CRIM20452 Modelling Criminological Data

Laura Bui & Reka Solymosi

2021-03-09

Contents

Welcome

Course Director: Dr Reka Solymosi

Teaching team: Dr Laura Bui, Carlos Iglesias, Ioana Macoveciuc, and Denny Vlaeva Welcome to Modelling Criminological Data. This is the online tutorial for the class's weekly lab sessions.

The class aims to introduce you to some basic principles and ideas required to understand data analysis. This will help you develop a better grasp of the statistics batted around in the media, by politicians, and by scientists around a variety of issues. In this class, we use the data analysis software `R` to explore and analyse data, interpret statistical results, and to encourage curiosity.

Data is ubiquitous today and affects all aspects of everyday life. Criminal justice agencies are increasingly adopting a 'problem solving' and 'evidence-led' discourse that requires seeking individuals with the skills to perform basic data analytical tasks in order to document patterns of crime problems, factors associated with them, and to evaluate responses to these problems. Also, there is increasing recognition that data analysis skills are helpful in many other professional sectors.

Taking this course meets the eligibility criterion to apply for the University of Manchester Q-Step summer internships.

Site edited and compiled by Laura Bui, based on previous teaching material from Reka Solymosi and Juanjo Medina-Ariza and Wooditch et al.'s (forthcoming) 'A Beginner's Guide to Statistics for Criminology and Criminal Justice Using R'

Chapter 1

A First Lesson About R

Operators & Functions, Objects, and Packages

Learning Outcomes:

- Install `R` and `R Studio`
- Understand how `R Studio` is set up
- Learn what operators, functions, and objects are and their relevance in `R`
- Understand the purpose of packages
- Install your first package

Today's learning tools:

***Total number of activities*: 8**

***Data*:**

- N/A

***Packages*:**

- `dplyr`

Functions introduced (and packages to which they belong)

- `c()` : Concatenates elements to create vectors (`base R`)
- `class()` : Check the class of an object (`base R`)

- `data.frame()` : Create a new data frame object (`base R`)
- `install.packages()` : Installs non-base R packages (`base R`)
- `library()` : Loads the installed non-base R package (`base R`)
- `list()` : Create a list (`base R`)
- `log()` : Computes the natural logarithm (`base R`)
- `View()` : View data in new window (`base R`)

In this lesson, you will be introduced to the programming language, `R`. After installing the related software and getting a basic idea of the R Studio interface, you will learn three `R` basics: operators (and functions), objects, and packages.

Unlike other statistical software like SPSS and STATA, `R` is a free, open-source software for performing statistical analysis and data visualization. In addition, `R` offers more analytical solutions, flexibility, and customization than these commonly used statistical software, and its popularity has increased substantially over the years.

We learn `R` because we hope that this is an important tool that you will continue to use in the future. As it is free and has a community feel to it where anyone can create and upload new techniques, the idea is that you can use `R` long after this course. Even if data analysis is not in the future for you, learning how to conduct and interpret statistical output is a good skill to have – much of our knowledge of the world includes statistics, so understanding the numbers and how they were derived are advantages. `R` uses a language called **object-oriented programming**, and though it may seem daunting at first, practice makes familiarity. Also, you can impress your friends with all your coding.

1.1 Install R & R Studio

As `R` and `R Studio` are free software, you should be able to install these on your own computers at home. You may be working with different IT, so there are different options for a successful install. Our first activity will be to decide what approach to working with `R` and `R Studio` will be best for you.

1.1.1 Activity 1: Identifying your operating system

In this activity, you need to answer a question about your computer/IT that you will be working with for this class. That question is:

- **What is your operating system**? Operating system refers to the software that your computer is running to deliver the basic functions. You may have, for example:
  - *Windows or Linux* - if you have these, you are most likely going to have an easy time installing `R` and `R Studio`, so you should give the installation instructions below a try.
  - *Apple* - if you have a Mac, there are some extra steps to install `R` and `R Studio`. Specifically, there will be an additional programme to download called Xcode, and additional steps to take.
  - *Chromebook* - Installing `R` and `R Studio` on a Chromebook involves installing Linux. Like with a Mac, there are some additional steps you will need to take, and some patience.

In your group google sheets, write down which operating system you have. This will guide which directions to follow later.

1.1.2 Activity 2: Install R & R Studio

1.1.2.1 Some notes specific to your operating system.

Before you move on to the installation steps, look at your answer from Activity 1, and read or watch the advice specific to your operating system:

- Windows: click here for instructions
- Chromebook: read the tutorial here
- Mac, follow the guidance in the video here and then, you will also need to install command line tools, for that you can watch another video here
- Linux: for ubuntu see the video here and if you have questions, let the teaching team know!

Once you have watched or read the instructions for your relevant operating system, you are now ready to actually have a go at downloading the software for yourself. Before you start, write in the google doc any questions or concerns, and once ready, install!

Install `R`:

1. Go to https://www.r-project.org/
2. Click the download R link under the *Getting Started* heading

3. You will be asked to select a Comprehensive R Archive Network (CRAN) mirror. Click the URL closest to your location

4. Click whichever download link is appropriate for your operating system (see above).

5. Then click the *install R for the first-time* link and proceed to install `R`

Install R Studio:

1. Go to https://rstudio.com/

2. Click the *Download* link

3. Scroll down and then click the DOWNLOAD button for the free version of `RStudio`

4. You will be taken to a website to download the free version of `RStudio` that is appropriate for your computer. Download and then install it.

1.1.2.2 Plan B: accessing R Studio remotely throught a web browser

It might be that the above does not work and you find that there are some issues specific to your computer, or something just is not working. In that case, you have two options:

- *Option 1:* You can remotely access one of the university PCs from your browser (Firefox, Chrome, Safari, etc). You can find instructions how to do this here , and the university IT helpdesk can help you access this too. If you do this, you will be able to use the version of `RStudio` installed in the computer clusters.

- *Option 2*: You can access an online version of `R Studio`, which you can access through any web browser (Firefox, Chrome, Safari, etc). To do this, you visit https://rstudio.cloud/, click on 'Get Started For Free', choose the free account, and click on 'sign up'. Then you can always visit this website and log in to use `R Studio` in the cloud. Note that you should start a `New Project` and name it *Modelling Crime Data*, and then all your work will be saved in this project. More on projects will be found in Lesson Two, next week.

1.2 Getting to know RStudio

You only need to open Rstudio (not both R and RStudio); R runs automatically in the background when RStudio is open. Using RStudio makes it easier to work with R than using R itself. If you do not believe this, try working directly in R!

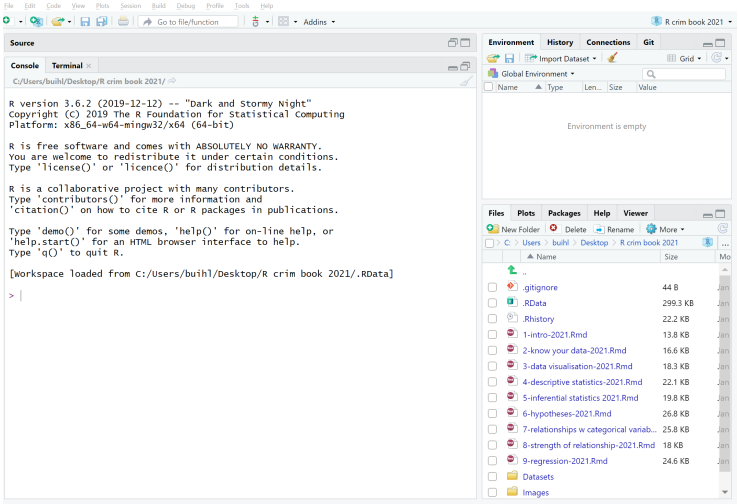


Figure 1.1: **Figure 1.1** R Studio interface

Figure 1.1 shows you what RStudio looks like when you first open it: three open panes.

The biggest one to your left is the main console, and it tells you what version of R you have.

When you start working in Rstudio, it is best to have *four* open panes. **How?** Let us explore in the next activity.

1.2.0.1 Activity 3: Opening up the script pane

Figure 1.2: Click in the *File* drop down Menu, select *New File*, then *R Script*.

You can shift between different panes and re-size them with your mouse too.

1.2.1 The four panes of R Studio

The purposes of the four panes in Figure 1.3 are the following:

1. **Script and data view**: where you type your programming code that tells R what you want to do. These are essentially instructions that you

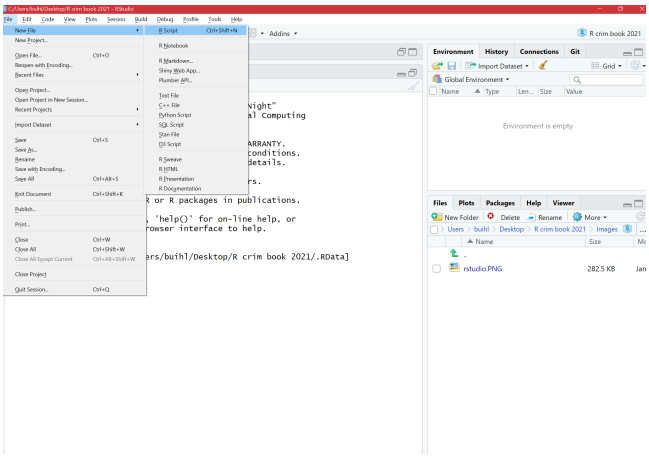


Figure 1.2: **Figure 1.2** Creating an R Script

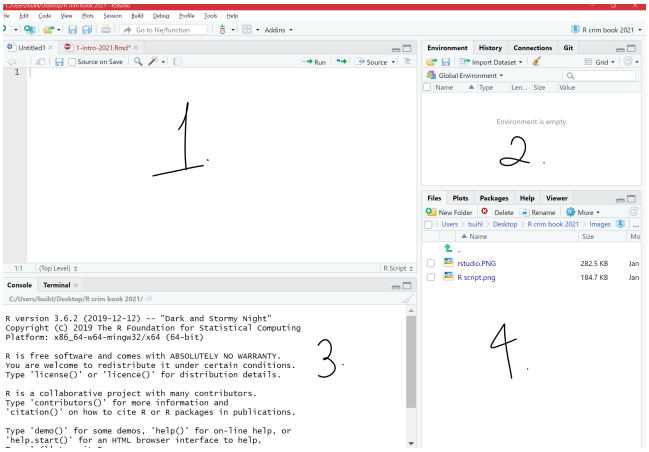


Figure 1.3: **Figure 1.3** Main window in `RStudio`

type and save as a **script**, so that you can return to it later to remember what you did and to share it with others so that they can reproduce what you did.

2. **Environment and history view**:

i) *2.1 Environment* tab- gives you the names of all the (data) objects that you have defined during your current R session, including number of observations and rows in those objects. We learn more about objects later.

ii) *2.2 History* tab- shows you a history of all the code you have previously evaluated in the main console.

3. **Main console**: this is considered R's heart, and it is where R evaluates the codes that you run. You can type your codes directly in the console, but for the sake of good habits, type them in the script and data view so you can save a record of them. Only type and run code from here if you want to debug or do some quick analysis.

4. **File directory, Plots, Packages, Help**:

i) *4.1 Files* tab- allows you to see the files in the folder that is currently set as your working directory.

ii) *4.2 Plots* tab- you will see any data visualizations that you produce here. You have not produced any yet, so it is empty now.

iii) *4.3 Packages* tab- you will see the packages that are currently available to install. We will explain what these are soon, but know that they are an essential feature when working with R.

iv) *4.4 Help* tab- you can access further information on the various packages.

1.2.1.1 Activity 4: Interacting with the 4 panes

In the previous activity, you opened up the 'script' pane. We now write some code in it, and see what happens.

To do this, go to your open version of R Studio, and type in the script pane the following:

```
print("Hello world!")
```

When you have typed this, you will have typed your first bit of code. Yet nothing is happening? That is because you also have to **RUN** the code.

You can do this by highlighting the code you wish to run, and clicking on 'run' in the top right hand corner:

....

When you 'run' the code, it will print the text 'Hello World!'(above in Figure 1.4) in the bottom pane, which is the **console**. That means you have written and executed your first line of code.

In the rest of the session, we will be unpacking how this all works, and getting more familiar and comfortable with using `R Studio`.

1.2.2 Customising R Studio

Did you know you can change the way `RStudio` looks? Click in the *Tools* menu and select *Global Options*. A pop-up window appears with various options:

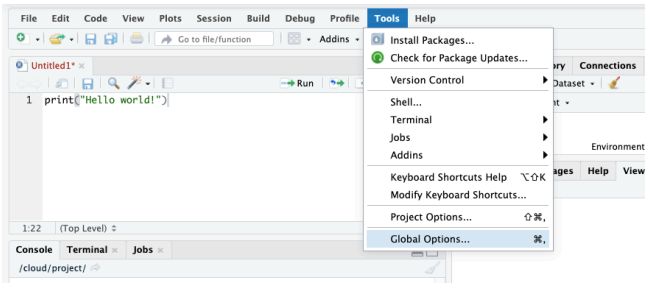


Figure 1.4: **Figure 1.5** Selecting 'Global Options'

Select *Appearance*. Here, you can change things related to the appearance of `R Studio` like the font type and size, including the theme background that `R` will use as the interface.

Tomorrow Night Bright theme in Figure 1.7 is a recommendation because it is easier on your eyes with its dark background. You can preview and then click *Apply* to select the one you like.

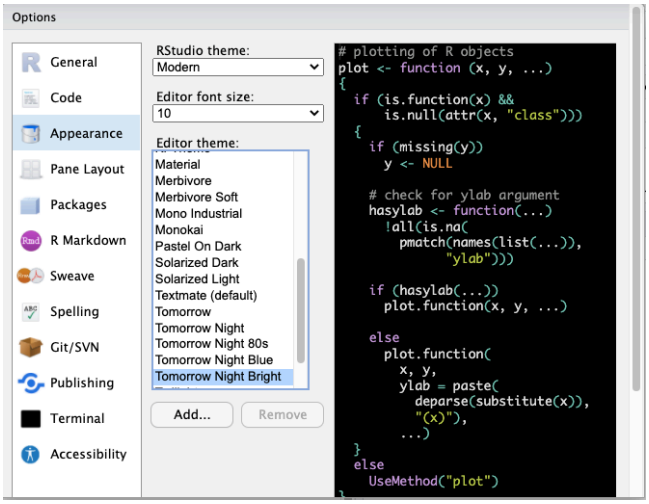


Figure 1.5: **Figure 1.6** Choosing a theme background

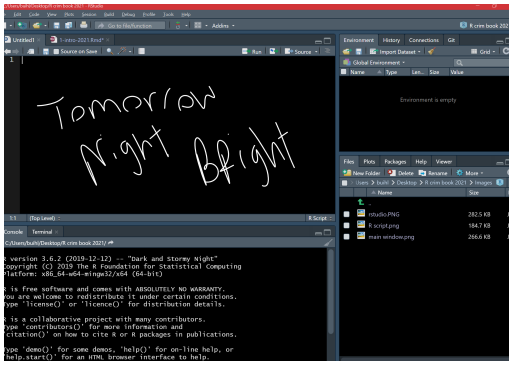


Figure 1.6: **Figure 1.7** Tomorrow Night Bright Theme

1.3 Today's 3 (TOPICS)

Now that we have a basic understanding of R, we are ready to learn some building blocks. Each session of *Modelling Criminological Data* will focus on three substantive topics, which are the key learning outcomes for each week. Today's topics are **operators and functions**, **objects**, and **packages**. Once you have mastered these three topics, you will be ready to start using R for data analysis!

1.3.1 Operators and Functions

Operators are symbols that tell R to do something specific like assigning values to vectors. They fall into four categories: arithmetic, relational, logical, and assignment. For now, let us type out and run some arithmetic operators:

```
# Addition
5 + 5
```

```
## [1] 10
```

```
# Subtraction
10 - 3
```

```
## [1] 7
```

```
# Multiplication
2 * 5
```

```
## [1] 10
```

```
# Division
6 / 2
```

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

Did you *Run* the arithmetic operators? The answers will have appeared, just like the above. If you are wondering what the # symbol means in the coding above, these are called **comments**. They are annotations you can use in your `R script` to tell R that what comes right after the # are not code to be ran. Comments are like your personal notes alongside your coding.

There are other operators we will come across like the *and* operator (`&`) and the *or* operator (`|`), which will be useful when we start to manipulate our data sets.

1.3.1.1 Activity 5: Play around with operators

Return to your open version of `R Studio`, and use this to find the answer to the following:

```
12329 + 342980
18390 / 348
```

The results should appear in the console pane and is known as your output.

Functions are similar to operators in that they *do* things. The difference is that they are called by a certain name, usually a name which represents what they do, and they are followed by brackets `()`. Within the brackets, you can put whatever it is that you want the function to work with. For example, the code we wrote in Activity 4 was the `print()` function. This function told `R` to print into the console whatever we put in the brackets ("Hello world!").

Same idea with a personalised greeting: if you want to print 'Hello Reka', you will need to have "Hello Reka" inside the brackets:

```
print("Hello Reka")
```

```
## [1] "Hello Reka"
```

Hello Reka.

There are so many functions in `R`. We will be learning many of them throughout our class. `Print` is fun, but most of the time, we will be using functions to help us with our data analysis. You might remember some Excel formulas we learned last semester from *Making Sense of Criminological Data*. For example, getting the minimum, maximum, or mean of a list of numbers. `R` does this using functions in a very similar way.

For example, if we have a bunch of numbers, we just find the appropriate function to get the summary we want:

```
mean(c(10, 34, 5, 3, 77))
```

```
## [1] 25.8
```

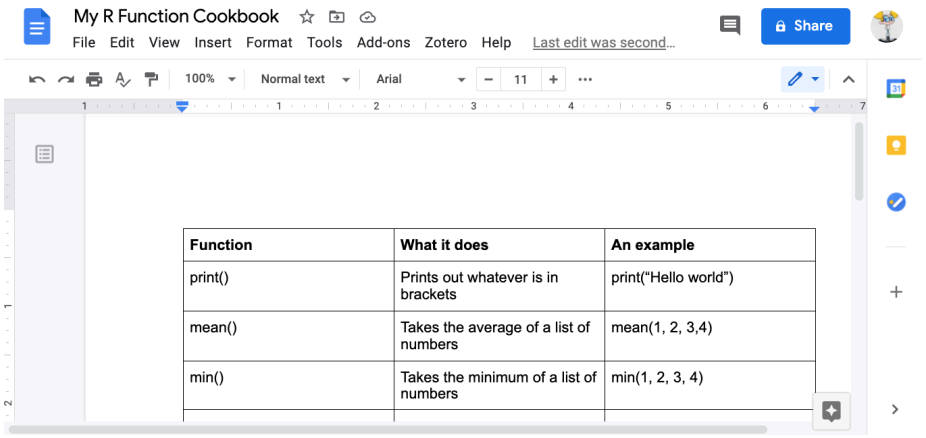
```
min(c(10, 34, 5, 3, 77))
```

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

```
max(c(10, 34, 5, 3, 77))
```

```
## [1] 77
```

How exactly can you find the function you need? Throughout this class, you will learn a list that appears at the top of each lesson. A recommendation is to also create a 'function cookbook', where you write down a list of functions, what the functions do, and some examples. Here is an example:



Function	What it does	An example
print()	Prints out whatever is in brackets	print("Hello world")
mean()	Takes the average of a list of numbers	mean(1, 2, 3,4)
min()	Takes the minimum of a list of numbers	min(1, 2, 3, 4)

Figure 1.7: **Figure 1.8** An example of a function cookbook

You can use google to make your cookbook, and the website stackoverflow, in particular, can help you find the function you need. But be wary, especially in the beginning, that you understand what the function does. There can be several different functions for the same action. One good approach is to add a function of interest to your cookbook and ask the teaching team about what it does, and how it might be different to other functions that do the same thing.

1.3.1.2 Activity 6: Play around with functions

Have a guess of (or google) what you think is the function to get the median. Once you have your answer, write it in the shared google docs. Then, use it to get the median of the numbers 10, 34, 5, 3, 77.

Write the answer in your shared google doc (or note it down for yourself if in the quiet room).

The answer is further below, after the note:

NOTE: `R` is case-sensitive! For example:

```
# Calculating the logarithm
Log(100)

# ERROR!
```

```
# Instead, it should be:
log(100)
```

```
## [1] 4.60517
```

Okay, now you know these, the answer to Activity 6 was…

```
median(c(10, 34, 5, 3, 77))
```

```
## [1] 10
```

Now let us move on to our second key topic for today: objects!

1.3.2 Objects

Everything that exists in `R` is an **object**. Think of objects as boxes where you put things in. Imagine a big, empty cardboard box. We can create this big empty box in `R` by simply giving it a name. Usually, you want your object ('box') to have a good descriptive name, which will tell others what is in it. Imagine moving house. If you have a cardboard box full of plates, you might want to label it 'plates'. That way, when carrying, you know to be careful, and when unpacking, you know its contents will go in the kitchen. On the other hand, if you named it 'box1', then this is a lot less helpful when it comes to unpacking.

1.3.2.1 Activity 7: Creating an object

Let us create an object called 'plates'. To do this, you go to your script, and type 'plates'.

But if you run this code, you will get an error. Let's see:

....

You will see the phrase 'Error: object plates not found'. This is because you have not yet put anything inside the plates 'box'. Remember objects are like

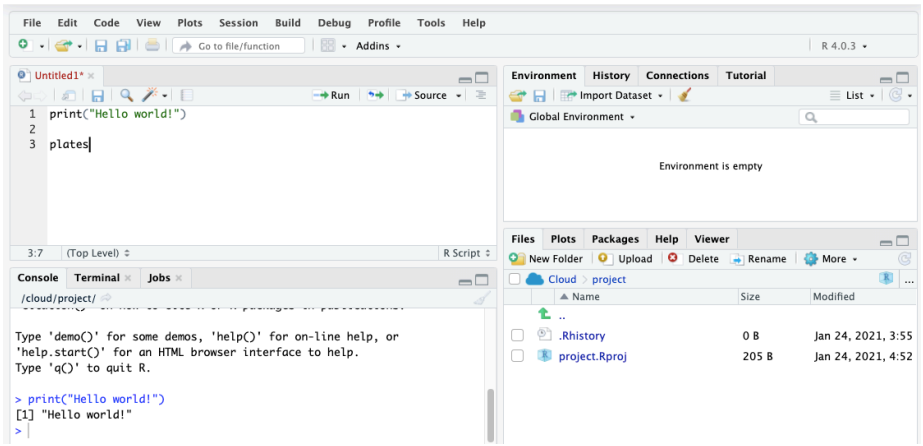


Figure 1.8: **Figure 1.9** Typing 'Plates' in the script

boxes,so there must be something inside our object 'plates'. In order for this object to exist, you have to put something inside it, or in R-speak *assign it some value*.

Therefore, we make an object by using an *assignment operator* (`<-`). In other words, we assign something to an object (i.e., put something in the box). For example:

```
plates <- "yellow plate"
```

Now if we run this, we will see no error message, but instead, we will see the `plates` object appear in our *environment pane*:

....

Here are some more examples to illustrate:

```
# Putting '10' in the 'a' box
a <- 10

# Putting 'Hello!' in the 'abc123' box
abc123 <- "Hello!"
```

In these examples, we are putting the value of `10` into the object `a`, and the value of 'Hello!' into the object `abc123`.

Earlier, we introduced you to the Environment and History pane. We mentioned that it lists objects you defined. After making the 'a' and 'abc123' objects, they should appear in that very pane under the `Environment` tab.

1.3.2.2 Types of objects

Why are objects important? They are so because we will be storing everything in our data analysis process in these objects. Depending on what is inside them, they can become a different type of object. Here are some examples:

Data structures are important objects that store your data, and there are five main types but we focus on three for this course:

1. *(atomic) vector*: an ordered set of elements that are of the same *class*. Vectors are a basic data structure in `R`. Below are five different classes of vectors:

```
# 1. numeric vector with three elements
my_1st_vector <- c(0.5, 8.9, 0.6)

# 2. integer vector with addition of L at the end of the value
my_2nd_vector <- c(1L, 2L, 3L)

# 3. logical vector
my_3rd_vector <- c(TRUE, FALSE, FALSE)
# 'my_4th_vector' creates a logical vector using abbreviations of True and False, but you should
my_4th_vector <- c(T, F)

# 4. character vector
my_5th_vector <- c("a", "b", "c")

# 5. complex vector (we will not use this for our class)
my_6th_vector <- c(1+0i, 2+4i)
```

2. *lists*: technically they, too, are vectors but they are more complex because they are not restricted on the length, structure, or class of the included elements. For example, to create a list containing strings, numbers, vectors and a logical, use the `list()` function, and inside the brackets, put everything that you want to combine into a list:

```
list_data <- list("teal", "sky blue", c(10, 5, 10), TRUE, 68.26, 95.46, 99.7)
```

Above, we created `list_data`, an object that contains all those things that we put inside the `list()` function. This function serves to create a list from combining everything that is put inside its brackets.

We then use the `class()` function to confirm that the objects have been defined as a list:

```
class(list_data)
```

```
## [1] "list"
```

3. *data frames*: also stores elements but differ from lists because they are defined by their number of columns and rows; the vectors (columns) must be of the same length. Data frames can contain different classes but each column must be of the same class. (More on class next week.) For example, if you want to combine some related vectors to make a data frame on violent American cities, use the function `data.frame()`:

```
# Making some relevant vectors
TopVioCities <- c("St. Louis", "Detroit", "Baltimore") # some violent US cities
VioRatePer1k = c(20.8, 20.6, 20.3) # their violence rates per 1,000 persons
State <- c("Missouri", "Michigan", "Maryland") # in what states are these cities found

#Join the newly created vectors to make a data frame called 'df'
df<-data.frame(TopVioCities, VioRatePer1k, State)
```

We can then view the data frame, 'df', with the `View()` function in which a tab will appear containing our vectors:

```
View(df)
```

1.3.2.3 Activity 8: Doing things to objects

We have learned what functions are (i.e., things that do things) and what operators are (i.e., symbols that do things) as well as what are objects (i.e., the boxes that hold things). We also saw some functions which helped us create objects. Functions can also do things to objects. For example, we saw the function `class()` that told us what kind of object `list_data` was, and `View()` which allowed us to have a look at our dataframe we called `df`.

Let us look back at our `plates` object. Remember it was the object that held our kitchen items, specifically that beloved yellow plate. We added 'yellow plate' to it. Now let us add some more plates (we like to collect plates apparently) and let us use the concatenate `c()` function for this again:

```
plates <- c("yellow plate", "purple plate", "silver plate", "orange bowl")
```

Let us say that we suddenly forgot what was in our object called 'plates'. Like what we learned earlier, we use the function `print()` to see what is inside this object:

```
print(plates)
```

```
## [1] "yellow plate" "purple plate" "silver plate" "orange bowl"
```

This can apply to obtaining the mean, the minimum, and maximum. You could assign those statistics to an object this time:

```
nums <- c(10, 34, 5, 3, 77)
```

Now if we want to know the mean, we can take the mean of the object `nums`, which we just created:

```
mean(nums)
```

```
## [1] 25.8
```

The object we will use most frequently though is data frames. Much like how we worked with spreadsheets in Excel in *Making Sense of Criminological Data*, we will be working with dataframes in `R`.

We had created a dataframe called `df` previously. If you have not yet created this dataframe in your own `R Studio`, do this now. You should have the object `df` in your environment. When you run `View(df)`, you should see this dataset:

To do something to an entire dataframe, we would use the name of the object (`df`) to refer to it. In the case of the `View()` function, we want to see all that is contained in `df`, so we will type `View(df)`. On the other hand, if we want to refer to only one *variable* or column in the data, (recall last semester - each variable is held in each column) there is a special notation to do this – $

To refer to a variable inside a dataframe, you use:

$dataframename + \$ + variablename$

For example, to refer to the variable `VioRatePer1k`, we use the notation `df$VioRatePer1k`.

And if we wanted to view only that variable, we use:

```
View(df$VioRatePer1k)
```

You should see:

Say we wanted to know the mean violence rate across our units of analysis, the cities, for example, we would take the numeric column to calculate this:

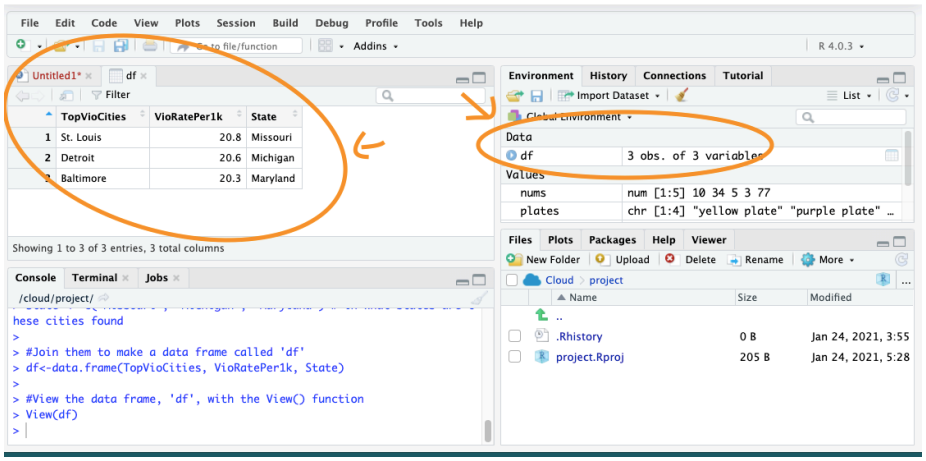


Figure 1.9: **Figure 1.12** The data frame 'df'

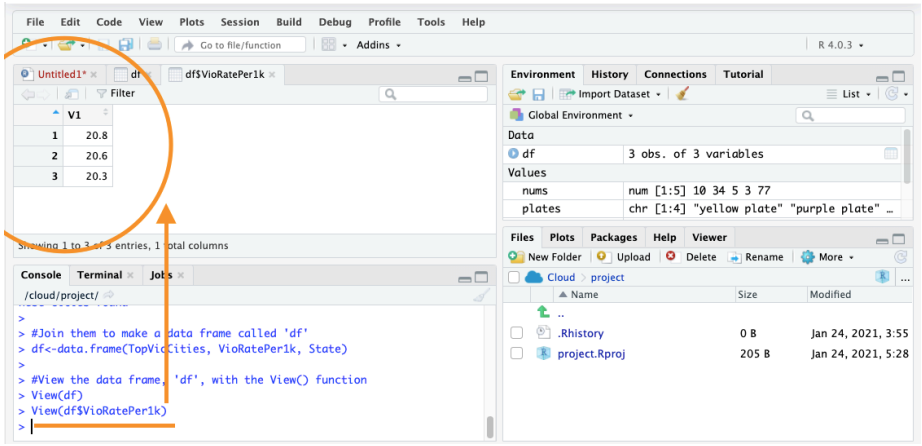


Figure 1.10: **Figure 1.13** Viewing df$VioRatePer1k

```
mean(df$VioRatePer1k)
```

```
## [1] 20.56667
```

1.3.3 Packages

Packages are a very important element of R. Throughout the course, and hopefully afterwards, you will find yourself installing numerous open source software packages that allow R to do new and different things. There are loads of packages out there. In early 2020, there were over 150,000 packages available. Anyone can write one, so you will need to be careful with which ones you use as the quality can vary. Official repositories, like CRAN, are your best bet for packages as they will have passed some quality controls.

You can see what packages are available in your local install by looking at the *packages* tab in the *File directory, Plots, Packages* pane (Figure 1.14).

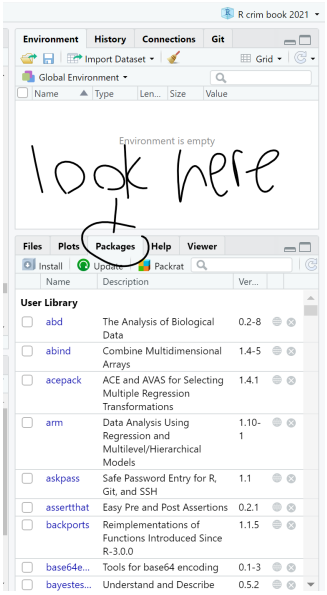


Figure 1.11: **Figure 1.14** Packages Tab

A number of the packages we will use belong to a set of packages called **tidyverse**. These packages help make your data tidy. According to Statistician and Chief Scientist at RStudio, Hadley Wickham, (also an author of some of your readings) transforming your data into *tidy data* is one of the most important steps of the data analysis process. It will ensure your data are in the format you need to conduct your analyses.

Packages can be installed using the `install.packages()` function. Remember that while you only need to install packages once, they need to be loaded with the `library()`function each time you open up `RStudio`. Let us install the package `dplyr` from `tidyverse` and load it:

```
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
```

A lot of code and activity appears in the console. Warnings may manifest. Most of the time, the warnings explain what is being loaded and confirm that the package is successfully loaded. If there is an error, you will have to figure out what the warnings are telling you to successfully load the package. This happens and is normal.

To double check that you have actually installed `dplyr`, go to that *File Directory, Plots, Packages* pane and click on the *Packages* tab. The list of packages is in alphabetical order and `dplyr` should be there. If there is a tick in its box, it means that this package is currently loaded and you can use it; if there is no tick, it means that it is inactive, and you will have to bring it up with `library()`, or just tick its box (Figure 1.15).

On *masking*: sometimes packages introduce functions that have the same name as those that are already loaded, maybe from another package, into your session. When that happens, the newly loaded ones will override the previous ones. You can still use them but you will have to refer to them explicitly by bringing them up by specifying to which package they belong using `library()`.

Keep `dplyr` in mind as next week we will learn more about it!

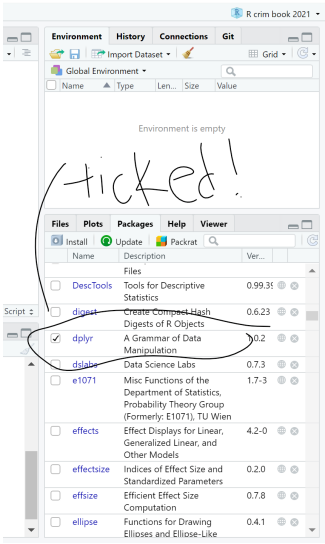


Figure 1.12: **Figure 1.15** dplyr Package Ticked

1.4 SUMMARY

Today you installed both `R` and `RStudio` and had a gander around the `RStudio` interface. If you were bored with its default look, you could customise the interface. When working with `RStudio`, it is best to have the four panes. One had to do with the all-important **script** and we use `#` in the coding when we want to make **comments**.

We learned about some important features of `R`. First were the **operators**. These are symbols that tell `R` what to do and come in four types. One we will use quite a bit is the *assignment operator*, which is the symbol, `<-`. Also important were **functions**, which are similar to operators except their names tell you what they do. Second were **objects**, which are pervasive when working with `R`. They are like boxes that you put stuff in, and we learned about three specific types of **data structure** objects: vectors, lists, and data frames. Third were **packages**. These are open source software that expand what `R` can do. We installed `dplyr` as an example, and we will use this `tidyverse` package in our next session.

That is all for today. Do not forget to do your homework task and the quiz by next week!

Chapter 2

Getting to know your data

Variables, Labels, and Subsetting

Learning Outcomes:

- Create a project in `R` to refer back to for each session
- Understand what variables are and how to examine them in `R`
- Learn how to make new variables
- Learn how to label variables and their values
- Learn how to subset select observations and variables

Today's Learning Tools:

***Total number of activities*: 9**

Data:

- National Crime Victimization Survey (NCVS)

Packages:

- `dplyr`
- `here`
- `haven`
- `labelled`
- `sjlabelled`

Functions introduced (and packages to which they belong)

- `%>%` : Known as the pipe operator, and allows users to pass one output of a code to become the input of another (`dplyr`)
- `as_factor()` : Changes the class of an object to factor class (`haven`)
- `attributes()` : Access object attributes, such as value labels (`base R`)
- `case_when()` : Allows users to vectorize multiple if / if else statements (`dplyr`)
- `count()` : Counts the number of occurrences (`dplyr`)
- `dim()` : Returns the number of observations and variables in a data frame (`base R`)
- `dir.create()` : creates a new folder in a project (`base R`)
- `factor()` Creates a factor (`base R`)
- `filter()` : Subsets a data frame to rows when a condition is true (`dplyr`)
- `get_labels()` : Returns value labels of labelled data (`sjlabelled`)
- `here()` : Finds a project's files based on the current working directory (`here`)
- `mutate()` : Creates new vectors or transforms existing ones (`dplyr`)
- `names()` : Returns the names of the variables in the data frame (`base R`)
- `read_spss()` : Imports SPSS .sav files (`haven`)
- `recode()` : Substitutes old values of a factor or a numeric vector by new ones (`base R`)
- `remove_labels()` : Removes value labels from a variable (`sjlabelled`)
- `remove_var_label()` : Removes a variable's label (`labelled`)
- `select()` : Select columns to retain or drop (`dplyr`)
- `table()`: Generates a frequency table (`base R`)
- `var_label()` : Returns or sets a variable label (`labelled`)

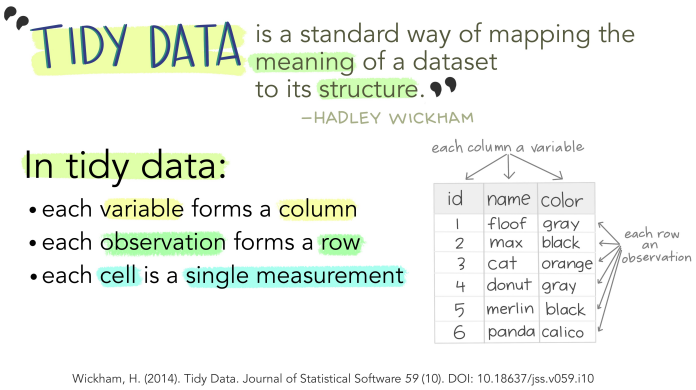
2.1 The Tidyverse

Last time, we installed our first package, `dplyr`. This is one of a number of packages from what is known as **tidyverse**.

The tidyverse contains packages that help us carry out **wrangling** (i.e., cleaning variables), analysis, plotting, and modelling of our data. The ethos of tidyverse is that working and using tidy data makes our lives easier in the long run, allows us to stick to specific conventions, and enables us to share with others who follow this approach. Essentially, tidy data makes data analysis easy to carry out and to communicate to others.

So what is tidy data?

Figure 2.1: **Figure 2.1** Tidyverse logo



Like what was mentioned in last week's lesson, our columns represent our variables, but, also, our cases (or better known in the tidyverse as **observations**) are our rows, whereby each cell is a value of that column's given variable for the observation in that row.

In this class, we will be working with tidy data. Generally, if you have messy data, which is common in the real world of data analysis, your first task is to wrangle it until it is in a tidy shape similar to the ones described in Figures 2.2 and 2.3.

In today's lesson, we use `dplyr` and other `tidyverse` packages to explore and get to know our data.

Being familiar with our data requires knowing what they comprise and what form they take. 'Tidying' our data through wrangling – labelling, reformatting, recoding, and creating new variables – will help with this step in the data analysis process. Today we will learn another three topics related to tidy data: **variables**, **labels**, and **subsetting**.

Let us get started and load `dplyr`:

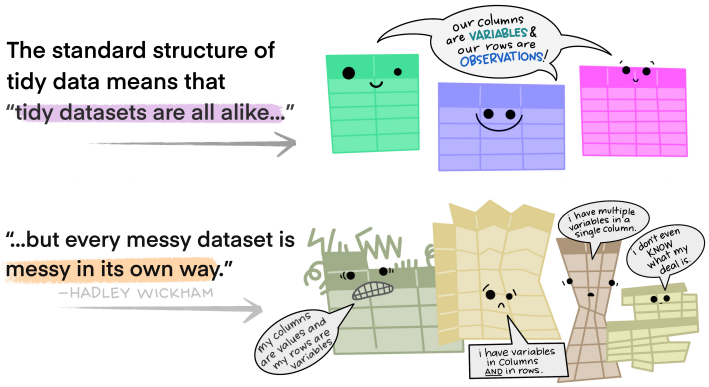


Figure 2.2: **Figure 2.3** Tidy data versus messy data by Alison Horst

```
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
```

```
# Like last week, you can also check the 'Packages' tab in the 'Files, Plots...' pane
```

2.2 R Projects – Getting Your Work Files Organised

Although today is focused on tidy data, it is also helpful if whatever you are working on is also tidy and found in one easily accessible folder. By saving work inside a **project**, you can find files such as data and scripts related to specific work in a single working directory. Let us get into the habit of doing this:

2.2.1 Activity 1: Making yourself a project

- Click on the top right tab called *Project: (None)* - Figure 2.4

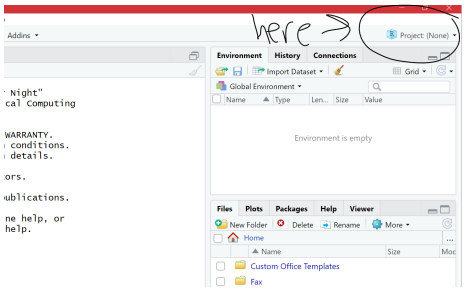


Figure 2.3: **Figure 2.4** Click on Project: (None)

In the options that appear, choose *Existing Directory*. The reason is you may have already created a folder for this work if you had saved your script from last week and from what you have done so far today. For example, Reka had saved a folder called 'modelling2021' with her scripts in it and will have her project in that same place too (see Figures 2.5 and 2.6).

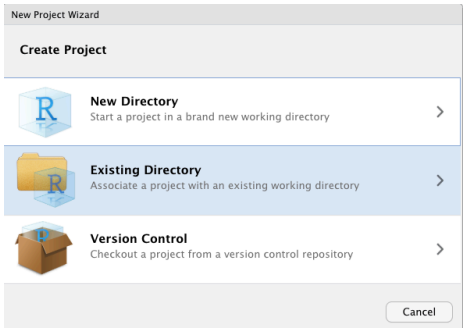


Figure 2.4: **Figure 2.5** Existing directory

Another option appears (Figure 2.6). Use the 'Browse…' button to select the folder where you have so far saved your script(s). Once you have done this, click on *Create Project* and your new project will be launched.

This will now open up a new `RStudio` window with your project. In the future, you can start right where you left off by double clicking your project, which is a `.Rproj` file, to open it. It helps if you keep all your work for this class in one place, so `R` can read everything from that folder.

Now if you did not save any scripts or files thus far, so want to create a project in a new place, you can do the following:

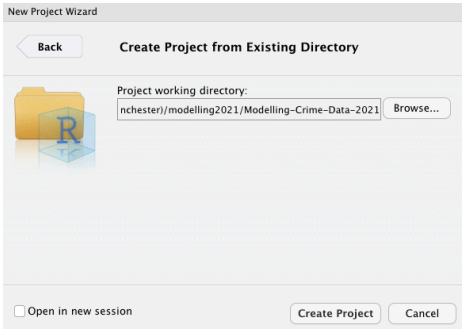


Figure 2.5: **Figure 2.6** Create project from existing directory

- Click on *New Project* (Figure 2.5). A window with different options appears. Create your project in *New Directory* and then click *New Project*
- Choose a name for your project (e.g., r_crim_course) and a location (i.e., working directory, which will be explained a little further below) where your project will be created

Inside your project, you can organise it by having separate files: one for scripts, one for data, one for outputs, and another for visuals. You can make these with the function `dir.create()`:

```
# For example, to make a sub-folder called 'Datasets' in your project folder, type thi
dir.create("Datasets")
```

Another example: to create the online tutorial for this class, we have a project called `Modelling-Crime-Data-2021`. Figure 2.7 shows that this title appears in the top right hand corner and its contents, including sub-folders, appear in under the *Files* tab in the *Files, Plots…* pane.

The files in your *Files, Plots…* pane tell you in what folder you are automatically working. This is known as your **working directory**, the default location that appears when you open `RStudio`. Wherever your `R project` (that file ending in `.Rproj`) is saved will be the working directory.

In your group google sheets, type the name of your `R` project and in which location it is in. Now decide whether this is a good location for your `R` project. For example, is the location name too long? (Something like 'C: Users xx Desktop xx xx Manchester xx xx xx xx xx xx' is too long and you might run into problems later.) Or are there files that are for your other course units in there too? If doubtful about the location of your project, move it somewhere else you think is more appropriate.

You can read on why projects are useful here: https://www.r-bloggers.com/2020/01/rstudio-projects-and-working-directories-a-beginners-guide/

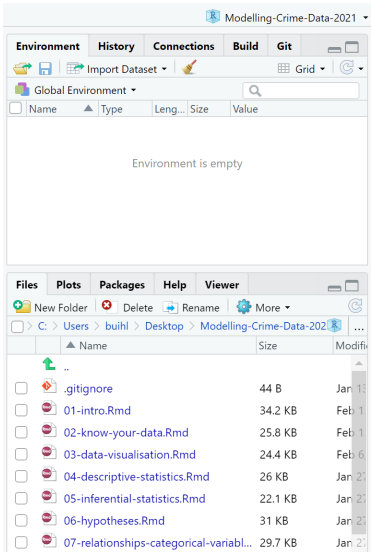


Figure 2.6: **Figure 2.7** How a project appears

2.2.1.1 The `here` package

Whenever you want to locate certain files within your Project, use the `here` package:

```
# First, you must install it if you have not done so:
install.packages("here")
```

```
# Then bring it up with 'library()' because it may not come up automatically after installing it
library(here)
```

```
## here() starts at /Users/reka/Dropbox (The University of Manchester)/modelling2021/Modelling-Cr
```

Using the `here` package is better than typing out the exact location of your file, which can be tedious. The next section shows how to use the `here()` function from said package to import data from the National Crime Victimization Survey (NCVS).

