The R Handbook for ESM Students

Anthony Schmidt

2020-05-27

[Introduction 4](#_Toc41472751)

[R Basics 4](#_Toc41472752)

[Installing R 4](#_Toc41472753)

[Installing RStudio 4](#_Toc41472754)

[RStudio Features 5](#_Toc41472755)

[Setting up RStudio 6](#_Toc41472756)

[R Files 9](#_Toc41472757)

[.R - R Scripts 9](#_Toc41472758)

[.rproj - R Projects 9](#_Toc41472759)

[.RData - R Data Files 11](#_Toc41472760)

[.RMD - R Markdown 12](#_Toc41472761)

[R Data Structures 12](#_Toc41472762)

[Vector 12](#_Toc41472763)

[Dataframe 12](#_Toc41472764)

[Tibble 13](#_Toc41472765)

[List 13](#_Toc41472766)

[Installing Packages 13](#_Toc41472767)

[CRAN 13](#_Toc41472768)

[Github 14](#_Toc41472769)

[Loading Packages 14](#_Toc41472770)

[Importing Data 14](#_Toc41472771)

[Importing a CSV 15](#_Toc41472772)

[Importing an Excel file 16](#_Toc41472773)

[Importing an SPSS data file 16](#_Toc41472774)

[RStudio Tips to Help Write Efficient Code 17](#_Toc41472775)

[Commenting 17](#_Toc41472776)

[Sectioning Scripts 17](#_Toc41472777)

[Data Object Names 18](#_Toc41472778)

[Shortcuts 18](#_Toc41472779)

[Data Preparation and Cleaning in R 18](#_Toc41472780)

[Introduction to the Tidyverse 18](#_Toc41472781)

[Extracting a variable 19](#_Toc41472782)

[Make new variables (columns) 19](#_Toc41472783)

[Viewing Your Data 19](#_Toc41472784)

[The RStudio Environment 20](#_Toc41472785)

[Structure of Data - str() 20](#_Toc41472786)

[Variable/Column Names - names() 21](#_Toc41472787)

[First n Rows - head() 21](#_Toc41472788)

[Renaming Variables 23](#_Toc41472789)

[Renaming Multiple Variables 24](#_Toc41472790)

[Cleaning Names with janitor 25](#_Toc41472791)

[Summary Stats - describe() 26](#_Toc41472792)

[Summary Stats - describeBy() 27](#_Toc41472793)

[Summary Stats - summary() 28](#_Toc41472794)

[Summary Stats - skim() 28](#_Toc41472795)

[Crosstabs - table() 29](#_Toc41472796)

[Keeping and Dropping Variables 33](#_Toc41472797)

[Keeping and Dropping Rows 34](#_Toc41472798)

[More Frequencies and Descriptives 34](#_Toc41472799)

[Spotting Coding Mistakes 35](#_Toc41472800)

[Modifying Data - mutate() 36](#_Toc41472801)

[Reordering Categories - factor() 36](#_Toc41472802)

[Clearing Whitespace in Text - str\_trim() 38](#_Toc41472803)

[Combining Categories - case\_when() 40](#_Toc41472804)

[Splitting variables with split() 42](#_Toc41472805)

[Descriptives 43](#_Toc41472806)

[Mean - mean() 43](#_Toc41472807)

[Median - median() 44](#_Toc41472808)

[Using the summarize() function - summarize() 44](#_Toc41472809)

[Analyzing Data by Groups - group\_by() 44](#_Toc41472810)

[Spotting Outliers 45](#_Toc41472811)

[Detecting Numerical Outliers 45](#_Toc41472812)

[Detecting outliers with Z-scores 48](#_Toc41472813)

[Dealing with Outliers 48](#_Toc41472814)

[Assessing Normality 52](#_Toc41472815)

[Histograms 52](#_Toc41472816)

[Density (Curve) Plots 53](#_Toc41472817)

[QQ Plots 53](#_Toc41472818)

[Skewness and Kurtosis 54](#_Toc41472819)

[Tests of Normality 54](#_Toc41472820)

[Transforming Variables 55](#_Toc41472821)

[Working with Missing Data 55](#_Toc41472822)

[Identifying missing data 55](#_Toc41472823)

[Per Variable 55](#_Toc41472824)

[Entire Data Frame 56](#_Toc41472825)

[Dropping missing data 58](#_Toc41472826)

[Replacing missing data with 0s 58](#_Toc41472827)

[Mean imputation 59](#_Toc41472828)

[Multiple imputation 59](#_Toc41472829)

[More Data Manipulation in R 60](#_Toc41472830)

[Wide and Long Data 60](#_Toc41472831)

[Long Data 60](#_Toc41472832)

[Wide Data 60](#_Toc41472833)

[Pivoting data from wide to long - pivot\_longer() 60](#_Toc41472834)

[Pivoting data from long to wide - pivot\_wider() 60](#_Toc41472835)

[Joining multiple data objects 60](#_Toc41472836)

[Statistics in R 60](#_Toc41472837)

[T-Tests 60](#_Toc41472838)

[Chi Square 60](#_Toc41472839)

[Regression 60](#_Toc41472840)

[ANOVA 60](#_Toc41472841)

[MANOVA 60](#_Toc41472842)

[ANCOVA 60](#_Toc41472843)

[Logistic Regression 60](#_Toc41472844)

[Factor Analysis 60](#_Toc41472845)

[Resouces 60](#_Toc41472846)

# Introduction

Welcome to the R Handbook for ESM Students. This handbook will be a hands-on guide to help you learn R. It will take you from install, to set up, from data cleaning to analysis.

This guide uses real data to help you practice with R. Specifically, it uses survey data from the [RStudio Learning R Survey](https://github.com/rstudio/learning-r-survey).

# R Basics

This chapter will introduce you to installing R, becoming familiar with the RStudio environment, installing packages, and loading data.

## Installing R

Your adventure with R begins with downloading and installing the required software, which is free! This will first require you to install **R**, which is the underlying language and software we will be working with. R is available for Windows, Mac OS, and Linux.

**You can download R here:** [**https://cran.rstudio.com/**](https://cran.rstudio.com/)**.**

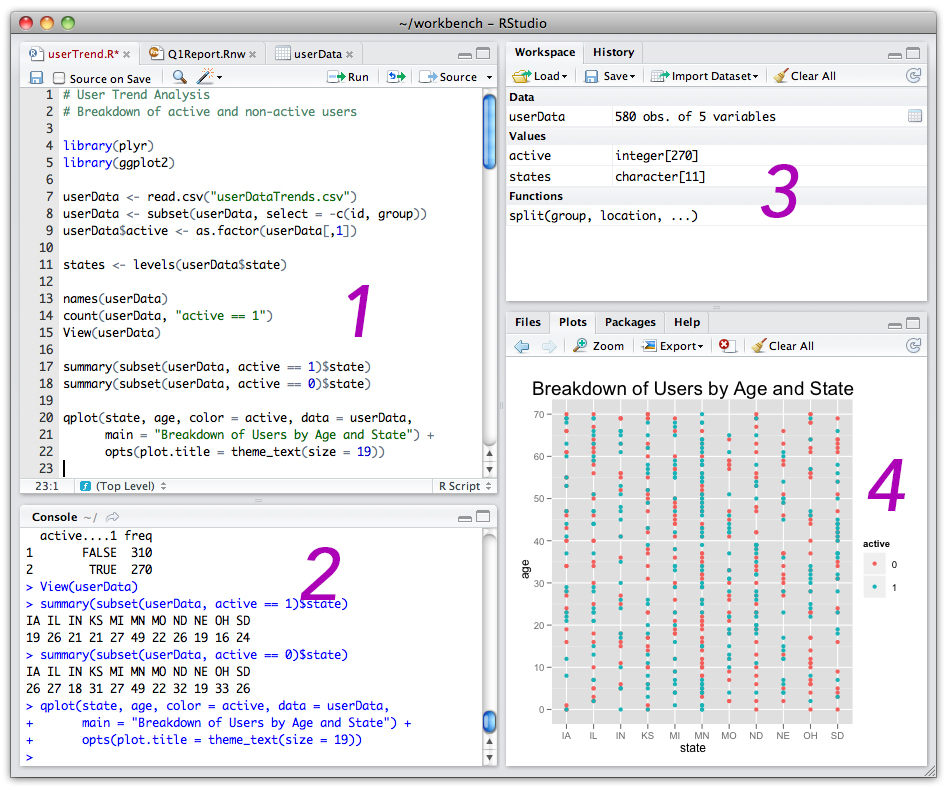
* For Windows users, you will want to download R Base. [Here is a direct link with further instructions](https://cran.rstudio.com/bin/windows/base/).
* For Mac users, you will [follow the directions here](https://cran.rstudio.com/bin/macosx/).

## Installing RStudio

Once R is installed, you will download and install **RStudio Desktop**. This is the environment (software program) in which you will work with R. RStudio is also available for Windows, Mac and various versions of Linux..

**You can download RStudio here:** [**https://rstudio.com/products/rstudio/download/#download**](https://rstudio.com/products/rstudio/download/#download)**.**

## RStudio Features



RStudio Default Environment

The RStudio environment is organized into four panes:

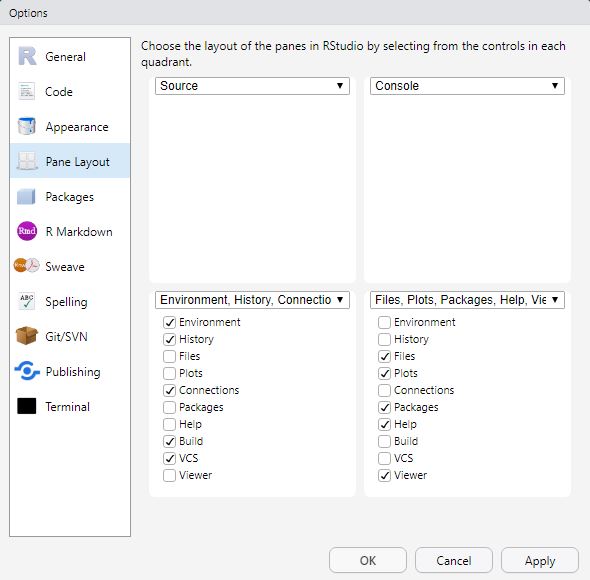
1. **Source** - The source is where you will write and execute your code. This is your script or syntax.
2. **Console** - The console displays output, warnings, and more.
   * The console is where you will see the output of your code. For example, if you run a script to calculate a mean in the source, the mean will appear in the console.
   * The console will also show any error messages or warnings. These will appear in red.
   * Any packages that you will install should be written here, not in your script. This is because you will only install a package once. If it is in your script and you run all of your script, it will try to install the packages multiple times.
     + Note: You can also include the install command in your script (at the top) and use a # sign to comment it out so that it doesn’t run when the script us run. This is useful if you are sharing a script and the other user may need to install the packages or if you may need to install them in the future.
     + Whether you write it in the script or the console is a personal choice.
3. **Environment** - The environment is mainly used to show the data frames and other files you have loaded. It will tell you the datafile name, how many rows (*obs.*) and how many columns (*variables*).
4. **Files, Plots, Packages, Help, and Viewer**
   * The Files tab will show local files in your R Project.
   * The Plots tab will show any visualizations you have created.
   * The Packages tab will allow you to see, load, and unload any packages you have installed. You can also install packages through this tab.
     + We will usually install packages using the install.packages() function and load packages using the library() function.
   * The Help tab will show explanations and examples of functions.
     + We can ask for help in the console by using ?. For example, ?mean will give additional information about the mean() function.
   * Viewer is used for any webpages that get produced through your code.

## Setting up RStudio

Once RStudio is installed, I recommend setting it up so that the environment is more comfortable to work in.

One way to do this is to reconfigure the panes so that they are easier to navigate.

