Contents

4.11 SD versus estimate of SD
4.12 The Big Short

5 Inference and Modeling
5.1 Motivating Example: Election Forecasting
5.2 The Sampling Model for Polls
5.3 Populations, samples, parameters and estimates
5.4 Central Limit Theorem in Practice
5.5 Confidence Intervals
5.6 p-values
5.7 Statistical Models
5.8 Bayesian Statistics
5.9 Bayes in practice
5.10 Election Forecasting

6 Data Wrangling
6.1 Importing Data
6.2 Tidy data
6.3 Reshaping data
6.4 Combining tables
6.5 Parsing html
6.6 String Processing
6.7 Regular Expressions
6.8 Parsing Dates and Times
6.9 Text mining

7 Regression
7.1 Motivating Example: Money Ball
7.2 Motivation
7.3 Correlation
7.4 Stratification
7.5 Bivariate Normal Distribution
7.6 Confounding
7.7 Multivariate Regression
7.8 Linear Models
7.9 Least Squares Estimates (LSE)
7.10 Advanced dplyr: tibbles and do
7.11 Broom
7.12 Building a better offensive metric for baseball
7.13 On base plus slugging (OPS)
7.14 Regression Fallacy
7.15 Measurement error models
7.16 Correlation is not causation

8 Machine Learning
8.1 Motivating example: Netflix Challenge
8.2 Loss Functions
8.3 Matrices
8.4 Cross validation
8.5 Multivariate Regression
8.6 Logistic Regression
8.7 Smoothing
8.8 k Nearest Neighbors
8.9 Decision Trees
8.10 Random Forrests
8.11 Distance
8.12 Dimension Reduction
8.13 Regularization

9 Data Science Toolbox
9.1 The unix shell
9.2 RStudio
9.3 R markdown
9.4 git and GitHub

Contents

1 Introduction to R

In this book we will be using the R software environment for all our analysis. Throughout the book you will learn R and data analysis techniques simultaneously. However, we need to introduce basic R syntax to get you going. In this chapter, rather than cover every R skill you need, we introduce just enough so that you can follow along the remaining chapters where we provide more in depth coverage, building upon what you learn in this chapter. We find that we better retain R knowledge when we learn it to solve a specific problem.

In this chapter, as done throughout the book, we will use a motivating case study. We ask a specific question related to crime in United States and provide a relevant dataset. Some basic R skills will permit us to answer the motivating question.

1.1 US gun murders

Imagine you live in Europe and are offered a job in a US company with many locations across all states. It is a great job but news with headlines such as **US Gun Homicide Rate Higher Than Other Developed Countries** have you worried. Charts like this make you worry even more:

Or even worse, this version from everytown.org:

But then you are reminded that the US is a large and diverse country with 50 very different states as well as the District of Columbia (DC).

California, for example, has a larger population than Canada and 20 US states have populations larger than that of Norway's. In some respects the variability across states in the US is akin to the variability across countries in Europe. Furthermore, although not in the charts above, the murder rates in Lithuania, Ukraine, and Russia are higher than 4 per 100,000. So perhaps the news reports that worried you are too superficial. You have options of where to live and want to find out how safe each state is. We will gain some insights by examining data related to gun homicides in the US using R.

Now before we get started with our example, we need to cover logistics as well as some of the very basic building blocks that we need to gain more advanced R skills. Be aware that for some of these, it is not immediately obvious how it is useful, but later in the book you will appreciate having the knowledge under your belt.

1.2 Why R?

R is not a programming language like C or Java, it was not created by software engineers for software development. Instead, it was developed by statisticians as an interactive environment for data analysis. You can read the full history here [LINK TO HISTORY OF S ETC..]. The interactivity is an indispensable feature in data science because, as you will soon learn, the ability to quickly explore data is a necessity for success in this field. However, like in other programming languages, you can save your work as scripts which you can easily execute at any moment. These scripts serve as a record of the analysis you performed, a key feature that facilitated reproducible work. If you are an expert programmer, you should not expect R to follow the conventions you are used to as you will be disappointing. If you bare with us, you will come to appreciate the unequal power of R when it comes to data analysis and data visualization in particular.

Other attractive features of R are the following:

1. Is is free and open soucre
2. It runs across all major platforms: Windows, Mac Os, Unix/Linux.
3. Scripts and data objects can be shared seamlessly across platforms.
4. There is a large, growing, and active community of R users and a result there are numerous resources for learning [ADD LINKS] and asking questions [ADD LINKS].
5. It easy for others to contribute add-ons which enables developers to share software implementations of new data science methodologies. This gives R users early access to the latest methods and to tools which are developed for a wide variety of disciplines including ecology, molecular biology, social sciences, geography, just to name a few.

1.3 Getting Started

1.3.1 Installing R

You can download R freely from the Comprehensive R Archive Network (CRAN). It is relatively straightforward, but if you need further help you can try the following resources:

- Installing R on Windows
- Installing R on Mac
- Installing R on Ubuntu

If you want to try out R without installing it you can access a web based consoles such as R fiddle.

1.3.2 The R console

Interactive data analysis usually occurs on the *R console* that executes commands as you type them. There are several ways to gain access to an R console. One way is to simply start R on your computer. The console looks something like this:

As a quick example, try using the console to calculate a 15% tip on a meal that cost $19.71:

```
0.15 * 19.71
#> [1] 2.96
```

1.3.3 Scripts

One of the great advantages of R over point and click analysis software is that you can save your work as scripts. You can edit and save these scripts using a text editor. We highly recommend working on an interactive *integrated development environment* (IDE) such as RStudio, which includes an editor with many R specific features, as well as a console to execute your code. [TODO: add arrows pointing at script and console]. RStudio is also free and below we include instructions on how to install it.

Note that most web-based R consoles, also provide a pane to edit scripts, but not all permit you to save the scripts for later use.

1.3.4 Installing RStudio

Instructions on to install RStudio are here and for Windows we have special instructions here.

Once you install RStudio you can simply start that program rather than R as it automatically starts R. But don't be confused, R is a different piece of software and you can't run RStudio without first installing R.

If you are going to follow along with RStudio as you read this book we recommend you first read [chapter section XX]. RStudio includes many useful feature other than providing a script editor and you want to know about them.

1.4 The R ecosystem

The functionality provided by a fresh install of R is only a small fraction of what is possible. In fact, we refer to what you get after your first install as *base R*. The extra functionality comes from add-ons available from developers. There are currently hundreds of these available from CRAN and many others shared via other repositories such as GitHub. However because not everybody needs all available functionality we instead make different components available via *packages*. R makes it very easy to install packages from within R. For example, to install the `dslabs` package which we use to share dataset and code related to this book you would type:

```
install.packages("dslabs")
```

In RStudio you can navigate to the *Tools* tab and select install packages. We can then load the package into our R sessions using the `library` function:

```
library(dslabs)
```

As you go through this book you will see that we load packages without installing them. This is because once you install a package, it remains installed and only needs to be loaded with `library`. If you try to load a package and get an error, it probably means you need to install it first.

1.5 The Very Basics

Before we get started with the motivating dataset we need to cover the very basics of R.

1.5.1 Objects

Suppose a high-school student asks us for help solving several quadratic equation of the form $ax^2+bx+c=0$. The quadratic equation gives us the solutions:

$$\frac{-b-\sqrt{b^2-4ac}}{2a} \text{ and } \frac{-b+\sqrt{b^2-4ac}}{2a}$$

which of course change depending on the values of a, b, and c. One advantage of programming languages is that we can define variables and wrtie expression with these variables, similar to how we do in math, but obtain a numeric solution. We will write out the expression a below, but if we are asked to solve $x^2+x-1=0$ then we define

```
a <- 1
b <- 1
c <- -1
```

which stores the values to use later. Note that we use `<-` to assign values to the variables.

Note: We can use `=` instead of `<-` but we recommend against using it to avoid confusion.

Try defining the three variables above on your console. Note that R does not print anything when we make this assignment. This is good. Had you made a mistake you would receive an error message.

To see the value stored in a variable we simply ask R to evaluate `a` and it shows the stored value:

```
a
#> [1] 1
```

A more explicit way to ask R to show us the value stored in `a` is using `print` like this:

```
print(a)
#> [1] 1
```

We use the term *object* to describe stuff that is stored in R. Variables are examples, but objects can also be more complicated entities such as functions, which are described below.

1.5.2 The Workspace

As we define object in the console, we are actually changing the *workspace*. You can see all the variables saved in your workspace by typing:

```
ls()
#> [1] "a" "b" "c"
```

In RStudio the *Environment* tab shows the values

[Screen shot?]

We should see `a`, `b`, and `c`. If you try to recover the value of a variable that is not in your workspace you receive an error. For example if you type `x` you will receive the following message: `Error: object 'x' not found`.

Now, since these values are saved in variables, to obtain a solution to our equation we use the quadratic formula to obtain the answer:

```
(-b + sqrt(b^2 - 4*a*c) ) / ( 2*a )
#> [1] 0.618
(-b - sqrt(b^2 - 4*a*c) ) / ( 2*a )
#> [1] -1.62
```

1.5.3 Functions

Once you define variables, a data analysis processes can usually be described as a series of *functions* applied to the data. R includes several predefined function and most of the analysis pipelines we construct make extensive use these.

Note that we already used the `install.packages`, `library`, and `ls` functions. And we used the function `sqrt` to solve the quadratic equation above. There are many more prebuild functions and even more can be added through packages. Note that these functions do not appear in the workspace because but they are available for immediate use.

In general, to evaluate a function we need to uses parentheses. If you type `ls` the function is not evaluated and instead R shows you the code the defines the function. If you type `ls()` the function is evaluated and, as seen above, we see objects in the workspace.

Unlike `ls`, most fucntions require one or more *arguments*. Here is an example of how we assign object to the argument of the function `log`:

```
log(8)
#> [1] 2.08
log(a) ## a is defined above
#> [1] 0
```

You can find out what the function expects and what it does by reviewing the very useful manuals included in R. You get help by using the `help` function like this:

```
help("log")
```

and for most functions we can use the shorthand:

```
?log
```

The help file will show you what arguments the function is expecting. For exmple `log` needs `x` and `base` to run. However, some arguments are required and others are optional. You can determine which arguments are options by noting, in the help document, that a default value is assigned with `=`. For example, the base of the function `log` defaults to `base = exp(1)` making it a natural log.

Note: if you want a quick look at the arguments without opening the help system you can type:

```
args(log)
#> function (x, base = exp(1))
#> NULL
```

You can change the default values by simply assigning another object:

```
log(8, base = 2)
#> [1] 3
```

Note that we have not being specificing the argument `x` as such:

```
log(x = 8, base = 2)
#> [1] 3
```

The above code work, but we can save ourselves some typing because, if no argument name is used, R assumes you are entering arguments in the order shown in the help file or by `args`. So by not using the names, it assumes the arguments are `x` followed by `base`:

```
log(8,2)
#> [1] 3
```

If using the arguments names, then we can include them in whatever order we want:

```
log(base = 2, x = 8)
#> [1] 3
```

Note that to specificy arguments we use `=`, we can't use `<-`.

There are some exceptions to the rule that functions need the parenthesis to be evaluated. Among these, the most commonly used are the arithmetic and relational operators. For example:

```
2 ^ 3
#> [1] 8
```

You can see the arithmentic operators by typing

You can get help using

```
help("+") ## or
?"-"
```

and the relational operators typing (note the quotes)

```
?">"
```

1.5.4 Other prebuilt objects

There are several datasets that are included for users to practice and test out functions. You can see all the available datasets by typing

```
data()
```

This shows you the object name for these datasets. These datasets are objects that can be used by simply typing the name, for example typing

```
co2
```

will show you Mauna Loa atmospheric CO2 concentration data.

There are also other mathematical objects such the constant π and ∞

```
pi
#> [1] 3.14
Inf+1
#> [1] Inf
```

1.5.5 Variable names

Here we used the letters a, b, and c as variable names, but variable names can be almost anything. Some basic rules in R is that they have to start with a letter, can't contain spaces and we avoid variables that are predefined in R, for example don't use `install.packages` as a variable name, don't type `install.packages <- 2`.

A nice convention to follow is to use meaningful words that describe what is stored, stick to lower case, and use underscores to replace spaces. For example, for the quadratic equations we could use something like this:

```
solution_1 <- (-b + sqrt(b^2 - 4*a*c) ) / ( 2*a )
solution_2 <- (-b - sqrt(b^2 - 4*a*c) ) / ( 2*a )
```

1.5.6 Saving your Workspace

Note that values remain in the workspace until you end your session or you erase them with the function `rm`. But workspaces can be saved for later use. In fact, when you quit R it asks you if you want to do this. If you do, the next time you start R it will restore the workspace. We actually recommend against saving the workspace this way because as you start working on different projects it will become harder to keep track of what is saved. Instead we recommend you assign the workspace a specific name. You can do this using the function `save` or `save.image`. To load you use the function `load`. When saving a workspace we recommend the suffix `rda` or `RData`. In RStudio by navigating to the *Session* tab and *Save Workspace as*. You can later load it using the *Load Workspace* options in the same tab. The pre-built functions `save`, `save.image` and `load`. You can read the help pages to learn more.

1.5.7 Scripts

To solve another equation such as $3x^2 + 2x - 1$ we can copy and paste the code above but this time redefine the variables and recompute the solution:

```
a <- 3
b <- 2
c <- -1
(-b + sqrt(b^2 - 4*a*c) ) / ( 2*a )
(-b - sqrt(b^2 - 4*a*c) ) / ( 2*a )
```

By creating and saving a script with the code above, we would not need to retype everything and simply change the variable names. Try writing the script above into a editor and note how much easier it is to change the variables and receive an answer.

1.5.8 Comments

If a line of R code start with the symbol `#` it is not evaluated. We can use this to write reminder of why we wrote the code we wrote. For example in the script above we could add

```
## Code to compute solution to quadratic equation of the form ax^2 + bx + c
## define the variables
a <- 3
b <- 2
c <- -1

## now compute the solution
(-b + sqrt(b^2 - 4*a*c) ) / ( 2*a )
(-b - sqrt(b^2 - 4*a*c) ) / ( 2*a )
```

1.6 Data types

Variables in R can be of different types. For example we need to distinguish numbers from character strings and tables from simple lists of numbers. The function `class` helps us determine what type of object we have:

```
a <- 2
class(a)
#> [1] "numeric"
```

To work efficiently in R it is important to learn the different types of variable and what we can do with these.

1.6.1 Data Frames

Up to now, the variables we have defined are just one number. This is not very useful for storing data. The most common way of storing a dataset in R is in a *data frame*. Conceptually, we can think of a data frame as table with rows representing observations and the different variables reported for each observatin defining the columns. Data frames are particularly useful for datasets because we can combine different data types into one object.

We stored the data for our motivating example in a data frame. You can access this dataset by loading the `dslabs` library and loading the `murders` dataset using the `data` function:

```
library(dslabs)
data(murders)
```

To see that this is in fact a data frame we type

```
class(murders)
#> [1] "data.frame"
```

1.6.2 Examining an object

The function `str` is useful to find out more about the structure of an object

```
str(murders)
#> 'data.frame':    51 obs. of  5 variables:
#>  $ state     : chr  "Alabama" "Alaska" "Arizona" "Arkansas" ...
#>  $ abb       : chr  "AL" "AK" "AZ" "AR" ...
#>  $ region    : Factor w/ 4 levels "Northeast","South",..: 2 4 4 2 4 4 1 2 2 2 ...
#>  $ population: num  4779736 710231 6392017 2915918 37253956 ...
#>  $ total     : num  135 19 232 93 1257 ...
```

This tells us much more about the object. We see that the table has 51 rows (50 states plus DC) and five variables. We can show the first six lines using the function `head`:

```
head(murders)
#>        state abb region population total
#> 1    Alabama  AL  South    4779736   135
#> 2     Alaska  AK   West     710231    19
#> 3    Arizona  AZ   West    6392017   232
#> 4   Arkansas  AR  South    2915918    93
#> 5 California  CA   West   37253956  1257
#> 6   Colorado  CO   West    5029196    65
```

In this data set each state is considered an obsrvation and five varialbles reported for each state.

Before we go any further in answering our original question about different states, let's get to know the components of this object better.

1.6.3 The accessor

For our analysis we will need to access the different variables, represented by columns, included in this data frame. To access these variables we use the accessor operator `$` in the following way:

```
murders$population
#>  [1]  4779736   710231  6392017  2915918 37253956  5029196  3574097
#>  [8]   897934   601723 19687653  9920000  1360301  1567582 12830632
#> [15]  6483802  3046355  2853118  4339367  4533372  1328361  5773552
#> [22]  6547629  9883640  5303925  2967297  5988927   989415  1826341
#> [29]  2700551  1316470  8791894  2059179 19378102  9535483   672591
#> [36] 11536504  3751351  3831074 12702379  1052567  4625364   814180
#> [43]  6346105 25145561  2763885   625741  8001024  6724540  1852994
#> [50]  5686986   563626
```

But how did we know to use `population`? Above, by applying the function `str` to the obejct `murders`, we revealed the names for each of the five variables stored in this table. We can quickly access the variables names using:

```
names(murders)
#> [1] "state"      "abb"        "region"     "population" "total"
```

It is important to know that the order of the entries in `murders$population` preserve the order of the rows in our data table. This will later permit us to manipulate one variable based on the results of another, for

example we will be able to order the state names by number of murders.

Tip: R comes with a very nice auto-complete functionality that saves us the trouble of typing out all the name. Try typing `murders$p` then hitting the *tab* key on your keyboard. RStudio has many useful auto-complete features options.

1.6.4 Vectors: numerics, characters, and logical

Note that the object `murders$population` is not one number but several. We call these types of objects *vectors*. A single number is technically a vector but in general we vectors refer to objects with several entries. The function `length` tells you how many entries in the vector:

```
pop <- murders$population
length(pop)
#> [1] 51
```

This particular vector is *numeric* since population sizes are numbers:

```
class(pop)
#> [1] "numeric"
```

In a numeric vector, every entry must be a number.

To store character strings, vectors can also be of class *character*. For example the state names are characters:

```
class(murders$state)
#> [1] "character"
```

As with numeric vectors, all entries in a character verctor need to be a character.

Another important type are *logical vectors*. These must be either `TRUE` or `FALSE`.

```
z <- 3 == 2
z
#> [1] FALSE
class(z)
#> [1] "logical"
```

Here the `==` is a relational operator asking if 3 is equals to 2. Remember that in R, if you just use one `=` you actually assign. You can see the other *relational operators* by typing

```
?Comparison
```

In future sections you will see how useful relational operators can be.

Advanced: Mathematically, the values in `pop` are integers and there is an integer class in R. However, by default, numbers are assigned class numeric even when they are round integers, for example `class(1)` returns numeric. You can turn them into class integer with `as.integer(1)` or add by adding an `L` like this: `1L`. Note the class by typing: `class(1L)`

1.6.5 Factors

In the `murders` dataset we might expect the region to also be a character vector. However it is not:

```
class(murders$region)
#> [1] "factor"
```

it is a *factor*. Factors are useful for storing categorical data. Notice that there are only 4 regions:

```
levels(murders$region)
#> [1] "Northeast"     "South"         "North Central" "West"
```

So, in the background, R stores these *levels* as integers and keeps a map to keep track of the labels. This is more memory efficient than storing all the characters. However, factors are also a source of confusion as they can easily be confused with characters but behave differently in different context. We will see more of this later.

In general, we recommend avoiding factors as much as possible although they are sometimes necessary to fit model containing categorical data.

1.6.6 Lists

Data frames are a special case of *lists*. We will covers lists in more detail later but know that they are useful because you can store any combination of other types, here is an example of list we created for you:

```
record
#> $name
#> [1] "John Doe"
#>
#> $student_id
#> [1] 1234
#>
#> $grades
#> [1] 95 82 91 97 93
#>
#> $final_grade
#> [1] "A"
class(record)
#> [1] "list"
```

We won't be using lists until later but you might encounter one in your own exploration of R. Note that, as with data frames, you can extract the components with the accessor `$`. In fact, data frames are a type of list.

```
record$student_id
#> [1] 1234
```

We can also use double brackets like this:

```
record[["student_id"]]
#> [1] 1234
```

You should get used to the fact that in R there are several ways to do the same thing, in particular accessing entries.

1.7 Vectors

The most basic unit available in R to store data are *vectors*. As we have seen complex datasets can usually be broken down into components that are vectors. For example in a data frame, each column is a vector. Here we learn more about this important class.

1.7.1 Creating vectors

We can create vectors using the function `c`, which stands for concatenate. We use `c` to *concatenate* entires in the following way:

```
codes <- c(380, 124, 818)
codes
#> [1] 380 124 818
```

We can also create character vectors. We use the quotes to denote that the entries are characters rather than variables names.

```
country <- c("italy","canada","egypt")
```

By now you should know that if you type

```
country <- c(italy, canada, egypt)
```

you recieve an error becuase the variables `italy`, `canada` and `egypt` are not defined: R looks for variables with those names and returns an error.

1.7.2 Names

Sometimes it is useful to name the entries of a vector. For example, when defining a vector of country codes we can use the names to connect the two:

```
codes <- c(italy = 380, canada = 124, egypt = 818)
codes
#>  italy canada  egypt
#>    380    124    818
```

The object `codes` continues to be a numeric vector:

```
class(codes)
#> [1] "numeric"
```

but with names

```
names(codes)
#> [1] "italy"  "canada" "egypt"
```

If the use of strings without quotes looks confusing, know that you can use the quotes as well

```
codes <- c("italy" = 380, "canada" = 124, "egypt" = 818)
codes
#>  italy canada  egypt
#>    380    124    818
```

There is no difference between this call and the previous one: one of the many ways R is quirky compared to other languages.

We can also assign names using the `names` functions

```
codes <- c(380, 124, 818)
country <- c("italy","canada","egypt")
names(codes) <- country
codes
#>  italy canada  egypt
#>    380    124    818
```

1.7.3 Sequences

Another useful function for creating vectors generates sequences

```
seq(1, 10)
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

The first argument defines the start, and the second the end. The default is to go up in increments of 1, but a third argument let's us tell it how much to jump by:

```
seq(1, 10, 2)
#> [1] 1 3 5 7 9
```

If we want consecutive integers we can use the following shorthand

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

Note that when we use these function, R produces integers, not numerics, because they are typically used to index something:

```
class(1:10)
#> [1] "integer"
```

However, note that as soon as we create something it's not an integer the class changes:

```
class(seq(1, 10))
#> [1] "integer"
class(seq(1, 10, 0.5))
#> [1] "numeric"
```

1.7.4 Subsetting

We use square brackets to access specific elements of a list. For the vector `codes` we defined above, we can access the seconde element using

```
codes[2]
#> canada
#>    124
```

You can get more than one entry by using a multi-entry vector as an index:

```
codes[c(1,3)]
#> italy egypt
#>   380   818
```

The sequences defined above are particularly useful if we want to access, say, the first two elements

```
codes[1:2]
#>  italy canada
#>    380    124
```

If the elements have names, we can also access the entries using these names. Here are two examples

```
codes["canada"]
#> canada
#>    124
codes[c("egypt","italy")]
#> egypt italy
#>   818   380
```

1.7.5 Coercion

In general, *coercion* is an attempt by R to be flexible with data types. When an entry does not match the expected, R tries to guess what we meant before throwing an errors. This can also lead to confusion. Failing to understand *coercion* can drive programmer crazy when attempting to code in R since it behavoes quite diffentlyfrom most other languages in this regard. Let's learn about it with a some examples.

We said that vectors must be all of the same type. So if we try to combine, say, numbers and characters you might expect an error

```
x <- c(1, "canada", 3)
```

But we don't get one, not even a warning! What happened? Look at `x` and its class:

```
x
#> [1] "1"      "canada" "3"
class(x)
#> [1] "character"
```

R *coerced* the data into characters. It guessed that because you put a character string in the vector you meant the 1 and 3 to actually be characters strings `"1"` and "`3`". The fact that not even a warning is issued is an example of how coercion can cause many unnoticed errors in R.

R also offers functions to force a specific coercion. For example you can turn numbers into characters with

```
x <- 1:5
y <- as.character(x)
y
#> [1] "1" "2" "3" "4" "5"
```

And you can turn it back with `as.numeric`.

```
as.numeric(y)
#> [1] 1 2 3 4 5
```

This function is actually quite useful as datasets that include numbers as character strings are common.

1.7.6 Not Availables (NA)

When these coercion function encounter an impossible case it gives us a warning and turns the entry into an special value called an `NA` for "not available". For example:

```
x <- c("1", "b", "3")
as.numeric(x)
#> Warning: NAs introduced by coercion
#> [1]  1 NA  3
```

R does not have any guesses for what number you want when you type `b` so it does not try.

Note that as a data scientist you will encounter the `NA` often as they are used for missing data, a common problem in real life dataset.

1.8 Sorting

Now that we have some basic R knowledge under our belt, let's try to gain some insights into the safety of differnet states in the conext of gun murders.

1.8.1 sort

We want to rank the states from least to most gun murders. The function `sort` sorts a vector in increasing order. So we can see that the largest number of gun murders by typing

```
library(dslabs)
data(murders)
sort(murders$total)
#>  [1]    2    4    5    5    7    8   11   12   12   16   19   21   22   27
#> [15]   32   36   38   53   63   65   67   84   93   93   97   97   99  111
#> [29]  116  118  120  135  142  207  219  232  246  250  286  293  310  321
#> [43]  351  364  376  413  457  517  669  805 1257
```

However, this does not give us information about which states have which murder totals. For example, we don't know which state had 1257 murders in 2010.

1.8.2 order

The function `order` is closer to what we want. It takes a vector and returns the vector of indexes that sort the input vector. This may sound confuisng so let's look at a simple example: we create a vector and sort it:

```
x <- c(31, 4, 15, 92, 65)
sort(x)
#> [1]  4 15 31 65 92
```

Rather than sort the vector, the function `order` gives us back the index that, if used to index the vector, will sort it:

```
index <- order(x)
x[index]
#> [1]  4 15 31 65 92
```

If we look at this index we see why it works:

```
x
#> [1] 31  4 15 92 65
order(x)
#> [1] 2 3 1 5 4
```

Note that the second and fourth entry of `x` are the smallest so `order(x)` starts with `2`. The next smallest is the third entry so the second entry is `3` and so on.

How does this help us order the states by murders? First remember that you the entries of vectors you access with `$` follow the same order as the rows in the table. So, for example, these two vectors, containing the state names and abbreviations respectively, are match by their order:

```
murders$state[1:10]
#>  [1] "Alabama"              "Alaska"               "Arizona"
#>  [4] "Arkansas"             "California"           "Colorado"
#>  [7] "Connecticut"          "Delaware"             "District of Columbia"
#> [10] "Florida"
murders$abb[1:10]
#>  [1] "AL" "AK" "AZ" "AR" "CA" "CO" "CT" "DE" "DC" "FL"
```

So this means we can now order the state names by their total murders by first obtaining the index that orders the vectors according to murder totals, and the indexing the state names or abbreviation vector:

```
ind <- order(murders$total)
murders$abb[ind]
```

```
#>  [1] "VT" "ND" "NH" "WY" "HI" "SD" "ME" "ID" "MT" "RI" "AK" "IA" "UT" "WV"
#> [15] "NE" "OR" "DE" "MN" "KS" "CO" "NM" "NV" "AR" "WA" "CT" "WI" "DC" "OK"
#> [29] "KY" "MA" "MS" "AL" "IN" "SC" "TN" "AZ" "NJ" "VA" "NC" "MD" "OH" "MO"
#> [43] "LA" "IL" "GA" "MI" "PA" "NY" "FL" "TX" "CA"
```

We see that California had the most murders.

1.8.3 `max` and `which.max`

If we are only interested in the entry with the largest value we can use `max` for the value

```
max(murders$total)
#> [1] 1257
```

and `which.max` for the index of the largest value

```
i_max <- which.max(murders$total)
murders$state[i_max]
#> [1] "California"
```

For the minimum we can use `min` and `which.min` in the same way.

So is California the most dangerous state? In a next section we argue that we should be considering rates not totals. Before doing that we introduce one last order related function: `rank`

1.8.4 `rank`

Although less useful than `order` and `sort`, the function `rank` is also related to order. For any given list it gives you a vector with the rank of the first entry, second entry, etc… of the vector. Here is a simple example

```
x <- c(31, 4, 15, 92, 65)
rank(x)
#> [1] 3 1 2 5 4
```

To summarize let's look at the results of the three functions we have introduced

original	sort	order	rank
31	4	2	3
4	15	3	1
15	31	1	2
92	65	5	5
65	92	4	4

1.9 Vector arithmetics

California had the most murders. But does this mean it is the most dangerous state. What if it just has many more people than any other state. We can very quickly confirm that, indeed, California has the largest population:

```
murders$state[which.max(murders$population)]
#> [1] "California"
```

with over 37 million inhabitants! It is thereforeunfair to compare the totals if we are interested in learning how safe the state is.

What we really should be computing is the murders per capita. The reports we describe in the motivating section used murders per 100,000 as the unit. To compute this quantiy, the powerful vector arithmetic capabilities of R come in handy.

1.9.1 Rescaling

In R, arithmetic operations on vectors occur *element wise*. For a quick example suppose we have height in inches

```
heights <- c(69, 62, 66, 70, 70, 73, 67, 73, 67, 70)
```

and want to covert to centimeters. Note what happens when we multiply `heights` by 2.54:

```
heights * 2.54
#>  [1] 175 157 168 178 178 185 170 185 170 178
```

it multiplied each element by 2.54. Similarly if we want to compute how many inches taller or shorter than the average, 69 inches, we can subtract it from every entry like this

```
heights - 69
#>  [1]  0 -7 -3  1  1  4 -2  4 -2  1
```

1.9.2 Two vectors

If we have two vectors of the same length, and we sum them in R, they get added entry by entry like this

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} a+e \\ b+f \\ c+g \\ d+h \end{pmatrix}$$

The same holds for other mathematical operation such as `-`, `*` and `/`.

This implies that to compute the murder rates we can simply type

```
murder_rate <- murders$total / murders$population * 100000
```

Once we do this, we notice that California is no longer near the top of the list. In fact, we can use what we have learned to order the states by murder rate:

```
murders$state[order(murder_rate)]
#>  [1] "Vermont"              "New Hampshire"        "Hawaii"
#>  [4] "North Dakota"         "Iowa"                 "Idaho"
#>  [7] "Utah"                 "Maine"                "Wyoming"
#> [10] "Oregon"               "South Dakota"         "Minnesota"
#> [13] "Montana"              "Colorado"             "Washington"
#> [16] "West Virginia"        "Rhode Island"         "Wisconsin"
#> [19] "Nebraska"             "Massachusetts"        "Indiana"
#> [22] "Kansas"               "New York"             "Kentucky"
#> [25] "Alaska"               "Ohio"                 "Connecticut"
#> [28] "New Jersey"           "Alabama"              "Illinois"
#> [31] "Oklahoma"             "North Carolina"       "Nevada"
#> [34] "Virginia"             "Arkansas"             "Texas"
#> [37] "New Mexico"           "California"           "Florida"
#> [40] "Tennessee"            "Pennsylvania"         "Arizona"
#> [43] "Georgia"              "Mississippi"          "Michigan"
```

```
#> [46] "Delaware"             "South Carolina"       "Maryland"
#> [49] "Missouri"             "Louisiana"            "District of Columbia"
```

1.10 Indexing

R provides a powerful and convinient way of indexing vectors. We can, for example, subset a vector based on properties of another vector. We continue our us murders example to demonstrate

1.10.1 Subsetting with logicals

We have now calculated the murder rate using

```
murder_rate <- murders$total / murders$population * 100000
```

Say you are moving from Italy where, according to the ABC news report, the murder rate is only 0.71 per 100,000. You would prefer to move to a state with a similar rate. Another powerful feature of R is that we can we can use logicals to index vectors. Note that if we compare a vector to a single number, it actually performs the test for each entry. Here is an example related to the question above

```
ind <- murder_rate < 0.71
```

Or if we want to know if its less or equal we can use

```
ind <- murder_rate <= 0.71
ind
#>  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
#> [12]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
#> [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
#> [34] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
#> [45] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

Note that we get back a logical vector with `TRUE` for each entry smaller or equal than 0.71. To see which states these are, we can leverage the fact that vectors can be indexed with logicals.

```
murders$state[ind]
#> [1] "Hawaii"        "Iowa"          "New Hampshire" "North Dakota"
#> [5] "Vermont"
```

Note that to count how many are TRUE, the function `sum` returns the sum of the entries of a vector and logical vectors get *coerced* to numeric with `TRUE` coded as 1 and `FALSE` as 0. Thus we can count the states using

```
sum(ind)
#> [1] 5
```

1.10.2 Logical Operators

Suppose we like the mountains and we want to move to a safe state in the West region of the country. We want the murder rate to be at most 1. So we want two different things to be true. Here we can use the logical operator *and* which in R is `&`. This operation results in a true, only when both logicals are true. To see this consider this example

```
TRUE & TRUE
#> [1] TRUE
TRUE & FALSE
#> [1] FALSE
```

```
FALSE & FALSE
#> [1] FALSE
```

We can form two logicals:

```
west <- murders$region=="West"
safe <- murders$rate<=1
```

and we can use the `&` to get a vector of logicals that tells us which states satisfy our condition

```
ind <- safe & west
murders$state[ind]
#> character(0)
```

1.10.3 `which`

Suppose we want to look up California's murder rate. For this type of operation, it is convenient to convert vectors of logicals into indexes instead of keeping long vectors of logicals. The function `which` tells us which entries of a logical vector are TRUE. So we can type:

```
ind <- which(murders$state == "California")
ind ##this is the index that matches the California entry
#> [1] 5
murder_rate[ind]
#> [1] 3.37
```

1.10.4 `match`

If instead of just one state we want to find out the murder rates for several, say New York, Florida, and Texas, we can use the function `match`. This function tells us which indexes of a second vector match each of the entries of a first vector:

```
ind <- match(c("New York", "Florida", "Texas"), murders$state)
ind
#> [1] 33 10 44
```

Now we can look at the murder rates:

```
murder_rate[ind]
#> [1] 2.67 3.40 3.20
```

1.10.5 `%in%`

If rather than an index we want a logical that tells us whether or not each element of a first vector is in a second we can use the function `%in%`. So, say you are not sure if Boston, Dakota and Washington are states, you can find out like this

```
c("Boston", "Dakota", "Washington") %in% murders$state
#> [1] FALSE FALSE  TRUE
```

Advanced: Note that there is a connection between `match` and `%in%` `throughwhich`: the following two lines of code are equivalent:

```
match(c("New York", "Florida", "Texas"), murders$state)
#> [1] 33 10 44
which(murders$state%in%c("New York", "Florida", "Texas"))
#> [1] 10 33 44
```

1.11 Basic Data Wrangling

Up to now we have been changing vectors by reordering them and subsetting them through indexing. But once we start more advanced analyses, we will want to prepare data tables for data analysis. We refer to the task as data wrangling. For this purpose we will introduce the `dplyr` package which provides intuitive functionality for working with tables. In Chapter [xx] we will cover the `dplyr` package in much more depth.

Once you install `dplyr` you can load it using

```
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
```

This package introduces functions that perform the most common operations in data warngling and uses names for these functions that are relatively easy to remember. For example to change the data table by adding a new column we use `mutate`, to filter the data table to subset of rows we use `filter` and to subset the data by selecting specific columns we use `select`. We can also perform a series of operation, for example select and then filter, by sending the results of one function to another using a what is called the *pipe operator*: `%>%`Some details are included below.

1.11.1 Adding a column with `mutate`

We want all the necessary information for our analysis to be included in the data table. So the first task is to add the murder rate to our data frame. The function mutate takes the data frame as a first argument and the name and values of the variable in the second using the convention `name = values`. So to add murder rate we use:

```
murders <- mutate(murders, rate = total / population * 100000)
```

Note that here we used `total` and `population` in the function, which are objects that are **not** defined in our workspace. What is happening is that `mutate` knows to look for these variables in the `murders` data frame. So the intuitive line of code above does exactly what we want. We can see the new column is added:

```
head(murders)
#>        state abb region population total rate
#> 1    Alabama  AL  South    4779736   135 2.82
#> 2     Alaska  AK   West     710231    19 2.68
#> 3    Arizona  AZ   West    6392017   232 3.63
#> 4   Arkansas  AR  South    2915918    93 3.19
#> 5 California  CA   West   37253956  1257 3.37
#> 6   Colorado  CO   West    5029196    65 1.29
```

Also note that we have over-written the original `murders` object. However, this does *not* change the object that is saved and we load with `data(murders)`. If we load the `murders` data again, the original will over-write our mutated version.

Note: If we reload the dataset from the `dslabs` package it will rewrite our new data frame with the original.

1.11.2 Subsetting with `filter`

Now suppose that we want to filter the data table to only show the entries for which the murder rate is lower than 0.71. To do this we use the `filter` function which takes the data table as argument and then the conditional statement as the next. Like mutate, we can use the data table variable names inside the function and it will know we mean the columns and not objects in the Workspace.

```
filter(murders, rate <= 0.71)
#>           state abb        region population total  rate
#> 1        Hawaii  HI          West    1360301     7 0.515
#> 2          Iowa  IA North Central    3046355    21 0.689
#> 3 New Hampshire  NH     Northeast    1316470     5 0.380
#> 4  North Dakota  ND North Central     672591     4 0.595
#> 5       Vermont  VT     Northeast     625741     2 0.320
```

1.11.3 Selecting columns with `select`

Although our data table only has six columns, some data tables include hundreds. If we want to view just a few we can use the `select` function. In the code below we select three columns, assign this to a new object and then filter the new object:

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
#>           state        region  rate
#> 1        Hawaii          West 0.515
#> 2          Iowa North Central 0.689
#> 3 New Hampshire     Northeast 0.380
#> 4  North Dakota North Central 0.595
#> 5       Vermont     Northeast 0.320
```

Note that in the call to `select`, the first argument, `murders`, is an object but `state`, `region`, and `rate` are variable names.

1.11.4 The pipe: `%>%`

In the code above we want to show the three variables for states that have murder rates below 0.71. To do this we defined an intermediate object. In `dplyr` we can write code that looks more like our description of what we want to:

original data → select → filter

For such operation, we can use the pipe `%>%`. The code looks like this:

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
#>           state        region  rate
#> 1        Hawaii          West 0.515
#> 2          Iowa North Central 0.689
#> 3 New Hampshire     Northeast 0.380
#> 4  North Dakota North Central 0.595
#> 5       Vermont     Northeast 0.320
```

This line of code is equivalent to the two lines of code above. Note that when using the pipe we no longer need to specify the required argument as the `dplyr` functions assume that whatever is being *piped* is what should be operated on.

1.11.5 Creating a data frame

It is sometimes useful for us to create our own data frames. You can do this using the `data.frame` function:

```
grades <- data.frame(names = c("John", "Juan", "Jean", "Yao"),
                     exam_1 = c(95, 80, 90, 85),
                     exam_2 = c(90, 85, 85, 90))
grades
#>   names exam_1 exam_2
#> 1  John     95     90
#> 2  Juan     80     85
#> 3  Jean     90     85
#> 4   Yao     85     90
```

Warning: By default the function `data.frame` turns characters into factors:

```
class(grades$names)
#> [1] "factor"
```

To avoid this we use the rather cumbersome argument `stringsAsFactors`:

```
grades <- data.frame(names = c("John", "Juan", "Jean", "Yao"),
                     exam_1 = c(95, 80, 90, 85),
                     exam_2 = c(90, 85, 85, 90),
                     stringsAsFactors = FALSE)
class(grades$names)
#> [1] "character"
```

Now that we understand the basics of data frames we are ready to continue our quest to characterize each state as it relates to gun murders. A naive approach to determining the most dangerous state would be to

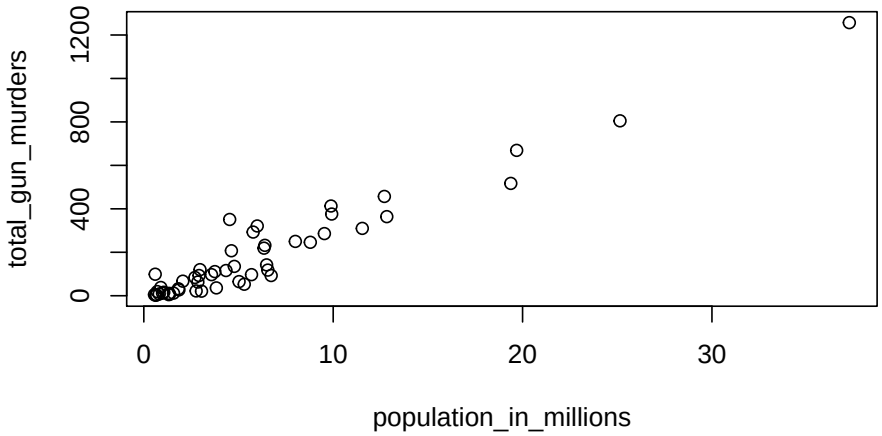
1.12 Basic plots

Exploratory data visualization is perhaps the strength of R. One can quickly go from idea to data to plot with a unique balance of flexibility and ease. For example, Excel may be easier than R but it is no where near as flexible. D3 may be more flexible and powerful than R, but it takes much longer to generate a plot.The next chapter dedicated to this topic, but here we introduce some very basic plotting functions.

1.12.1 Scatter plots

Earlier we inferred that states with larger population are likely to have more murders. This can confirmed with an exploratory visualization that plots these two quantities against each other:

```
population_in_millions <- murders$population/10^6
total_gun_murders <- murders$total
plot(population_in_millions, total_gun_murders)
```



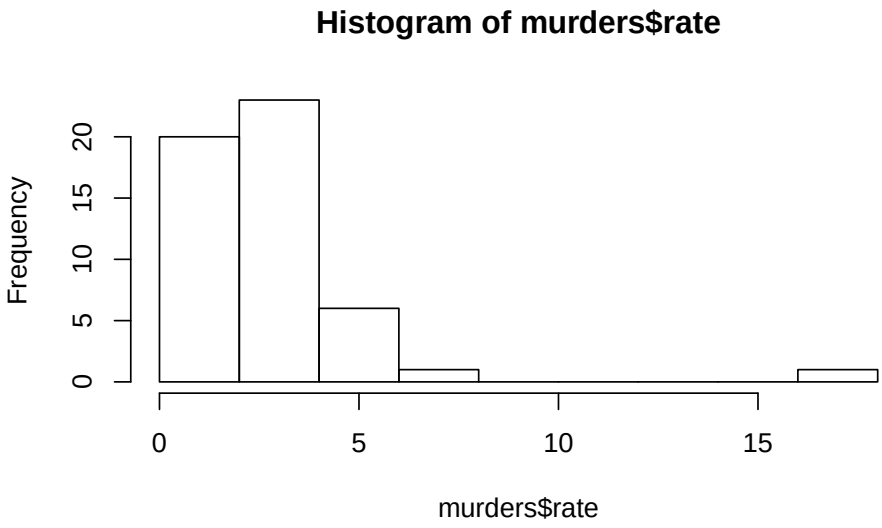
We can clearly see a relationship. **Advanced**: For a quick plot that avoids accessing variables twice, we can use the `with` function

```
with(murders, plot(population, total))
```

1.12.2 Histograms

We will describe histograms as they related to distribution in the next chapter. Here we will simply note that histograms are a powerful graphical summary of a list of numbers that gives you a general overview of the types of values you have. We can make a histograms of our murder rates by simply typing

```
hist(murders$rate)
```



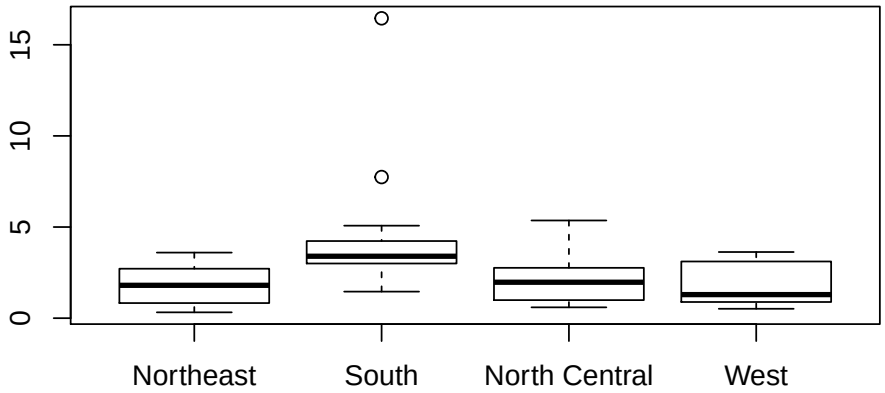
We can see that there is a wide range of values with most of them between 2 and 3 and one very extreme case with a murder rate of more than 15:

```
murders$state[which.max(murders$rate)]
#> [1] "District of Columbia"
```

1.12.3 Boxplot

Boxplots will be described in more detail in the next chapter as well. But here we say that they provide a more terse summary than the histogram but they are easier to stack with other boxplots. Here we can use them to compare the different regions

```
boxplot(rate~region, data = murders)
```



We can see that the South has larger murder rates than the other three regions.

1.13 Importing Data

In this chapter we used a data set already stored in an R object. A data scientist will rarely have such luck and will have to import data into R from either a file, a database, or other source. We cover this in more detail in Chapter XX. But because it is so common to read data from a file, we will briefly describe the key approach and function, in case you want to use your new knowledge on one of your own data sets.

Small datasets such as the one used in this chapter are typically commonly stored as Excel files. Although there are R packages designed to read Excel (xls) format, you generally want to avoid this format and save files as comma delimited (Comma-Separated Value/CSV) or tab delimited (Tab-Separated Value/TSV/TXT) files. These plain-text formats make it easier to share data, since commercial software is not required for working with the data.

1.13.0.1 Paths and the Working Directory

The first step is to find the file containing your data and know its *path*. When you are working in R it is useful to know your *working directory*. This is the folder in which R will save or look for files by default. You can see your working directory by typing:

```
getwd()
```

You can also change your working directory using the function `setwd`. Or you can change it through RStudio by clicking on "Session".

The functions that read and write files (there are several in R) assume you mean to look for files or write files in the working directory. Our recommended approach for beginners will have you reading and writing to the working directory. However, you can also type the full path, which will work independently of the working directory.

We have included the us murders data in CSV file as part of the `dslabs` package. We recommend placing your data in your working directory.

Because knowing where packages store files is rather advanced, we proved the following code that finds the directory and copies the file

```
dir <- system.file(package="dslabs") #extracts the location of package
filename <- file.path(dir,"extdata/murders.csv")
file.copy(filename, "murders.csv")
#> [1] FALSE
```

You should be able to see the file in your working directory and can check using

```
list.files()
#>  [1] "_book"                    "_bookdown.yml"
#>  [3] "_build.sh"                "_common.R"
#>  [5] "_deploy.sh"               "_output.yml"
#>  [7] "book_cache"               "book_files"
#>  [9] "book.bib"                 "book.Rmd"
#> [11] "data-science-book.Rproj" "dataviz"
#> [13] "DESCRIPTION"              "draft_exercises"
#> [15] "exercises"                "index.Rmd"
#> [17] "inference"                "intro"
#> [19] "LICENSE"                  "man"
#> [21] "murders.csv"              "NAMESPACE"
#> [23] "old-old-scpt.pl"          "old-scpt.pl"
#> [25] "packages.bib"             "preamble.tex"
#> [27] "prob"                     "R"
#> [29] "README.md"                "regression"
#> [31] "scpt.pl"                  "style.css"
#> [33] "toc.css"                  "wrangling"
```

1.13.1 `read.csv`

We are ready to read in the file. There several functions for reading in tables. Here we introduce one included in base R:

```
dat <- read.csv("murders.csv")
head(dat)
#>        state abb region population total
#> 1    Alabama  AL  South    4779736   135
#> 2     Alaska  AK   West     710231    19
#> 3    Arizona  AZ   West    6392017   232
#> 4   Arkansas  AR  South    2915918    93
#> 5 California  CA   West   37253956  1257
#> 6   Colorado  CO   West    5029196    65
```

We can see that we have read in the file.

Warning: `read.csv` automatically converts characters to factors. Note for example that:

```
class(dat$state)
#> [1] "factor"
```

You can avoid this using

```
dat <- read.csv("murders.csv", stringsAsFactors = FALSE)
class(dat$state)
#> [1] "character"
```

With this call the region variable is no longer a factor but we can easily change this with:

```
dat <- mutate(dat, region =  as.factor(region))
```

1.14 Programming basics

We teach R because it greatly facilitates data analysis, the main topic of this book. Coding in R we can efficiently perform exploratory data analysis, build data analysis pipelines and prepare data visualization

to communicate results. However R is not just a data analysis environment but a programming language. Advanced R programmers can develop complex packages and even improve R itself, but we do not cover advanced progamming in this book. However,in this section we introduce three key programming concepts: conditional expressions, for-loops and functions. These are not just key building blocks for advanced programming, but ocassionaly come in handy during data analysis. We also provide a list of power function that we dot not cover in the book but are worth knowing about as they are powerful tools commonly by expert data analysists.

1.14.1 Conditionals expressions

Conditionals expressions are one of the basic features of programming. The most common conditional expression is the if-else statement. In R, we can actually perform quite a bit of data analysis without conditionals. However, they do come up occasionally and if once you start writing your own functions and packages you will definitely need them.

Here is a very simple example showing the general structure of an if-else statement. The basic idea is to print the reciprocal of `a` unless `a` is 0:

```
a <- 0

if(a!=0){
  print(1/a)
} else{
  print("No reciprocal for 0.")
}
#> [1] "No reciprocal for 0."
```

Let's look at one more example using the US murders data frame.

Here is a very simple example that tells us which states, if any, have a murder rate lower than 0.5 per 100,000. The if statement protects us from the case in which no state satisfies the condition.

```
ind <- which.min(murder_rate)

if(murder_rate[ind] < 0.5){
  print(murders$state[ind])
} else{
  print("No state has murder rate that low")
}
#> [1] "Vermont"
```

If we try it again with a rate of 0.25 we get a different answer:

```
if(murder_rate[ind] < 0.25){
  print(murders$state[ind])
} else{
  print("No state has a murder rate that low.")
}
#> [1] "No state has a murder rate that low."
```

A related function that is very useful is `ifelse`. This function takes three arguments: a logical and two possible answers. If the logical is `TRUE` the first answer is returned and if `FALSE` the second. Here is an example

```
a <- 0
ifelse(a > 0, 1/a, NA)
#> [1] NA
```

The function is particularly useful because it works on vectors. It examines each element of the logical vector and returns corresponding answers from the accordingly.

```
a <- c(0,1,2,-4,5)
result <- ifelse(a > 0, 1/a, NA)
```

This table helps us see what happned:

a	is_a_positive	answer1	answer2	result
0	FALSE	Inf	NA	NA
1	TRUE	1.00	NA	1.0
2	TRUE	0.50	NA	0.5
-4	FALSE	-0.25	NA	NA
5	TRUE	0.20	NA	0.2

Here is an example of how this function can be readily used to replace all the missing values in a vector with zeros:

```
data(na_example)
no_nas <- ifelse(is.na(na_example), 0, na_example)
sum(is.na(no_nas))
#> [1] 0
```

Two other useful function are `any` and `all`. The `any` function takes a vector of logical and returns `TRUE` if any of the entries is `TRUE`.The `all` function takes a vector of logical and returns `TRUE` if all of the entries is `TRUE`. Here is an example.

```
z <- c(TRUE, TRUE, FALSE)
any(z)
#> [1] TRUE
all(z)
#> [1] FALSE
```

1.14.2 Defining Functions

As you become more experienced you will find yourself needeing to perform the same operations over and over. A simple example is computing average. We can compute the average of a vector `x` using the `sum` and `length` functions: `sum(x)/length(x)`. But because we do this so often it is much more efficient to write a function that performs this operation and thus someone already wrote the `mean` function. However, you will encounter situations in which the function does not already exist so R permits you to write your own. A simple version of function that computes the average can be defined like this

```
avg <- function(x){
  s <- sum(x)
  n <- length(x)
  s/n
}
```

Now `avg` is a function that computes the mean:

```
x <- 1:100
identical(mean(x), avg(x))
#> [1] TRUE
```

Note that variables defined inside a function are not saved in the workspace. So while we use `s` and `n` when we call `avg`, there values are created and changed only during the call. Here are illustrative example:

```
s <- 3
avg(1:10)
```

```
#> [1] 5.5
s
#> [1] 3
```

Note how `s` is still 3 after we call `avg`.

In general, functions are objects, so we assign them to variable names with `<-`. The function `function` tells R you are about to define a function. The general form of a function definition looks like this

[INSERT DIAGRAM]

Also not that The functions you define can have multiple arguments as well as default values. For example we can define a function that computes either the arithmetic or geometric average depending on a user defined variable like this

```
avg <- function(x, arithmetic = TRUE){
  n <- length(x)
  ifelse(arithmetic, sum(x)/n, prod(x)^(1/n))
}
```

We will learn more about how to create functions through experience as we face more complex tasks.

1.14.3 For loops

The formula for the sum $1 + 2 + \cdots + n$ is $n(n+1)/2$. What if we weren't sure that was the right function, how could we check? Using what we learned about functions we can create on that computes the S_n:

```
compute_s_n <- function(n){
  x <- 1:n
  sum(x)
}
```

Now if we can compute S_n for various vales of n, say $n = 1, \ldots, 25$ how do we do it? Do we write 25 lines of code calling `compute_s_n`? No, that is what for loops are for in programming. Note that we are performing exactly the same task over and over and that the only thing that is changing is the value of n. For loops let us define the range that our variable takes (in our example $n = 1, \ldots, 10$), then change the value as you *loop* and evaluate expression as you loop. The general form looks like

[INSERT CARTOON]

Perhaphs the simplest example of a for loop is this useless piece of code:

```
for(i in 1:5){
  print(i)
}
#> [1] 1
#> [1] 2
#> [1] 3
#> [1] 4
#> [1] 5
```

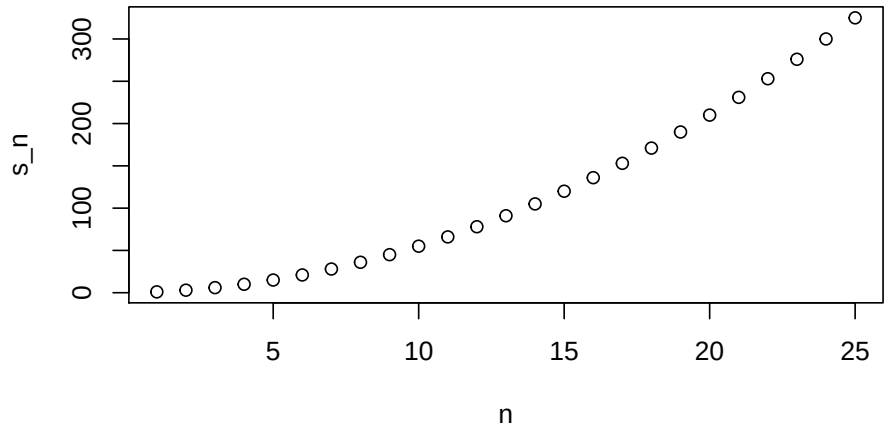
And here is the for loop we would write for our S_n example:

```
m <- 25
s_n <- vector(length = m) # create an empty vector
for(n in 1:m){
  s_n[n] <- compute_s_n(n)
}
```

In each iteration $n = 1$, $n = 2$, etc..., we compute S_n and store it in the nth entry of `s_n`.

Now we can create a plot to get search for a pattern

```
n <- 1:m
plot(n, s_n)
```

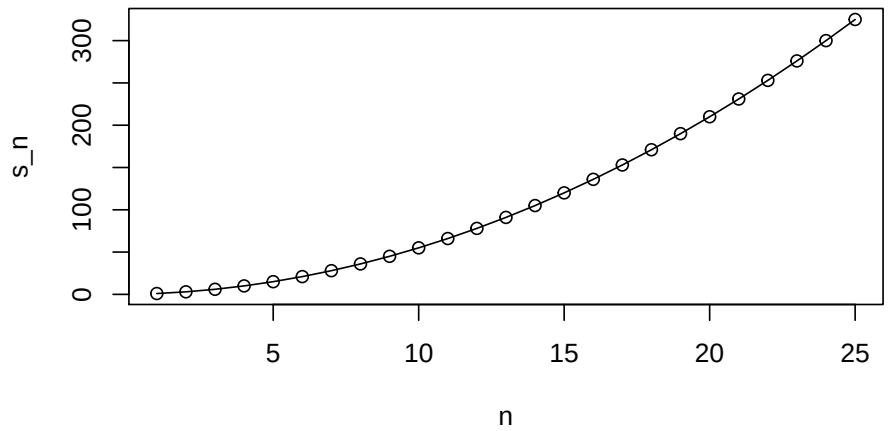


If you noticed that it appears to be a quadratic, you are on the right track because the formula is $n(n+1)/2$ which we can confirm with a table

```
#>   s_n formula
#> 1   1       1
#> 2   3       3
#> 3   6       6
#> 4  10      10
#> 5  15      15
#> 6  21      21
```

We can also overlay the two results by using the function `lines` to draw a line over the previously plotted points:

```
plot(n, s_n)
lines(n, n*(n+1)/2)
```



1.14.4 Other functions

It turns out that we rarely use for loops in R. This is because there are usually more powerful ways to perform the same task. Functions that are typically used instead of for loops are the apply family: `apply`, `sapply`, `tapply`, and `mapply`. We do not cover these functions in this book but they are worth learning if you intend to go beyond this introduction. Other functions that are widely used are `split`, `cut`, and `Reduce`.

Contents

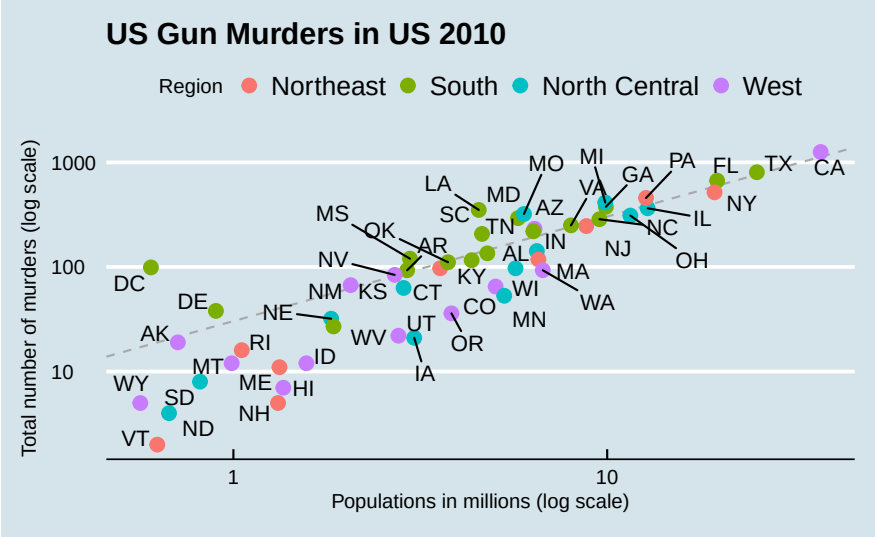
1 Data Visualization and Exploratory Data Analysis

1.1 Introduction

Looking at the numbers and character strings that define a dataset is rarely useful. To convince yourself, print and stare at the US murders data table:

```
library(tidyverse)
library(dslabs)
data(murders)
head(murders)
#>        state abb region population total
#> 1    Alabama  AL  South    4779736   135
#> 2     Alaska  AK   West     710231    19
#> 3    Arizona  AZ   West    6392017   232
#> 4   Arkansas  AR  South    2915918    93
#> 5 California  CA   West   37253956  1257
#> 6   Colorado  CO   West    5029196    65
```

What do you learn from staring at this table? How quickly can you determine which states have the largest populations? Which states have the smallest? How large is a typical state? Is there a relationship between population size and total murders? How do murder rates vary across regions of the country? For most human brains it is quite difficult to extact this infromation just from looking at the numbers. In contrast, the answer to all the questions above are readily avaialble from examining this plot



We are reminded of the saying "a picture is worth a thousand words". Data visualization provides a powerful way to communicate a data-driven finding. In some cases, the visualization is so convincing that no follow-up analysis is required.

The growing availability of informative datasets and software tools has led to increased reliance on data visualizations across many industries, academia, and government. An visible example are news organizations that are increasingly embracing *data journalism* and including effective *infographics* as part of their reporting.

A particularly effective example is a Wall Street Journal published an article showing data related to the impact of vaccines on battling infectious diseases. One of the graphs shows measles cases by US state through the years with a vertical line demonstrating when the vaccine was introduced.

Another striking examples comes from the New York Times showing data on
scores from the NYC Regents Exams. These scores are collected for several reasons including to determine if a student graduates from high school. In New York City you need a 65 to pass. The distribution of the test scores forces us to notice something somewhat problematic:

The most common test score is the minimum passing grade, with vew few just below. This unexpected result is consistent with students close to passing having their scores bumped up.

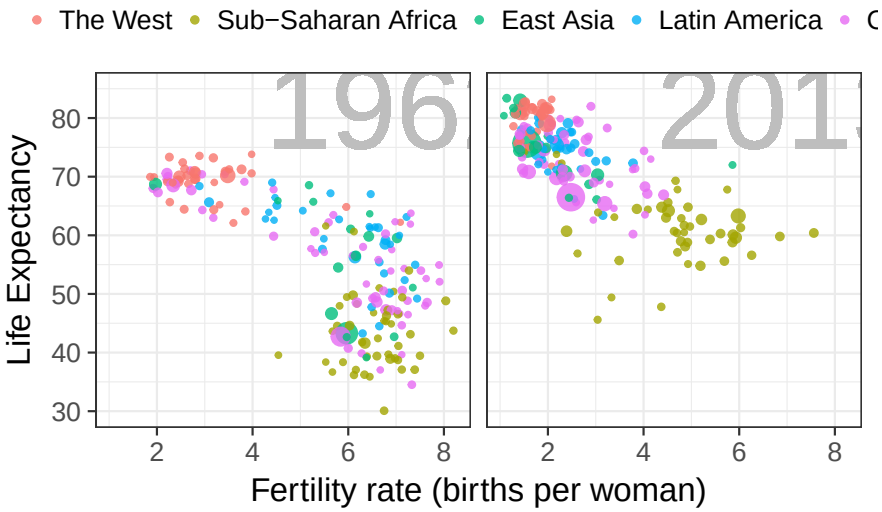
This is an example of how data visualization can lead to discoveries which would otherwise be missed if we simply subject the data to a battery of data analsysis tools or proceedures. Data visualization is the strongest tool of what we call exploratory data analysis (EDA). John W. Tukey (pictured), consider the father of EDA, once said that

> "The greatest value of a picture is when it forces us to notice what we never expected to see."

```
knitr::include_graphics("http://upload.wikimedia.org/wikipedia/en/e/e9/John_Tukey.jpg")
```

We note that many wildely used data analysis tools were initiated by discoveries made via EDA. EDA is perhaps the most important part of data analysis yet often overlooked.

Data visualization is also now pervasive in philantropic and educational organizations. In the talks New Insights on Poverty and The Best Stats You've Ever Seen, Hans Roslings forced us to to notice the unexpected with a series of plots related to world health and economics. In his videos, he used animated grpahs to show us how the world was changing and old narratives are no longer true.



It is also important to note that mistakes, biases, systematic errors and other unexpected problems often lead to data that should be handled with care. Failure to discover these problems often leads to flawed analyses and false discoveries. As an example, consider that measurement devices sometimes fail and that most data analysis procedures are not designed to detect these. Yet, these data analysis procedures will still give you an answer. The fact that it can be hard or impossible to notice an error just from the reported results, makes data visualization particularly important.

Data visualization is a powerful approach to detecting these problems. We refer to this particular task as *exploratory data analysis* (EDA).

In this chapter we will learn the basics of data visualization and exploratory data analysis. We will use three motivating examples. We will use the ggplot2 package to code. To learn the very basics, we will start with a somewhat artificial example: heights reported by students. Then we will cover two the examples mentioned above 1) world health and economics and 2) infectious disease trends in the United States.

Note that there is much more to data visualization than what we cover here. More

- ER Tufte (1983) The visual display of quantitative information. Graphics Press.
- ER Tufte (1990) Envisioning information. Graphics Press.
- ER Tufte (1997) Visual explanations. Graphics Press.
- A Gelman, C Pasarica, R Dodhia (2002) Let's practice what we preach: Turning tables into graphs. The American Statistician 56:121-130
- NB Robbins (2004) Creating more effective graphs. Wiley

We also do not cover interactive graphics, a topic that is a too advanced for this book. Some useful resources for those interested in learning more

- https://shiny.rstudio.com/
- https://d3js.org/

1.2 Distributions

You may have noticed that numerical data is often summarized with the *average* value. For example, the quality of a high school is sometimes summarized with one number: the average score in a standardized test. Occasionally a second number is reported: the *standard deviation*. So, for example, you might read a report stating that scores were 680 plus or minus 50 (the standard deviation). Note that the report has summarized an entire vector of scores with with just two numbers. Is this appropriate? Is there any important piece of information we are missing by only looking at this summary rather than the entire list?

Our first data visualization building block is learning to summarize lists of factors or numeric vectors. The most basic statistical summary of a list of objects or numbers is it's distribution. Once a vector has been

summarized as distribution, there are several data visualization techniques to effectively relay this information.

1.2.1 Data types

We will be working with two type of variables: categorical and numeric. Each can be divided into two other groups: categorical can be ordinal or not numerical variables can be discrete or continuous.

Variables that is defined by a small number of groups as *categorical data*. Two simple examples are sex (male or female), regions (Northeast, South, North Central, West). Some categorical data can be ordered, for example spiciness (mild, medium, hot), even if they are not numbers per se. In statistics text books they sometimes refer to these as *ordinal* data.

Example of numerical data are population sizes, murder rates, and heights. Note that some numerical data can be treated as ordered categorical. We can further divide numerical data into continuous and discrete. Continuous variables are those that can take any value such heights, if measured with enough precision. For example a pair of twins may be 68.12 and 68.11 inches respectively. Counts, such as population sizes, are discrete because they have to be round numbers.

Note that discrete numeric data can be considered ordinal. Although this is technically true, we usually reserve the term ordinal data for variables belonging to a small number of different groups, with each group having many members. In contrast, when we have many groups with few cases in each group we typically refer as discrete numerical variable. So, for example, the number of packs of cigarettes,a person smokes a day rounded to the closes pack, would be considered original while the actual number of cigarettes would be considered a numerical variable. But indeed, there are examples that can be considered both when it comes to visualizing data.

1.2.2 Motivating problem

Here we introduce a new motivating problem. It is an artificial one, but it will help us illustrate the concepts needed to understand distributions.

Pretend that we have to describe the heights of our classmates to ET, an extraterrestrial that has never seen humans. As a first step we need to collect data. To do this we ask students to report their heights in inches. We ask them to provide sex information because we know there are two different distributions. We collect the data and save it in a data frame:

```
library(dslabs)
data(heights)
head(heights)
#>      sex height
#> 1   Male     75
#> 2   Male     70
#> 3   Male     68
#> 4   Male     74
#> 5   Male     61
#> 6 Female     65
```

One way to convey the heights to ET is to simply send him this list of `nrow(heights)` heights. But there are much more effective ways to convey this information and understanding the concept of a distribution will help. To simplify the explanation, at first we focus on male heights.

1.2.3 Distribution Function

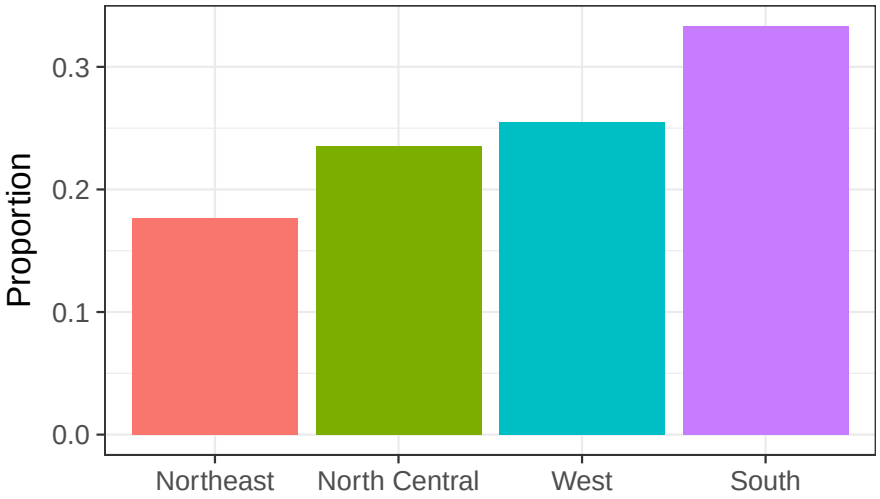
It turns that, in some cases, these two numbers are pretty much all we need to understand the data. We will learn data visualization techniques that will help us determine when this two number summary is appropriate.

These same techniques will serve as an alternative for when two numbers is not enough.

The most basic statistical summary of a list of objects or numbers is it's distribution. The simplest way to think of a distribution is as a compact description of a list with many entries. This concept should not be new for most of you. For example, with categorical data, the distribution simply describes the proportion of each unique category. For example the sex represented in the heights dataset is:

```
#>
#> Female   Male
#>  0.234  0.766
```

This two category *frequency table* is the simplest form of a distribution we can form. We don't really need to visualize it since one number describes everything we need to know **23**% are females and the rest are males. When there more categories then a simple barplot describes the distribution. Here is an example with the US state regions:



This particular plot is simply showing us four numbers: one for each category. We usually use barplots to display a few numbers. Although, this particular plot, a graphical representation of a frequency table, does not provide much more insight than a table itself, it is a first a example of how we convert a vector into a plot that succinctly summarizes all the information in the vector. Once the data is numerical, the task of displaying distributions is more challenging.

1.2.4 Cumulative Distribution Functions

Numerical data, that are not categorical, also have distributions. In general, when data is not categorical, reporting the frequency of each entry is not an effective summary since most entries are unique. For example, while several students reported a height of 68 inches, only one student reported a height of `68.503937007874` inches and only one student reported a height `68.8976377952756` inches. We assume that they converted from 174 and 175 centimeters respectively.

Statistics text books teach us that a more useful way to define a distribution for numeric data is to define a function that reports the proportion of the data below a for all possible values of a. This function is called the cumulative distribution function (CDF). In Statistics the following notation is used:

$$F(a) = \Pr(x \leq a)$$

Here is a plot of F for the height data:

Like the frequency table does for categorical data, the CDF defines the distribution for numerical data. From the plot we can see, for example, that 34% of the values are below 65, since $F(66) = 0.138$, or that 90% of

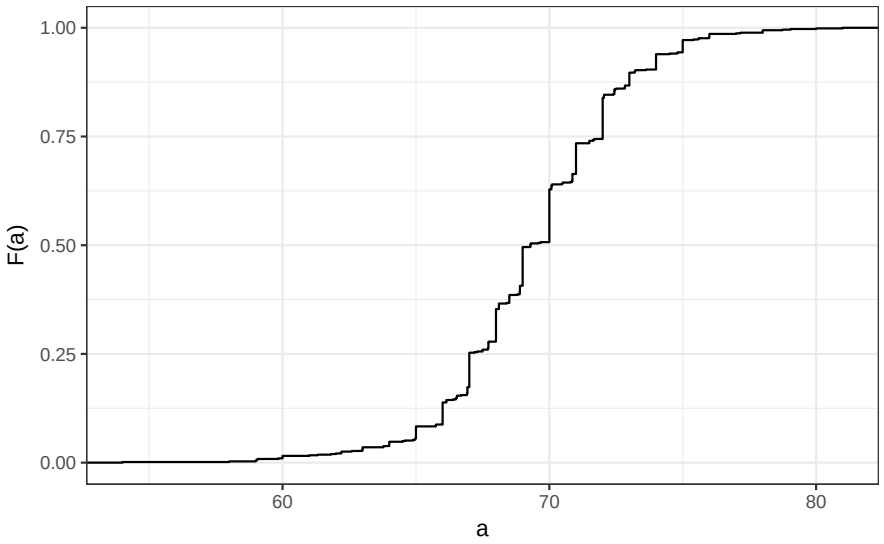


Figure 1: Empirical cummulative distribution function for male height.

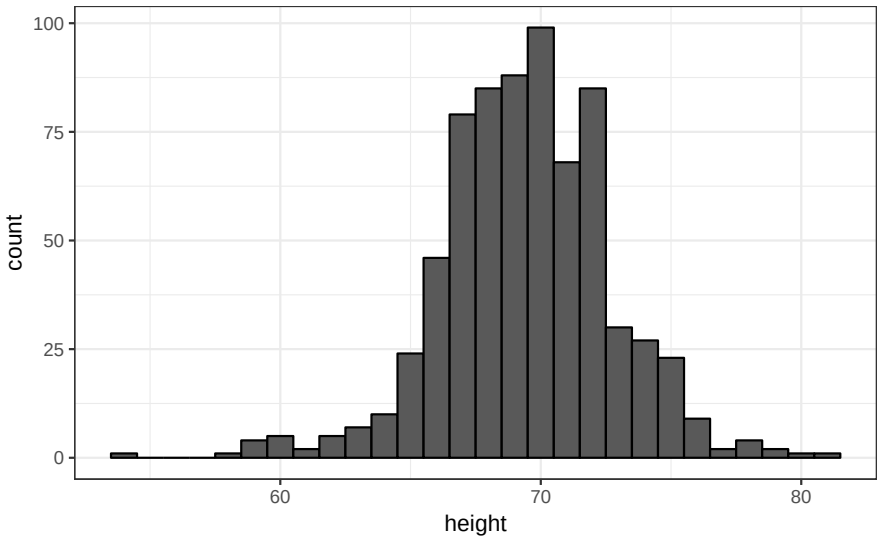
the values are below 72, since $F(72) = 0.839$, etc… In fact, we can report the proportion of values between any two heights, say a and b by computing $F(b) - F(a)$. This means that if we send this plot above to ET, he will have all the information needed to reconstruct the entire list. Paraphrasing the expression "a picture is worth a thousands word", in this case a picture is as informative as 708 numbers.

A final note: because CDFs can be defined mathematically, as opposed to using data as we do here, the word *empirical* is added to distinguish and we use the term empirical CDF (ECDF) instead.

1.2.5 Histograms

Although the CDF concept is widely discussed in statistics textbooks, the plot is actually not very popular in practice. The main reason is that it does not easily convey characteristics of interest such as: at what value is the distribution centered? Is the distribution symmetric? What ranges contain 95% of the values? Histograms are much preferred because it greatly facilitates answering such questions. Histograms sacrifice just a bit of information to produce plots that are much easier to interpret.

The simplest way to make a histograms is to divide the span of our data into non-overlapping bins of the same size. Then for each bin we count the number of values that fall in that interval. The histogram plots these counts as bars with the base of the bar the interval. Here is the histogram for the height data splitting the range of values into one inch intervals $[58.5, 59.5], (55.5, 56.5], ..., (80.5, 81.5]$



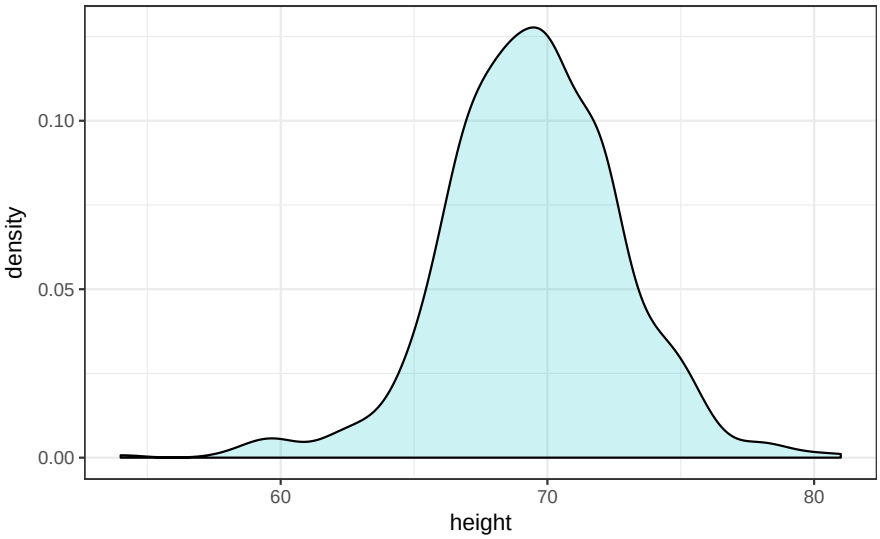
Note that a histogram is similar to a barplot but it differs in that the x-axis is numerical, not categorical.

If we send this plot to ET, he will immediately learn some important properties about our data. First, the range of the data is from 55 to 81 with the majority (more than 95%) between 63 and 75 inches. Second, the heights are close to symmetric around 69 inches. Also note that by adding up counts, ET could obtain a very good approximation of the proportion of the data in any interval. Therefore, this histogram above is not only easy to interpret but provides almost all the information contained in the raw list of 708 heights with just 23 bin counts.

So what information do we lose? Note that all values in each interval are treated the same when computing bin heights. So, for a example, the histogram does not distinguish between 64 64.1 64.2 inches. Given that these differences are almost unnoticeable to the eye, the practical implications are negligible and we were able to summarize the data to just 23 numbers.

1.3 Smoothed Density

Smooth density plots are aesthetically more appealing than histograms. Here is what a smooth density plot looks like for our heights data:

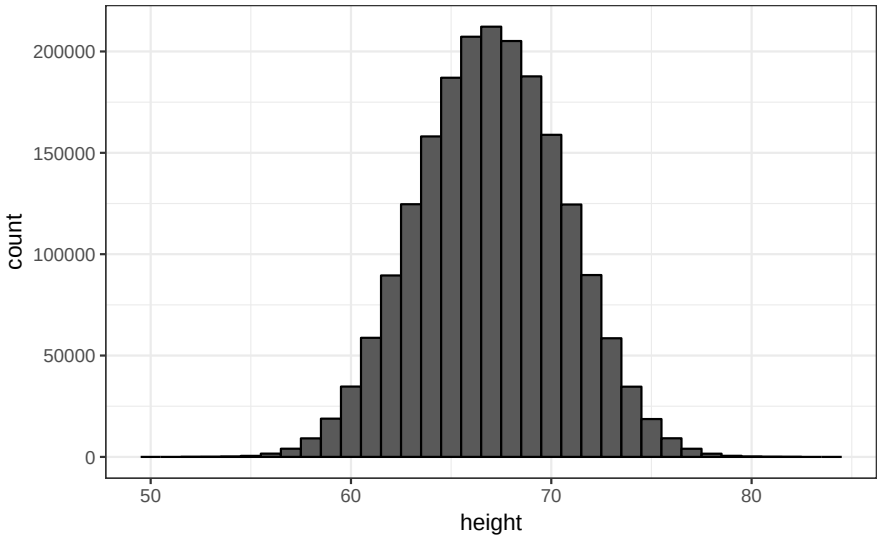


Note that we no longer have sharp edges at the interval boundaries and that many of the local peaks have been removed. Also, notice that the scale of the y-axis changed from counts to *density*.

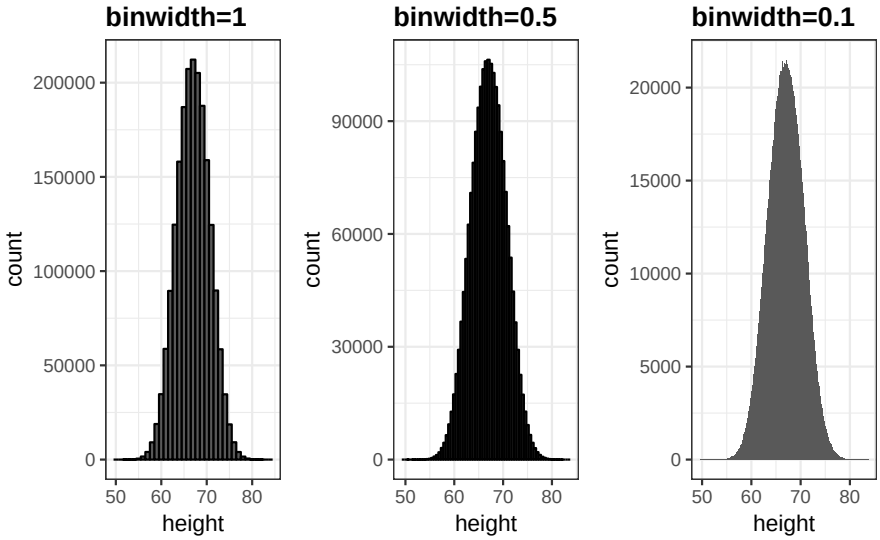
To understand the smooth densities we have to understands *estimates* a topic we don't cover until a later chapter. However, we provide a heuristic explanation to help you understand the basics and you can use this useful data visualization tool.

The main new concept you have to understand is that we assume that our list of observed values comes from a much much larger list of unobserved values. So in the case of heights, you can imagine our list of 924 students comes from a hypothetical list containing all the heights of all the students in all the world measured very precisely. Let's say there are 1,000,000 of these. This list values, like any list of values, has a distribution and this is really what we want to report to ET since it is much general. Unfortunately we don't get to see it.

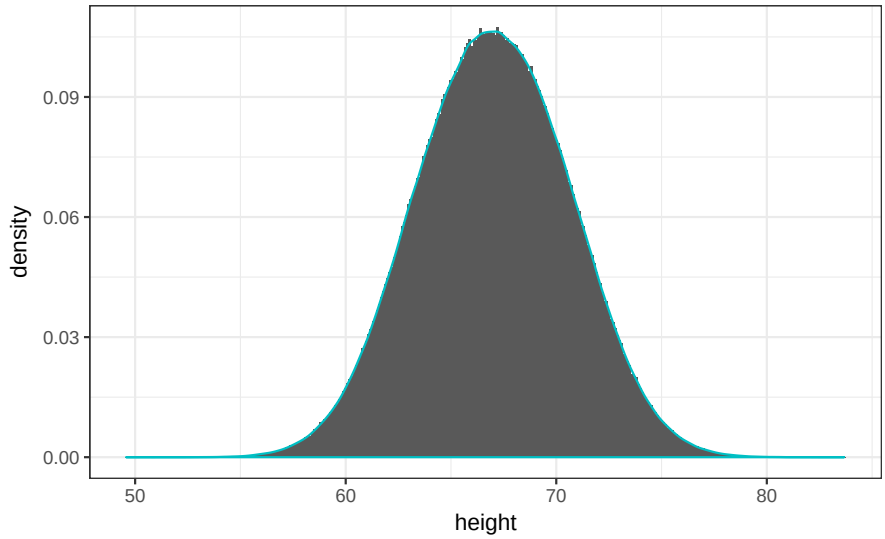
However, we make an assumption that helps us perhaps approximate it. Because we have 1,000,000 values, measured very precisely, we can make a histogram with very very small bins. The assumption is that if we do this, consecutive bins will be similar, this is what we mean by smooth: we don't have big jumps. So here a hypothetical histogram with bins of size 1



The smaller we make the bins the smoother the histogram gets. Here are the histograms with bin width of 1, 0.5 and 0.1:

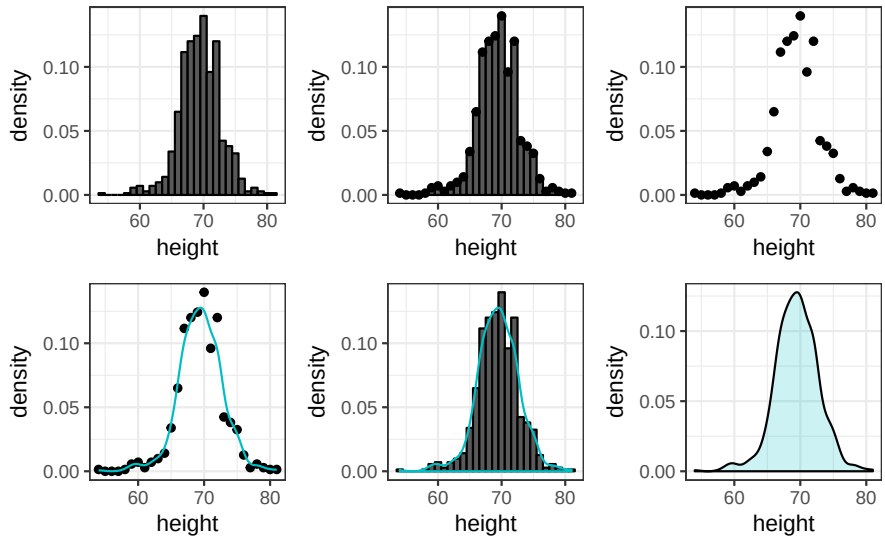


The smooth density is basically the curve that goes through the top of the histogram bars when the bins are very very small. To make the curve not depend on the hypothetical size of the hypothetical list we compute the curve on frequencies rather than counts



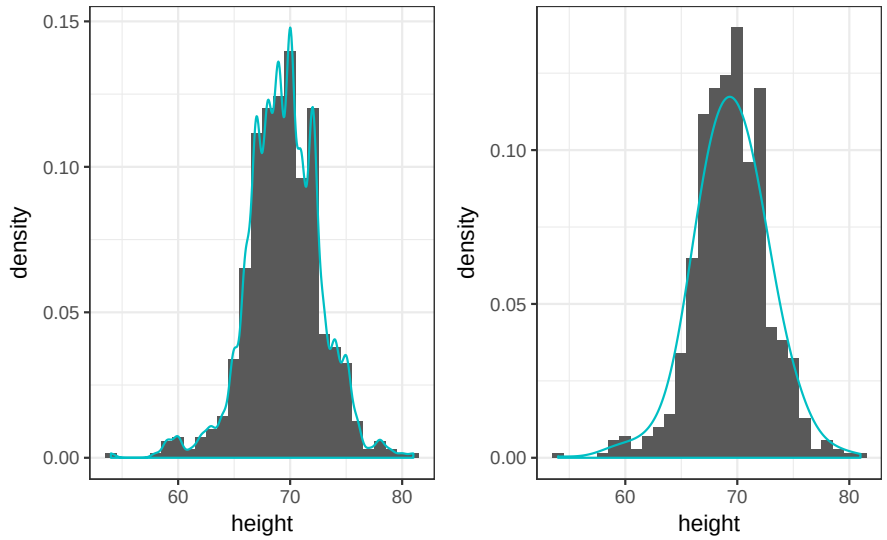
Now, back to reality. We don't have millions of measurements, instead we have 708 and we can't make a histogram with very small bins.

