

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 do environmental economics research

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

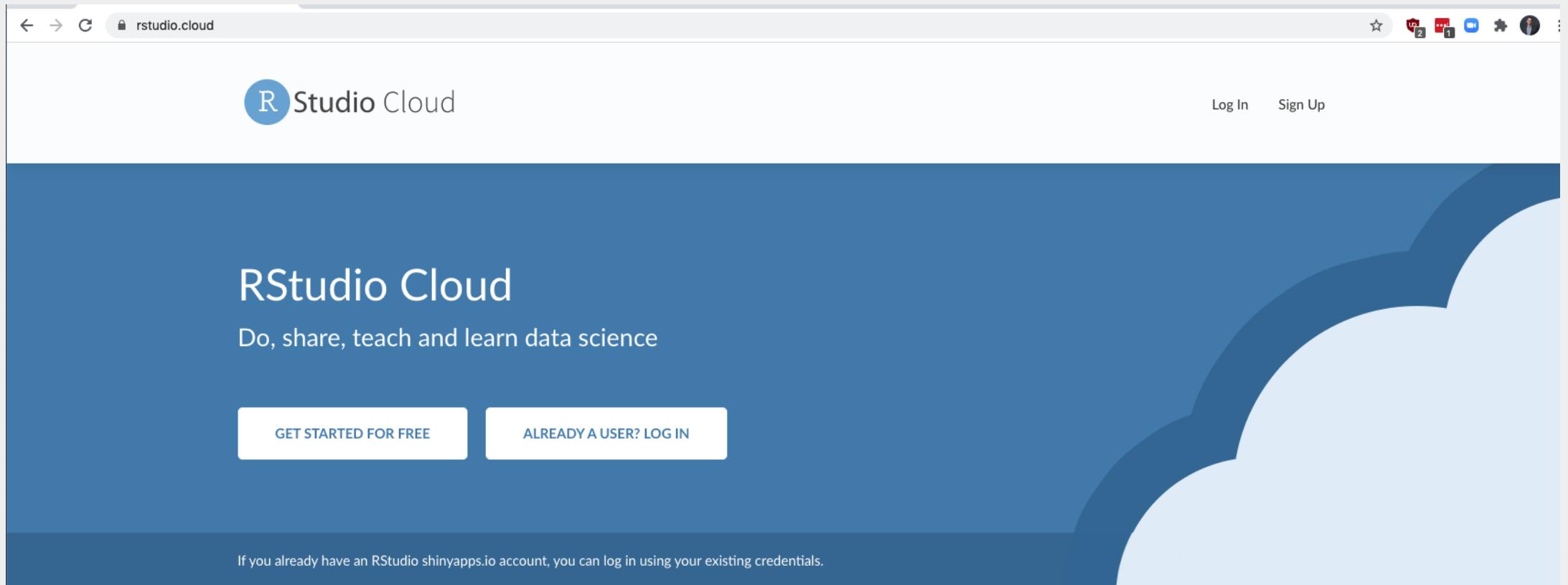
Why?

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 bottom part of the page has a white background with a dark blue footer bar.

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 from github

The screenshot shows the RStudio Cloud interface at rstudio.cloud/projects. The left sidebar includes sections for Spaces (Your Workspace selected), Learn (Guide, What's New, Primers, Cheat Sheets), Help (Current System Status, RStudio Community), and Info (Plans & Pricing, Terms and Conditions). The main content area displays 'Your Projects' with a 'Sort By name' dropdown and a 'no projects' message. A 'New Project' button and a 'New Project from Git Repository' link are available. On the right, an 'Info' panel for user 'Ivan Rudik' shows account usage: 'Projects: 0 of 15' and 'Project hours: 0.4 of 15'. It also includes a link to upgrade the account.

Your Workspace Projects About

R Studio Cloud Ivan Rudik

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

Your Projects

Sort By name

New Project

New Project from Git Repository

no projects

This is your personal workspace.

Learn more about [Your Workspace](#) in the [Guide](#).

Account Usage

Projects: 0 of 15

Project hours: 0.4 of 15

[Upgrade your account](#) to use more project hours, create more projects, and access other premium features.

6 / 122

Getting started: new project from github

The screenshot shows the RStudio Cloud interface at rstudio.cloud/projects. The left sidebar includes links for 'Your Workspace' (selected), 'New Space', 'Guide', 'What's New', 'Primer', 'Cheat Sheets', 'Current System Status', and 'RStudio Community'. The main area displays 'Your Projects' with a sorting dropdown set to 'By name'. A central modal window titled 'New Project from Git Repository' contains a text input field with the URL <https://github.com/irudik/aem6510>. Below the input is an 'OK' button. To the right of the modal is an 'Info' panel with the message: 'This is your personal workspace. Learn more about Your Workspace in the Guide.' At the bottom right of the page is an 'Account Usage' section showing 'Projects: 0 of 15' and 'Project hours: 0.4 of 15', along with a link to upgrade the account.