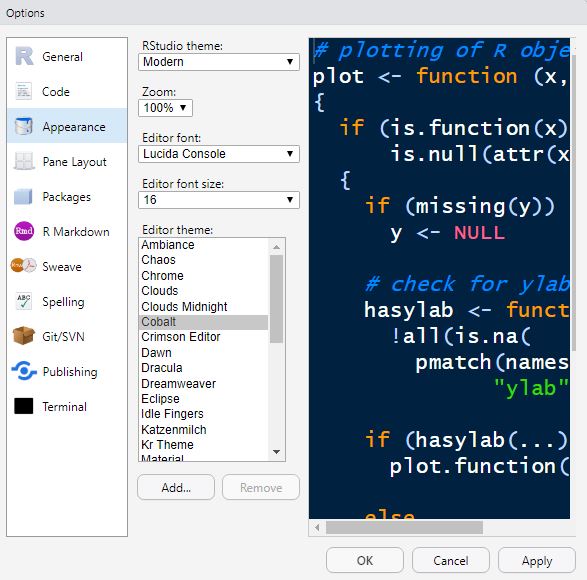
At the top of RStudio, click on **Tools > Global Options** and select **Pane Layout**. Change your options to the ones in the image below. This will allow you to read your output right next to your script. You can also easily minimize your environment, as you will not need to look there very often.



Pane Layout

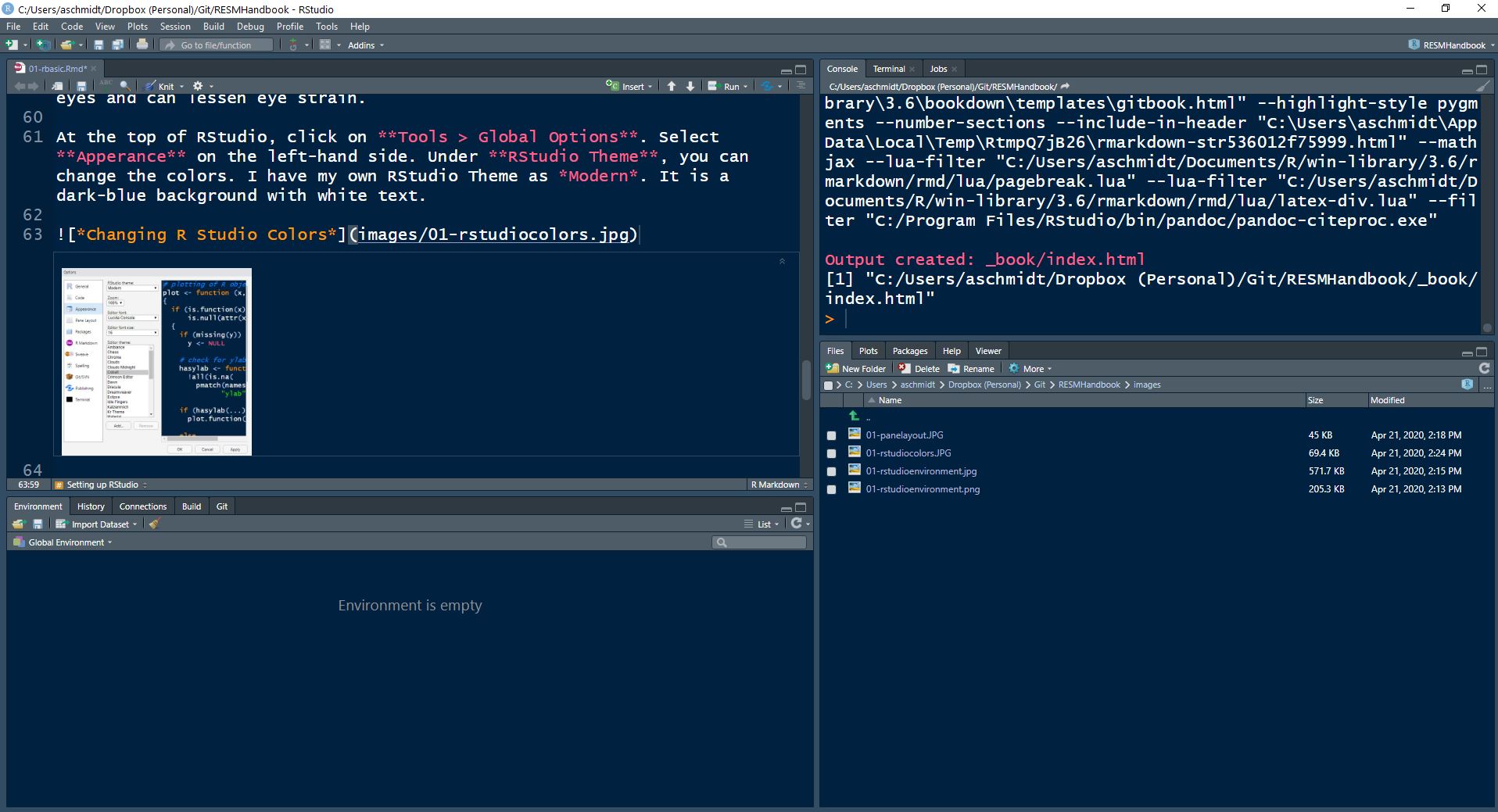
Next, you can switch the colors of R to something more comfortable for your eyes. A darker background reduces the amount of light hitting your eyes and can lessen eye strain.

At the top of RStudio, click on **Tools > Global Options**. Select **Apperance** on the left-hand side. Under **RStudio Theme**, you can change the colors. I have my own RStudio Theme as *Modern*. It is a dark-blue background with white text.



Changing R Studio Colors

Changing your theme to *Modern* and switching your panes as I recommended will make your RStudio look like this:



A More Comfortable RStudio

## R Files

Before working with R, there are several different R filetypes that you should learn about.

### .R - R Scripts

Files that end in .R are your basic R scripts. You can create a single R script for your project, or create one for each phase, i.e., a data cleaning script, a script for RQ1, a script for RQ2, etc.

### .rproj - R Projects

R Projects are very important to utilize. Without creating an R Project, you will need to include a file path to your working directory and run this code every time you start your script.

To set a working directory, you would write the following in your **source** pane, inside the R script. You would then highlight the code or place your cursor somewhere inside the code and click **Run** at the top right of the **Source** pane, or Click *Ctrl+Enter* (or on a Mac, **Cmd+Enter**).

setwd("C:/path/to/your/directory")

*Quotes, correct capitalization, and forward slashes are important here.*

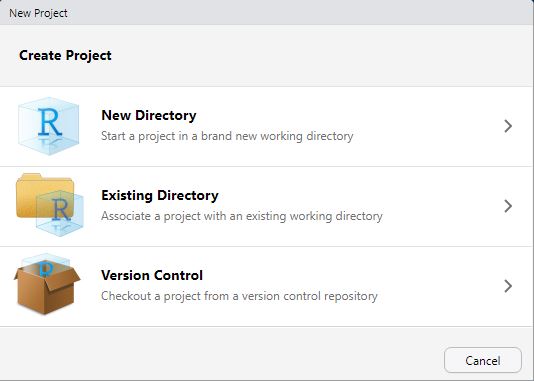
If you move your files to a new directory, you will have to rewrite this command or nothing in your script will run.

You can also set a working directory through RStudio by going to **Session > Set Working Directory** and selecting one of the choices. You will have to do this every time you open the R script.

R Projects avoid all of this. Any file that is in the same folder as the .Rproj file is automatically linked to the R Script. That means you can create a folder with your project, move that folder anywhere or give it to anyone, and when they double-click the .Rproj file, it will work!

Any subdirectories will still need to use a short path name, i.e., read\_csv("subfolder/file.csv"), but this is much simpler than utilizing the long path name.

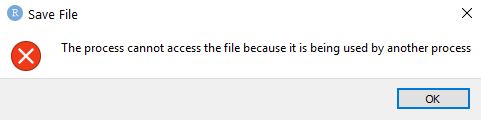
To create an R Project, simply go to **File > New Project…**. You will see the following screen:



New Project Menu

* If you want to create a new directory, select **New Directory**. On the next screen, choose **New Project**, locate where you want to put your project, and click **Create Project**
* If you have a directory already and you just want to make it into a project, click **Existing Directory**
* The **Version Control** option is useful if you will be placing your project on Github (see section on [Github](#github)).

There is a small issue with R Projects if you save them in Dropbox on your local machine. As Dropbox is constantly trying to sync your files to the cloud, it causes some interference with RStudio. You will see this message flash every 10 mins or so. It is nothing serious, just annoying. Click OK (or press the spacebar) and continue working. This [forum post may provide a solution to the problem](https://community.rstudio.com/t/mostly-resolved-dropbox-box-sync-google-drive-etc/6098).



Annoying Message

### .RData - R Data Files

As you begin working in R, you will inevitably have a number of dataframes saved to your **Environment** pane. These may be datasources you loaded, dataframes you created, or raw data, cleaned data, analyses, models, etc. While you could simply re-run your script, you can also save everything in your Environment to an R Data file. When you load the R Data file, all of those dataframes will reload. An additional bonus is that these files are smaller due to compression!

This is particularly useful if you are creating the data frames in one script and will use them in another script.

To save your Environment to an R Data file, you have a few options. You can run these in your script or in the console. That’s up to you.

#### Save .RDS

This allows you to save a **single** object in the environment:

# Save an object to a file  
saveRDS(object, file = "my\_data.rds")  
  
# Restore the object  
readRDS(file = "my\_data.rds")

#### Save .RData

This allows you to save specific objects:

# Saving on object in RData format  
save(data1, file = "data.RData")  
  
# Save multiple objects  
save(data1, data2, file = "data.RData")  
  
# To load the data again  
load("data.RData")

#### Save your entire workspace

You can save your entire environment:

# Save all objects  
save.image(file = "my\_work\_space.RData")  
  
# Load your environment  
load("my\_work\_space.RData")

### .RMD - R Markdown

R Markdown combines R and document or webpage authoring in a single file. R Markdown is useful if you want to write code and a report, render that report to a Word file, a PDF, or publish it online at [RPubs](http://www.rpubs.com). However, it is beyond the scope of this handbook. Please see the [free, online R Markdown book, *R Markdown: The Definitive Guide*](https://bookdown.org/yihui/rmarkdown/).

## R Data Structures

A number of different objects can exist in the RStudio environment. Here are a few definitions.

### Vector

Vectors are basically one single column of a data frame. All elements in this dimension must be the same type (e.g. integers, strings, logical elements like TRUE and FALSE).

The following commands create vectors.

name <- c("Mike", "Will", "Dustin", "Lucas", "Elle")   
age <- c(20, 25, 30, 35, 40)

**A note on c**

The c in the commands above mean to “combine” into a vector. It is used in many functions where you need to combine several items toegther.

### Dataframe

A data frame is the easiest type of data to work with. Consider it equivelent to an Excel spreadhseet. It has columns and rows. Each column can be a different type (character, factor, numeric, date).

You can import data and it usually imports as a data frame, or you can make your own by combining vectors:

dataframe\_example <- data.frame(name=c("Thor", "Iron Man"),  
 age=c(1000, 48),  
 earth=c(FALSE, TRUE))  
dataframe\_example

## name age earth  
## 1 Thor 1000 FALSE  
## 2 Iron Man 48 TRUE

### Tibble

Tibbles are very similar to dataframes. They are often referred to when talking about Tidyverse packages or functions. Mostly, a tibble, when shown in the *console* will include its data type with it (character, factor, integer, etc.).

tibble\_example <- as\_tibble(dataframe\_example)  
  
tibble\_example

## # A tibble: 2 x 3  
## name age earth  
## <fct> <dbl> <lgl>  
## 1 Thor 1000 FALSE  
## 2 Iron Man 48 TRUE

[Read more about Tibbles here](https://blog.rstudio.com/2016/03/24/tibble-1-0-0/).

### List

A list is a collection of objects. For example, a list can hold multiple dataframes, or a vector, a matrix, etc. They are much more difficilt to work with.