So instead we make the histogram, computing frequencies rather than counts, using bin sizes appropriate for our data, and we draw a smooth curve that goes through the tops of the histogram bars:



Note that *smooth* is a relative term. We can actually control how *smoothness* of the curve that defines the smooth density through an option in the function that computes the smooth density. Here are two examples using different degrees of smoothness on the same histogram:

```
p1 <- heights %>% filter(sex=="Male")%>% ggplot(aes(height)) +
  geom_histogram(aes(y=..density..), binwidth = 1) +
  geom_density(col="#00BFC4", adjust = 0.5)
p2 <- heights %>% filter(sex=="Male") %>% ggplot(aes(height)) +
  geom_histogram(aes(y=..density..), binwidth = 1) +
  geom_density(col="#00BFC4", adjust = 2)
grid.arrange(p1,p2, ncol=2)
```

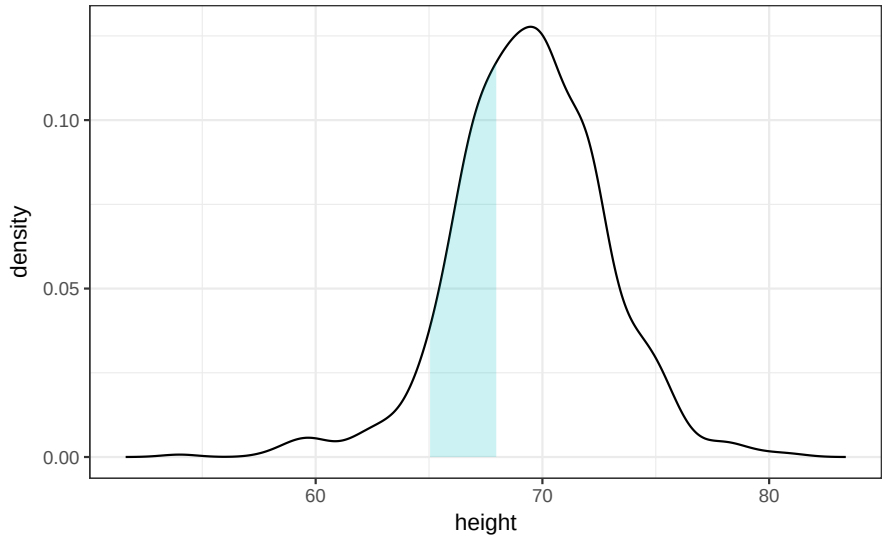


We need to make this choice with care as the resulting visualizations can change our interpretation of the data. We should select a degree of smoothness that we can defend as being representative of the underlying data. In th case of height, we really do have reason to believe that there should the proportion of people with similar heights should be the same. For example, the proportion that is 72 inches should be more similar to the proportion that is 71, then to the proportion that is 78 or 65. This implies that the curve should be pretty smooth; more like the example on the right than on the left.

While the histogram is an assumption free summary, the smoothed density is based on assumptions.

1.3.0.1 Interpreting the y-axis

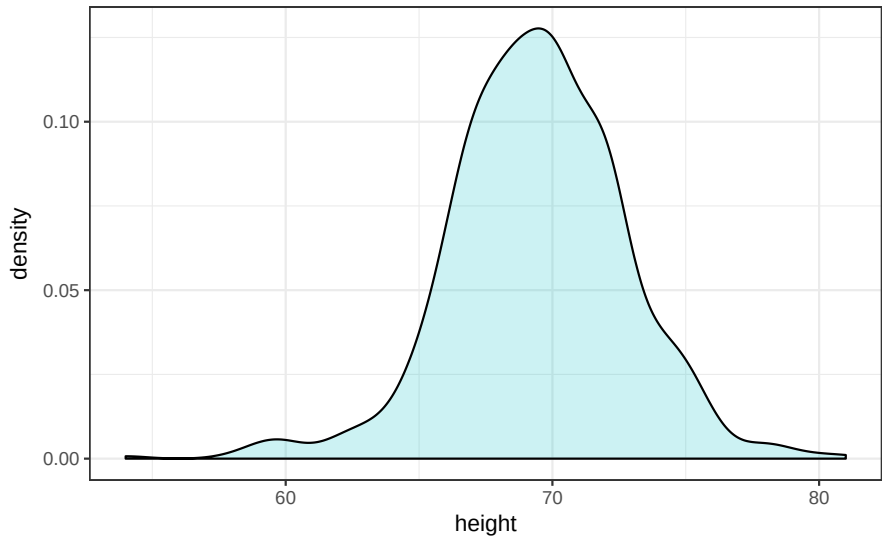
Finally, we point out that interpreting the y-axis of a smooth density plot is not straight forward. It is scaled so that the area under the density curve adds up to 1. So if you imagine we form a bin with a base 1 unit in length, the y-axis value tells us the proportion of values in that bin. But this is only true for bins of size 1. For other sized intervals, the best way to determine the proportion of data in that interval is by computing the proportion of the total area contained in that interval. For example here are the proportion of values between 65 and 68:



The proportion of this area is about 0.31 meaning that about that proportion is between 65 and 68 inches.

Understanding this we are ready to use the smooth density as a summary. For this dataset we would feel

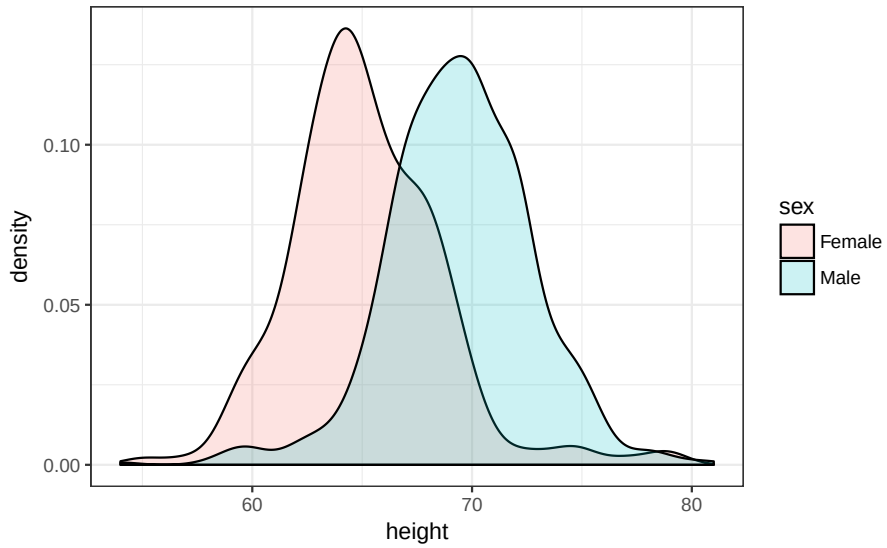
quite comfortable with the smoothness assumption and therefore with sharing this aesthetically pleasing figure with ET, which he could use to understand our male heights data:



1.3.0.2 Densities Permit Stratification

As a final note, we point out that an advantage of smooth densities over histograms for visualization purposes is that makes it easier to compare two distributions. This in large part because the jagged edges of the histogram add clutter. Here is an example comparing male and female heights:

```
heights %>% ggplot(aes(height, fill=sex)) + geom_density(alpha = 0.2)
```



With the right argument, `ggplot` automatically shades the intersecting region with a different color.

1.4 Normal Distribution

Histograms and density plots provide excellent summaries of a distribution. But can we summarize even further? We often see the average and standard deviation used as summary statistics: a two number summary! To understand what these summaries are and why they are so widely used we need to understand the normal distribution.

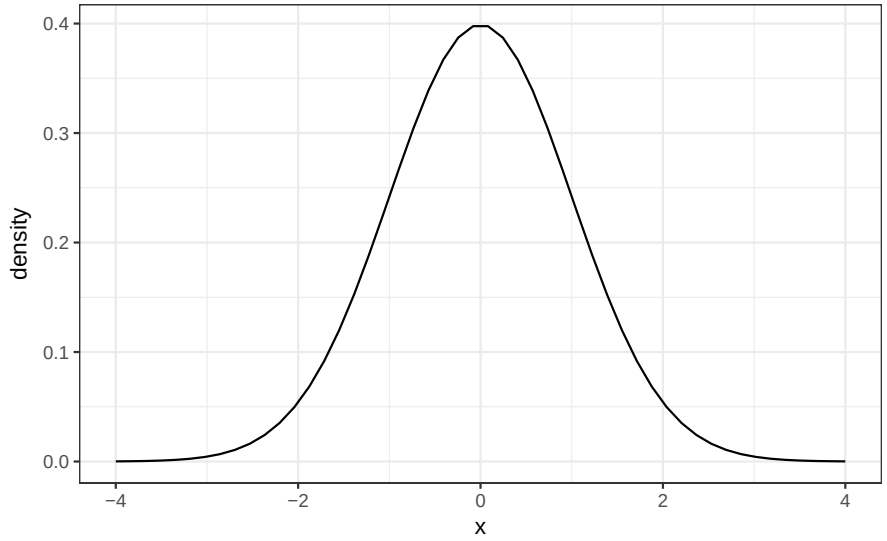
The normal distribution, also known as the bell curve and as the Gaussian distributions, is one of the most famous mathematical concepts in history. A reason for this is that approximately normal distributions occur in many situations. Examples include gambling winnings, heights, weights, blood pressure, standardized test scores, and experimental measurement errors. There are explanations for this, but we explain this in a later chapter. Here we focus on how the normal distribution helps us summarize data.

Rather than using data, the normal distribution is defined with a mathematical formula. For any interval (a, b) the proportion of values in that interval can be computed using this formula:

$$\Pr(a < x < b) = \int_a^b \frac{1}{\sqrt{2\pi}s} \exp\left\{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right\} dx$$

You don't need to memorize or understand the details of the formula. But note that it is completely defined by just two parameters m and s, the rest of the symbols in the formula represent the interval ends that we determine, a and b, and known mathematical constants π and e. These two, parameters m and s are referred to as the *average*, also called the *mean*, and the *standard deviation* (SD) of the distribution respectively.

The distribution is symmetric, centered at the average, and most values (about 95%) are withing 2 SDs from the average. Here is what it looks like when the average is 0 and the SD is 1:



The fact that the distribution is defined by just two parameters implies that if a dataset is approximated by a normal distribution, all the information needed to describe the distribution can be encoded in just two numbers: the average and the standard deviation, which we now define for an arbitrary list of numbers.

For a list of numbers contained in a vector `x` the average is defined as

```
average <- sum(x) / length(x)
```

and the SD is defined as

```
SD <- sqrt( sum( (x-mu)^2) / length(x))
```

which can be interpreted as the average distance between values and their average.

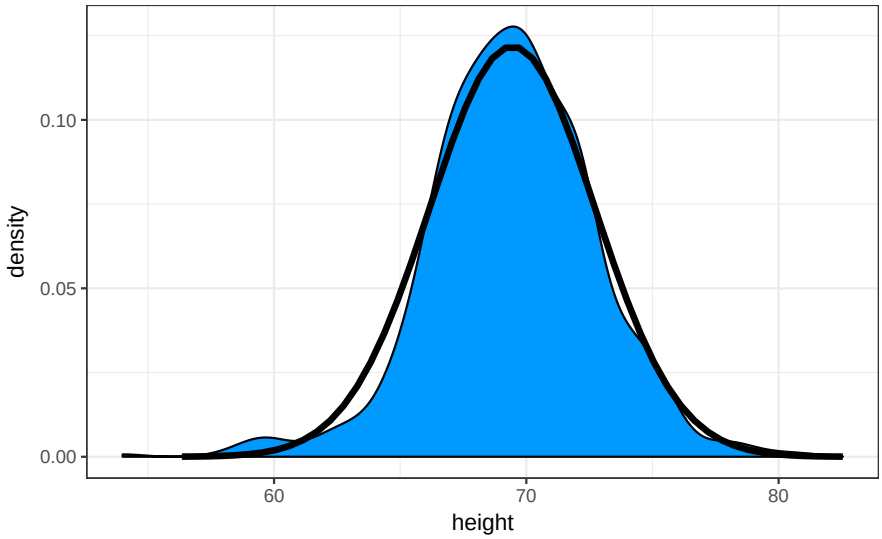
Let's compute the values for the height for males which we will store in the object x:

```
index <- heights$sex=="Male"
x <- heights$height[index]
```

The pre-built functions `mean` and `sd` [Footnote: SD divide by n-1] can be used here:

```
average <- mean(x)
SD <- sd(x)
c(average=average,SD=SD)
#> average      SD
#>   69.44    3.27
```

Here is a plot of the smooth density and the normal distribution with mean average = 69.444 and SD = 3.275



We note that the it does appear to be quite a good approximation. We now will see how well this approximation works at predicting proportion of values within intervals.

1.4.0.1 Standardized units

For data that is approximately normally distributed it is convenient to think in term of *standard units*. The standard unit of a value tells us how many standard deviations away from the average it is. Specifically, for a value x we define it as $z = (x - \text{average})/\text{SD}$. If you look back at the formula for the normal distribution you notice that what is being exponentiation is $-z/2$. The maximum of of $\exp -z^2/2$ is when $z = 0$ which explains why the maximum of the distribution is at the mean. It also explains the symmetry since $-z/2$ is symmetric around 0.

If we convert the normally distributed data to standard units we can quickly know if, for example, a person is about average ($z = 0$), one of the largest ($z = 2$), one of the smallest ($z = -2$) or an extremely rare occurrence ($z > 3$ or $z < -3$). Note that it does not matter what the original units are, these rules apply to data that is approximately normal.

In R we can obtain standard units using the function `scale`

```
z <- scale(x)
```

Now to see how many men are within 2 SDs from the average we simply type:

```
mean(abs(z) < 2)
#> [1] 0.951
```

Note that it is about 95% which is what the normal distribution predicts! To further confirm that in fact the approximation is a good one we can use quantile-quantile plots.

1.4.1 Quantile-quantile QQ plots

A systematic way to assess how well the normal distribution fits the data is to check if the observed and predicted proportions match. In general, the approach of the QQ-plot is as follows

1. Define a series of proportions $p = 0.05, \ldots .95$
2. For each p determine the value q so that the proportion of values in the data below q is p. The qs are referred to as the *quantiles*.

To give a quick a example, for the male heights data we have that

```
mean(x <= 69.5)
#> [1] 0.504
```

50% are shorter or equal to 69 inches. This implies that if $p = 0.50$ then $q = 69.5$.

Now we define a series of p

```
p <- seq(0.05, 0.95, 0.05)
```

If the quantiles for the data match the quantiles for the normal then it must be because the data follows a normal distribution.

To obtain the quantiles from the data we can use the `quantile` function like this:

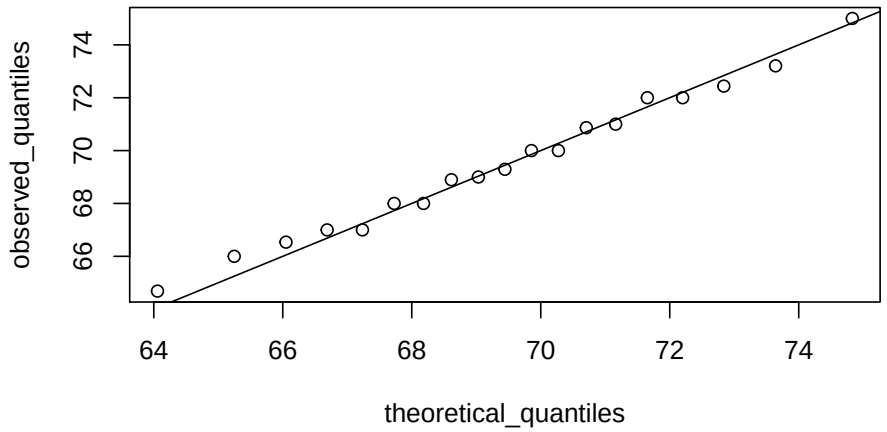
```
observed_quantiles <- quantile(x, p)
```

To obtain the theoretical normal distribution quantiles, with the corresponding average and SD, we use the `qnorm` function:

```
theoretical_quantiles <- qnorm( p, mean = mean(x), sd = sd(x))
```

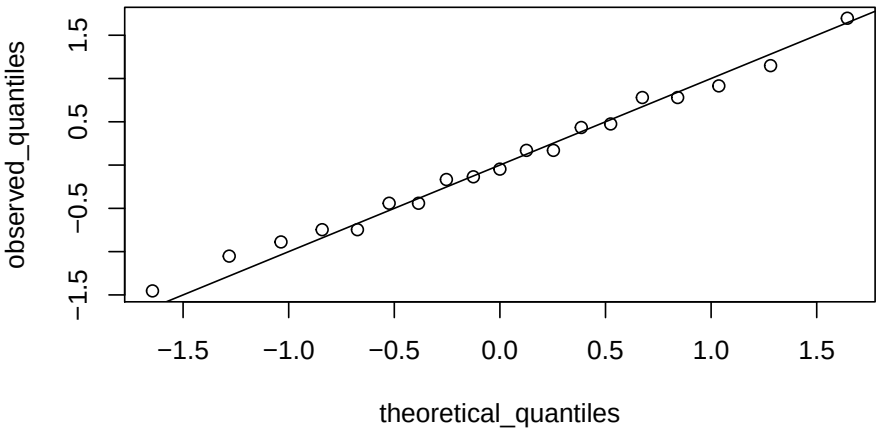
To see if the match or not we plot them against each other and draw the identity line:

```
plot(theoretical_quantiles, observed_quantiles)
abline(0,1)
```



Note that this code become much cleaner if we use standard units:

```
observed_quantiles <- quantile(z, p)
theoretical_quantiles <- qnorm(p)
plot(theoretical_quantiles, observed_quantiles)
abline(0,1)
```



1.4.1.1 Percentiles

Before we move on, let's define some terms that are commonly used in exploratory data analysis.

Percentiles are special case of *quantiles* that are commonly used. The percentiles are the quantiles you obtain when setting the p at $0.01, 0.02, ..., 0.99$. We call, for example, the case of $p = 0.25$ the 25-th percentile, which gives us a number for which 25% of the data is below. The most famous percentile is the 50th also known as the *median*.

Note that for the normal distribution the *median* and average are the same, but this is generally not the case.

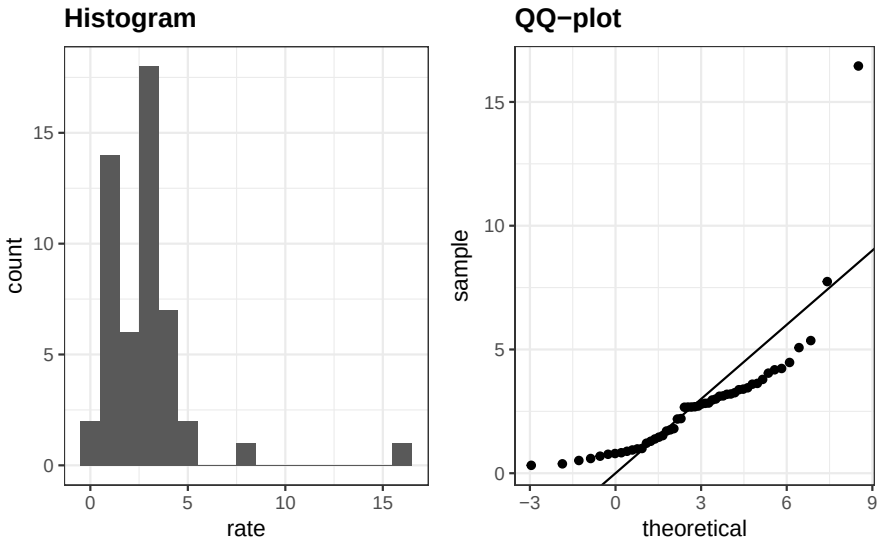
Another special case that receives a name are the *quartiles* which are obtained when setting $p = 0.25, 0.50,$ and 0.75.

1.4.2 Summarizing male heights with two numbers

Using the histogram, density plots and qq-plots we have become convinced that the male height data is well approximated with a normal distribution. In this case, we report back to ET a very succinct summary: Male heights follow a normal distribution with an average of 69.444 inches and a SD of 3.275 inches. With this information ET will have everything he needs to know what to expect when he meets our male students.

1.4.3 Boxplots

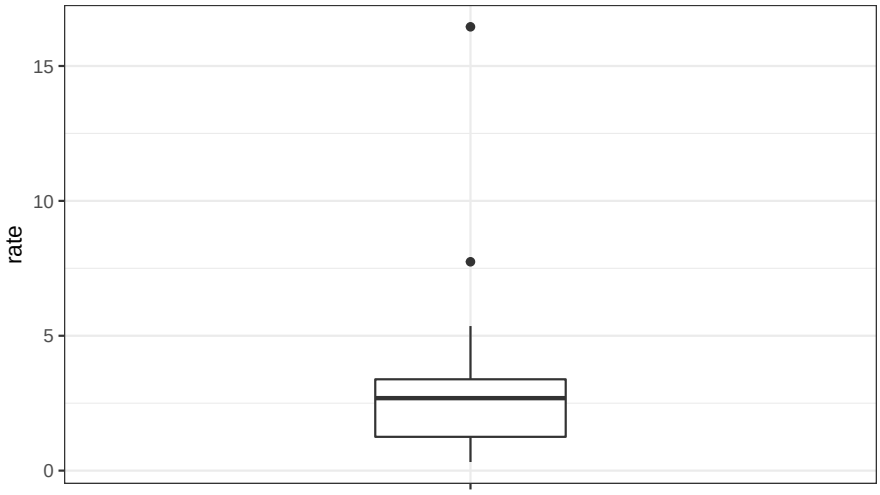
To introduce boxplots we will go back to the US murders data. Suppose want to summarize the murder rate distribution. Using the data visualization technique we have learned we can quickly notice that the normal approximation does not apply here:



In this case, the histogram, or a smooth density plot, would serve as a relatively succinct summary.

Now, suppose those used to receiving just two numbers as summaries ask us for a more compact summary.

Here Tukey offers some advice. Provide a five number summary composed of the range along with the quartiles (the 25th, 50th, and 75th percentiles). Tukey further suggested that we ignore *outliers* when computing the range and instead plot these as independent points. We provide a detailed explanation of outliers later in the chapter. Finally, he suggested we plot these numbers as a "box" with "whiskers"" like this:

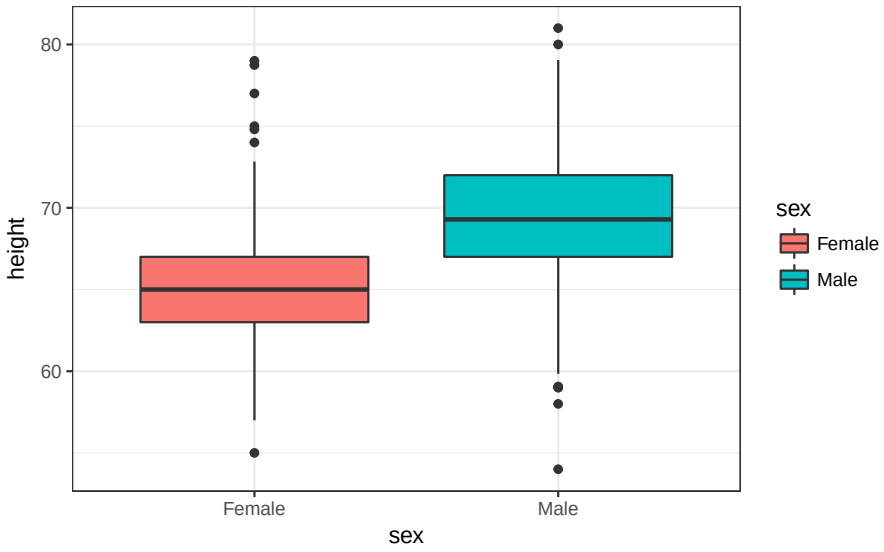


with the box defined by the 25% and 75% percentile and the whiskers showing the range. The distance between these two are called the *interquartile* range. The two points are outliers according to Tukey's definition. The median is shown with a horizontal line. Today, we call these *boxplots*.

From just this simple plot we know that the median is about 2.5, that the distribution is not symmetric, and that the range is 0 to 5 for the great majority of states with two exceptions.

They are even more useful when we want to quickly compare two or more distributions. For example, here are the heights for men and women.

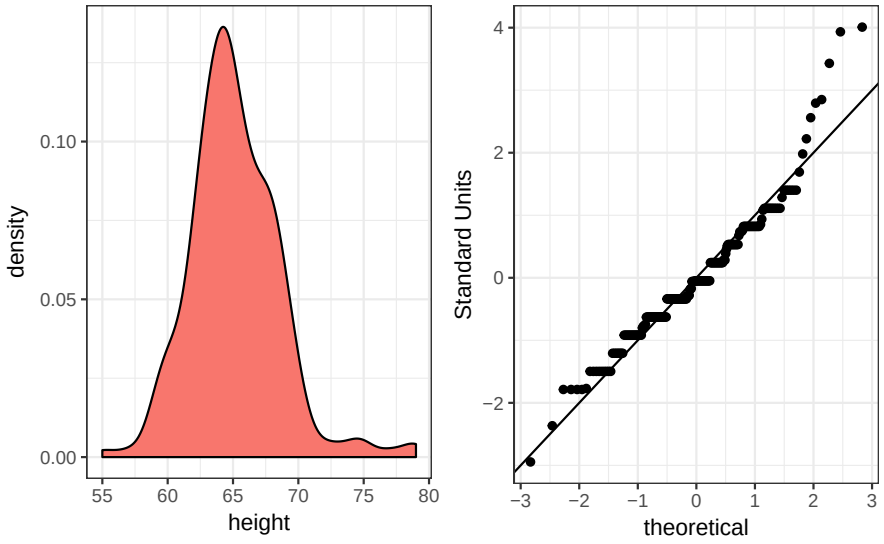
```
heights %>% ggplot(aes(x=sex, y=height, fill=sex)) +
  geom_boxplot()
```



The plot immediately reveals that males are, on average, taller than females. The standard deviations appear to be similar.

1.4.4 Are women heights normally distributed?

Note we have to give ET a full summary of our heights as we have not yet summarized female heights. We expect that they will follow a normal distribution, just like males. However, exploratory plots reveal that the approximation is not as useful:



We see something we did not see for the males. The density plot has a second "bump". Also the qqplot shows that the highest points, tend to be taller than expected by the normal distribution. When reporting back to ET we might need to provide a histogram rather than just the average and standard deviation for the female heights.

However, go back and read Tukey's quote. We have noticed what didn't expected to see. If we look at other female height distributions we do find that they are well approximated with a normal distribution. How are our female students? Is our class a requirement for the female basketball team? Are small proportion of females claiming to be taller than they are? Another, perhaps more likely explanation, is that in the form students used to entire their heights, `FEMALE` was the default sex and some males entered their heights but

forgot to change the sex variable. In any case, data visualization has helped discover a potential flaw in our data.

1.5 Robust Summaries

```
library(tidyverse)
ds_theme_set()
```

1.5.1 Outliers

We previously described how boxplots show *outliers* but we did not provide a precise definition. Here we discuss outliers, approaches that can help detect them, and summaries that take into account their presence.

Outliers are very common in data science. Data recording can be complex and it is common to observe data points generated in error. For example, an old monitoring device may read out nonsenscial measurements before completely failing. Human error is also a source of outliers with data entry is done manually. For example, one individual may mistakingly enter their height in centimeters instead of inches.

Now, how do we distinguish an outlier from measurement that were to big or to small just due to this expected variability? This is not always an easy question to answer but we try to provide some guidance. We start with a simple example.

Suppose a collegaue is charged with collecting demography data for a group of males. The data is stored in the obejct

```
#>  num [1:500] 5.59 5.8 5.54 6.15 5.83 5.54 5.87 5.93 5.89 5.67 ...
```

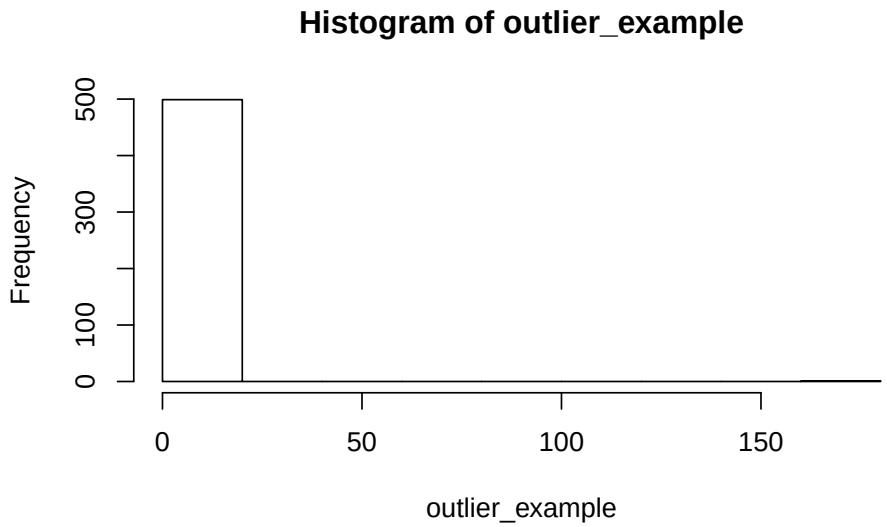
Our colleage uses the fact that heights are usually well approximated by a normal distribution and summarizes the data with average and standard deviation:

```
mean(outlier_example)
#> [1] 6.1
sd(outlier_example)
#> [1] 7.8
```

and writes a report on the interesting fact that this group of males is much taller than usual. The average height is over six feet tall! Using your data science skills you notice something else that is unexpected. The standard deviation is over 7 inches. Adding and subtracting two standard deviations, you note that 95% of this population will have heights between -9.489, 21.697 inches which does not make sense.

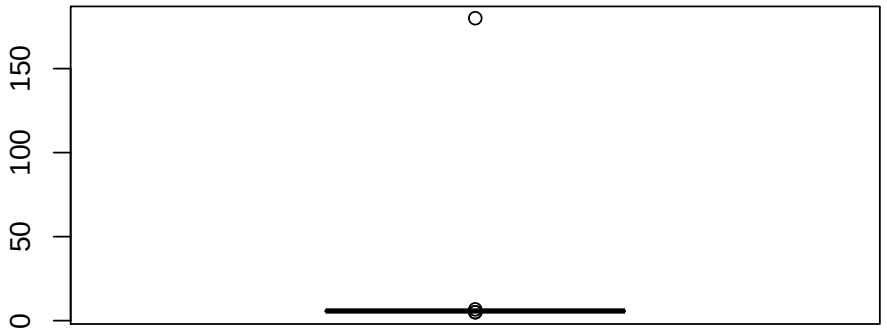
A quick plot reveals the problem:

```
hist(outlier_example)
```



There appears to be at least one value that is nonsensical, since we know that a height of 180 feet is impossble. The boxplot detects this point as on outlier:

```
boxplot(outlier_example)
```



1.5.2 Median

When we have an outlier like this, the average can become very large. Note that, mathematically, we can make the average as large as we want by simply changing one number: with 500 data points, we can increase the average by any amount Δ by adding $\Delta \times$ 500 to a single number. The median, defined as the value for which half the values and smaller and the other half is bigger, is robust to such outliers. No matter how large we make the largest point, the median remains the same.

With this data the median is

```
median(outlier_example)
#> [1] 5.74
```

which is about 5 feet and 9 inches.

The median is what boxplot display as a horizontal line.

1.5.3 The Inter Quartile Range

The box in boxplots are defined by the first and third quartile. These are meant to provide an idea of the variability in the data: 50% of the data is within this range. The difference between the 3rd and 1st quartile (or 75-th and 25-th percentiles)is referred to as the inter quartile range (IQR). Note that, as the median, this quantity will be robust to outliers as large values do not affect it. We can do some math to see that for

normal data the IQR / 1.349 approximates the standard deviation of the data had an outlier not be present. We can see that this works well in our example since we get a standard deviation estiamte of

```
IQR(outlier_example) / 1.349
#> [1] 0.245
```

which is about 3 inches.

1.5.4 Tukey's definition of an outlier

In R, points falling outside the whiskers of the boxplot are referred to as *outliers*. This definition of outlier was introduced by Tukey. The top whisker ends at the 75-th percentile plus 1.5 × IQR. Similarly the bottom whisker ends at the 25-th percentile minus 1.5× IQR. If we define the first and third quartiles as Q_1 and Q_3 respectively then an outlier is anything outside the range: $[Q_1 - 1.5 \times (Q_3 - Q1), Q_3 + 1.5 \times (Q_3 - Q1)]$.

When the data is normally distributed the standard units of these values are

```
q3=qnorm(0.75)
q1=qnorm(0.25)
iqr <- q3-q1
r <- c(q1 - 1.5*iqr , q3 + 1.5*iqr)
r
#> [1] -2.7  2.7
```

Using the `pnorm` function we see that 99.3% of the data falls in this interval.

Note that this is not such an extreme event: if we have 1000 datapoints that are randomly distributed we expect to see about 7 outside of this range. But these would not be outliers since we expect to see them under the typical variation.

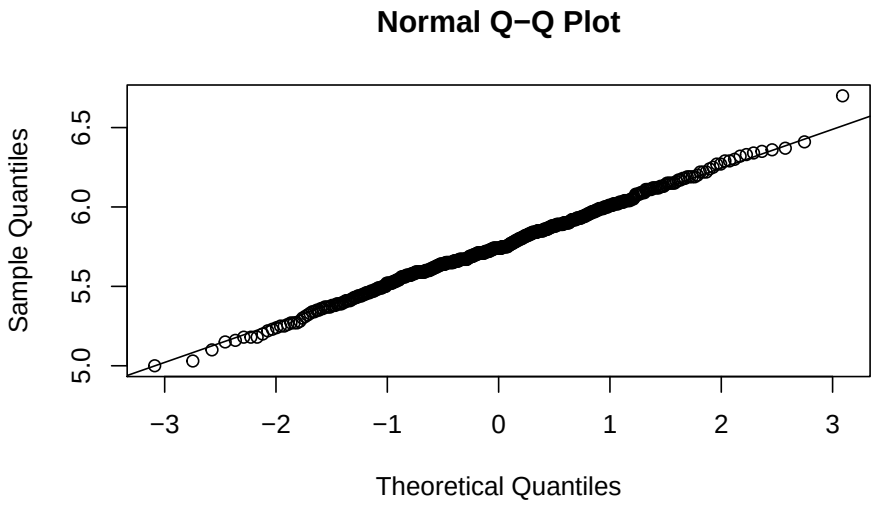
If we want an outlier to be rarer, we can increase the 1.5 to a larger number. Tukey also used 3 and called these *far out* outliers. With a normal distribution 100% of the data falls in this interval. This translates into about 1 in a million chance of being above the range and for being below the range. In the `geom_boxplot` function this can be controlled by the `outlier.size` argument, which defaults to 1.5.

The 180 feet measurement is well beyond the range of the height data:

```
max_height <- quantile(outlier_example, 0.75) + 3*IQR(outlier_example)
```

If we take this value out, we can see that the data is in fact normally distributed as expected:

```
x <- outlier_example[outlier_example < max_height]
qqnorm(x)
qqline(x)
```



1.5.5 Median Absolute Deviation

Another estimate the standard deviation in the presence of outliers we can use the median absolote deviation (MAD). This quantify first computes the median, the for each value the distance to the median, and then the median of these distances. For technical reasons not discussed here, this quantity needs to be multiplied by 1.4826 to assure it approximates the actual standard deviation. The `mad` function already incorporates this correction. For the height data we get a MAD of

```
mad(outlier_example)
#> [1] 0.237
```

which is about 3 inches.

1.5.6 Self-reported heights

```
data(reported_heights)
str(reported_heights)
#> 'data.frame':    1052 obs. of  3 variables:
#>  $ time_stamp: chr  "9/2/2014 13:40:36" "9/2/2014 13:46:59" "9/2/2014 13:59:20" "9/2/2014 14:51:53"
#>  $ sex       : chr  "Male" "Male" "Male" "Male" ...
#>  $ height    : chr  "75" "70" "68" "74" ...
```

Height is a character so we create a new column with the numeric version

```
reported_heights <- reported_heights %>%
  mutate(original_heights = height, height = as.numeric(height))
```

We get a warning about NAs. This is because some of the self reported heights were not numbers: We can see why we get these

```
reported_heights %>%
  filter(is.na(height)) %>% head
#>          time_stamp    sex height original_heights
#> 1 9/2/2014 15:16:28   Male     NA            5' 4"
#> 2 9/2/2014 15:16:37 Female     NA            165cm
#> 3 9/2/2014 15:16:52   Male     NA              5'7
#> 4 9/2/2014 15:16:56   Male     NA            >9000
```

```
#> 5 9/2/2014 15:16:56   Male     NA             5'7"
#> 6 9/2/2014 15:17:09 Female     NA             5'3"
```

We can see the that some students self reported their heights in the feet and inches format rather than report inches. Others used centimeters and other were just trolling. For now we will remove these entries:

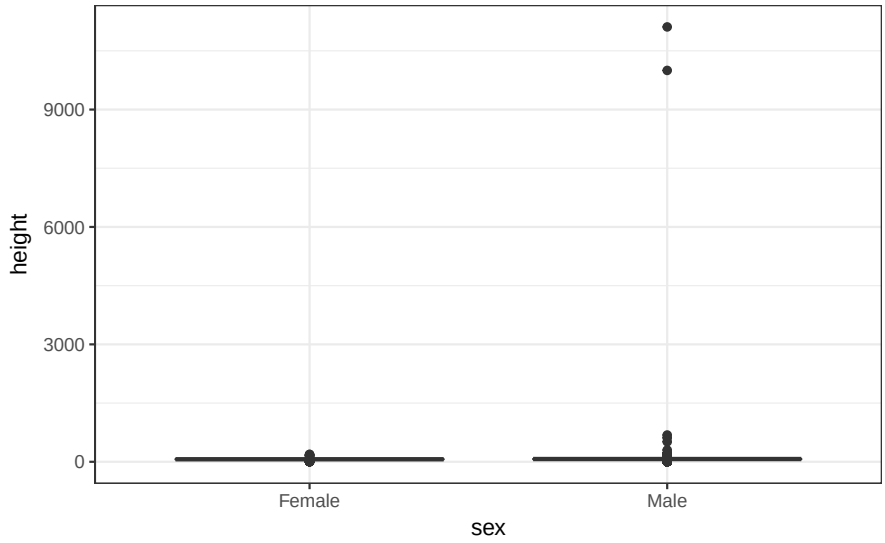
```
reported_heights <- filter(reported_heights, !is.na(height))
```

If we compute the average and standard deviation we notice we obtain strange results and different from the median and MAS:

```
reported_heights %>% group_by(sex) %>%
  summarize(average = mean(height), sd = sd(height),
            median = median(height), MAD = mad(height))
#> # A tibble: 2 x 5
#>      sex average    sd median   MAD
#>    <chr>   <dbl> <dbl>  <dbl> <dbl>
#> 1 Female    63.1  27.3   64.2  3.78
#> 2   Male   104.4 541.5   70.0  4.45
```

This suggest that we have outliers and this is confirmed by simply creating a boxplot:

```
reported_heights %>% ggplot(aes(sex, height)) + geom_boxplot()
```



We can see some rahter extreme values. To see what these values are we can quickly look at the larges values using the `arrange` function:

```
reported_heights %>% arrange(desc(height)) %>% top_n(10, height)
#>             time_stamp    sex height original_heights
#> 1    9/3/2014 23:55:37   Male  11111            11111
#> 2   4/10/2016 22:45:49   Male  10000            10000
#> 3    8/10/2015 3:10:01   Male    684              684
#> 4   2/27/2015 18:05:06   Male    612              612
#> 5    9/2/2014 15:16:41   Male    511              511
#> 6    9/7/2014 20:53:43   Male    300              300
#> 7  11/28/2014 12:18:40   Male    214              214
#> 8  11/24/2015 10:39:45   Male    192              192
#> 9  12/26/2014 10:00:12   Male    190              190
#> 10  11/6/2016 10:21:02 Female    190              190
```

The first seven entries look like strange errors. However the next few look like they were entered as centimeters. Note, for example, that 184 is six feet tall.

We can review all the non-sensical answers by looking at the data considered to be *far out* by Tukey:

```
max_height <- quantile(reported_heights$height, .75) + 3*IQR(reported_heights$height)
min_height <- quantile(reported_heights$height, .25) - 3*IQR(reported_heights$height)
c(min_height, max_height )
#> 25% 75%
#>  44  93
```

```
outliers <- reported_heights %>%
  filter(!between(height, min_height, max_height)) %>%
  arrange(height)
outliers$height
#>   [1]     0.00     0.70     1.00     1.00     1.00     1.60     1.70
#>   [8]     2.00     5.00     5.00     5.10     5.10     5.10     5.10
#>  [15]     5.11     5.11     5.11     5.11     5.20     5.20     5.20
#>  [22]     5.25     5.25     5.30     5.40     5.40     5.50     5.50
#>  [29]     5.50     5.50     5.50     5.50     5.50     5.50     5.50
#>  [36]     5.50     5.50     5.51     5.60     5.60     5.60     5.60
#>  [43]     5.60     5.60     5.60     5.69     5.70     5.70     5.70
#>  [50]     5.70     5.70     5.70     5.75     5.80     5.80     5.80
#>  [57]     5.80     5.80     5.90     5.90     5.90     5.90     6.00
#>  [64]     6.00     6.00     6.00     6.00     6.00     6.00     6.00
#>  [71]     6.00     6.00     6.00     6.00     6.00     6.00     6.00
#>  [78]     6.00     6.00     6.00     6.10     6.10     6.20     6.20
#>  [85]     6.30     6.50     6.50     6.50     6.70     7.00    12.00
#>  [92]    19.00    22.00    23.00    25.00    34.00   100.00   103.20
#>  [99]   111.00   120.00   120.00   150.00   150.00   152.00   157.00
#> [106]   158.00   158.00   158.00   159.00   160.00   160.00   162.00
#> [113]   162.00   163.00   164.00   164.00   165.00   165.00   165.00
#> [120]   167.00   167.00   167.00   168.00   168.00   168.00   168.00
#> [127]   169.00   169.00   169.00   169.00   170.00   170.00   170.00
#> [134]   170.00   170.00   170.00   170.00   172.00   172.00   172.00
#> [141]   172.00   172.00   172.00   172.00   173.00   173.00   173.00
#> [148]   174.00   174.00   174.00   174.00   175.00   175.00   175.00
#> [155]   175.00   175.00   176.00   176.00   177.00   178.00   178.00
#> [162]   178.00   178.00   178.00   179.00   180.00   180.00   180.00
#> [169]   180.00   180.00   180.00   180.00   180.00   182.00   183.00
#> [176]   183.00   183.00   184.00   184.00   184.00   185.00   185.00
#> [183]   190.00   190.00   192.00   214.00   300.00   511.00   612.00
#> [190]   684.00 10000.00 11111.00
```

Examining these heights we see two common mistakes: entries in centimeters, which turn out to be too large, and entries of the form `feet.inches` which turn out to be very small. Some of the the even smaller values, such as 1.6, could be entries in meters.

Using data wrangling techniques we will learn wasy to correct these values and turn them into inches. Here we note that we were able to detect this problem using careful data exploration to unconver problems with the data: the first step in the great majority of data science projects.

1.6 Summarizing data with dplyr

An important part of exploratory data analysis is summarizing data. We learned about the average and standard deviation as a two summary statistic that provides all the necessary information to summarize data that is normally distributed. We also learned that better summaries can be achieved by splitting data into groups before using the normal approximation. For example, in our heights dataset we described the height of men and women separately. In this section we cover two new `dplyr` verbs that make these computations easier: `summarize` and `group_by`. We learn to access resulting values using what we call the *dot placeholder*. Finally, we also learn to use `arrange` which helps us examine data after sorting.

```
library(tidyverse)
```

1.6.1 Summarize

The `summarize` function in `dplyr` provides a way to compute summary statistics with intuitive and readable code. We start with a simple example based on heights

```
library(dslabs)
data(heights)
```

that computes the average and standard deviation for males:

```
s <- heights %>%
  filter(sex == "Male") %>%
  summarize(average = mean(height), standard_deviation = sd(height))
s
#>   average standard_deviation
#> 1    69.4               3.27
```

This takes our original data table as input, filters to keep only males and then produces a new, summarized, table with just the average and the standard deviation of heights. We get to pick the names of the columns of the resulting table. For example, above we decided to use `average` and `standard_deviation`, but we could have used other names just the same.

Because the resulting table, stored in `s`, is a data frame, we can access the components with the accessor `$`, which in this case will be a numeric:

```
s$average
#> [1] 69.4
s$standard_deviation
#> [1] 3.27
```

As with most other `dplyr` functions, `summarize` is aware of the variable names and we can use them directly. So when inside the call to the `summarize` function we write `mean(height)` it is accessing the column with the name, and then computing the average of the respective numeric vector. Note that we can compute any other summary that operates on vectors and returns a single value. For example, we can add the median, min and max like this:

```
heights %>%
  filter(sex == "Male") %>%
  summarize(median = median(height), minimum = min(height), maximum = max(height))
#>   median minimum maximum
#> 1   69.3      54      81
```

Note that we can obtain these three values wit just one line using the `quantiles` function. For example `quantile(x, c(0,0.5,1))` returns the min, median, and max of the vector `x`. However, if we attempt to use a function that returns two or more values:

```
heights %>%
  filter(sex == "Male") %>%
  summarize(range = quantile(height, c(0, 0.5, 1)))
```

we will receive an error: `Error: expecting result of length one, got : 2`. With the function `summarize` we can only call functions that return a single value. In a later chapter we will learn how to deal with functions that return more than one value.

As another example of how we can use the `summarize` function let's compute the average murder rate for the United States. Remember our data table includes total murders and population size for each state and we have already used `dplyr` to add a murder rate column:

```
data(murders)
murders <- murders %>% mutate(murder_rate = total/population*100000)
```

However, note that the US murder is **not** the average of the state murder rates:

```
summarize(murders, mean(murder_rate))
#>   mean(murder_rate)
#> 1              2.78
```

because in this computation the small states are given the same weight as the large ones. The US murder rate is proportional to the total US murders divided by the total US population. So the correct computation is:

```
us_murder_rate <- murders %>% summarize( rate = sum(total) / sum(population) * 100000)
us_murder_rate
#>   rate
#> 1 3.03
```

This computation counts larger states proportionally to their size and this results in a larger value.

1.6.2 Using the dot to accessing the pipped data

The `us_murder_rate` object defined above represents just one number. Yet we are storing it in a data frame

```
class(us_murder_rate)
#> [1] "data.frame"
```

since, as most `dplyr` functions, `summarize` always returns a data frame.

This might be problematic if we want to use the result with functions that require a numeric value. Here we show a useful trick to access values stored in data piped via `%>%`: when a data object is piped it can be accessed using the dot `.`. To understand what we mean take a look at this line of code:

```
us_murder_rate %>% .$rate
#> [1] 3.03
```

Note that is returns the value in the `rate` column of `us_murders_rate` making it equivalent to `us_murders_rate$rate`. To understand this line, you just need to think of `.` as a placeholder for the data that is being passed through the pipe. Because this data object is a data frame, we can access it's columns with the `$`.

To get a number from the original data table with one line of code we can type:

```
us_murder_rate <- murders %>%
  summarize( rate = sum(total) / sum(population) * 100000) %>%
  .$rate
```

```
us_murder_rate
#> [1] 3.03
```

which is now a numeric:

```
class(us_murder_rate)
#> [1] "numeric"
```

We will see other instances in which using the `.` is useful. For now, we will only use it to produce numeric vectors from pipelines constructed with `dplyr`.

1.6.3 Group then summarize

A common operation in data exploration is to first split data into groups and then compute summaries for each group. For example, we may want to compute the average and standard deviation for men and women heights separately. The `group_by` function helps us do this.

If we type this:

```
heights %>% group_by(sex)
#> # A tibble: 924 x 2
#> # Groups:   sex [2]
#>      sex height
#>   <fctr>  <dbl>
#> 1   Male     75
#> 2   Male     70
#> 3   Male     68
#> 4   Male     74
#> 5   Male     61
#> 6 Female     65
#> # ... with 918 more rows
```

the result does not look very different from `heights`, except we see this `Groups: sex [2]` when we print the object. Although not immediatly obvious from it's apperance, this is now a special data frame called a *grouped data frame* and `dplyr` functions, in particular summarize, will behave differently when acting on this object. Conceptually you can think of this table as many tables, with the same columns but not necessarily the same number of rows, stacked together in one object. Note what happens when we summarize the data after grouping:

```
heights %>%
  group_by(sex) %>%
  summarize(average = mean(height), standard_deviation = sd(height))
#> # A tibble: 2 x 3
#>      sex average standard_deviation
#>   <fctr>   <dbl>              <dbl>
#> 1 Female    65.2               3.45
#> 2   Male    69.4               3.27
```

The summarize function applies the summarization to each group separately.

For another example, let's compute the median murder rate in the four regions of the country:

```
murders %>%
  group_by(region) %>%
  summarize(median_rate = median(murder_rate))
#> # A tibble: 4 x 2
#>          region median_rate
#>          <fctr>       <dbl>
```

```
#> 1     Northeast        1.80
#> 2         South        3.40
#> 3 North Central        1.97
#> 4          West        1.29
```

1.6.4 Sorting data tables

When examining a dataset it is often convenient to sort the table by the different columns. We know about the `order` and `sort` function, but for ordering entire tables, the function `dplyr` function `arrange` is useful. For example, here we order the states by population size we type:

```
murders %>% arrange(population) %>% head()
#>                  state abb        region population total murder_rate
#> 1              Wyoming  WY          West     563626     5       0.887
#> 2 District of Columbia  DC         South     601723    99      16.453
#> 3              Vermont  VT     Northeast     625741     2       0.320
#> 4         North Dakota  ND North Central     672591     4       0.595
#> 5               Alaska  AK          West     710231    19       2.675
#> 6         South Dakota  SD North Central     814180     8       0.983
```

Note that we get to decide which column to sort by. To see the states by population, from smallest to largest, we arrange by `murder_rate` instead:

```
murders %>% arrange(murder_rate) %>% head()
#>           state abb        region population total murder_rate
#> 1       Vermont  VT     Northeast     625741     2       0.320
#> 2 New Hampshire  NH     Northeast    1316470     5       0.380
#> 3        Hawaii  HI          West    1360301     7       0.515
#> 4  North Dakota  ND North Central     672591     4       0.595
#> 5          Iowa  IA North Central    3046355    21       0.689
#> 6         Idaho  ID          West    1567582    12       0.766
```

Note that the default behavior is to order in ascending order. In `dplyr`, the function `desc` transforms a vector to be in descending order. So if we want to sort the table in descending order we can type

```
murders %>% arrange(desc(murder_rate)) %>% head()
#>                  state abb        region population total murder_rate
#> 1 District of Columbia  DC         South     601723    99       16.45
#> 2            Louisiana  LA         South    4533372   351        7.74
#> 3             Missouri  MO North Central    5988927   321        5.36
#> 4             Maryland  MD         South    5773552   293        5.07
#> 5       South Carolina  SC         South    4625364   207        4.48
#> 6             Delaware  DE         South     897934    38        4.23
```

1.6.4.1 Nested Sorting

If we are ordering by a column with ties we can use a second column to break the tie. Similarly, a third column can be used to break ties between first and second and so on. Here we order by `region` then within region we order by murder rate

```
murders %>% arrange(region, murder_rate) %>% head()
#>           state abb    region population total murder_rate
#> 1       Vermont  VT Northeast     625741     2       0.320
#> 2 New Hampshire  NH Northeast    1316470     5       0.380
#> 3         Maine  ME Northeast    1328361    11       0.828
```

```
#> 4  Rhode Island  RI Northeast    1052567    16       1.520
#> 5 Massachusetts  MA Northeast    6547629   118       1.802
#> 6      New York  NY Northeast   19378102   517       2.668
```

1.6.4.2 The top n

In the code above we have used the function `head` to avoid having the page fill with the entire data. If we want to see a larger proportion we can use the `top_n` function. Here are the

```
murders %>% top_n(10, murder_rate)
#>                  state abb        region population total murder_rate
#> 1              Arizona  AZ          West    6392017   232        3.63
#> 2             Delaware  DE         South     897934    38        4.23
#> 3  District of Columbia  DC         South     601723    99       16.45
#> 4              Georgia  GA         South    9920000   376        3.79
#> 5            Louisiana  LA         South    4533372   351        7.74
#> 6             Maryland  MD         South    5773552   293        5.07
#> 7             Michigan  MI North Central    9883640   413        4.18
#> 8          Mississippi  MS         South    2967297   120        4.04
#> 9             Missouri  MO North Central    5988927   321        5.36
#> 10      South Carolina  SC         South    4625364   207        4.48
```

Note that `top_n` picks the highest `n` based on the column given as a second argument. However, the rows are not sorted.

If the second argument is left blank, then it just takes the first `n` columns. This means that to see the top 10 states ranked by murder rate, sorted by murder rate we can type:

```
murders %>% arrange(desc(murder_rate)) %>% top_n(10)
#>                  state abb        region population total murder_rate
#> 1  District of Columbia  DC         South     601723    99       16.45
#> 2            Louisiana  LA         South    4533372   351        7.74
#> 3             Missouri  MO North Central    5988927   321        5.36
#> 4             Maryland  MD         South    5773552   293        5.07
#> 5       South Carolina  SC         South    4625364   207        4.48
#> 6             Delaware  DE         South     897934    38        4.23
#> 7             Michigan  MI North Central    9883640   413        4.18
#> 8          Mississippi  MS         South    2967297   120        4.04
#> 9              Georgia  GA         South    9920000   376        3.79
#> 10              Arizona  AZ          West    6392017   232        3.63
```

1.7 A first introduction to ggplot2

We have now described several data visualization techniques and are ready to learn how to create them in R. Throughout the book we will be using the `ggplot2` package. We can load it, along with `dplyr`, as part of the tidyverse:

```
library(tidyverse)
```

Note that many other approaches are available for creating plots in R. In fact the plotting capabilities that come with a basic installation of R are already quite powerful. We have seen examples of these already with the functions `plot`, `hist` and `boxplot`. There are also other packages for creating graphics such as `grid` and `lattice`. We chose to use `ggplot2` in this book because it breaks plots into components in a way that permits beginners to create relatively complex and aesthetically pleasing plots using syntax that is intuitive and relatively easy to remember.

One reason `ggplot2` is generally more intuitive for beginners is that it uses a *grammar of graphics* [CITE], the *gg* in `ggplot2`. This is analogous to the way learning grammar can help a beginner construct hundreds of different sentences by learning just a a handful of verbs, nouns and adjective without having to memorize each specific sentence. Similarly, by learning a handful of `ggplot2` building blocks and its grammar, you will be able to create hundreds of different plots.

Another reason `ggplot2` makes it easier for beginner is that its default behavior is carefully chosen to satisfy the great majority of cases and are aesthetically pleasing. As a result, it is possible to create informative and elegant graphs with relatively simple and readable code.

One limitation, is that ggplot is designed to work exclusively with data tables in which rows are observation and columns are variables. However, a substantial percentage of datasets that beginners work with are, or can be converted into, this format. An advantage of this approach, it that assuming that our data follows this format simplifies the code and learning the grammar.

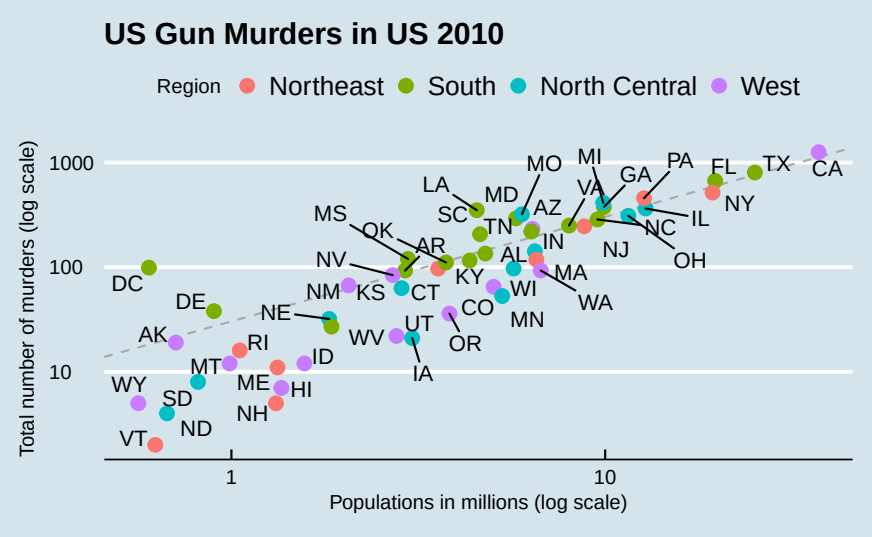
1.7.1 The Cheat Sheet

To use `ggplot2` you will have to learn several functions and arguments. These are hard to memorize so we highly recommend you have the a ggplot2 sheet cheat handy. You can get a copy here: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf or simply perform a internet search for "ggplot2 cheat sheet"

1.7.2 The components of a graph

We construct a graph that summarizes the US murders dataset.

```
library(dslabs)
data(murders)
```



We can clearly see how much states vary across population size and the total number of murders. Not surprisingly, we also see a clear relationship between murder totals and population size. A state falling on the dashed grey line has the same murder rate as the US average. The four geographic regions are denoted with color and depicts how most southern states have murder rates above the average.

This data visualization shows us pretty much all the information in the data table. The code needed to make this plot is relatively simple. We will learn to create the plot part by part.

1.7.3 The components of a graph

The first step in learning `ggplot2` is to be able to break a graph apart into components. Let's break down the plot above and introduce some of the `ggplot2` terminology. The main three components to note are:

1. **Data**: The US murders data table is being summarized. We refer to this as the **data** component.
2. **Geometry**: The plot above is a scatter plot. This is referred to as the **geometry** component. Other possible geometries are barplot, histograms, smooth densities, qqplots, and boxplots.
3. **Aesthetic mapping**: The x-axis values are used to display population size, the y-axis values are used to display the total number of murders, text is used to identify the states, and colors are used to denote the four different regions. These are the **aesthetic mappings** component. How we define the mapping depends on what **geometry** we are using.

We also note that:

4. The range of the x-axis and y-axis appears to be defined by the range of the data. They are both on log-scales. We refer to this as the **scale** component.
5. There are labels, a title, a legend, and we use the style of The Economist magazine.

We will now construct the plot piece by piece.

1.7.4 Creating a blank slate `ggplot` object

The first step in creating a `ggplot2` graph is to define a `ggplot` object. We do this with the function `ggplot` which initializes the graph. If we read the help file for this function we see that the first arguments is used to specify what data is associated with this object:

```
ggplot(data = murders)
```

We can also pipe the data. So this line of code is equivalent to the one above:

```
murders %>% ggplot()
```

Note that it renders a plot, in this case a blank slate since no geometry has been defined. The only style choice we see is a grey background.

What has happened above is that the object was created and because it was not assigned, it was automatically evaluated. But note that we can define an object, for example like this:

```
p <- ggplot(data = murders)
class(p)
#> [1] "gg"     "ggplot"
```

To render the plot associated with this object we simply print the object `p`. The following two lines of code produce the same plot we see above:

```
print(p)
p
```

1.7.5 Layers

In ggplot we create graphs by adding *layers*. Layers can define geometries, compute summary statistics, define what scales to use, or even change styles. To add layers, we use the the symbol `+`. In general a line of code will look like this:

DATA %>% `ggplot()` + LAYER 1 + LAYER 2 + … + LAYER N

Usually, the first added layer defines the geometry. We want to make a scatter plot. So what geometry do we use?

1.7.5.1 Geometry

Taking a quick look at the cheat sheet we see that the function used to create plots with this geometry is `geom_point`.

[INLCUDE SCREEN SHOT OF CHEAT SHEET]

We will see that geometry function names follow this pattern: `geom` and the name of the geometry connected by an underscore.

For `geom_point` to know what to do, we need to provide data and a mapping. We have already connected the object `p` with the `murders` data table and if we add as a layer `geom_point` we will default to using this data. To find out what mapping are expected we read the **Aesthetics** section of the help file `geom_point` help file:

> Aesthetics
>
> geom_point understands the following aesthetics (required aesthetics are in bold):
>
> x
>
> y
>
> alpha
>
> colour

and, as expected, we see that at least two arguments are required `x` and `y`.

1.7.5.2 `aes`

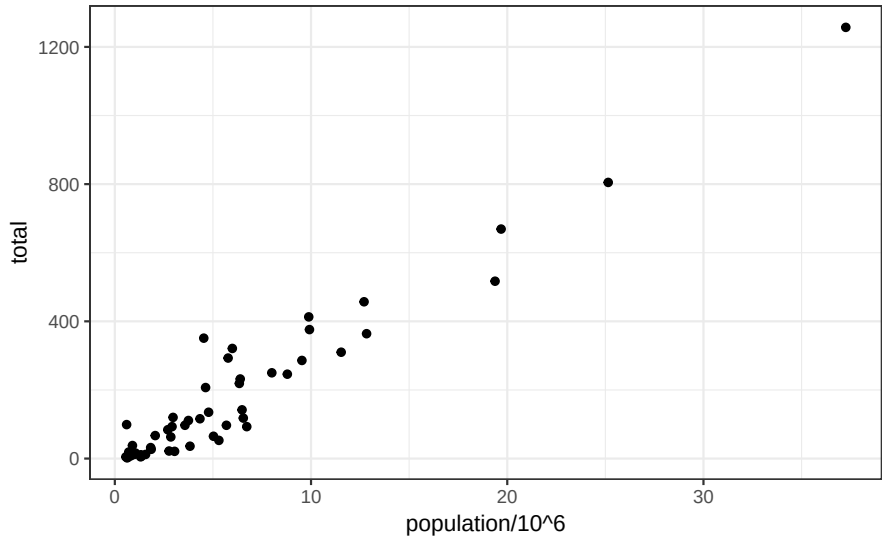
`aes` will be one of the functions that you will most use. The function connects data with what we see on the graph. We refer to this connect as the **aesthetic mappings**. The outcome of this function is often used as the argument of a geometry function. This example produces a scatter plot of total murders versus population in millions:

```
murders %>% ggplot() +
  geom_point(aes(x = population/10^6, y = total))
```

Note that we can drop the `x =` and `y =` if we wanted to as these are the first and second expected arguments as seen in the help page.

Also note that we can add a layer to the `p` object that has defined above as `p <- ggplot(data = murders)`:

```
p + geom_point(aes(population/10^6, total))
```



Note that the scale and labels are defined by default when adding this layer. Also notice that we use the variable names from the object component: `population` and `total`.

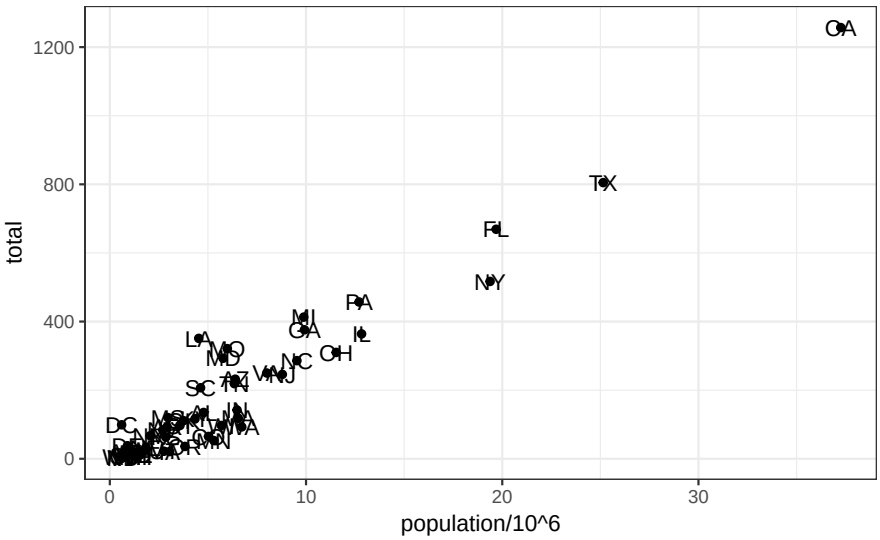
Keep in mind that the behavior of recognizing the variables from the data component, is quite specific to `aes`. With most function, if you try to access the values of `population` or `total` outside of `aes` you receive an error.

1.7.5.3 Adding other layers

A second layer in the plot we wish to make involves adding a label to each point to identify the state. The `geom_label` and `geom_text` functions permit us to add text to the plot, without and with a rectangle behind the text respectively.

Because each state (each point) has a label we need a aesthetic mapping to make the connection. By reading the help file we learn that we supply the mapping between point and label through the `label` argument of `aes`. So the code looks like this:

```
p + geom_point(aes(population/10^6, total)) +
  geom_text(aes(population/10^6, total, label = abb))
```



We have successfully added a second layer to the plot.

As an example of the unique behavior or `aes` mentioned above, note that this call

```
p_test <- p + geom_text(aes(population/10^6, total, label = abb))
```

is fine, this call

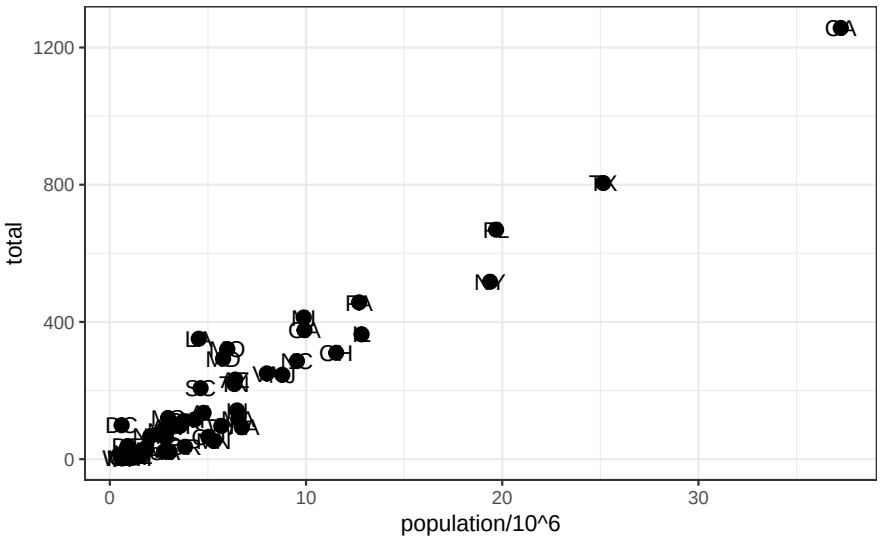
```
p_test <- p + geom_text(aes(population/10^6, total), label = abb)
```

will give you error as `abb` is not found once it is outside of `aes` function and `geom_text` does not know where to find `abb` as it is not a global variable.

1.7.5.4 Tinkering with other arguments

Note that each geometry function has many arguments other than `aes` and `data`. They tend to be specific to the function. For example, in the plot we wish to make, the points are larger than the default ones. In the help file we see that `size` is an aesthetic and we can change it like this:

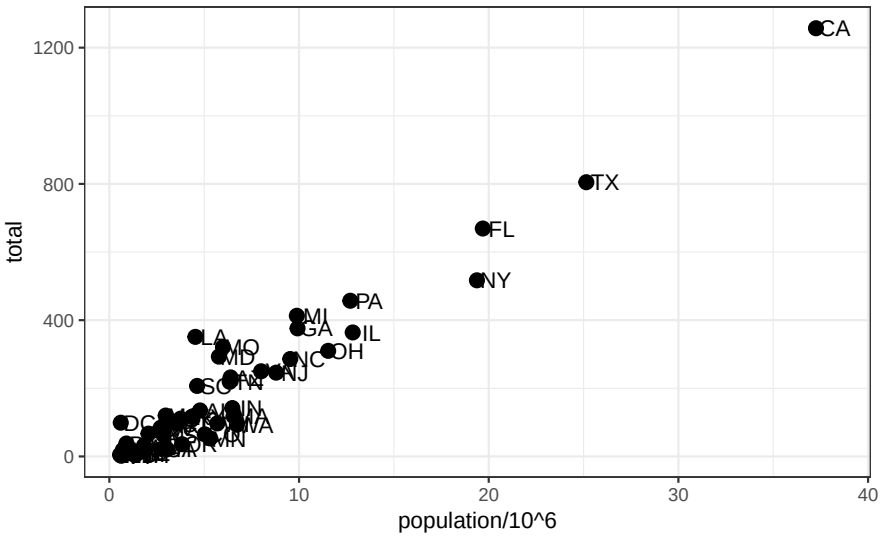
```
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb))
```



Note that `size` is **not** a mapping, it affects all the points so we do not need to include it inside `aes`.

Now that the points are larger, it is hard to see the labels. If we read the help file for `geom_text` we see learn of the `nudge_x` argument with moves the text slightly to the right:

```
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb), nudge_x = 1)
```



This is preferred as it makes it easier to read the text.

1.7.6 Global aesthetic mappings

Note that in the previous line of code, we define the mapping `aes(population/10^6, total)` twice, once in each geometry. We can avoid this by using a *glogbal* aesthetic mapping. We can do this when we define the blank slate `ggplot` object. Remember that the function `ggplot` contains an argument that permits us to define aesthetic mappings:

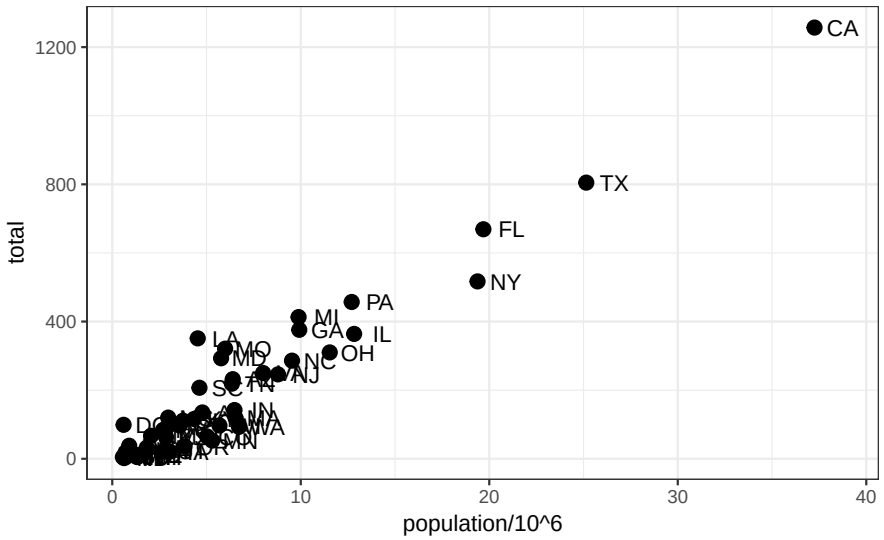
```
args(ggplot)
#> function (data = NULL, mapping = aes(), ..., environment = parent.frame())
#> NULL
```

If we define a mapping in `ggplot`, then all the geometries that are added as layers will default to this mapping. We redefine `p`:

```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb))
```

and then we can simply use code like this:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 1.5)
```

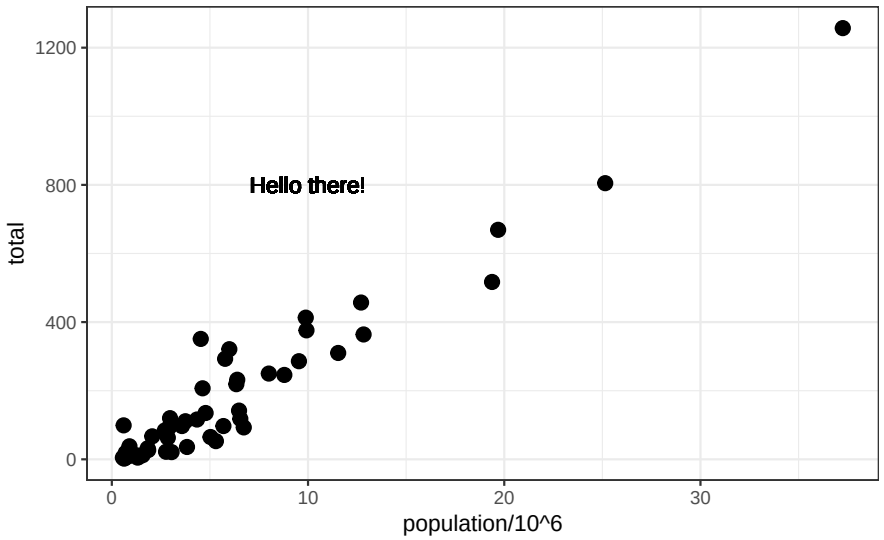


We keep the `size` and `nudge_x` argument in `geom_point` and `geom_text` respectively because we only want to increase the size of points and nudge only the labels. Also note that the `geom_point` function does not need a `label` argument and therefore ignores it.

1.7.6.1 Local aesthetic mappings overide global ones

If we need to, we can override the global mapping by defining a new mapping within each layer. These *local* definitions overrides the *global*. Here is an example:

```
p + geom_point(size = 3) +
  geom_text(aes(x = 10, y = 800, label = "Hello there!"))
```

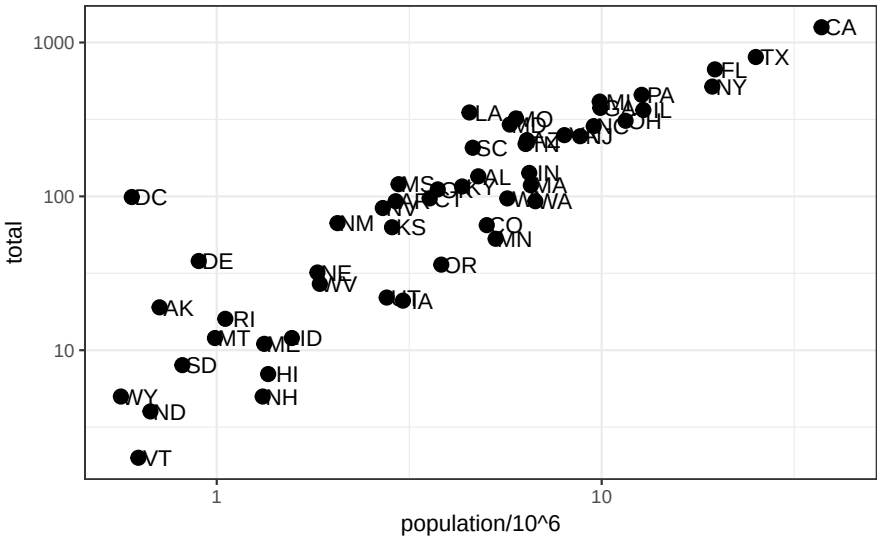


Clearly, the second call to `geom_text` does not use the `population` and `total`.

1.7.7 Scales

First, our desired scales are in log-scale. This is not the default so this change needs to be added through a *scales* layer. A quick look at the cheat sheet reveals the `scale_x_continuous` let's use edit the behavior of scales. We use them like this:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```



Because we are in the log-scale now, the *nudge* must be made smaller.

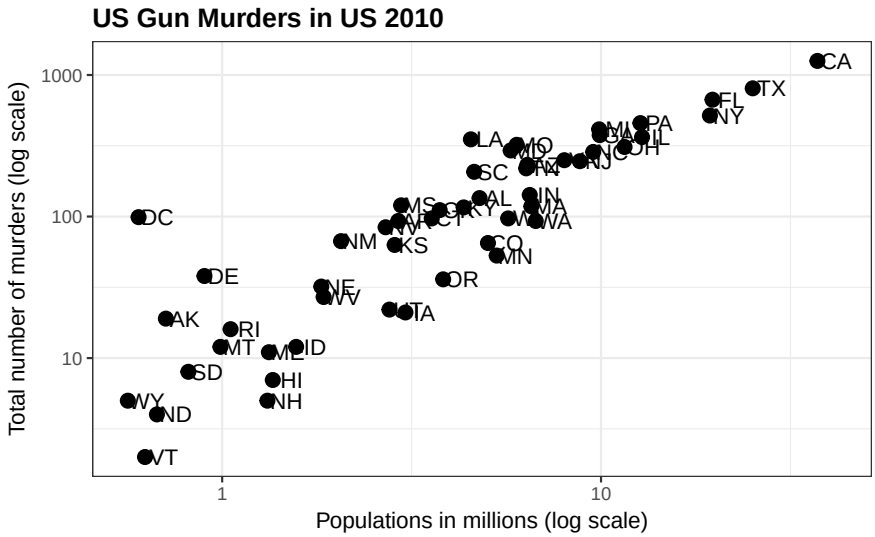
This particular transformation is so common that `ggplot` provides specialized functions:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10()
```

1.7.8 Labels and Titles

Similarly, the cheat sheet quickly reveals that to change labels and add a title we use the following functions:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in US 2010")
```



We are almost there! All we have to do is color, a legend and optional changes to the style.

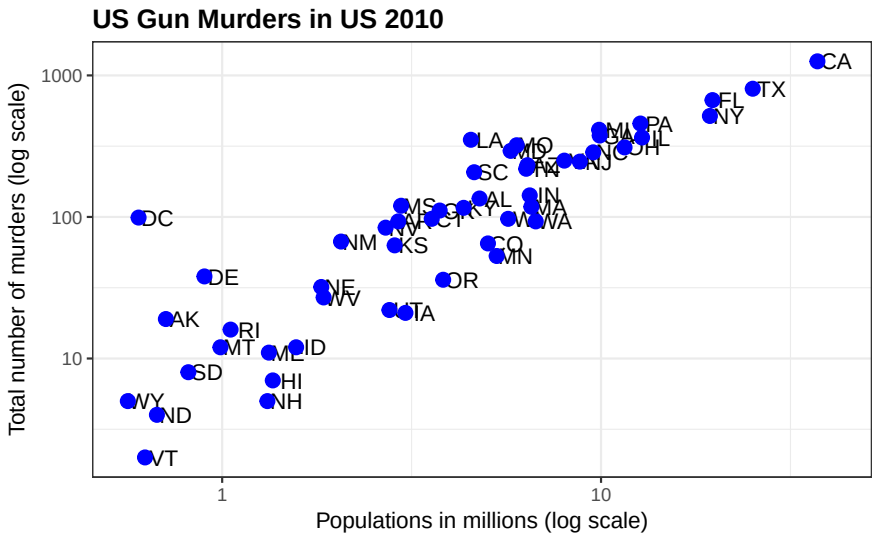
1.7.8.1 Categories as colors

Note that we can change the color of the points using the `col` argument in the `geom_point` function. To facilitate exposition we will redefine `p` to be everything except the points layer:

```
p <-  murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in US 2010")
```

and then test out what happens by adding different calls to `geom_point`. We can make all the points blue by adding the `color` argument

```
p + geom_point(size = 3, color ="blue")
```

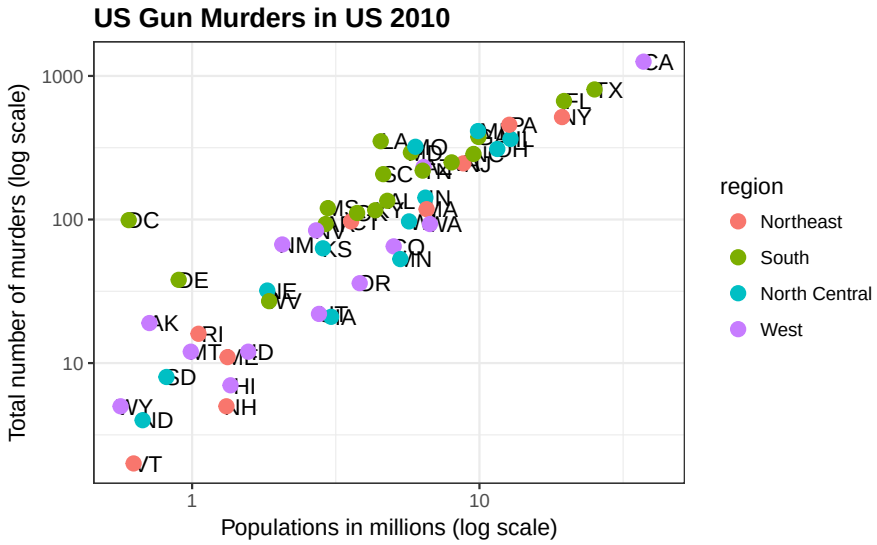


This, of course, is not what we want. We want to assign color depending on the geographical region. A nice

default behavior of `ggplot2` is that if we assign a categorical variable to color, it automatically assigns a different color to each category. It also adds a legend!

To map each point to a color, we need to use `aes` since this mapping. So we use the following code:

```
p + geom_point(aes(col=region), size = 3)
```



The `x` and `y` mappings are inherited from those already defined in `p`. So we do not redefine them. We also move `aes` to the first argument since that where the mappings are expected in this call.

Here we see yet another useful default behavior: `ggplot2` has automatically added a legend that maps color to region.

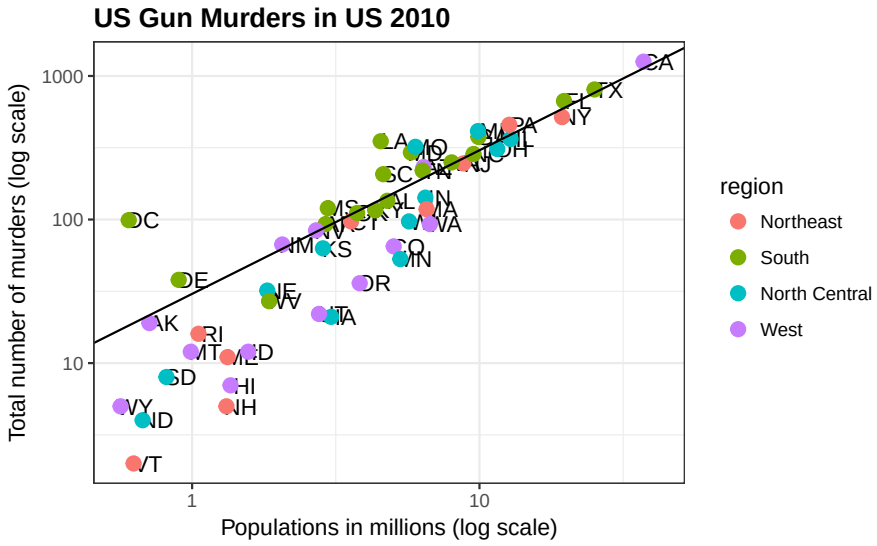
1.7.9 Adding a line

We want to add a line that represents the average murder rate for the entire country. Note that once we determine the per million rate to be r, this line is defined by the formula: $y = rx$ with y and x our axes: total murders and population in millions respectively. In the log-scale this line turns into: $\log(y) = \log(r) + \log(x)$. So in our plot it's a line with slope 1 and intercept $\log(r)$. To compute this value we use what we our `dplyr` skills:

```
r <- murders %>%
  summarize(rate = sum(total) /  sum(population) * 10^6) %>% .$rate
```

To add a line we use the `geom_abline` function. `ggplot` uses `ab` in the name to remind us we are supplying the intercept (`a`) and slope (`b`). The default line has slope 1 and intercept 0 so we only have to define the intercept:

```
p + geom_point(aes(col=region), size = 3) +
  geom_abline(intercept = log10(r))
```



We can change the line type and color of the lines using arguments. Also we draw it first so it doesn't go over our points.

```
p <- p + geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3)
```

Note that we redefined `p`

1.7.10 Other adjustements

The default plots created by `ggplot` are already very useful. But often, we need to make minor tweaks to the default behavior. Although it is not always obvious how to make these even with the cheat sheet, `ggplot2` is very flexible.

For example, note that we can make changes to the legend via the `scale_color_discrete` function. For example, in our plot the word *region* is capitalized and we can change like this:

```
p <- p + scale_color_discrete(name = "Region")
```

1.7.11 Add-on packages

The power of `ggplot2` is augmented further due to the availability of add-on packages. The remaining changes required to put the finishing touches on our plot require the `ggthemes` and `ggrepel` packages.

The style of a `ggplot` graph can be changed using the `theme` functions. Several themes are included as part of the `ggplot2` package. In fact, for most of the plots in this book we use a function in the `dslabs` package that automatically sets a default theme:

```
ds_theme_set()
```

Many other themes added by the package `ggthemes`. Among those are the `theme_economist` theme that we used. After installing the package, you can change the style of by adding a layer:

```
library(ggthemes)
p + theme_economist()
```

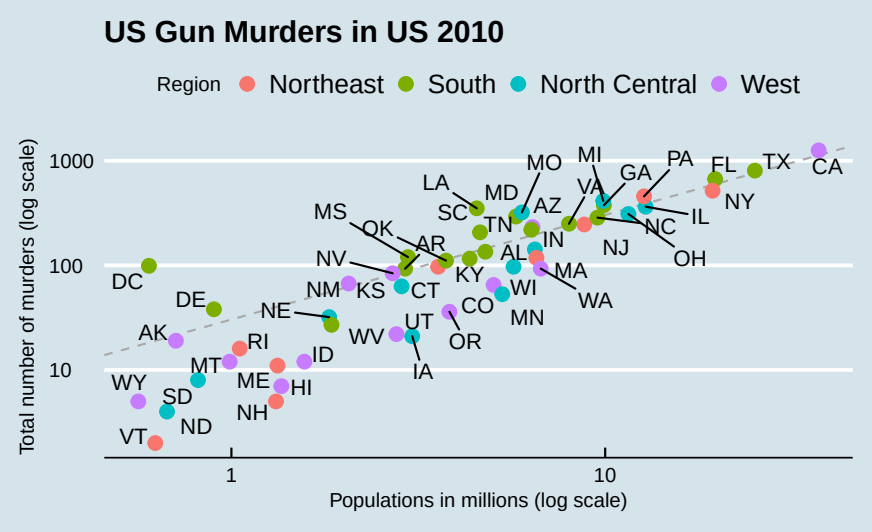
You can see how some of the other themes look like by simply changing the function. For example you might try the `theme_fivethirtyeight()` theme instead.

The final difference has to do with the position of the labels. Note that in our plot, some of the labels fall on top of each other. The an add-on package `ggrepel` includes a geometry that adds labels ensuring that they don't fall on top of each other. We simple change `geom_text` with `geom_text_repell`.

1.7.12 Putting it all together

So now that we are done testing we can write one piece of code that produces our desired plot from scratch.

