr package. The arguments are as follows:

rename( > NAME OF DATAFRAME,

> NEW_VARIABLE_NAME = OLD_VARIABLE_NAME

)

**For example:** `rename(demo, id = SEQN)`

**select()** is the function we use to only return the varibles we want - it is housed in the dplyr package. The arguments are as follows:

select( > NAME OF DATAFRAME,

> VARIABLES, (can be written as a vector)

)

**For example:** `select(demo, c(id,gender,age))`

**For example:** `select(demo, id:age))`

### 5.4.1 4.1 Try it yourself

In the bpx dataframe, keep the following variables and top 5 rows only:

- SEQN
- PEASCST1
- PEASCTM1
- BPXSY1
- BPXDI1

### 5.4.2 4.2 Try it yourself

rename the variables in the following way:

- SEQN -> id
- PEASCST1 -> bp_status
- PEASCTM1 -> bpt_sec
- BPXSY1 -> systolic
- BPXDI1 -> diastolic

**DO QUESTION 4 OF THE QUIZ NOW** > Given the following code: > select(dataset,A,B) > What is the purpose of the code?

**DO QUESTION 5 OF THE QUIZ NOW** > Which code below is the correct one to rename column A to B in a dataframe called dataset?

## 5.5 3. Filter

Sometimes, we want to focus on a subset of the dataset that satisifying some conditions for further analysis. In this case, we need to filter the dataset based on variables' values and conditions.

For example, we use the following code to filter the observations that patients are 40 years old at screening.

```
filter(final_demo, age == 40)
```

```
## [1] id     gender age    race   edu
## <0 rows> (or 0-length row.names)
```

We may also interested in the observations that patients are 40 **or** 41 years old at screening:

```
filter(final_demo, age == 40 | age == 41)
```

```
## [1] id     gender age    race   edu
## <0 rows> (or 0-length row.names)
```

Another way of writing the code above is to use the `%in%` operator:

`x %in% y` is equivalent to the condition that **the value of x is in one of the values of y**

```
filter(final_demo, age %in% c(40, 41))
```

```
## [1] id     gender age    race   edu
## <0 rows> (or 0-length row.names)
```

#### 5.5.0.1 Functions debunked

**filter()** is the function we use to subset the dataset based on their values and conditions - it is housed in the dplyr package. The arguments are as follows:

filter( > NAME OF DATASET,

CONDITION 1

CONDITION 2

...

CONDITION n

)

**For example:** `filter(new_demo, gender == 1)`

**Note:**

1, pay attention to the difference between `=` and `==` > `=` is assignment operator

`==` is comparison operator

More comparison operators are `<`, `<=`,`>`, `>=`,`!=`.

2, logical operators

Mutiple conditions can be combined using logical operators. Common logical operators are:

`and` is `&`

`or` is `|`

`not` is `!`

**For example:** `filter(new_demo, age == 40 | age == 41)`

#### 5.5.0.2 Missing values

Note, we don't have any missing values in the age column (can be checked using `is.na(new_demo['age'])`).

What if we have missing values and how does filter() treat missing values?

filter() only keeps the rows where the condition is **TRUE** and remove the rows where the condition is failed due to **FALSE OR NA**.

If you want to keep the NAs, you need to add the condition explicitly: `is.na(VARIABLE_NAME)` and use `|` to combine with other conditions.

**For example:** `filter(new_demo, is.na(income) | income <= 50)`

### 5.5.1 4.3 Try it yourself

In the demo dataframe, find all the records that:

4.3.1 the participant who is a male

4.3.2 the participant who is a male and is older tha 50 years old

4.3.3 the education level is missing

**DO QUESTION 6 OF THE QUIZ NOW** > Fill in the blank to answer '4.3.1 Try it yourself':

> filter(final_demo, gender ___)

**DO QUESTION 7 OF THE QUIZ NOW** > Fill in the blank to answer the second question in '4.3.2 Try it yourself':

> filter(final_demo, gender == 'Male' ___ age > 50)

**DO QUESTION 8 OF THE QUIZ NOW** > Fill in the blank to answer the third question in '4.3.3 Try it yourself':

> filter(final_demo, ___)

## 5.6 4.Re-order the rows

Suppose we want to re-order the observations by certain variables, we can use the arrange() function in R.

First, let's look at the top 5 rows in the demo dataframe:

```
final_demo
```

```
##       id gender age               race                              edu
## 1  73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 2  73558   Male  54 Non-Hispanic White High school graduate/GED or equi
## 3  73559   Male  72 Non-Hispanic White        Some college or AA degree
## 4  73560   Male   9 Non-Hispanic White                             <NA>
## 5  73561 Female  73 Non-Hispanic White        College graduate or above
## 6  73562   Male  56   Mexican American        Some college or AA degree
## 7  73563   Male   0 Non-Hispanic White                             <NA>
## 8  73564 Female  61 Non-Hispanic White        College graduate or above
## 9  73565   Male  42     Other Hispanic High school graduate/GED or equi
```

```
## 10 73566 Female  56 Non-Hispanic White High school graduate/GED or equi
```

We notice that the column "age" doesn't follow any order. We use the following code to re-order the dataframe by age:

```
arrange(final_demo, age)
```

```
##       id gender age               race                          edu
## 1  73563   Male   0 Non-Hispanic White                         <NA>
## 2  73560   Male   9 Non-Hispanic White                         <NA>
## 3  73565   Male  42     Other Hispanic High school graduate/GED or equi
## 4  73558   Male  54 Non-Hispanic White High school graduate/GED or equi
## 5  73562   Male  56   Mexican American        Some college or AA degree
## 6  73566 Female  56 Non-Hispanic White High school graduate/GED or equi
## 7  73564 Female  61 Non-Hispanic White       College graduate or above
## 8  73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 9  73559   Male  72 Non-Hispanic White        Some college or AA degree
## 10 73561 Female  73 Non-Hispanic White       College graduate or above
```

If we want to change it to descending order, use desc():

```
arrange(final_demo, desc(age))
```

```
##       id gender age               race                          edu
## 1  73561 Female  73 Non-Hispanic White       College graduate or above
## 2  73559   Male  72 Non-Hispanic White        Some college or AA degree
## 3  73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 4  73564 Female  61 Non-Hispanic White       College graduate or above
## 5  73562   Male  56   Mexican American        Some college or AA degree
## 6  73566 Female  56 Non-Hispanic White High school graduate/GED or equi
## 7  73558   Male  54 Non-Hispanic White High school graduate/GED or equi
## 8  73565   Male  42     Other Hispanic High school graduate/GED or equi
## 9  73560   Male   9 Non-Hispanic White                         <NA>
## 10 73563   Male   0 Non-Hispanic White                         <NA>
```

What if we want to change the order by multiple columns?

We can simply add multiple columns in the arguments! It's also useful when there are ties in the values of one column and the subsquent columns are used to break the ties.

For example, re-order the observations by age,id:

```
arrange(final_demo,age,id)
```

```
##       id gender age               race                          edu
## 1  73563   Male   0 Non-Hispanic White                         <NA>
## 2  73560   Male   9 Non-Hispanic White                         <NA>
## 3  73565   Male  42     Other Hispanic High school graduate/GED or equi
## 4  73558   Male  54 Non-Hispanic White High school graduate/GED or equi
```

```
## 5  73562   Male  56   Mexican American       Some college or AA degree
## 6  73566 Female  56 Non-Hispanic White High school graduate/GED or equi
## 7  73564 Female  61 Non-Hispanic White       College graduate or above
## 8  73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 9  73559   Male  72 Non-Hispanic White       Some college or AA degree
## 10 73561 Female  73 Non-Hispanic White       College graduate or above
```

#### 5.6.0.1 Functions debunked

**arrange()** is the function we use to change the order of the observations by columns - it is housed in the dplyr package. The arguments are as follows:

arrange( > NAME OF DATAFRAME,

COLUMN 1

COLUMN 2

...

COLUMN n

)

**For example:** `arrange(new_demo, gender)`

Use `desc(COLUMN_NAME)` to reorder the dataframe by *COLUMN_NAME* in descending order.

#### 5.6.0.2 Missing values

In the last example, we notice that there are missing values in column 'income' and 'income_ratio'.

How does arrange() deal with missing value? **All NAs will be retained at the end.** Check the output above!

### 5.6.1 4.4 Try it yourself

Re-order the rows in the bpx dataset by Blood Pressure Time in Seconds (bpt_sec) in descending order:

**DO QUESTION 9 OF THE QUIZ NOW** > Fill in the blank to answer the question in '4.4 Try it yourself':

`___`(final_bpx,`___`)

## 5.7 5. Add new variables

Suppose we want to gain more information about our observations and we want to constrcut a new variable based on the variables we have. The mutate() and transmute() in R helps us to add new variables into a dataframe.

For example, we want to add a new variable called born_year in the demo dataframe and born_year is calculated in this way: born_year = 2021- age:

```
mutate(final_demo,
       born_year = 2021 - age
      )
```

```
##       id gender age               race                              edu
## 1  73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 2  73558   Male  54 Non-Hispanic White High school graduate/GED or equi
## 3  73559   Male  72 Non-Hispanic White        Some college or AA degree
## 4  73560   Male   9 Non-Hispanic White                             <NA>
## 5  73561 Female  73 Non-Hispanic White        College graduate or above
## 6  73562   Male  56   Mexican American        Some college or AA degree
## 7  73563   Male   0 Non-Hispanic White                             <NA>
## 8  73564 Female  61 Non-Hispanic White        College graduate or above
## 9  73565   Male  42     Other Hispanic High school graduate/GED or equi
## 10 73566 Female  56 Non-Hispanic White High school graduate/GED or equi
##    born_year
## 1       1952
## 2       1967
## 3       1949
## 4       2012
## 5       1948
## 6       1965
## 7       2021
## 8       1960
## 9       1979
## 10      1965
```

If we want to keep the added variables only, use transmute() instead:

```
transmute(final_demo,
    born_year = 2021 - age
     )
```

```
##    born_year
## 1       1952
## 2       1967
## 3       1949
## 4       2012
## 5       1948
## 6       1965
## 7       2021
## 8       1960
## 9       1979
```

```
## 10       1965
```

#### 5.7.0.1 Functions debunked

**mutate()** is the function we use to create new variables based on variables we have and add them to the original dataframe - it is housed in the dplyr package. The arguments are as follows:

mutate( > NAME OF DATAFRAME,

NEW_VARIABLE = FUNCTION OF EXISTING VARIBALES

... )

**For example:** `mutate(new_demo, rescale_income = income/2)`

**transmute()** is the function we use to create new variables based on variables we have and only keep the added variables - it is housed in the dplyr package. The arguments are as follows:

transmute( > NAME OF DATAFRAME,

NEW_VARIABLE = FUNCTION OF EXISTING VARIBALES

...

)

**For example:** `transmute(new_demo, rescale_income = income/2)`

You can find more details in how to use creation functions to create new variables in Chapter I. Explore Chapter 3.

### 5.7.1 4.5 Try it yourself

4.5.1. Create a new variable called called **rescale_bpt_sec** that records the Blood Pressure Time in miuntes. Keep both original and new variables.

4.5.2. Create **rescale_bpt_sec** in the same way above and **only keep new variables**.

Note: Try to avoid using select().

**DO QUESTION 10 OF THE QUIZ NOW** > What are the `?`s to answer '4.5.1 & 4.5.2 Try it yourself'?

`_?_`(final_bpx,

rescale_bpt_sec = bpt_sec/60

)

## 5.8 6.Summary statistics and group_by

Information about the whole dataframe such as mean is very useful for data analysis. However, things could be very complex when aggregating multiple functions. We'll start with the simple one: find a summary statistic from the whole dataset.

For example, find the average age in the whole demo dataframe:

```
summarize(final_demo,average_age = mean(age,na.rm = TRUE))
```

```
##   average_age
## 1        49.2
```

It gets more complex when we change the unit of analysis into groups, i.e, find summary statistics grouping by variables. To do so, we first convert the dataframe into a grouped dataframe using group_by() and then apply the summarize() to the grouped dataframe.

For example, find the average age in the demo dataframe per gender:

```
by_gender <- group_by(final_demo,gender)
summarize(by_gender, average_age = mean(age, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##   gender average_age
##   <fct>        <dbl>
## 1 Male          43.1
## 2 Female        63.3
```

#### 5.8.0.1 Functions debunked

**summarize()** is the function we use to compute the summary statistics for the whole dataframe - it is housed in the dplyr package. The arguments are as follows:

summarize( > NAME OF DATAFRAME,

NAME OF SUMMARY STATISTIC = FUNCTION()

)

**For example:** `summarize(new_demo, mean(age,na.rm = TRUE))`

**group_by()** is the function we use to create a grouped dataframe grouping by one or more variables - it is housed in the dplyr package. The arguments are as follows:

group_by( > NAME OF DATAFRAME,

NAME OF VARIABLE

)

**For example:** `group_by(new_demo,age)` **Note:** if there is missing value in the variable, group_by() treats it as a new group

### 5.8.1 4.6 Try it yourself

Find the average age in the demo dataframe per education level

**DO QUESTION 11 OF THE QUIZ NOW** > Fill in the blanks to answer the question in '4.6 Try it yourself':

> by_edu <- group_by(___)
>
> summarize(___, average_age = ___(age, na.rm = TRUE))

### 5.8.2 Group_by() extension

group_by() is also useful when conjuncting with filter() and mutate().

For example, return the observations which has more than 2 records in each education group.

```
by_edu <-group_by(final_demo,edu)
filter(by_edu,n() > 2)
```

```
## # A tibble: 4 x 5
## # Groups:   edu [1]
##   id          gender age         race               edu
##   <labelled> <fct>  <labelled> <fct>              <fct>
## 1 73557      Male   69         Non-Hispanic Black High school graduate/GED or e~
## 2 73558      Male   54         Non-Hispanic White High school graduate/GED or e~
## 3 73565      Male   42         Other Hispanic     High school graduate/GED or e~
## 4 73566      Female 56         Non-Hispanic White High school graduate/GED or e~
```

Here's another example: using group_by() with mutate() to compute the difference in each age and the average mean in each gender group

```
by_gender <-group_by(final_demo,gender)
mutate(by_gender,
      diff_age = age - mean(age, na.rm = T))
```

```
## # A tibble: 10 x 6
## # Groups:   gender [2]
##    id         gender age        race           edu                          diff_age
##    <labelle> <fct>  <labelle> <fct>          <fct>                        <labelled>
##  1 73557     Male   69        Non-Hispanic ~ High school graduate/GE~  25.857143
##  2 73558     Male   54        Non-Hispanic ~ High school graduate/GE~  10.857143
##  3 73559     Male   72        Non-Hispanic ~ Some college or AA degr~  28.857143
##  4 73560     Male    9        Non-Hispanic ~ <NA>                     -34.142857
```

```
## 5 73561     Female 73        Non-Hispanic ~ College graduate or abo~   9.666667
## 6 73562     Male   56        Mexican Ameri~ Some college or AA degr~  12.857143
## 7 73563     Male    0        Non-Hispanic ~ <NA>                     -43.142857
## 8 73564     Female 61        Non-Hispanic ~ College graduate or abo~  -2.333333
## 9 73565     Male   42        Other Hispanic High school graduate/GE~  -1.142857
## 10 73566    Female 56        Non-Hispanic ~ High school graduate/GE~  -7.333333
```

#### 5.8.2.1 Functions debunked

**n()** is the function we use to count the number of observations in each group - it is housed in the dplyr package. It can only be used with the existence of summarize(), filter(), and mutate() and there is **no** argument in it.

### 5.8.3 4.7 Try it yourself

Return the observations which has more than 3 records in each gender group.

### 5.8.4 4.8 Try it yourself

Compute the difference in each age and the average mean in each education level group

**DO QUESTION 12 OF THE QUIZ NOW** > Fill in the blanks to answer '4.7 Try it yourself':

> by_gender <-group_by(final_demo,gender)
>
> filter(by_edu,___)

**DO QUESTION 13 OF THE QUIZ NOW** > Fill in the blanks to answer '4.8 Try it yourself':

> by_edu <-group_by(final_demo,edu)
>
> mutate(by_edu, ___)

## 5.9 7. Pipe

You may have noticed that when doing multiple-step operations, we need to assign the output to a new variable every time when a step is done. It becomes more annoying when there are more steps to do. The pipe operator %>% - which is housed in the magrittr package - saves us from create many unnecessary variables: it takes the output from one function as an input to the following function.

Here's example without using pipe: we want to first keep the observations in the demo dataframe with an age greater than 40 and then create a new variable called born_year calculated by 2021 - age.

