

# Lecture 10

R and the tidyverse // regression

---

Ivan Rudik  
AEM 4510

# Roadmap

- What is R?
- What is the tidyverse?
- How do we import and manipulate data?

Our goal is to take a hands on approach to learning how we actually **do** environmental economics

A good chunk of this lecture comes from Grant McDermott's **data science for economists** notes, and **RStudio education**

# RStudio Cloud

---

# Getting started

We will be using [rstudio.cloud](#) for our coding

# Getting started

We will be using [rstudio.cloud](#) for our coding

Why?

# Getting started

We will be using **rstudio.cloud** for our coding

Why?

You don't need to download/install anything

# Getting started

We will be using **rstudio.cloud** for our coding

Why?

You don't need to download/install anything

I can prepare the packages and code and make it easy to download

# Getting started

We will be using **rstudio.cloud** for our coding

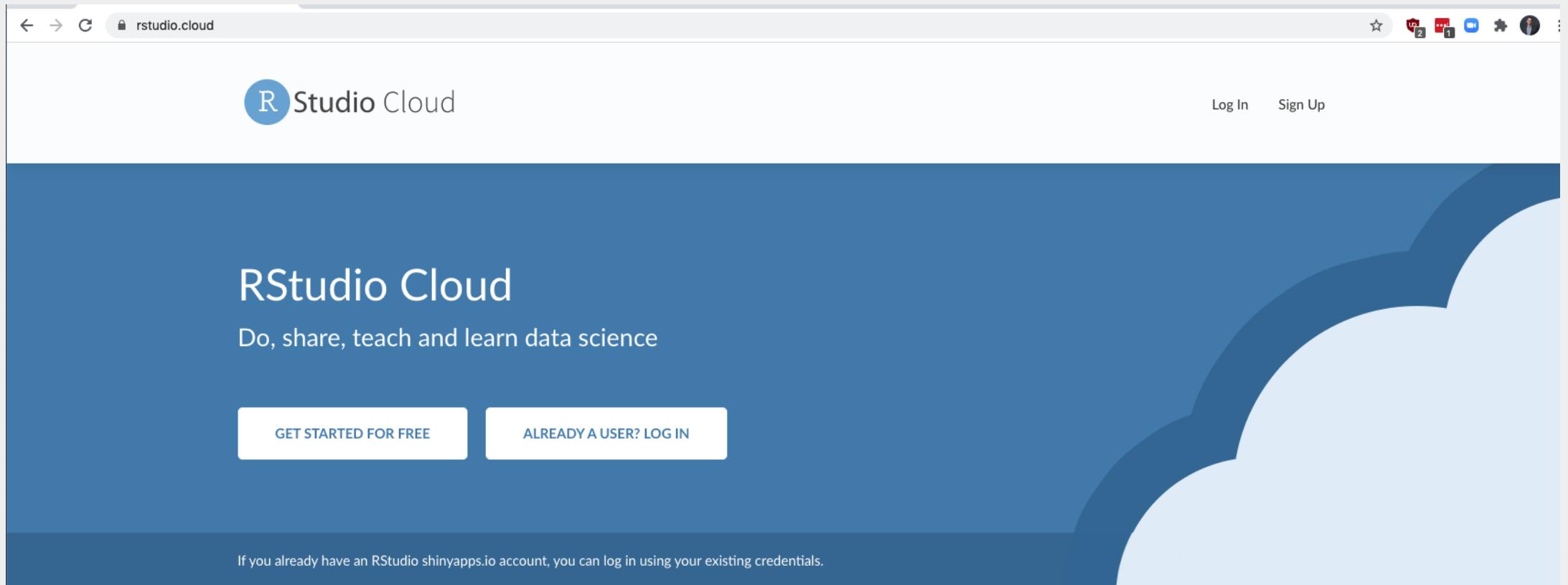
Why?

You don't need to download/install anything

I can prepare the packages and code and make it easy to download

Let's get everything going...

# Getting started: login



The screenshot shows the RStudio Cloud login page. At the top, there's a header with a back arrow, forward arrow, refresh button, and a lock icon followed by "rstudio.cloud". To the right of the address bar are several browser extension icons. Below the header, the "R Studio Cloud" logo is on the left, and "Log In" and "Sign Up" links are on the right. The main section has a blue background with a white cloud graphic on the right. The text "RStudio Cloud" and "Do, share, teach and learn data science" is centered. Two buttons are present: "GET STARTED FOR FREE" and "ALREADY A USER? LOG IN". A note at the bottom of this section says, "If you already have an RStudio shinyapps.io account, you can log in using your existing credentials." The footer is white with a dark blue horizontal bar containing the RStudio Cloud logo and the text "Data science without the hardware hassles".

R Studio Cloud

Log In    Sign Up

RStudio Cloud

Do, share, teach and learn data science

GET STARTED FOR FREE    ALREADY A USER? LOG IN

If you already have an RStudio shinyapps.io account, you can log in using your existing credentials.

## Data science without the hardware hassles

RStudio Cloud is a lightweight, cloud-based solution that allows anyone to do, share, teach and learn data science online.

- Analyze your data using the RStudio IDE, directly from your browser.

---

\$ AVAILABLE PRICING PLANS

---

🕒 RSTUDIO CLOUD GUIDE

---

🌐 RSTUDIO.COM

---

# Getting started: new project

The screenshot shows the RStudio Cloud interface at the URL [rstudio.cloud/spaces/125964/projects](https://rstudio.cloud/spaces/125964/projects). The left sidebar has sections for 'Spaces' (Your Workspace, AEM 4510, AEM 6510, New Space), 'Learn' (Guide, What's New, Primers, Cheat Sheets), and 'Help'. The main area shows 'AEM 4510' by Ivan Rudik with tabs for 'Projects' (selected), 'Members', and 'About'. The 'Projects' tab shows 'All Projects' with a dropdown for 'List' (All projects) and 'Sort' (By name). A modal menu is open under the 'New Project' button, listing '+ New Project' and 'New Project from Git Repository'. Below the modal are buttons for 'Delete', 'Move', and 'Export'. A project named 'In-10' by Ivan Rudik is listed, created on Mar 29, 2021, 10:22 AM.

AEM 4510  
Ivan Rudik

Projects Members About

All Projects

List All projects Sort By name

In-10  
Ivan Rudik  
Created Mar 29, 2021 10:22 AM

New Project

+ New Project  
New Project from Git Repository

Delete Move Export

# Getting started: wait for deployment

The screenshot shows the RStudio Cloud interface at [rstudio.cloud/project/1795327](https://rstudio.cloud/project/1795327). The left sidebar has a dark theme with white text. The 'Your Workspace' section is selected. The main area shows a large, semi-transparent circular progress bar with a blue dot icon and the text 'Deploying Project'. At the top right, there are user icons for 'Ivan Rudik' and other users, along with a gear icon for settings.

← → C 🔒 rstudio.cloud/project/1795327

R Studio Cloud Your Workspace / aem6510

Spaces

Your Workspace New Space

Learn

Guide What's New Primers Cheat Sheets

Help

Current System Status RStudio Community

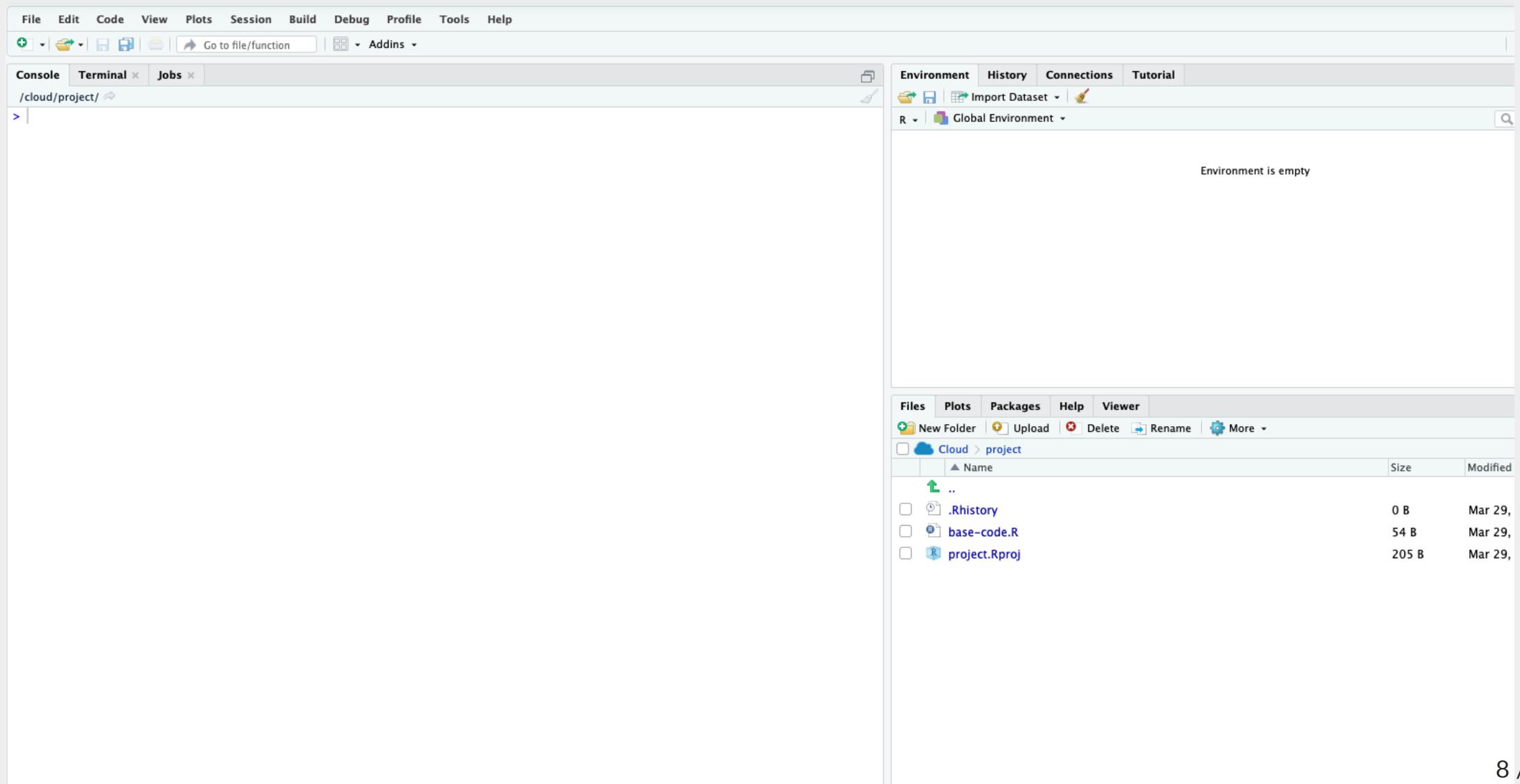
Info

Plans & Pricing Terms and Conditions

Deploying Project

7 / 162

# Click on base-code in bottom-right



# Code script open!

The screenshot shows the RStudio IDE interface with the following components:

- Top Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Toolbar:** Includes icons for New File, Open, Save, Print, Go to file/function, and Addins.
- Code Editor:** A tab labeled "base-code.R" containing the following R code:

```
1 # load packages
2 library(tidyverse)
3
4
5 # start code here
```
- Environment Panel:** Shows the Global Environment with the message "Environment is empty".
- Files Panel:** Shows a project structure in the Cloud:
  - New Folder
  - Upload
  - Delete
  - Rename
  - More

Contents of the project:

  - ..
  - .Rhistory
  - base-code.R
  - project.Rproj
- Console:** Shows the path "/cloud/project/" and a prompt ">".
- Status Bar:** Displays the time "5:18" and the text "(Top Level)".

# Now we're set

Now we're all set with our coding environment

# Now we're set

Now we're all set with our coding environment

You can write code in the top window and save it as a file

# Now we're set

Now we're all set with our coding environment

You can write code in the top window and save it as a file

Or you can just enter it in the console in the bottom if you don't want to save it

# Now we're set

Now we're all set with our coding environment

You can write code in the top window and save it as a file

Or you can just enter it in the console in the bottom if you don't want to save it

Highlight code in the top window and press cmd+enter to run those highlighted lines

# Quick intro to R

---

# What is R?

## What is R?

R is a language and environment for statistical computing and graphics.

## What is RStudio?

RStudio is an integrated development environment (IDE) for R, a programming language for statistical computing and graphics.

# What is R?

## What is R?

R is a language and environment for statistical computing and graphics.

## What is RStudio?

RStudio is an integrated development environment (IDE) for R, a programming language for statistical computing and graphics.

# Why are we using R?

1. R is free

# Why are we using R?

1. R is free
2. R is relatively easy to learn (especially with `tidyverse` which we will be using)

# Why are we using R?

1. R is free
2. R is relatively easy to learn (especially with `tidyverse` which we will be using)
3. R is widely used for statistical analysis and data science in business, economics, natural sciences

# Why are we using R?

1. R is free
2. R is relatively easy to learn (especially with `tidyverse` which we will be using)
3. R is widely used for statistical analysis and data science in business, economics, natural sciences
4. R has a large online community for help (e.g. StackOverflow), and lots and lots of packages that help you do your analysis smoothly

# Let's see what we can do in R

Next let's see how to use R

# Let's see what we can do in R

Next let's see how to use R

First we will start with the basics

# Let's see what we can do in R

Next let's see how to use R

First we will start with the basics

Then we will start doing more complicated exercises that are important for doing data analysis and economics work

# Let's see what we can do in R

Next let's see how to use R

First we will start with the basics

Then we will start doing more complicated exercises that are important for doing data analysis and economics work

Please follow along in RStudio/RStudio Cloud

# Arithmetic operations

R can do all the standard arithmetic operations

```
1+2 ## add
```

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

```
6-7 ## subtract
```

```
## [1] -1
```

```
5/2 ## divide
```

```
## [1] 2.5
```

# Logical operations

You also have logical operations

```
1 > 2
```

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

```
(1 > 2) | (1 > 0.5) # / is the or operator
```

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

```
(1 > 2) & (1 > 0.5) # & is the and operator
```

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

# Logical operations

We can negate expressions with: !

This is helpful for filtering data

```
is.na(1:10)
```

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

```
!is.na(1:10)
```

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

NA means **not available** (i.e. missing)

# Logical operators

For value matching we use: `%in%`

To see whether an object is contained within (i.e. matches one of) a list of items, use `%in%`.

```
4 %in% 1:10
```

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

```
4 %in% 5:10
```

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

This is kind of like an `any` command in other languages

# Logical operators

To evaluate whether two expressions are equal, we need to use **two** equal signs

```
1 = 1 ## This doesn't work
```

```
## Error in 1 = 1: invalid (do_set) left-hand side to assignment
```

```
1 == 1 ## This does.
```

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

```
1 != 2 ## Note the single equal sign when combined with a negation.
```

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

# Assignment

In R, we can use either `=` or `←` to handle assignment<sup>1</sup>

<sup>1</sup> The `←` is really a `<` followed by a `-`. It just looks like an arrow because of the font on the slides

# Assignment

In R, we can use either `=` or `←` to handle assignment<sup>1</sup>

`←` is normally read aloud as "gets"

You can think of it as a (left-facing) arrow saying **assign in this direction**

<sup>1</sup> The `←` is really a `<` followed by a `-`. It just looks like an arrow because of the font on the slides

# Assignment

```
a ← 10 + 5  
a
```

```
## [1] 15
```

# Assignment

You can also use `=` for assignment

```
b = 10 + 10  
b
```

```
## [1] 20
```

# Which assignment operator to use?

Most R folks prefer `←` for assignment

# Which assignment operator to use?

Most R folks prefer `←` for assignment

It doesn't really matter though, other languages strictly use `=`

# Which assignment operator to use?

Most R folks prefer `←` for assignment

It doesn't really matter though, other languages strictly use `=`

**Use whatever you prefer, just be consistent**

# Help

If you are struggling with a (named) function or object in R, simply type ?

command here

```
?rowSums
```

# Help

Also try `vignette()` for a more detailed introduction to many packages

```
# Try this:  
vignette("dplyr")
```

# Help

Also try `vignette()` for a more detailed introduction to many packages

```
# Try this:  
vignette("dplyr")
```

Vignettes are a very easy way to learn how and when to use a package

# Object-oriented programming

In R:

"Everything is an object and everything has a name."

# What are objects?

We won't go into details but here are some objects that we'll be working with regularly:

- vectors
- matrices
- data frames
- lists
- etc.

# What are objects?

We won't go into details but here are some objects that we'll be working with regularly:

- vectors
- matrices
- data frames
- lists
- etc.

A lot of these are probably familiar if you have coding experience

# Data frames

The most important object we will be working with is the **data frame**

# Data frames

The most important object we will be working with is the **data frame**

You can think of it basically as an Excel spreadsheet

# Data frames

The most important object we will be working with is the **data frame**

You can think of it basically as an Excel spreadsheet

```
## Create a small data frame called "df".  
df ← data.frame(x = 1:2, y = 3:4)  
df
```

```
##   x y  
## 1 1 3  
## 2 2 4
```

# Data frames

The most important object we will be working with is the **data frame**

You can think of it basically as an Excel spreadsheet

```
## Create a small data frame called "df".  
df ← data.frame(x = 1:2, y = 3:4)  
df
```

```
##   x y  
## 1 1 3  
## 2 2 4
```

This is essentially just a table with columns named `x` and `y`

# Data frames

The most important object we will be working with is the **data frame**

You can think of it basically as an Excel spreadsheet

```
## Create a small data frame called "df".  
df ← data.frame(x = 1:2, y = 3:4)  
df
```

```
##   x y  
## 1 1 3  
## 2 2 4
```

This is essentially just a table with columns named `x` and `y`

Each row is an observation telling us the values of both `x` and `y`

# Working with multiple objects

In R we can have multiple data frames in memory at once

```
df2 <- data.frame(x = rnorm(10), y = rnorm(10)) # x and y are each 10 normal random numbers  
df # df still exists even though we made df2!
```

```
##   x  y  
## 1 1  3  
## 2 2  4
```

```
df2 # here's df2
```

```
##           x          y  
## 1 -0.5931213  0.61789560  
## 2  1.2577549 -0.21840102  
## 3  1.4785890  0.65103015  
## 4  0.2589759 -0.41870007  
## 5 -0.2724455 -0.56776428  
## 6  0.7611042  0.52591216  
## 7 -1.2124237 -0.42724465  
## 8  0.1835111  0.53510305  
## 9 -0.4444444  0.56776428  
## 10 0.52591216 0.18351111
```

# Built in dataframes

R (and its packages) also has a bunch of built in dataframes with special names that you can call upon any time, we will be using these for examples

```
mtcars
```

```
##          mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant       18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360    14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE     16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SLI    17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
```

# Built in dataframes

R (and its packages) also has a bunch of built in dataframes with special names that you can call upon any time, we will be using these for examples

This one is in the `dplyr` package that we will load later

```
starwars
```

```
## # A tibble: 87 x 14
##   name    height  mass hair_color skin_color eye_color birth_year sex gender homeworld species film
##   <chr>     <int> <dbl> <chr>       <chr>       <chr>      <dbl> <chr> <chr>       <chr>   <list>
## 1 Luke...     172     77 blond      fair        blue         19 male   masculin... Tatooine Human <chr>
## 2 C-3PO       167     75 <NA>       gold        yellow       112 none   masculin... Tatooine Droid  <chr>
## 3 R2-D2        96     32 <NA>       white, bl... red          33 none   masculin... Naboo  Droid  <chr>
## 4 Dart...      202    136 none      white        yellow       41.9 male   masculin... Tatooine Human <chr>
## 5 Leia...      150     49 brown     light        brown        19 femal... feminin... Alderaan Human <chr>
## 6 Owen...      178    120 brown, gr... light        blue         52 male   masculin... Tatooine Human <chr>
## 7 Beru...      165     75 brown     light        blue         47 femal... feminin... Tatooine Human <chr>
## 8 R5-D4       97      32 <NA>       white, red red          NA none   masculin... Tatooine Droid  <chr>
```

# Reserved words

R has a bunch of key/reserved words that serve specific functions

See [here](#) for a full list, including (but not limited to):

```
if  
else  
while # looping  
function  
for # looping  
TRUE  
FALSE  
NULL # null/undefined  
Inf # infinity  
NaN # Not a number  
NA # Not available / missing
```

# Semi-reserved words

There are other words that are sort of reserved, in that they have a particular meaning

The most important one is `c` which binds and concatenates objects together

```
my_vector ← c(1, 2, 5)  
my_vector
```

```
## [1] 1 2 5
```

This created a vector/row consisting of 1, 2, 5

# Namespace conflicts

Try loading up `dplyr` in RStudio

```
library(dplyr)
```

What warning gets reported?