[Read more about lists here](https://www.tutorialspoint.com/r/r_lists.htm).

## Installing Packages

Many packages come pre-loaded. Packages perform different jobs in R. Packages allow you to use specific functions in R. They allow R to do new things.

For example, the psych package contains functions to perform various statistical analyses. It contains functions such as cohen.d() to calculate Cohen’s *d*, cor.plot() to create a correlation plot, or describe() to describe data.

### CRAN

CRAN is the official R package repository. To install a CRAN package, you simply use the install.packages() function. The package name is entered in “quotes” inside the parentheses. Package names are case sensitive. The example below shows how to install the tidyverse package, a package you will use often. This should be entered into the **console** not the **source/script** pane.

Enter the following into the **console** and hit **enter**:

install.packages("tidyverse)

You can also use R Studio to search for and install packages by clicking on the “Pacakges” tab and clicking “Insall”.

### Github

One reason R is a popular statistical programming language is because it is open source. That means anyone can make and distribute a package of useful functions. Many packages can be found on Github, an online platform for software development. Github is a great source for packages, but exercise caution as packages here may still be in development or may be beta versions not full tested and deployed.

If you wish to install a package from Github, first, you’ll need to install the devtools package:

Enter the following into the **console** and hit **enter**:

install.packages("devtools")

After installation, you can begin installing packages from Github:

Enter the following into the **console** and hit **enter**" devtools::install\_github("waffle")

**A note on ::**

You may notice this command started with devtools::. What does :: mean? We can use the double colons when we want to use a function from a package but don’t want to load the package. For example, we could have loaded the devtools package and then used the install\_github() function, but that would be two steps and require more memory for the loading of devtools. Instead, we have just one step and less memory usage!

## Loading Packages

To load packages, we use the library() function. It’s good practice to write all the packages you load at the **top** of the R script. If you are writing a script and realize you need to load a new package, add it in at the top. This way, when you go to run your script again, all packages are loaded first before anything else gets processed.

A typical R script might look like this at the **top**:

library(tidyverse)  
library(psych)  
library(ggplot2)

To run such a command, you would highlight the entire block of text and hit run (Ctrl/Cmd+Enter).

## Importing Data

In this section, we will learn how to import data in different formats.

To practice these and other skills in this handbook, you first need to create an R Project. Follow the steps above to create an **“R ESM Handbook Practice”** project.

**A note on organization**

I highly recommend placing raw data files inside a “data” folder in your R Project. This will keep your directories organized and easy to navigate.

### Importing a CSV

CSV files are quite common. While they look like standard Excel files, they are simply text files with columns separated by commas (and sometimes tabs).

#### Importing a Local CSV File

To load a csv file, we will use the read\_csv() command.

Download the following CSV file and save it within your *data* folder in your **“R ESM Handbook Practice”** R Project.

[**Download the “2019 English R Community Survey Responses.csv” file.**](/data-files/01/2019%20English%20R%20Community%20Survey%20Responses.csv)

Use the following command in your script and run it:

read\_csv("data/2019 English R Community Survey Responses.csv")

If you noticed, you got a lot of text output in your console. That’s not very useful, is it?

We need to save this information to a data object in the environment. To do that we will use the <- arrow operator.

Try this again:

rsurvey\_csv <- read\_csv("data/2019 English R Community Survey Responses.csv")

Now you will see the data was saved to the environment. It should say: rsurvey\_csv and have 1838 obs. (rows or individual cases) and 52 variables (columns).

**A note about <-**

In R, <- is the most common assignment operator. You can also use the equal sign =, but that has several other uses whereas <- only has one - to assign data to an object.

#### Importing a CSV File from the Internet

You can load data directly from the internet using the same function as before. Just use a web address instead of the file path.

rsurvey\_csv\_from\_net <- read.csv("https://github.com/rstudio/learning-r-survey/blob/master/2019/data/2019%20English%20R%20Community%20Survey%20Responses%20.csv?raw=true")

For more information, use the ?read.csv help command.

### Importing an Excel file

Download the following Excel file and save it within your *data* forlder in your **“R ESM Handbook Practice”** R Project.

[**Download the “2019 English R Community Survey Responses.xlsx” file.**](/data-files/01/2019%20English%20R%20Community%20Survey%20Responses.xlsx)

To read in an Excel file, we need to first download and install the readxl package.

In the **console**, use the following command:

install.packages("readxl")

After it downloads and installs, we can use the library() function to **load** the readxl package.

Write and run:

library(readxl)

Now we can load the Excel file with the read\_excel() function:

rsurvey\_excel <- read\_excel("data/2019 English R Community Survey Responses.xlsx")

If your Excel file has a number of different worksheets, we could also specify which sheet you want to load with:

rsurvey\_excel <- read\_excel("data/2019 English R Community Survey Responses.xlsx", sheet="Form Responses 1")

Note that the quotation marks are around the file path name and the sheet name.

For more information, use the ?read\_excel help command.

### Importing an SPSS data file

Download the following SPSS file and save it within your *data* folder in your **“R ESM Handbook Practice”** R Project.

[**Download the “2019 English R Community Survey Responses.sav” file.**](/data-files/01/2019%20English%20R%20Community%20Survey%20Responses.sav)

SPSS .sav files can be read into R using the haven package.

haven is part of the tidyverse suite of packages. You will learn about the tidyverse in a later section. For now, you can install the tidyverse package in the **console**, using the following command:

install.packages("tidyverse")

After it downloads and installs, use the library() function to load the package. Add this line near the top of your script:

library(haven)

Now we can load the SPSS file with the read\_sav():

rsurvey\_spss <- read\_.spss\_sav("data/2019 English R Community Survey Responses.sav")

You can also import SAS and Stata files, too. For more information, use the ?haven or ?read\_.spss\_sav help commands. You can also visit the [haven website](https://haven.tidyverse.org/).

## RStudio Tips to Help Write Efficient Code

### Commenting

To stay organized, it is important to comment your code. To make a comment in your code, you can simply use the hashtag sign #. Here are some examples:

# Load foreign package for SPSS  
library(foreign)

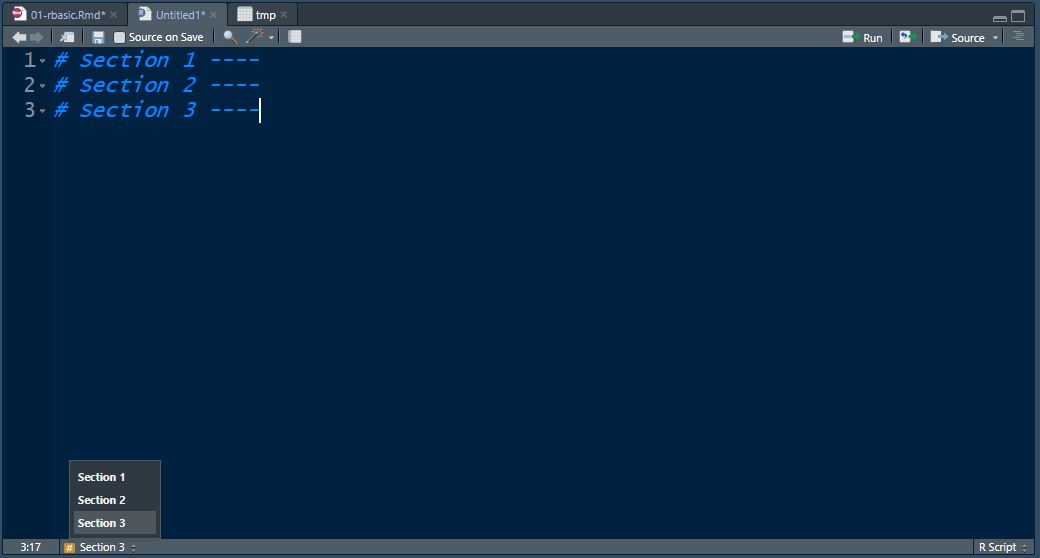
library(foreign) # Load foreign package for SPSS

rsurvey\_spss <- read.spss("data/2019 English R Community Survey Responses.sav",   
use.value.labels = FALSE, #I don't need value labels  
to.data.frame = TRUE) #I don't want to work with a list!

### Sectioning Scripts

If you add at least four ---- dashes after a comment, it creates a section in your script. You can then navigate to different sections using the dropdown menu at the bottom of the **source** pane.

# Section 1 ----  
# Section 2 ----  
# Section 3 ----



Sections Example

### Data Object Names

When you seek help online for R, you will notice a lot of articles use example data object names as d, foo, bar, or mydata. I do not recommend this. Instead, always try to give your data objects descriptive names, as we did above. This way, you can always figure out what your data objects refer to.

### Shortcuts

RStudio has created a lot of different keyboard shortcuts. I highly recommend [visiting this link to explore them](https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts0).

To see some of them demonstrated, you can [watch this video](https://www.youtube.com/watch?v=D89TnL7NxjQ).

# Data Preparation and Cleaning in R

This chapter will introduce you to viewing your data, summarizing your data, and cleaning your data following recommendations from the [*Brief Introduction to the 12 Steps of Data Cleaning*](https://www.slideshare.net/jamorrow/brief-introduction-to-the-12-steps-of-evaluagio) (Morrow, 2013).

## Introduction to the Tidyverse

The Tidyverse is a set of packages that make R easier to use. All the packages work together and share an underlying grammar and philosophy. That’s right - philosophy. The Tidyverse operates on the assumption that data should be “tidy”.

According to Hadley Wickham,Chief Scientist at RStudio and one of the creators of the Tidyverse:

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In tidy data: 1. Each variable forms a column. 2. Each observation forms a row. 3. Each type of observational unit forms a table.

You can read more in [*Tidy Data* from the Journal of Statistic Software](https://www.jstatsoft.org/article/view/v059i10/v59i10.pdf).

The Tidyverse not only helps keep data “tidy,” but makes programming easier compared to the base R syntax. Compare:

### Extracting a variable

**Tidyverse**

select(iris, Species, Petal.Width) # by name  
select(iris, 5, 4) # by column index

**Base R**

iris[, c("Species", "Petal.Width")] # by name  
iris[, c(5, 4)] # by column index

### Make new variables (columns)

**Tidyverse**

iris %>%  
mutate(Petal.Ratio = Petal.Length/Petal.Width,  
 Sepal.Ratio = Sepal.Length/Sepal.Width)

**Base R**

iris$Petal.Ratio <- iris$Petal.Length/iris$Petal.Width  
iris$Sepal.Ratio <- iris$Sepal.Length/iris$Sepal.Width

(see more examples [here](https://tavareshugo.github.io/data_carpentry_extras/base-r_tidyverse_equivalents/base-r_tidyverse_equivalents.html))

Most of this handbook will use Tidyverse functions, as they are clearly easier to work with.

## Viewing Your Data

First, let’s load some data. We will be working with [R Studio’s “Learning R Survey”](https://github.com/rstudio/learning-r-survey).

We will load the data from the URL:

raw\_data <- read.csv("https://github.com/rstudio/learning-r-survey/blob/master/2019/data/2019%20English%20R%20Community%20Survey%20Responses%20.csv?raw=true", fileEncoding = "UTF-8")

We also added fileEncoding = "UTF-8" to make sure any text was formatted correctly. UTF-8 is a common encoding format.

### The RStudio Environment

There are several ways to view this data using the R Studio environment:

1. Type the name in the **console**.

* This is not useful for a large data frame.

1. Double-click the data object in the **environment**.

* This will open up the dataframe in the **source** pane, and you can easily browse the data.
* You can do the same thing using the view() function, i.e. view(raw\_data).

### Structure of Data - str()

You can use str() (from base R) to view how your data is structured. That is, whether variables are numeric, characters, factors. If they are factors, you can also see what their levels are. I recommend using str() in the **console** not your script.