We need to do this in two steps.The first step is to filter the dataframe and save the output as a new variable *temp*:

```
temp <- filter(final_demo, age > 40)
temp
```

```
##      id gender age               race                            edu
## 1 73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 2 73558   Male  54 Non-Hispanic White High school graduate/GED or equi
## 3 73559   Male  72 Non-Hispanic White        Some college or AA degree
## 4 73561 Female  73 Non-Hispanic White        College graduate or above
## 5 73562   Male  56   Mexican American        Some college or AA degree
## 6 73564 Female  61 Non-Hispanic White        College graduate or above
## 7 73565   Male  42     Other Hispanic High school graduate/GED or equi
## 8 73566 Female  56 Non-Hispanic White High school graduate/GED or equi
```

The second step is to create the new variable:

```
temp <- mutate(temp, born_year = 2021 - age)
temp
```

```
##      id gender age               race                            edu
## 1 73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 2 73558   Male  54 Non-Hispanic White High school graduate/GED or equi
## 3 73559   Male  72 Non-Hispanic White        Some college or AA degree
## 4 73561 Female  73 Non-Hispanic White        College graduate or above
## 5 73562   Male  56   Mexican American        Some college or AA degree
## 6 73564 Female  61 Non-Hispanic White        College graduate or above
## 7 73565   Male  42     Other Hispanic High school graduate/GED or equi
## 8 73566 Female  56 Non-Hispanic White High school graduate/GED or equi
##   born_year
## 1      1952
## 2      1967
## 3      1949
## 4      1948
## 5      1965
## 6      1960
## 7      1979
## 8      1965
```

The intermediate variable *temp* can be avoided by using the pipe operatore:

```
final_demo %>%
  filter(age > 40) %>%
  mutate(born_year = 2021 - age)
```

```
##      id gender age               race                            edu
## 1 73557   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 2 73558   Male  54 Non-Hispanic White High school graduate/GED or equi
```

```
## 3 73559   Male  72 Non-Hispanic White        Some college or AA degree
## 4 73561 Female  73 Non-Hispanic White        College graduate or above
## 5 73562   Male  56   Mexican American        Some college or AA degree
## 6 73564 Female  61 Non-Hispanic White        College graduate or above
## 7 73565   Male  42     Other Hispanic High school graduate/GED or equi
## 8 73566 Female  56 Non-Hispanic White High school graduate/GED or equi
##   born_year
## 1      1952
## 2      1967
## 3      1949
## 4      1948
## 5      1965
## 6      1960
## 7      1979
## 8      1965
```

### 5.9.1 4.9 Try it yourself

Re-write the following code using pipe operator

```
# temp <- filter(final_bpx, systolic > 120)
# temp <- mutate(temp, bpt_min = bpt_sec/60)
# temp
```

**DO QUESTION 14 OF THE QUIZ NOW** > Fill in the blanks to answer the question in '4.9 Try it yourself':

> ___ %>% filter(___,systolic > 120) %>% mutate(___,bpt__min = bpt__sec/60)

## 5.10 8. Summary of dealing with missing values

Dealing with missing values can be complex. You need to be careful when using functions since different functions have different ways in dealing with missing values.

filter(): excludes missing values. If you do want to retain missing values, use **is.na()** explicitly.

arrange(): retains missing values and sorts them at the end

select(): retains missing values

mutate(): retains missing values

summarize(): retains missing values. If you want to remove missing values, set **na.rm = TRUE** in aggregation functions.

group__by(): treats missing values as a group

## 5.11 ALTERNATIVES TO NHANESTRANSLATE()

### 5.11.1 case_when()

`case_when()` is another useful function that we can use that will aid our data analysis. This function acts like `mutate()`, except it changes the actual values of the variable. Here is an example of how we can use `case_when()` and `mutate()` together to change values within a variable:

```
x <- c(97, 36, 55, 50, 49, 65, 46, 87, 100)
```

```
case_when(x < 50 ~ "Fail",
          x > 65 ~ "Pass",
          TRUE  ~ "Retake recommended")
```

```
## [1] "Pass"               "Fail"               "Retake recommended"
## [4] "Retake recommended" "Fail"               "Retake recommended"
## [7] "Fail"               "Pass"               "Pass"
```

In the example above, we have a vector of student grades. Using `case_when()`, we have translated the grades to "Fail", "Pass", and "Retake recommended" depending on the student grades. * **Fail** if students receive a grade of less than 50, * **Pass** if students receive a grade of greater than 65, * **Retake recommended** if students receive a grade between 50 and 65.

## 5.12 Translating NHANES using case_when()

Another general method for us to translate our NHANES data into conventional language is to use `mutate()` and `case_when()`! However, this method requires us to know what each numerical value of each variable means. For example, if we want to translate the RIDRETH3 values, then we need to check the Codebook and Frequencies for the following translations:

- 1: "Mexican American"
- 2: "Other Hispanic"
- 3: "Non-Hispanic White"
- 4: "Non-Hispanic Black"
- 6: "Non-Hispanic Asian"
- 7: "Other Race - Including Multi-Racial"
- .: "Missing"

Knowing this, we can use `mutate()` and `case_when()` like so:

```
translated_demo <- demo %>%
    mutate(Race = case_when(
          RIDRETH3 == 1 ~ "Mexican American",
          RIDRETH3 == 2 ~ "Other Hispanic",
          RIDRETH3 == 3 ~ "Non-Hispanic White",
          RIDRETH3 == 4 ~ "Non-Hispanic Black",
```

```
          RIDRETH3 == 6 ~ "Non-Hispanic Asian",
          RIDRETH3 == 7 ~ "Other Race - Including Multi-Racial",
          RIDRETH3 == "." ~ "Missing"
      )) %>%
    select(ID = SEQN, Race)
```

```
head(translated_demo, 10)
```

```
##       ID               Race
## 1  73557 Non-Hispanic Black
## 2  73558 Non-Hispanic White
## 3  73559 Non-Hispanic White
## 4  73560 Non-Hispanic White
## 5  73561 Non-Hispanic White
## 6  73562   Mexican American
## 7  73563 Non-Hispanic White
## 8  73564 Non-Hispanic White
## 9  73565     Other Hispanic
## 10 73566 Non-Hispanic White
```

But what if we want to have less categories or want to combine some of the categories? What do we do then?

`case_when()` is still the way to go! In this case, our codes should look like this:

```
less_categories <- demo %>%
    mutate(Race = case_when(
          RIDRETH3 == 1 ~ "Hispanic",
          RIDRETH3 == 2 ~ "Hispanic",
          RIDRETH3 == 3 ~ "White",
          RIDRETH3 == 4 ~ "Black",
          RIDRETH3 == 6 ~ "Asian",
          RIDRETH3 == 7 ~ "Other",
          RIDRETH3 == "." ~ "Missing"
      )) %>%
    select(ID = SEQN, Race)
```

```
head(less_categories, 10)
```

```
##       ID     Race
## 1  73557    Black
## 2  73558    White
## 3  73559    White
## 4  73560    White
## 5  73561    White
## 6  73562 Hispanic
## 7  73563    White
## 8  73564    White
```

```
## 9  73565 Hispanic
## 10 73566    White
```

### 5.12.1 Functions debunked

**case_when()** is a function that we use to change or translate the values of our variables into something else that is still meaningful. The arguments are as follows:

case_when( > **A VARIABLE == value as it is written in the original dataset ~ "NEW/TRANSLATED VALUE"** )

Note that if none of the cases match, then R will automatically regard it as a missing (NA) value.

**For example:** `case_when(x < 50 ~ "Fail", x > 65 ~ "Pass", TRUE ~ "Retake recommended")`

## 5.13 `recode()` from dplyr

Another alternative is `recode()`. This function works the same way as `case_when()`, except you only need to identify the variable once like so:

```
# demo_translate2 <- demo %>%
#     mutate(Race = dplyr::recode(RIDRETH3,
#            `1` = "Hispanic",
#            `2` = "Hispanic",
#            `3` = "White",
#            `4` = "Black",
#            `6` = "Asian",
#            `7` = "Other",
#           .default = "Missing"
#     )) %>%
#     select(ID = SEQN, Race)
```

```
# head(demo_translate2, 10)
```

## 5.14 `recode()` from car

Another option is to use `recode()` from the car package. This function is mostly used for translating numerical data into more meaningful character data or strings.

```
library(car)
```

```
## Loading required package: carData
```

```
##
```

```
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

While looking at the codes below, take note of the different types of quotation marks: single (' ') or double ("") and the semi-colon (;) as the argument separator. Note also how we can use `1:2` instead of separating them into 1 and 2 like what we did with earlier functions.

```
demo_translate3 <- demo %>%
    mutate(Race = car::recode(RIDRETH3,
        "1:2 = 'Hispanic';
         3 = 'White';
         4 = 'Black';
         6 = 'Asian';
         7 = 'Other';
         else = 'Missing'"
    )) %>%
    select(ID = SEQN, Race)
```

```
head(demo_translate3, 10)
```

```
##       ID     Race
## 1  73557    Black
## 2  73558    White
## 3  73559    White
## 4  73560    White
## 5  73561    White
## 6  73562 Hispanic
## 7  73563    White
## 8  73564    White
## 9  73565 Hispanic
## 10 73566    White
```

## 5.15 TAKEAWAYS

By the end of this tutorial, you should be familiar with the dplyr package and be able to do basic data wrangling by yourself.

For more study materials on the dplyr package, check out this textbook.

# Chapter 6

# Data Visualization with ggplot2

## 6.1 INSTRUCTIONS

This tutorial will introduce us to data visualization on R, specially using the ggplot R package. For this purpose, we will focus on the basic and most commonly used functions of the ggplot package. We will be able to read through the step-by-step instructions on how to use each function as well as its different arguments.

Accompanying this tutorial is **a short Google quiz** for your own self-assessment. The instructions of this tutorial will clearly indicate when you should answer which question.

## 6.2 LEARNING OBJECTIVES

- Be familiar with the basics of ggplot including how to graph basic scatterplot (with smoothed conditional means), line graph, bar graph, and bar chart using geometric functions.
- Get a brief understanding of coordinate functions with special emphasis on `coord_flip()` and `coord_polar()`.
- Be able to create gridded subplots using facet functions.
- Know how to customize a graph to our own liking - including changing the texts, font, size, and color.
- Know how to export the final graph into a png file using `ggsave()`.

## 6.3 1. SET UP

### 6.3.1 Loading required packages

For this tutorial, we will only be focusing on the ggplot2 package. But we will also be using a few functions from the readr and dplyr packages from the previous tutorials.

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

#install.packages("readr")
library(readr)

#install.packages("ggplot2")
library(ggplot2)
```

### 6.3.2 Importing Data

Like we have learned in the previous tutorial, the first step after loading the required packages is to import the data that we will be working with! In this tutorial, we will be working with the same Demographics and Blood Pressure datasets from NHANES. However, to make things easier for us, a dataset consisting of both Demographics and Blood Pressure information have already been created, translated, and combined. As a challenge (this is completely optional), you can try to recreate this data frame on your own! Here is more information about the data frame we will be using for this tutorial: * Its name is "demo_bpx.csv" - it is a csv file * It contains the Respondent Sequence Number of each participant, along with their reported Gender, Race, Systolic and Diastolic Blood Pressures, and Blood Pressure Time in seconds. The first three are from the DEMO_H dataset, the rest are from the BPX_H dataset. * It only contains information of the first 200 participants. * The first column (X1) is automatically added by R.

```
demo_bpx <- read_csv("data/demo_bpx.csv")
```

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

## Rows: 210 Columns: 7

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): Gender, Race
## dbl (5): ...1, ID, Systolic, Diastolic, BPT

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(demo_bpx)
```

```
## # A tibble: 6 x 7
##     ...1    ID Gender Race               Systolic Diastolic   BPT
##    <dbl> <dbl> <chr>  <chr>                 <dbl>     <dbl> <dbl>
## 1      1 73557 Male   Non-Hispanic Black      114        76   620
## 2      2 73558 Male   Non-Hispanic White      160        80   766
## 3      3 73559 Male   Non-Hispanic White      140        76   665
## 4      4 73560 Male   Non-Hispanic White      102        34   803
## 5      5 73561 Female Non-Hispanic White      134        88   949
## 6      6 73562 Male   Mexican American        158        82  1064
```

#### 6.3.2.1 DO QUESTION 1 OF THE QUIZ NOW

> **REVIEW** What is the name of the core that the packages readr, dplyr, and ggplot2 belong to?

## 6.4 2. GGPLOT AND POINT GEOMETRICS

Now that our dataset is all set up, it's time to plot our first graph! First, we need to start with an empty canvas and to do this, we need `ggplot()` as our base. Try running `ggplot()` alone. What do you see?

```
ggplot()
```

intro2R_files/figure-latex/unnamed-chunk-183-1.pdf

If you answered "nothing" then you are completely right! Again, `ggplot()` alone only acts as a blank canvas. Usually, we would only have the dataset that we want to plot in the `()` of `ggplot()`. For example:

```
ggplot(demo_bpx)
```

intro2R_files/figure-latex/unnamed-chunk-184-1.pdf

As for the actual graph, in order to plot it, we need geometric functions.

### 6.4.1 Point Geometrics

Point geometrics, `geom_point()`, lets you graph scatterplots. The most basic argument that you can nest in `geom_point()` is `aes(x, y)` which basically tells `geom_point()` the x and y variables that you want to graph! `aes` stands for "aesthetics", we will cover more of this later in this tutorial.

```
ggplot(demo_bpx) +
    geom_point(aes(x = Systolic,
                   y = Diastolic),
               na.rm = TRUE,
               show.legend = TRUE)
```

`intro2R_files/figure-latex/unnamed-chunk-185-1.pdf`

As you can see, point geometrics (and all other geometric functions) always have to go after our `ggplot()` function. In addition, note that `ggplot()` and `geom_point()` are separated by a `+`.

#### 6.4.1.1 Functions debunked

**ggplot** is our blank canvas - this function is housed in the ggplot2 package! The arguments are as follows:

ggplot( > **NAME OF DATAFRAME**,

> aes(x = **VARIABLE ON THE X AXIS**, y = **VARIABLE ON THE Y AXIS**)

)

**geom_point** is the function we use to draw scatterplots - it is also housed in the ggplot2 package. The arguments that will be covered in this tutorial are as follows - you are welcomed to explore this function in more detail on your own:

geom_point( > aes(**AESTHETICS** - to be covered later in the tutorial),

> na.rm = **TRUE OR FALSE**,
>
> show.legend = **TRUE OR FALSE**

)

**For example:** `ggplot(demo_bpx) + geom_point(aes(x = Systolic, y = Diastolic, color = Gender), na.rm = TRUE)`

#### 6.4.1.2 DO QUESTIONS 2 & 3 OF THE QUIZ NOW

**HINT**: We've covered how to look for help within and outside of R in our very first tutorial: Basics of R and RStudio.

> What do you think the argument `na.rm = TRUE` does?
>
> What do you think the argument `show.legend = TRUE` does?

### 6.4.2 1.1 Try it yourself

Plot a scatterplot to show the relationship between Diastolic Blood Pressure (x-axis) and Blood Pressure Time in Seconds (y-axis).

#### 6.4.2.1 DO QUESTION 4 OF THE QUIZ NOW

> What does the "Try it yourself" graph above look like?

### 6.4.3 Aesthetics

Besides the x and y axes, there are other aesthetics that you can use to customize your graph as well! For example, you can change the color, size, opacity, and shape of your data points based on a particular variable of the dataset!

#### 6.4.3.1 Colors

We can tell ggplot to use different colors for different genders using the argument `color =` like so:

```
ggplot(demo_bpx) +
  geom_point(aes(x = Systolic,
                 y = Diastolic,
                 color = Gender),
             na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-186-1.pdf

**6.4.3.1.1 Missing Values** Looking at the graph above, we can see that there are a few NA data points for the variable gender. Let's rename all of these NA values to "Unstated", instead, to more accurately represent our data. To do this, we need to use the `replace_na()` function from the tidyr package like so:

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

```
demo_bpx %>%
    replace_na(list(Gender = "Unstated")) %>%
    ggplot() +
        geom_point(aes(x = Systolic,
                       y = Diastolic,
                       color = Gender),
                   na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-188-1.pdf

#### 6.4.3.2 Shapes

We can also use different shapes to represent the different genders in our dataset. To do this, we use the argument `shapes =`. This argument is more appropriate to use this argument when we are trying to distinguish between discrete variables since there are no in-between shapes to accurately reflect continuous variables!