2.3 Importing Data

Here is another `tidyverse` package to install:

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

```
library(haven)
```

`haven` enables `R` to understand various data formats used by other statistical packages such as SPSS and STATA. We will need this package to open data in its diverse forms. When we worked with Excel last semester in *Making Sense of Criminological Data*, we could only open certain type of files. With `R`, we can open a wide range of data files, which opens up many possibilities of analysis for us. Let us give this a try now.

2.3.1 Activity 2: Importing and Viewing Data

Go to the class Blackboard. Then click on *Learning materials* and then on *Week 2* where you will find the dataset 'NCVS lone offender assaults 1992 to 2013'.

Download the data into the *Datasets* sub-folder in your working directory you had created using the function `dir.create()`.

This data comes from The National Crime Victimization Survey (NCVS) from the US. You can read more about it here.

Now how to bring this dataset up in `RStudio`? We must follow these steps to *read* the data:

- 1. We note what kind of dataset file, 'NCVS lone offender assaults 1992 to 2013', is. The extension is `.sav` and this means that it is a file from the statistical package SPSS, so the function we need is `read_spss()` from the `haven` package.

- 2. When importing data with the `here()` function, you must specify where the file is and what it is called in the brackets. In this case, we need to specify that it is in the sub-folder 'Datasets' and it is called 'NCVS lone offender assaults 1992 to 2013.sav'. So, the code to find the file would be: `here("Datasets", "NCVS lone offender assaults 1992 to 2013.sav")`.

Completing these steps, we now can load our dataset and place it inside a data frame called `ncvs`:

```
# Importing our SPSS dataset and naming it 'ncvs'
ncvs <- read_spss(here("Datasets", "NCVS lone offender assaults 1992 to 2013.sav"))
```

What you are saying to `R` is the following:

> My data, *NCVS lone offender assaults 1992 to 2013*, is a .sav file. Therefore, it is an SPSS dataset and is located in the sub-folder called *Datasets* in my default working directory. `R`, please extract it from *here*, understand it, and place it in a data frame object called `ncvs`, so I can find it in `RStudio`.

To view the new data frame, `ncvs`, type:

```
View(ncvs)
```

A tab appears labelled 'ncvs'. In it, you can view all its contents. In your group google sheets, type how many 'entries' and 'columns' there are in our data frame, `ncvs`.

There are other ways to load data that are of different formats. For more information, see this cheatsheet for importing data. But for now, you can rely on us showing you the functions you need.

2.4 Today's 3 (TOPICS)

We now have data to tidy in `R`, so onto our three main topics for this week: **variables**, **labels**, and **subsetting**.

2.4.1 Variables

Variables can be persons, places, things, or ideas that we are interested in studying. For example, height and favourite football team.

Last week, we learned a little on how to examine what our variables are. Let us revisit this. One of the first things we do to get to know our `ncvs` data frame is to identify the number of rows and columns by using the function `View()` as we did above, or, a more direct way, the function `dim()`:

```
dim(ncvs)
```

```
## [1] 23969    47
```

Like your answer from Activity 2, the data frame`ncvs` has 47 columns, meaning that it has 47 variables. What are these variables all called? We can get their names using the `names` function:

```
names(ncvs)
```

```
##  [1] "YEAR"                "V2119"               "V2129"
##  [4] "V3014"               "V3016"               "V3018"
##  [7] "V3021"               "V3023"               "V3023A"
## [10] "V3024"               "V2026"               "V4049"
## [13] "V4234"               "V4235"               "V4236"
## [16] "V4237"               "V4237A"              "V4238"
## [19] "V4239"               "V4240"               "V4241"
## [22] "V4242"               "V4243"               "V4244"
## [25] "V4245"               "V4246"               "V4246A"
## [28] "V4246B"              "V4246C"              "V4246E"
## [31] "V4246F"              "V4246G"              "V4247"
## [34] "V4528"               "injured"             "privatelocation"
## [37] "reportedtopolice"    "weaponpresent"       "medicalcarereceived"
## [40] "filter_$"            "relationship"        "Policereported"
## [43] "victimreported"      "thirdpartyreport"    "maleoff"
## [46] "age_r"               "vic18andover"
```

We observe that a number of these variable names are codes, which is somewhat similar to the Crime Survey of England and Wales (CSEW) data we worked with last semester, whereby the variable *polatt7*, for example, was trust in police.

You could view the data frame like you did previously (with the function `View()`) to find out what the variables actually are, or you could look it up in the data dictionary here. The advantage of the data dictionary – which will accompany all well-documented datasets – is it will tell you precisely what the variables are and how they measure their characteristics in that particular dataset. For example, the data dictionary that accompanies the NCVS tells us that the variable `V3014` is age.

Measurement

What about the level of measurement for these variables? Different variable types refer to different levels of measurement.

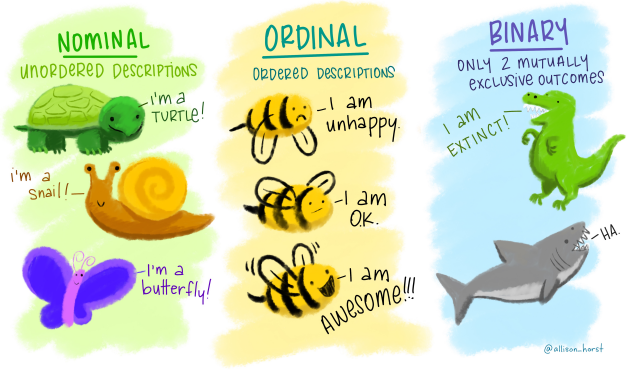


Figure 2.7: **Figure 2.8** Categorial variables by Allison Horst

For **categorical variables**, we can have variables that are nominal (no order), ordinal (have an order), or binary (only two possible options, like 'yes' and 'no').

Numeric variables can be classified two separate ways. Last semester, we discussed the difference between *interval* and *ratio* variables. Interval variables have the same distance between observations, but have no true zero. The temperature in Celsius is one example. Ratio variables, on the other hand, do have a true zero point. Calculating a ratio from the values of these variables makes sense, but for values from interval variables, it does not. For example, it is pretty meaningless to state that 20 degrees Celsius is twice as hot as 10 degrees Celsius ('0' in Celsius is not the absolute lowest temperature). But if Reka has £30 pounds in her bank account and Laura has £60, it is meaningful to say Laura has twice as much savings as Reka does.

Another way to classify numeric variables is to distinguish between discrete and continuous variables. *Discrete* numeric variables have set values that make sense. For example, crime is one such variable. It is understandable to have 30 burglaries in May and 50 burglaries in December, but it is not understandable to have 45.2482 burglaries. *Continuous* numeric variables, however, can take on any value between a lower and upper bound and be meaningful. For example, weight and height. Here is an apt illustration:

2.4.1.1 Variables in `R`

So, how are these levels of measurement among variables relevant in `R`? Nominal and ordinal variables are encoded as a **factor** class because they are categorical characteristics, so take on a limited number of values; factors are like the integer vector introduced last week but each integer is a label. Numeric variables, on the other hand, are encoded as a **numeric** class.

Figure 2.8: **Figure 2.9** Discrete versus continuous numeric variables by Allison Horst

2.4.1.1.1 Activity 3: Identifying a variable's level of measurement Identifying the level of measurement should be straightforward when examining each variable. In some cases, however, `R` may not quite grasp what kind of variable you are working with. Thus, you will need to find out what `R` thinks your variable is classed as.

How do you ask `R` what your variable is? Let us find out by using our `ncvs` data frame.

First, do you remember from last week how to refer to one specific variable in your data frame?

It is: `dataframe$variablename`

If we want to find out about the variable `injured` (whether the person was injured or not) from our data frame `ncvs`, for example, we can refer to the variable specifically. Let us use the `attributes()` function to examine this variable.

```
# To see the class of a specific variable, such as the variable 'injured', we use:
attributes(ncvs$injured)
```

```
## $label
## [1] "Victim Sustained Injuries During Vicitmization"
##
## $format.spss
## [1] "F8.2"
```

```
##
## $display_width
## [1] 10
##
## $class
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $labels
## uninjured   injured
##         0         1
```

```
# The $ symbol allows us to access specific variables in a data frame object

# The $ symbol must be accompanied by the name of the data frame!
```

We can see the label ('Victim Sustained Injuries During Vicitmization'), and the values (at the bottom) indicating '0' for 'uninjured' and '1' for 'injured'. This appears to be a categorical variable with 2 possible values; therefore, a *binary* variable.

Now your turn: find out what is the class of the variable `weaponpresent`. In your googledoc, type out the answer and the code you used to get that answer.

2.4.1.2 Formatting Classes and Value Labels

In some cases, you may want to make changes to how variables are classed. For example, in our data frame `nvcs`, some of the variables are classed as `haven_labelled`.

What is this, you ask? When we use the `haven()` function to import data, `R` keeps the information associated with that file – specifically the value labels that were in the dataset. In practice, therefore, you can find categorical data in `R` embedded in very different types of vectors (e.g., character, factor, or haven labelled) depending on decisions taken by whomever created the data frame.

Although we understand why these variables are labelled as `haven_labelled`, they do not help us understand what class these variables actually are. If we know a variable is classed as factor, We can change it to be so. For example, we want to change the class of variable `V3018` to be accurately classed as factor:

```
# V3018 is a binary variable indicating sex
attributes(ncvs$V3018)
```

```
## $label
## [1] "SEX (ALLOCATED)"
##
## $format.spss
## [1] "F1.0"
##
## $display_width
## [1] 7
##
## $class
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $labels
##            Male          Female         Residue Out of universe
##               1               2               8               9
```

```
# It is indeed classed as 'haven-labelled'

# Naming the newly created factor 'sex' so we do not erase the original variable
# Specify the order we want our variable labels using 'labels= c()'
ncvs$sex <- factor(ncvs$V3018, labels = c("male", "female"))

table(ncvs$sex)
```

```
##
##   male female
##  12533  11436
```

```
attributes(ncvs$sex)
```

```
## $levels
## [1] "male"   "female"
##
## $class
## [1] "factor"
```

The new variable, `sex`, a duplicate of `V3018`, is no longer a 'haven_labelled' type variable. It is now classed as a factor. But remember, 'class' and 'factor' are `R` lingo – **we would still describe this as a categorical, binary variable!** We include `R` language so that you know what it means and how it links to what you have learned in your data analysis classes.

2.4.1.3 Recoding and Creating New Variables

What if we want to create a new variable? Here are three scenarios where we would want to do so:

Scenario 1: we want a variable like the `injured` variable, but for the values, we instead want to see 'uninjured' and 'injured' and not '0' and '1';

Scenario 2: we want a composite variable, like the fear of crime composite variable from last semester that comprised many different scores;

Scenario 3: we want to change an existing variable that is *ordinal* with four outcomes into a *binary* variable with only two outcomes.

We address all three scenarios in turn. Recoding and creating new variables is called **data wrangling** and the package `dplyr` is most appropriate for doing so.

Figure 2.9: **Figure 2.9** Data wrangling by Allison Horst

2.4.1.4 Activity 4: Creating a new variable by recoding an existing one

For scenario 1, we want to recode our existing `injured` variable. We view a frequency table of this variable to understand why. The frequency table tells us the number of times each value for the variable occurs. This is similar to the Pivot Table function in Excel from last semester.

In R, the way to create a frequency table for one variable is to use the function `table()`. Inside the brackets, you would type the data frame and the variable you want to create the frequency table for. We want a frequency table for the `injured` variable:

```
table(ncvs$injured)
```

```
##
##     0     1
## 16160  7809
```

The frequency table shows that 16,160 people answered '0' to the question of whether they were injured, while 7,809 answered '1'. Since we ran our `attribute()` function earlier, we know that '0' means 'uninjured', and '1' means 'injured'. Often in data, '0' represents the absence of the feature being measured and '1' means the presence of such feature.

Although we know what the numbers represent, we might forget or someone else unfamiliar with the data views the variable and may not know what the values mean. In this case, it would be helpful to change the numbers '0' and '1' to what they represent.

To do so, we create a new variable whereby a new column appears in the data frame. It is similar to when you create an object. Remember:

```
name <- "Reka"
```

The only difference is that we must attach this new object (which appears as a column) to the dataframe, and that the number of things we put in this object needs to match the number of rows in that data frame. As we are creating a new variable from one that already has the same number of rows, this is not an issue.

Let us again create a duplicate variable of the `injured` variable:

```
# Create the new variable 'injured_new' from 'injured'
ncvs$injured_new <-ncvs$injured
```

View the `ncvs` data frame. Notice that a new column, `injured_new`, appeared at the end with the exact same contents as the variable `injured`. We will now change those '0' and '1' values.

A function to change values is `as_factor()` from the `haven` package. This function takes the labels assigned to the values of the variable, and changes those original values into these very labels.

```
# Remember: You can check what package each function we learn today belongs to by referring to th

ncvs$injury_r <- as_factor(ncvs$injured)
```

In the data frame `ncvs`, you will see this new column `injured_r`. If we make the frequency table with this new variable, we see that the values are readily understandable as 'uninjured' and 'injured':

```
table(ncvs$injury_r)
```

```
##
## uninjured   injured
##     16160      7809
```

This a lot easier than 'VLOOKUP' from last semester!

2.4.1.5 Activity 5: Creating a composite variable from more than 1 existing variable

We turn to Scenario 2: we want to create a new variable in our `ncvs` data frame that indicates the severity of the victimization experienced by the respondent.

That severity will be measured by two variables: (1) whether the offender had a weapon and (2) whether the victim sustained an injury during their victimization. These are not necessarily the best variables to use in measuring victimization severity; this example, however, should illustrate how you might combine variables to create a new one.

Before we do this, we need to know if we can actually do so by getting to know those variables of interest. By using the function `count()`, we get a good sense of what the values represent and the number of respondents in each of those values for both variables.

```
# Is the appropriate package, 'dplyr', loaded?

# Using count ( ) for 'injured' and 'weaponpresent'
count(ncvs, injured)
```

```
## # A tibble: 2 x 2
##         injured     n
## *     <dbl+lbl> <int>
## 1 0 [uninjured] 16160
## 2 1 [injured]    7809
```

```
count(ncvs, weaponpresent)
```

```
## # A tibble: 3 x 2
##   weaponpresent     n
## *         <dbl> <int>
## 1             0 15814
## 2             1  6652
## 3            NA  1503
```

This function tells us that `injured`, a binary variable, is stored as numbers, where the 0 value means the victim was uninjured and the 1 value means they were injured. Also, the `weaponpresent` variable is (should be) a binary variable stored as numbers. Here, more victims report that the offender did not use a weapon during the offence (n= 15,814) as opposed to using one (n= 6652). But there are also a number of missing values for this question (n= 1503).

Now what if we wanted to combine these, so we can have a score of severity, which takes into consideration presence of weapon and injury?

There is a particular function from the `dplyr` package that is very handy for creating a new variable from more than 1 variable. It is called `mutate`. The `mutate()` function will create a new column in our data frame that comprises the sum of both of these variables, keeping the old variables too:

```
# Create the new variable with mutate:

# 1. The first argument inside the `mutate` function is the data frame into which we w
# 2. After the comma, we insert the equation that adds together the variables 'injured
# 3. This new variable is saved in the data frame `ncvs` (to do so, it overwrites the

ncvs <- mutate(ncvs, severity = injured + weaponpresent)
```

Now view the data frame to see the new variable `severity`. The `severity` variable is ordinal, where '0' is the least severe (neither a weapon was used nor the victim injured), '1' is more severe (either the offender wielded a weapon or the victim reported being injured), and '2' is the most severe (the respondent reported being injured and the offender had a weapon).

Let us see the new variable in the frequency table:

```
table(ncvs$severity)
```

```
##
##     0     1     2
##  9945 10862  1659
```

You can then add value labels to reflect your understanding of the summed scores. To do so, you can over-write the existing `severity` variable (instead of making an additional duplicate variable).

You do so by specifying it on the left side of the `<-` (assignment operator). Then, on the right side, you use the `recode()` function. Inside the brackets of the `recode()` function, we specify the variable we want to change, and then we follow with a list of values. Notice that the numbers must be between the crooked quote marks ``:

```
ncvs$severity <- recode(ncvs$severity, `0` = "not at all severe", `1` = "somewhat severe", `2` =
```

View this new version of the `severity` variable in a frequency table:

```
table(ncvs$severity)
```

```
##
## not at all severe   somewhat severe       very severe
##              9945             10862              1659
```

The above example was simple, but often, we will want to make more complex combinations of variables. This is known as **recoding**. And it will constitute our next activity.

2.4.1.6 Activity 6: Recoding

Now to Scenario 3: we want to turn the variable `relationship` into a binary variable called `notstranger` whereby the offender was either a stranger (0) or was known to the victim (1).

First, we use the `table()` function to create a frequency table for `relationship`, and it has four categories:

```
table(ncvs$relationship)
```

```
##
##    0    1    2    3
## 6547 2950 4576 9227
```

What do these categories mean? We can use the `as_factor()` function from the `haven` package to find out:

```
table(as_factor(ncvs$relationship))
```

```
##
##        stranger  slightly known casual acquiant      well known      Don't know
##            6547            2950            4576            9227               0
```

There are four categories of relationship in addition to a 'don't know' category, but there are no observations in it.

We want to turn this into a binary variable. So let us use `mutate()` and a new function called `case_when()`.

Think of `case_when()` as an 'if' logical statement. It allows us to make changes to a variable that are conditional on some requirement. For example, we specify that values greater than '0' (values 1 to 3) mean 'not a stranger' and values equal to '0' mean 'stranger':

```
ncvs <- mutate(ncvs,
               notstranger = case_when(
                 relationship == 0 ~ "Stranger",
                 relationship > 0 ~ "Not a stranger"))
```

To verify that we have recoded a new binary variable:

```
table(ncvs$notstranger)
```

```
##
## Not a stranger       Stranger
##          16753           6547
```

It seems that most victimisation is perpetrated by non-strangers!

2.4.2 Labels

Variables sometimes come with labels – these are very brief descriptions of the variable itself and what its values are. We are familiar with *variable* labels because of our previous activities. Now, *value* labels are very useful when we have a nominal or ordinal level variable in our dataset that has been assigned numeric values. To have a look at what are your variable and value labels, use the function `attributes()`:

```
attributes(ncvs$injured)
```

```
## $label
## [1] "Victim Sustained Injuries During Vicitmization"
##
## $format.spss
## [1] "F8.2"
##
## $display_width
## [1] 10
##
## $class
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $labels
## uninjured   injured
##         0         1
```

```
# You can also use var_label() and get_labels() too, but attributes() shows both types of labels
```

Returning to a familiar variable, `injured`, the output in the console shows that 'uninjured' is labelled '0' and 'injured' is labelled '1'. Maybe, though, you do not like the labels that are attached to the variable values. Perhaps they do not make sense or they do not help you to understand better what this variable measures. If so, we can remove and change the labels.

2.4.2.1 Activity 7: Removing labels

We return to the `injured` variable from the `ncvs` dataframe. We, again, are going to make a duplicate variable of `injured` to learn how to remove and add labels. We do this because it is good practice to leave your original variables alone in case you need to go back to them.

```
# Make a new variable that is a duplicate of the original one, but naming it 'injured_no_labels'
ncvs$injured_no_labels <- ncvs$injured

attributes(ncvs$injured_no_labels)
```

```
## $label
## [1] "Victim Sustained Injuries During Vicitmization"
##
## $format.spss
## [1] "F8.2"
```

```
##
## $display_width
## [1] 10
##
## $class
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $labels
## uninjured   injured
##         0         1
```

To remove labels, we will need to load two new packages: `labelled` and `sjlabelled`. Can you do that?

After loading the two new packages, we remove variable and value labels:

```
# Remove variable labels
ncvs$injured_no_labels <- remove_var_label(ncvs$injured_no_labels)

# Check that they were removed
var_label(ncvs$injured_no_labels)
```

```
## NULL
```

```
# Remove value labels
ncvs$injured_no_labels <- remove_labels(ncvs$injured_no_labels, labels = c(1:2))

# Check that they were removed
get_labels(ncvs$injured_no_labels)
```

```
## NULL
```

Now to add a label:

```
# Add variable label
var_label(ncvs$injured_no_labels) <- "Whether Victim Sustained Injuries"

# Check that they were added
var_label(ncvs$injured_no_labels)
```

```
## [1] "Whether Victim Sustained Injuries"
```

```
# Add value labels
ncvs$injured_no_labels <-add_labels(ncvs$injured_no_labels, labels = c(`uninjured` = 0, `injured`

# Check that they were added
get_labels(ncvs$injured_no_labels)
```

```
## [1] "uninjured" "injured"
```

Nothing to add in the googledoc so far, since Activity 4, so onto the next activity.

2.4.2.2 Note: pipes

In R, %>% represents a **pipe operator**. This is a nifty shortcut in R coding. It is in reference to René Magritte's *The Treachery of Images*. The pipe operator means that we only need to specify the data frame object once at the beginning as opposed to typing out the name of the data frame repeatedly. In all subsequent functions, the object is 'piped' through. If you were to read the code out loud, you might say 'and then' whenever you come across the pipe operator. We will use this now.

2.4.3 Subsetting

2.4.3.1 Activity 8: Ways to subset data

Through `tidyverse` functions, we can subset our data frames or vectors based on some criteria. Using the function `select()`, we can subset variables by number or name:

```
# Using select () to subset by the first two variables
ncvs_df_1 <- ncvs %>% select(1:2)
```

If we wanted to select only the variables `injured`, `weaponpresent`, and `severity`:

```
# Using select () to subset by three select variables
ncvs_df_2 <- ncvs %>% select(injured, weaponpresent, severity)
```

Using the function `slice()`, we can subset rows by number. To get only the first row:

```
# Get the first row
first_row_of_ncvs <- ncvs %>% slice(1)
```

To get more rows, you can use the 'from:to' notation. To get the first two rows, for example, you say 'from 1 to 2', that is '1:2':

```
# Get the first two rows
first_two_rows_of_ncvs <- ncvs %>% slice(1:2)
```

You can subset select rows and columns by taking `slice()` and combining it with `select()`. For example:

```
# Get the first two variables and first two rows
first_two_rows_cols <- ncvs %>% select(1:2) %>% slice(1:2)
```

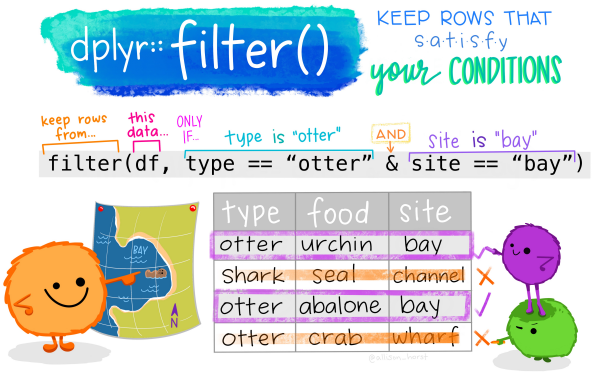


Figure 2.10: **Figure 2.11** Filter by Allison Horst

Use the `filter()` function to subset observations (i.e., rows) based on conditions. For example, we only want those for which the `injured` variable was equal to 1, so we run:

```
only_injured <- ncvs %>% filter(injured == 1)
```

These filters can be combined using conditions and (`&`) and or (`|`) except we call this subset of the data frame 'knew_of_and_injured':

```
# We want a subset called 'knew_of_and_injured' which comprises responses greater than 0 in the
knew_of_and_injured <- ncvs %>%
filter(relationship > 0 & injured == 1)
```

Say if we wanted the first five rows of `knew_of_and_injured`. How would we do that? In your group googledoc, type out the code you think will help you create a (sub-)subset `knew_of_and_injured` of its first five rows. Call this new subset 'injuredfiveknew'.

2.4.3.2 Activity 9: Subsetting, the Sequel

We now have a subset called `injuredfiveknew`. Say we only want to keep the variables `V3014` (age) and `V3018` (sex). How would you make an object that only contains these two variables from `injuredfiveknew`?

Recall that you would need to use the function `select()`to select variables. But in this example, instead of inserting ' : ' like in the previous code, you would need to insert a ','. Understanding what ':' means and viewing the order of the variables in `injuredfiveknew` will give you insight into why.

In your group googledoc sheet, write out the code that you would use to do so. Name this new object that contains the two variables `five_ageandincome`

2.5 SUMMARY

Today you were further introduced to **tidyverse** packages that helped you to tidy your data. First, we learned to put our work into a **project** and then how to import data using a package called `haven`. Whenever we specify a data frame, we learned a nifty short-cut: the **pipe operator** - `%>%` - which allows us to specify the data frame only once when we code.

Our three main topics today had to do with helping us tidy. There was a lot of **data wrangling** too. One topic were the variables themselves where we learned about the **factor** and **numeric** classes, and how to make and **recode** new variables. Two, we learned how to remove and add variable and value labels so that we can understand what our variables are measuring. Three, we then learned to subset our data, whereby we make new dataframes that include only

the columns – variables – or rows – observations – we want. We tidied our data using the TIDYVERSE WAY!

P.S. Well done today, to get through all this. What you are learning now will serve as the building blocks for your later data analysis, and we recognise it is all new and scary. But keep practicing, and you will get the hang of this in no time! And of course: don't forget to do your homework!

2.5.1 Answers to activities (if applicable)

- 1. N/A
- 2. 23,969 entries and 47 columns
- 3. numeric and class(ncvs$weaponpresent)
- 4. N/A
- 5. N/A
- 6. N/A
- 7. N/A
- 8. injuredfiveknew <- KnewOfandInjured %>% slice(1:5)
- 9. five_ageandincome <- injuredfiveknew %>% select(4,6)

Chapter 3

Data Visualization

Layers and Graphs

Learning Outcomes:

- Understand what the layered approach to Grammar of Graphics and its corresponding package `ggplot2` are
- Learn how to use `ggplot2` to create different visualizations

Today's Learning Tools:

***Total number of activities*: 12**

Data:

- Official crime data from England and Wales

Packages:

- `ggplot2`
- `ggthemes`
- `readr`
- `here`

Functions introduced (and packages to which they belong)

- `aes()` : Mapping aesthetics to variables (`ggplot2`)
- `as.factor()` : Convert to factor, including specifying levels (`base R`)
- `facet_wrap()` : Facet graphics by one or more variables (`ggplot2`)
- `geom_histogram()` : Geometry layer for histograms (`ggplot2`)
- `geom_line()` : Geometry layer for line charts (`ggplot2`)
- `geom_point()` : Geometry layer for scatterplots (`ggplot2`)
- `ggplot()` : Initialize a ggplot graphic, i.e., specify data, aesthetics (`ggplot2`)
- `labs()` : Specify labels for ggplot object, e.g., title, caption (`ggplot2`)
- `range()` : Compute the minimum and maximum values (`base R`)
- `read_csv()` : Read in comma separated values file (`readr`)
- `scale_color_brewer()` : Default color scheme options (`ggplot2`)
- `scale_color_viridis_d()` : Colour-blind-considerate palettes from `viridis` package (`ggplot2`)
- `theme()` : Customise ggplot graphics (`ggplot2`)
- `theme_economist ()` : Changes the graphic to resemble ones found in *The Economist* (`ggthemes`)
- `theme_minimal()` : Default minimalist theme for `ggplot` graphics (`ggplot2`)

3.1 Grammar of Graphics

Today we learn the basics of data visualization in `R` using a package called `ggplot2` within the `tidyverse`, one of the most popular packages for making high-quality, reproducible graphics. The framework on which `ggplot2` is based derives from Hadley Wickham's (2010) A Layered Grammar of Graphics.

Wickham advances the work of *Grammar of Graphics* by proposing a layered approach to describe and build graphics in a structured manner. Layers, according to Wickham, are used to describe the basic features that underlie any graphic or visualization. Every layer comprises five parts: data, aesthetics, statistical transformation (stat), geometric object (geom), and position adjustment (position).

The three primary ones we focus on to ease you into this, however, are (1) the data, (2) the aesthetics, and (3) the geometric objects:

1. Data: usually the data frame with tidy rows and columns.
2. Aesthetics: the visual characteristics that represent the data, or variables. At its simplest, these could be an x- and y-axis, and can extend to other aesthetics like colour, size, and shape, which are mapped to variables.

3. Geometric objects: also known as 'geoms' and represent the data points in the visualization, such as dots or lines.

Table 3. Simple dataset with variables mapped into aesthetic space.

x	*y*	Shape
25	11	circle
0	0	circle
75	53	square
200	300	square

Figure 3.1: **Figure 3.1** Visualize Table 3

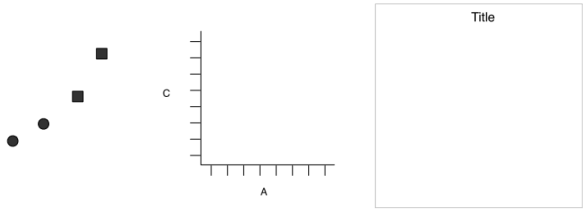


Figure 1. Graphics objects produced by (from left to right): geometric objects, scales and coordinate system, plot annotations.

Figure 3.2: **Figure 3.2** Using the *Grammar of Graphics* approach to convert elements of Table 3 into its visual components

Why is this important? Visualizing your data helps you to have a better understanding of them before moving on to more advanced and complex analyses. Numbers themselves can be deceptive and complicated, so by visualizing them, you can identify any patterns or anomalies.

3.1.1 Activity 1: Getting Ready

Before we 'layer up', let us do the following:

1. Open up your existing `R`project like you had learned last week
2. Load the `ggplot2` and `readr` packages using the `library()` function. If you have not already installed these packages, you will need to do so first, using the `install.packages()` function. *If you are unclear about the difference between these two functions, see 1.3.3 in Lesson 1 of this online tutorial site or ask now!*

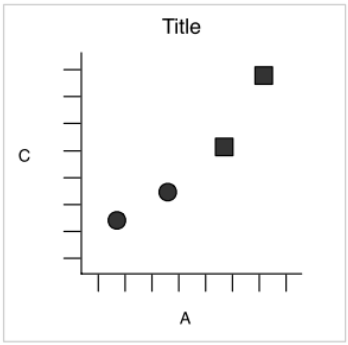


Figure 2. The final graphic, produced by combining the pieces in Figure 1.

Figure 3.3: **Figure 3.3** Integrating those visual parts of Table 3 into a graphic

3. Download the first two datasets (gmp_2017.csv and gmp_monthly_2017.csv) from Blackboard, in the subfolder 'Data for this week' in the Week 3 learning materials. The datasets contain crime data from Greater Manchester, England. Load these two datasets using the `read_csv()` function into two separate data frames. Respectively name the data frames `burglary_df` and `monthly_df` by using the `<-` assignment operator.

Make sure the datasets are saved in the sub-folder called 'Datasets' that you created last week *inside* your project directory/ folder. If this is the case, we can use the `here()` function to tell `R` where exactly to locate the datasets. (See lecture short 3.1 'The R Prep routine' on Blackboard.)

Create first data frame object, `burglary_df` and load into it the 'gmp_2017.csv' data file:

```
# gmp_2017.csv
burglary_df <- read_csv(here("Datasets", "gmp_2017.csv"))
```

Now your turn to make the second data frame. Create the second data frame, `monthly_df` and load into it the 'gmp_monthly_2017.csv' data file.

You should have two data frames in your environment now. How many observations (rows) and variables (columns) are in `burglary_df`? How about in `monthly_df`?

3.2 ggplot2

This week we will be learning how to use `R` to create various exciting data visualizations. You may remember some of the key principles of data visualizations from last semester: graphics should be truthful,functional, beautiful, insightful, and enlightening. Also, accessible!

Here is a recent powerful visualization from The New York Times about deaths from COVID-19 in the USA (Figure 3.4): https://twitter.com/beccapfoley/status/1363338950518206472

`R` has amazing data visualization qualities. It is increasingly used by professionals and academics to create graphics that embody those principles of data visualization.

For example, read this article about how the BBC uses `R` to create their graphics: How the BBC Visual and Data Journalism team works with graphics in R (See Figure 3.5)

As you learn to create graphs in `ggplot2`, you will see that the code reflects building these graphs as layers. This will take getting used to if you are only familiar with building graphs from a template in Excel. You can also reuse chunks of code to create different kinds of graphics or identical ones in other datasets.

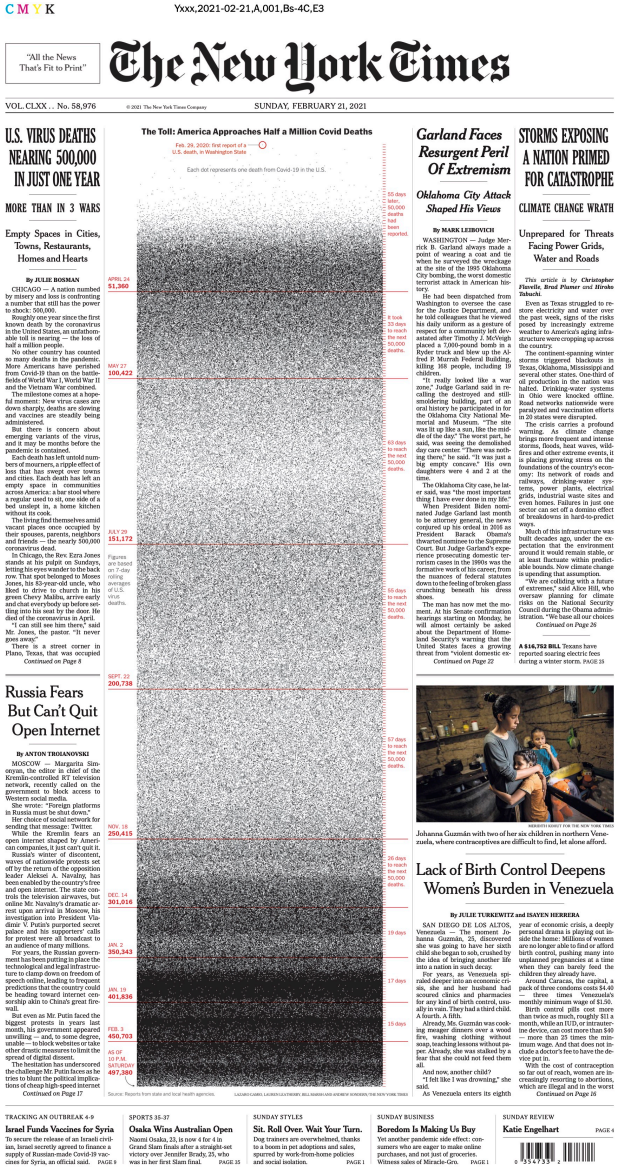
Today we focus on the `ggplot2` package to learn our three substantive topics: **layers**, **graphs for categorical data**, and **graphs for numeric data**.

3.2.0.1 Activity 2: Getting to Know New Data

Before we dive into those topics, we need to get into the habit of getting to know our data. Below is a brief 'data dictionary' of the variables from both of our data:

A. **burglary_df** has nine variables:

- *LSOAcode*: LSOA stands for Lower Super Output Area, and is a statistical unit of geography. It represents neighbourhoods that each contain 300-400 households. Essentially, LSOA means neighbourhood.
- *burglary_count*: the number of burglaries in each LSOA in 2017
- *LAname*: the name of the Local Authority into which the LSOA falls
- *IMD score*: the IMD refers to the Index of Multiple Deprivation. This measure combines seven domains of deprivation into one index: (1) income, (2) employment, (3) education, (4) health, (5) crime, (6) barriers to housing or services, and (7) living environment. It is the score that a particular LSOA has on this measure. You can read more about the measure here: IMD infographic



C M Y K Yxxx,2021-02-21,A,001,Bs-4C,E3

"All the News That's Fit to Print"

The New York Times

VOL. CLXX .. No. 58,976 © 2021 The New York Times Company SUNDAY, FEBRUARY 21, 2021

U.S. VIRUS DEATHS NEARING 500,000 IN JUST ONE YEAR

MORE THAN IN 3 WARS

Empty Spaces in Cities, Towns, Restaurants, Homes and Hearts

By JULIE BOSMAN

CHICAGO — A nation numbed by misery and loss is confronting a number that still has the power to shock: 500,000.

Roughly one year since the first known death by the coronavirus in the United States, an unfathomable toll is nearing — the loss of half a million people.

No other country has counted so many deaths in the pandemic. More Americans have perished from Covid-19 than on the battlefields of World War I, World War II and the Vietnam War combined.

The milestone comes at a hopeful moment: New virus cases are down sharply, deaths are slowing and vaccines are steadily being administered.

But there is concern about emerging variants of the virus, and it may be months before the pandemic is contained.

Each death has left untold numbers of mourners, a ripple effect of loss that has swept over towns and cities. Each death has left an empty space in communities across America: a bar stool where a regular used to sit, one side of a bed unslept in, a home kitchen without its cook.

The living find themselves amid vacant places once occupied by their spouses, parents, neighbors and friends — the nearly 500,000 coronavirus dead.

In Chicago, the Rev. Ezra Jones stands at his pulpit on Sundays, letting his eyes wander to the back row. That spot belonged to Moses Jones, his 83-year-old uncle, who liked to drive to church in his green Chevy Malibu, arrive early and chat everybody up before settling into his seat by the door. He died of the coronavirus in April.

"I can still see him there," said Mr. Jones, the pastor. "It never goes away."

There is a street corner in Plano, Texas, that was occupied

Continued on Page 8

Russia Fears But Can't Quit Open Internet

By ANTON TROIANOVSKI

MOSCOW — Margarita Simonyan, the editor in chief of the Kremlin-controlled RT television network, recently called on the government to block access to Western social media.

She wrote: "Foreign platforms in Russia must be shut down."

Her choice of social network for sending that message: Twitter.

While the Kremlin fears an open internet shaped by American companies, it just can't quit it.

Russia's winter of discontent, waves of nationwide protests set off by the return of the opposition leader Aleksei A. Navalny, has been enabled by the country's free and open internet. The state controls the television airwaves, but online Mr. Navalny's dramatic arrest upon arrival in Moscow, his investigation into President Vladimir V. Putin's purported secret palace and his supporters' calls for protest were all broadcast to an audience of many millions.

For years, the Russian government has been putting in place the technological and legal infrastructure to clamp down on freedom of speech online, leading to frequent predictions that the country could be heading toward internet censorship akin to China's great firewall.

But even as Mr. Putin faced the biggest protests in years last month, his government appeared unwilling — and, to some degree, unable — to block websites or take other drastic measures to limit the spread of digital dissent.

The hesitation has underscored the challenge Mr. Putin faces as he tries to blunt the political implications of cheap high-speed internet

Continued on Page 17

The Toll: America Approaches Half a Million Covid Deaths

Feb. 29, 2020: first report of a U.S. death, in Washington State

Each dot represents one death from Covid-19 in the U.S.

APRIL 24 **51,360**

MAY 27 **100,422**

JULY 29 **151,172**

SEPT. 22 **200,738**

NOV. 18 **250,415**

DEC. 14 **301,016**

JAN. 2 **350,343**

JAN. 19 **401,636**

FEB. 3 **450,703**

AS OF 10 P.M. SATURDAY **497,380**

55 days later, 50,000 deaths had been reported.

It took 53 days to reach the next 50,000 deaths.

63 days to reach the next 50,000 deaths.

66 days to reach the next 50,000 deaths.

57 days to reach the next 50,000 deaths.

36 days to reach the next 50,000 deaths.

19 days

17 days

15 days

Figures are based on 7-day rolling averages of U.S. virus deaths.

Source: Reports from state and local health agencies. LAZARO GAMIO, LAUREN LEATHERBY, BILL MARSH AND ANDREW ROSENBERG/THE NEW YORK TIMES

Garland Faces Resurgent Peril Of Extremism

Oklahoma City Attack Shaped His Views

By MARK LEIBOVICH

WASHINGTON — Judge Merrick B. Garland always made a point of wearing a coat and tie when he surveyed the wreckage at the site of the 1995 Oklahoma City bombing, the worst domestic terrorist attack in American history.

He had been dispatched from Washington to oversee the case for the Justice Department, and he told colleagues that he viewed his daily uniform as a gesture of respect for a community left devastated after Timothy J. McVeigh placed a 7,000-pound bomb in a Ryder truck and blew up the Alfred P. Murrah Federal Building, killing 168 people, including 19 children.

"It really looked like a war zone," Judge Garland said in recalling the destroyed and still-smoldering building, part of an oral history he participated in for the Oklahoma City National Memorial and Museum. "The site was lit up like a sun, like the middle of the day." The worst part, he said, was seeing the demolished day care center. "There was nothing there," he said. "It was just a big empty concave." His own daughters were 4 and 2 at the time.

The Oklahoma City case, he later said, was "the most important thing I have ever done in my life."

When President Biden nominated Judge Garland last month to be attorney general, the news conjured up his ordeal in 2016 as President Barack Obama's thwarted nominee to the Supreme Court. But Judge Garland's experience prosecuting domestic terrorism cases in the 1990s was the formative work of his career, from the nuances of federal statutes down to the feeling of broken glass crunching beneath his dress shoes.

The man has now met the moment. At his Senate confirmation hearings starting on Monday, he will almost certainly be asked about the Department of Homeland Security's warning that the United States faces a growing threat from "violent domestic ex-

Continued on Page 22

STORMS EXPOSING A NATION PRIMED FOR CATASTROPHE

CLIMATE CHANGE WRATH

Unprepared for Threats Facing Power Grids, Water and Roads

This article is by ***Christopher Flavelle, Brad Plumer*** *and* ***Hiroko Tabuchi.***

Even as Texas struggled to restore electricity and water over the past week, signs of the risks posed by increasingly extreme weather to America's aging infrastructure were cropping up across the country.

The continent-spanning winter storms triggered blackouts in Texas, Oklahoma, Mississippi and several other states. One-third of oil production in the nation was halted. Drinking-water systems in Ohio were knocked offline. Road networks nationwide were paralyzed and vaccination efforts in 20 states were disrupted.

The crisis carries a profound warning. As climate change brings more frequent and intense storms, floods, heat waves, wildfires and other extreme events, it is placing growing stress on the foundations of the country's economy: Its network of roads and railways, drinking-water systems, power plants, electrical grids, industrial waste sites and even homes. Failures in just one sector can set off a domino effect of breakdowns in hard-to-predict ways.

Much of this infrastructure was built decades ago, under the expectation that the environment around it would remain stable, or at least fluctuate within predictable bounds. Now climate change is upending that assumption.

"We are colliding with a future of extremes," said Alice Hill, who oversaw planning for climate risks on the National Security Council during the Obama administration. "We base all our choices

Continued on Page 26

A $16,752 BILL Texans have reported soaring electric fees during a winter storm. PAGE 15

MEREDITH KOHUT FOR THE NEW YORK TIMES

Johanna Guzmán with two of her six children in northern Venezuela, where contraceptives are difficult to find, let alone afford.

Lack of Birth Control Deepens Women's Burden in Venezuela

By JULIE TURKEWITZ and ISAYEN HERRERA

SAN DIEGO DE LOS ALTOS, Venezuela — The moment Johanna Guzmán, 25, discovered she was going to have her sixth child she began to sob, crushed by the idea of bringing another life into a nation in such decay.

For years, as Venezuela spiraled deeper into an economic crisis, she and her husband had scoured clinics and pharmacies for any kind of birth control, usually in vain. They had a third child. A fourth. A fifth.

Already, Ms. Guzmán was cooking meager dinners over a wood fire, washing clothing without soap, teaching lessons without paper. Already, she was stalked by a fear that she could not feed them all.

And now, another child?

"I felt like I was drowning," she said.

As Venezuela enters its eighth year of economic crisis, a deeply personal drama is playing out inside the home: Millions of women are no longer able to find or afford birth control, pushing many into unplanned pregnancies at a time when they can barely feed the children they already have.

Around Caracas, the capital, a pack of three condoms costs $4.40 — three times Venezuela's monthly minimum wage of $1.50.

Birth control pills cost more than twice as much, roughly $11 a month, while an IUD, or intrauterine device, can cost more than $40 — more than 25 times the minimum wage. And that does not include a doctor's fee to have the device put in.