This is a very wide dataframe (52 variables/columns), so I will use the following code just to view the structure of the first 5 columns.

str(raw\_data[1:5])

## 'data.frame': 1838 obs. of 5 variables:  
## $ Timestamp : Factor w/ 1827 levels "1/1/2020 0:02:42",..: 235 236 237 238 178 179 180 181 182 183 ...  
## $ How.would.you.rate.your.level.of.experience.using.R. : Factor w/ 5 levels "","Beginner",..: 3 2 4 4 4 3 4 3 4 4 ...  
## $ Compared.with.other.technical.topics.you.ve.learned.in.school.and.on.the.job..on.a.scale.of.1.to.5..how.difficult.do.you.expect.learning.R.to.be.: int NA NA NA NA NA NA NA NA NA NA ...  
## $ From.what.you.know.about.R..how.long.do.you.expect.that.it.will.take.for.you.to.learn.enough.to.use.R.productively. : Factor w/ 4 levels "","Months","Weeks",..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ How.do.you.think.you.would.go.about.the.process.of.learning.R. : Factor w/ 4 levels "","By myself using a book or online documentation that is not part of a formal course",..: 1 1 1 1 1 1 1 1 1 1 ...

Here is what we did:

* str(raw\_data) tells R you want to look at the structure of the data object.
* [1:5] tells R you just want to look at columns 1 to 5

Here is what the output means:

1. “Timestamp” contains dates but is being read as a **factor**. Factors are like categories. Multiple data points can share a factor.

* This can be converted into a date later using the lubridate package

1. “How.would.you.rate.your.level.of.experience.using.R.” is also a factor. It has 5 levels or categories. “Beginner” is one level. The numbers after the list correspond to the levels of the factors. For example, “Beginner” might be 1 or 2 or 3. We will investigate this more later.
2. “Compared.with.other….” has the class **integer** (int). This is a number. The first 10 numbers are all NA (missing).

If you just want to examine a single variable, you can use the **$** operator:

str(raw\_data$In.what.country.do.you.currently.reside.)

## Factor w/ 93 levels "","Afghanistan",..: 90 57 90 90 90 90 90 90 62 90 ...

In the output, we can see:

* Factor - This is a factor (a categorical variable)
* w/ 93 levels - It has 93 categories (countries)
* "", "Afghanistan",..: - The first category is "" (i.e. missing), the second category is Afghanistan…and so on, in alphabetical order
* 90 57 90 90 90 - These numbers represent the numbers (randomly) assigned to each category.

If you just want to see the class of a variable, you can use the class function:

class(raw\_data$What.year.did.you.first.start.learning.R.)

## [1] "integer"

### Variable/Column Names - names()

The names() (from base R) function is useful if you just want a list of the variable names.

names(raw\_data[1:5])

## [1] "Timestamp"   
## [2] "How.would.you.rate.your.level.of.experience.using.R."   
## [3] "Compared.with.other.technical.topics.you.ve.learned.in.school.and.on.the.job..on.a.scale.of.1.to.5..how.difficult.do.you.expect.learning.R.to.be."  
## [4] "From.what.you.know.about.R..how.long.do.you.expect.that.it.will.take.for.you.to.learn.enough.to.use.R.productively."   
## [5] "How.do.you.think.you.would.go.about.the.process.of.learning.R."

### First n Rows - head()

The head() (from base R’s utils package) function gives you a slightly more detailed look at your data. It will give you the column name and the first 10 data points (by default):

head(raw\_data[1:5])

## Timestamp  
## 1 12/13/2019 9:50:30  
## 2 12/13/2019 9:50:38  
## 3 12/13/2019 9:51:19  
## 4 12/13/2019 9:53:51  
## 5 12/13/2019 10:01:03  
## 6 12/13/2019 10:04:42  
## How.would.you.rate.your.level.of.experience.using.R.  
## 1 Expert  
## 2 Beginner  
## 3 Intermediate  
## 4 Intermediate  
## 5 Intermediate  
## 6 Expert  
## Compared.with.other.technical.topics.you.ve.learned.in.school.and.on.the.job..on.a.scale.of.1.to.5..how.difficult.do.you.expect.learning.R.to.be.  
## 1 NA  
## 2 NA  
## 3 NA  
## 4 NA  
## 5 NA  
## 6 NA  
## From.what.you.know.about.R..how.long.do.you.expect.that.it.will.take.for.you.to.learn.enough.to.use.R.productively.  
## 1   
## 2   
## 3   
## 4   
## 5   
## 6   
## How.do.you.think.you.would.go.about.the.process.of.learning.R.  
## 1   
## 2   
## 3   
## 4   
## 5   
## 6

You can also request more or less data points per column:

head(raw\_data[1:5], n = 3)

## Timestamp  
## 1 12/13/2019 9:50:30  
## 2 12/13/2019 9:50:38  
## 3 12/13/2019 9:51:19  
## How.would.you.rate.your.level.of.experience.using.R.  
## 1 Expert  
## 2 Beginner  
## 3 Intermediate  
## Compared.with.other.technical.topics.you.ve.learned.in.school.and.on.the.job..on.a.scale.of.1.to.5..how.difficult.do.you.expect.learning.R.to.be.  
## 1 NA  
## 2 NA  
## 3 NA  
## From.what.you.know.about.R..how.long.do.you.expect.that.it.will.take.for.you.to.learn.enough.to.use.R.productively.  
## 1   
## 2   
## 3   
## How.do.you.think.you.would.go.about.the.process.of.learning.R.  
## 1   
## 2   
## 3

Or you can request data from a single variable:

head(raw\_data$How.likely.are.you.to.recommend.R.to.a.colleague..friend..or.family.member., n = 20)

## [1] 10 10 10 9 10 10 9 10 10 10 10 10 10 10 10 10 9 9 9  
## [20] 9

head() looks at the head or top of the data. tail() looks at the bottom:

tail(raw\_data$How.likely.are.you.to.recommend.R.to.a.colleague..friend..or.family.member., n = 20)

## [1] 9 10 10 6 10 6 9 10 10 10 8 10 10 10 10 10 8 NA 10  
## [20] 9

## Renaming Variables

If you have noticed, the variable names for the R survey are very long. This is not convenient to work with.

R Studio provides us with a useful [code book](https://github.com/rstudio/learning-r-survey/blob/master/2019/Learning%20R%20Internet%20Survey%20-%20Question%20Names.tsv) to help us fix this problem.

A common way of renaming columns is to use the rename() function. This function is part of one of the most useful packages you will download and install: tidyverse.

To begin, install tidyverse through the **console**: install.packages("tidyverse"). Then load it at the top of your script: library(tidyverse). Don’t forget to run this line of code!

The first column’s name is already simple: *Timestamp*. So, let’s rename the second column.

First, let’s get the column name (run this in the **console**:

names(raw\_data[2])

## [1] "How.would.you.rate.your.level.of.experience.using.R."

Now, we can use the rename() function to change this to “Qr\_experience”, as listed in the codebook. We will assign this to a new data object for comparison purposes. Put this in your script.

renamed <- raw\_data %>%  
 rename("Qr\_experience" = "How.would.you.rate.your.level.of.experience.using.R.")

Let’s see the new name:

# new name  
names(renamed[1:2])

## [1] "Timestamp" "Qr\_experience"

So, what did we do?

* renamed <- creates a new data object
* raw\_data says which data object we are working with
* %>% is the pipe operator. This means something like “and”
* rename("new\_name" = "old name") sets the name name and the old name. Note the quotation marks

We can read this line as a sentence: Create a data object, “renamed”. Use the “raw\_data” object and rename “new\_name” from “old\_name”.

## Renaming Multiple Variables

We can also rename multiple columns the same way:

renamed <- raw\_data %>%  
 rename("Qr\_experience" = "How.would.you.rate.your.level.of.experience.using.R.",  
 "Qr\_difficulty\_experienced" = "Compared.with.other.technical.topics.you.ve.learned.in.school.and.on.the.job..on.a.scale.of.1.to.5..how.difficult.do.you.expect.learning.R.to.be.")

Let’s check:

# new name  
names(renamed[2:3])

## [1] "Qr\_experience" "Qr\_difficulty\_experienced"

You can also use names() to change the names of the data frame like this. You can use c to write a list of names and then apply it to the names of your data frame. The names are applied in the order they are written and any misisng names are set to NA:

names(raw\_data) <- c("Q1", "Q2", "Q3")

Let’s check:

# new name  
names(raw\_data[1:2])

## [1] "Q1" "Q2"

However, raw\_data has 52 variables! This would take a *long* time to do. Instead of writing 52+ lines of code, we can use the names() function to set the names of our data frame (raw\_data) equal to the names of *another* data frame.

The R Studio survey provides us with a .tsv file of proper variable names for the columns. Let’s load that first:

qnames <- read\_tsv("https://raw.githubusercontent.com/rstudio/learning-r-survey/master/2019/data/2019-english-question-names-only.tsv")

Let’s use the method above to make a copy of “raw\_data” and then rename it:

# make a copy of the data frame  
renamed <- raw\_data  
  
# rename the columns based on qnames  
names(renamed) <- names(qnames)

Let’s check

names(renamed[1:4])

## [1] "Qtime" "Qr\_experience"   
## [3] "Qr\_difficulty" "Qr\_length\_to\_success"

Note that renaming follows the order of the “qnames” data frame.

## Cleaning Names with janitor

Let’s make sure all the names are lowercase. This will make typing them in later analyses easier, as you don’t ever need to remember what is capital and what is not. Install the janitor package:

**Console**: install.packages("janitor").

Run the following code:

renamed <- renamed %>%   
 janitor::clean\_names()

Let’s check:

names(renamed[1:10])

## [1] "qtime" "qr\_experience"   
## [3] "qr\_difficulty" "qr\_length\_to\_success"   
## [5] "qhow\_to\_learn\_r" "qreason\_to\_learn"   
## [7] "qr\_use" "qtools"   
## [9] "qobstacles\_to\_starting" "qr\_year"

The clean\_names() function will do a number of things:

* convert to lower case
* change spaces into underscores
* change % signs to “percentage”

There are many cool options, so please check ?janitor::clean\_names.

### Summary Stats - describe()

The psych package offers a very useful funcion called describe(). This function gives you summary statistics about your data. In particular, it will give you:

* item name
* item number
* number of valid cases
* mean
* standard deviation
* trimmed mean (with trim defaulting to .1)
* median (standard or interpolated
* mad: median absolute deviation (from the median).
* minimum
* maximum
* skew
* kurtosis
* standard error

Load the psych package at the top of your script using library(psych). Then, you can execute this command.

The following command describes the third variable (according to our str() function, that variable is numeric):

describe(renamed[1:5])

## vars n mean sd median trimmed  
## qtime\* 1 1838 913.24 526.89 912.5 913.05  
## qr\_experience\* 2 1838 3.41 0.78 4.0 3.53  
## qr\_difficulty 3 8 3.50 0.53 3.5 3.50  
## qr\_length\_to\_success\* 4 1838 1.01 0.12 1.0 1.00  
## qhow\_to\_learn\_r\* 5 1838 1.01 0.13 1.0 1.00  
## mad min max range skew kurtosis  
## qtime\* 676.07 1 1827 1826 0.00 -1.20  
## qr\_experience\* 0.00 1 5 4 -1.02 0.21  
## qr\_difficulty 0.74 3 4 1 0.00 -2.23  
## qr\_length\_to\_success\* 0.00 1 4 3 20.39 456.26  
## qhow\_to\_learn\_r\* 0.00 1 4 3 19.22 399.20  
## se  
## qtime\* 12.29  
## qr\_experience\* 0.02  
## qr\_difficulty 0.19  
## qr\_length\_to\_success\* 0.00  
## qhow\_to\_learn\_r\* 0.00

See more options using ?describe in the **console**.

### Summary Stats - describeBy()

The psych package also provides describeBy (note the capital “B”) to break an integer variable down by some factor or category. For example, the survey provides “qyear\_born”, which we can use to calculate age. Then, we can see summary stats of age by experience) from “qr\_experience”.