# Namespace conflicts

Try loading up `dplyr` in RStudio

```
library(dplyr)
```

What warning gets reported?

The warning *masked from 'package:X'* is about a **namespace conflict**

# Namespace conflicts

Try loading up `dplyr` in RStudio

```
library(dplyr)
```

What warning gets reported?

The warning *masked from 'package:X'* is about a **namespace conflict**

`dplyr` and the `stats` package (which gets loaded automatically when you start R) have functions named `filter` and `lag`

# Namespace conflicts

Whenever a namespace conflict arises, the most recently loaded package will gain preference

# Namespace conflicts

Whenever a namespace conflict arises, the most recently loaded package will gain preference

The `filter()` function now refers specifically to the `dplyr` variant

# Namespace conflicts

Whenever a namespace conflict arises, the most recently loaded package will gain preference

The `filter()` function now refers specifically to the `dplyr` variant

But what if we want the `stats` variant?

# Namespace conflicts

Whenever a namespace conflict arises, the most recently loaded package will gain preference

The `filter()` function now refers specifically to the `dplyr` variant

But what if we want the `stats` variant?

1. Temporarily use `stats::filter()`
2. Permanently assign `filter <- stats::filter`

# Solving namespace conflicts

Use `package :: function()`

# Solving namespace conflicts

Use `package::function()`

Explicitly call a conflicted function from a package using the `package::function()` syntax:

```
stats::filter(1:10, rep(1, 2))
```

```
## Time Series:  
## Start = 1  
## End = 10  
## Frequency = 1  
## [1] 3 5 7 9 11 13 15 17 19 NA
```

# Solving namespace conflicts

We can also use `::` for more than just conflicted cases.

# Solving namespace conflicts

We can also use `::` for more than just conflicted cases.

It can clarify where a function or dataset comes from and make the code clearer:

```
dplyr::starwars ## Print the starwars data frame from the dplyr package  
scales::comma(c(1000, 1000000)) ## Use the comma function, which comes from the scales package
```

# Solving namespace conflicts

We can also use `::` for more than just conflicted cases.

It can clarify where a function or dataset comes from and make the code clearer:

```
dplyr::starwars ## Print the starwars data frame from the dplyr package  
scales::comma(c(1000, 1000000)) ## Use the comma function, which comes from the scales package
```

The `::` syntax also means that we can call functions without loading package first. E.g. As long as `dplyr` is installed on our system, then

```
dplyr::filter(iris, Species="virginica") will work.
```

# Solving namespace conflicts

Assign `function ← package::function`

# Solving namespace conflicts

```
Assign function <- package::function
```

You can permanently assign a conflicted function name to a particular package

# Solving namespace conflicts

```
Assign function <- package::function
```

You can permanently assign a conflicted function name to a particular package

This will hold for the remainder of your current R session, or until you change it back:

```
filter <- stats::filter ## Note the lack of parentheses.  
filter <- dplyr::filter ## Change it back again.
```

# Indexing

How do we index in R?

# Indexing

How do we index in R?

We've already seen an example of indexing in the form of R console output:

```
1+2
```

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

The `[1]` above denotes the first (and, in this case, only) element of our output

# Indexing

How do we index in R?

We've already seen an example of indexing in the form of R console output:

```
1+2
```

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

The `[1]` above denotes the first (and, in this case, only) element of our output

In this case, a vector of length one equal to the value "3"

# Indexing

Try the following in your console to see a more explicit example of indexed output:

```
rnorm(n = 100, mean = 0, sd = 1) # 100 random variables with mean 0 standard deviation 1
```

```
## [1]  0.09624156 -0.92935568 -2.26134254  1.59193398  0.17399060 -0.06339205  0.78440285 -0.1117587  
## [10] 1.07125342  0.11530815 -1.02561823 -1.82490703  0.91480452 -0.44733162  1.03644635  0.7316104  
## [19] -0.65382881  1.42584562 -1.47432514  0.67487397 -1.04597700  0.74043296  0.30407875 -0.4612921  
## [28]  0.98012905 -0.73136772  1.43594170 -0.53043288  2.93487869  0.62946193 -0.15306036  0.1063545  
## [37] -0.73639126  0.74534237 -2.09415019 -0.90588744 -0.79974789 -0.80881428 -0.43920683  0.7727759  
## [46] -0.15552591  0.82839637 -0.66804680  0.21666360  2.29278223  1.68942658  0.46718001  0.1117758  
## [55]  0.29735613 -0.10137821 -1.02488839 -0.54620863  0.50612632 -0.06138068  2.26030944  0.5382938  
## [64]  0.44353567 -0.33670815  0.30232161 -2.08361213  0.08230098  0.32622700 -0.30511533 -0.1585478  
## [73]  0.43771517 -0.50065523  0.58507690  0.83178380 -2.02858363  0.78691988  0.47775139 -0.1056907  
## [82]  0.33789474 -0.45268679 -0.94185101 -1.55154566  1.01961483  0.36500751  0.63929872  0.8107302  
## [91] -0.86738960 -0.92409066  1.51547209 -0.42188791 -1.95797775 -0.71004799  0.76039718 -1.3605695  
## [100] 1.10161612
```

# Indexing: []

We can also use [] to index objects that we create in R.

```
a <- 11:20  
a
```

```
## [1] 11 12 13 14 15 16 17 18 19 20
```

```
a[4] ## Get the 4th element of object "a"
```

```
## [1] 14
```

```
a[c(4, 6)] ## Get the 4th and 6th elements
```

```
## [1] 14 16
```

# Indexing: []

It also works on larger arrays (vectors, matrices, data frames, and lists). For example:

```
starwars[1, 1] ## Show the cell corresponding to the 1st row & 1st column of the data frame.
```

```
## # A tibble: 1 x 1
##   name
##   <chr>
## 1 Luke Skywalker
```

# Indexing: []

It also works on larger arrays (vectors, matrices, data frames, and lists). For example:

```
starwars[1, 1] ## Show the cell corresponding to the 1st row & 1st column of the data frame.
```

```
## # A tibble: 1 x 1
##   name
##   <chr>
## 1 Luke Skywalker
```

What does `starwars[1:3, 1]` give you?

## Indexing: []

We haven't covered them properly yet (patience), but **lists** are a more complex type of array object in R

## Indexing: []

We haven't covered them properly yet (patience), but **lists** are a more complex type of array object in R

They can contain a random assortment of objects that don't share the same characteristics

# Indexing: []

We haven't covered them properly yet (patience), but **lists** are a more complex type of array object in R

They can contain a random assortment of objects that don't share the same characteristics

- e.g. a list can contain a scalar, a string, and a data frame, or even another list

# Indexing: []

The relevance to indexing is that lists require two square brackets [[]] to index the parent list item and then the standard [] within that parent item:

```
my_list <- list(  
  a = "hello",  
  b = c(1,2,3),  
  c = data.frame(x = 1:5, y = 6:10)  
)  
my_list[[1]] ## Return the 1st list object
```

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

```
my_list[[2]][3] ## Return the 3rd element of the 2nd list object
```

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

# Indexing: \$

Lists provide a nice segue to our other indexing operator: \$

- Let's continue with the my\_list example from the previous slide.

```
my_list
```

```
## $a
## [1] "hello"
##
## $b
## [1] 1 2 3
##
## $c
##   x   y
## 1 1   6
## 2 2   7
## 3 3   8
## 4 4   9
## 5 5  10
```

# Indexing: \$

Lists provide a nice segue to our other indexing operator: \$.

- Let's continue with the `my_list` example

```
my_list
```

```
## $a
## [1] "hello"
##
## $b
## [1] 1 2 3
##
## $c
##   x  y
## 1 1  6
## 2 2  7
## 3 3  8
## 4 4  9
## 5 5 10
```

# Indexing: \$

We can call these objects directly by name using the dollar sign, e.g.

```
my_list$a ## Return list object "a"
```

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

```
my_list$b[3] ## Return the 3rd element of list object "b"
```

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

```
my_list$c$x ## Return column "x" of list object "c"
```

```
## [1] 1 2 3 4 5
```

# Indexing: \$

The `$` form of indexing also works for other object types

In some cases, you can also combine the two index options:

```
starwars$name[1] # first element of the name column of the starwars data frame
```

```
## [1] "Luke Skywalker"
```

# Indexing: \$

However, note some key differences between the output from this example and that of our previous `starwars[1, 1]` example:

```
starwars$name[1]
```

```
## [1] "Luke Skywalker"
```

```
starwars[1, 1]
```

```
## # A tibble: 1 × 1
##   name
##   <chr>
## 1 Luke Skywalker
```

# Removing objects

Use `rm()` to remove an object or objects from your working environment.

```
a <- "hello"  
b <- "world"  
rm(a, b)
```

You can also use `rm(list = ls())` to remove all objects in your working environment (except packages), or just start a new R session instead

# The tidyverse

---

# What is "tidy" data?

What we are going to learn is how to use a set of packages called the **tidyverse**

# What is "tidy" data?

What we are going to learn is how to use a set of packages called the **tidyverse**

These sets of packages make working with data **extremely easy** and **intuitive**

# What is "tidy" data?

## Resources:

- [Vignette](#) (from the `tidyr` package)
- [Original paper](#) (Hadley Wickham, 2014 JSS)

# What is "tidy" data?

Resources:

- [Vignette](#) (from the `tidyr` package)
- [Original paper](#) (Hadley Wickham, 2014 JSS)

Key points:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

# What is "tidy" data?

Resources:

- [Vignette](#) (from the `tidyr` package)
- [Original paper](#) (Hadley Wickham, 2014 JSS)

Key points:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

Basically, tidy data is more likely to be [long \(i.e. narrow\)](#) than wide

# Checklist

Install tidyverse: `install.packages('tidyverse')` (already done on cloud)

Install nycflights13: `install.packages('nycflights13', repos = 'https://cran.rstudio.com')`

# Tidyverse vs. base R

Lots of debate over tidyverse vs base R

# Tidyverse vs. base R

Lots of debate over tidyverse vs base R

The answer is **obvious**: We should teach the tidyverse first

- Good documentation and support
- Consistent philosophy and syntax
- Nice front-end for big data tools
- For data cleaning, plotting, the tidyverse is elite

# Tidyverse vs. base R

Base R is still great

- Base R is extremely flexible and powerful
- The tidyverse can't do everything
- Using base R and the tidyverse together is often a good idea

# Tidyverse vs. base R

One point of convenience is that there is often a direct correspondence between a tidyverse command and its base R equivalent:

tidyverse	base
?readr::read_csv	?utils::read.csv
?dplyr::if_else	?base::ifelse
?tibble::tibble	?base :: data.frame

Tidyverse functions typically have extra features on top of base R

# Tidyverse vs. base R

One point of convenience is that there is often a direct correspondence between a tidyverse command and its base R equivalent:

tidyverse	base
?readr::read_csv	?utils::read.csv
?dplyr::if_else	?base::ifelse
?tibble::tibble	?base :: data.frame

Tidyverse functions typically have extra features on top of base R

There are always many ways to achieve a single goal in R

# Tidyverse packages

Let's load the tidyverse meta-package and check the output.

```
library(tidyverse)
```

# Tidyverse packages

Let's load the tidyverse meta-package and check the output.

```
library(tidyverse)
```

We have actually loaded a number of packages: **ggplot2**, **tibble**, **dplyr**, etc

# Tidyverse packages

Let's load the tidyverse meta-package and check the output.

```
library(tidyverse)
```

We have actually loaded a number of packages: **ggplot2**, **tibble**, **dplyr**, etc

We can also see information about the package versions and some  
**namespace conflicts**

# Tidyverse packages

The tidyverse actually comes with a lot more packages than those that are just loaded automatically

```
tidyverse_packages()
```

```
## [1] "broom"        "cli"          "crayon"        "dbplyr"        "dplyr"         "forcats"       "ggplot2"       "hav
## [9] "hms"          "httr"          "jsonlite"      "lubridate"     "magrittr"      "modelr"        "pillar"        "pur
## [17] "readr"         "readxl"        "reprex"        "rlang"         "rstudioapi"   "rvest"         "stringr"       "tib
## [25] "tidyverse"     "xml2"
```

e.g. the **lubridate** package is for working with dates and the **rvest** package is for webscraping

# Tidyverse packages

The tidyverse actually comes with a lot more packages than those that are just loaded automatically

```
tidyverse_packages()
```

```
## [1] "broom"        "cli"          "crayon"        "dbplyr"        "dplyr"         "forcats"       "ggplot2"       "haven"        "hms"          "httr"          "jsonlite"      "lubridate"     "magrittr"      "modelr"        "pillar"        "purrr"         "readr"         "readxl"        "reprex"        "rlang"         "rstudioapi"    "rvest"         "stringr"       "tibble"        "tidyverse"     "xml2"
```

e.g. the **lubridate** package is for working with dates and the **rvest** package is for webscraping

These packages have to be loaded separately

# Tidyverse packages

We're going to focus on two workhorse packages:

1. **dplyr**
2. **tidyr**

These are the packages for cleaning and wrangling data

# Tidyverse packages

We're going to focus on two workhorse packages:

1. **dplyr**
2. **tidyr**

These are the packages for cleaning and wrangling data

They are thus the ones that you will likely make the most use of

# Tidyverse packages

We're going to focus on two workhorse packages:

1. **dplyr**
2. **tidyr**

These are the packages for cleaning and wrangling data

They are thus the ones that you will likely make the most use of

Data cleaning and wrangling is important and knowing how to do it well is a good skill for any data-oriented job

# Pipes: %>%

The pipe operator `%>%` lets us perform a sequence of operations in a very nice and tidy way

# Pipes: %>%

The pipe operator `%>%` lets us perform a sequence of operations in a very nice and tidy way

Let's consider a fake example to get the idea for why its really beneficial

# Pipes: %>%

Let's say we wanted to apply a sequence of operations that tells the computer what you did throughout your day:

1. Wake up
2. Get out of bed
3. Comb hair
4. Go downstairs
5. Drink a cup
6. Grab hat
7. Catch bus

# Pipes: %>%

If you were to code this up in a traditional way it might look one of two ways:

A bunch of lines doing all the different steps

```
me <- wake_up(me)
me <- get_out_of_bed(me)
me <- comb_hair(me)
me <- go(me, "downstairs")
me <- drink(me, "cup")
me <- grab(me, "hat")
me <- catch(me, "bus")
```

# Pipes: %>%

If you were to code this up in a traditional way it might look one of two ways:

Or if you're a little crazy then do it all in one line

```
me ← catch(grab(drink(go(comb_hair(get_out_of_bed(wake_up(me)))), where = "downstairs"), what = ')
```

These are kind of tedious or messy and out of the order you'd think

# Pipes: %>%

With pipes we can do everything at once, but have it be **in order**:

```
me <- me %>%
  wake_up() %>%
  get_out_of_bed() %>%
  comb_hair() %>%
  go("downstairs") %>%
  drink("cup") %>%
  grab("hat") %>%
  catch("bus")
```

This makes everything **very intuitive** to read and code!

# Pipes: %>%

Here's a real example: suppose we wanted to figure out the average highway miles per gallon of Audi's in the `mpg` dataset:

```
mpg
```

```
## # A tibble: 234 x 11
##   manufacturer model      displ  year   cyl trans   drv   cty   hwy fl class
##   <chr>        <chr>     <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
## 1 audi         a4          1.8  1999     4 auto(l5) f       18    29  p    compact
## 2 audi         a4          1.8  1999     4 manual(m5) f      21    29  p    compact
## 3 audi         a4          2    2008     4 manual(m6) f      20    31  p    compact
## 4 audi         a4          2    2008     4 auto(av)   f      21    30  p    compact
## 5 audi         a4          2.8  1999     6 auto(l5) f      16    26  p    compact
## 6 audi         a4          2.8  1999     6 manual(m5) f      18    26  p    compact
## 7 audi         a4          3.1  2008     6 auto(av)  f      18    27  p    compact
## 8 audi         a4 quattro  1.8  1999     4 manual(m5) 4     18    26  p    compact
## 9 audi         a4 quattro  1.8  1999     4 auto(l5)  4     16    25  p    compact
## 10 audi        a4 quattro  2    2008     4 manual(m6) 4    20    28  p    compact
## # ... with 224 more rows
```

# Pipes: %>%

There's two ways you might do this without taking advantage of pipes:

# Pipes: %>%

There's two ways you might do this without taking advantage of pipes:

The first is to do it step-by-step, line-by-line which requires a lot of variable assignment

```
audis_mpg <- filter(mpg, manufacturer == "audi")
audis_mpg_grouped <- group_by(filter(mpg, manufacturer == "audi"), model)
summarise(audis_mpg_grouped, hwy_mean = mean(hwy))
```

```
## # A tibble: 3 x 2
##   model      hwy_mean
##   <chr>       <dbl>
## 1 a4          28.3
## 2 a4 quattro  25.8
## 3 a6 quattro  24
```

# Pipes: %>%

Next you could do it all in one line which is hard to read

```
summarise(group_by(filter(mpg, manufacturer="audi")), model), hwy_mean = mean(hwy))
```

```
## # A tibble: 3 x 2
##   model      hwy_mean
##   <chr>       <dbl>
## 1 a4          28.3
## 2 a4 quattro  25.8
## 3 a6 quattro  24
```

# Pipes: %>%

Or, you could use **pipes** %>%:

```
mpg %>% filter(manufacturer="audi") %>% group_by(model) %>% summarise(hwy_mean = mean(hwy))
```

```
## # A tibble: 3 x 2
##   model      hwy_mean
##   <chr>       <dbl>
## 1 a4          28.3
## 2 a4 quattro  25.8
## 3 a6 quattro  24
```

# Pipes: %>%

Or, you could use **pipes** %>%:

```
mpg %>% filter(manufacturer="audi") %>% group_by(model) %>% summarise(hwy_mean = mean(hwy))
```

```
## # A tibble: 3 x 2
##   model     hwy_mean
##   <chr>      <dbl>
## 1 a4          28.3
## 2 a4 quattro  25.8
## 3 a6 quattro  24
```