With the cost of contraception so far out of reach, women are increasingly resorting to abortions, which are illegal and in the worst

Continued on Page 16

0 75747 75 2

Figure 3.4: **Figure 3.4** NYT visualization of COVID-19 deaths in USA

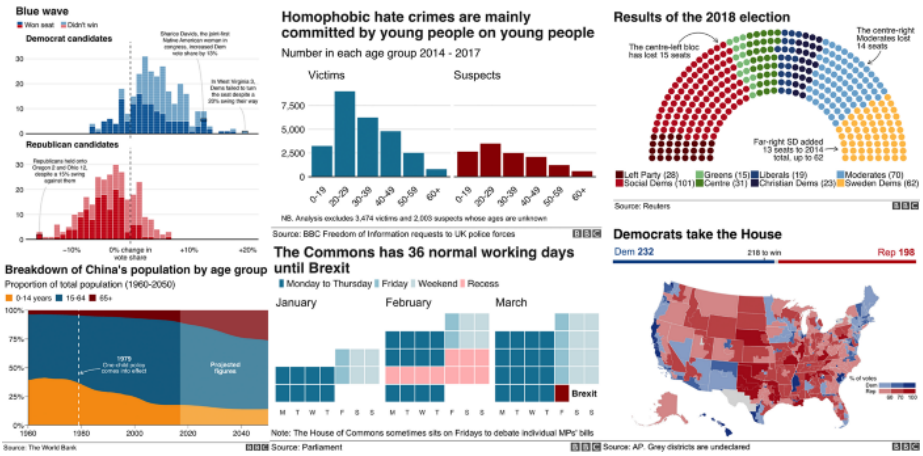


Figure 3.5: **Figure 3.5** How the BBC Visual and Data Journalism team works with graphics in R

- *IMDrank*: another version of the IMD except it ranks all 32,844 LSOAs from highest to lowest deprivation.
- *IMDdeci*: indicates in what decile of deprivation the LSOA falls.
- *incscore*: score for income deprivation; this is one of the seven indicators comprising the IMD. Measures the proportion of the population experiencing deprivation relating to low income.
- *LSOAname*: the name of that particular LSOAs
- *pop*: the population size of LSOAs

B. **monthly_df** has three variables:

- *Month*: month of the year whereby 1 = January, 2 = February… and 12 = December
- *crime_type*: type of crime (for police.uk categories, see: download police.uk data categories vs home office offence codes)
- *n*: number of each type of crime for each month in 2017

Have a look at both datasets. Let us use the function `View()` first. What do you think are the levels of measurement for each variable? Make a note of this.

Now use the the function `class()` to look at how `R` treats each variable for each dataset. How does this match with what you think is the level of measurement for each variable? Keep this in mind!

Notice that although `monthly_df` has only several variables, there is a lot of information in this data frame. There are counts for various different crimes for each month of the year. Because of this, the data are in *long format*. This is a

useful and typical format for longitudinal data because our time variable (i.e., months) is contained in one variable. This will make it easier to create graphics showing change over time.

In addition, when you read a good quantitative research paper, its methods section will state the number of cases analysed with 'n =', whereby 'n' represents 'number of cases'. This is known as sample size. For example, if the sample size of a study is two-hundred, the methods section would state it as 'n= 200'.

In your group google doc, type out the following: the number of observations for each data frame. Type out these values following 'n ='. In addition, state what you think the unit of analysis is in each data frame.

3.3 Today's 3

Now that we have gotten to know our data, onto our three main topics: **layers**, **graphs for nominal and ordinal data**, and **graphs for interval and ratio data**.

3.3.1 Layers

Layers often relate to one another and have similar features. For example, they may share the same underlying data on which they are built. An example is this scatterplot overlayed with a smoothed regression line to describe the relationship between two variables:

From: https://cfss.uchicago.edu/notes/grammar-of-graphics/

One layer is the scatterplot itself and it contains the necessary components (e.g., data, aesthetics, and geom). The second layer is the smoothing line. Another example of multiple layers is this one that uses two different but related datasets in one graphic:

From: http://applied-r.com/building-layered-plots/

The advantage of the layered approach is its ease for the user and the developer of these statistical graphics. As a user, you can repeatedly update a graphic, changing one feature at a time instead of having to work with the overall graphic; as a developer, you can add more components to your graphic while being able to use what you have already. The idea is that you can tweak your graphic by component or layer while leaving the rest of your graphic intact. This is in constrast to changing the entire graphic just to tweak a small detail.

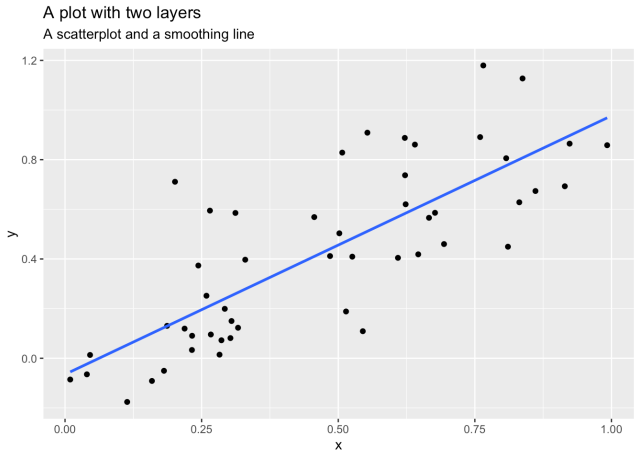


Figure 3.6: **Figure 3.6** A graphic with two layers

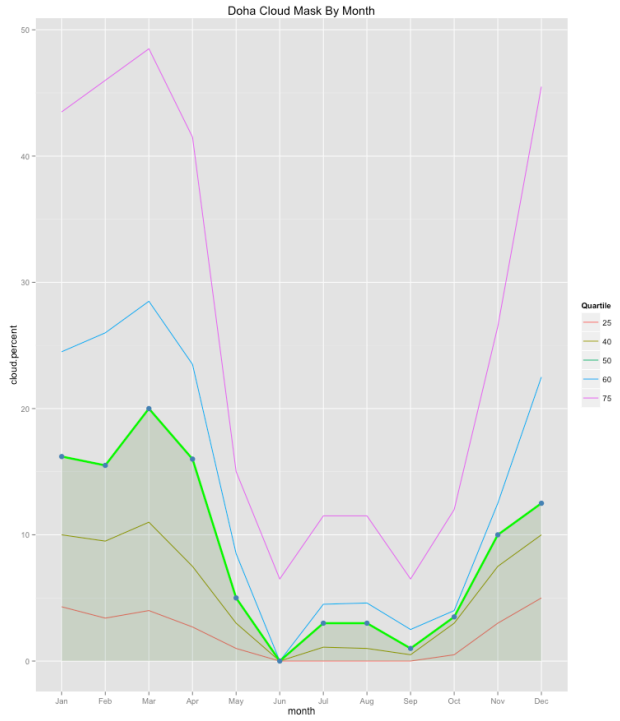


Figure 3.7: **Figure 3.7** Different data sets and aesthetics defined by layer

To understand a layer and how it is generated in `ggplot2`, we begin with a scatterplot. This is a graph that shows the relationship between two continuous variables – in other words, between two numeric variables.

For example, we may be interested to find out whether there is a relationship between area-level income deprivation and burglaries. Are areas with higher income deprivation scores burgled more? Perhaps there is less money to spend on security systems, so there is easier access to homes in these areas. Or is it areas with lower income deprivation scores that are burgled more? Maybe because these areas are wealthier so there are more valuable goods to take. To visualize this relationship, we build a scatterplot using the layered approach.

3.3.1.1 Activity 3: Using ggplot2 to create a scatterplot

Our research question, therefore, is: *What is the relationship between income and burglary counts?*

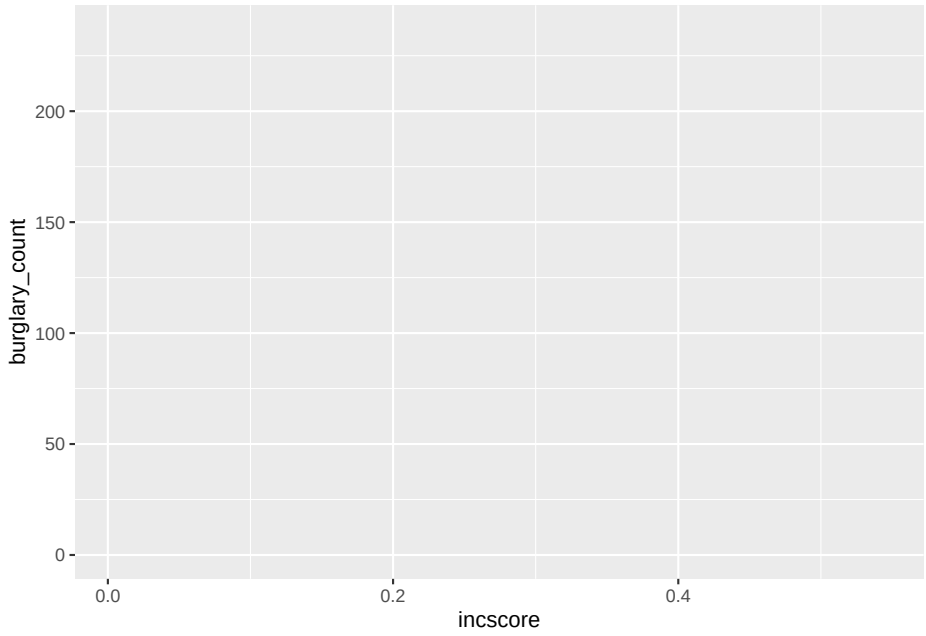
Following the grammar of graphics layered approach, let us first specify the data component. To do this, we take the `ggplot()` function, and inside it, we put our data frame object, which contains our data. To let the function know this is the data, we can also write `data =` before it:

```
# If there is an error about function missing, has the package ggplot2 been installed
ggplot(data = burglary_df)
```

The *Plots* window will appear and will be blank, like what you see above. We have told `R` what data we want to plot, but we have yet to specify what variables we want across our x and y axes, or other fields. To do this, we will need *aesthetics*, the second component of the layer.

Recall that aesthetics are about mapping the visual properties. As our research question is about the relationship between income (`incscore`) and burglary count (`burglary_count`), we will map these variables to the x and y axes. We specify the aesthetics inside the `aes()` function (i.e.,**aes**thetics), indicating which variable we want on the x axis (`x =`) and which one on the y axis (`y =`).

```
ggplot(data = burglary_df, mapping = aes(x = incscore, y = burglary_count))
```

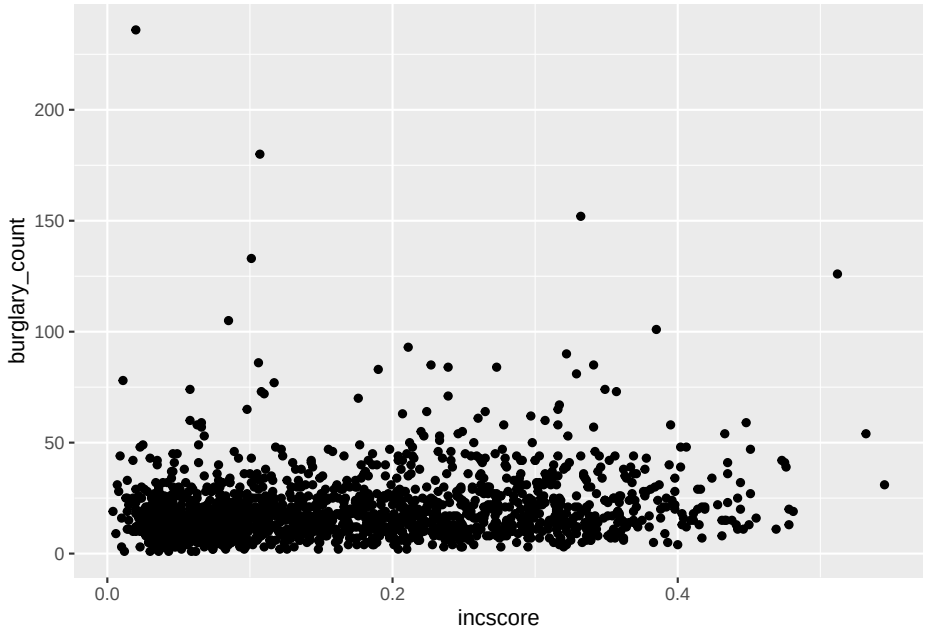


You will now see an update in the *Plots* window! It shows the axis and the axes labels. `R` now knows that we want to plot the income variable on our x axis and the burglary count variable on the y. The labels are applied automatically, but you can change these, which we will soon learn.

Let us learn how to finally put our data on the plot. To do this, we must assign a **geometry**.

This last component is the geometric object, and is what also makes the layer in `R` code. It has numerous functions, each beginning with the argument `geom_`. To create a scatterplot, we use the function `geom_point()`:

```
ggplot(data = burglary_df, mapping = aes(x = incscore, y = burglary_count)) +
  geom_point()
```



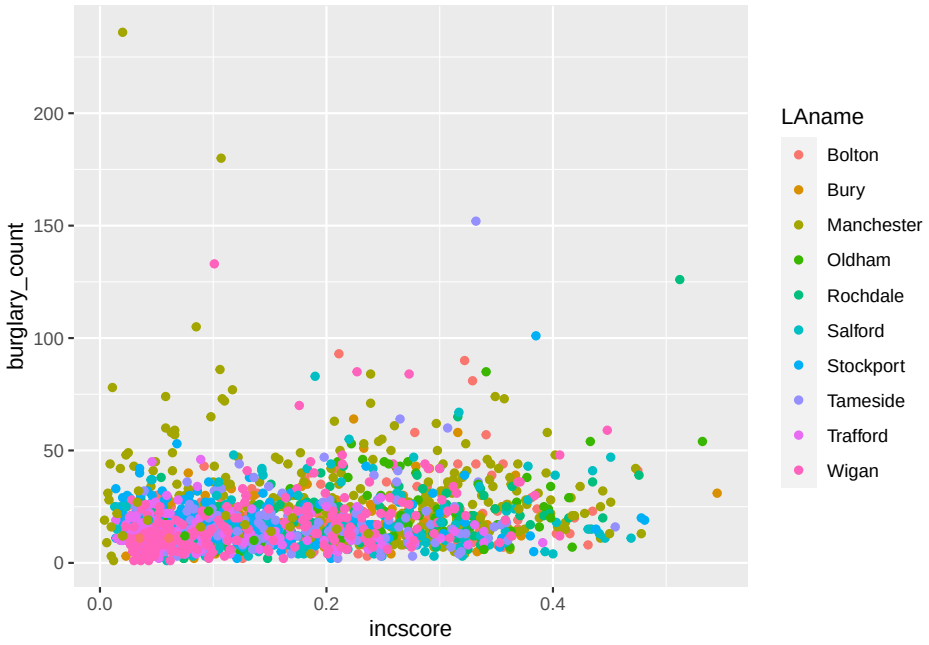

```
# We are saying, 'Hey R, may you please make me a graphic using the data "burglary_df"
```

Now the scatterplot in all its glory appears. What can you say about the relationship between deprivation and burglary victimisation? In your group google doc, type out what you think is the relationship between these two variables.

3.3.1.2 Activity 4: Adding a Third Variable to a Scatterplot

The real power in layered graphics comes from the ability to keep piling on the layers and tweaking components that make up the layers. For example, it would be helpful if we included other information like how local authorities may play a role in this relationship. Maybe income is positively associated with burglaries in one local authority, such as Bolton, but not in another, such as Bury. To explore this, we might want to colour each point by the variable `LAname` using the `colour` argument:

```
ggplot(data = burglary_df, mapping = aes(x = incscore, y = burglary_count, colour = LAname)) +
  geom_point()
```



In your group google doc, state what you observe from this new visual. What sort of relationship does it seem to show between burglary counts and income? Do you think local authority plays a role? Do note the default colours, and think back to what we learned about accessibility last semester. In the next activity, we will address this!

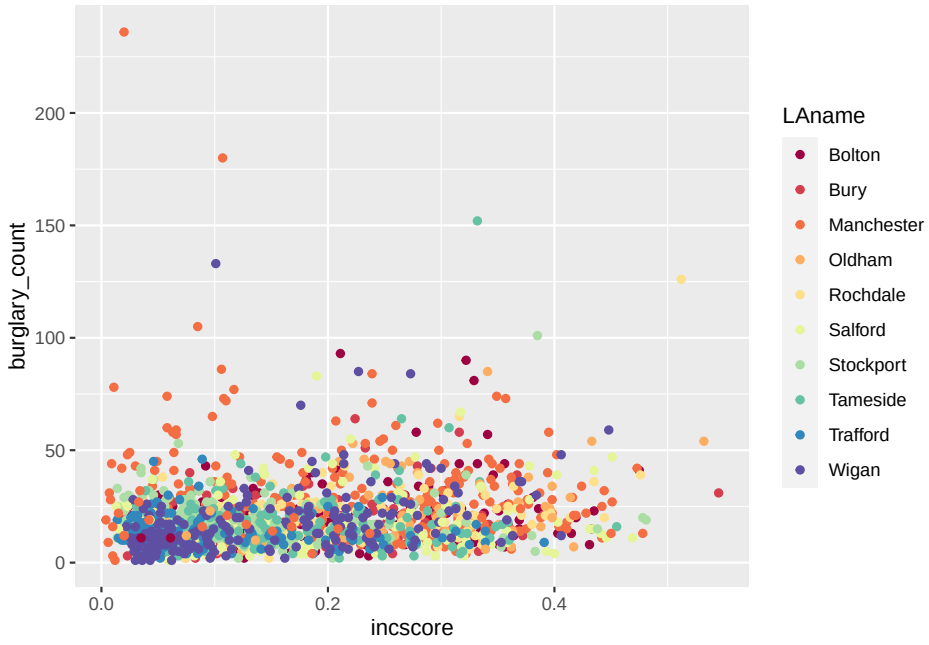
As previously mentioned, a number of other functions are available for the `geom_` function such as shape, linetype, size, fill, and alpha (transparency). But these depend on the geometry being used and the class of variable being mapped to the aesthetic. For example, try the shape aesthetic for `LAname`, which will vary the shape of each point according to local authority (hint: replace `colour =` with `shape =`). A warning message appears – what is it telling you? How can you see this reflected on the plot?

This relates to the grammar of graphics philosophy on how people understand information, and `ggplot2` will warn you when it thinks the graphic is difficult to interpret or, worse, misleading.

3.3.1.3 Activity 5: Colours!

The colour palette that appeared for our previous scatterplot is a default one, and there are a number of default colour palettes available. We can specify a colour palette using an additional layer with the function `scale_color_brewer()`. For example:

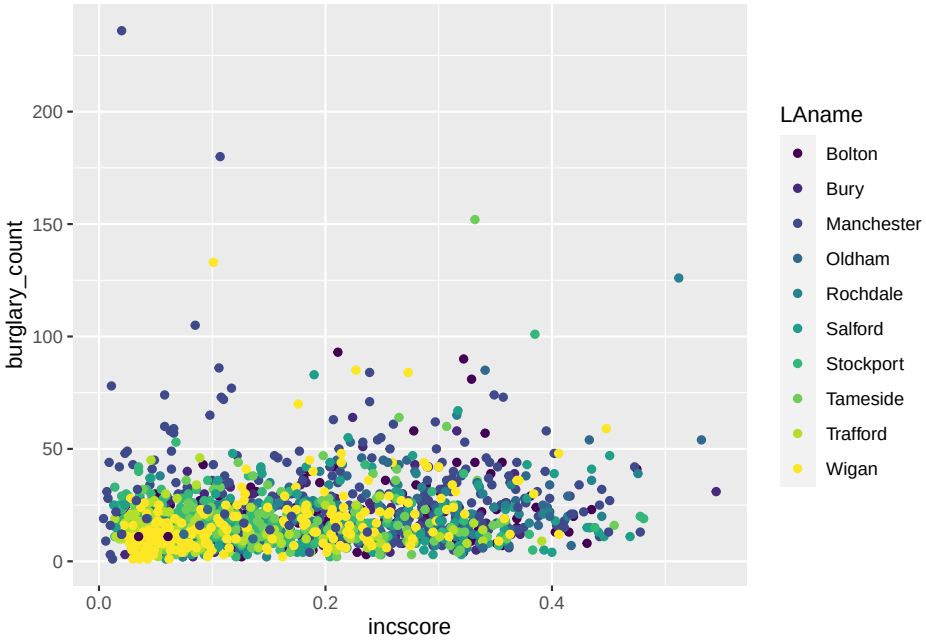
```
ggplot(data = burglary_df, mapping = aes(x = incscore, y = burglary_count, color = LAna
  geom_point() +
  scale_colour_brewer(palette = "Spectral")
```



Does that look familiar? Last semester we used the website colorbrewer2.org/. The `scale_colour_brewer()` function takes palettes from this resource, and applies them to your charts.

In addition, the `viridis` package has a number of colour palettes that consider colour blindness where readers are unable to differentiate colours on the graphic. It is already integrated into `ggplot2`, so there is no need to install it separately. For example, we take the previous code and replace the last line with a viridis colour palette for categorical variables:

```
ggplot(data = burglary_df, mapping = aes(x = incscore, y = burglary_count, color = LAna
  geom_point() +
  scale_color_viridis_d() # or scale_color_viridis_c() for a continuous variable
```

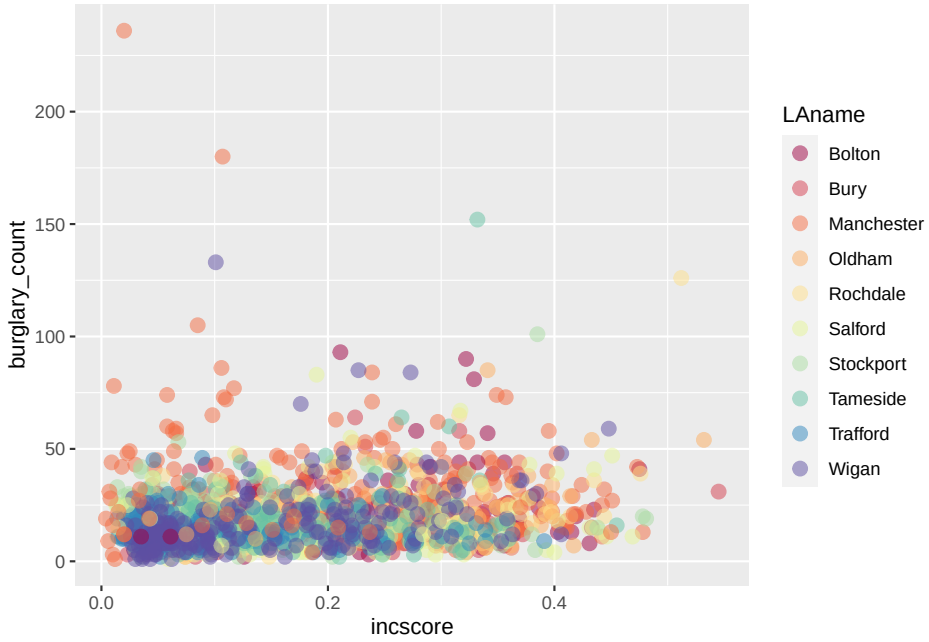


Now we have built up a graph to visualise the bivariate relationship between income deprivation score and burglary count per neighbourhood. Then, we introduced a third variable (i.e., multi-variable analysis) and brought in colour to add to the local authority. In the upcoming exercises, we will learn how to tweak the graphics so to make them even more our own!

3.3.1.4 Activity 6: Sizes & Transparency

Recall that the geometric object is about the data points themselves. You can change their appearance within the `geom_point` function. To increase the size of the points and make them transparent, we use the size and alpha arguments. The default for size is 1, so anything lower than 1 will make points smaller while anything larger than 1 will make the points bigger. The default for alpha is also 1, which means total opaqueness, so you can only go lower to make things more transparent. For example:

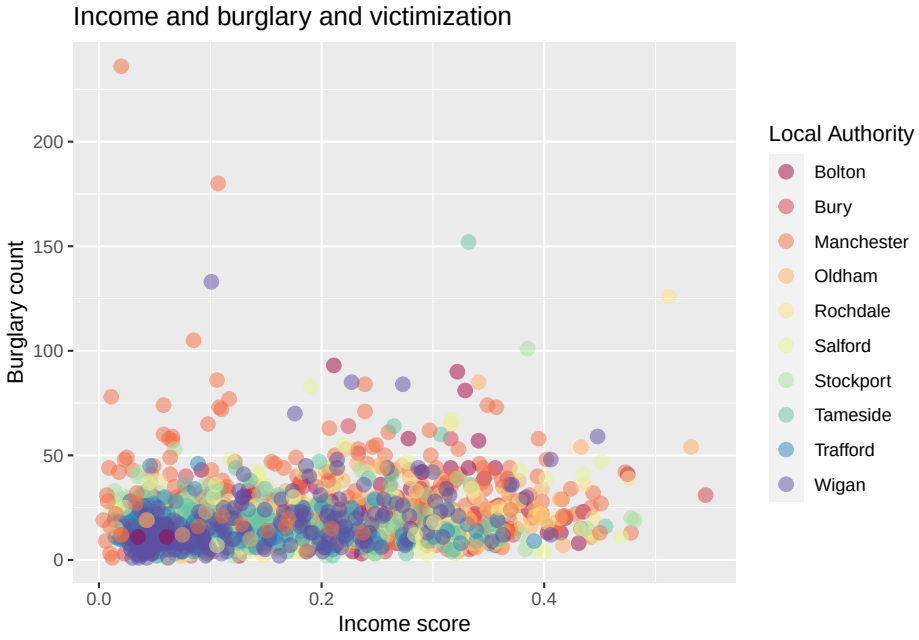
```
# Below, we want our data points to be a bit bigger ( 3 ) and more transparent ( 0.5 ) than the d

ggplot(data = burglary_df, mapping = aes(x = incscore, y = burglary_count, color = LAname)) +
  geom_point(size = 3, alpha = 0.5) +
  scale_color_brewer(palette = "Spectral")
```



3.3.1.5 Activity 7: Labels

The graphic is looking good, but the labels will need tweaking if we do not like the default ones, which are our variable names. We use the `labs()` function to change the labels for the x and y axes as well as the graphic title:

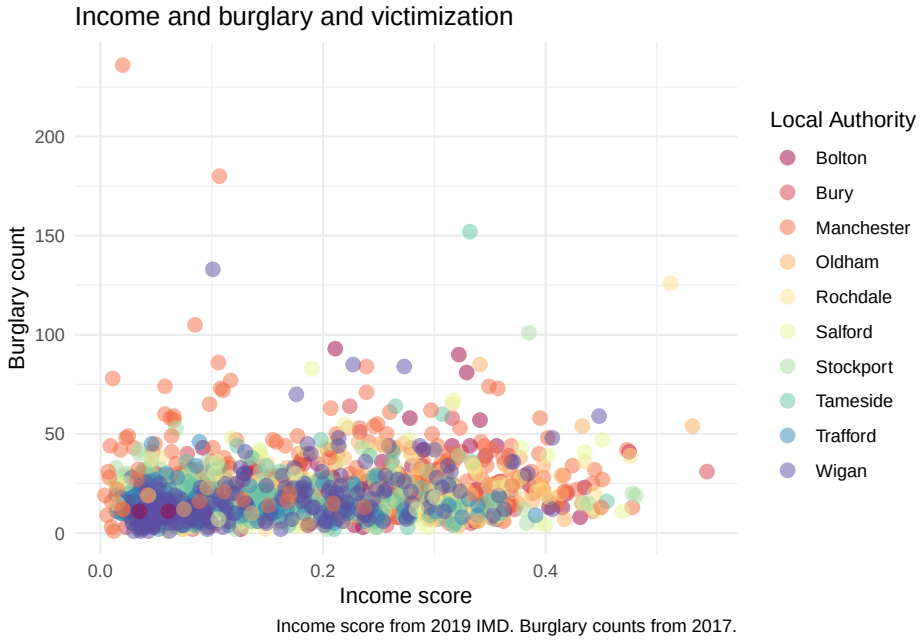
```
ggplot(data = burglary_df, mapping = aes(x = incscore, y = burglary_count, color = LAna
  geom_point(size = 3, alpha = 0.5) +
  scale_color_brewer(palette = "Spectral") +
  labs(x = "Income score", y = "Burglary count", title = "Income and burglary and victi
```



3.3.1.6 Activity 8: Built-in Themes

A final touch to your graphic can be the use of **themes**. These change the overall appearance of your graphic. The default theme we have for our graphic is `theme_gray()`, but we can go for a minimalist look by using `theme_minimal()`. There are a number of these customised themes like one that is inspired by *The Economist* (`theme_economist()`). To use special themes like *The Economist* install the package `ggthemes` and load it.

```
# Changing graphic to the minimal theme with the last line of code
ggplot(data = burglary_df, mapping = aes(x = incscore, y = burglary_count, color = LAname)) +
  geom_point(size = 3, alpha = 0.5) +
  scale_color_brewer(palette = "Spectral") +
  labs(x = "Income score", y = "Burglary count", title = "Income and burglary and victimization",
  theme_minimal()
```



3.3.2 Graphs for Categorical Data

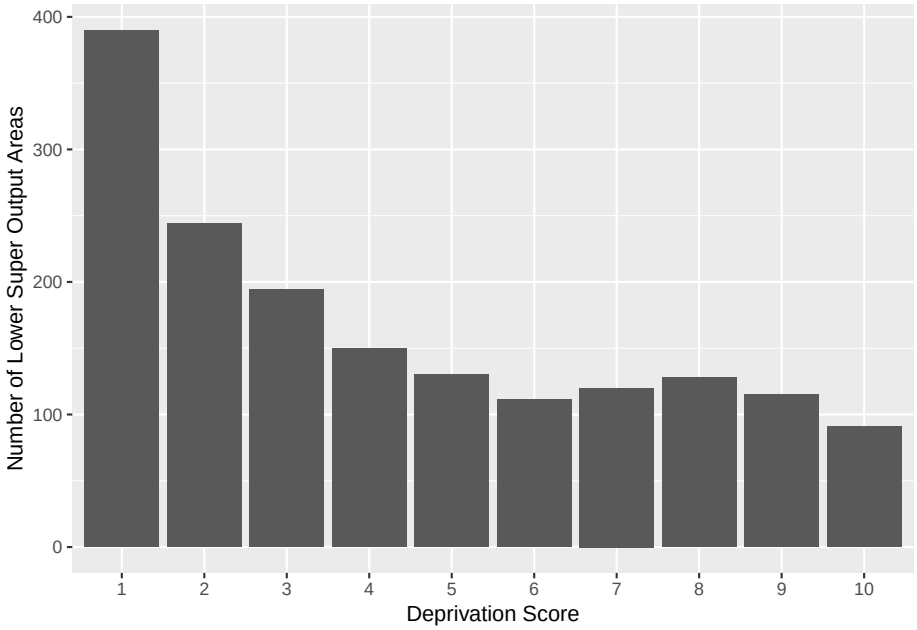
3.3.2.1 Activity 9: Bar graphs

To explore the count distribution of categorical variables, use the bar graph. We may be interested to know the number of neighbourhoods falling into each indice of the multiple deprivation (IMD) decile, a key indicator of criminality. Those in decile 1 are within the most deprived 10% of LSOAs nationally; those in decile 10 are considered within the least deprived 10% of LSOAs nationally.

The variable of interest, `IMDdeci`, is classed as numeric but we will need to treat it as a factor or else the default x-axis will include non-integer values like 7.5 .

Instead of making a new object, `ggplot` can calculate these frequencies by only specifying the x-axis and using the function `as.factor()`:

```
ggplot(data = burglary_df) +
  geom_bar(mapping = aes(x = as.factor(IMDdeci))) + # convert IMDdeci to a factor with
  labs(x = "Deprivation Score", y = "Number of Lower Super Output Areas")
```

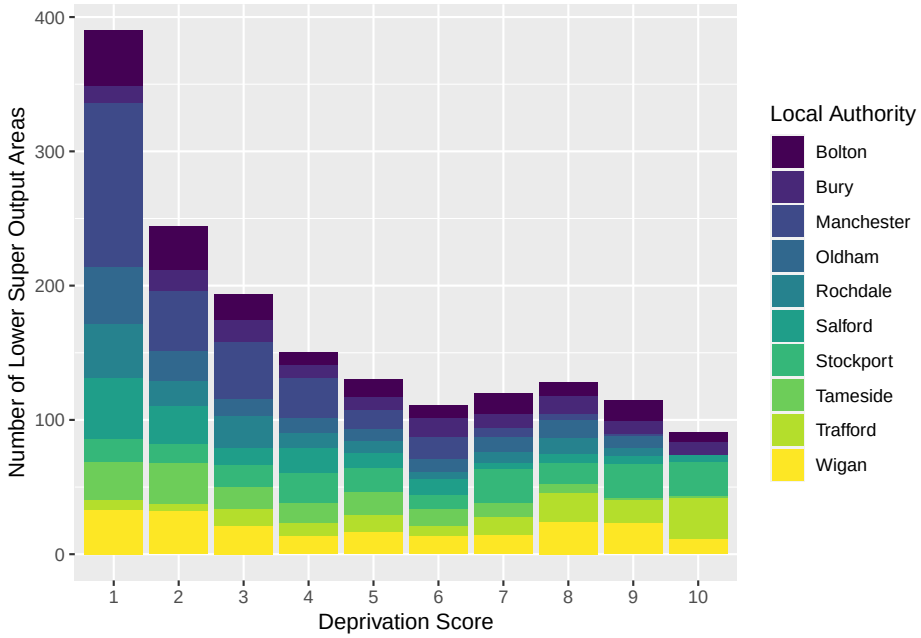


3.3.2.2 *Grouped Bar Graphs*

If we are interested in the distribution of deprivation like the previous bar graph but would like to see it by local authority, we can use the grouped bar graph, and specify the colour with the `fill =` parameter inside the `aes()` function.

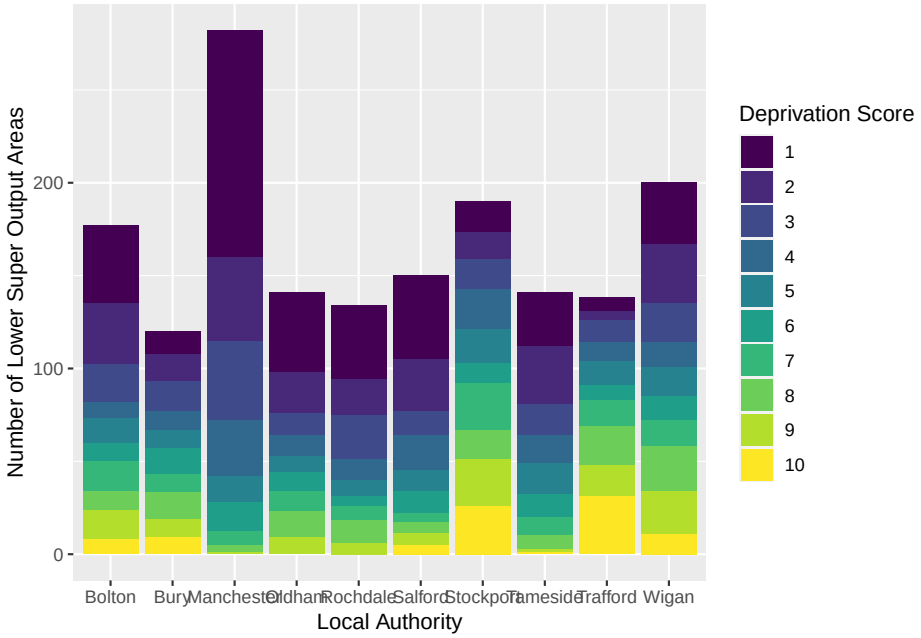
Think back, however, to what we learned last semester, about row and column percentages, and how in bivariate analysis it is important to make decisions that will affect how our readers interpret our data. For example, if we fill the colour of each bar by local authority, we get a chart like this:

```
# The 'dodge' position means that the bars avoid or 'dodge' each other
ggplot(data = burglary_df) +
  geom_bar(mapping = aes(x = as.factor(IMDdeci), fill = LAname), position = "stack") +
  scale_fill_viridis_d() +
  labs(x = "Deprivation Score", y = "Number of Lower Super Output Areas", fill = "Local Authority
```



How do the different local authorities compare in terms of the distribution of deprived neighbourhoods? It can be a bit tough to say. But what about if we flip these:

```
# We switch the positions of LAname and IMDdeci in the second line of code in this exa
ggplot(data = burglary_df) +
  geom_bar(mapping = aes(x = LAname, fill = as.factor(IMDdeci)), position = "stack") +
  scale_fill_viridis_d() +
  labs(x = "Local Authority", y = "Number of Lower Super Output Areas", fill = "Depriva
```

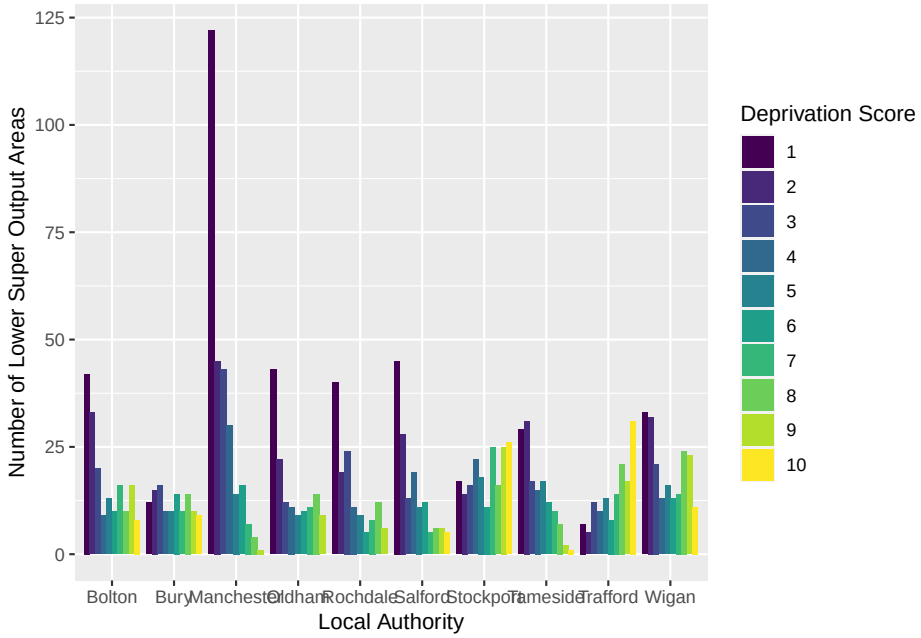


This seems clearer.

In your google doc, state what you observe from the bar graph and grouped bar graph depicting the relationship between deprivation and LSOAs.

Note: in the previous example, you might have noticed – inside the `geom_bar()` function, after the closing bracket of the `aes()` function – the parameter `, position = "stack"`. This means that we get a *stacked* bar chart. You could change this to `, position = "dodge"`, which would give you a clustered bar chart. Like so:

```
# Instead of stack, the 'dodge' position means that the bars avoid or 'dodge' each other
ggplot(data = burglary_df) +
  geom_bar(mapping = aes(x = LAname, fill = as.factor(IMDdeci)), position = "dodge") +
  scale_fill_viridis_d() +
  labs(x = "Local Authority", y = "Number of Lower Super Output Areas", fill = "Deprivation Score
```



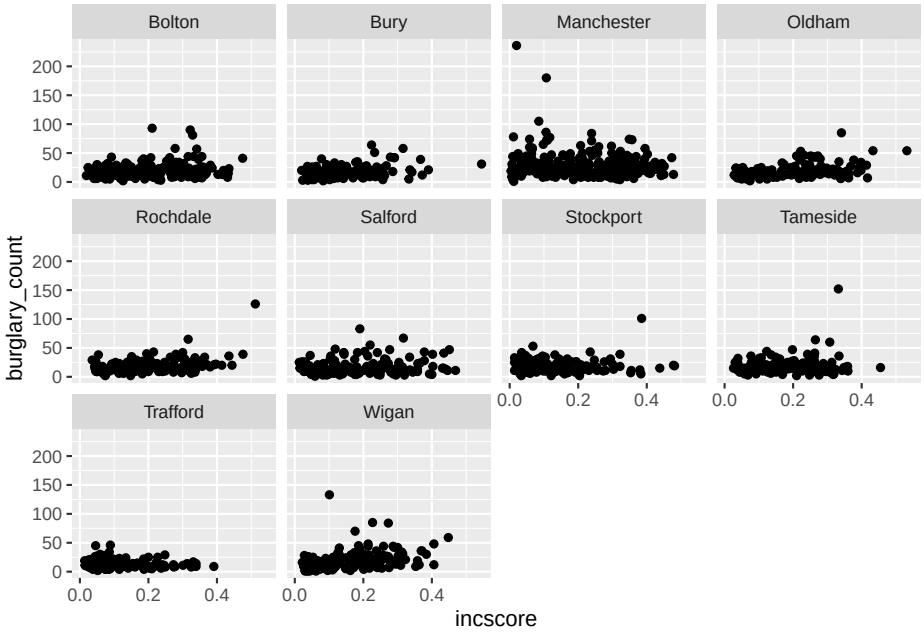
This might also be another way to display your data. There are advantages and disadvantages to both the stacked and clustered bar charts, and it has to do with what they hide and show!

3.3.2.3 *Multiple Bar Graphs*

Another useful visual is multiple bar graphs. This creates different plots for each level of a factor instead of putting lots of information into one graphic. Using the function `facet_wrap ()`, we make the information obtained from the previous scatterplot, where we coloured in each point by local authority, clearer.