```
library(ggthemes)
library(ggrepel)

### First define the slope of the line
r <- murders %>%
  summarize(rate = sum(total) /  sum(population) * 10^6) %>% .$rate

## Now make the plot
murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) +
  geom_text_repel() +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in US 2010") +
  scale_color_discrete(name = "Region") +
  theme_economist()
```



1.7.13 Other examples

Now let's try to make the summary plots we have described.

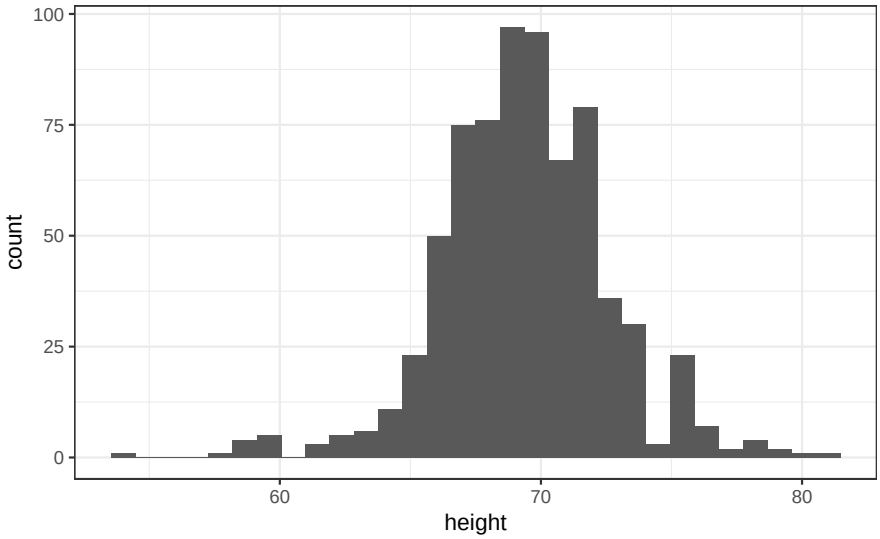
1.7.13.1 Histogram

Let's start with the histogram. In a first step we need to use `dplyr` to filter the data:

```
heights %>% filter(sex=="Male")
```

Now that we have a dataset, the next step is deciding what geometry we need. If you guessed `geom_histogram` you guessed right. Looking at the help file for this function we learn that the only required argument is `x`, the variable for which we will construct a histogram. So the code looks like this:

```
p <- heights %>%
  filter(sex=="Male") %>%
  ggplot(aes(x = height))

p + geom_histogram()
```



As before, we can drop the `x =`.

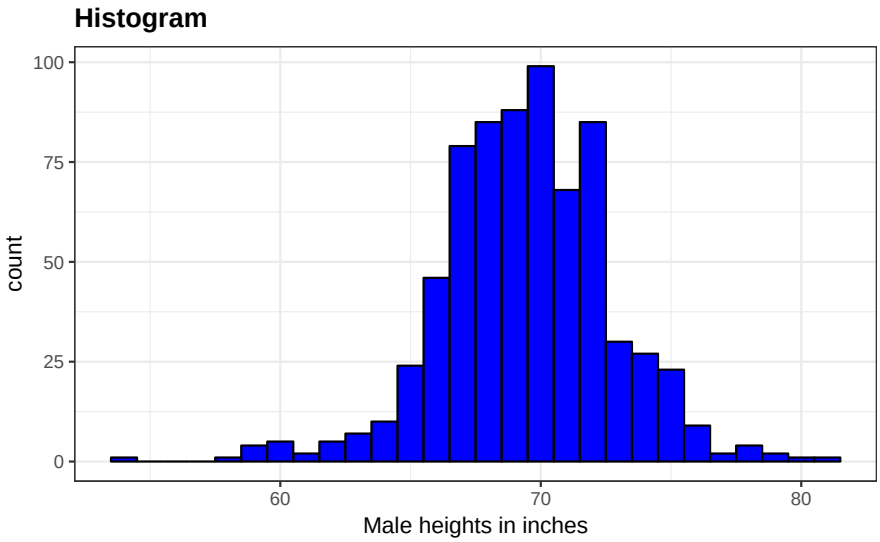
Note that this call gives us a message:

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

We previously used a bin size of 1 inch, so the code looks like this:

```
p + geom_histogram(binwidth = 1)
```

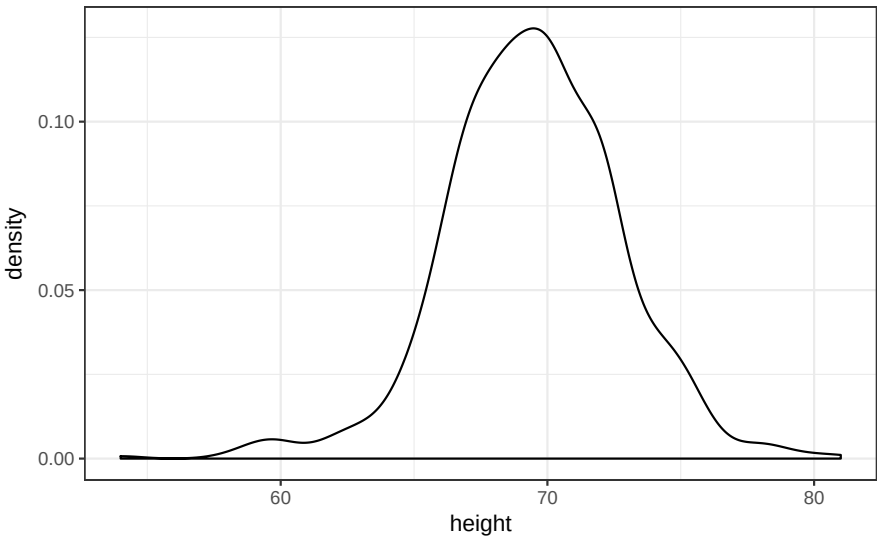
Finally, if we want to add color for aesthetic reasons we use the arguments described in the help file. We also add labels and a title:



1.7.13.2 Density

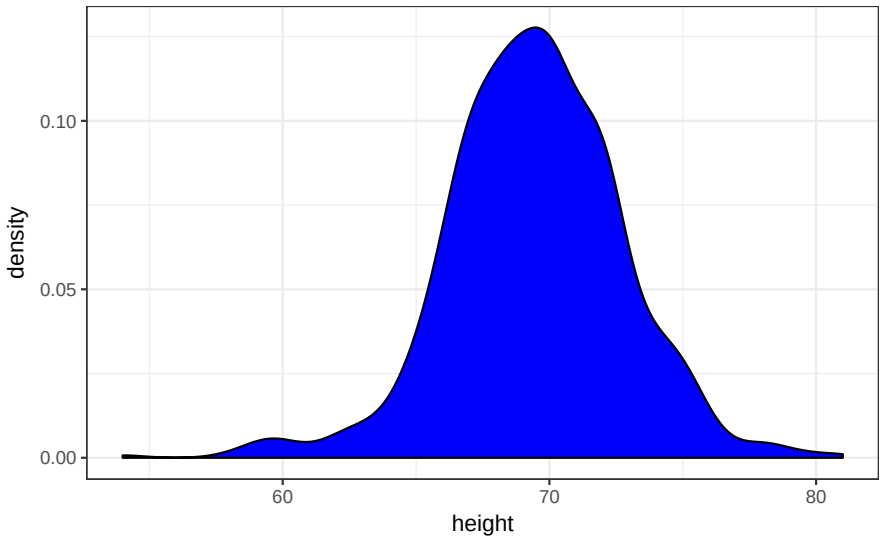
To create a smooth density we need a different geometry: we used `geom_density` instead.

```
p + geom_density()
```



Do fill in with color we can use the `fill` argument.

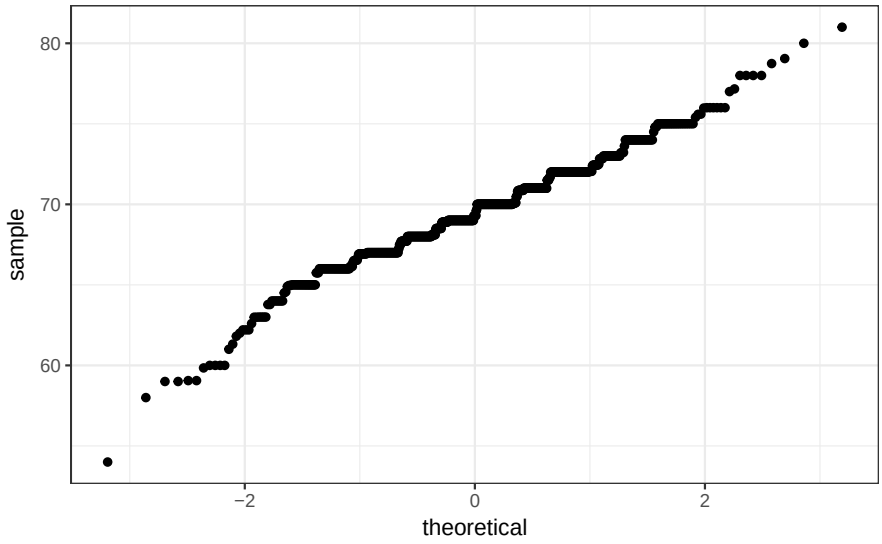
```
p + geom_density(fill="blue")
```



1.7.13.3 QQ-plots

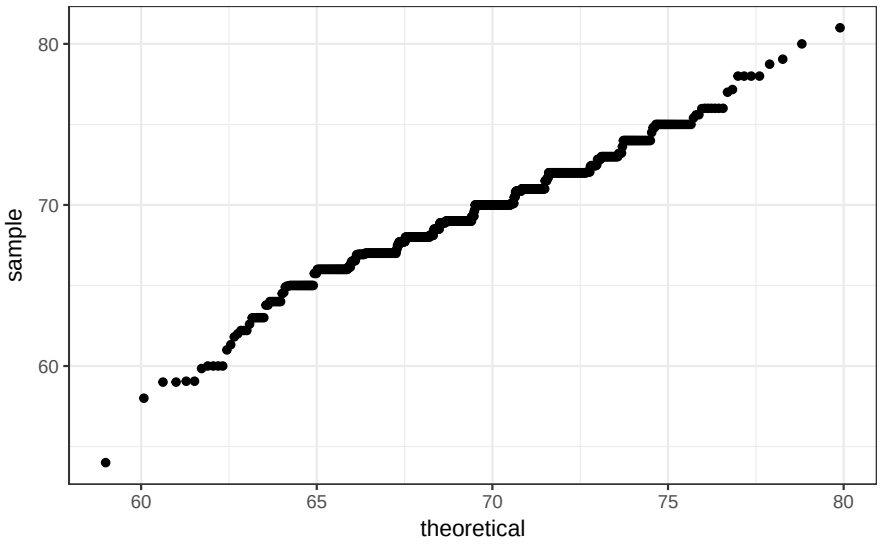
For qq-plots we use the `geom_qq` geometry. From the help file we learn that we need to specify the `sample` (we will learn about samples in a later chapter)

```
p <- heights %>% filter(sex=="Male") %>%
  ggplot(aes(sample = height))
p + geom_qq()
```



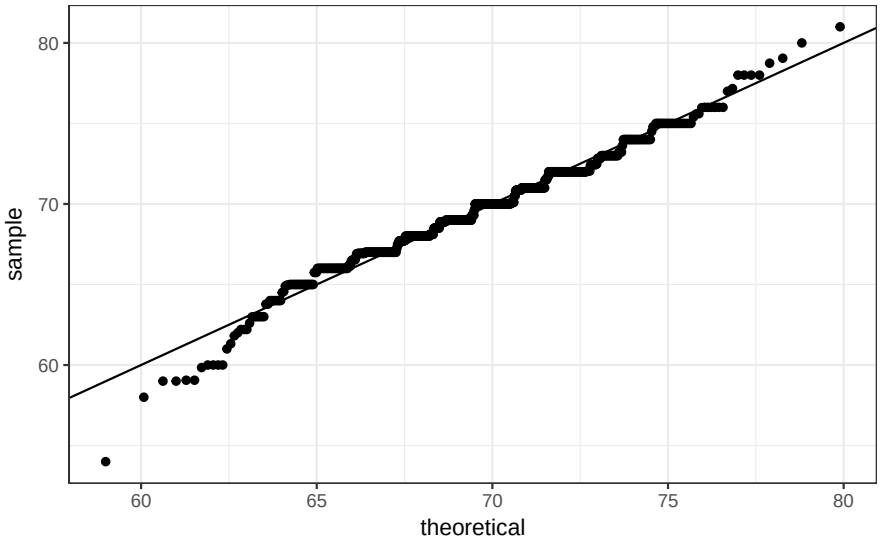
By default it is compared to the normal distribution with average 0 and standard deviation 1. To change this, again from the help file, we use the `dparams` arguments.

```
params <- heights %>% filter(sex=="Male") %>%
  summarize(mean = mean(height), sd = sd(height))

p  +  geom_qq(dparams = params)
```



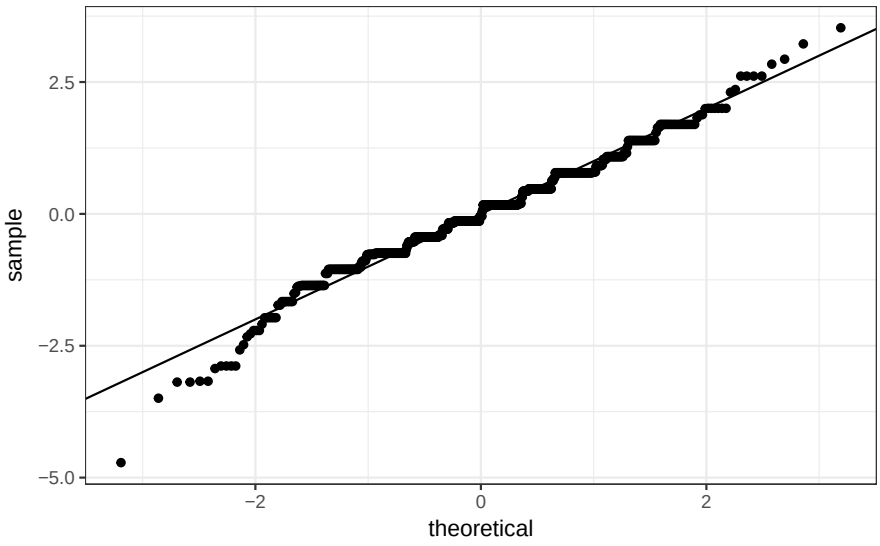
Adding an identity line is a s simple as assign another layer. For straight lines we use the `geom_abline` function. To help you remember the name the `ab` is to remind us that we need to supply an intercept (a) and slope (b) to draw the line $y = a + bx$. The default is the identity `a=0` and `b=1`

```
p +  geom_qq(dparams = params) +
  geom_abline()
```



Another option here is to scale the data first and the make a qqplot against the standard normal

```
heights %>% filter(sex=="Male") %>%
  ggplot(aes(sample = scale(height))) +
  geom_qq() +
  geom_abline()
```



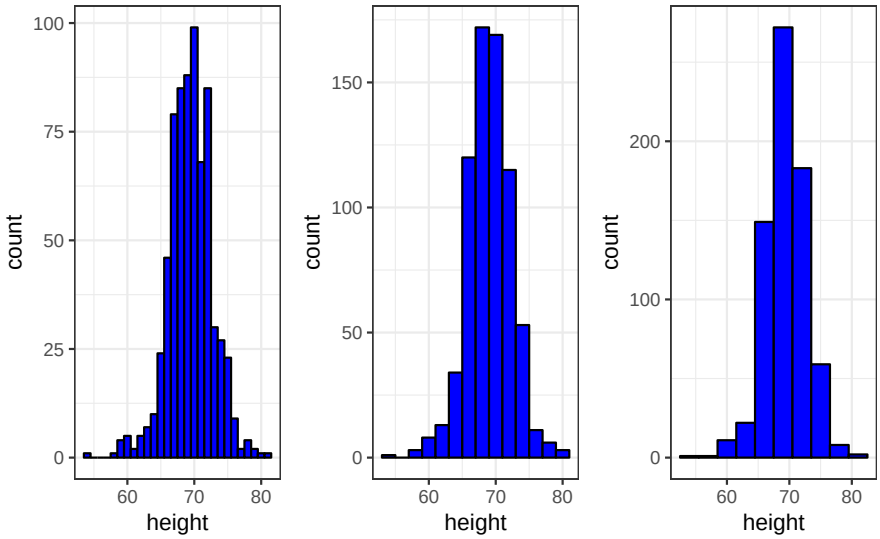
1.7.14 Grids of plots

There are often reasons to graph plots next to each other. The `gridExtra` package permits us to do that:

```
p <- heights %>% filter(sex=="Male") %>% ggplot(aes(x = height))
p1 <- p + geom_histogram(binwidth = 1, fill = "blue", col="black")
p2 <- p + geom_histogram(binwidth = 2, fill = "blue", col="black")
p3 <- p + geom_histogram(binwidth = 3, fill = "blue", col="black")
```

To print them all side-by-side, we can use the function `grid.arrange` in the `gridExtra` package:

```
library(gridExtra)
grid.arrange(p1,p2,p3, ncol = 3)
```



1.7.15 Summary

We will learn a lot more ggplot through practice and examples.

1.8 Case Study: Trends in world health and economics

In this section we will demonstrate how relatively simple `ggplot` code can create insightful and aesthetically pleasing plots that help us better understand trends in world health and economics. We later augment the code somewhat to perfect the plots and describe some general principles to guide for data visualization.

1.9 Example 1: Life Expecacty and Fertility Rates

Hans Rosling was the co-founder of the Gapminder Foundation, an organization dedicated to educating the public by using data to dispel common myths about the so-called developing world. The organization uses data to show how actual trends in health and economics contradict the narratives that emanate from sensationalist media coverage of catastrophes, tragedies and other unfortunate events. As stated in the Gapminder Foundation's website

> Journalists and lobbyists tell dramatic stories. That's their job. They tell stories about extraordinary events and unusual people. The piles of dramatic stories pile up in peoples' minds into an over-dramatic worldview and strong negative stress feelings: "The world is getting worse!", "It's we vs. them!" , "Other people are strange!", "The population just keeps growing!" and "Nobody cares!"

Hans Rosling conveyed actual data-based trends in a dramatic way of his own, using effective data visualization. This section is based on two talks that exemplify this approach to education: New Insights on Poverty and The Best Stats You've Ever Seen. Specifically, in this section, we set out to answer the following two questions using data:

1. Is it a fair characterization of today's world to say it is divided into western rich nations and the developing world in Africa, Asia and Latin America?
2. Has income inequality across countries worsened during the last 40 years?

To answer these question we will be using the `gapminder` dataset provided in `dslabs`. This dataset was created using a number of spreadsheets available from the Gapminder Foundation. You can access the table like this:

```
library(dslabs)
data(gapminder)
head(gapminder)
#>               country year infant_mortality life_expectancy fertility
#> 1             Albania 1960            115.4           62.9      6.19
#> 2             Algeria 1960            148.2           47.5      7.65
#> 3              Angola 1960            208.0           36.0      7.32
#> 4 Antigua and Barbuda 1960               NA           63.0      4.43
#> 5           Argentina 1960             59.9           65.4      3.11
#> 6             Armenia 1960               NA           66.9      4.55
#>   population      gdp continent          region
#> 1    1636054       NA    Europe Southern Europe
#> 2   11124892 1.38e+10    Africa Northern Africa
#> 3    5270844       NA    Africa   Middle Africa
#> 4      54681       NA  Americas       Caribbean
#> 5   20619075 1.08e+11  Americas   South America
#> 6    1867396       NA      Asia    Western Asia
```

1.9.0.1 Hans Rosling's Quiz

As done in the *New Insights on Poverty* video, we start by testing our knowledge regarding differences in child mortality across different countries.

For each of the six pairs of countries below, which country do you think had the highest child mortality in 2015? Which pairs do you think are most similar?

1. Sri Lanka or Turkey
2. Poland or South Korea
3. Malaysia or Russia
4. Pakistan or Vietnam
5. Thailand or South Africa

When answering these questions without data, the non-European countries are typically picked as having higher mortality rates: Sri Lanka over Turkey, South Korea over Poland, and Malaysia over Russia. It is also common to assume that countries considered to be part of the developing world, Pakistan, Vietnam, Thailand and South Africa, have similarly high mortality rates.

To answer these questions **with data** we can use `dplyr`. For example for the first comparison we see that

```
library(tidyverse)
gapminder %>%
  filter(year == 2015 & country %in% c("Sri Lanka","Turkey")) %>%
  select(country, infant_mortality)
#>     country infant_mortality
#> 1 Sri Lanka              8.4
#> 2    Turkey             11.6
```

Turkey has the higher rate.

We can use this code on all comparisons and find the following:

country	infant_mortality	country1	infant_mortality1
Sri Lanka	8.4	Turkey	11.6
Poland	4.5	South Korea	2.9
Malaysia	6.0	Russia	8.2
Pakistan	65.8	Vietnam	17.3
Thailand	10.5	South Africa	33.6

We see that the European countries have higher rates: Poland has a higher rate than South Korea, and Russia has a higher rate than Malaysia. We also see that Pakistan has a much higher rate than Vietnam and South Africa a much higher rate than Thailand. It turns out that most people do worse than if they were guessing, which implies we are more than ignorant, we are misinformed.

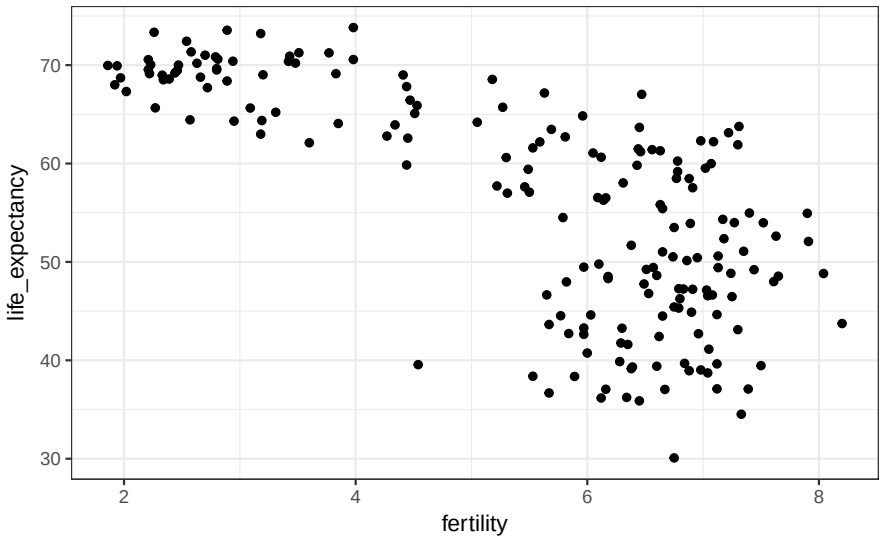
1.9.1 Life expectancy and fertility rates

The reason for this stems from the preconceived notion that the world is divided into two groups: the western world (Western Europe and North America), characterized by long life spans and small families, versus the developing world (Africa, Asia, and Latin America) characterized by short life spans and and large families. But, does the data support this dichotomous view of two groups?

The necessary data to answer this question is also available in our `gapminder` table. Using our newly learned data visualization skills we will be able to answer this question.

The first plot we make to see what data have to say about this world view is a scatter plot of life expectancy versus fertility rates (average number of children per woman). We will start by looking at data from about 50 years ago, when perhaps this view was cemented in our minds.

```
ds_theme_set()

filter(gapminder, year==1962) %>%
  ggplot( aes(fertility, life_expectancy)) +
  geom_point()
```

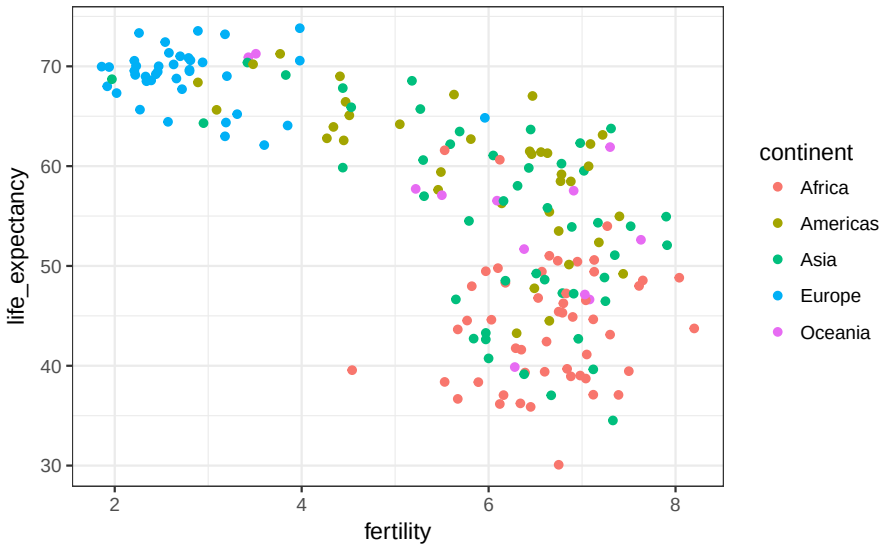


Most points fall into two distinct categories:

1. Life expectancy around 70 years and 3 or less children per family
2. Life expectancies lower then 65 years and with more than 5 children per family.

To confirm that indeed these countries are from the regions we expect, we can use color to represent continent.

```
filter(gapminder, year==1962) %>%
  ggplot( aes(fertility, life_expectancy, color = continent)) +
  geom_point()
```



So in 1962, "the west versus developing world" view was grounded in some reality. But is this still the case 50 years later?

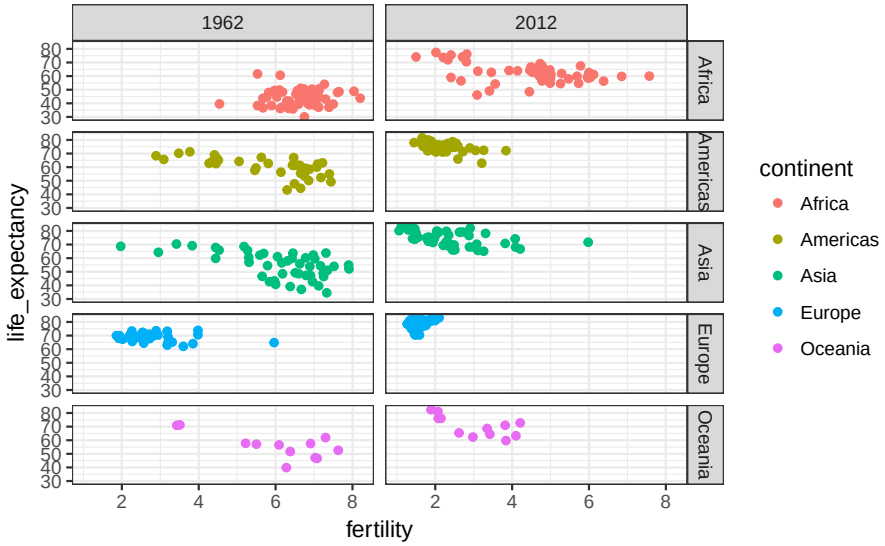
1.9.2 Faceting

We could easily plot the 2012 data in the same way we did for 1962. But to compare, side by side plots are preferable. In `ggplot` we can achieve this by *faceting variables*: we stratify the data by some variable and make the same plot for each strata.

To achieve faceting we add a layer with the function `facet_grid`, which automatically separates the plots.

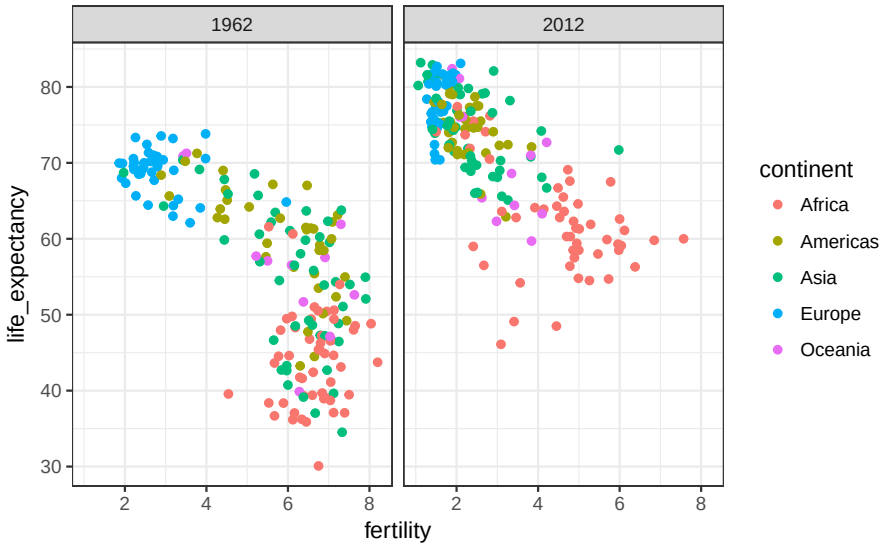
This function lets you facet by up to two variable using columns to represent one variable and rows to represent the other. The function expects the row and column variables separated by a `~`. Here is an example of a scatter plot with `facet_grid` added as the last layer:

```
filter(gapminder, year%in%c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid(continent~year)
```



We see a plot for each continent/year pair. However, this is just an example, and more than what we want, which is simply to compare 1962 and 2012. In this case, there is just one variable and we use `.` to let facet know that we not using one of the variable :

```
filter(gapminder, year%in%c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid( . ~ year)
```



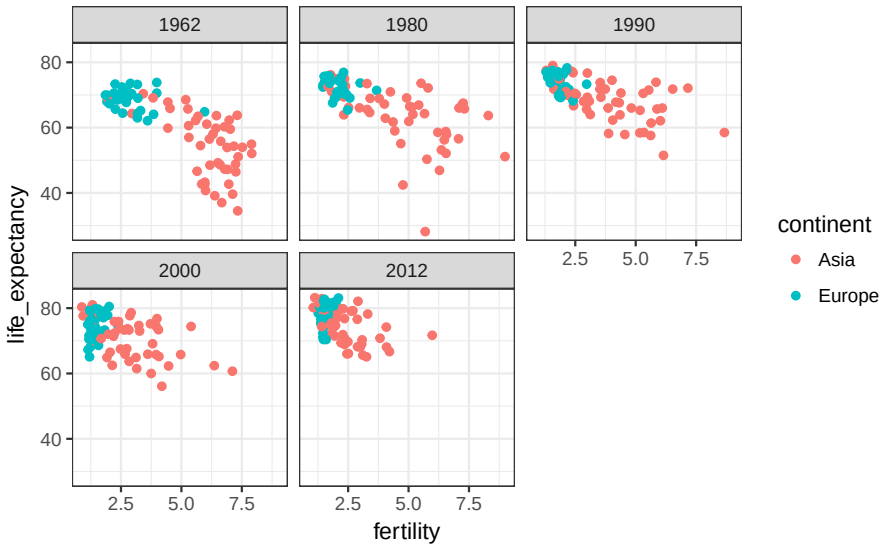
This plot clearly shows that the majority of countries have moved from the *developing world* cluster to the *western world* one. In 2012, the western versus developing world view no longer makes sense. This is particularly clear when comparing Europe to Asia, which includes several countries that have made great

improvements.

1.9.2.1 `facet_wrap`

To explore how this transformation happened through the years, we can make the plot for several years. For example we can add 1970, 1980, 1990, 2000. If we do this, we will not want all the plots on the same row, the default behavior of `facet_grid`, as they will become to thin to show the data. Instead we will want to use multiple rows and columns. The function `facet_wrap` permits us to do this, as it automatically wraps the series of plots so that each displays has view-able dimensions:

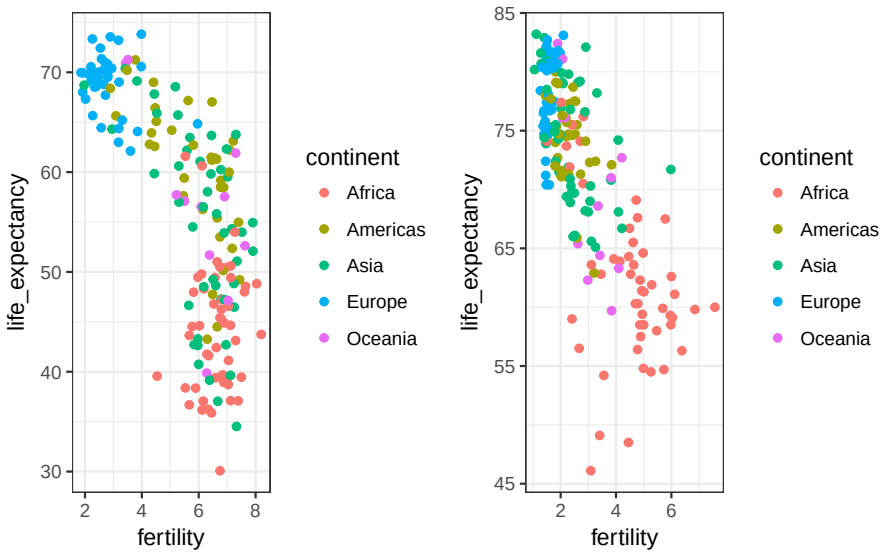
```
years <- c(1962, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia")
gapminder %>%
  filter(year %in% years & continent %in% continents) %>%
  ggplot( aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_wrap(~year)
```



This plot clearly shows how most Asian countries have improved at a much faster rate than European ones.

1.9.3 Fixed scales for better comparisons

Note that the default choice of the range of the axes is an important one. When not using `facet`, this range is determined by the the data shown in the plot. When using `facet`, this range is determined by the data shown in all plots and therefore kept fixed across plots. This makes comparisons across plots much easier. For example, in the plot above we see that life expectancy has increased and the fertility has decreased across most countries. We see this because the cloud of points moves. This is not the case if we adjust the scales:



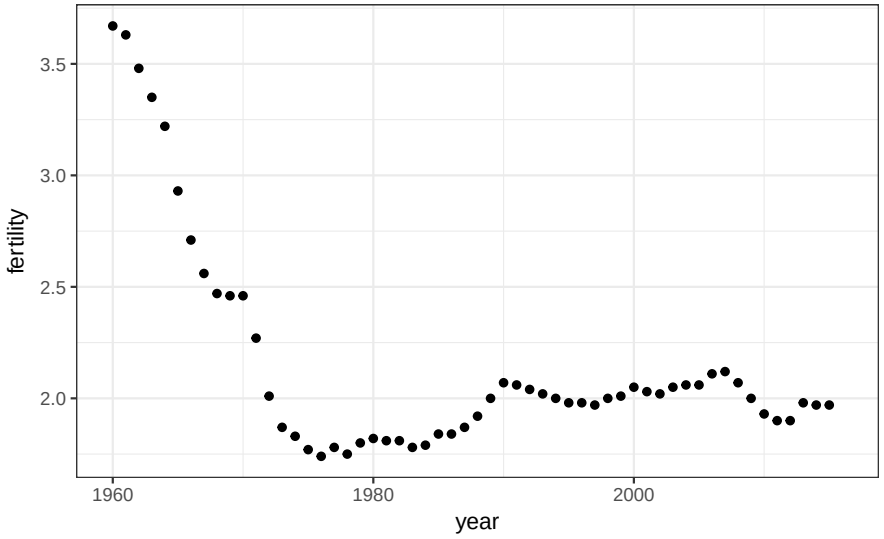
In the plot above we we have to pay special attention to the range to notice that the plot on the right has larger life expectancy.

1.9.4 Time Series Plots

The visualizations above effectively illustrates that data no longer support the western versus developing world view. Once we see these plots new questions emerge. For example, which countries are improving more, which ones less? Was the improvement constant during the last 50 years or was there more accelerated during certain periods? For a closer look that may help answer these question, we intrude *time series plots*.

Time series plots have time in the x-axis and an outcome or measurement of interest on the y-axis. For example, here is a trend plot for the United States fertility rates:

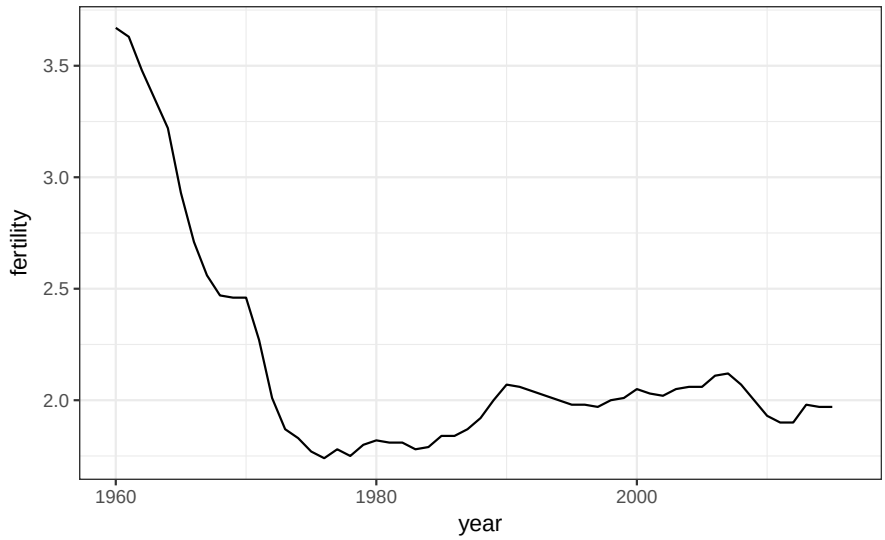
```
gapminder %>% filter(country == "United States") %>%
  ggplot(aes(year,fertility)) +
  geom_point()
```



We see that the trend is not linear at all. Instead we see a sharp drop during the 60s and 70s to below 2. Then the trend comes come back to 2 and stabilizes during the 90s.

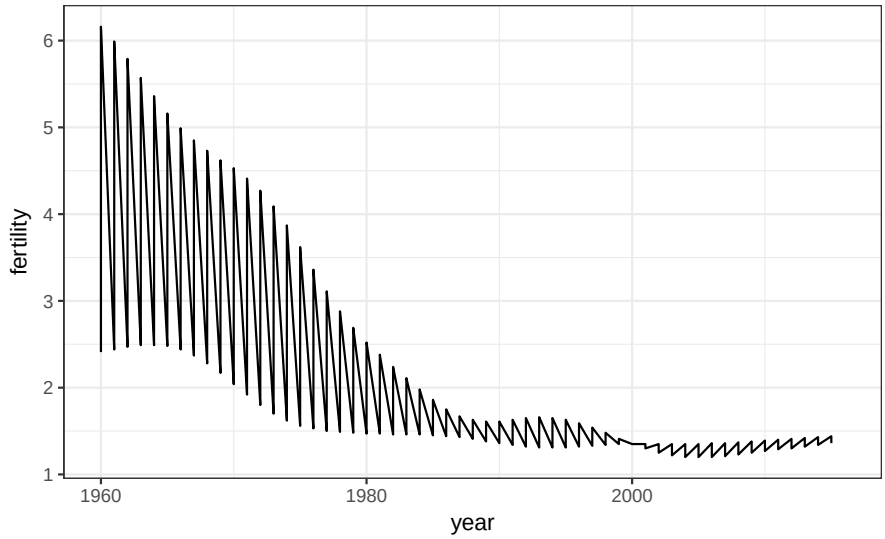
When the points are regularly and densly spaced, as they are here, we create curves by joining the points with lines, to convey that these data are from a single country. To do this we use the `geom_line` function instead of `geom_point`.

```
gapminder %>% filter(country == "United States") %>%
  ggplot(aes(year,fertility)) +
  geom_line()
```



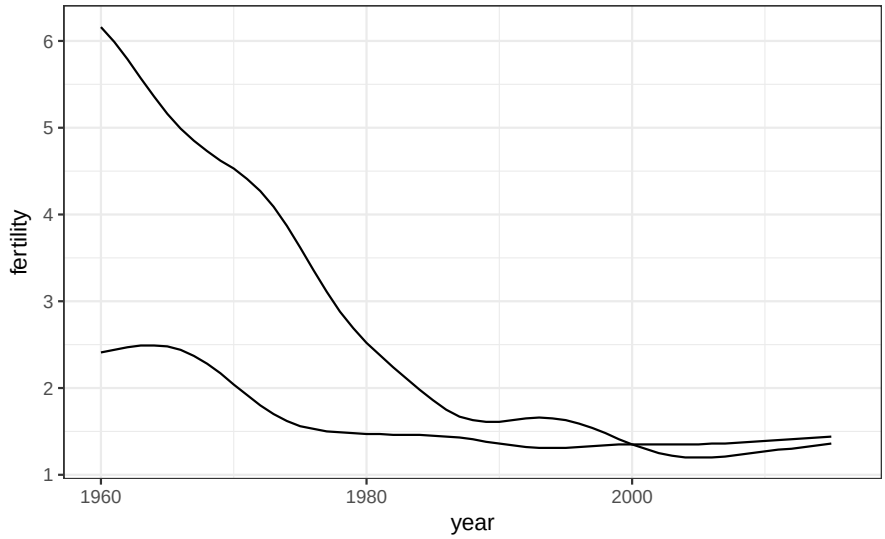
This is particularly helpful when we look a two countries. If we subset the data to include two countries, one from European and one from Asia, then copy to code above:

```
countries <- c("South Korea","Germany")
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year,fertility)) +
  geom_line()
```



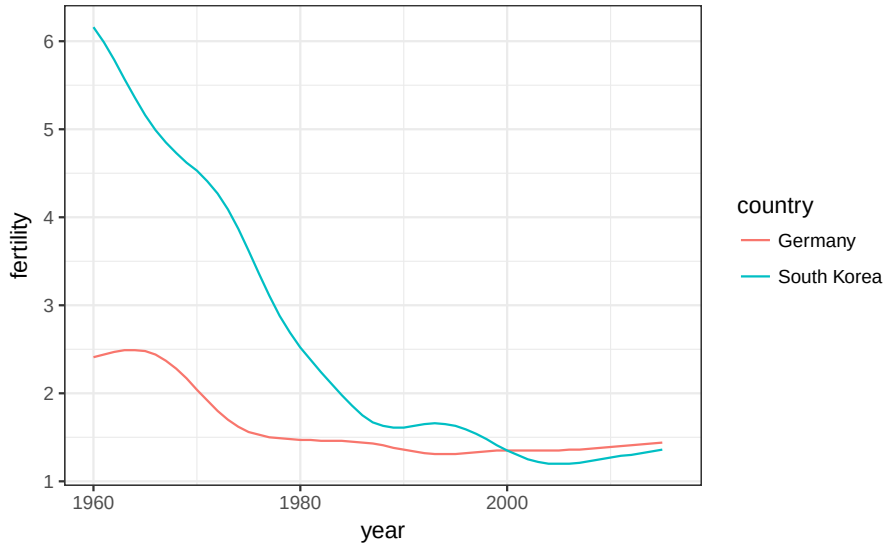
Note that this is **not** the plot that we want. Rather than a line for each country, the points for both countries are joined. This is actually expected since we have not told `ggplot` anything about wanting two separate lines. To let `ggplot` know that there are two curves that need to be made separately, we assign each point to a `group`, one for each country:

```
countries <- c("South Korea","Germany")
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year,fertility, group = country)) +
  geom_line()
```



But which line goes with which country? We can assign colors to make this distinction. A useful side-effect of using the `color` argument to assign different colors to the different countries is that the data is automatically grouped:

```
countries <- c("South Korea","Germany")
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year,fertility, col = country)) +
  geom_line()
```



The plot clearly shows how South Korea's fertility rate dropped drastically during the 60s and 70s and by 1990 had a similar rate to Germany.

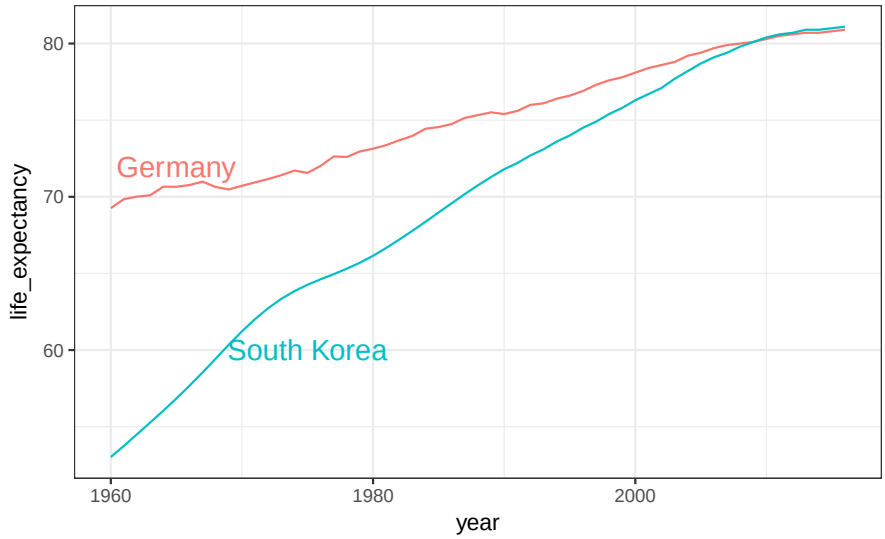
1.9.4.1 We prefer labels over legends

For trend plots we recommend labeling the lines rather than using legends as the viewer can quickly see

which line is which country. This suggestion actually applies to most plots: labeling is usually preferred over legends.

We demonstrate how we can do this using the life expectancy data. We define a data table with the label locations and then use a second mapping just for these labels:

```
labels <- data.frame(country = countries, x = c(1975,1965), y = c(60,72))

gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, life_expectancy, col = country)) +
  geom_line() +
  geom_text(data = labels, aes(x, y, label = country), size = 5) +
  theme(legend.position = "none")
```



The plot clearly shows how an improvement in life expectancy followed the drops in fertility rates. While in 1960 Germans lived more than 15 years more South Koreans, by 2010 the gap is completely closed. It exemplifies the improvement that many non-western countries have achieved in tha last 40 years.

1.9.5 Example 2: Income Distribution

Another commonly held notions is that wealth distribution across the world has become worse during the last decades. When general audiences are asked if poor countries have become poorer and rich countries become richer, the majority answers yes. By using stratification, histograms, smooth densities, and boxplots we will be able to understand if this is in fact the case. We will also learn how transformations can sometimes help provide more informative summaries and plots.

1.9.6 Transformations

The `gapminder` data table includes a column with the countries gross domestic product (GDP). GDP measure the market value of goods and services produced by a country in a year. The GDP per person is often used as a rough summary of how rich a country is. Here we divide this quantity by 365 to obtain the more interpretable measure *dollars per day*. Using current US dollars as a unit, a person surviving on an income of less than $2 a day is defined to be living in *absloute povery*. We add this variable to the data table:

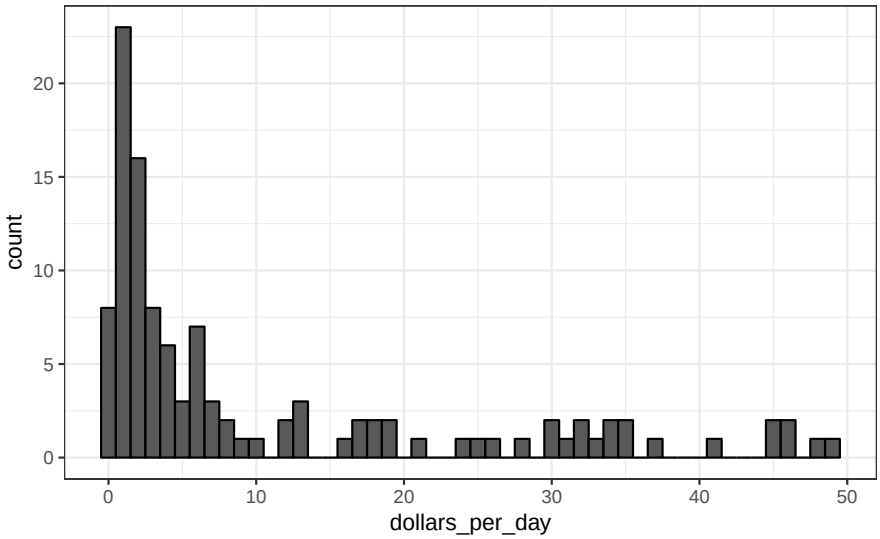
```
gapminder <- gapminder %>%
  mutate(dollars_per_day = gdp/population/365)
```

Note that the GDP values are adjusted for inflation and represent current US dollars, so these values are meant to be comparable across the years. Also note that these are country averages and that within each country there is much variability. All the graphs and insights described below relate to country averages not to individuals.

1.9.6.1 Country income distribution

Here is a histogram of per day incomes from 1970:

```
past_year <- 1970
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black")
```



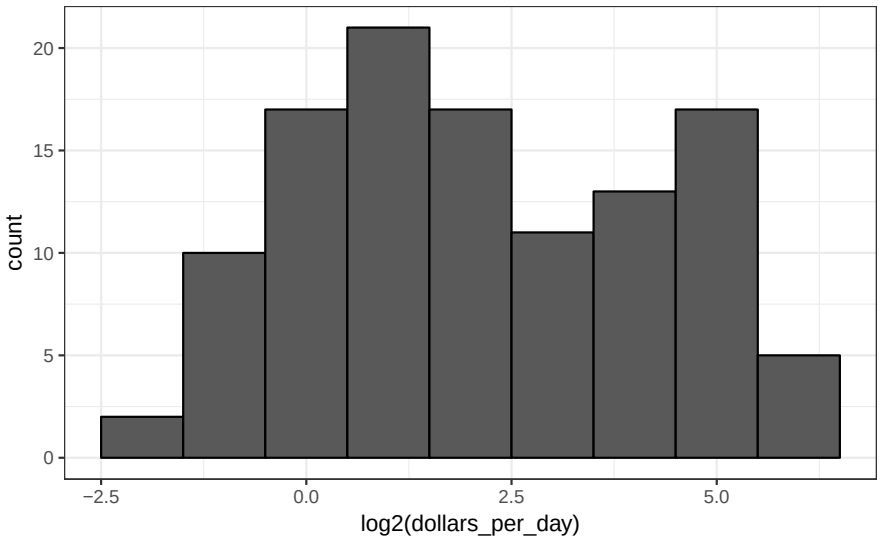
We use the `color = "black"` argument to draw a boundary and clearly distinguish the bins.

In this plot we see that for the majority of countries, averages are below $10 a day. However, the majority of the x-axis is dedicate to the 35 countries with averages above $10. So the plot is not very informative about countries with values below $10 a day.

It might be more informative to quickly be able to see how many countries have average daily incomes of about $1 (extremely poor), $2 (very poor), $4 (poor), $8 (middle), $16 (well off), $32 (rich), $64 (very rich) per day. These changes are multiplicative and log transformations change multiplicative changes into additive ones: when using base 2, a doubling of a value turns into an increase by 1.

Here is the distribution if we apply a log base 2 transform:

```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(log2(dollars_per_day))) +
  geom_histogram(binwidth = 1, color = "black")
```



In a way this provides a *close up* of the mid to lower income countries.

1.9.6.2 Which base?

In the case above we used base 2 in the log transformations. Other common choices are base e (the natural log) and base 10.

In general, we do not recommend using the natural log for data exploration and visualization. This is because while $2^2, 2^3, 2^4, \ldots$ or $10^1, 10^2, \ldots$ are easy to compute in our heads, the same is not true for $e^2, e^3, \ldots$.

In the dollars per day example, we used base 2 instead of base 10 because the resulting range is easier to interpret. The range of the values being plotted is 0.327, 48.885.

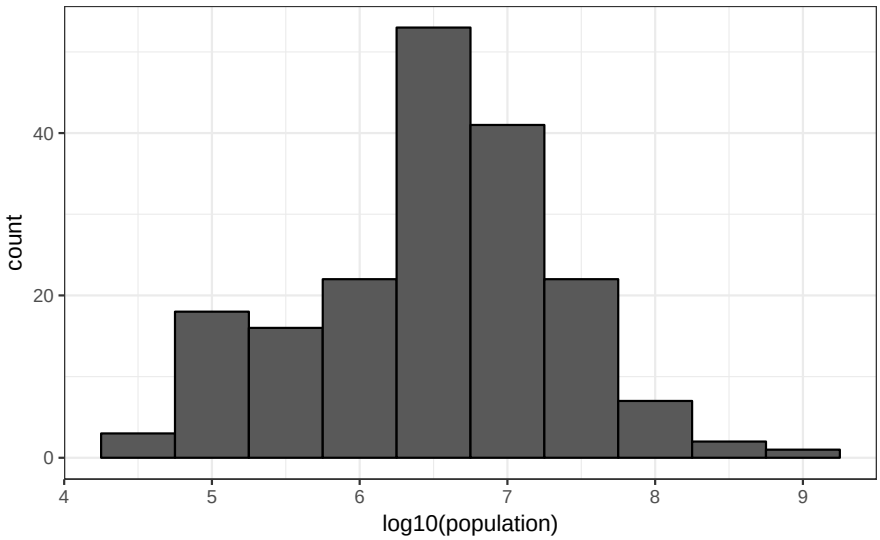
In base 10 this turns into a range that includes very few integers: just 0 and 1. With base two, our range includes -2, -1, 0, 1, 2, 3, 4 and 5. It is easier to compute 2^x and 10^x when x is an integer and between -10 and 10, so we prefer to have more small integers in the scale. Another consequence of a limited range is that choosing the binwidth is more challenging. With log base 2, we know that a binwidth of 1 will translate to a bin with range x to $2x$.

As an example in which base 10 makes more sense consider population sizes. A log base 10 makes more sense since the range for these is:

```
filter(gapminder, year == past_year) %>%
  summarize(min = min(population), max = max(population))
#>     min      max
#> 1 46075 8.09e+08
```

Here is the histogram of the transformed values:

```
gapminder %>% filter(year == past_year) %>%
  ggplot(aes(log10(population))) +
  geom_histogram(binwidth = 0.5, color = "black")
```



Here we quickly see that country populations range between ten thousand and ten billion.

1.9.6.3 Transform the values or the scale?

There are two ways we can use log transformations in plots. We can log the values before plotting them or use log scales in the axes. Both approaches are useful and have different strengths. If we log the data we can more easily interpret intermediate values in the scale. For example, if we see

—-1—-x—-2———3—-

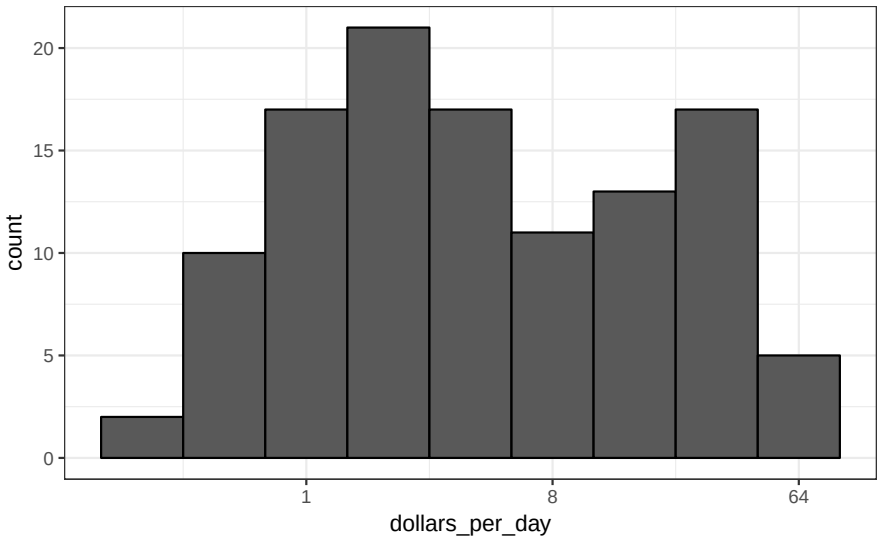
for log transformed data we know that the value of x is about 1.5. If the scales are logged

—-1—-x—-10——100—

then, to determine `x`, we need to compute $10^{1.5}$, which is not easy to do in our heads. The advantage of using logged scales is that we see the original values on the axes. However, the advantage of showing logged scales is that the original values are displayed in the plot, which are easier to interpret. For example, we would see "32 dollars a day" instead of "5 log base 2 dollar a day".

As we learned earlier, if we want to scale the axis with logs we can use the `scale_x_ccontinuous` function. So instead of logging the values first, we apply this layer:

```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2")
```



Note that the log base 10 transformation has it's own function: `scale_x_log10()`, but currently base 2 does not. Although we could easily define our own.

Note that there are other transformation avaiable through the `trans` argument. As we learn later on, the square root (`sqrt`) transformation, for example, is useful when considering counts. The logistic transformation (`logit`) is useful when plotting proportions between 0 and 1. The `reverse` transformation is useful when we want smaller values to be on the right or on top.

1.9.7 Modes

In statistics these bumps are sometimes referred to as *modes*. The mode of a distribution is the value with the highest frequency. The mode of the normal distribution is the average. When a distribution, like the one above, doesn't monotonically decrease from the mode we call the locations where it goes up and down again local modes and say that the distribution has multiple modes.

The histogram above suggest that the 1970 country income distribution has two modes: one at about 2 dollars per day (1 in the log 2 scale) and another at about 32 dollars per day (5 in the log 2 scale). This *bimodality* is consistent with a dichotomous world made up of countries with average incomes less than $8 (3 in the log 2 scale) a day and countries above that.

1.9.8 Stratify and boxplot

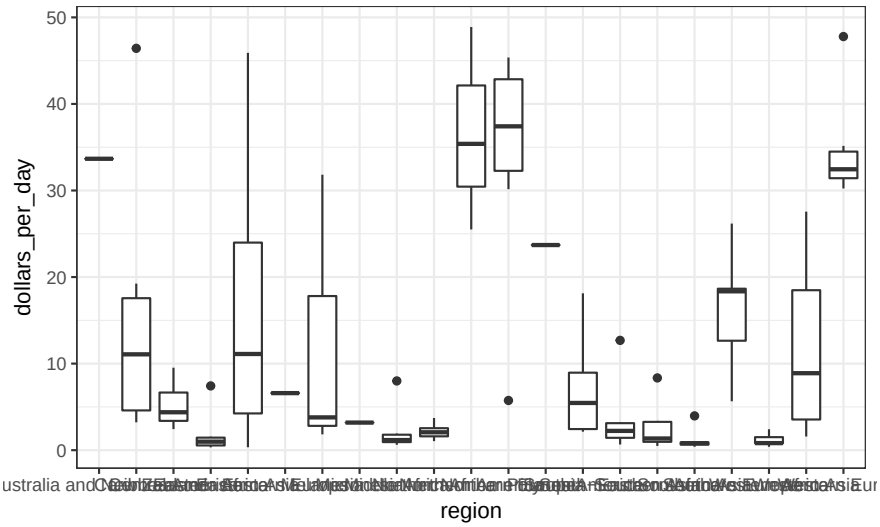
The histogram showed us that the income distribution values shows a dichotomy. However, the histogram does not show us if the two groups of countries are *west* versus the *developing* world.

To see distributions by geographical region we first stratify the data into regions, and then examine the distribution for each. Because of the number of regions

```
length(levels(gapminder$region))
#> [1] 22
```

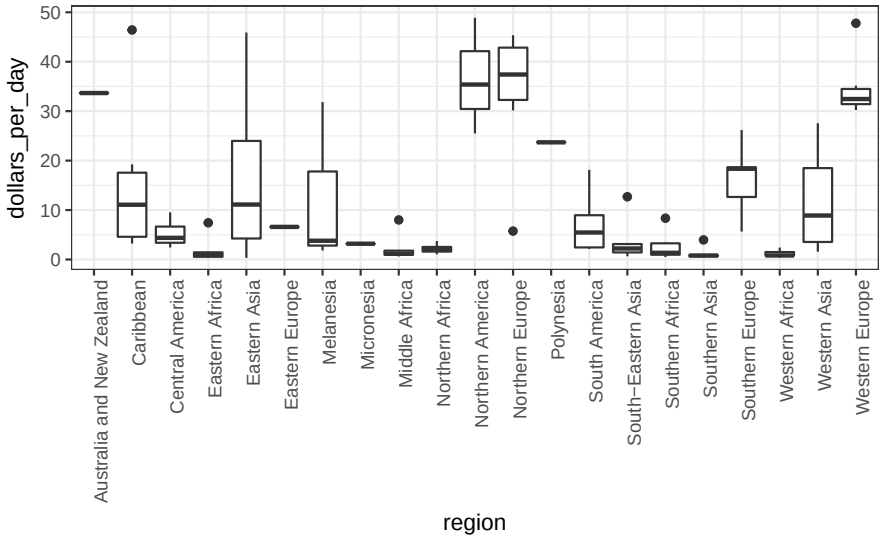
looking at histograms or smooth densities for each will not be useful. Instead, we can stack boxplots next to each other:

```
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(region, dollars_per_day))
p + geom_boxplot()
```



Note that we can't read the region names because the default gpplot behavior is to write the labels horizontally and, here, we run of of room. We can easily fix this by rotating the labels. Consulting the sheet cheat we find we can rotate the names by changing the `theme` through `element_text`. The `hjust=1` justifies the text so that it is next to the axis.

```
p + geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



We can already see that there is indeed a west versus the rest dichotomy.

1.9.8.1 Do not order alphabetically

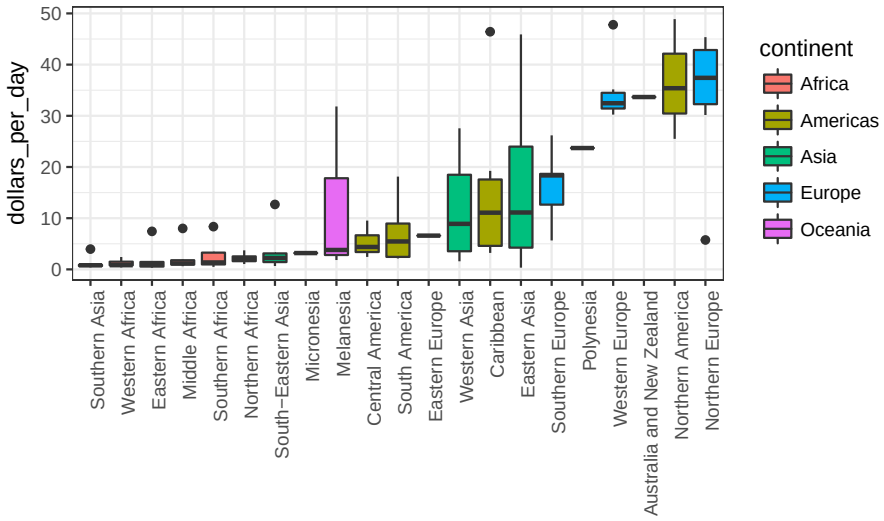
There are a few more adjustments we can make to this plot help uncover this reality. First, it helps to order the regions in the boxplots from poor to rich rather than alphabetically. This can be achieved using the `reorder` function. This function let's us change the order of the levels of a factor variable based on a summary computed on a numeric vector. A character vector gets coerced into a factor: Here is an example. Note how the order of levels change:

```
fac <- factor(c("Asia", "Asia", "West", "West", "West"))
levels(fac)
#> [1] "Asia" "West"
```

```
value <- c(10, 11, 12, 6, 4)
fac <- reorder(fac, value, FUN = mean)
levels(fac)
#> [1] "West" "Asia"
```

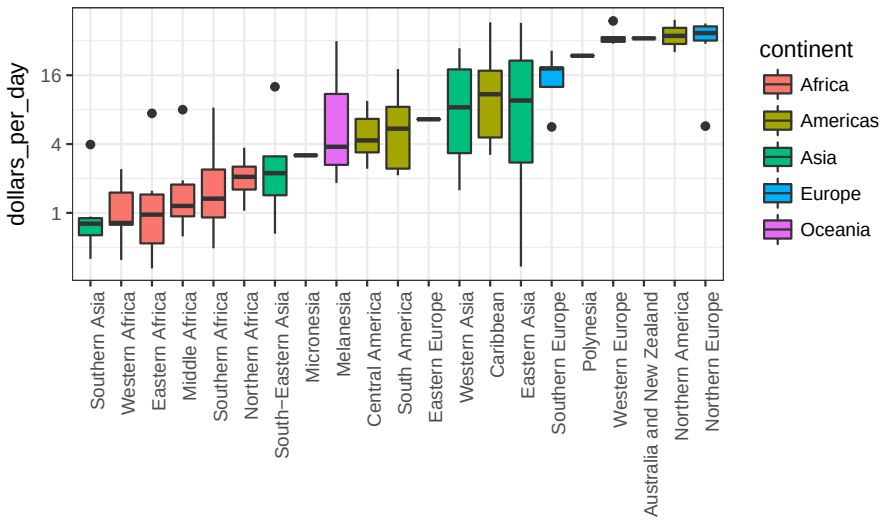
Second, we can use color to distinguish the different continents, a visual cue that helps find specific regions. Here is the code:

```
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
  ggplot(aes(region, dollars_per_day, fill = continent)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("")
p
```



This plot shows two clear groups, with the rich group composed of North America, Northern and Western Europe, New Zealand and Australia. As with the histogram, if we remake the plot using a log scale

```
p + scale_y_continuous(trans = "log2")
```

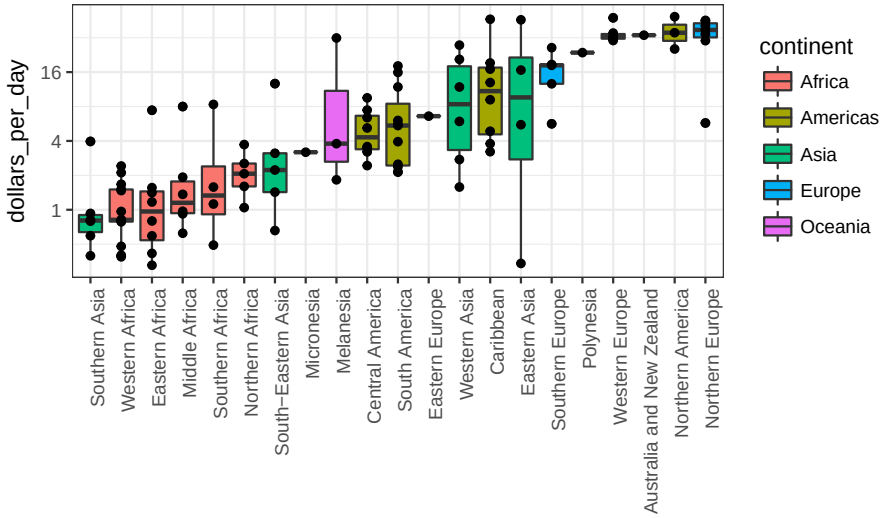


we are able to better see differences within the devoloping world.

1.9.8.2 Show the data

In many cases we do not show the data because it adds clutter to the plot and obfuscates the message. In the example above we don't have that many points. We can add this layer using geom_point() this is not the case here and adding points, actually lets us see all the data

```
p + scale_y_continuous(trans = "log2") + geom_point(show.legend = FALSE)
```



1.9.9 Comparing Distributions

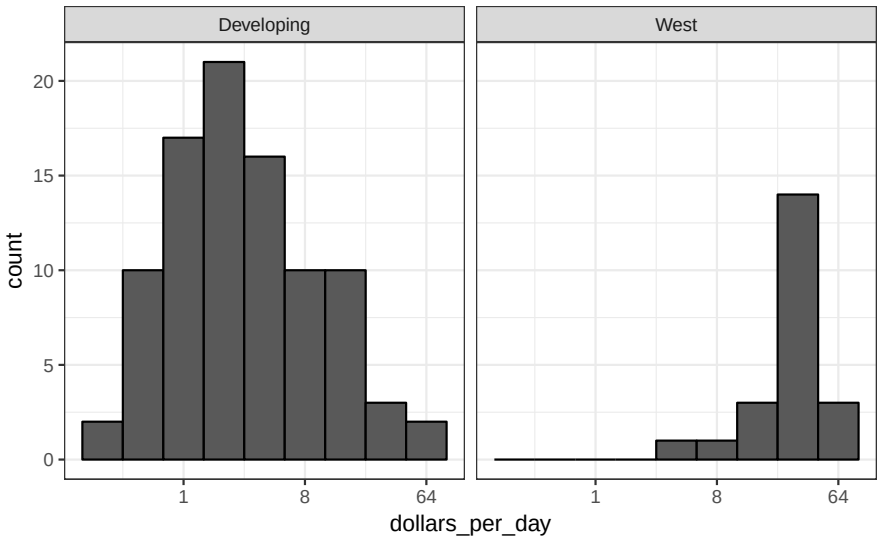
The exploratory data analysis above has revealed two characteristics about average income distribution in 1970. Using a histogram we found a bimodal distribution with the modes relating to poor and rich countries. Then by stratifying by region an examining boxplots we found that the rich countries were mostly in Europe and Northern America, along with Australia and New Zealand. We define a vector with these regions:

```
west <- c("Western Europe","Northern Europe","Southern Europe",
          "Northern America","Australia and New Zealand")
```

Now we want to focus on comparing the differences in distributions across time.

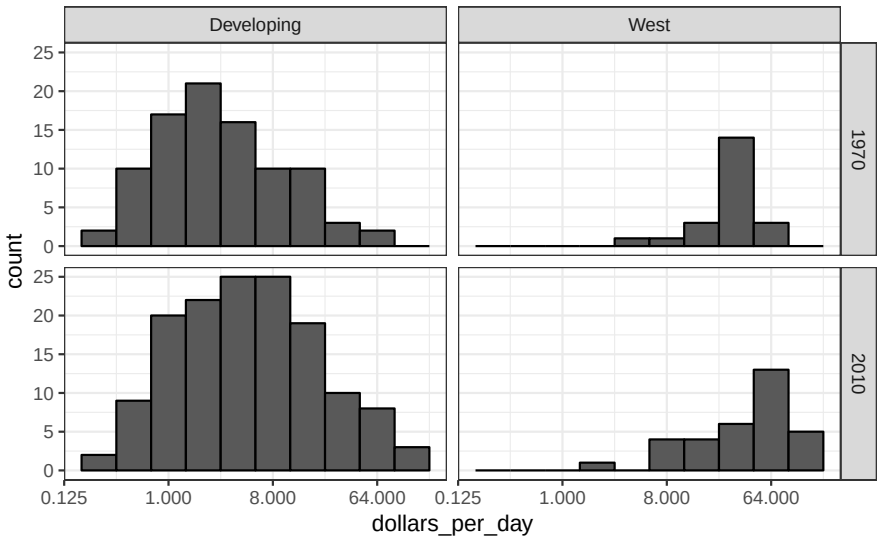
We start by confirming that the bimodality observed in 1970 is explained by a west versus devoloping world dichotomy. We do this by creating histograms for the groups previously identified. Note that we create the two groups with am `ifelse` inside a `mutate` and that we use `facet_grid` to make a histogram for each group:

```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  mutate(group = ifelse(region%in%west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(. ~ group)
```



Now we are ready to see if the separation is worse today than it was 40 years ago. We do this by faceting by both region and year:

```
past_year <- 1970
present_year <- 2010
gapminder %>%
  filter(year %in% c(past_year, present_year) & !is.na(gdp)) %>%
  mutate(group = ifelse(region%in%west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(year ~ group)
```



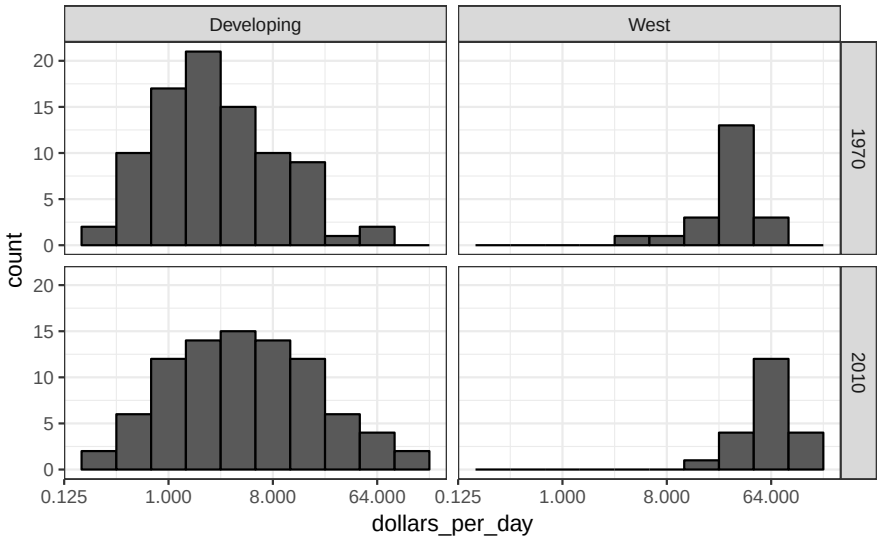
Before we interpret the findings of this plot, we note that there are more countries represented in the 2010 histograms than in 1970: the total counts are larger. One reason for this is that several countries were founded after 1970, for example the Soviet Union turned into several countries including Russia and Ukraine during the 1990s. Another reason is that data was available for more countries during 2010.

We remake the plots using only countries with data available for both years. In the data wrangling chapter we will learn `tidyverse` tools that permit us to write efficient code for this, but here simple code using the `intersect` function:

```
country_list_1 <- gapminder %>%
  filter(year == past_year & !is.na(dollars_per_day)) %>% .$country

country_list_2 <- gapminder %>%
  filter(year == present_year & !is.na(dollars_per_day)) %>% .$country

country_list <- intersect(country_list_1, country_list_2)
```

These 108 account for 86 % of the world population, so this subset should be representative.

Let's remake the plot but only for this subset by simply adding `country %in% country_list` to the filter function:



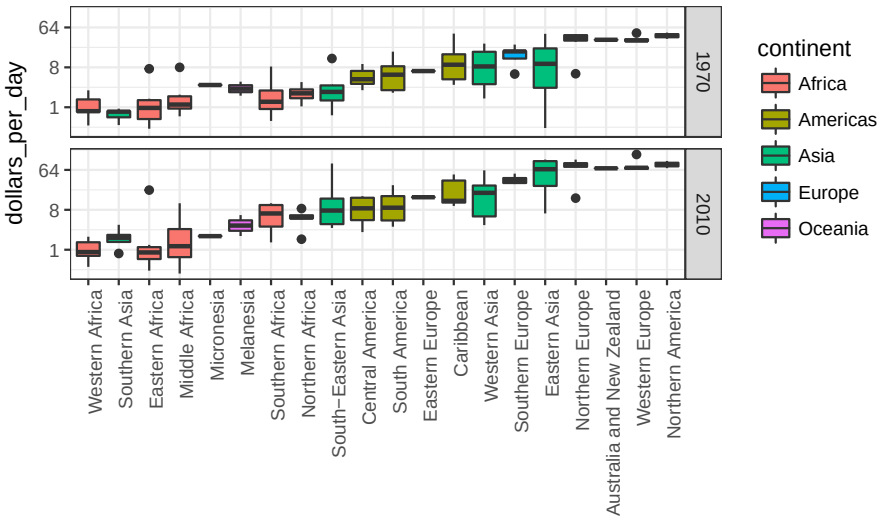
We now see that while the rich countries have become a bit richer, percentage-wise, the poor countries appear to have improved more. In particular we see that the proportion of *developing* countries earning more than $16 a day increases substantially.

To see which specific regions improved the most we can remake the boxplots we made above but now adding 2010

```
p <- gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
  ggplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("") +
  scale_y_continuous(trans = "log2")
```

and then using facet to compare the two years:

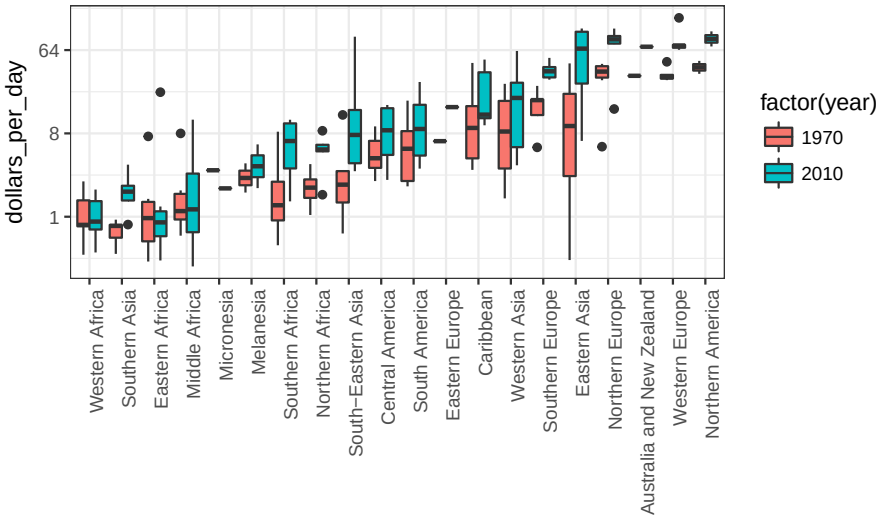
```
p + geom_boxplot(aes(region, dollars_per_day, fill = continent)) +
  facet_grid(year~.)
```



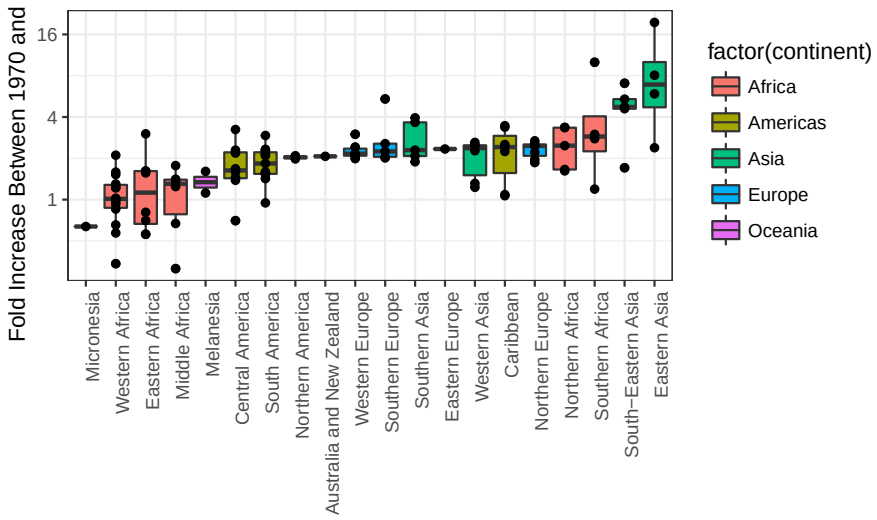
Here, we pause to introduce another powerful ggplot2 feature. Because we want to compare each region before and after, it would be convenient to have the 1970 boxplot next to the 2010 boxplot for each region. In general, comparisons are easier when data are plotted next to each other.

So, instead of faceting, we keep the data from each year together, but ask ggplot to color (or fill) them depending on the year, ggplot automatically separates them and puts the two boxplots next to each other. Because year is a number, we turn it into a factor so that each category because ggplot automatically assigns a color to each category of a factor:

```
p + geom_boxplot(aes(region, dollars_per_day, fill = factor(year)))
```



Finally, we point out that if what we are most interested is in comparing before and after values, it might make more sense to plot the ratios, or difference in the log scale. We are still not ready to learn to code this but here is what the plot would look like:

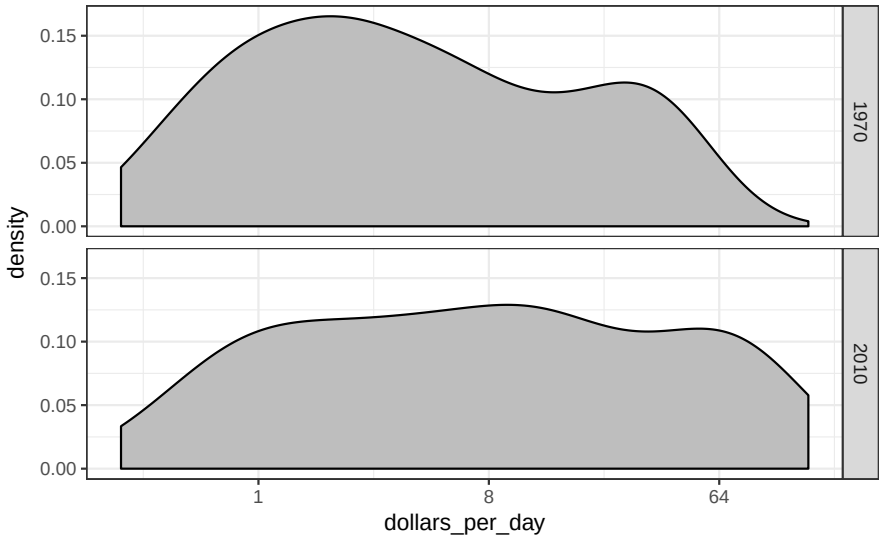


1.9.10 Density plots

We have used data exploration to discover that income gap between rich and poor countries has closed considerably during the last 40 years. We used a series of histograms and boxplots to see this. Here we suggest a succint way to convey this message with just one plot. We will use smooth density plots.

Let's start by noting that density plots for income distribution in 1970 and 2010 deliver the message that the gap is closing:

```
gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  ggplot(aes(dollars_per_day)) +
  geom_density(fill = "grey") +
  scale_x_continuous(trans = "log2") +
  facet_grid(year~.)
```



In the 1970 plot we see two clear modes, poor and rich countries. In 2010 it appears that some of the poor countries have shifted towards the right, closing the gap.

The next message we need to covey is that the reason for this change in distribution is that poor countries became richer rather than some rich countries becoming poorer. To do this all we need to do is assig a color

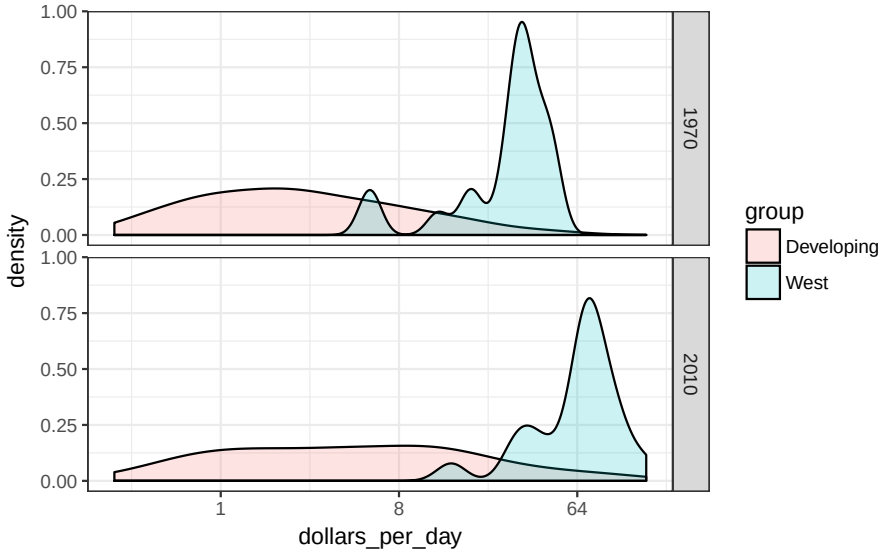
to the groups we identified during data exploration.

However, before we can do this we need to learn how to make these smooth densities in a way that preserves information of how many countries are in each group. To understand why we need this, note the discrepancy in the size of each group:

group	n
Developing	87
West	21

but when we overlay two densities, the default is to have the area represented by each distribution add up to 1 regardless of the size of each group:

```
gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day, fill = group)) +
  scale_x_continuous(trans = "log2") +
  geom_density(alpha = 0.2) +
  facet_grid(year ~ .)
```



which make it appear as if there are the same number of countries in each group. To change this, we will need to learn to access computed variables with `geom_density` function.

1.9.11 Accessing Computed Variables

To have the areas of these densities be proportional to the size of the groups, we can simply multiply the y-axis values by the size of the group. From the `geom_density` help file we see that the functions computes a variable called `count` that does exactly this. We want this variable to be on the y-axis rather than the density.