It performs the operations from left to right, exactly like you'd think of them:  
take this object (mpg), do this (filter), then do this (group by car model), then  
do this (take the mean of highway miles)

# Use vertical space

Pipes are even more readable if we write it over several lines:

```
mpg %>%
  filter(manufacturer=="audi") %>%
  group_by(model) %>%
  summarise(hwy_mean = mean(hwy))
```

```
## # A tibble: 3 x 2
##   model      hwy_mean
##   <chr>        <dbl>
## 1 a4            28.3
## 2 a4 quattro    25.8
## 3 a6 quattro    24
```

Using vertical space costs nothing and makes for much more readable code

# dplyr

---

# Aside: dplyr 1.0.0 release

Please make sure that you are running at least **dplyr** 1.0.0 before continuing.

```
packageVersion("dplyr")
```

```
## [1] '1.0.2'
```

```
# install.packages('dplyr') ## install updated version if < 1.0.0
```

# The five key dplyr verbs

1. `filter`: Subset/filter rows based on their values
2. `arrange`: Reorder/arrange rows based on their values
3. `select`: Select columns/variables
4. `mutate`: Create new columns/variables
5. `summarise`: Collapse multiple rows into a single summary value, potentially by a grouping variable

# The five key dplyr verbs

1. `filter`: Subset/filter rows based on their values
2. `arrange`: Reorder/arrange rows based on their values
3. `select`: Select columns/variables
4. `mutate`: Create new columns/variables
5. `summarise`: Collapse multiple rows into a single summary value, potentially by a grouping variable

Let's practice these commands together using the `starwars` data frame that comes pre-packaged with `dplyr`

# Starwars

Here's the `starwars` dataset, it has 87 observations of 14 variables

```
starwars
```

```
## # A tibble: 87 x 14
##   name    height  mass hair_color skin_color eye_color birth_year sex gender homeworld species film
##   <chr>     <int> <dbl> <chr>       <chr>       <chr>        <dbl> <chr> <chr>       <chr>       <chr>
## 1 Luke...     172     77 blond      fair        blue          19 male   masculin... Tatooine Human  <chr>
## 2 C-3PO       167     75 <NA>       gold        yellow        112 none   masculin... Tatooine Droid   <chr>
## 3 R2-D2       96      32 <NA>       white, bl... red           33 none   masculin... Naboo   Droid   <chr>
## 4 Dart...      202    136 none      white        yellow        41.9 male   masculin... Tatooine Human  <chr>
## 5 Leia...      150     49 brown      light       brown          19 femal... feminin... Alderaan Human  <chr>
## 6 Owen...      178    120 brown, gr... light       blue           52 male   masculin... Tatooine Human  <chr>
## 7 Beru...      165     75 brown      light       blue           47 femal... feminin... Tatooine Human  <chr>
## 8 R5-D4       97      32 <NA>       white, red red            NA none   masculin... Tatooine Droid   <chr>
## 9 Bigg...      183     84 black      light       brown          24 male   masculin... Tatooine Human  <chr>
## 10 Obi-...      182     77 auburn, w... fair        blue-gray        57 male   masculin... Stewjon Human  <chr>
## # ... with 77 more rows, and 1 more variable: starships <list>
```

# 1) dplyr::filter

Why filter?

# 1) dplyr::filter

Why filter?

Filtering subsets your data

# 1) dplyr::filter

Why filter?

Filtering subsets your data

This means that you can take out data that meet certain characteristics

# 1) dplyr::filter

Why filter?

Filtering subsets your data

This means that you can take out data that meet certain characteristics

e.g. if you want to run your analysis only on low income areas, or if you want to focus on years after 2005

# 1) dplyr::filter

Here we are subsetting the observations of humans that are at least 190cm

```
starwars %>%  
  filter(  
    species = "Human",  
    height >= 190  
)
```

```
## # A tibble: 4 x 14  
##   name  height  mass hair_color skin_color eye_color birth_year sex  gender homeworld species films  
##   <chr>   <int> <dbl> <chr>       <chr>       <chr>          <dbl> <chr> <chr> <chr>       <chr> <list>  
## 1 Dart...     202    136 none        white       yellow         41.9 male  masculin... Tatooine Human <chr...  
## 2 Qui-...     193     89 brown       fair        blue           92   male  masculin... <NA>   Human <chr...  
## 3 Dooku      193     80 white       fair        brown          102  male  masculin... Serenno Human <chr...  
## 4 Bail...     191     NA black       tan         brown          67   male  masculin... Alderaan Human <chr...  
## # ... with 1 more variable: starships <list>
```

# 1) dplyr::filter

You can filter using regular expressions with grep-type commands or the `stringr` package

```
starwars %>%  
  filter(stringr::str_detect(name, "Skywalker"))
```

```
## # A tibble: 3 x 14  
##   name    height   mass hair_color skin_color eye_color birth_year sex gender homeworld species films  
##   <chr>     <int>   <dbl> <chr>       <chr>       <chr>      <dbl> <chr> <chr> <chr>       <chr> <list>  
## 1 Luke...     172     77 blond      fair        blue         19 male  masculin... Tatooine Human <chr...  
## 2 Anak...     188     84 blond      fair        blue        41.9 male  masculin... Tatooine Human <chr...  
## 3 Shmi...     163     NA black      fair        brown        72 female feminin... Tatooine Human <chr...  
## # ... with 1 more variable: starships <list>
```

This subsets the observations for individuals whose names contain "Skywalker"

# 1) dplyr::filter

A very common `filter` use case is identifying/removing missing data cases:

```
starwars %>%  
  filter(is.na(height))
```

```
## # A tibble: 6 x 14  
##   name  height  mass hair_color skin_color eye_color birth_year sex  gender homeworld species films  
##   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>    <chr>    <chr>  <list>  
## 1 Arve...     NA     NA brown     fair       brown           NA male  mascul... <NA>    Human  <chr...  
## 2 Finn       NA     NA black     dark       dark            NA male  mascul... <NA>    Human  <chr...  
## 3 Rey        NA     NA brown     light      hazel           NA fema... femin... <NA>    Human  <chr...  
## 4 Poe ...     NA     NA brown     light      brown           NA male  mascul... <NA>    Human  <chr...  
## 5 BB8        NA     NA none      none      black           NA none  mascul... <NA>    Droid  <chr...  
## 6 Capt...     NA     NA unknown  unknown    unknown          NA <NA>  <NA>    <NA>    <NA>  <chr...  
## # ... with 1 more variable: starships <list>
```

# 1) dplyr::filter

To remove missing observations, use negation:

```
starwars %>%  
  filter(!is.na(height))
```

```
## # A tibble: 81 x 14  
##   name    height  mass hair_color skin_color eye_color birth_year sex gender homeworld species film  
##   <chr>     <int> <dbl> <chr>       <chr>       <chr>        <dbl> <chr> <chr>       <chr>       <chr>       <list>  
## 1 Luke...     172     77 blond      fair        blue          19 male  masculin… Tatooine Human <chr>  
## 2 C-3PO      167     75 <NA>       gold        yellow        112 none  masculin… Tatooine Droid <chr>  
## 3 R2-D2       96      32 <NA>      white, bl… red           33 none  masculin… Naboo  Droid <chr>  
## 4 Dart...     202    136 none      white        yellow        41.9 male  masculin… Tatooine Human <chr>  
## 5 Leia...     150      49 brown      light       brown          19 fema… feminin… Alderaan Human <chr>  
## 6 Owen...     178     120 brown, gr… light       blue           52 male  masculin… Tatooine Human <chr>  
## 7 Beru...     165      75 brown      light       blue           47 fema… feminin… Tatooine Human <chr>  
## 8 R5-D4       97      32 <NA>      white, red red            NA none  masculin… Tatooine Droid <chr>  
## 9 Bigg...     183      84 black      light       brown          24 male  masculin… Tatooine Human <chr>  
## 10 Obi-…     182      77 auburn, w… fair        blue-gray        57 male  masculin… Stewjon Human <chr>  
## # ... with 71 more rows, and 1 more variable: starships <list>
```

## 2) dplyr::arrange

`arrange` sorts the data frame based on the variables you supply:

```
starwars %>%  
  arrange(birth_year)
```

```
## # A tibble: 87 x 14  
##   name    height  mass hair_color skin_color eye_color birth_year sex gender homeworld species film  
##   <chr>     <int> <dbl> <chr>       <chr>       <chr>        <dbl> <chr> <chr>       <chr>       <chr>      <list>  
## 1 Wick...     88    20  brown      brown       brown          8   male  masculin... Endor      Ewok      <chr>  
## 2 IG-88     200   140 none       metal       red            15  none  masculin... <NA>       Droid      <chr>  
## 3 Luke...    172    77  blond      fair        blue           19  male  masculin... Tatooine   Human     <chr>  
## 4 Leia...    150    49  brown      light       brown          19  femal... feminin... Alderaan   Human     <chr>  
## 5 Wedg...    170    77  brown      fair        hazel          21  male  masculin... Corellia   Human     <chr>  
## 6 Plo ...    188    80  none       orange     black           22  male  masculin... Dorin      Kel Dor    <chr>  
## 7 Bigg...    183    84  black      light       brown          24  male  masculin... Tatooine   Human     <chr>  
## 8 Han ...    180    80  brown      fair        brown          29  male  masculin... Corellia   Human     <chr>  
## 9 Land...    177    79  black      dark        brown          31  male  masculin... Socorro    Human     <chr>  
## 10 Boba...   183    78.2 black     fair        brown         31.5 male  masculin... Kamino    Human     <chr>  
## # ... with 77 more rows, and 1 more variable: starships <list>
```

# 1) dplyr::arrange

Why arrange?

# 1) dplyr::arrange

Why arrange?

Arrange sorts your data

# 1) dplyr::arrange

Why arrange?

Arrange sorts your data

This makes it easy to check and see patterns in the data

## 2) dplyr::arrange

We can also arrange items in descending order using `arrange(desc())`

```
starwars %>%  
  arrange(desc(birth_year))
```

```
## # A tibble: 87 x 14  
##   name    height  mass hair_color skin_color eye_color birth_year sex gender homeworld species film  
##   <chr>     <int> <dbl> <chr>       <chr>       <chr>        <dbl> <chr> <chr>       <chr>       <chr>      <list>  
## 1 Yoda       66     17  white       green       brown          896 male  masculin <NA>       Yoda's... <chr>  
## 2 Jabba...    175    1358 <NA>       green-tan... orange          600 herm... masculin Nal Hutta Hutt      <chr>  
## 3 Chew...     228    112  brown       unknown      blue           200 male  masculin Kashyyyk Wookiee <chr>  
## 4 C-3PO      167     75 <NA>       gold        yellow         112 none  masculin Tatooine Droid     <chr>  
## 5 Dooku      193     80  white       fair        brown          102 male  masculin Serenno Human    <chr>  
## 6 Qui-...     193     89  brown       fair        blue            92 male  masculin <NA>       Human    <chr>  
## 7 Ki-A...     198     82  white       pale        yellow         92 male  masculin Cerea     Cerean    <chr>  
## 8 Fini...     170     NA  blond       fair        blue           91 male  masculin Coruscant Human    <chr>  
## 9 Palp...     170     75  grey        pale        yellow         82 male  masculin Naboo     Human    <chr>  
## 10 Clie...    183     NA  brown       fair        blue           82 male  masculin Tatooine Human    <chr>  
## # ... with 77 more rows, and 1 more variable: starships <list>
```

### 3) dplyr::select

Why select?

### 3) dplyr::select

Why select?

Select lets you choose columns to keep or drop

### 3) dplyr::select

Why select?

Select lets you choose columns to keep or drop

This means you are essentially getting rid of variables, likely ones you do not need

### 3) dplyr::select

Why select?

Select lets you choose columns to keep or drop

This means you are essentially getting rid of variables, likely ones you do not need

e.g. if you used an emissions rate and marginal damage per unit of pollution variable to construct the marginal damage of output, you may not need the first two variables any more

### 3) dplyr::select

Use commas to select multiple columns out of a data frame, deselect a column with "-", select across multiple columns with "first:last":

```
starwars %>%  
  select(name:skin_color, species, -height)
```

```
## # A tibble: 87 x 5  
##   name           mass hair_color   skin_color species  
##   <chr>        <dbl> <chr>       <chr>      <chr>  
## 1 Luke Skywalker     77  blond       fair       Human  
## 2 C-3PO              75  <NA>        gold       Droid  
## 3 R2-D2              32  <NA>        white, blue Droid  
## 4 Darth Vader        136 none         white      Human  
## 5 Leia Organa         49  brown        light      Human  
## 6 Owen Lars          120 brown, grey  light      Human  
## 7 Beru Whitesun lars  75  brown        light      Human  
## 8 R5-D4              32  <NA>        white, red Droid  
## 9 Biggs Darklighter   84  black        light      Human  
## 10 Obi-Wan Kenobi    77  auburn, white fair      Human
```

### 3) dplyr::select

You can also rename your selected variables in place

```
starwars %>%  
  select(alias = name, crib = homeworld)
```

```
## # A tibble: 87 x 2  
##   alias      crib  
##   <chr>     <chr>  
## 1 Luke Skywalker Tatooine  
## 2 C-3PO        Tatooine  
## 3 R2-D2        Naboo  
## 4 Darth Vader Tatooine  
## 5 Leia Organa Alderaan  
## 6 Owen Lars   Tatooine  
## 7 Beru Whitesun lars Tatooine  
## 8 R5-D4        Tatooine  
## 9 Biggs Darklighter Tatooine  
## 10 Obi-Wan Kenobi Stewjon  
## # ... with 77 more rows
```

### 3) dplyr::select

If you just want to rename columns without subsetting them, you can use

rename:

```
starwars %>%  
  rename(alias = name, crib = homeworld)
```

```
## # A tibble: 87 x 14  
##   alias height mass hair_color skin_color eye_color birth_year sex gender crib species films ve  
##   <chr>    <int> <dbl> <chr>       <chr>       <chr>        <dbl> <chr> <chr> <chr> <chr> <lis> <l  
## 1 Luke...     172     77 blond      fair        blue          19 male  masculin Tato... Human <chr... <c  
## 2 C-3PO       167     75 <NA>       gold        yellow        112 none  masculin Tato... Droid <chr... <c  
## 3 R2-D2        96      32 <NA>       white, bl... red           33 none  masculin Naboo Droid <chr... <c  
## 4 Dart...      202     136 none       white        yellow        41.9 male  masculin Tato... Human <chr... <c  
## 5 Leia...      150      49 brown      light        brown         19 female feminin Alder... Human <chr... <c  
## 6 Owen...      178     120 brown, gr... light        blue          52 male  masculin Tato... Human <chr... <c  
## 7 Beru...      165      75 brown      light        blue          47 female feminin Tato... Human <chr... <c  
## 8 R5-D4        97      32 <NA>       white, red red           NA none  masculin Tato... Droid <chr... <c  
## 9 Bigg...      183      84 black      light        brown         24 male  masculin Tato... Human <chr... <c  
## 10 Obi-       182      77 auburn    w fair        blue-gray       57 male  masculin Stew... Human <chr... <c  
## # ... with 77 more rows, and 1 more variable: value <dbl>
```

### 3) dplyr::select cont.

The `select(contains(PATTERN))` option provides a nice shortcut in relevant cases.

```
starwars %>%  
  select(name, contains("color"))  
  
## # A tibble: 87 x 4  
##   name           hair_color    skin_color eye_color  
##   <chr>          <chr>        <chr>      <chr>  
## 1 Luke Skywalker   blond       fair        blue  
## 2 C-3PO            <NA>        gold        yellow  
## 3 R2-D2            <NA>        white, blue red  
## 4 Darth Vader     none        white        yellow  
## 5 Leia Organa      brown       light       brown  
## 6 Owen Lars        brown, grey light       blue  
## 7 Beru Whitesun lars brown       light       blue  
## 8 R5-D4            <NA>        white, red red  
## 9 Biggs Darklighter black       light       brown  
## 10 Obi-Wan Kenobi   auburn     white, fair blue-gray
```

### 3) dplyr::select

The `select( ... , everything())` option is another useful shortcut if you only want to bring some variable(s) to the "front" of a data frame

```
starwars %>%  
  select(species, homeworld, everything()) %>%  
  head(5)
```

```
## # A tibble: 5 x 14  
##   species homeworld name   height  mass hair_color skin_color eye_color birth_year sex   gender films  
##   <chr>     <chr>    <chr>   <int>  <dbl>   <chr>       <chr>       <chr>           <dbl> <chr> <chr>   <list>  
## 1 Human     Tatooine  Luke...    172     77  blond      fair        blue            19   male  masculin... <chr...  
## 2 Droid      Tatooine  C-3PO     167     75 <NA>       gold        yellow          112  none  masculin... <chr...  
## 3 Droid      Naboo     R2-D2      96      32 <NA>      white, bl... red             33   none  masculin... <chr...  
## 4 Human     Tatooine  Dart...    202    136  none       white        yellow          41.9 male  masculin... <chr...  
## 5 Human     Alderaan  Leia...    150     49  brown      light       brown            19   fema... feminin... <chr...  
## # ... with 1 more variable: starships <list>
```

### 3) dplyr::select

You can also use `relocate` to do the same thing

```
starwars %>%  
  relocate(species, homeworld) %>%  
  head(5)
```

```
## # A tibble: 5 x 14  
##   species homeworld name   height  mass hair_color skin_color eye_color birth_year sex   gender films  
##   <chr>     <chr>    <chr>   <int> <dbl>   <chr>       <chr>       <chr>           <dbl> <chr> <chr>   <list>  
## 1 Human     Tatooine  Luke...     172     77  blond      fair        blue            19   male  masculin... <chr...  
## 2 Droid     Tatooine  C-3PO      167     75 <NA>       gold        yellow          112  none  masculin... <chr...  
## 3 Droid     Naboo     R2-D2      96      32 <NA>       white, bl... red             33   none  masculin... <chr...  
## 4 Human     Tatooine  Dart...     202    136  none       white        yellow          41.9 male  masculin... <chr...  
## 5 Human     Alderaan  Leia...     150     49  brown      light        brown            19   fema... feminin... <chr...  
## # ... with 1 more variable: starships <list>
```

## 4) dplyr::mutate

Why mutate?

## 4) dplyr::mutate

Why mutate?

You may need to create new variables

## 4) dplyr::mutate

Why mutate?