Getting started: wait for deployment

[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



8

Click on class-code in bottom-right

The screenshot shows the RStudio Cloud interface for a workspace named 'aem6510'. The left sidebar contains links for 'Your Workspace', 'New Space', 'Learn', 'Guide', 'What's New', 'Primers', 'Cheat Sheets', 'Help', 'Current System Status', and 'RStudio Community'. The main area has tabs for 'Console' (active), 'Terminal', and 'Jobs'. The 'Console' tab displays the R startup message and a single command prompt line starting with '>'. The 'Environment' tab shows an 'Empty Environment'. The 'Files' tab displays a file tree with the following contents:

Name	Size	Modified
..		
.gitignore	570 B	Oct 20, 2020, 1:48 PM
.Rhistory	0 B	Oct 20, 2020, 1:48 PM
aem6510.Rproj	205 B	Oct 20, 2020, 1:48 PM
class-code		
lecture-notes		
LICENSE	1 KB	Oct 20, 2020, 1:48 PM
README.md	11.9 KB	Oct 20, 2020, 1:48 PM

Click on class-code.Rproj

The screenshot shows the RStudio Cloud interface for a workspace named 'aem6510'. The left sidebar contains links for 'Spaces', 'Learn', 'Help', and 'Info'. The main area has tabs for 'Console', 'Terminal', and 'Jobs'. The 'Console' tab shows the message 'Restarting R session...'. The 'Environment' tab indicates 'Environment is empty'. The 'Files' tab shows a directory structure for 'Cloud > project > class-code' with files: '.Rprofile' (26 B, Oct 20, 2020, 1:48 PM), 'class-code.Rproj' (205 B, Oct 20, 2020, 1:48 PM), 'code-here.R' (101 B, Oct 20, 2020, 1:48 PM), 'renv' (17.6 KB, Oct 20, 2020, 1:48 PM), and 'renv.lock'.

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

File Edit Code View Plots Session Build Debug Profile Tools Help

Console Terminal Jobs

/cloud/project/

Restarting R session...

Environment History Connections Git Tutorial

Import Dataset Global Environment

Environment is empty

Files Plots Packages Help Viewer

New Folder Upload Delete Rename More

Cloud > project > class-code

Name	Size	Modified
..		
.Rprofile	26 B	Oct 20, 2020, 1:48 PM
class-code.Rproj	205 B	Oct 20, 2020, 1:48 PM
code-here.R	101 B	Oct 20, 2020, 1:48 PM
renv		
renv.lock	17.6 KB	Oct 20, 2020, 1:48 PM

Click yes

The screenshot shows the RStudio Cloud interface. On the left, there's a sidebar with sections like 'Spaces', 'Learn', 'Help', and 'Info'. The main area has tabs for 'Console', 'Terminal', and 'Jobs'. The 'Console' tab shows the message 'Restarting R session...'. To the right, there are panes for 'Environment', 'Files', and 'Plots'. A 'Confirm Open Project' dialog box is overlaid on the interface, asking 'Do you want to open the project /cloud/project/class-code?'. The 'Yes' button is highlighted with a blue border.

rstudio.cloud/project/1795327

R Studio Cloud Your Workspace / aem6510

File Edit Code View Plots Session Build Debug Profile Tools Help

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

Console Terminal Jobs

/cloud/project/

Restarting R session...

Environment History Connections Git Tutorial

Import Dataset

Global Environment

Environment is empty

Files Plots Packages Help Viewer

New Folder Upload Delete Rename More

Cloud > project > class-code

Confirm Open Project

Do you want to open the project /cloud/project/class-code?

Yes No

renv.lock

	Size	Modified
.Rproj	26 B	Oct 20, 2020, 1:48 PM
.R	205 B	Oct 20, 2020, 1:48 PM
renv.lock	101 B	Oct 20, 2020, 1:48 PM
	17.6 KB	Oct 20, 2020, 1:48 PM

11 / 122

Check package status (not required)