#create an age variable  
renamed$age <- 2020-renamed$qyear\_born  
  
#Describe experience by age  
describeBy(renamed$age, group=renamed$qr\_experience, mat=TRUE)

## Warning in min(x, na.rm = na.rm): no non-missing arguments to  
## min; returning Inf

## Warning in max(x, na.rm = na.rm): no non-missing arguments to  
## max; returning -Inf

## item group1 vars n mean sd median  
## X11 1 1 0 NaN NA NA  
## X12 2 Beginner 1 224 36.26786 12.968732 34  
## X13 3 Expert 1 506 36.62648 9.625914 35  
## X14 4 Intermediate 1 993 36.62034 10.948554 35  
## X15 5 None 1 8 44.12500 13.798939 42  
## trimmed mad min max range skew kurtosis  
## X11 NaN NA Inf -Inf -Inf NA NA  
## X12 34.76111 11.8608 19 122 103 1.9371531 8.039461  
## X13 35.35468 7.4130 22 123 101 2.3352574 13.003583  
## X14 35.29560 8.8956 20 142 122 2.1278872 11.839006  
## X15 44.12500 12.6021 20 65 45 -0.1578919 -1.082360  
## se  
## X11 NA  
## X12 0.8665099  
## X13 0.4279241  
## X14 0.3474419  
## X15 4.8786616

What did we do?

* renamed$age <- - Assign whatever is after the <- to a new column, “age” in “renamed”
* 2020-renamed$qyear\_born - Subtract renamed$qyear\_born from the current year, 2020.
* describeBy(age$age, - The numeric variable
* group=age$qr\_experience - The grouping variable
* mat=TRUE) - Show a matrix in the output instead of a more complicated list

### Summary Stats - summary()

The summary() function will give similar statistics as describe(), though with less detail. For **factors**, it will produce a count per level (category). For **integers** it will produce min, max, quartiles, median, mean, and number of missing data.

The following summary() function calls columns 2 (a factor) and 3 (an integer)

summary(raw\_data[2:3])

## Q2 Q3   
## : 31 Min. :3.0   
## Beginner : 233 1st Qu.:3.0   
## Expert : 529 Median :3.5   
## Intermediate:1037 Mean :3.5   
## None : 8 3rd Qu.:4.0   
## Max. :4.0   
## NA's :1830

### Summary Stats - skim()

skim is another function that produces summary statistics. It will also produce a small histogram to show the distribution of your data (for numeric/integers).

First, install the skimr package in the **console** using:

install.packages("skimr")

Add library(skimr) to the top of your script after installation.

Then, we can use the skim() function like this:

skim(raw\_data[2:3])

## Warning in sorted\_count(x): Variable contains value(s) of ""  
## that have been converted to "empty".

Data summary

|  |  |
| --- | --- |
| Name | raw\_data[2:3] |
| Number of rows | 1838 |
| Number of columns | 2 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| factor | 1 |
| numeric | 1 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: factor**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | ordered | n\_unique | top\_counts |
| Q2 | 0 | 1 | FALSE | 5 | Int: 1037, Exp: 529, Beg: 233, emp: 31 |

**Variable type: numeric**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| Q3 | 1830 | 0 | 3.5 | 0.53 | 3 | 3 | 3.5 | 4 | 4 | ▇▁▁▁▇ |

### Crosstabs - table()

Base R provides the table() function to allow you to view a crosstabulation count of one variable by one or more other variables

We can use the code below to get a list of experience levels by country:

table(renamed$qcountry, renamed$qr\_experience)

##   
## Beginner Expert  
## 31 18 33  
## Afghanistan 0 0 0  
## Albania 0 0 0  
## Algeria 0 7 5  
## Andorra 0 0 1  
## Argentina 0 1 3  
## Armenia 0 0 0  
## Australia 0 5 20  
## Austria 0 0 5  
## Bangladesh 0 0 0  
## Belarus 0 0 2  
## Belgium 0 2 3  
## Brazil 0 9 20  
## Bulgaria 0 0 0  
## Cameroon 0 1 0  
## Canada 0 11 25  
## Chile 0 0 1  
## China 0 0 1  
## Colombia 0 1 4  
## Congo, Democratic Republic of the 0 0 0  
## Costa Rica 0 0 1  
## Cote d'Ivoire 0 0 0  
## Croatia 0 0 0  
## Cyprus 0 0 0  
## Czech Republic 0 6 9  
## Denmark 0 1 10  
## Dominican Republic 0 0 0  
## Ecuador 0 1 0  
## Egypt 0 0 1  
## El Salvador 0 0 0  
## Estonia 0 0 0  
## Finland 0 1 1  
## France 0 4 18  
## Germany 0 8 21  
## Greece 0 1 1  
## Hong Kong 0 0 0  
## Hungary 0 0 0  
## Iceland 0 0 1  
## India 0 11 7  
## Indonesia 0 0 0  
## Iran 0 0 2  
## Ireland 0 4 3  
## Israel 0 0 1  
## Italy 0 1 6  
## Japan 0 2 1  
## Kenya 0 1 1  
## Korea, South 0 1 1  
## Latvia 0 0 2  
## Lithuania 0 1 1  
## Luxembourg 0 1 1  
## Macedonia 0 0 0  
## Malawi 0 1 0  
## Malaysia 0 0 0  
## Malta 0 0 0  
## Mexico 0 3 4  
## Nepal 0 1 0  
## Netherlands 0 3 10  
## New Zealand 0 1 4  
## Nigeria 0 1 0  
## Norway 0 3 3  
## Pakistan 0 0 1  
## Peru 0 0 1  
## Philippines 0 0 0  
## Poland 0 2 4  
## Portugal 0 3 0  
## Puerto Rico 0 0 3  
## Reunion 0 0 1  
## Russia 0 10 8  
## Saudi Arabia 0 0 0  
## Senegal 0 0 1  
## Serbia and Montenegro 0 2 0  
## Singapore 0 2 1  
## Slovakia 0 1 1  
## Slovenia 0 0 1  
## South Africa 0 2 3  
## Spain 0 3 7  
## Sudan 0 0 0  
## Suriname 0 0 0  
## Sweden 0 2 5  
## Switzerland 0 0 10  
## Taiwan 0 0 2  
## Tanzania 0 0 0  
## Thailand 0 0 2  
## Tunisia 0 0 0  
## Turkey 0 1 1  
## Uganda 0 0 1  
## Ukraine 0 0 2  
## United Arab Emirates 0 0 0  
## United Kingdom 0 24 33  
## United States of America 0 69 203  
## Uruguay 0 0 1  
## Venezuela 0 0 2  
## Vietnam 0 0 2  
##   
## Intermediate None  
## 66 0  
## Afghanistan 1 0  
## Albania 1 0  
## Algeria 13 1  
## Andorra 1 0  
## Argentina 2 0  
## Armenia 1 0  
## Australia 26 0  
## Austria 5 0  
## Bangladesh 1 0  
## Belarus 0 0  
## Belgium 7 0  
## Brazil 18 0  
## Bulgaria 1 0  
## Cameroon 0 0  
## Canada 52 0  
## Chile 2 0  
## China 3 0  
## Colombia 6 1  
## Congo, Democratic Republic of the 1 0  
## Costa Rica 1 0  
## Cote d'Ivoire 1 0  
## Croatia 3 0  
## Cyprus 1 0  
## Czech Republic 13 0  
## Denmark 13 0  
## Dominican Republic 2 0  
## Ecuador 1 0  
## Egypt 1 0  
## El Salvador 1 0  
## Estonia 1 0  
## Finland 6 0  
## France 31 0  
## Germany 55 0  
## Greece 0 0  
## Hong Kong 1 0  
## Hungary 8 0  
## Iceland 1 0  
## India 16 0  
## Indonesia 2 0  
## Iran 1 0  
## Ireland 9 0  
## Israel 2 0  
## Italy 6 0  
## Japan 3 0  
## Kenya 6 0  
## Korea, South 0 0  
## Latvia 1 0  
## Lithuania 2 0  
## Luxembourg 1 0  
## Macedonia 1 0  
## Malawi 1 0  
## Malaysia 6 0  
## Malta 1 0  
## Mexico 5 0  
## Nepal 1 0  
## Netherlands 18 0  
## New Zealand 6 0  
## Nigeria 2 0  
## Norway 13 0  
## Pakistan 0 0  
## Peru 3 0  
## Philippines 2 0  
## Poland 6 0  
## Portugal 2 0  
## Puerto Rico 0 0  
## Reunion 0 0  
## Russia 23 1  
## Saudi Arabia 1 0  
## Senegal 0 0  
## Serbia and Montenegro 0 0  
## Singapore 3 0  
## Slovakia 0 0  
## Slovenia 1 0  
## South Africa 9 0  
## Spain 6 0  
## Sudan 1 0  
## Suriname 1 0  
## Sweden 11 0  
## Switzerland 11 0  
## Taiwan 0 0  
## Tanzania 2 0  
## Thailand 0 0  
## Tunisia 2 0  
## Turkey 6 0  
## Uganda 0 0  
## Ukraine 1 0  
## United Arab Emirates 1 0  
## United Kingdom 80 1  
## United States of America 412 3  
## Uruguay 3 0  
## Venezuela 0 1  
## Vietnam 0 0

## Keeping and Dropping Variables

We can use select() from tidyverse’s dplyr package to keep specific variables or drop other variables.

**Keeping** variables:

keep <- renamed %>% select(qr\_experience, qr\_year)

To **drop** variables, use the *minus* sign.

dropped <- renamed %>% select(-qr\_experience, -qr\_year)

For multiple variables, you can use commas for single columns and colons for a range of columns:

keep\_multi <- renamed %>%  
 select(qr\_experience, -qr\_year, qr\_difficulty:qobstacles\_to\_starting)

Note that select also places variables in the order specificed.

Alternatively, you can do the same in base R with subset:

keep <- subset(renamed, select = qr\_experience)  
dropped <- subset(renamed, select = -qr\_experience)

select has more advanced functions such as select\_if and select\_at. See ?select or ?select\_if for more information.

## Keeping and Dropping Rows

Whereas select() works on columns, slice() works on rows:

* slice(1:100) keeps rows 1 to 100
* slice(-1:-100) removes rows 1 to 100
* slice(1) keeps just row 1
* slice(-1) removes just row 1

## More Frequencies and Descriptives

You can use any of the following functions to get frequencies of your data:

* summary()
* describe() from the psych package
* skim() from the skimr package

Let’s use the jmv package the get some frequencies:

**Console**: install.packages("jmv")

We will use the descriptives() function. This will provide nice output. We will use the parameter freq=TRUE to get frequencies for factors.

library("jmv")  
  
renamed %>%  
 select(qr\_experience, qr\_year) %>%  
 descriptives(freq = TRUE)

##   
## DESCRIPTIVES  
##   
## Descriptives   
## ---------------------------------------   
## qr\_experience qr\_year   
## ---------------------------------------   
## N 1838 1676   
## Missing 0 162   
## Mean 2008   
## Median 2015   
## Minimum 2   
## Maximum 2019   
## ---------------------------------------   
##   
##   
## FREQUENCIES  
##   
## Frequencies of qr\_experience   
## --------------------------------------------------------   
## Levels Counts % of Total Cumulative %   
## --------------------------------------------------------   
## 0.0 0.0   
## Beginner 233 12.7 0.0   
## Expert 529 28.8 0.0   
## Intermediate 1037 56.4 0.0   
## None 8 0.4 0.0   
## --------------------------------------------------------

## Spotting Coding Mistakes

You may have noticed that the descriptives for “qr\_year” had a minimum of 2. The question reads “What year did you first start learning R?”. This should only contain year data. By running frequencies, you can spot such errors. Let’s take a closer look.