```
ggplot(demo_bpx) +
  geom_point(aes(x = Systolic,
                 y = Diastolic,
                 shape = Gender),
             na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-189-1.pdf

#### 6.4.3.3 Size

You can also change the size of your data points using the argument `size =` and then any number. You can also use this argument to distinguish data points of different genders with different point sizes, but this is not recommended. If you want to plot points of different sizes, it is most appropriate if you use it to distinguish a particular continuous variable.

In the example below, not how the argument `size = 2` is outside of the aesthetics bracket. This tells R that we want ALL of our data points to be of the same size 2.

```
ggplot(demo_bpx) +
  geom_point(aes(x = Systolic, y = Diastolic),
             size = 2,
             na.rm = TRUE,)
```

intro2R_files/figure-latex/unnamed-chunk-190-1.pdf

#### 6.4.3.4 Opacity

Similar to size, if using opacity as an indication of different categories of a variable is most appropriate if the variable is continuous. So in the example below, the graph shows all data points with the same opacity because the argument `alpha = 11` is outside of the aesthetics brackets.

Also note that alpha values range from 0 to 1. The other neat thing about changing the data points opacity is that we can identify where data points overlap. For example, in the graph below, you can see some data points are darker than others. This means that those data points have multiple replicates!

```
ggplot(demo_bpx) +
  geom_point(aes(x = Systolic, y = Diastolic),
             alpha = 0.5,
             na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-191-1.pdf

Why do you think some point aesthetics better demonstrate discrete variables while others better demonstrate continuous variables?

#### 6.4.3.5 Jitter Position

After the graph above, you, hopefully, should have noticed that there are a lot of overlapping points in our dataset. To address this issue of overplotting, we can add random noise to each point to spread points out because no two points are likely to have same random noise. We can do this by nesting the position/argument jitter to our scatterplot like so:

```
ggplot(demo_bpx) +
  geom_point(aes(x = Systolic,
                 y = Diastolic,
                 color = Gender),
             na.rm = TRUE,
             position = "jitter")
```

intro2R_files/figure-latex/unnamed-chunk-192-1.pdf

### 6.4.4 1.2 Try it yourself

Plot a scatterplot using the Diastolic (x-axis) and Systolic (y-axis) variables in the data frame demo_bpx where all of the data points are blue.

#### 6.4.4.1 DO QUESTION 5 OF THE QUIZ NOW

Which is the correct code for the question above?

## 6.5 3. MULTIPLE GEOMETRIC FUNCTIONS UNDER ONE GGPLOT

Now that you're more familiar with `ggplot()` and `geom_point()`, we can try layering multiple graph types in one single ggplot canvas! For example, we can layer a line graph on top of a scatterplot.

```
ggplot(demo_bpx) +
  geom_point(aes(x = Systolic, y = Diastolic, color = Gender),
             na.rm = TRUE) +
  geom_smooth(aes(x = Systolic, y = Diastolic),
              method = "loess",
              formula = y ~ x,
              na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-193-1.pdf

To clean up our codes even more, we can nest the aesthetics into the `ggplot()` function instead of the geometric functions. But note that everything you nest in your `ggplot()` will be applied to all following geometrics.

```
ggplot(demo_bpx, aes(x = Systolic, y = Diastolic)) +
    geom_point(aes(color = Gender),
               na.rm = TRUE) +
    geom_smooth(method = "loess",
                formula = y ~ x,
                na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-194-1.pdf

#### 6.5.0.1 Functions debunked

**geom_smooth** is how we show the smoothed conditional means line on our scatterplot. The arguments are as follows:

geom_smooth( > mapping = aes(**AESTHETICS**),

> method = **"SMOOTHING METHOD (FUNCTION)"**, e.g. loess, gam, lm, glm
>
> formula = **FORMULA TO USE IN SMOOTHING FUNCTION**, usually y ~ x when there are less than 1,000 observations
>
> na.rm = **TRUE OR FALSE**,
>
> show.legend = **TRUE OR FALSE**,
>
> se = **TRUE OR FALSE**

)

#### 6.5.0.2 DO QUESTION 6 OF THE QUIZ NOW

> What do you think the argument `se = FALSE` does?

## 6.6 5. OTHER GEOMETRIC FUNCTIONS

Aside from `geom_point()`, there are other geometric functions that we can use to plot different types of graphs. Note that this list of geometrics functions is not extensive, and you are encouraged to explore more about them on R using the `?` or `??` command or on this website about ggplot!

### 6.6.1 Bar graph

We use `geom_bar()` to create bar graphs. Different than `geom_point()`, `geom_bar()` only needs us to define either the x or y aesthetic, not both. This is because one of the axes needs to be the count of whatever variable we chose! For example, if we want to count how many individuals there are of each reported gender:

```
ggplot(demo_bpx) +
    geom_bar(aes(x = Gender))
```

intro2R_files/figure-latex/unnamed-chunk-195-1.pdf

We can also tell R to calculate the proportion of each gender instead of counting:

```
ggplot(demo_bpx) +
    geom_bar(aes(x = Gender, y = ..prop.., group = 1))
```

intro2R_files/figure-latex/unnamed-chunk-196-1.pdf

#### 6.6.1.1 Fill Position

A neat argument/position that we can nest in `geom_bar()` is fill. Fill lets us add another variable to our graph and further divides up our columns into separate categories. For example, if we want to know the different combination of genders and races in our dataset:

```
ggplot(demo_bpx) +
    geom_bar(aes(x = Gender, fill = Race))
```

intro2R_files/figure-latex/unnamed-chunk-197-1.pdf

#### 6.6.1.2 Dodge Position

Another option is the dogdge argument/position. This argument separates our columns into smaller, side-by-side columns so we can easily compare the different variables.

```
ggplot(demo_bpx) +
  geom_bar(aes(x = Gender, fill = Race),
           position = "dodge")
```

intro2R_files/figure-latex/unnamed-chunk-198-1.pdf

Another way that we can choose to present our bar graph is in the form of a circular graph. To do this, we can add another function `coord_polar()` to our ggplot canvas.

```
ggplot(demo_bpx, aes(x = Gender)) +
  geom_bar() +
  coord_polar()
```

intro2R_files/figure-latex/unnamed-chunk-199-1.pdf

```
ggplot(demo_bpx, aes(x = Race)) +
  geom_bar() +
  coord_polar()
```

intro2R_files/figure-latex/unnamed-chunk-200-1.pdf

### 6.6.2 Line Graph

Another graph type that the ggplot R package offers is line graph. To plot a line graph, we use the function `geom_line()`.

```
ggplot(demo_bpx) +
    geom_line(aes(x = Systolic, y = Diastolic), na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-201-1.pdf

Line graphs may also be an interesting way for us to demonstrate ranges of a particular continuous variable of a categorical variable. For example, we can see the range of Systolic Blood Pressures of different races with the following graph:

```
ggplot(demo_bpx) +
    geom_line(aes(x = Systolic, y = Race), na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-202-1.pdf

### 6.6.3 Boxplot

If you are not a fan of the line graph above, boxplots is another option that we can explore together. To plot a boxplot, we use `geom_boxplot()` like so:

```
ggplot(demo_bpx, aes(x = Systolic, y = Race)) +
  geom_boxplot(na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-203-1.pdf

```
ggplot(demo_bpx, aes(x = Gender, y = Diastolic)) +
  geom_boxplot(na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-204-1.pdf

### 6.6.4 Frequency Polygon

Frequency polygon is another option that we can explore. They are similar to bar graphs, except frequency polygon visualize the counts with lines. We can plot frequency polygons using `geom_freqpoly()` like so:

```
ggplot(demo_bpx) +
    geom_freqpoly(aes(x = Systolic), binwidth = 5, na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-205-1.pdf

Since `geom_freqpoly()` divides the variable in the x axis into bins before counting the number of observations in each bin, we can further customize how we want our frequency polygon to look by changing the binwidth.

### 6.6.5 1.3 Try it yourself

Try increasing and decreasing the binwidth of a frequency polygon. What differences do you see? What does binwidth actually mean?

```
ggplot(demo_bpx) +
    geom_freqpoly(aes(x = Systolic), binwidth = 1, na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-206-1.pdf

```
ggplot(demo_bpx) +
    geom_freqpoly(aes(x = Systolic), binwidth = 20, na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-207-1.pdf

We can also layer multiple `geom_freqpoly()` on each other and give them different colors:

```
ggplot(demo_bpx) +
    geom_freqpoly(aes(x = Systolic, color = "Systolic"), binwidth = 10, na.rm = TRUE) +
    geom_freqpoly(aes(x = Diastolic, color = "Diastolic"), binwidth = 10, na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-208-1.pdf

You may notice that the x axis label for the graph above is incorrect! And the legend title is also wrong. Don't worry! We will go over how to manually add and edit graph elements later in this tutorial.

#### 6.6.5.1 DO QUESTIONS 7 & 8 OF THE QUIZ NOW

> Increasing the binwidth of a bar graph makes the graph more detailed. (True or False)
>
> Match the geometrics with the correct graph type.

## 6.7 6. FACET FUNCTIONS

Aside from using aesthetics, `facet_wrap()` is another good option to create subplots based on categorical variables. You can create divide the data up into subplots by 1 or 2 variables. Run the codes below to see what these two situations would look like.

```
ggplot(demo_bpx) +
  geom_point(aes(x = Diastolic, y = Systolic), na.rm = TRUE) +
  facet_wrap(~ Gender, nrow = 3)
```

intro2R_files/figure-latex/unnamed-chunk-209-1.pdf

```
ggplot(demo_bpx) +
  geom_point(aes(x = Diastolic, y = Systolic), na.rm = TRUE) +
  facet_grid(Race ~ Gender)
```

intro2R_files/figure-latex/unnamed-chunk-210-1.pdf

#### 6.7.0.1 DO QUESTION 9 ON CANVAS NOW

> It is most useful to use facet functions when plotting continuous variables.

### 6.7.1 1.4 Try it yourself

Try recreating the graph above but without any missing (NA) values.

**HINT**: Filter out any information we do not need using logical operators!

## 6.8 7. CUSTOMIZING GRAPH ELEMENTS

We can customize how our graph looks like with different graph elements including graph title, axes labels, legendes, etc. In this section, we will cover as many graph elements as possible.

But for your own information and exploration, more information on editing graph elements can be found here. Here is also a list of colors that R recognizes that may come in handy.

Going back to the frequency polygon above where the x axis label is incorrect, this is what the code for the correct graph would look like:

```
ggplot(demo_bpx) +
    geom_freqpoly(aes(x = Systolic, color = "Systolic"), binwidth = 10, na.rm = TRUE) +
    geom_freqpoly(aes(x = Diastolic, color = "Diastolic"), binwidth = 10, na.rm = TRUE) +
    labs(x = "Blood Pressure (Hg mm)", color = "Type of Blood Pressure")
```

intro2R_files/figure-latex/unnamed-chunk-211-1.pdf

The first three lines of codes are similar to what we have previously, and the last one is new. Let's go over the last line of codes together in the following functions debunked.

#### 6.8.0.1 Functions Debunked

`labs` lets us change the axes labels, graph title, and legend title! The basic arguments are as follows:

**labs**(

title = "**TITLE OF GRAPH**",

x = "**X AXIS LABEL**",

y = "**Y AXIS LABEL**",

color = "**LEGEND TITLE**"

)

**For example:** `labs(x = "Blood Pressure (Hg mm)", color = "Legend")`

We can customize our graphs even more by changing the texts' fonts, emphasis, or even size! To do this, we can nest multiple arguments in a function called `theme()`. Going back to the main objective of our tutorial: create a graph that shows the relationship between Diastolic and Systolic Blood Pressure of the first 200 Males and Females in the 2013-2014 NHANES datasets, let us recall what that graph originally looks like:

```
ggplot(demo_bpx) +
    geom_point(aes(x = Systolic, y = Diastolic, color = Gender),
               na.rm = TRUE,
               position = "jitter") +
    geom_smooth(aes(x = Systolic, y = Diastolic),
                method = "loess",
                formula = y ~ x,
                na.rm = TRUE)
```

intro2R_files/figure-latex/unnamed-chunk-212-1.pdf

Now, we can change the axes labels, legend title, and add a graph title using `labs()`:

```
ggplot(demo_bpx) +
    geom_point(aes(x = Systolic, y = Diastolic, color = Gender),
               na.rm = TRUE,
               position = "jitter") +
    geom_smooth(aes(x = Systolic, y = Diastolic),
                method = "loess",
                formula = y ~ x,
```

```
               na.rm = TRUE) +
    labs(title = "Systolic vs. Diastolic Blood Pressures of Different Genders",
         x = "Systolic Blood Pressure (mm Hg)",
         y = "Diastolic Blood Pressure (mm Hg)",
         color = "Genders of Respondents")
```

intro2R_files/figure-latex/unnamed-chunk-213-1.pdf

After that, we can use `theme()` to change the font, emphasis, size, and color of our texts.

```
ggplot(demo_bpx) +

    geom_point(aes(x = Systolic, y = Diastolic, color = Gender),
            na.rm = TRUE,
            position = "jitter") +

    geom_smooth(aes(x = Systolic, y = Diastolic),
               method = "loess",
               formula = y ~ x,
               na.rm = TRUE) +

    labs(title = "Systolic vs. Diastolic Blood Pressures of Different Genders",
         x = "Systolic Blood Pressure (mm Hg)",
         y = "Diastolic Blood Pressure (mm Hg)",
         color = "Genders of Respondents") +

    theme(plot.title = element_text(family = "Helvetica", face = "bold", size = 20, color = "cyan
          axis.title = element_text(family = "Helvetica", size = 15),
          axis.text = element_text(family = "Helvetica", size = 12),
          legend.title = element_text(family = "Helvetica", face = "italic", size = 15),
          legend.text = element_text(family = "Helvetica", size = 12))
```

intro2R_files/figure-latex/unnamed-chunk-214-1.pdf

#### 6.8.0.2 Functions Debunked

Some basic arguments of `theme()` are as follows:

**theme**(

plot.title = element_text(family = "**FONT NAME**", face = "**EMPHASIS TYPE**, size = **SIZE NUMBER**, color ="**A COLOR THAT R RECOGNIZES**"),

axis.title = [*same arguments as before*],

axis.text = [*same arguments as before*],

legend.title = [*same arguments as before*],

legend.text = [*same arguments as before*]

)

**For example:** `theme(plot.title = element_text(family = "Helvetica", face = "bold", size = 20, color = "cyan4"))`

#### 6.8.0.3 DO QUESTION 10 OF THE QUIZ NOW

What is the difference between `legend.title` and `legend.text`?

### 6.8.1 1.5 Try it yourself

Customize your own graph using the functions we just learned above!

## 6.9 8. SAVING OUR GRAPH

To explort our graph into a file like png, pdf, or jpeg, we can use the function `ggsave()` followed by the name that we want the file to have. To simplify this process, we can give our graph a name such as "Final_plot" using `<-`.

```
Final_plot <-

    ggplot(demo_bpx) +

    geom_point(aes(x = Systolic, y = Diastolic, color = Gender),
               na.rm = TRUE,
               position = "jitter") +

    geom_smooth(aes(x = Systolic, y = Diastolic),
                method = "loess",
                formula = y ~ x,
                na.rm = TRUE) +
```

```
    labs(title = "Systolic vs. Diastolic Blood Pressures of Different Genders",
         x = "Systolic Blood Pressure (mm Hg)",
         y = "Diastolic Blood Pressure (mm Hg)",
         color = "Genders of Respondents") +

    theme(plot.title = element_text(family = "Helvetica", face = "bold", size = 20, color = "cyan
          axis.title = element_text(family = "Helvetica", size = 15),
          axis.text = element_text(family = "Helvetica", size = 12),
          legend.title = element_text(family = "Helvetica", face = "italic", size = 15),
          legend.text = element_text(family = "Helvetica", size = 12))
```

```
ggsave("images/Final plot.png")
```

```
## Saving 6.5 x 4.5 in image

## Warning in grid.Call(C_stringMetric, as.graphicsAnnot(x$label)): font family not
## found in Windows font database

## Warning in grid.Call(C_stringMetric, as.graphicsAnnot(x$label)): font family not
## found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_stringMetric, as.graphicsAnnot(x$label)): font family not
## found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## font family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database
```

By default, `ggsave()` will save the last plot that we ran before it. But we can also tell it exactly which plot we are referring to using another argument after the file name.