You may need to create new variables

e.g. if I give you nominal house prices and the rate of house price inflation, you need to combine these two things to make a new **real house price** variable

## 4) dplyr::mutate

You can create new columns from scratch as transformations of existing columns:

```
starwars %>%  
  select(name, birth_year) %>%  
  mutate(dog_years = birth_year * 7) %>%  
  mutate(comment = paste0(name, " is ", dog_years, " in dog years.))
```

```
## # A tibble: 87 x 4  
##   name           birth_year  dog_years comment  
##   <chr>          <dbl>      <dbl> <chr>  
## 1 Luke Skywalker     19        133  Luke Skywalker is 133 in dog years.  
## 2 C-3PO              112       784  C-3PO is 784 in dog years.  
## 3 R2-D2              33        231  R2-D2 is 231 in dog years.  
## 4 Darth Vader        41.9      293. Darth Vader is 293.3 in dog years.  
## 5 Leia Organa         19        133  Leia Organa is 133 in dog years.  
## 6 Owen Lars            52       364  Owen Lars is 364 in dog years.  
## 7 Beru Whitesun lars    47       329  Beru Whitesun lars is 329 in dog years.  
## 8 R5-D4              NA        NA   R5-D4 is NA in dog years.
```

## 4) dplyr::mutate

Note: `mutate` creates variables in order, so you can chain multiple mutates in a single call

```
starwars %>%  
  select(name, birth_year) %>%  
  mutate(  
    dog_years = birth_year * 7, ## Separate with a comma  
    comment = paste0(name, " is ", dog_years, " in dog years.")  
  )
```

```
## # A tibble: 87 x 4  
##   name      birth_year  dog_years comment  
##   <chr>        <dbl>     <dbl> <chr>  
## 1 Luke Skywalker      19       133  Luke Skywalker is 133 in dog years.  
## 2 C-3PO                 112      784  C-3PO is 784 in dog years.  
## 3 R2-D2                  33      231  R2-D2 is 231 in dog years.  
## 4 Darth Vader             41.9    293. Darth Vader is 293.3 in dog years.  
## 5 Leia Organa              19       133  Leia Organa is 133 in dog years.  
## 6 Owen Lars                52      364  Owen Lars is 364 in dog years.
```

## 4) dplyr::mutate

Boolean, logical and conditional operators all work well with `mutate` too:

```
starwars %>%
  select(name, height) %>%
  filter(name %in% c("Luke Skywalker", "Anakin Skywalker")) %>%
  mutate(tall1 = height > 180) %>% # TRUE or FALSE
  mutate(tall2 = ifelse(height > 180, "Tall", "Short")) ## Same effect, but can choose labels
```

```
## # A tibble: 2 x 4
##   name           height tall1 tall2
##   <chr>          <int> <lgl> <chr>
## 1 Luke Skywalker     172 FALSE Short
## 2 Anakin Skywalker    188 TRUE  Tall
```

## 4) dplyr::mutate

Lastly, combining `mutate` with `across` allows you to easily work on a subset of variables:

```
starwars %>%  
  select(name:eye_color) %>%  
  mutate(across(where(is.character), toupper)) %>% # Take all character variables, uppercase them  
  head(5)
```

```
## # A tibble: 5 x 6  
##   name           height  mass hair_color skin_color eye_color  
##   <chr>        <int> <dbl> <chr>      <chr>      <chr>  
## 1 LUKE SKYWALKER     172    77 BLOND      FAIR       BLUE  
## 2 C-3PO              167    75 <NA>       GOLD       YELLOW  
## 3 R2-D2               96    32 <NA>      WHITE, BLUE  RED  
## 4 DARTH VADER        202   136 NONE       WHITE      YELLOW  
## 5 LEIA ORGANA         150    49 BROWN     LIGHT      BROWN
```

## 5) dplyr::summarise

Why summarise?

## 5) dplyr::summarise

Why summarise?

Often we want to get summary statistics or *collapse* our data

## 5) dplyr::summarise

Why summarise?

Often we want to get summary statistics or *collapse* our data

e.g. if I gave you a data set of each individual's marginal damage in the US,  
we may want to aggregate up to county-level marginal damages

# 5) dplyr::summarise

Summarising useful in combination with the `group_by` command

```
starwars %>%  
  group_by(species, gender) %>% # for each species-gender combo  
  summarise(mean_height = mean(height, na.rm = TRUE)) # calculate the mean height
```

```
## # A tibble: 42 x 3  
## # Groups:   species [38]  
##   species   gender   mean_height  
##   <chr>     <chr>       <dbl>  
## 1 Aleena    masculine      79  
## 2 Besalisk  masculine     198  
## 3 Cerean    masculine     198  
## 4 Chagrian  masculine     196  
## 5 Clawdite  feminine      168  
## 6 Droid     feminine      96  
## 7 Droid     masculine     140  
## 8 Dug       masculine     112  
## 9 Ewok      masculine     88  
## 10 Geonosian masculine     182
```

## 5) dplyr::summarise

Note that including "na.rm = TRUE" is usually a good idea with summarise functions, it keeps NAs from propagating to the end result

```
## Probably not what we want
starwars %>%
  summarise(mean_height = mean(height))
```

```
## # A tibble: 1 x 1
##   mean_height
##       <dbl>
## 1        NA
```

## 5) dplyr::summarise

We can also use `across` within `summarise`:

```
starwars %>%  
  group_by(species) %>% # for each species  
  summarise(across(where(is.numeric), mean, na.rm = T)) %>% # take the mean of all numeric varian  
  head(5)
```

```
## # A tibble: 5 x 4  
##   species    height   mass birth_year  
##   <chr>      <dbl>   <dbl>     <dbl>  
## 1 Aleena       79     15       NaN  
## 2 Besalisk     198    102       NaN  
## 3 Cerean       198     82       92  
## 4 Chagrian     196     NaN       NaN  
## 5 Clawdite     168     55       NaN
```

# Other dplyr goodies

`group_by` and `ungroup`: For (un)grouping

- Particularly useful with the `summarise` and `mutate` commands

# Other dplyr goodies

`group_by` and `ungroup`: For (un)grouping

- Particularly useful with the `summarise` and `mutate` commands

`slice`: Subset rows by position rather than filtering by values

- E.g. `starwars %>% slice(c(1, 5))`

# Other dplyr goodies

`pull`: Extract a column from as a data frame as a vector or scalar

- E.g. `starwars %>% filter(gender=="female") %>% pull(height)`

# Other dplyr goodies

`pull`: Extract a column from as a data frame as a vector or scalar

- E.g. `starwars %>% filter(gender=="female") %>% pull(height)`

`count` and `distinct`: Number and isolate unique observations

- E.g. `starwars %>% count(species)`, or `starwars %>% distinct(species)`

# Other dplyr goodies

There are also a whole class of **window functions** for getting leads and lags, percentiles, cumulative sums, etc.

- See `vignette("window-functions")`.

## dplyr::xxxx\_join

The last set of commands we need are the `join` commands

## dplyr::xxxx\_join

The last set of commands we need are the `join` commands

These are the same as `merge` in stata but with a bit more functionality

## dplyr::xxxx\_join

The last set of commands we need are the `join` commands

These are the same as `merge` in stata but with a bit more functionality

Why join?

## dplyr::xxxx\_join

The last set of commands we need are the `join` commands

These are the same as `merge` in stata but with a bit more functionality

Why join?

Suppose we want to understand how pollution affects housing prices

## dplyr::xxxx\_join

The last set of commands we need are the `join` commands

These are the same as `merge` in stata but with a bit more functionality

Why join?

Suppose we want to understand how pollution affects housing prices

You may need to combine data sets: one data sets on house prices in all counties, and another data set on pollution levels in all counties

# dplyr::xxxx\_join

How does joining work?

## dplyr::xxxx\_join

How does joining work?

We need two datasets (e.g. housing prices and pollution)

# dplyr::xxxx\_join

How does joining work?

We need two datasets (e.g. housing prices and pollution)

Each dataset has the same set of **key** variables (e.g. county)

# dplyr::xxxx\_join

How does joining work?

We need two datasets (e.g. housing prices and pollution)

Each dataset has the same set of **key** variables (e.g. county)

When we join, we match up the two datasets on the key, and combine them into one

# dplyr::xxxx\_join

How does joining work?

We need two datasets (e.g. housing prices and pollution)

Each dataset has the same set of **key** variables (e.g. county)

When we join, we match up the two datasets on the key, and combine them into one

In our housing example, each row of the joined dataset would now tell us the house price, the county its in, and the county's pollution

# dplyr::xxxx\_join

We merge data with **join operations**:

- `inner_join(df1, df2)`
- `left_join(df1, df2)`
- `right_join(df1, df2)`
- `full_join(df1, df2)`

(You can visualize the operations [here](#))

# dplyr::xxxx\_join

Lets use the data that comes with the the [nycflights13](#) package.

```
library(nycflights13)
```

```
flights
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier fli
##   <int> <int> <int>     <int>           <int>     <dbl>     <int>           <int>     <dbl> <chr>   <i
## 1 2013     1     1      517             515       2     830            819      11 UA    1
## 2 2013     1     1      533             529       4     850            830      20 UA    1
## 3 2013     1     1      542             540       2     923            850      33 AA    1
## 4 2013     1     1      544             545      -1    1004            1022     -18 B6    1
## 5 2013     1     1      554             600      -6    812            837     -25 DL    1
## 6 2013     1     1      554             558      -4    740            728      12 UA    1
## 7 2013     1     1      555             600      -5    913            854      19 B6    1
## 8 2013     1     1      557             600      -3    709            723     -14 EV    5
## 9 2013     1     1      557             600      -3    838            846      -8 B6    1
## 10 2013    1     1      558             600      -2    753            745      8 AA    1
## # ... with 336,766 more rows, and 7 more variables: origin <chr>, dest <chr>, air_time <dbl>, 104st162ce
## #   hour <dbl>, minute <dbl>, time_hour <dbl>,
```

# dplyr::xxxx\_join

planes

```
## # A tibble: 3,322 x 9
##   tailnum  year type          manufacturer    model engines seats speed engine
##   <chr>    <int> <chr>        <chr>           <chr>    <int> <int> <int> <chr>
## 1 N10156  2004 Fixed wing multi engine EMBRAER     EMB-145XR  2      55   NA Turbo-fan
## 2 N102UW   1998 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  2     182   NA Turbo-fan
## 3 N103US   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  2     182   NA Turbo-fan
## 4 N104UW   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  2     182   NA Turbo-fan
## 5 N10575   2002 Fixed wing multi engine EMBRAER     EMB-145LR   2      55   NA Turbo-fan
## 6 N105UW   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  2     182   NA Turbo-fan
## 7 N107US   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  2     182   NA Turbo-fan
## 8 N108UW   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  2     182   NA Turbo-fan
## 9 N109UW   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  2     182   NA Turbo-fan
## 10 N110UW  1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214  2     182   NA Turbo-fan
## # ... with 3,312 more rows
```

# Joining operations

Let's perform a left join on the flights and planes datasets

- Note: I'm going to subset columns after the join, but only to keep text on the slide

# Joining operations

Let's perform a left join on the flights and planes datasets

- Note: I'm going subset columns after the join, but only to keep text on the slide

```
left_join(flights, planes) %>%  
  select(year, month, day, dep_time, arr_time, carrier, flight, tailnum, type, model)
```

```
## Joining, by = c("year", "tailnum")
```

```
## # A tibble: 336,776 x 10  
##   year month   day dep_time arr_time carrier flight tailnum type  model  
##   <int> <int> <int>    <int>    <int> <chr>    <int> <chr>  <chr> <chr>  
## 1 2013     1     1      517      830  UA        1545 N14228 <NA> <NA>  
## 2 2013     1     1      533      850  UA        1714 N24211 <NA> <NA>  
## 3 2013     1     1      542      923  AA        1141 N619AA <NA> <NA>  
## 4 2013     1     1      544     1004  B6        725  N804JB <NA> <NA>  
## 5 2013     1     1      554      812  DL        461  N668DN <NA> <NA>
```

# Joining operations

Note that dplyr made a reasonable guess about which columns to join on (i.e. columns that share the same name), and told us what it chose

```
## Joining, by = c("year", "tailnum")
```

There's an obvious problem here: the variable `year` does not have a consistent meaning across our joining datasets

# Joining operations

Note that dplyr made a reasonable guess about which columns to join on (i.e. columns that share the same name), and told us what it chose

```
## Joining, by = c("year", "tailnum")
```

There's an obvious problem here: the variable `year` does not have a consistent meaning across our joining datasets

In one it refers to the *year of flight*, in the other it refers to *year of construction*

Luckily, there's an easy way to avoid this problem: try `?dplyr :: join`

# Joining operations

You just need to be more explicit in your join call by using the `by =` argument

```
left_join(  
  flights,  
  planes %>% rename(year_built = year), ## Not necessary w/ below line, but helpful  
  by = "tailnum" ## Be specific about the joining column  
 ) %>%  
 select(year, month, day, dep_time, arr_time, carrier, flight, tailnum, year_built, type, model)  
 head(3) ## Just to save vertical space on the slide
```

```
## # A tibble: 3 x 11  
##   year month   day dep_time arr_time carrier flight tailnum year_built type  
##   <int> <int> <int>    <int>    <int> <chr>    <int> <chr>     <int> <chr>  
## 1  2013     1     1      517      830  UA        1545 N14228    1999 Fixed wing multi engine 737-  
## 2  2013     1     1      533      850  UA        1714 N24211    1998 Fixed wing multi engine 737-  
## 3  2013     1     1      542      923  AA        1141 N619AA    1990 Fixed wing multi engine 757-
```

# Joining operations

Note what happens if we again specify the join column but don't rename the ambiguous year:

```
left_join(flights,
  planes, ## Not renaming "year" to "year_built" this time
  by = "tailnum") %>%
  select(contains("year"), month, day, dep_time, arr_time, carrier, flight, tailnum, type, model)
  head(3)

## # A tibble: 3 x 11
##   year.x year.y month   day dep_time arr_time carrier flight tailnum type
##   <int>  <int> <int> <int>    <int>    <int> <chr>    <int> <chr>   <chr>
## 1  2013    1999     1     1      517      830  UA        1545 N14228 Fixed wing multi engine 737-824
## 2  2013    1998     1     1      533      850  UA        1714 N24211 Fixed wing multi engine 737-824
## 3  2013    1990     1     1      542      923  AA        1141 N619AA Fixed wing multi engine 757-223
```

# Joining operations

Note what happens if we again specify the join column but don't rename the ambiguous year:

```
left_join(flights,
  planes, ## Not renaming "year" to "year_built" this time
  by = "tailnum") %>%
  select(contains("year"), month, day, dep_time, arr_time, carrier, flight, tailnum, type, model)
  head(3)

## # A tibble: 3 x 11
##   year.x year.y month   day dep_time arr_time carrier flight tailnum type
##   <int>  <int> <int> <int>    <int>    <int> <chr>    <int> <chr>   <chr>
## 1  2013    1999     1     1      517      830  UA        1545 N14228 Fixed wing multi engine 737-824
## 2  2013    1998     1     1      533      850  UA        1714 N24211 Fixed wing multi engine 737-824
## 3  2013    1990     1     1      542      923  AA        1141 N619AA Fixed wing multi engine 757-223
```

Make sure you know what "year.x" and "year.y" are

# tidyr

---

# Key `tidyr` verbs

1. `pivot_longer`: Pivot wide data into long format (i.e. "melt", "reshape long")
2. `pivot_wider`: Pivot long data into wide format (i.e. "cast", "reshape wide")
3. `separate`: Split one column into multiple columns
4. `unite`: Combine multiple columns into one

# Key tidy verbs

1. `pivot_longer`: Pivot wide data into long format (i.e. "melt", "reshape long")
2. `pivot_wider`: Pivot long data into wide format (i.e. "cast", "reshape wide")
3. `separate`: Split one column into multiple columns
4. `unite`: Combine multiple columns into one

Let's practice these verbs together in class

# 1) tidyverse::pivot\_longer

```
stocks ← data.frame(  
  time = as.Date('2009-01-01') + 0:1,  
  stock_X = rnorm(2, 0, 1),  
  stock_Y = rnorm(2, 0, 2),  
  stock_Z = rnorm(2, 0, 4)  
)  
stocks
```

```
##          time    stock_X    stock_Y    stock_Z  
## 1 2009-01-01 -0.6576081  0.3679567  4.378233  
## 2 2009-01-02 -1.0493144 -0.5491154 -2.952837
```

We have 4 variables, the date and the stocks

How do we get this in tidy form?

# 1) tidyverse::pivot\_longer

```
stocks %>% pivot_longer(-time, names_to = "stock", values_to = "price")
```

We need to pivot the stock name variables X, Y, Z longer

1. Choose non-time variables: -time
2. Decide what variable holds the names: names\_to = "stock"
3. Decide what variable holds the values: values\_to = "price"

# 1) tidyverse::pivot\_longer

```
stocks %>% pivot_longer(-time, names_to = "stock", values_to = "price")
```

```
## # A tibble: 6 x 3
##   time      stock    price
##   <date>    <chr>    <dbl>
## 1 2009-01-01 stock_X -0.658
## 2 2009-01-01 stock_Y  0.368
## 3 2009-01-01 stock_Z  4.38
## 4 2009-01-02 stock_X -1.05
## 5 2009-01-02 stock_Y -0.549
## 6 2009-01-02 stock_Z -2.95
```

# 1) tidyverse::pivot\_longer

Let's quickly save the "tidy" (i.e. long) stocks data frame for use on the next slide

```
tidy_stocks ← stocks %>%  
  pivot_longer(-time, names_to = "stock", values_to = "price")
```

## 2) tidyverse::pivot\_wider

```
tidy_stocks %>% pivot_wider(names_from = stock, values_from = price)
```

```
## # A tibble: 2 x 4
##   time      stock_X stock_Y stock_Z
##   <date>    <dbl>    <dbl>    <dbl>
## 1 2009-01-01 -0.658    0.368    4.38
## 2 2009-01-02 -1.05     -0.549   -2.95
```

```
tidy_stocks %>% pivot_wider(names_from = time, values_from = price)
```

```
## # A tibble: 3 x 3
##   stock   `2009-01-01` `2009-01-02`
##   <chr>    <dbl>      <dbl>
## 1 stock_X    -0.658     -1.05
## 2 stock_Y     0.368     -0.549
## 3 stock_Z     4.38      -2.95
```

## 2) tidyverse::pivot\_wider

```
tidy_stocks %>% pivot_wider(names_from = stock, values_from = price)
```

```
## # A tibble: 2 x 4
##   time      stock_X stock_Y stock_Z
##   <date>    <dbl>    <dbl>    <dbl>
## 1 2009-01-01 -0.658    0.368    4.38
## 2 2009-01-02 -1.05     -0.549   -2.95
```

```
tidy_stocks %>% pivot_wider(names_from = time, values_from = price)
```

```
## # A tibble: 3 x 3
##   stock   `2009-01-01` `2009-01-02`
##   <chr>    <dbl>      <dbl>
## 1 stock_X    -0.658     -1.05
## 2 stock_Y     0.368     -0.549
## 3 stock_Z     4.38      -2.95
```

Note that the second example has effectively transposed the data

### 3) tidyverse::separate

```
economists ← data.frame(name = c("Adam.Smith", "Paul.Samuelson", "Milton.Friedman"))
economists
```

```
##           name
## 1      Adam.Smith
## 2  Paul.Samuelson
## 3 Milton.Friedman
```

```
economists %>% separate(name, c("first_name", "last_name"))
```

```
##   first_name last_name
## 1      Adam     Smith
## 2      Paul Samuelson
## 3    Milton   Friedman
```

### 3) tidyverse::separate

```
conomists ← data.frame(name = c("Adam.Smith", "Paul.Samuelson", "Milton.Friedman"))
conomists
```

```
##           name
## 1      Adam.Smith
## 2  Paul.Samuelson
## 3 Milton.Friedman
```

```
conomists %>% separate(name, c("first_name", "last_name"))
```

```
##   first_name last_name
## 1      Adam     Smith
## 2      Paul Samuelson
## 3    Milton   Friedman
```

This command is pretty smart. But to avoid ambiguity, you can also specify the separation character with `separate( ... , sep=".")`