Although the scatterplot uses numeric variables, we use this information by local authority, and `LAname` is classed as factor:

```
# To facet the scatterplot by local authority, ' ~ ' is used next to 'LAname'
ggplot(data = burglary_df) +
  geom_point(mapping = aes(x = incscore, y = burglary_count)) +
  facet_wrap(~LAname)
```




```
# By default, facet_wrap fixes the y-axis of each graph to make comparisons easier
```

It is important to choose an appropriate graph, so thought is required. May the following graphic be fair warning:

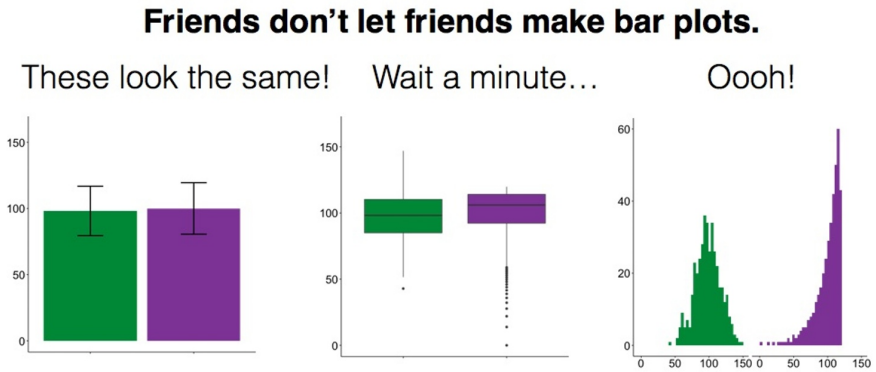


Figure 3.8: **Figure 3.8** Friends don't let friends do this

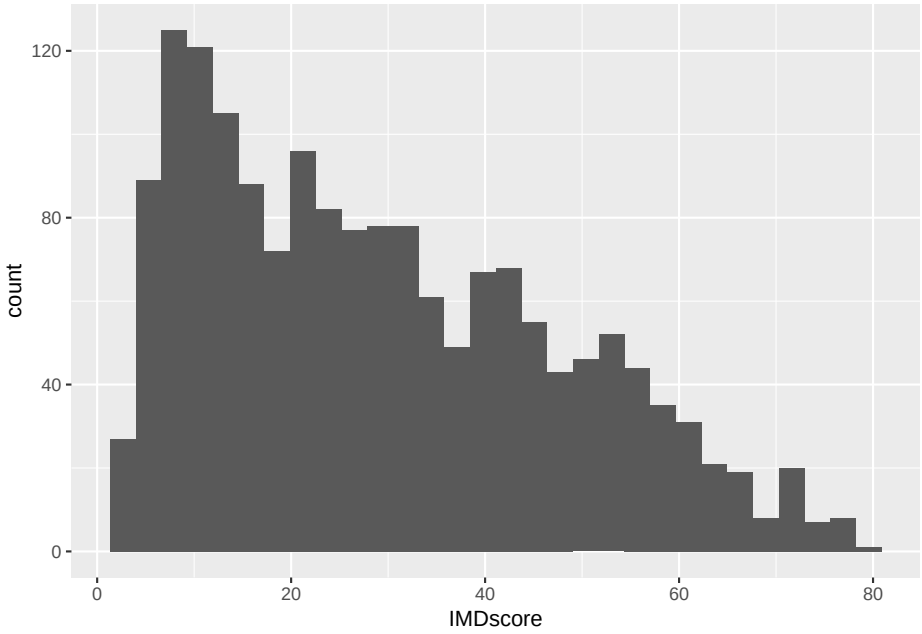
3.3.3 Graphs for Numeric Data

3.3.3.1 Activity 10: Histograms

Histograms differ from scatterplots because they require only one numeric variable and they create bins, visualized as bars, to count the frequency of each bar. We create a histogram of the variable `IMDscore`, the overall score of deprivation in each neighbourhood:

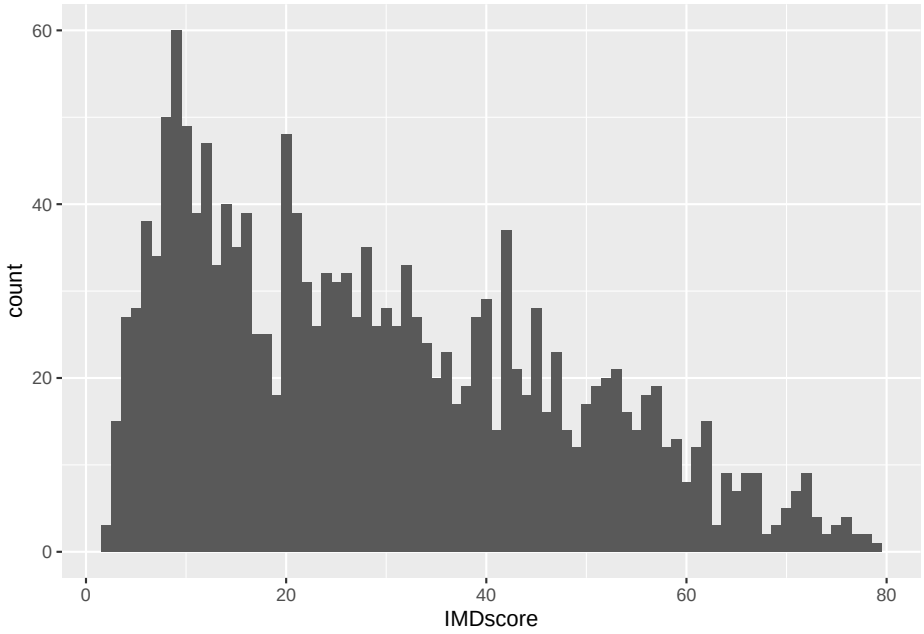
```
ggplot(data = burglary_df, mapping = aes(x = IMDscore)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



How the histogram looks, and therefore, the conclusions drawn from the visual, are sensitive to the number of **bins**. You should try different specifications of the bin-width until you find one that tells the complete and concise story of the data. Specifying the size of bins is done using the argument `bin_width =`; the default size is 30 and you can change this to get a rougher or a more granular idea:

```
# Using bin-width of 1 we are essentially creating a bar for every increase in level o
ggplot(data = burglary_df, mapping = aes(x = IMDscore)) +
  geom_histogram(binwidth = 1)
```



Play around with the bin widths to find which one looks the most visually comprehensible. In your google doc, state what bin width you think does the job, and why.

3.3.3.2 Activity 11: Line Graphs

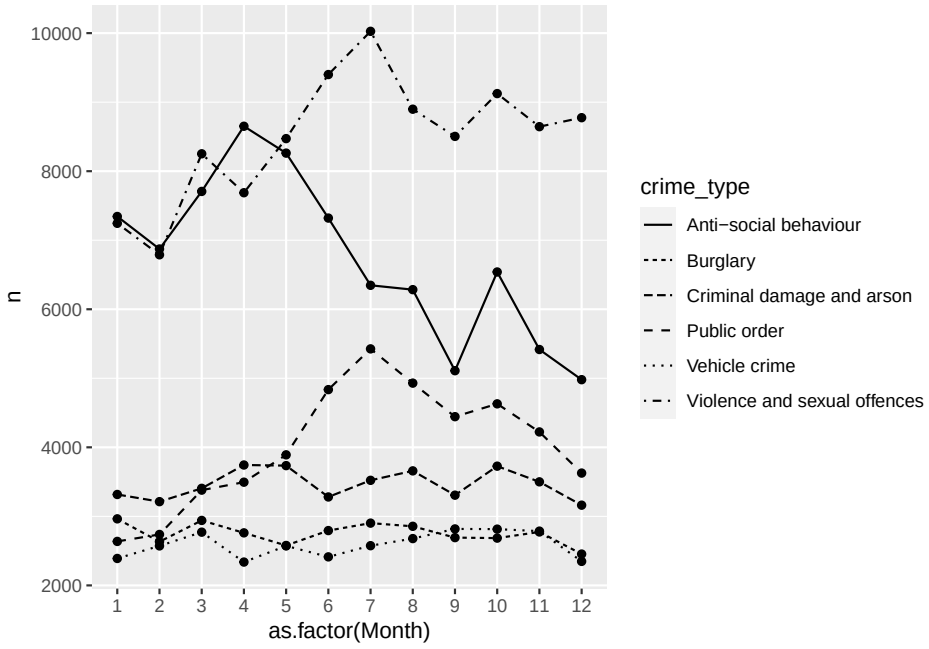
A very good way of visualizing trends over time are line graphs. For this, let us use the `monthly_df` data frame. Ensure it is loaded into your environment. Have a look at the structure of these data to get to know them.

The data is in long format: we have a single month variable for 12 months' worth of data (rather than one variable or column for each month). Setting longitudinal data up in long format allows us to specify the aesthetics (e.g., x- and y-axes) easily.

For a line graph, we use the function `geom_line()`. We plan to plot the counts to show the trends of different crimes over the course of the year. Our time variable, `month`, should run along the x-axis, and the count variable, `n`, should run on the y-axis.

To show each crime type separately, we use the group aesthetic and a new aesthetic, `linetype`, which uses different patterns for each group. To show our time measurement points, we also add `geom_point ()`:

```
ggplot(data = monthly_df, aes(x = as.factor(Month), y = n, group = crime_type, linetype
  geom_line() +
  geom_point()
```




```
# We are saying, 'Hey R, may you please make a graphic using the data frame "monthly_d
```

Now, in your google doc, state what you observe from this line graph on different crimes throughout the months.

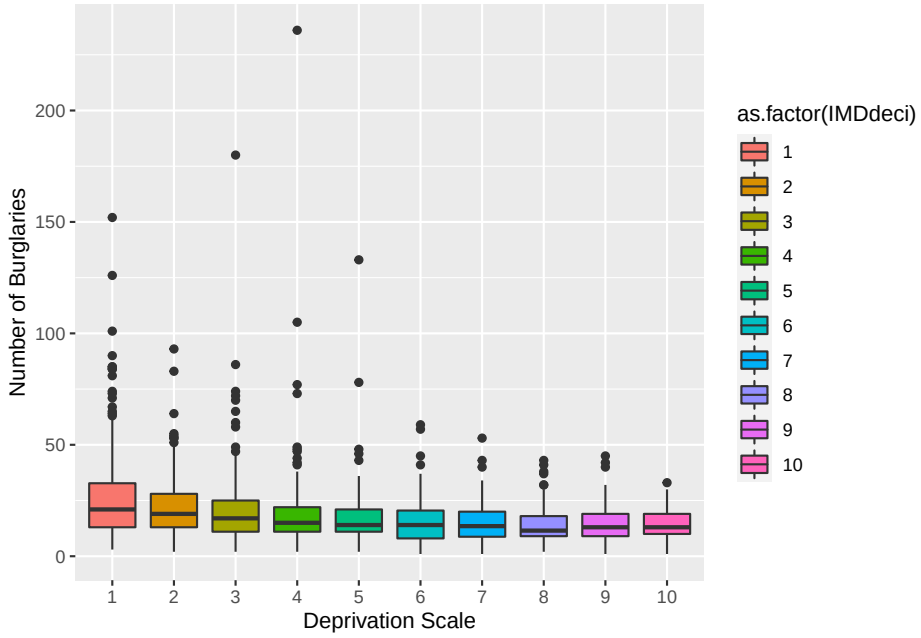
3.3.3.3 Activity 12: Boxplots

If you want more information about your data than a histogram can give, a box (and whisker) plot may be what you are looking for. A boxplot visualizes the minimum, maximum, interquartile range, the median, and any unusual observations - your five-number summary!

For this example, we return to the `burglary_df` data, and focus on the distribution of burglary counts. In addition, we make a boxplot for *each* deprivation decile to show how LSOAs in the most deprived deciles often have higher median burglary counts.

We send all of this information, the boxplots, into one object called `my_whisker_plot`, to which we can return in case:

```
my_whisker_plot <- ggplot(data = burglary_df) +
geom_boxplot(mapping = aes(x = as.factor(IMDdeci), y = burglary_count, group = as.factor(IMDdeci)

my_whisker_plot +
  labs(x = "Deprivation Scale", y = "Number of Burglaries")
```



In your group google doc, state what you observe from your newly created boxplot.

3.4 SUMMARY

Today was about using a popular package called `ggplot2` for effectively visualizing data. We learn this because it is important to understand our data before embarking on more complex analyses. If we do not understand our data beforehand, we risk misinterpreting what the data are trying to tell us. The package is based on this approach called *Grammar of Graphics*, which proposes a 'grammar' to describe and construct statistical graphics. The package, however, sees the creation of data visuals accomplished through **layers**.

Our first topic we learned was about these layers, comprising data, aesthetics, and geometric objects. As a final touch, **themes** changed the overall appearance of your graphic. Our second and third topics introduced us to **graphs** that we can use for all types of variables, and how to create them in `ggplot2`. One example was the histogram, whereby adjusting the width of the **bins** helped for better interpretation of the data.

Hurrah: homework time!

3.4.1 ANSWERS TO ACTIVITIES:

- 1. burglary_df: 1,673 observations, 9 variables ; monthly_df: 73 observations, 3 variables
- 2. (1) burglary_df: n= 1,673 LSOAs; (2) monthly_df: n= 72 months
- 3. Hard to say…
- 4. Roughly: most areas have burglary counts below 50
- 5. N/A
- 6. N/A
- 7. N/A
- 8. N/A
- 9. Roughly: The most number of LSOAs are in decile 1, meaning they are within the 10% of the most deprived LSOAS, and in the final and stacked grouped bar graph, LSOAs from Manchester have a large number in decile 1.
- 10. N/A
- 11. Roughly: Violence and sexual offences are the highest compared to other crimes and reaches its peak in the summer months. Antisocial behaviour rises from February to April, then declines, hitting a low point in September before rising again.
- 12. Roughly: Decile 1 has highest median (about 25) compared to other deciles but with many outliers. Decile 4 has the most extreme outlier of about 250 burglaries.

Chapter 4

Descriptive Statistics

Central Tendency, Outliers, and Dispersion

Learning Outcomes

- Revisit what descriptive statistics are and their importance in understanding your data
- Learn / review measures of the central tendency and dispersion and how to conduct them
- Learn how to identify outliers and skewness

Today's Learning Tools:

***Total number of activities*: 12**

Data:

- Law Enforcement Management and Administrative Statistics (LEMAS)-Body-Worn Camera Supplement (BWCS) Survey
- 2004 Survey of Inmates in State and Federal Correctional Facilities (SISFCF)

Packages:

- `dplyr`
- `ggplot2`
- `here`

- `modeest`
- `moments`
- `qualvar`
- `skimr`

Functions introduced (and packages to which they belong)

- `diff()` : Computes differences between values in a numeric vector (`base R`)
- `dim()` : Check the dimensions of an `R` object (`base R`)
- `DM()` : Computes deviation from the mode (`qualvar`)
- `group_by()` : Group observations by variable(s) for performing operations (`dplyr`)
- `IQR()` : Compute interquartile range (`base R`)
- `is.na()` : Returns `TRUE` when values are missing, `FALSE` if not (`base R`)
- `mean()` : Compute arithmetic mean (`base R`)
- `max()` : Returns the maximum value (`base R`)
- `min()` : Returns the minimum value (`base R`)
- `mlv()` : Compute the mode (`modeest`)
- `quantile()` : Compute quantiles as per specified probabilities (`base R`)
- `sd()` : Computes standard deviation of a numeric vector (`base R`)
- `skewness()` : Calculate degree of skewness in a numeric vector (`modeest`)
- `skim()` : Provide summary statistics specific to object class (`skimr`)
- `summarize()` : Create new summary variable(s), e.g., counts, mean (`dplyr`)
- `summary()` : Produce summary of model results (`base R`)
- `table()` : Generates a frequency table (`base R`)
- `var()` : Computes variance (`base R`)

4.1 Revisiting Descriptive Statistics

The field of statistics is divided into two main branches: descriptive and inferential. Much of what you will cover today on descriptive statistics will be a review from last semester, but learning to conduct them in `R` will be a new learning experience.

So, what are descriptive statistics? If general statistics is concerned with the study of how best to collect, analyse, and make conclusions about data, then this specific branch – descriptive statistics – is concerned with the sample distribution and the population distribution from which the sample is drawn. Basically, descriptive statistics are ways of describing your data.

Similar to last week's lesson on data visualization, describing your data is important because it helps you identify patterns and anomalies. In addition, it gives others a succinct understanding of your data, so it facilitates good communication. We revisit and learn another three substantive topics today: the **central tendency**, **outliers**, and **dispersion**.

4.1.1 Activity 1: Our preparation routine

We start by doing the usual routine, but with new data on police body-worn cameras:

1. Open up your existing `R` project
2. Load the required packages (listed at the top of this lesson). Some of these are familiar from previous weeks, so you only need to use the function `library()`to load them. For packages that we have not used before and are not installed in our `R`, you will need to use the function `install.packages()` first before loading it with `library()`:

```
# Remember, we only need to do this once whenever we meet a new package and would like to keep it

install.packages("modeest")
install.packages("moments")
install.packages("qualvar")
install.packages("skimr")
```

```
# All of these are now in our R library of packages, but we need to load them, or 'bring them to

library(dplyr)
library(ggplot2)
library(here)
library(modeest)
library(moments)
library(qualvar)
library(skimr)
```

3. You will have downloaded the datasets from Blackboard as usual, ensuring the datasets have been moved into the subfolder you had created from Lesson 2 called 'Datasets'.
4. For now, open only the 2016 LEMAS-BWCS dataset (37302-0001-Data.rda). These data are from the Inter-university Consortium for Political and Social Research (ICPSR) website and you can find them by using the dataset name, which refers to the ICPSR study number. This

time, data are stored in an R data file format (.rda). Always check for the type of file format the data are in - this determines what codes and packages (if any) you will need to use to load the data into R.

5. For this type of dataset, as it is in an '.rda' data file format, we will need to use the `load()` function to import the data frame into our environment, specifying your working directory using `here()` like we have in Lesson 2, section 2.1.

```
load(here("Datasets", "37302-0001-Data.rda"))
```

6. Name the data frame `bwcs` by using the `<-` assignment operator. Another way of looking at it is you are putting the dataset into a 'box' that you will call 'bwcs'. It will appear in the Environment pane like all objects you create in R.

```
bwcs <- da37302.0001
```

7. Use the function `View(bwcs)` to ensure the file is loaded and to get to know your data. You can also use the function `dim(bwcs)` to see the number of observations and variables.

4.2 Today's 3

Two primary ways of describing your data have to do with the central tendency and their dispersion. We also learn about outliers.

4.2.1 Central Tendency

Central tendency refers to a descriptive summary of the centre of the data's distribution; it takes the form of a single number that is meant to represent the middle or average of the data. The **mean**, **median**, and **mode** are common statistics of the central tendency.

The *mean* is the average and it is useful when we want to summarise a variable quickly. Its disadvantage is that it is sensitive to extreme values, so the mean can be distorted easily.

To address this sensitivity, the *median* is a better measure because it is a robust estimate, which means that it is not easily swayed by extreme values. It is the middle value of the distribution.

The *mode* helps give an idea of what is the most typical value in the distribution, which is the most frequently occurring case in the data. While mean and median apply to numeric variables, the mode is most useful when describing categorical variables. For example, you may want to know the most frequently occurring category.

4.2.1.1 Activity 2: Recap of how to obtain the mean and median

In Lesson 1, you were introduced to the functions `mean()` and `median()`. There is no direct function in `base R` to calculate the mode, but we will learn one way to do so soon. First, find only the mean and median for the following six numbers:

```
# 345, 100, 250, 100, 101, 300
```

Type your answers in the group google doc.

4.2.1.2 Activity 3: Levels of measurement

From today's data, we are interested in exploring the adoption of body-worn cameras (BWCs) and their usage in a sample of agencies. The variables that will help us explore this is `Q_10A` because it measures whether the agency adopted BWCs as of 2016 and `Q_12` because it measures the number of cameras that agencies reported using.

Let us say we want to learn about these variables. The first step to choosing the appropriate analysis for our variables, `Q_10A` and `Q_12`, is establishing: *what are their levels of measurement*?

To review levels of measurement, refer to Lesson 2 (section 2.4.1). Have a think and record your answers for each variable in the google doc.

....

Figure 4.1 What are the levels of measurement?

Now that we have established the level of measurement for our variables, what type of object does `R` think these variables are? In other words, what are these variables classed as? Recall that we can use the `class()` function to answer this:

```
# We start with variable Q_10A:

class(bwcs$Q_10A)
```

```
## [1] "factor"
```

`Q_10A` is classed as a factor. Another function that will return what the variable is classed as but also the specific levels is the `attributes()` function:

```
attributes(bwcs$Q_10A)
```

```
## $levels
## [1] "(1) Agency has acquired in any form (including testing)"
## [2] "(2) Agency has not acquired"
##
## $class
## [1] "factor"
```

This factor variable has two levels: '(1) Agency has acquired in any form (including testing)' and '(2) Agency has not acquired'.

Now, we want to know the **mode** – which level or category appears most frequently in our data set. We use the function `mlv()` (acronym for **m**ost **l**ikely **v**alues) from the `modeest` package to answer this question:

```
# Double check that the package modeest is loaded for this code to work

mlv(bwcs$Q_10A)
```

```
## [1] (2) Agency has not acquired
## 2 Levels: (1) Agency has acquired in any form (including testing) ...
```

The output returns the answer as: '(2) Agency has not acquired'. It also, conveniently, prints all the levels so we can see what are the other categories.

If you want to cross check your answer obtained from the `mlv()` function, create a frequency table using the `table()` function:

```
table(bwcs$Q_10A)
```

```
##
## (1) Agency has acquired in any form (including testing)
##                                                    1915
##                                (2) Agency has not acquired
##                                                    2013
```

Indeed, there are 2,013 observations where the agency had not acquired any body worn cameras, which is more than the 1,915 observations where the agency had acquired body worn cameras in any format.

We could also run the `mlv()` function and save the output in an object rather than print the answers into the console. Here, we save to the object called `mode_adopted`:

```
mode_adopted <- mlv(bwcs$Q_10A)
mode_adopted
```

```
## [1] (2) Agency has not acquired
## 2 Levels: (1) Agency has acquired in any form (including testing) ...
```

```
# This new object will appear in your Environment pane
```

Our exploration into the adoption of BWCs has so far yielded the knowledge that the majority of agencies had not acquired BWCs as of 2016. But how about those that have done so? To what extent do they use BWCs? This is where variable `Q_12` comes in handy:

4.2.1.3 Activity 4: The extent of body worn camera use

Similar to the previous activity, obtain the class of the variable `Q_12`. Type the code you used and what class the variable is in your group google doc.

After you do so, we get to know more about `Q_12`. First, let us get the minimum value:

```
 min(bwcs$Q_12,  na.rm=TRUE)
```

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

And then the maximum:

```
max(bwcs$Q_12, na.rm=TRUE)
```

```
## [1] 1200
```

Above we used the functions `min()` and `max()`, and inside we put the name of the specific variable we want the minimum and maximum of and the name of the data frame to which it belongs (hence, `bwcs$Q_12`). We also added the code `na.rm=TRUE` after the comma.

What the heck is `na.rm`? When calculating measures of central tendency, you need to tell `R` that you have missing (i.e., NA) observations. Using `na.rm = TRUE` will exclude observations that have already been defined as missing. If we do not specify this code, it will instead return `NA` for agencies that have acquired BWCs because `Q_12` has missing data.

Now let us find the average number of BWCs used by agencies that have responded 'Yes, acquired' in `Q10_A`. We will use two ways to determine the mean: the `dplyr` way and the usual `base R` way. When we refer to `base R`, it means that these functions are already included in `R`, so there is no need to install additional packages.

We start with `dplyr`. Now, we will do two things: first, we use the `group_by()` function in order to group our observations by the values of the `Q_10A` variable. The values are '1' and '2', and are essentially the categories '(1) Agency has acquired in any form (including testing)' and '(2) Agency has not acquired'. Then, we use the `summarise()` function, to tell us some summary statistics (in this case the mean, using the `mean()` function) *for each group*.

We did a similar thing last semester whereby we obtained these descriptive statistics for *each* value of a particular variable when we examined the relationship between age and the reason someone was arrested. Here, we are looking for the values of how many body worn cameras are used (`Q_12`) *between* whether they are were acquired or not (`Q_10A`):

```
# If you do not remember what '%>%' means, refer to Lesson 2, section 2.4.2.2
bwcs %>%
  group_by(Q_10A) %>%
  summarize(mean_deployed = mean(Q_12, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##   Q_10A                                                    mean_deployed
## * <fct>                                                            <dbl>
## 1 (1) Agency has acquired in any form (including testing)           31.8
## 2 (2) Agency has not acquired                                       NaN
```

We see that in agencies where BWCs were acquired, the average number of cameras is about 32. But in agencies where BWCs were not acquired, the average is `NaN`. What is this?

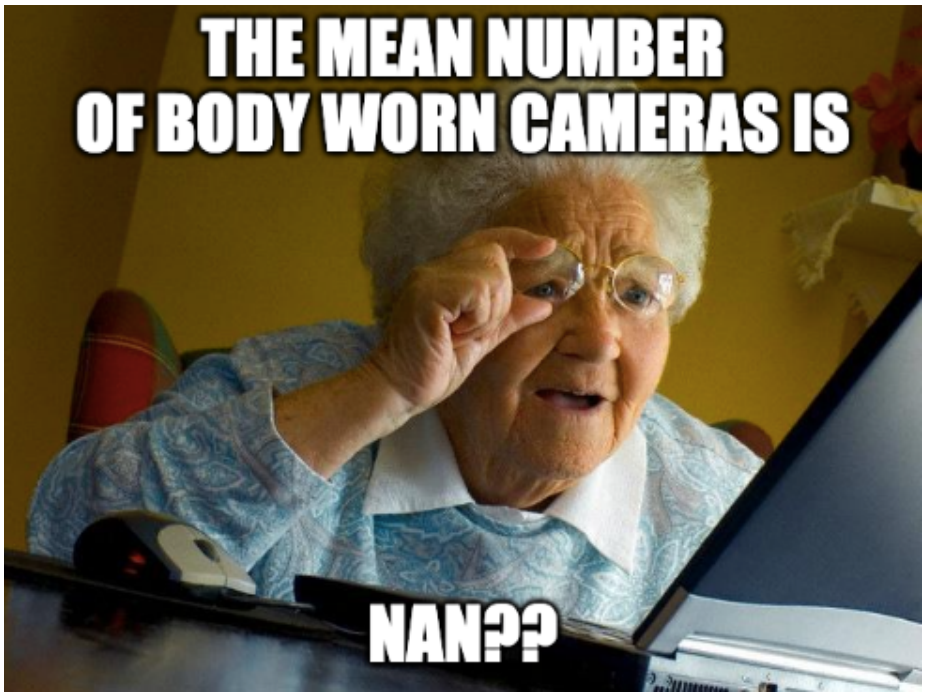


Figure 4.1: **Figure 4.2** NaN??

`NaN` in `R` stands for **N**ot **a** **N**umber. The reason is *all* the values for the number of cameras used (in reference to `Q_12`) among agencies that had not acquired cameras (in reference `Q_10A`) is missing (`NA`). It makes sense that this should be so, because if agencies responded in `Q_10A` that they had not acquired any BWCs, then why would we expect them to give a number in `Q_12` on how many cameras they used? `Q_12` does not apply to them.

Now, the above was the `dplyr` way. But you could use the `base R` way with the `mean()` function:

```
mean(bwcs$Q_12, na.rm = TRUE)
```

```
## [1] 31.82024
```

The answers from both ways are the same: 31.8 . This is because we know that one group (agencies that have not acquired BWCs) do not contribute to the average. We learn both methods because there may be situations where one is more appropriate than the other (e.g., maybe you want to see means for the individual categories when they are not `NaN`).

Next, we identify the median, as, remember, the mean is susceptible to outliers. To get the median for each group, we use `summarise()` but, this time, we include the `median()` rather than the `mean()` function:

```
# Use the same format as above, but this time use the median() function
# dplyr way
bwcs %>% group_by(Q_10A) %>% summarise(med_deployed = median(Q_12, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##   Q_10A                                                    med_deployed
## * <fct>                                                           <dbl>
## 1 (1) Agency has acquired in any form (including testing)             8
## 2 (2) Agency has not acquired                                        NA
```

What about the `base R` way?

```
# Base R way
median(bwcs$Q_12, na.rm = TRUE)
```

```
## [1] 8
```

Again, both methods to obtain the median are the same, but the median differs a lot from the mean: it is 8.

What does this mean? There must be a few agencies that use a *lot* of body worn cameras, which distorts our mean to appear as if the majority of agencies use 32 cameras. When the median and the mean are very different from each other, we have what is known as *skew*.

Now what if we want to calculate these descriptive statistics at once? This is where the `dplyr` way is most handy. To obtain these statistics together, we specify that we want to create four columns that will appear in the output as a table:

- (1) *mean_deployed*, using the `mean()` function
- (2) *median_deployed*, using the `median()` function
- (3) *mode_deployed*, using the `mlv()` function
- (4) *total*, using the `sum()` function

```
# Table containing the mean, median, mode, and total number of BWCs used
bwcs %>%
  group_by(Q_10A) %>%
  summarise(mean_deployed = mean(Q_12, na.rm = TRUE),
            median_deployed = median(Q_12, na.rm = TRUE),
            mode_deployed = mlv(Q_12, method='mfv', na.rm = TRUE),
            total = sum(Q_12, na.rm = TRUE))
```

```
## # A tibble: 2 x 5
##   Q_10A                          mean_deployed median_deployed mode_deployed total
## * <fct>                                  <dbl>           <dbl>         <dbl> <dbl>
## 1 (1) Agency has acquired in ~          31.8               8             1 60363
## 2 (2) Agency has not acquired           NaN               NA           NaN     0
```

Another handy package for descriptive statistics is `skimr`. The function `skim()` produces measures of central tendency and measures of dispersion (we will learn more on these in the later section), and number of missing values.

A great feature is that it also includes a histogram of the numeric variables specified. If you do not want to specify any variables, `skim()` will summarise your entire data frame, and this may be good, but it depends on the size of your dataset:

```
# Ensure the package 'skimr' is loaded or this will not work
# Produce a summary of your Q_12 variable, grouped by Q_10 using skim()
bwcs %>%
  group_by(Q_10A) %>%
  skim(Q_12)
```

skim_type	skim_variable	Q_10A	n_missing
numeric	Q_12	(1) Agency has acquired in any form (including testing)	18
numeric	Q_12	(2) Agency has not acquired	2013

4.2.2 Outliers

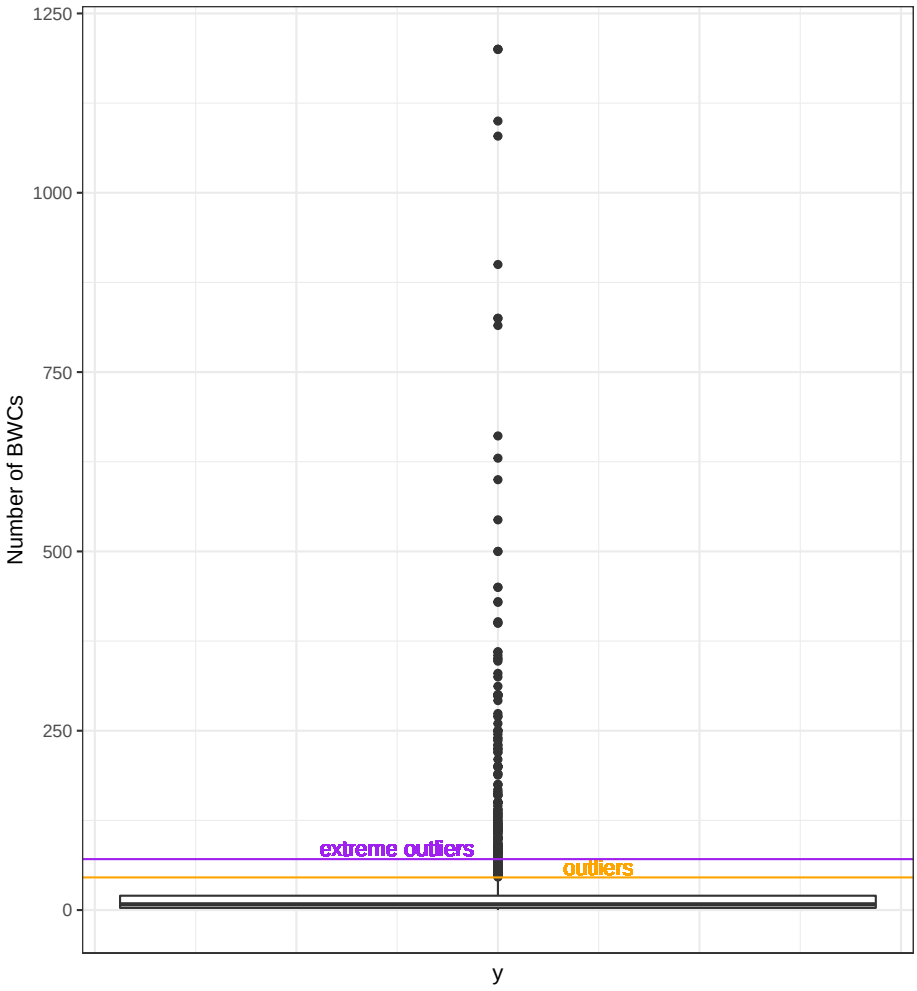
Recall the main disadvantage of the mean: it is not robust to extreme values, otherwise known as **outliers**.

What exactly constitutes an *outlier*? Generally, an outlier is any observation that shows an extreme value on one of our variables. There are different ways to define what an outlier is, but, in this class, we use the **interquartile range** (IQR) to do so.

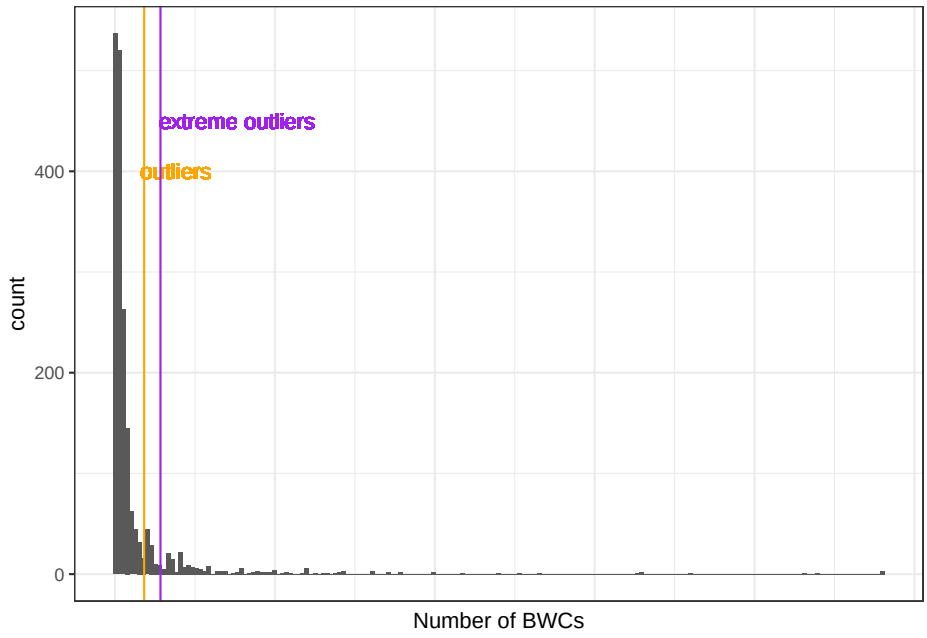
The IQR is another robust estimate and is the 75th percentile observation (the third quartile [Q3]) minus the 25th percentile observation (the first quartile [Q1]) in your distribution:

$$IQR = Q3 - Q1$$

Now you can also see where the outliers, and extreme outliers are. Outliers are anything above the orange line, extreme outliers are anything above the purple line:



We could also show this in a histogram, where everything to the right of the orange line is an outlier, and everything to the right of the purple is an extreme outlier:



–>

A popular method for determining outliers is known as **Tukey fences**. According to this method, outliers are seen as falling outside a lower or an upper inner fence. They are calculated from the following:

$$Lower\ inner\ fence = Q1 - (1.5 * IQR)$$

$$Upper\ inner\ fence = Q3 + (1.5 * IQR)$$

In addition, extreme outliers are seen as falling outside a lower and upper outer fence and can be calculated from the following:

$$Lower\ outer\ fence = Q1 - (3 * IQR)$$

$$Upper\ outer\ fence = Q3 + (3 * IQR)$$

To visualise this, see Figure 4.3:

4.2.2.1 Activity 5: Determining outliers using Tukey fences

We obtain the IQR of our `Q_12` variable using the `IQR` function, and save this into an object called `bwc_deployed_iqr`:

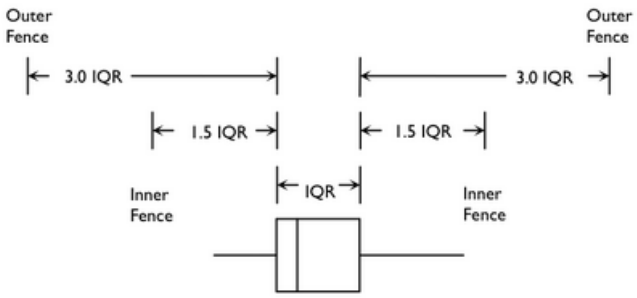


Figure 4.2: **Figure 4.3** Tukey fences

```
bwc_deployed_iqr <- IQR(bwcs$Q_12, na.rm = TRUE)
```

It is now saved as an object and holds the value for the IQR:

```
# Our 'box' called 'bwc_deployed_iqr' that holds the IQR of the variable Q_12
bwc_deployed_iqr
```

```
## [1] 17
```

Next, we obtain the 1st and 3rd quartiles of the `Q_12` variable using the `quantile()` function and place them in objects too:

```
bwc_deployed_1st <- quantile(bwcs$Q_12, 0.25, na.rm = TRUE)
bwc_deployed_3rd <- quantile(bwcs$Q_12, 0.75, na.rm = TRUE)
```

Now, we calculate the Tukey fences and put them each in separate objects.

The lower inner fence:

```
lower_inner_fence <- bwc_deployed_1st - 1.5 * bwc_deployed_iqr
lower_inner_fence
```

```
##   25%
## -22.5
```

The upper inner fence:

```
upper_inner_fence <- bwc_deployed_3rd + 1.5 * bwc_deployed_iqr
upper_inner_fence
```

```
##  75%
## 45.5
```

You can also calculate the 'outer fences' which are considered extreme outliers:

```
lower_outer_fence <- bwc_deployed_1st - 3 * bwc_deployed_iqr
lower_outer_fence
```

```
## 25%
## -48
```

```
upper_outer_fence <- bwc_deployed_3rd + 3 * bwc_deployed_iqr
upper_outer_fence
```

```
## 75%
##  71
```

Let us see how many agencies are considered outliers and extreme outliers on how many BWCs they use. We revisit the `filter()` function for this. (See Lesson 2, section 2.4.3.1 .) Remember that we filter to keep all rows that meet a specific condition. In this case, we want all observations whose value for the variable `Q_12` is an outlier (outside the inner fences but within the outer fences) and an extreme outlier (outside the outer fences). We will save them in separate objects called `outliers` and `outliers_extreme`:

```
outliers <- bwcs %>%
  filter(Q_12 > upper_inner_fence| Q_12 < lower_inner_fence)
```

and

```
outliers_extreme <- bwcs %>%
  filter(Q_12 > upper_outer_fence| Q_12 < lower_outer_fence)
```

These new objects are now in our environment (that Environment pane in the top right of your `RStudio`). We can look at their characteristics on the number of BWCs used with the `summary()` function:

```
summary(outliers$Q_12)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    46.0    60.0   100.0   162.1   167.2  1200.0
```

and

```
summary(outliers_extreme$Q_12)
```

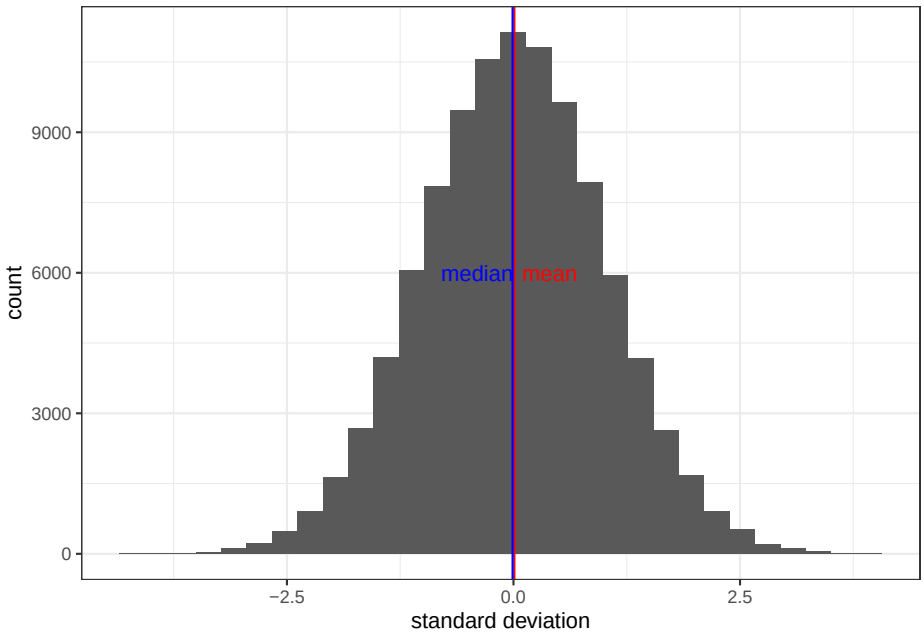
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    72.0   100.0   130.0   215.4   241.2  1200.0
```

In your group google doc, type your answers to the following: (1) lower and upper inner fences, (2) lower and upper outer fences, and (3) minimum of `outliers` and `outliers_extreme`.

4.2.2.2 On Skewness

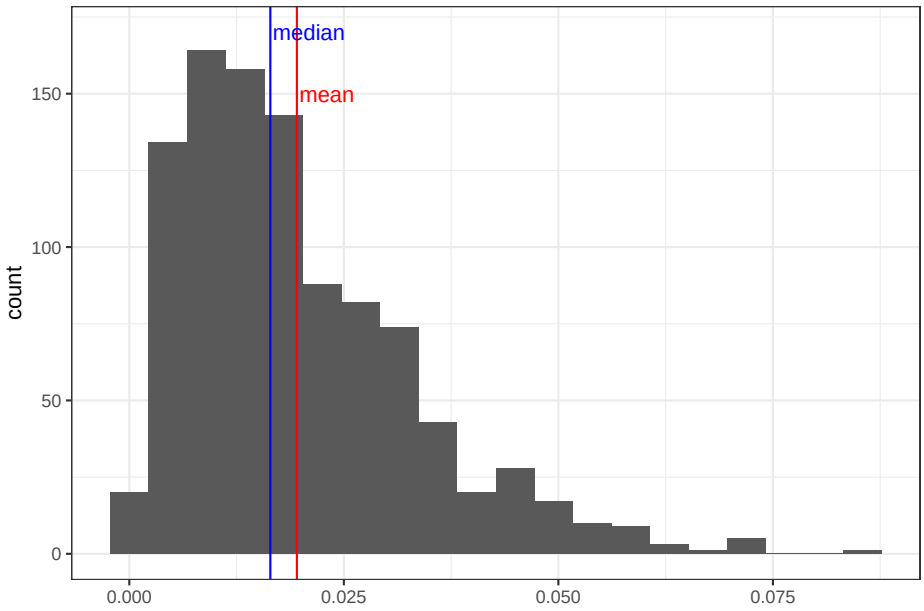
Related to outliers is **skewness**. This has to do with the shape of the distribution. You might remember from last semester we discussed something called the bell curve, or normal distribution. This is how you expect your data to look if the mean and the median are the same exact value, and your data are distributed equally on either side of this value.

Here is a normal distribution (Figure 4.4):

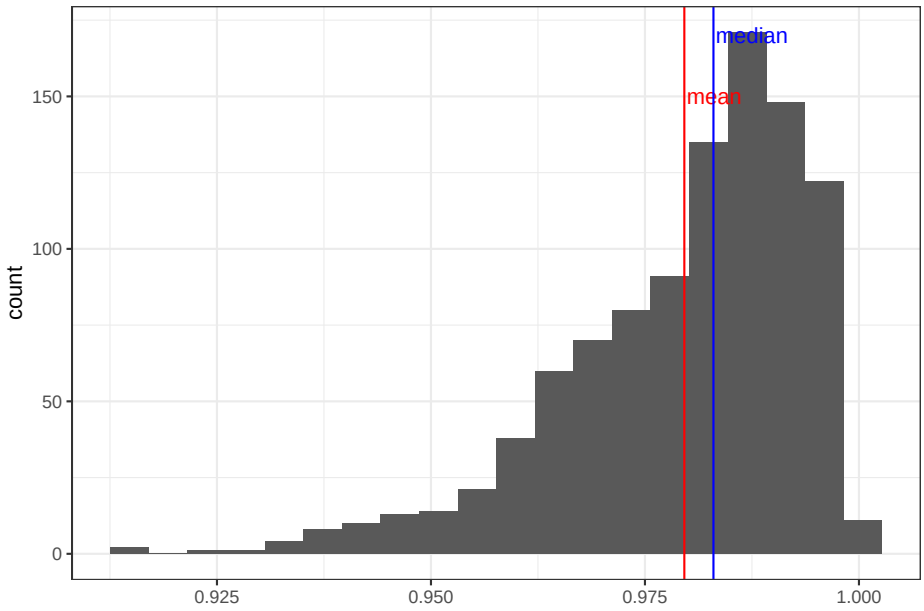


A skewed distribution would pull the observations more so to the left or to the right, and you would have a long tail on either side. This would also cause your mean and median to be further apart.

Here is a right skewed distribution (Figure 4.5):



And here is a left skewed distribution (Figure 4.6):



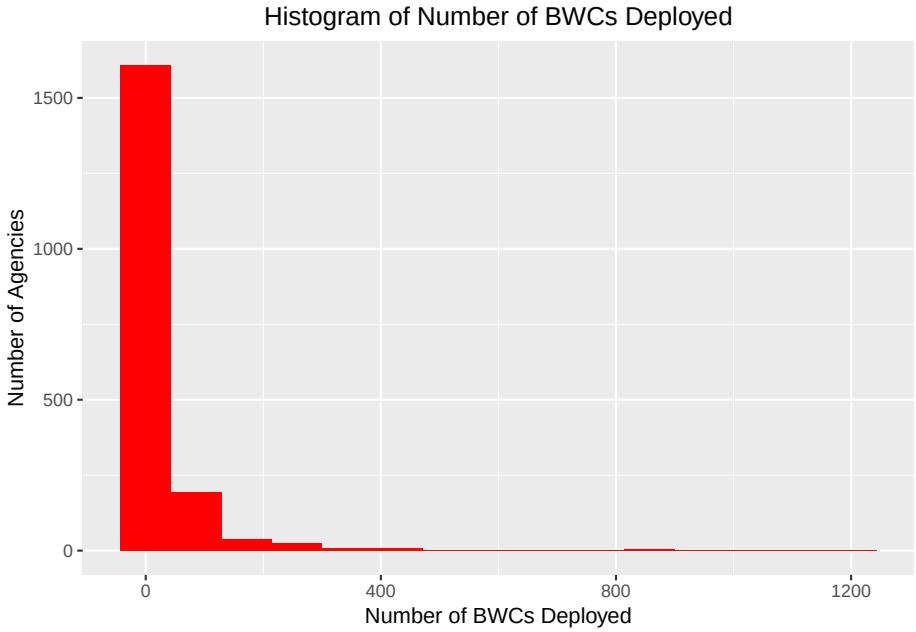
So, how can we tell if our data are skewed? We can tell by visualising it!

4.2.2.3 Activity 6: Visualizing skewness using a histogram

When your distribution is skewed, the majority of cases veer more to the left or right of the distribution, with a tail of outliers. An initial way of checking for skewness is to use a histogram, like from last week.

```
# To create a histogram of number of BWCs used, we will use the `ggplot()` function and the geome
# Ensure the package ggplot2 is loaded

ggplot(data = bwcs, mapping = aes(x = Q_12)) +
  geom_histogram(bins = 15, fill = "red") +
  labs(x = "Number of BWCs Deployed", y = "Number of Agencies") +
  ggtitle("Histogram of Number of BWCs Deployed") +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## Warning: Removed 2031 rows containing non-finite values (stat_bin).
```



What do you observe from the histogram? Discuss in your groups, or reflect if you are in the quiet room. What do you think the number of BWCs may depend on? Size of the agency? Willingness to adopt BWCs? Are you surprised by this data?

Figure 4.3: **Figure 4.7** Police Body Worn Cameras

From our histogram, we see that most agencies used fewer than 250 BWCs and only a small proportion deployed more than 800.