```
ggsave("images/Final plot-1.png", Final_plot)
```

```
## Saving 6.5 x 4.5 in image

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database
```

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## font family not found in Windows font database
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database
```

We can also change the width and height of our saved file. Remember to specify the units as well!

```
ggsave("images/Final plot-2.png", Final_plot, width = 40, height = 30, units = 'cm')
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
```

```
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## font family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## family not found in Windows font database
```

More information on `ggsave()` can be found here.

Now we can check our working directory to see if all of our graphs are there.

```
dir()
```

```
##  [1] "_book"
##  [2] "_bookdown.yml"
##  [3] "_bookdown_files"
##  [4] "_build.sh"
```

```
##  [5] "_deploy.sh"
##  [6] "_output.yml"
##  [7] "0-r-and-rstudio-set-up.Rmd"
##  [8] "1-introduction-to-r.Rmd"
##  [9] "2-importing-data-into-r-with-readr.Rmd"
## [10] "3-introduction-to-nhanes.Rmd"
## [11] "4-data-analysis-with-dplyr.Rmd"
## [12] "5-data-visualization-with-ggplot.Rmd"
## [13] "6-date-time-data-with-lubridate.Rmd"
## [14] "7-data-summary-with-tableone.Rmd"
## [15] "8-Exercise-Solutions.Rmd"
## [16] "9-references.Rmd"
## [17] "book.bib"
## [18] "data"
## [19] "DESCRIPTION"
## [20] "Dockerfile"
## [21] "docs"
## [22] "header.html"
## [23] "images"
## [24] "index.Rmd"
## [25] "intro2R.Rmd"
## [26] "intro2R_cache"
## [27] "intro2R_files"
## [28] "LICENSE"
## [29] "now.json"
## [30] "packages.bib"
## [31] "preamble.tex"
## [32] "R.Rproj"
## [33] "README.md"
## [34] "style.css"
## [35] "toc.css"
```

If there is a present file that you think should not be there, we can use the function `file.remove()` followed by the name of the file to remove it.

```
file.remove("Rplot001.png")
```

```
## Warning in file.remove("Rplot001.png"): cannot remove file 'Rplot001.png',
## reason 'No such file or directory'
```

```
## [1] FALSE
```

Then, we can check our directory again and all of the relevant files should be there!

```
dir()
```

```
##  [1] "_book"
##  [2] "_bookdown.yml"
```

```
##  [3] "_bookdown_files"
##  [4] "_build.sh"
##  [5] "_deploy.sh"
##  [6] "_output.yml"
##  [7] "0-r-and-rstudio-set-up.Rmd"
##  [8] "1-introduction-to-r.Rmd"
##  [9] "2-importing-data-into-r-with-readr.Rmd"
## [10] "3-introduction-to-nhanes.Rmd"
## [11] "4-data-analysis-with-dplyr.Rmd"
## [12] "5-data-visualization-with-ggplot.Rmd"
## [13] "6-date-time-data-with-lubridate.Rmd"
## [14] "7-data-summary-with-tableone.Rmd"
## [15] "8-Exercise-Solutions.Rmd"
## [16] "9-references.Rmd"
## [17] "book.bib"
## [18] "data"
## [19] "DESCRIPTION"
## [20] "Dockerfile"
## [21] "docs"
## [22] "header.html"
## [23] "images"
## [24] "index.Rmd"
## [25] "intro2R.Rmd"
## [26] "intro2R_cache"
## [27] "intro2R_files"
## [28] "LICENSE"
## [29] "now.json"
## [30] "packages.bib"
## [31] "preamble.tex"
## [32] "R.Rproj"
## [33] "README.md"
## [34] "style.css"
## [35] "toc.css"
```

We've reached the end of our tutorial! Note that this tutorial only covers the basics of ggplot. ggplot is a large package and you are encouraged to explore the many functions that it offers on your own. And remember that it takes practice to be fluent in this language!

## 6.10 9. SUMMARY AND TAKEAWAYS

By the end of this tutorial, you should be somewhat familiar with the ggplot package including how to create a basic graph on R as well as how to customize it to your liking.

If you are interested in learning more about ggplot, this website is also a good resource for you to tap into.

Here is also a cheat sheet of more ggplot functions.

# Chapter 7

# Date & Time Data with lubridate

## 7.1 INSTRUCTIONS

In this tutorial, we will be exploring how to deal with date/time data in R using the lubridate package. This incudes creating new, retrieving information from existing, modifying, and conduct different arithmetic calculations on date/time data. We will also be plotting date/time data to visualize our dataset.

Accompanying this tutorial is **a short Google quiz** for your own self-assessment. The instructions of this tutorial will clearly indicate when you should answer which question.

## 7.2 LEARNING OBJECTIVES

- Understand the basics of the lubridate package and its simple datetime functions.
- Know how to create date, time, and datetime data on R.
- Know how to retrieve information from date/time data
- Know how to modify date/time data
- Know how to add, subtract, multiply, divide date/time data
- Be familiar with graphs with date/time data.

## 7.3 1. SET UP

For this tutorial, we will need the **lubridate** package with functions that will allow us to deal with date/time data. This package is also part of the tidyverse core that also houses readr, dplyr, and ggplot2.

```
#install.packages(lubridate)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

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

We will also be using the **Friend visits.csv** dataset. This dataset was especially prepared for this tutorial, and it includes specific data that will help us understand lubridate more! In short, this made-up dataset contains information about the times that our host(s) had friends over and the times that they left.

Let's import this dataset into R now using `read_csv()` from the package readr.

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

```
visits <- read_csv("data/Friends visits.csv")
```

```
## Rows: 365 Columns: 8

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (8): Friend, Year, Month, Day, Hour, Minute, Second, Left_time

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(visits)
```

```
## # A tibble: 6 x 8
##   Friend  Year Month   Day  Hour Minute Second Left_time
##    <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>     <dbl>
## 1      1  2015     1     1    19      4     22       830
## 2      2  2015     1     2     9     20     19       850
## 3      3  2015     1     3    22     23      9       923
## 4      4  2015     1     4     9     22     51      1004
## 5      5  2015     1     5    18      1     16       812
## 6      6  2015     1     6     5     59      2       740
```

We will also need the **dplyr** and **ggplot2** packages to plot a few graphs.

```
#install.packages(dplyr)
library(dplyr)

#install.packages(ggplot2)
library(ggplot2)
```

## 7.4 2. EXPLORING FRIENDS VISITS DATASET

Before we jump into lubridate and date/time data, let's first explore the Friends visits dataset that we will be using today.

```
head(visits, 20)
```

```
## # A tibble: 20 x 8
##    Friend  Year Month   Day  Hour Minute Second Left_time
##     <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>     <dbl>
##  1      1  2015     1     1    19      4     22       830
##  2      2  2015     1     2     9     20     19       850
##  3      3  2015     1     3    22     23      9       923
##  4      4  2015     1     4     9     22     51      1004
##  5      5  2015     1     5    18      1     16       812
##  6      6  2015     1     6     5     59      2       740
##  7      7  2015     1     7     9     45     55       913
##  8      8  2015     1     8    12      3     33       709
##  9      9  2015     1     9    17      9     59       838
## 10     10  2015     1    10    16     44     43       753
## 11     11  2015     1    11     0     31     28       849
## 12     12  2015     1    12    19     40     54       853
## 13     13  2015     1    13     7     55      6       924
## 14     14  2015     1    14    13     23     15       923
## 15     15  2015     1    15    14     51     17       941
## 16     16  2015     1    16    21     11     24       702
## 17     17  2015     1    17     0     32      7       854
## 18     18  2015     1    18     9     31     32       851
## 19     19  2015     1    19     4     56     41       837
## 20     20  2015     1    20    20     24     22       844
```

As we can see, the dataset Friends visits contains information about all of the visits from friends that the host(s) received in the year 2015. The information includes date and time of each friend's visit as well as the time that they left.

Right now, the none of the columns are recognized as date or time in R because all of the information are scattered across multiple columns. In addition, the column "Left_time" also clumps together the hours and minutes into one incoherent number. For this specific column, we need to clearly identify the hour and minute so that R can recognize it as containing date/time.

But before we jump even further into this tutorial, let's actually filter out information from this dataset. Let's say we only want to retain visits that are before 9 PM.

```
visits <- filter(visits, Hour < 21)
```

Now, let's check our dataset again.

```
head(visits)
```

```
## # A tibble: 6 x 8
##   Friend  Year Month   Day  Hour Minute Second Left_time
##    <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>     <dbl>
## 1      1  2015     1     1    19      4     22       830
## 2      2  2015     1     2     9     20     19       850
## 3      4  2015     1     4     9     22     51      1004
## 4      5  2015     1     5    18      1     16       812
## 5      6  2015     1     6     5     59      2       740
## 6      7  2015     1     7     9     45     55       913
```

## 7.5 3. CREATING DATE/TIME DATA

Date/time data are data tht conveys information about, you guessed it, date and/or time! There are three relevant data types when we talk about date/time data: 1. **Date** - only has the date (e.g. 2020-05-15) 2. **Time** - only has the time (e.g. 20:45:00) 3. **Datetime** - has both the date and time (e.g. 2020-05-15 20:45:00)

Now that we know the different types of date/time data, know that there are also several ways for us to create date/time data: we can create them from raw **strings**, from an **existing date/time data**, or from a **dataset**.

### 7.5.1 Strings

Firstly, we can use `ymd()` with a quoted string to create a new date/time data.

```
ymd("2021-06-20")
```

```
## [1] "2021-06-20"
```

We can also change the order of the letters in `ymd()` to match which information came first (year, month, or day). For example, R reads `ymd()` as "year, month, date" and `dmy()` as "day, month, year".

```
dmy("15 Feb, 2010")
```

```
## [1] "2010-02-15"
```

Similarly, `mdy()` is also an option. In the example below, the string is just a series of numbers that R will reads as month, day, year.

```
mdy("07082016")
```

```
## [1] "2016-07-08"
```

If we do choose to only write a series of number, then our string can be unquoted as well. R will still be able to identify that this is a string and will read it accordingly!

```
mdy(07082016)
```

```
## [1] "2016-07-08"
```

In addition, we can also generate hour, minute, and second information by using `ymd_hms()`. Different than the functions above, we cannot change the order of "hms".

```
ymd_hms("2021-06-20-5-49-34")
```

```
## [1] "2021-06-20 05:49:34 UTC"
```

In R, the default time zone is Coordinated Universal Time, or UTC for short. But we can also change the time zone of our date/time data by using the `tz` argument like so:

```
ymd_hms("2021-06-20-5-49-34", tz = "America/Vancouver")
```

```
## [1] "2021-06-20 05:49:34 PDT"
```

```
ymd_hms("2021-06-20-5-49-34", tz = "Etc/GMT+7")
```

```
## [1] "2021-06-20 05:49:34 -07"
```

To check the full list of time zones that R recognizes, we can use the function `OlsonNames()`.

```
head(OlsonNames(), 10)
```

```
##  [1] "Africa/Abidjan"     "Africa/Accra"       "Africa/Addis_Ababa"
##  [4] "Africa/Algiers"     "Africa/Asmara"      "Africa/Asmera"
##  [7] "Africa/Bamako"      "Africa/Bangui"      "Africa/Banjul"
## [10] "Africa/Bissau"
```

We can also use the following code to check what our current time zone is.

```
Sys.timezone()
```

```
## [1] "America/Los_Angeles"
```

If after running the code above and R returns NA or UTC for you, it means the software cannot correctly identify where we are. For this, we can tell R directly what our time zone is. For example, if we are in Vancouver, BC, Canada right now, we would write the following code:

```
Sys.setenv(TZ = "America/Vancouver")
```

To confirm if the correct time zone has been set, we can use the functions `today()` or `now()`.

```
today()
```

```
## [1] "2021-07-25"
```

```
now()
```

```
## [1] "2021-07-25 23:51:31 PDT"
```

### 7.5.2 6.1 Try it yourself

After running the `today()` and `now()` codes above, what do you see?

Try to also change the time zone to where you are or to something else. Now what do you see when you run `today()` and `now()`?

#### 7.5.2.1 DO QUESTIONS 1-3 OF THE QUIZ NOW

> Which of the following is a date data?
>
> What data type is this: 2017-09-19 19:00:00?
>
> What is different about the outputs' data types of `today()` and `now()`? (Select all that apply)

### 7.5.3 Existing Date/Time Data

You may have recognized that while `today()` gives us only the current date, `now()` gives us both the date and time of our current location - in this case, America/Vancouver.

We can actually convert date data to datetime and vice versa using `as_datetime()` and `as_date()`.

```
## from date to datetime

as_datetime(today())
```

```
## [1] "2021-07-25 UTC"
```

```
## from datetime to date

as_date(now())
```

```
## [1] "2021-07-25"
```

### 7.5.4 Dataset

Lastly, we can also create date/time data using a dataset. For this section, we will be using the flights dataset from the nycflights13 package that we have briefly explored previously.

Let's quickly look at our data again.

```
head(visits)
```

```
## # A tibble: 6 x 8
##   Friend  Year Month   Day  Hour Minute Second Left_time
##    <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>     <dbl>
## 1      1  2015     1     1    19      4     22       830
## 2      2  2015     1     2     9     20     19       850
## 3      4  2015     1     4     9     22     51      1004
## 4      5  2015     1     5    18      1     16       812
## 5      6  2015     1     6     5     59      2       740
## 6      7  2015     1     7     9     45     55       913
```

As we can see, all of these columns contain data regarding date and time.

Unfortunately, because the information about datetime is divided up into different columns, R does not recognize it as date/time data. What we need to do is combine and convert all of these columns into datetime. To do this, we can use the function `make_datetime()`.

```
visits_datetime <- visits %>%
    mutate(Visit_time = make_datetime(Year, Month, Day, Hour, Minute))
```

```
head(visits_datetime)
```

```
## # A tibble: 6 x 9
##   Friend  Year Month   Day  Hour Minute Second Left_time Visit_time
##    <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>     <dbl> <dttm>
## 1      1  2015     1     1    19      4     22       830 2015-01-01 19:04:00
## 2      2  2015     1     2     9     20     19       850 2015-01-02 09:20:00
## 3      4  2015     1     4     9     22     51      1004 2015-01-04 09:22:00
## 4      5  2015     1     5    18      1     16       812 2015-01-05 18:01:00
## 5      6  2015     1     6     5     59      2       740 2015-01-06 05:59:00
## 6      7  2015     1     7     9     45     55       913 2015-01-07 09:45:00
```

Now if we look at our new column `Visit_time`, we should see that all of the information that we have mentioned in our code have been combined into one single value that takes the form of datetime!

So we know how to combine information of different columns to form one cohesive one, what about the Left_time colum? How can we turn values like 1951 into 19:51:00? Once again, `make_datetime()` is the answer!

But first, we need to quickly create a new function that will help us make this process easier. This new function will retain information about the year, month, and date as well as divide the time into hours and minutes.

```
make_datetime_new <- function(year, month, day, time) {
  make_datetime(year, month, day, time %/% 100, time %% 100)
```

```
}
```

#### 7.5.4.1 DO QUESTION 4 OF THE QUIZ NOW

> **REVIEW:** What is the difference between %/% and %%?

Here, we are telling R to write a new `make_datetime()` function named `make_datetime_new()`. Our new function then has 5 arguments, the year, the month, the day, the time %/% 100, and the time %% 100. The last two arguments will give us the hour and minutes of the day. In other words, 830 %/% 100 = 8 and 830 %% 100 is 30, so combined, 830 becomes 8:30!

Now if we apply this new function to our flights_datetime's arr_time column, we should see the new "arr_time" column with the arrival time of flights in date/time form.

```
Friends_visits <- visits_datetime %>%
    mutate(Left_time = make_datetime_new(Year, Month, Day, Left_time))
```

```
head(Friends_visits)
```

```
## # A tibble: 6 x 9
##   Friend  Year Month   Day  Hour Minute Second Left_time
##    <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl> <dttm>
## 1      1  2015     1     1    19      4     22 2015-01-01 08:30:00
## 2      2  2015     1     2     9     20     19 2015-01-02 08:50:00
## 3      4  2015     1     4     9     22     51 2015-01-04 10:04:00
## 4      5  2015     1     5    18      1     16 2015-01-05 08:12:00
## 5      6  2015     1     6     5     59      2 2015-01-06 07:40:00
## 6      7  2015     1     7     9     45     55 2015-01-07 09:13:00
## # ... with 1 more variable: Visit_time <dttm>
```

Another simpler way to use `make_datetime()` is to insert the numerical values of the year, month, day, and time directly.