## 4) tidyverse

```
gdp ← data.frame(  
  yr = rep(2016, times = 4),  
  mnth = rep(1, times = 4),  
  dy = 1:4,  
  gdp = rnorm(4, mean = 100, sd = 2)  
)  
gdp
```

```
##      yr mnth dy      gdp  
## 1 2016     1   1 98.39997  
## 2 2016     1   2 100.23089  
## 3 2016     1   3 98.77900  
## 4 2016     1   4 99.06514
```

## 4) tidyverse

```
## Combine "yr", "mnth", and "dy" into one "date" column
gdp %>% unite(date, c("yr", "mnth", "dy"), sep = "-")
```

```
##       date      gdp
## 1 2016-1-1 98.39997
## 2 2016-1-2 100.23089
## 3 2016-1-3 98.77900
## 4 2016-1-4 99.06514
```

## 4) tidyverse::unite

Note that `unite` will automatically create a character variable:

```
gdp_u <- gdp %>% unite(date, c("yr", "mnth", "dy"), sep = "-") %>% as_tibble()  
gdp_u
```

```
## # A tibble: 4 x 2  
##   date      gdp  
##   <chr>     <dbl>  
## 1 2016-1-1  98.4  
## 2 2016-1-2  100.  
## 3 2016-1-3  98.8  
## 4 2016-1-4  99.1
```

## 4) tidyverse::unite

Note that `unite` will automatically create a character variable:

```
gdp_u <- gdp %>% unite(date, c("yr", "mnth", "dy"), sep = "-") %>% as_tibble()  
gdp_u
```

```
## # A tibble: 4 x 2  
##   date      gdp  
##   <chr>    <dbl>  
## 1 2016-1-1  98.4  
## 2 2016-1-2  100.  
## 3 2016-1-3  98.8  
## 4 2016-1-4  99.1
```

If you want to convert it to something else (e.g. date or numeric) then you will need to modify it using `mutate`

## 4) tidyverse

```
library(lubridate)
gdp_u %>% mutate(date = ymd(date))
```

```
## # A tibble: 4 x 2
##   date      gdp
##   <date>    <dbl>
## 1 2016-01-01  98.4
## 2 2016-01-02 100.
## 3 2016-01-03  98.8
## 4 2016-01-04  99.1
```

# Regression and ordinary least squares

---

# Why?

## Motivation

Let's start with a few **basic, general questions**

# Why?

## Motivation

Let's start with a few **basic, general questions**

1. What is the goal of econometrics?
2. Why do economists (or other people) study or use econometrics?

# Why?

## Motivation

Let's start with a few **basic, general questions**

1. What is the goal of econometrics?
2. Why do economists (or other people) study or use econometrics?

**One simple answer:** Learn about the world using data

# Why?

## Example

GPA is an output from endowments (ability), and hours studied (inputs), and pollution exposure (externality)

# Why?

## Example

GPA is an output from endowments (ability), and hours studied (inputs), and pollution exposure (externality)

One might hypothesize a model:  $\text{GPA} = f(I, P, \text{SAT}, H)$

where  $H$  is hours studied,  $P$  is pollution exposure, SAT is SAT score and I is family income

# Why?

## Example

GPA is an output from endowments (ability), and hours studied (inputs), and pollution exposure (externality)

One might hypothesize a model:  $\text{GPA} = f(I, P, \text{SAT}, H)$

where  $H$  is hours studied,  $P$  is pollution exposure, SAT is SAT score and I is family income

We expect that GPA will rise with some variables, and decrease with others

# Why?

## Example

GPA is an output from endowments (ability), and hours studied (inputs), and pollution exposure (externality)

One might hypothesize a model:  $\text{GPA} = f(I, P, \text{SAT}, H)$

where  $H$  is hours studied,  $P$  is pollution exposure, SAT is SAT score and I is family income

We expect that GPA will rise with some variables, and decrease with others

But who needs to expect?

# Why?

## Example

GPA is an output from endowments (ability), and hours studied (inputs), and pollution exposure (externality)

One might hypothesize a model:  $\text{GPA} = f(I, P, \text{SAT}, H)$

where  $H$  is hours studied,  $P$  is pollution exposure, SAT is SAT score and I is family income

We expect that GPA will rise with some variables, and decrease with others

But who needs to expect?

# How?

We can write down a linear regression model of the relationship between GPA and (H, P, SAT, PCT):

$$\text{GPA}_i = \beta_0 + \beta_1 I_i + \beta_2 P_i + \beta_3 \text{SAT}_i + \beta_4 H_i + \varepsilon_i$$

The left hand side of the equals sign is our **dependent variable** GPA

The right hand side of the equals sign contains all of our **independent variables** (I, P, SAT, H), and an error term  $\varepsilon_i$  (described later)

The subscript  $i$  means that the variable contains the value for some person  $i$  in our dataset where  $i = 1, \dots, N$

# How?

$$\text{GPA}_i = \beta_0 + \beta_1 I_i + \beta_2 P_i + \beta_3 \text{SAT}_i + \beta_4 H_i + \varepsilon_i$$

We are interested in how pollution P affects GPA

# How?

$$\text{GPA}_i = \beta_0 + \beta_1 I_i + \beta_2 P_i + \beta_3 \text{SAT}_i + \beta_4 H_i + \varepsilon_i$$

We are interested in how pollution P affects GPA

This is given by  $\beta_2$

# How?

$$\text{GPA}_i = \beta_0 + \beta_1 I_i + \beta_2 P_i + \beta_3 \text{SAT}_i + \beta_4 H_i + \varepsilon_i$$

We are interested in how pollution  $P$  affects GPA

This is given by  $\beta_2$

Notice that  $\beta_2 = \frac{\partial \text{GPA}_i}{\partial P_i}$

# How?

$$\text{GPA}_i = \beta_0 + \beta_1 I_i + \beta_2 P_i + \beta_3 \text{SAT}_i + \beta_4 H_i + \varepsilon_i$$

We are interested in how pollution P affects GPA

This is given by  $\beta_2$

Notice that  $\beta_2 = \frac{\partial \text{GPA}_i}{\partial P_i}$

$\beta_2$  tells us how GPA changes, given a 1 unit increase in pollution!

# How?

$$\text{GPA}_i = \beta_0 + \beta_1 I_i + \beta_2 P_i + \beta_3 \text{SAT}_i + \beta_4 H_i + \varepsilon_i$$

We are interested in how pollution  $P$  affects GPA

This is given by  $\beta_2$

Notice that  $\beta_2 = \frac{\partial \text{GPA}_i}{\partial P_i}$

$\beta_2$  tells us how GPA changes, given a 1 unit increase in pollution!

Our goal will be to estimate  $\beta_2$ , we denote estimates with hats:  $\hat{\beta}_2$

# How?

How do we estimate  $\beta_2$ ?

# How?

How do we estimate  $\beta_2$ ?

First, suppose we have a set of estimates for all of our  $\beta$ s, then we can estimate the GPA ( $\widehat{GPA}_i$ ) for any given person based on just (I, P, SAT, H):

$$\widehat{GPA}_i = \hat{\beta}_0 + \hat{\beta}_1 I_i + \hat{\beta}_2 P_i + \hat{\beta}_3 \text{SAT}_i + \hat{\beta}_4 H_i$$

# How?

We estimate the  $\beta$ s with **linear regression**, specifically ordinary least squares

**Ordinary least squares:** choose all the  $\beta$ s so that the sum of squared errors between the *real* GPAs and model-estimated GPAs are minimized:

$$SSE = \sum_{i=1}^N (GPA_i - \widehat{GPA}_i)^2$$

# How?

We estimate the  $\beta$ s with **linear regression**, specifically ordinary least squares

**Ordinary least squares:** choose all the  $\beta$ s so that the sum of squared errors between the *real* GPAs and model-estimated GPAs are minimized:

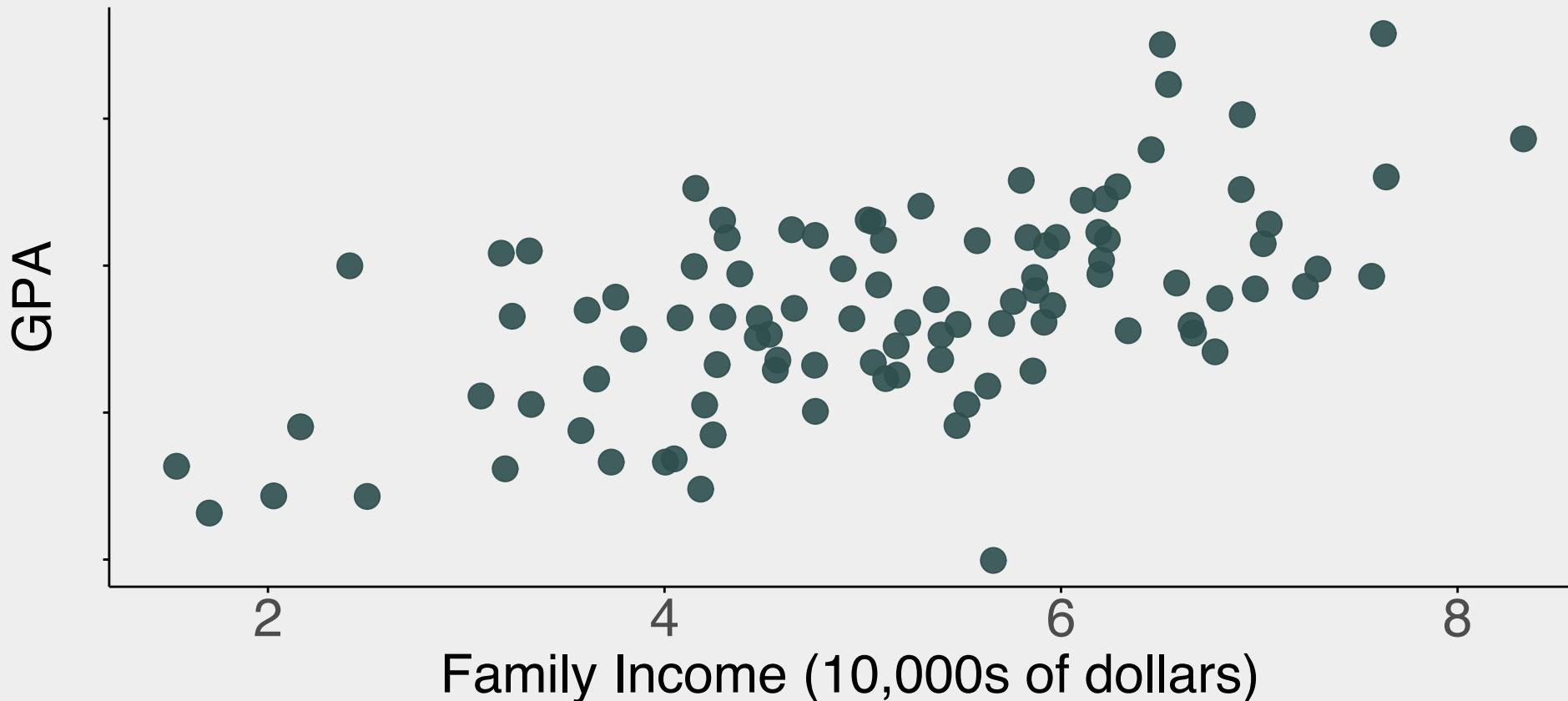
$$SSE = \sum_{i=1}^N (GPA_i - \widehat{GPA}_i)^2$$

Choosing the  $\beta$ s in this fashion gives us the best-fit line through the data

# How?

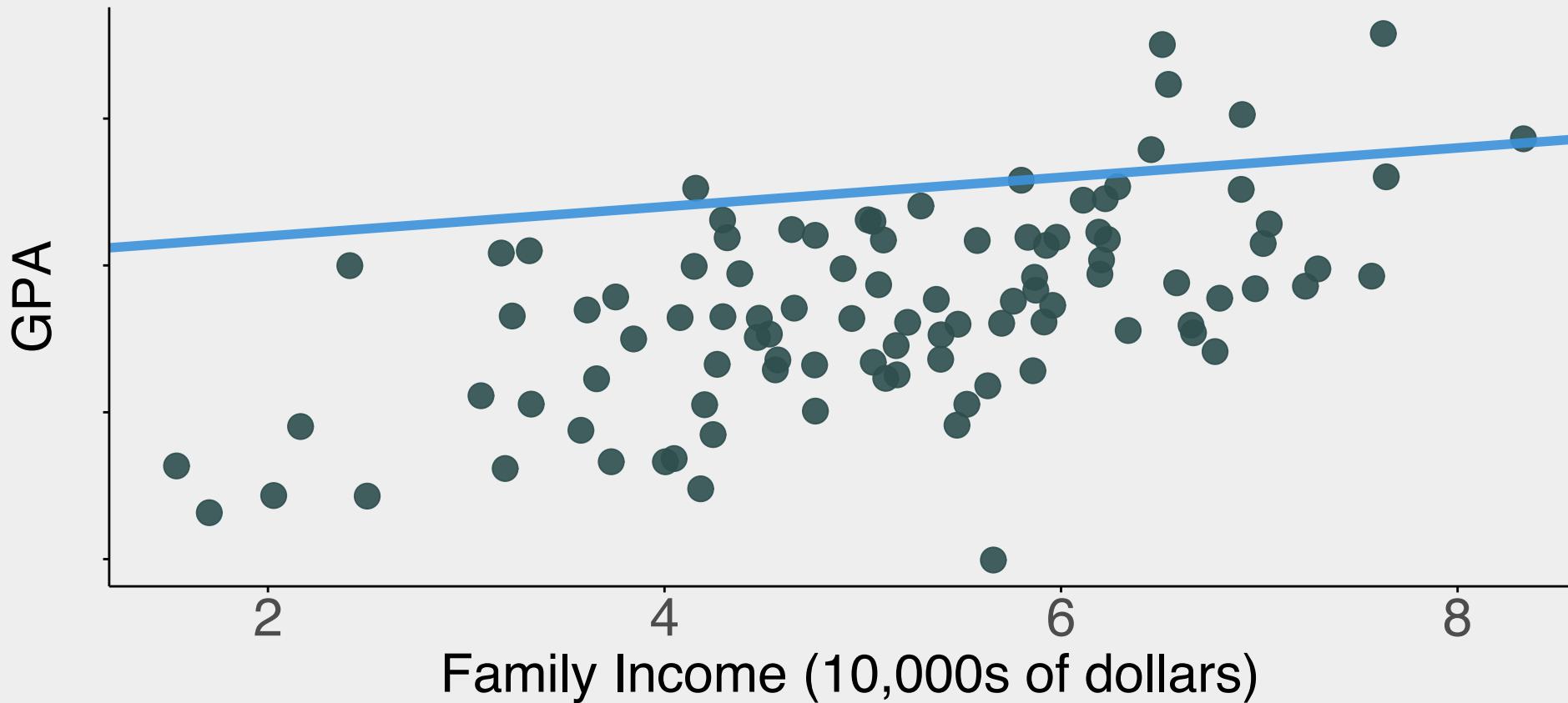
# Simple example

Suppose we were only looking at GPA and family income I



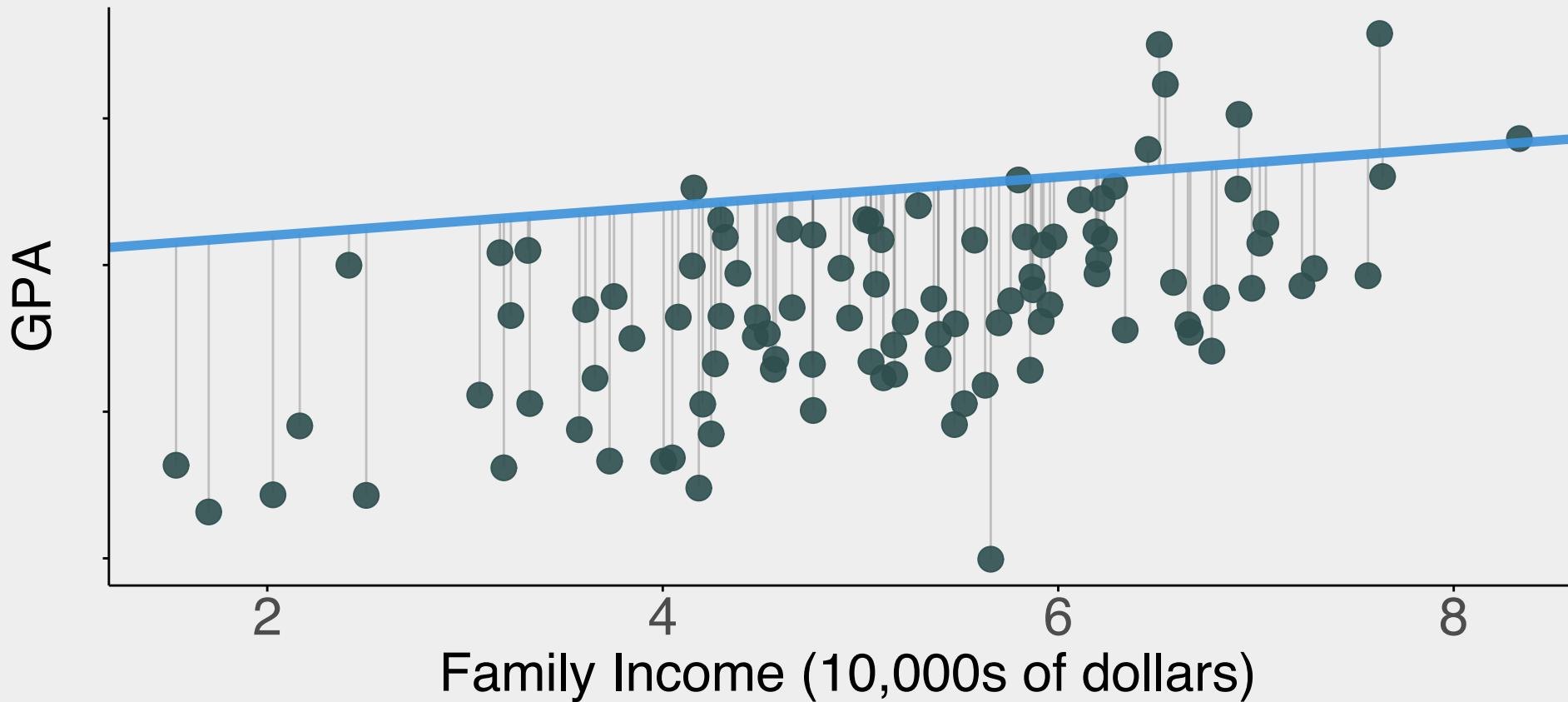
# Simple example

For any line  $(\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x)$



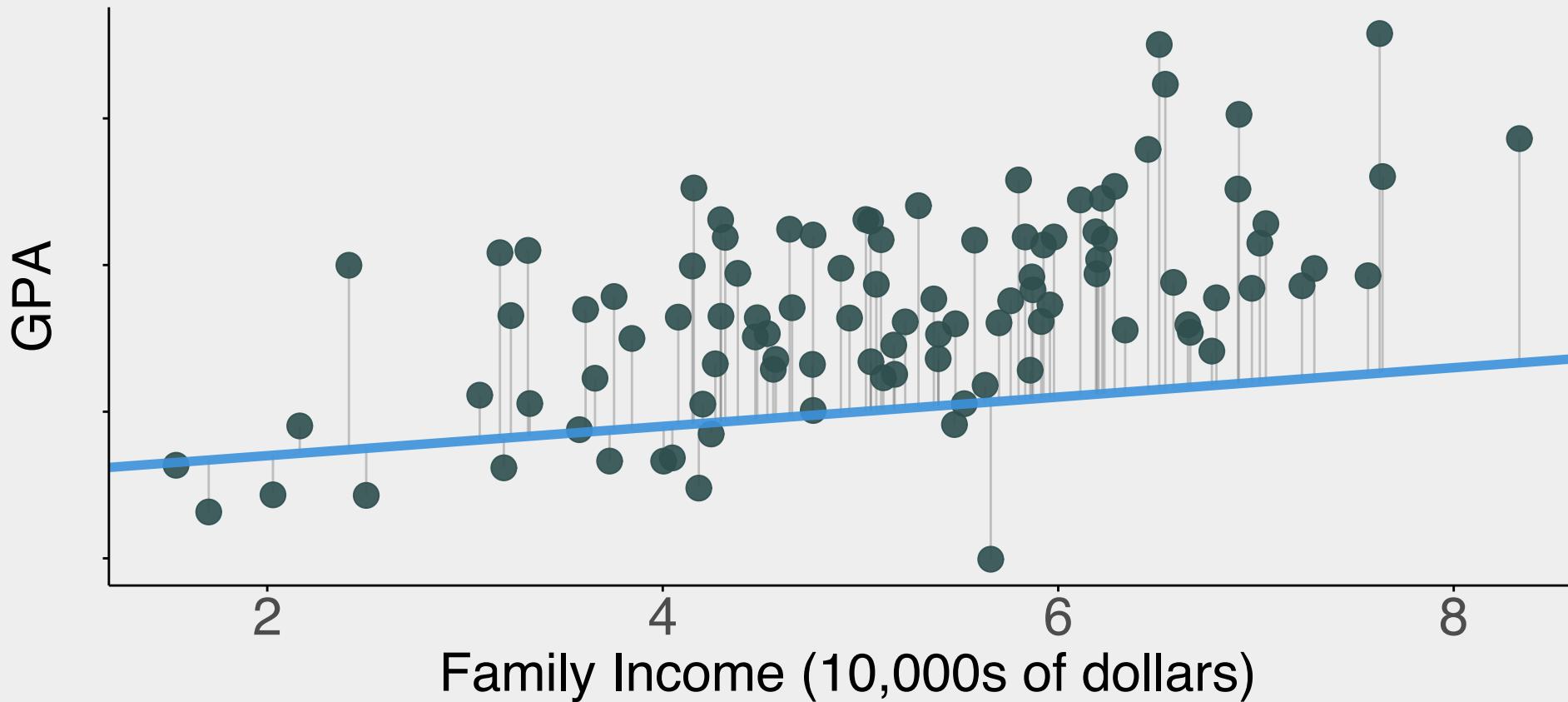
# Simple example

For any line  $(\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x)$ , we can calculate errors:  $e_i = y_i - \hat{y}_i$