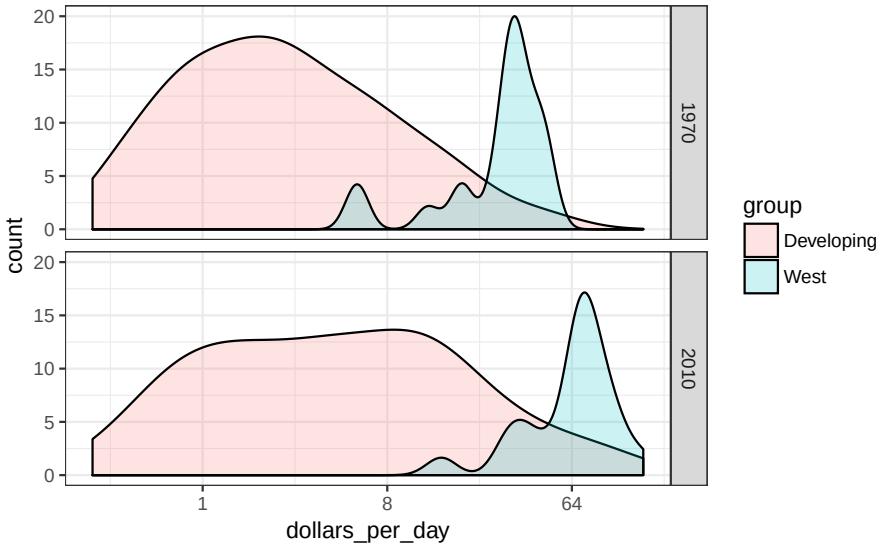
In ggplot we access these variables by surrounding them the name by `...` So we will use the following mapping:

```
aes(x = dollars_per_day, y = ..count..)
```

We can now create the desired plot by simply chaging the mapping in the previous code chunk:

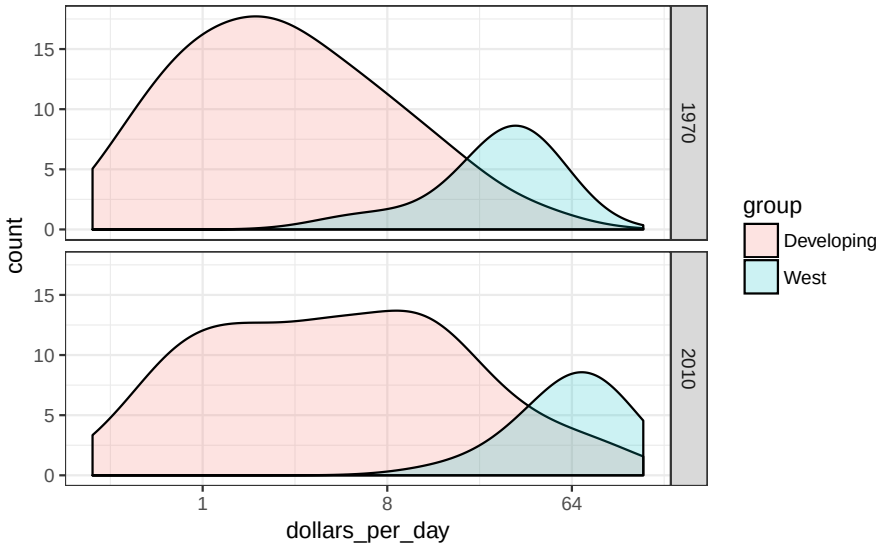
```
p <- gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
```

```
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day, y = ..count.., fill = group)) +
  scale_x_continuous(trans = "log2")

p + geom_density(alpha = 0.2) + facet_grid(year ~ .)
```



If we want the densities to be smoother, we use the `bw` argument. We tried a few and decided on 0.75:

```
p + geom_density(alpha = 0.2, bw = 0.75) + facet_grid(year ~ .)
```



This plot now shows what is happening very clearly. The developing world distribution is changing. A third mode appears consisting of the countries that most closed the gap.

1.9.12 'case_when'

We can actually make this figure somewhat more informative. From the exploratory data analysis we noticed that many of the countries that most improved were from Asia. We can easily alter the plot to show key regions separately.

We introduce the `case_when` function useful for defining groups. It currently does not have a data argument (this might change) so we need to access the components of our data table using the dot placeholder:

```
gapminder <- gapminder %>%
  mutate(group = case_when(
    .$region %in% west ~ "West",
    .$region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
    .$region %in% c("Caribbean", "Central America", "South America") ~ "Latin America",
    .$continent == "Africa" & .$region != "Northern Africa" ~ "Sub-Saharan Africa",
    TRUE ~ "Others"))
```

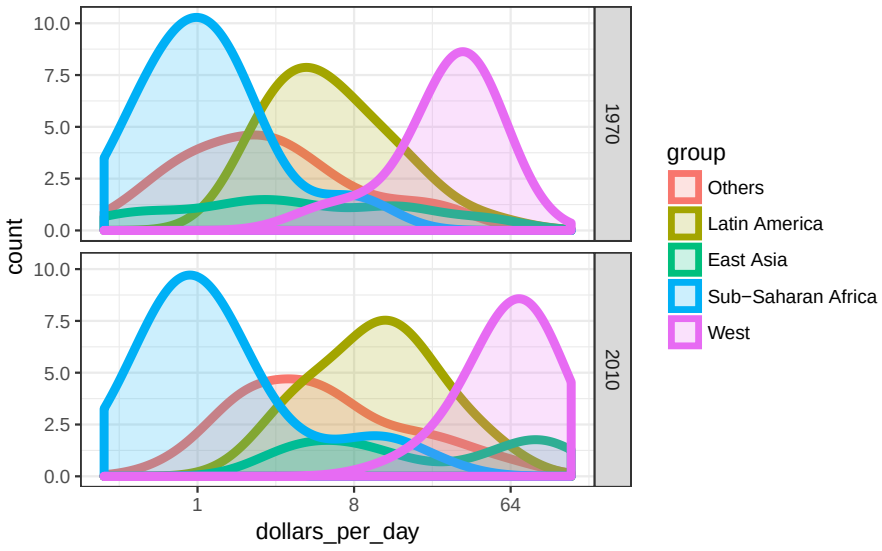
We turn this `group` variable into a factor to control the order of the levels:

```
gapminder <- gapminder %>%
  mutate(group = factor(group, levels = c("Others", "Latin America", "East Asia","Sub-Saharan Africa",
```

We pick this particular order for a reason that becomes clear later.

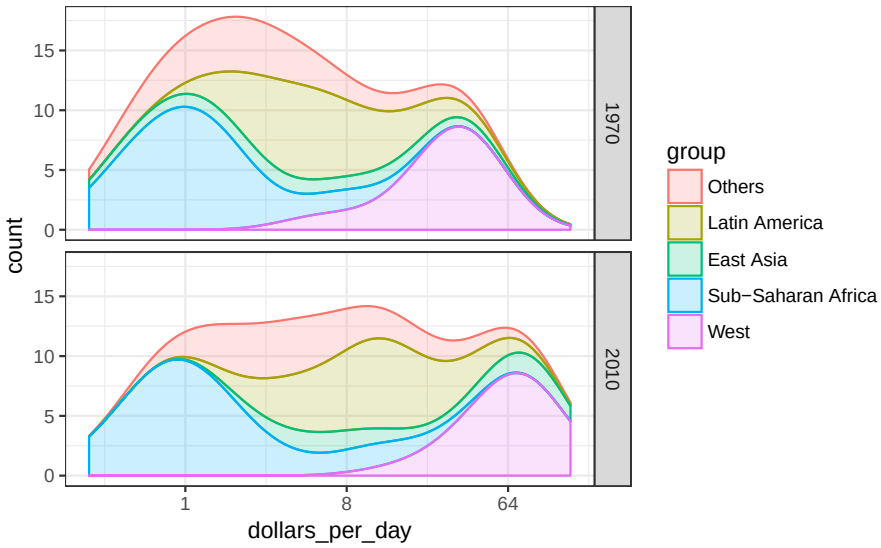
Now we can now easily plot the densities for each. We use `color` and `size` to clearly see the tops:

```
p <- gapminder %>%
    filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  ggplot(aes(dollars_per_day, y = ..count.., fill = group, color = group)) +
  scale_x_continuous(trans = "log2")

p + geom_density(alpha = 0.2, bw = 0.75, size = 2) + facet_grid(year ~ .)
```



The plot cluttered and somewhat hard to read. A clearer picture is sometimes achieved by stacking the densities on top of each other:

```
p + geom_density(alpha = 0.2, bw = 0.75, position = "stack") + facet_grid(year ~ .)
```

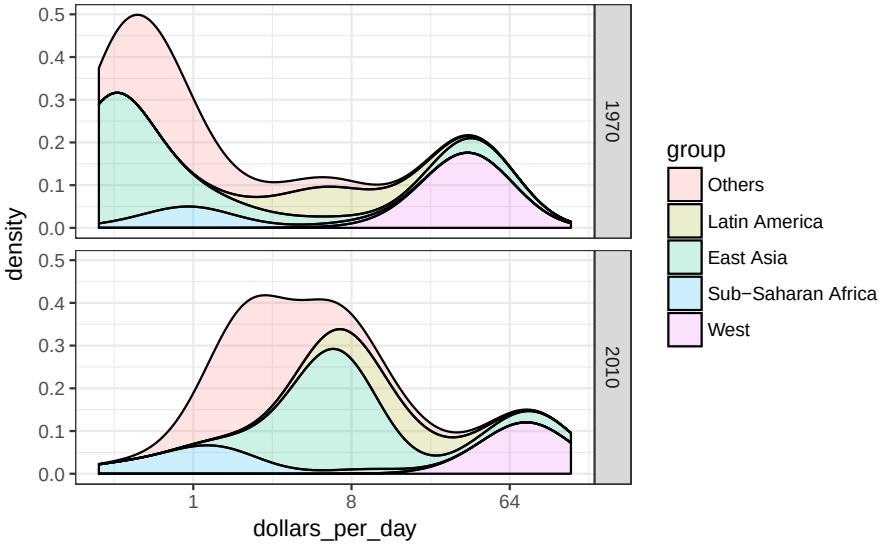


Here we can clearly see how the distributions for East Asia, Latin America and Others shift markedly to the right. While Sub-Saharan Africa remains stagnant.

Note that we order the levels of the group so that The West density be plotted first, then Sub-Saharan Africa. Having the two extremes be plotted first let's us see the remaining bimodality better.

1.9.12.1 Weighted densities

As a final point, we note that these distributions weigh every country the same. So if most of the population is improving, but living in a very large country, such as China, we might not appreciate this. We can actually weight the smooth densities using the `weight` mapping argument. The plot then looks like this:



This particular figure shows very clearly how the income distribution gap is closing with most of the poor remaining in Sub-Saharan Africa.

1.10 Ecological Fallacy

Throughout this section we have been comparing regions of the world. We have seen that on average, some regions do better than others. Here we focus on describing the importance of variability within the groups.

Here we will focus on the relationship between country child survival rates and average income. We start by comparing these quantities across regions. We define a few more regions:

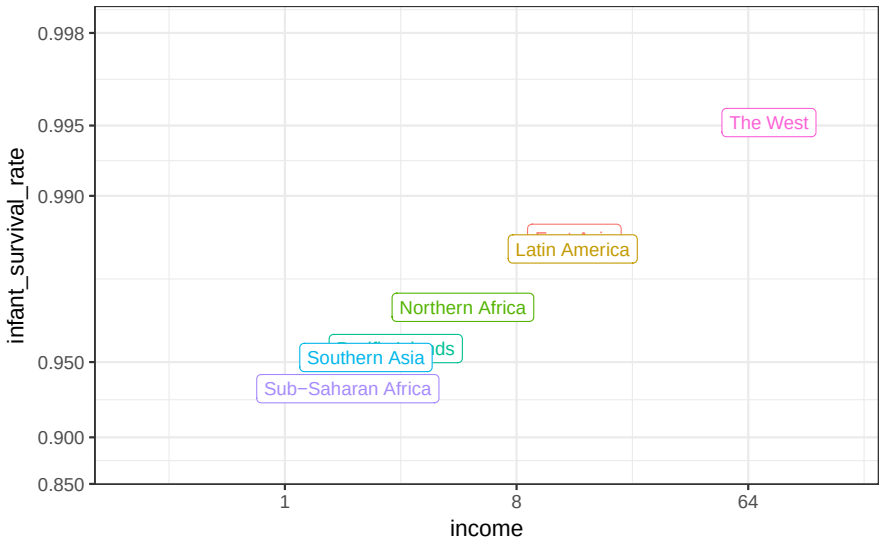
```
gapminder <- gapminder %>%
  mutate(group = case_when(
    .$region %in% west ~ "The West",
    .$region %in% "Northern Africa" ~ "Northern Africa",
    .$region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
    .$region == "Southern Asia"~ "Southern Asia",
    .$region %in% c("Central America", "South America", "Caribbean") ~ "Latin America",
    .$continent == "Africa" & .$region != "Northern Africa" ~ "Sub-Saharan Africa",
    .$region %in% c("Melanesia", "Micronesia", "Polynesia") ~ "Pacific Islands"))
```

We then compute these quantities for each region.

```
surv_income <- gapminder %>%
  filter(year %in% present_year & !is.na(gdp) & !is.na(infant_mortality) & !is.na(group)) %>%
  group_by(group) %>%
  summarize(income = sum(gdp)/sum(population)/365,
            infant_survival_rate = 1-sum(infant_mortality/1000*population)/sum(population))

surv_income %>% arrange(income)
#> # A tibble: 7 x 3
#>                group income infant_survival_rate
#>                <chr>  <dbl>                <dbl>
#> 1 Sub-Saharan Africa   1.76                0.936
#> 2      Southern Asia   2.07                0.952
#> 3    Pacific Islands   2.70                0.956
#> 4    Northern Africa   4.94                0.970
#> 5      Latin America  13.24                0.983
#> 6          East Asia  13.44                0.985
#> # ... with 1 more rows
```

This shows a dramatic difference. While in the west less than 0.5% children die, in Sub-Saharan Africa the rate is higher than 6%! The relationship between these two variables is almost perfectly linear.

```
surv_income %>% ggplot(aes(income, infant_survival_rate, label = group, color = group)) +
  scale_x_continuous(trans = "log2", limits = c(0.25, 150)) +
  scale_y_continuous(trans = "logit", limit = c(0.875, .9981),
                     breaks = c(.85,.90,.95,.99,.995,.998)) +
  geom_label(size = 3, show.legend = FALSE)
```



In this plot we introduce the use of the `limit` argument which let's us change the range of the axes. We are making the range larger than the data needs because we will later compare this plot to one with more variability and we want the ranges to be the same. We also introduce the `breaks` argument which lets us set the location of the axis labels. Finally we introduce a new transformation, the logistic transformation.

1.10.0.1 Logistic transformation

The logistic or logit transformation for a proportion or rate p is defined as

$$f(p) = \log\left(\frac{p}{1-p}\right)$$

When p is a proportion or probability, the quantity that is being logged, $p/(1-p)$ is called the *odds*. In the case p is the proportion of a children that survived. The odds tell us how many more children are expcted to survive than to die. The log transformation makes this symmetric. If the rates are the same, then the log odds is 0. Fold increases or decreases turn into positive and negative increments respectively.

This scale is useful when we want to highlight differences near 0 or 1. For survival rates this is important because a survival rate of 90% is unacceptable, while a survival of 99% is relatively good. We would much prefer a survival rate closer to 99.9%. We want our scale to highlight these difference and the logit does this. Note that 99.9/0.1 is about 10 times bigger than 99/1 which is about 10 times larger than 90/10. And by using the log these fold changes turn into constant increases.

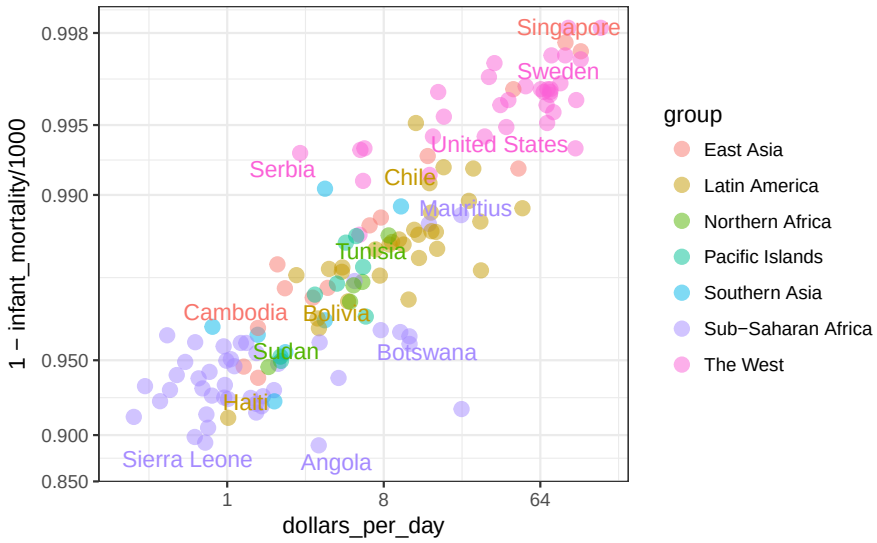
1.10.1 Show the data

Now, back to our plot. Based on the plot above, do we conclude that a country with a low income is destined to have low survival rate? Do we conclude that all survival rates in Sub-Saharan Africa are all lower than in Southern Asia which in turn are lower than in the Pacific Islands, and so on?

Jumping to this conclusion based on a plot showing averages is referred to as the *ecological fallacy*. The almost perfect relationship between survival rates and income is only observed for the averages at the region level. Once we show all the data we see a somewhat more complicated story:

```
library(ggrepel)
highlight <- c("Sierra Leone", "Mauritius",  "Sudan", "Botswana", "Tunisia",
               "Cambodia","Singapore","Chile", "Haiti", "Bolivia",
               "United States","Sweden", "Angola", "Serbia")
```

```
gapminder %>% filter(year %in% present_year & !is.na(gdp) & !is.na(infant_mortality) & !is.na(group) ) %
  ggplot(aes(dollars_per_day, 1 - infant_mortality/1000, col = group, label = country)) +
  scale_x_continuous(trans = "log2", limits=c(0.25, 150)) +
  scale_y_continuous(trans = "logit",limit=c(0.875, .9981),
                     breaks=c(.85,.90,.95,.99,.995,.998)) +
  geom_point(alpha = 0.5, size = 3) +
  geom_text_repel(size = 4, show.legend = FALSE,
    data = filter(gapminder, year %in% present_year & country %in% highlight))
```



Specifically, we see that there is a large amount of variability. We see that countries from the same regions can be quite different and that countries with the same income can have different survival rates. For example, while on average, Sub-Saharan Africa, had the worse health and economic outcomes, there is wide variability within that group. For example note that Mauritius and Botswana are doing better than Angola and Sierra Leone with Mauritius comporable to Western countries.

1.11 Some Data Visualization Principles

We have already provided some rules to follow as we created plots for our examples. Here we aim to provide some general principles we can use as a guides for effective data visualization. Much of this section is based on a talk by Karl Broman titled "Creating effective figures and tables" including some of the figures which were made with code that Karl makes available on his GitHub repository, and class notes from Peter Aldhous' Introduction to Data Visualization course. Following Karl's approach, we show some examples of plot styles we should avoid, explain how to improve them, and use these as motivation for a list of principles. We compare and contrast plots that follow these principles to those that don't.

The principles are mostly based on research related to how humans detect patterns and make visual comparisons. The preferred approaches are those that best fit the way our brains process visual information. When deciding on a visualization approach it is also important to keep our goal in mind. We may be comparing a viewable number of quantities, describing distribution for categories or numeric values, comparing the data from two groups, or describing the relationship between two variables. As final note, we also note that for a data scientist it is important to adapt and optimize graphs to the audience. For example, an exploratory plot made for ourselves will be different than a chart intended to communicate a finding to a general audience.

We will be using these libraries:

```
library(tidyverse)
library(gridExtra)
```

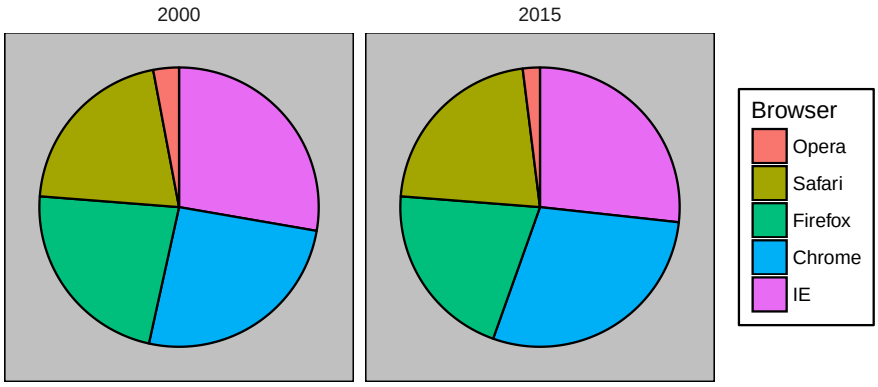


Figure 2: Pie chart of browser usage.

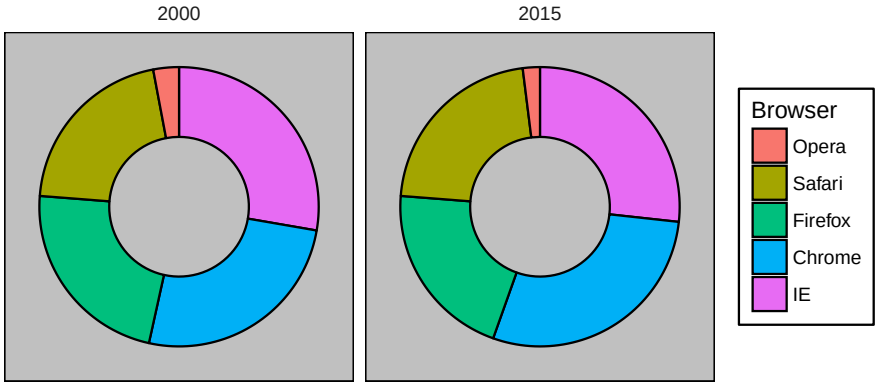


Figure 3: Pie chart of browser usage.

```
library(dslabs)
ds_theme_set()
```

1.11.1 Encoding data using visual cues

We start by describing some principles for encoding data. There are several approaches at our disposal including position, aligned lengths, angles, area, brightness, and color hue.

To illustrate how some of these strategies compare let's suppose we want to report the results from two hypothetical polls asking regarding browser preference taken in 2000 and then 2015. Here, for each year, we are simply comparing four quantities, four percentages. A widely used graphical representation of percentages, popularized by Microsoft Excel, is the pie chart:

Here we are representing quantities with both areas and angles since both the angle and area of each pie slice is proportional to the quantity it represents. This turns out to be a sub optimal choice since, as demonstrated by perception studies, humans are not good at precisely quantifying angles and are even worse when only are is available. The donut chart is an example of a plot that uses only area:

To see how hard it is to quantify angles and are note that the rankings and all the percentages in the plots above changed fro 2000 to 2015. Can you determine the actual percentages and rank the browsers' popularity? Can you see how the percentages changed from 2000 to 2015? It is not easy to tell from the plot. In fact, the `pie` R function help file states

> "Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a prefe

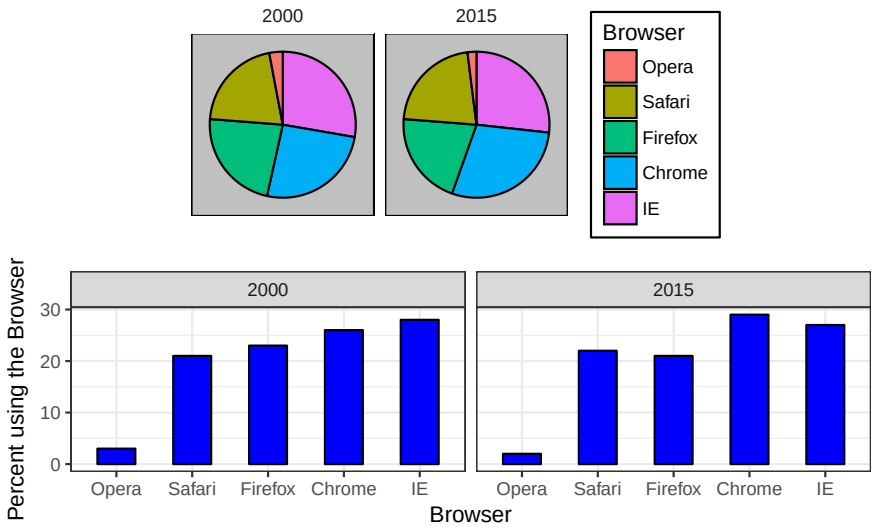


Figure 4: Barplot of browser usage.

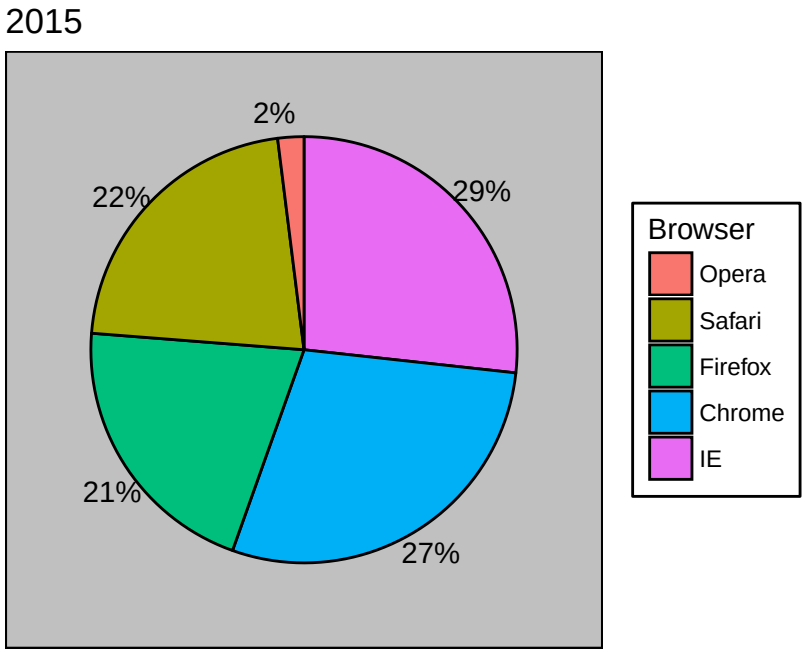
In this case, simply showing the numbers is not only clearer, but it would saves on print cost if making a paper version.

Browser	2000	2015
Opera	3	2
Safari	21	22
Firefox	23	21
Chrome	26	29
IE	28	27

The preferred way to plot quantities is to use length and position since humans are much better at judging linear measure. The bar plot uses bars use this approach by using bars of length proportional to the quantities of interest. By adding horizontal lines at strategically chosen values, in this case at every multiple of 10, we ease the quantifying through the position of the top of the bars. Compare and contrast the information we can extract from the two figures.

Notice how much easier it is to see the differences in the barplot. In fact, we can now determine the actual percentages by following a horizontal line to the x-axis.

If for some reason you need to make a pie chart, do include the percentages as numbers to avoid having to infer them from the angles or area:



In general, position and length are the preferred ways to display quantities over angles which are preferred to area. Brightness and color are even harder to quantifying that angles and area but, as we will see later, they are sometimes useful when more than two dimensions are being displayed.

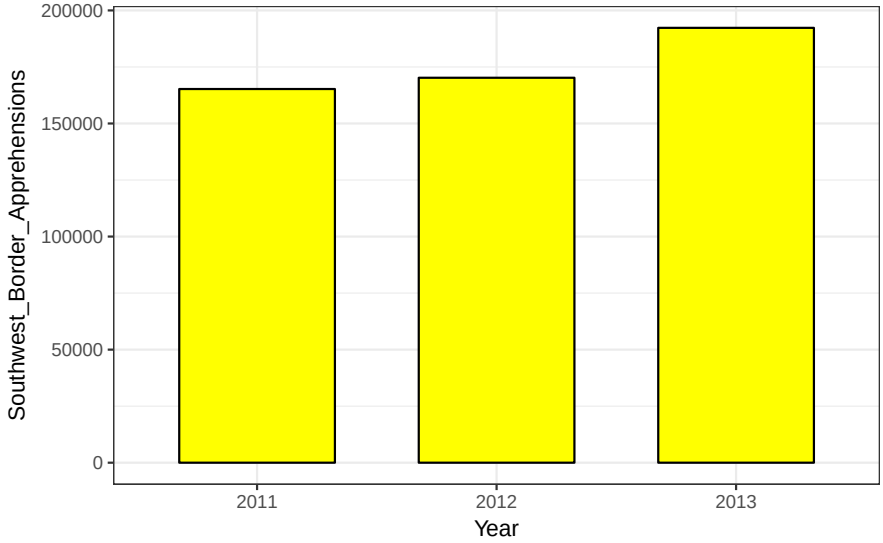
1.11.2 Know when to include 0

When using barplots it is dishonest not to start the bars at 0. This is because, by using a barplot, we are implying the length is proportional to the quantities being displayed. By avoiding 0, relatively small difference can be made to look much bigger than they actually are. This approach is often used by politicians or media organizations trying to exaggerate a difference. Here is a illustrative example:

```
knitr::include_graphics("http://paldhous.github.io/ucb/2016/dataviz/img/class2_8.jpg")
```

(Source: Fox News, via Peter Aldhous via Media Matters via Fox News) via Media Matters.

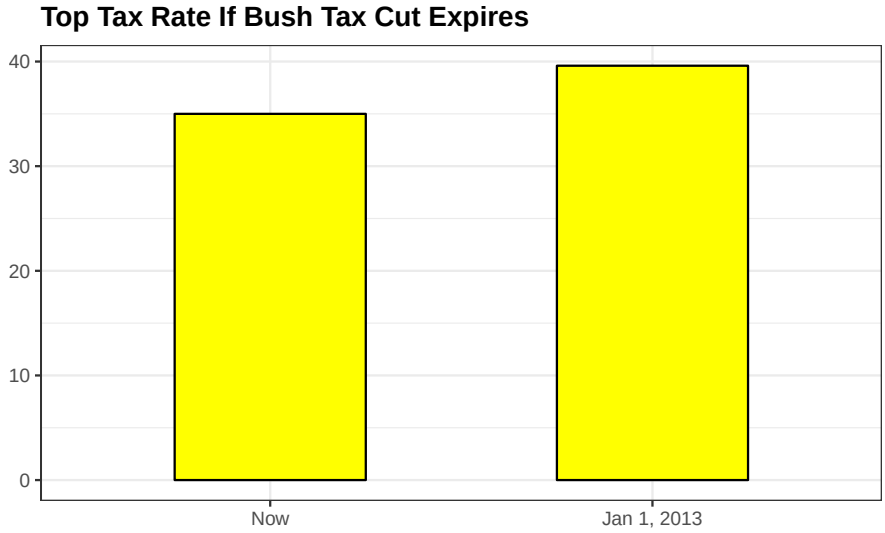
From the plot above, it appears that apprehensions have almost tripled when in fact they have only increased by about 16%. Starting the graph at 0 illustrates this clearly:



Here is another example, described in detail here.

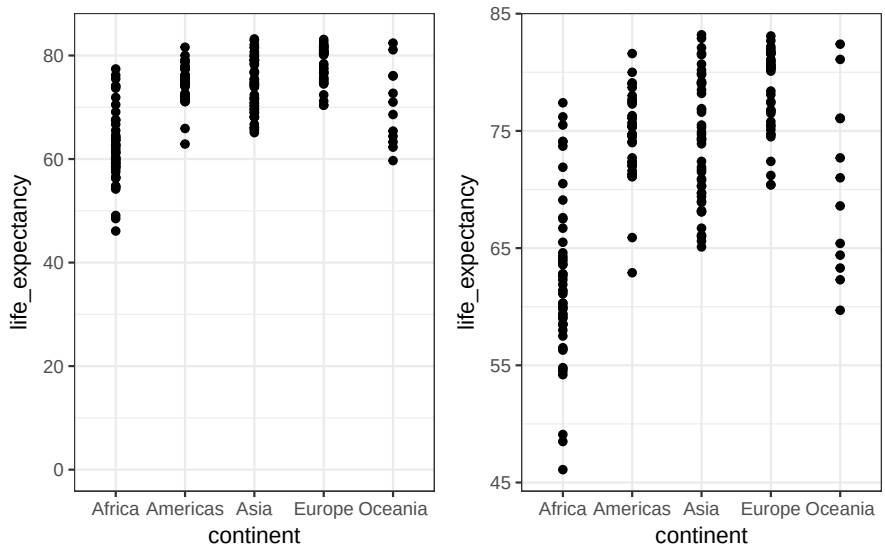
```
knitr::include_graphics("http://i2.wp.com/flowingdata.com/wp-content/uploads/2012/08/Bush-cuts.png")
```

which makes a 13% increase look like a five fold change. Here is the appropriate plot:



When using position rather than length, then it is not necessary to include 0. This is particularly the case when we want to compare differences between groups relative the variability seen within the groups. Here is illustrative example showing country average life expectancy stratified into continents in 2012:

```
p1 <- gapminder %>% filter(year == 2012) %>%
  ggplot(aes(continent, life_expectancy)) +
  geom_point()
p2 <- p1 +
  scale_y_continuous(limits = c(0, 84))
grid.arrange(p2, p1, ncol = 2)
```



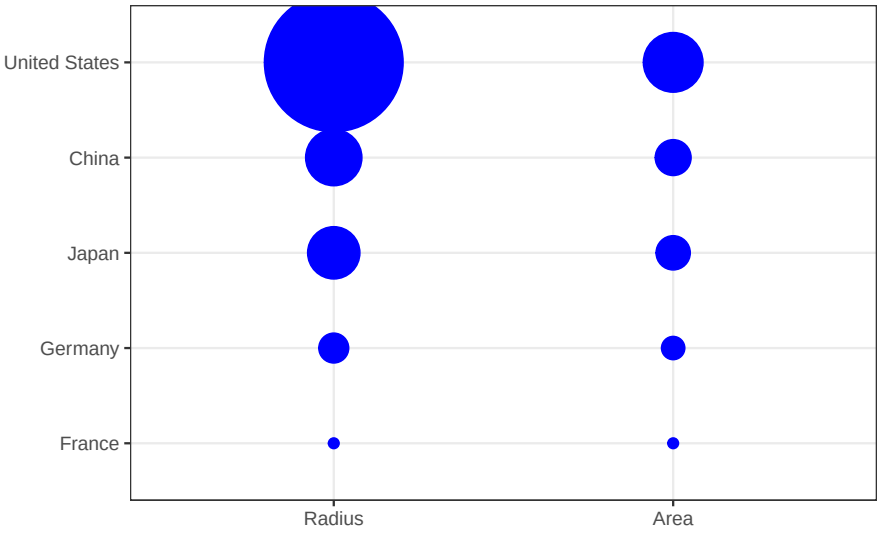
The plot on the left, which includes 0, the space between 0 and 43 adds no information and makes it harder to appreciate the between and within variability.

1.11.3 Do not distrort quantities

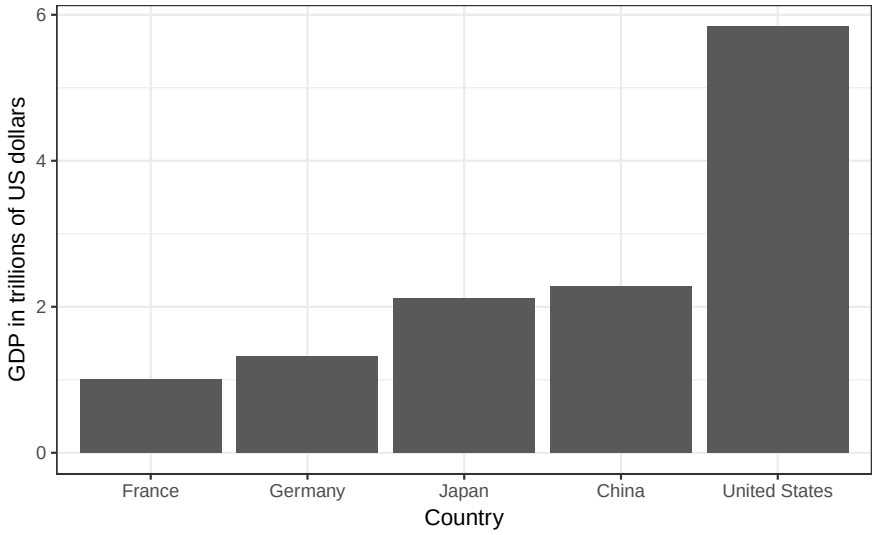
During President Barack Obama's 2011 State of the Union Address the following chart was used to compare the US GDP to the GDP of four competing nations:

```
knitr::include_graphics("http://paldhous.github.io/ucb/2016/dataviz/img/class2_30.jpg")
```

Note judging by the area of the circles the US appears to have an economy over five times larger than China and over 30 times larger than France. However, when looking at the actual numbers one sees that this is not the case. The actual ratios are 2.6 and 5.8 times bigger than China and France respectively. The reason for this distortion is that the radius, rather than the area, was made to be proportional to the quantity which implies that the proportion between the areas is squared: 2.6 turns into 6.5 and 5.8 turns into 34.1. Here is a comparison of the circles we get if we make the value proportional to the radius and to the area:



Not surprisingly, ggplot defaults to using area rather than radius. Of course, in this case, we really should not be using area at all since we can use position and length:



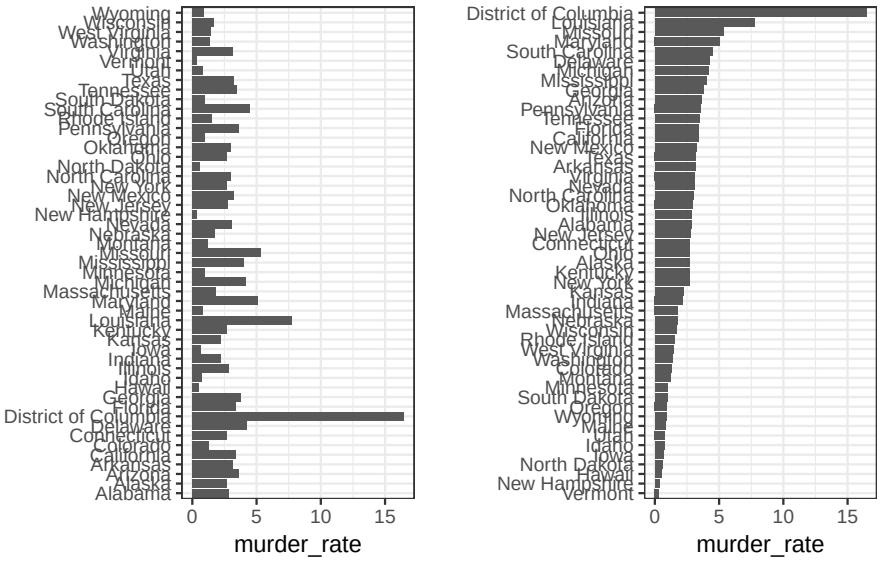
1.11.4 Order by a meaningful value

When one of the axes is used to show categories, as is done in barplots, the default ggplot behavior is to order the categories alphabetically when they are defined by character strings. If they are defined by factors,

they are ordered by the factor levels. We rarely want to use alphabetical order. Instead we should order by a meaningful quantity. In all the cases above, the barplots where ordered by the values being displayed. The exception was the graph showing barplots comparing browsers. In this case we kept the order the same across the barplots to ease the comparison. Instead we ordered by the average value of 2000 and 2015. We previously learned how to use the `reorder` function, which helps achieve this goal.

To appreciate how the right order can help convey a message, suppose we want to create a plot to compare the murder rate across states. We are particularly interested in the most dangerous and safest states. Note the difference when we order alphabetically (the default) versus when we order by the actual rate:

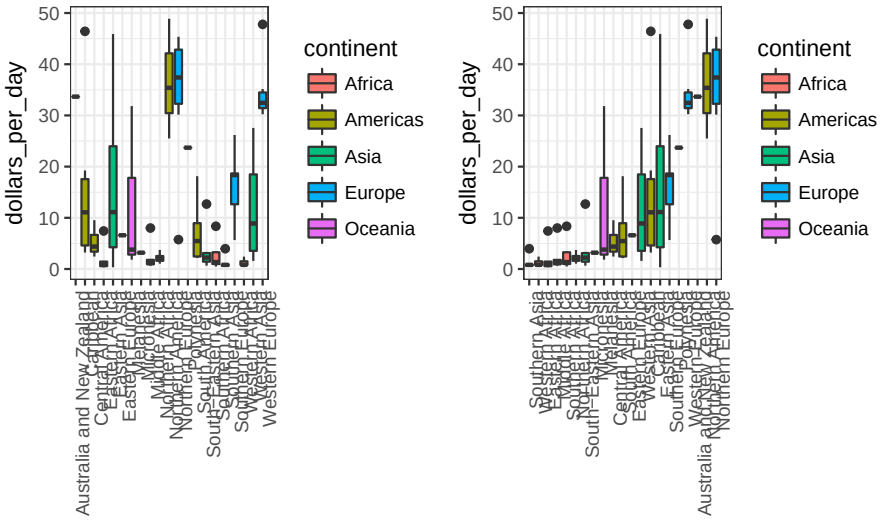
```
data(murders)
p1 <- murders %>% mutate(murder_rate = total / population * 100000) %>%
  ggplot(aes(state, murder_rate)) +
  geom_bar(stat="identity") +
  coord_flip() +
  xlab("")
p2 <- murders %>% mutate(murder_rate = total / population * 100000) %>%
  mutate(state = reorder(state, murder_rate)) %>%
  ggplot(aes(state, murder_rate)) +
  geom_bar(stat="identity") +
  coord_flip() +
  xlab("")
grid.arrange(p1, p2, ncol = 2)
```



Note that the `reorder` function lets us reorder groups as well. Earlier we saw an example related to income distributions across regions. Here are the two versions plotted against each other:

```
past_year <- 1970
p1 <- gapminder %>%
  mutate(dollars_per_day = gdp/population/365) %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(region, dollars_per_day, fill = continent)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("")
p2 <- gapminder %>%
  mutate(dollars_per_day = gdp/population/365) %>%
  filter(year == past_year & !is.na(gdp)) %>%
```

```
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
  ggplot(aes(region, dollars_per_day, fill = continent)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("")
grid.arrange(p1, p2, ncol=2)
```

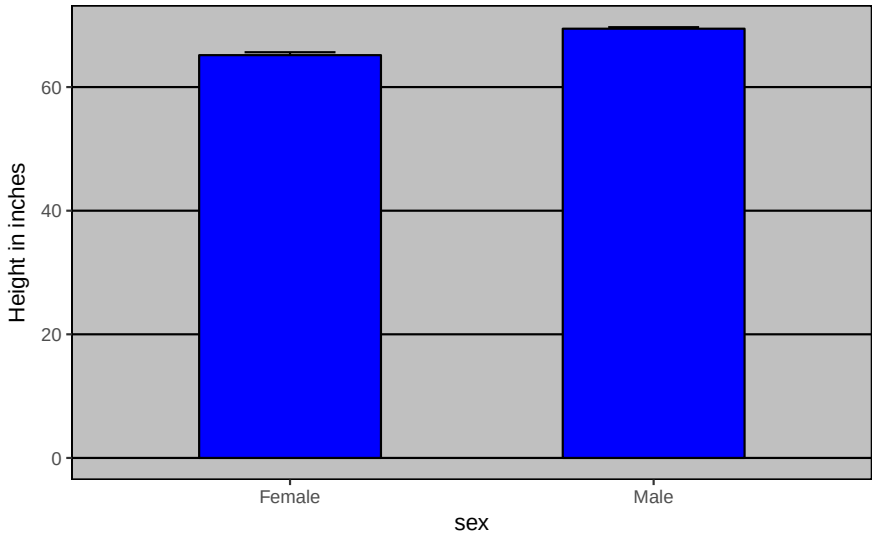


The first is orders the regions alphabetically while the second orders them by the group's median.

1.11.5 Show the data

We have focused on displaying single quantities across categories. We now shift our attention to displaying data, with a focus on comparing groups.

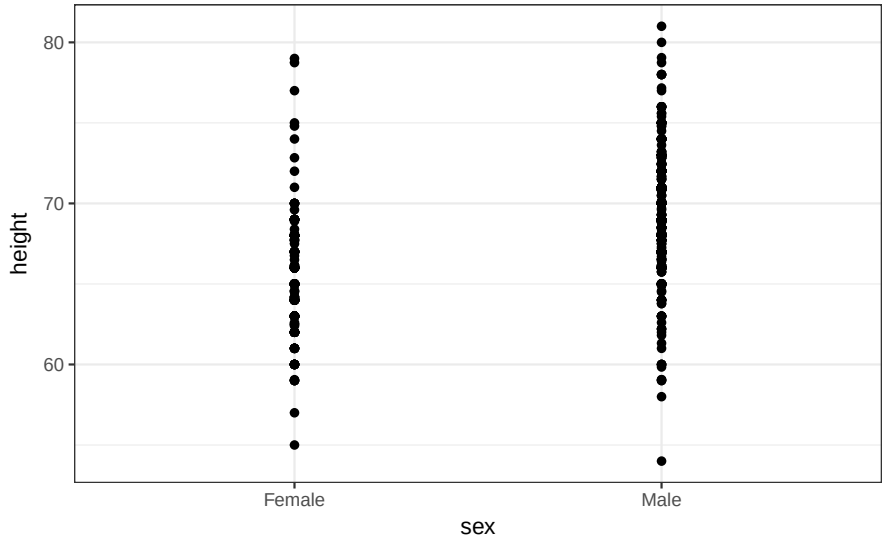
To motivate our first principle, show the data, we go back to our artificial example of describing heights to ET, an extraterrestrial. This time let's assume ET is interested is the difference in heights between males and females. A commonly seen plot used for comparisons between groups, popularized by software such as Microsoft Excel, shows the average and standard errors (standard errors are defined in a later chapter, but don't confuse them with the standard deviation of the data). The plot looks like this:



The average of each group is represented by the top of each bar and the antennae expand to the average plus two standard errors. If all ET receives is this plot he will have little information on what to expect if he meets a group of human males and females. The bars go to 0, does this mean there are tiny humans measuring less than one foot? Are all males taller than the tallest females? Is there a range of heights? ET can't answer these questions since we have provided almost no information on the height distribution.

This brings us to our first principle: show the data. This simple ggplot code already generates a more informative plot than the barplot by simply showing all the data points:

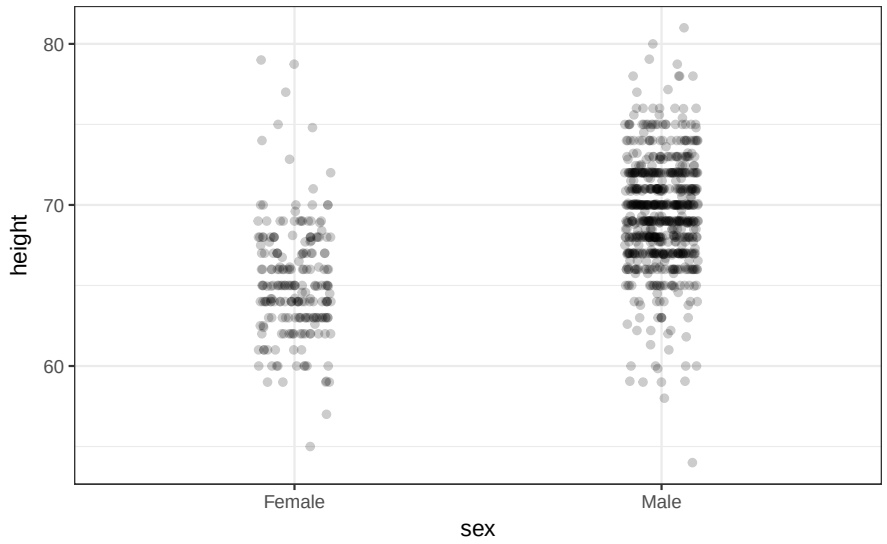
```
heights %>% ggplot(aes(sex, height)) + geom_point()
```



For example, we get an idea of the range of the data. However this plot has limitations as well since we can't really see all the 216 and 708 points plotted for females and males respectively, and many points are plotted above each other. As we have described, visualizing the distribution is much more informative. But before doing this, we point out two ways we can improve a plot showing all the points.

The first is to add *jitter*: adding a small random shift to each point. In this case adding horizontal jitter does not alter the interpretation, since the height of the points do not change, but we minimize the number of point that fall on top of each other and therefore get a better sense of how the data is distributed. A second improvement comes from using *alpha blending*: making the points somewhat transparent. The more points fall on top of each other, the darker the plot which also helps us get a sense of how the points are distributed. Here is the same plot with jitter and alpha blending:

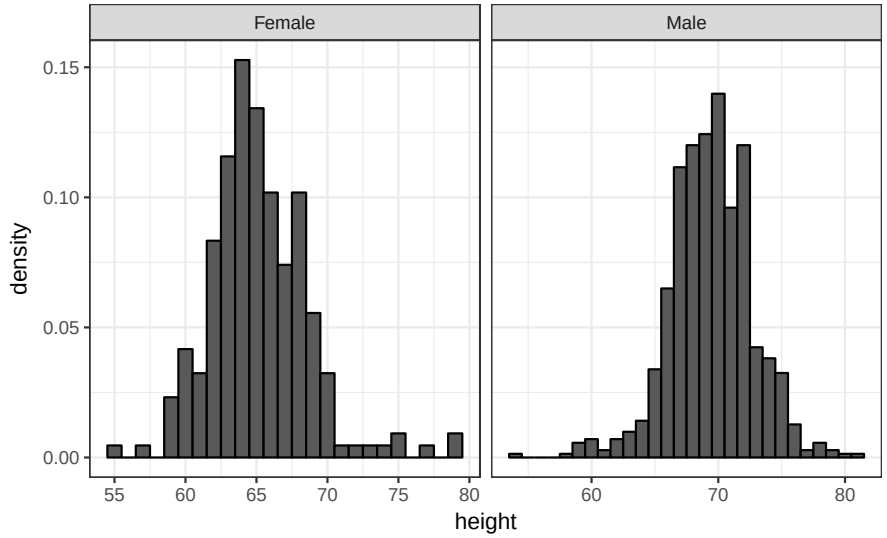
```
heights %>% ggplot(aes(sex, height)) + geom_jitter(width = 0.1, alpha = 0.2)
```



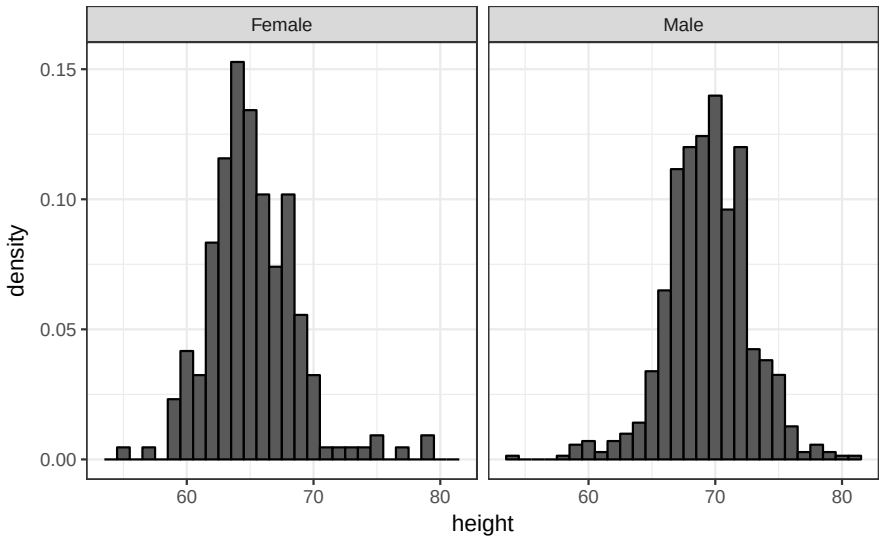
Now we start getting a sense that, on average, males are taller than females. We also note dark horizontal demonstrating that many report values are rounded to the nearest integer.

1.11.6 Ease comparisons: Use common axes

Since there are so many points it is more effective to show distributions, rather than show individual points. We therefore show histograms for each group:



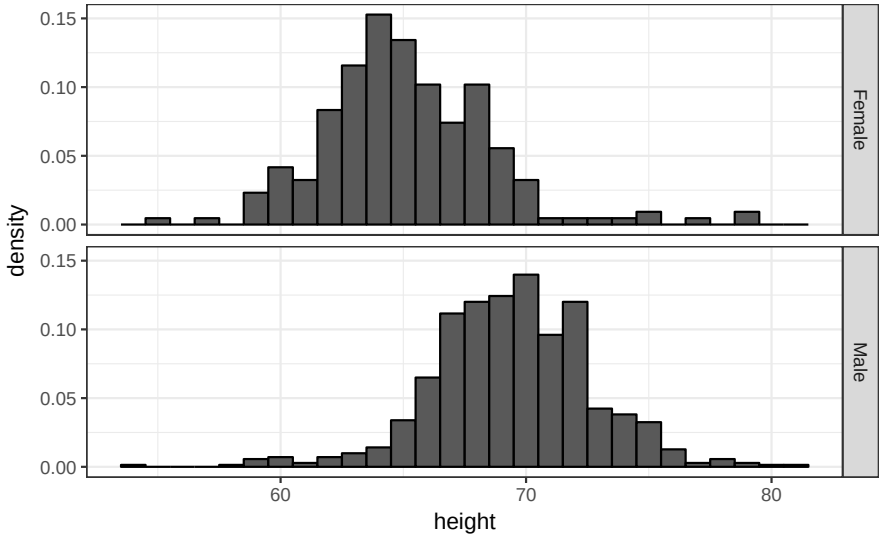
However, from this plot it is not immediately obvious that males are, on average, taller than females. We have to look carefully to notice that the x-axis has a higher range of values in the male histogram. An important principle here is to **keep the axes the same** when comparing data across to plots. Note how to comparison becomes easier:



1.11.7 Ease comparisons: align plots vertically to see horizontal changes and horizontally to see vertical changes

In these histograms, the visual cue related to decreases or increases in height are shifts to the left or right respectively: horizontal changes. Aligning the plots vertically helps us see this change when the axis are fixed:

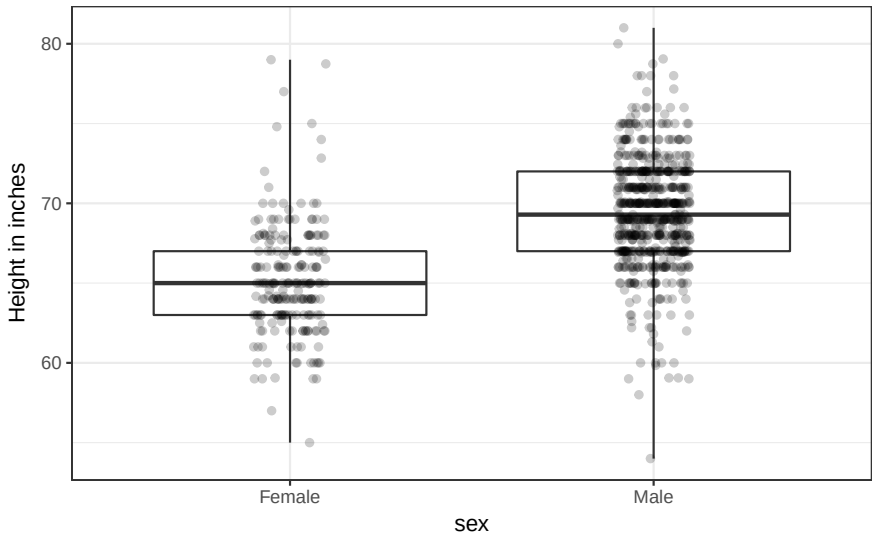
```
p2 <- heights %>%
  ggplot(aes(height, ..density..)) +
  geom_histogram(binwidth = 1, color="black") +
  facet_grid(sex~.)
p2
```



This plot makes it much easier to notice that men are, on average, taller.

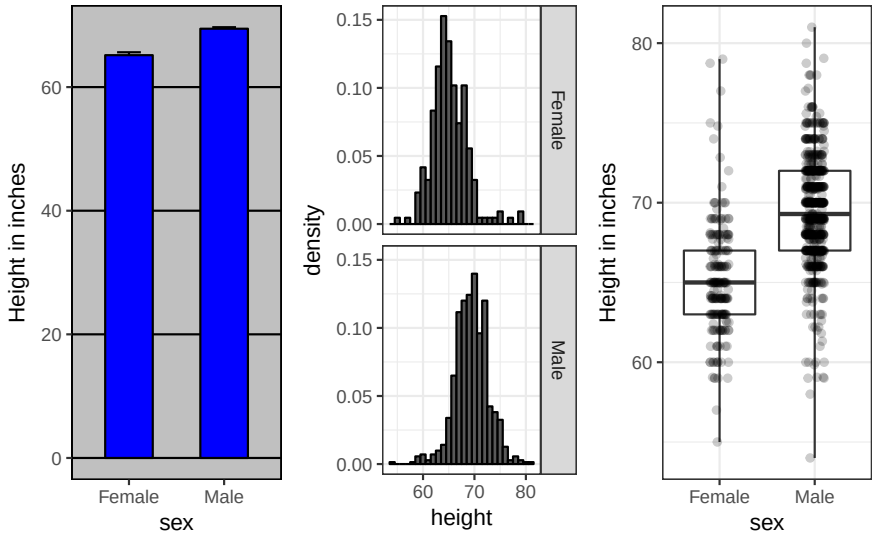
If instead of histograms we want the more compact summary provided by boxplot, then we align the horizontally, since, by default, boxplots move up and down with changes in height. Following our *show the data* principle we add overlay all the data points:

```
p3 <- heights %>%
  ggplot(aes(sex, height)) +
  geom_boxplot(coef=3) +
  geom_jitter(width = 0.1, alpha = 0.2) +
  ylab("Height in inches")
p3
```



Now contrast and compare these three plots, based on exactly the same data:

```
grid.arrange(p1, p2, p3, ncol = 3)
```



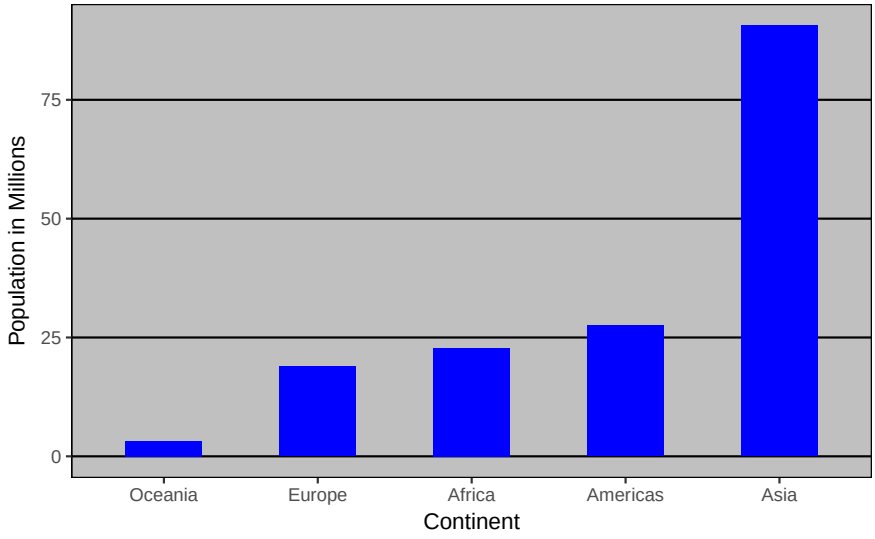
Note how much more we learn from the two plots on the right. Barplots are useful for showing one number, but not very useful when wanting to describe distributions.

1.11.8 Consider transformations

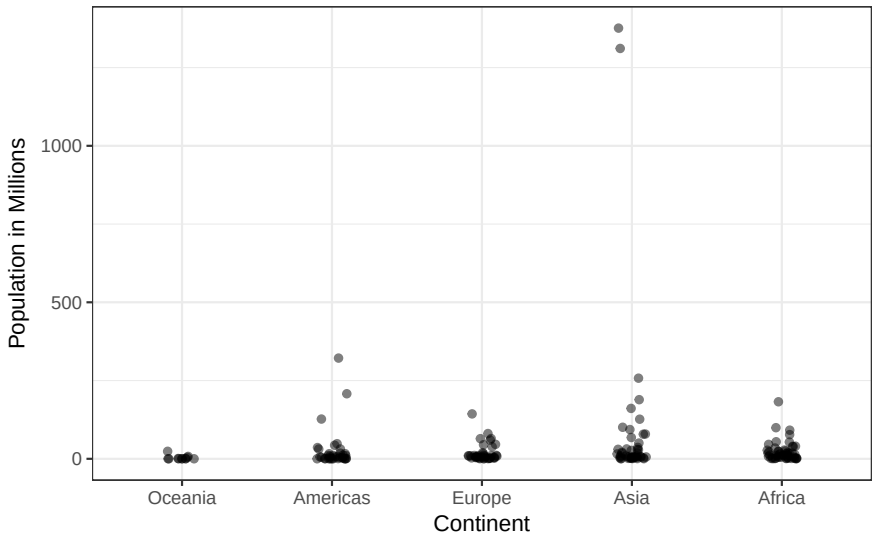
We have motivated the use the log transformation in cases were the changes are multiplicative. Population size was an example in which we found a log transformation to yield a more informative transformation.

The combination of incorrectly using barplot and when a log transformation is merited can be particularly distorting. As an example consider this barplot showing the average population sizes for each continent in

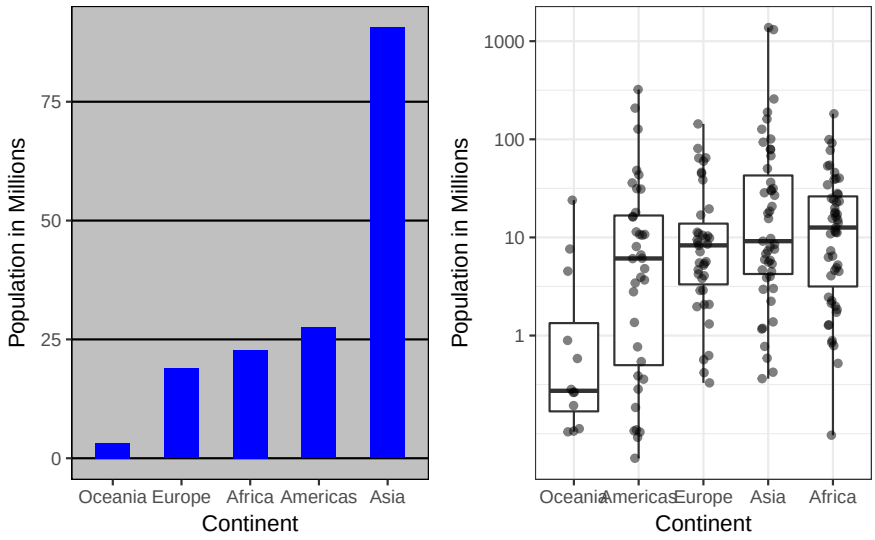
2015:



From this plot one would conclude that countries in Asia are much more populous than other continents. Following the *show the data* principle we quickly notice that this is due to two very large countries, which we assume are India and China:



Here, using a log transformation provides a much more informative plot. We compare the original barplot to a boxplot using the log scale transformation for the y-axis:

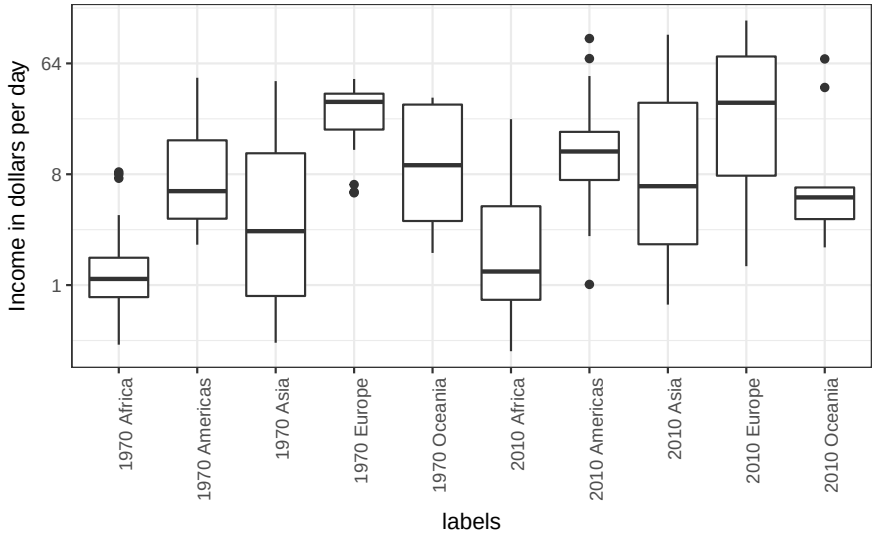


Note in particular that with the new plot we realize that countries in Africa actually has a larger median population size than those in Asia.

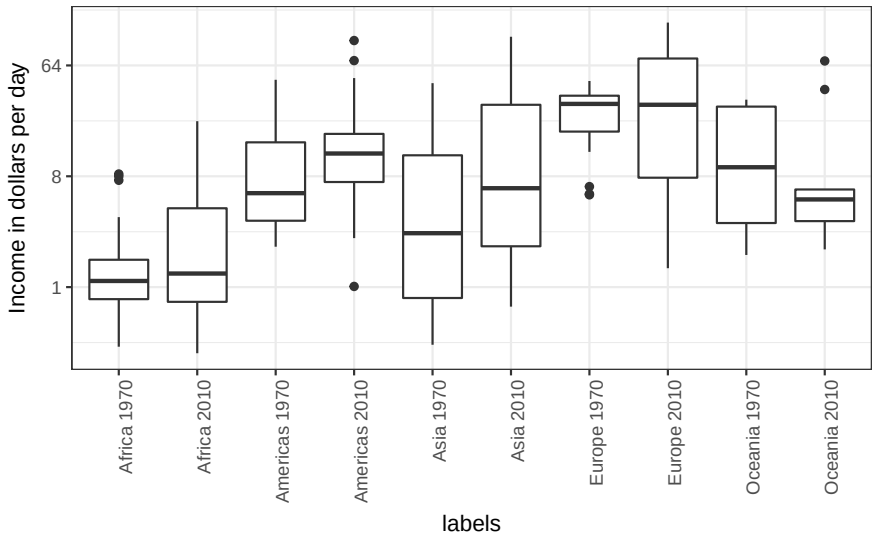
Other transformation you should consider are the logistic transformation, useful to better see fold changes in odds, and the square root transformation, useful for count data.

1.11.9 Ease comparisons: Visual cues to be compared should be adjacent

When comparing income data between 1970 and 2010 across region we made a figure similar to the one below. A difference is that here we look at continents instead of regions, but this is not relevant to the point we are making.

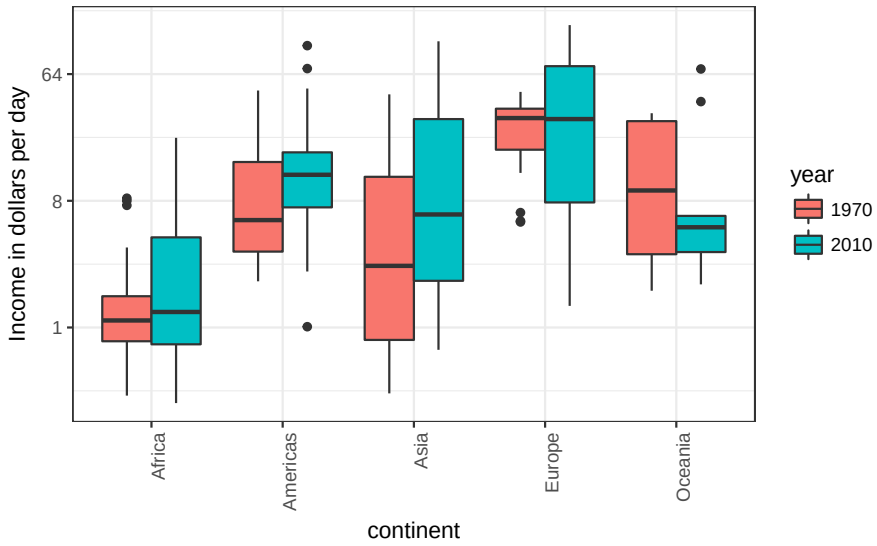


Note that, for each continent, we want to compare the distributions from 1970 to 2010. The default in ggplot is to order alphabetically so the labels with 1970 come before the labels with 2010, making the comparisons challenging. Note how much easier it is to make the comparison when the boxplots are next to each other:



1.11.10 Ease comparison: use color

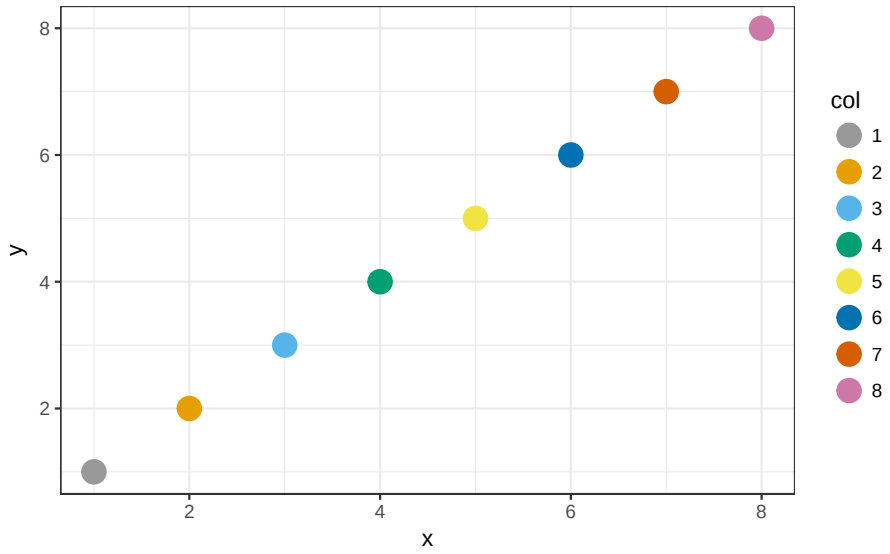
The comparison becomes even easier to make if we use color to denote the two things we want compared.



1.11.11 Think of the color blind

About 10% of the population is color blind. Unfortunately, the default colors used in ggplot are not optimal for this group. However, ggplot does it make it easy to change the color palette used in the plots. Here is an example of how we can use color blind friendly pallet described here:

```
color_blind_friendly_cols <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E
p1 <- data.frame(x=1:8, y=1:8, col = as.character(1:8)) %>% ggplot(aes(x, y, color = col)) + geom_point
p1 + scale_color_manual(values=color_blind_friendly_cols)
```



There are several resources that help you select colors, for example this one.

1.11.12 Use scatter-plots to examine the relationship between two variables

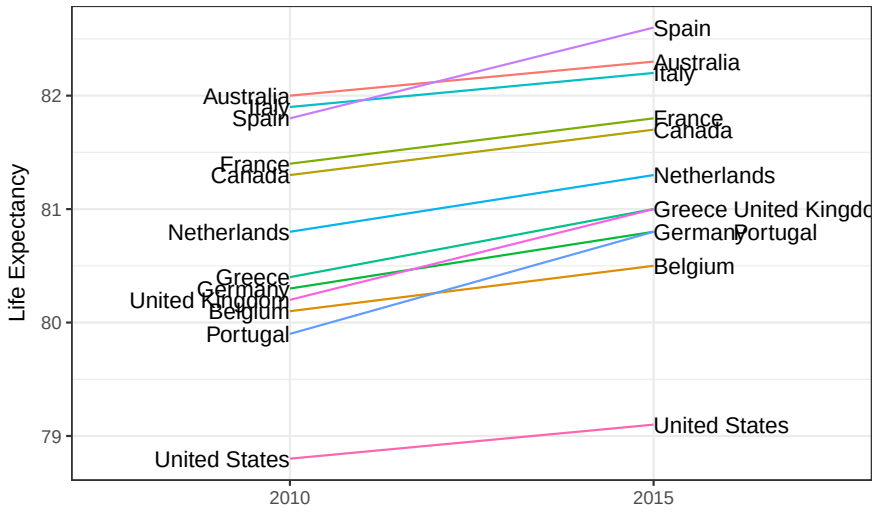
In every single instance in which we have examined the relationship between two variables, total murders versus population size, life expectancy versus fertility rates, and child mortality versus income, we have used scatter plots. This is the plot we generally recommend.

1.11.12.1 Slope charts

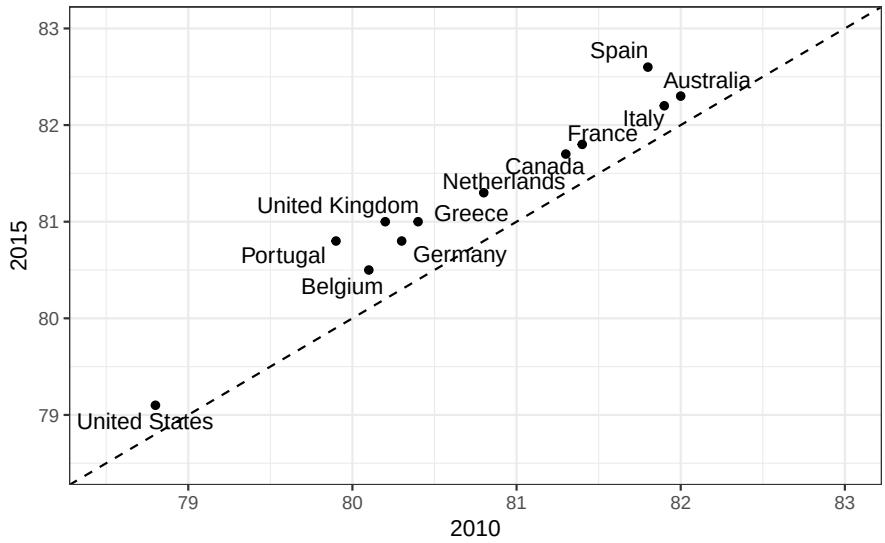
One exception where another type of plot may be more informative is when you are comparing variables of the same type but at different time points and for a relatively small number of comparisons. For example, comparing life expectancy between 2010 and 2015. In this case we might recommend a *slope chart*.

There is not geometry for slope chart in ggplot2 but we can construct one using `geom_lines`. We need to do some tinkering to add labels. Here is a comparisons for large western countries:

```
west <- c("Western Europe","Northern Europe","Southern Europe",
          "Northern America","Australia and New Zealand")

dat <- gapminder %>%
  filter(year%in% c(2010, 2015) & region %in% west &
           !is.na(life_expectancy) & population > 10^7)

dat %>%
  mutate(location = ifelse(year == 2010, 1, 2),
         location = ifelse(year == 2015 & country%in%c("United Kingdom","Portugal"), location+0.22, loca
         hjust = ifelse(year == 2010, 1, 0)) %>%
  mutate(year = as.factor(year)) %>%
  ggplot(aes(year, life_expectancy, group = country)) +
  geom_line(aes(color = country), show.legend = FALSE) +
  geom_text(aes(x = location, label = country, hjust = hjust),
            show.legend = FALSE) +
  xlab("") + ylab("Life Expectancy")
```



An advantage of the slope chart is that it permits us to quickly get an idea of changes based on the slope of the lines. Note that we are using angle as the visual cue. But we also have position to determine the exact values. Comparing the improvements is a bit harder with a scatter plot:



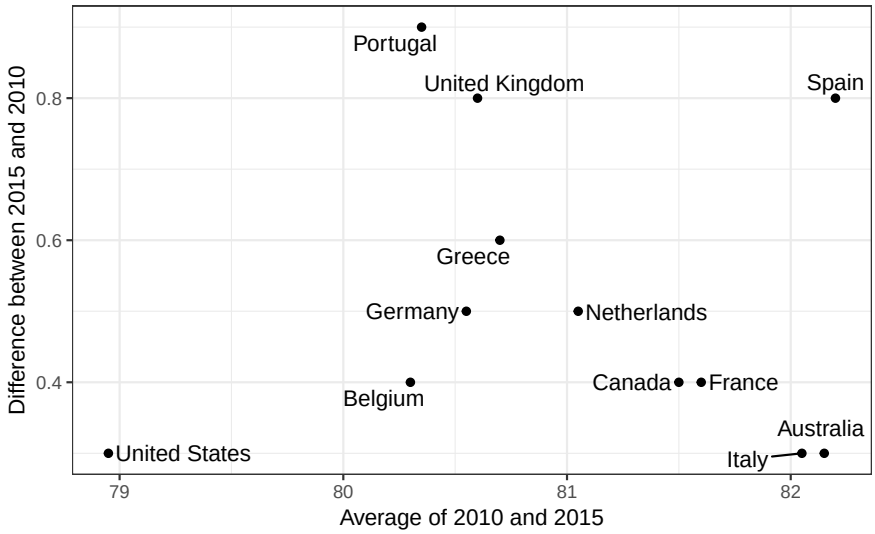
Note that in the scatter plot we have followed the principle *use common axes* since we are comparing these before and after. However, if we have many points the slope charts stop being useful as it becomes hard to

1.11.12.2 Bland-Altman plot

Since what we are interested in the difference, it makes sense to dedicate one of our axes to it. The Bland-Altman plot, also know as the Tukey mean-difference plot and the MA-plot, shows the difference versus the average:

```
library(ggrepel)
dat %>%
   mutate(year = paste0("life_expectancy_", year)) %>%
   select(country, year, life_expectancy) %>% spread(year, life_expectancy) %>%
  mutate(average = (life_expectancy_2015 + life_expectancy_2010)/2,
         difference = life_expectancy_2015 - life_expectancy_2010) %>%
  ggplot(aes(average, difference, label = country)) +
  geom_point() +
```

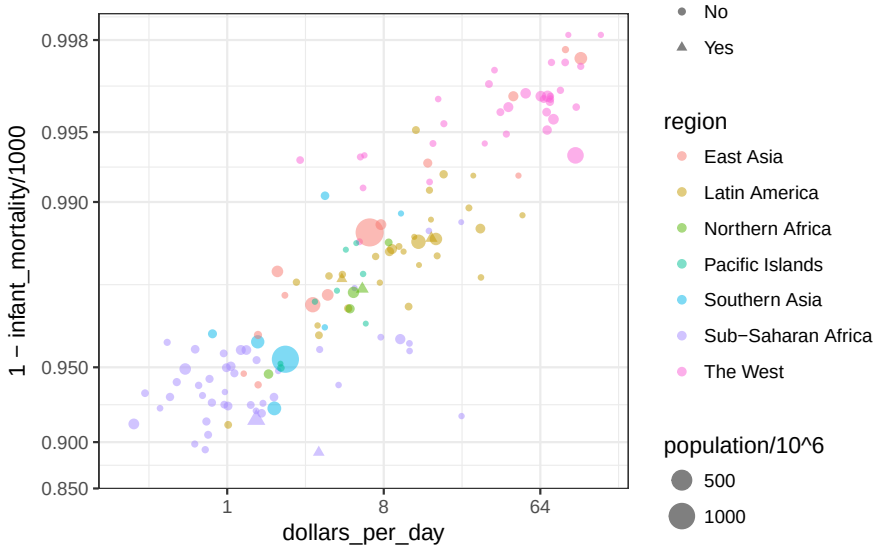
```
  geom_text_repel() +
  geom_abline(lty = 2) +
  xlab("Average of 2010 and 2015") + ylab("Difference between 2015 and 2010")
```



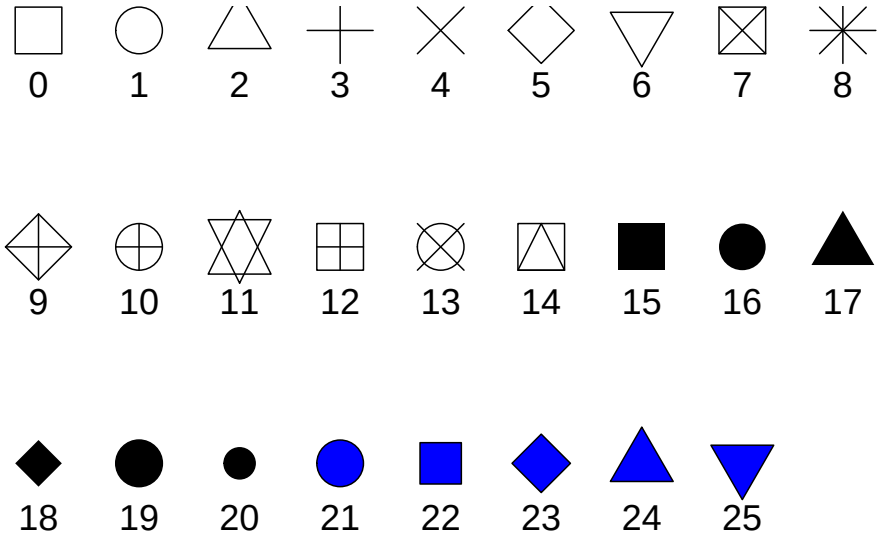
Here we quicky see which countries have improved the most as it is represented by the y-axis. We also get an idea of the overall value from the x-axis.

1.11.13 Encoding a third variable

We previously showed a scatter plot showing the relationship between infant survival and average income. Here is a version of this plot where we encode three variables: OPEC membership, region, and population:



Note that we encode categorical variables with color hue and shape. These shape can be controlled with `shape` argument. Below are the shapes available for use in R. Note that for the last five, the color goes inside.



For continuous variables we can use color, intensity or size. We now show an example of how we do this with a case study.

1.11.13.1 Case Study: Vaccines

Vaccines have helped save millions of lives. In the 19th century, before herd immunization was achieved through vaccination programs, deaths from infectious diseases, like smallpox and polio, were common. However, today, despite all the scientific evidence for their importance, vaccination programs have become somewhat controversial.

The controversy started with a paper published in 1988 and lead by Andrew Wakefield claiming there was a link between the administration of the measles, mumps and rubella (MMR) vaccine, and the appearance of autism and bowel disease. Despite much scientific contradicting this finding, sensationalists media reports and fear mongering from conspiracy theorists, led parts of the public to believe that vaccines were harmful. Some parents stopped vaccinating their children. This dangerous practice can be potentially disastrous given that the Center for Disease Control (CDC) estimates that vaccinations will prevent more than 21 million hospitalizations and 732,000 deaths among children born in the last 20 years (see Benefits from Immunization during the Vaccines for Children Program Era — United States, 1994-2013, MMWR). The1988 paper has since been retracted and Andrew Wakefield was eventually "struck off the UK medical register, with a statement identifying deliberate falsification in the research published in The Lancet, and was thereby barred from practicing medicine in the UK." (source: Wikipedia). Yet misconceptions persist. In part due to self-proclaimed activist that continue to dispel misinformation about vaccines.

Effective communication of data is a strong antidote to misinformation and fear mongering. Earlier we showed a an example provided by a [Wall Street Journal] article](http://graphics.wsj.com/infectious-diseases-and-vaccines/?mc_cid=711ddeb86e) showing data related to the impact of vaccines on battling infectious diseases. Here we reconstruct that example.

The data used for these plots were collected, organized and distributed by the Tycho Project. They include weekly reported counts data for seven diseases from 1928 to 2011, from all fifty states. We include the yearly totals in the `dslabs` package:

```
data(us_contagious_diseases)
str(us_contagious_diseases)
#> 'data.frame':    18870 obs. of  6 variables:
#>  $ disease       : Factor w/ 7 levels "Hepatitis A",..: 1 1 1 1 1 1 1 1 1 1 ...
#>  $ state         : Factor w/ 51 levels "Alabama","Alaska",..: 1 1 1 1 1 1 1 1 1 1 ...
#>  $ year          : num  1966 1967 1968 1969 1970 ...
#>  $ weeks_reporting: int  50 49 52 49 51 51 45 45 45 46 ...
```

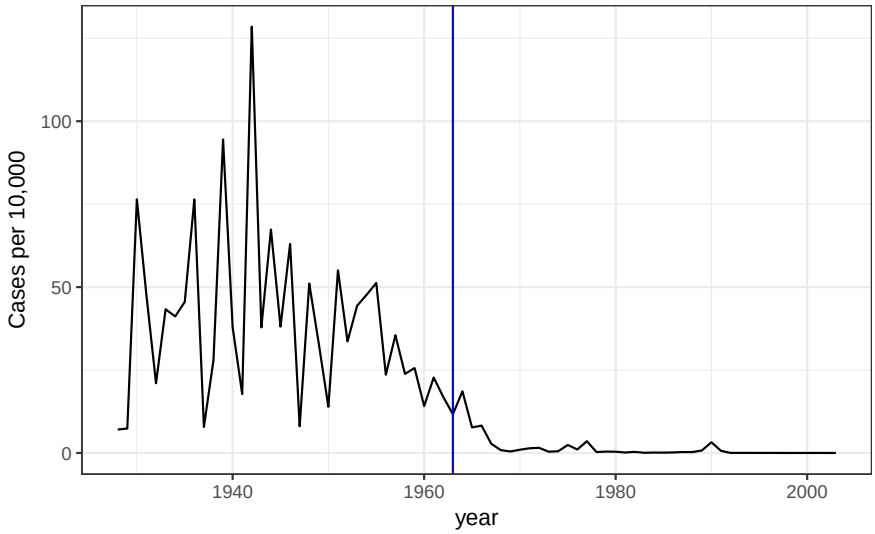
```
#>  $ count          : num  321 291 314 380 413 378 342 467 244 286 ...
#>  $ population     : num  3345787 3364130 3386068 3412450 3444165 ...
```

We create a temporary object `dat` to that stores only the Measles data, includes a per 100,000 rate, orders states by average value of disease and removes Alaska and Hawaii since they only became states in the late 50s.

```
the_disease <- "Measles"
dat <- us_contagious_diseases %>%
  filter(!state%in%c("Hawaii","Alaska") & disease == the_disease) %>%
  mutate(rate = count / population * 10000) %>%
  mutate(state = reorder(state, rate))
```

We can now easily plot disease rates per year. Here are the Measles data from California:

```
dat %>% filter(state == "California") %>%
  ggplot(aes(year, rate)) +
  geom_line() + ylab("Cases per 10,000")  +
  geom_vline(xintercept=1963, col = "blue")
```



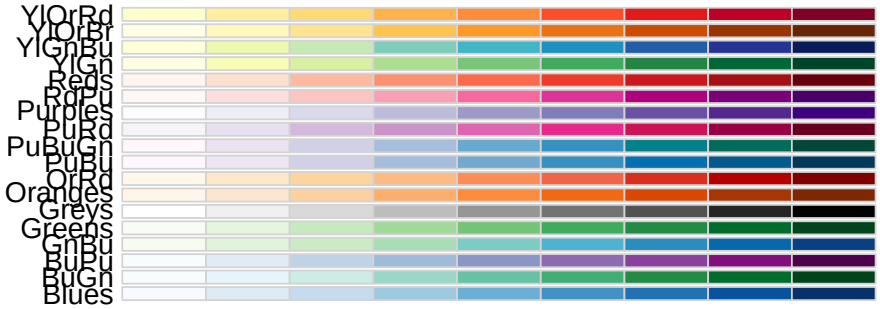
We add a vertical line at 1963 since this is when the vaccine was introduced [Control, Centers for Disease; Prevention (2014). CDC health information for international travel 2014 the yellow book. p. 250. ISBN 9780199948505].