```
make_datetime(2014, 8, 22, 9)
```

```
## [1] "2014-08-22 09:00:00 UTC"
```

#### 7.5.4.2 DO QUESTION 5 OF THE QUIZ NOW

> Using the Friends_visits data frame, which of the following code will give us a new column named "Year_Month" that only has information of the year and month of each friend visit?

### 7.5.5 Functions Debunked

**make_datetime()** can be used to create new datetime data. The arguments are as follows:

make_datetime(

**VALUES OF YEAR,**

**VALUES OF MONTH,**

**VALUES OF DAY,**

**VALUES OF HOUR,**

**VALUES OF MINUTE,**

**VALUES OF SECOND,**

)

**For example:** * Combine columns of a dataset: `flights_datetime %>% make_datetime(year, month, day, hour, minute)` * Insert values directly: `make_datetime(2014, 8, 22, 9)`

### 7.5.6 6.2 Try it yourself

Try creating a new column named "Day_visit" that only contains information of the Year, Month, and Day columns using the Friends_visits dataframe that we just created.

## 7.6 4. RETRIEVING INFORMATION FROM DATE/TIME DATA

We have learned how to create date/time data, but how do we retrieve information from the date/time data that we created? Hopefully, by the end of this section, we would be able to answer this question!

Let's first create a simple datetime value from a string using `ymd_hms()`. And let's also use the column "Visit_time" of our Friends_visits data frame.

```
DT <- ymd_hms("2020-04-19 09:45:00")
```

```
head(Friends_visits$Visit_time)
```

```
## [1] "2015-01-01 19:04:00 UTC" "2015-01-02 09:20:00 UTC"
## [3] "2015-01-04 09:22:00 UTC" "2015-01-05 18:01:00 UTC"
## [5] "2015-01-06 05:59:00 UTC" "2015-01-07 09:45:00 UTC"
```

### 7.6.1 Year

Now, if we want to know the year of our datetime value, we can use the function `year()` with the name of our datetime variable between the `()`.

```
year(DT)
```

```
## [1] 2020
```

```
head(
    year(Friends_visits$Visit_time)
    )
```

```
## [1] 2015 2015 2015 2015 2015 2015
```

The reason why the outputs to `year(Friends_visits$Visit_time)` are all 2015s is because if we look at the first 6 records of the columns "Visit_time", all year values are 2015!

Similarly, we can also use the function `yday()` to find out what day of the year our datetime value falls into. For example, April 19 is the **110th** day of the year.

```
yday(DT)
```

```
## [1] 110
```

Using the same function on our Friends_visits data frame, we can see that the first 6 records of the column "Visit_time" contains the first, second, fourth, fifth, sixth, and seventh day of the year.

```
head(
    yday(Friends_visits$Visit_time)
    )
```

```
## [1] 1 2 4 5 6 7
```

### 7.6.2 Month

`mday()` is similar to `yday()` except it gives us what day of the month our datetime value falls into. In this case, "4-19" is the **19th** day of April!

```
mday(DT)
```

```
## [1] 19
```

What days of the month do the first 6 records of Visit_time contain?

```
head(
    mday(Friends_visits$Visit_time)
    )
```

```
## [1] 1 2 4 5 6 7
```

At this point, we should already have figured out the pattern, so `month()` will give us the month of our datetime value, normally, with a numerical output. If we want to convert month from a numerical "4" to "Apr", we would add the `label` argument like so:

```
month(DT, label = TRUE)
```

```
## [1] Apr
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

```
head(
    month(Friends_visits$Visit_time, label = TRUE)
    )
```

```
## [1] Jan Jan Jan Jan Jan Jan
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

### 7.6.3 Week

`wday()` will give us what day of the week our datetime value falls into. Similarly, we can use `label` to change numerical values to "Mon", "Tues", "Wed", etc. If we want the full "Monday" instead of just "Mon", we would add the `abbr` argument like so:

```
wday(DT, label = TRUE, abbr = FALSE)
```

```
## [1] Sunday
## 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
```

```
head(
    wday(Friends_visits$Visit_time, label = TRUE, abbr = FALSE)
    )
```

```
## [1] Thursday  Friday    Sunday    Monday    Tuesday   Wednesday
## 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
```

### 7.6.4 6.3 Try it yourself

Using the functions introduced above, solve the following questions: 1. What day of the week is April 2, 2014? 2. What day of the year is 2017-09-15? 3. What day of the month is 20190830? 4. Find the months of the last 11 records of the column Visit_hour.

### 7.6.5 Plotting Retrieved Information

Let's try to aply the functions that we just learned above by retrieving information from the Friends_visits dataset that we previously created.

Here is the dataset again:

```
head(Friends_visits)
```

```
## # A tibble: 6 x 9
##   Friend  Year Month   Day  Hour Minute Second Left_time
##    <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl> <dttm>
```

```
## 1       1  2015     1     1    19      4     22 2015-01-01 08:30:00
## 2       2  2015     1     2     9     20     19 2015-01-02 08:50:00
## 3       4  2015     1     4     9     22     51 2015-01-04 10:04:00
## 4       5  2015     1     5    18      1     16 2015-01-05 08:12:00
## 5       6  2015     1     6     5     59      2 2015-01-06 07:40:00
## 6       7  2015     1     7     9     45     55 2015-01-07 09:13:00
## # ... with 1 more variable: Visit_time <dttm>
```

Let's say we want to plot a bar graph to show how many visits each month in 2015 the hosts have using the Visit_time column. First we want to extract the months from the Visit_time values. After that, we want to place the extracted months in the "month" column.

To complete our goals, we need to use both `mutate()` and `month()` like so:

```
head(
Friends_visits %>%
    mutate(Month = month(Visit_time, label = TRUE))
    )
```

```
## # A tibble: 6 x 9
##   Friend  Year Month   Day  Hour Minute Second Left_time
##    <dbl> <dbl> <ord> <dbl> <dbl>  <dbl>  <dbl> <dttm>
## 1       1  2015 Jan       1    19      4     22 2015-01-01 08:30:00
## 2       2  2015 Jan       2     9     20     19 2015-01-02 08:50:00
## 3       4  2015 Jan       4     9     22     51 2015-01-04 10:04:00
## 4       5  2015 Jan       5    18      1     16 2015-01-05 08:12:00
## 5       6  2015 Jan       6     5     59      2 2015-01-06 07:40:00
## 6       7  2015 Jan       7     9     45     55 2015-01-07 09:13:00
## # ... with 1 more variable: Visit_time <dttm>
```

We should be able to see that all of the "1"s under the month (second) column, have successfully turned into "Jan".

Finally, all we need is a `ggplot()` canvas and a `geom_bar()` function! **(Notice the transition from pipe %>% to +).**

```
Friends_visits %>%
    mutate(Month = month(Visit_time, label = TRUE)) %>%
    ggplot(aes(Month)) +
    geom_bar()
```

intro2R_files/figure-latex/unnamed-chunk-265-1.pdf

#### 7.6.5.1 DO QUESTION 6 OF THE QUIZ NOW

What is the difference between `%>%` and `+`? (Select all that apply)

### 7.6.6 6.4 Try it yourself

Using the same Friends_visit dataset, create a similar graph as above but the x-axis is days of the week. In other words, create a bar graph that shows how many visits there are in each day of the week.

## 7.7 5. UPDATING & PLOTTING DATE/TIME DATA

### 7.7.1 Update

To modify or update a piece of date/time data, we would use `update()`. Updating a date/time data can mean changing it completely:

```
(DT <- ymd_hms("2020-04-19 09:45:00"))
```

```
## [1] "2020-04-19 09:45:00 UTC"
```

```
update(DT, year = 2021, month = 6, day = 21, hour = 9, minute = 13)
```

```
## [1] "2021-06-21 09:13:00 UTC"
```

The cool thing about the `update()` function is that it will automatically adjust the date and time if the value that we want to change our current date/time to is too large. For example, April only has 30 days. Now look what happens when we try to update our date to 2020-04-31.

```
DT %>%
    update(day = 31)
```

```
## [1] "2020-05-01 09:45:00 UTC"
```

`update()` automatically adjusts it to May 1st instead because April 31st does not exist!

### 7.7.2 6.5 Try it yourself

Try to update our month to 13. What happened to our date/time? What is the output?

**update()** can be used to update existing datetime data. The arguments are as follows:

update(

year = **VALUES OF YEAR,**

month = **VALUES OF MONTH,**

day OR mday OR yday = **VALUES OF DAY,**

hour = **VALUES OF HOUR,**

minute = **VALUES OF MINUTE,**

second = **VALUES OF SECOND**

)

**For example:** * `update(DT, year = 2021, month = 6, day = 21, hour = 9, minute = 13)` * `update(ymd_hms("2020-04-19 09:45:00"), year = 2021, month = 6, day = 21, hour = 9, minute = 13)`

### 7.7.3 Plotting Date/Time

So far, we have learned how to plot graphs with date data type, such as month and weekday, in the x-axis. But what if we want to plot our time data in the x-axis instead? Although it sounds simple, plotting time data may be a bit harder than it seems because there is no function in the lubridate package that allows us to only retain time data. In other words, there is no `hms()` function... in lubridate.

To loophole around this issue, we can use `update()`! Basically, we want to update all dates in our data to the same date, let's say January 1 2015, so that when we plot the graph, the graph will be representative of friends visits in each hour. It will make more sense when we start plotting the graph.

Let's check the dataset again before we start plotting.

```
head(Friends_visits)
```

```
## # A tibble: 6 x 9
##   Friend  Year Month   Day  Hour Minute Second Left_time
##    <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl> <dttm>
## 1      1  2015     1     1    19      4     22 2015-01-01 08:30:00
## 2      2  2015     1     2     9     20     19 2015-01-02 08:50:00
## 3      4  2015     1     4     9     22     51 2015-01-04 10:04:00
## 4      5  2015     1     5    18      1     16 2015-01-05 08:12:00
## 5      6  2015     1     6     5     59      2 2015-01-06 07:40:00
## 6      7  2015     1     7     9     45     55 2015-01-07 09:13:00
## # ... with 1 more variable: Visit_time <dttm>
```

Recall that in this dataset, we have already converted the Visit_time column to contain data in the date/time form. However, right now, R recognizes this column using date AND time, but we only want R to distinguish the different times for our graph. So, as we have discussed before, we want to trick R into thinking that all 365 records are of the same date, just different times. To do this, we use `update()`.

```
head(
    Friends_visits %>%
    mutate(Visit_hour = update(Visit_time, yday = 1))
    )
```

```
## # A tibble: 6 x 10
##   Friend  Year Month   Day  Hour Minute Second Left_time
##    <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl> <dttm>
## 1      1  2015     1     1    19      4     22 2015-01-01 08:30:00
## 2      2  2015     1     2     9     20     19 2015-01-02 08:50:00
## 3      4  2015     1     4     9     22     51 2015-01-04 10:04:00
## 4      5  2015     1     5    18      1     16 2015-01-05 08:12:00
## 5      6  2015     1     6     5     59      2 2015-01-06 07:40:00
## 6      7  2015     1     7     9     45     55 2015-01-07 09:13:00
## # ... with 2 more variables: Visit_time <dttm>, Visit_hour <dttm>
```

Now if we look at the new "Visit_hour" column, all records should have the same date: 2015-01-01. Perfect!

The next step is to actually plot this on a frequency polygon like so:

```
Friends_visits %>%
    mutate(Visit_hour = update(Visit_time, yday = 1)) %>%
    ggplot(aes(Visit_hour)) +
    geom_freqpoly(binwidth = 3600)
```

intro2R_files/figure-latex/unnamed-chunk-271-1.pdf

This graph is a bit tricky to read because even though the x-axis says that all data points are within Jan 01, we know this is not true - the data displayed are from Jan 01 to Dec 31! Again, the reason why our x-axis is labelled as Jan 01 is because we have tricked R into thinking that all data points are of the same date so that our graph can plot the number of friend visits depending on times.

What information can you conclude from the graph above? Around what timeframe do our host(s) receive the most friends visits?

Note that each binwidth (i.e. binwidth = 1) is equal to one second. So `binwidth = 3600` means we are clumping all flights within each 60 minutes (1 hour) together into one single data point in our frequency polygon.

### 7.7.4 6.6 Try it yourself

Recreate the frequency polygon above but change the binwidth so that all flights within each 30 minutes are clumped into one single data point. How do the graphs differ? Can you think of a few scenarios where one would be preferred?

#### 7.7.4.1 DO QUESTION 7 OF THE QUIZ NOW

> What should the binwidth be if we want to have each data point to represent a separate 2 hour interval?

#### 7.7.4.2 hms package

There is actually another package in R (and also part of the tidyverse core) that deals with time data specifically: hms. This function provides an alternative to the ggplot code we just wrote above.

We will not go over this package or function in detail in this tutorial. You can explore hms more via this link.

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

```
##
## Attaching package: 'hms'
```

```
## The following object is masked from 'package:lubridate':
##
##     hms
```

```
head(
    Friends_visits %>%
    mutate(Visit_hour_hms = hms(Second, Minute, Hour))
    )
```

```
## # A tibble: 6 x 10
##   Friend  Year Month   Day  Hour Minute Second Left_time
##    <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl> <dttm>
## 1      1  2015     1     1    19      4     22 2015-01-01 08:30:00
## 2      2  2015     1     2     9     20     19 2015-01-02 08:50:00
## 3      4  2015     1     4     9     22     51 2015-01-04 10:04:00
## 4      5  2015     1     5    18      1     16 2015-01-05 08:12:00
## 5      6  2015     1     6     5     59      2 2015-01-06 07:40:00
## 6      7  2015     1     7     9     45     55 2015-01-07 09:13:00
## # ... with 2 more variables: Visit_time <dttm>, Visit_hour_hms <time>
```

We can see that our “Visit_hour_hms” column actually only retain time (hour, minute, second) data directly. Using this column, we can plot our frequency polygon like so:

```
Friends_visits %>%
    mutate(Visit_hour_hms = hms(Second, Minute, Hour)) %>%
    ggplot(aes(Visit_hour_hms)) +
    geom_freqpoly(binwidth = 900)
```

intro2R_files/figure-latex/unnamed-chunk-274-1.pdf

If we compare this graph with the one above, we should see that they are identical! Except the x-axis in this graph is much clearer and accurate because it only contains the time of day.

## 7.8 6. ARITHMETIC OPERATORS WITH DATE/TIME

### 7.8.1 Basic Arithmetic

Just like any other types of data on R, we can conduct basic arithmetic operations using date/time data.

```
5 * (years(2) + months(11))
```

```
## [1] "10y 55m 0d 0H 0M 0S"
```

```
days(50) + hours(25) + minutes(2)
```

```
## [1] "50d 25H 2M 0S"
```

```
ymd_hms("2020-04-19 09:45:00") +
    days(50) + hours(25) + minutes(2)
```

```
## [1] "2020-06-09 10:47:00 UTC"
```

We can also combine arithmetic operators with `today()` or `now()` like the code below to find out someone's age!

```
(age <- today() - ymd("2000-02-15"))
```

```
## Time difference of 7831 days
```

As you may have seen, telling someone you are 7,797 days old is quite a mouthful, and also pretty impractical. A way for us to make this number more comprehensible is by converting it to years using the function `as.duration()`.

```
as.duration(age)
```

```
## [1] "676598400s (~21.44 years)"
```

#### 7.8.1.1 DO QUESTION 8 OF THE QUIZ NOW

We can use `now()` to calculate our age as well. (True or False)

### 7.8.2 Account for Leap Years and Daylight Savings

A potential problem with the functions that you are introduced to in the pr is that they do not account for time changes within the year. For example, a leap year can add an extra day to the year and daylight savings may add or subtract an hour from a day.

To solve this problem, we can use `dyears()` and `ddays()` instead of the normal `years()` and `days()`.

#### 7.8.2.1 Leap Year

We know that 2020 was a leap year. Let is check the difference between adding one year to 2020-01-01 using `years()` and `dyears()`.

```
## Normal

ymd("2020-01-01") + years(1)
```

```
## [1] "2021-01-01"
```

```
## Considers Leap Year

ymd("2020-01-01") + dyears(1)
```

```
## [1] "2020-12-31 06:00:00 UTC"
```

As we can see, `dyears()` accounts for an extra day in February, so it recognizes that 365 days (1 year) after 2020-01-01 is only 2020-12-31, not 2021-01-01.

#### 7.8.2.2 Daylight Savings

Similarly, `ddays()` recognizes that daylight savings in Vancouver started in early morning 2021-03-14, and correctly calculates that 1 day (24 hours) after 2021-03-14 1AM is actually 2021-03-15 2AM!