Let’s start by viewing all the data in this question. Since it is year data, let’s arrange it from smallest to greatest and look at the first 10 observations.

renamed %>%  
 select(qr\_year) %>%  
 arrange(qr\_year) %>%  
 head(n = 10)

## qr\_year  
## 1 2  
## 2 6  
## 3 13  
## 4 18  
## 5 207  
## 6 1977  
## 7 1985  
## 8 1989  
## 9 1989  
## 10 1990

Let’s review what we did in the code:

* renamed %>% tells R to use “renamed” and…
* select(qr\_year) %>% select the “qr\_year” variable and…
* arrange(qr\_year) %>% arrange the data based on the “qr\_year” variable and…
* head(n = 10) get the first 10 responses.

From the results, we saw that the first 5 responses are invalid years.

## Modifying Data - mutate()

We have a variable, “qr\_year” with five invalid cases. They should be years, but are some other numbers. Let’s change those to mising observations using mutate(). mutate() allows us to change and create variables (literally mutate the data).

#run this to change the variable  
renamed <- renamed %>%  
 mutate(qr\_year2 = ifelse(qr\_year < 1977, NA, qr\_year))  
  
#run this to check  
renamed %>%  
 select(qr\_year, qr\_year2) %>%  
 arrange(qr\_year) %>%  
 head(n=10)

## qr\_year qr\_year2  
## 1 2 NA  
## 2 6 NA  
## 3 13 NA  
## 4 18 NA  
## 5 207 NA  
## 6 1977 1977  
## 7 1985 1985  
## 8 1989 1989  
## 9 1989 1989  
## 10 1990 1990

What we did:

* renamed <- means we will save our fixed data to the renamed data object. Essentially, we are overwriting the data object that already exists.
* renamed %>% tells R to use “renamed” and…
* mutate(qr\_year2 create a new variable, “qr\_year2”
* = which is equal to the following command
* ifelse(qr\_year < 1977, NA, qr\_year)) if qr\_year is less than 1977, write NA, otherwise (else), write the year from qr\_year

ifelse is a conditional statement that says: ifelse(if this condition is true, then do this, otherwise do this).

## Reordering Categories - factor()

Recode allows you to take a numeric value in a dataset and assign it a label, for example make 1 = “apple”, or the opposite, for example, make “apple” = 1.

Many of the factors (categories) in our R survey dataset have already been coded to a numeric value, automatically. However, this does not mean the mapping of the number to the factor is correct.

Let’s run this code in the **console** to check. Since we are just checking, we don’t really want to include this in our script, so we can do basic exploration in the console.

renamed %>% count(qr\_experience)

## # A tibble: 5 x 2  
## qr\_experience n  
## <fct> <int>  
## 1 "" 31  
## 2 "Beginner" 233  
## 3 "Expert" 529  
## 4 "Intermediate" 1037  
## 5 "None" 8

You will see that 1 = "“, which is missing data, 2 =”Beginner“, 3 =”Expert“, 4 =”Intermediate" and 5 = “None”. If we were doing a statistical analysis without modifying our data, our results would be severely inaccurate.

Let’s recode this data. We will just do one variable as an example.

recoded <- renamed %>%   
 select(qr\_experience) %>%  
 mutate(qr\_experience2 = factor(qr\_experience,  
 levels=c("None","Beginner", "Intermediate", "Expert", NA ))  
 )

Let’s check if we are correct. You can run this in the **console**:

recoded %>% count(qr\_experience2)

## Warning: Factor `qr\_experience2` contains implicit NA, consider  
## using `forcats::fct\_explicit\_na`

## # A tibble: 5 x 2  
## qr\_experience2 n  
## <fct> <int>  
## 1 None 8  
## 2 Beginner 233  
## 3 Intermediate 1037  
## 4 Expert 529  
## 5 <NA> 31

Now our factors are in the correct order and if we wanted to do some statistical test, it would be much more accurate.

What did we do?

* recoded <- creates a new data object. We could have overwritten our previous object, but this is to show you the result of recoding.
* renamed %>% tells R to use the “renamed” data object and…
* select(qr\_experience) %>% means we are only going to keep this variable in our data (to make demonstration easier), and
* mutate( means we are going to make a change
* qr\_experience2 = we will create a new variable “qr\_experience2” equal to the next command
  + Note: we could have done mutate(qr\_experience) and overwritten our column, but if we create a new variable, we can compare them side by side (for demonstration purposes)
* factor(qr\_experience, make a factor from “qr\_experience”
* levels=c("None","Beginner", "Intermediate", "Expert", NA )) set the level names and orders to these.
  + Note that NA refers to missing data and is **not** in quotes
* ) this final parentheses closes the mutate function. If you place your cursor after it, the corresponding opening parentheses will be highlighted. This is an easy way to check that you have closed all functions, useful for debugging problems. )

## Clearing Whitespace in Text - str\_trim()

On the Learning R survey, there was a question about what industry people were in. This resulted in 135 different responses. However, many of these can be combined to make several large groups. First, let’s view the jobs.

renamed %>%  
 count(qindustry)

## # A tibble: 135 x 2  
## qindustry n  
## <fct> <int>  
## 1 "" 44  
## 2 "Academia" 2  
## 3 "Accommodation and Food Services" 12  
## 4 "Advertising" 1  
## 5 "Aerospace" 1  
## 6 "Aerospace " 1  
## 7 "Agriculture" 4  
## 8 "Agriculture " 1  
## 9 "Agriculture and animal science" 1  
## 10 "Agrifood " 1  
## # ... with 125 more rows

The count() function counts *unique* values. However, you will notice that many are repeated. That is likely due to random whitespace in or around the text entries. We can use several functions str\_trim() from tidyverse’s stringr package to fix this.

stringr is loaded when the tidyverse is loaded, so we don’t need to load another library.

str\_trim() removes whitespace from the start and end of a string. Let’s also use make sure everything is lowercase using either base R’s tolower() or stringr’s str\_to\_lower(). They both do the same thing. You can also use str\_squish to remove **all** white space within a string.

renamed <- renamed %>%  
 mutate(qindustry = str\_squish(tolower(qindustry)))

Let’s check. You can type this in the console. We will select only “qindustry” and “qindustry2” to compare them:

renamed %>%  
 count(qindustry)

## # A tibble: 125 x 2  
## qindustry n  
## <chr> <int>  
## 1 "" 44  
## 2 "academia" 2  
## 3 "accommodation and food services" 12  
## 4 "advertising" 1  
## 5 "aerospace" 2  
## 6 "agriculture" 5  
## 7 "agriculture and animal science" 1  
## 8 "agrifood" 1  
## 9 "analytics consulting company" 1  
## 10 "any" 1  
## # ... with 115 more rows

Hmm. That reduced the number of unique values to 125. That is not ideal, but we have a few more tools we can use to fix this.

First, what did we do?

* renamed <- - saves the following commands to the “renamed” data object
* renamed %>% - means to use the renamed data object “and then”
* mutate( - change the data
* qindustry = - make the column “qindustry” equal to the following
* str\_squish( - trim the whitespace in the following string
* tolower(qindustry))) - make the string lowercase and close all parentheses

**A Note about Parentheses ( )**

In R, it can be easy to become lost with your open and closing parentheses. RStudio provides parenthesese highlighting. Place your cursor on the *right* side of any parentheses (open or close) and it will highlight to corresponding one. This is a quick way to debug a common problem.

## Combining Categories - case\_when()

Looking through the list of jobs, there are some clear categories that we can combine jobs into. For example, *agriculture*, *agriculture and anaimal science*, and *agrifood* can all be combine into an “agriculture” category. Here is what we will do. We will use three functions to do this: \* mutate() to change or create new variables \* case\_when() as a kind of if/else command \* str\_detect() to select specified text

# combine into "agriculture"  
renamed <- renamed %>%  
 mutate(  
 industry = case\_when(  
 str\_detect(qindustry, "agri") ~ "agriculture",  
 TRUE ~ qindustry  
 )  
 )  
  
# and check  
renamed %>% count(industry)

## # A tibble: 123 x 2  
## industry n  
## <chr> <int>  
## 1 "" 44  
## 2 "academia" 2  
## 3 "accommodation and food services" 12  
## 4 "advertising" 1  
## 5 "aerospace" 2  
## 6 "agriculture" 7  
## 7 "analytics consulting company" 1  
## 8 "any" 1  
## 9 "arts and entertainment" 14  
## 10 "automotive" 1  
## # ... with 113 more rows

We now have 123 rows. Let’s see what we did, and then expand it:

* renamed <- - assign to the “renamed” object
* renamed %>% - use “renamed” and then
* mutate( - change or create a variable
* industry = case\_when( - create the variable “industry” when
* str\_detect(qindustry, "agri") - R detects “agri” in the “qindustry” columns"
* ~ "agriculture", - and rename them “agriculture”
* TRUE ~ qindustry - everything else is named the name they have in “qindustry”

Let’s exand this to categorize as much as possible. To do this efficiently, I will run renamed$industry in the *console* and scroll through it as I create categories

# combine into "agriculture"  
renamed <- renamed %>%  
 mutate(  
 industry = case\_when(  
 str\_detect(qindustry, "agri") ~ "agriculture",  
 str\_detect(qindustry, "health") ~ "health",  
 str\_detect(qindustry, "education|academia|university|research") ~ "education and research",  
 str\_detect(qindustry, "marketing|business|trade|ecommerce") ~ "business",  
 str\_detect(qindustry, "information|analytics|software|cybersecurity|digital|telec") ~ "information technologies",  
 str\_detect(qindustry, "envi|forest|geo|natural|wildlife|sustain") ~ "environment",  
 str\_detect(qindustry, "law|legal") ~ "law",  
 str\_detect(qindustry, "media|journalism") ~ "media",  
 str\_detect(qindustry, "profit|") ~ "non-profit",  
 TRUE ~ "other"  
 )  
 )  
  
# and check  
renamed %>% count(industry)

## # A tibble: 9 x 2  
## industry n  
## <chr> <int>  
## 1 agriculture 7  
## 2 business 133  
## 3 education and research 704  
## 4 environment 33  
## 5 health 184  
## 6 information technologies 196  
## 7 law 4  
## 8 media 12  
## 9 non-profit 565

This code got out data down to 9 categories. The categories are neither perfect nor exhaustive, but they are functional. This is a lesson in working with open-ended data on surveys. A question like this should really have limited options, otherwise, there is a lot to clean!

**A Note About Spacing and Tabs**

You may have noticed that in the code above, mutate() was on its own line and there were two parentheses also on their own line. You may have also noticed mutate() was indented as was str\_detect(). These are not required for the code to function, but are useful to keep the code organized.

It is recommend to start a new function after the %>% pipe operator on a new line. It is also recommended to create a new line after a parentheses( if it will contain multiple or long arguments.

What did we do differently? \* "law|legal" - You will notice we used the | pipe symbol. This stands for “or”. This command means to look for “law” or “legal”. \* "envi|forest|geo|natural|wildlife|sustain" - You will notice we did not use complete words. str\_detect() looks for patterns of string and does not need the complete words unless there are many similar string. + This means “envi” could look for *environment*, *environments*, or *environmental*. \* TRUE ~ "other" - We set everything else to the “other” category.

## Splitting variables with split()

The Learning R survey contained a few check-all question types. Depending on the survey platform, these questions either create a new column for each choice or combine all choices in a single column. The Learning R survey did the latter. For example, the question “What applications do you use R for most? (check all that apply)” for the variable qused\_for looks like this:

head(renamed$qused\_for)

## [1] Statistical analysis, Data transformation, Modeling, Visualization, Machine learning, Text processing  
## [2] Statistical analysis, Data transformation, Visualization   
## [3] Statistical analysis, Data transformation, Visualization   
## [4] Data transformation   
## [5] Statistical analysis, Data transformation, Modeling, Visualization   
## [6] Statistical analysis, Data transformation, Modeling, Visualization, Machine learning, Text processing  
## 370 Levels: Building packages ... Web app development using Shiny

How can we work with this type of data? The first thing we can do is split the data so that choices are separated into columns. There are several functions we can use to do this. Normally, we can do this with separate() from tidyverse’s tidyr package. However, we need to know how many columns to split the data into.