Now, can we show data for all states in one plot? We have three variables to show: year, state and rate. In the WSJ figure they use the x-axis for year, the y-axis for state and color hue to represent rates. However, the color scale they use, which goes from yellow to blue to green to orange to red. can be improved.

When choosing colors to quantify a numeric variable we chose between two options sequential and diverging.

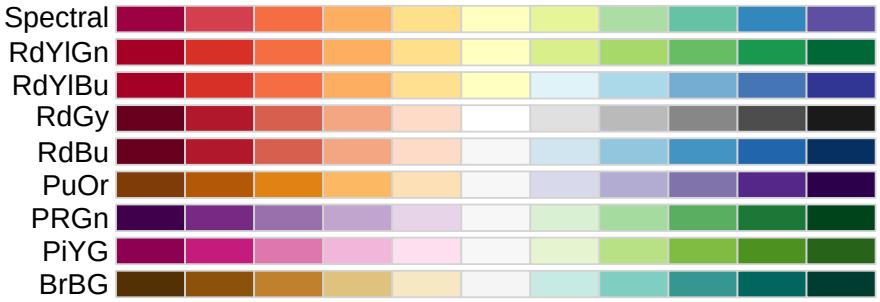
Sequential colors are suited for data that goes from high to low. High values are clearly distinguished from low values. Here are some examples offered by the package `RColorBrewer`

```
library(RColorBrewer)
display.brewer.all(type="seq")
```



Diverging colors are used to represent values that diverge from a center. We put equal emphasis on both ends of the data range: higher than the center and lower than the center. An example of when we would use a divergent pattern would be if we were to show height in standard deviations away from the average. Here are some examples of divergent patterns:

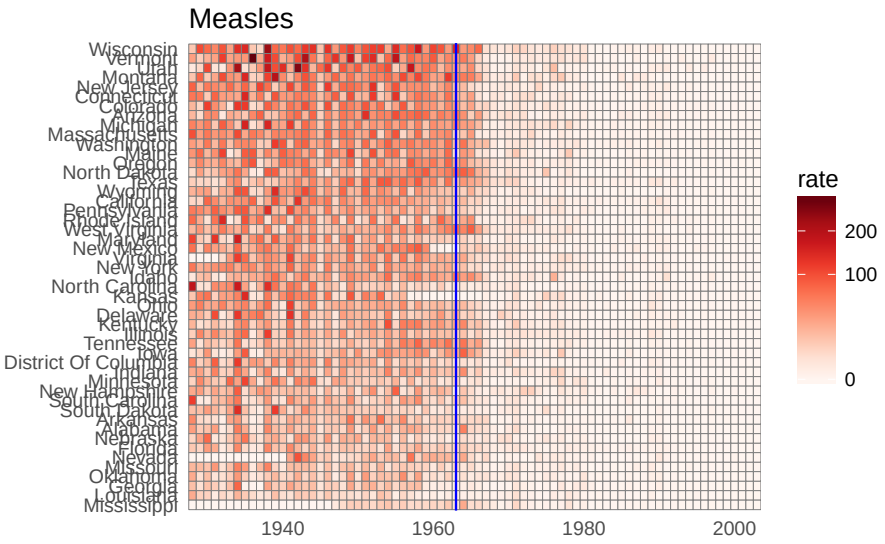
```
library(RColorBrewer)
display.brewer.all(type="div")
```



In our example we want to use a sequential palette since there is no meaningful center, just low and high rates.

We use the geometry `geom_tile` to tile the region with colors representing disease rates. We use a square root transformation to avoid having the really high counts dominate the plot.

```
dat %>% ggplot(aes(year, state,  fill = rate)) +
  geom_tile(color = "grey50") +
  scale_x_continuous(expand=c(0,0)) +
  scale_fill_gradientn(colors = brewer.pal(9, "Reds"), trans = "sqrt") +
  geom_vline(xintercept=1963, col = "blue") +
  theme_minimal() +  theme(panel.grid = element_blank()) +
  ggtitle(the_disease) +
  ylab("") +
  xlab("")
```



This plot makes a very striking argument for the contribution of vaccines. However, one limitation of this plot is that it uses color to represent quantity which we earlier explained makes it a bit harder to know exactly how high it is going. Position and lengths are better cues. If we are willing to lose state information, we can make a version of the plot that shows the values with position. We can also show the average for the US which we compute like this:

```
avg <- us_contagious_diseases %>%
  filter(disease==the_disease) %>% group_by(year) %>%
  summarize(us_rate = sum(count, na.rm=TRUE)/sum(population, na.rm=TRUE)*10000)
```

Now to make the plot we simply use the `geom_line` geometry:

```
dat %>% ggplot() +
  geom_line(aes(year, rate, group = state),  color = "grey50",
            show.legend = FALSE, alpha = 0.2, size = 1) +
  geom_line(mapping = aes(year, us_rate),  data = avg, size = 1, color = "black") +
  scale_y_continuous(trans = "sqrt", breaks = c(5,25,125,300)) +
  ggtitle("Cases per 10,000 by state") +
  xlab("") +
  ylab("") +
  geom_text(data = data.frame(x=1955, y=50), mapping = aes(x, y, label="US average"), color="black") +
  geom_vline(xintercept=1963, col = "blue")
```

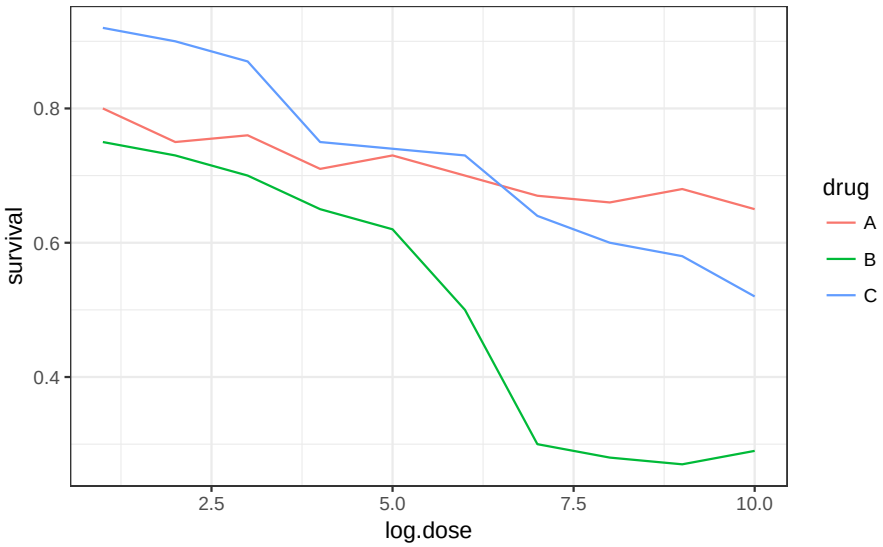
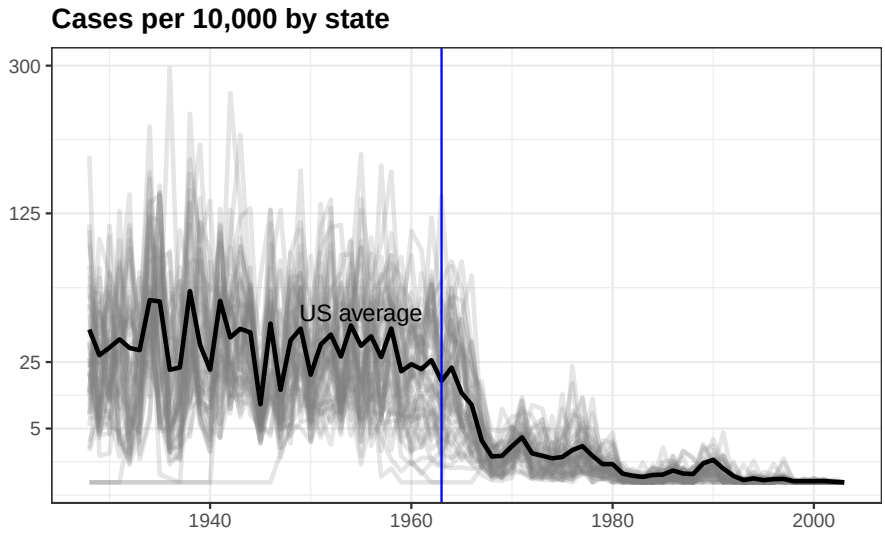


Figure 5: This plot demonstrates that using color is more than enough to distinguish the three lines.



In theory we could use color to represent the categorical value state, but it is hard to pick 50 distinct colors.

1.11.14 Avoid pseudo three dimensional plots

The figure below, taken from the scientific literature [CITE: DNA Fingerprinting: A Review of the Controversy Kathryn Roeder Statistical Science Vol. 9, No. 2 (May, 1994), pp. 222-247] shows three variables: dose, drug type and survival. Although your screen/book page is flat and two dimensional, the plot tries to imitate three dimensions and assigned a dimension to each variable.

```
knitr::include_graphics("https://raw.githubusercontent.com/kbroman/Talk_Graphs/master/Figs/fig8b.png")
```

Humans are not good at seeing in three dimensions (which explains why it is hard to parallel park) and our limitation is even worse when with pseudo-three-dimensions. To see this, try to determine the values of the survival variable in the plot above. Can you tell when the purple ribbon intersects the red one? This is an example in which was easily use color to represent the categorical variable:

Note how much easier it is to determine the survival values.

1.11.15 Avoid gratuitousthree dimensional plots

Pseudo 3D is sometimes used completely gratuitously: plots are made to look 3D even when the 3rd dimension does not represent a quantity. This only adds confusion and makes it harder to relay your message. Here are two examples:

```
knitr::include_graphics("https://raw.githubusercontent.com/kbroman/Talk_Graphs/master/Figs/fig1e.png")
```



```
knitr::include_graphics("https://raw.githubusercontent.com/kbroman/Talk_Graphs/master/Figs/fig2d.png")
```


1.11.15.1 Avoid too any significant digits

By default, statistical software like R returns many significant digits. The default behavior in R is to show 7 significant digits. So many digits often adds no information and the visual clutter than can makes it hard for the consumer of your table to understand the message. As an example here are the per 10,000 disease rates for California across the five decades

state	year	Measles	Pertussis	Polio
California	1940	37.8826320	18.3397861	18.3397861
California	1950	13.9124205	4.7467350	4.7467350
California	1960	14.1386471	0.0000000	0.0000000
California	1970	0.9767889	0.0000000	0.0000000
California	1980	0.3743467	0.0515466	0.0515466

We are reporting precision up to 0.00001 cases per 10,000, a very small value in the context the changes that are occurring across the dates. In this case 2 significant figure is more than enough and makes the point that rates are decreasing clearly:

state	year	Measles	Pertussis	Polio
California	1940	37.9	18.3	18.3
California	1950	13.9	4.7	4.7
California	1960	14.1	0.0	0.0
California	1970	1.0	0.0	0.0
California	1980	0.4	0.1	0.1

Useful ways to change the number of significant digits or to round number are `signif` and `round`. You can define the number of significant digits use globally by siting options like this: ‘

Another principle, related to displaying tables, is to place values being compared on columns rather than rows. Note that our table above is easier to read than this one:

state	disease	1940	1950	1960	1970	1980
California	Measles	37.9	13.9	14.1	1	0.4
California	Pertussis	18.3	4.7	0.0	0	0.1
California	Polio	18.3	4.7	0.0	0	0.1

1.11.16 Know your audience

Graphs can be used for our 1) own exploratory data analysis, 2) to convey a message to experts, or 3) to help tell a story to a general audience. Make sure that the intended audience of your final produce understands each element of the plot.

As a simple example, consider that for your own exploration it may be more useful to log data and then plot. While for a general audience, not familiar with converting logged values back to the original measurements, using a log-scale for the axis will be better.

1.11.17 Further reading:

- ER Tufte (1983) The visual display of quantitative information. Graphics Press.
- ER Tufte (1990) Envisioning information. Graphics Press.
- ER Tufte (1997) Visual explanations. Graphics Press.
- WS Cleveland (1993) Visualizing data. Hobart Press.
- WS Cleveland (1994) The elements of graphing data. CRC Press.
- A Gelman, C Pasarica, R Dodhia (2002) Let's practice what we preach: Turning tables into graphs. The American Statistician 56:121-130.
- NB Robbins (2004) Creating more effective graphs. Wiley.
- Nature Methods columns
- A Cairo (2013) The Functional Art: An Introduction to Information Graphics and Visualization. New Riders
- N Yau (2013) Data Points: Visualization That Means Something. Wiley

Contents

1 Probability

1.1 Introduction

Knowing how to compute probabilities gives you an edge in these games. As a result throughout history many smart people, including famous mathematicians such as Cardano, Fermat and Pascal, spent time and energy thinking through the math of games of chance. As a result, Probability Theory was born. Probability continues to be highly useful in modern games of chance. For example, in poker, we can compute the probability of winning a hand based on the cards on the table. Casino rely on probability theory to develop games that guarantee a profit.

It turns out probability theory is useful in many other contexts and in particular, those that depend on data affected by chance in some way. As a result, knowledge of probability is indispensable for data science.

In games of chance probability has a very intuitive definition, for example we know what it means that the chance of pair dice coming up seven is 1 in 6. However, this is not the case in other contexts. Today probability is being used much more broadly with the word *probability* commonly used in everyday language. Google's auto-complete of "What are the chances of" give us "having twins", "rain today", "getting struck by lightning", and "getting cancer".

We also see the word probability used by election forecasters. In 2008 Nate Silver gave Obama a 94% chance of winning. In 2012 it was a 90%. He won both elections. In 2016 Silver was not as certain, and gave Hillary Clinton only a 71% of winning. She lost. But 71% is still more than 50%. Was Mr. Silver wrong? And what does probability mean in this context anyway? Are there die being tossed somewhere? To answer these questions we will need to learn, among other things, some probability theory. We cover election forecasting in the next chapter, since we also need to understand the concepts of *statitical inference* which we explain in that chapter. But note that statistical inference builds upon probability theory.

The motivation for this chapter are the circumstances surrounding the financial crisis of 2007-2008. Part of what caused this financial crisis was that the risk of certain securities sold by financial institutions was underestimated. Specifically, the risk of mortgage-backed securities (MBS) and collateralize debt obligations (CDO) were grossly underestimated. These MBS and CDO were sold at prices that assumed most homeowners would make their monthly payments, and the probability of this not happening was low. A combination of factors resulted in many more defaults than were expected. This resulted in a price crash of these securities. As a consequence, banks lost so much money that they needed government bailouts to avoid closing down completely.

To begin to understand this very complicated event, we need to understand the basics of probability. We will introduce important concepts such as random variables, independence, Monte Carlo simulations, expected

values, standard errors, and the Central Limit Theorem. Before we start we try to understand the financial crisis using probability, we will use several examples related to games of chance as these are simple and illustrative.

1.2 Discrete Probability

We start by covering some basic principles related to categorical data. The subset of probability is referred to as *discrete probability*. It will help us understand the probability theory we will later introduce for numeric and continuous data, which is much more common in data science applications. Discrete probability is more useful in card games and we use these as examples.

1.2.1 Relative Frequency

The word probability is used in everyday language. For example Google's auto-complete of "What are the chances of" give us "getting pregnant", "having twins", and "rain tomorrow". Answering questions about probability is often hard if not impossible. Here we discuss a mathematical definition of *probability* that does permit us to give precises answers to certain questions.

For example, if I have 2 red beads and 3 blue beads inside an urn and I pick one at random what is the probability of picking a red one? Our intuition tells us that the answer is 2/5 or 40%. A precise definition can be given by noting that there are five possible outcomes of which two satisfy the condition necessary for the even "pick a red bead". Because each of the five outcomes has the same chance of occurring we conclude that the probability is .4 for red and .6 for blue.

A more tangible way way to think about the probability of an event is as the proportion of times the event occurs when we repeat the experiment over and over, independently, and under the same conditions.

1.2.2 Notation

We use the notation $\Pr(A)$ to denote the probability of event A happening. We use the very general term *event* to refer to things that can happen when something happens by chance. For example, in our previous example the event was "picking a red bead". In a political poll in which we call 100 likely voters at random an example of an event is "calling 48 Democrats and 52 Republicans".

In data science applications, we will often deal with continuous variables. In these cases events will often be things like "is this person taller than 6 feet". In this case we write events in a more mathematical form: $X \geq 6$. We will see more of these examples later. Here we focus on categorical data.

1.2.3 Monte Carlo Simulations

Computers provide a way to actually perform the simple random experiment described above: pick a bead at random from a bag with three blue beads and two red ones. Random number generators permit us to mimic the process of picking at random.

An example is the `sample` function in R. We demonstrate its use in the code below. First, we use the function `rep` to generate the urn:

```
beads <- rep( c("red", "blue"), times = c(2,3))
beads
#> [1] "red"  "red"  "blue" "blue" "blue"
```

and then use `sample` to pick a bead at random:

```
sample( beads, 1)
#> [1] "red"
```

This line of code produces one random outcome. We want to repeat this experiment "over and over". However, it is of course impossible to repeat forever. Instead, we repeat the experiment a large enough number of times to make the results practically equivalent. This is an example of a *Monte Carlo* simulation. N

Note that much of what mathematical and theoretical statistician study, something we do not cover int his book, relates to providing rigorous definitions of "practically equivalent" as well as study as how close a large number of experiments gets us to what happens in the limit. Later in this section we provide a practical approach to deciding what is "large enough".

To perform our first Monte Carlo simulation we use the `replicate` function, which permits us to repeat the same task any number of times. Here we repeat the random event $B = 10{,}000$ times:

```
B <- 10000
events <- replicate(B, sample(beads, 1))
```

We can now see if in fact, our definition is in agreement with this Monte Carlo simulation approximation. We can use `table` to see the distribution:

```
tab <- table(events)
tab
#> events
#> blue  red
#> 6053 3947
```

and `prop.table` gives us the proportions:

```
prop.table(tab)
#> events
#>  blue   red
#> 0.605 0.395
```

The numbers above are the estimated probabilities provided by this Monte Carlo simulation. Statistical theory, not covered here, tells us the as B gets larger as the estimates get closer to 3/5=.6 and 2/5=.4.

This is a simple and not very useful example, but we will use Monte Carlo simulation to estimate probabilities in cases in which it is harder to compute the exact ones. Before we go into more complex examples we use simple ones to demonstrate the computing tools available in R.

1.2.4 With and without replacement

The function `sample` has an argument that permits us to pick more than one element from the urn. However, by default, this selection occurs *without replacement*: after a bead is selected, it is not put back in the bag. Note what happens when we ask to randomly select five beads:

```
sample(beads, 5)
#> [1] "red"  "blue" "blue" "red"  "blue"
sample(beads, 5)
#> [1] "blue" "red"  "blue" "blue" "red"
sample(beads, 5)
#> [1] "blue" "red"  "red"  "blue" "blue"
```

This results in re-arrangements that always have three blue and two red beads. If we ask that six beads be selected, we get an error

```
sample(beads, 6)
```

```
Error in sample.int(length(x), size, replace, prob) :    cannot take a sample larger than
the population when 'replace = FALSE'
```

However, the `sample` function can be used directly, without the use of `replicate`, to repeat the same experiment of picking one out of the 5 beads, over and over, under the same conditions. To do this we sample *with replacement*: return the bead back to the urn after selecting it.

We can tell `sample` to do this chanting the `replace` argument , which defaults to `FALSE` to `replace=TRUE`

```
events <- sample(beads, B, replace = TRUE)
prop.table(table(events))
#> events
#>  blue   red
#> 0.599 0.401
```

Note that, not surprisingly, we get results very similar to those previously obtained with `replicate`.

1.2.5 Probability Distributions

Defining a distribution for categorical outcomes is relatively straight forward. We simply assign a probability to each category. In cases that can be thought of as beads in an urn, for each bead type their proportion defines the distribution.

If we are randomly calling likely voters from a population that is 44% Democrat, 44% republican and 10% undecided and 2% green party, these proportions define the probability for each group. The probability distribution is:

$$\Pr(\text{picking a Republican}) = 0.44 \Pr(\text{picking a Democrat}) = 0.44 \Pr(\text{picking an undecided}) = 0.10 \Pr(\text{picking a Green}) = 0.02$$

1.2.6 Independence

We say two events are independent if the outcome of one does not affect the other. The classic example are coin tosses. Every time we toss a fair coin the probability of seeing heads is 1/2 regardless of what previous tosses have revealed. The same is true when we pick beads from an urn with replacement. In the example above the probability of red is 0.40 regardless of previous draws.

Many examples of events that are not independent come from card games. When we deal the first card, the probability of getting a king is 1/13 since there are thirteen possibilities: Ace, Deuce, Three, ..., Ten, Jack, Queen, King and Ace. Now if we deal a king for the first card,and don't replace it into the deck, now the probabilities of getting a king in the second card are less because there are only three kings left: the probability is 3 out of 51. These events are therefore **not independent**. The first outcomes affected the next.

To see an extreme case of non-independent events consider our example of drawing five beads at random **without** replacement:

```
x <- sample(beads, 5)
```

If you have to guess the color of the first bead you predict blue since blue has a 60% chance. But if I show you the result of the last four outcomes:

```
x[2:5]
#> [1] "blue" "blue" "blue" "red"
```

would you still guess blue? Of course not. Now you know that the probability of red 1. The events are not independent so the probabilities changes.

1.2.7 Conditional Probabilities

When events are not independent, *conditional probabilities* are useful. We already saw an example of a conditional probability: we computed the probability that a second dealt card is a king given that the first was a king. In probability we use the following notation

$$\Pr(\text{Card 2 is a king} \mid \text{Card 1 is a king}) = 3/51$$

We use the | as shorthand for "given that" or "conditional on".

Note that when two events, say A and B, are independent we have

$$\Pr(A \mid B) = \Pr(A)$$

This the mathematical way of saying: the fact that B happened does not affect the probability of A happening. In fact, this can be considered the mathematical definition of independence.

1.3 Multiplication rule

If we want to know the probability of two events, say A and B, occurring, we can use the multiplication rule.

$$\Pr(A \text{ and } B) = \Pr(A)\Pr(B \mid A)$$

Let's use Black Jack as an example. In Black Jack you get two assigned two random cards. Then can ask for more. The goal is to get closer to 21 than the dealer, without going over. Face cards are worth 10 points and aces worth 11 or 1 (you choose).

So, in a black jack game, to calculate the chances of getting a 21 in the following way: draw an ace and then a face card, we compute the probability of the first being and ace and multiply by the probability of a face card given that the first was an ace: $1/13 \times 12/52 \approx 0.02$

The multiplicative rule also applies to more than two events. We can use induction to expand for more events:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B \mid A)\Pr(C \mid A \text{ and } B)$$

When we have independent events then the multiplication rule becomes simpler:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B)\Pr(C)$$

But we have to be very careful here, as assuming independence can result in very different, and incorrect, probability calculations when we don't actually have independence.

As an example, imagine a court case in which the suspect was described to have a mustache and a beard. The defendant has a mustache and a beard and the prosecution brings in an "expert" to testify that 1/10 men have beards and 1/5 have mustaches so using the multiplication rule we conclude that only $1/10 \times 1/5$ or 0.02 have both.

But to multiply like this we need to assume independence! The conditional probability of a man having a mustache conditional on them having a beard is .95. So the correct calculation probability is much higher: 0.09.

Note that the multiplication rule also gives us a general formula for computing conditional probabilities:

$$\Pr(B \mid A) = \frac{\Pr(A \text{ and } B)}{\Pr(A)}$$

To illustrate how we use these formulas and concepts in practice we will show several examples related to card games.

1.3.1 Combinations and Permutations

In our very first example we imagined an urn with five beads. To review, to compute the probability distribution of one draw, we simply listed out all the possibilities, there were 5, and then, for each event, counted how many of these possibilities were associated with it. So, for example, because tout of the five possible outcomes, three were blue, the probability of blue is 3/5.

For more complicated examples these computations are not straightforward. For example, what is the probability that if I draw five cards without replacement I get all cards of the same suit, what is called a flush in poker? Discrete probability teach us how to make this computations, here we focus on how to use R code.

First let's construct a deck of cards. For this we will use the `expand.grid` and `paste` functions. We use `paste` to create strings by joining smaller strings. For example, if we have the number of suit of card we can get the card name like this:

```
number <- "Three"
suit <- "Hearts"
paste(number, suit)
#> [1] "Three Hearts"
```

It also works on pairs of vectors performing the operation element-wise:

```
paste(letters[1:5], as.character(1:5))
#> [1] "a 1" "b 2" "c 3" "d 4" "e 5"
```

The function `expand.grid` gives us all the combinations of two lists. For example if you have blue and black pants and white, grey and plaid shirts all your combinations are:

```
expand.grid(pants = c("blue", "black"), shirt = c("white", "grey", "plaid"))
#>   pants shirt
#> 1  blue white
#> 2 black white
#> 3  blue  grey
#> 4 black  grey
#> 5  blue plaid
#> 6 black plaid
```

So here is how we generate a deck of cards:

```
suits <- c("Diamonds", "Clubs", "Hearts", "Spades")
numbers <- c("Ace", "Deuce", "Three", "Four","Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Q
deck <- expand.grid( number=numbers, suit=suits)
deck <- paste( deck$number, deck$suit)
```

With the deck constructed, we can now double check that the probability of a king in the first card is 1/13. We simply compute the proportion of possible outcomes that satisfy our condition:

```
kings <- paste("King", suits)
mean(deck %in% kings)
#> [1] 0.0769
```

which is 1/13.

Now, how about the conditional probability of the second card being a king given that the first was a king ? Earlier we deduced that if one king is already out of the deck and there are 51 left then this probability is 3/51.

But let's confirm by listing out all possible outcomes. To do this we can use the `permutatiosn` function from the `gtools` package. This function computes, for any list of size `n` all the different ways we can select `r` items. So here are all the ways we can chose 2 numbers from a list of 5:

```
library(gtools)
permutations(3, 2)
#>      [,1] [,2]
#> [1,]    1    2
#> [2,]    1    3
#> [3,]    2    1
#> [4,]    2    3
#> [5,]    3    1
#> [6,]    3    2
```

Notice that the order matters here: 3, 1 is different than 1,3. Also (1,1), (2,2) and (3,3) don't appear because once we pick a number it can't appear again.

Optionally, we can add a vector. So if you want to see five random seven digit phone numbers, out of all possible phone numbers you could type:

```
all_phone_numbers <- permutations(10, 7, v = 0:9)
n <- nrow(all_phone_number)
index <- sample(n, 5)
all_phone_number[index,]
```

To compute all possible ways we can chose two cards, when the order matters, we type:

```
hands <- permutations(52, 2, v = deck)
```

This is a matrix with dimensions two columns and 2652 rows. We can get the first and second card like this:

```
first_card <- hands[,1]
second_card <- hands[,2]
```

Now the cases for which the first hand was a king can be computed like this:

```
sum(first_card %in% kings)
#> [1] 204
```

To get the conditional probability we compute what fraction of these have a king in the second card:

```
sum(first_card %in% kings & second_card %in% kings) /
  sum(first_card %in% kings)
#> [1] 0.0588
```

which is exactly 3/51 as we had already deduced. Note that the code above equivalent to

```
mean(first_card %in% kings & second_card %in% kings) /
  mean(first_card %in% kings)
#> [1] 0.0588
```

which is a R version of

$$\frac{\Pr(A \text{ and } B)}{\Pr(A)}$$

Now what if the order does not matter? For example in Black Jack if you get an Ace and face card it is called a *Natural 21* and you win automatically. If we want to compute the probability of this happening we want to enumerate the *combinations* not permutations since the order does not matter. Note the differences:

```
permutations(3,2)
#>      [,1] [,2]
#> [1,]    1    2
#> [2,]    1    3
#> [3,]    2    1
#> [4,]    2    3
#> [5,]    3    1
#> [6,]    3    2
combinations(3,2)
#>      [,1] [,2]
#> [1,]    1    2
#> [2,]    1    3
#> [3,]    2    3
```

In the second line the outcome does not include (2,1) because (1,2) already was enumerated. Similarly for (3,1) and (3,2).

So to compute the probability of a *Natural 21* in Blackjack we can do this:

```
aces <- paste("Ace", suits)

facecard <- c("King", "Queen", "Jack", "Ten")
facecard <- expand.grid( number=facecard, suit=suits)
facecard <- paste( facecard$number, facecard$suit)

hands <- combinations(52, 2, v = deck)
mean(hands[,1] %in% aces & hands[,2] %in% facecard)
#> [1] 0.0483
```

Note that in the last line we assume the ace comes first. This is only because we know the way `combination` generates enumerates possibilities and it will list this case. But to be safe we could have written this and get the same answer:

```
mean((hands[,1] %in% aces & hands[,2] %in% facecard) |
       (hands[,2] %in% aces & hands[,1] %in% facecard))
#> [1] 0.0483
```

1.3.1.1 Monte Carlo Example

Instead of using `combinations` to deduce the exact probability of a Natural 21 we can use a Monte Carlo to estimate this probability. In this case we draw two cards over and over and keep track of how many 21s we get. We can use the function sample to draw to cards without replacements:

```
hand <- sample(deck, 2)
hand
#> [1] "Eight Spades" "Ten Spades"
```

And then check if one card is an ace and the other a face card or a 10. Going forward we include 10 when we say *face card*. Now we need the check both possibilities

```
(hands[1] %in% aces & hands[2] %in% facecard) |
  (hands[2] %in% aces & hands[1] %in% facecard)
#> [1] FALSE
```

We repeat this 10,000 times and get a very good approximation of the probability of a Natural 21.

```
B <- 10000
results <- replicate(B, {
```

```
  hand <- sample(deck, 2)
  (hand[1] %in% aces & hand[2] %in% facecard) |
    (hand[2] %in% aces & hand[1] %in% facecard)
})
mean(results)
#> [1] 0.0488
```

Note that here we do have to check both possibilities: Ace first or Ace second because we are not using the `combinations` function.

1.3.2 Birthday Problem

Support you are in a classroom with 50 people. If we assume this a randomly selected group of 50 people, what is the chance that at least two people have the same birthday? Although it is somewhat advanced, we can deduce this mathematically. We do this later. Using a Monte Carlo simulation it is not that complicated. For simplicity, we assume nobody was born on February 29. This actually doesn't change the answer much.

First note that birthdays can be represented as numbers between 1 and 365, so a sample of 50 birthdays can be obtained like this:

```
n <- 50
bdays <- sample(1:365, n, replace = TRUE)
```

To check if in this particular set of 50 people we have at least two with the same birthday we can use the function `duplicated` which returns `TRUE` whenever an element of a vector is a duplicate. Here is an example:

```
duplicated(c(1,2,3,1,4,3,5))
#> [1] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
```

The second time 1 and 3 appear we get a `TRUE`. So to check if two birthdays were the same we simply use the `any` and `duplicated` functions like this:

```
any(duplicated(bdays))
#> [1] TRUE
```

In this case, we see that it did happen. At least two people had the same birthday.

To estimate the probability we repeat this experiment, sample 50 birthdays, over and over:

```
B <- 10000
results <- replicate(B,{
  bdays <- sample(1:365, n, replace=TRUE)
  any(duplicated(bdays))
  })
mean(results)
#> [1] 0.97
```

Where you expecting it to be this height?

People tend to underestimate these probabilities. To get an intuition as to why it is so high think about what happens when the group is close to 365. At this stage we run out of days and the probability is one.

1.3.3 sapply: a better way to do for loops

Say we want to use this knowledge to bet with friends about two people having the same birthday is a group of people. When are the chances larger 50%? Larger the 75%?

Let's create a look-up table. We can quickly create a function to compute this for any group sizes:

```
compute_prob <- function(n, B=10000){
  same_day <- replicate(B, {
    bdays <- sample(1:365, n, replace=TRUE)
    any(duplicated(bdays))
  })
  mean(same_day)
}
```

And now we can use for-loop to run it for several group size:

```
n <- seq(1,60)
```

Now, for-loops are rarely the preferred approach in R. In general, we try to perform operations on entire vectors. Arithmetic operations, for example, operate on vectors in a element-wise fashion:

```
x <- 1:10
sqrt(x)
#>  [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
y <- 1:10
x*y
#>  [1]   1   4   9  16  25  36  49  64  81 100
```

No need for for-loops. But not all functions work this way. For example, the function we just wrote does not work element-wise since it is expecting a scalar. This piece of code does not run the function on each entry of `n`:

```
compute_prob(n)
```

The function `sapply` permits us to perform element-wise operations on any function. Here is how it works:

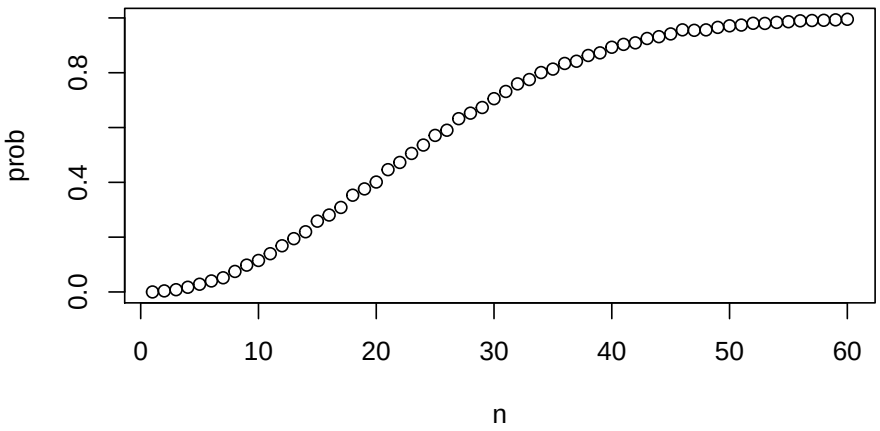
```
x <- 1:10
sapply(x, sqrt)
#>  [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

It loops through the elements of the first argument of `sapply` and sends those as values to first argument of the function specified as the second argument to `sapply`. So for our case we can simply type:

```
prob <- sapply(n, compute_prob)
```

We can now make a plot of the estimated probabilities of two people having the same birthday in a group of size n:

```
prob <- sapply(n, compute_prob)
plot(n, prob)
```



Now let's compute the exact probabilities rather than use Monte Carlo approximations. Not only do we get the exact answer using math but the computations are much faster since we don't have to generate experiments.

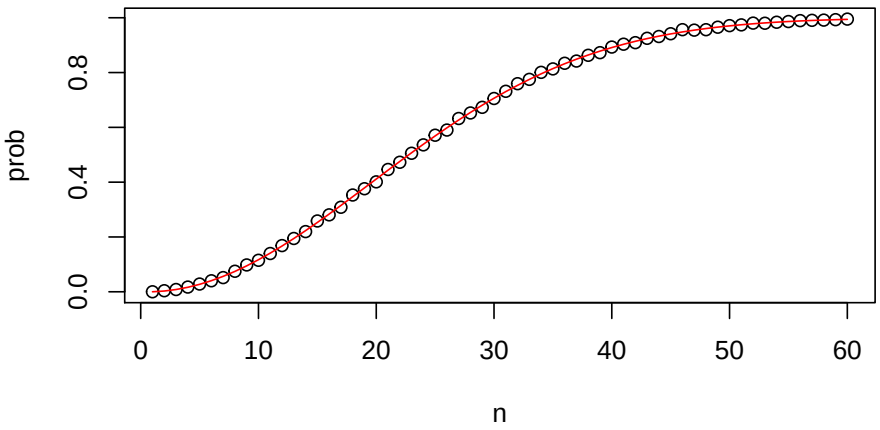
To make the math simpler, instead of computing the probability of it happening we will compute the probability of it not happening. We use the multiplication rule.

Let's start with the first person. The probability that person 1 has a unique birthday is 1. The probability that the person 2 has a unique birthday, given that person 1 already took one is 364/365. Then for person 3, given the first two have unique birthdays, leaves 363 days to choose from. We continue this way and find the chances of all 50 people having a unique birthday is:

$$1 \times \frac{364}{365} \times \frac{363}{365} \dots \frac{365-n+1}{365}$$

We can write a function that does this for any number.

```
exact_prob <- function(n){
  prob_unique <- seq(365,365-n+1)/365
  1 - prod( prob_unique)
}
eprob <- sapply(n, exact_prob)

plot(n, prob)
lines(n, eprob, col = "red")
```



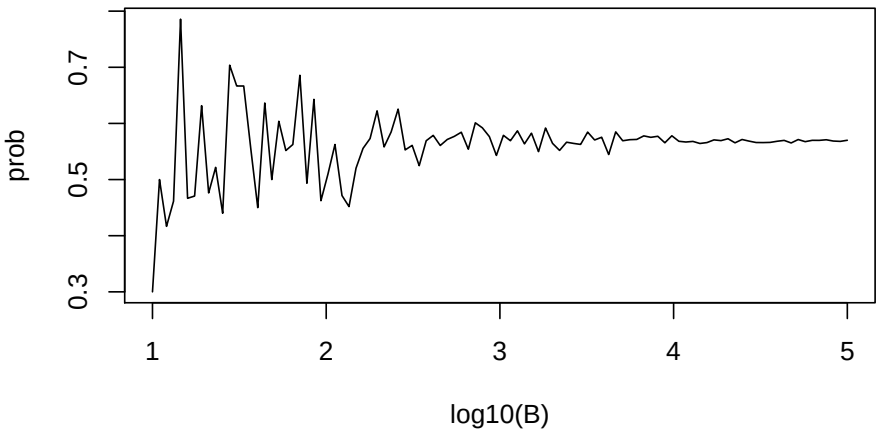
This plot shows that the Monte Carlo simulation provided a very good estimate of the exact probability. Had it not been possible to compute the exact probabilities we would have still been able accurately estimate the probabilities

1.3.4 How many Monte Carlo experiments are enough

In the examples above we used \$B=\$10,000 Monte Carlo experiments. It turns out that this provided very accurate estimates. But in more complex calculation 10,000 may not nearly enough. Also for some calculations 10,000 experiments might not be computationally feasible. In practice we won't know what the answer is so we won't know if our Monte Carlo estimate is accurate. We know that the larger B the better the approximation but how big do we need it to be? This is actually a challenging questions and answering it often requires advanced theoretical statistics training.

One practical approach we will describe her is to check for the stability of the estimate. Here is an example with the birthday problem for a group of 22 people.

```
B <- 10^seq(1, 5, len = 100)
compute_prob <- function(B, n=25){
  same_day <- replicate(B, {
    bdays <- sample(1:365, n, replace=TRUE)
    any(duplicated(bdays))
  })
  mean(same_day)
}
prob <- sapply(B, compute_prob)
plot(log10(B), prob, type = "l")
```



In this plot we can see that the values start to stabilize, vary less than .01, around 1000. Note that the exact probability, which we know in this case, is 0.569.

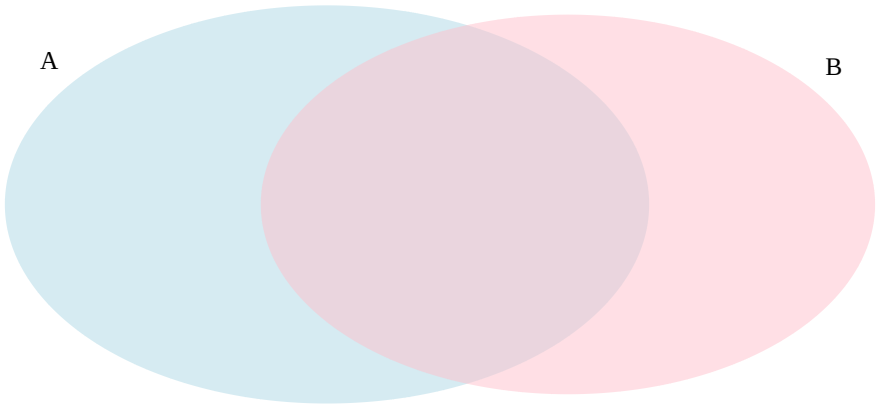
1.3.5 Addition Rule

Another way to compute the probability of a Natural 21 is to notice that it is the probability of an ace followed by facecard or a face card followed by an ace. Here we use the addition rule

$$ Pr(A or B) = Pr(A) + Pr(B) - Pr(A and B)

$$ This rule is intuitive: think of a venn diagram. If we simply add the probabilities we count the intersection twice.

```
#> Loading required package: grid
#> Loading required package: futile.logger
#>
#> Attaching package: 'futile.logger'
#> The following object is masked from 'package:gtools':
#>
#>     scat
#> (polygon[GRID.polygon.1], polygon[GRID.polygon.2], polygon[GRID.polygon.3], polygon[GRID.polygon.4], t
```



In the case of Natural the intersection is empty since both hands can't happen simulaneously. The probability of ace followed by a face card to be $1/13 \times 16/51$ and the probability of a face card followed by an ace is $16/52 \times 4/51$. These two are actually the same which makes sense due to symmetry. In any case we get the same result using the addititon rule:

```
1/13*16/51 + 16/52*4/51 + 0
#> [1] 0.0483
```

1.3.6 Monty Hall Problem

In the 1970s there was a game show called "Let's Make a Deal". Monty Hall was the host. At some point in the game contestant were asked to pick one of three doors. Behind one door there was a prize. There other doors had goat to show you lost. If the contestand did no pick the prized door his or her first try, Monty Hall would open one of the two remaining doors and show the contestant there was no prize. Then he would and ak "Do you want to switch doors?" What would you do?

We can use probability to show that if you stick to the original door your chances of winning a car 1 in 3 but if you switch your chances double to 2 in 3! This seems counterintuitive. Many people incorreclty think both chances are in 1 in 2 since you are chosing between 2. You can watch a detailed explanation here or read one here. Here we use a Monte Carlo simulation to see which strategy is better. Note that this code is written longer than it should be for pedagocial purposes.

Let's start with the stick strategy

```
B <- 10000
stick <- replicate(B, {
  doors <- as.character(1:3)
  prize <- sample(c("car","goat","goat"))
  prize_door <- doors[prize == "car"]
  my_pick  <- sample(doors, 1)
  show <- sample(doors[!doors %in% c(my_pick, prize_door)],1)
  stick <- my_pick
  stick == prize_door
})
mean(stick)
#> [1] 0.336
```

As we write the code we note that the lines starting with `my_pick` and `show` have no influence on the last logical operation. From this we should realize that the chance is 1 in 3, what we started out with.

Now let's repeat the exercise but consider the switch strategy:

```
switch <- replicate(B, {
  doors <- as.character(1:3)
```

```
  prize <- sample(c("car","goat","goat"))
  prize_door <- doors[prize == "car"]
  my_pick  <- sample(doors, 1)
  show <- sample(doors[!doors %in% c(my_pick, prize_door)], 1)
  stick <- my_pick
  switch <- doors[!doors%in%c(my_pick, show)]
  switch == prize_door
})
mean(switch)
#> [1] 0.672
```

The Monte Carlo estimate confirms the 2/3 calculation. The helps us gain some insight by showing that we are removing a door, `show`, that is definitly not a winner from our choices. We also see that unless we get it right when we first pick win: 1 - 1/3 = 2/3.

1.4 Continuous Probability

We earlier explained why when summarizing a list of numeric values, such as heights, it is not useful to construct a distribution that defines a proportion to each possible outcome. Note, for example, that if we measure every single person in a very large population with extremely high precision, because no two people are exactly the same height we need to assign a proportion to each observed value and attain no useful summary at all. Similarly, when defining probability distributions, it is not useful to assign a very small probability to every single height. Just like when using distributions to summarize numeric data, it is much more practical to define a function that operates on intervals rather than single values. The standard way of doing this is using the *cumulative distribution functions* (CDF).

We previously described empirical cumulative distribution function (eCDF) as a basic summary of a list of numeric values. As an example, we defined the height distribution for male student adults. Here we define the vector x to contain the male heights:

```
library(tidyverse)
library(dslabs)
data(heights)
x <- heights %>% filter(sex=="Male") %>% .$height
```

We defined the empirical distribution function as

```
F <- function(a) mean(x<=a)
```

which, for any value `a`, gives the proportion of values in the list `x` that are smaller or equal than `a`.

Note that we have not yet introduced probability. Let's do this by asking, if I pick one of the male students at random, what is the chance that he is taller than 70.5 inches? Because every student has the same chance of being picked the answer to this is equivalent to the proportion of students that are taller than 70.5 feet. Using the CDF we obtain an answer by typing:

```
1 - F(70)
#> [1] 0.371
```

Once a CDF is defined, we can use this to compute the probability of any subset. For example the probability of a student being between height `a` and height `b` is

```
F(b)-F(a)
```

Because we can compute the probability for any possible event this way, the cumulative probability function defines the probability distribution for picking a height at random from our vector of heights `x`.

1.4.1 Theoretical distribution

In the data visualization chapter we introduced the normal distribution as a useful approximation to many naturally occurring distribution, including that of height. The cumulative distribution for the normal distribution is defined by a mathematical formula which in R can be obtained with the function `pnorm`. We say that a random quantity is normally distributed with average `avg` and standard deviation `s` if it's probability distribution is defined by

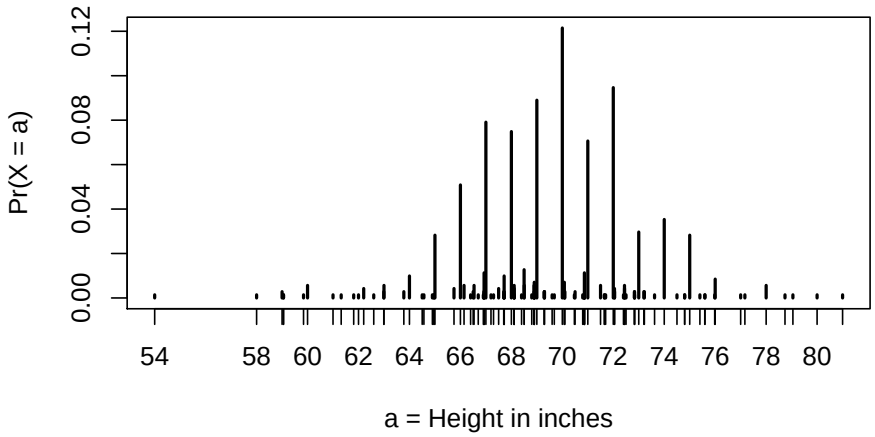
```
F(a) = rnorm(a, avg, s)
```

This is useful because if we are willing to use the normal approximation for, say, height we don't need the entire data set to answer questions such as "what is the probability that a randomly selected student is taller then 70 inches?". We just need the average height and standard deviation:

```
avg <- mean(x)
s <- sd(x)
1 - pnorm(70.5, avg, s)
#> [1] 0.374
```

1.4.2 Approximations

The normal distribution is derived mathematically: we do no need data to define it. For a practicing data scientists, pretty much everything we do involves data. Data is always, technically speaking, discrete. For example we could consider our adult data categorical with each specific height a unique category. The probability distribution is defined by the proportion of students reporting each height. Here is a plot of that probability distribution:



While most students rounded up their heights to the nearest inch, others reported values with more precision. For example, one student reported his height to be 69.6850393700787 which is 177 centimeters. The probability assigned to this height is 0.001 or 1 in 708. The probability of for 70 inches is much higher 0.121, but does it really make sense to think of that the probability of being exactly 70 inches is the same as being 69.6850393700787? Clearly it is much more useful for data analytic purposes to treat this outcome as a continuous numeric variable, keeping in mind that very few people, or perhaps none, are exactly 70 inches, but rather that people round to the nearest inch.

With continuous distributions the probability of a singular value is not even defined. For example it does not make sense to ask what is the probability that a normally distributed value is 70. Instead we define probabilities for intervals. So we could ask what is the probability that someone is between 69.99 and 70.11.

In cases like height, in which the data is rounded, the normal approximation is particularly useful if we deal with intervals that include exactly one round number. So for example, the normal distribution is useful for approximating the proportion of students reporting between 69.5 and 70.5:

```
mean(x <= 68.5) - mean(x <= 67.5)
#> [1] 0.12
mean(x <= 69.5) - mean(x <= 68.5)
#> [1] 0.124
mean(x <= 70.5) - mean(x <= 69.5)
#> [1] 0.14
```

Note how close we get with the normal approximation:

```
pnorm(68.5, avg, s) - pnorm(67.5, avg, s)
#> [1] 0.11
pnorm(69.5, avg, s) - pnorm(68.5, avg, s)
#> [1] 0.12
pnorm(70.5, avg, s) - pnorm(69.5, avg, s)
#> [1] 0.12
```

However, the approximation is not as useful for other intervals. For example note how the approximation breaks when we try to estimate

```
mean(x <= 70.9) - mean(x<=70.1)
#> [1] 0.024
```

with

```
pnorm(70.9, avg, s) - pnorm(70.1, avg, s)
#> [1] 0.0923
```

In general we call this situation as discretization. Although the true height distribution is continuous, the reported heights tend to be more common at discrete values, in this case, due to rounding. As long as we are aware of how deal with this reality, the normal approximation can still be a very useful tool.

1.4.3 The probability density

For categorical distributions we can define the probability of a category. For example a roll of a die, let's call it X, can be 1,2,3,4,5 or 6. The probability of 4 is defined as

$$\Pr(X = 4) = 1/6$$

The CFD can then easily be defend:

$$F(4) = \Pr(X \leq 4) = \Pr(X = 4) + \Pr(X = 3) + \Pr(X = 2) + \Pr(X = 1)$$

Although for continuous distributions the probability of a single value $\Pr(X = x)$ is not defined there is a theoretical definition that has a similar interpretation. The probability density at x is defined as the function $f(a)$ such that

$$F(a) = \Pr(X \leq a) = \int_{-\infty}^{a} f(x)\, dx$$

For those that know Calculus, remember that the integral is related to a sum: it is the sum of bars with widths approximating 0. If you don't know Calculus you can think of $f(x)$ as a curve for which the area under that curve up the value a gives you the probability of $X \leq a$.

For example to use the normal approximation to estimate the probability of someone being taller than 76 inches we use

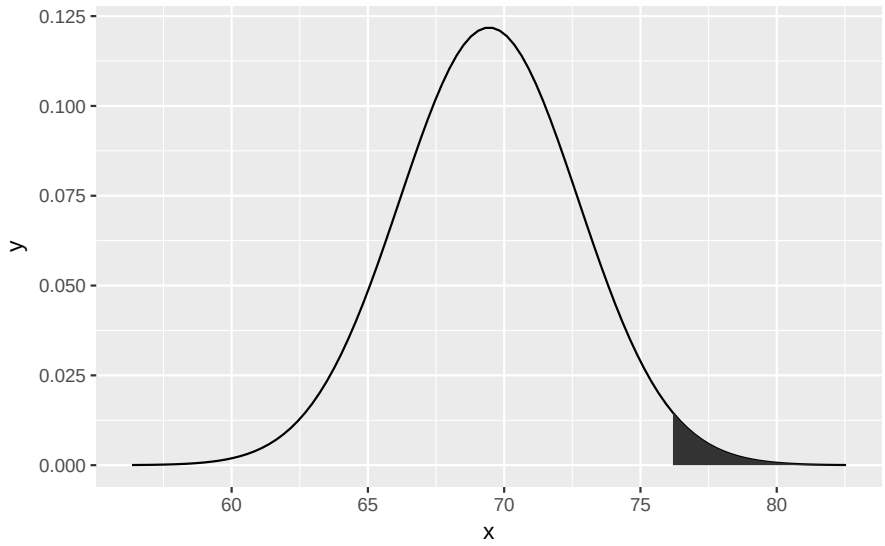


Figure 1: Integral of a function.

```
1 - pnorm(76, avg, s)
#> [1] 0.0227
```

which mathematically is the grey are below:

The curve you see is the probability density for the normal distribution. In R using the function `dnorm`.

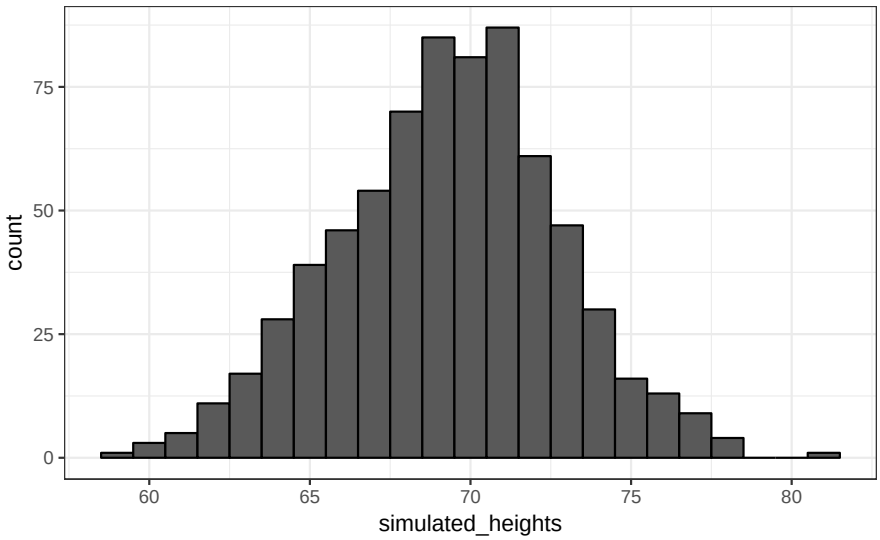
Although it may not be immediately obvious why knowing about probability densities is useful, understanding this concept will be essential to those wanting to fit models to data for which predefined functions are not available.

1.4.4 Monte Carlo simulations

R provides functions to generate normally distributed outcomes. Specifically, the `rnorm` function takes three arguments: size, average (defaults to 0), and standard deviation (defaults to 1) and produced these random numbers. Here is an example of how we could generate data that looks like our reported heights:

```
n <- length(x)
avg <- mean(x)
s <- sd(x)
simulated_heights <- rnorm(n, avg, s)
```

Not surprisingly the distribution looks normal:



This is one of the most useful functions in R as it will permit us to generate data that mimics natural events and answer questions related to what could happen by chance by running Monte Carlo Simulations.

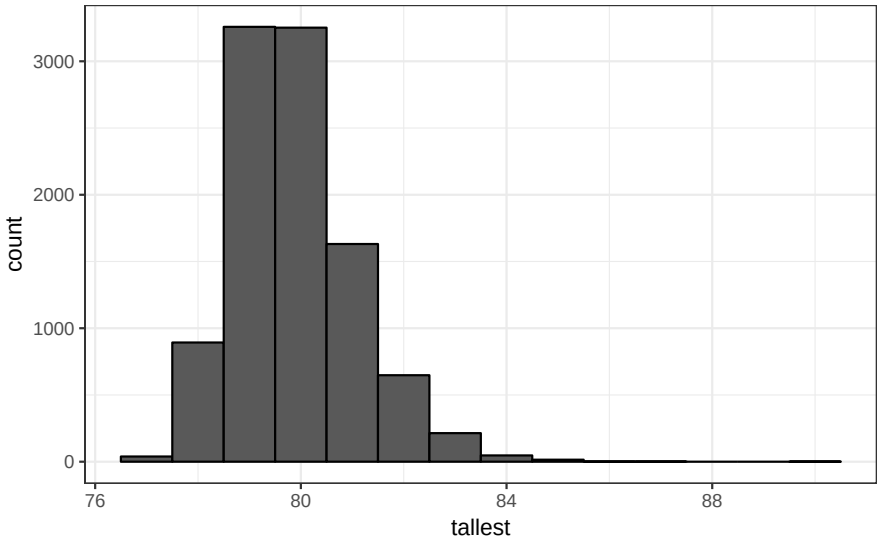
For example, if we pick 800 males at random, what is the distribution of the tallest person? How rare is a seven footer? The following Monte Carlo helps us answer that question:

```
B <- 10000
tallest <- replicate(B, {
  simulated_data <- rnorm(800, avg, s)
  max(simulated_data)
})
```

A seven footer is quite rare:

```
mean(tallest >= 7*12)
#> [1] 0.0034
```

Here is the resulting distribution:

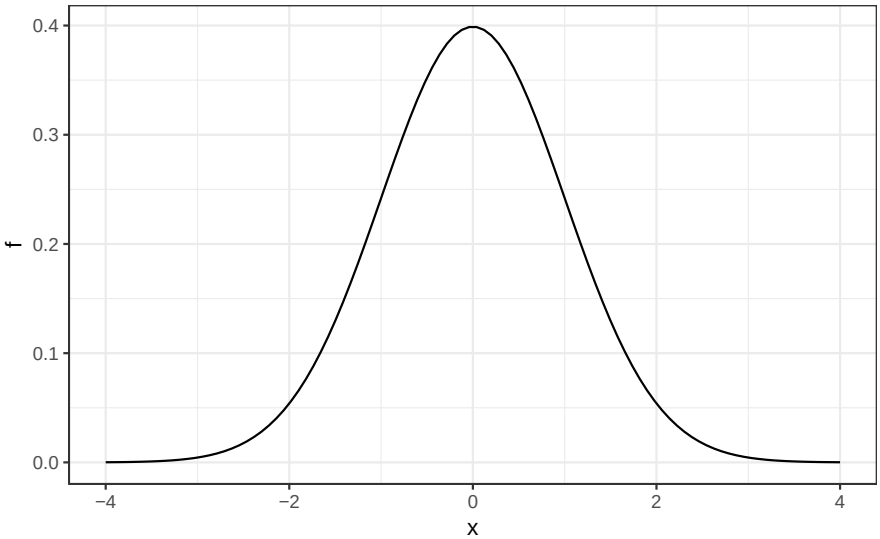


Note that it does not look normal.

1.4.5 Other Continuous Distributions

The normal distribution is not the only useful theoretical distribution. Other continuous distribution that we may encounter are the student-t, chi-squared, exponential, gamma, beta, and beta-binomial. R provides functions to compute the density, the quantiles, the cumulative distribution functions and to generate Monte Carlo simulations. R uses a convention that let's use remember the names. Namely using the letters `d`, `q`, `p` and `r` in front of a shorthand for the distribution. We have already seen the functions `dnorm`, `pnorm` and `rnorm` for the normal distribution. The functions `qnorm` gives us the quantiles. For example, we can draw a distribution like this:

```
x <- seq(-4, 4, length.out = 100)
data.frame(x, f = dnorm(x)) %>% ggplot(aes(x, f)) + geom_line()
```



For example for the student-t the shorthand `t` is used so the functions are `dt` for the density, `qt` for the quantiles, `pt` for the cumulative distribution function, and `rt` for Monte Carlos simulation.

1.5 Setting the random seed

Before we continue, we briefly explain the following important line of code:

```
set.seed(1)
```

Throughout this book, we use random number generators. This implies that many of the results presented can actually change by chance. This implies that a frozen version of the book may show a different result than what you obtain when you try to code shown in the book. This is actually fine since the results are random and change from time to time. However, if we want to to ensure that results are exactly the same every time you run them you can set R's random number generation seed to a specific number. Above we set it to 1.

```
?set.seed
```

In the exercises we may ask you to set the seed to assure that the results you obtain are exactly what we expect them to be.

1.6 Random Variables

Random variable are numeric outcomes resulting from a random processes. We can easily generate random variables using some the simple examples we have shown. For example define `X` to be 1 if a bead is blue and

red otherwise

```
beads <- rep( c("red", "blue"), times = c(2,3))
X <- ifelse(sample(beads, 1) == "blue", 1, 0)
```

Here `X` is a random variable: every time we select a new bead the outcome changes randomly:

```
ifelse(sample(beads, 1) == "blue", 1, 0)
#> [1] 0
ifelse(sample(beads, 1) == "blue", 1, 0)
#> [1] 1
ifelse(sample(beads, 1) == "blue", 1, 0)
#> [1] 1
```

Sometimes it's 1 some times it's 0.

In data science we often deal with data that is affected by chance in some way: the data comes from a random sample, the data is affected by measurement error or the data measures some outcome that is random in nature. Being able to quantify the uncertainty introduced by randomness is one of the most important jobs of a data scientists. Statistical inference offers a framework for doing this as well as several practical tools. The first step is to learn how to mathematically describe random variables. We start with games of chance.

1.7 Sampling Models

Many data generation procedures, those that produce the data we study, can be modeled quite well as draws from a urn. For example, we can model the process of polling likely voters as drawing 0s (Republicans) and 1s (Democrats) from an urn containing the 0 and 1 code for all likely voters. In epidemiological studies, we often assume that the subjects in our study are a random sample from the population of interest. The data related to a specific outcome can be modeled as random sample from urn containing that outcome for the entire population of interest. Similarly, in experimental research, we often assume that the individual organisms we are studying, for example worms, flies, or mice, are a random sample from a larger population. Randomized experiments can also be modeled by draws from an urn given the way individuals are assigned into groups: when getting assigned you draw your group at random. Sampling models are therefore ubiquitous in data science. Casino games offer a plethora of examples of real world situations in which sampling models are used to answer specific questions. We will therefore start with such examples.

Suppose a very small casino hires you to consult on weather they should set up roulette wheels. To keep the example simple, we will assume that 1,000 people will play and that the only game you can play is to bet on red or black. The casino wants you to predict how much money they will make, or lose. They want a range of values and, in particular, they want to know what is the chance of losing money. If this probability is too high they will pass on the installing roulette wheels since they can't take the risk given that they need to pay their employees and keep the lights on.

We are going to define a random variable `S` that will represent the casino's total winnings. Let's start by constructing the urn. A roulette wheel has 18 red pockets, 18 black pockets and two green ones. So playing a color in one game of roulette is equivalent to drawing from this urn:

```
color <- rep(c("Black","Red","Green"), c(18,18,2))
```

The 1,000 outcomes from 1,000 people playing are independent draws from this urn. If red comes up, the gambler wins and the casino loses a dollar so we draw a -$1. Otherwise the casino wins a dollar and we draw a $1. We code these draws like this:

To construct our random variable `S` we can use this code:

```
n <- 10000
X <- sample(ifelse( color=="Red", -1, 1),  n, replace = TRUE)
X[1:10]
#>  [1]  1 -1 -1 -1 -1  1  1  1 -1  1
```

Because we know the proportions of 1s and -1s, we can generate the draws with one line of code, without defining `color`:

```
X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
```

We call this a **sampling model** since we are modelling the random behavior of roulette with the sampling of draws from an urn.

The total winnings `S` is simply the sum of these 1,000 independent draws:

```
X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
S <- sum(X)
S
#> [1] 664
```

1.7.1 The probability distribution of a random variable

If you run the code above you see that S changes every time. This is, of course, because `S` is a **random variable**. The probability distribution of a random variables tells us the probability of the observed value falling in any given interval. So, for example, if we want to know the probability that we lose money, we are asking the probability that `S` in the interval $S < 0$.

Note that if we can define a cumulative distribution function $F(a) = \Pr(S \leq a)$ then we will be able to answer any question related to the probability of events defined by our random variable S, including the event $S < 0$. We call this F the random variable's *distribution function*.

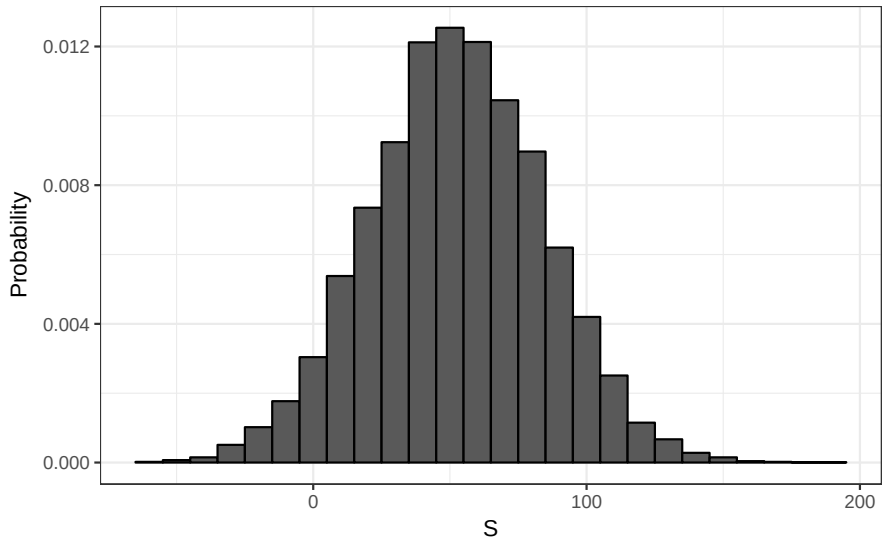
We can estimate the distribution function for the random variable S by using a Monte Carlo simulation to generate many realizations of the random variable. With this code we run the experiment of having 1,000 people play roulette, over and over, specifically `B`=1,000,000 times

```
n <- 1000
B <- 10000
S <- replicate(B, {
  X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
  sum(X)
})
```

So now we can ask: "in our simulations, how often did we get sums larger than `a`?".

```
mean(S <= a)
```

This will be a very good approximation of $F(a)$. In fact we can visualize the distribution by creating a histogram showing the probability $F(b) - F(a)$ for several intervals $(a, b]$:



Now we can easily answer the casino's question, "how likely is it that we lose money?"

```
mean(S<0)
#> [1] 0.046
```

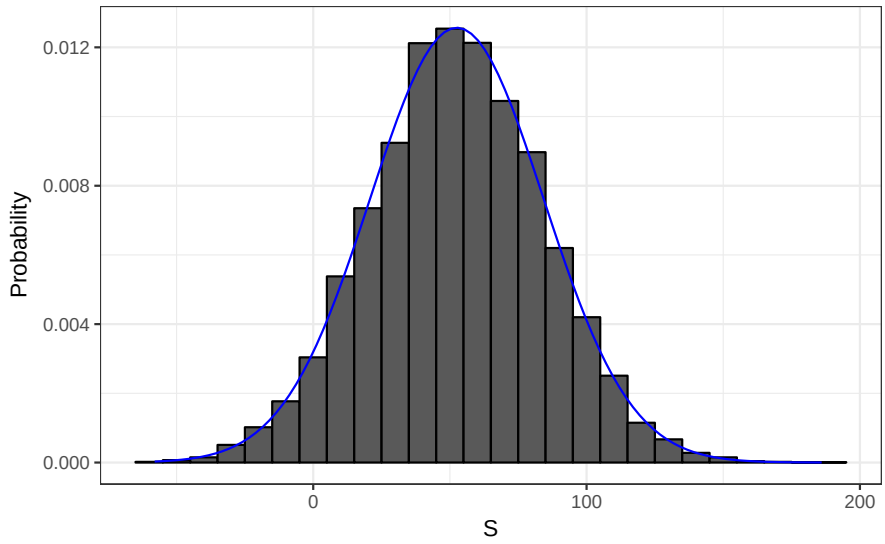
which we can see is quite low.

From the histogram above we see that the distribution appears to be approximately normal. A qq-plot will confirm that the normal approximation is close to perfect. If in fact the distribution is normal then all we need to define the distribution is the average and the standard deviation. Because we have the original values from which the distribution is created, we can easily compute these:

```
mean(S)
#> [1] 52.5
sd(S)
#> [1] 31.7
```

If we add a normal density, with this average and standard deviation, to the histogram above we see that it matches very well:

```
s <- seq(min(S), max(S), length = 100)
normal_density <- data.frame(s = s, f=dnorm(s, mean(S), sd(S)))
data.frame(S=S) %>% ggplot(aes(S, ..density..)) +
  geom_histogram(color = "black", binwidth = 10)  +
  ylab("Probability") +
  geom_line(data = normal_density, mapping=aes(s,f), color="blue")
```



This average and this standard deviation have special names: they are referred to as the *expected value* and *standard error* of the random variable S. We will say more about these in the next section.

It turns out that statistical theory provides a way to derive the distribution of random variable defined as independent random draws from an urn. Specifically, in our example above we can show that if $(S+n)/2$ follows a binomial distribution. We therefore do no need to run for Monte Carlo simulations to know the probability distribution of S. We did this for illustrative purposes. For the details of the binomial distribution you can consult any basis probability book or even Wikipedia. However, we will discuss an incredibly useful approximation provided by mathematical theory that applies generally to sums and averages of draws from any urn: the Central Limit Theorem (CLT).

1.7.2 Distribution versus probability distributions

Before we continue, let's make an important distinction and connection between the distribution of a list of numbers and a probability distribution. In the visualization chapter we described how any list of numbers $x_1, \dots, x_n$ has a distribution. The definition is quite straight forward. We define $F(a)$ as the function that answers the question "what proportion of the list is less than or equal to a ?". Because they are useful summaries when the distributions is approximately normal, we define the average and standard deviation. These are defined with a straight forward operation of the list:

```
avg <- sum(x)/length(x)
s <- sqrt(sum((x - avg)^2) / length(x))
```

A random variable X has a distribution function. To define this we do need a list of numbers. It is a theoretical concept. In this case to define the distribution as the $F(a)$ that answers the question "what is the probability that X is less than or equal to a ?" There is not list of numbers. However, if X is defined by drawing from an urn with numbers in it, then there is a list: the list of of numbers inside the urn. In this case, the distribution of that list is the probability distribution of X and the average and standard deviation of that list are the expected value and standard error of the random variable. Another way to think about it, that does not involve an urn, is to run a Monte Carlo simulation an generate a very large list of outcomes of X. These outcomes are a list of numbers. The distribution of this list will be a very good approximation of the probability distribution of X. The longer the list the better the approximation. The average and standard deviation of this list will approximate the expected value and standard error of the random variable.

1.7.3 Notation for Random Variables

Note that in statistical textbooks capital letters are used to denote random variables and we follow this convention here. Lower case letters are used for observed values. You will see some notation that includes both. For example, you will see events defined as $X \leq x$. Here X is a random variable, making it a random event, and x is an arbitrary value and not random. So, for example, X might represents the number on a die roll and x will represent an actual value we see. So in this case, the probability of $X = x$ is 1/6 regardless of the value of x. Note that this notation is a bit strange because when we ask questions about probability since X is not an observed quantity. Instead it's a random quantity that we will see in the future. We can talk about what we expect it to be, what values are probable, but not what it is. But once we have data, we do see a realization of X. So data scientists talk of what could have been after we see what actually happened.

1.8 Central Limit Theorem

The Central Limit Theorem (CLT) tells us that when the number of draws, also called the *sample size*, is large the probability distribution of the sum of the independent draws is approximately normal. Because sampling models are used for so many data generation processes, the CLT is considered one of the most important mathematical insights in history.

Previously, we discussed that if we know that the distribution of a list of numbers is approximated by the normal distribution all we need to describe the list are the average and standard deviation. We also know that the same applies to a probability distributions. If a random variable has a probability distribution that is approximated with the normal distribution then all we need to describe the probability distribution are the average and standard deviation, referred to as the expected value and standard error.

1.9 The Expected Value and Standard Error

We have described sampling models for draws. We will now go over the mathematical theory that let's us approximate the probability distributions for the sum of draws. Once we do this we will be able to help the casino predict how much money they will make. The same approach we use for the sum of draws will be useful for describing the distribution of averages and proportion which we will need to understand how polls work.

The first important concept to learn is the *expected value*. In statistics books it is common to letter E like this $\mathrm{E}[X] = \mu$ to denote that the expected value of the random variable X is μ. The Greek letter μ to commonly used to represent expected values. A random variable will vary around its expected value in a way that if you take the average of many many draws, the average of the draws will approximate the expected value, getting closer and closer the more draws you take.

A useful formula is that the expected value of the random variable defined by one draw is the average of the numbers in the urn. For example in our urn used to model betting on red on roulette we have 20 one dollars, and 18 negative one dollars, so the expected value is:

$$\mathrm{E}[X] = (20 + -18)/38$$

which is about 5 cents. It is a bit counter-intuitive to say that X varies around 0.05 when the only values it takes is 1 and -1. An intuitive way to think about the expected value is that if we play the game over and over, the casino wins, on average, 5 cents per game. A Monte Carlo simulation confirms this:

```
B <- 10^6
X <- sample(c(-1,1), B, replace = TRUE, prob=c(9/19, 10/19))
mean(X)
#> [1] 0.0515
```

In general, if the urn has two possible outcomes, say a and b, with proportions p and $1-p$ respectively, the average is

$$ap + b(1-p).$$

To see this, notice that if there are n beads in the urn then we have np as, $n(1-p)$ bs and because the average is the sum, $n \times a \times p + n \times b \times (1-p)$, divided by the total n we get that the average is $ap + b(1-p)$.

Now the reason we define the expected value is because this mathematical definition turns out to be useful for approximating the probability distributions of sums which in turn is useful to described the distribution of averages and proportions. The first useful fact is that the expected value of the sum of the draws is:

$$\text{number of draws} \times \text{average of the numbers in the urn}$$

So if 1,000 people play roulette the casino expects to win, on average, about 1,000 × \$0.05 = \$50. But this is an expected value. How different can one observation be from the expected value? The casino really needs to know this. What is the range of possibilities? If negative numbers are too likely we may not install roulette wheels. Statistical theory once again answers this question. The *standard error* (SE) gives us an idea of the size of the variation around the expected value. In statistics books it is common to use SE[X] to denote the standard error of a random variable.

If our draws are independent, then the standard error of the sum is given by the equation:

$$\sqrt{\text{number of draws}} \times \text{standard deviation of the numbers in the urn}$$

Using the definition of standard deviation, we can derive, with a bit of math, that if a jar contains two values a and b with proportions p and $(1-p)$ respectively, the standard deviation is

$$|\, b - a \,|\, \sqrt{p(1-p)}.$$

So in our roulette example the standard deviation of the values inside the urn is: $|\, 1-(-1) \,|\, \sqrt{10/19 \times 9/19}$ or

```
2 * sqrt(90)/19
#> [1] 0.999
```

The standard error tells us the typical difference between a random variable and its expectation. So because one draw is, obviously, the sum of one draw we can use the formula above to calculate that the random variable defined by one draw has an expected value of 0.05 and a standard error of about 1. This makes sense since we either get 1 or -1 with 1 slightly favored over -1.

Using the formula above, the sum of 1,000 people playing has standard error of about \$32:

```
n <- 1000
sqrt(n) * 2 * sqrt(90)/19
#> [1] 31.6
```

So when 1,000 people bet on red the casino es expected to win \$50 with a standard error of \$32. So it seems like a safe bet. But we still really can't answer questions such as, how likely is it to lose money? Here the CLT will help.

1.10 Central Limit Theorem Approximation

The CLT tells us that the sum `S` is approximated by normal distribution. Using the formulas above we know that the expected value and standard error are:

```
n * (20-18)/38
#> [1] 52.6
sqrt(n) * 2 * sqrt(90)/19
#> [1] 31.6
```

Note that the theoretical values above match those obtained with the Monte Carlo simulation:

```
mean(S)
#> [1] 52.5
sd(S)
#> [1] 31.7
```

Using the CLT we can skip the Monte Carlo simulation and instead compute the probability of the casino losing money using this approximation:

```
mu <- n * (20-18)/38
se <-  sqrt(n) * 2 * sqrt(90)/19
pnorm(0, mu, se)
#> [1] 0.0478
```

which is also in very good agreement with our Monte Carlo result

```
mean(S < 0)
#> [1] 0.046
```

1.10.1 Averages and proportions

There are two useful mathematical results that we used above and often use when working with data. We list them below

1. The expected value of the sum of random variable is the expected value of their sum. We can write it like this:

$$\mathrm{E}[X_1 + X_2 + \cdots + X_n] = \mathrm{E}[X_1] + \mathrm{E}[X_2] + \cdots + \mathrm{E}[X_n]$$

   If the X are independent draws from the urn then they all have the same expect value, call it μ and thus

$$\mathrm{E}[X_1 + X_2 + \cdots + X_n] = n\mu$$

   which is another way of writing the result we show above for the sum of draws.

2. The expected value of random variable times a non-random constant is the expected value times the non-random constant. This is easier to explain with symbols:

$$\mathrm{E}[aX] = a \times \mathrm{E}[X]$$

   To see why this is intuitive, think of change of units: if we change the units of a random variable, say from dollars to cents, the expectation should change in the same way. A consequence of the above two facts is that the expected value of the average of independent draws from the same urn is the expected of the urn, call it μ again:

$$\mathrm{E}[(X_1 + X_2 + \cdots + X_n)/n] = \mathrm{E}[X_1 + X_2 + \cdots + X_n]/n = n\mu/n = \mu$$

3. The square of the standard error of the sum of **independent** random variables is the sum of the square of the standard error of each random error. This one is easier to understand in math form:

$$\mathrm{SE}[X_1 + X_2 + \cdots + X_n] = \sqrt{\mathrm{SE}[X_1]^2 + \mathrm{SE}[X_2]^2 + \cdots + \mathrm{E}[X_n]^2}$$

Note that the square of the standard error is referred to as the *variance* in statistical textbooks.

4. The standard error of random variable times a non-random constant is the standard error times the non-random constant. As with the expectation

$$\text{SE}[aX] = a \times \text{SE}[X]$$

   To see why this is intuitive, again think of units.

   A consequence of 3 and 4 is that the standard error of the the average of independent draws from the same urn is the standard deviation of the urn, call it σ:

5. If X is a normally distributed random variable , then if a and b are non-random constants, $aX + b$ is also a normally distributed random variable. Note that what we are doing is changing the units of the random variable by multiplying by a then shifting the center by b.

1.10.2 Law of large numbers

An important implication of the final result is that the standard error of the average becomes smaller and smaller as n grows larger. When n is very large then the standard error is practically 0 and the average of the draws converges to the average of the urn. This is known in statistical textbooks as the law of large numbers or the law of averages.

1.10.3 Misinterpreting law of averages

Note that the law of average is sometimes misinterpreted. For example if you toss a coin 5 times and see a head each time, you might hear someone argue that the next toss is probably a tail because of the law of averages: on average we should see 50% heads and 50% tails. A similar argument would be to say that red "is due" on roulette after seeing black come up five times in a row. These events are independent so the chance of a coin landing heads is 50% regardless of the previous 5. Similarly for the roulette outcome. The law of averages applies only when the number of draws is very large not is small samples. After a million tosses you will definitely see about 50% heads regardless of what the first five were. Another funny misuse of the law of average is in sports where you hear TV announcers predict a player is about to succeed because they have failed a few times in a row.

1.10.4 How large is large in CLT?

The CLT works when the number of draws is large. But large is a relative term. In many circumstances as few as 30 draws is enough to make the CLT useful. In specific instances as few as 10 is enough. However, these should not be considered general rules. Note, for example, that when the probability of success is very small, we need larger sample sizes. Consider for example the lottery. In the lottery the chances of winning are less than 1 in a million. Thousands of people play so the number of draws is very large. Yet the number of winners, the sum of the draws, range between 0 and 4. This sum is certainly not well approximated by a normal distribution so the CLT does not apply, even with the very large sample size. This is generally true when the probability of a success is very low. In these cases the Poisson distribution is more appropriate. We do not cover the theory here but you can learn about the Poisson distribution in any probability textbooks and even Wikipedia.

1.11 SD versus estimate of SD

The standard deviation of a list `x` (we use heights as an example) is defined as the square root of the average of the squared differences:

```
library(dslabs)
x <- heights$height
m <- mean(x)
s <- sqrt(mean((x-m)^2))
```

Using mathematical notation we write

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

However, note that the `sd` function returns a slightly different result:

```
identical(s,sd(x))
#> [1] FALSE
s-sd(x)
#> [1] -0.00204
```

This is because the `sd` function R does not return the `sd` of the list but rather uses a formula that estimates standard deviations of a population from a random sample $X_1, \ldots, X_N$ which, for reasons not discussed here, divide by the $N - 1$.

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

You can see that this is the case by typing

```
N <- length(x)
s-sd(x)*sqrt((N-1)/N)
#> [1] 4.44e-16
```

For all the theory discussed here you need to compute the actual standard deviation as defined:`sqrt(mean((x-m)^2))`. So be careful when using the `sd` function in R. However, note that throughout the book we sometimes use the `sd` function when we really want the actual SD. This is because when the list size is big these two are practically equivalent since $\sqrt{(N-1)/N}$ is close to 1.

1.12 The Big Short

1.12.1 Interest rates explained with chance model

In a way, these sampling models are also used by banks to decide interest rates. Suppose you run a small bank that has a history of identifying potential homeowners that can be trusted to make payments. In fact, historically, in a given year, only 2% of your customers default, meaning that they don't pay back the money you lent them. However, note that if you simply loan money to everybody without interest you will end up losing money due to this 2%. Although you know 2% of your clients will probably default you don't know which ones. However, by charging everybody just a bit extra you can make up the loses incurred due to that 2% and also pay the employees that work to make these loans happen. You can also make a profit, but if you set the interest rates too high, your clients will go to another bank. We use all these facts and some probability theory to decide what interest rate we should charge.

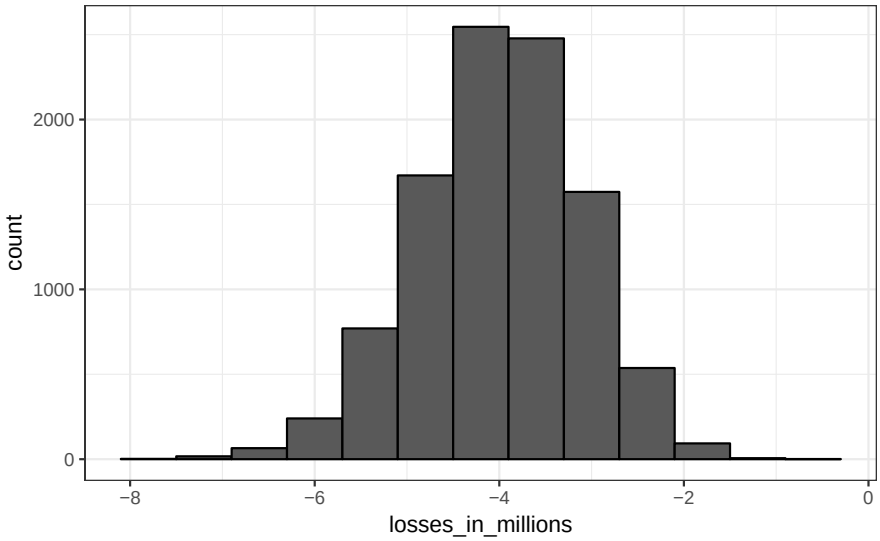
Suppose your bank will give out 1,000 loans for $180,000 this year. Also suppose your bank loses, after adding up all costs, $200,000 per foreclosure. For simplicity we assume this includes all operational costs. A sampling model for this scenario can be coded like this:

```
n <- 1000
loss_per_foreclosure <- -200000
p <- 0.02
defaults <- sample( c(0,1), n, prob=c(1-p, p), replace = TRUE)
sum(defaults * loss_per_foreclosure)
#> [1] -3e+06
```

Note that the total loss defined by the final sum is a random variable. Everytime you run the above code you get a different answer. We can easily construct a Monte Carlo simulation to get an idea of the distribution of this random variable.