```
## Normal

ymd_hms("2021-03-14 1:00:00",
        tz = "America/Vancouver") +
    days(1)
```

```
## [1] "2021-03-15 01:00:00 PDT"
```

```
## Considers Daylight Savings

ymd_hms("2021-03-14 1:00:00",
        tz = "America/Vancouver") +
    ddays(1)
```

```
## [1] "2021-03-15 02:00:00 PDT"
```

#### 7.8.2.3 DO QUESTIONS 9-10 OF THE QUIZ NOW

> Which of these functions account for date and time changes throughout the years?
>
> What happens when we use `ddays()` or `dyears()` in days or years that do not experience any special time/date changes?

## 7.9 7. SUMMARY AND TAKEAWAYS

After completing this tutorial, you should be more familiar the lubridate package and know the basic ways to deal with date/time data in R. This tutorial covered how to create new date/time data, retrieve information from existing date/time data, modify and plot date/time data, as well as conduct simple arithmetic operators using date/time data.

Date/time is quite interesting once you get the hang of it. This data type can give us very valuable information about the dataset that we are working with. Again, it is okay to make mistakes and be confused when we first got started, but know that fluency also comes from practice!

# Chapter 8

# Data Summary with tableone

## 8.1 INSTRUCTIONS

In this tutorial, we will be exploring how to summarize all variables of our datasets in one single table. We will familiarize ourselves with the R package tableone and its associated functions. This tutorial will show you how to be more efficient in analyzing data on R.

Accompanying this tutorial is **a short Google quiz** for your own self-assessment. The instructions of this tutorial will clearly indicate when you should answer which question.

## 8.2 LEARNING OBJECTIVES

- Understand the basics the tableone package and its applications.
- Efficiently summarize whole datasets into one single table.
- Be familiar with the function `CreateTableOne()` and a few of its basic arguments.
- Know how to tell tableone which variables are continuous and which variables are categorical.
- Be familiar with different `print()` arguments to customize a tableone.

## 8.3 1. SET UP

For this tutorial, the main package that we will be working with is the **tableone** package. We will also need the dplyr package for a few basic functions and data from the nhanesA package. Let's go ahead and load them in our session!

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

#install.packages("dplyr")
library(dplyr)

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

Alright, so we are going back to the NHANES dataset for this tutorial. Let's, once again, download the "DEMO_H" dataset and save it in an object called "demo_original".

```
demo_original <- nhanes("DEMO_H")
```

```
## Processing SAS dataset DEMO_H     ..
```

Just a reminder to everyone that this is what our raw dataset look like.

```
head(demo_original)
```

```
##    SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN RIDRETH1 RIDRETH3 RIDEXMON
## 1 73557        8        2        1       69       NA        4        4        1
## 2 73558        8        2        1       54       NA        3        3        1
## 3 73559        8        2        1       72       NA        3        3        2
## 4 73560        8        2        1        9       NA        3        3        1
## 5 73561        8        2        2       73       NA        3        3        1
## 6 73562        8        2        1       56       NA        1        1        1
##   RIDEXAGM DMQMILIZ DMQADFC DMDBORN4 DMDCITZN DMDYRSUS DMDEDUC3 DMDEDUC2
## 1       NA        1       1        1        1       NA       NA        3
## 2       NA        2      NA        1        1       NA       NA        3
## 3       NA        1       1        1        1       NA       NA        4
## 4      119       NA      NA        1        1       NA        3       NA
## 5       NA        2      NA        1        1       NA       NA        5
## 6       NA        1       2        1        1       NA       NA        4
##   DMDMARTL RIDEXPRG SIALANG SIAPROXY SIAINTRP FIALANG FIAPROXY FIAINTRP MIALANG
## 1        4       NA       1        2        2       1        2        2       1
## 2        1       NA       1        2        2       1        2        2       1
## 3        1       NA       1        2        2       1        2        2       1
## 4       NA       NA       1        1        2       1        2        2       1
## 5        1       NA       1        2        2       1        2        2       1
## 6        3       NA       1        2        2       1        2        2       1
##   MIAPROXY MIAINTRP AIALANGA DMDHHSIZ DMDFMSIZ DMDHHSZA DMDHHSZB DMDHHSZE
## 1        2        2        1        3        3        0        0        2
## 2        2        2        1        4        4        0        2        0
## 3        2        2       NA        2        2        0        0        2
## 4        2        2        1        4        4        0        2        0
```

```
## 5        2        2       NA        2        2        0        0        2
## 6        2        2        1        1        1        0        0        0
##   DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR DMDHSEDU WTINT2YR WTMEC2YR
## 1        1       69        1        3        4       NA 13281.24 13481.04
## 2        1       54        1        3        1        1 23682.06 24471.77
## 3        1       72        1        4        1        3 57214.80 57193.29
## 4        1       33        1        3        1        4 55201.18 55766.51
## 5        1       78        1        5        1        5 63709.67 65541.87
## 6        1       56        1        4        3       NA 24978.14 25344.99
##   SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR
## 1       1      112        4        4     0.84
## 2       1      108        7        7     1.78
## 3       1      109       10       10     4.51
## 4       2      109        9        9     2.52
## 5       2      116       15       15     5.00
## 6       1      111        9        9     4.79
```

As we can see, the data is quite overwhelming! Let's only select a few familiar variables to make the summary a bit more manageable and comprehensible.

```
demo <- select(demo_original,
               c("RIAGENDR", # Gender
                 "RIDAGEYR", # Age
                 "RIDRETH3", # Race
                 "DMDEDUC2") # Education
               )
```

```
head(demo)
```

```
##   RIAGENDR RIDAGEYR RIDRETH3 DMDEDUC2
## 1        1       69        4        3
## 2        1       54        3        3
## 3        1       72        3        4
## 4        1        9        3       NA
## 5        2       73        3        5
## 6        1       56        1        4
```

Awesome, our data is looking much better now!

We have learned how to analyze it with dplyr and visualize it with ggplot. But in this tutorial, we are going to learn how to summarize the data in this large dataset into one simple table.

## 8.4 2. WHAT IS TABLEONE?

tableone is an R package that helps us construct "Table 1", or the baseline table that we see in biomedical research papers. This package gives us access to a lot of useful data summary function that we can use to summarize both categorical and continuous data. In addition, we can also identify normal and nonnormal variables so that R can analyze it more accurately.

tableone is unique in that it is very simple and easy to use. One single function can do tremendous data summary as we will see in the later sections in this tutorial.

#### 8.4.0.1 DO QUESTIONS 1-2 OF THE QUIZ NOW

tableone is part of the tidyverse core. (True or False)

What sort of data can tableone summarize? (Select all that apply)

## 8.5 3. CREATING A TABLEONE

### 8.5.1 CreateTableOne

The simples way that we can use tableone is to use the function `CreateTableOne()` with the nested dataset between then `()` like so:

```
CreateTableOne(data = demo)
```

```
##
##                          Overall
##   n                      10175
##   RIAGENDR (mean (SD))    1.51 (0.50)
##   RIDAGEYR (mean (SD))   31.48 (24.42)
##   RIDRETH3 (mean (SD))    3.29 (1.61)
##   DMDEDUC2 (mean (SD))    3.52 (1.24)
```

As we can see in the output above, this function has cleanly summarize all of our data into one table. It gives us how many records there are in the dataset (n), as well as the mean and standard deviation of all of our variables!

It looks pretty neat right now, but recall that the variables **RIAGENDR (Gender), RIDAGEYR (Age), and RIDRETH3 (Race)** are all categorical! So it does not make any sense to have a mean for these variables at all.

But do not worry at all! There are actually several ways that we can solve this problem: 1. First solution is, we can use nhanesTranslate and these variables will instantly be converted to categorical, and 2. Second solution is, we can use the `factorVars` argument in `CreateTableOne()` to identify categorical variables.

### 8.5.2 Solution 1: nhanesTranslate & CreateTableOne

First, let's translate all of our variables using the `nhanesTranslate()` function that we have learned in previous tutorials like so.

```
demo_translate <- nhanesTranslate("DEMO_H",
                   c("RIAGENDR",
                     "RIDAGEYR",
                     "RIDRETH3",
                     "DMDEDUC2"),
                     data = demo)
```

```
## Translated columns: RIAGENDR RIDRETH3 DMDEDUC2
```

After that, for ease of communication, let's also change the column names to something that we can all understand.

```
names(demo_translate) <- c("Gender", "Age", "Race", "Education")
```

### 8.5.3 7.1 Try it yourself

**Challenge**: Why do you think we need to change the names of our variables AFTER we translate them?

**Hint**: Think about the `data = demo` argument in `nhanesTranslate()`

Now, this is what our dataset should look like. Look familiar?

```
head(demo_translate)
```

```
##   Gender Age               Race                        Education
## 1   Male  69 Non-Hispanic Black High school graduate/GED or equi
## 2   Male  54 Non-Hispanic White High school graduate/GED or equi
## 3   Male  72 Non-Hispanic White         Some college or AA degree
## 4   Male   9 Non-Hispanic White                              <NA>
## 5 Female  73 Non-Hispanic White         College graduate or above
## 6   Male  56   Mexican American         Some college or AA degree
```

This table should look exactly like the one that you have seen in previous tutorials! The only difference here is that, **in this tutorial, we are using and summarizing the ENTIRE dataset!** We will not be scaling down to only analyzing or visualizing the first or last few rows!

Now if we use the `CreateTableOne()` function again but on our new `demo_translate` object, we should be able to see a quite different table.

```
(tab_nhanes <- CreateTableOne(data = demo_translate))
```

```
##
##                                       Overall
##   n                                   10175
##   Gender = Female (%)                  5172 (50.8)
##   Age (mean (SD))                     31.48 (24.42)
##   Race (%)
##      Mexican American                  1730 (17.0)
```

```
##      Other Hispanic                      960 ( 9.4)
##      Non-Hispanic White                 3674 (36.1)
##      Non-Hispanic Black                 2267 (22.3)
##      Non-Hispanic Asian                 1074 (10.6)
##      Other Race - Including Multi-Rac    470 ( 4.6)
##   Education (%)
##      Less than 9th grade                 455 ( 7.9)
##      9-11th grade (Includes 12th grad    791 (13.7)
##      High school graduate/GED or equi   1303 (22.6)
##      Some college or AA degree          1770 (30.7)
##      College graduate or above          1443 (25.0)
##      Refused                               2 ( 0.0)
##      Don't Know                            5 ( 0.1)
```

The count of records (n) is still there and we are still provided with the mean and standard deviation of participants' age. However, instead of a single mean and standard deviation for gender, race, and education, we now have all of the categories of these variables fleshed out. In addition, we are also given the count and percentage of each category!

You may have also noticed that "Female" is the only gender that is shown in this table. This is because this variable only has two levels: Female and Male. For this reason, we can infer the count and percentage of the other category just based on the one that tableone gives us. There is a way that we can force tableone to show all categories of a variable. We will cover this in a later section of this tutorial.

#### 8.5.3.1 DO QUESTIONS 3-4 OF THE QUIZ NOW

> What kind of information is summarized when the data is continuous?
>
> What kind of information is summarized when the data is categorical?

### 8.5.4 Solution 2: Identify Numerical Categorical Data

Before we hop to this second solution, again, let's rename all of our variables to something more comprehensible so that everything is easier to understand. In this subsection, however, we will be renaming our `demo` dataset, instead of the `demo_translate` dataset that we renamed earlier.

```
names(demo) <- c("Gender", "Age", "Race", "Education")
```

Okay, now we are ready to go! Note that this second solution is more transferrable and will work for datasets that do not come from NHANES.

The second way that we can help tableone know which variable is categorical is by telling it directly using the argument `factorVars`. `factorVars` is especially useful for identifying numerical categorical data like the ones that we have.

Coupled with `factorVars` is also `vars`. `vars` is used to select which variables we want to keep in our tableone. Combined what we have learned about `CreateTableOne()` so far with `factorVars` and `vars`, this is what our function with clearly identified numerical categorical data should look like:

```
CreateTableOne(data = demo,
               vars = c("Gender", "Age", "Race", "Education"),
               factorVars = c("Gender", "Race", "Education")
              )
```

```
##
##                    Overall
##   n                10175
##   Gender = 2 (%)    5172 (50.8)
##   Age (mean (SD))  31.48 (24.42)
##   Race (%)
##      1              1730 (17.0)
##      2               960 ( 9.4)
##      3              3674 (36.1)
##      4              2267 (22.3)
##      6              1074 (10.6)
##      7               470 ( 4.6)
##   Education (%)
##      1               455 ( 7.9)
##      2               791 (13.7)
##      3              1303 (22.6)
##      4              1770 (30.7)
##      5              1443 (25.0)
##      7                 2 ( 0.0)
##      9                 5 ( 0.1)
```

As we can see, this tableone that we just created should look somewhat familiar to the table that we created above. The only difference is that because we did not use nhanesTranslate, all of the categories in our categorical variables are numerical. This will not be an issue if we know which number corresponds to which gender, race, or education level of the participants. Other than that, the counts and percentages of these categorical variables should be identical.

If the amount of vectors `c()` and strings in the code above is a bit confusing and hard on our eyes, we can also define `factorVars` and `vars` before inputting them into `CreateTableOne()` like so:

```
vars <- c("Gender", "Age", "Race", "Education")
```

```
factorVars <- c("Gender", "Race", "Education")
```

```
CreateTableOne(data = demo,
               vars = vars,
```

```
              factorVars = factorVars
             )
```

```
##
##                    Overall
##   n                10175
##   Gender = 2 (%)    5172 (50.8)
##   Age (mean (SD)) 31.48 (24.42)
##   Race (%)
##      1              1730 (17.0)
##      2               960 ( 9.4)
##      3              3674 (36.1)
##      4              2267 (22.3)
##      6              1074 (10.6)
##      7               470 ( 4.6)
##   Education (%)
##      1               455 ( 7.9)
##      2               791 (13.7)
##      3              1303 (22.6)
##      4              1770 (30.7)
##      5              1443 (25.0)
##      7                 2 ( 0.0)
##      9                 5 ( 0.1)
```

We should be able to see that both tables in this subsection are identical!

### 8.5.5 7.2 Try it yourself

Create a tableone without the `vars` argument. What do you see? Do you think the `vars` argument is necessary in our case? If not, in what situation(s) do you think it would be necessary?

## 8.6 4. OTHER ARGUMENTS TO CUSTOMIZE TABLEONE

There are other arguments of `CreateTableOne()` that we can use to customize and adjust our tableone!

### 8.6.1 Show All Levels

Recall how our Gender variable only shows the "Female" category. If we want both categories "Female" and "Male" to be shown, we can add `showAllLevels = TRUE` to our `print()` function like so:

```
print(tab_nhanes,
      showAllLevels = TRUE)
```

```
##
##                  level                            Overall
##   n                                               10175
##   Gender (%)     Male                              5003 (49.2)
##                  Female                            5172 (50.8)
##   Age (mean (SD))                                 31.48 (24.42)
##   Race (%)       Mexican American                  1730 (17.0)
##                  Other Hispanic                     960 ( 9.4)
##                  Non-Hispanic White                3674 (36.1)
##                  Non-Hispanic Black                2267 (22.3)
##                  Non-Hispanic Asian                1074 (10.6)
##                  Other Race - Including Multi-Rac   470 ( 4.6)
##   Education (%)  Less than 9th grade                455 ( 7.9)
##                  9-11th grade (Includes 12th grad   791 (13.7)
##                  High school graduate/GED or equi  1303 (22.6)
##                  Some college or AA degree         1770 (30.7)
##                  College graduate or above         1443 (25.0)
##                  Refused                              2 ( 0.0)
##                  Don't Know                           5 ( 0.1)
```

Another way that we can show both Male and Femal is to use `cramVars`. But this argument only works on 2-level variables (i.e. variables with only 2 categories) because all categories will be placed in the same row.