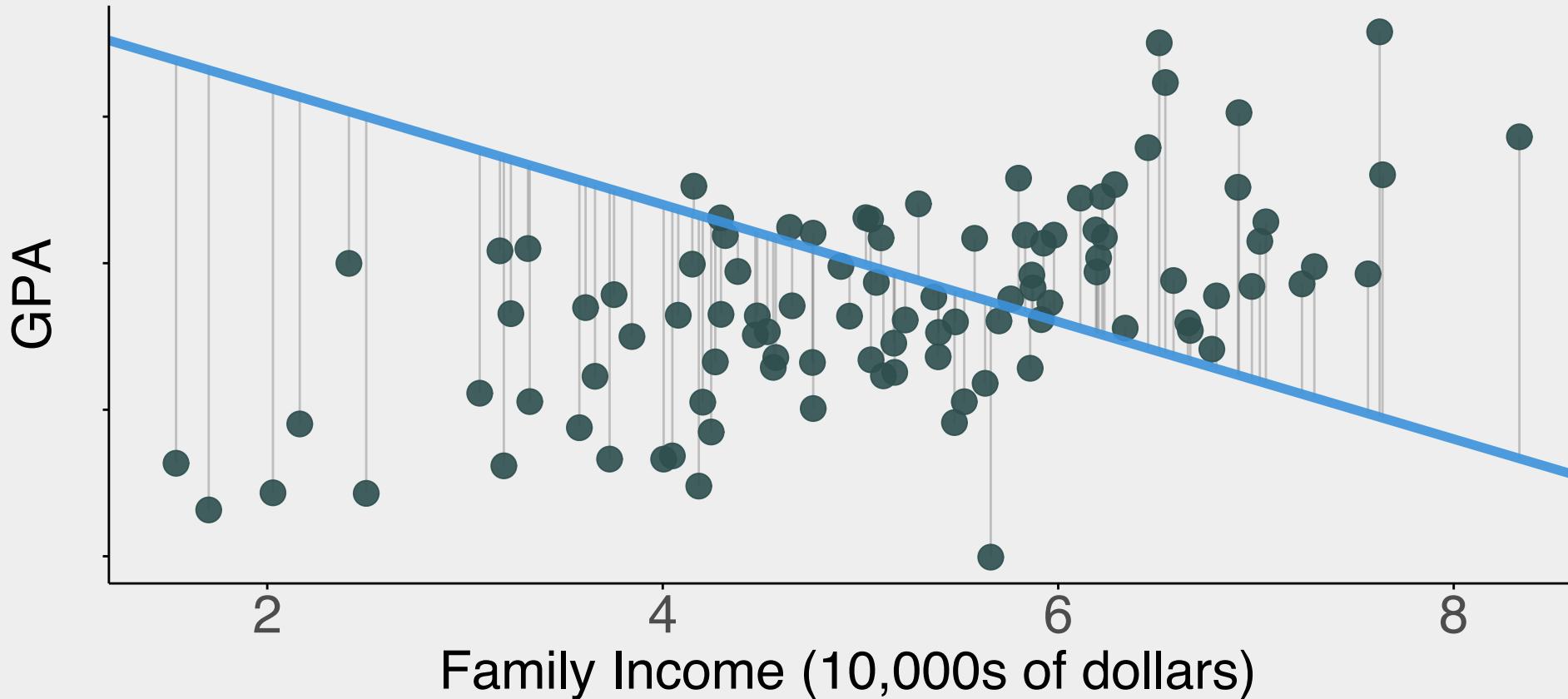
# Simple example

For any line  $(\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x)$ , we can calculate errors:  $e_i = y_i - \hat{y}_i$



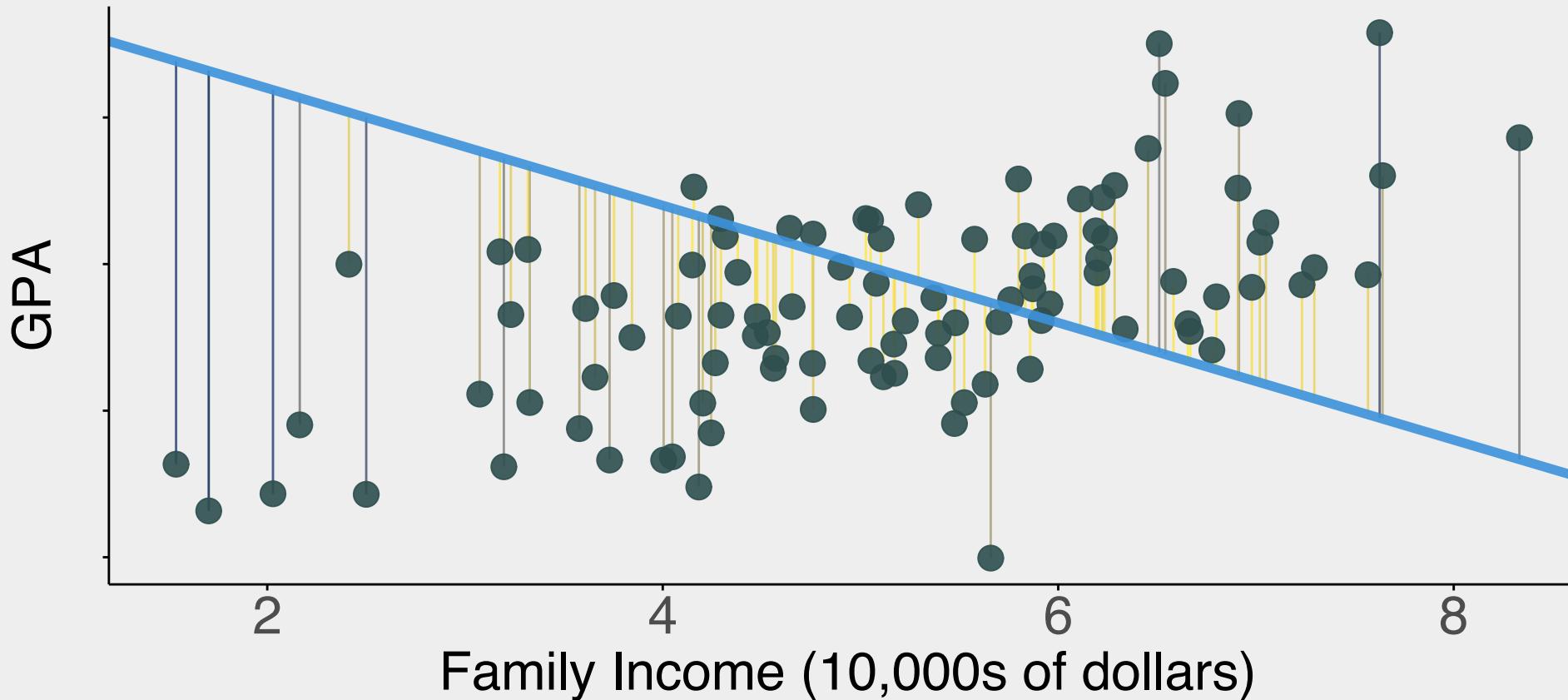
# Simple example

For any line  $(\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x)$ , we can calculate errors:  $e_i = y_i - \hat{y}_i$



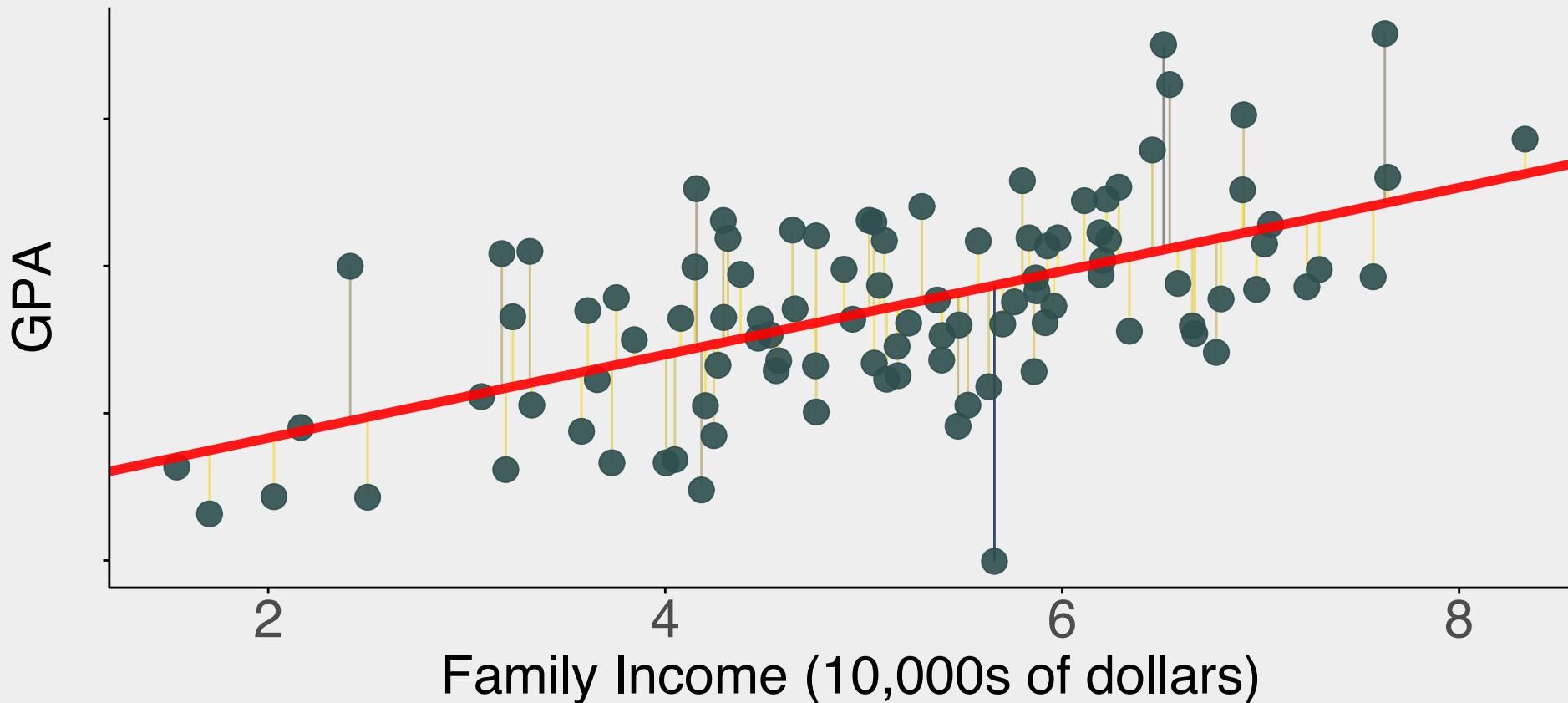
# Simple example

SSE squares the errors ( $\sum e_i^2$ ): bigger errors get bigger penalties



# Simple example

The OLS estimate is the combination of  $\hat{\beta}_0$  and  $\hat{\beta}_1$  that minimize SSE



# OLS error term

So OLS is just the best-fit line through your data

# OLS error term

So OLS is just the best-fit line through your data

# OLS error term

So OLS is just the best-fit line through your data

Why?

# OLS error term

So OLS is just the best-fit line through your data

Why?

Our model isn't perfect, the people in our dataset (i.e. our sample) may not perfectly match up to the entire population of people

# OLS error term

There's **a lot** of other stuff that determines GPAs!

# OLS error term

There's **a lot** of other stuff that determines GPAs!

We jam all that stuff into error term  $\varepsilon_i$ :

$$\text{GPA}_i = \beta_0 + \beta_1 I_i + \beta_2 P_i + \beta_3 \text{SAT}_i + \beta_4 H_i + \varepsilon_i$$

# OLS error term

There's **a lot** of other stuff that determines GPAs!

We jam all that stuff into error term  $\varepsilon_i$ :

$$\text{GPA}_i = \beta_0 + \beta_1 I_i + \beta_2 P_i + \beta_3 \text{SAT}_i + \beta_4 H_i + \varepsilon_i$$

So  $\varepsilon_i$  contains all the determinants of GPA that we aren't explicitly addressing in our model

# OLS properties

OLS has one **very** nice property relevant for this class:<sup>1</sup>

<sup>1</sup> The other is that it has the minimum variance of all unbiased estimators but that's not super important for us.

# OLS properties

OLS has one **very** nice property relevant for this class:<sup>1</sup>

**Unbiasedness:**  $E[\hat{\beta}] = \beta$

<sup>1</sup> The other is that it has the minimum variance of all unbiased estimators but that's not super important for us.

# OLS properties

**Unbiasedness:**  $E[\hat{\beta}] = \beta$

On average, our estimate  $\hat{\beta}$  exactly equals the **true**  $\beta$

# OLS properties

**Unbiasedness:**  $E[\hat{\beta}] = \beta$

On average, our estimate  $\hat{\beta}$  exactly equals the **true**  $\beta$

The key is **on average**: we are estimating our model using only some sample of the data

# OLS properties

**Unbiasedness:**  $E[\hat{\beta}] = \beta$

On average, our estimate  $\hat{\beta}$  exactly equals the **true**  $\beta$

The key is **on average**: we are estimating our model using only some sample of the data

The estimated  $\beta$  won't exactly be right for the entire population, but on average, we expect it to match

# OLS properties

**Unbiasedness:**  $E[\hat{\beta}] = \beta$

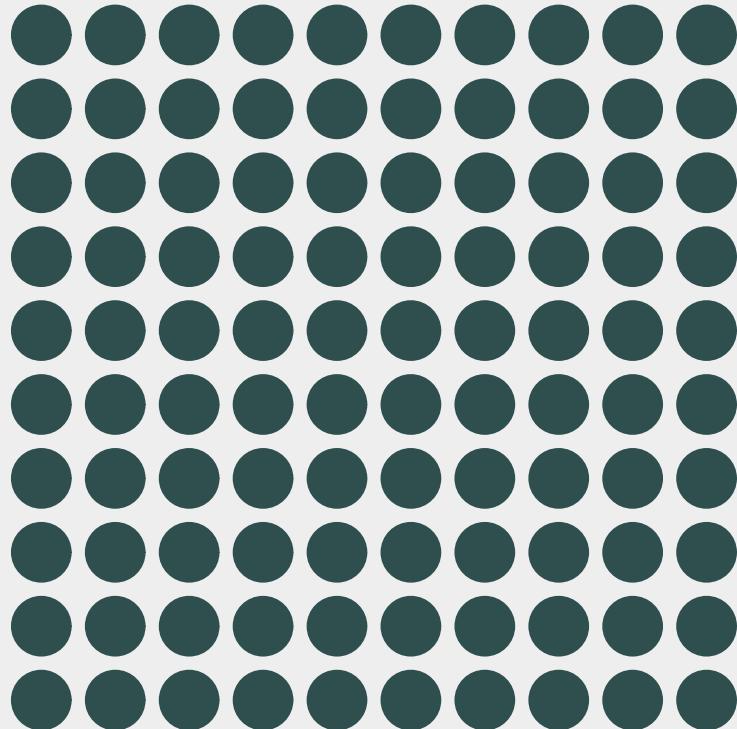
On average, our estimate  $\hat{\beta}$  exactly equals the **true**  $\beta$

The key is **on average**: we are estimating our model using only some sample of the data

The estimated  $\beta$  won't exactly be right for the entire population, but on average, we expect it to match

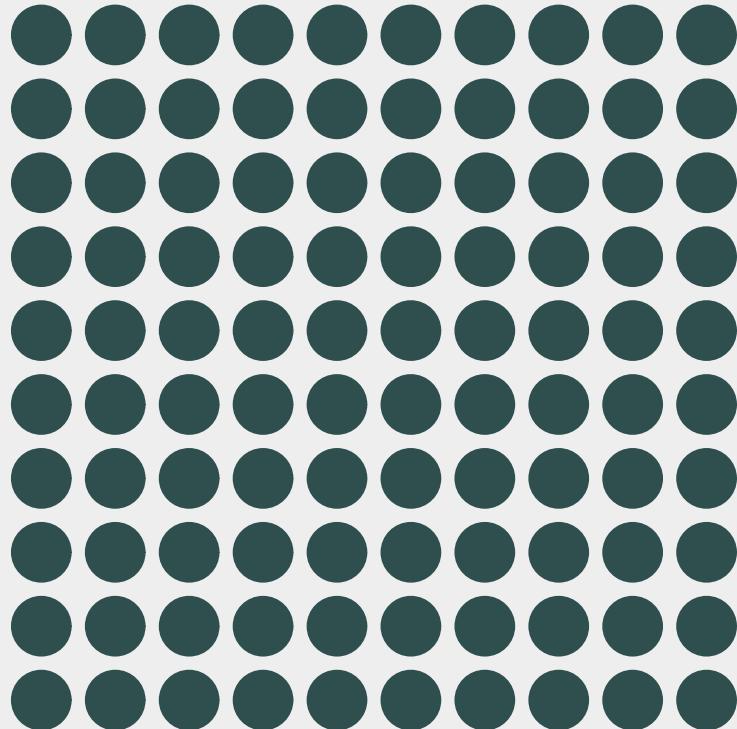
Let's see in an example where we only have a subsample of the full population of data