```
B <- 10000
losses <- replicate(B, {
    defaults <- sample( c(0,1), n, prob=c(1-p, p), replace = TRUE)
  sum(defaults * loss_per_foreclosure)
})
```

Here is the distribution of this random variable:



We don't really need a Monte Carlo simulation though. Using what we have learned, the CLT tells us that because our losses are a sum of independent draws, its distribution is approximately nomral with expected value and standard errors given by:

```
n*(p*loss_per_foreclosure + (1-p)*0)
#> [1] -4e+06
sqrt(n)*abs(loss_per_foreclosure)*sqrt(p*(1-p))
#> [1] 885438
```

We can now set an interest rate to guarantee that, on average, we break even. Basically, we need to add a quantiy x to each loan, which in this case are represent by draws, so that the expected value is 0. If we define l to be the loss per forclusure, we need

$$lp + x(1-p) = 0$$

which implies x is

```
- loss_per_foreclosure*p/(1-p)
#> [1] 4082
```

or an interest rate of 0.023.

We still have a problem though. Although this interest rate guarantees that on average we break even there is 50% chance that we lose money. If our bank loses money we have to close it down. So we need to pick an interest rate that makes it unlikely for this to happen. At the same time, if the interest rate is too high our clients will go to another bank so we must be willing to take some risks. So let's say that we want our chances of losing money to be 1 in 100, what does the x quantity need to be now? This one is a bit harder. We want the sum S to have

$$\Pr(S < 0) = 0.01$$

We know that S is approximately normal. The expected value of S is$\{lp + x(1-p)\}n$ wiht n the number of draws, which in this case represents loans. The standard error is $|x-l|\sqrt{np(1-p)}$. Because x is positive and l negative $|x-l| = x-l$. Note that these are just the formulas from above but using more compact symbols.

Now we are going to use a mathematical "trick" that is very common in statistics. We are going add and substract the same quantities to both sides of the event $S < 0$ so that the probability does not change and we end up with a standard normal random variable on the left which will then permit us to write down an equation with only x as an unkown. Here it goes.

If

$$\Pr(S < 0) = 0.01$$

then

$$\Pr\left(\frac{S - \mathrm{E}[S]}{\mathrm{SE}[S]} < \frac{-\mathrm{E}[S]}{\mathrm{SE}[S]}\right)$$

And remember $\mathrm{E}[S]$ and $\mathrm{SE}[S]$ are the expected value and stnard error of S respectively. All we did abobe was add and divide by the same quantiy on both sides. We did this because now the trem on the left is a standard normal random variable, whcih we will rename Z. Now we fill in the blanks with the actual formula for expected value and standard error:

$$\Pr\left(Z < \frac{-\{lp + x(1-p)\}n}{(x-l)\sqrt{np(1-p)}}\right) = 0.01$$

It looks complicated but remember that l, p and n are all known amounts so eventually we will turn them into numbers.

Now because the term on the left side is a normal ranndom with expected value 0 and standard error 1, it means that the quantity on the left must be equal to

```
qnorm(0.01)
#> [1] -2.33
```

for the equation to hold true. Remember that z =`qnorm(0.01)` gives us the value of z for which

$$\Pr(Z \leq z) = 0.01$$

So this means that right side of the complicated equation must be z=`qnorm(0.01)`

$$\frac{-\{lp + x(1-p)\}N}{(x-l)\sqrt{Np(1-p)}} = z$$

The trick works because we end up with an expression containing x that we know has to be equal to a known quanty z. Solving for x is now simply algebra, we solve for x:

$$x = -l\frac{np - z\sqrt{np(1-p)}}{N(1-p) + z\sqrt{np(1-p)}}$$

which is

```
l <- loss_per_foreclosure
z <- qnorm(0.01)
x <- -l*( n*p - z*sqrt(n*p*(1-p)))/ ( n*(1-p) + z*sqrt(n*p*(1-p)))
x
#> [1] 6249
```

Our interest rate now goes up to 0.035. This is still a very competitive interest rate. Note also that by chosing this interest rate we now have an expected profit per loan of

```
loss_per_foreclosure*p + x*(1-p)
#> [1] 2124
```

which is a total expected profit of about

```
n*(loss_per_foreclosure*p + x*(1-p))
#> [1] 2124198
```

dollars!

We can run a Monte Carlo simulation to double check our theoretical approximations:

```
B <- 10000
profit <- replicate(B, {
    draws <- sample( c(x, loss_per_foreclosure), n,
                        prob=c(1-p, p), replace = TRUE)
    sum(draws)
})
mean(profit)
#> [1] 2125971
mean(profit<0)
#> [1] 0.0113
```

1.12.2 The Big Short

One of your employees points out that since the bank is making 2124 per loan that you should give out more loans! Why just `n`? You explain that finding those `n` clients was hard. You need a group that is predictable and that keeps the chances of defaults low. He then points out that even if the probability of default is higher, as long as our expected value is positive, you can minimize your chances of losses by increasing n and relying on the law of large numbers.

He claims that even if the default rate is twice as high, say 4%, if we set the rate just a bit higher than

```
p <- 0.04
r <- (- loss_per_foreclosure*p/(1-p)) / 180000
r
#> [1] 0.0463
```

Say at 5%, we are guaranteed a positive expected value of

```
r <- 0.05
x <- r*180000
loss_per_foreclosure*p + x * (1-p)
#> [1] 640
```

and can minimize our chances of losing money by simply increasing n since

$$\Pr(S < 0) = \Pr\left(Z < -\frac{\mathrm{E}[S]}{\mathrm{SE}[S]}\right)$$

with Z a standard normal random variable as above. If we define μ and σ to be the expected value and standard deviation of the urn, that is of a single loan we have, using the formulas above, $\mathrm{E}[S] = n\mu$ and $\mathrm{E}[S] = \sqrt{n}\sigma$ so if we define z=`qnorm(0.01)` we have

$$-\frac{n\mu}{\sqrt{n}\sigma} = -\frac{\sqrt{n}\mu}{\sigma} = z$$

which implies that if let

$$n \geq z^2\sigma^2/\mu^2$$

we are guaranteed to have a probability of less than 0.01. The implication is that, as long as μ is positive, we can find an n that minimizes the probability of a loss. This is form of the law of large numbers: when n is large our average earning per loan converges to the expected earning μ.

With x fixed, now we can ask what n do we need for the probability to be 0.01. In our example, as if we give out

```
z <- qnorm(0.01)
n <- ceiling( (z^2*(x-l)^2*p*(1-p))/(l*p + x*(1-p))^2 )
n
#> [1] 22163
```

loans the probability of losing is about 0.01 and we are expected to earn a total of

```
n*(loss_per_foreclosure*p + x * (1-p))
#> [1] 14184320
```

dollars! We can confirm this with a Monta Carlo simulation

```
p <- 0.04
x <- 0.05*180000
profit <- replicate(B, {
    draws <- sample( c(x, loss_per_foreclosure), n,
                     prob=c(1-p, p), replace = TRUE)
    sum(draws)
})
mean(profit<0)
#> [1] 0.0098
```

This seems like a no brainer. Your colleague decides to leave your bank and start his own high risk morgage company. A few months later your colleages bank had gone bankrupt. A book is written and eventually a movie made relating the mistake your friend, and many others, made. What happened?

Your friend's scheam was mainly based on this mathematical forumla

$$\mathrm{SE}[(X_1 + X_2 + \cdots + X_n)/n] = \sigma/\sqrt{n}$$

By making n large, we minimize the standard error of our per-loan profit. However, for this rule to hold te Xs must be independent draws. The fact that one person defaults must be indepent of other defaulting. Note for example that in the extreme case of averaging the **same** event over and over, an event is certainly not independent from itself, we get a standard error that is $\sqrt{n}$ times bigger:

$$\mathrm{SE}[(X_1 + X_1 + \cdots + X_1)/n] = \mathrm{SE}[nX_1/n] = \sigma > \sigma/\sqrt{n}$$

To construct a more realistic simulation that the original one your friend ran, let's assume there is a global event that affects everybody with high risk mortganges and changes their probabilty. We will assume that with 50-50 chance all the probabilities go up or down slightly to somewhere between 0.03 and 0.05. But it happens to everybody at once, not just one person. This draws are no longer independent.

```
p <- 0.04
x <- 0.05*180000
profit <- replicate(B, {
    new_p <- 0.04 + sample(seq(-0.01, 0.01, length = 100), 1)
    draws <- sample( c(x, loss_per_foreclosure), n,
                     prob=c(1-new_p, new_p), replace = TRUE)
    sum(draws)
})
```

Note that our expected profit is still large

```
mean(profit)
#> [1] 1.4e+07
```

However, the probability of the bank having negative earning shoots up to

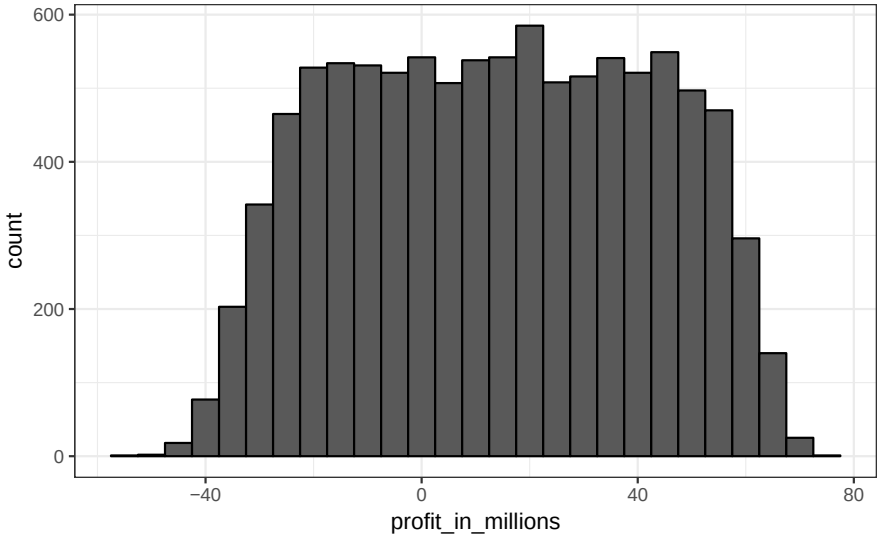
```
mean(profit<0)
#> [1] 0.349
```

Even scarier is that the probabiliyt of losing more than 10 million dollars is

```
mean(profit < -10000000)
#> [1] 0.244
```

To understand how this happens look at the distribution

```
data.frame(profit_in_millions=profit/10^6) %>% ggplot(aes(profit_in_millions)) + geom_histogram(color="
```



The theory completley breaks down and the random variable has much more variability than expected. The financial meltdown of 2007 was due, among other things, to financial "experts" assuming independence when there was none.

Contents

1 Inference and Modeling

The day before the 2008 presidential election Nate Silver's Fivethirtyeight stated that "Barack Obama appears poised for a decisive electoral victroy". They went further and predicted that Obama would win the election 349 electoral votes to 189 and the popular vote by a margin of 6.1%. Fivethirtyeight also attached a probabilistic statement to their prediction claiming that Obama had a 91% chance of winning the election. The predictions were quite accurate since in the final tally actual results were Obama won the electoral college 365 to 173 and the popular vote by 7.2% difference. Their performance in the 2008 election brought Fivethirtyeight to the attention of political pundints and TV personalities. The week before the 2012, Fivethirtyeight Nate Silver was giving a Obama a 90% chance of winning despite many of the experts thinking they were close, political commentator Joe Scarborough said during his show

> Anybody that thinks that this race is anything but a tossup right now is such an ideologue … they're jokes.".

To which Nate Silver responded via Twitter:

> If you think it's a toss-up, let's bet. If Obama wins, you donate $1,000 to the American Red Cross. If Romney wins, I do. Deal?

How was Mr. Silver so confident. We will demonstrate how *poll aggregators*, such as Fivethirtyeight, collected and combined data reported by different experts to produce imrpoved predictions. The two main statistical tools used by the aggregators are the topic of this chapter: inference and modeling. To begin to understand how election forecasting works we need to understand the basic data point they use: poll results.

1.1 Polls

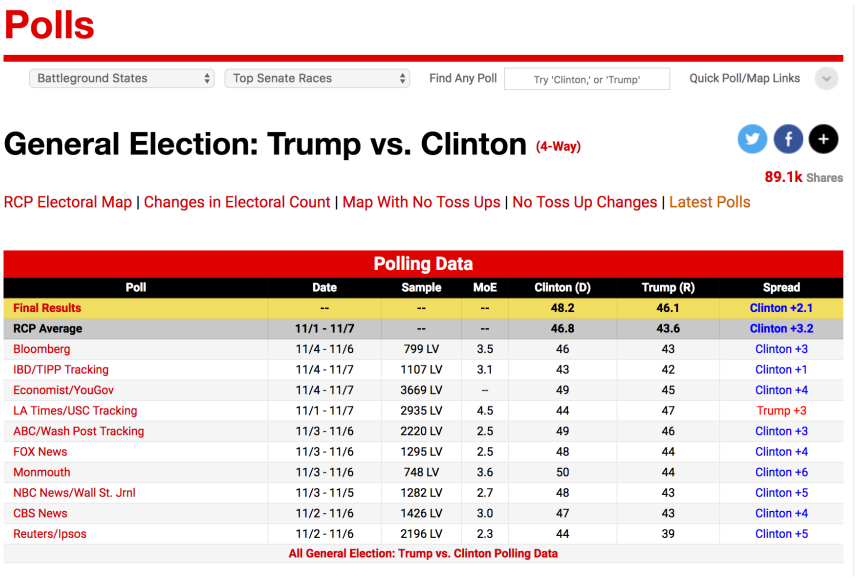
Opinion polling have been conducted since the 19-th century. The general goal of these is to describe the opinions held by a specific population on a given set of topics. In recent times, these polls have been pervasive during presidential elections. Polls are useful when asking everybody in the population is logistically impossible. The general strategy is to ask a smaller group, chosen at random, and then infer the opinions of the entire population from the opinions of the smaller group. Statistical theory is used to justify the process. This theory is referred to as *inference* and is the main topic of this section.

Perhaps the best know opinion polls are those conducted to determining which candidate is preferred by voters in a given election. Political strategists make extensive use of polls to determine, for example, how to invest resources. For example, they may want to know which geographical locations to focus get out the vote efforts.

Elections are a particularly interesting case of opinion polls because the actual opinion of the entire population is revealed on election day. Of course, it costs millions of dollars to run an actual election which makes polling a cost effective strategy for those that want to forecast the results.

Although typically the results of these polls are kept private, similar polls are conducted by news organizations because results tend to be of interest to the general public and are often made public. We will eventually be looking at such data.

Real Clear Politics is an example of a news aggregation that organizers and publishes poll results. For example, here are examples of polls reporting estimate of the popular vote for the 2016 presidential election:



Polls

Battleground States | Top Senate Races | Find Any Poll | Try 'Clinton,' or 'Trump' | Quick Poll/Map Links

General Election: Trump vs. Clinton (4-Way)

89.1k Shares

RCP Electoral Map | Changes in Electoral Count | Map With No Toss Ups | No Toss Up Changes | Latest Polls

Polling Data

Poll	Date	Sample	MoE	Clinton (D)	Trump (R)	Spread
Final Results	--	--	--	48.2	46.1	Clinton +2.1
RCP Average	11/1 - 11/7	--	--	46.8	43.6	Clinton +3.2
Bloomberg	11/4 - 11/6	799 LV	3.5	46	43	Clinton +3
IBD/TIPP Tracking	11/4 - 11/7	1107 LV	3.1	43	42	Clinton +1
Economist/YouGov	11/4 - 11/7	3669 LV	--	49	45	Clinton +4
LA Times/USC Tracking	11/1 - 11/7	2935 LV	4.5	44	47	Trump +3
ABC/Wash Post Tracking	11/3 - 11/6	2220 LV	2.5	49	46	Clinton +3
FOX News	11/3 - 11/6	1295 LV	2.5	48	44	Clinton +4
Monmouth	11/3 - 11/6	748 LV	3.6	50	44	Clinton +6
NBC News/Wall St. Jrnl	11/3 - 11/5	1282 LV	2.7	48	43	Clinton +5
CBS News	11/2 - 11/6	1426 LV	3.0	47	43	Clinton +4
Reuters/Ipsos	11/2 - 11/6	2196 LV	2.3	44	39	Clinton +5

All General Election: Trump vs. Clinton Polling Data

Although, in the United States, the popular vote does not determine the result of the election, we will use it here as an illustrative and simple example of how well polls work. Forecasting the election is a more complex process since it involves combining results from 50 states and DC.

Let's make some observation about the table above. First note that different polls, all taken days before the election, report a different *spread*: the estimated difference between support for the two candidates. Note also that the reported spreads hover around what ended up being the actual result: Clinton won the popular vote by 2.1%.We also see a column titled **MoE** which stands for *margin of error*.

In this section we show how the probability concepts we learned in the previous chapter can be applied to develop the statistical approaches that make polls an effective tool. We will learn the statistical concepts necessary to define *estimates* and *margins of errors*, and show how we can use these to forecast final results relatively well and also provide an estimate of the precision of our forecast. Once we learn this we will be able to understand two concepts that are ubiquitous in data science: *confidence intervals*, and *p-values*. Finally, to understand probabilistic statetmets about the probability of a candidate winning we will have to learn about Bayesian modelling. In the final sections we put it all together to recreate the simplified version of the Fivethirtyeight model and apply it to 2016 election.

We start by connecting probability theory to the task of using polls to learn about a population.

1.2 The Sampling Model for Polls

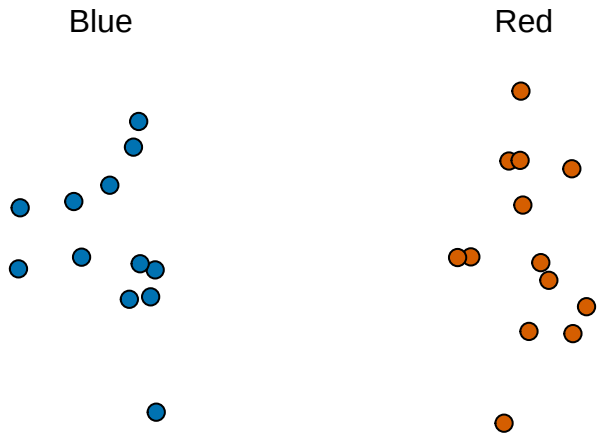
To help us understand the connection between polls and what we have learned let's construct a similar situation to the one pollsters face. We will use an urn instead of voters and of competing with other pollsters for media attention we will have a competition with $25 dollar prize. The challenge is to guess the spread between the proportion of blue and red beads in this urn:

Before making a prediction you can take a sample (with replacement) from the urn. To mimic the fact that running polls is expensive, it cost you $0.10 per bead you sample. So if your sample size of 250, and you win, you will break even as you will pay me $25 to collect your $25 prize. Your entry into the competition can be an interval. If the interval you submit contains the true proportion you get half what you paid and pass to the second phase of the competition. In the second phase the entry with the smallest interval is selected as the winner.

The `dslabs` package includes a function that shows a random draw from this urn:

```
library(tidyverse)
FALSE Loading tidyverse: ggplot2
FALSE Loading tidyverse: tibble
FALSE Loading tidyverse: tidyr
FALSE Loading tidyverse: readr
FALSE Loading tidyverse: purrr
FALSE Loading tidyverse: dplyr
FALSE Conflicts with tidy packages ----------------------------------------------
FALSE filter(): dplyr, stats
```

```
FALSE lag():    dplyr, stats
library(dslabs)
ds_theme_set()
take_poll(25)
```



Think about how you would construct your interval based on the data shown above.

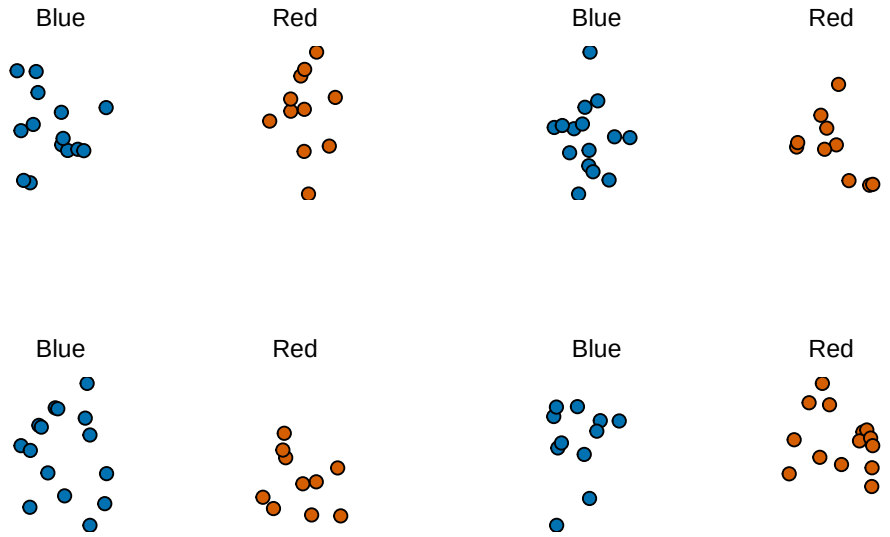
We have just described a simple sampling model for opinion polls. The beads inside the urn represent the individuals that will vote on election day. Those that will vote for the republican candidate are represented with red beads and the democrats with the blue beads. For simplicity assume there are no other colors, that there are just two parties.

1.3 Populations, samples, parameters and estimates

We want to predict the proportion of blue beads in the run, let's call this quantity p, which in turn tells us the proportion of red beads $1-p$, and the spread $p-(1-p)$ which simplifies to $2p-1$.

In statistical textbooks, the beads in the urn are called the *population*. The proportion of blue beads in the population p is called a *parameter*. The 25 beads we see in the plot above our called a *sample*. The task of statistical inference is to predict the parameter p using the observed data in the sample.

Can we do this with the 25 observations above? It is certainly informative. For example, given we see 13 red and 12 blue, it is unlikely that $p > .9$ or $p < .1$. But are we ready to predict with certainty that there are more red beads than blue? We want to construct an estimate of p using only the information we observe. An estimate can be thought of as a summary of the observed data that we think is informative about the parameter of interest. It seems intuitive to think that the proportion of blue beads in the sample, 0.48 must be at least related the actual proportion p. But do we simply predict p to be 0.48? First note that the sample proportion is a random variable. If we run the command `take_poll(25)` four times



we get a different answer each time since the sample proportion is a random variable.

Note that in the four random samples we show above, the sample proportions range from 0.44 to 0.60. By describing the distribution of this random variable we will be able to gain insights into how good this estimate is and how we can make it better.

1.3.1 The Sample Average

Taking an opinion poll is being modeled as taking a random sample from the urn. We are proposing the use of the proportion of blue beads in our sample as an *estimate* of the parameter p. Once we have this estimate we can can easily report an estimate for the spread $2p-1$, but for simplicity we will illustrate the concepts for estimating p. We will use our knowledge of probability to defend our use of the sample proportion and quantify how close we think it is from the population proportion p.

We start by defining the random variable $X=1$ if we pick a blue bead at random and 0 if it is red. This implies that the population is a list of 0s and 1s. If we sample N beads then the average of the draws $X_1,\dots,X_N$ is equivalent to the proportion of blue beads in our sample. This is because adding the Xs is equivalent to counting the blue beads and dividing it by the total N turns this into a proportion. We use the symbol $\bar{X}$ to represent this average. In general, in statistics text books, a bar on top of a symbol means the average.

The theory we just learned about the sum of draws becomes useful because if we know the distribution of the sum $N\bar{X}$, we know the distribution of the average $\bar{X}$ because N is a non-random constant.

For simplicity, let's assume that the draws are independent: after we see each sampled bead we return it to the urn. In this case what do we know about the distribution of the sum of draws? First we know that the expected value of the sum of draws is N times the average of the values in the urn. We know that the average of the 0s and 1s in the urn must be p, the proportion of blue beads.

Here we encounter an important difference with what we did in the Probability Chapter: we don't know what is in the urn. We know there are blue and red beads but we don't know how many of each. This is what we want to find out: we are trying to **estimate** p.

1.3.2 Parameters

Just like we use variables to define unknowns in systems of equations, in statistical inference we define *parameters* to define unknown parts of our models. In the urn model we are using to mimic a opinion poll, we do not know the proportion of blue beads in the urn. We define the parameters p to represent this quantity. Note that p is the average of the urn since if we take the average of the 1s (blue) and 0s (red) we

get the proportion of blue beads. Also note that our main goal is figuring out what is p. We are going to *estimate this parameter*.

Note that the ideas presented here on how we estimate parameters and provide insights into how good these estimates are, extrapolate to many data science tasks. For example, we may ask what is the difference in health improvement between patient receiving treatment and a control group. We may ask what are the health effects of smoking on a population? What are the differences in racial groups of fatal shootings by police? What is the rate of change in life expectancy in the US during the last 10 years? All these questions can be framed as a task of estimating a parameter from a sample.

1.3.3 Polling versus Forecasting

Before we continue let's make an important clarification related to the practical problem of forecasting the election. If a poll is conducted four months before the election it is estimating the p for that moment not for election day. But note that the p for election night might be different since people's opinions fluctuate through time. The polls provided the night before the election tend to be the most accurate since opinions don't change that much in a couple of days. However, forecasters try to build tools that model how opinions vary across time and try to predict the election night result taking into consideration the fact that opinions fluctuate. We will describe some approaches for doing this in our a later section.

1.3.4 Properties of our estimate: Expected value and standard error

To understand how good our estimate is, we will describe the statistical properties of the random variable defined above: the sample proportion $\bar{X}$. Note that $\bar{X}$ is the some of independent draws so the rules we covered in the probability chapter apply.

Using what we have learned, the expected value of the sum $N\bar{X}$ is $N\times$ the average of the urn, p. So dividing by the non-random constant N gives us that the expected value of the average $\bar{X}$ is p. We can write it using our mathematical notation:

$$\mathrm{E}(\bar{X}) = p$$

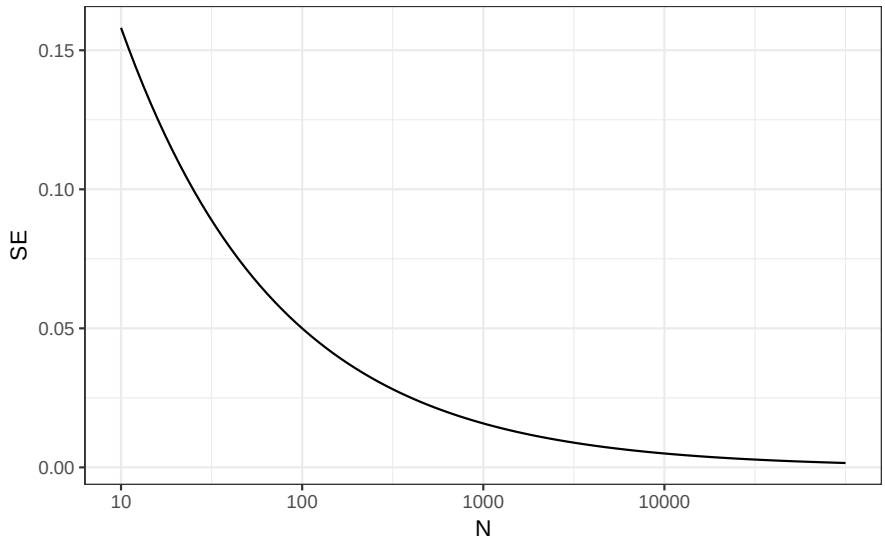
We can also use what we learned to figure out the standard error. We know it is the the standard error of the sum is $\sqrt{N}\times$ the standard deviation of the urn. Can we compute the standard error of the urn? We learned a formula that tells is that it is $(1-0)\sqrt{p(1-p)} = p(1-p)$. Because we are dividing the sum by N we arrive at the following formula for the standard error for the average:

$$\mathrm{SE}(\bar{X}) = \sqrt{p(1-p)/N}$$

This result reveals the power of polls. The expected value of the sample proportion $\bar{X}$ is the parameter of interest p and we can make the standard error as small as we want by increasing N. The law of large numbers tells us that with a large enough poll our estimate converges to p.

If we take a large enough poll to make our standard error about 0.01% we will be quite certain about who will win. But how large does the poll have to be for the standard error to be this small?

One problem is that we do not know p, so we can't compute the standard error. For illustrative purposes let's assume that $p = 0.51$ and make a plot of the standard error versus the sample size N:



From the plot we see that we would need a poll of over 10,000 people to get the standard error that low. We rarely see polls of this size due in part to costs. From the Real Clear Politics table we learn that the sample sizes in opinion polls range from 500-3,500 people. For a sample size of 1,000 and $p = 0.51$ the standard error is:

```
sqrt(p*(1-p))/sqrt(1000)
#> [1] 0.0158
```

or 1.5 percentage points. So even with large polls, for close elections, $\bar{X}$ can lead us astray if we don't realize it is a random variable. But we can actually say more about how close we get the p.

1.4 Central Limit Theorem in Practice

The CLT tells us that the distribution function for a sum of draws is approximately normal. We also learned that dividing a normally distributed random variable then dividing that random variable normally distributed variable. This implies that the distribution of $\bar{X}$ is approximately normal.

So in summary we have the $\bar{X}$ has an approximately normal distribution with expected value p and standard error $\sqrt{p(1-p)/N}$.

Now how does this help us? Suppose we want to know what is the probability that we are withing 1% from p. We are basically asking if

$$\Pr(|\bar{X} - p| \leq .01)$$

which is the same as:

$$\Pr(\bar{X} \leq p + .01) - \Pr(\bar{X} \leq p - .01)$$

Can we answer this question? Note that we can use the mathematical trick we learned in the previous chapter. Subtract the expected value and divide by the standard error to get a standard normal random variable, call it Z, on the left. Since p is the expected value and $\mathrm{SE}(\bar{X}) = \sqrt{p(1-p)/N}$ is the standard error we get:

$$\Pr\left(Z \leq \ .01/\mathrm{SE}(\bar{X})\right) - \Pr\left(Z \leq -.01/\mathrm{SE}(\bar{X})\right)$$

A problem is that we don't know p, so we don't know $\mathrm{SE}(\bar{X})$. But it turns out that the CLT still works if we use estimate the standard error by using $\bar{X}$ in place of p. We say that we *plug-in* the estimate. Our estimate of the standard error is therefore:

$$\hat{\mathrm{SE}}(\bar{X}) = \sqrt{\bar{X}(1-\bar{X})/N}$$

In statistics textbooks, we use a little hat to denote estimates. Note that the estimate can be constructed using the observed data and N.

Now we continue with our calculation but dividing by $\hat{\mathrm{SE}}(\bar{X}) = \sqrt{\bar{X}(1-\bar{X})/N}$ instead. In our first sample we had 12 blue and 13 red so $\bar{X} = 0.48$ and so our estimate of standard error is

```
X_hat <- 0.48
se <- sqrt(X_hat*(1-X_hat)/25)
se
#> [1] 0.0999
```

And now we can answer the question of the probability of being close to p. The answer is

```
pnorm(0.01/se) - pnorm(-0.01/se)
#> [1] 0.0797
```

So there is a small chance that we will be close. A poll of only $N = 25$ people is not really very useful. At least for a close elections.

Earlier we mentioned the *margin of error*. Now we can define it because it is simply two times the standard error which we can now estimate and in our case it is:

```
2*se
#> [1] 0.2
```

Why do we multiply by 2? Because if you ask what is the probability that we are withing two standard error from p we get:

$$\Pr\left(Z \leq 2\mathrm{SE}(\bar{X})/\mathrm{SE}(\bar{X})\right) - \Pr\left(Z \leq -2\mathrm{SE}(\bar{X})/\mathrm{SE}(\bar{X})\right)$$

which is

$$\Pr\left(Z \leq 2\right) - \Pr\left(Z \leq -2\right)$$

which we know is about 95%:

```
pnorm(2)-pnorm(-2)
#> [1] 0.954
```

So there is a 95% that $\bar{X}$ will be within $2 \times \hat{SE}(\bar{X})$, in our case 0, to p. Note that 95% is somewhat of an arbitrary choice and sometimes other percentages are used, but it is the most commonly used value to define *margin of error*.

In summary, the CLT tells us that our poll based on a sample size of 25 is not a very useful. We don't really learn much when the margin of error is this large. All we can really say is that the popular vote will not be won by a large margin. This is why pollsters tend to use larger sample sizes.

From the table above we see that typical sample sizes from 700 to 3500. To see how this gives us a much more practical result, note that if we had obtained a $\bar{X}$=0.48 with a sample size of 2,000 our standard error $\hat{\mathrm{SE}}(\bar{X})$ would have been 0.011. So our result is an estimate of 48% with a margin of error is 2%. In this case, the result is much more informative and would make us think that there are more red balls than blue. But keep in mind, we this is hypothetical. We did not take a poll of 2,000 since we don't want ruin the competition.

1.4.1 A Monte Carlo simulation

Suppose we want to use a Monte Carlo simulation to corroborate the tools we have built using probability theory. To create a the simulation we would write code like this:

```
B <- 10000
N <- 1000
Xhat <- replicate(B, {
  X <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
  mean(X)
})
```

The problem is, of course, we don't know `p`. We could construct an urn like the one pictured above and run an analog (without a computer) simulation. It would take a long time, but you could take 10,000 samples, count the beads and keep track of the proportions of blue. We can use the function `take_poll(n=1000)` instead of drawing from an actual urn, but it would still take time to count the beads and enter the results.

So, one thing we do to corroborate theoretical results is to pick one, or several values of `p`, and run the simulations. Let's set `p=0.45`. We can then simulate a poll

```
p <- 0.45
N <- 1000

X <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
Xhat <- mean(X)
```

In this particular sample our estimate is `Xhat`. We can use that code to a Monte Carlo simulation:

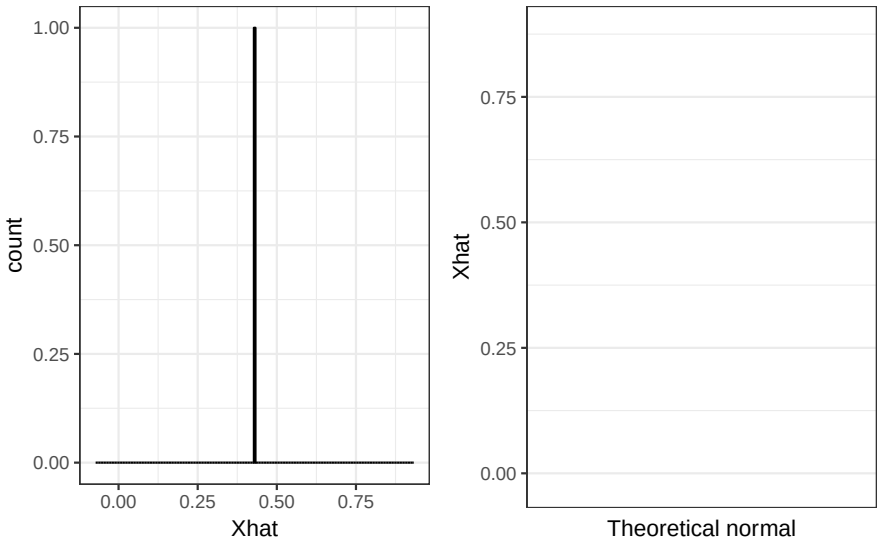
```
B <- 10000
Xhat <- replicate(B, {
  X <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
  mean(X)
})
```

To review, the theory tells us that $\bar{X}$ has is approximately normally distributed, has expected value $p=$0.45 and standard error $\sqrt{p(1-p)/N} = 0.016$. The simulation confirms this

```
mean(Xhat)
#> [1] 0.431
sd(Xhat)
#> [1] NA
```

A histogram and qq-plot confirm that the the normal approximation is accurate as well:

```
#>
#> Attaching package: 'gridExtra'
#> The following object is masked from 'package:dplyr':
#>
#>     combine
```



Again, note that in a real life we would never be able to run such an experiment because we don't know p. But we could run it for various values of p and N and see that the theory does indeed work well for most values. You can easily do this by re-running the code above after changing `p` and `N`.

1.4.2 The spread

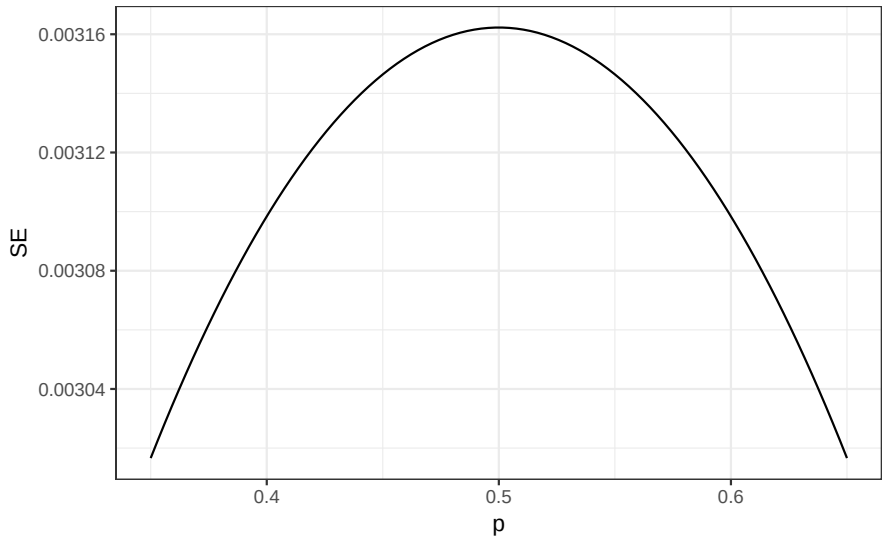
The competition is to predict the spread not the proportion p. However, because we are assuming there are only two parties, we know that the spread if $p - (1 - p) = 2p - 1$. So everything we have done can easily be adapted to an estimate of $2p - 1$. Once we have our estimate $\bar{X}$ and $\hat{\text{SE}}(\bar{X})$ we estimate the spread with $2\bar{X} - 1$ and, since we are multiplying by 2, the standard error is $2\hat{\text{SE}}(\bar{X})$. Note that subtracting 1 does not add any variability so it does not affect the standard error.

So for our 25 sample above, our estimate p is `.48` with margin of error `.20` so our estimate of the spread is `0.04` with margin of error `.40`. Again, not a very useful sample size. But the point is that once we have an estimate and standard error for p, we have it for the spread $2p - 1$.

1.4.3 Bias: Why not run a very large poll?

Note that for realistic value of p, say from 0.35 to 0.65, if we run a very large poll with 100,000 people, theory would tell us that we would predict the election perfectly since the largest possible margin of error is around 0.3%. Here are the calculations:

```
N <- 100000
p <- seq(0.35, 0.65, length = 100)
SE <- sapply(p, function(x) 2*sqrt(x*(1-x)/N))
data.frame(p=p, SE = SE) %>% ggplot(aes(p, SE)) +
  geom_line()
```

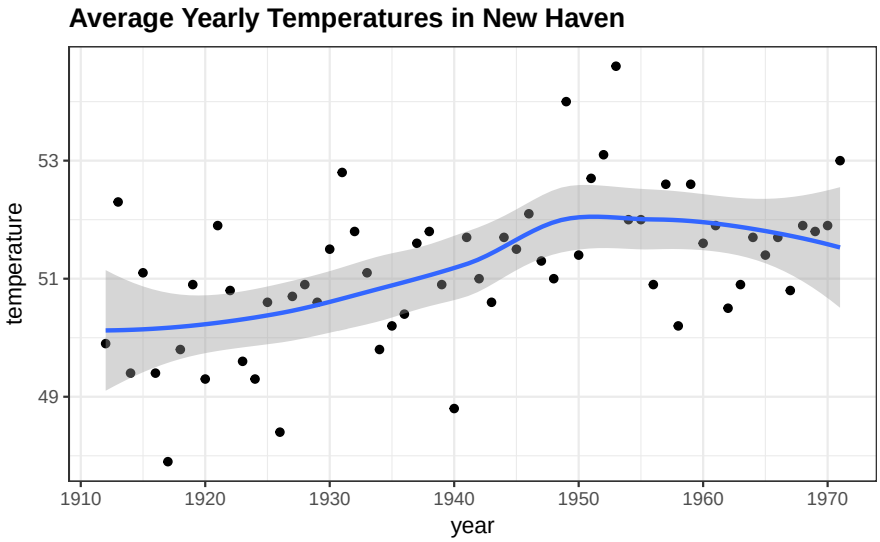


One reason is that running such a poll is very expensive. But perhaps a more important reason is that theory has it's limitations. Polling is much more complicated than picking beads from an urn. People might lie to you and other might not have phones. But perhaps the most different way an actual poll is from an urn model is that we actually don't know for sure who is in our population and who is not. How do we know who is going to vote? Are we reaching all possible voters? So even if our margin of error is very small it might not be exactly right that our expected value is p. We call this bias. Historically, we observe that polls are indeed biased although not by that much. The typical bias appears to be about 1-2%. This makes election forecasting a bit more interesting and we will talk about how to model this in a later chapter.

1.5 Confidence Intervals

Confidence intervals are a very useful concept that is widely used by data scientists. A version of these that very commonly seen come from the `ggplot` geometry `geom_smooth`. Here is an example using a temperature dataset available in R:

```
data("nhtemp")
data.frame(year = as.numeric(time(nhtemp)), temperature=as.numeric(nhtemp)) %>%
  ggplot(aes(year, temperature)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Average Yearly Temperatures in New Haven")
```



We will later learn how the curve is formed, but note the shaded area around the curve. The shaded area around the curve is created using the concept of confidence intervals.

In our competition we were asked to give an interval. If the interval you submit includes the p you get half the money you spent on your "poll" back and pass to the next stage of the competition. One way to pass to the second round is to report a very large interval. For example, the interval $[0, 1]$ is guaranteed to include p. However, with an interval this big, we have no chance of winning the competition. Similarly, if you are an election forecaster and predict the spread will be between -100% and 100% you will be ridiculed for stating the obvious. Even a smaller interval such as saying the spread will be between -10 and 10% will not be considered serious.

On the other-hand the smaller the interval we report, the smaller our chances of winning the prize. Similarly, a bold pollster that reports very small intervals and misses the mark most of the time will not be considered a good pollster. We want to be somewhere in between.

We can use the statistical theory we have learned to compute the probability of any given interval including p. Similarly,if we are asked to create an interval with, say, a 95% chance of including p, we can do that as well. These are called 95% confidence intervals.

Note, that when pollster report an estimate and a margin of error, they are, in a way, reporting a 95% confidence interval. Let's show how this works mathematically.

We want to know that probability that the interval $[\bar{X} - 2\hat{\text{SE}}(\bar{X}), \bar{X} - 2\hat{\text{SE}}(\bar{X})]$ contains the true proportion p. First, note that the start and end of this intervals are random variables: every time they we take a sample they change. To illustrate this run Monte Carlo simulation above twice. We use the same parameters as above

```
p <- 0.45
N <- 1000
```

And note that the interval here

```
X <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
X_hat <- mean(X)
SE_hat <- sqrt(X_hat*(1-X_hat)/N)
c(X_hat - 2*SE_hat, X_hat + 2*SE_hat)
#> [1] 0.418 0.480
```

is different from this one:

```
X <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
X_hat <- mean(X)
```

```
SE_hat <- sqrt(X_hat*(1-X_hat)/N)
c(X_hat - 2*SE_hat, X_hat + 2*SE_hat)
#> [1] 0.416 0.478
```

Keep sampling and creating intervals and you will see the random variation.

To determine the probability that the interval includes p we need to compute this:

$$\Pr\left(\bar{X} - 2\hat{\text{SE}}(\bar{X}) \leq p \leq \bar{X} + 2\hat{\text{SE}}(\bar{X})\right)$$

By subtracting and dividing the same quantities in all parts of the equation we get that the above is equivalent to:

$$\Pr\left(-2 \leq \frac{\bar{X} - p}{\hat{\text{SE}}(\bar{X})} \leq 2\right)$$

The term in the middle is an approximately normal random variable with expected value 0 and standard error 1, which we have been denoting with Z, so we have

$$\Pr\left(-2 \leq Z \leq 2\right)$$

which we can quickly compute using

```
pnorm(2) - pnorm(-2)
#> [1] 0.954
```

proving the we have a 95% probability.

Note that if we want to have a larger probability, say 99%, we need to multiply by whatever `z` satisfies the following:

$$\Pr\left(-z \leq Z \leq z\right) = 0.99$$

Note that by using

```
z <- qnorm(0.995)
z
#> [1] 2.58
```

will do it because by definition `pnorm(qnorm(0.995)` is 0.995 and by symmetry `pnorm(1-qnorm(0.995))` is 1 - 0.995, we have that

```
pnorm(z)-pnorm(-z)
#> [1] 0.99
```

is `0.995 - 0.005 = 0.99`. We can use this approach for any percentile q: we use $1-(1-q)/2$. Why this number? Because $1-(1-q)/2+(1-q)/2 = q$.

Note that to get exactly 0.95 confidence interval, we actually use a slightly smaller number than 2:

```
qnorm(0.975)
#> [1] 1.96
```

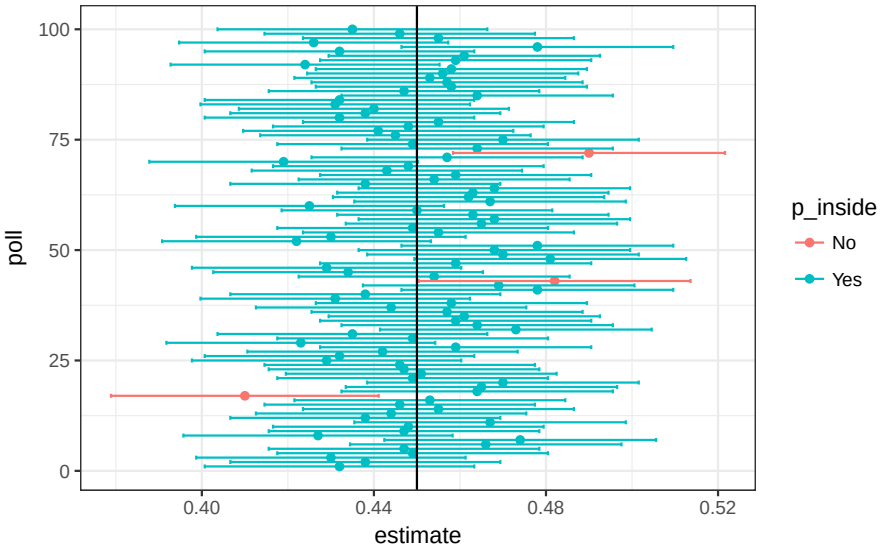
1.5.1 A Monte Carlo Simulation

We can run a Monte Carlo simulation to confirm that in fact that a 95% confidence interval includes p 95% of the time.

```
set.seed(1)
```

```
B <- 10000
inside <- replicate(B, {
  X <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
  X_hat <- mean(X)
  SE_hat <- sqrt(X_hat*(1-X_hat)/N)
  between(p, X_hat - 2*SE_hat, X_hat + 2*SE_hat)
})
mean(inside)
#> [1] 0.955
```

The following plot shows the first 100 confidence intervals. In this case we created the simulation so The black line



1.5.2 The Correct Language

When using the theory we described above it is important to remember that it is the intervals that are random not p. In the plot above we can see the random intervals moving around and p, represented with the vertical line, staying in the same place. The proportion of blue in the urn p is not. So the 95% relates to the probability that this random interval falls on top of p. Saying the p has a 95% of being between this and that is technically an incorrect statement. Again, because p is not random.

1.5.3 Power

Pollsters are not successful for providing correct confidence intervals but rather for predicting who will win. When we took a 25 bead sample size, the confidence interval for the spread:

```
N <- 25
X_hat <- 0.48
(2*X_hat - 1) + c(-2,2)*2*sqrt(X_hat*(1-X_hat)/sqrt(N))
#> [1] -0.934  0.854
```

includes 0. If this were a poll and we were forced to make a declaration, we would have to say it was a "toss-up".

A problem with our poll results is that, given the sample size and the value of p, we would have to sacrifice on the probability of an incorrect call to create an interval that does not include 0.

This does not mean that the election is close. It only means that we have a small sample size. In statistical textbooks this is called lack of *power*. In the context of polls, *power* is the probability of detecting spreads different from 0.

By increasing our sample size, we lower our standard error and therefore have a much better chance of detecting the direction of the spread.

1.6 p-values

p-values are ubiquitous in the scientific literature. They are related to confidence interval so we introduce the concept here.

Let's consider the blue and red beads. Suppose that rather than wanting an estimate of the spread or the proportion of blue, I am interested only in the question: are there more blue beads or red beads? I want to know if the spread $2p - 1 > 0$.

Suppose we take a random sample of $N = 100$ and we observe 52 blue beads which gives us $2\bar{X} - 1 = 0.04$. This seems to be pointing to their being more blue than red since 0.04 is larger than 0. However, as data scientists we need to be skeptical. We know there is chance involved in this process and we could get a 52 even when the actual spread is 0. We call this a *null hypothesis*. The null hypothesis is the skeptics hypothesis: the spread is $2p - 1 = 0$. We have observed a random variable $2 * \bar{X} - 1 = 0.52$ and the p-value is the answer to the question how likely is it to see a value this large, when the null hypothesis is true. So we write

$$\Pr(\mid \bar{X} - 0.5 \mid > 0.02)$$

assuming the $2p - 1 = 0$ or $p = 0.5$. Under the null hypothesis we know that

$$\sqrt{N}\frac{\bar{X} - 0.5}{\sqrt{0.5(1 - 0.5)}}$$

is standard normal. So we can compute the probability above, which is the p-value.

$$\Pr\left(\sqrt{N}\frac{\mid \bar{X} - 0.5 \mid}{\sqrt{0.5(1 - 0.5)}} > \sqrt{N}\frac{0.02}{\sqrt{0.5(1 - 0.5)}}\right)$$

```
N=100
z <- sqrt(N)*0.02/0.5
1 - (pnorm(z) - pnorm(-z))
#> [1] 0.689
```

This is the p-value. In this case there is actually a large chance of seeing 52 or larger under the null hypothesis.

Note that there is a close connection between p-values and confidence intervals. If a 95% confidence interval of the spread does not include 0, we know that the p-value must be smaller than 0.05.

To learn more about p-values you can consult any statistcs textbook. However, in general we prefer reporting confidence intervals over p-values since it gives us an idea of the size of the estimate. The p-value simply reports a probability and says nothing about the significance of the finding in the context of the problem.

1.7 Statistical Models

"All models are wrong, but some are useful" -George E. P. Box

1.7.1 Poll Aggregators

As we described earlier, in the 2012 Nate Silver was giving a Obama a 90% chance of winning. Yet, none of the individual polls where that close. Political commentator Joe Scarborough said during his show

> Anybody that thinks that this race is anything but a tossup right now is such an ideologue … they're jokes.".

To which Nate Silver responded:

> If you think it's a toss-up, let's bet. If Obama wins, you donate $1,000 to the American Red Cross. If Romney wins, I do. Deal?

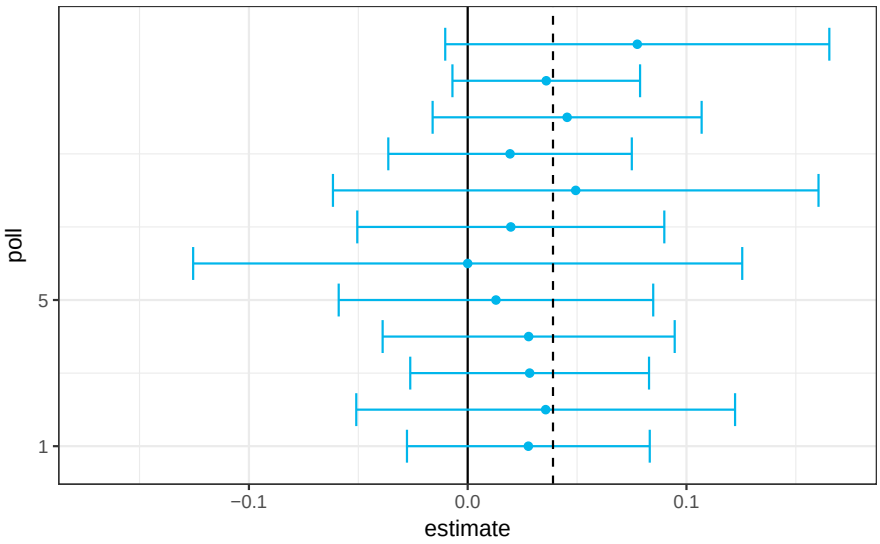
How was Mr. Silver so confident. We will demonstrate Mr. Silver saw that Mr. Scarborough did not using a Monte Carlo simulation. We generate results for 12 polls taken the week before the election. We mimic sample sizes from actual polls and construct. We construct and report 95% confidence intervals for each of the 12 polls:

```
d <- 0.039
Ns <- c(1298, 533, 1342, 897, 774, 254, 812, 324, 1291, 1056, 2172, 516)
p <- (d + 1)/2

confidence_intervals <- sapply(Ns, function(N) {
  X <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
  X_hat <- mean(X)
  SE_hat <- sqrt(X_hat*(1-X_hat)/N)
  2*c(X_hat, X_hat - 2*SE_hat, X_hat + 2*SE_hat)-1
})
```

Let's save the results from this simulation in a data frame:

```
polls <- data.frame(poll=1:ncol(confidence_intervals),
                    t(confidence_intervals),
                    sample_size=Ns)
names(polls)<-c("poll", "estimate", "low", "high", "sample_size")
polls
#>    poll estimate      low   high sample_size
#> 1     1   0.0277 -0.02776 0.0832        1298
#> 2     2   0.0356 -0.05093 0.1222         533
#> 3     3   0.0283 -0.02626 0.0829        1342
#> 4     4   0.0279 -0.03888 0.0946         897
#> 5     5   0.0129 -0.05896 0.0848         774
#> 6     6   0.0000 -0.12549 0.1255         254
#> 7     7   0.0197 -0.05047 0.0899         812
#> 8     8   0.0494 -0.06159 0.1604         324
#> 9     9   0.0194 -0.03629 0.0750        1291
#> 10   10   0.0455 -0.01603 0.1069        1056
#> 11   11   0.0359 -0.00697 0.0788        2172
#> 12   12   0.0775 -0.01026 0.1653         516
```

Here is a visualization of what the intervals the pollsters would have reported for the difference between Obama and Romney:



Not surprisingly, all 12 polls report confidence intervals that include the election night result (dashed line). However, all 12 polls include 0 (solid black line) as well. Therefore, individually, if asked for a prediction the pollsters would have to agree with Scarborough: it's a toss up. Below we describe how they are missing a key insight.

Poll aggregators, such as Nate Silver, realized that by combining the results of different polls you could greatly improve precision. By doing this, effectively, we are conducting a poll with a huge sample size. As a result we can report a smaller 95% confidence interval, and therefore a more precise prediction.

Although as aggregators we do not have access to the raw poll data we can use mathematics to reconstruct what we would have obtained had we made one large poll with

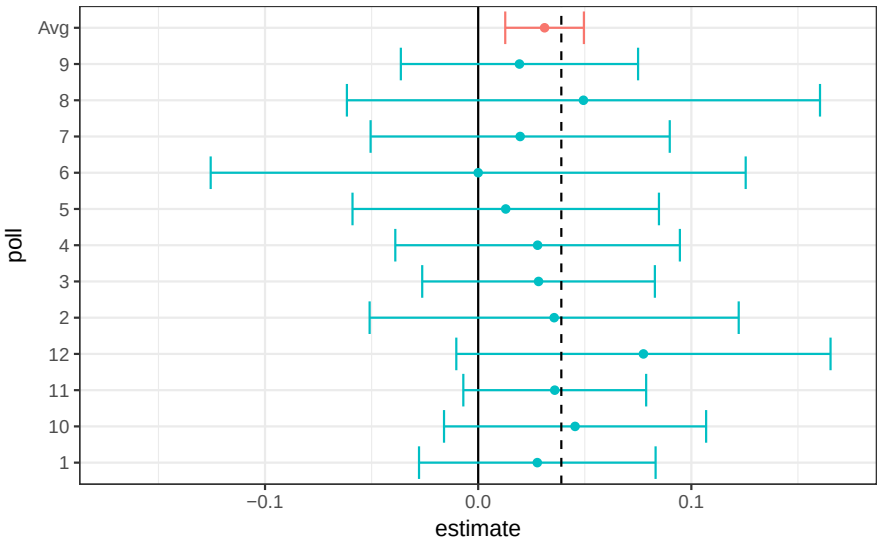
```
sum(polls$sample_size)
#> [1] 11269
```

participants. Basically we construct an estimate of the spread, let's call it d, with a weighted average in the following way:

```
d_hat <- polls %>%
  summarize(avg = sum(estimate*sample_size) / sum(sample_size)) %>%
  .$avg
```

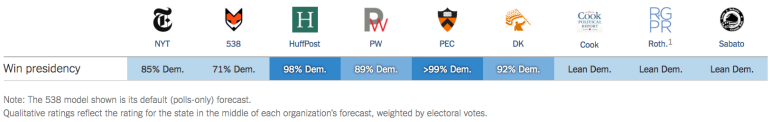
Once we have an estimate of d we can construct an estimate for the proportion voting for Obama which we can then use to estimate the standard error. Once we do this we see that our margin of error is 0.018.

Thus we can predict that the spread will be 3.1 plus or minus 1.8, which not only includes the actual result but is quite far from including 0. Once we combine the 12 polls we become quite certain that Obama will win the popular vote.



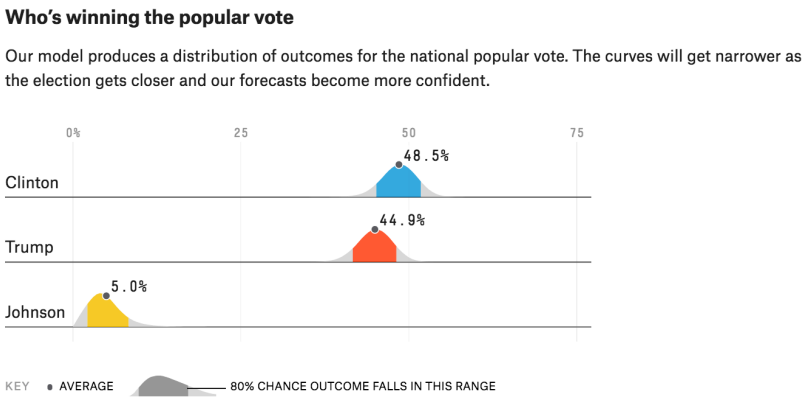
Note that this was just a simulation to illustrate the idea. The actual data science exercise of forecasting elections is much more complicated. It involves modeling. Below we explain how pollsters fit multilevel models to the data and use this to forecast election results. In the 2008 and 2012 US presidential elections Nate Silver used this approach to make an almost perfect predictions and silence the pundits.

Since the 2008 elections other organizations have started their own election forecasting group that, like Nate Silver, aggregate polling data and use statistical models to make predictions. In 2016, forecasters underestimated the Trump's chances of winning greatly.



For example, the Princeton Election Consortium gave Trump less than 1% while the Huffington Post gave him a 2%. In contrast, Fivethirtyeight had this probability at 29%, higher than tossing two coins and getting two heads. By understanding statistical models and how these forecasters use them, we will start to understand how this happened.

Although not nearly as interesting as predicting the electoral college, for illustrative purposes we will start by looking at predictions for the popular vote. Fivethirtyeight predicted a 3.6% advantage for Clinton, included the actual result of 2.1% (48.2% to 46.1%) in their interval, and was much more confident about Clinton winning this, giving her a 81.4% chance of winning.



We introduce actual data from the 2016 US presidential election to show how models are motivated and build to produce these predictions.

1.7.2 Poll data

We use public polling data organized by fivethirtyeight for the 2016 presidential election. The data is included as part of of the `dslabs` package:

```
data(polls_us_election_2016)
names(polls_us_election_2016)
#>  [1] "state"            "startdate"        "enddate"
#>  [4] "pollster"         "grade"            "samplesize"
#>  [7] "population"       "rawpoll_clinton"  "rawpoll_trump"
#> [10] "rawpoll_johnson"  "rawpoll_mcmullin" "adjpoll_clinton"
#> [13] "adjpoll_trump"    "adjpoll_johnson"  "adjpoll_mcmullin"
```

The table includes results for national polls as well as state polls taken during the year before the election. For this first illustrative example, we will filter the data to include national polls that happened during the week before the election. We also remove polls that fivethirtyeight has determined not to be reliable and graded with "B" or less. Some polls have not been graded and we include those:

```
polls <- polls_us_election_2016 %>%
  filter(state == "U.S." & enddate >= "2016-10-31" &
           (grade %in% c("A+","A","A-","B+") | is.na(grade)))
```

We add a spread estimate:

```
polls <- polls %>%
  mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

For illustrative purposes, we will assume that there are only two parties and call p the proportion voting for Clinton and $1-p$ the proportion voting for Trump. We are interested in the spread $2p-1$. Let's call the spread d (for difference).

Note that we have 49 estimates of the spread.
The theory we learned tells us that these estimates are a random variable with a probability distribution that is approximately normal. The expected value is the election night spread d and the standard error is $\sqrt{p(1-p)/N}$. Assuming the urn model we described earlier are good ones, we can use this information to construct a confidence interval based on the aggregated data. The estimated spread is:

```
d_hat <- polls %>% summarize(d_hat = sum(spread * samplesize) / sum(samplesize)) %>% .$d_hat
```

and the standard error is:

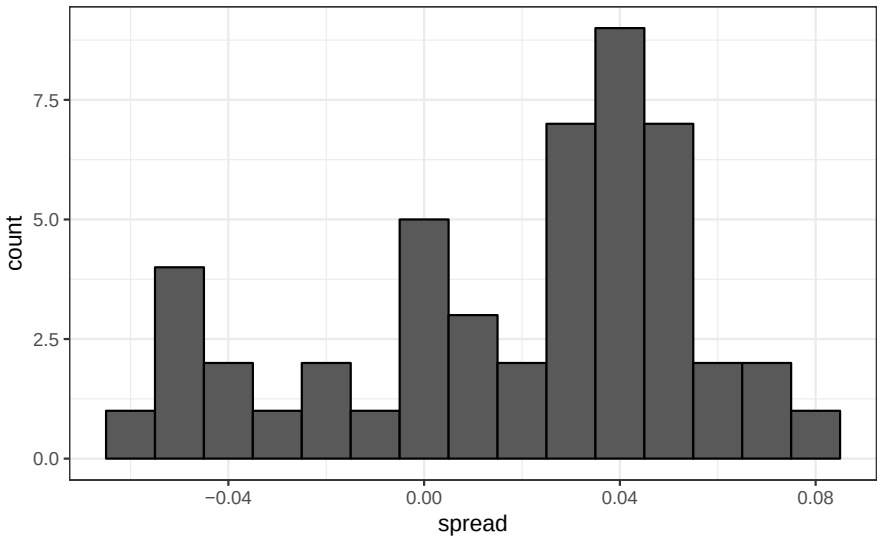
```
p_hat <- (d_hat+1)/2
moe <- 1.96 * 2 * sqrt(p_hat*(1-p_hat)/sum(polls$samplesize))
moe
#> [1] 0.00662
```

So we report a spread of 1.43% with a margin of error of 0.66%. On election night we find out that the actual percentage was 2.1% which is outside a 95% confidence interval.

What happened?

A histogram of the reported spreads shows a problem:

```
polls %>%
  ggplot(aes(spread)) +
  geom_histogram(color="black", binwidth = .01)
```



The data does not appear to be normally distributed and the standard error appears to be larger than 0.007. The theory is not quite working here.

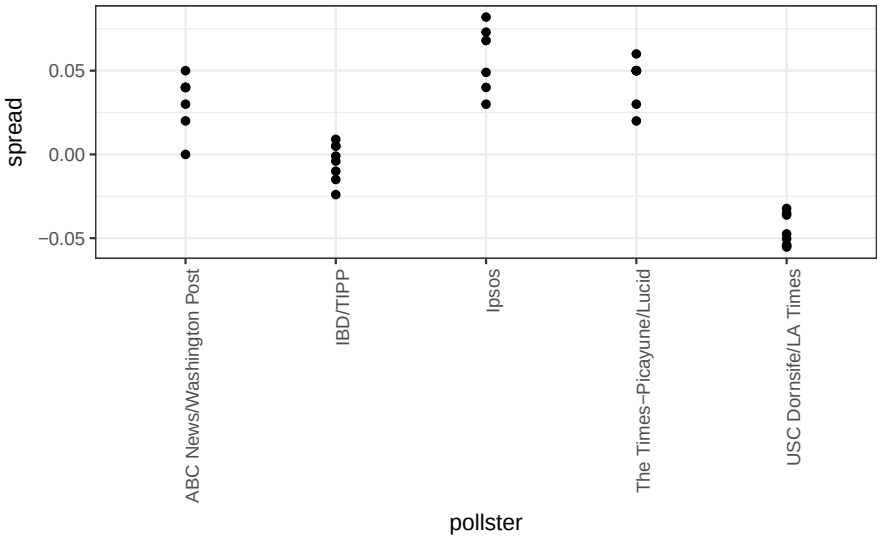
1.7.3 Pollster bias

Notice that various pollsters are involved and some are taking several polls a week:

```
polls %>% group_by(pollster) %>% summarize(n())
#> # A tibble: 15 x 2
#>                                                pollster `n()`
#>                                                  <fctr> <int>
#> 1                                ABC News/Washington Post     7
#> 2                                      Angus Reid Global     1
#> 3                                CBS News/New York Times     2
#> 4 Fox News/Anderson Robbins Research/Shaw & Company Research     2
#> 5                                               IBD/TIPP     8
#> 6                                          Insights West     1
#> # ... with 9 more rows
```

Let's visualize the data for the pollsters that are regularly polling:

```
polls %>% group_by(pollster) %>%
  filter(n() >= 6) %>%
  ggplot(aes(pollster, spread)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



This plot reveals a unexpected result. First note that the standard error predicted by theory for each poll:

```
polls %>% group_by(pollster) %>%
  filter(n() >= 6) %>%
  summarize(se = 2 * sqrt( p_hat * (1-p_hat) / median(samplesize)))
#> # A tibble: 5 x 2
#>                   pollster     se
#>                     <fctr>  <dbl>
#> 1 ABC News/Washington Post 0.0265
#> 2                 IBD/TIPP 0.0333
#> 3                    Ipsos 0.0225
#> 4 The Times-Picayune/Lucid 0.0196
#> 5    USC Dornsife/LA Times 0.0183
```

is between 0.018 and 0.033 which agrees with the within poll variation we see. However, there appears to be differences *across the polls*. Note for example how the USC Dornsife/LA Times pollster is predicting a 4% win for Trump, while Ipsos is predicting a win larger than 5% for Clinton. The theory we learned says nothing about different pollsters producing polls with different expected values. All the polls should have the same expected value. Fivethirtyeight refers to these differences as "house effects". We can also call them *pollster bias*.

Rather than use the urn model theory we are instead going to develop a data-driven model.

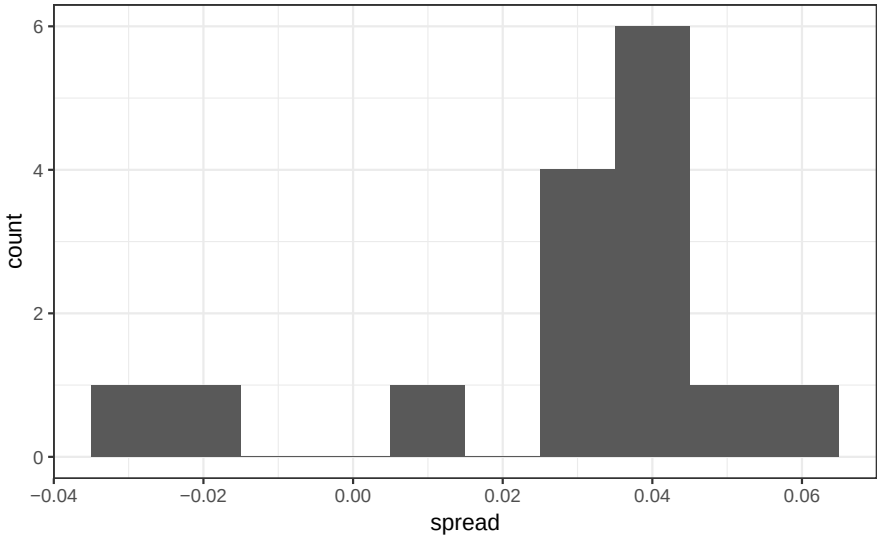
1.7.4 Data driven model

For each pollster, let's collect their last reported result before the election:

```
one_poll_per_pollster <- polls %>% group_by(pollster) %>%
  filter(enddate == max(enddate)) %>%
  ungroup()
```

Here is a histogram of the data for these 15 pollsters:

```
one_poll_per_pollster %>%
  ggplot(aes(spread)) + geom_histogram(binwidth = 0.01)
```



In the previous section we saw that using the urn model theory to combine these results might not be appropriate due to the pollster effect. Instead we will model this spread data directly.

The new model can also be thought as an urn model although the connection is not as direct. Rather than 0s (republicans) and 1s (democrats) our urn now contains poll results from all possible pollsters. We *assume* that the expected value of our urn is the actual spread $d = 2p - 1$.

Because rather than 0s and 1s, our urn contains continuous numbers between -1 and 1, the standard deviation of the urn is no longer $\sqrt{p(1-p)}$. Rather than voter sampling variability, the standard error now includes the pollster to pollster variability. Our new urn, also includes the sampling variability from the polling. Regardless, this standard deviation is now an unknown parameter. In statistics textbooks the Greek symbol σ is used to represent this parameter.

In summary, we have two unknown parameters: the expected value d and the standard deviation σ.

Our task is to estimate d. Because we model the observed values $X_1, \dots X_N$ as a random sample from the urn, the CLT still works in this situation because it is an average of independent random variables. For a large enough sample size N, the probability distribution of the sample average $\bar{X}$ is approximately normal with expected value μ and standard error $\sigma/\sqrt{N}$. If we are willing to consider $N = 15$ large enough, we can use this to construct confidence intervals.

A problem is that we don't know σ. But theory tells us that we can estimate the urn model σ with the *sample stadnard deviation* defined as:

$$s = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \bar{X})^2$$

Note that unlike for the population standard deviation definition, we now divide by $N - 1$. This makes s a better estimate of σ. There is a mathematical explanation for this, which is explained in most statistics textbooks, but we don't cover it here.

The `sd` function in R computes the sample standard deviation:

```
sd(one_poll_per_pollster$spread)
#> [1] 0.0242
```

We are now ready to form a new confidence interval based on our new data driven model:

```
results <- one_poll_per_pollster %>%
  summarize(avg = mean(spread), se = sd(spread)/sqrt(length(spread))) %>%
  mutate(start = avg - 1.96*se, end = avg + 1.96*se)
round(results*100,1)
#>   avg  se start end
#> 1 2.9 0.6   1.7 4.1
```

Note that our confidence interval is wider now as it incorporates the pollster variability. It does include the election night result of 2.1%. Also note that it was small enough not to include 0 which means we were confident Clinton would win the electoral vote.

Are we know ready to declare a probability of Clinton winning the popular vote? Not yet. In our model d is a fixed parameters so we can't talk about probabilities. To provide probabilities, we will need to learn about Bayesian statistics.

1.8 Bayesian Statistics

```
library(tidyverse)
library(dslabs)
ds_theme_set()
```

What does it mean when an election forecasters tell us that a given candidate has a 90% chance of winning? In the context of the urn model this would be equivalent to stating the the probability $p > 0.5$ is 90%. But, as we discussed, in the urn model p is a fixed parameter and it does not make sense to talk about probability. With Bayesian statistics, we assume it is in fact random.

Forecaster also use models to describe variability at different levels. For example, sampling variability, pollster to pollster variability, day to day variability, and election to election variability. One of the most successful approaches used for this are hierarchical models, which can be explained in the context of Bayesian statistics.

1.8.1 Bayes theorem

We start by reviewing Bayes theorem. We do this using a hypothetical cystic fibrosis test as an example. Suppose a test for cystic fibrosis has an accuracy of 99%. We will use the following notation:

$$\text{Prob}(+ \mid D = 1) = 0.99, \text{Prob}(- \mid D = 0) = 0.99$$

with + meaning a positive test and D representing if you actually have the disease (1) or not (0).

Suppose we select a random person and they test positive, what is the probability that they have the disease? We write this as $\text{Prob}(D = 1 \mid +)$? The cystic fibrosis rate is 1 in 3,900 which implies that $\text{Prob}(D = 1) = 0.00025$. To answer this question we will use Bayes Theorem, which in general tells us that:

$$\Pr(A \mid B) = \frac{\Pr(B \mid A)\Pr(A)}{\Pr(B)}$$

This equation applied to our problem becomes:

$$\begin{aligned}\Pr(D = 1 \mid +) &= \frac{P(+ \mid D = 1) \cdot P(D = 1)}{\Pr(+)} \\ &= \frac{\Pr(+ \mid D = 1) \cdot P(D = 1)}{\Pr(+ \mid D = 1) \cdot P(D = 1) + \Pr(+ \mid D = 0)\Pr(D = 0)}\end{aligned}$$

Plugging in the numbers we get:

$$\frac{0.99 \cdot 0.00025}{0.99 \cdot 0.00025 + 0.01 \cdot (.99975)} = 0.02$$

This says that despite the test having 0.99 accuracy, the probability of having the disease given a positive test is only 0.02. This may appear counter-intuitive to some. The reason this is the case is because we have to factor in the very rare probability that a person, chosen at random, has the disease. To illustrate this we run a Monte Carlo simulation.

1.8.1.1 Simulation

The following simulation is meant to help you visualize Bayes Theorem. We start by randomly selecting 100,000 people from a population in which the disease in question has a 5% prevalence.

```
prev <- 0.00025
N <- 100000
outcome <- sample(c("Disease","Healthy"), N, replace = TRUE, prob = c(prev,1-prev))
```

Note that there are very few with the disease

```
N_D <- sum(outcome == "Disease")
N_D
#> [1] 23
N_H <- sum(outcome == "Healthy")
N_H
#> [1] 99977
```

Also there are many without the disease which makes probably that we will see some false positives given that the test is not perfect. Now each person gets the test which is correct 90% of the time.

```
accuracy <- 0.99
test <- vector("character",N)
test[outcome=="Disease"]  <- sample(c("+","-"), N_D, replace=TRUE, prob = c(accuracy, 1 - accuracy))
test[outcome=="Healthy"]  <- sample(c("-","+"), N_H, replace=TRUE, prob = c(accuracy, 1 - accuracy))
```

Because there are so many more controls than cases, even with a low false positive rate, we get more controls than cases in the group that tested positive (code not shown):

```
table(outcome, test)
#>           test
#> outcome       -     +
#>   Disease     0    23
#>   Healthy 99012   965
```

From this table we see that the proportion of positive tests that have the disease is 23 out of 988. We can run this over and over again to see that in fact the probability converges to about 0.022 The proportions of in the top plot shows $\Pr(D = 1)$. The bottom left shows $\Pr(D = 1 \mid +)$ and the bottom right shows $\Pr(D = 0 \mid +)$.

1.9 Bayes in practice

José Iglesias is a professional baseball player. In April 2013, when he was starting his career, he was performing rather well:

Month	At Bats	H	AVG
April	20	9	.450

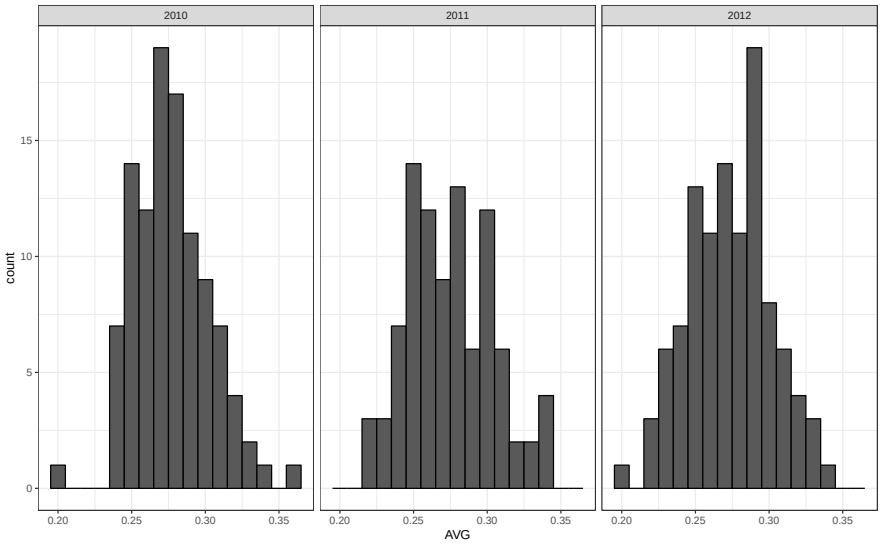


Figure 1: Batting average histograms for 2010, 2011, and 2012.

Month	At Bats	H	AVG

The batting average (`AVG`) statistic is one way of measuring success. Roughly speaking, it tells us the success rate when batting. An `AVG` of .450 means José has been successful 45% of the times he has batted (`At Bats`) which is rather high, historically speaking. Note, for example, that no one has finished a season with an `AVG` of .400 or more since Ted Williams did it in 1941! To illustrate the way hierarchical models are powerful, we will try to predict José's batting average at the end of the season. In a typical season, players have about 500 at bats.

With the techniques we have learned up to now, referred to as *frequentist techniques*, the best we can do is provide a confidence interval. We can think of outcomes from hitting as a binomial with a success rate of p. So if the success rate is indeed .450, the standard error of just 20 at bats is:

$$\sqrt{\frac{.450(1-.450)}{20}} = .111$$

This means that our confidence interval is .450-.222 to .450+.222 or .228 to .672.

This prediction has two problems. First, it is very large, so not very useful. Second, it is centered at .450 which implies that our best guess is that this new player will break Ted Williams' record.

If you follow baseball, this last statement will seem wrong and this is because you are implicitly using a hierarchical model that factors in information from years of following baseball. Here we show how we can quantify this intuition.

First, let's explore the distribution of batting averages for all players with more than 500 at bats during the previous three seasons:

We note that the average player had an `AVG` of .275 and the standard deviation of the population of players was 0.027. So we can see already that .450 would be quite an anomaly since it is over six stander deviations away from the mean.

So is José lucky or the best batter seen in the last 50 years? Perhaps it's a combination of both. But how lucky and how good is he? If we become convinced that he is lucky, we should trade him to a team that trusts the .450 observation and is maybe overestimating his potential.

1.9.1 The hierarchical model

The hierarchical model provides a mathematical description of how we came to see the observation of .450. First, we pick a player at random with an intrinsic ability summarized by, for example, p, then we see 20 random outcomes with success probability p.

We use a model to represents two levels of variability in our data. First each player is assigned a nature ability to hit at birth. We will use the symbol p to represent this ability. You can think of θ as the batting average you would converge to if this particular player batted over and over again.

Based on the plots above we assume that p has a normal distribution. With expected value .270 and standard error 0.027.

Now the second level of variability has to do with luck when batting. Regardless of how good the player is, sometimes you have bad luck and sometimes you have good luck. At each at bat this player has a probability of success p. If we add up these success and failures, then the CLT tells us that the observed average, call it Y, has a normal distribution with expected value p and standard error $\sqrt{p(1-p)/\sqrt{N}}$ with N the number of at bats.

Statistical text books will write the model like this:

$$\begin{aligned} p &\sim N(\mu, \tau^2) \text{ describes randomness in picking a player} \\ Y \mid p &\sim N(p, \sigma^2) \text{ describes randomness in the performance of this particular player} \end{aligned}$$

with $\mu = .270$, $\tau = 0.027$, and $\sigma^2 = p(1-p)/N$.

Note the two levels (this is why we call them hierarchical): 1) Player to player variability and 2) variability due to luck when batting. In a Bayesian framework, the first level is called a *prior distribution* and the second the *sampling distribution*.

Now, let's use this model for José's data. Suppose we want to predict his innate ability in the form of his *true* batting average p. This would be the hierarchical model for our data:

$$\begin{aligned} p &\sim N(.275, .027^2) \\ Y \mid p &\sim N(p, .111^2) \end{aligned}$$

We now are ready to compute a posterior distribution to summarize our prediction of p. The continuous version of Bayes rule can be used here to derive the *posterior probability function*, which is the distribution of p assuming we observe $Y = y$. In our case we can show follows a normal distribution with expected value:

$$\begin{aligned} \mathrm{E}(p \mid y) &= B\mu + (1-B)Y \\ &= \mu + (1-B)(Y-\mu) \\ B &= \frac{\sigma^2}{\sigma^2 + \tau^2} \end{aligned}$$

Note that this is a weighted average of the population average μ and the observed data Y. The weight depends on the SD of the population τ and the SD of our observed data σ. This weighted average is sometimes referred to as *shrinking* because it *shrinks* estimates towards a prior mean. In the case of José Iglesias, we have:

$$\begin{aligned} \mathrm{E}(p \mid Y = .450) &= B \times .275 + (1-B) \times .450 \\ &= .275 + (1-B)(.450 - .275) \\ B &= \frac{.111^2}{.111^2 + .027^2} = 0.944 \\ \mathrm{E}(p \mid Y = 450) &\approx .285 \end{aligned}$$

The standard error can be shown to be:

$$\text{SE}(p \mid y)^2 = \frac{1}{1/\sigma^2 + 1/\tau^2} = \frac{1}{1/.111^2 + 1/.027^2} = 0.00069$$

and the standard deviation is therefore 0.026. So we started with a frequentest 95% confidence interval that ignored data from other players and summarized just José's data: .450 ± 0.220. Then we used a Bayesian approach that incorporated data from other players and other years to obtain a posterior probability. This is actually referred to as an empirical Bayes approach because we used data to construct the prior. From the posterior we can report what is called a 95% credible interval by reporting a region, centered at the mean, with a 95% chance of occurring. In our case, this turns out to be: .285 ± 0.052.

The Bayesian credible interval suggests that if another team is impressed by the .450 observation, we should consider trading José as we are predicting he will be just slightly above average. Interestingly, the Red Sox traded José to the Detroit Tigers in July. Here are the José Iglesias batting averages for the next five months.

Month	At Bat	Hits	AVG
April	20	9	.450
May	26	11	.423
June	86	34	.395
July	83	17	.205
August	85	25	.294
September	50	10	.200
Total w/o April	330	97	.293

Although both intervals included the final batting average, the Bayesian credible interval provided a much more precise prediction. In particular, it predicted that he would not be as good the remainder of the season.

1.10 Election Forecasting

1.10.1 Bayesian Approach

Pollsters tend to make probabilistic statements about the results of the election. "The chance that Obama wins the electoral colleges is 91%" is a probabilistic statement about the parameter d. We showed that for the 2016 election, Fivethirtyeight eight gave Clinton a 81.4% chance of winning the popular vote. To do this they use the Bayesian approach we described.

We assume a hierarchical model similar to what we did to predict the performance of a baseball player

Statistical text books will write the model like this:

$$d \sim N(\mu, \tau^2) \text{ describes our best guess had we not seen any polling data}$$
$$\bar{X} \mid d \sim N(d, \sigma^2) \text{ describes randomness due to sampling and the pollster effect}$$

For our best guess we note that before any poll data is available we can use data sources other than polling data. A popular approach is to use what are called *fundamentals* which are based on properties about the current economy that historically appear to have an effect in favor or against of the incumbent party. We won't use these here. Instead we will use $\mu = 0$ which is interpreted as a model that simply does not provide any information on who will win. For the standard deviation we will use recent historical data that shows the winner of the popular vote has an average spread of about 3.5%. Therefore we set $\tau = 0.035$.

Now we can use the formulas for the posterior distribution for the parameter d: the probability of $d > 0$ given the observed poll data:

Contents

1 Regression

1.1 Motivating Example: Money Ball

Moneyball: The Art of Winning an Unfair Game is a book by Michael Lewis about the Oakland Athletics baseball team and its general manager, the person tasked with building the team, Billy Beane.

Baseball teams use *scouts* to help them decide what players to hire. These scouts evaluate players by observing them perform. Scouts tend to favor athletic players with observable physical abilities. For this reason scouts tend to agree on who the best players are and as consequence these players tend to be in high demand. This in turn drives up their salaries.

In 1989-1991 the A's had one of the highest payrolls in baseball. They were able to buy the best players and during that time they were one of the best teams in baseball. However, in 1995 the A's team owner changed, and the new management cut the budget drastically leaving then general manager, Sandy Alderson, with one of the lowest payrolls in baseball. He could no longer afford the most sought after players. Alderson began using a statistical approach to find inefficiencies in the market. Alderson was a mentor to Billy Beane, who succeed him in 1998 and fully embraced data science, as opposed to scouts, as a method for finding low cost players that data predicted would help the team win. Today this strategy has been adapted by most baseball teams. As we will see, regression plays a large role in this approach.

As motivation for this chapter we will go back to 2002 and try to build a baseball team with a limited budget. Note that, in 2002, the Yankees' payroll of $125,928,583 more than tripled the Oakland A's $39,679,746 budget.

Statistics have been used in baseball since its beginnings. Note that the dataset we will be using, included in the `Lahman` library, goes back to the 19th century. For example, a summary statistics we will describe soon, *Batting average*, has been used to summarize a batter's success for decades. Other statistics such as home runs (HR), runs batted in (RBI) and stolen bases (SB) are reported for each player in the game summaries included in the sports section of newspapers and players rewarded for high numbers.

Although summary statistics were widely used in baseball, data analysis per se was not. These statistics were arbitrarily decided on without much thought as to weather they actually predicted or were related to helping a team win.

This changed with Bill James. In the late 1970s, this aspiring writer and baseball started publishing articles describing more in-depth analysis of baseball data. He named the approach of using data to predict what outcomes best predicted if a team won sabermetrics. Until Billy Bean made sabermetrics the center of his baseball operation, Billy Bean's work was mostly ignored by the baseball world. Today pretty much every team uses the approach and which has gone beyond baseball into other sports.

In this chapter, to simplify the exercise we will focus on scoring runs and ignore pitching and fielding. We will see how regression analysis can help develop strategies to build a competitive baseball team with a constrained budget. The approach can be divided into two separate data analysis. In the first we determine which recorded player-specific statistics predict runs. In the second we examine if players were undervalued based on what our first analysis predicts.

1.1.1 Baseball Basics

We actually don't need to understand all the details about the game of baseball, which has over 100 rules, to see how regression will help us find undervalues players. Here we distill the sports to the basic knowledge ones need to know to effectively attack the data science problem.

The goal of a baseball game is to score more runs (points) than the other team. Each team has 9 batters that bat in a predetermined order. After the 9th batter hits, we start with the first again. Each time they come to bat we call it a plate appearance (PA). At each PA, the other team's *pitcher* throws the ball and you try to hit it. The PA ends with an binary outcome: you either make an *out* (failure) and sit back down or you don't (success) and you get to run around the bases, and potentially score a run. Each team gets nine tries, referred to as *innings*, to score runs and each inning ends after three outs (failures).

Here is a success and here is a failure. From these videos we see how luck is involved in the process. When you bat, you want to hit the ball hard. If you hit it hard enough it is a HR, the best possible outcome as you get at least one automatic run. But sometimes, due to chance, you hit the ball very hard and a defender catches it resulting in an out. In contrast, sometimes you hit the ball softly but it lands just in the right place. The fact that there is chance involved hints at why probability models will be involved.

Now there are several ways to succeed. Understanding this distinction will be important for our analysis. When you hit the ball you want to pass as many *bases* as possible. There are four bases with the fourth one called *home plate*. Home plate is is where you start by trying to hit, so the bases form a cycle.

If you get home you score a run. We are simplifying a bit but there are five ways you can succeed (not make an out):

- Bases on balls (BB) - the pitcher does not pitch well so you got to first base.
- Single - You hit the ball and get to first base.
- Double (X2B) - You hit the ball and get to second base
- Triple (X3B) - You hit the ball and get to third base
- Home Run (HR) - You hit the ball and go all the way home and score a run.

Here is an example of a HR.

If you get to a base, you still have the chance of getting home and scoring a run if the next batter hits successfully. While you are *on base* you can also try to steal a base (SB). If you run fast enough you can go try to go from first to second without the other team tagging you. Here is an example of a stolen base.

1.1.2 No awards for BB

Historically, the *batting average* has been considered the most important offensive statistic. To define this average we define a *hit* (H) and an *at bat* (AB). Singles, doubles, triples and home runs are hits. The fifth way to be successful, BB, is not a hit. An AB is the number of times you either get a hit or make an out, BBs are excluded. The batting average is simply H/AB and is considered the main measure of a success rate. Today this success rate ranges from 20% to 38%. We refer to the batting average in thousands so for example if your success rate is 25%, we call it batting 250.

One of Bill James first important insights is that the batting average ignore BB, but a BB is a success. So a player that gets many more BB than the average player might not be recognized if he does not excel in batting average. But is this player not helping produce runs? No award is given to the player with the most BB. In contrast, total stolen bases were considered important and an award given to the player with the most. But players with high totals of SB, also made more outs as they did not always succeed. Does a player with high SB totals help produce runs?

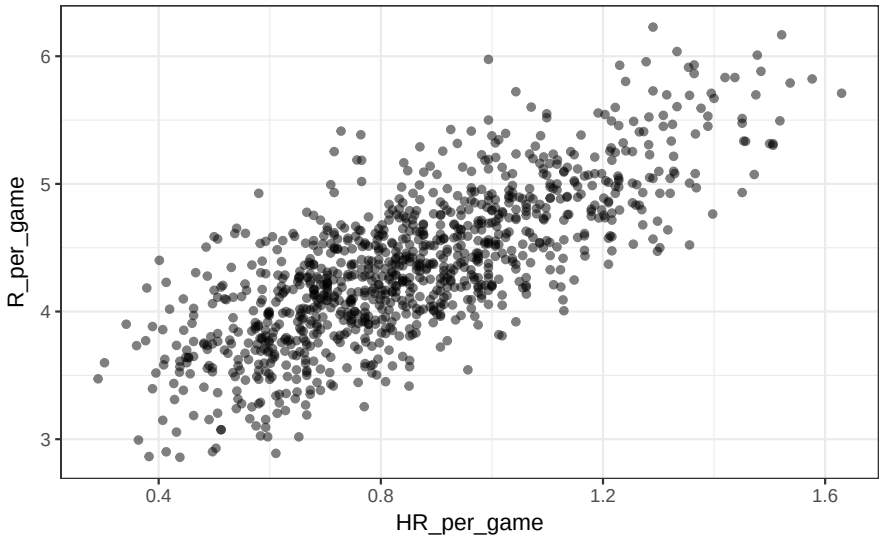
Can we use data science to determine if it's better to pay for BB or SB?

1.1.3 Base on Ball or Stolen bases?

One of the challenges in this analysis is that it is not obvious how to determine if a player produces runs because so much depends on his teammates. We do keep track of the number of runs scored by a player. But note that if you hit after someone who hits many HRs you will score many runs. But these runs don't necessarily happen if we hire this player but not his HR hitting teammate. However, we can examine team-level statistics. How do teams with many SB compare to teams with few? How about BB? We have data! Let's examine some.

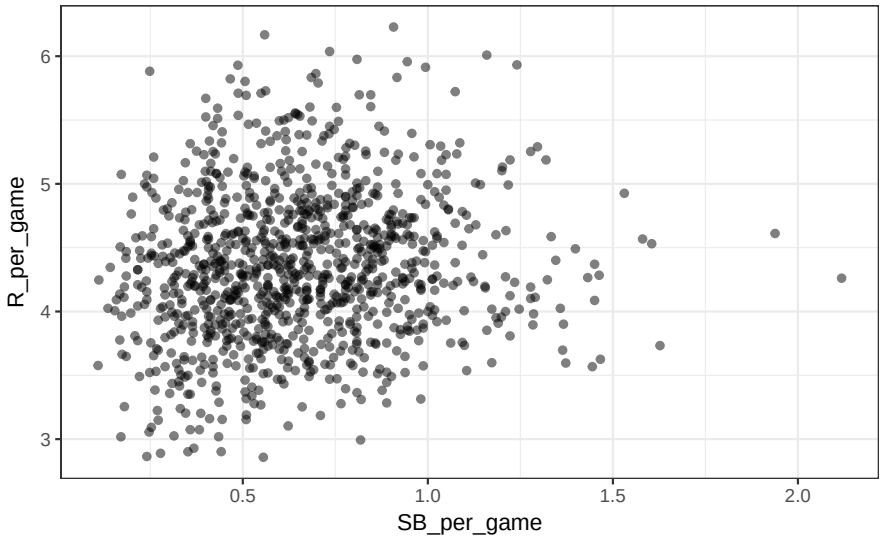
Let's start with an obvious one: HRs. Do teams that hit more home runs score more runs? We examine data from 1961 to 2001. The visualization of choice when exploring the relationship between two variables, such as HRs and wins, is a scatter plot.