```
print(tab_nhanes,
     cramVars = "Gender")
```

```
##
##                                       Overall
##   n                                       10175
##   Gender = Male/Female (%)            5003/5172 (49.2/50.8)
##   Age (mean (SD))                         31.48 (24.42)
##   Race (%)
##      Mexican American                      1730 (17.0)
##      Other Hispanic                         960 ( 9.4)
##      Non-Hispanic White                    3674 (36.1)
##      Non-Hispanic Black                    2267 (22.3)
##      Non-Hispanic Asian                    1074 (10.6)
##      Other Race - Including Multi-Rac       470 ( 4.6)
##   Education (%)
##      Less than 9th grade                    455 ( 7.9)
##      9-11th grade (Includes 12th grad       791 (13.7)
##      High school graduate/GED or equi      1303 (22.6)
##      Some college or AA degree             1770 (30.7)
##      College graduate or above             1443 (25.0)
##      Refused                                  2 ( 0.0)
##      Don't Know                               5 ( 0.1)
```

#### 8.6.1.1 DO QUESTION 5 OF THE QUIZ NOW

What is the difference between `showAllLevels` and `cramVars`?

### 8.6.2 Nonnormal

Right now, our tableones assume that the data of all of our continuous variables are normal, but what if our data is not normal?

If we know that some or all of our continous variables are not normal, we can tell R this by using the `nonnormal` argument of `print()`. For example, if our Age variable is nonnormal, then:

```
print(tab_nhanes,
      showAllLevels = TRUE,
      nonnormal = "Age"
     )
```

```
##
##                        level                              Overall
##   n                                                       10175
##   Gender (%)           Male                                5003 (49.2)
##                        Female                              5172 (50.8)
##   Age (median [IQR])                                      26.00 [10.00, 52.00]
##   Race (%)             Mexican American                    1730 (17.0)
##                        Other Hispanic                       960 ( 9.4)
##                        Non-Hispanic White                  3674 (36.1)
##                        Non-Hispanic Black                  2267 (22.3)
##                        Non-Hispanic Asian                  1074 (10.6)
##                        Other Race - Including Multi-Rac     470 ( 4.6)
##   Education (%)        Less than 9th grade                  455 ( 7.9)
##                        9-11th grade (Includes 12th grad     791 (13.7)
##                        High school graduate/GED or equi    1303 (22.6)
##                        Some college or AA degree           1770 (30.7)
##                        College graduate or above           1443 (25.0)
##                        Refused                                2 ( 0.0)
##                        Don't Know                             5 ( 0.1)
```

In the table above, we can see that instead of the usual mean and standard deviation, we are provided with the median and interquartile range (IQR) for our nonnormal Age variable!

### 8.6.3 7.3 Try it yourself

How do you know if a variable is nonnormal? Try using the function `summary()` and look at the number under skew. How do you decide if something is normal or nonnormal? Is the decision to make “Age” nonnormal accurate?

#### 8.6.3.1 DO QUESTION 6 OF THE QUIZ NOW

The decision to make "Age" nonnormal is accurate. (True or False)

### 8.6.4 Show Categorical or Continuous Variables Only

We also have the option to only create tableones with only categorical or continuous variables.

```
## Categorical variables only

tab_nhanes$CatTable
```

```
##
##                                        Overall
##   n                                    10175
##   Gender = Female (%)                  5172 (50.8)
##   Race (%)
##      Mexican American                  1730 (17.0)
##      Other Hispanic                     960 ( 9.4)
##      Non-Hispanic White                3674 (36.1)
##      Non-Hispanic Black                2267 (22.3)
##      Non-Hispanic Asian                1074 (10.6)
##      Other Race - Including Multi-Rac   470 ( 4.6)
##   Education (%)
##      Less than 9th grade                455 ( 7.9)
##      9-11th grade (Includes 12th grad   791 (13.7)
##      High school graduate/GED or equi  1303 (22.6)
##      Some college or AA degree         1770 (30.7)
##      College graduate or above         1443 (25.0)
##      Refused                              2 ( 0.0)
##      Don't Know                           5 ( 0.1)
```

```
## Continuous variables only

print(tab_nhanes$ContTable, nonnormal = "Age")
```

```
##
##                       Overall
##   n                   10175
##   Age (median [IQR]) 26.00 [10.00, 52.00]
```

### 8.6.5 Strata

In a way, strata is like the function `group_by()` in dplyr or facets in ggplot. It groups data together into groups or "strata" and then summarizes each group individually.

Note that while `showAllLevels` and `nonnormal` are arguments of the function `print()`, `strata` is an argument of the function `CreateTableOne()`.

For example, if we want to separate our data summary by Gender, we would need to write a code like so:

```
strata <- CreateTableOne(data = demo_translate,
                         vars = c("Age", "Race", "Education"), ## Note that Gender is n
                         factorVars = c("Race","Education"), ## Again, Gender is not in
                         strata = "Gender"
                         )
```

```
print(strata,
      nonnormal = "Age",
      cramVars = "Gender")
```

```
##                                       Stratified by Gender
##                                        Male                   Female
##   n                                     5003                   5172
##   Age (median [IQR])                   25.00 [9.00, 51.00] 28.00 [10.00, 52.00]
##   Race (%)
##      Mexican American                    833 (16.7)           897 (17.3)
##      Other Hispanic                      449 ( 9.0)           511 ( 9.9)
##      Non-Hispanic White                 1811 (36.2)          1863 (36.0)
##      Non-Hispanic Black                 1152 (23.0)          1115 (21.6)
##      Non-Hispanic Asian                  521 (10.4)           553 (10.7)
##      Other Race - Including Multi-Rac    237 ( 4.7)           233 ( 4.5)
##   Education (%)
##      Less than 9th grade                 230 ( 8.3)           225 ( 7.5)
##      9-11th grade (Includes 12th grad    393 (14.2)           398 (13.2)
##      High school graduate/GED or equi    665 (24.1)           638 (21.2)
##      Some college or AA degree           754 (27.3)          1016 (33.7)
##      College graduate or above           713 (25.9)           730 (24.2)
##      Refused                               0 ( 0.0)             2 ( 0.1)
##      Don't Know                            3 ( 0.1)             2 ( 0.1)
##                                       Stratified by Gender
##                                        p      test
##   n
##   Age (median [IQR])                    0.001 nonnorm
##   Race (%)                              0.317
##      Mexican American
##      Other Hispanic
##      Non-Hispanic White
##      Non-Hispanic Black
##      Non-Hispanic Asian
##      Other Race - Including Multi-Rac
##   Education (%)                        <0.001
```

```
##       Less than 9th grade
##       9-11th grade (Includes 12th grad
##       High school graduate/GED or equi
##       Some college or AA degree
##       College graduate or above
##       Refused
##       Don't Know
```

Let's unpack this table together. Firstly, we have the usual mean and standard deviation OR median and IQR for each category of each variable. Except now, we can see that all of the variables and their categories are summarized by or stratified by Gender.

Second of all, we can also see a second table below our usual table with p-values and test. This only appears when we have stratified our data into two groups for comparison. The default test for categorical variables is `chisq.test()` and the default for continuous variables is `oneway.test()` (regular ANOVA). tableone also considers nonnorm as present by the word "nonnorm" under "test" in the table above. Otherwise, we also have the option to use `krushal.test()` for nonnormal continuous variables.

### 8.6.6 7.4 Try it yourself

Create a tableone using the demo_translate dataset. Keep all variables and stratified the data using "Age". What do you see? Do you think this is a helpful tableone?

#### 8.6.6.1 DO QUESTION 7 OF THE QUIZ NOW

> Which of the following is the least appropriate to stratify our dataset by?

## 8.7 5. EXPORT TABLEONE

Finally, let's export our tableone!

Recall that we can use the function `write.csv()` to export data from R to a csv file. But before we can use this function, we need to save the table into an object using `print()` like so first:

```
tab_csv <- print(strata,
                 nonnormal = "Age",
                 printToggle = FALSE)
```

#### 8.7.0.1 DO QUESTION 8 OF THE QUIZ NOW

> What does the argument `printToggle = FALSE` do?

Now we can use our `write.csv()` function like normal.

```
write.csv(tab_csv, file = "data/NHANES_Summary.csv")
```

Tada! Now our table is saved as a csv file in our working directory!

```
dir()
```

```
##  [1] "_book"
##  [2] "_bookdown.yml"
##  [3] "_bookdown_files"
##  [4] "_build.sh"
##  [5] "_deploy.sh"
##  [6] "_output.yml"
##  [7] "0-r-and-rstudio-set-up.Rmd"
##  [8] "1-introduction-to-r.Rmd"
##  [9] "2-importing-data-into-r-with-readr.Rmd"
## [10] "3-introduction-to-nhanes.Rmd"
## [11] "4-data-analysis-with-dplyr.Rmd"
## [12] "5-data-visualization-with-ggplot.Rmd"
## [13] "6-date-time-data-with-lubridate.Rmd"
## [14] "7-data-summary-with-tableone.Rmd"
## [15] "8-Exercise-Solutions.Rmd"
## [16] "9-references.Rmd"
## [17] "book.bib"
## [18] "data"
## [19] "DESCRIPTION"
## [20] "Dockerfile"
## [21] "docs"
## [22] "header.html"
## [23] "images"
## [24] "index.Rmd"
## [25] "intro2R.Rmd"
## [26] "intro2R_cache"
## [27] "intro2R_files"
## [28] "LICENSE"
## [29] "now.json"
## [30] "packages.bib"
## [31] "preamble.tex"
## [32] "R.Rproj"
## [33] "README.md"
## [34] "style.css"
## [35] "toc.css"
```

#### 8.7.0.2 DO QUESTIONS 9-10 OF THE QUIZ NOW

Which of the following arguments can be nested in `CreateTableOne()`?

Which of the following arguments can be nested in `print()`?

## 8.8 6. ALTERNATIVES TO TABLEONE

Data summary is one of the many applications that R specializes at. With this said, there are multiple other R packages that also do data summary aside from tableone. We will not go over any of these packages, but know that each package has its own strengths and so are most optimally used in different situations.

Here are the other data summary packages and its main data summary function:

### 8.8.1 base R

In base R, we have `summary()` and `by()`:

```
summary(demo_translate)
```

```
##     Gender          Age                                        Race
##  Male  :5003   Min.   : 0.00   Mexican American              :1730
##  Female:5172   1st Qu.:10.00   Other Hispanic                : 960
##                Median :26.00   Non-Hispanic White            :3674
##                Mean   :31.48   Non-Hispanic Black            :2267
##                3rd Qu.:52.00   Non-Hispanic Asian            :1074
##                Max.   :80.00   Other Race - Including Multi-Rac: 470
##
##                              Education
##  Some college or AA degree       :1770
##  College graduate or above       :1443
##  High school graduate/GED or equi:1303
##  9-11th grade (Includes 12th grad: 791
##  Less than 9th grade             : 455
##  (Other)                         :   7
##  NA's                            :4406
```

```
by(demo_translate, demo_translate$Gender, summary)
```

```
## demo_translate$Gender: Male
##     Gender          Age                                        Race
##  Male  :5003   Min.   : 0.00   Mexican American              : 833
##  Female:   0   1st Qu.: 9.00   Other Hispanic                : 449
##                Median :25.00   Non-Hispanic White            :1811
##                Mean   :30.69   Non-Hispanic Black            :1152
##                3rd Qu.:51.00   Non-Hispanic Asian            : 521
##                Max.   :80.00   Other Race - Including Multi-Rac: 237
##
##                              Education
##  Some college or AA degree       : 754
```

```
##  College graduate or above       : 713
##  High school graduate/GED or equi: 665
##  9-11th grade (Includes 12th grad: 393
##  Less than 9th grade             : 230
##  (Other)                         :   3
##  NA's                            :2245
## ------------------------------------------------------------
## demo_translate$Gender: Female
##     Gender          Age                                 Race
##  Male  :   0   Min.   : 0.00   Mexican American              : 897
##  Female:5172   1st Qu.:10.00   Other Hispanic                : 511
##                Median :28.00   Non-Hispanic White            :1863
##                Mean   :32.25   Non-Hispanic Black            :1115
##                3rd Qu.:52.00   Non-Hispanic Asian            : 553
##                Max.   :80.00   Other Race - Including Multi-Rac: 233
##
##                              Education
##  Some college or AA degree       :1016
##  College graduate or above       : 730
##  High school graduate/GED or equi: 638
##  9-11th grade (Includes 12th grad: 398
##  Less than 9th grade             : 225
##  (Other)                         :   4
##  NA's                            :2161
```

### 8.8.2 Hmisc

In Hmisc, we have `describe()`:

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

```
## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:dplyr':
##
##     src, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
describe(demo_translate)
```

```
## demo_translate
##
##  4  Variables      10175  Observations
## ---------------------------------------------------------------------------
## Gender
##        n  missing distinct
##    10175        0        2
##
## Value        Male Female
## Frequency    5003   5172
## Proportion  0.492  0.508
## ---------------------------------------------------------------------------
## Age : Age in years at screening
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##    10175        0       81        1    31.48    27.75        1        3
##      .25      .50      .75      .90      .95
##       10       26       52       68       75
##
## lowest :  0  1  2  3  4, highest: 76 77 78 79 80
## ---------------------------------------------------------------------------
## Race
##        n  missing distinct
##    10175        0        6
##
## lowest : Mexican American                 Other Hispanic                 Non-Hispanic White
## highest: Other Hispanic                   Non-Hispanic White             Non-Hispanic Black
##
## Mexican American (1730, 0.170), Other Hispanic (960, 0.094), Non-Hispanic White
## (3674, 0.361), Non-Hispanic Black (2267, 0.223), Non-Hispanic Asian (1074,
## 0.106), Other Race - Including Multi-Rac (470, 0.046)
## ---------------------------------------------------------------------------
## Education
##        n  missing distinct
##     5769     4406        7
##
## lowest : Less than 9th grade              9-11th grade (Includes 12th grad High school graduat
## highest: High school graduate/GED or equi Some college or AA degree       College graduate or
##
## Less than 9th grade (455, 0.079), 9-11th grade (Includes 12th grad (791,
## 0.137), High school graduate/GED or equi (1303, 0.226), Some college or AA
## degree (1770, 0.307), College graduate or above (1443, 0.250), Refused (2,
## 0.000), Don't Know (5, 0.001)
## ---------------------------------------------------------------------------
```

### 8.8.3 psych

In psych, we have `describe()` and `describeBy()`. Note how the categorical variables are marked with an asterisk (*).

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

```
##
## Attaching package: 'psych'

## The following object is masked from 'package:Hmisc':
##
##     describe

## The following object is masked from 'package:car':
##
##     logit

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
describe(demo_translate)
```

```
##            vars     n  mean    sd median trimmed   mad min max range  skew
## Gender*       1 10175  1.51  0.50      2    1.51  0.00   1   2     1 -0.03
## Age           2 10175 31.48 24.42     26   29.82 28.17   0  80    80  0.44
## Race*         3 10175  3.14  1.35      3    3.11  1.48   1   6     5  0.04
## Education*    4  5769  3.52  1.23      4    3.62  1.48   1   7     6 -0.47
##            kurtosis   se
## Gender*       -2.00 0.00
## Age           -1.09 0.24
## Race*         -0.51 0.01
## Education*    -0.69 0.02
```

```
describeBy(demo_translate, demo_translate$Gender)
```

```
##
##  Descriptive statistics by group
## group: Male
##            vars    n  mean    sd median trimmed   mad min max range  skew
## Gender*       1 5003  1.00  0.00      1    1.00  0.00   1   1     0   NaN
## Age           2 5003 30.69 24.39     25   28.89 28.17   0  80    80  0.48
## Race*         3 5003  3.16  1.34      3    3.14  1.48   1   6     5  0.03
## Education*    4 2758  3.49  1.25      4    3.58  1.48   1   7     6 -0.40
##            kurtosis   se
## Gender*         NaN 0.00
## Age           -1.07 0.34
## Race*         -0.49 0.02
```

```
## Education*    -0.78 0.02
## ------------------------------------------------------------
## group: Female
##            vars    n  mean    sd median trimmed   mad min max range  skew
## Gender*       1 5172  2.00  0.00      2    2.00  0.00   2   2     0   NaN
## Age           2 5172 32.25 24.43     28   30.72 29.65   0  80    80  0.40
## Race*         3 5172  3.12  1.35      3    3.09  1.48   1   6     5  0.06
## Education*    4 3011  3.55  1.21      4    3.65  1.48   1   7     6 -0.53
##            kurtosis   se
## Gender*         NaN 0.00
## Age           -1.12 0.34
## Race*         -0.54 0.02
## Education*    -0.59 0.02
```

### 8.8.4 desctable

In desctable, we have `desctable()`:

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