# OLS properties

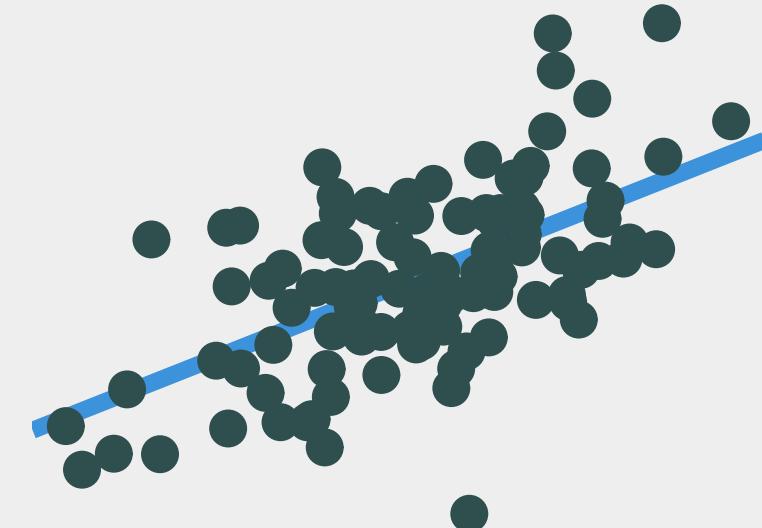


Population

# OLS properties



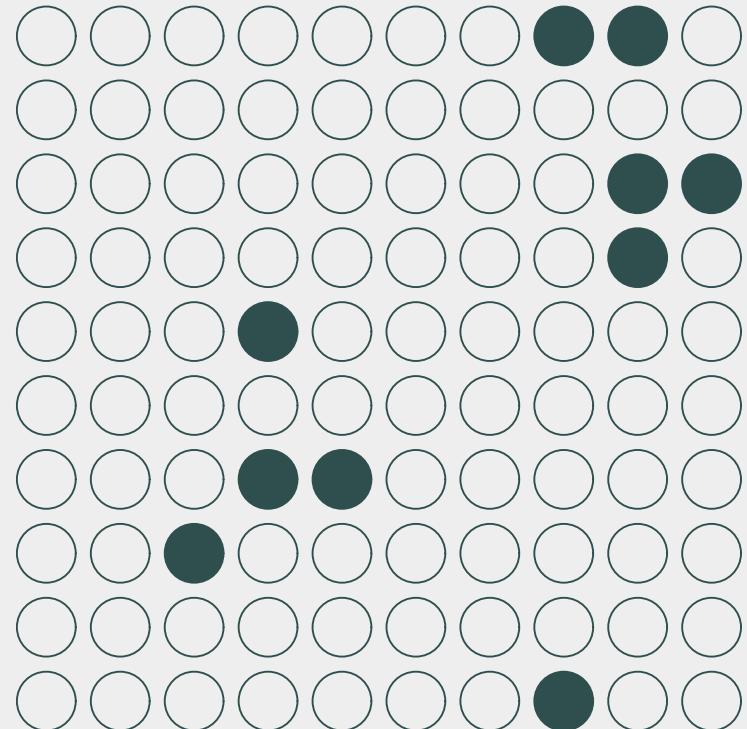
Population



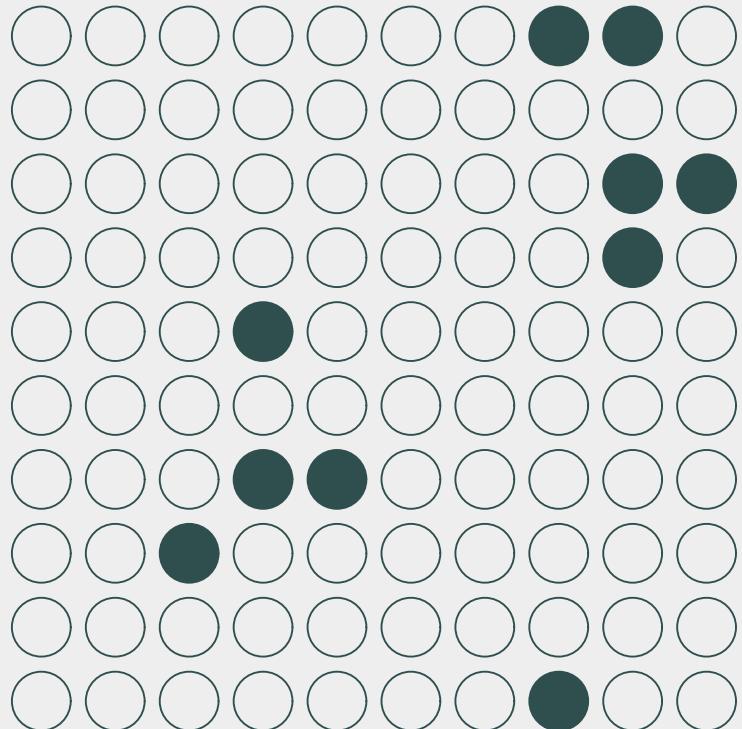
Population relationship

$$y_i = 2.53 + 0.57x_i + u_i$$

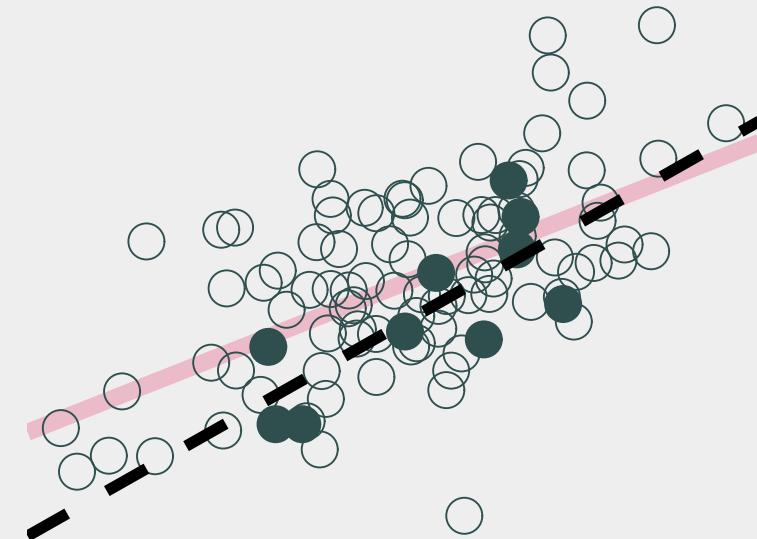
$$y_i = \beta_0 + \beta_1 x_i + u_i$$



**Sample 1:** 10 random individuals



**Sample 1:** 10 random individuals

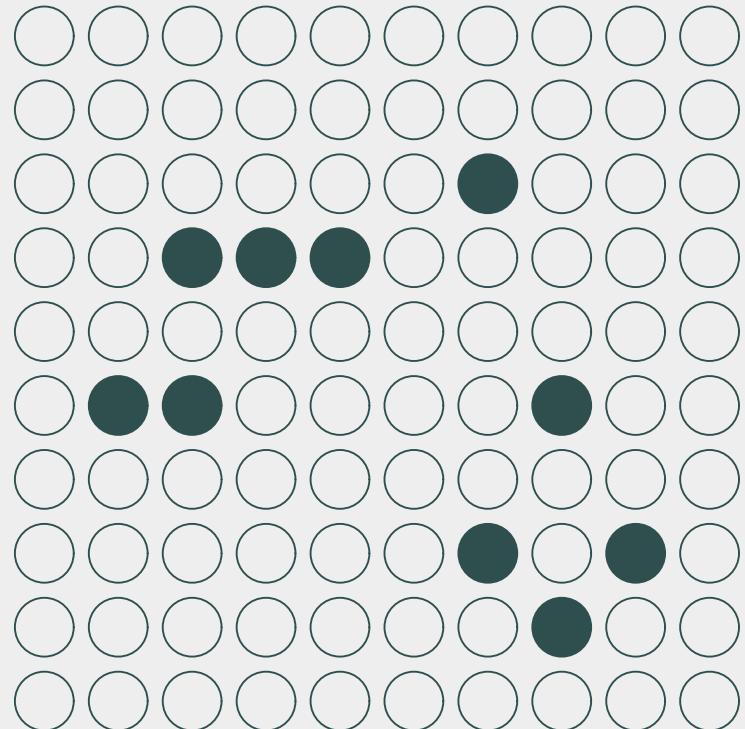


**Population relationship**

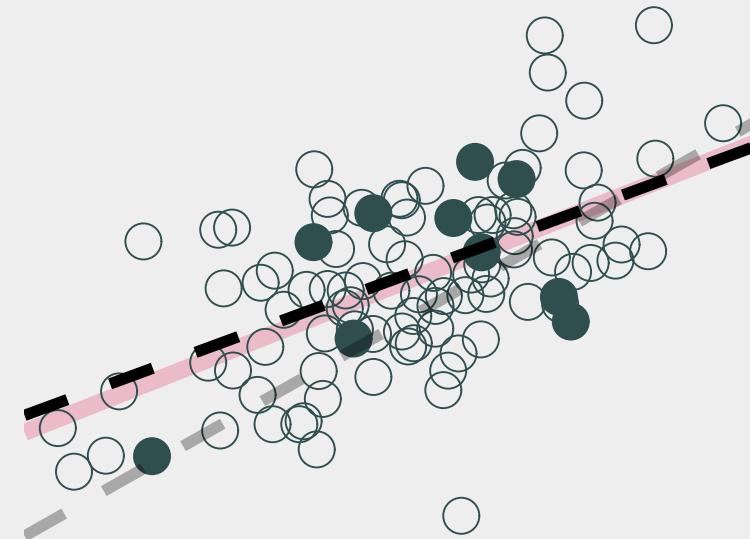
$$y_i = 2.53 + 0.57x_i + u_i$$

**Sample relationship**

$$\hat{y}_i = 0.72 + 0.81x_i$$



Sample 2: 10 random individuals

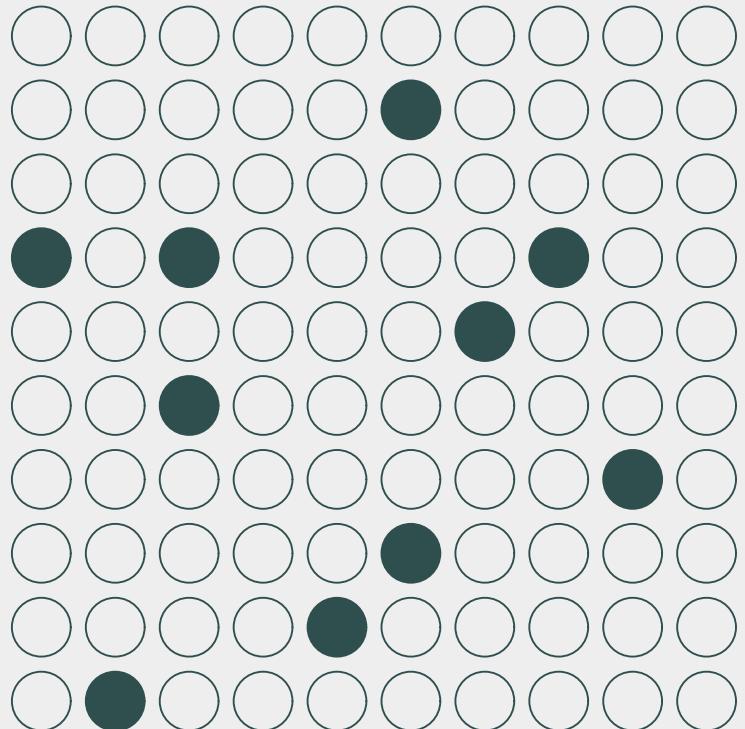


**Population relationship**

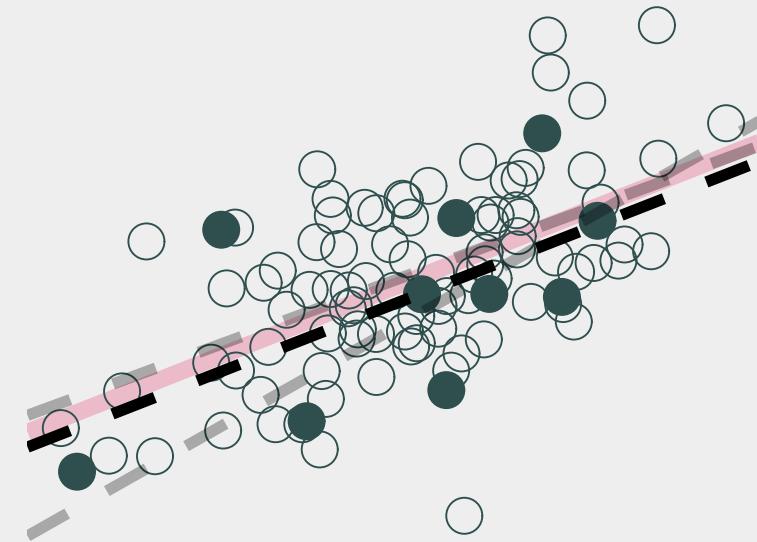
$$y_i = 2.53 + 0.57x_i + u_i$$

**Sample relationship**

$$\hat{y}_i = 2.82 + 0.53x_i$$



**Sample 3:** 10 random individuals



**Population relationship**

$$y_i = 2.53 + 0.57x_i + u_i$$

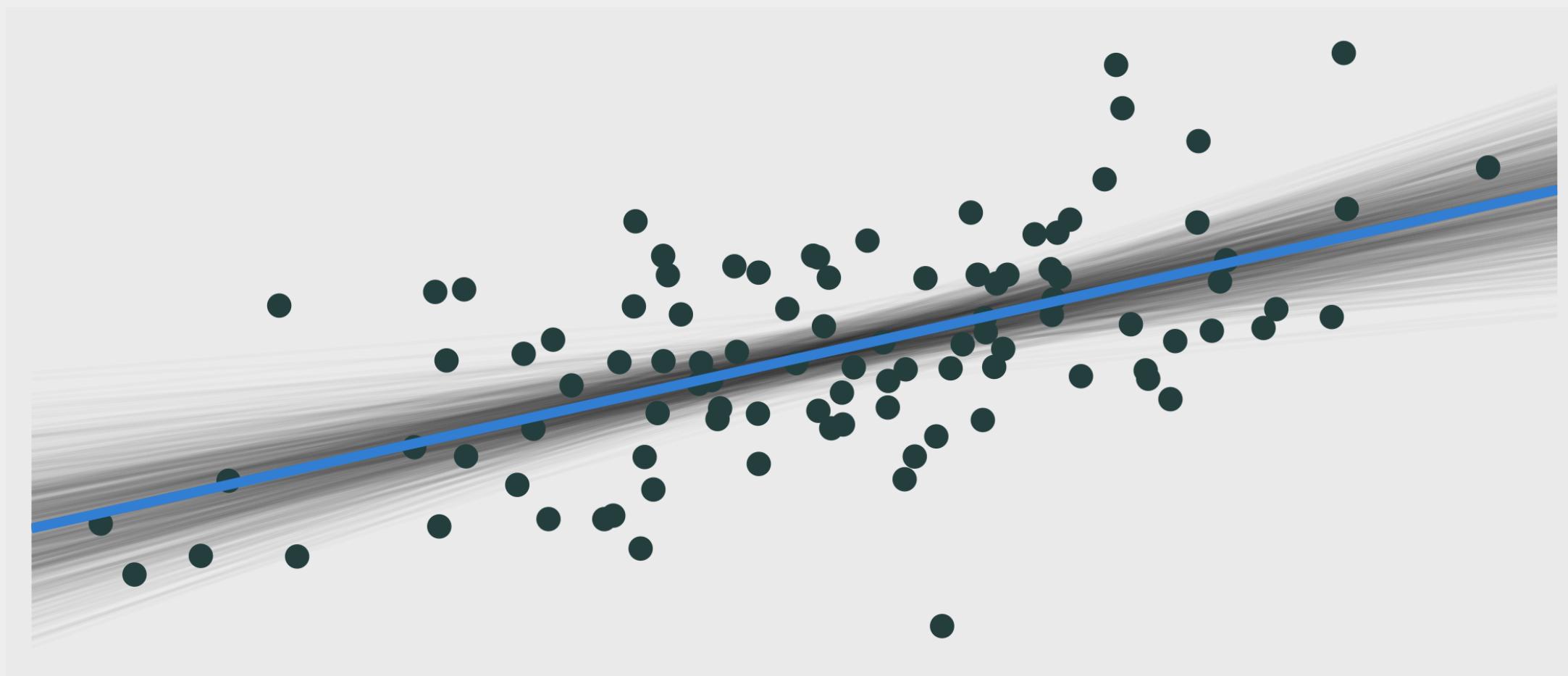
**Sample relationship**

$$\hat{y}_i = 2.32 + 0.56x_i$$

Let's repeat this 1,000 times.

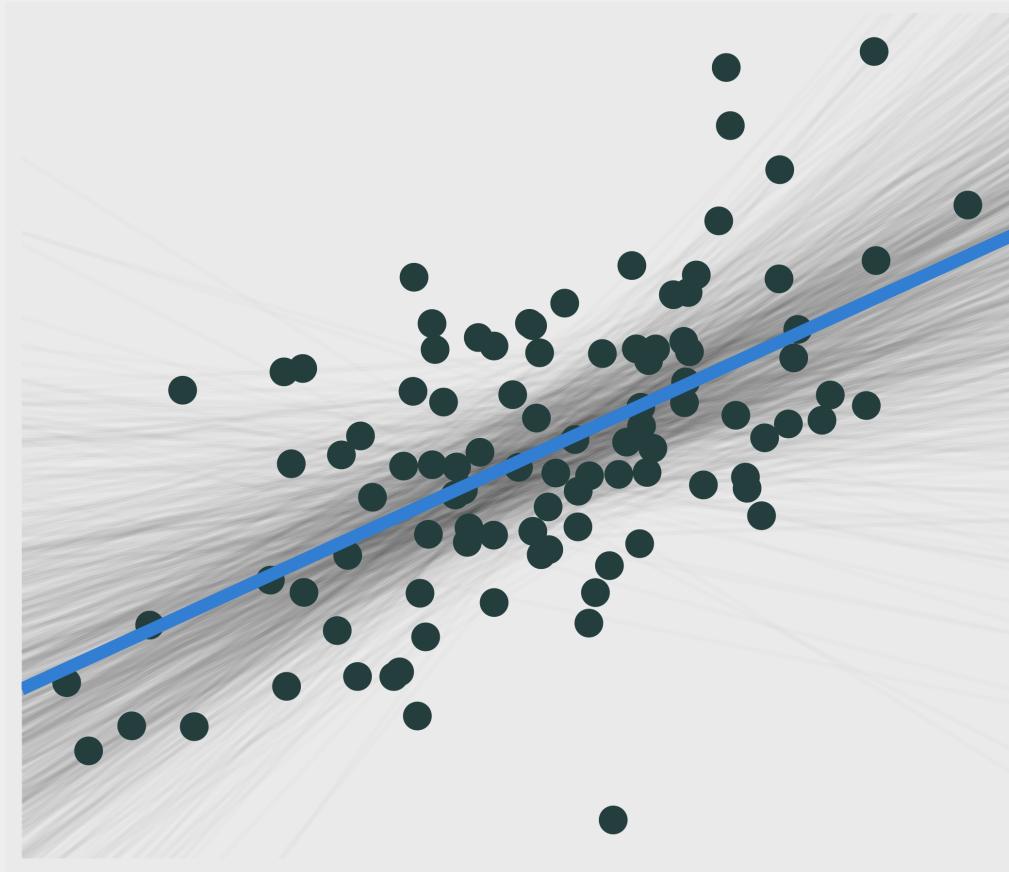
(This exercise is called a (Monte Carlo) simulation.)