```
library(Lahman)
library(tidyverse)
library(dslabs)
ds_theme_set()

Teams %>% filter(yearID %in% 1961:2001 ) %>%
  mutate(HR_per_game = HR/G, R_per_game = R/G) %>%
  ggplot(aes(HR_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
```



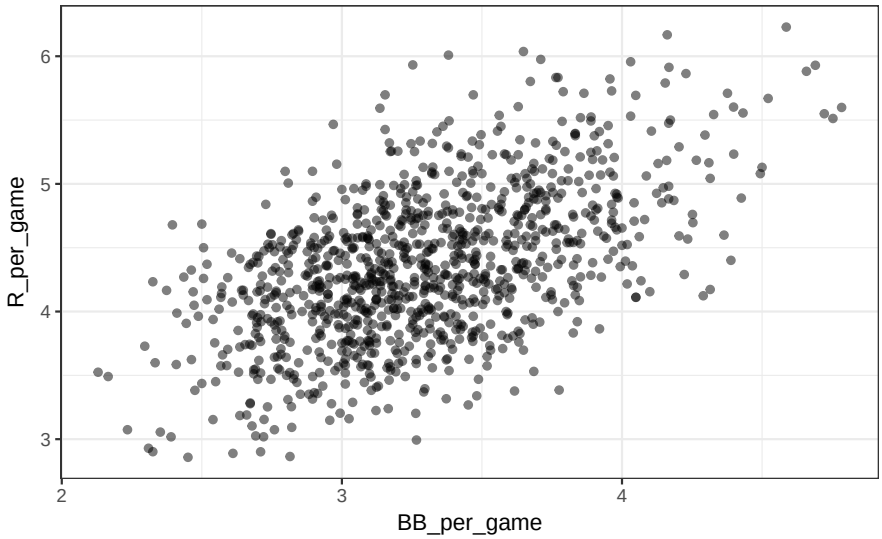
The plot shows a strong association: teams with more HRs tended to score more runs. Now let's examine the relationship between stolen bases and wins

```
Teams %>% filter(yearID %in% 1961:2001 ) %>%
  mutate(SB_per_game = SB/G, R_per_game = R/G) %>%
  ggplot(aes(SB_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
```



Here the relationship is not as clear. Finally, let's examine the relationship between BB and runs.

```
Teams %>% filter(yearID %in% 1961:2001 ) %>%
  mutate(BB_per_game = BB/G, R_per_game = R/G) %>%
  ggplot(aes(BB_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
```

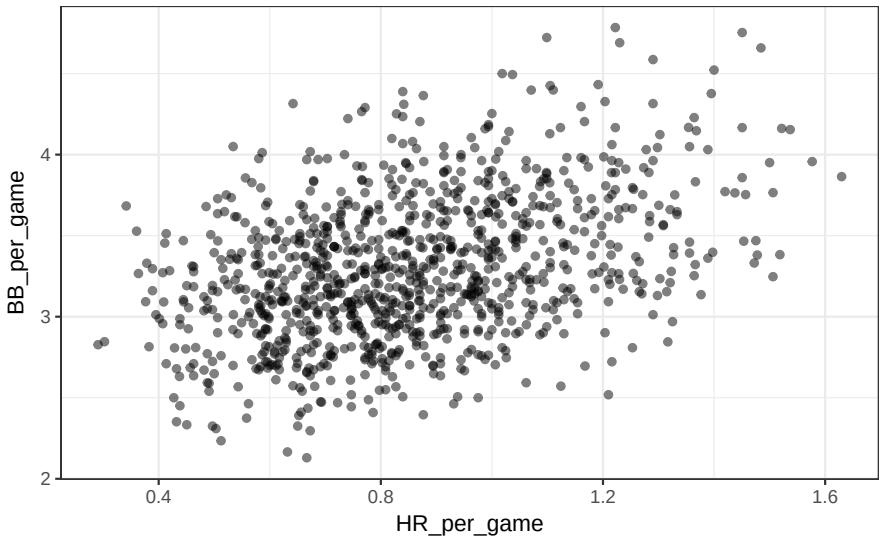


Here again we see a clear association. But does this mean that increasing BB will cause an increase in runs? Here we give an important warning:

Assocation is not causation

In fact, it looks like BB and HRs are also associated

```
Teams %>% filter(yearID %in% 1961:2001 ) %>%
  mutate(HR_per_game = HR/G, BB_per_game = BB/G) %>%
  ggplot(aes(HR_per_game, BB_per_game)) +
  geom_point(alpha = 0.5)
```



We know that, by definition, HR cause runs because. It could be that HRs also cause BB make it appear as if BB cause runs. This is called confounding, an important concept we will learn about.

Linear regression will help us parse all this out and quantify the associations. This will then help us determine what players to recruit. Specifically, we will try to predict things like how many more runs will a team scores if we increase the number BBs but keep the HRs fixed? Regression will help us answer this question.

1.2 Motivation

```
library(tidyverse)
library(dslabs)
ds_theme_set()
set.seed(0)
```

Up to now this book has focused mainly on *univariate* variables. However, in data science applications it is very common to be interested in the relationship between two or more variables. In our baseball example we are interested in the relationship, for example, between Bases on Balls and Runs. We will come back to this example, but we introduce necessary concepts needed to understand regression using a simpler example. We actually use the dataset from which regression was born.

The example is from Genetics. Francis Galton studied the variation and heredity of human traits. Among many other traits, Galton collected and studied height data from families to try to understand heredity. While doing this he developed the concepts of correlation and regression and a connection to pairs of data that follow a normal distribution. Note that at the time this data was collected, what we know today about genetics was not yet understood. A very specific question Galton tried to answer was how well can we predict a son's height based on the parents' height. The technique he developed to answer this question, regression, can also be applied to our baseball question: do bases on ball predict runs?

We have access to Galton's family height data through the `HistData` package. We will create a dataset with the heights of fathers and the first son of each family.

```
library(HistData)
data("GaltonFamilies")
galton_heights <- GaltonFamilies %>%
  filter(childNum == 1 & gender == "male") %>%
  select(father, childHeight) %>%
  rename(son = childHeight)
```

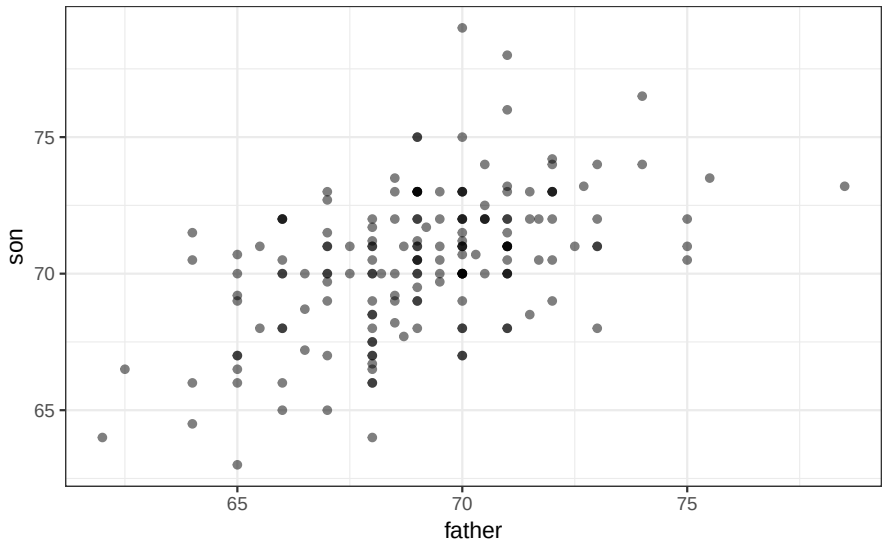


Figure 1: Heights of father and son pairs plotted against each other.

Supposed we were asked summarize these data. Since both distributions are well approximated by the normal distribution, we could use the two averages and two standard deviations as summaries:

```
galton_heights %>%
  summarize(mean(father), sd(father), mean(son), sd(son))
#>   mean(father) sd(father) mean(son) sd(son)
#> 1         69.1       2.55      70.5    2.56
```

However, this summary fails to describe an important characteristic of the data:

```
galton_heights %>% ggplot(aes(father, son)) +
  geom_point(alpha = 0.5)
```

the trend that the taller the father, the taller the son.

We will learn that the correlation coefficient is a summary of how two variable move together and then see how this is used to predict.

1.3 Correlation

The correlation coefficient is defined for a list of pairs $(x_1, y_1), \ldots, (x_n, y_n)$ as:

$$\rho = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \mu_x}{\sigma_x}\right)\left(\frac{y_i - \mu_y}{\sigma_y}\right)$$

with μ_x, μ_y the averages of $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ respectively and σ_x, σ_y the standard deviations. The Greek letter ρ is commonly used in statistics books to denote the correlation. The reason is that ρ is the Greek letter for r, the first letter of regression. Soon we learn about the connection between correlatio and regression.

To understand why this equation does in fact summarize how two variables move together consider the i-th entry of x is $\left(\frac{x_i-\mu_x}{\sigma_x}\right)$ SDs away from the average. Similarly, the y_i that is paired with x_i, is $\left(\frac{y_1-\mu_y}{\sigma_y}\right)$ SDs away from the average y. If x and y are unrelated the product $\left(\frac{x_i-\mu_x}{\sigma_x}\right)\left(\frac{y_i-\mu_y}{\sigma_y}\right)$ will be positive ($+\times+$ and $-\times-$) as often as negative ($+\times-$ and $-times+$) and will average out to about 0. This correlation is this average and therefore unrelated variables will have 0 correlation. If instead the quantities vary together

then we are averaging mostly positive products ($+ \times +$ and $- \times -$) and we get a positive correlation. If they vary in opposite directions we get a negative correlation.

Note that the correlation is between -1 and 1. We can show this mathematically. To see this, consider that we can't have higher correlation than when we compare a list to itself (perfect correlation). In this case the correlation is

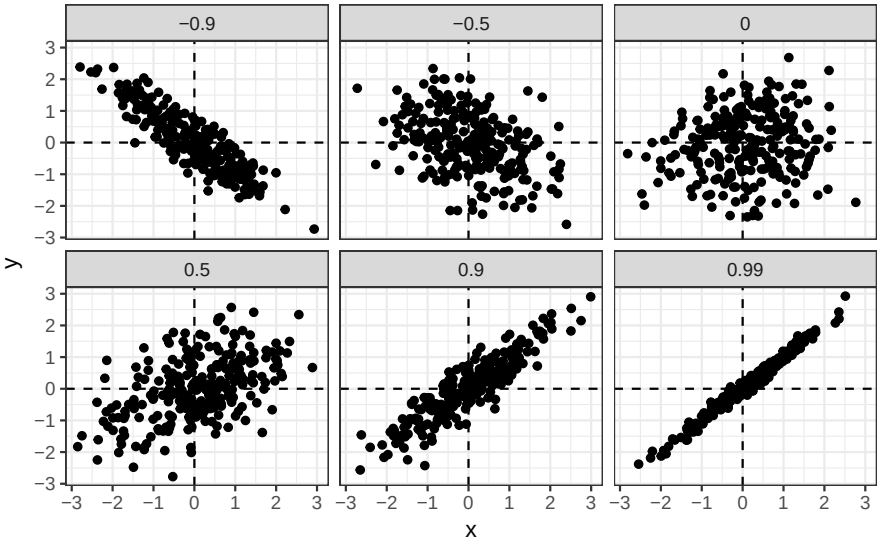
$$\rho = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \mu_x}{\sigma_x}\right)^2 = 1/\sigma^2\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)^2 = 1$$

A similar arguments with x and it's exact opposite proves the correlation has to be bigger or equal to -1.

So, for example, the correlation between a variable and itself is 1 and the correlation between a variable and it's negative is -1. For other pairs, the correlation is in between -1 and 1. For example, the correlation between father and son's heights is about 0.5:

```
galton_heights %>% summarize(cor(father, son))
#>   cor(father, son)
#> 1            0.501
```

To see what data looks like for different values of ρ, here are six examples of pairs with correlations ranging from -0.9 to 0.99:



1.3.1 Sample Correlation is a Random Variable

Before we continue connecting correlation to regression, lets remind ourselves about random variability.

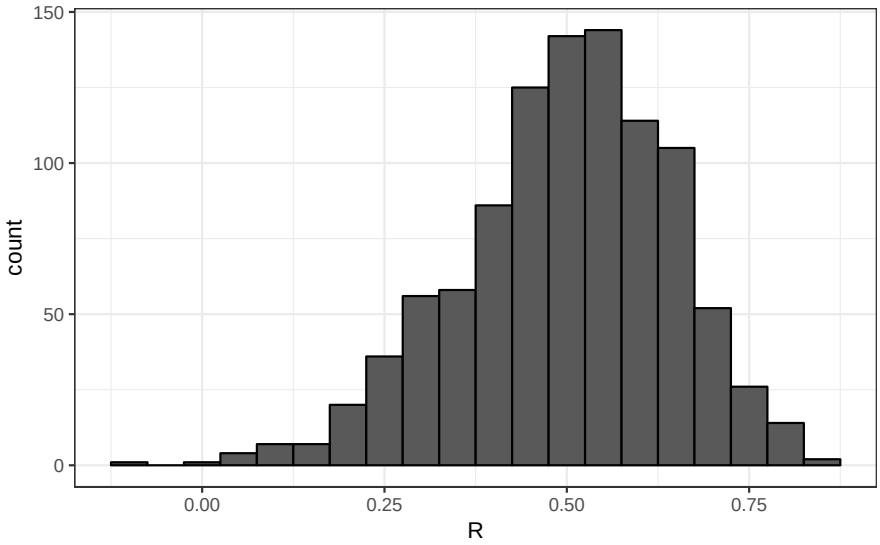
In most data science application we do not observe the population but rather a sample. As with the average and standard deviation, the **sample** correlation is the most commonly used estimate of the population correlation. This implies that the correlation we compute and use as a summary is a random variable.

As in illustration let's assume that the 179 pairs of fathers and sons is our entire population. A less fortunate geneticists can only afford measurements from a random sample of 25 pairs. The sample correlation of the sample

```
R <- sample_n(galton_heights, 25, replace = TRUE) %>%
  summarize(cor(father, son))
```

is a random variable. We can run a Monte Carlo simulation to see its distribution:

```
B <- 1000
N <- 25
R <- replicate(B, {
  sample_n(galton_heights, N, replace = TRUE) %>%
    summarize(r=cor(father, son)) %>% .$r
})
data.frame(R) %>% ggplot(aes(R)) + geom_histogram(binwidth = 0.05, color = "black")
```



We see that is the expected value is the population correlation:

```
mean(R)
#> [1] 0.498
```

and that it has a relatively high standard error relative to it's size.

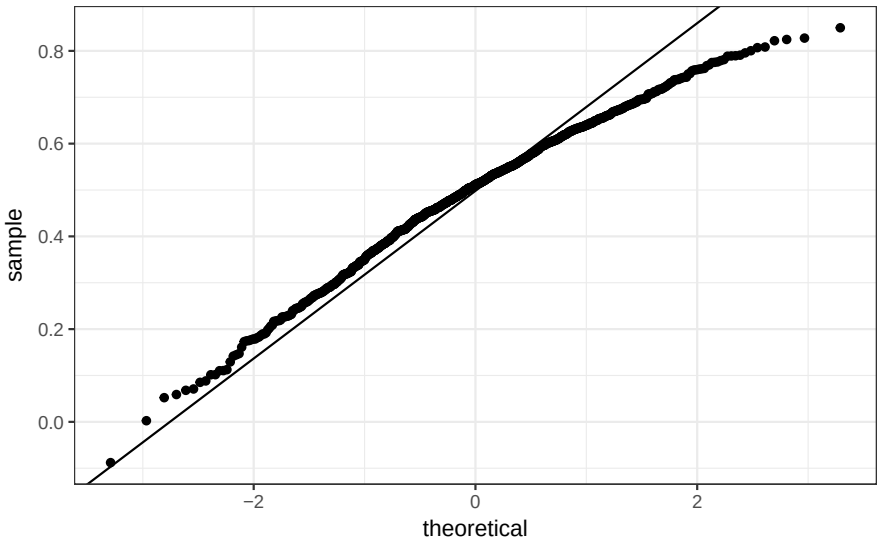
```
sd(R)
#> [1] 0.145
```

That there large, relative to its difference from 0, standard deviation is something to keep in mind when interpreting correlations.

Also note that because the sample correlation is an average of independent draws, the central limit actually applies. Therefore for large enough N the distribution of `R` is approximately normal with expected value ρ. The standard deviation is somewhat complex to derive: it is $\sqrt{\frac{1-r^2}{N-2}}$.

In our example, $N = 25$ does not seem to be large enough to make the approximation a good one:

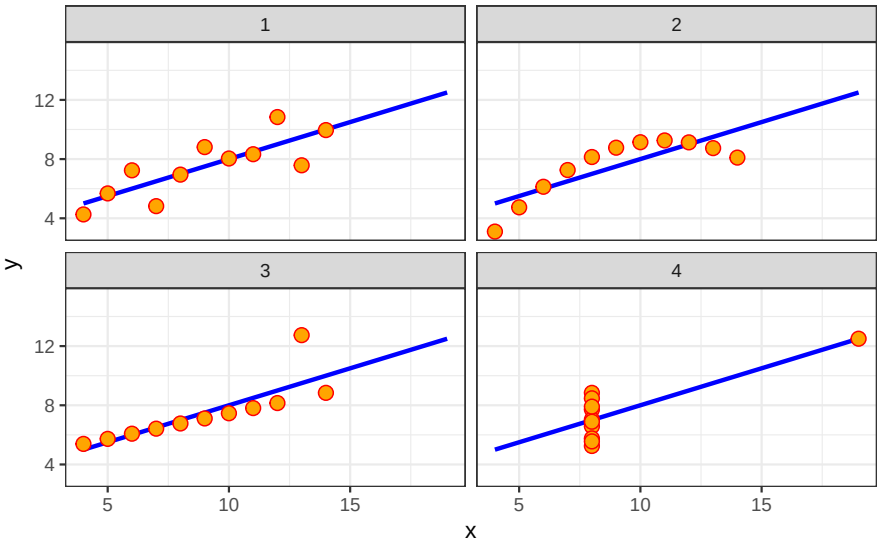
```
data.frame(R) %>%
  ggplot(aes(sample=R)) +
  stat_qq() +
  geom_abline(intercept = mean(R),
              slope = sqrt((1-mean(R)^2)/(N-2)))
```



If you increase N you will see the distribution converging to normal.

1.3.2 When is correlation a useful summary?

Correlation is not always a good summary of the relationship between two variables. A famous example used to illustrate this, are the following four artificial datasets, referred to as Anscombe's quartet. All these pairs have a correlation of 0.82:



Correlation is only meaningful in a particular context. To help us understand when it is that correlation is meaningful, as a summary statistic, we will get back to the example of predicting son's height using the father's height. This will help motivate and define linear regression. We start by demonstrating how correlation can be useful for prediction.

1.4 Stratification

Suppose we are asked to guess the height of a randomly selected son and we don't know his father's height. Because the distribution of son heights is approximately normal, we know the average height, 70.455, is the value with the highest proportion and would be the prediction with the chances of minimizing the error. But what if we are told that the father is 72 inches tall, do we sill guess 70.455 for the son?

It turns out that if we were able to collect data from a very large number of fathers that are 72 inches the distribution of their sons' heights would be normally distributed. This implies that the average of this distribution would be our best prediction. Can we figure out what this average is?

We call this approach *stratifying* or *conditioning*. The distribution of the strata is called the *conditional distribution* and the average of this distribution the *conditional aveage*. In our case we are computing the average son height *conditioned* on the father being 72 inches tall. A challenge when using this approach in practice is that for continuous data we don't have many data points matching exactly one value. For example, we have only

```
sum(galton_heights$father == 72)
#> [1] 8
```

fathers that are exactly 72 inches. If we change the number to 72.5 we only get

```
sum(galton_heights$father == 72.5)
#> [1] 1
```

The small samples sizes will result in averages with large standard errors and not useful for prediction.

For now, we will take the approach of creating strata of fathers with very similar heights. Specifically, we will round father heights to the nearest inch and assume that they are all 72 inches. If we do this, we end up with the following prediction for the son of a father that is 72 inches tall:

```
conditional_avg <- galton_heights %>%
  filter(round(father) == 72) %>%
  summarize(avg = mean(son)) %>% .$avg
conditional_avg
#> [1] 71.8
```

Note that a 72 inch father is taller than average. Specificaly, (72 - `mean(galton_heights$father)`/`sd(galton_heights$fath` = 1.139 standard deviations taller than the average father. Note that our prediction 71.836 is also taller than average, but only 0.54 standard deviations larger than the average son. The sons of 72 inch fathers haver *regressed* some to the average height. We note that the reduction in how much taller is about 0.5, which happens to be the correlation. As we will see. this is not a coincidence.

If we want to make a prediction of any height, not just 72, we could apply the same approach to each strata. Stratification followed by boxplots lets us see the distribution of each group:

```
galton_heights %>% mutate(father_strata = factor(round(father))) %>%
  ggplot(aes(father_strata, son)) +
  geom_boxplot() +
  geom_point()
```

Not suprisingly, the centers of the groups is increasing with height.

```
galton_heights %>%
  mutate(father = round(father)) %>%
  group_by(father) %>%
  summarize(son_conditional_avg = mean(son)) %>%
  ggplot(aes(father, son_conditional_avg)) +
  geom_point()
```

Furthermore, these centers appear to follow a linear relationship. Below we plot the averages of each group. If we take into account that these averages are random variable, with standard errors, it does appear that they follow a straight line:

```
r <- galton_heights %>% summarize(r = cor(father, son)) %>% .$r
galton_heights %>%
  mutate(father = round(father)) %>%
```

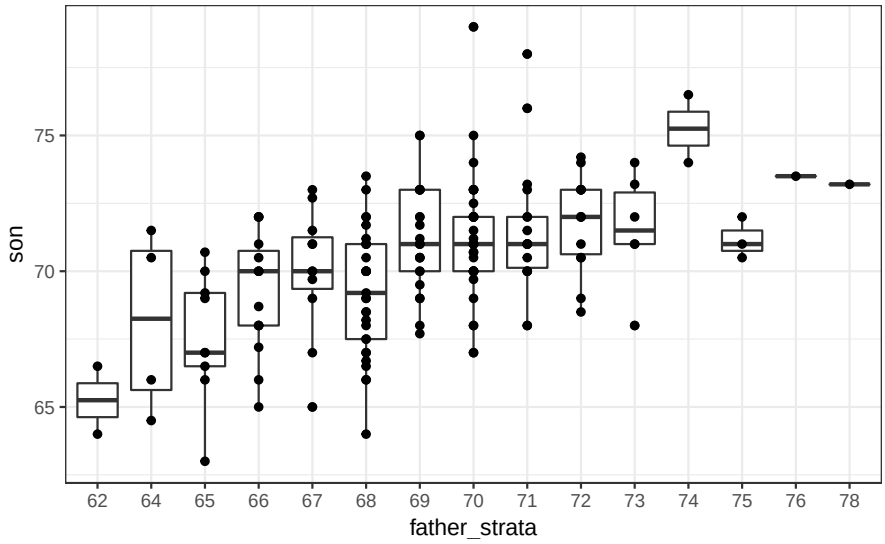


Figure 2: Boxplot of son heights stratified by father heights.

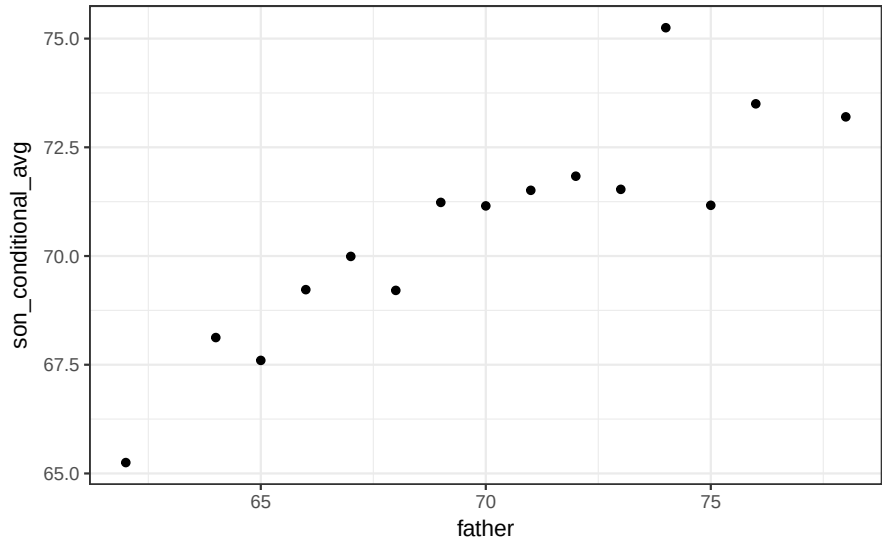
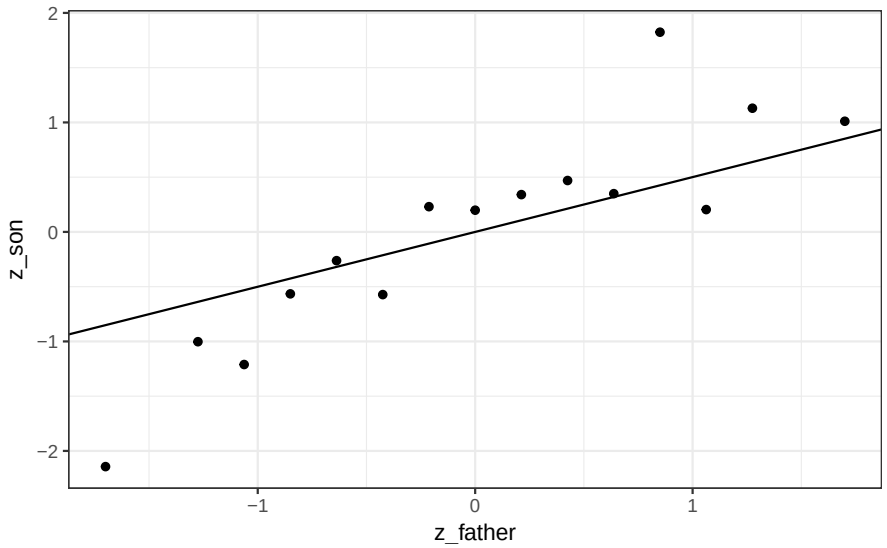


Figure 3: Boxplot of son heights stratified by father heights.

```
  group_by(father) %>%
  summarize(son = mean(son)) %>%
  mutate(z_father = scale(father), z_son = scale(son)) %>%
  ggplot(aes(z_father, z_son)) +
  geom_point() +
  geom_abline(intercept = 0, slope = r)
```



In the next section we explain the line these averages appear to follow is what we call the *regression line* and describe Galton's theoretical justification for using this line to estimate conditional means. Here we define it and compute the regression line for the data at hand.

1.4.1 The Rergression Line

If we are predicting a random variable Y knowing the value of another $X = x$ using a regression line then we predict that for every standard deviation, σ_X, that x increases above the average mu_X, Y increase ρ standard deviations σ_Y above the average μ_Y with ρ the correlation between X and Y. The formula for the regression is therefore

$$\left(\frac{Y - \mu_Y}{\sigma_Y}\right) = \rho\left(\frac{x - \mu_X}{\sigma_X}\right)$$

We can rewrite it like this

$$Y = \mu_Y + \rho\left(\frac{x - \mu_X}{\sigma_X}\right)\sigma_Y$$

If there is perfect correlation, we predict an increase that is the same number of SDs. If there is 0 correlation, then we don't use x at all for the prediction and simply predict the average μ_Y. For values between 0 and 1, the prediction is somewhere in between. If the correlation is negative we predict a reduction instead of an increase. For negative values, we predict in the opposite direction.

Note that if the correlation is positive and lower than 1 our prediction is closer, in standard units, to the average height than the value using to predict, x, is to the average x. This is why we call it *regression*: the son regresses to the average height. In fact the title of Galton's paper was:

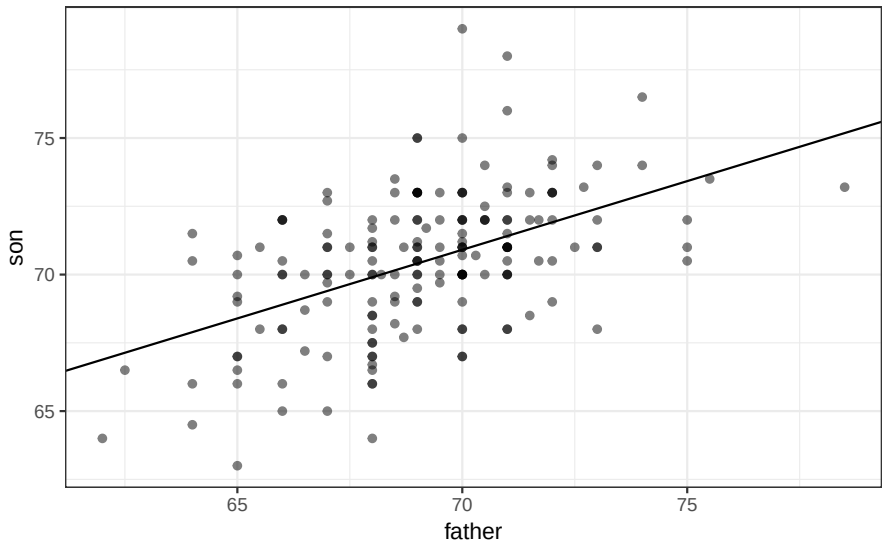
> Regression toward mediocrity in hereditary stature

To add regression lines to plots we will need the above formula in the form

$$y = b + mx \text{ with slope } m = \rho\frac{\sigma_y}{\sigma_x} \text{ and intercept } b = \mu_y - m\mu_x$$

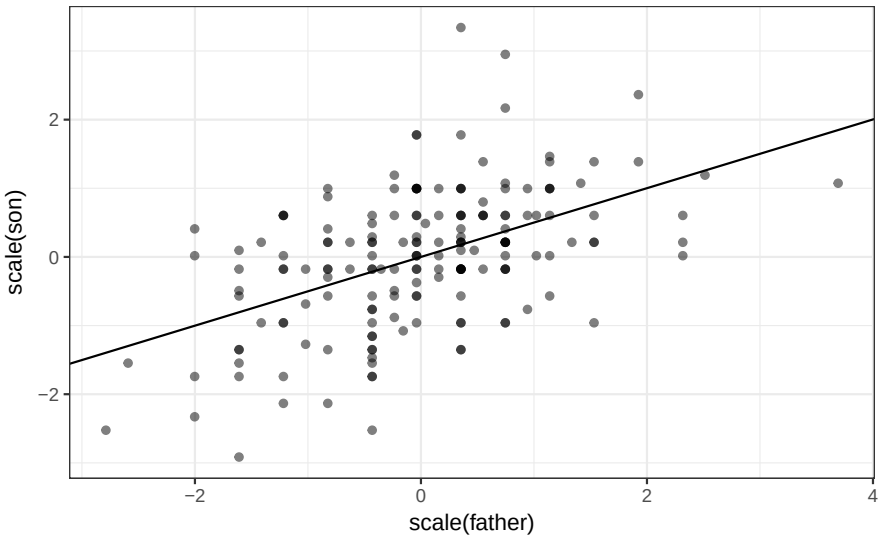
Here we add the regression line to the original data:

```
mu_x <- mean(galton_heights$father)
mu_y <- mean(galton_heights$son)
s_x <- sd(galton_heights$father)
s_y <- sd(galton_heights$son)
r <- cor(galton_heights$father, galton_heights$son)

m <-  r * s_y / s_x
b <- mu_y - m*mu_x

galton_heights %>%
  ggplot(aes(father, son)) +
  geom_point(alpha = 0.5) +
  geom_abline(intercept = b, slope = m )
```



Note that if we first standardize the variables, that is subtracact the average and divide by the standard deviation, then the regression line has intercept 0 and slope equal to the correlation ρ. Here is the same plot but using standard units:

```
galton_heights %>%
  ggplot(aes(scale(father), scale(son))) +
  geom_point(alpha = 0.5) +
  geom_abline(intercept = 0, slope = r)
```



1.4.2 Regression improves precision

Let's compare two approaches to prediction

1. Round father's heights to closes inch, stratify and then take the average
2. Compute the regression line and use it to predict

We use a Monte Carlo simulation sampling $N = 50$ famlies

```
B <- 1000
N <- 50

set.seed(1)
conditional_avg <- replicate(B, {
  dat <- sample_n(galton_heights, N)
  dat %>% filter(round(father) == 72) %>%
    summarize(avg = mean(son)) %>%
    .$avg
  })

regression_prediction <- replicate(B, {
  dat <- sample_n(galton_heights, N)
  mu_x <- mean(dat$father)
  mu_y <- mean(dat$son)
  s_x <- sd(dat$father)
  s_y <- sd(dat$son)
  r <- cor(dat$father, dat$son)

  mu_y + r*(72 - mu_x)/s_x*s_y
})
```

Although the expected value of these two random variables is about the same

```
mean(conditional_avg, na.rm = TRUE)
#> [1] 71.9
mean(regression_prediction)
#> [1] 71.9
```

The standard error for the regression prediction is substantially smaller

```
sd(conditional_avg, na.rm = TRUE)
#> [1] 0.865
sd(regression_prediction)
#> [1] 0.403
```

The regression line is therefore much more stable than the conditional mean. There is an intuitive reason for this. The conditional average is computed on a relatively small subset: the fathers that are about 72 inches tall. In fact, in some of the permutations we have no data, which is why we use `na.rm=TRUE`. The regression always uses all the data. So why not always use the regression for prediction? Because it is not always appropriate. For example, Anscombe provided cases for which the data does not have a linear relationship. So are we justified in using the regression line to predict? Galton answered this in the postive for height data.

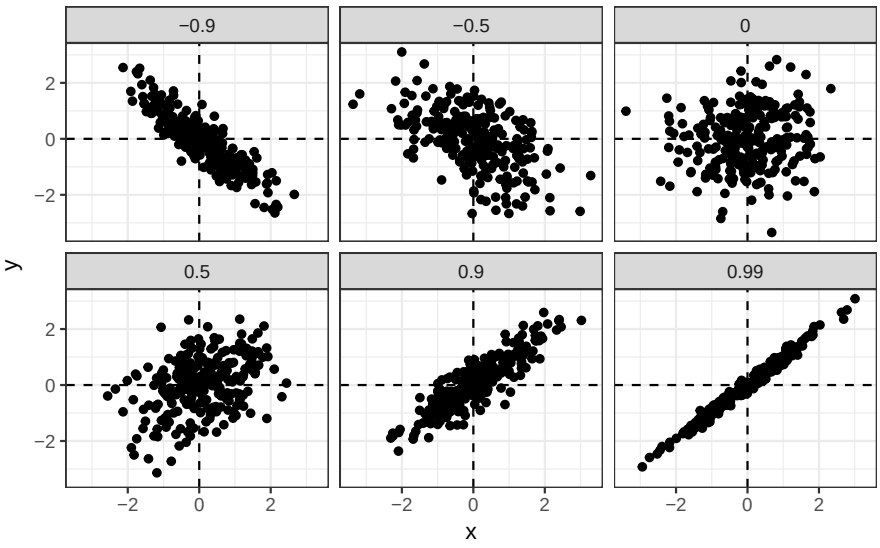
1.5 Bivariate Normal Distribution

Correlation and the regression line are widely used summary statistic but it is often misused or misinterpreted. Anscombe's examples provide toy examples of dataset in which summarizing with correlation would be a mistake. But there are many more real life examples.

The main way we motivate the use of correlation involves what is called the *bivariate normal distribution*.

When a pair of random variables are approximated by the bivariate normal distribution, scatter plots look like ovals. They can be thin (high correlation) or circle-shaped (no correlation). We saw some of these above:

```
n <- 250
cors <- c(-0.9,-0.5,0,0.5,0.9,0.99)
sim_data <- lapply(cors,function(r) MASS::mvrnorm(n,c(0,0), matrix(c(1,r,r,1),2,2)))
sim_data <- Reduce(rbind, sim_data)
sim_data <- cbind( rep(cors, each=n), sim_data)
colnames(sim_data) <- c("r","x","y")
as.data.frame(sim_data) %>% ggplot(aes(x,y)) +facet_wrap(~r) + geom_point() +geom_vline(xintercept = 0,l
```



A more technical way to define the bivariate normal distribution is the following: if X is a normally distributed random variable, Y is also a normally distributed random variable, and for any stratum of X, say $X = x$, Y is approximately normal in that stratum, then the pair is approximately bivariate normal.

When we fix $X = x$ in this way we then refer to the resulting distribution of the Ys in the strata as the *conditional distribution* of Y given $X = x$. We write it like notation like this:

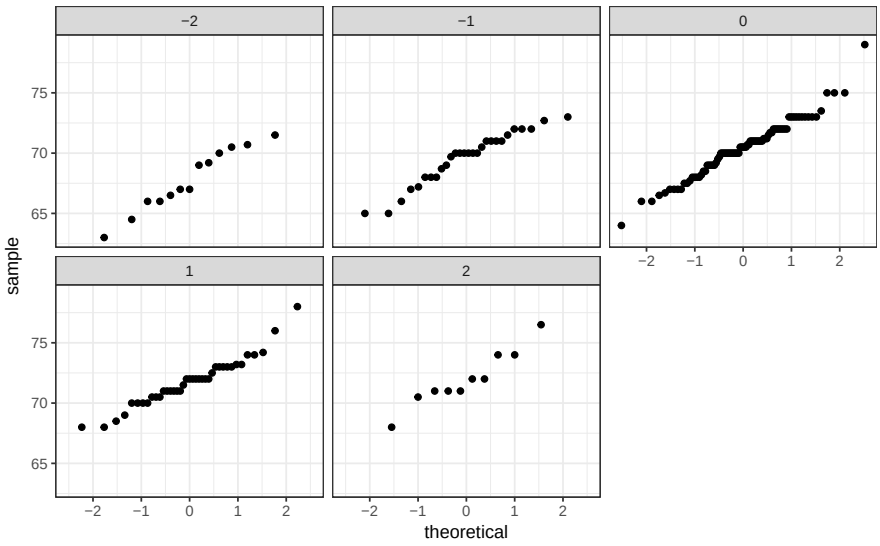


Figure 4: qqplots of son heights for four strata defined by father heights.

$f_{Y|X=x}$ is the notation for conditional distribution and $\mathrm{E}(Y \mid X = x)$ is the notation for conditional expected value.

If we think the height data is well approximated by the bivariate normal distribution then we should see the normal approximation hold for each strata. Here we stratify the son heights by the standardized father heights and see that the assumption appears to hold:

```
galton_heights %>%
  mutate(z_father = round((father - mean(father))/sd(father))) %>%
  filter(z_father %in% -2:2) %>%
  ggplot() +
  stat_qq(aes(sample=son)) +
  facet_wrap(~z_father)
```

Now we come back to defining correlation. Galton showed, using mathematical statistics, that when two variables follow a bivariate normal distribution, then for any given value of x, the expected value of the Y in pairs for which $X = x$ is:

$$\mathrm{E}(Y|X = x) = \mu_Y + \rho\frac{X - \mu_X}{\sigma_X}\sigma_Y$$

Note that this is the regression line: the line has slope

$$\rho\frac{\sigma_Y}{\sigma_X}$$

and intercept $\mu_y - m\mu_X$. It is equivalent to the line we showed earlier:

$$\frac{\mathrm{E}(Y \mid X = x) - \mu_Y}{\sigma_Y} = \rho\frac{x - \mu_X}{\sigma_X}$$

This implies that, if our data is approximately bivariate, the regression line gives the conditional probability. Therefore, we can obtain a much more stable estimate of the conditional expectation by fingind the regression line and using it to predict.

In summary, if our data is approximately bivariate, the the conditional expectation, the best prediction of Y given we know the value of X, is given by the regression line.

1.5.1 Variance explained

The theory also tells us that the standard deviation of the *conditional* distribution described above is:

$$\text{SD}(Y \mid X = x) = \sigma_Y \sqrt{1 - \rho^2}$$

To see why this is intuitive, note that without conditioning, $\text{SD}(Y) = \sigma_Y$, we are looking at the variability of all the sons. But once we condition, we are only looking at the variability of the sons with a tall, 72 inch, fahter. This group will all tend to be somewhat tall so the standard deviation is reduced.

Specifically it is reduced to $\sqrt{1 - \rho^2} = \sqrt{1 - 0.25} = 0.86$ of what it was originally. We could say that the fathers height "explains" 14% of the sons height variability.

The statement X explains such and such percent of the variablity is commonly used in academic papers. In this case this percent acutally refers to the variance (the SD squared). So if the data is bivariate normal the variance is reduced by $1 - \rho^2$ so we say that X explains $1 - (1 - \rho^2) = \rho^2$ (the correlation squared) of the variance.

But it is important to remember that the "variance explained" statement only makes sense when the data is approximated by a bivariate normal distribution.

1.5.2 Warning: There are two regression lines

We computed a regression line to predict the son's height from father's height. We used this calculations

```
mu_x <- mean(galton_heights$father)
mu_y <- mean(galton_heights$son)
s_x <- sd(galton_heights$father)
s_y <- sd(galton_heights$son)
r <- cor(galton_heights$father, galton_heights$son)
m_1 <-  r * s_y / s_x
b_1 <- mu_y - m_1*mu_x
```

which gives us the function $\text{E}(Y \mid X = x) = 35.712 + 0.503\ x$.

What if we want to predict the father's height based on the son's? It is important to know that this is not determined by computing the inverse function $x = \{\text{E}(Y \mid X = x) - 35.712\ \}/\ 0.503$.

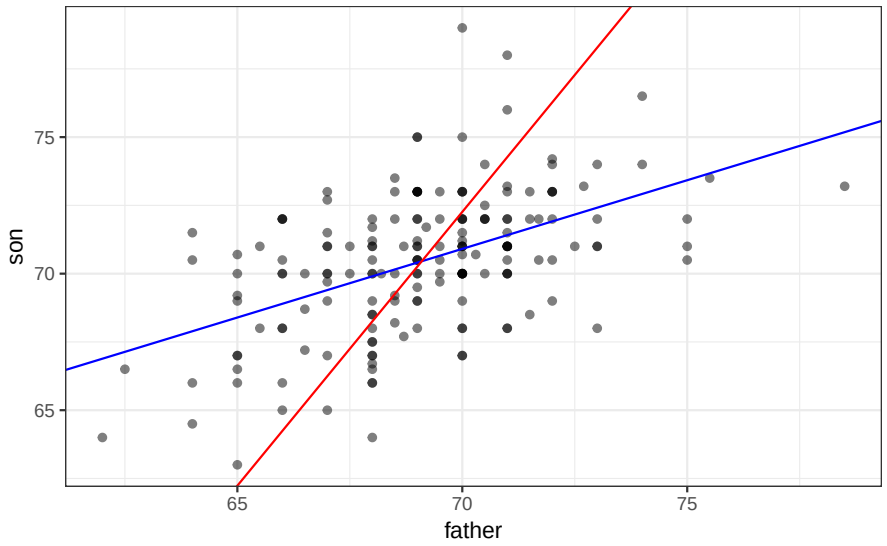
We need to compute $\text{E}(X \mid Y = y)$. Because the data is approximately bivariate normal, the theory described above tells us that this conditional expectation will follow a line with slope and intercept:

```
m_2 <-  r * s_x / s_y
b_2 <- mu_x - m_2*mu_y
```

So we get $\text{E}(X \mid Y = y) = 33.965 + 0.499\text{y}$. Note that again we see regression to the average: the prediction for the father is closer to the average father than the son heights y is to the son average.

Here is a plot showing the two regression lines:

```
galton_heights %>%
  ggplot(aes(father, son)) +
  geom_point(alpha = 0.5) +
  geom_abline(intercept = b_1, slope = m_1, col = "blue") +
  geom_abline(intercept = -b_2/m_2, slope = 1/m_2, col = "red")
```

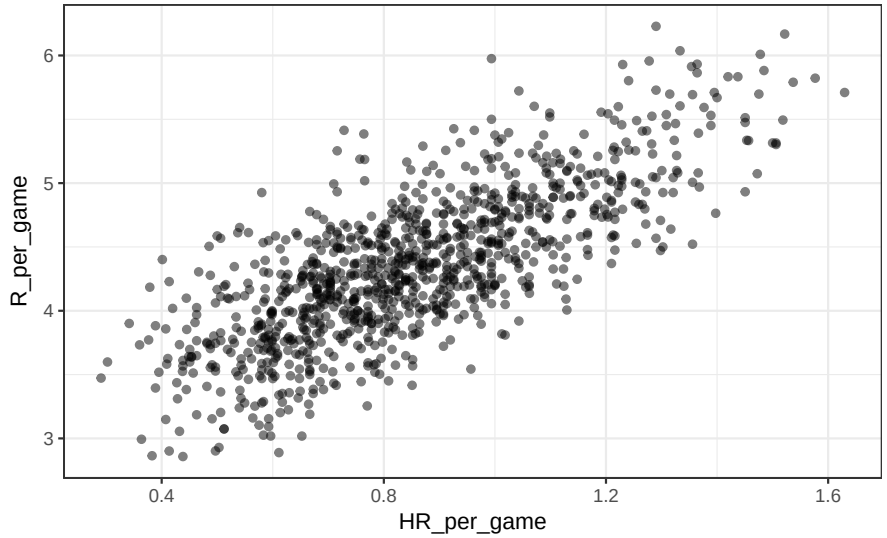


with blue for the predicting son heights with father heights and red for predicting father heights with son heights.

1.5.3 Baseball example

Now let's turn our attention back to the original problem: predicting runs. First we note that the Home Run and Run data appear to be bivariate normal:

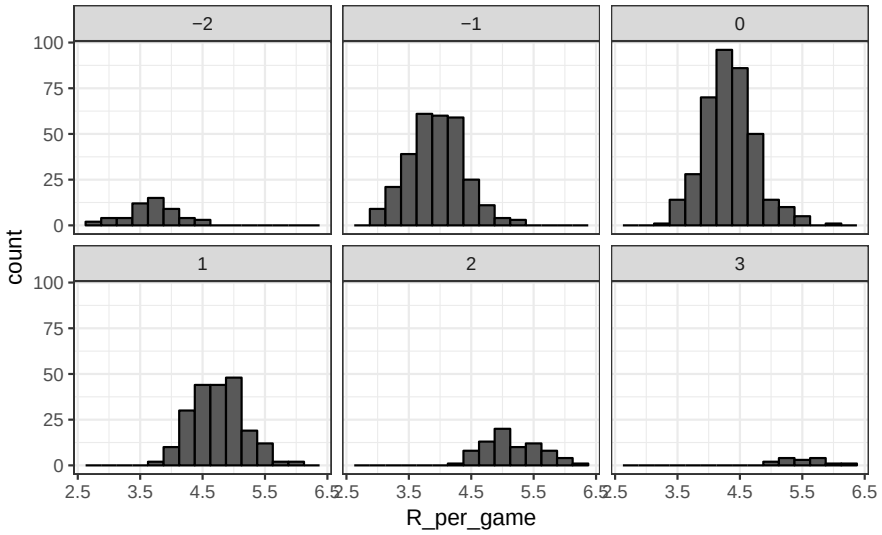
```
library(Lahman)
p <- Teams %>% filter(yearID %in% 1961:2001 ) %>%
  mutate(HR_per_game = HR/G, R_per_game = R/G) %>%
  ggplot(aes(HR_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
p
```



We can see that the histogramsof each strata confirm that the conditional distributions are normal:

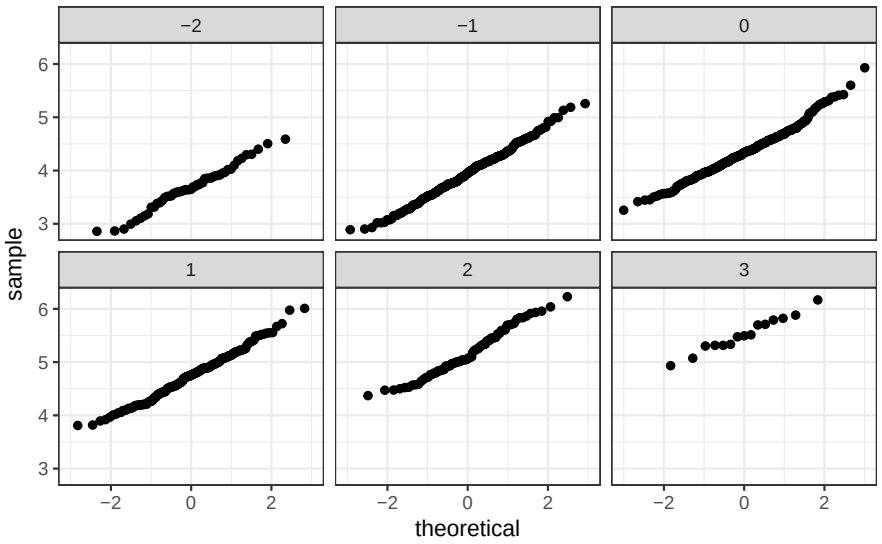
```
Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(z_HR = round((HR - mean(HR))/sd(HR)),
         R_per_game = R/G) %>%
```

```
  filter(z_HR %in% -2:3) %>%
  ggplot(aes(x=R_per_game)) +
  geom_histogram(binwidth = 0.25, color = "black") +
  facet_wrap(~z_HR)
```



The qq-plots confirm the that the normal approximation holds

```
Teams %>% filter(yearID %in% 1961:2001 ) %>%
  mutate(z_HR = round((HR - mean(HR))/sd(HR)),
         R_per_game = R/G) %>%
  filter(z_HR %in% -2:3) %>%
  ggplot() +
  stat_qq(aes(sample=R_per_game)) +
  facet_wrap(~z_HR)
```



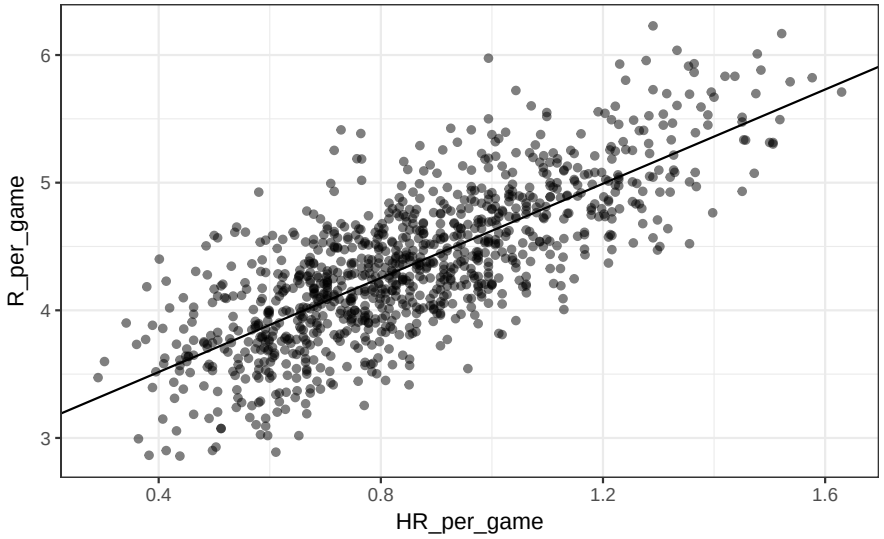
Now we are ready to use linear regression to predict the number of runs a team will score if we know how many home runs the team hits. All we need to do is compute the five summary statistics:

```
summary_stats <- Teams %>%
  filter(yearID %in% 1961:2001 ) %>%
  mutate(HR_per_game = HR/G, R_per_game = R/G) %>%
```

```
  summarize(avg_HR = mean(HR_per_game),
            s_HR = sd(HR_per_game),
            avg_R = mean(R_per_game),
            s_R = sd(R_per_game),
            r = cor(HR_per_game, R_per_game))
summary_stats
#>   avg_HR  s_HR avg_R   s_R     r
#> 1  0.855 0.243  4.36 0.589 0.762
```

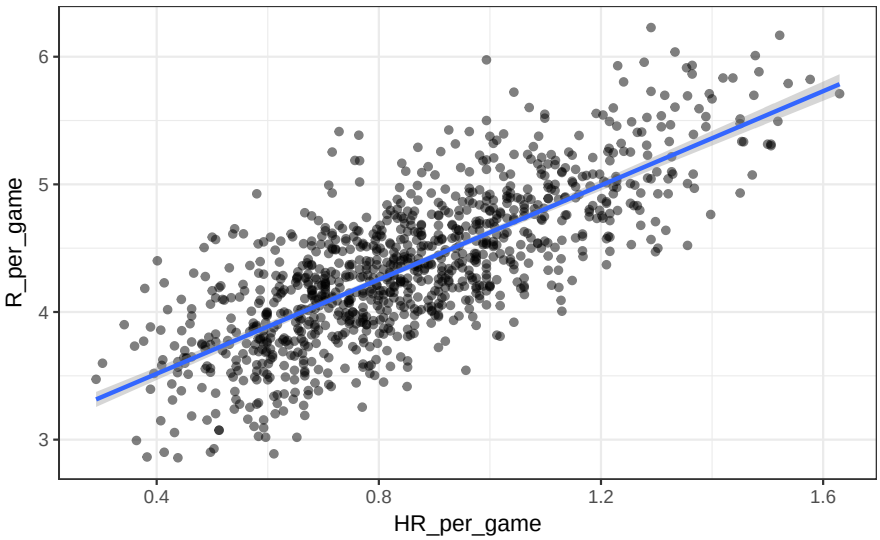
and use the formulas given above to create the regression lines:

```
reg_line <- summary_stats %>% summarize(slope = r*s_R/s_HR,
                                        intercept = avg_R - slope*avg_HR)
p + geom_abline(intercept = reg_line$intercept, slope = reg_line$slope)
```



Note that we will soon learn of R functions, such as `lm`, that make fitting regression lines much easier. Also note that the ggplot2 function `geom_smooth` computes and adds a regression line to plot along with confidence intervals, which we learn about later. We use the argument `method = "lm"` which stands for *linear model*, the title of the next section.

```
p + geom_smooth(method = "lm")
```



In the example above, the slope is 1.845. So this tells us that teams that hit 1 more HR per game than the average team score 1.845 more runs per game than the average team. Given that the most common final score is a differnce of a run, this can certainly lead to a large increase in wins. Not surprisingly, HR hitters are very expensive. Because we are working on a budget we will need to find some other way to increase wins. So let's move our attention to BB.

1.6 Confounding

If we find the regression line for predicting runs from bases on balls we a get slope of 0.735.

So does this mean that if we go and hire low salary players with many BB, that increase the number of walks per game by 2, our team will score 1.471 more runs per game?

We are again reminded that association is not causation. The data does provide strong evidence that a team with two more BB per game than the average team scores 1.471 runs per game. But this does not mean that BB are the cause.

If we do compute the regression line slope for singles we get 0.449, a lower value.

Note that a single gets you to first base just like a BB. Those that know about baseball, will tell you that with a single, runners on base have a better chance of scoring than with BB. So how can BB be more predictive of runs? The reason this happen is because of confounding. Note the correlation between HR, BB, and singles:

```
Teams %>%
  filter(yearID %in% 1961:2001 ) %>%
  mutate(Singles = (H-HR-X2B-X3B)/G, BB = BB/G, HR = HR/G) %>%
  summarize(cor(BB, HR), cor(Singles, HR), cor(BB,Singles))
#>   cor(BB, HR) cor(Singles, HR) cor(BB, Singles)
#> 1       0.404           -0.174          -0.0561
```

It turns out that pitchers, afraid of HRs, will sometimes avoid throwing strikes to HR hitters. As a result HR hitters tend to have more BB. Thus a team with many HR will also have more BB and as a result it may appear that BB cause Runs but it is actually the HR that cause the runs. We say that BB are *confounded* with HR. But could it be that BB still help? To find out we somehow have to adjust for the HR effect. Regression can help with this as well.

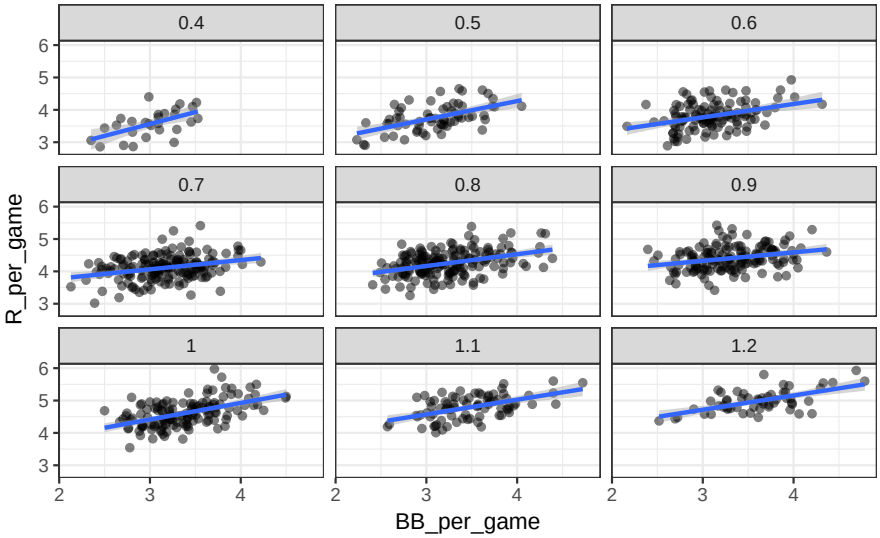
1.6.1 Stratification

A first approach is to keep HRs fixed at a certain value and then examine the relationship. As we did when we stratified fathers by rounding to the closest inch, here we can stratify HR per game to the closest tenth (we filter out the strata with few points):

```
dat <- Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(HR_strata = round(HR/G,1),
         BB_per_game = BB/G,
         R_per_game = R/G) %>%
  filter(HR_strata >= 0.4 & HR_strata <=1.2)
```

and then make a scatter plot for each strata:

```
dat %>%
  ggplot(aes(BB_per_game, R_per_game)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm") +
  facet_wrap(~HR_strata)
```

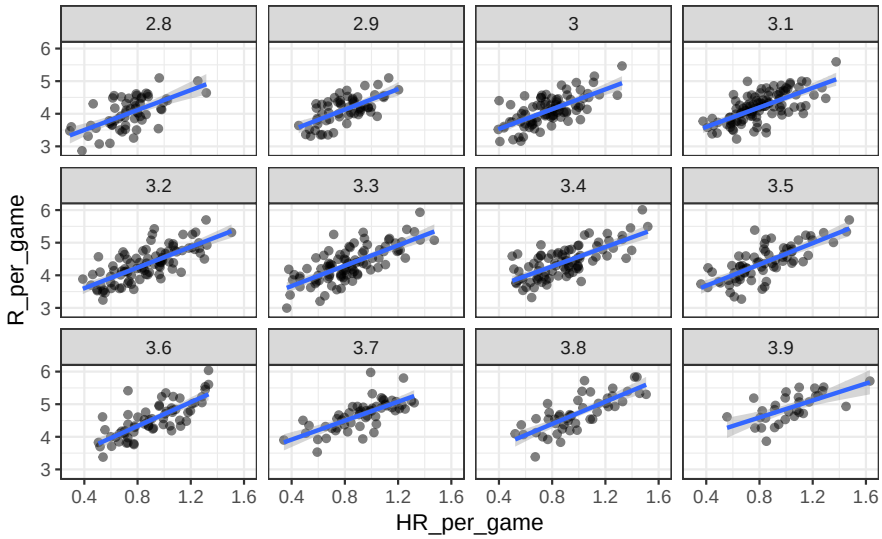


Remember that the regression slope for predicting runs with BB was `bb_slope`. Once we stratify by HR these slopes are substantially reduced:

```
dat %>%
  group_by(HR_strata) %>%
  summarize(slope = cor(BB_per_game, R_per_game)*sd(R_per_game)/sd(BB_per_game))
#> # A tibble: 9 x 2
#>   HR_strata slope
#>       <dbl> <dbl>
#> 1       0.4 0.734
#> 2       0.5 0.566
#> 3       0.6 0.412
#> 4       0.7 0.285
#> 5       0.8 0.365
#> 6       0.9 0.261
#> # ... with 3 more rows
```

These values are closer the the slope we obtained from singles 0.449, which is more consistent with our intuition since both singles and BB get us to first base, they should have about the same predictive power.

Now, although our understanding of the application tells us that HR cause BB but not the other way around, we can still check if stratifying by BB makes the effect of BB go down. We use the same code except we swap HR and BBs to get this plot:



In this case, the slopes:

```
#> # A tibble: 12 x 2
#>   BB_strata slope
#>       <dbl> <dbl>
#> 1       2.8  1.53
#> 2       2.9  1.57
#> 3       3.0  1.52
#> 4       3.1  1.49
#> 5       3.2  1.58
#> 6       3.3  1.56
#> # ... with 6 more rows
```

do not change much from the original 1.845.

Regardless, it seems that if we stratify by HR we have bivariate distributions for runs versus BB. Similarly if we stratify by BB, we have approximate bivariate normal distributions for HR versus runs.

1.7 Multivariate Regression

It is somewhat complex to be computing regression lines for each strata. We are essentially fitting models like this

$$\mathrm{E}[R \mid BB = x_1, HR = x_2] = \beta_0 + \beta_1(x_2)x_1 + \beta_2(x_1)x_2$$

with the slopes for x_1 changing for different values of x_2 and vice versa. Is there an easier approach?

Note that, if we take random variability into account, the slopes in the strata don't appear to change much. If these slopes are in fact the same, this implies that $\beta_1(x_2)$ and $\beta_2(x_1)$ are constants. Which in turn implies that the expectation of runs conditioned on HR and BB can be written like this:

$$\mathrm{E}[R \mid BB = x_1, HR = x_2] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

This model implies that if the number of HR is fixed at x_2 we observe a linear relationship between runs and BB with an intercept of $\beta_0 + \beta_2 x_2$. The model also implies that as the number of HR grows, the intercept grows is linear as well and determined by $\beta_1 x_1$.

In this analysis, referred to as *multivariate regression* we say that the BB slope β_1 is *adjusted* for the HR effect. If the model is correct then confounding has been accounted for. But how do we estimate β_1 and β_2 from the data? For this, we learn about linear models and least squares estimates.

1.8 Linear Models

Since Galton's original development, regression has become one of the most widely used tools in data science. One reason for this has to do with the fact that regression permits us to find relationships between two variables while adjusting for others, as we have just shown for BB, HR and runs in baseball. This has been particularly popular in fields where randomized experiments are hard to run, such as Economics and Epidemiology. When we are not able to randomly assign each individual to a treatment or control group, confounding is particularly prevalent. For example, consider estimating the effect of eating fast foods on life expectancy using data collected from a random sample of people in a jurisdiction. Fast food consumers are more likely to be smokers, drinkers, and have lower incomes. Therefore, a naive regression model may lead to an overestimate of a negative health effect. So how do we do adjust for confounding in practice?

We have described how if data is bivariate normal then the conditional expectations follow the regression line. That the conditional expectation is a line is not an extra assumption but rather a result derived. However, in practice it is common to explicitly write down a model that describes the relationship between two or more variables using a *linear model*.

We note that "Linear" here does not refer to lines exclusively, but rather to the fact that the conditional expectation is linear combinations of known quantities. In math we call a linear combination of variables, say x, y and z and combination that multiply them by a constant and then adds then with shifts permuted, for example $2 + 3x - 4y + 5z$. So $\beta_0 + \beta_1 x_1 + \beta_2 x_2$ is a linear combination of x_1 and x_2. The simplest linear model is a constant β_0, the second simplest is a line $\beta_0 + \beta_1 x$.

For Galton's data we would denote the N observe red father's heights with $x_1, \ldots, x_n$. Then we model the N son heights we are trying to predict with

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,\ i = 1, \ldots, N$$

with x_i is the father's height, which is fixed (not random) due to the conditioning, and Y_i is the random son's height that we want to predict. We further assume that ε_i are independent from each other, have expected value 0 and the standard deviation, call it σ, does not depend on i. We know the x_i, but to have a useful model for prediction we need β_0 and β_1. We estimate these from the data. Once we do we can predict son's heights for any father's height x.

Note that if we further assume that the ε is normally distributed, then this model is exactly the same one we derived earlier. A somewhat nuanced difference is that in the first approach we assumed the data was bivariate normal and the linear model was derived not assumed. In practice, linear models are just assumed without necessarily assuming normality: the distribution of the εs is not specified. But, nevertheless, if your data is bivariate normal, the above linear model holds. If your data is not bivariate normal, then you will need to have other ways of justifying the model.

One reason linear models are popular is that they are interpret able. In the case of Galton's data we can interpret the data like this: due to inherited genes, the son's height prediction grows by β_1 for each inch we increase the father height x. Because not all sons with fathers of height x are of equal height, we need the term ε, which explains the remaining variability. This remaining variability includes the mother's genetic effect, environmental factors, and other biological randomness.

Note that, given how we wrote the model above, the intercept β_0 is not very interpret able as it is the predicted height of a son with a father with no height. Due to regression to the mean the prediction will

usually be a bit larger than 0. To make the slope parameter more interpret able we can rewrite the model slightly as

$$Y_i = \beta_0 + \beta_1(x_i - \bar{x}) + \varepsilon_i,\ i = 1, \ldots, N$$

x_i to $x_i - \bar{x}$ in which case β_0 would be the height when $x_i = \bar{x}$ which is the son of an average father.

1.9 Least Squares Estimates (LSE)

For linear models to be useful we have to estimate the unknown βs. The standard approach in science is to find the values that minimize the distance of the fitted model to the data. The following is called the least squares (LS) equation and we will see it often in this chapter. For Galton's data we would write

$$RSS = \sum_{i=1}^{n} \{Y_i - (\beta_0 + \beta_1 x_i)\}^2$$

This quantity is called the residual sum of squares (RSS). Once we find the values that minimize the RSS, we will call the values the least squares estimates (LSE) and denote them with $\hat{\beta}_0$ and $\hat{\beta}_1$.

Let's write a function that computes the RSS for any pair of values β_0 and β_1:

```
rss <- function(beta0, beta1, data){
  resid <- galton_heights$son - (beta0+beta1*galton_heights$father)
  return(sum(resid^2))
}
```

So for any pair of values we get an RSS. Here is a plot of the RSS as a function of β_1 when we keep the β_0 fixed at 25.

```
beta1 = seq(0, 1, len=nrow(galton_heights))
results <- data.frame(beta1 = beta1,
                      rss = sapply(beta1, rss, beta0 = 25))
results %>% ggplot(aes(beta1, rss)) + geom_line() +
  geom_line(aes(beta1, rss), col=2)
```

We can see a clear minimum for β_1 at around 0.65. However, this minimum for β_1 is for when $\beta_0 = 25$. But we don which we don't know if it minimizes the pair $(25, 0.65)$ minimize the equation across all pairs.

Trial and error here is not going to work. Instead, we will use calculus: take the partial derivatives, set them to 0 and solve. Of course, if we have many parameters, these equations can get rather complex. But there are functions in R that do these calculations for us. We will learn these soon. To learn the mathematics behind this you can consult a book on linear models.

1.9.1 The `lm` function

In R we can obtain the least squares estimates using the the `lm` function. To fit the model

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

with Y_i the son's height and x_i the fathers height we write:

```
fit <- lm(son ~ father, data = galton_heights)
fit
#>
#> Call:
```

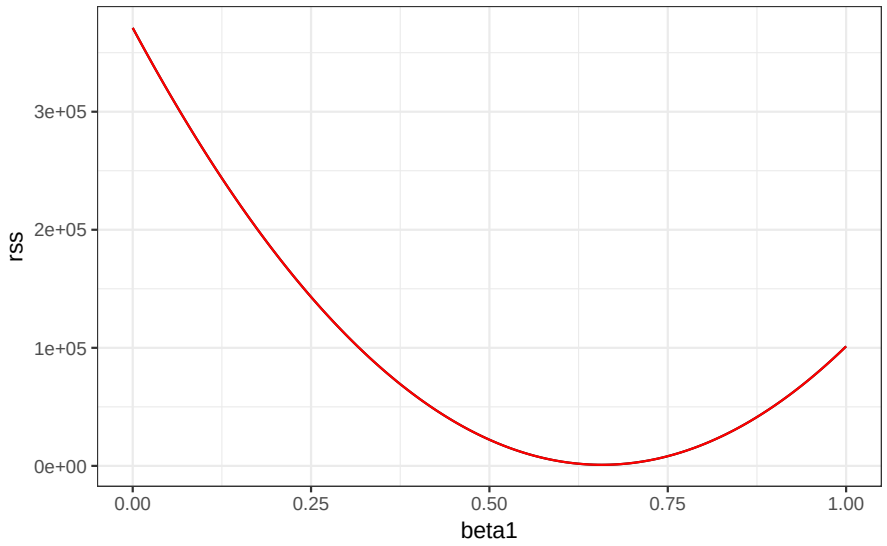


Figure 5: Residual sum of squares obtained for several values of the parameters.

```
#> lm(formula = son ~ father, data = galton_heights)
#>
#> Coefficients:
#> (Intercept)       father
#>      35.712        0.503
```

and obtain the lest squares estimates. The general way we use `lm` is by using the character `~` to let `lm` which is the variable we are predicting (left) and which we are using to predict (right). The intercept is added automatically to the model that will be fit.

The object `fit` includes more information about the fit. We can use the function `summary` to extract more of this information

```
summary(fit)
#>
#> Call:
#> lm(formula = son ~ father, data = galton_heights)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -5.902 -1.405  0.092  1.342  8.092
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  35.7125     4.5174    7.91  2.8e-13 ***
#> father        0.5028     0.0653    7.70  9.5e-13 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.22 on 177 degrees of freedom
#> Multiple R-squared:  0.251,  Adjusted R-squared:  0.246
#> F-statistic: 59.2 on 1 and 177 DF,  p-value: 9.47e-13
```

To understand some of the columns included in this summary we need to remember that the LSE are random variables. Mathematical statistics gives us some ideas of the distribution of these random variables

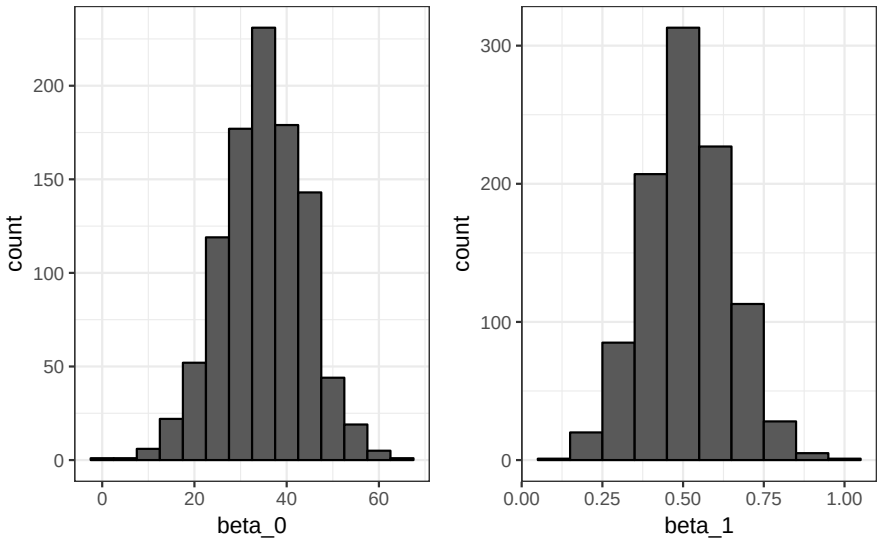
1.9.2 LSE are random variables

The LSE are derived from the data $Y_1, \ldots, Y_N$, which are random. This implies that our estimates are random variables. To see this we can run a Monte Carlo simulation in which we assume the son and father height data defines a population, take a random sample of size $N = 50$ and compute the regression slope coefficient for each one:

```
B <- 1000
N <- 50
lse <- replicate(B, {
  sample_n(galton_heights, N, replace = TRUE) %>%
   lm(son ~ father, data = .) %>% .$coef
})
lse <- data.frame(beta_0 = lse[1,], beta_1 = lse[2,])
```

We can see the variability of the estimates by plotting their distributions:

```
library(gridExtra)
#>
#> Attaching package: 'gridExtra'
#> The following object is masked from 'package:dplyr':
#>
#>     combine
p1 <- lse %>% ggplot(aes(beta_0)) + geom_histogram(binwidth = 5, color = "black")
p2 <- lse %>% ggplot(aes(beta_1)) + geom_histogram(binwidth = 0.1, color = "black")
grid.arrange(p1, p2, ncol = 2)
```



The reason these look normal is because the central limit theorem applies here as well: for large enough N the least squares estimates will be approximately normal with expected value β_0 and β_1 respectively. The standard errors are bit complicated to compute but mathematical theory does allow us to compute them and they are included in the summary provided by the `lm` function. Here it is for one of our simulated data sets:

```
 sample_n(galton_heights, N, replace = TRUE) %>%
   lm(son ~ father, data = .) %>% summary
#>
#> Call:
#> lm(formula = son ~ father, data = .)
#>
```

```
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -5.098 -1.094 -0.125  1.353  4.115
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   43.743      7.041    6.21  1.2e-07 ***
#> father         0.393      0.102    3.86  0.00034 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.87 on 48 degrees of freedom
#> Multiple R-squared:  0.237,  Adjusted R-squared:  0.221
#> F-statistic: 14.9 on 1 and 48 DF,  p-value: 0.000342
```

You can see that the standard errors estimates reported by the `summary` are close to the standard errors from the simulation:

```
lse %>% summarize(se_0 = sd(beta_0), se_1 = sd(beta_1))
#>   se_0  se_1
#> 1 8.92 0.129
```

The `summary` function also reports t-statistics (`t value`) and p-values (`Pr(>|t|)`). The t-statistic is not actually based on the central limit theorem but rather on the assumption that the εs follow a normal distribution. Under this assumption, mathematical theory tells is that the LSE divided by their standard error, $\hat{\beta}_0/\widehat{\text{SE}}(\hat{\beta}_0)$ and $\hat{\beta}_1/\widehat{\text{SE}}(\hat{\beta}_1)$ follow a t-distribution whit $N-p$ degrees of freedom with p the number of parameters in our model. In the case of height $p = 2$. The two p-value are testing the null hypothesis that $\beta_0 = 0$ and $\beta_1 = 0$ respectively. Note that, as we described previously, for large enough N the CLT works and the t-distribution becomes almost the same as the normal distribution. Also note that we can construct confidence intervals, but we will soon learn about broom, an add-on package that makes this easy.

We note here that although we do not show examples in this book, hypothesis testing with regression models is very commonly used in Epidemiology and Economics to make statements such as "the effect of A on B was statistically significant after adjusting for X, Y and Z". Note that several assumptions have to hold for these statements to hold.

1.9.2.1 Advanced note

Although interpretation is not straight-forward, it is also useful to know that the LSE can be strongly correlated.

```
lse %>% summarise(cor(beta_0, beta_1))
#>   cor(beta_0, beta_1)
#> 1              -0.999
```

However, the correlation depends on how the predictors are defined or transformed. Here we standardize father heights which changes x_i to $x_i - \bar{x}$

```
B <- 1000
N <- 50
lse <- replicate(B, {
  sample_n(galton_heights, N, replace = TRUE) %>%
  mutate(father = father - mean(father)) %>%
  lm(son ~ father, data = .) %>% .$coef
})
cor(lse[1,], lse[2,])
#> [1] -0.187
```

1.9.3 Predicted values are random variables

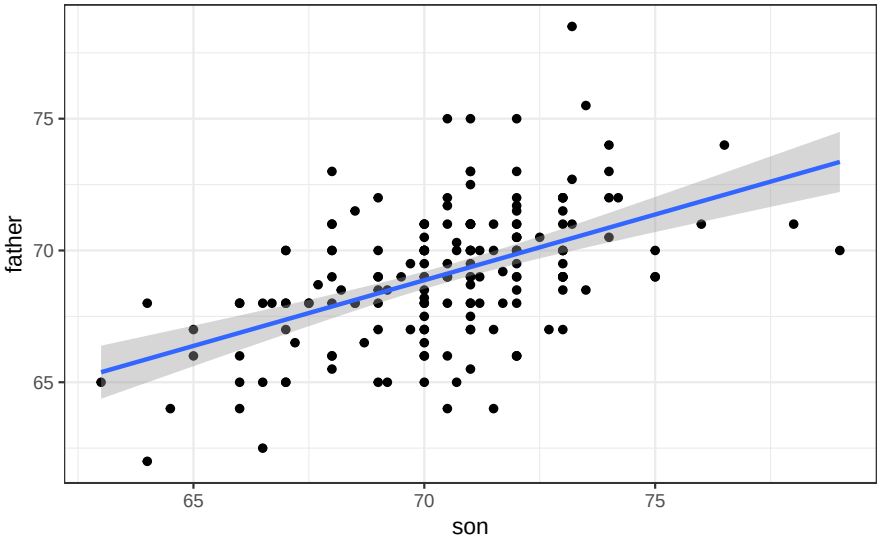
Once we fit our model, we can obtain prediction of Y by plugin the estimates into the regression model. For example, if the father's height is x, then our prediction $\hat{Y}$ for the son's height we will:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

When we plot $\hat{Y}$ versus x we see the regression line.

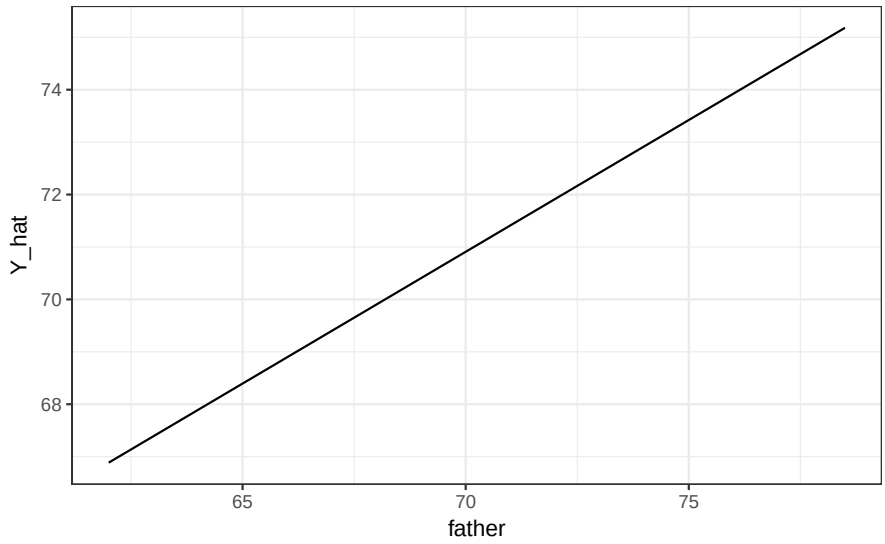
Note that the prediction $\hat{Y}$ is also a random variable and mathematical theory tells us what the standard errors are. If we assume the errors are normal our have a large enough sample size, we can use theory to construct confidence intervals as well. In fact, the ggplot2 layer `geom_smooth(method = "lm")` that we previously used plots $\hat{Y}$ and surrounds it by confidence intervals:

```
galton_heights %>% ggplot(aes(son, father)) +
  geom_point() +
  geom_smooth(method = "lm")
```



The R function `predict` takes an `lm` object as input and returns the prediction

```
galton_heights %>%
  mutate(Y_hat = predict(lm(son ~ father, data = .))) %>%
  ggplot(aes(father, Y_hat)) +
  geom_line()
```



If requested, the standard errors and other information from which we can construct confidence intervals:

```
fit <- galton_heights %>% lm(son ~ father, data = .)
Y_hat <- predict(fit, se.fit = TRUE)
names(Y_hat)
#> [1] "fit"            "se.fit"         "df"             "residual.scale"
```

1.10 Advanced dplyr: tibbles and do

Let's go back to the baseball. In a previous example we estimated regression lines to predict runs for BB in different HR strata. We first constructed a data frame similar to this:

```
dat <- Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(HR = round(HR/G, 1),
         BB = BB/G,
         R = R/G) %>%
  select(HR, BB, R) %>%
  filter(HR >= 0.4 & HR<=1.2)
```

Then, to compute the regression line in each strata, since we didn't know the `lm` function, we used the formula directly like this:

```
dat %>%
  group_by(HR) %>%
  summarize(slope = cor(BB,R)*sd(R)/sd(BB))
#> # A tibble: 9 x 2
#>      HR slope
#>   <dbl> <dbl>
#> 1   0.4 0.734
#> 2   0.5 0.566
#> 3   0.6 0.412
#> 4   0.7 0.285
#> 5   0.8 0.365
#> 6   0.9 0.261
#> # ... with 3 more rows
```

We argued that the slopes are similar and that the differences were perhaps due to random variation. To provide a more rigorous defense of the slopes being the same, which led to our multivariate model, we could

compute confidence intervals for each slope. We have not learned the formula for this, but the `lm` function provides enough information to construct them.

First, note that if we try to use the `lm` function to get the estimated slope like this:

```
dat %>%
  group_by(HR) %>%
  lm(R ~ BB, data = .) %>%
  .$coef
#> (Intercept)          BB
#>       2.199       0.638
```

we don't get what we want. The `lm` function ignored the `group_by`. This is expected because `lm` is not part of the tidyverse and does not know how to handle the outcome of `group_by`, a grouped *tibble*.

1.10.1 Tibbles

When `summarize` receives the output of `group_by` it somehow knows which rows of the tables go with which groups. Where is this information stored in the data.frame?