We can use the following code to count how many commas appear in each row and then get the highest value:

renamed$qused\_for %>%  
 str\_count(pattern = ",") %>%   
 max()

## [1] 10

There is at least one row with 10 commas, meaning 11 items were selected. We can now use separate to separate the string.

renamed <- renamed %>%  
 separate(qused\_for, sep = ",",  
 into = paste0("use\_", 1:11))

## Warning: Expected 11 pieces. Missing pieces filled with `NA` in  
## 1837 rows [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,  
## 16, 17, 18, 19, 20, ...].

The original data object had 56 variables. The new data object now has 66. This means we have removed the “qused\_for” variable and replaced it with 11 more.

What did we do?

* renamed <- - assign the new resulting data to “renamed”
* renamed %>% - use renamed and then
* separate(qused\_for, sep = ",", - separate “qused\_for” by using the commas
* into = paste0("use\_", 1:11)) - into 11 columns, with the name “use\_” pasted to each column, from 1 to (:) 11.
  + Note: we could have written into = c("use\_1", "use\_2"...) but that would have taken a LONG time. Using paste0() is much faster.

Working with check-all data can be tricky because it requires a lot of columns. We really need to do more transformation of the data. We will revisit this variable in the section on pivoting data.

## Descriptives

So far, we have looked at functions to explore the data and do basic data manipulation and cleaning. We also need to learn functions that allow us to describe specific elements of the data frame.

### Mean - mean()

This is a simple function to allows you to find the mean of data. You can use it via base R. Before doing so, note that there is NA data, which cannot be calculated in the mean, so we need to use na.rm = TRUE to remove the NAs.

#base R  
mean(renamed$age, na.rm = T)

## [1] 36.61121

You should note that for both, what was returned was unexpected. For the base R sytnax, “NA” was returned. For the tidyverse syntax, an error was returned. Both of these occured because the column has NAs which cannot be calculated in the mean() function. We need to remove the NAs:

**A Note About TRUE and FALSE**

There are many functions that require a logical argument. To save time, you can usually write T or F instead of the whole word.

### Median - median()

You can do the same calculation with the median() function:

median(renamed$age, na.rm = T)

## [1] 35

### Using the summarize() function - summarize()

We can use the tidyverse’s summarize() function to do multiple descriptives in one command. summarize() is similar to mutate() in that it create new columns. However, instead of appending a column to a data frame, it literally summarizes and condenses all data into the number of columns specififed. Here is an example:

renamed %>%  
 drop\_na(age) %>%  
 summarize(avg = mean(age),  
 mid = median(age),  
 n = n())

## avg mid n  
## 1 36.61121 35 1731

What did we do?

* renamed %>% - use “renamed” and then
  + Note: We are not assigning this to anything because we are just exploring the data. There is no need to create a data object.
* drop\_na(age) %>% - rather than write na.rm=T for each function, we will just drop all NAs for “age” first and then
* summarize( - create a summary data frame
* avg = mean(age) - create a column “avg” with the mean of “age”
* mid = median(age) - create a column “mid” with the median of “age”
* n = n() - create a column “n” with the sample size of “age”

### Analyzing Data by Groups - group\_by()

What if we want to view the proportion of ages in the data set, or the ages broken down by group? We can use the group\_by() function:

renamed %>%  
 drop\_na(age) %>%  
 group\_by(age) %>%  
 summarize(n = n()) %>%  
 ungroup() %>%  
 mutate(  
 proportion = prop.table(n)  
 )

## # A tibble: 63 x 3  
## age n proportion  
## <dbl> <int> <dbl>  
## 1 19 1 0.000578  
## 2 20 5 0.00289   
## 3 21 7 0.00404   
## 4 22 23 0.0133   
## 5 23 32 0.0185   
## 6 24 43 0.0248   
## 7 25 57 0.0329   
## 8 26 64 0.0370   
## 9 27 77 0.0445   
## 10 28 73 0.0422   
## # ... with 53 more rows

Here is what we did:

* renamed %>% - use “renamed” and then
* drop\_na(age) %>% - drop the NAs based on “age” and then
* group\_by(age) %>% - group the data by “age” and then
* summarize(n = n()) %>% - create a summary data frame that counts the sample size for each group and then
* ungroup() %>% - ungroup the data and then
* mutate(proportion = prop.table(n)) - create a new column, “proportion” that takes the proportion of each age based on the count of “n”

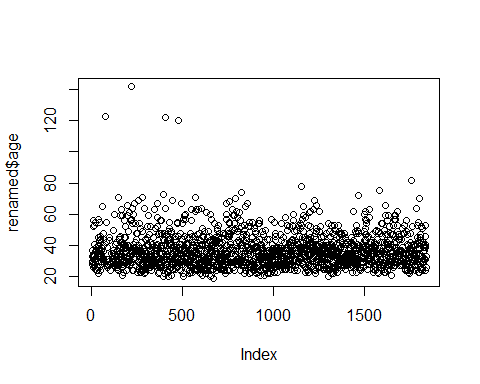
## Spotting Outliers

In this section, we will look at different ways to identify outliers in univariate data (multivariate data will be covered in [Regression Diagnostics]). Spotting outliers is an important part of the data cleaning process. Many statistical tests rely on the assumption of no extreme scores that may bias the tests. In addition, identifying outliers can help you find mistakes or other problems in the data.

### Detecting Numerical Outliers

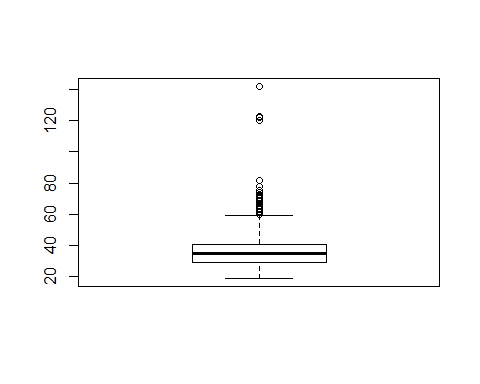
We can use a simple scatterplot for numerical data.

plot(renamed$age)



We could also use a boxplot:

boxplot(renamed$age)



The graphs above shows that there are four possible outliers. These outliers have ages over 100. We can search for them in the data to see what is going on with them:

renamed %>%  
 filter(age > 100) %>%  
 select(age, qyear\_born)

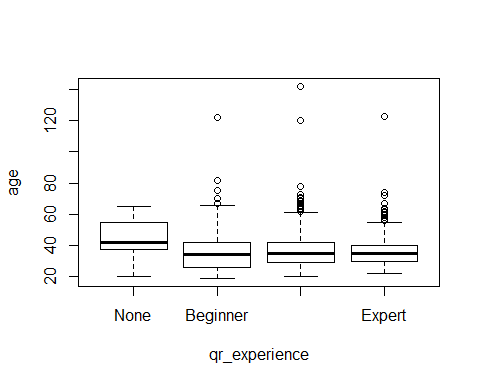
## age qyear\_born  
## 1 123 1897  
## 2 142 1878  
## 3 122 1898  
## 4 120 1900

I highly doubt these are serious responses, so we would need to deal with them in some way (see [Dealing with Outliers](#dealing-with-outliers)).

#### Detecting Categorical Outliers

We can also see if a category contains any numeric outliers based on another variable. For example, if we want to see if there are any age outliers for the experience variable, we can use a simple boxplot:

plot(age ~ qr\_experience, data=renamed)

 The graph above shows the four same extreme outliers. It also shows beginner, intermediate, and expert responses have some higher ages, but they don’t suggest any surprising outliers, especially because each of the three have high ages.

What did we do?

* plot( - make a plot
* age ~ qr\_experience, - plot age by (~) experience
* data=renamed) - from the “renamed” data frame

### Detecting outliers with Z-scores

One way to detect outliers is to standardize values and select values greater than or less than some specific value. For example, Tabachnick and Fidell (2013) suggest any score above 3.29 may be an outlier for continuous, univariate data.

There are many ways to create Z-scores in R. You could do it manually:

renamed$z\_of\_age <- (renamed$age - mean(renamed$age, na.rm=T))/sd(renamed$age, na.rm = T)  
  
# to check  
summary(renamed$z\_of\_age)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## -1.6178 -0.6992 -0.1480 0.0000 0.4032 9.6809 107

But, we could also load a package and use a function to do it quickly. For example, the psycho package has a standardize() function.

# install package if needed  
# install.packages("psycho")  
library(psycho)  
renamed$z\_of\_age <- standardize(renamed$age)  
  
# to check  
summary(renamed$z\_of\_age)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## -1.6178 -0.6992 -0.1480 0.0000 0.4032 9.6809 107

### Dealing with Outliers

#### Removal

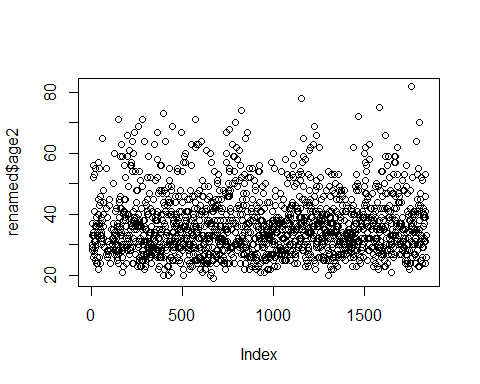
One thing you can do is “delete” your outliers. We will set outliers to NA.

# set ages over 100 to NA  
renamed <- renamed %>%  
 mutate(age2 = ifelse(age > 99, NA, age))  
  
# check if we made the correct change  
renamed %>%  
 filter(qyear\_born < 1901) %>%  
 select(age2, qyear\_born)

## age2 qyear\_born  
## 1 NA 1897  
## 2 NA 1878  
## 3 NA 1898  
## 4 NA 1900

Let’s look at the plot again:

plot(renamed$age2)

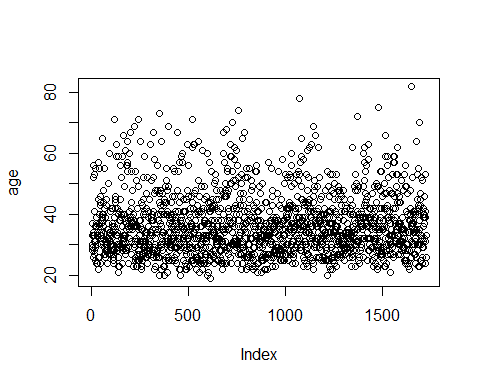


There are no clear outliers for the “age” variable.

#### Filtering

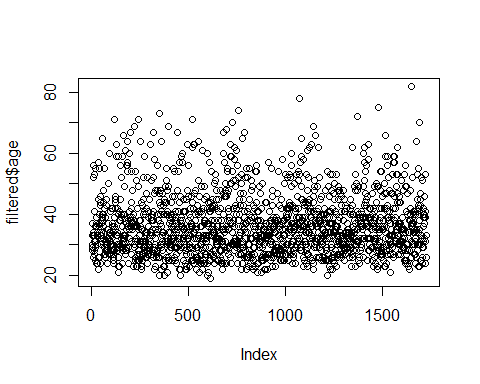
Rather than transforming that data, you could simply use filter() to temporarily remove it for some function. For example:

renamed %>%  
 filter(age < 100) %>%  
 with(plot(age))



In the formula above, with() is a base R function that uses the data from a tidyverse chain (functions with %>%). Using strictly base R would require more steps, including creating a new dataframe:

filtered <- subset(renamed, age < 99)  
plot(filtered$age)



**A Note about “Deleting” Values**

It is never a good idea to delete or overwrite your raw data. Recall that we have a “raw\_data” data object that we have not transformed in any way. All transformations have been done in new data objects and, sometimes, in new variables. This is good practice so that we can assess our changes and revert our changes if necessary.

#### Winsorizing

Winsorizing is a way to replace the most extreme scores with another value. The psych package has a winsor() function that will replace the top and bottom values with the next highest score.