Another way of checking for skewness is through the `skewness()` function from the `moments` package. Skewness is determined by the following:

- 0 = no skewness as it is a symmetrical distribution
- positive value = means distribution is positively skewed or skewed right
- negative value = means distribution is negatively skewed or skewed left

We calculate skewness for our variable `Q_12`and put it into an object called `bwc_skew`:

```
bwc_skew <- skewness(bwcs$Q_12, na.rm = TRUE)
```

Now print the result. Will the skewness coefficient indicate negative, positive, or no skewness?

```
bwc_skew
```

```
## [1] 7.640767
```

We see a positive skew!

From the two ways of checking for skewness, `Q_12` is an asymmetric distribution known as a **positively skewed distribution** – one tail of the distribution is longer than the others because of outliers on the right side.

4.2.3 Measures of Dispersion

What is meant by dispersion is the spread of values from the one value chosen to represent them all. That one value is usually the mean, or in the case of categorical variables, the mode. This is important because the single number to summarise our data might often mask variation. Last semester, we read *The Tiger That Isn't* and you might remember the section about the 'white rainbow'. We know the vibrancy of each of the colours of the rainbow, but if we combined these colours, they would form a bland white band in the sky - the average of all the colours is white. If we just represented the rainbow by the average, we would miss quite a lot!

Let us explore how we measure dispersion, and how we can interpret this to draw conclusions about the data.

Figure 4.4: **Figure 4.8** Double rainbow?!

4.2.3.1 Activity 7: Loading the other dataset

We learn about measures of dispersion using another dataset from the ICPSR website. The Survey of Inmates in State and Federal Correctional Facilities (SISFCF) survey has been conducted periodically since 1976, and the current survey (2004) data (04572-0001-Data.rda) can, too, be accessed at the ICPSR website and downloaded as an .rda file. Follow the same steps with loading the data as with the previous data on BWCs. Now, for this dataset, we do the following as well:

```
load(here("Datasets", "04572-0001-Data.rda"))
```

1. After the data loads into `RStudio`, it is now called 'da04572.0001'. Put this into an object called `inmatesurvey04` by using the `<-` assignment operator.

```
inmatesurvey04 <- da04572.0001
```

Once finished with this activity, you will find your new data frame, `inmatesurvey04`, in the *Environment* pane.

4.2.3.2 Dispersion in Nominal and Ordinal Variables

We learn how to conduct two measures of dispersion in `R` for categorical variables: the variation ratio and the index of qualitative variation:

The **variation ratio** (VR) tells us the proportion of cases that are not in the modal category – basically cases not in the most frequent or 'popular' category. The formula for the variation ratio (VR) is:

$$VR = 1–(\frac{N\ modalcat}{N\ total})$$

N modalcat refers to the frequency of observations in the modal category, and *N total* refers to the total number of observations. This formula states that VR is equal to 1 minus the proportion of observations that are in the modal category, which is the same as saying the proportion of observations that are not in the modal category.

4.2.3.3 Activity 8: Calculating the variation ratio

Now, for example, we are interested in knowing whether the modal category describes the overall work histories prior to arrest of federal inmates pretty accurately, or if there is a lot of variation in responses from inmates. To do so, we use the variable `V1748` which tells us about their work histories prior to arrest.

We can look at a frequency table:

```
table(inmatesurvey04$V1748)
```

```
##
##  (1) Full-time  (2) Part-time (3) Occasional (7) Don't know    (8) Refused
##           2180            284             65              0              1
```

Or we can use `skim`:

skim_type	skim_variable	n_missing	complete_rate	factor.ordered	factor.n_unique	factor.top_counts
factor	data	1156	0.6863809	FALSE	4	(1): 2180, (2): 284

```
skim(inmatesurvey04$V1748)
```

Then, we check if we have any observations defined as missing using `is.na` to get a summary of missing cases:

```
table(is.na(inmatesurvey04$V1748))
```

```
##
## FALSE  TRUE
##  2530  1156
```

This tells us that there are 2,530 complete observations and 1,156 missing ones.

We could tell from the tables above that 'Full-time' is the mode, but we can also get `R` to provide the mode and save it in an object:

```
mode_employment <- mlv(inmatesurvey04$V1748, na.rm = TRUE)
mode_employment
```

```
## [1] (1) Full-time
## 5 Levels: (1) Full-time (2) Part-time (3) Occasional ... (8) Refused
```

What about the variation ratio? To get this, we create two objects: one for total number of observations and, in the other object, only the numbers who reported the modal category, which is Full-time (1).

To create the first object, get the number of total observations. Store this value in the object `n_employed`:

```
# The '!' next to 'is.na' is telling R to only keep observations that are not missing
n_employed <- inmatesurvey04 %>%
  filter(!is.na(V1748)) %>%
  summarize(n = n())
```

To create the second object, store the value 'full-time' in the object `n_mode`:

```
n_mode <- inmatesurvey04 %>%
  filter(V1748 == "(1) Full-time" ) %>%
  summarize(n = n())
```

Finally, we can calculate the proportion. Store this value in the object `proportion_mode`:

```
proportion_mode <- n_mode/n_employed
```

We can use this to get the variation ratio. To do this, subtract the proportion of cases in the modal category from 1 .

```
vratio <- 1 - proportion_mode
vratio
```

```
##           n
## 1 0.1383399
```

The VR is 0.1383, meaning that the work history among federal inmates prior to arrest is relatively concentrated in the modal category of full-time employment. This is because the smaller the VR, the larger the proportion of cases there are concentrated in the model category.

4.2.3.4 Activity 9: Onto the Index of Qualitative Variation (IQV)

The IQV is different from the VR because it considers dispersion across all categories, whereas the VR only views dispersion in terms of modal versus non-modal.

The IQV lies between 0 and 1 and tells you the variability of the distribution. The smaller the value of the IQV, the smaller the amount of variability of the distribution. We use the `DM()` function from the `qualvar` package, which stores frequencies of a categorical variable in a vector:

```
# Get the index of qualitative variation for the same variable, V1748
IQV<-as.vector(table(inmatesurvey04$V1748))

DM(IQV, na.rm = TRUE)
```

```
## [1] 0.1729249
```

What is the value that you get in the output? Type the value in your google doc and what you think this value means.

4.2.3.5 Dispersion in Interval and Ratio Variables

These measures of dispersion are the most familiar because they are defined as the spread of values around a *mean*. We revisit three of these: the range, variance, and standard deviation.

4.2.3.6 Activity 10: The range

The **range** is the difference between the maximum and minimum value in a given distribution. We use the ratio-level variable `V2254`, a measure of the age at which inmates stated they first started smoking cigarettes regularly as an example:

```
# Get to know the variable
attributes(inmatesurvey04$V2254)
```

```
## $value.labels
##                Refused             Don't know Never smoked regularly
##                     98                     97                      0
```

Note that this variable has codes that do not indicate the number of times smoked. These are 0 (never smoked); 97 (Don't know); and 98 (Refused). We notice that these categories are not about smoking and including these will give us an incorrect result. To resolve this, we must convert irrelevant categories into missing data.

First, we create a new variable so we do not write over the original, calling it `age_smoker_range` using the `mutate()` function. This is a duplicate of the `V2254` variable, but we will recode each of the irrelevant values (0, 97, 98) as `NA` by using the `na_if()` function from the `dplyr` package.

```
# We make this new variable using the values from the variable 'V2254'
# Then we include it into our data frame 'inmatesurvey04' by specifying the assignment

inmatesurvey04 <- inmatesurvey04 %>%
  mutate(age_smoker_range = V2254,
           age_smoker_range = na_if(age_smoker_range, 0),
          age_smoker_range = na_if(age_smoker_range, 97),
          age_smoker_range = na_if(age_smoker_range, 98),
         )
```

The `range ()` function prints the minimum and maximum values.

```
smoker_range <- range(inmatesurvey04$age_smoker_range, na.rm = TRUE)

smoker_range
```

```
## [1]  1 57
```

```
# We calculate the difference of that object to obtain the value
diff(smoker_range)
```

```
## [1] 56
```

What is the range? Input the answer in your google doc.

BUT you can see something strange while figuring out the range: the minimum is one year old! This may be something we want to exclude in future analyses as it might not be valid – maybe a mistake in coding – so conducting these descriptive statistics that you have been doing can help with tidying your data.

4.2.3.7 Activity 11: Variance

Now, the **variance** (s^2) is about spread, specifically the sum of the squared deviations from the mean, then divided by the total number of observations. The equation looks like this:

$$s^2 = \frac{\sum(x-\mu)^2}{N}$$

We use the function `var()` from the `stats` package that comes already preinstalled in `base R`.

We use the variable `V2529`, which records the number of times inmates reported being written up for verbally assaulting a prison staff member as an example:

```
# First, let us get to know the variable
summary(inmatesurvey04$V2529)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    1.00    1.00    1.00   31.45    3.00  998.00    3545
```

```
attributes(inmatesurvey04$V2529)
```

```
## $value.labels
##    Refused Don't know
##        998        997
```

Now let us create a new variable so we do not write over the original one. We do this so we can recode '997' and '998' values as missing - you can see these stand for 'Refused' and 'Don't know' respectively.

```
inmatesurvey04 <- inmatesurvey04 %>%
  mutate(verb_assault_var = V2529,
          verb_assault_var = na_if(verb_assault_var, 997),
          verb_assault_var = na_if(verb_assault_var, 998),
         )
```

Now let us calculate the variance:

```
var1 <- var(inmatesurvey04$verb_assault_var, na.rm = TRUE)
var1
```

```
## [1] 82.64384
```

The variance is 82.6, which is fairly dispersed, but if you got to know the data and this variable, you will notice that one inmate self-reported 99 verbal assaults, so this will have had an influence on the variance.

4.2.3.8 Activity 12: Standard deviation

Last, we learn about the **standard deviation** (SD), which is the square root of the variance. The smaller it is, the more concentrated the observations are around the mean, while larger values reflect a wider distance from the mean. In other words, the SD tells us how well the mean represents our data. Too large, then the mean is a poor representation, because there is too much variability. We use the variable `verb_assault_var` as an example:

```
# Calculate the standard deviation of the 'verb_assault_var' variable and save it to a

sd <- sd(inmatesurvey04$verb_assault_var, na.rm = TRUE)
sd
```

```
## [1] 9.090866
```

Recall that the SD is the square root of the variance. You can double check to see if the SD, squared, matches the variance. Create a new object, `var2`, by taking the square of the standard deviation:

```
var2 <- sd^2
var2
```

```
## [1] 82.64384
```

Yes, it does match!

The SD is 9.09087, meaning that this number of verbal assaults is one standard deviation above and below the mean, so the majority of reported verbal assaults, about 68% of them (more on the 68-95-97 rule next week), should fall within 9.09 above and below the mean.

4.3 SUMMARY

Descriptive statistics was the name of the game today. These are measures we use to describe our data. First, we learned about measures of central tendency: the **mean**, **median**, and **mode**. Second, pesky **outliers** were examined along with their cousin **skewness**. A way of addressing outliers was through the robust estimate, the **interquartile range**. With skewness, visualizations were used to ascertain whether the distribution could be **positively skewed**. Third, we learned all about measures of dispersions: two for nominal and ordinal variables – the **variation ratio** and the **index of qualitative variation** – and three for interval and ratio variables – the **range**, **variance**, and **standard deviation**. In some situations we found ourselves, missing values appeared. This is actually very common in data, so we addressed these with the `R` argument **na.rm**.

Homework time!

4.3.1 Answers to Activities (where applicable)

```
1. -
2. 199.33= mean(c(345, 100, 250, 100, 101, 300));  175.5= median(c(345, 100, 250, 100, 101, 300))
3. Q_10A: binary categorical; Q_12: ratio/ discrete numeric
4. class(bwcs$Q_12) # we see this is a numeric variable
5. outliers: any agency that has 46 or more (upper inner fence) or -22.5 (lower inner fence thoug
6. -
7. -
8. -
9. The value is 0.1729, meaning that there is not much dispersion across the categories. This sup
10. The range of the age when federal inmates started smoking is 56.
11. -
12. -
```

Chapter 5

Inferential Statistics

Samples, Standard Errors, and Confidence Intervals

Learning Outcomes:

- Understand what inferential statistics are and why they are used
- Learn how samples can be used to draw conclusions about the population
- Learn about standard errors and confidence intervals and how to calculate them

Today's Learning Tools:

***Total number of activities*: 8**

Data:

- Synthetic data we make ourselves

Packages:

- `dplyr`
- `ggplot2`
- `mosaic`

Functions introduced (and packages to which they belong)

- `bind_rows()` : Combine data frame(s) together row-wise (`dplyr`)

- `geom_density()` : Geometry layer for density plots (`ggplot2`)
- `geom_errorbar()` : Draw error bars by specifying maximum and minimum value (`ggplot2`)
- `do()` : Loop for resampling (`mosaic`)
- `geom_vline()` : Geometry layer for adding vertical lines (`ggplot2`)
- `if_else()` : Tests conditions for true or false, taking on values for each (`dplyr`)
- `rnorm()` : Create synthetic normally distributed data (`base R`)
- `round()` : Rounds to nearest whole number or specified number of decimals (`base R`)
- `sample()` : Randomly sample from a vector or data frame (`mosaic`)
- `set.seed()` : Random number generator start point (`base R`)

5.1 Generalising About the World from Data

Last week we revisited a familiar sort of statistics: descriptive. But we also learned how to conduct these statistics using `R`. Today, we learn the other main branch of statistics: inferential.

Whereas descriptive statistics are concerned with summarising and describing your data, **inferential (or frequentist) statistics** are concerned with using the data to say something about the world in which we live. For example, we can make conclusions on body worn camera use in agencies across the country as a whole from data on only a handful of agencies. Using samples drawn from our population of interest, we can conduct statistical analyses to generalise to our lives and what we observe around us.

Inferences made from inferential statistics are not bound to one dataset and sample, and that is the strength of this type of statistics. It is able to *generalise*, like in the previous example on body worn cameras. Because, however, we will be saying something that is applicable to the 'real' world, we must understand the theory for which makes this possible.

Today's learning experience is the most theoretical of this course unit. To understand later inferential statistical analyses is to first understand the base on which they stand. Our three substantive topics today are: **samples**, **standard errors**, and **confidence intervals**.

5.1.1 Activity 1: Our preparation routine

As usual, we begin by opening your existing `R` project, then installing (if need) and loading the required packages listed above under the 'Packages' subheading.

5.2 Today's 3

As you continue the remainder of this course unit, you will observe how important it is to collect accurate information to conduct inferential statistics. Your findings and conclusions are only as good as their basis, and if that basis is a shoddy collection of data, what you have to say will reflect that. An important way to collect accurate information is to ensure that what we have is representative of that real world. This is where samples arrive to play.

5.2.1 Samples

Say we are curious about how widespread robbery in England and Wales has been in the past 12 months. We could obtain police-recorded data to tell us this information. We also, however, know from previous criminology classes that many people do not report crimes to the police, so this data is limited, unable to tap into what is known as 'the dark figure of crime'.

One way to address this limitation is with self-report victimisation surveys, such as the Crime Survey for England and Wales. It would be ideal to survey everyone in England and Wales about whether they have been a victim of robbery in the past year and how many times they have been robbed. Surveying the entire **population**, however, is impractical because of time and financial constraints. So, although the Crime Survey addresses the limitation of police-recorded data, it is still unable to obtain information on the entire population of interest – everyone living in England and Wales.

Sure, eventually mass collection of data from the population may be possible in the future as we have glimpsed with the sheer amount gathered by social media corporations, but even then, access and availability remain issues. Because of these problems in collecting information from the population, we use a sample.

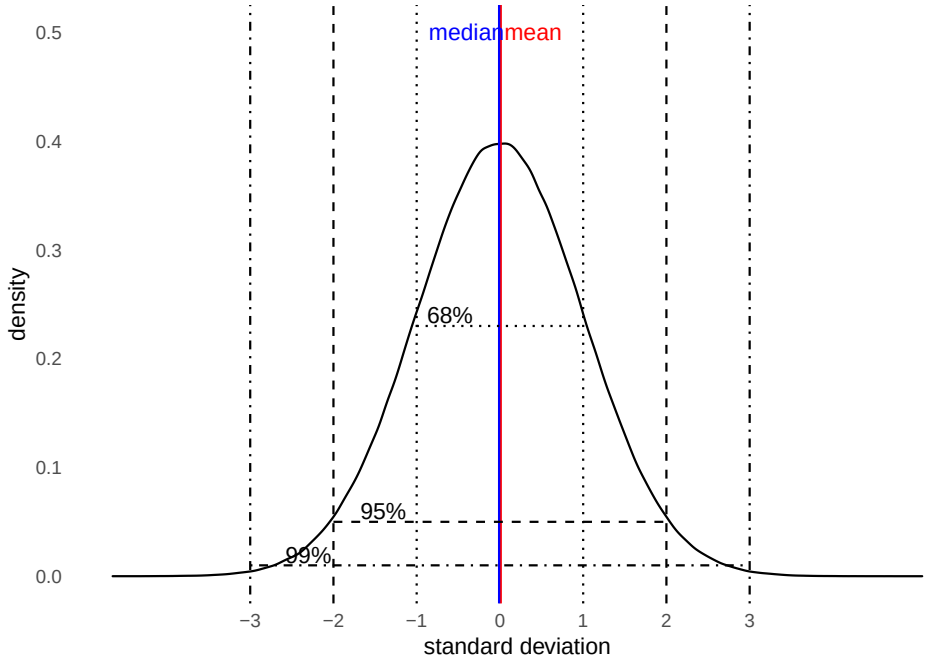
A **sample** is a small selection of that population of interest. You may recall from last semester and from your first year research methods classes the different approaches to sampling – different ways to select individuals to form your sample. Examples are random sampling and convenience sampling. The aim of good sampling is to acheive **external validity**.

If our sample has high external validity, it means that our sample is representative of our population of interest. Therefore, we can be confident that whatever we say and conclude about our sample can be generalised to the population. That is what the Crime Survey of England and Wales does: sample from the population to get an estimate of how widespread crime and victimisation are in the whole of the population.

How do we know that a sample is good at representing the population of interest? In the real world, as we have learned, it is often impossible to get data from the whole population. To illustrate how we can trust our statistics from our sample to represent the estimates in the population from which they are derived, also known as **parameters**, we will create a fake population from which we draw samples using **synthetic data**.

Last week, we learned about distributions. Specifically, we focused on the normal distribution. This is also called a bell curve, because when we squint a little, the shape looks like a bell. Remember that normal distrbutions are symmetrical (there is no skew) and the mean is the same as the median.

In addition, there were measures of distribution, or dispersion. For example, the standard deviation. Later we learn more about a nifty fact on the normal distribution. For now, the normal distribution below illustrates this nifty fact: 68% of your data will fall within +/- 1 standard deviation of your mean; 95% of your data within +/- 2 standard deviations of your mean; and 99% of your data within +/- 3 standard deviations of your mean.



Much of what we will be learning in the coming weeks will make the assumption that our data are normally distributed. Of course, there will be exceptions. This is something you will need to check for before embarking on further statistical analyses. For today, though, we focus on the normal distribution and we create synthetic data on intelligence quotient (IQ) scores of American probationers to show how a sample can be representative of the population.

5.2.1.1 Activity 2: Making normally distributed synthetic data

The synthetic data will consist og randomly generated numbers to represent the intelligence quotient (IQ) scores of every probationer in the US, which is a population of about 3.6 million. For this example, we assume the mean IQ scores to be 100 and the standard deviation to be 15. We create this population distribution by using the function `nrnorm()` and assigning this to a vector object called `prob_iq`. Within the `nrnorm()` function, we specify the parameters `n =`, `mean =`, and `sd =`. That is the **n**umber of observations we want (3.6 million, one for each of the probationers in the US, remember this is the **population**), the **mean** IQ score we want the population to have (that is 100, specified above), and the dispersion around this mean, given by the standard deviation in the `sd =` parameter (specified above as 15 IQ points).

```
# Inside nrnorm(), we specify the parameters:
# n = number of observations, which is 3.6 milion -- our population of US probationers
# mean = IQ score of 100
# sd = dispersion of 15 IQ points around the mean

prob_iq <- rnorm(n = 3600000, mean = 100, sd = 15)
```

We now have a vector of numbers, all randomly created. Let us get some descriptive statistics:

```
mean(prob_iq)
```

```
## [1] 99.99722
```

```
median(prob_iq)
```

```
## [1] 99.99733
```

```
sd(prob_iq)
```

```
## [1] 14.99602
```

By doing so, we are verifying that the mean and SD are indeed 100 and 15, but you may notice a few discrepancies:

- 1. The mean is not *exactly* 100 and the SD is not *exactly* 15, although they are very close to those values.

- 2. Your values may be slightly different to that of the lab notes. The reason is `R` *randomly* generates these numbers for you. If you re-run the code with `rnorm` above to re-create your `prob_iq`, you will get another set of numbers!

To ensure we have the same set of numbers, we can set a specific seed from which the random numbers should 'grow'. If we choose the same seed, then we will get the same set of random numbers and, thus, the same results.

We set a seed using the function `set.seed()`, which ensures it generates the exact same distribution for this session. Then we re-create the `prob_iq` object:

```
set.seed(1612) # Use this number!

prob_iq <- rnorm(n = 3600000, mean = 100, sd = 15)
```

Again, obtain the mean, median, and the SD to verify that we have the same results:

```
mean(prob_iq)
```

```
## [1] 100.0032
```

```
median(prob_iq)
```

```
## [1] 100.0035
```

```
sd(prob_iq)
```

```
## [1] 14.99679
```

The values, though slightly different from those stated in our `rnorm()` code, should match those in the lab notes.

With this set of random IQ scores, let us build a data frame using the function `data.frame()` in which we will create two columns: one for a unique identifier for each probationer called `probationer_id` and another for IQ scores called `IQ`. The data frame object will be called `prob_off`:

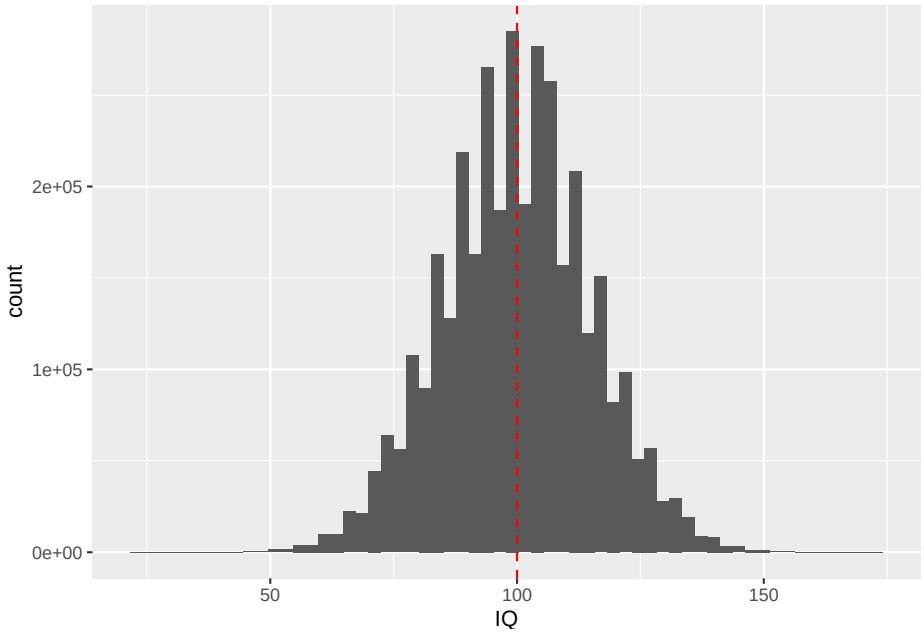
```
prob_off <- data.frame(probationer_id = 1:3600000,      # create a column whose number
                       IQ = prob_iq )     # create a column whose numbers are the rand
```

We now need one last step to complete our fake population data. The values for IQ score must be whole numbers (integers). To achieve this, we will use the `round()` function. In the `round()` function you specify two things, first the object you wish to round (the numbers in the IQ column of our prob_off dataframe so `prob_off$IQ`) and the number of decimals you would like displayed (since we want whole numbers, we want 0 decimals!).

```
prob_off$IQ <- round(prob_off$IQ, 0)
```

Now we finally have our complete fake population of 3.6 million US probationers and their IQ scores. You can have a look at this data with the `View()` function. Let's also visualise this distribution will help us identify its shape. We use `ggplot2`:

```
# Have you loaded the 'ggplot2' library?


ggplot(prob_off) +
  geom_histogram(mapping = aes(x = IQ), bins = 60) +
  geom_vline(xintercept = mean(prob_off$IQ), col = "red", linetype = "dashed") # We add a red li
```



Visualising the distribution of IQ scores, we observe that the scores are normally distributed, which is bell-shaped, and there is no skewness on other side. The majority of probationers have an IQ around the mean of 100.

5.2.1.2 Activity 3: Taking a sample from (our synthetic) population

So above we created a dataset of all probationers in the US, all 3.6 million of them. This is our **population**. When we look at the mean, median, and standard deviation of this population, these numbers are the *true* estimates of IQ scores in the population of American probationers.

Now if we take a sample from this population, how accurate would our sample estimates be compared to the population estimates?

First, we make a sample that is taken from our population. We draw a random sample of 100 probationers using the function `sample()` from the `mosaic` package:

```
library(mosaic)
```

In the `sample()` function, we specify two parameters. `x =` specifies what data set to sample from, and `size =` specifies the size of the sample to take. So in this case, to randomly select 100 probationers from the `prob_off` dataframe we use:

```
# We take a sample of 100 from our data frame, 'prob_off', and put it into an object c
sample1 <- sample(x = prob_off, size = 100)
```

Now that we have our sample `sample1`, we can take some descriptive statistics of 'sample1'

```
mean(sample1$IQ) # 101.59
```

```
## [1] 101.59
```

```
median(sample1$IQ) # 101.5
```

```
## [1] 101.5
```

```
sd(sample1$IQ) # 16.27485
```

```
## [1] 16.27485
```

The results seem very close to the 'true' estimates from our population of probationers. Let's try another sample with another seed:

```
set.seed(90201)
sample2 <- sample(x = prob_off, size = 100)
```

Now that we have our second, sample `sample2`, we can take some descriptive statistics once more:

```
mean(sample2$IQ) # 100.81
```

```
## [1] 100.81
```

```
median(sample2$IQ) # 102.5
```

```
## [1] 102.5
```

```
sd(sample2$IQ) # 15.08206
```

```
## [1] 15.08206
```

You can see we have slightly different numbers. This is because we took a different sample! The results are still close to our population measures, but they are different to sample 1. Alright, let's do this one more time!

```
set.seed(42)
sample3 <- sample(x = prob_off, size = 100)
```

Now the descriptives:

```
mean(sample3$IQ) # 99.41
```

```
## [1] 99.41
```

```
median(sample3$IQ) # 101
```

```
## [1] 101
```

```
sd(sample3$IQ) # 15.53497
```

```
## [1] 15.53497
```

Again, we are getting slightly different numbers, as we have sampled a separate 100 probationers from our population of 3.6 million. Depending on which sample we chose, our conclusions would be different. This variation in estimates is known as **sampling variability**, an unavoidable consequence of randomly sampling observations from the population.

For example, above, we took 3 different samples. Let's look at the mean of each sample again:

```
mean(sample1$IQ)
```

```
## [1] 101.59
```

```
mean(sample2$IQ)
```

```
## [1] 100.81
```

```
mean(sample3$IQ)
```

```
## [1] 99.41
```

Each time we get a slightly different value for the mean, depending on the sample which we look at.

Sampling variability

Getting different estimates each time we randomly sample from the population is a real problem facing researchers: each time you take a sample from your population of interest, we need to be confident that its estimates are similar to the true but unknown estimates of that population.

Sampling variability makes up what is known as the **sampling distribution**. Sampling distribution of a sample statistic (for example the mean) refers to the distribution of that value if we were to take many many many many many many many samples, and each time, record the value. Then if we took all of these values, they will follow a normal distribution, the mean of which will be the true population value! This is really cool, and we will explore it in the next activity.

5.2.1.3 Activity 4: The sampling distribution

Let's demonstrate this with taking 1,000 samples of 100 people each, from the 3.6 million probationers. Imagine we got funding to run 1,000 surveys, each time we take a random sample of 100 probationers, and we take the mean IQ for the sample.

Remember the function to get one sample of 100 people above? It was `sample(x = prob_off, size = 100)`. Now to repeat this 1,000 times we can use the `do()` function, which tells R to do something multiple times. So to take 1,000 samples of 100 probationers we want to *do* `sample(x = prob_off, size = 100)` 1,000 times, like so:

```
sample1000 <- do(1000) * sample(x = prob_off, size = 100)
```

This might take a while (you are sampling 1,000 times after all!) so give R time to process. When done, you will see the sample1000 object appear in your environment. Have a look at it with `View()`.

You can see we have the prisoner ID, and we have the IQ score, but we also have these new variables, `.row` and `.index`. `.row` refers to the probationer's position in their particular sample, while `.index` refers to the sample into which they belong. So if the `.index` says 1, they were in the 1st sample, 2, the 2nd sample, and so on up to 1000. To select a particular sample for example the 2nd one, you could use the `filter()` function in order to select all cases where `.index == 2`. But we want to keep all 1,000 samples. Instead, what we want is the **average IQ for each sample**.

You may recall how to get the mean for each value of a categorical variable from last week! Specifically, we used `group_by()` and `summarise()`. Let's use these again, to create a new object, `sample_means1000`, which has the mean IQ for each sample (all 1,000 of them!):

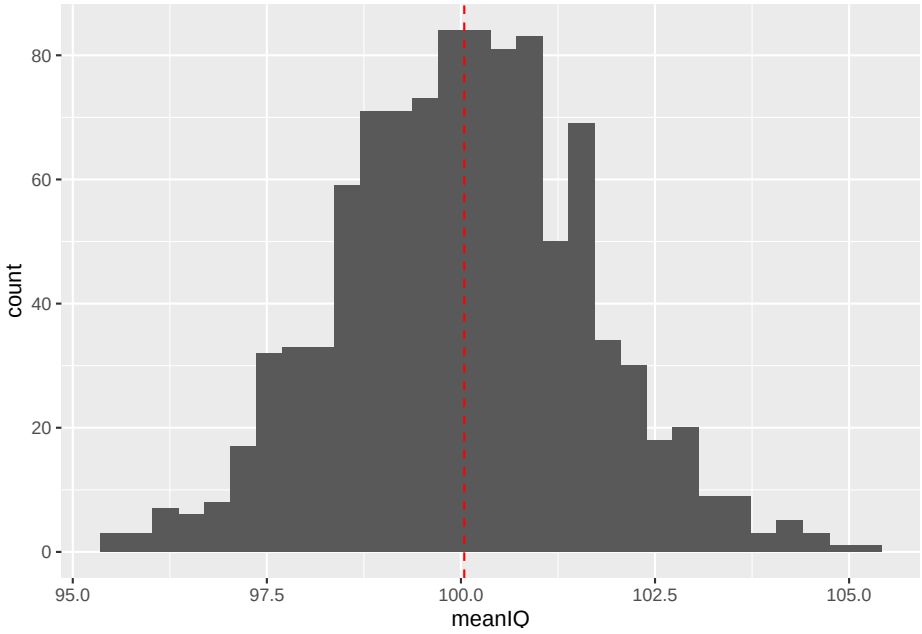
```
sample_means1000 <- sample1000 %>%
  group_by(.index) %>% # Group by .index  (the sample id)
  summarize(meanIQ = mean(IQ)) # Creating new variable of mean IQ
```

The resulting dataframe (`sample_means1000`) has 2 columns, one for sample id and one for the mean score of IQ for that specific sample. It has 1,000 observations - one for each sample!

So what does our variability look like? We can visualise this sampling distribution to compare to the previous population distribution

```
ggplot(data = sample_means1000) +
  geom_histogram(mapping = aes(x = meanIQ)) +
  geom_vline(mapping = aes(xintercept = mean(meanIQ)), col = "red", linetype = "dashed")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The histogram of our sampling distribution shows this very important concept in inferential statistics: in the case of an **unbiased estimator** what we see is that whenever we draw a sample, sometimes we *over estimate* the parameter, and sometimes we *under estimate* the parameter. In this case, this means that sometimes our sample mean is *larger* than our population mean, and other times our sample mean is *smaller* than the population mean. Usually the mean of the randomly selected sample will fall close to the populatin mean, but occasionally, it will fall far. Now - what is really exciting, is that if you randomly draw repeated samples from the same population and calculate the mean of each sample, then plot the frequency of those means, you will get the *normal distribution* – that bell-shaped curve! It doesn't even matter if your underlying data are normally distributed! Your sample statistics will be!

This indicates that most samples drawn from the population will have a mean close to the true population mean, and in general over/under estimate it in about equal amounts!

So according to our sampling distribution of probationer IQ scores, drawing a sample with a mean IQ score that is radically different from that of the population would be unlikely. See above, how 68% of observations in a normal distribution fall within +/- 1 standard deviation of the mean? And 95% within +/- 2 standard deviations. This should be reassuring!

However in real life, you are not ever going to conduct 1,000 surveys on 100 probationers each. Instead, you will have one survey, and you will have to make

sure that your one sample is a good one. Random sampling is one approach, we see here, but another thing to think about is that of the **sample size**. The next activity explores this.

5.2.1.4 Activity 5: Sample sizes

We came out pretty well in the above example, with our samples of 100 probationers. But is 100 a good number? What if we had repeated samples of 30 instead of 100. What about 1,000?

Well let's create 3 sets of 1,000 samples this time, to see how the sample size might affect our sampling distribution of our mean!

Let's make the 3 samples new dataframe objects, and call them `sample30`, `sample100`, and `sample1000`:

```
# 30 probationers in each sample
sample30 <- do(1000) * sample(x = prob_off, size = 30)

# 100 probationers in each sample
sample100 <- do(1000) * sample(x = prob_off, size = 100)


# 1000 probationers in each sample
sample1000 <- do(1000) * sample(x = prob_off, size = 1000)
```

Now we have 3 dataframes, all with 1,000 samples of varying sample sizes. The first with 1,000 30-person samples, the second with 1,000 100-person samples, and the 3rd with 1,000 1000-person samples. Which one of these would you trust to best represent the population? Why? Do you come across anything like this in your own lives? Discuss in your groups if you're in the chatty rooms!

In the meantime, let's also create a new dataframe for each set of 1,000 samples where we calculate the mean IQ for each one of them.

```
# Calculate the means IQ scores for each sample

sample_means30 <- sample30 %>%
  group_by(.index) %>%
  summarize(meanIQ = mean(IQ))

sample_means100 <- sample100 %>%
  group_by(.index) %>%
  summarize(meanIQ = mean(IQ))
```

```
sample_means1000 <- sample1000 %>%
  group_by(.index) %>%
  summarize(meanIQ = mean(IQ))
```

We now have 3 dataframes, each with 1,000 samples, but one with 30 people per sample, one with 100 peope per sample, and one with 1,000 people per sample. Let's bind these together, but first, for each one, create a new column called "sample_size" which tell us which one has how many people in each sample.

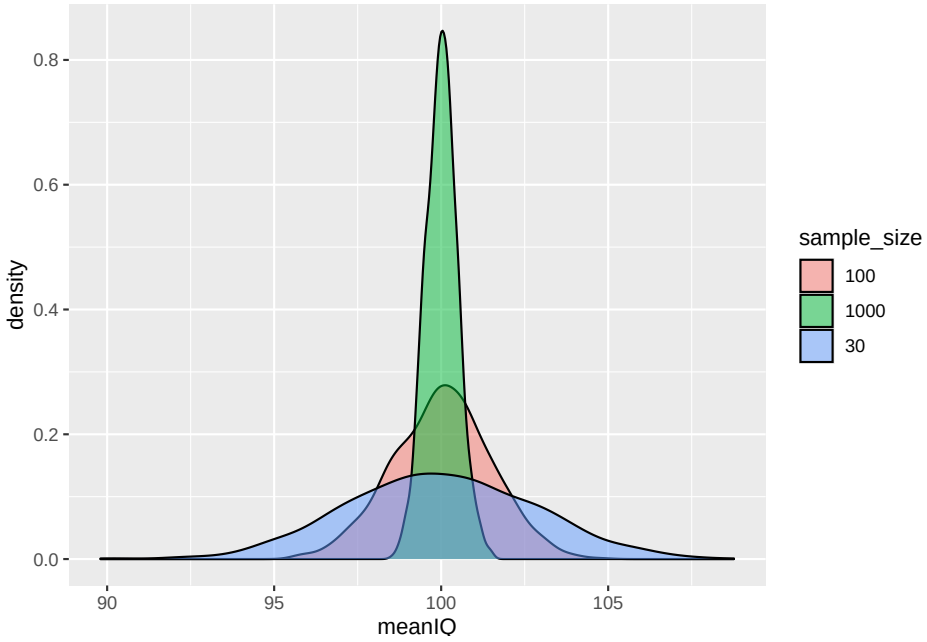
```
sample_means30 <- sample_means30 %>% mutate(sample_size = "30")

sample_means100 <- sample_means100 %>% mutate(sample_size = "100")

sample_means1000 <- sample_means1000 %>% mutate(sample_size = "1000")
```

We use the function `bind_rows()` to *bind* these different data frames of different sample sizes into one complete dataframe, to answer our question:

```
sample.means.total <- bind_rows(sample_means30, sample_means100, sample_means1000)
```

Now if you look at this dataframe, it has 3,000 samples in it, 1,000 with 30 people in each, 1,000 with 100 people in each, and 1,000 with 1,000 people in each. Let's look at the distribution of the mean IQ for each of these three sets. We can use `geom_density` to create a density plot. Let's fill by sample_size, and let's set the opacity (`alpha =` to 0.5 so we can see through each one for they will overlap)

```
# Density plot for comparison
ggplot(data = sample.means.total) +
  geom_density(mapping = aes(x = meanIQ, fill = sample_size), alpha = 0.5)
```



From the density plot, all three sample distributions are normally distributed and have similar means to that of the population. (remember that the mean of the sampling distribution will be the true population parameter!)

Notice, however, that the *larger the sample size, the more likely that the sample means are closer to those of the population.* The distribution of sample sizes of 1,000, for example, is tight and pointy, indicating that the IQ scores cluster very closely to the mean of the sampling distribution (and therefore the true population mean).

Whereas if we look at the distribution of the sample sizes of 30, it is flatter and wider - its scores are more spread away from the true population mean. The implication is that if we draw small sized samples, we have a higher chance of having a sample that does not reflect the true population at all. With small sample, we run the risk of getting a sample statistic that is further away from the population parameter than we do with larger samples. So generally, larger the sample the better. Otherwise, our findings and generalisations will be inaccurate.

So how do we know what sample size is big enough? For this, we carry out something known as a **power analysis** but we will get to that next week. For now, think "bigger is better" when it comes to sample size!

So great, we can say all these things in our hypothetical world of the many many many samples from the same population. But like we said earlier, we don't really get a chance to do that in the real world. So how can we trust our samples? To do this, we can quantify our sample variabilty using **standard**

errors.

5.2.2 Standard Errors

5.2.2.1 Activity 6: The central limit theorem

We can summarise the variability of the sampling distribution in an estimate called **standard error** (SE). It is essentially the standard deviation of the sampling distribution. We demonstrate how sample size affects the SE, in that the larger the sample size, the smaller the SE and vice versa:

```
sd(sample_means30$meanIQ)
```

```
## [1] 2.802161
```

```
sd(sample_means100$meanIQ)
```

```
## [1] 1.468766
```

```
sd(sample_means1000$meanIQ)
```

```
## [1] 0.474146
```

You can see that as the sample sizes get larger, we see a smaller result for the standard deviation of our sampling distribution - or the standard error of our sample.

What we have learned is succinctly referred to as the **Central Limit Theorem**. This theorem states that as sample sizes get larger, the means of the sampling distribution approaches normal distribution; it is able to reflect the true population estimate.

With the synthetic data, we have demonstrated how samples can estimate the population, which is usually unknown to us. The SE is helpful for when we want to know the extent to which the mean of the sample we have, drawn from a population whose estimates are unknown to us, is an accurate estimation of the true mean in that population.

But I promised a way to quantify our sample variability without having to do 1000 repeat samples. Well, excitingly, we can calculate the standard error from just one sample! Let's do this by using a sample of 1,000 people. Let's make a new one of these:

```
set.seed(1234)
new_1000_sample <- sample(prob_off, 1000)
```

Now we can get the standard error by dividing the standard deviation of the IQ variable in this sample by the square root of our sample size (in this case 1000):

$SE = \sigma/\sqrt(n)$

In R:

```
sd(new_1000_sample$IQ)/sqrt(1000)
```

```
## [1] 0.4887421
```

The SE is 0.4887421. What does this mean? Well remember that the standard error is the standard deviation of the sampling distribution of our sample statistic (here the mean). And remember what percentage of observations, within a normal distribution, fall within some standard deviations away from the mean?

- 68% between +/- 1 standard deviation from the mean
- 95% between +/- 2 standard deviations away from the mean
- 99% between +/- 3 standard deviations away from the mean

So, with our standard error above, we can say that almost all of the samples (95%) will produce a statistic (in this case a mean) which are within +/- 2* 0.4887421, so within +/- 0.9774842 or about 1 IQ point away from the true mean IQ for the whole population of 3.6 million probationers!

This is how we can use the theory of the sampling distribution of the mean to then look into our one sample, and be able to estimate how representative are the estimates we derive from them about the whole population. How cool is that?! It is the power of statistics all in our hands!

5.2.3 Confidence Intervals

The last thing we will learn about is a better way of communicating the extent of inaccuracy in sample estimates. Communicating uncertainty when talking about statistics is an incredibly important topic! In statistics, we are making generalisations, we are making inferences about a population based on some data we collected from a sample. This means that we will always have some element of error and uncertainty in our conclusions.

One way to clearly quantify and communicate our uncertainty is to use **confidence intervals** (CIs). These appear as an interval that tells you the margin of error – far away is your sample statistic from the population parameter. We calculate them by, first, returning to our normal distribution, and its **68-95-99 rule**, or known as an empirical rule, which states that 68% of cases in the distribution will fall within one standard deviation above and below the mean; 95% within two SD; and 99% within three SD.

5.2.3.1 Activity 7: The 68-95-97 rule in action

Two observations to note: first, last week we learned about standard deviations and that there was mention of 68% of verbal assaults falling within one SD; it was a reference to this rule. Second, there is a contradiction with the numbers. If 95% of values fall within 1.96 SD, then why does the empirical rule state that 95% of values will fall within 2 SD, and why have you been saying this so far? Well, I'm simply rounding up. The former (1.96) is the precise number and the latter (2) is an approximation, meant to help you memorise this rule easier than if the value was a non-integer like 1.96.

If 95% of values of the normal distribution fall within 1.96 SD of the mean, we are able to calculate the upper and lower boundaries of this particular confidence interval using this 1.96 value (also known as z-value) from our sample using the following formula:

$\bar{x} \pm 1.96 * sd/\sqrt{n}$

Let's look back at our `sample1` object. Then take the mean of IQ in this sample:

```
mean(sample1$IQ, na.rm = TRUE)
```

```
## [1] 101.59
```

Now to create the lower bound for the confidence interval around this sample statistic, we take this mean minus 1.96 times the standard deviation:

```
mean(sample1$IQ, na.rm = TRUE) - 1.96*sd(sample1$IQ)/sqrt(100)
```

```
## [1] 98.40013
```

And to get the upper bound, you add the same value (1.96*the standard deviation) to the mean:

```
mean(sample1$IQ, na.rm = TRUE) + 1.96*sd(sample1$IQ)/sqrt(100)
```

```
## [1] 104.7799
```

Seeing the dashes that represent the confidence interval shows us that IQ scores will vary away from the mean of our sample, but 95% of them will fall within this interval. Similar to what we learned about repeated samples, if we took 100 resamples of our population of probationers and obtained the sample means, the true population mean will fall within the confidence interval 95% of the time. Thus, only 5% of the time will our resamples fail to obtain the true population mean.

So here, we can conclude from our sample that the mean IQ for all probationers will be somewhere between 98.4001302 and 101.908987.

What if we took a different sample though? You can repeat the steps above for sample 2 and sample 3, and you will see the following conclusions:

- **Sample 2**: we conclude that the mean IQ for all probationers will be somewhere between 97.8539159 and 103.7660841.
- **Sample 3**: we conclude that the mean IQ for all probationers will be somewhere between 96.3651461 and 102.4548539.

With each different sample we get a slightly different upper and lower bound. So how can we trust this? Well since we know that we have 95% of observations within +/- 1.96 standard deviations from the mean of the sampling distribution, we can conclude that on the whole, the confidence intervals derived from 95% of our samples will contain the true population parameter!

Don't believe me? Let's plot this!

First let's take our population parameter, the true mean IQ for all probationers in the US:

```
# Create a vector containing the true population mean from prob_off
true.mean <- mean(prob_off$IQ)
```

Then, let's take another 100 samples of 100 probationers in each sample:

```
set.seed(1897)
new_sample_100 <- do(1000) * sample(prob_off, size = 100)
```

Now for each sample, calculate the mean and the lower and upper CIs (remember above the formula to get these CIs!)