# Population vs. sample



# Population vs. sample

**Question:** Why do we care about *population vs. sample*?



On **average**, our regression lines match the population line very nicely

However, **individual lines** (samples) can really miss the mark

Differences between individual samples and the population lead to **uncertainty** for us in the true effect

# Population vs. sample

**Answer:** Uncertainty matters!

# Population vs. sample

**Answer:** Uncertainty matters!

$\hat{\beta}$  itself is random, it will depend on the sample of data we have

# Population vs. sample

**Answer:** Uncertainty matters!

$\hat{\beta}$  itself is random, it will depend on the sample of data we have

When we take a sample and run a regression, we don't know if it's a 'good' sample ( $\hat{\beta}$  is close to  $\beta$ ) or a 'bad sample' (our sample differs greatly from the population)

# Unbiasedness

For OLS to be unbiased and give us, on average, the causal effect of some X on some Y we need a few assumptions to hold

# Unbiasedness

For OLS to be unbiased and give us, on average, the causal effect of some X on some Y we need a few assumptions to hold

Whether or not these assumptions are true is why you often hear *correlation is not causation*

# Unbiasedness

For OLS to be unbiased and give us, on average, the causal effect of some X on some Y we need a few assumptions to hold

Whether or not these assumptions are true is why you often hear *correlation is not causation*

If we want some  $\hat{\beta}_1$  on a variable  $x$  to be unbiased we need the following to be true:

$$E[x\varepsilon] = 0 \quad \leftrightarrow \quad \text{correlation}(x, \varepsilon) = 0$$

# Unbiasedness

For OLS to be unbiased and give us, on average, the causal effect of some X on some Y we need a few assumptions to hold

Whether or not these assumptions are true is why you often hear *correlation is not causation*

If we want some  $\hat{\beta}_1$  on a variable  $x$  to be unbiased we need the following to be true:

$$E[x\varepsilon] = 0 \quad \leftrightarrow \quad \text{correlation}(x, \varepsilon) = 0$$

The variable you are interested in **cannot** be correlated with the error term

# Unbiasedness

The variable you are interested in **cannot** be correlated with the error term

# Unbiasedness

The variable you are interested in **cannot** be correlated with the error term

What does this mean in words?

# Unbiasedness

The variable you are interested in **cannot** be correlated with the error term

What does this mean in words?

The error term contains all variables that determine  $y$ , but we *omitted* from our model

# Unbiasedness

The variable you are interested in **cannot** be correlated with the error term

What does this mean in words?

The error term contains all variables that determine  $y$ , but we *omitted* from our model

We are assuming that our variable of interest,  $x$ , is not correlated with any of these omitted variable

# Unbiasedness

The variable you are interested in **cannot** be correlated with the error term

What does this mean in words?

The error term contains all variables that determine  $y$ , but we *omitted* from our model

We are assuming that our variable of interest,  $x$ , is not correlated with any of these omitted variable

If  $x$  is correlated with any of them, then we will have something called **omitted variable bias**

# Omitted variable bias

Here's an intuitive example

# Omitted variable bias

Here's an intuitive example

Suppose we wanted to understand the effect of lead exposure  $P$  on GPAs

# Omitted variable bias

Here's an intuitive example

Suppose we wanted to understand the effect of lead exposure  $P$  on GPAs

lead harms children's brain development, especially before age 6

# Omitted variable bias

Here's an intuitive example

Suppose we wanted to understand the effect of lead exposure  $P$  on GPAs

lead harms children's brain development, especially before age 6

We should expect early-life lead exposure to reduce future GPAs

# Omitted variable bias

Our model might look like:

$$\text{GPA}_i = \beta_0 + \beta_1 \text{P}_i + \varepsilon_i$$

# Omitted variable bias

Our model might look like:

$$\text{GPA}_i = \beta_0 + \beta_1 \text{P}_i + \varepsilon_i$$

We want to know  $\beta_1$

# Omitted variable bias

Our model might look like:

$$\text{GPA}_i = \beta_0 + \beta_1 P_i + \varepsilon_i$$

We want to know  $\beta_1$

What would happen if we took a sample of *real world data* and used OLS to estimate  $\hat{\beta}_1$ ?

# Omitted variable bias

We would have omitted variable bias

# Omitted variable bias

We would have omitted variable bias

Why? What are some examples?

# Omitted variable bias

We would have omitted variable bias

Why? What are some examples?

**Who** is more likely to be exposed to lead?

# Omitted variable bias

We would have omitted variable bias

Why? What are some examples?

**Who** is more likely to be exposed to lead?

Poorer families likely have more lead exposure, why?

# Omitted variable bias

We would have omitted variable bias

Why? What are some examples?

**Who** is more likely to be exposed to lead?

Poorer families likely have more lead exposure, why?

Richer families can move away, pay to replace lead paint, lead pipes, etc

# Omitted variable bias

We would have omitted variable bias

Why? What are some examples?

**Who** is more likely to be exposed to lead?

Poorer families likely have more lead exposure, why?

Richer families can move away, pay to replace lead paint, lead pipes, etc

This means lead exposure is correlated with lower income

# Omitted variable bias

Why does this correlation cause us problems?

# Omitted variable bias

Why does this correlation cause us problems?

Family income *also* matters for GPA, it is in  $\varepsilon_i$ , so our assumption that  $\text{correlation}(x, \varepsilon) = 0$  is violated

# Omitted variable bias

Why does this correlation cause us problems?

Family income *also* matters for GPA, it is in  $\varepsilon_i$ , so our assumption that  $\text{correlation}(x, \varepsilon) = 0$  is violated

Children from richer families tend to have higher GPAs

# Omitted variable bias

If we just look at the effect of lead exposure on GPAs without addressing its correlation with income, lead exposure will look worse than it actually is

# Omitted variable bias

If we just look at the effect of lead exposure on GPAs without addressing its correlation with income, lead exposure will look worse than it actually is

This is because our data on lead exposure is also proxying for income (since  $\text{correlation}(x, \varepsilon) = 0$  )

# Omitted variable bias

If we just look at the effect of lead exposure on GPAs without addressing its correlation with income, lead exposure will look worse than it actually is

This is because our data on lead exposure is also proxying for income (since  $\text{correlation}(x, \varepsilon) = 0$ )

So  $\hat{\beta}_1$  will pick up the effect of both!

# Omitted variable bias

If we just look at the effect of lead exposure on GPAs without addressing its correlation with income, lead exposure will look worse than it actually is

This is because our data on lead exposure is also proxying for income (since  $\text{correlation}(x, \varepsilon) = 0$ )

So  $\hat{\beta}_1$  will pick up the effect of both!

Our estimate  $\hat{\beta}_1$  is **biased** and overstates the negative effects of lead

# Omitted variable bias

How do we fix this bias?

# Omitted variable bias

How do we fix this bias?

Make income not omitted: control for it in our model

# Omitted variable bias

How do we fix this bias?

Make income not omitted: control for it in our model

If we have data on family income  $I$  we can instead write our model as:

$$\text{GPA}_i = \beta_0 + \beta_1 P_i + \beta_2 I_i + \varepsilon_i$$

$I$  is no longer omitted

# Omitted variable bias

How do we fix this bias?

Make income not omitted: control for it in our model

If we have data on family income  $I$  we can instead write our model as:

$$\text{GPA}_i = \beta_0 + \beta_1 P_i + \beta_2 I_i + \varepsilon_i$$

$I$  is no longer omitted

Independent variables in our model that we include to address bias are called  
**controls**

# Regression in R

---

# How do we actually run regressions?

Now we are going to learn how to run regressions in R

# How do we actually run regressions?

Now we are going to learn how to run regressions in R

There's a lot of regression packages, built-in is `lm`, or you can also use a new one called `fixest`

# How do we actually run regressions?

Now we are going to learn how to run regressions in R

There's a lot of regression packages, built-in is `lm`, or you can also use a new one called `fixest`

We will also be using the `broom` package to make our output look nice

# How do we actually run regressions?

Now we are going to learn how to run regressions in R

There's a lot of regression packages, built-in is `lm`, or you can also use a new one called `fixest`

We will also be using the `broom` package to make our output look nice

They work almost identically

# Using regression packages

To run a regression we need 3 things:

# Using regression packages

To run a regression we need 3 things:

1. The package that will run the regression
2. Our regression formula
3. The dataframe that contains our data

# Using regression packages

To run a regression we need 3 things:

1. The package that will run the regression
2. Our regression formula
3. The dataframe that contains our data

In general, we will always run a regression like this:

```
package_name(formula_here, data = dataframe_here)
```

# Using regression packages

To run a regression we need 3 things:

1. The package that will run the regression
2. Our regression formula
3. The dataframe that contains our data

In general, we will always run a regression like this:

```
package_name(formula_here, data = dataframe_here)
```

You can then store the output by assigning it to a variable: `results =`  
`package_name(formula_here, data = dataframe_here)`

# Using regression packages

Let's start by using the built-in `lm` package along with the `starwars` dataset

```
starwars
```

```
## # A tibble: 87 x 14
##   name    height  mass hair_color skin_color eye_color birth_year sex gender homeworld species film
##   <chr>     <int> <dbl> <chr>       <chr>       <chr>        <dbl> <chr> <chr>       <chr>       <chr> <list>
## 1 Luke...     172     77 blond      fair        blue          19 male  masculin Tatooine Human <chr>
## 2 C-3PO       167     75 <NA>       gold        yellow        112 none  masculin Tatooine Droid <chr>
## 3 R2-D2        96     32 <NA>       white, bl... red           33 none  masculin Naboo  Droid <chr>
## 4 Dart...      202    136 none      white        yellow        41.9 male  masculin Tatooine Human <chr>
## 5 Leia...      150     49 brown      light       brown          19 femal feminin Alderaan Human <chr>
## 6 Owen...      178    120 brown, gr... light       blue           52 male  masculin Tatooine Human <chr>
## 7 Beru...      165     75 brown      light       blue           47 femal feminin Tatooine Human <chr>
## 8 R5-D4        97     32 <NA>       white, red red            NA none  masculin Tatooine Droid <chr>
## 9 Bigg...      183     84 black      light       brown          24 male  masculin Tatooine Human <chr>
## 10 Obi-...      182     77 auburn, w... fair        blue-gray        57 male  masculin Stewjon Human <chr>
## # ... with 77 more rows, and 1 more variable: starships <list>
```

# Using regression packages

Suppose we wanted to see what was the effect of height on mass:

$$mass_i = \beta_0 + \beta_1 height_i + \varepsilon_i$$

# Using regression packages

Suppose we wanted to see what was the effect of height on mass:

$$mass_i = \beta_0 + \beta_1 height_i + \varepsilon_i$$

We can run the following code:

```
# package_name(formula_here, data = dataframe_here)
lm(mass ~ height, data = starwars)
```

```
##
## Call:
## lm(formula = mass ~ height, data = starwars)
##
## Coefficients:
## (Intercept)      height
## -13.8103        0.6386
```

# Using regression packages

We can clean up the output a bit by piping it to `broom::tidy`:

```
# package_name(formula_here, data = dataframe_here)
lm(mass ~ height, data = starwars) %>%
  broom::tidy()
```

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>      <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -13.8      111.     -0.124    0.902
## 2 height       0.639      0.626     1.02     0.312
```

# Using regression packages

The first column gives us our estimates:  $\hat{\beta}_0, \hat{\beta}_1$

# Using regression packages

The first column gives us our estimates:  $\hat{\beta}_0, \hat{\beta}_1$

The last column is the **p-value**: the probability that we would have gotten an estimate at least that large, if the *true* value was actually 0

Smaller values generally mean it is more likely that height has an effect on mass: the probability that we could have gotten our estimated value if height didn't matter is very low

# Interpreting coefficient estimates

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -13.8     111.     -0.124   0.902
## 2 height       0.639     0.626     1.02    0.312
```

What do these coefficient estimates mean?

# Interpreting coefficient estimates

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -13.8     111.     -0.124   0.902
## 2 height       0.639     0.626     1.02    0.312
```

What do these coefficient estimates mean?

The estimate for the  $\beta$  on height means: if we increase height by 1cm, mass increases by 0.639kg

# Interpreting coefficient estimates

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -13.8     111.     -0.124   0.902
## 2 height       0.639     0.626     1.02    0.312
```

What do these coefficient estimates mean?

The estimate for the  $\beta$  on height means: if we increase height by 1cm, mass increases by 0.639kg

This just comes from our previous example where  $\beta_1 = \frac{\partial \text{mass}_i}{\partial \text{height}_i}$

# Interpreting coefficient estimates

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -13.8     111.     -0.124   0.902
## 2 height       0.639     0.626     1.02    0.312
```

How do we interpret  $\beta_0$ , the estimate of the *intercept*?

# Interpreting coefficient estimates

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -13.8     111.     -0.124   0.902
## 2 height       0.639     0.626     1.02    0.312
```

How do we interpret  $\beta_0$ , the estimate of the *intercept*?

Well, it is just the estimated mass, given someone had zero height:

$$\hat{\beta}_0 = \hat{\beta}_0 + \hat{\beta}_1 \times 0$$

# Interpreting coefficient estimates

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -13.8     111.     -0.124   0.902
## 2 height       0.639     0.626     1.02    0.312
```

How do we interpret  $\beta_0$ , the estimate of the *intercept*?

Well, it is just the estimated mass, given someone had zero height:

$$\hat{\beta}_0 = \hat{\beta}_0 + \hat{\beta}_1 \times 0$$

It's kind of a nonsense interpretation here since no one has zero height

# Interpreting coefficient estimates

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -13.8     111.     -0.124   0.902
## 2 height       0.639     0.626     1.02    0.312
```

How do we interpret  $\beta_0$ , the estimate of the *intercept*?

Well, it is just the estimated mass, given someone had zero height:

$$\hat{\beta}_0 = \hat{\beta}_0 + \hat{\beta}_1 \times 0$$

It's kind of a nonsense interpretation here since no one has zero height

Generally we don't read too much into the intercept terms

# Interpreting coefficient estimates

Now, what if we changed our model a bit so it was instead:

$$\log(\text{mass}_i) = \beta_0 + \beta_1 \log(\text{height}_i) + \varepsilon_i$$

What does  $\beta_1$  mean now?

# Interpreting coefficient estimates

Now, what if we changed our model a bit so it was instead:

$$\log(\text{mass}_i) = \beta_0 + \beta_1 \log(\text{height}_i) + \varepsilon_i$$

What does  $\beta_1$  mean now?

$$\beta_1 = \frac{\partial \log(\text{mass}_i)}{\partial \log(\text{height}_i)}$$

# Interpreting coefficient estimates

Now, what if we changed our model a bit so it was instead:

$$\log(\text{mass}_i) = \beta_0 + \beta_1 \log(\text{height}_i) + \varepsilon_i$$

What does  $\beta_1$  mean now?

$$\beta_1 = \frac{\partial \log(\text{mass}_i)}{\partial \log(\text{height}_i)}$$

But we can rewrite this as:

$$\beta_1 = \frac{\partial \log(\text{mass}_i)}{\partial \log(\text{height}_i)} = \frac{\partial \log(\text{mass}_i)}{\partial \text{mass}_i} \frac{\partial \text{mass}_i}{\partial \text{height}_i} \frac{\partial \text{height}_i}{\partial \log(\text{height}_i)}$$

# Interpreting coefficient estimates

$$\beta_1 = \frac{\partial \log(\text{mass}_i)}{\partial \log(\text{height}_i)} = \frac{\partial \log(\text{mass}_i)}{\partial \text{mass}_i} \frac{\partial \text{mass}_i}{\partial \text{height}_i} \frac{\partial \text{height}_i}{\partial \log(\text{height}_i)}$$

And this is equal to:

$$\beta_1 = \frac{\partial \log(\text{mass}_i)}{\partial \log(\text{height}_i)} = \frac{1}{\text{mass}_i} \frac{\partial \text{mass}_i}{\partial \text{height}_i} \frac{\text{height}_i}{1}$$

# Interpreting coefficient estimates

$$\beta_1 = \frac{\partial \log(\text{mass}_i)}{\partial \log(\text{height}_i)} = \frac{\partial \log(\text{mass}_i)}{\partial \text{mass}_i} \frac{\partial \text{mass}_i}{\partial \text{height}_i} \frac{\partial \text{height}_i}{\partial \log(\text{height}_i)}$$

And this is equal to:

$$\beta_1 = \frac{\partial \log(\text{mass}_i)}{\partial \log(\text{height}_i)} = \frac{1}{\text{mass}_i} \frac{\partial \text{mass}_i}{\partial \text{height}_i} \frac{\text{height}_i}{1}$$

And finally:

$$\beta_1 = \frac{\partial \log(\text{mass}_i)}{\partial \log(\text{height}_i)} = \frac{\text{height}_i}{\text{mass}_i} \frac{\partial \text{mass}_i}{\partial \text{height}_i}$$

which is the definition of the elasticity of mass with respect to height

# Interpreting coefficient estimates

$$\log(\text{mass}_i) = \beta_0 + \beta_1 \log(\text{height}_i) + \varepsilon_i$$

In a *log-log* model,  $\beta_1$  tells us the percent change in mass, given a percent change in height

# Interpreting coefficient estimates

$$\log(\text{mass}_i) = \beta_0 + \beta_1 \log(\text{height}_i) + \varepsilon_i$$

In a *log-log* model,  $\beta_1$  tells us the percent change in mass, given a percent change in height

Let's run the regression:

```
# package_name(formula_here, data = dataframe_here)
lm(log(mass) ~ log(height), data = starwars) %>%
  broom::tidy()
```

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic    p.value
##   <chr>      <dbl>     <dbl>     <dbl>      <dbl>
## 1 (Intercept) -3.84      1.17     -3.27 0.00181
## 2 log(height)  1.58      0.228      6.93 0.00000000410
```

# Interpreting coefficient estimates

```
# package_name(formula_here, data = dataframe_here)
lm(log(mass) ~ log(height), data = starwars) %>%
  broom::tidy()
```

```
## # A tibble: 2 x 5
##   term      estimate std.error statistic    p.value
##   <chr>      <dbl>     <dbl>     <dbl>      <dbl>
## 1 (Intercept) -3.84      1.17     -3.27 0.00181
## 2 log(height)  1.58      0.228      6.93 0.00000000410
```

A 1% increase in height is associated with a 1.58% increase in mass!