```
dat %>%  group_by(HR) %>% head()
#> # A tibble: 6 x 3
#> # Groups:   HR [5]
#>      HR    BB     R
#>   <dbl> <dbl> <dbl>
#> 1   0.9  3.56  4.24
#> 2   0.7  3.97  4.47
#> 3   0.8  3.37  4.69
#> 4   1.1  3.46  4.42
#> 5   1.0  2.75  4.61
#> 6   0.9  3.06  4.58
```

Note that there are no columns with this information. But, if you look closely at the output above you notice the line `A tibble: 6 x 3`. We can learn the class of the returned object using

```
dat %>%  group_by(HR) %>% class()
#> [1] "grouped_df" "tbl_df"     "tbl"        "data.frame"
```

The `tbl`, pronounced tibble, or `tbl_df`, is a special kind of data frame. We have seen them before because `tidyverse` functions such as `group_by` and `summarize` always return this type of data frame. The `group_by` function returns a special kind of `tbl`, the `grouped_df`. We will say more about these latter.

The manipulation verbs, `select`, `filter`, `mutate`, and `arrange` preserve the class of the input: if they receive a data frame they return a data frame. But tibbles are the default data frame in the tidyverse.

Tibbles are very similar to data frames. You can think of them as a modern version of data frames. Here we briefly described three important differences.

1.10.1.1 Tibbles diplsay better

The print method for tibbles is more readable than that of data frame. To see this compare these two outputs:

```
Teams
```

`Teams` is a data frame with many rows and columns. Nevertheless the output shows everything, wraps around and is hard to read. It is so bad we don't print it here, we let you print it on your screen. If you convert this data frame to a tibble data frame, the output is much more readable:

```
as_tibble(Teams)
#> # A tibble: 2,805 x 48
#>   yearID   lgID teamID franchID divID  Rank     G Ghome     W     L DivWin
#>    <int> <fctr> <fctr>   <fctr> <chr> <int> <int> <int> <int> <int>  <chr>
#> 1   1871     NA    BS1      BNA  <NA>     3    31    NA    20    10   <NA>
#> 2   1871     NA    CH1      CNA  <NA>     2    28    NA    19     9   <NA>
#> 3   1871     NA    CL1      CFC  <NA>     8    29    NA    10    19   <NA>
#> 4   1871     NA    FW1      KEK  <NA>     7    19    NA     7    12   <NA>
#> 5   1871     NA    NY2      NNA  <NA>     5    33    NA    16    17   <NA>
#> 6   1871     NA    PH1      PNA  <NA>     1    28    NA    21     7   <NA>
#> # ... with 2,799 more rows, and 37 more variables: WCWin <chr>,
#> #   LgWin <chr>, WSWin <chr>, R <int>, AB <int>, H <int>, X2B <int>,
#> #   X3B <int>, HR <int>, BB <int>, SO <int>, SB <int>, CS <int>,
#> #   HBP <int>, SF <int>, RA <int>, ER <int>, ERA <dbl>, CG <int>,
#> #   SHO <int>, SV <int>, IPouts <int>, HA <int>, HRA <int>, BBA <int>,
#> #   SOA <int>, E <int>, DP <int>, FP <dbl>, name <chr>, park <chr>,
#> #   attendance <int>, BPF <int>, PPF <int>, teamIDBR <chr>,
#> #   teamIDlahman45 <chr>, teamIDretro <chr>
```

Also note that the output adjusts to your window size.

1.10.1.2 Subsets of Tibbles are Tibbles

If you subset the columns of data frame you may get back an object that is not a data frame. For example

```
class(Teams[,20])
#> [1] "integer"
```

is not a data frame. With tibbles this does not happen:

```
class(as_tibble(Teams)[,20])
#> [1] "tbl_df"     "tbl"        "data.frame"
```

This is useful in the tidyverse since functions require data frames as input.

With tibbles, if you want to access the vector that defines a column, and not get back a data frame, you need to use the accessor `$`:

```
class(as_tibble(Teams)$HR)
#> [1] "integer"
```

A related feature is that tibbles will give you a warning if you try to access a column that does not exist. If we accidentally write `hr` instead of `HR` this

```
Teams$hr
#> NULL
```

returns a `NULL` with no warning, which can make it harder to debug. In contrast this

```
as_tibble(Teams)$hr
#> Warning: Unknown or uninitialised column: 'hr'.
#> NULL
```

gives you an informative warning.

1.10.1.3 Tibbles can have complex entries

While the columns or data frames need to be vectors of numbers, strings or Boolean, tibbles can have more complex objects such as lists or functions. Also note that we can create tibbles with `tibble` or `data_frame` functions:

```
tibble(id = c(1, 2, 3), func = c(mean, median, sd))
#> # A tibble: 3 x 2
#>      id   func
#>   <dbl> <list>
#> 1     1  <fun>
#> 2     2  <fun>
#> 3     3  <fun>
```

1.10.1.4 Tibbles can be grouped

The function `group_by` returns a special kind of tibble: a grouped tibble. This class stores information that lets you know which rows are in which groups. The tidyverse functions, in particular the `summarize` function, are aware of the group information. In our example above we saw that the `lm` function, which is not part of the tidyverse, does not know how to deal with grouped tibbles. The object is basically converted to a regular data frame and the function runs ignoring the groups. This is why we only get one pair of estimates:

```
dat %>%
  group_by(HR) %>%
  lm(R ~ BB, data = .)
#>
#> Call:
#> lm(formula = R ~ BB, data = .)
#>
#> Coefficients:
#> (Intercept)           BB
#>       2.199        0.638
```

To make these non-tidyverse functions work integrate with the tidyverse we will learn about `do`.

1.10.2 `do`

The tidyverse functions know how to interpret grouped tibbles. Furthermore, to facilitate stringing commands through the pipe `%>%` tidyverse function consistently return data frames, since this assures that the output of one is accepted as the input of another.

But most R functions do not recognize grouped tibbles nor do they return data frames. The `lm` function is an example. The `do` functions serves as a bridge between R functions such as `lm` and the tidyverse. The `do` function understands grouped tibbles and always returns a data frame.

So, let's try to use the `do` function to fit a regression line to each HR strata:

```
dat %>%
  group_by(HR) %>%
  do(fit = lm(R ~ BB, data = .))
#> Source: local data frame [9 x 2]
#> Groups: <by row>
#>
#> # A tibble: 9 x 2
#>      HR      fit
#> * <dbl>   <list>
#> 1   0.4 <S3: lm>
#> 2   0.5 <S3: lm>
```

```
#> 3   0.6 <S3: lm>
#> 4   0.7 <S3: lm>
#> 5   0.8 <S3: lm>
#> 6   0.9 <S3: lm>
#> # ... with 3 more rows
```

Notice that we did in fact fit a regression line to each strata. The `do` function will create a data frame with the first column being the strata value and a column named `fit` (we chose the name, but it can be anything). The column will contain the result of the `lm` call. Therefore, the returned tibble has a column with `lm` objects which is not very useful.

Also note that if we do not name a column then `do` will return the actual output of `lm`, not a data frame, and this will result in an error since `do` is expecting a data frame as output.

```
dat %>%
  group_by(HR) %>%
  do(lm(R ~ BB, data = .))
```

```
Error: Results 1, 2, 3, 4, 5, ... must be data frames, not lm
```

For a useful data frame to be constructed, the output of the function must be a data frame too. We could build a function that returns only what we want in the form of a data frame:

```
get_slope <- function(data){
  fit <- lm(R ~ BB, data = data)
  data.frame(slope = fit$coefficients[2],
             se = summary(fit)$coefficient[2,2])
}
```

And the use `do` **without** naming the output, since we are already getting a data frame:

```
dat %>%
  group_by(HR) %>%
  do(get_slope(.))
#> # A tibble: 9 x 3
#> # Groups:   HR [9]
#>      HR slope     se
#>   <dbl> <dbl>  <dbl>
#> 1   0.4 0.734 0.2076
#> 2   0.5 0.566 0.1102
#> 3   0.6 0.412 0.0974
#> 4   0.7 0.285 0.0705
#> 5   0.8 0.365 0.0652
#> 6   0.9 0.261 0.0754
#> # ... with 3 more rows
```

If we name the output then we get a column containing a data frames:

```
dat %>%
  group_by(HR) %>%
  do(slope = get_slope(.))
#> Source: local data frame [9 x 2]
#> Groups: <by row>
#>
#> # A tibble: 9 x 2
#>      HR               slope
#> * <dbl>              <list>
#> 1   0.4 <data.frame [1 x 2]>
```

```
#> 2   0.5 <data.frame [1 x 2]>
#> 3   0.6 <data.frame [1 x 2]>
#> 4   0.7 <data.frame [1 x 2]>
#> 5   0.8 <data.frame [1 x 2]>
#> 6   0.9 <data.frame [1 x 2]>
#> # ... with 3 more rows
```

which is not very useful.

We cover one last feature of `do`. If the data frame being returned has more than one row, these will be concatenated appropriately. Here is an example in which we return both estimated parameters:

```
get_lse <- function(data){
  fit <- lm(R ~ BB, data = data)
  data.frame(term = names(fit$coefficients),
    slope = fit$coefficients,
    se = summary(fit)$coefficient[,2])
}
dat %>%
  group_by(HR) %>%
  do(get_lse(.))
#> # A tibble: 18 x 4
#> # Groups:   HR [9]
#>      HR        term slope     se
#>   <dbl>      <fctr> <dbl>  <dbl>
#> 1   0.4 (Intercept) 1.359 0.6307
#> 2   0.4          BB 0.734 0.2076
#> 3   0.5 (Intercept) 2.007 0.3437
#> 4   0.5          BB 0.566 0.1102
#> 5   0.6 (Intercept) 2.533 0.3046
#> 6   0.6          BB 0.412 0.0974
#> # ... with 12 more rows
```

If you think this is all a bit too complicated, you are not alone. To simplify things, we introduce the broom package which was designed to facilitate the use of model fitting functions, such as `lm`, with the tidyverse.

1.11 Broom

Our original task was to provide an estimate and confidence interval for the slope estimates of each strata. The broom package will make this quite easy.

Broom has three main functions, all of which extract information from the object returned by `lm` and return it in a tidyverse friendly data frame. These functions are `tidy`, `glance` and `augment`. The `tidy` function returns estimates and related information as a data frame:

```
library(broom)
fit <- lm(R ~ BB, data = dat)
tidy(fit)
#>          term estimate std.error statistic  p.value
#> 1 (Intercept)    2.199    0.1135      19.4 1.06e-70
#> 2          BB    0.638    0.0344      18.5 1.37e-65
```

We can add other important summaries such as confidence intervals:

```
tidy(fit, conf.int = TRUE)
#>          term estimate std.error statistic  p.value conf.low conf.high
```

```
#> 1 (Intercept)    2.199    0.1135      19.4 1.06e-70     1.98     2.421
#> 2          BB    0.638    0.0344      18.5 1.37e-65     0.57     0.705
```

Because the outcome is a data frame we can immediately use it with `do` to and string together the commands that produce the table we are after:

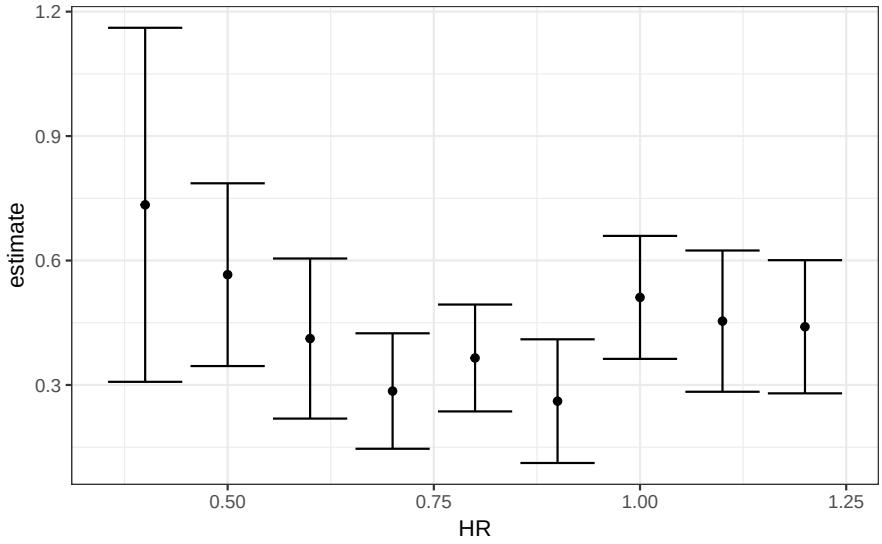
```
dat %>%
  group_by(HR) %>%
  do(tidy(lm(R ~ BB, data = .), conf.int = TRUE))
#> # A tibble: 18 x 8
#> # Groups:   HR [9]
#>      HR        term estimate std.error statistic  p.value conf.low
#>   <dbl>       <chr>    <dbl>     <dbl>     <dbl>    <dbl>    <dbl>
#> 1   0.4 (Intercept)    1.359    0.6307      2.16 4.05e-02   0.0631
#> 2   0.4          BB    0.734    0.2076      3.54 1.54e-03   0.3076
#> 3   0.5 (Intercept)    2.007    0.3437      5.84 2.07e-07   1.3202
#> 4   0.5          BB    0.566    0.1102      5.14 3.02e-06   0.3456
#> 5   0.6 (Intercept)    2.533    0.3046      8.32 2.43e-13   1.9293
#> 6   0.6          BB    0.412    0.0974      4.23 4.80e-05   0.2190
#> # ... with 12 more rows, and 1 more variables: conf.high <dbl>
```

Because a data frame is returned we can filter and select the rows and columns we want:

```
dat %>%
  group_by(HR) %>%
  do(tidy(lm(R ~ BB, data = .), conf.int = TRUE)) %>%
  filter(term == "BB") %>%
  select(HR, estimate, conf.low, conf.high)
#> # A tibble: 9 x 4
#> # Groups:   HR [9]
#>      HR estimate conf.low conf.high
#>   <dbl>    <dbl>    <dbl>     <dbl>
#> 1   0.4    0.734    0.308     1.161
#> 2   0.5    0.566    0.346     0.786
#> 3   0.6    0.412    0.219     0.605
#> 4   0.7    0.285    0.146     0.425
#> 5   0.8    0.365    0.236     0.494
#> 6   0.9    0.261    0.112     0.410
#> # ... with 3 more rows
```

A table like this can then be easily visualized with ggplot2:

```
dat %>%
  group_by(HR) %>%
  do(tidy(lm(R ~ BB, data = .), conf.int = TRUE)) %>%
  filter(term == "BB") %>%
  select(HR, estimate, conf.low, conf.high) %>%
  ggplot(aes(HR, y = estimate, ymin = conf.low, ymax = conf.high)) +
  geom_errorbar() +
  geom_point()
```



No we return to discussing our original task of determining if slopes changed. The plot we just made, using `do` and `broom`, shows that the confidence intervals overlap which provides a nice visual confirmation that our assumption that the slope does not change is safe.

The other functions provided by broom, `glance` and `augment` relate to model specific and observation specific outcomes respectively. Here we can see the model fit summaries `glance` returns:

```
glance(fit)
#>   r.squared adj.r.squared sigma statistic  p.value df logLik  AIC  BIC
#> 1     0.266         0.265 0.454       343 1.37e-65  2   -596 1199 1213
#>   deviance df.residual
#> 1      195         947
```

You can learn more about these summaries in any regression text book.

We will see an example of `augment` in the next section.

1.12 Building a better offensive metric for baseball

In trying to answer how well BB predict runs, data exploration led us to a model

$$\mathrm{E}[R \mid BB = x_1, HR = x_2] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Here the data is approximately normal and conditional distributions were also normal. Thus we are justified to pose a linear model:

$$Y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon_i$$

with Y_i runs per game, x_1 walks per game, and x_2. To use `lm` here we need to let it know we have two predictor variables. So we use the `+` symbol as follows:

```
fit <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB = BB/G, HR = HR/G,  R = R/G) %>%
  lm(R ~ BB + HR, data = .)
```

We can use `tidy` to see a nice summary:

```
tidy(fit, conf.int = TRUE)
#>          term estimate std.error statistic   p.value conf.low conf.high
#> 1 (Intercept)    1.744    0.0823      21.2  7.30e-83    1.583      1.91
#> 2          BB    0.387    0.0270      14.3  1.20e-42    0.334      0.44
#> 3          HR    1.561    0.0490      31.9 1.75e-155    1.465      1.66
```

When we fit the model with only one variable the estimated slopes were 0.735 and 1.845 for BB and HR respectively. Note that when fitting the multivariate model both go down, with the BB effect decreasing much more.

Now, if we want to construct a metric to pick players, we need to consider singles, doubles, and triples as well. Can we build a model that predicts runs based on all these outcomes?

Now we are going to take somewhat a "leap of faith" and assume that these five variables are jointly normal. This means that if we pick any one of them, and hold the other four fixed, the relationship with the outcome is linear and the slope does not depend on the four values held constant. If this is true then a linear model for our data is

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3} + \beta_4 x_{i,4} + \beta_5 x_{i,5} + \varepsilon_i$$

with x_1, x_2, x_3, x_4, x_5 representing BB, singles, doubles, triples, and HR respectively.

Using `lm` we can quickly find the LSE for the parameters using:

```
fit <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB = BB/G,
         singles = (H-X2B-X3B-HR)/G,
         doubles = X2B/G,
         triples =X3B/G,
         HR=HR/G,
         R=R/G) %>%
  lm(R ~ BB + singles + doubles + triples + HR, data = .)
```

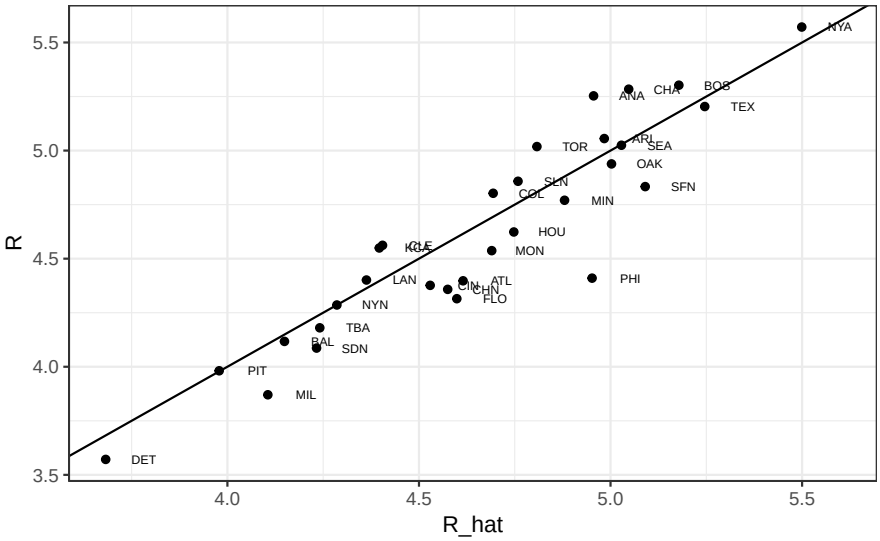
We can see the coefficients using `tidy`:

```
coefs <- tidy(fit, conf.int = TRUE)
coefs
#>          term estimate std.error statistic   p.value conf.low conf.high
#> 1 (Intercept)   -2.769    0.0862     -32.1 5.32e-157   -2.938    -2.600
#> 2          BB    0.371    0.0117      31.6 2.08e-153    0.348     0.394
#> 3     singles    0.519    0.0127      40.8 9.81e-217    0.494     0.544
#> 4     doubles    0.771    0.0226      34.1 8.93e-171    0.727     0.816
#> 5     triples    1.240    0.0768      16.1  2.24e-52    1.089     1.391
#> 6          HR    1.443    0.0244      59.3  0.00e+00    1.396     1.491
```

To see how well our metric actually predicts runs we can predict the number of runs for each team in 2002 using the function `predict` then make a plot:

```
Teams %>%
  filter(yearID %in% 2002) %>%
  mutate(BB = BB/G,
         singles = (H-X2B-X3B-HR)/G,
         doubles = X2B/G,
         triples =X3B/G,
         HR=HR/G,
         R=R/G)  %>%
```

```
  mutate(R_hat = predict(fit, newdata = .)) %>%
  ggplot(aes(R_hat, R, label = teamID)) +
  geom_point() +
  geom_text(nudge_x=0.1, cex = 2) +
  geom_abline()
```



Our model does quite a good job as demonstrated by the fact that points from the observed versus predicted plot fall close to the identity line.

So instead of using batting average or just number of HR as a measure of picking players, we can use our fitted model to form a more metric that relates more directly to run production. Specifically, to define a metric for player A, we imagine a team made up of players just like player A and use our fitted regression model to predict how many runs this team would produce. The formula would look like this: -2.769 + 0.371 $\times$ BB + 0.519 $\times$ singles + 0.771 $\times$ doubles + 1.24 $\times$ triples + 1.443 $\times$ HR +

To define a player specific metric we have a bit more work to do. A challenge here is that we derived the metric for teams, based on team-level summary statistics. For example, the HR value that is entered into the equation is HR per game for the entire team. If we compute the HR per game for a player it will be much lower as the total is accumulated by 9 batters. Furthermore, if a player only plays part of the game, and gets less opportunities than average, it is still considered a game played. For players, a rate that takes into account opportunities is the per-plate-appearance rate.

To make the per-game team rate comparable to the per-plate-appearance player rate comparable we compute the average number of team plate appearances per game:

```
pa_per_game <- Batting %>% filter(yearID == 2002) %>%
  group_by(teamID) %>%
  summarize(pa_per_game = sum(AB+BB)/max(G)) %>%
  .$pa_per_game %>%
  mean
```

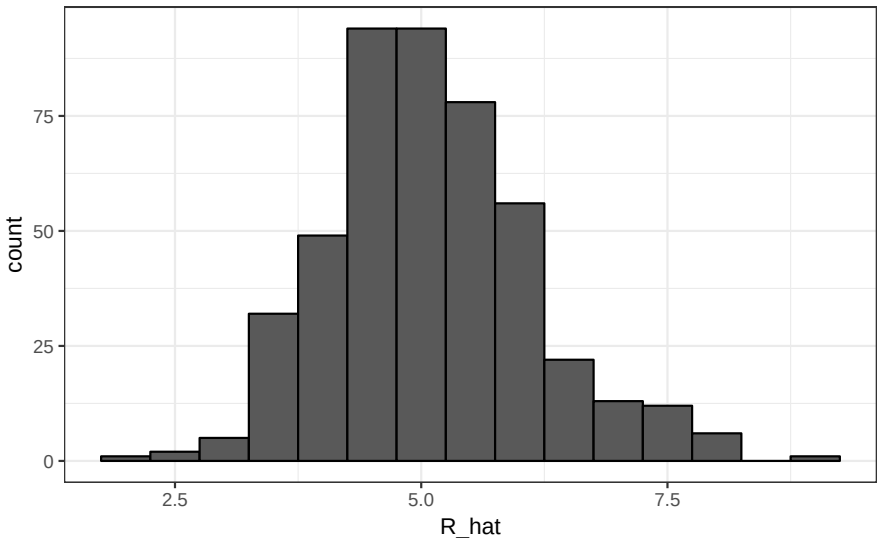
We compute the per-plate-appearance rates for players available in 2002 on data from 1999-2001. To avoid small sample artifacts we filter players with few plate appearances. Here is the entire calculation in one line:

```
players <- Batting %>% filter(yearID %in% 1999:2001) %>%
  group_by(playerID) %>%
  mutate(PA = BB + AB) %>%
  summarize(G = sum(PA)/pa_per_game,
    BB = sum(BB)/G,
    singles = sum(H-X2B-X3B-HR)/G,
```

```
    doubles = sum(X2B)/G,
    triples = sum(X3B)/G,
    HR = sum(HR)/G,
    AVG = sum(H)/sum(AB),
    PA = sum(PA)) %>%
  filter(PA >= 300) %>%
  select(-G) %>%
  mutate(R_hat = predict(fit, newdata = .))
```

The player specific predicted runs computed here can be interpreted as the number of runs we predict a team will score if all batters are exactly like that player. The distribution shows that there is wide variability across players:

```
players %>% ggplot(aes(R_hat)) +
  geom_histogram(binwidth = 0.5, color = "black")
```



To actually build the team we will need to know their salaries as well as their defensive position. For this we join the `players` data frame we just created with the player information data frame included in some of the other Lahman data tables. We will learn more about the join function in a later chapter.

Start by adding the 2002 salary of each player:

```
players <- Salaries %>%
  filter(yearID == 2002) %>%
  select(playerID, salary) %>%
  right_join(players, by="playerID")
```

Next we add their defensive position. This is a somewhat complicated task because players play more than one position each year. Here we pick the one the position player most played using the `top_n` function. To make sure we only pick one position in the case of ties pick the first row of the resulting data frame. We also remove the `OF` position which stands for outfielder, a generalization of three positions left field (LF), center filed (CF), and right field (RF). We also remove pitchers since they don't bat in the league the Athletics play.

```
players <- Fielding %>% filter(yearID == 2002) %>%
  filter(!POS %in% c("OF","P")) %>%
  group_by(playerID) %>%
  top_n(1, G) %>%
  filter(row_number(G) == 1) %>%
```

```
  ungroup() %>%
  select(playerID, POS) %>%
    right_join(players, by="playerID") %>%
  filter(!is.na(POS)  & !is.na(salary))
```

Finally we add their name and last name:

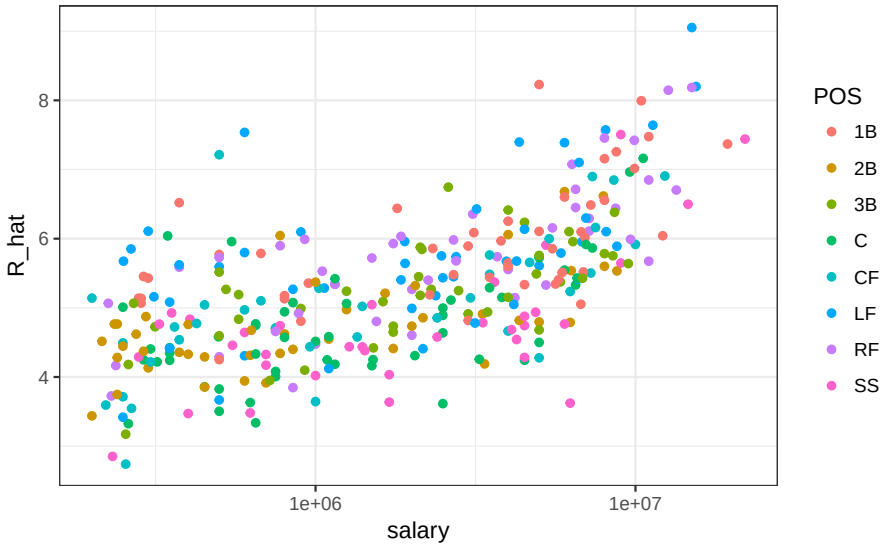
```
players <- Master %>%
  select(playerID, nameFirst, nameLast, debut) %>%
  right_join(players, by="playerID")
```

If you are a baseball fan you will recognize the top 10 players:

```
players %>% select(nameFirst, nameLast, POS, salary, R_hat) %>%
  arrange(desc(R_hat)) %>%
  top_n(10)
#> Selecting by R_hat
#>    nameFirst    nameLast POS   salary R_hat
#> 1      Barry       Bonds  LF 15000000  9.05
#> 2       Todd      Helton  1B  5000000  8.23
#> 3      Manny     Ramirez  LF 15462727  8.20
#> 4      Sammy        Sosa  RF 15000000  8.19
#> 5      Larry      Walker  RF 12666667  8.15
#> 6      Jason      Giambi  1B 10428571  7.99
#> 7    Chipper       Jones  LF 11333333  7.64
#> 8      Brian       Giles  LF  8063003  7.57
#> 9     Albert      Pujols  LF   600000  7.54
#> 10     Nomar Garciaparra  SS  9000000  7.51
```

Note the very high salaries for most players. In fact, we see that players with a higher metric have higher salaries:

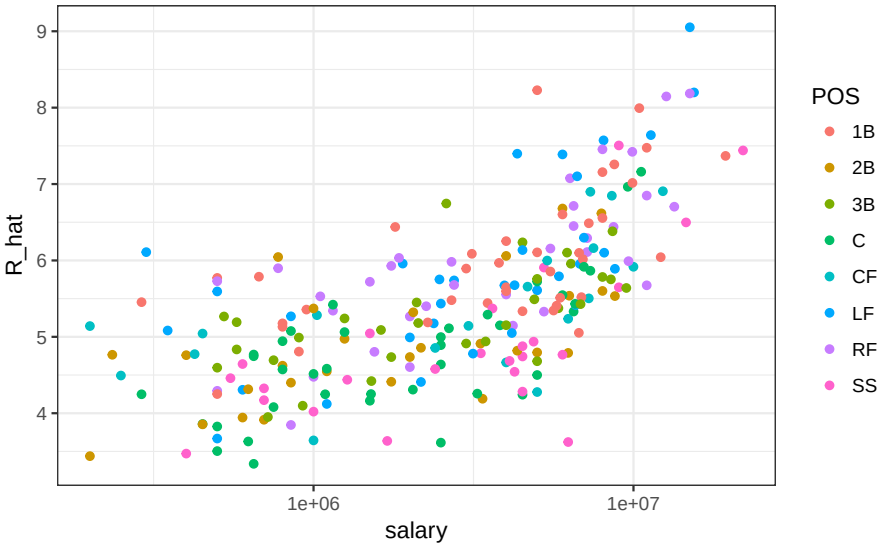
```
players %>% ggplot(aes(salary, R_hat, color = POS)) +
  geom_point() +
  scale_x_log10()
```



We do see some low cost players with very high metrics. These will be great for our team. Unfortunately, these are likely young players that have not yet been able to negotiate a salary and not available. For example, the lowest earner in our top 10 list, Albert Pujols. was a rookie in 2001. Here is the plot without

players that debuted before 1997:

```
players %>% filter(debut < 1998) %>%
 ggplot(aes(salary, R_hat, color = POS)) +
  geom_point() +
  scale_x_log10()
```



We can search for good deals by looking at players that produce many more runs that others with similar salaries. We can use this table to decide what players to pick and keep our total salary below the 40 million dollars Billy Bean had to work with. This can be done using what compute scientist called linear programming. This is not something we teach but we include the code anyway:

```
library(reshape2)
#>
#> Attaching package: 'reshape2'
#> The following object is masked from 'package:tidyr':
#>
#>     smiths
library(lpSolve)

players <- players %>% filter(debut <= 1997 & debut > 1988)
constraint_matrix <- acast(players, POS ~ playerID, fun.aggregate = length)
#> Using R_hat as value column: use value.var to override.
npos <- nrow(constraint_matrix)
constraint_matrix <- rbind(constraint_matrix, salary = players$salary)
constraint_dir <- c(rep("==", npos), "<=")
constraint_limit <- c(rep(1, npos), 50*10^6)
lp_solution <- lp("max", players$R_hat,
                  constraint_matrix, constraint_dir, constraint_limit,
                  all.int = TRUE)
```

This algorithm chooses these 9 players:

```
our_team <- players %>%
  filter(lp_solution$solution == 1) %>%
  arrange(desc(R_hat))
our_team %>% select(nameFirst, nameLast, POS, salary, R_hat)
#>   nameFirst    nameLast POS   salary R_hat
#> 1     Larry      Walker  RF 12666667  8.15
```

```
#> 2     Nomar Garciaparra  SS  9000000  7.51
#> 3      Luis    Gonzalez  LF  4333333  7.40
#> 4      Mike      Piazza   C 10571429  7.16
#> 5       Jim     Edmonds  CF  7333333  6.90
#> 6      Phil       Nevin  3B  2600000  6.75
#> 7      Greg    Colbrunn  1B  1800000  6.44
#> 8     Terry    Shumpert  2B   775000  6.04
```

We note that these players all have above average BB and HR rates while the same is not true for singles.

```
my_scale <- function(x) (x - median(x))/mad(x)
players %>% mutate(BB = my_scale(BB),
                   singles = my_scale(singles),
                   doubles = my_scale(doubles),
                   triples = my_scale(triples),
                   HR = my_scale(HR),
                   AVG = my_scale(AVG),
                   R_hat = my_scale(R_hat)) %>%
  filter(playerID %in% our_team$playerID) %>%
  select(nameFirst, nameLast, BB, singles, doubles, triples, HR, AVG, R_hat) %>%
  arrange(desc(R_hat))
#>   nameFirst    nameLast      BB singles doubles triples     HR   AVG R_hat
#> 1     Larry      Walker  1.0605  0.6554   0.922   1.562  1.566 2.835 2.904
#> 2     Nomar Garciaparra  0.0274  1.6371   3.118   0.336  0.625 3.197 2.245
#> 3      Luis    Gonzalez  0.7046  0.0000   1.413   0.537  1.355 1.829 2.133
#> 4      Mike      Piazza  0.3129 -0.0547  -0.242  -1.274  2.035 1.252 1.891
#> 5       Jim     Edmonds  1.8074 -1.1409   0.674  -0.674  1.264 0.579 1.621
#> 6      Phil       Nevin  0.4909 -0.6479   0.764  -1.098  1.548 0.728 1.463
#> 7      Greg    Colbrunn  0.2703  0.6546   0.784   0.585  0.475 1.375 1.148
#> 8     Terry    Shumpert -0.1576  0.1221   1.326   3.908 -0.123 0.859 0.744
```

1.13 On base plus slugging (OPS)

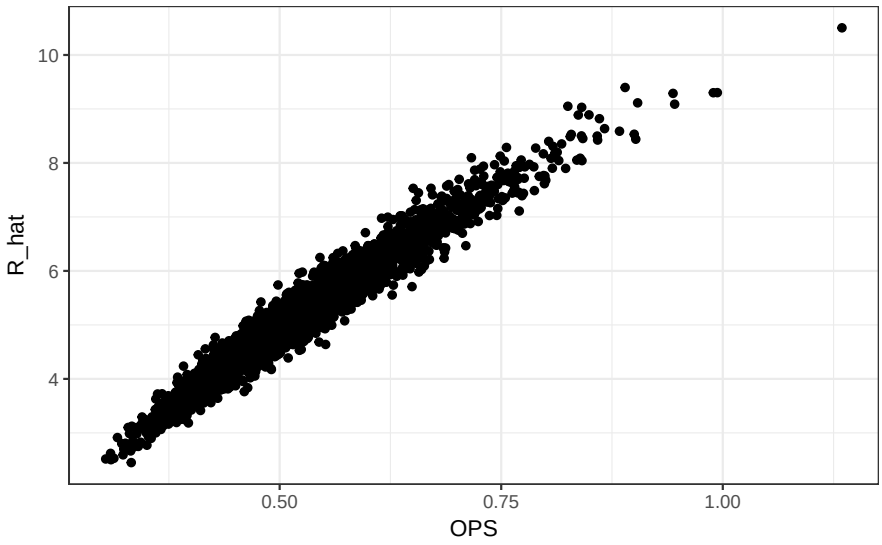
Since the 1980s, sabbermetricians have used a summary statistic different from batting average to evaluate players. They realized walks were important and that doubles, triples and HR should be weighted much more than singles and proposed the following metric:

$$\frac{\text{BB}}{\text{PA}} + \frac{\text{Singles} + 2\text{Doubles} + 3\text{Triples} + 4\text{HR}}{\text{AB}}$$

The called this on-base-percentage plus slugging percentage (OPS). Although the sabbermetricians probably did not use regression, this metric is impressively close to what one gets with regression:

```
Batting %>%
  filter(yearID %in% 1990:2001) %>%
  group_by(playerID, yearID) %>%
  filter(BB + AB >= 300) %>%
  mutate(PA = BB + AB,
         singles = (H-X2B-X3B-HR),
         OPS = BB / PA +
           (singles + 2*X2B + 3*X3B + 4*HR)/AB,
         G = PA/pa_per_game,
         BB = BB/G,
         singles = singles/G,
```

```
        doubles = X2B/G,
        triples = X3B/G,
        HR = HR/G) %>%
  ungroup() %>%
  mutate(R_hat = predict(fit, newdata = .)) %>%
  ungroup %>%
  ggplot(aes(OPS, R_hat)) +
  geom_point()
```



1.14 Regression Fallacy

Wikipedia defines the *sophomore slump* as

> A sophomore slump or sophomore jinx or sophomore jitters refers to an instance in which a second, or sophomore, effort fails to live up to the standards of the first effort. It is commonly used to refer to the apathy of students (second year of high school, college or university), the performance of athletes (second season of play), singers/bands (second album),television shows (second seasons) and movies (sequels/prequels).

In Major League Baseball the rookie of the year (ROY) award is given to the first year player that is judged to have performed the best. The *sophmore slump* phrase is used to describe the observation that ROY award winners don't do as well during their second year. Note for example that in this recent Fox Sports article asks "Will MLB's tremendous rookie class of 2015 suffer a sophomore slump?".

Does the data confirm the existence of a sophomore slump? Let's take a look Examining the data for batting average we see that this observation holds true.

The data is available in the Lahman library but we have to do some work to create a table with the statistics for all the ROY. First we create a table with player ID, their names, and their most played position.

```
library(Lahman)
playerInfo <- Fielding %>%
  group_by(playerID) %>%
  arrange(desc(G)) %>%
  slice(1) %>%
  ungroup %>%
  left_join(Master, by="playerID") %>%
  select(playerID, nameFirst, nameLast, POS)
```

Now we will create a table with only the ROY award winners and add their batting statistics. We filter out out pitchers, since pitchers are not given awards for batting and we are going to focus on offense. Specifically we will focus on batting average since it is the summary that most pundits talk about when discussing the sophomore slump:

```
ROY <- AwardsPlayers %>%
  filter(awardID == "Rookie of the Year") %>%
  left_join(playerInfo, by="playerID") %>%
  rename(rookie_year = yearID) %>%
  right_join(Batting, by="playerID") %>%
  mutate(AVG = H/AB) %>%
  filter(POS != "P")
```

Now we will keep only the rookie and sophomore seasons and remove players that did not play sophomore seasons

```
ROY <- ROY %>%
  filter(yearID == rookie_year | yearID == rookie_year+1) %>%
  group_by(playerID) %>%
  mutate(rookie = ifelse(yearID == min(yearID), "rookie", "sophomore")) %>%
  filter(n() == 2) %>%
  ungroup %>%
  select(playerID, rookie_year, rookie, nameFirst, nameLast, AVG)
```

Finally, we will use the spread function to have one column for the rookie and sophomore years batting averages:

```
ROY <- ROY %>%  spread(rookie, AVG) %>% arrange(desc(rookie))
```

We can see the top performs in their first year

```
ROY
#> # A tibble: 97 x 6
#>    playerID rookie_year nameFirst nameLast rookie sophomore
#>       <chr>       <int>     <chr>    <chr>  <dbl>     <dbl>
#> 1 mccovwi01        1959    Willie  McCovey  0.354     0.238
#> 2 suzukic01        2001    Ichiro   Suzuki  0.350     0.321
#> 3 bumbral01        1973        Al   Bumbry  0.337     0.233
#> 4  lynnfr01        1975      Fred     Lynn  0.331     0.314
#> 5 pujolal01        2001    Albert   Pujols  0.329     0.314
#> 6 troutmi01        2012      Mike    Trout  0.326     0.323
#> # ... with 91 more rows
```

and just by eyeballing we see the sophomore slump. In fact the proportion of players that have a lower batting average their sophomore year is

```
mean(ROY$sophomore - ROY$rookie <= 0)
#> [1] 0.68
```

So is it "jitters" or "jinx"? To answer this question let's turn our attention to all players that played the 2013 and 2014 seasons and batted more than 130 times (minimum to win Rookie of the Year). We perform a similar operations as we did above

```
two_years <- Batting %>%
  filter(yearID %in% 2013:2014) %>%
  group_by(playerID, yearID) %>%
  filter(sum(AB) >= 130) %>%
  summarize(AVG = sum(H)/sum(AB)) %>%
  ungroup %>%
```

```
  spread(yearID, AVG) %>%
  filter(!is.na(`2013`) & !is.na(`2014`)) %>%
  left_join(playerInfo, by="playerID") %>%
  filter(POS!="P") %>%
  select(-POS) %>%
  arrange(desc(`2013`)) %>%
  select(-playerID)
```

Note that the same pattern arises when we look at the top performers: batting averages go down for the most of the top performers.

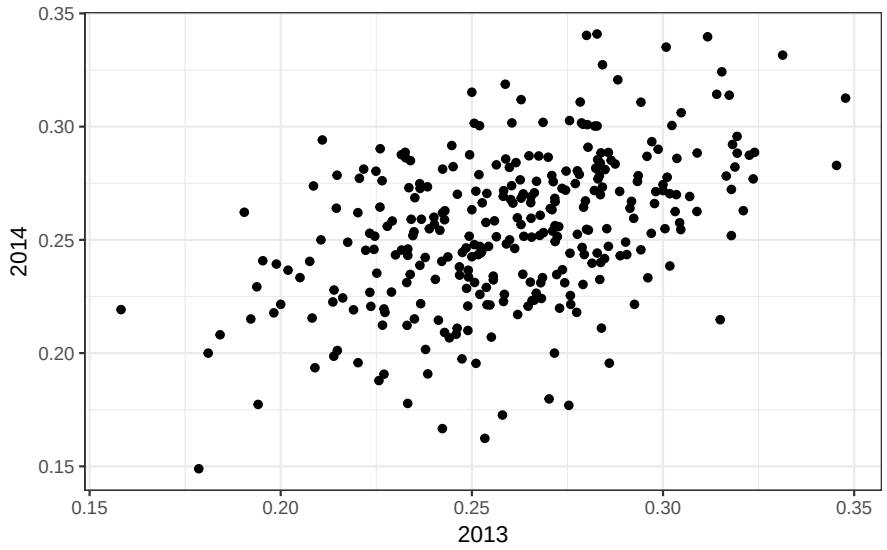
```
two_years
#> # A tibble: 312 x 4
#>   `2013` `2014` nameFirst nameLast
#>    <dbl>  <dbl>     <chr>    <chr>
#> 1  0.348  0.313    Miguel  Cabrera
#> 2  0.345  0.283    Hanley  Ramirez
#> 3  0.331  0.332   Michael  Cuddyer
#> 4  0.324  0.289   Scooter  Gennett
#> 5  0.324  0.277       Joe    Mauer
#> 6  0.323  0.287      Mike    Trout
#> # ... with 306 more rows
```

But these are not rookies! Also note what happens to the worst performers of 2013:

```
arrange(two_years, `2013`)
#> # A tibble: 312 x 4
#>   `2013` `2014` nameFirst nameLast
#>    <dbl>  <dbl>     <chr>    <chr>
#> 1  0.158  0.219     Danny Espinosa
#> 2  0.179  0.149       Dan    Uggla
#> 3  0.181  0.200      Jeff   Mathis
#> 4  0.184  0.208    Melvin    Upton
#> 5  0.190  0.262      Adam  Rosales
#> 6  0.192  0.215     Aaron    Hicks
#> # ... with 306 more rows
```

There batting averages go up! Is this some sort of reverse sophomore slump? It is not. There is no such thing as the sophomore slump. This is all explained with a simple statistical fact: the correlation for performance in two separate years is high, but not perfect:

```
two_years %>% ggplot(aes(`2013`, `2014`)) +
  geom_point()
```



The correlation is

```
summarize(two_years, cor(`2013`,`2014`))
#> # A tibble: 1 x 1
#>   `cor(\`2013\`, \`2014\`)`
#>                       <dbl>
#> 1                      0.46
```

The data look very much like a bivariate normal distribution which means we predict 2014 batting average Y for any given player that had 2013 batting average X with:

$$\frac{Y - .255}{.032} = 0.46 \left(\frac{X - .261}{.023} \right)$$

Because the correlation is not perfect regression tells us that on average, expect high performs from 2013 will to do a bit worse in 2014. It's not a jinx, it's just due to the chance. The ROY are selected from the top values of X so it is excepted that Y will regress to the mean.

1.15 Measurement error models

Up to now, all our linear regression examples have been applied to two or more random variables. We assume the pairs are bivariate normal and used this to motivates a linear model. This approach cover most real life examples of linear regression. The other major application comes from measurement errors models. In these application, it is common to have a non-random covariate, such as time, and randomness is introduced from measurement error rather than sampling or natural variability.

To understand these models, imagine you are Galileo in the 16th century trying to describe the velocity of a falling object. An assistant climbs the Tower of Pisa and drops a ball, while several other assistants record the position at different times. Let's simulate some data using the equations we know today and adding some measurement error:

```
falling_object <- rfalling_object()
```

The assistants hand the data to Galileo and this is what he sees:

```
falling_object %>%
  ggplot(aes(time, observed_distance)) +
  geom_point() +
```

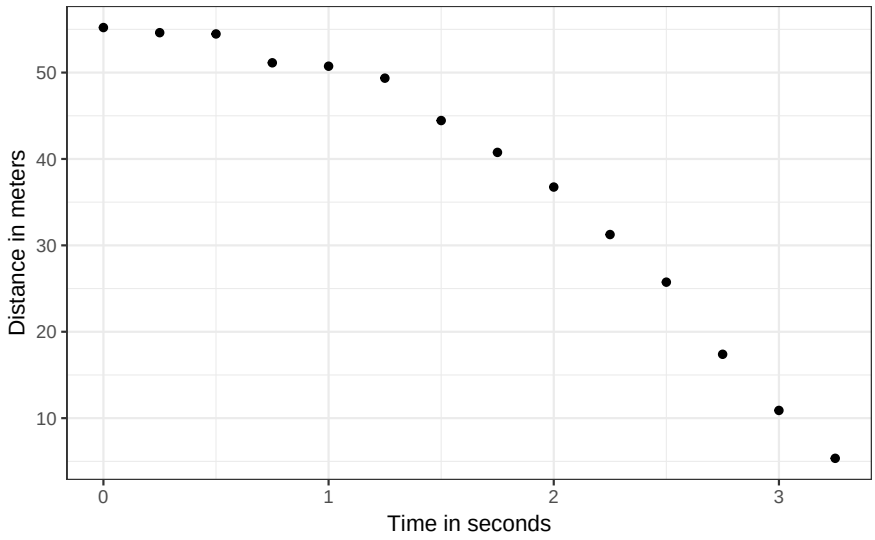


Figure 6: Simulated data for distance travelled versus time of falling object measured with error.

```
  ylab("Distance in meters") +
  xlab("Time in seconds")
```

Galileo does not know the exact equation, but by looking at the plot above he deduces that the position should follow a parabola, which we can write like this

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$

The data does not fall exactly on a parabola. Galileo knows this is due to measurement error. His helpers make mistakes when measuring the distance. To account for this he models the data with:

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i, i = 1, \ldots, n$$

With Y_i representing distance in meters, x_i representing time in seconds, and ε accounting for measurement error. The measurement error is assumed to be random, independent from each other, and having the same distribution for each i. We also assume that there is no bias which means the expected value $\mathrm{E}[\varepsilon] = 0$.

Note that this is a linear model because it is a linear combination of known quantities, x and x^2 are known, and unknown parameters (the β s). Unlike our previous examples x are fixed quantities, we are not conditioning.

To pose a new physical theory and start making predictions about other falling objects Galileo needs actual numbers, rather than unknown parameters. The LSE seem like a reasonable approach. How do we find the LSE?

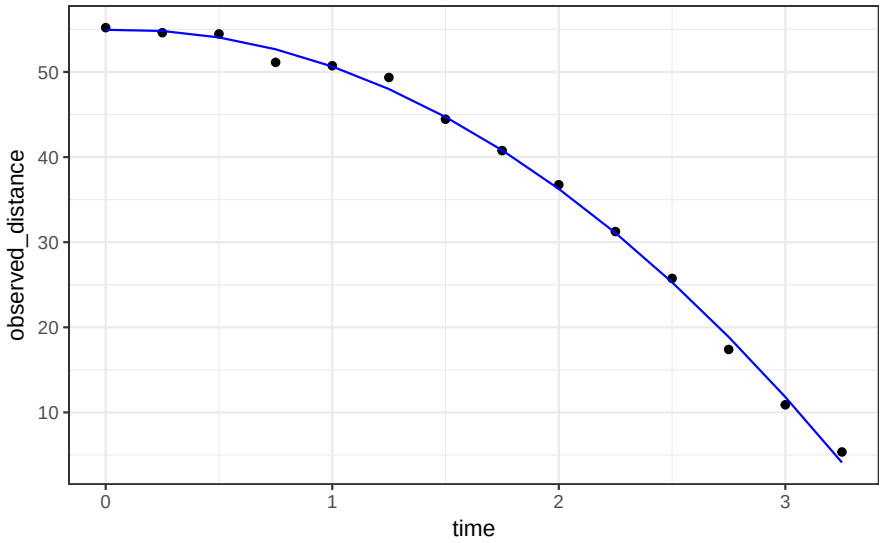
Note that the LSE calculations do not require the errors to be approximately normal. The `lm` function will find the β s that will minimize the residual sum of squares:

```
fit <- falling_object %>%
  mutate(time_sq = time^2) %>%
  lm(observed_distance~time+time_sq, data=.)
tidy(fit)
#>          term estimate std.error statistic  p.value
#> 1 (Intercept)    54.96     0.652    84.323 8.13e-17
```

```
#> 2        time     0.73     0.931     0.784 4.50e-01
#> 3     time_sq    -5.04     0.276   -18.237 1.44e-09
```

To check if the estimated parabola fits the data. The broom function `augment` let's us do this easily

```
augment(fit) %>%
  ggplot() +
  geom_point(aes(time, observed_distance)) +
  geom_line(aes(time, .fitted), col = "blue")
```



Thanks to my high school physics teacher, I know that the equation for the trajectory of a falling object is:

$$d = h_0 + v_0 t - 0.5 \times 9.8 t^2$$

with h_0 and v_0 the starting height and velocity respectively. The data we simulated above followed this equation and added measurement error to simulate `n` observations for dropping the ball ($v_0 = 0$) from the tower of Pisa ($h_0 = 56.67$).

These are consistent with the parameter estimates

```
tidy(fit, conf.int = TRUE)
#>          term estimate std.error statistic  p.value conf.low conf.high
#> 1 (Intercept)    54.96     0.652    84.323 8.13e-17    53.52     56.39
#> 2        time     0.73     0.931     0.784 4.50e-01    -1.32      2.78
#> 3     time_sq    -5.04     0.276   -18.237 1.44e-09    -5.65     -4.43
```

The tower of Pisa height is within the confidence interval for β_0, the initial velocity 0 is in the confidence interval for β_1 (not the p-value is larger than 0.05), and the acceleration constant is in a confidence interval for -2 $\times \beta_2$.

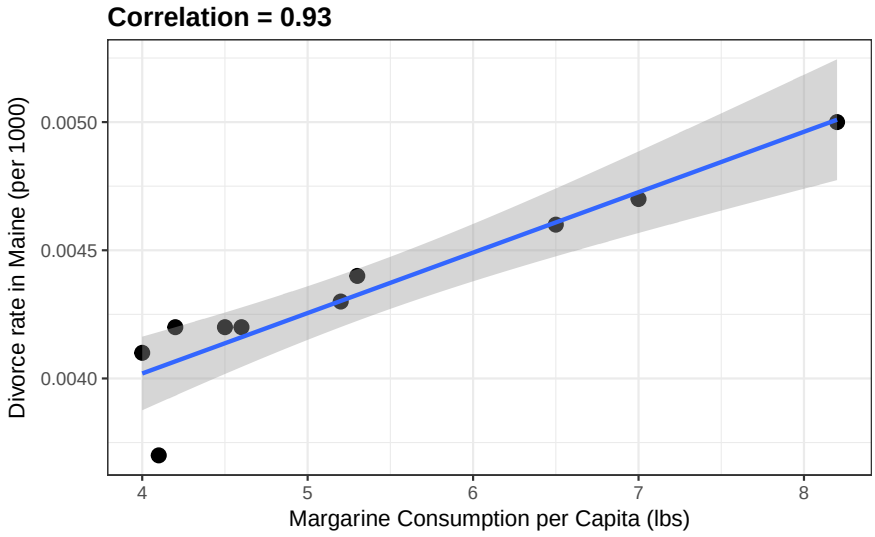
1.16 Correlation is not causation

```
library(tidyverse)
library(tidyr)
library(broom)
library(dslabs)
ds_theme_set()
```

Correlation is not causation is perhaps the most important lesson one learns in a statistics class. In this chapter we have described tools useful for quantifying associations between variables. But we must be careful not to overinterpret these results. Here we examine three ways that can lead to misinterpreting data.

1.16.1 Spurious Correlation

The following comical example underscores that correlation is not causation. It show a very strong correlation between divorce rates and margarine consumption.



Does this mean margarine causes divorces? The divorces cause people to eat more margarine? Of course not! This is just what we call a *spurious correlation*. You can see many more absurd examples of this wesbsite completely dedicated to *spurious correaltions*. There are many reasons that a variable X can correlated with a variable Y without either being the cause.

The cases presented in the *spurious correlation* site are all examples of what is generally called *data dredging*, *data fishing*, or *data snooping*. It's basically a form of what in the US they call *cherry picking*: you look through many results produced by a random process and pick the one that shows a relationship that supports your theory.

A Monte Carlo simulation can be used to show how this can happen. We will save the results of our simulation into a tibble. The first columns denotes group, and we have 1,000,000 of them. For each group we generate 25 observations which are stored in the second and third columns. These are just random independent normally distributed data. So we know, because we constructed the simulation, that X and Y are not correlated.

```
N <- 25
G <- 100000
sim_data <- tibble(group = rep(1:G, each = N), x = rnorm(N*G), y = rnorm(N*G))
```

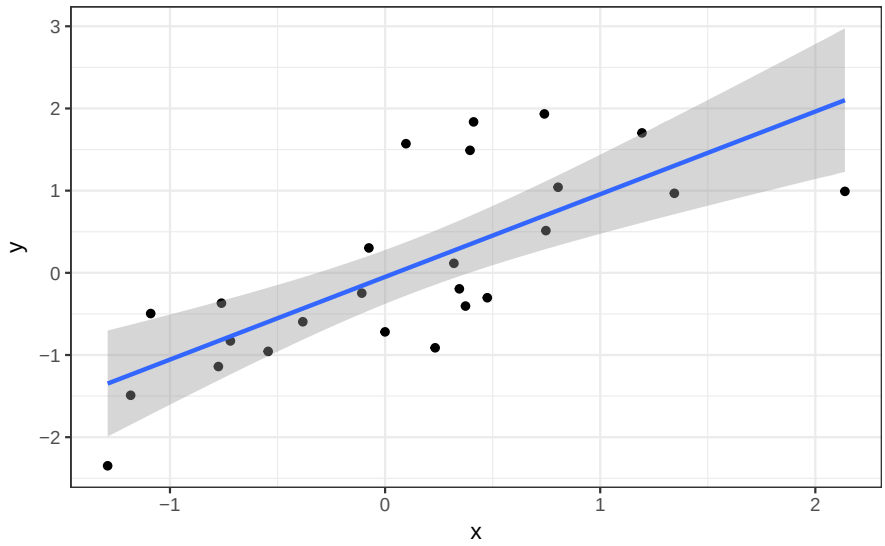
Next, we compute the correlation between `x` and `y` for each group and look at the max:

```
res <- sim_data %>% group_by(group) %>% summarize(r = cor(x,y)) %>% arrange(desc(r))
res
#> # A tibble: 100,000 x 2
#>   group     r
#>   <int> <dbl>
#> 1  2859 0.737
#> 2 71619 0.723
#> 3 57653 0.720
#> 4 15807 0.720
```

```
#> 5 30572 0.711
#> 6 52128 0.691
#> # ... with 9.999e+04 more rows
```

We see a correlation of 0.737 and if you just plot the data from that group it shows a convincing plot that x and y:

```
sim_data %>% filter(group == res$group[which.max(res$r)]) %>%
  ggplot(aes(x,y)) +
  geom_point() +
  geom_smooth(method = "lm")
```



It is just a mathematical fact that if we observe 1,000,000 random correlations that are expected to be 0, the largest one will be close 1.

Note that if we performed regression on this group and interpreted the p-value, we would incorrectly claim this was a statistically significant relation:

```
sim_data %>% filter(group == res$group[which.max(res$r)]) %>%
 do(tidy(lm(y~x, data = .)))
#>          term estimate std.error statistic  p.value
#> 1 (Intercept)  -0.0499     0.159    -0.315 7.56e-01
#> 2           x   1.0062     0.193     5.223 2.69e-05
```

This particular form of data dredging is referred to as *p-hacking*. P-hacking is a topic of much discussion because it it appears to be a problem for scientific publications. Because publishers tend to reward statistically significant results over negative results, there is an incentive to report significant results. In epidemiology and the social sciences, for example, researchers may look for associations between an adverse outcome and several exposures and report only the one exposure that resulted in a small p-value. Furthermore, they might try fitting several different models to adjust for confounding and pick the one that yields the smallest p-value. In experimental disciplines, an experiment might be repeated more than once and only the one that resulted in a small p-value reported. This does not necessarily happen due to unethical behavior, but rather to statistical ignorance. In advanced statistics course you learn methods to adjust for these multiple comparisons.

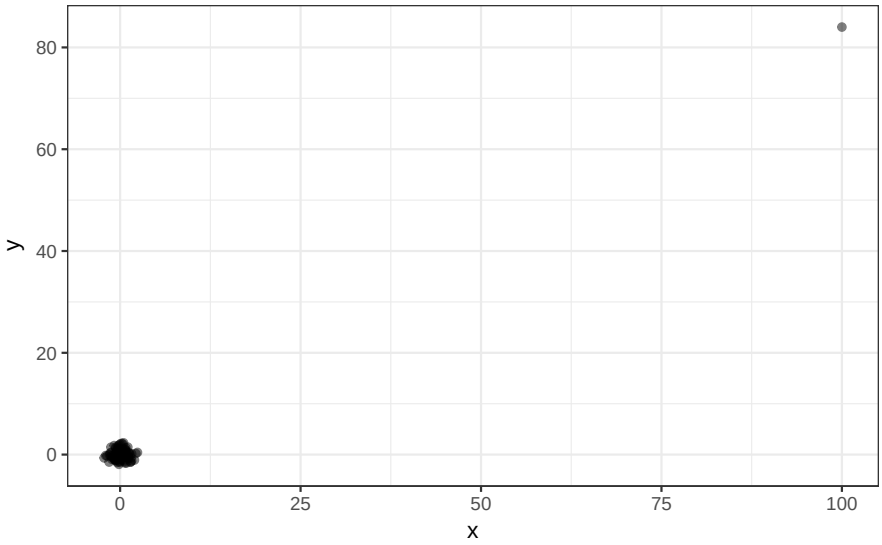
1.16.2 Outliers

Suppose we take measurements from two independent outcomes, X and Y. We standardize the measurements.

```
set.seed(1)
x=rnorm(100,0,1)
y=rnorm(100,0,1)
```

But make a mistake and forget to standardize entry 23.

```
x[23] <- 100
y[23] <- 84
tibble(x,y) %>% ggplot(aes(x,y)) + geom_point(alpha = 0.5)
```



Not suprinsingle the correlation is very high:

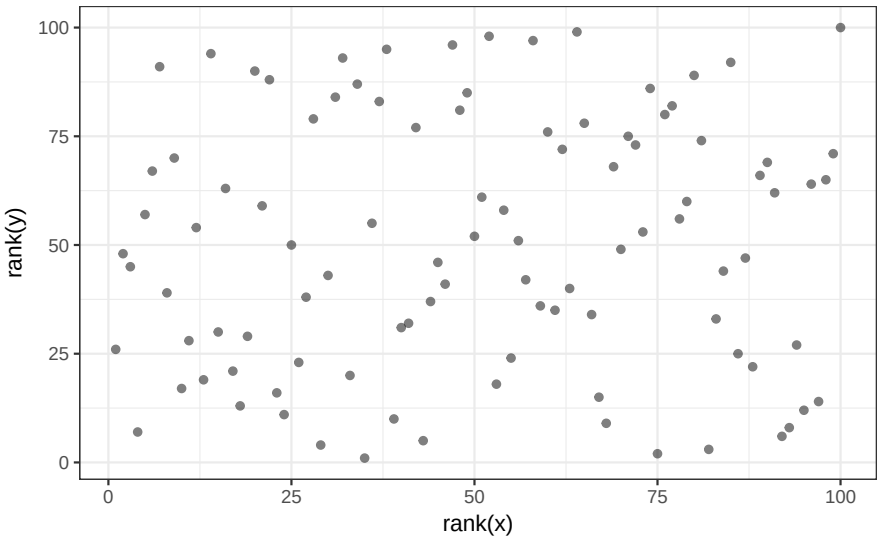
```
cor(x,y)
#> [1] 0.99
```

But this is driven by the one outlier. If we remove this outlier, the correlation is greatly reduced

```
cor(x[-23], y[-23])
#> [1] -0.00107
```

There is a alternative way to the sample correlation, for estimating the population correlation which is robust to outliers. It is called *Spearman correlation*. The idea is simple: compute the correlation on the ranks of the values:

```
tibble(x,y) %>% ggplot(aes(rank(x),rank(y))) + geom_point(alpha = 0.5)
```



The outlier is no longer associated with a very large value and the correlation comes way down:

```
cor(rank(x), rank(y))
#> [1] 0.0658
```

which can also be calculated like this:

```
cor(x,y, method = "spearman")
#> [1] 0.0658
```

There are also methods for robust fitting of linear models which can you learn about in, for example, this book: Robust Statistics: Edition 2 Peter J. Huber Elvezio M. Ronchetti

1.16.3 Reverse cause an effect

Another way associations is confused with causation is when the cause and effect are confused. An example of this is claiming that tutoring makes students perform worse because they test lower than peers that are not tutored. Here the tutoring is not causing the low test scores but the other way around. A form of this claim was actually made in an op-en in the New York Times titled Parental Involvement Is Overrated. Consider this quote from the article:

> When we examined whether regular help with homework had a positive impact on children's academic performance, we were quite startled by what we found. Regardless of a family's social class, racial or ethnic background, or a child's grade level, consistent homework help almost never improved test scores or grades… Even more surprising to us was that when parents regularly helped with homework, kids usually performed worse.

A very likely possibility is that the kids needing regular parental help, get this help because they don't perform well in school.

We can easily construct an example of cause and effect reversal using the father and son height data. Note that if we fit the model

$$X_i = \beta_0 + \beta_1 y_i + \varepsilon_i, i = 1, \ldots, N$$

to the father and son height data, with X_i the father height and y_i the son height we do get a statistically significant result:

```
library(HistData)
data("GaltonFamilies")
GaltonFamilies %>%
  filter(childNum == 1 & gender == "male") %>%
  select(father, childHeight) %>%
  rename(son = childHeight) %>%
  do(tidy(lm(father ~ son, data = .)))
#>          term estimate std.error statistic  p.value
#> 1 (Intercept)   33.965    4.5682      7.44 4.31e-12
#> 2         son    0.499    0.0648      7.70 9.47e-13
```

The model fits the data very well. If we look at the mathematical formulation of the model above, it could easily be incorrectly interpreted as to suggest that the son being tall caused the father to be tall. But what we know about genetics and biology tells us it's the other way around. The model is technically correct. What is wrong here is the interpretatin.

1.16.4 Confounders

Confounders are perhaps the most common reason that leads to associations begin misinterpreted. If X and Y are correlated, we call Z a *confounder* if changes in Z causes changes in both X and Y. Earlier we say how HR was a confounder that resulted in a higher correlation than expected when studying the relationship between BB and R. In some cases we can use linear models to account for confounders. But it is not always the case.

Incorrect interpretation due to confounders are ubiquotous in the lay press. They are sometimes hard to detect. Here we present an example that resulted in somewhat of a controversy in academia.

1.16.4.1 Gender contributes to personal research funding success in The Netherlands

A 2014 PNAS paper analyzed success rates from funding agencies in the Netherlands and concluded that their

> results reveal gender bias favoring male applicants over female applicants in the prioritization of their "quality of researcher" (but not "quality of proposal") evaluations and success rates, as well as in the language use in instructional and evaluation materials.

The main evidence for this conclusion comes down to a comparison of the percentages. Table S1 in the paper includes the information we need:

```
#>             discipline applications_total applications_men
#> 2    Chemical sciences                122              83
#> 3    Physical sciences                174             135
#> 4              Physics                 76              67
#> 5           Humanities                396             230
#> 6   Technical sciences                251             189
#> 7    Interdisciplinary                183             105
#> 8  Earth/life sciences                282             156
#> 9      Social sciences                834             425
#> 10    Medical sciences                505             245
#>    applications_women awards_total awards_men awards_women
#> 2                  39           32         22           10
#> 3                  39           35         26            9
#> 4                   9           20         18            2
#> 5                 166           65         33           32
#> 6                  62           43         30           13
#> 7                  78           29         12           17
```

```
#> 8                  126           56         38           18
#> 9                  409          112         65           47
#> 10                 260           75         46           29
#>    success_rates_total success_rates_men success_rates_women
#> 2                 26.2              26.5                25.6
#> 3                 20.1              19.3                23.1
#> 4                 26.3              26.9                22.2
#> 5                 16.4              14.3                19.3
#> 6                 17.1              15.9                21.0
#> 7                 15.8              11.4                21.8
#> 8                 19.9              24.4                14.3
#> 9                 13.4              15.3                11.5
#> 10                14.9              18.8                11.2
```

We can construct the two by two table used for the conclusion above

```
two_by_two <- research_funding_rates %>%
  select(-discipline) %>%
  summarize_all(funs(sum)) %>%
  summarize(yes_men = awards_men,
            no_men = applications_men - awards_men,
            yes_women = awards_women,
            no_women = applications_women - awards_women) %>%
  gather %>%
  separate(key, c("awarded", "gender")) %>%
  spread(gender, value)
two_by_two
#>   awarded  men women
#> 1      no 1345  1011
#> 2     yes  290   177
```

Compute the difference in percentage

```
two_by_two %>% mutate(men = round(men/sum(men)*100, 1), women = round(women/sum(women)*100, 1)) %>% fil
#>   awarded  men women
#> 1     yes 17.7  14.9
```

note that it's lower for women and find that it is almost statistically significant at the 0.05 level:

```
two_by_two %>% select(-awarded) %>% fisher.test() %>% tidy
#>   estimate p.value conf.low conf.high                                method
#> 1    0.812  0.0456    0.658         1 Fisher's Exact Test for Count Data
#>   alternative
#> 1   two.sided
```

So there appears to be some evidence of an association. But can we infer causation here? Is gender bias causing this observed difference.

A response was published a few months later titled *No evidence that gender contributes to personal research funding success in The Netherlands: A reaction to van der Lee and Ellemers* which concluded

> However, the overall gender effect borders on statistical significance, despite the large sample. Moreover, their conclusion could be a prime example of Simpson's paradox; if a higher percentage of women apply for grants in more competitive scientific disciplines (i.e., with low application success rates for both men and women), then an analysis across all disciplines could incorrectly show "evidence" of gender inequality.

What is this Simpson's Paradox they refer to? To explain Simpson's paradox we will describe a similar data sets related to gender bias.

1.16.5 Simpson's Paradox

A very clear example of how Simpson's Paradox has resulted in incorrect conclusions comes from another controversy related to gender bias. Admission data from U.C. Berkeley 1973 showed that more men were being admitted than women: 44% men were admitted compared to 30% women. PJ Bickel, EA Hammel, and JW O'Connell. Science (1975). Here is the data:

Probability of

```
data(admissions)
admissions
#>    major gender admitted applicants
#> 1      A    men       62        825
#> 2      B    men       63        560
#> 3      C    men       37        325
#> 4      D    men       33        417
#> 5      E    men       28        191
#> 6      F    men        6        373
#> 7      A  women       82        108
#> 8      B  women       68         25
#> 9      C  women       34        593
#> 10     D  women       35        375
#> 11     E  women       24        393
#> 12     F  women        7        341
```

The percent of men and womena that got accepted are

```
admissions %>% group_by(gender) %>%
  summarize(percentage = round(sum(admitted*applicants)/sum(applicants),1))
#> # A tibble: 2 x 2
#>   gender percentage
#>    <chr>      <dbl>
#> 1    men       44.5
#> 2  women       30.3
```

A statistical test clearly rejects the hypothesis that gender and admission are independent:

```
admissions %>% group_by(gender) %>%
  summarize(total_admitted = round(sum(admitted/100*applicants)),
            not_admitted = sum(applicants) - sum(total_admitted)) %>%
  select(-gender) %>%
  do(tidy(fisher.test(.)))
#>   estimate  p.value conf.low conf.high                                method
#> 1     1.84 4.84e-22     1.62      2.09 Fisher's Exact Test for Count Data
#>   alternative
#> 1   two.sided
```

But closer inspection shows a paradoxical result. Here are the percent admissions by major:

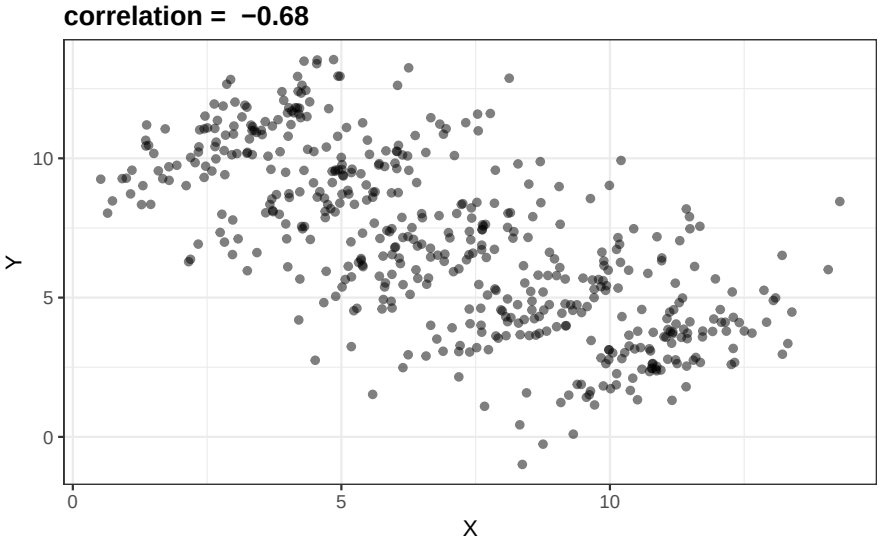
```
admissions %>% select(major, gender, admitted) %>%
  spread(gender, admitted) %>%
  mutate(women_minus_men = women - men)
#>   major men women women_minus_men
#> 1     A  62    82              20
#> 2     B  63    68               5
#> 3     C  37    34              -3
#> 4     D  33    35               2
#> 5     E  28    24              -4
```

```
#> 6     F    6     7                  1
```

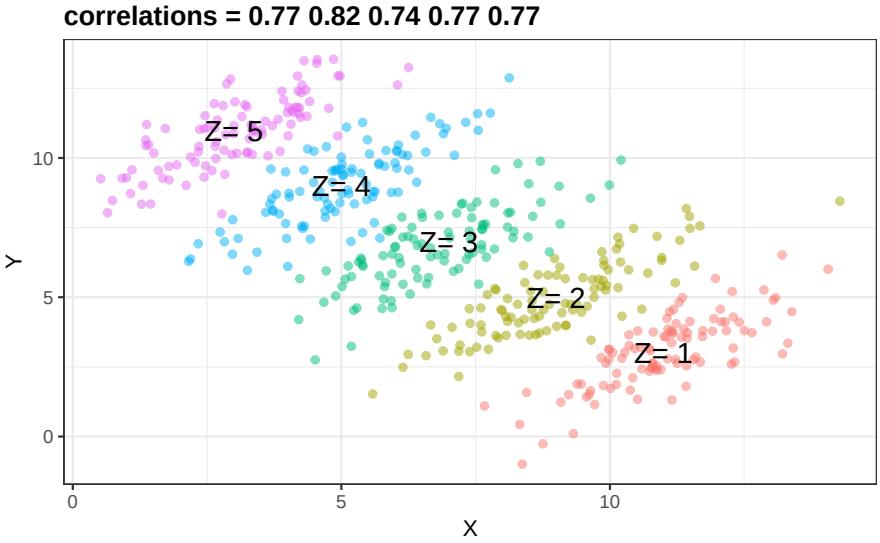
Four out of the six majors favor women. More importantly all the difference are much smaller than the 14.2 difference that we see when examining the totals.

The paradox is that analyzing the totals suggests a dependence between admission and gender, but when the data is grouped by major, this dependence seems to disappear. What's going on? This actually can happen if an unacounted confounder is driving most of the variability.

Before continuing with the admissions example, we show a very clear illustration of how this can happen:



You can see that X and Y are negatively correlated. However, once we stratify by Z (shown in different colors), which we have not yet showns, another pattern emerges.

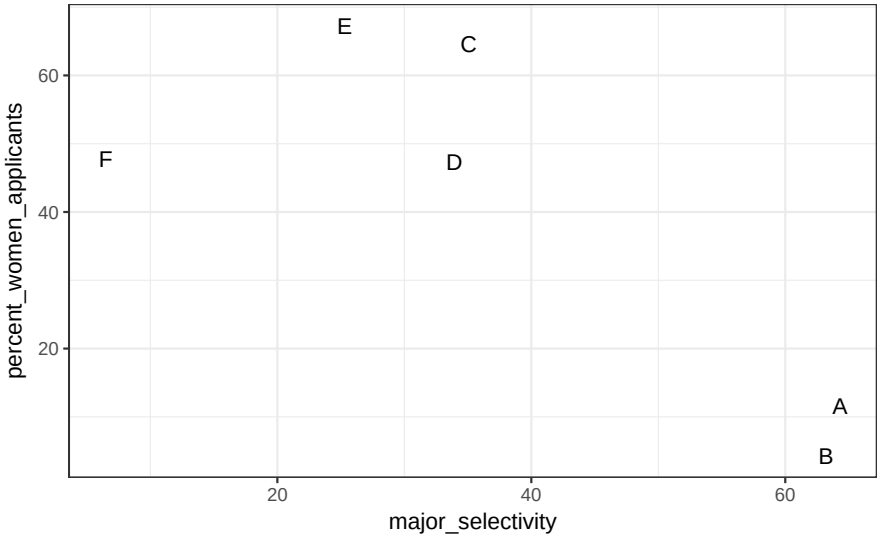


It is really Z that is negatively correlated with X. If we stratify by Z the X and Y are actually positively correlated:

So let's define three variables: X is 1 for men and 0 for women, Y is 1 for those admitted and 0 otherwise, and Z quantifies how selective is the major. A gender bias claim would be based on the fact that $\Pr(Y = 1|X = x)$ is higher for $x = 1$ then $x = 0$. But Z is an important confounder to consider. Clearly Z is associated with Y, as the more selective a major the lower $\Pr(Y = 1|Z = z)$. But is major selectivity Z associated with gender X?

One way to see this is to plot the total percent admitted to a major versus the percent of women that make up the applicants:

```
admissions %>%
  group_by(major) %>%
  summarize(major_selectivity = sum(admitted*applicants)/sum(applicants),
            percent_women_applicants = sum(applicants*(gender=="women")/sum(applicants))*100) %>%
  ggplot(aes(major_selectivity, percent_women_applicants, label = major)) +
  geom_text()
```



There seems to association. The plot suggests that women were much more likely to apply to "hard" majors: gender and majors difficulty are confounded. Compare, for example, major B and major E. Major B is much harder to enter than major E and over 60% of applicants to major B were women while less than 30% of the applicants of major E were women.

1.16.5.1 Confounding explained graphically

The following plot shows the percent of applicants that were accepted by gender.

```
admissions %>%
  mutate(percent_admitted = admitted*applicants/sum(applicants)) %>%
  ggplot(aes(gender, y = percent_admitted, fill = major)) +
  geom_bar(stat = "identity", position = "stack")
```

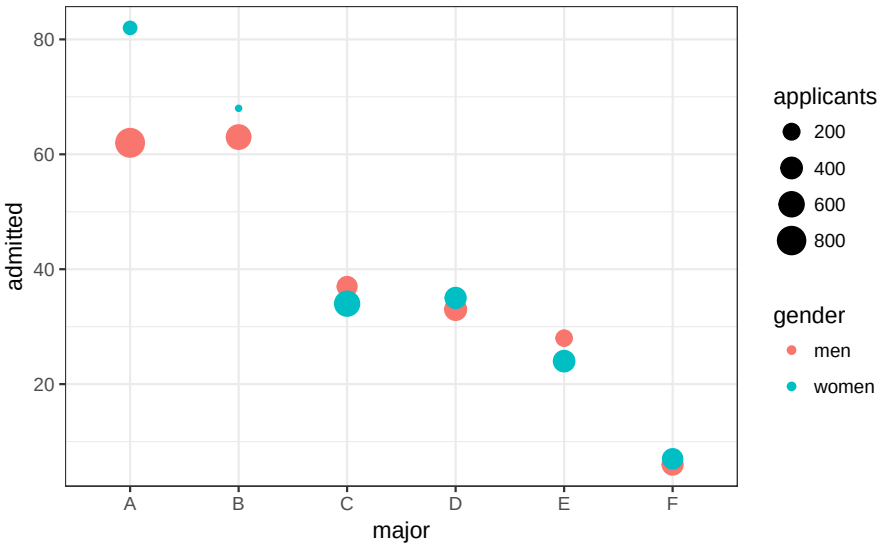
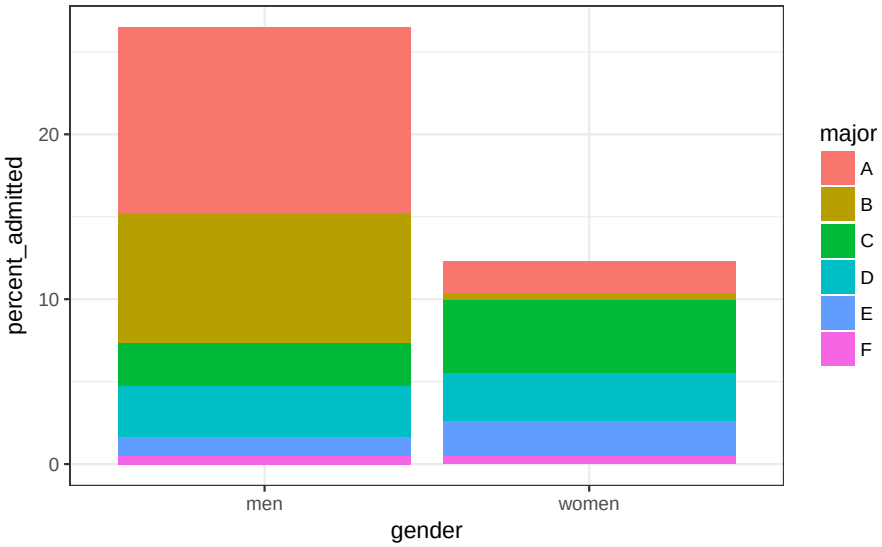


Figure 7: Admission percentage by major for each gender.



It also breaks down the acceptance rates by major: the size of the colored bars represent the percent of each major stundents were admitted to. This breakdown let's us see that the majority of accepted men come from two majors: A and B. It also let's us see that few women applied to these majors. What the plot does **not** show us is what the percent admitted by major.

1.16.6 Average after stratifying

In this plot, we can see that if we condition or stratify by major, and then look at differences, we control for the confounder and this effect goes away.

```
admissions %>%
  ggplot(aes(major, admitted, col = gender, size = applicants)) +
  geom_point()
```

Now we see that major by major there is not much difference. The size of the dot represents the number of applicants, and explains the paradox: The easiest majors have large red dots and small blue dots.

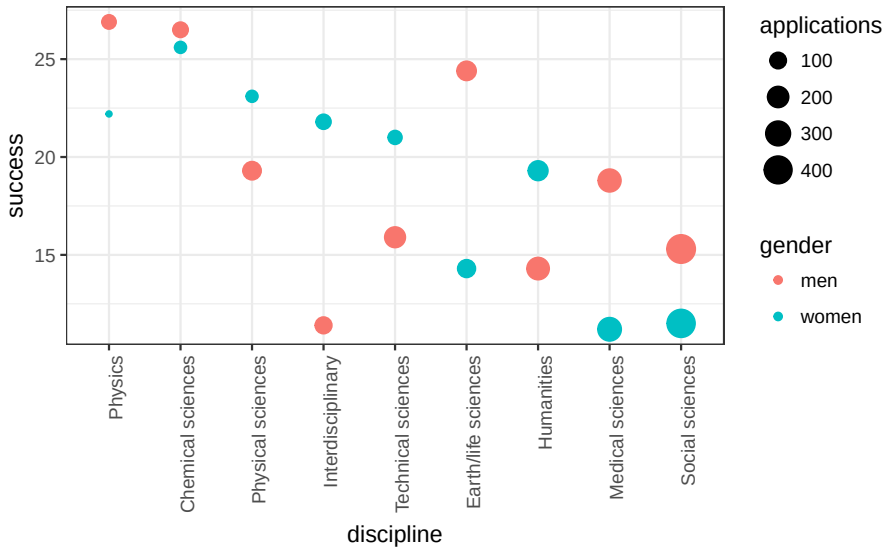
If we average the difference by major we find that the percent ad actually 3.5% higher for women.

```
admissions %>%  group_by(gender) %>% summarize(average = mean(admitted))
#> # A tibble: 2 x 2
#>   gender average
#>    <chr>   <dbl>
#> 1    men    38.2
#> 2  women    41.7
```

1.16.7 Back to the Reseaech Funding

In the UC Berkeley admissions examples, the overall differences were explained by difference across disciplines. We use the same approach on the research funding data and looks at comparisons by discipline:

```
dat <- research_funding_rates %>%
  rename(success_total = success_rates_total,
         success_men = success_rates_men,
         success_women = success_rates_women) %>%
  gather(key, value, -discipline) %>%
  separate(key, c("type", "gender")) %>%
  spread(type, value) %>%
  filter(gender != "total") %>%
  mutate(discipline = reorder(discipline, applications, sum))
dat %>%
  ggplot(aes(discipline, success, size = applications, color = gender)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  geom_point()
```



Here we see that some fields favor men and other women. We see that the two fields with the largest difference favoring men, are also the fields with the most applications. However, are any of these differences statistically significant? Note that even when there is no bias we will see differences due to random variability in the review process as well as random variability across candidates. If we perform a fisher test in each discipline we see that most differences result in p-values larger than 0.05.

```
do_fisher_test <- function(m, x, n, y){
  tab <- tibble(men = c(x, m-x), women = c(y, n-y))
  tidy(fisher.test(tab)) %>%
```

```
    rename(odds = estimate) %>%
    mutate(difference = y/n - x/m)
}
res <- research_funding_rates %>%
  group_by(discipline) %>%
  do(do_fisher_test(.$applications_men, .$awards_men,
                    .$applications_women, .$awards_women)) %>%
  ungroup() %>%
  select(discipline, difference, p.value) %>%
  arrange(difference)
res
#> # A tibble: 9 x 3
#>            discipline difference p.value
#>                 <chr>      <dbl>   <dbl>
#> 1 Earth/life sciences   -0.10073  0.0367
#> 2    Medical sciences   -0.07622  0.0175
#> 3             Physics   -0.04643  1.0000
#> 4     Social sciences   -0.03803  0.1274
#> 5   Chemical sciences   -0.00865  1.0000
#> 6   Physical sciences    0.03818  0.6514
#> # ... with 3 more rows
```

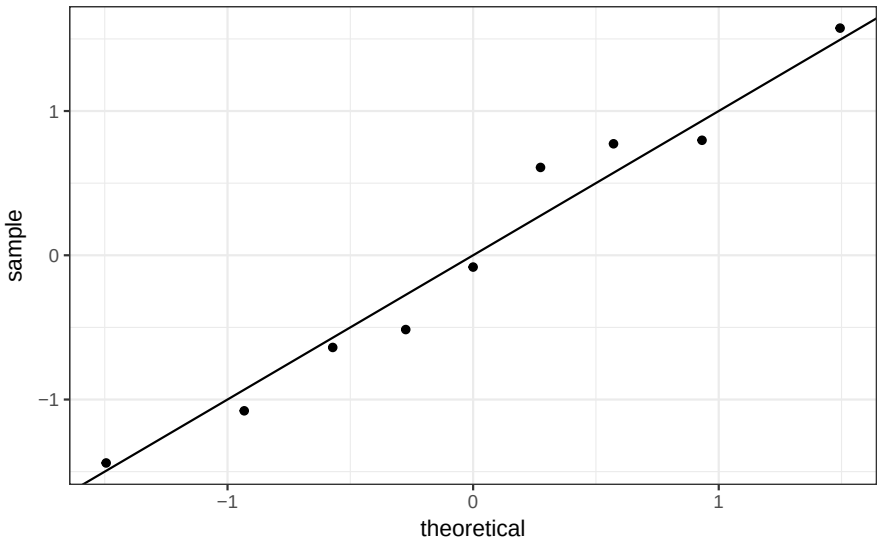
We see that for Earth/life science there is a difference of 10% favoring men and this has a p-value of 0.04. But is this a spurious correlation? We perforemd 9 tests. Peporting only the one case with a p-value less than 0.05 would be cherry picking.

The overall average of the difference is only -0.3%, which is much smaller than the standard error:

```
res %>% summarize(overall_avg = mean(difference),
                  se = sd(difference)/sqrt(n()))
#> # A tibble: 1 x 2
#>   overall_avg     se
#>         <dbl>  <dbl>
#> 1    -0.00311 0.0226
```

Furthermore, note that the difference appear to follow a normal distribution

```
res %>% ggplot(aes(sample = scale(difference))) + stat_qq() + geom_abline()
```



which points that the possibility that the observed differences are just due to change.