```
# Select the sample of 1,000 samples, each with 100 probationers and place in object, 'new.sample
new.sample.ci100 <- new_sample_100 %>%
  group_by(.index) %>%
  summarise(sample_mean = mean(IQ),
            sample_sd = sd(IQ),
            lower_ci = sample_mean-1.96*sample_sd/sqrt(100),
            upper_ci = sample_mean+1.96*sample_sd/sqrt(100))
```

If you have a look at this new dataframe `new.sample.ci100` you can see that for each one of our 100 samples, we have the sample mean, as well as the sd and the calculated upper and lower CI. Let's revisit what we learned from recoding, and use the `if_else()` function to create an additional variable, which tells us whether or not each one of the CIs contains the true population mean (`true.mean` above).

```
new.sample.ci100 <- new.sample.ci100 %>%
  mutate(capture.mean = if_else(condition = lower_ci > true.mean | upper_ci < true.mean
# If not, capture.mean will be "no"
```

We can now use this to produce a table to show new variable and how many Cis captured the true population mean:

```
table(new.sample.ci100$capture.mean)
```

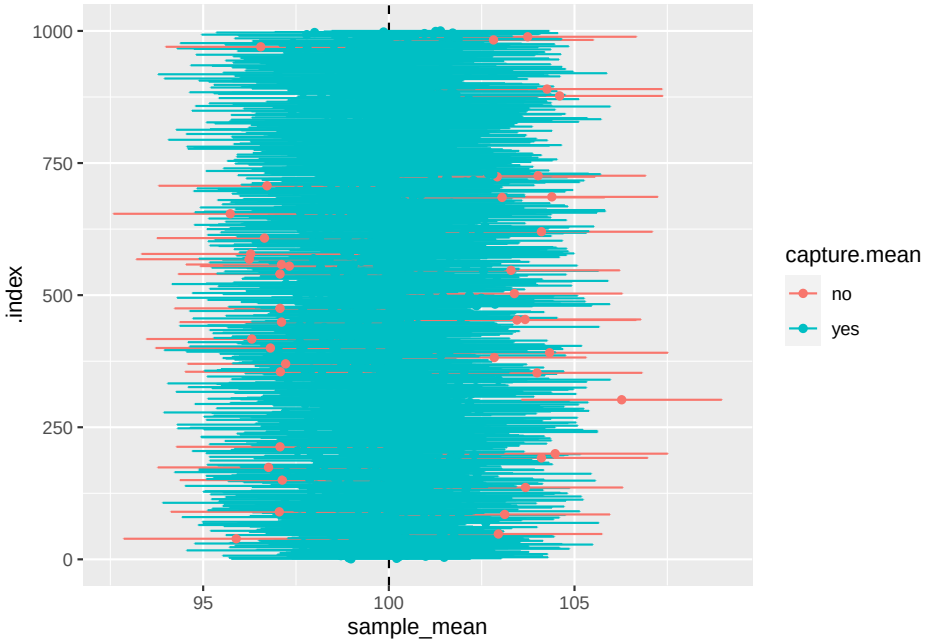
```
##
##  no yes
##  42 958
```

Looks like our samples fared better than we would hope, in this example 958% contained our true population mean, and 42% did not, but it's still close to what we expected. What if we were to do this with 1000 samples of 100? What do you think the yes vs nos would look like then? What about 10,000 samples? Discuss in your groups.

5.2.3.2 Activity 8: Visualising confidence intervals

Finally, to illustrate, we can visualise this to better understand what we have just found:

```
ggplot(data = new.sample.ci100) +
  geom_vline(mapping = aes(xintercept = true.mean), linetype = "dashed") +
  geom_errorbarh(mapping = aes(xmin = lower_ci, xmax = upper_ci, y = .index, colour =
  geom_point(mapping = aes(y = .index, x = sample_mean, colour = capture.mean))
```



The visual shows the result obtained in the table, but here, you can see all 100 of the scores, their CIs, and how 95% of the time, they capture the true population mean. In reality, we have no way of knowing whether we have captured the true population estimates in our sample, but the use of the confidence interval gives us *confidence* that we are on the right track, so reporting CIs in your results is good practice for presenting your findings.

5.3 SUMMARY

Today was a theoretical demonstration of why **samples** can be used to estimate what is happening in the **population**. Samples with high **external validity** can do so. This is the foundation of inferential statistics, the use of samples to draw conclusions about the population. We used **synthetic data** to show why. Despite **sampling variability**, the means of the **sampling distribution** demonstrate that it is able to approximate the normal distribution and, therefore, the true population estimates. This is further demonstrated by the **central limit theorem**, which clarifies that sample size matters in producing more accurate estimates of the population. We learned about the standard error and then onto **confidence intervals**, which is useful in establishing how accurate our estimates are, because in reality, rarely are the population estimates known.

Homework time!

Chapter 6

Hypotheses

Statistical Significance, Binominal Test, Single Sample Significance Tests

Learning Outcomes:

- Know what hypotheses are and how to use them in inferential statistics
- Understand what statistical significance is and how to interpret p-values
- Know what hypothesis tests are and how to conduct a few of them in `R` using the z and t values

Today's Learning Tools:

Data:

- Synthetic data

Packages:

- `DescTools`
- `dplyr`
- `ggplot2`
- `tigerstats`

Functions introduced (and packages to which they belong)

- `BinomCI()` : Compute confidence intervals for binomial proportions (`DescTools`)

- `cat()` : Combines/concatenates character values and prints them (`base R`)
- `nrow()` : Counts the number of rows (`base R`)
- `pnorm()` : Probability of random variable following normal distribution (`stats`)
- `pnormGC()` : Compute probabilities for normal random variables (`tigerstats`)
- `prop.test()` : Test null hypothesis that proportions in groups are the same (`base R`)
- `prop_z_test()` : Single-sample z-test for proportions we created in this chapter
- `scale()` : Mean centers or rescales a numeric variable (`base R`)
- `which()` : Provides the position of the elements such as in a row (`base R`)
- `z_test()` : Function created in this chapter for a single-sample z-test

---

6.1 Hypothesis Testing

Last time we were introduced to inferential statistics, which uses a sample to draw conclusions about the population. We did not cover types of analyses, however. Instead, as a basis for understanding why samples can be used to say something about the population, we learned the theory that demonstrated why this was so.

Today we learn the first step in making predictions about our world: hypothesis testing. In crime and criminal justice research, **hypotheses**, essentially predictions, are made frequently – a new intervention programme will reduce reoffending; low self-control is predictive of later criminal behaviour; adverts on reporting sexual harassment on public transport will increase awareness and reduce sexual victimisation rates.

When we test our hypothesis, we test to see if our prediction is true for the population of interest. For example, if we hypothesize that the new intervention programme will reduce reoffending among at-risk young people in Manchester, and our result shows this, we then can generalise our result to the population of all at-risk young people in Manchester.

In hypothesis testing, we aim to reject the **null hypothesis**. This states that there is no relationship between our variables. Using the previous example, our null hypothesis (H_0) would be that the new intervention programme will not reduce reoffending – there will be no difference in reoffending rates between those who participated and those who did not participate in the programme. Our hypothesis (H_A) is considered the *alternative* (hence the 'A') of the null

hypothesis, which means that there is a relationship between our variables, or in other words, a difference between groups.

Example Hypotheses :

H_0: There is no difference in reoffending rates between those who participated and those who did not participate in the new intervention programme

H_A: There is a difference in reoffending rates between those who participated in the new intervention programme and those who did not participate in said programme

Therefore, when we make predictions, we always state two hypotheses: H_0 and H_A. Stating hypotheses can take two forms: a **directional hypothesis** where you specify the direction or the relationship expected and a **non-directional hypothesis** where you are only interested in whether there is a difference between groups. Our example is a non-directional hypothesis because we do not state specifically whether we want reoffending rates to be lower or higher in each group of young people.

Confidence is key in hypothesis testing. If we reject the null hypothesis, we want to be confident that it is actually false. In criminology, and across the social sciences, we want to be at least 95% confident in our result. This implies that we are willing to be wrong 5% of the time. We cannot be 100% confident because we usually do not have information from the entire population, so find ourselves trying to decide whether our null hypothesis is false without being so sure of the true result in the population. Because of this uncertainty, we need to be wary of **Type 1 error**. This error, known also as a false positive, is when we reject the null hypothesis when it is actually true. In addition, there is also *Type 2 error*, a false negative, which is when we fail to reject the null hypothesis even though it is actually false.

Hypotheses are created before the researcher collects outcome data for the study and conducts any analysis. This is good practice and ethical as well, because if the hypotheses are stated after data has been collected and analysed, then the researcher may be tempted to change the hypotheses, so will dishonestly influence the tests of statistical significance. Also, this affects Type 1 error because we are increasing its likelihood through bad practice – we want to leave that 5% of getting it wrong to chance only.

As we are concerned with rejecting the null hypothesis with high confidence that it is actually false, we will need to identify the risk of making a type 1 error and hope that the risk is as small as possible in a test of **statistical significance**. How to calculate this risk?

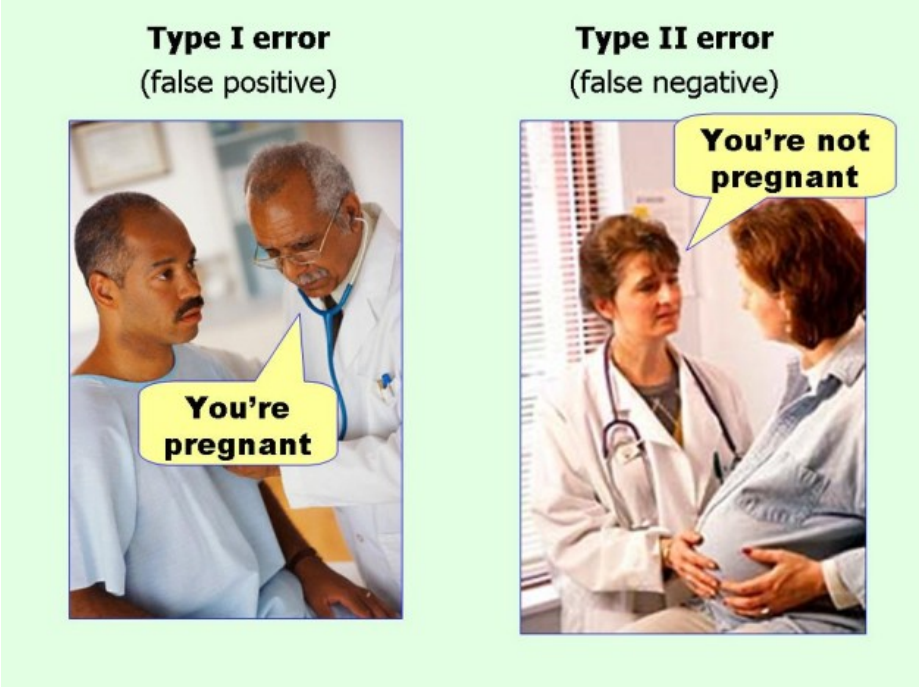


Figure 6.1: **Figure 6.1** Type 1 and Type 2 errors

6.2 Today's 3

To understand hypotheses in action, we learn three substantive concepts today: **statistical significance**, **binominal test**, and **single sample significance tests**.

6.2.1 Statistical Significance

What is meant by **statistical significance**? It is a misleading term because it seems to mean that the result is significant or important, but that is not correct. When we test for statistical significance, we are testing to see if the probability of the null hypothesis being true is less than the level of type 1 error specified.

The level specified is called the **significance level** and is denoted by the alpha-level ( ). In the social sciences, it is usually set at $= 0.05$ to indicate that we want to be 95% confident in our rejection of the null hypothesis. If the probability of the null hypothesis is less than $= 0.05$, for example, then we can reject it. If, however, the probability is greater than $= 0.05$, then we have failed to reject the null hypothesis.

The probability obtained is the **p-value**, short for probability value, and it tells us how likely that certain result or effect will happen if the null hypothesis is true. That is why if the probability is less then 5%, then it is unlikely to happen.

The terminology is important here: we can only say we 'reject' or 'fail to reject' the null hypothesis; we cannot say we 'accept' the null hypothesis because testing for statistical significance is not about finding out if the null hypothesis is correct.

Another important point: lately, there has been substantial calls to get rid of p-values and statistical significance. The problem is that the p-value is often misunderstood and even misused. It has come to be misinterpreted as either the study worked (a good study) or did not work (a bad study). **That is why including confidence intervals is good practice**. Even though the practice of relying on p-values is controversial, we learn about statistical significance in this course unit because many studies still use it. Understanding what it actually is will prevent misinterpretation.

Now that we have established what is statistical significance, we must identify the most appropriate test for statistical significance. Selecting an appropriate test depends on a number of assumptions. The remainder of the course unit will introduce you to a number of these tests. For today, we learn about specific hypothesis tests using the binomial distribution and then the normal distribution, and each have their own set of assumptions. We begin with hypothesis testing using the binomial distribution.

6.2.2 A Binomial Test

Hypothesis testing using the binomial distribution have the following assumptions:

1. *Level of measurement:* the variable is binary, meaning it measures only two possible categories.
2. *Shape of the population distribution:* None
3. *Sample:* high external validity
4. *Hypothesis:* stated before collection and analysis of data

We will, once again, create synthetic data of an evaluation of our example intervention programme. The sample comprises 100 at-risk young people who are randomly assigned to two groups: 50 to a treatment group that receives the intervention that aims to prevent them from committing future offences and 50 to the control group that receives no intervention. We evaluate this programme to see if it works and if it can be generalised to all at-risk youth in Manchester. As the outcome has only two options – success and failure – we rely on the **binomial distribution**.

In `R`, we run the `prop.test ()` function to test the null hypothesis that the proportions, or probabilities of success, are the same or equal in several groups. Say we find that in the treatment group, 10 out of the 50 went on to reoffend whereas, in the control group, 28 out of the 50 went on to reoffend.

We could stop here and conclude, 'Yes, there is a difference between groups, and by golly, the intervention works.' But remember that chance, or random variation, is inherent in all phenomena, so this observation could just be a mere fluke. That is why we conduct a test of statistical significance. This one is called a 'two-sample proportion test':

```
# The first concatenate (c () ) contains the numbers that went onto reoffend
# The second concatenate contains the total numbers in each group
prop.test(x = c(10, 28), n = c(50, 50))
```

```
##
##  2-sample test for equality of proportions with continuity correction
##
## data:  c(10, 28) out of c(50, 50)
## X-squared = 12.267, df = 1, p-value = 0.0004611
## alternative hypothesis: two.sided
```

```
## 95 percent confidence interval:
##  -0.5567014 -0.1632986
## sample estimates:
## prop 1 prop 2
##   0.20   0.56
```

The output is a little ugly, but you can refer to the help documentation for more detail on the output by typing `?prop.test`. Let us focus on some of the results:

- P-value: probability of the null hypothesis is true – it is less than $= 0.05$
- Alternative hypothesis: the direction of the hypothesis – We expect between-group differences in either direction so it is two-sided
- prop1 and prop2: proportions attributed to each group – 20% in one group (10/ 50) and 56% in the other (28/ 50)

The p-value is less than the specified $= 0.05$, so we can say that the difference is statistically significant. We have evidence to reject the null hypothesis that no difference in reoffending exists between the treatment and control groups. This suggests that *something* is happening because of the intervention, but we are not sure what.

Now, if we want a directional hypothesis? For example, we expect that reoffending reduces in the treatment group compared to the control group:

```
# We add 'alternative=' and specify 'less' to indicate we expect the treatment group to have a sm
prop.test(x = c(10, 28), n = c(50, 50), alternative = "less")
```

```
##
##  2-sample test for equality of proportions with continuity correction
##
## data:  c(10, 28) out of c(50, 50)
## X-squared = 12.267, df = 1, p-value = 0.0002306
## alternative hypothesis: less
## 95 percent confidence interval:
##  -1.0000000 -0.1917075
## sample estimates:
## prop 1 prop 2
##   0.20   0.56
```

Again, the p-value is less than $= 0.05$, so we have sufficient indication to reject our null hypothesis. In other words, there is evidence to suggest that the at-risk youth in the treatment did offend less than in the control group due to the intervention, and this finding is statistically significant – we can generalise this result to the population which they represent.

Now if, for example, we found out that the therapist hired to deliver the intervention was a fraud, we may be worried the treatment group did worse than the control group. We then would expect a directional hypothesis where reoffending is higher in the treatment group than that of the control. We run the two-sample proportion test again, but with a slight difference to the direction:

```
# We specify 'greater' following 'alternative' to indicate our expected direction for
prop.test(x = c(10, 28), n = c(50, 50), alternative = "greater")
```

```
##
##  2-sample test for equality of proportions with continuity correction
##
## data:  c(10, 28) out of c(50, 50)
## X-squared = 12.267, df = 1, p-value = 0.9998
## alternative hypothesis: greater
## 95 percent confidence interval:
##  -0.5282925  1.0000000
## sample estimates:
## prop 1 prop 2
##   0.20   0.56
```

The p-value is greater than the specified 0.05, meaning that we have failed to reject the null hypothesis, and we do not have adequate evidence to support our hypothesis that our intervention increases reoffending.

With this example, we have committed some bad practice: we did multiple hypothesis tests. This is a no-no: stick to one hypothesis and test that. Our purpose here, however, was to demonstrate how to run the binomial test. Recall that it is important to state your hypotheses before you collect and analyse your data.

Including confidence intervals (CIs) is good practice, and it is possible to create them for binomial proportions. For example, we would like to build CIs around the proportion of the outcome for each group. We use the `BinomCI ()` function in the `DescTools` package to do so:

```
# CIs for treatment group where 10 reoffended
BinomCI(10, 50)
```

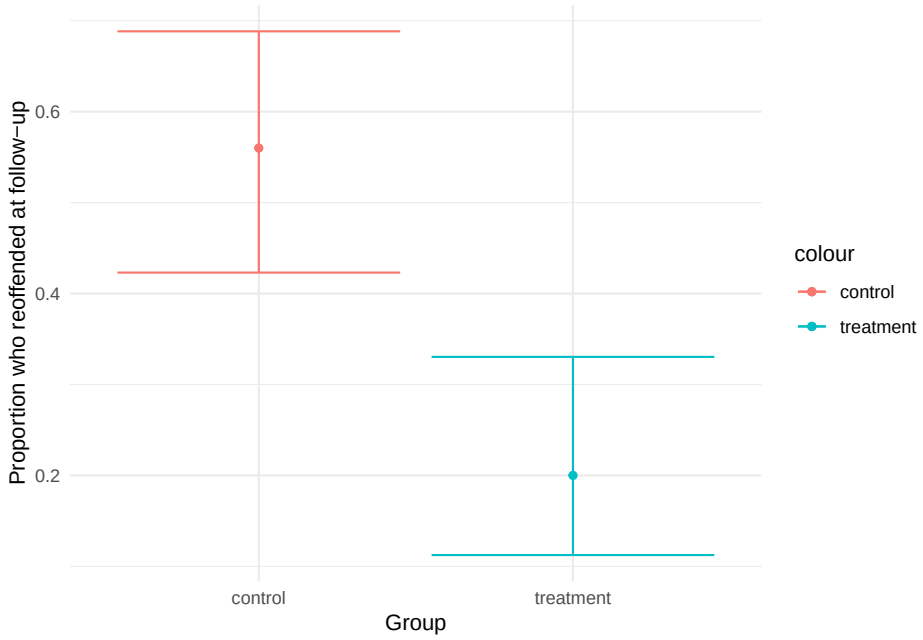
```
##      est    lwr.ci    upr.ci
## [1,] 0.2 0.1124375 0.3303711
```

```
# CIs for control group where 28 reoffended
# By default, confidence level is 95% but have added in 'conf.level' to show that you
BinomCI(28, 50, conf.level = 0.95)
```

```
##       est    lwr.ci   upr.ci
## [1,] 0.56 0.4230603 0.688378
```

In the CI for the treatment group, 11 to 33% of young people exposed to the intervention will reoffend, whereas the CI for the control group indicates that 42 to 69% of the young people not exposed to the intervention will reoffend. This seems like a large difference. To get a better understanding, we visualise this using `ggplot`:

```
# Taking the previous coding and placing them in objects
treatment_group <- BinomCI(10, 50, conf.level = 0.95)
control_group <- BinomCI(28, 50, conf.level = 0.95)

# Creating two error bar layers, one for each group
ggplot() +
  geom_errorbar(mapping = aes(ymin = treatment_group[2], ymax = treatment_group[3], x = "treatmer
  geom_point(mapping = aes(y = treatment_group[1], x = "treatment", colour = "treatment")) + geom
  geom_point(mapping = aes(y = control_group[1], x = "control", colour = "control")) +
  xlab("Group") +
  ylab("Proportion who reoffended at follow-up") +
  theme_minimal()
```



Visualising our results, we see further support that the intervention reduces reoffending: the confidence intervals for each group do not overlap and the proportion for the control group is lower than that of the treatment group.

When the CIs between groups do not overlap, this is good because it indicates that the two groups likely come from two different populations.

6.2.3 Single-sample significance tests

For this section, we learn some hypothesis tests that use the normal distribution.

Hypothesis testing using the normal distribution have the following assumptions:

1. *Level of measurement:* the variable is interval or ratio level
2. *Shape of the population distribution:* normal distribution
3. *Sample:* high external validity
4. *Hypothesis:* stated before collection and analysis of data

Unlike the binominal test, which compared groups, we will be comparing a single group – our sample – to the population. This may sound strange because we have learned that information about the population is rare, so we must make do with uncertainty. In some cases, however, we may know the population parameter and these tests can be used. When comparing our sample to a known population, we use the z- distribution; if we have to compare with an unknown population, we use the t-distribution. The z- and t- distributions are types of normal distributions.

The normal distribution has some predictable characteristics about it. One is that half of the distribution will always be below the mean, and the other half will be above the mean. We demonstrate this by creating a synthetic data of 1.5 million US prisoner IQ scores, drawn from a population that is normally distributed (= 100; SD = 15). We then test whether half of our population have an IQ above the mean. We introduce two new functions: `which ()` to select a subset of prisoners who have an IQ of 100 + and `nrow ()`, which divides the number of prisoners with an IQ of 100+ by the total number of prisoners:

```
# Make synthetic data which includes the variables 'prisoner_id' and 'IQ'
# 'prisoner_id' has 1 to 1.5 million IDs while 'IQ' has scores generated by 'rnorm' fu
# Place data frame in object called 'PrisonerIQ'
PrisonerIQ <- data.frame( prisoner_id = 1:1500000, IQ = round(rnorm(1500000, mean = 10

# Using 'which' function to make subset of prisoners with IQ above 100
# 'which' is to the left of the ',' at the end of code to specify that we are selectin
# Place subset in object called 'IQ_Over_100'
IQ_Over_100 <- PrisonerIQ[ which(PrisonerIQ$IQ>100) ,]
```

```
# Divide 'IQ_Over_100' by total number of prisoner IQ scores
nrow(IQ_Over_100)/nrow(PrisonerIQ)
```

```
## [1] 0.4866147
```

The result should be close to .50 indicating that half of the population will have an IQ higher than the mean. This illustrates a useful feature of the normal distribution: the percentage of cases between its mean and points at a measured distance are fixed. This is referred to as the standard deviation (SD) unit, and the **z-score** is used to represent it. Z-scores range from -4 standard deviations below the mean and +4 standard deviations above the mean.

To create z-scores, we use the function `scale ()`. For the next example, we take the IQ of five prisoners and change the IQ of the first prisoner from 102 to 115 so that it is easy to show that this prisoner's z-score is 1. The reason is the prisoner's IQ score of 115 is one standard deviation above the population mean. (Remember: = 100; SD = 15):

```
# View the first 5 prisoner IQs
PrisonerIQ[1:5,]
```

```
##   prisoner_id  IQ
## 1           1  99
## 2           2  97
## 3           3 111
## 4           4  92
## 5           5 126
```

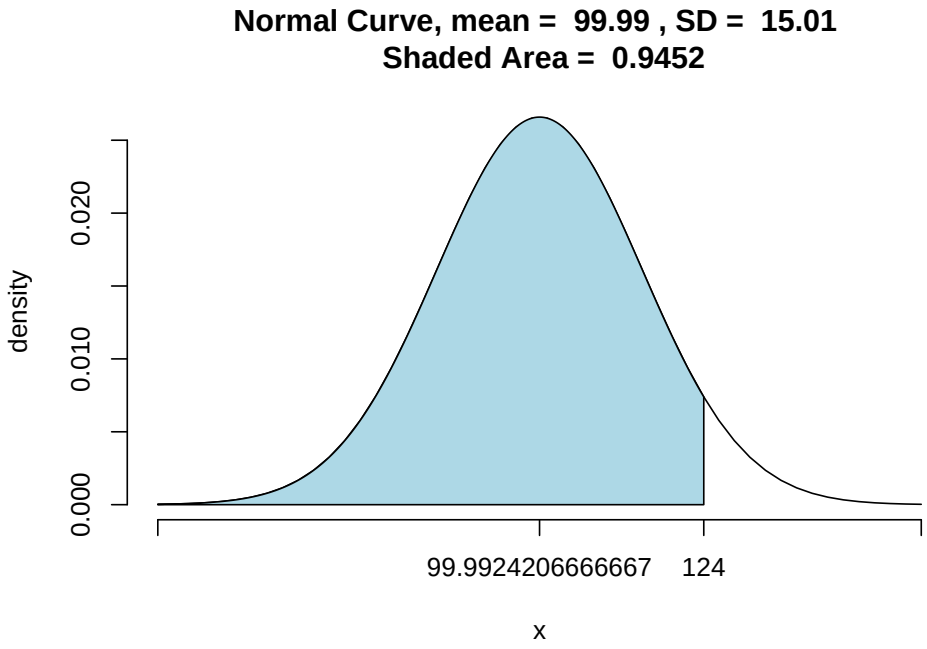
```
# Change the IQ of prisoner #1
PrisonerIQ$IQ[1] <- 115

# Create a variable storing z-scores of IQs
PrisonerIQ$z_scoreIQ <- scale(PrisonerIQ$IQ)

# Check to make sure prisoner #1 has a z-score around 1
PrisonerIQ[1,]
```

```
##   prisoner_id  IQ z_scoreIQ
## 1           1 115  1.000097
```

Where the z-score becomes practical is illustrated in the following example: say if a probation officer is writing a report for a prisoner who is about to be up for parole. The prisoner has an IQ of 124. The officer wants to give a good idea of

how this score compares to all other prisoners. An apt way of doing this is to state the proportion of prisoners who have lower IQs. This can be done using the `pnormGC()` function from the `tigerstats` package:

```
# The mean should be 100 but, in reality, it is not precise, so we calculate it and pu
m<-mean(PrisonerIQ$IQ)

# Same with sd; it should be 15 but we calculate it to get a precise estimate and put
sd<-sd(PrisonerIQ$IQ)

# Enter the prisoner's IQ score and specify 'below' following 'region' because we are
# interested in IQ scores below 124
pnormGC(124, region="below", mean=m, sd=sd,graph=TRUE)
```



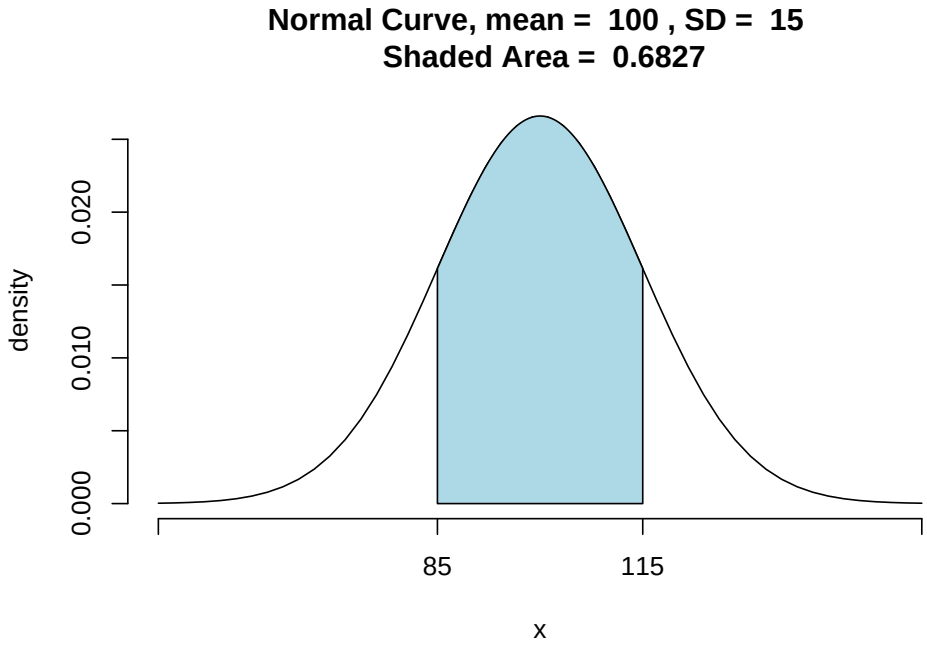

```
## [1] 0.9451843
```

The output shows 0.9453, meaning that the prisoner has a higher IQ than over 94% of the prison population.

Recall from the previous week, the 68-95-99.7 rule. This is helpful to keep in mind with z-scores, as a z-score indicates how far away a score is from the mean based on the standard normal distribution. The rule posits that 68% of cases in the distribution fall within one standard deviation above and below the mean; 95% within two SD; and 99.7% within 3 SD. We demonstrate this rule by using

the `pnormGC ()` function to get the proportion of prisoners that have an IQ between 85 to 115, which is one SD above and below the mean:

```
# Specify 'between' for 'region' because we are interested in proportion of prisoners between the
pnormGC(bound=c(85, 115),region="between", mean=100,sd=15,graph=TRUE)
```




```
## [1] 0.6826895
```

And, yes, it shows that 68% of the IQ scores do fall within 1 SD of the mean.

6.2.3.1 Single sample z-tests for means

Returning to the parole board example, say if the officer wanted to know, with 99% confidence, if the average IQ at this specific prison is significantly different from those of all prisons in the UK. The officer conducts an IQ assessment of all 233 prisoners at their prison and finds average IQ is 103 (SD =18). As the population parameter is known on prisoner IQ, a **single sample z-test** is appropriate. This test examines whether a sample is drawn from a specific population with a known or hypothesized mean. Here are the officer's hypotheses:

H_0: The mean IQ of the population from which our sample of prisoners was drawn is the same as the mean IQ of the UK prison population (mean = 100).

H_A: The mean IQ of the population from which our sample of prisoners was drawn is not the same as the mean IQ of the UK prison population (mean 100).

We create our own function called ' z_test ' so that other prisons can easily compare their IQ scores to that of all prisoners. In addition, we use `cat()`, which combines our string text to label our z-score in the output and the computed z-score together.

```
# Specifying inputs in the order that the user needs to enter them : mean, standard de
z_test<-function(xbar, sd, n, mu) {
  z <- (xbar-mu) / (sd / sqrt(n)) # Equation for one-sample z-test
  return(cat('z =', z)) # Report z-score to the user

} # End function

# Test it with our example by supplying estimates
z_test(103, 18, 233, 100)
```

```
## z = 2.544056
```

Including the p-value would be helpful so we edit our code and use the function `pnorm ()` to compute the probability value.

```
# Same as above
z_test<-function(xbar, sd, n, mu) {
z <- (xbar-mu) / (sd / sqrt(n))

# This is the new bit: we multiply 'pnorm' by 2 to indicate that our hypothesis is non
p<-2 * pnorm(-abs(z))

return( cat('z =', z,
'\np-value =', p)) # Added code to return p-value
}

# Test it with our example again
z_test(103, 18, 233, 100)
```

```
## z = 2.544056
## p-value = 0.01095734
```

Remember that the officer wanted to be 99% confident, and this means that the significance level would be set at $= 0.01$ and not $= 0.05$ (this is if we are 95% confident). As so, our p-value is greater than the alpha level so we fail to reject the null hypothesis.

6.2.3.2 Single Sample z-tests for Proportions

Our next example is to do with evaluating a new prison education programme. The foundation supporting the programme would like to achieve a success rate of 75% among 100,000 prisoners participating in he programme. Success is defined as completion of the six-month course.

After the programme ran, there is conflicting information about its success: managers of the programme claim they achieved higher than the 75% success rate, while a journalist investigating the programme claimed it was below 75%. You want to get to the bottom of this, so you collect information from 150 of the prisoners who enrolled on the programme using independent random sampling.

Your data shows that 85% of the participants successfully completed the programme. What to make of your result? Let's set up the hypotheses where we want a non-directional alternative hypothesis:

H_0: The success rate of the program is 0.75% (P = 0.75).

H_A: The success rate of the program is not 0.75% (P 0.75)

To test this, we use a **single-sample z-test for proportions** because we are concerned with comparing the percentages or proportions between our sample and the known population. We create another new function:

```
# Specifying inputs: p is proportion of success, P is proportion of success in population, n is s
prop_z_test<-function(p, P, n) {

Numerator<-(p - P)

PQ<- P * (1-P)

# Standard error
Denominator<-sqrt(PQ / n)
z<- Numerator / Denominator

# Return the z-value and p-value to the user
p<-2 * pnorm(-abs(z))

return( cat('z =', z,
'\np-value =', p))
}

# Let's test it using the values from our problem
prop_z_test(0.85, 0.75, 150)
```

```
## z = 2.828427
## p-value = 0.004677735
```

The z-score is 2.828427 and the p-value is statistically significant. We reject the null hypothesis and conclude that the success rate is not 75%.

6.2.3.3 Single-sample t-tests for Means

When the population parameter is unknown and we want to compare our sample to it, we use the t-distribution. From the previous example, let us say that average test scores were also collected for those prisoners who completed the six-month education course.

The foundation defined success as 65 for the test. Again, managers claimed the average scores were higher than this, whereas the journalist claimed the average was below 65. You have collected test score information from 50 prisoners and find that the mean is 60 and SD is 15. What conclusions can be made about the *larger population of prisoners* at the 95% confidence level?

Hypotheses

H_0: The mean test score for prisoners who have completed the program is 65 (= 65).

H_A: The mean test score for prisoners who have completed the program is not 65 (65).

We conduct a **single-sample t-test for means**, which is similar to the previous z-tests except it is for an unknown population, which in this case, is the overall population of prisoners and not just the ones who had completed the six-month education programme. We modify the code from the `z_test ()` function:

```
# Specifying inputs: xbar is sample mean, sd is sample standard deviation, n is sample
single_t_test<-function(xbar, sd, n, mu) {

# Equation for one-sample z-test
t <- (xbar-mu) / (sd / sqrt(n - 1))

# Report t-score and p-value to the user
p<-2 * pnorm(-abs(t))

return( cat('t =', t,
'\np-value =', p))
}

# Test it with our example
single_t_test(60, 15, 51, 65)
```

```
## t = -2.357023
## p-value = 0.01842213
```

The t-value of -2.357023 is statistically significant, so we have sufficient support to reject the null hypothesis. We conclude that the mean test score for prisoners who completed the programme is not 65.

6.3 SUMMARY

Today, we learned that to make predictions about the population from our sample, we must create **hypotheses**. When we test our hypothesis, we aspire to reject the **null hypothesis**, which tells us no differences exist. To ensure we reject the null accurately, however, we must be wary of **type 1 error** so we consider this error in tests of **statistical significance** and in evaluating our **p-values**. These hypothesis tests we learned today in `R` used the **binomial distribution** as well as the normal distribution, and required us to set our hypotheses at the outset as either **directional** or **non-directional**. Hypothesis tests that used the normal distribution were for **single samples** and statistical significance was determined by **z scores** and **t values**.

Homework time…

Chapter 7

Relationships with Categorical Variables

Independent and Dependent Variables, T-test, Chi-square

Learning Outcomes:

- Learn to arrange variables as independent and dependent variables
- Learn to test for statistical significance in relationships with categorical variables
- Understand how to interpret outputs from t-tests and the chi-square test

Today's Learning Tools:

Data:

- National Youth Survey (NYS)
- British Crime Survey (BCS)

Packages:

- `dplyr`
- `forcats`
- `gmodels`
- `haven`
- `here`
- `labelled`
- `skimr`
- `tidyverse`

Functions introduced (and packages to which they belong)

- `chisq.test()` : Produces the chi-square test (`base R`)
- `CrossTable()` : Produces contingency tables (`gmodels`)
- `fct_explicit_na()` : Provides missing values an explicit factor level (`forcats`)
- `fisher.test()` : Produces Fisher's exact test (`base R`)
- `merge()` : Merge datasets by common row or column names (`base R`)
- `n()` : Count observations within `summarize()`, `mutate()`, `filter()` (`dplyr`)
- `read_dta()` : Imports a .dta Stata file (`haven`)
- `t.test()`: Performs one and two sample t-tests on vectors of data (`base R`)
- `var.test()` : Performs an F-test to compare the variances of two samples from normal populations (`base R`)
- `with()` : Evaluates an expression, often to specify data you want to use (`base R`)

7.1 Associating with Categorical Variables

We are familiar with categorical variables, which take the forms of nominal and ordinal level characteristics. In `R`, these variables are encoded as a factor class, and for shorthand, are referred to as **factors**.

In inferential statistics, sometimes we want to make predictions about relationships with factors. For example, we want to understand the relationship between sex and alcohol consumption. Perhaps we think males will have higher alcohol consumption than females. The variable, sex, in this example, is a binary factor with the two categories, male and female, whereas alcohol consumption is a numeric variable.

Today, we learn how to conduct more inferential statistical analyses, but this time with categorical variables.

Before we start, we do the following:

1. Open up our existing `R` project
2. Install and load the required packages listed above
3. Open the National Youth Survey (NYS) datasets (*nys_1_ID.dta* and *nys_2_ID.dta*) using the function `read.dta ()`, specifying the working directory with `here()`

4. Name the data frames `nys_1` and `nys_2` respectively
5. Get to know the datasets with the codes `View(nys_1)` and `View(nys_2)`
6. There is some code on scientific notation that we will ignore, so run `options(scipen=999)` to turn it off.

```
View(nys_1)
View(nys_2)

options(scipen=999)
```

7.2 Today's 3

We further our understanding of inferential statistics by learning more about variables and a couple of new statistical analyses that examine a relationship with factor variables. Our three topics today are: **independent and dependent variables**, the **t-test**, and the **chi-square**.

7.2.1 Independent and Dependent Variables

In learning to set hypotheses last week, we primarily wanted to know whether there was a relationship between certain variables. Inadvertently, we also were arranging our variables in a way that indicated one was the explanation and the other was the outcome.

Using our previous example on sex and alcohol consumption, we can arrange them so that sex is the **independent variable** (X), which means that it is assumed to have an impact on the amount of alcohol consumption, the **dependent variable** (Y), which is considered to be influenced by X. We believe that sex (X) has an influence on level of alcohol consumption (Y), and hypothesise, based on previous research, that males will have higher levels of alcohol consumption than females. We then conduct an appropriate statistical test to find out.

Although independent and dependent variables are not terms usually used for today's analyses, we will use them to indicate which variable plays the role of explanation and which plays the part of outcome. In addition, it is good to get into the mindset of arranging your variables as such. This helps to clarify how your variables will relate to each other.

An important point to remember is that although the independent variable (IV) is considered to impact on the dependent variable (DV), it *does not mean the IV causes the DV*. We can only arrange these variables in the direction we think is suitable and make statements about whether they are related to each other. It is in experimental research designs that we can speak of causality.

7.2.2 The T-Test

7.2.2.1 Independent Sample T-test

We return to that previous example of sex and alcohol consumption. We would like to know whether there is a difference between the mean alcohol consumption level of males and females. In this example, we use the **independent sample t-test** or *unpaired t-test*. This test requires that the IV be a binary factor while the DV be either a ratio or interval variable and be normally distributed – unless **N** (the number of the sample) is large. Both variables meet the assumptions. We are interested in knowing whether there is a significant difference between males and females in the mean number of times they reported being drunk in the past year. Our null and alternative hypotheses are as follows:

H_0: There is no significant difference in frequency of getting drunk between males and females in the past year.

H_A: There is a significant difference in frequency of getting drunk between males and females in the past year.

We use the data frame, `nys_1`, to address our research question: Is there a difference in the amount of drunkenness between males and females? Before conducting our t-test, we need to check our variables of interest to see if they need any recoding. We use the function `summarize ()` to examine the sex variable (`sex`) and use the function `n ()` to obtain frequencies:

```
# Examining distribution of sex variable
nys_1 %>% group_by(sex) %>% summarize(n = n())
```

```
## # A tibble: 2 x 2
##          sex     n
## *  <dbl+lbl> <int>
## 1 1 [Male]     918
## 2 2 [Female]   807
```

According to the output, there are 918 males and 807 females. In addition, the codes for each value are shown: males are coded '1' whereas females are coded

'2'. We may want to recode these values to make a dummy variable (binary but codes are 0 and 1) so that females are '1' and males are '0':

```
# First, recode 'sex' and store it in a new variable called 'female'
nys_1$female <- recode(nys_1$sex, '2' = 1, '1' = 0)

# Next, label the values so that female is '1' and male is '0'
nys_1 <- nys_1 %>% add_value_labels(female = c("Female"=1, "Male"=0))

# Check they were added correctly by getting frequencies
count(nys_1, female)
```

```
## # A tibble: 2 x 2
##       female     n
## *  <dbl+lbl> <int>
## 1 0 [Male]     918
## 2 1 [Female]   807
```

Now we check out our dependent variable `drunk` using the `skim()` function from the `skimr` package familiar from lesson 4:

```
# Use skim() function from skimr package to examine drunk variable
skim(nys_1, drunk)
```

skim_type	skim_variable	n_missing	complete_rate	numeric.mean	numeric.sd	numeric.p0	numeric.p
numeric	drunk	6	0.9965217	1.242001	10.48457	0	

```
# Checking out DV by variable 'female'
nys_1 %>% group_by(female) %>% skim(drunk)
```

skim_type	skim_variable	female	n_missing	complete_rate	numeric.mean	numeric.sd	numeric.p0	n
numeric	drunk	0	3	0.9967320	1.614208	11.745701	0	
numeric	drunk	1	3	0.9962825	0.818408	8.821284	0	

From the output, it seems that most youth did not report getting drunk in the past year; this is evidenced by the mean. The mean, however, shows that males have a slightly higher level of being drunk than females (1.61 as opposed to 0.82). Is this a statistically significant difference?

Our observation of a difference between means prompts us to test whether this difference is an actual difference, or if it is attributed to chance.

Before we conduct the test to tell us this, we conduct the **test for equality of variance**. This is an F-test that evaluates the null hypothesis that the variances of the two groups are equal. When we want to compare variances, we conduct this test as the variances may affect how we specify our t-test.

```
# DV comes first in code, then the IV
var.test(nys_1$drunk~nys_1$female)
```

```
##
##  F test to compare two variances
##
## data:  nys_1$drunk by nys_1$female
## F = 1.7729, num df = 914, denom df = 803, p-value < 2.2e-16
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  1.549899 2.026885
## sample estimates:
## ratio of variances
##           1.772941
```

The information to focus on in the output are the alternative hypothesis, the F-statistic, and associated p-value. The alternative hypothesis states that the variances are not equal to 1, meaning they are not equal to each other. The p-value is very small, so we reject the null hypothesis that the variances are equal to each other. Conducting the F-test is an important step before the t-test because now we must set `var.equal` to `FALSE` in the following independent sample t-test.