For example, the code below trims the top and bottom 2% of the data set and replaces it with the next highest and lowest values.

library(psych)  
  
winsored <- renamed %>%  
 mutate(winsor = winsor(age, trim=.02, na.rm=T))  
  
# let's check  
winsored %>%  
 select(age, winsor) %>%  
 filter(age < 20 | age > 70)

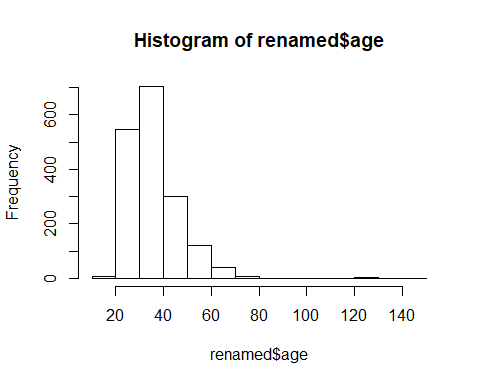
## age winsor  
## 1 123 63.4  
## 2 71 63.4  
## 3 142 63.4  
## 4 71 63.4  
## 5 73 63.4  
## 6 122 63.4  
## 7 120 63.4  
## 8 71 63.4  
## 9 19 22.0  
## 10 74 63.4  
## 11 78 63.4  
## 12 72 63.4  
## 13 75 63.4  
## 14 82 63.4

## Assessing Normality

### Histograms

Normality can be visually assessed with a histogram. A quick histogram can be made using base R’s hist() function:

hist(renamed$age)

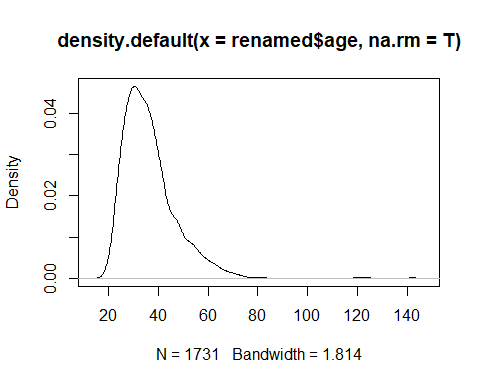


More advanced histograms can be made with ggplot. See [Data Visualization with ggplot]

### Density (Curve) Plots

You can also use a density plot to visually assess normality:

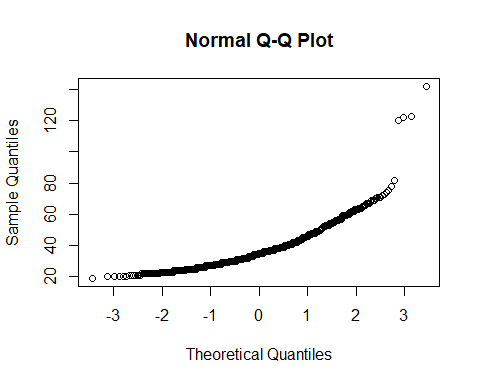
plot(density(renamed$age, na.rm=T))



### QQ Plots

A QQ plot can also be used to assess normality:

qqnorm(renamed$age)



### Skewness and Kurtosis

There are a number of ways to get skewness and kurtosis.

You can get the values alongside other stats with the psych package’s describe()

You can also use skewness() and kurtosis() functions from the moments package:

# install if needed  
# install.packages("moments")  
  
library(moments)  
  
skewness(renamed$age, na.rm = T)

## [1] 2.137981

kurtosis(renamed$age, na.rm = T)

## [1] 14.4558

### Tests of Normality

Univariate tests of normality are also easily done in R. The stats package (included with R) contains both the Kolmogorov-Smirnov (K-S) test and the Shapiro-Wilk’s test. The Shapiro-Wilk’s test is more common and easier to compute in R:

shapiro.test(renamed$age)

##   
## Shapiro-Wilk normality test  
##   
## data: renamed$age  
## W = 0.86577, p-value < 2.2e-16

The above p-value is significant at less than 0.05, signifying non-normality.

See ?shapio.test and ?ks.test for more information.

### Transforming Variables

If you wish to deal with non-normality by transforming variables, you can use these methods ([DataNovia, n.d.](https://www.datanovia.com/en/lessons/transform-data-to-normal-distribution-in-r/" \l "transformation-methods)):

* Square Roots
  + sqrt(x) for positive skew
  + sqrt(max(x+1)-x) for negative skew
* Log
  + log10(x) for positive skew
  + log10(max(x+1)-x) for negative skew
* Inverse
  + 1/x for positive skew
  + 1/max(x+1) for negative skew

## Working with Missing Data

Missing data in R is represented by NA. <https://towardsdatascience.com/data-cleaning-with-r-and-the-tidyverse-detecting-missing-values-ea23c519bc62>

## Identifying missing data

There are a number of ways to identify missing data in R.

### Per Variable

We have already seen a number of ways to view and summarize data [Viewing Data]. There are several other functions that can be useful.

You can use is.na to look at NA values:

mean(is.na(renamed$age))

## [1] 0.05821545

This means that around 6% of the “age” variable is misisng.

Conversely, you can also use complete.cases to see how many cases are *not* missing:

mean(complete.cases(renamed$age))

## [1] 0.9417845

94% of the age variable is *not* missing.

### Entire Data Frame

We can look at an entire data frame by using summarize\_all, part of the summarize family of verbs in dplyr (part of tidyverse).

missing <- renamed %>%  
 summarize\_all(funs(mean(is.na(.))))

This will produce a new data object where each column has 1 row, and that row is the percent missing.

What we did:

* missing <- - assign everything to a new data frame (this step is optional and you could just explore everything in the console)
* renamed %>% - use renamed and
* summarize\_all( - summarize every column
* funs( - use the following function (“funs”) to summarize everything
* mean(is.na(.)))) - take the mean of missing data.
  + the . means to use the entire data frame
  + We need to include it because is.na requires an argument

**A Note on summarize**

summarize is a family of verbs in dplyr that includes summarize, summarize\_if, summarize\_at (for specific columns), and summarize\_all. You may also see it spelled as summarise - this is the same function. Check out ?summarize\_all.

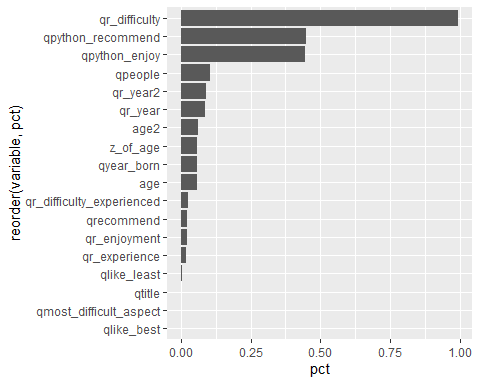
Let’s quickly view what that looks like using glimpse(). This will print the columns and first row veritcally in the console. Because there are 67 columns, we will just look at the first 10.

glimpse(missing[1:10])

## Rows: 1  
## Columns: 10  
## $ qtime <dbl> 0  
## $ qr\_experience <dbl> 0.01686616  
## $ qr\_difficulty <dbl> 0.9956474  
## $ qr\_length\_to\_success <dbl> 0  
## $ qhow\_to\_learn\_r <dbl> 0  
## $ qreason\_to\_learn <dbl> 0  
## $ qr\_use <dbl> 0  
## $ qtools <dbl> 0  
## $ qobstacles\_to\_starting <dbl> 0  
## $ qr\_year <dbl> 0.08813928

You can also get a quick visual by running this code:

missing %>%  
 pivot\_longer(qtime:age2, names\_to = "variable",   
 values\_to = "pct") %>%  
 filter(pct > 0) %>%  
 filter(!str\_detect(variable, "use\_")) %>%  
 ggplot(aes(x=reorder(variable, pct), y=pct)) +  
 coord\_flip() +  
 geom\_col()



This code introduces some new concepts that will be covered later, so the following explanation will not go into detail, but here is what we did:

* missing %>% - use the “missing” data frame
* pivot\_longer(qtime:age2, names\_to = "variable", values\_to = "pct") %>% - our “missing” data frame had 1 row and 67 columns. By using pivot\_longer, we “flipped” our data frame so it had 2 columns (“variable” and “pct”) and 67 rows. This is easier to visualize.
* filter(pct > 0) %>% - we removed any columns with no missing data
* filter(!str\_detect(variable, "use\_")) %>% we removed any columns that started with “use\_” because those are check-all questions.
  + note the ! at the beginning means “is not” or “does not”, so this says to filter any columns that does not start with “use\_to”
* ggplot(aes(x=reorder(variable, pct), y=pct)) + - this line introduces a powerful visualization tool, ggplot, which will be covered later. It says to use “variable on the x-axis (and order it by”pct" so that it is in descending order) and use “pct” on the y-axis.
  + Note that a quirk of ggplot is that it uses + instead of %>% despite being part of the tidyverse
* coord\_flip() + this will flip the visual to make the bar chart horizontal
* geom\_col() - this means we will make a column (bar) chart

## Dropping missing data

You can remove missing data by using drop\_na().

If you use drop\_na() without including columns in the (), it will drop rows that are missing values in any column. This will severely reduce your data frame. If we run the following code, it will drop everything and you will have 0 observations because every row has across has at least 1 missing value somewhere.

missing\_dropped\_all <- renamed %>%  
 drop\_na()  
  
head(missing\_dropped\_all[1:5])

## [1] qtime qr\_experience   
## [3] qr\_difficulty qr\_length\_to\_success  
## [5] qhow\_to\_learn\_r   
## <0 rows> (or 0-length row.names)

This is useful if you have selected (with select() or subset()) a single column or a small group of columns and you do want to drop any missing data, but in general, you may want to specify columns:

missing\_dropped\_some <- renamed %>%  
 drop\_na(age)

This will remove all rows with misisng data in the age variable. Compared “renamed” to “missing\_dropped\_some” and you will see less observations.

## Replacing missing data with 0s

Sometimes missing data is meaningful and it would be useful in including them in various analyses. NA values are typically dropped from analyses, but we can include them by changing them to 0s. To do so, we can use mutate():

zero\_ages <- renamed %>%  
 mutate(age = ifelse(is.na(age), 0, age))

Let’s compare the mean age of our new data frame and our old one:

mean(renamed$age, na.rm=T) #note: you must remove NAs for this to run

## [1] 36.61121

mean(zero\_ages$age)

## [1] 34.47987

## Mean imputation

**If warranted**, we can easily replace missing data with means (or medians) like so:

mean\_imputation <- renamed %>%  
 mutate(age = ifelse(is.na(age), mean(age, na.rm=T), age))  
  
mean(mean\_imputation$age)

## [1] 36.61121

## Multiple imputation

Multiple imputation is currently beyond the scope of this guide. However, you can learn more at any of the following sites:

* [Getting Started with Multiple Imputation in R](https://uvastatlab.github.io/2019/05/01/getting-started-with-multiple-imputation-in-r/)
* [Tutorial on 5 Powerful R Packages used for imputing missing values](https://www.analyticsvidhya.com/blog/2016/03/tutorial-powerful-packages-imputing-missing-values/)
* [How do I perform multiple imputation in R?](https://stats.idre.ucla.edu/r/faq/how-do-i-perform-multiple-imputation-using-predictive-mean-matching-in-r/)
* [Imputing missing data with R; MICE package](https://www.r-bloggers.com/imputing-missing-data-with-r-mice-package/)

# More Data Manipulation in R

## Wide and Long Data

### Long Data

### Wide Data

## Pivoting data from wide to long - pivot\_longer()

## Pivoting data from long to wide - pivot\_wider()

## Joining multiple data objects

# Statistics in R

## T-Tests

## Chi Square

## Regression

## ANOVA

## MANOVA

## ANCOVA

## Logistic Regression

## Factor Analysis

# Resources

<http://flavioazevedo.com/stats-and-r-blog/2016/9/13/learning-r-on-youtube>