```
## Loading required package: pander

##
## Attaching package: 'desctable'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test, IQR
```

```
desctable(demo_translate)
```

```
##                                                     N           % Median IQR
## 1                                      Gender 10175          NA     NA  NA
## 2                                Gender: Male  5003 49.16953317     NA  NA
## 3                              Gender: Female  5172 50.83046683     NA  NA
## 4                                         Age 10175          NA     26  42
## 5                                        Race 10175          NA     NA  NA
## 6                     Race: Mexican American  1730 17.00245700     NA  NA
## 7                       Race: Other Hispanic   960  9.43488943     NA  NA
## 8                   Race: Non-Hispanic White  3674 36.10810811     NA  NA
## 9                   Race: Non-Hispanic Black  2267 22.28009828     NA  NA
## 10                  Race: Non-Hispanic Asian  1074 10.55528256     NA  NA
## 11      Race: Other Race - Including Multi-Rac   470  4.61916462     NA  NA
## 12                                  Education  5769          NA     NA  NA
## 13              Education: Less than 9th grade   455  7.88698215     NA  NA
## 14 Education: 9-11th grade (Includes 12th grad   791 13.71121512     NA  NA
## 15 Education: High school graduate/GED or equi  1303 22.58623678     NA  NA
```

```
## 16        Education: Some college or AA degree  1770 30.68122725     NA  NA
## 17        Education: College graduate or above  1443 25.01300052     NA  NA
## 18                          Education: Refused     2  0.03466805     NA  NA
## 19                       Education: Don't Know     5  0.08667013     NA  NA
```

### 8.8.5 skimr

In skimr, we have `skim()`:

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

```
#skim(demo_translate)
```

## 8.9 7. SUMMARY AND TAKEAWAYS

Congratulations on finishing tutorial 7 on Data Summary with tableone! After this tutorial, you should be familiar with the R package tableone as well as the function `CreateTableOne()`. In addition, you should also be familiar with the different arguments of `print()` to customize your own tableone.

There are a lot more powerful functions in the tableone package. You are free to explore them on your own using this document.

# APPENDIX

## .10 Exercise solutions

```r
require(dplyr)
```

## .11 Introduction to R

This notebook contains the solutions for all of the **Try it yourself** sections of **tutorial 1: Introduction to R**. Make sure that you have at least tried to solve these sections first before viewing this notebook.

### .11.1 1.1

a. Can you replicate and solve these problems in R?

- 2^2
- $2 \times 2$
- $2 + 5 \times (5 \div 4)$^6
- what is the remainder of $52 \div 5$
- what is the whole number solution to $82 \div 8$

```r
2^2
```

```
## [1] 4
```

```r
2 * 2
```

```
## [1] 4
```

```r
2 + 5 * (5 / 4)^6
```

```
## [1] 21.07349
```

```r
52 %/% 5
```

```
## [1] 10
```

```
82 %% 8
```

```
## [1] 2
```

b. Can you solve for x using R?

a <- 9 + 3 * 6

x <- a ÷ 2

```
a <- 9 + 3 * 6
x <- a / 2
x
```

```
## [1] 13.5
```

```
## x is equal to 13.5!
```

### .11.2 1.2

Translate the following into R and find the output: * 8 times 3 is greater than 8? * eleven divided by seven is not equal to 2? * 9 is less than or equal to 18?

```
8 * 3 > 8
```

```
## [1] TRUE
```

```
11 / 7 != 2
```

```
## [1] TRUE
```

```
9 <= 18
```

```
## [1] TRUE
```

### .11.3 1.3

Can you try storing a string? Assign the string “hello world, I am here” to the variable named start.

```
start <- "hello world, I am here"
start
```

```
## [1] "hello world, I am here"
```

**Note how the string is in `""` but the variable name is not. Why do you think this is?** Because `start` is now a known variable that stores a string. In other words, `start` does not mean “start”, instead it means “hello world, I am here”.

### .11.4 1.4

It is important to note that **R is case-sensitive**. This means that it distinguishes capitalized from non-capitalized characters, so logical and Logical are read as two separate things by R!

Try typing Logical with a capitalized "L". How does R respond to this?

```
#Logical
    ### an error message that says "object 'Logical' not found" pops up
```

### .11.5 1.5

Why do you think numeric, character, and logical are not in "" but Number, Text, and T/F are?

Because numeric, character, and logical are known variables that hold meanings (and that we already defined), while Number, Text, and T/F are not. They are actually just texts that we use to rename our column names - they do not hold any meanings.

### .11.6 1.6

1. What are 2 ways that we can print rows 1 to 5 of the data frame faithful?

```
## 1. print(faithful[1:12, ])

## 2. faithful[1:12, ]
```

2. What is the value of the cell in the fourth row and second column of the data frame faithful?

```
## faithful[4,2]
```

### .11.7 1.7

Try it yourself Write a code to find the structure of the variable waiting in the faithful dataset.

```
## str(faithful$waiting)
```

### .11.8 1.8

Remember those variables that we created earlier in the tutorial? Try finding the lengths of data frame and numeric.

**Challenge:** Psst! There are actually 2 ways for you to find the length of numeric.

```
## length(dataframe)

## length(numeric) #OR
## length(dataframe$numeric)
```

### .11.9 1.9

Recall that in order for us to refer to a variable in a dataset, we need to first type the dataset name following by a $ before we can type the variable name.

A way to avoid repeating `faithful$` everytime is to attach the dataset using `attach(faithful)`.

Try attaching the dataset faithful then find the mean of the variable eruptions without using $!

```
## attach(faithful)
## mean(eruptions)
```

## .12 Importing Data into R with readr

This notebook contains the solutions for all of the **Try it yourself** sections of **tutorial 2: Importing Data into R with readr**. Make sure that you have at least tried to solve these sections first before viewing this notebook.

### .12.1 2.1

Can you try importing the `bpx.csv` file into R using the function `read_csv()`?

```
#read_csv("../input/import/bpx.csv")
```

### .12.2 2.2

Can you identify the mistakes of the following codes?

**a.** The pathway is not in `""`

**b.** “DEMO” should not be capitalized

**c.** “Read_csv” should not be capitalized

**d.** The entire pathway needs to be in `""` instead of just the file name

### .12.3 2.3

Just by looking at the actual data frame, can you guess what type of data `col_double()` and `col_character()` are?

(**HINT**: doubles? integers? logical? character?)

Here is a list of `col_x()` and what they mean: * `col_double()` – Doubles * `col_integer()` – Integers * `col_logical()` – True/False * `col_date()` – Date * `col_time()` – Time * `col_datetime` – Date and Time * `col_character()` – Text, Character, and everything else

This list is not extensive, so you can use `?cols` to learn more about column specification!

### .12.4 2.4

You may also notice that the header of *Skip_2* is incorrect. This is because R recognizes the header of our data as the first row, thus omiting it when importing `demo.csv` into R.

Let's say this is not what we really want. What we actually want to do is to remove the first two rows of actual data while keeping the header. What do you think we have to do to achieve this?

(**HINT**: Recall what we learn about extracting rows in tutorial 1)

```
#DEMO[3:10, ]
```

### .12.5 2.5

Import the `bpx.xlsx` into R using the `read_excel()` function.

```
#read_excel("../input/import/bpx.xlsx")
```

## .13 Introduction to NHANES

This notebook contains the solutions for all of the **Try it yourself** sections of **Tutorial 3: Introduction to NHANES**. Make sure that you have at least tried to solve these sections first before viewing this notebook.

### .13.1 3.1

Find all the Examination Data in survey cycle 2013-2014

```
## nhanesTables('EXAM', 2013)
```

### .13.2 3.2

Import the blood pressure dataset in the Examination Data in survey cycle 2013-2014

```
## bpx <- nhanes('BPX_H')
```

### .13.3 3.3

Translate the following variables in the BPX dataset

BPXPULS - Pulse regular or irregular?

BPAARM - Arm selected

```
## bpxtranslate <- nhanesTranslate('BPX_H',  c('BPXPULS', 'BPAARM'), data = bpx)
```

## .14 Data Analysis with dplyr

This notebook contains the solutions for all of the **Try it yourself** sections of **Tutorial 4: Data Analysis with dplyr**. Make sure that you have at least tried to solve these sections first before viewing this notebook.

### .14.1 4.1

In the bpx dataframe, keep the following variables and top 5 rows only:

- SEQN
- PEASCST1
- PEASCTM1
- BPXSY1
- BPXDI1

```
## new_bpx <- select(bpx,
##                   c(SEQN,
##                    PEASCST1,
##                    PEASCTM1,
##                    BPXSY1,
##                    BPXDI1))
## new_bpx <- head(new_bpx,5)
```

### .14.2 4.2

- SEQN -> id
- PEASCST1 -> bp_status
- PEASCTM1 -> bpt_sec
- BPXSY1 -> systolic
- BPXDI1 -> diastolic

```
## new_bpx <- rename(bpx,
##                   id = SEQN,
##                   bp_status = PEASCST1,
##                   bpt_sec = PEASCTM1,
##                   systolic = BPXSY1,
##                   diastolic = BPXDI1)
```

## .14.3 4.3

In the demo dataframe, find all the records that:

### .14.3.1 4.3.1 the participant who is a male

```
## filter(final_demo, gender == "Male")
```

### .14.3.2 4.3.2 the participant who is a male and is older tha 50 years old

```
## filter(final_demo, gender == 'Male' && age > 50)
```

### .14.3.3 4.3.3 the education level is missing

```
## filter(final_demo, is.na(edu))
```

## .14.4 4.4

Re-order the rows in the bpx dataset by Blood Pressure Time in Seconds (bpt_sec) in descending order:

```
## arrange(final_bpx,desc(bpt_sec))
```

## .14.5 4.5

### .14.5.1 4.5.1

Create a new variable called called **rescale_bpt_sec** that records the Blood Pressure Time in miuntes. Keep both original and new variables.

```
##  mutate(final_bpx,rescale_bpt_sec = bpt_sec/60)
```

### .14.5.2 4.5.2

Create **rescale_bpt_sec** in the same way above and **only keep new variables**.

Note: Try to avoid using select().

```
##  transmute(final_bpx,rescale_bpt_sec = bpt_sec/60)
```

## .14.6 4.6

Find the average age in the demo dataframe per education level

```
##  by_edu <- group_by(edu)

## summarize(by_edu, average_age = mean(age, na.rm = TRUE))
```

### .14.7 4.7

Return the observations which has more than 3 records in each gender group.

```
## by_gender <-group_by(final_demo,gender)

## filter(by_edu,n()>3)
```

### .14.8 4.8

Compute the difference in each age and the average mean in each education level group

```
## by_edu <-group_by(final_demo,edu)

## mutate(by_edu, diff_age = age - mean(age, na.rm = T))
```

### .14.9 4.9

Re-write the following code using pipe operator

```
## temp <- filter(final_bpx, systolic > 120)
## temp <- mutate(temp, bpt_min = bpt_sec/60)
## temp
```

```
## final_bpx %>%
##    filter(systolic > 120) %>%
##    mutate(bpt_min = bpt_sec/60)
```

## .15 Data Visualization with ggplot2

This notebook contains the solutions for all of the **Try it yourself** sections of tutorial 5: Data Visualization with ggplot. Make sure that you have at least tried to solve these sections first before viewing this notebook.

### .15.1 1.1

Plot a scatterplot to show the relationship between Diastolic Blood Pressure (x-axis) and Blood Pressure Time in Seconds (y-axis).

```
## ggplot(data = demo_bpx) +
##   geom_point(aes(x = Diastolic,
```

```
##                  y = BPT),
##              na.rm = TRUE)
```

### .15.2 1.2

Plot a scatterplot using the Diastolic (x-axis) and Systolic (y-axis) variables in the data frame demo_bpx where all of the data points are blue.

```
## ggplot(demo_bpx) +
##   geom_point(aes(x = Systolic,
##                  y = Diastolic),
##              color = "blue",
##              na.rm = TRUE)
```

### .15.3 1.3

Try increasing and decreasing the binwidth of a frequency polygon. What differences do you see? What does binwidth actually mean?

Higher binwidths give us less fluctuations than lower binwidths. When we are increasing the binwidths, we are actually increasing the length of intervals, which leads to less intervals. In other words, each data point are a bit further away from each other, so when connected, the graph looks less detailed and smoother.

### .15.4 1.4

Try recreating the graph above but without any missing (NA) values.

**HINT**: Filter out any information we do not need using logical operators!

First, we need to filter out all of the NA values:

```
## no_NA <- filter(demo_bpx, Race != "NA" & Gender != "NA")
```

Then, we can graph our facets normally:

```
## ggplot(no_NA) +
##   geom_point(aes(x = Diastolic, y = Systolic), na.rm = TRUE) +
##   facet_grid(Race ~ Gender)
```

## .16 Date and Time Data with lubridate

This notebook contains the solutions for all of the **Try it yourself** sections of **tutorial 6: Date & Time Data with lubridate**. Make sure that you have at least tried to solve these sections first before viewing this notebook.

### .16.1 6.1

After running the `today()` and `now()` codes above, what do you see?

Try to also change the time zone to where you are or to something else. Now what do you see when you run `today()` and `now()`?

After running `today()` and `now()`, you should see that `today()` is a date data and `now()` is a datetime data.

Changing the time zone means that the date and time will be different if we run `today()` and `now()` again.

### .16.2 6.2

Try creating a new column named "Day_visit" that only contains information of the Year, Month, and Day columns using the Friends_visits dataframe that we just created.

```
## head(
##   Friends_visits %>%
##   mutate(Hour_visit = make_datetime(Year, Month, Day))
##    )
```

### .16.3 6.3

Using the functions introduced above, solve the following questions: 1. What day of the week is April 2, 2014? 2. What day of the year is 2017-09-15? 3. What day of the month is 20190830? 4. Find the months of the last 11 records of the column Visit_hour.

```
#1. wday(mdy("April 2, 2014"), label = TRUE)

#2. yday(ymd("2017-09-05"))

#3. mday(ymd(20190830))

#4. tail(month(Friends_visits$Visit_time, label = TRUE), 11)
```

### .16.4 6.4

Using the same Friends_visit dataset, create a similar graph as above but the x-axis is days of the week. In other words, create a bar graph that shows how many visits there are in each day of the week.

```
## Friends_visits %>%
##  mutate(Week_day = wday(Visit_time, label = TRUE, abbr = FALSE)) %>%
##  ggplot(aes(Week_day)) +
##  geom_bar()
```

### .16.5 6.5

Try to update our month to 13. What happened to our date/time? What is the output?

```
# DT %>%
#     update(month = 13)

## The year turns to 2021 because a year only has 12 months!
```

### .16.6 6.6

Recreate the frequency polygon above but change the binwidth so that all flights within each 30 minutes are clumped into one single data point. How do the graphs differ? Can you think of a few scenarios where one would be preferred?

```
## Friends_visits %>%
##    mutate(Visit_hour = update(Visit_time, yday = 1)) %>%
##    ggplot(aes(Visit_hour)) +
##    geom_freqpoly(binwidth = 900)
```

## .17 Data Summary with tableone

This notebook contains the solutions for all of the **Try it yourself** sections of **tutorial 7: Data Summary with tableone**. Make sure that you have at least tried to solve these sections first before viewing this notebook.

### .17.1 7.1

**Challenge**: Why do you think we need to change the names of our variables AFTER we translate them?

**Hint**: Think about the `data = demo` argument in `nhanesTranslate()`

Because after we use the function `names()` our data is no longer recognizable by R as being connected to the DEMO_H dataset that we downloaded from NHANES.

### .17.2 7.2

Create a tableone without the `vars` argument. What do you see? Do you think the `vars` argument is necessary in our case? If not, in what situation(s) do you think it would be necessary?

The `vars` argument tells R which variables you want to include in your tableone. If it is missing from `CreateTableOne()`, this means that we want R to select **ALL** variables of the original table in our tablone. Therefore, in our case, `vars` is not neccesary because we are selecting all variables anyway. But in cases where we only want to select some variables to include in our tableone, `vars` is an absolute must!

### .17.3 7.3

How do you know if a variable is nonnormal? Try using the function `summary()` and look at the number under skew. How do you decide if something is normal or nonnormal? Is the decision to make “Age” nonnormal accurate?

After you plug the dataset name into `summary()`, you should see a numerical value below “skew” that tells us the skewness of the data. The further it is from 0 (both negative and positive), the further it is from normal! The decision to make “Age” nonnormal is arguably not quite accurate because the skewness of “Age” is only 0.4.

### .17.4 7.4

Create a tableone using the demo_translate dataset. Keep all variables and stratified the data using “Age”. What do you see? Do you think this is a helpful tableone?

```
## CreateTableOne(data = demo_translate,
##               vars = c("Race", "Education", "Gender"),
##               factorVars = c("Race","Education", "Gender"),
##               strata = "Age"
##              )
```

Just like how we should not use continuous variables to create facets (check ggplot tutorial), we should also not stratified our data using continuous variables. When we stratify our data using numerical continuous variables, the tableone becomes too big to handle. The information that we get is also not meaningful as there is a lot of values including lots of NA values. If we want to stratify our data by age, it may be better to use age groups instead of single age values.