```
# Run the t-test with var.equal option set to false
t.test(nys_1$drunk~nys_1$female, var.equal = FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  nys_1$drunk by nys_1$female
## t = 1.5994, df = 1677.3, p-value = 0.1099
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.1800974  1.7716968
## sample estimates:
## mean in group 0 mean in group 1
##        1.614208        0.818408
```

```
# You can save your results in an object, too, to refer back to later:
t_test_results <- t.test(nys_1$drunk~nys_1$female, var.equal = FALSE)

# Print independent t-test results
t_test_results
```

```
##
##  Welch Two Sample t-test
##
## data:  nys_1$drunk by nys_1$female
## t = 1.5994, df = 1677.3, p-value = 0.1099
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.1800974  1.7716968
## sample estimates:
## mean in group 0 mean in group 1
##        1.614208        0.818408
```

From the output, focus on the means, alternative hypothesis, the t-statistics, the p-value, and the 95% confidence interval (CI).

First, the means show that males got drunk an average of 1.6 times the year before, whereas females got drunk an average of 0.8 times, similar to the output of our descriptive statistics. Second, the alternative hypothesis (a true difference in means) is not equal to 0, which is what we expect. Third, the t-statistic reports 1.5994 with an associated p-value of 0.1099, which is greater than $=$ 0.05, so we fail to reject the null hypothesis. Fourth, the 95% CI tells us that, 95% of the time, the true t-value in the population will fall between -0.1801 and 1.772. We conclude that the true difference in means is not significantly different from 0 – there is *no significant difference in frequency of getting drunk between males and females in the past year.*

7.2.2.2 Dependent Sample T-test

When you are interested in comparing means of two groups that are related to each other, the **dependent sample t-test** or *paired sample t-test* is appropriate. What this means is that the responses are paired together because there is a prior connection between them. For example, I have two groups where the first group comprises test scores before an intervention and the second group comprises test scores after that intervention of the same people from the first group – a dependent sample t-test is called for. Another example: the first group are of brothers and the second group comprises sisters to those in the first group. This, too, would call for a dependent sample t-test.

We return to our NYS sample, which is drawn from a longitudinal study whereby the same people are surveyed over multiple time periods. In this example, we compare the behaviours of youth from Wave 1 to Wave 2 to address the research question: *'Do youth report a significantly different number of instances where they steal something over $50 in Wave 2 than in Wave 1?'* Our non-directional null and alternative hypotheses are as follows:

H_0: There is no significant difference in the number of times a youth reported stealing something worth more than $50 between Wave 1 and Wave 2.

H_A: There is a significant difference in the number of times a youth reported stealing something worth more than $50 between Wave 1 and Wave 2.

As responses from both Waves 1 and 2 are required, cases without this pair of responses will be dropped automatically when our analysis is conducted. This t-test also requires the DV to be stored in two separate variables and the level of measurement to be ratio or interval. Let us get to know our data:

```
# View NYS wave 2 data
View(nys_2)
```

The variable `CASEID` is the case identification numbers and these are found across all waves. We want to examine Waves 1 and 2, so will need to merge the waves together using the variable `CASEID`:

```
# Merging waves 1 and 2 of data together by CASEID
# Putting this merged data into the data frame object called 'nys_merged'
nys_merged <-merge(nys_1, nys_2, by="CASEID")
```

Now, for the dependent sample t-test whereby we use the variables `thftg50` and `thftg50_w2`:

```
# Let us run the dependent samples t-test
# Specify TRUE for the paired option
t.test(nys_merged$thftg50_w2, nys_merged$thftg50, paired= TRUE)
```

```
##
##  Paired t-test
##
## data:  nys_merged$thftg50_w2 and nys_merged$thftg50
## t = 1.7707, df = 1648, p-value = 0.07679
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.004178788  0.081801589
## sample estimates:
## mean of the differences
##               0.0388114
```

The results indicate that we fail to reject the null hypothesis as the p-value is above $= 0.05$. We conclude that there is no significant difference in thefts of items totaling more than $50 between the paired Waves 1 and 2 responses.

7.2.3 Chi-square

We now open the 2007–2008 wave of the British Crime Survey dataset using the `read.dta ()` function. We name the data frame `BCS0708`. Use the `View()` and `dim()` functions to get to know your data.

The **chi-square statistic** is a test of statistical significance for two categorical variables. It tells us how much the observed distribution differs from the one expected under the null hypothesis. To begin to even understand this definition, we start with cross tabulations or **contingency tables**. These appear as a table that shows the crossed frequency distributions of more than one variable simultaneously and are particularly helpful in exploring relationships between categorical variables that do not have too many categories. Specifically, cross-tabulations have the same meaning except they are applied only to relationships between two categorical variables.

We produce a cross tabulation of victimisation (`bcsvictim`) and whether rubbish is a problem in the area where the respondent lives (`rubbcomm`). According to *Broken Windows Theory*, proposed by James Q. Wilson and George Kelling, we should see a relationship between these two variables.

First, we check out our variables:

```
# Seeing what class these variables are
class(BCS0708$bcsvictim)
```

```
## [1] "haven_labelled" "vctrs_vctr"     "double"
```

```
class(BCS0708$rubbcomm)
```

```
## [1] "haven_labelled" "vctrs_vctr"     "double"
```

```
# 0-Not victim of crime; 1-Victim of crime
attributes(BCS0708$bcsvictim)
```

```
## $label
## [1] "experience of any crime in the previous 12 months"
##
## $format.stata
## [1] "%8.0g"
##
## $class
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $labels
## not a victim of crime       victim of crime
##                     0                     1
```

```
# 5-point Likert-scale ranging from 1 (very common) to 4 (not common)
attributes(BCS0708$rubbcomm)
```

```
## $label
## [1] "in the immediate area how common is litter\\rubbish"
##
## $format.stata
## [1] "%8.0g"
##
## $class
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $labels
##       very common     fairly common   not very common not at all common
##                 1                 2                 3                 4
##         not coded
##                 5
```

```
table(BCS0708$bcsvictim)
```

```
##
##    0    1
## 9318 2358
```

```
# If you want the frequency distribution with labels, use 'as_factor ()'
table(as_factor(BCS0708$bcsvictim))
```

```
##
## not a victim of crime       victim of crime
##                  9318                  2358
```

```
table(as_factor(BCS0708$rubbcomm))
```

```
##
##       very common     fairly common   not very common not at all common
##               204              1244              4154              5463
##         not coded
##                 0
```

```
# Checking for any missing values using 'sum ()' to count number of NA data
sum(is.na(BCS0708$bcsvictim))
```

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

```
# Value of 5 indicates missing
sum(is.na(BCS0708$rubbcomm))
```

```
## [1] 611
```

For the DV, we are missing 611 cases. For this course unit, keep in mind that all the statistical tests you will learn rely on **full cases analysis** whereby the tests exclude cases that have missing values. There are appropriate ways in dealing with missing data but this is beyond the scope of this class.

There are a couple of ways of producing cross tabulations in R:

```
# Basic R
table(as_factor(BCS0708$bcsvictim),
      as_factor(BCS0708$rubbcomm))
```

```
##
##                         very common fairly common not very common
##   not a victim of crime         141           876            3173
##   victim of crime                63           368             981
##
##                         not at all common not coded
##   not a victim of crime              4614         0
##   victim of crime                     849         0
```

```
# dplyr
results <- BCS0708 %>% # 'fct_explicit_na()' function from the forcats package to get an explicit
  group_by(fct_explicit_na(as_factor(rubbcomm))) %>%
  # We use the function 'mean()' as the variable is binary and because it is coded as 0 and 1.
  summarize( count = n(), outcome_1 = mean(bcsvictim))

# Auto-print the results stored in the newly created object
results
```

```
## # A tibble: 5 x 3
##   `fct_explicit_na(as_factor(rubbcomm))` count outcome_1
## * <fct>                                  <int>     <dbl>
## 1 very common                              204     0.309
## 2 fairly common                           1244     0.296
## 3 not very common                         4154     0.236
## 4 not at all common                       5463     0.155
## 5 (Missing)                                611     0.159
```

The `dplyr` coding seems more complicated than that of `basic R` but its output is clearer to read than the one produced by `R`, and it is more detailed. The proportion of victimised individuals are within each of the levels of the categorical, ordered measure of rubbish in the area. Victimisation appears higher (31%) in the areas where rubbish in the streets is a very common problem.

To further explore contingency tables, the best package for this is `gmodels`. It allows you to produce cross tabulations in a format similar to the one used by commercial statistical packages SPSS and SAS. We use the function `CrossTable()`, then the `with()` function to identify the data frame at the outset instead of having to include it with each variable.

```
# Define the rows in your table (rubbcomm) and then the variable that will define the
with(BCS0708, CrossTable(as_factor(rubbcomm),
                         as_factor(bcsvictim),
prop.chisq = FALSE,
format = c("SPSS")))
```

```
##
##    Cell Contents
## |-------------------------|
## |                   Count |
## |             Row Percent |
## |          Column Percent |
## |           Total Percent |
## |-------------------------|
##
## Total Observations in Table:  11065
##
##                      | as_factor(bcsvictim)
## as_factor(rubbcomm)  | not a victim of crime  |       victim of crime  |
## ---------------------|------------------------|------------------------|--------------
##          very common |                    141  |                     63  |
##                      |                69.118% |                30.882% |
##                      |                 1.602% |                 2.786% |
##                      |                 1.274% |                 0.569% |
## ---------------------|------------------------|------------------------|--------------
##       fairly common  |                    876  |                    368  |
##                      |                70.418% |                29.582% |
##                      |                 9.950% |                16.276% |
##                      |                 7.917% |                 3.326% |
## ---------------------|------------------------|------------------------|--------------
##     not very common  |                   3173  |                    981  |
##                      |                76.384% |                23.616% |
##                      |                36.040% |                43.388% |
##                      |                28.676% |                 8.866% |
```

```
## --------------------|-----------------------|-----------------------|-----------------------|
##   not at all common |                  4614 |                   849 |                  5463 |
##                     |               84.459% |               15.541% |               49.372% |
##                     |               52.408% |               37.550% |                       |
##                     |               41.699% |                7.673% |                       |
## --------------------|-----------------------|-----------------------|-----------------------|
##        Column Total |                  8804 |                  2261 |                 11065 |
##                     |               79.566% |               20.434% |                       |
## --------------------|-----------------------|-----------------------|-----------------------|
##
##
```

Cells for the central two columns are the total number of cases in each category, comprising the *row percentages*, the *column percentages*, and the *total percentages*.

The contingency table shows that 63 people in the category 'rubbish is very common' were victims of a crime; this represents 30.88% of all the people in the 'rubbish is very common' category (your row percent), 2.79% of all the people in the 'victim of a crime' category (your column percent), and 0.57% of all the people in the sample.

There is quite a lot of proportions that the contingency table will churn out, but you are only interested in the proportions or percentages that allow you to make meaningful comparisons. Again, this is where talk about independent and dependent variables come in. Although talking about variables like this is not common in these analyses as they would be, say, in regression, an analysis we learn later, it is good to get into the mindset of arranging our variables in this way.

If you believe in broken windows theory, you will think of victimisation as the outcome we want to explain (Y) and rubbish in the area as a possible explanation for the variation in victimisation (X). If so, you will need to request percentages that allow you to make comparisons across `rubbcomm` for the outcome, `bcsvictim`.

This is a very **important** point: often, cross tabulations are interpreted the wrong way because percentages were specified incorrectly. Two rules to help ensure you interpret cross-tabs (short for cross tabulations) correctly:

1. If your dependent variable is defining the rows, then you ask for the *column percentages*
2. If your dependent variable is defining the columns, then you ask for the *row percentages*

Also, *you make the comparisons across the right percentages in the direction where they do not add up to a hundred percent.* For example, 30.88% of people

who live in areas where rubbish is very common have been victimised, whereas only 15.54% of people who live in areas where rubbish is not at all common have been victimised in the previous year. To make it easier, we can ask R to only give us the percentages in which we are interested:

```
# prop.c is column and prop.t is total
with(BCS0708, CrossTable(as_factor(rubbcomm),
                         as_factor(bcsvictim),
                         prop.chisq = FALSE, prop.c = FALSE, prop.t = FALSE, format =
                           c("SPSS")))
```

```
##
##    Cell Contents
## |-------------------------|
## |                   Count |
## |             Row Percent |
## |-------------------------|
##
## Total Observations in Table:  11065
##
##                      | as_factor(bcsvictim)
## as_factor(rubbcomm)  | not a victim of crime  |       victim of crime  |
## ---------------------|------------------------|------------------------|--------------
##         very common  |                    141 |                     63 |
##                      |                69.118% |                30.882% |
## ---------------------|------------------------|------------------------|--------------
##       fairly common  |                    876 |                    368 |
##                      |                70.418% |                29.582% |
## ---------------------|------------------------|------------------------|--------------
##     not very common  |                   3173 |                    981 |
##                      |                76.384% |                23.616% |
## ---------------------|------------------------|------------------------|--------------
##   not at all common  |                   4614 |                    849 |
##                      |                84.459% |                15.541% |
## ---------------------|------------------------|------------------------|--------------
##        Column Total  |                   8804 |                   2261 |
## ---------------------|------------------------|------------------------|--------------
##
##
```

Now with this output, we only see the row percentages. **Marginal frequencies** appear as the right column and bottom row. *Row marginals* show the total number of cases in each row. For example, 204 people perceive rubbish as very common in the area where they are living and 1,244 perceive rubbish as fairly common in their area. *Column marginals* show the total number of cases in each column: 8,804 non-victims and 2,261 victims.

In the central cells, these are the total number for each combination of categories. For example, for row percentages, 63 people who perceive rubbish as very common in their area and who are a victim of a crime represent 30.88% of all people who have reported that rubbish is common (63 out of 204). For column percentages (shown previously), 63 people who live in areas where rubbish is very common and are victims represent 2.79% of all people who were victims of crime (63 out of 2,261).

We now have an understanding of cross-tabs so that we can interpret the **chi-square statistic**. Our null and alternative hypotheses are as follows:

H_0: Victimisation and how common rubbish is in the area are not related to each other. (They are considered independent of each other.)

H_A: Victimisation and how common rubbish is in the area are significantly related to each other. (They are considered dependent on each other.)

The test does the following: (1) compares these expected frequencies with the ones we actually observe in each of the cells, (2) then averages the differences across the cells, and (3) produces a standardised value, ^{2} (the chi-square statistic).

We then look at a chi-square distribution to see how probable or improbable this value is. But, in practice, the p-value helps us ascertain this more quickly.

Expected frequencies are the number of cases you would expect to see in each cell within a contingency table if there was no relationship between the two variables. **Observed frequencies** are the cases that we actually see in our sample. We use the `CrossTable ()` function again:

```
with(BCS0708, CrossTable(as_factor(rubbcomm),
                         as_factor(bcsvictim),
                         expected = TRUE, prop.c = FALSE, prop.t = FALSE, format =
                           c("SPSS") ))
```

```
##
##    Cell Contents
## |-------------------------|
## |                   Count |
## |         Expected Values |
## | Chi-square contribution |
## |             Row Percent |
## |-------------------------|
##
## Total Observations in Table:  11065
##
##                      | as_factor(bcsvictim)
## as_factor(rubbcomm) | not a victim of crime  |       victim of crime  |             Row Total
```

```
## --------------------|-----------------------|-----------------------|--------------
##          very common |                   141  |                    63  |
##                      |               162.315  |                41.685  |
##                      |                 2.799  |                10.899  |
##                      |                69.118% |                30.882% |
## --------------------|-----------------------|-----------------------|--------------
##       fairly common |                   876  |                   368  |
##                      |               989.804  |               254.196  |
##                      |                13.085  |                50.950  |
##                      |                70.418% |                29.582% |
## --------------------|-----------------------|-----------------------|--------------
##     not very common |                  3173  |                   981  |
##                      |              3305.180  |               848.820  |
##                      |                 5.286  |                20.583  |
##                      |                76.384% |                23.616% |
## --------------------|-----------------------|-----------------------|--------------
##   not at all common |                  4614  |                   849  |
##                      |              4346.701  |              1116.299  |
##                      |                16.437  |                64.005  |
##                      |                84.459% |                15.541% |
## --------------------|-----------------------|-----------------------|--------------
##        Column Total |                  8804  |                  2261  |
## --------------------|-----------------------|-----------------------|--------------
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  184.0443     d.f. =  3     p =  1.180409e-39
##
##
##
##        Minimum expected frequency: 41.68495
```

The output shows that, for example, 63 people lived in areas where rubbish was very common and experienced victimisation in the past year. Under the null hypothesis of no relationship, however, we should expect this value to be 41.69. Thus, more people are in this cell than we would expect under the null hypothesis.

The chi-square value is 184.04, with 3 degrees of freedom (df). The df is obtained by the number of rows minus one, then multiplied by the number of columns minus one: $(4-1)*(2-1)$. The probability associated with this particular value is nearly zero (1.180e-39). This value is considerably lower than the standard

alpha level of 0.05. We conclude that there is a significant relationship between victimisation and the presence of rubbish. We reject the null hypothesis.

If you do not want to use `CrossTable ()`, you can use `chisq.test ()` to give you the chi-square value:

```
chisq.test(BCS0708$rubbcomm, BCS0708$bcsvictim)
```

```
##
##  Pearson's Chi-squared test
##
## data:  BCS0708$rubbcomm and BCS0708$bcsvictim
## X-squared = 184.04, df = 3, p-value < 2.2e-16
```

The chi-square statistic only tells us whether there is a relationship or not between two variables; it says nothing about strength of relationship or exactly what differences between observed and expected frequencies are driving the results.

For the chi-square to work though, it needs to have a sufficient number of cases in each cell. Notice that `R` was telling us that the minimum expected frequency is 41.68. One rule of thumb is that all expected cell counts should be above 5. If we have a small number in the cells, one alternative is to use **Fisher's exact test**:

```
with(BCS0708, fisher.test(rubbcomm, bcsvictim))
```

```
# If you get an error message about increasing the size of your workspace, do so with this code:
fisher.test(BCS0708$rubbcomm, BCS0708$bcsvictim, workspace = 2e+07, hybrid = TRUE)
```

```
##
##  Fisher's Exact Test for Count Data hybrid using asym.chisq. iff
##  (exp=5, perc=80, Emin=1)
##
## data:  BCS0708$rubbcomm and BCS0708$bcsvictim
## p-value < 2.2e-16
## alternative hypothesis: two.sided
```

We did not need this test for our example, but we use it to illustrate how to use Fisher's Exact Test when counts in cells are low. The p-value from Fisher's exact test is still smaller than = 0.05, so we reach the same conclusion that the relationship observed can be generalised to the population.

Last, we had observed that there were differences between the observed and expected frequencies. This is called a **residual**. Some differences seem larger

than others. For example, there were about 21 more people that lived in areas where rubbish was very common and they experienced victimisation than what were expected under the null hypothesis. When you see large differences, it is unsurprising to also expect that the cell in question may be playing a particularly strong role in driving the relationship between rubbish and victimisation.

We are not sure, however, of what is considered a large residual. A statistic that helps address this is the **adjusted standardised residuals**, which behaves like a z-score. Residuals indicate the difference between the expected and the observed counts on a standardised scale. When the null hypothesis is true, there is only about a 5% chance that any particular standardised residual exceeds 2 in absolute value.

Whenever you see differences that are greater than 2, the difference between expected and observed frequencies in that particular cell is significant and is driving the results of your chi-square test. Values above $+3$ or below -3 are considered convincing evidence of a true effect in that cell:

```
# Include argument 'asresid= TRUE' to include adjusted standardised residuals
with(BCS0708, CrossTable(as_factor(rubbcomm), as_factor(bcsvictim), expected = TRUE,
                         prop.chisq = FALSE, prop.c = FALSE, prop.t = FALSE, asresid =
                           TRUE, format = c("SPSS") ))
```

```
##
##    Cell Contents
## |-------------------------|
## |                   Count |
## |         Expected Values |
## |             Row Percent |
## |           Adj Std Resid |
## |-------------------------|
##
## Total Observations in Table:  11065
##
##                      | as_factor(bcsvictim)
## as_factor(rubbcomm)  | not a victim of crime  |       victim of crime  |
## ---------------------|------------------------|------------------------|--------------
##         very common  |                    141  |                     63  |
##                      |               162.315  |                41.685  |
##                      |                69.118% |                30.882% |
##                      |                -3.736  |                 3.736  |
## ---------------------|------------------------|------------------------|--------------
##       fairly common  |                    876  |                    368  |
##                      |               989.804  |               254.196  |
##                      |                70.418% |                29.582% |
##                      |                -8.494  |                 8.494  |
```

```
## --------------------|-----------------------|-----------------------|-----------------------|
##     not very common |                  3173 |                   981 |                  4154 |
##                     |              3305.180 |               848.820 |                       |
##                     |                76.384% |                23.616% |               37.542% |
##                     |                -6.436 |                 6.436 |                       |
## --------------------|-----------------------|-----------------------|-----------------------|
##   not at all common |                  4614 |                   849 |                  5463 |
##                     |              4346.701 |              1116.299 |                       |
##                     |                84.459% |                15.541% |               49.372% |
##                     |                12.605 |               -12.605 |                       |
## --------------------|-----------------------|-----------------------|-----------------------|
##        Column Total |                  8804 |                  2261 |                 11065 |
## --------------------|-----------------------|-----------------------|-----------------------|
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  184.0443     d.f. =  3     p =  1.180409e-39
##
##
##
##        Minimum expected frequency: 41.68495
```

The column representing the outcome of interest (victimisation present), shows the adjusted standardised residual is lower than −12 for the 'not at all common' category. That is the largest residual for the DV. The expected count under the null hypothesis in this cell is much higher than the observed count.

7.3 SUMMARY

We learned a few inferential statistical analyses to examine relationships with categorical variables, known as **factors** in R. These variables, in today's analyses, were arranged as **independent variables** or **dependent variables**. When analysing a relationship between a categorical and a numeric variable, the t-test was used. We learned to conduct **independent sample** and **dependent sample** t-tests.

Before we performed the former t-test, we conducted the **test for equality of variance**. In the latter section, we learned to conduct a chi-square test, a

test of statistical significance between two categorical variables. This involved **contingency tables**, whereby we examined specifically **cross tabulations**.

The chi-square statistic contrasts **expected** and **observed** frequencies in the cross-tab. When counts in any one cell is lower than five, **Fisher's Exact Test** is used instead. We also check the **adjusted standardised residuals** to identify which contrast of frequencies are driving the observed results.

Homework time!

Chapter 8

Strength of Relationships

Between Categorical Nominal, Categorical Ordinal, & Numeric Variables

Learning Outcomes:

- Learn how to conduct analyses that identify the effect sizes of relationships
- Understand the output and make correct interpretations of it

Today's Learning Tools:

Data:

- Seattle Neighborhoods and Crime Survey
- Patrick Sharkey's data

Packages:

- `DescTools`
- `dplyr`
- `ggplot2`
- `GoodmanKruskal`
- `haven`
- `here`
- `tibble`

Functions introduced (and packages to which they belong)

- `add_row()` : Add rows to a data frame (`tibble`)
- `cor()` : Produces the correlation of two variables (`base R`)
- `cor.test()` : Obtains correlation coefficient (`base R`)
- `CramerV()` : Conducts the Cramer's V measure of association (`DescTools`)
- `GoodmanKruskalGamma()` : Conducts Goodman-Kruskal gamma measure of association (`DescTools`)
- `Phi()` : Conducts the phi measure of association (`DescTools`)
- `rm()` : Remove object from R environment (`base R`)
- `SomersDelta()` : Conducts the Somers' D measure of association (`DescTools`)

8.1 Measures of Association

So far in the course, we have learned ways of measuring whether there is a relationship between variables. If there is a relationship between variables, it means that the value of IV can be used to predict the value of the DV. Although we are able to test for statistical significance, we are unable to say anything about how *strong* these associations between variables are.

In crime and criminal justice research, not only are we interested in whether there is a relationship between variables, but we are also interested in the size of that relationship. Knowing the strength of the relationship is useful because it indicates the size of the difference. It addresses the question of 'To what extent is this relationship generalisable?' instead of merely 'Is there a relationship or not?'

The strength of relationship is known more as the **effect size**, and simply quantifies the magnitude of difference between two variables. Today's lesson is about the effect size for relationships between categorical variables, and then ones between numeric variables.

We begin by :

1. Opening our project
2. Loading the required packages
3. Opening the Seattle Neighborhoods and Crime Survey dataset using the function `read.dta ()` and naming the data frame object as `seattle_ df`
4. Get to know the data by using the functions `View()` and `dim()`, and to see if it has been loaded successfully

---

8.2 Today's 3

We will learn how to run effect sizes and how to interpret them. First, we learn how to do so among categorical, nominal variables; then, second, with categorical, ordinal variables; and, third, with numeric variables.

8.2.1 Between Categorical, Nominal Variables

Following last week's lesson on the chi-square analysis, the effect size for the relationship between two categorical variables can be conducted after obtaining the 2 statistic. A number of measures are available to test *how related* the two variables are to each other. We learn three of them: *phi*, *Cramer's V*, and *Goodman and Kruskal's lambda and tau*.

8.2.1.1 Phi

This measure builds directly off of the chi-square statistic, but it is for the strength of association between two *binary* variables. It is used for 2×2 tables to obtain a measure of association ranging from 0 to 1, whereby higher values indicate a stronger relationship. What this measure does is account for the sample size under observation, as the chi-square statistic is influenced by it. We obtain the **phi** coefficient by dividing the chi-square value by the sample size and taking the square root of that result, but we can do this in R. For example, we would like to know whether there is a relationship between sex and reporting victimisation to the police, and what is the strength of that relationship. We obtain the relevant variables, then conduct a chi-square analysis and obtain the phi coefficient:

```
# Create copies of the variables, rename them so it is easier to remember
# sex is variable 'QDEM3'
table(seattle_df$QDEM3)
```

```
##
##    1    2
## 1145 1075
```

```
# Use factor () function to create factor variable 'sex'
seattle_df$sex <- factor(seattle_df$QDEM3, levels = c(1, 2), labels = c("female" ,
                                                                        "male"))


# Reported victimisation to police is 'Q58E'
table(seattle_df$Q58E)
```

```
##
##   -1    0    1    8    9
## 1582  379  255    2    2
```

```
# Use factor () function to create factor variable 'reported_to_police'
seattle_df$reported_to_police <- factor(seattle_df$Q58E, levels = c(0, 1), labels =
                                        c("no" , "yes"))

# Chi-square analysis
# Place variable objects into new object 'chi'
chi<-table(seattle_df$sex, seattle_df$reported_to_police)
# Make sure the relationship is significant first
chisq.test(chi)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  chi
## X-squared = 5.8156, df = 1, p-value = 0.01588
```

```
# Chi-square is significant ( p= 0.01588)

# Calculate the phi coefficient
Phi(seattle_df$sex, seattle_df$reported_to_police)
```

```
## [1] 0.09903062
```

The phi value is about 0.1, and as the range is 0 to 1, it seems that this value is closer to 0 than it is to 1. We conclude that this is a weak association between a respondent's sex and whether they reported their victimisation to the police.

8.2.1.2 Cramer's V

For tables that are larger than 2x2, meaning these categorical variables are not binary, **Cramer's V** is appropriate (although using Cramer's V on a 2x2 table will produce the same value as that of the phi coefficient). Interpreting this value is similar to phi: a range from 0 to 1 with higher values indicating a stronger relationship. An advantage of Cramer's V is that it is not swayed by sample size, so if you had suspected that the chi-square statistic was the result of a large sample size, this value can help determine that:

```
# Cramer's V test for the same relationship detailed above to show that it is exactly like phi:
CramerV(seattle_df$sex, seattle_df$reported_to_police)
```

```
## [1] 0.09903062
```

```
# Now for non-binary variable, Q52, 'How often do you worry about being attacked in your neighbou

# Checking it out
table(seattle_df$Q52)
```

```
##
##    1    2    3    4    8    9
## 1584  354  167   98   12    5
```

```
# Make a copy of the Q52 variable
seattle_df$worry_attacked <- factor(seattle_df$Q52, levels = c(1, 2, 3, 4), labels =
                                      c("Less than once a month" , "once a month",
                                        "about o nce a week", "everyday"))

# Run a chi-square test to make sure there is a significant relationship
# Save the result in the object "chi2"
chi2 <- table(seattle_df$worry_attacked, seattle_df$sex)
chisq.test(chi2)
```

```
##
##  Pearson's Chi-squared test
##
## data:  chi2
## X-squared = 38.092, df = 3, p-value = 2.702e-08
```

```
# Conduct the Cramer's V test of association on the contingency table
# You can use the object 'chi2' for convenience
CramerV(chi2)
```

```
## [1] 0.1314953
```

Like the phi, the value shows a relatively small relationship between residents' sex and how often they worry about being attacked in their neighbourhood.

8.2.2 Between Categorical, Ordinal Variables

In some common situations in crime and criminal justice research, we work with survey data that have ordered response categories. For example, a likert scale of how strongly someone feels about abolishing the death penalty.

We learn two measures that deal with relationships between categorical, ordinal variables: Goodman-Kruskal's Gamma and Somers' D. Both use concordant and discordant pairs of observations to estimate the effect size. **Concordant pairs** are observations where the rankings are consistent for both variables, and **discordant pairs** are observations whose rankings are inconsistent for both variables. If a pair of observations has the same rank on the variables of interest, they are considered a **tied pair**.

8.2.2.1 Goodman-Kruskal Gamma

Gamma can take on values ranging from -1 to $+1$, where: 0 indicates no relationship at all; a negative value indicates a negative relationship; and a positive value a positive relationship. The closer the value is to either -1 or $+1$, the stronger the relationship is between variables.

For this example, we test the association between two measures commonly used to gauge community informal social control. The first variable, `Q20A`, is based on a survey question asking respondents how likely it is that their neighbours would do something about a group of neighbourhood children skipping school. The second variable, `Q20B`, asks respondents how likely it is that neighbours would do something if children were spray painting graffiti on a neighbourhood building.

```
# First, check and recode your variables if necessary
table(seattle_df$Q20A)
```

```
##
##   1   2   3   4   8   9
## 432 701 784 207  87   9
```

```
# Recode Q20A- check the codebook on the variable's ranking
seattle_df$intv.truancy <- factor(seattle_df$Q20A, levels = c(1, 2, 3, 4), labels =
                                    c("very likely" , "likely", "unlikely", "very
                                      unlikely"))

# Review everything was coded correctly
table(seattle_df$intv.truancy)
```

```
##
##                                          very likely
##                                                  432
##                                               likely
##                                                  701
##                                             unlikely
##                                                  784
## very\n                                      unlikely
##                                                  207
```

```
class(seattle_df$intv.truancy)
```

```
## [1] "factor"
```

```
# Now do the same for the graffiti variable
table(seattle_df$Q20B)
```

```
##
##    1    2    3    4    8    9
## 1198  761  177   43   35    6
```

```
seattle_df$intv.graff<- factor(seattle_df$Q20B, levels = c(1, 2, 3, 4), labels =
                                 c("very likely" , "likely", "unlikely", "very unlikely"))

table(seattle_df$intv.graff)
```

```
##
##   very likely        likely      unlikely very unlikely
##          1198           761           177            43
```

```
class(seattle_df$intv.graff)
```

```
## [1] "factor"
```

```
# Assess the relationship between these two variables
GoodmanKruskalGamma(seattle_df$intv.truancy, seattle_df$intv.graff, conf.level = .95)
```

```
##     gamma    lwr.ci    upr.ci
## 0.6315274 0.5898587 0.6731960
```

```
# We can also run it using a table saved as an R object
x <- table(seattle_df$intv.truancy, seattle_df$intv.graff)
gamma_neighbor_intervene <- GoodmanKruskalGamma(x, conf.level = .95)
```

We specified the `conf.level` option and get a 95% confidence interval around the gamma coefficient. The value we receive is 0.63, which, in addition to the confidence interval that does not overlap with 0, indicates a fairly strong statistically significant and positive association between the two variables.

8.2.2.2 Somers' D

Somer's D provides a value between −1 and +1, whereby values closer to −1 and +1 indicate better prediction ability. This commonly used measure for ordinal variables indicates how much improvement in the prediction of the dependent variable is attributed to information we know from the independent variable.

We use the `SomersDelta` function from the `DescTools` package to conduct Somers' D. We examine the relationship between the respondents' perception of their neighbour's willingness to exert informal social control and how likely the respondent would miss their neighbourhood if they moved away (`Q7`):

```
# Recode the variable Q7
seattle_df$miss_neigh <- factor(seattle_df$Q7, levels = c(1, 2, 3, 4), labels = c("very

# Check our newly coded variable
table(seattle_df$miss_neigh)
```

```
##
##   very likely        likely      unlikely very unlikely
##          1094           760           244           102
```

```
# To run Somers' D, first save the contingency table in the R object "z"
z <- table(seattle_df$intv.graff, seattle_df$miss_neigh)

# Now run the Somers' D measure
# 'direction' tells R which variable should be considered the IV
# It defaults to row, so if you excluded this option, the function would still provide
SomersDelta(z, direction = "row", conf.level = 0.95)
```

```
##    somers    lwr.ci    upr.ci
## 0.1867064 0.1502957 0.2231171
```

Somers' D is positive, but close to 0, meaning that perception of neighbours' willingness to intervene is a poor predictor of how much respondents would miss their neighbourhood if they moved. The confidence interval around the estimate does not overlap with 0 meaning that the association is statistically significant, even though the association is weak.

8.2.3 Between Numeric Variables

This section is about measuring the strength of relationships between two ratio/interval variables. Load Professor Patrick Sharkey's dataset (sharkey.csv), which is a study on the effect of nonprofit organisations on the levels of crime, using the `read_csv()` function. Alternatively, you can load the dataset from the Dataverse website.

Name the data frame `sharkey`, and if you dislike scientific notation, you can turn it off in `R` by using `options(scipen=999)`.

First, we examine our bivariate relationship between variables of interest through a scatterplot. The reason for this is we need to figure out if the variables have a **linear relationship**, and this determines what test we need to use. The extent to which two variables move together is called **covariation**: one variable can increase and so will the other (positive linear relationship); one variable can decrease while the other increases (negative linear relationship).

We focus on the year 2012, which is the most recent year in the dataset, and on only a few select variables. We will place them in a data frame named, `df`. To do this, we will use the `filter()` and `select()` functions from the `dplyr` package. Then, we will remove the dataset from `R` using the `rm()` function.

```
df <- filter(sharkey, year == "2012")
df <- select(df, place_name, state_name, black, lesshs, unemployed, fborn,
             incarceration, log_viol_r, largest50)

# Goodbye sharkey
rm(sharkey)

# View the number of cities located in each of the 44 states
table(df$state_name)
```

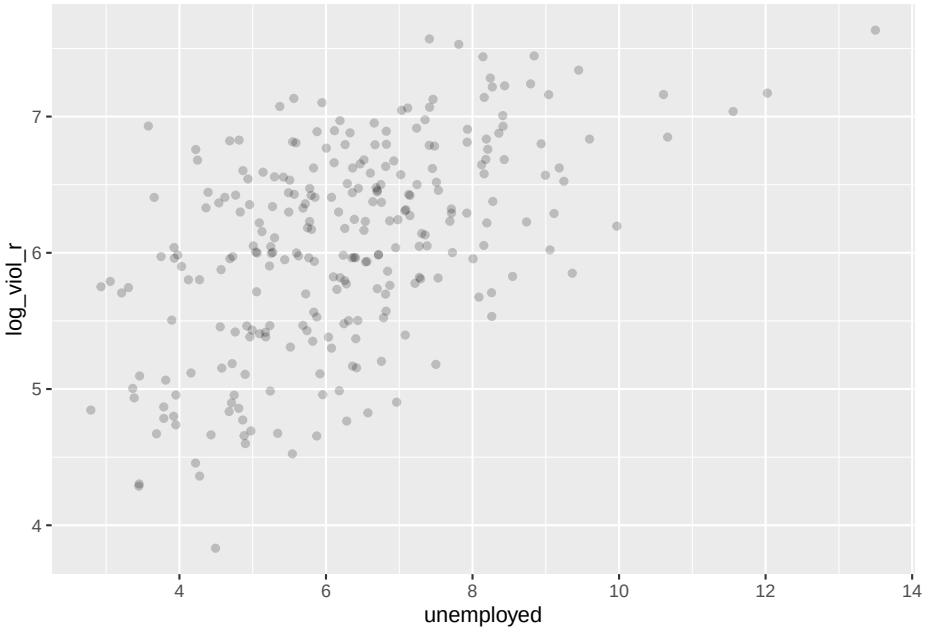
```
##
##              Alabama               Alaska              Arizona
##                    4                    1                    9
##             Arkansas           California             Colorado
##                    1                   65                   10
```

```
##          Connecticut District of Columbia              Florida
##                    5                    1                   18
##              Georgia                Idaho             Illinois
##                    2                    1                    8
##              Indiana                 Iowa               Kansas
##                    3                    3                    5
##            Louisiana             Maryland        Massachusetts
##                    4                    1                    3
##             Michigan            Minnesota           Mississippi
##                    6                    3                    1
##             Missouri              Montana             Nebraska
##                    5                    1                    2
##               Nevada        New Hampshire           New Jersey
##                    3                    1                    4
##           New Mexico             New York       North Carolina
##                    1                    5                    9
##         North Dakota                 Ohio             Oklahoma
##                    1                    5                    4
##              Oregon         Pennsylvania         Rhode Island
##                    4                    4                    1
##       South Carolina         South Dakota            Tennessee
##                    3                    1                    6
##                Texas                 Utah             Virginia
##                   30                    4                    7
##           Washington            Wisconsin
##                    6                    3
```

The variables we now have contain information on the demographic composition of those cities (percent black population, percent without high school degree, percent unemployed, percent foreign born) and criminal justice characteristics (incarceration rate and the rate of sworn full-time police officers). In addition, we have measures of the violence rate and a binary variable that tells us if the city is one of the 50 largest in the country (value '1' means one of the 50 largest).

As the first step in exploring relationships is to visualise them, we create a scatterplot between the log of the violence rate (`log_viol_r`) and unemployment (`unemployed`) using the `ggplot()` function in the `ggplot2` package:

```
ggplot(df, aes(x = unemployed, y = log_viol_r)) +
# Jitter adds a little random noise
# This makes points less likely to overlap one another in the plot
  geom_point(alpha=.2, position="jitter")
```



Is there a linear relationship between violence and unemployment? Does it look as if cities that have a high score on the x-axis (unemployment) also have a high score on the y-axis (violent crime)? It is hard to see but there is a trend: cities with more unemployment seem to have more violence. Notice, for example, how at high levels of unemployment, there are places with high levels of violence.

As there is a linear relationship, we conduct a **Pearson's correlation** test. This test tells you whether the two variables are significantly related to one another – whether they covary. A p-value is provided to determine this. Also provided is a Pearson's r value, which indicates the strength of the relationship between the two variables. The value ranges from -1 (a negative linear relationship) to 1 (a positive linear relationship). Values that are closer to 1 or -1 suggest a stronger relationship.

The test calculates this value, the Pearson's r, by, first, examining the extent to which each case deviates from the mean of each of the two variables, and then multiplies these deviations:

$$covariation\ of\ scores = \sum_{i=1}^{n}(x_{1i} - \bar{x_1})(x_{2i} - \bar{x_2})$$

Second, it standardises the covariation by taking the square root of the value obtained from the sums of squared deviations from the mean of both variables. This is because, sometimes, the variables may use different units of measurement from each other. For example, one variable measures in inches and the

other variable measures in decades. Thus, Pearson's r is the ratio between the covariation of scores and this standardisation of covariation:

$$Pearson's\ r = \frac{\sum_{i=1}^{n}(x_{1i} - \bar{x_1})(x_{2i} - \bar{x_2})}{\sqrt{[\sum_{i=1}^{n}(x_{1i} - \bar{x_1})^2][\sum_{i=1}^{n}(x_{2i} - \bar{x_2})^2]}}$$

Our null and alternative hypotheses are as follows:

H_0: There is no correlation between the violence rate and unemployment.

H_A: There is a correlation between the violence rate and unemployment.

We use the `cor ()` function and `cor.test` from `base R` to conduct a Pearson's correlation:

```
cor(df$log_viol_r, df$unemployed)
```

```
## [1] 0.5368416
```

```
cor.test(~ log_viol_r + unemployed, data=df, method = "pearson", conf.level = 0.95)
```

```
##
##  Pearson's product-moment correlation
##
## data:  log_viol_r and unemployed
## t = 10.3, df = 262, p-value < 0.00000000000000022
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.4449525 0.6175447
## sample estimates:
##       cor
## 0.5368416
```

The `cor()` function gives the correlation but `cor.test()` gives more detail. Both give a positive correlation of 0.54. The coefficient is also an indication of the strength of the relationship. Jacob Cohen (1988) suggests that within the social sciences, a correlation of 0.10 may be defined as a small relationship; a correlation of 0.30, a moderate relationship; and a correlation of 0.50, a large relationship.

As our relationship was linear and our results are statistically significant, we reject the null hypothesis. We conclude that there is a statistically significant relationship between the violence rate and unemployment.

Now what about bivariate relationships where linearity is not met? This would call for either **Kendall's tau** and **Spearman's correlation**, nonparametric

versions of Pearson's correlation. Kendall's tau is more accurate when you have a small sample size compared to Spearman's rho.

We again use the function `cor.test ()` to conduct these:

```
# We purposely add an outlier so that the relationship is no longer linear
# This will produce a fictitious city with a very high level of unemployment and the lowest level
# For convenience, we will first further reduce the number of variables
df_1 <- select(df, unemployed, log_viol_r)

# To add cases to data frame, use the add_row() function from the tibble package
df_1 <- add_row(df_1, unemployed = 20, log_viol_r = 1)

# Conducting a Kendall
cor.test(~ log_viol_r + unemployed, data=df_1, method = "kendall", conf.level = 0.95)
```

```
##
##  Kendall's rank correlation tau
##
## data:  log_viol_r and unemployed
## z = 8.3925, p-value < 0.00000000000000022
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##      tau
## 0.345979
```

```
# Conducting a Spearman
cor.test(~ log_viol_r + unemployed, data=df_1, method = "spearman", conf.level = 0.95)
```

```
## Warning in cor.test.default(x = c(7.2830276, 6.79316, 6.478019, 5.949461, :
## Cannot compute exact p-value with ties
```

```
##
##  Spearman's rank correlation rho
##
## data:  log_viol_r and unemployed
## S = 1574389, p-value < 0.00000000000000022
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.4923882
```

The correlation coefficient is represented by tau for Kendall's rank (ranges from 0 to 1) and by Spearman's r (or rho) value (ranges −1 to 1) for Spearman's rank.

8.3 SUMMARY

Today was all about **effect sizes** where we learned how to produce them. For relationships between nominal variables, we have the following options: **phi** and **Cramer's V**. Then, for relationships between ordinal variables, we used **concordant and discordant pairs** to find effect sizes through **Gamma** and **Somers' D**. We then learned to produce effect sizes for relationships between numeric relationships, and an important assumption had to do with **covariation**: **Pearson's correlation** when the relationship is **linear** and nonparametric tests, **Kendall's tau** and **Spearman's correlation**, when the relationship is not.

Homework time!

Chapter 9

Regression

OLS & Logistic Regressions

Learning Outcomes:

- Learn how and when to use ordinary least squares and logistic regressions
- Understand what their outputs mean

Today's Learning Tools:

Data:

- Crime Survey for England and Wales (CSEW) from 2013 to 2014 sweep teaching dataset
- `Arrests` from the `effects` package

Packages:

- `arm`
- `car`
- `effects`
- `ggplot2`
- `here`
- `lessR`
- `sjplot`
- `tidyverse`

Functions introduced (and packages to which they belong)

- `complete.cases()` : Returns only complete cases that do not have NAs (`base R`)
- `display()` : Gives a clean printout of lm, glm, and other such objects (`arm`)
- `lm()` : Fit linear models (`base R`)
- `Logit()` : Fit logistic regression models with less typing (`lessR`)
- `qplot()` : Creates a variety of plots/graphs (`base R`)
- `relevel()` : Reorders the levels of a factor (`base R`)
- `tab_model()` : Creates HTML tables summarising regression models (`sjplot`)
- `vif()` : Calculate the variance inflation for OLS or other linear models (`car`)

9.1 Multiple and Simultaneous Relationships

Our learning on inferential statistics, so far, has been on single relationships between two variables. One of the major drawbacks of previous analyses of single relationships is that, even though you are able to test for statistical significance, you cannot ascertain prediction very well. In other words, you were unable to say for certain that one variable predicted another variable; we merely could say 'there was a relationship' or 'there was no relationship.'

Although we got into the habit of arranging our variables as independent and dependent variables, these terms are more relevant to regression. The reason is, not only can we establish significant relationships, we also can say with relatively more clarity, how that relationship looks like – which variable is impacting on what other variable.

Another major drawback of previous analyses on bivariate relationships is that you are less certain about whether the relationship and its effect size would still hold in the face of other variables. For example, would a significant relationship between peer group and violence still exist when the relationship between unemployment and violence is considered? This last lesson is on analyses that establish predictive relationships and test multiple relationships between two variables at the same time.

9.2 Today's 3

Our main topic of the day is **regression** and we learn two forms: ordinary least squares (OLS) and logistic. The former is applied to dependent variables that are measured at interval or numeric level, while the latter is applied to dependent variables that are binary and measured at the nominal level. The three topics are: assumptions of OLS regression; interpreting OLS regression; and logistic regression.

9.2.1 Assumptions of OLS Regression

Ordinary least squares (OLS) regression is a popular technique used to explore whether one or multiple independent variables (X) can predict or explain the variation in the dependent variable (Y). When multiple independent variables are included simultaneously, they are called **covariates**; when only one independent variable is used, the OLS regression is called **bivariate regression**. OLS regression has been the mainstay analysis in the social sciences.

To begin, we do the following:

1. Open up your existing `R` project
2. Install and load the required packages shown above
3. Import the dataset 'csew1314_teaching.csv', using the `read_csv()` function
4. Name the data frame `df`

For our example, we are interested in the bivariate relationship between age (X) and perception of antisocial behaviour (Y). Our research question is: 'Does age predict perceived level of antisocial behaviour in one's neighbourhood?' Our null and alternative hypotheses are as follows:

H_0: Age does not predict perceived level of antisocial behaviour in one's neighbourhood.

H_A: Age predicts perceived level of antisocial behaviour in one's neighbourhood.

Our initial step is to get to know our data:

```
# Age variable
summary(df$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   16.00   36.00   51.00   51.19   66.00   99.00     118
```

```
# Antisocial variable
summary(df$antisocx)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -4.015  -0.559   0.177   0.000   0.788   1.215   26695
```

Both variables have missing data. You could ignore the 'NAs', but when we run regression, it will conduct a *listwise deletion*, which is when cases are dropped from your analysis if they have missing data on either the independent variable (IV) or the dependent variable (DV). As we now want descriptive statistics of cases we will be using, we will delete the cases that will be dropped anyway. We use the `complete.cases ()` function to keep cases that have valid responses in both the IV and DV:

```
# Start with 35,371 cases
nrow(df)
```

```
## [1] 35371
```

```
# Retain only complete cases
df <- df[(complete.cases(df$age) & complete.cases(df$antisocx)), ]

# Left with 8,650 cases after dropping NAs
nrow(df)
```

```
## [1] 8650
```

```
# Summaries of just the 8,650 cases
summary(df$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   16.00   36.00   50.00   50.82   66.00   98.00
```

```
summary(df$antisocx)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -4.014557 -0.552077  0.178767  0.001823  0.788219  1.215267
```

To conduct OLS regression, or linear regression for short, a number of assumptions must be met, so you will need to test for them. Failure to do so will result in a possibly incorrect model and drawing conclusions from it would be silly. We only, however, briefly go through the first four assumptions because going into detail is beyond the scope of this class.

The purpose of today is more to introduce you to regression and learn how to use it and interpret the output. For this class, we find that the assumption of linearity is important to go into detail. Going through this assumption in detail will give you a better understanding of OLS regression itself and how it operates. We now go through each of the five assumptions of OLS regression before using the technique.

9.2.1.1 Independence of errors

Errors of the prediction, which make up the regression line, are assumed to be independent of each other. The term *heteroscedasticity* is used to describe this violation of independence of errors. When we violate this assumption, our points in the scatterplot resemble the shape of a funnel. If there is dependency between the observations, you would need to use other models that are not covered in this course.

9.2.1.2 Equal Variances of errors

If the variance of our residuals is unequal, we will need different estimation methods, but this issue is minor. The reason it is a minor issue is that regression is considered a *robust* estimation, meaning that it is not too sensitive to changes in the variance.

9.2.1.3 Normality of errors

Residuals are assumed to be normally distributed. Gelman and Hill (2007) believe this to be the least important of the assumptions because regression inferences tend to be robust regarding non-normality of the errors. Your results, however, may be sensitive to large outliers so you will need to drop them if appropriate.

9.2.1.4 Multicollinearity

When you are including more than one IV, **multicollinearity** may be a concern. Independent variables that are highly correlated with each other mean that

they are tapping into the same construct. For example, if you include two IVs, parental attachment and parental discipline, you may find that they are highly related to each other because both measure a similar idea.

One indication is that if you run a Pearson's correlation for the two IVs and get a value of 0.6 or 0.7, you should start investigating further. Another way to check for multicollinearity is through the *variance inflation factor* (VIF). How high the VIF should be before multicollinearity is a concern is debatable: here, anything above 5 should be a matter of concern. With this analysis, IVs should be numeric. We include an additional covariate, confidence in the police (`confx`), to illustrate. The VIF for our variables is conducted with the `vif ()` function in the `car` package:

```
cal_vif <- lm(antisocx ~ age + confx, data = df)

vif(cal_vif)
```

```
##      age    confx
## 1.003988 1.003988
```

The VIF values are below 5 and indicate that including both variables, `age` and `confx`, in the same model does not pose any multicollinearity concerns.

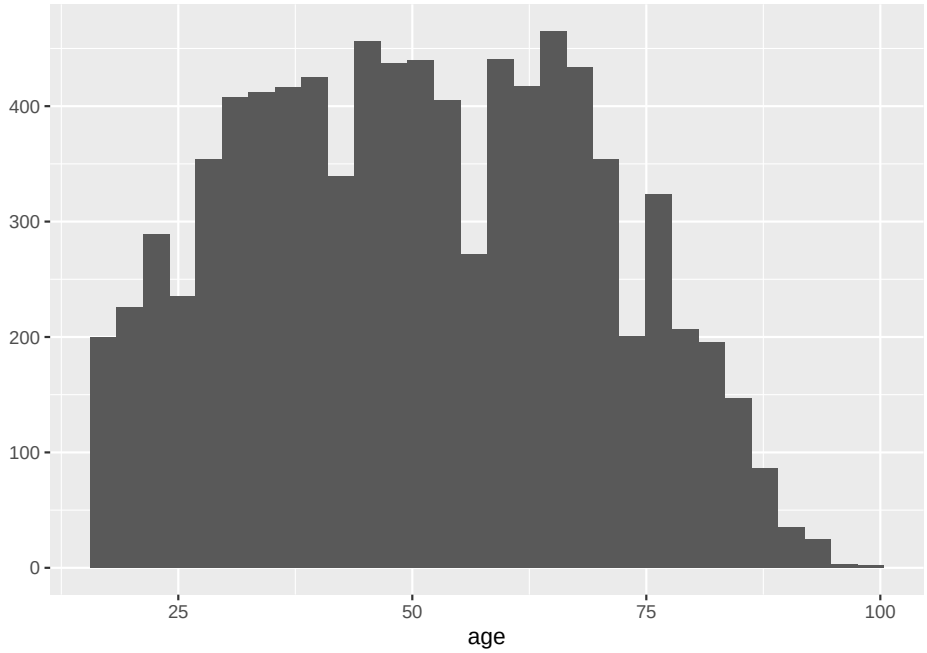
9.2.1.5 Linearity

Similar to Pearson's correlation, a linear relationship must be established before conducting the analysis. If the relationship is non-linear, your predicted values will be wrong, and it will systematically miss the true pattern of the mean of Y.

Say, for example, you randomly drew a case from the dataset. What would be your best guess on the value of that case's perceived antisocial behaviour in the neighbourhood (`antisocx`)? Likely, you say a value near the mean (M = 0.001823) of the sample. Next, we visualise our variable distributions:

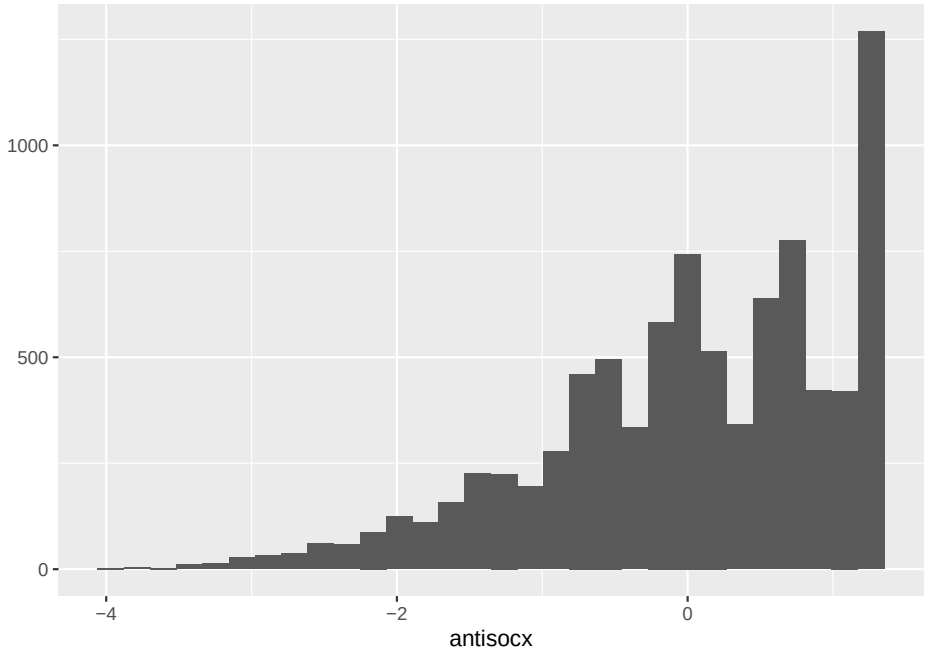
```
qplot(x = age, data = df)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

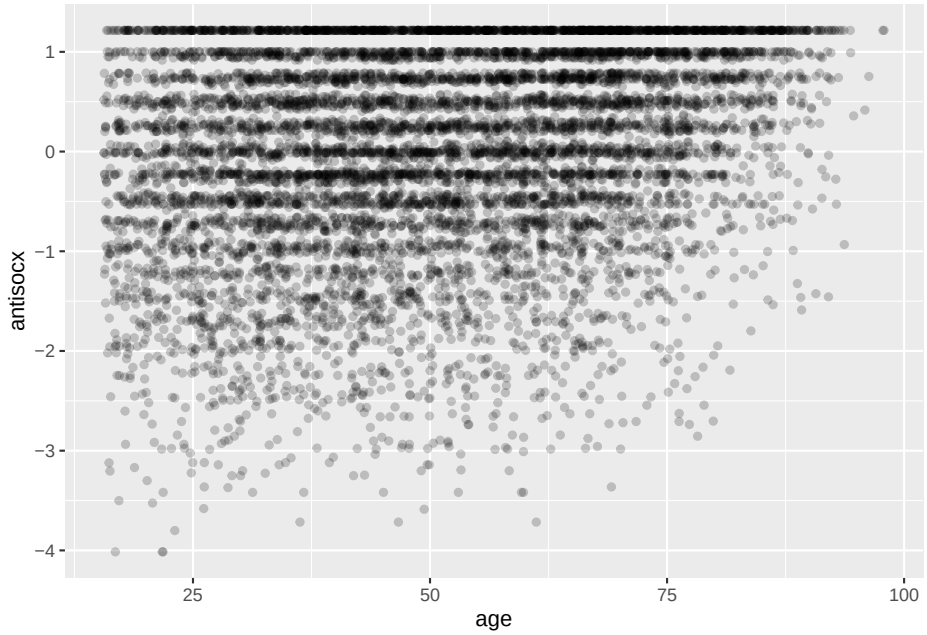



```
qplot(x = antisocx, data = df)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
# antisocx is negatively skewed

# Checking out how the variables covary together
ggplot(df, aes(x = age, y = antisocx)) +
  geom_point(alpha=.2, position="jitter")
```

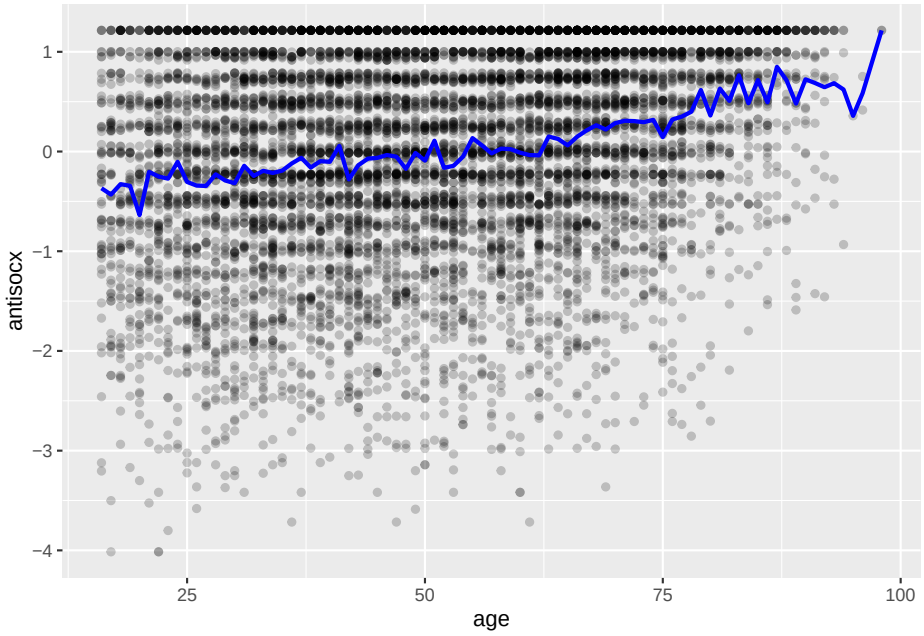


If you were to guess that case's perceived level of antisocial behaviour in the neighbourhood, but now based it on the scatterplot, and learned they were aged 30, what would your answer now be? Before, we based our guess on descriptive measures of central tendency and dispersion, which included all ages. Now, we are asked to guess knowing that the person is aged 30. If we were to now guess the level of perceived antisocial behaviour, we would try and guess the mean of those who are also aged 30. This is what is known as the *conditional mean* – the mean of Y for each value of X.

We create a trend line through the scatterplot, which represents the mean of the variable on the y-axis, `antisocx`, and we want the means computed for each age:

```
ggplot() +
  geom_point(data=df, aes(x = age, y = antisocx), alpha=.2) +
  geom_line(data=df, aes(x = age, y = antisocx), stat='summary', fun.y=mean,
            color="blue", size=1)
```

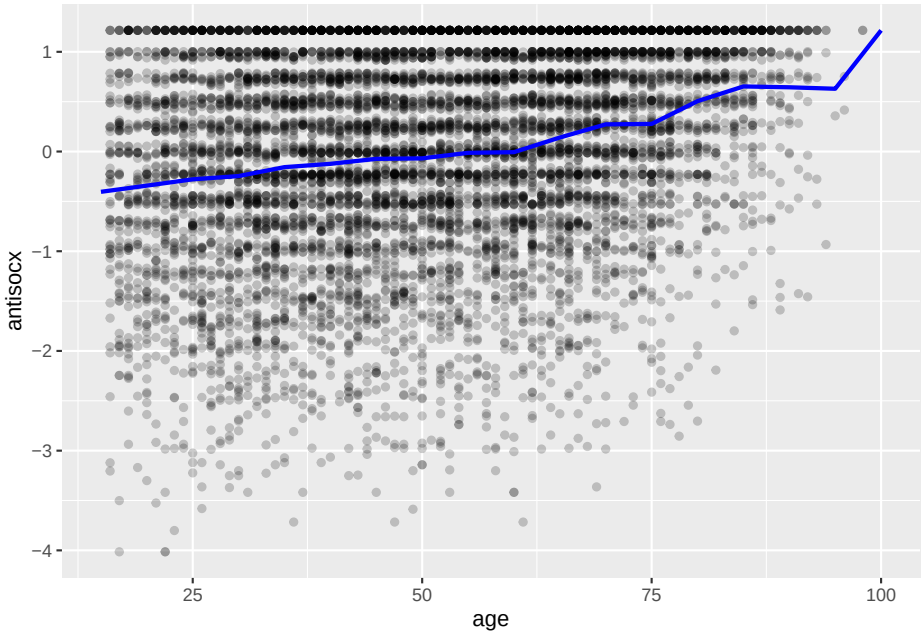
```
## No summary function supplied, defaulting to `mean_se()`
```



Plotting the conditional means shows us that the mean of perceived antisocial behaviour for those respondents around the age of 30 is around −0.3. This would be a better guess than the mean for all ages, 0.0018. The trend line gives us a better idea of what is going on in our scatterplot, but the line looks a bit rough. We can make it smoother:

```
# Make smoother by calculating average perceived antisocial behaviour for age in increments of 5
ggplot() +
  geom_point(data=df, aes(x = age, y = antisocx), alpha=.2) +
  geom_line(data=df, aes(x = round(age/5)*5, y = antisocx), stat='summary', fun.y=mean,
            color="blue", size=1)
```

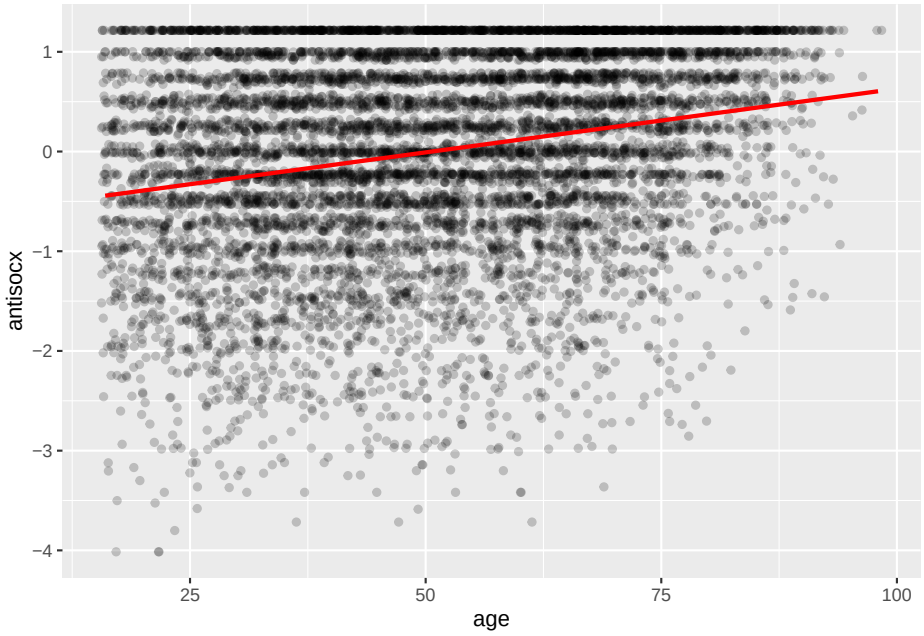
```
## No summary function supplied, defaulting to `mean_se()`
```



The blue trend line shows us that there is an upward trend, meaning that as age increases, so do perceptions of antisocial behaviour in the neighbourhood. Like what we did above, OLS regression tries to capture the trend or pattern of the data by drawing a straight line of predicted values as the model of the data:

```
# method=lm asks for the linear regression line
# se=FALSE asks not to print confidence interval
# Other arguments specify the colour, thickness of the line
ggplot(data = df, aes(x = age, y = antisocx)) +
  geom_point(alpha = .2, position = "jitter") +
  geom_smooth(method = "lm", se = FALSE, color = "red", size = 1)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



The red **regression line** gives you guesses or predictions for the value of perceived level of antisocial behaviour based on information that we have about age. The line can also be seen as one that tries to summarise the pattern of what is going on among points. The line is, of course, linear. It does not go through all the points, however. As Bock et al. (2012) highlight:

> 'Like all models of the real world, the line will be wrong – wrong in the sense that it can't match reality exactly. But it can help us understand how the variables are associated' (p. 179).

A map is never a perfect representation of the world; the same happens with statistical models. Yet, as with maps, models can be helpful.

9.2.2 Interpreting OLS Regression

In this section, we learn how to interpret bivariate and multiple regression output. Now that we know what the regression line is, how do we draw one? Two points are needed to do so:

1. We need to know where the line begins. The **intercept** is where it begins, which is the value of Y, our DV, when X (our IV) is 0.

2. We also need to know the angle of that line. This is referred to as the **slope**.

This translates into the equation:

$$y = b_0 + b_1 x_i$$

Where b_0 is the y-intercept and $b_1 x_i$ is the slope of the line. Linear regression models try to minimise the distance from every point in the scatterplot to the regression line. This is called *least squares estimation*. The farther these points are from the regression line, the more error your regression model will have.

It is not unusual to observe that some points will fall above the line (a positive error value), while other points will fall below it (a negative error value). If we wanted to sum these error values, the problem is that the positive values would cancel out the negative values, underreporting our overall error. This would then incorrectly suggest that our regression line was perfect.

To resolve this issue, the error values are squared before they are summed. This is called the *error sum of squares*. Our regression model is trying to find a line fit that has the least (squared) error. Hence, least squared estimation.

9.2.2.1 Bivariate Regression

To fit the model, we use the `lm()` function using the formula specification `(Y ~ X)`:

```
fit_1 <- lm(antisocx ~ age, data = df)

# Get to know the data
class(fit_1)
```

```
## [1] "lm"
```

```
attributes(fit_1)
```

```
## $names
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
##
## $class
## [1] "lm"
```

```
summary(fit_1)
```

```
##
## Call:
## lm(formula = antisocx ~ age, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8481 -0.5752  0.1297  0.7691  1.6581
##
## Coefficients:
##              Estimate Std. Error t value            Pr(>|t|)
## (Intercept) -0.647232   0.030455  -21.25 <0.0000000000000002
## age          0.012773   0.000563   22.68 <0.0000000000000002
##
## Residual standard error: 0.9705 on 8648 degrees of freedom
## Multiple R-squared:  0.05616,    Adjusted R-squared:  0.05606
## F-statistic: 514.6 on 1 and 8648 DF,  p-value: < 0.00000000000000022
```

```
# A more concise display of results is using 'display ()' from arm package:
display(fit_1)
```

```
## lm(formula = antisocx ~ age, data = df)
##             coef.est coef.se
## (Intercept) -0.65     0.03
## age          0.01     0.00
## ---
## n = 8650, k = 2
## residual sd = 0.97, R-Squared = 0.06
```

Now let us interpret the regression output. There are several points to focus on when communicating your results:

1. Beta (or regression) coefficient: this is the value that measures the impact of the independent variable on the dependent variable. It is the b_1 regression coefficient from the previous equation, the value that will shape the slope for this model. From our output, this value is 0.012773. When the value is positive, it tells us that for every 1 unit increase in X, there is a b_1 increase on Y; if, however, the coefficient is negative, then it represents a decrease on Y. Here, we interpret the result as: 'For every 1 year older, there is a 0.01 unit increase in level of perceived antisocial behaviour in the neighbourhood.' The coefficient can be interpreted as a measure of the effect size of this relationship. In addition, with very large sample sizes,

you should place less of an emphasis on p-values and more emphasis on the size of the beta coefficients because p-values are sensitive to sample size. For example, you may have all statistically significant covariates, even though the variables have small coefficients. This means that they do not have much of an effect on the DV.

2. P-value: The p-value for each beta coefficient tests the null hypothesis that the coefficient is equal to zero. A low p-value (< 0.05) means you can reject the null hypothesis and that the changes in the predictor's value are significantly related to changes in the DV value. For statistically non-significant coefficients, it is not recommended that you interpret them; you can, however, make statements about their general trend (positive or negative). In our output, the p-value is very small ($p < .001$). We conclude that age significantly predicts level of perceived antisocial behaviour and that we can reject the null hypothesis.

3. F-statistic: There is another p-value at the bottom of your regression output and belongs to the F-statistic. This assesses whether our overall model can predict the DV with high confidence, especially if we have multiple predictors. This, too, is statistically significant, meaning that at least one of our inputs must be related to our DV.

4. R^2: This is known as the percent of variance explained, and it is a measure of the strength of our model. This value ranges from 0 to 1. Towards 1, the better we are able to account for variation in our outcome Y with our IV(s). In other words, the stronger the relationship is between Y and X. Weisburd and Britt (2009: 437) suggest that, in criminal justice research, values greater than .40 are rare, but if we obtain such a value, it is considered a powerful model. If the value is lower than .20, the model is considered relatively weak. The output shows that R^2 for our model was about .06 (the multiple r-squared value). We interpret this as our model explains 6% of the variance in the level of perceived antisocial behaviour.

To understand R^2, we refer to **residuals** (e). These are the distance between each point in the dataset and the regression line. The distance represents the amount of error. Previously, this was mentioned in the context of least squares estimation. Residuals matter because it tells us how much variation is still unexplained and considers the fact that we cannot perfectly predict perceived antisocial behaviour each time by using the information on age. In actuality, our regression equation should be:

$$y = b_0 + b_1 x_i + e$$

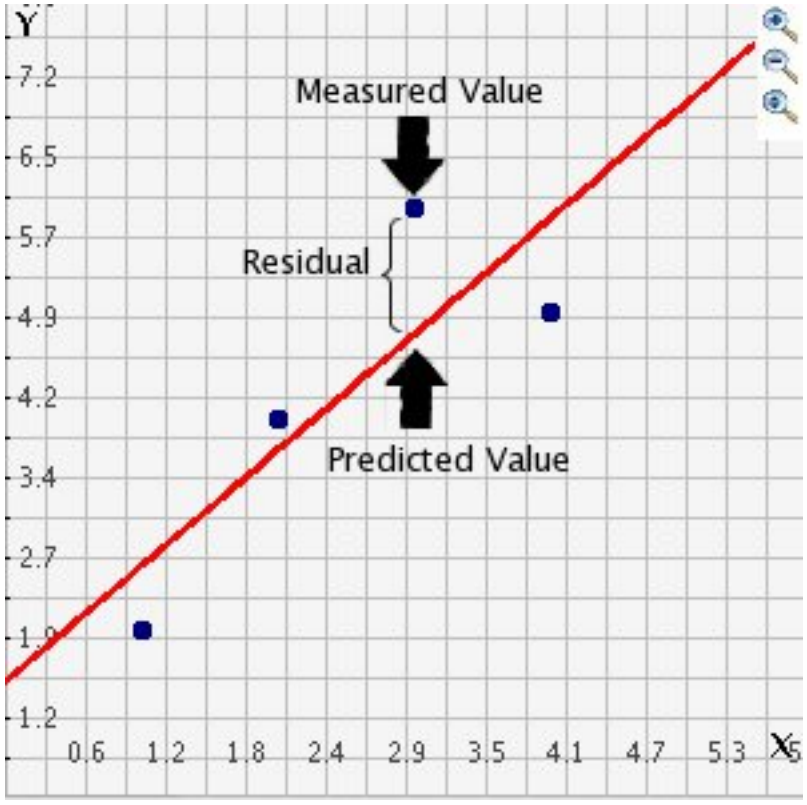


Figure 9.1: **Figure 9.1** Residuals visualized

9.2.2.2 Multiple Regression

At the start of the session, there was mention of one of the major drawbacks of looking at only bivariate relationships. One was that these analyses did not consider other possible explanations. It could be that you have a statistically significant relationship, but without considering other variables that may explain this relationship, your significant result could, in fact, be *spurious* – an association that should not be taken very seriously. For some ridiculous examples, click here.

Multiple regression is a way for assessing the relevance of competing explanations. For example, we think there is a relationship between age and perceived antisocial behaviour levels, but we have not considered whether other reasons may provide a better explanation for these perceptions. Maybe previous victimisation or fear of crime may be better contenders.

The equation for multiple regression is similar to bivariate regression except it includes more coefficients:

$$y = b_0 + b_1 x_i + b_2 x_{ii} + ...b_n x_n + e$$

For this example, we include additional variables to our model: sex (`sex`), previous victimisation in past 12 months (`bcsvictim`), and confidence in the police (`confx`). First, we check out these variables and tidy them as necessary. Then we will run our regression:

```
# Checking out what type of class

class(df$confx)
```

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

```
class(df$bcsvictim)
```

```
## [1] "integer"
```

```
# attach labels to bcsvictim and make ordering false
df$bcsvictim <- factor(df$bcsvictim, labels = c("not victim", "Yes victim"), ordered =

# Sex
class (df$sex)
```

```
## [1] "integer"
```

```
table(df$sex)
```

```
##
##    1    2
## 4064 4586
```

```
# Recode sex as dummy variable (0 and 1) where female is 0 and ordered is false
df$gender <- df$sex
df$gender[df$gender=="2"] <- "0"
df$gender <- factor(df$gender, labels = c("female", "male"), ordered = FALSE)
# Double check values match original variable
table(df$gender)
```

```
##
## female   male
##   4586   4064
```

```
# Now let us run the multiple regression

fit_2 <- lm(antisocx ~ age + gender + bcsvictim + confx, data = df)
summary(fit_2)
```

```
##
## Call:
## lm(formula = antisocx ~ age + gender + bcsvictim + confx, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8854 -0.5684  0.1142  0.7195  2.5249
##
## Coefficients:
##                       Estimate Std. Error t value             Pr(>|t|)
## (Intercept)         -0.5136682  0.0328072 -15.657 < 0.0000000000000002
## age                  0.0104148  0.0005699  18.276 < 0.0000000000000002
## gendermale           0.1003834  0.0207701   4.833              0.00000137
## bcsvictimYes victim -0.4310943  0.0290209 -14.855 < 0.0000000000000002
## confx                0.2038268  0.0106002  19.229 < 0.0000000000000002
##
## Residual standard error: 0.9361 on 8151 degrees of freedom
##   (494 observations deleted due to missingness)
## Multiple R-squared:  0.1247, Adjusted R-squared:  0.1243
## F-statistic: 290.3 on 4 and 8151 DF,  p-value: < 0.00000000000000022
```

We return to those four points to focus on when confronted with a regression output:

1. Regression coefficients: The usual interpretation is to state that Y changes for every one-unit increase in X *when the other variables in the model are held constant* (or are 'controlled for'). Therefore, a 0.01 increase in level of perceived antisocial behaviour in the neighbourhood is associated with a one-year increase in age, for example. This is similar to the bivariate regression. For categorical variables, however, the interpretation is different; notice how these variables have an additional label attached to them in the output. For example, `gender` includes 'male' and `bcsvictim` includes 'yes victim'. Labels that do not appear next to the name of the categorical variable in the model are called the *reference category*. This category is the lowest value. You can select your reference category based on the modal category or if it is a category in which you are least interested. We interpret them as the following: males have a perceived antisocial behaviour level that averages 0.10 units higher than that of females; victims have a perceived antisocial behaviour level that averages .43 units lower than that of non-victims. A one-point score increase for confidence in neighbourhood police is related to .20 increase in level of perceived antisocial behaviour. For each interpretation, covariates in the model are controlled for.

2. P-value: All coefficients are statistically significant, meaning that we can reject the null hypothesis that no differences exist on perceived antisocial behaviour.

3. F-statistic: Now that we have more than one IV in our model, we evaluate whether all of the regression coefficients are zero. The F-statistic is below the conventional significance level of $= 0.05$, so at least one of our IVs is related to our DV. You will also notice t-values for each IV. These are testing whether each of the IVs is associated with the DV when controlling for covariates in the model.

4. R^2: The R^2 is higher than that of our previous bivariate regression model. It will, however, inevitably increase whenever new variables are added to the model, regardless if they are only weakly related to the DV. Having said this, the new R^2 value suggests that the addition of the three new variables improves our previous model.

If you would like to professionally present your results, you can use the function `tab_model ()` from the `sjplot` package:

```
tab_model(fit_2)
```

antisocx

Predictors

Estimates

CI

p

(Intercept)

-0.51

-0.58 – -0.45

<0.001

age

0.01

0.01 – 0.01

<0.001

gender [male]

0.10

0.06 – 0.14

<0.001

bcsvictim [Yes victim]

-0.43

-0.49 – -0.37

<0.001

confx

0.20

0.18 – 0.22

<0.001

Observations

8156

R2 / R2 adjusted

0.125 / 0.124

```
# Change name of DV
tab_model(fit_2, dv.labels = "Perceived Antisocial Behaviour")
```

Perceived Antisocial Behaviour

Predictors

Estimates

CI

p

(Intercept)

-0.51

-0.58 – -0.45

<0.001

age

0.01

0.01 – 0.01

<0.001

gender [male]

0.10

0.06 – 0.14

<0.001

bcsvictim [Yes victim]

-0.43

-0.49 – -0.37

<0.001

confx

0.20

0.18 – 0.22

<0.001

Observations

8156

R2 / R2 adjusted

0.125 / 0.124

```
# Change name of IVs
tab_model(fit_2, pred.labels = c("(Intercept)", "Age", "Sex", "Victimisation", "Confidence in Pol
```

Perceived Antisocial Behaviour

Predictors

Estimates

CI

p

(Intercept)

-0.51

-0.58 – -0.45

<0.001

Age

0.01

0.01 – 0.01

<0.001

Sex

0.10

0.06 – 0.14

<0.001

Victimisation

-0.43

-0.49 – -0.37

<0.001

Confidence in Police

0.20

0.18 – 0.22

<0.001

Observations

8156

R2 / R2 adjusted

0.125 / 0.124

9.2.3 Logistic Regression

Now, what if our dependent variable is categorical and binary? This is where **logistic regression**, a technique belonging to a family of techniques called **generalized linear models**, is appropriate. Those assumptions of linearity and normality are not issues because this technique, itself, is not based on the normal distribution.

In criminology, it is often the case that our DV has only two categories. For example, victim or not a victim; arrested or not arrested. This form of regression models the probability of belonging to one of the levels of the binary outcome.

For this example, we are interested in the relationship between race and receiving harsher treatment for possession of marijuana. We use the dataset `Arrests` from the `effects` package. This data includes information on police treatment of individuals arrested for possession of marijuana in Toronto, Canada. Our DV is whether the arrestee was released with summons; if they were not, they received harsh treatment such as being brought to the police station or held for bail:

```
library(effects)
data(Arrests, package="effects") # It will tell you that the dataset are not found, bu

# Checking out our DV
table(Arrests$released)
```

```
##
##   No  Yes
##  892 4334
```

```
# Checking out the order of the levels in the DV
attributes(Arrests$released)
```

```
## $levels
## [1] "No"  "Yes"
##
## $class
## [1] "factor"
```

Logistic regression will predict probabilities associated with the level whose first letter is further in the alphabet — this would be 'yes', the respondent was released with a summons. But this is not our level of interest. We will need to reorder the levels in the DV using the `relevel` function:

```
# Reverse the order
#Rename the levels so that it is clear we now mean 'yes' to harsh treatment
Arrests$harsher <- relevel(Arrests$released, "Yes")
Arrests$harsher <- factor(Arrests$harsher, labels = c("No", "Yes"), ordered = TRUE)

#  Check that it matches the original variable 'released' but in reverse order
table(Arrests$harsher)
```

```
##
##   No  Yes
## 4334  892
```

```
table(Arrests$released)
```

```
##
##   No  Yes
##  892 4334
```

```
# We will also reverse the levels in our race variable so that 'white' is '0'
table(Arrests$colour)
```

```
##
## Black White
##  1288  3938
```

```
Arrests$race <- Arrests$colour
Arrests$race <- relevel(Arrests$race, "White")

# Check to see coding has changed
table(Arrests$race)
```

```
##
## White Black
##  3938  1288
```

Now that we have arranged the codes of our variables of interest, we fit the logistic regression model using the `glm ()` function and specify a logit model (`family="binomial"`). We include three other variables in the model to see to what extent race seems to matter in harsher police treatment even when we adjust or consider sex, employment, and previous police contacts:

```
fit_3 <- glm(harsher ~ race + checks + sex + employed, data=Arrests, family = "binomial
summary(fit_3)
```

```
##
## Call:
## glm(formula = harsher ~ race + checks + sex + employed, family = "binomial",
##     data = Arrests)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5226  -0.6156  -0.4407  -0.3711   2.3449
##
## Coefficients:
##             Estimate Std. Error z value             Pr(>|z|)
## (Intercept) -1.90346    0.15999 -11.898 < 0.0000000000000002
## raceBlack    0.49608    0.08264   6.003        0.00000000194
## checks       0.35796    0.02580  13.875 < 0.0000000000000002
## sexMale      0.04215    0.14965   0.282                0.778
## employedYes -0.77973    0.08386  -9.298 < 0.0000000000000002
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4776.3  on 5225  degrees of freedom
## Residual deviance: 4330.7  on 5221  degrees of freedom
## AIC: 4340.7
##
## Number of Fisher Scoring iterations: 5
```

The output is similar to the one produced for linear regression, although there are some differences such as the Akaike information criterion (AIC) and the z-statistic, which is a similar idea to the t-test in linear regression. The z-statistic is a test of statistical significance and assesses whether each variable in the model is associated with the DV.

All IVs are significantly associated with harsher police treatment except for sex. We see that every one unit increase in the number of previous police contacts, the log odds of receiving harsher treatment (versus being released on summons) increases by 0.36 adjusting for the other variables in the model.

Observe that variables that are categorical have a label next to them in the output. For example, sex has 'male' next to it; race has 'black' next to it; employed has 'yes' next to it. These are the categories you are comparing to the reference category (whichever category is coded '0'). So, being black increases the log odds of receiving harsh treatment by 0.49 compared to being white, while being employed decreases the log odds of harsh treatment by 0.77 compared to being unemployed.

This is good, but most people find interpreting log odds, well, odd – it is not very intuitive. It is common to use the **odds ratio** (OR) to interpret logistic regression. We convert these values into ORs:

```
exp(coef(fit_3))
```

```
## (Intercept)   raceBlack      checks     sexMale employedYes
##   0.1490516   1.6422658   1.4304108   1.0430528   0.4585312
```

```
# We can add confidence intervals
exp(cbind(OR = coef(fit_3), confint(fit_3)))
```

```
## Waiting for profiling to be done...
```

```
##                    OR     2.5 %    97.5 %
## (Intercept) 0.1490516 0.1081600 0.2026495
## raceBlack   1.6422658 1.3957633 1.9298763
## checks      1.4304108 1.3601419 1.5049266
## sexMale     1.0430528 0.7830594 1.4090622
## employedYes 0.4585312 0.3892103 0.5407210
```

```
# Another way of getting the results, which uses less typing, is to use the Logit() function from

Logit(harsher ~ checks + colour + sex + employed, data=Arrests, brief=TRUE)
```

```
##
## Response Variable:   harsher
## Predictor Variable 1:  checks
## Predictor Variable 2:  colour
## Predictor Variable 3:  sex
## Predictor Variable 4:  employed
##
## Number of cases (rows) of data:  5226
## Number of cases retained for analysis:  5226
##
##
##
##    BASIC ANALYSIS
##
## Model Coefficients
##
##              Estimate    Std Err  z-value  p-value   Lower 95%   Upper 95%
## (Intercept)   -1.4074     0.1724   -8.162    0.000     -1.7453     -1.0694
```

```
##      checks    0.3580     0.0258   13.875    0.000      0.3074      0.4085
## colourWhite   -0.4961     0.0826   -6.003    0.000     -0.6580     -0.3341
##     sexMale    0.0422     0.1496    0.282    0.778     -0.2511      0.3355
## employedYes   -0.7797     0.0839   -9.298    0.000     -0.9441     -0.6154
##
##
## Odds ratios and confidence intervals
##
##              Odds Ratio   Lower 95%   Upper 95%
## (Intercept)      0.2448      0.1746      0.3432
##      checks      1.4304      1.3599      1.5046
## colourWhite      0.6089      0.5179      0.7160
##     sexMale      1.0431      0.7779      1.3986
## employedYes      0.4585      0.3890      0.5404
##
##
## Model Fit
##
##     Null deviance: 4776.258 on 5225 degrees of freedom
## Residual deviance: 4330.699 on 5221 degrees of freedom
##
## AIC: 4340.699
##
## Number of iterations to convergence: 5
##
##
##
##
## >>> Note:  colour is not a numeric variable.
##
##
##
## >>> Note:  sex is not a numeric variable.
##
##
##
## >>> Note:  employed is not a numeric variable.
##
## Collinearity
##
##
## >>> No collinearity analysis because not all variables are numeric.
```

If the odds ratio is greater than 1, it indicates that the odds of receiving harsh treatment increases when the independent variable, too, increases. Previous police contacts increase the odds of harsher treatment by 43% and being black

increases the odds of harsher treatment by 64% (all the while adjusting for the other variables in the model). Employment, however, has an odds ratio of 0.45. If the OR is between 0 and 1, it is considered a negative relationship. Referring to the confidence intervals is also helpful: if 0 is within the interval, the result is statistically non-significant. Take the 95% CI of sex, for example.

For both types of regression, it is inappropriate to make comparisons between IVs. You cannot directly compare the coefficients of one IV to the other, unless they use the same metric. For more ways to interpret ORs in logistic regression, refer to the UCLA Institute for Digital Research and Education

9.2.3.1 Model Fit

Now, how well is the fit of our logistic regression model? In order words, how well does our IVs collectively predict the DV? Recall from OLS regression, we assessed how well our overall model predicts the DV by observing the *F-statistic* and R^2. This is not the case for logistic regression. The reason is, in our familiar linear regression, we understand residual variation (related to that regression line) through the error sum of squares. In logistic regression, there is not one way to understand residual variation, but several, because of its binary outcome. This results in different ways of understanding model fit: (1) quantitative prediction, focused on how close the prediction is to being correct; (2) qualitative prediction, focused on whether the prediction is correct or incorrect; or (3) both. We go through the first one.

9.2.3.2 *Quantitative Prediction*

A common measure for evaluating model fit is the **deviance**, also known as -2LL. It measures how accurate the prediction is by multiplying the log likelihood statistic by -2. This *log-likelihood statistic* measures how much unexplained variance remains after the model is fitted, whereby larger values reflect poorer fit. Larger values of -2LL, similarly, indicate worse prediction.

The logistic regression model can be estimated using an approach of probability called *maximum likelihood estimation*, which calculates the probability of obtaining a certain result given the IVs in the model. It does so by minimising that -2LL. This is like what OLS regression does with the error sum of squares – tries to minimise it. Thus, -2LL is to logistic regression as the error sum of squares is to OLS regression.

When we re-run our model, we find that -2LL is in the form of *null* and *residual* deviances towards the bottom of the output:

```
Logit(harsher ~ checks + colour + sex + employed, data=Arrests, brief=TRUE)
```

```
##
## Response Variable:   harsher
## Predictor Variable 1:  checks
## Predictor Variable 2:  colour
## Predictor Variable 3:  sex
## Predictor Variable 4:  employed
##
## Number of cases (rows) of data:  5226
## Number of cases retained for analysis:  5226
##
##
##
##    BASIC ANALYSIS
##
## Model Coefficients
##
##              Estimate    Std Err  z-value  p-value   Lower 95%   Upper 95%
## (Intercept)   -1.4074     0.1724   -8.162    0.000     -1.7453     -1.0694
##      checks    0.3580     0.0258   13.875    0.000      0.3074      0.4085
## colourWhite   -0.4961     0.0826   -6.003    0.000     -0.6580     -0.3341
##     sexMale    0.0422     0.1496    0.282    0.778     -0.2511      0.3355
## employedYes   -0.7797     0.0839   -9.298    0.000     -0.9441     -0.6154
##
##
## Odds ratios and confidence intervals
##
##              Odds Ratio   Lower 95%   Upper 95%
## (Intercept)      0.2448      0.1746      0.3432
##      checks      1.4304      1.3599      1.5046
## colourWhite      0.6089      0.5179      0.7160
##     sexMale      1.0431      0.7779      1.3986
## employedYes      0.4585      0.3890      0.5404
##
##
## Model Fit
##
##     Null deviance: 4776.258 on 5225 degrees of freedom
## Residual deviance: 4330.699 on 5221 degrees of freedom
##
## AIC: 4340.699
##
## Number of iterations to convergence: 5
##
##
##
##
```

```
## >>> Note:  colour is not a numeric variable.
##
##
##
## >>> Note:  sex is not a numeric variable.
##
##
##
## >>> Note:  employed is not a numeric variable.
##
## Collinearity
##
##
## >>> No collinearity analysis because not all variables are numeric.
```

The null deviance is the value of -2LL for a model without any IVs, whereas the residual deviance is the value of -2LL for our present model. We test to see if the null hypothesis, that the regression coefficients (IVs) equal zero, is true. We do so by calculating the *model chi-squared*. This is the difference between both deviances:

```
# All the information we need to calculate the model chi-squared is already in our model object
names(fit_3)
```

```
##  [1] "coefficients"      "residuals"         "fitted.values"
##  [4] "effects"           "R"                 "rank"
##  [7] "qr"                "family"            "linear.predictors"
## [10] "deviance"          "aic"               "null.deviance"
## [13] "iter"              "weights"           "prior.weights"
## [16] "df.residual"       "df.null"           "y"
## [19] "converged"         "boundary"          "model"
## [22] "call"              "formula"           "terms"
## [25] "data"              "offset"            "control"
## [28] "method"            "contrasts"         "xlevels"
```

```
# We now calculate said chi-squared by taking the difference of the deviances
with(fit_3, null.deviance - deviance)
```

```
## [1] 445.5594
```

Our obtained value is: 445.56. We do not know, though, how big or small this value is to determine how predictive our model is. We will need a test of statistical significance:

```
# Obtaining the p-value
with(fit_3, pchisq(null.deviance - deviance, df.null - df.residual, lower.tail = FALSE)
```

```
## [1] 3.961177e-95
```

```
# Incorporating previous calculation of difference between deviances
# df.null - df.residual: calculating the degrees of freedom (should equal the number o
```

The p-value is smaller than $= 0.05$, so we conclude that our model is a better fit than a model with no IVs (or predictors).

In addition, Professor Scott Menard (2010) suggests calculating the *likelihood ratio* R^2, also known as the Hosmer/Lemeshow R^2, by taking the difference of the previous deviances and dividing it by the null deviance. The likelihood ratio R^2 tells you the extent to which including your IVs in your model reduces the variation, or error, as measured by the null deviance. The value can range from 0 to 1, where higher values mean better predictive accuracy:

```
#Likelihood ratio R2
with(fit_3, (null.deviance - deviance)/null.deviance)
```

```
## [1] 0.09328629
```

This particular R^2 is considered a 'pseudo- R^2' as it is not the same as the one obtained in OLS/ linear regression. There are many kinds of pseudo ones but Menard recommended the likelihood ratio one because (1) of its conceptual similarity to the R^2 in OLS regression; (2) it is not easily swayed by the base rate (the number of cases that have the charateristic of interest); (3) it is easy to interpret; and (4) it can also be used in other generalised linear models, such as ones with categorical DVs with more than two levels.

9.3 SUMMARY

Regression allows us to look at relationships between variables while including other variables, called **covariates**, in the model. For OLS regression, assumptions must be met before embarking on fitting this model, and has to do with building the **regression line** whereby the line starts at the **intercept** and its angle is determined by the **slope**. **Residuals** form the equation to determine the R^2 and is the error of the model. When the DV is binary, logistic regression is appropriate. The **odds ratio** is an easier way to interpret logistic regression.

We assess model fit through quantitative prediction measures of deviance, and it is recommended to use the likelihood ratio R^2 instead of other pseudo R^2s

Homework time!