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

- Left Sidebar:** Includes links for "Your Workspace", "New Space", "Learn" (Guide, What's New, Primers, Cheat Sheets), "Help" (Current System Status, RStudio Community), and "Info" (Plans & Pricing, Terms and Conditions).
- Header:** Shows the URL "rstudio.cloud/project/1795327", the RStudio Cloud logo, and user "Ivan Rudik".
- Console Tab:** Displays the R environment setup and a command to check package status.
- Environment Tab:** Shows the Global Environment pane which is currently empty.
- Files Tab:** Shows a file tree for the project "class-code".

```
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

* Downloading renv 0.12.0 from CRAN ... trying URL 'http://package-proxy/src/contrib/renv_0.12.0.tar.gz'
Content type 'application/x-tar' length 1011356 bytes (987 KB)
=====
downloaded 987 KB

OK
* Installing renv 0.12.0 ... Done!
Successfully installed and loaded renv 0.12.0.
* Project '/cloud/project/class-code' loaded. [renv 0.12.0]
* The project may be out of sync -- use `renv::status()` for more details.
> renv::status()
```

Name	Size	Modified
..		
.Rprofile	26 B	Oct 20, 2020, 1:48 PM
class-code.Rproj	205 B	Oct 20, 2020, 1:52 PM
code-here.R	101 B	Oct 20, 2020, 1:48 PM
renv		
renv.lock	17.6 KB	Oct 20, 2020, 1:48 PM

Install packages (then press 'y')

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

- Console Tab:** Displays the command `library("tidyverse")` and its output, listing numerous packages and their versions.
- Environment Tab:** Shows the "Global Environment" pane with the message "Environment is empty".
- File Browser:** Shows the project structure under "Cloud > project > class-code".
- Bottom Status Bar:** Shows the command `> tidyverse::restore()`.

Key output from the Console tab:

```
/cloud/project/class-code/~/
[1] "tidyverse" [1] "0.3.1"
[2] "dplyr"     [2] "0.8.0"
[3] "ggplot2"   [3] "3.3.0"
[4] "gridExtra" [4] "2.3.0"
[5] "grid"      [5] "3.3.0"
[6] "grid"      [6] "3.3.0"
[7] "grid"      [7] "3.3.0"
[8] "grid"      [8] "3.3.0"
[9] "grid"      [9] "3.3.0"
[10] "grid"     [10] "3.3.0"
[11] "grid"     [11] "3.3.0"
[12] "grid"     [12] "3.3.0"
[13] "grid"     [13] "3.3.0"
[14] "grid"     [14] "3.3.0"
[15] "grid"     [15] "3.3.0"
[16] "grid"     [16] "3.3.0"
[17] "grid"     [17] "3.3.0"
[18] "grid"     [18] "3.3.0"
[19] "grid"     [19] "3.3.0"
[20] "grid"     [20] "3.3.0"
[21] "grid"     [21] "3.3.0"
[22] "grid"     [22] "3.3.0"
[23] "grid"     [23] "3.3.0"
[24] "grid"     [24] "3.3.0"
[25] "grid"     [25] "3.3.0"
[26] "grid"     [26] "3.3.0"
[27] "grid"     [27] "3.3.0"
[28] "grid"     [28] "3.3.0"
[29] "grid"     [29] "3.3.0"
[30] "grid"     [30] "3.3.0"
[31] "grid"     [31] "3.3.0"
[32] "grid"     [32] "3.3.0"
[33] "grid"     [33] "3.3.0"
[34] "grid"     [34] "3.3.0"
[35] "grid"     [35] "3.3.0"
[36] "grid"     [36] "3.3.0"
[37] "grid"     [37] "3.3.0"
[38] "grid"     [38] "3.3.0"
[39] "grid"     [39] "3.3.0"
[40] "grid"     [40] "3.3.0"
[41] "grid"     [41] "3.3.0"
[42] "grid"     [42] "3.3.0"
[43] "grid"     [43] "3.3.0"
[44] "grid"     [44] "3.3.0"
[45] "grid"     [45] "3.3.0"
[46] "grid"     [46] "3.3.0"
[47] "grid"     [47] "3.3.0"
[48] "grid"     [48] "3.3.0"
[49] "grid"     [49] "3.3.0"
[50] "grid"     [50] "3.3.0"
[51] "grid"     [51] "3.3.0"
[52] "grid"     [52] "3.3.0"
[53] "grid"     [53] "3.3.0"
[54] "grid"     [54] "3.3.0"
[55] "grid"     [55] "3.3.0"
[56] "grid"     [56] "3.3.0"
[57] "grid"     [57] "3.3.0"
[58] "grid"     [58] "3.3.0"
[59] "grid"     [59] "3.3.0"
[60] "grid"     [60] "3.3.0"
[61] "grid"     [61] "3.3.0"
[62] "grid"     [62] "3.3.0"
[63] "grid"     [63] "3.3.0"
[64] "grid"     [64] "3.3.0"
[65] "grid"     [65] "3.3.0"
[66] "grid"     [66] "3.3.0"
[67] "grid"     [67] "3.3.0"
[68] "grid"     [68] "3.3.0"
[69] "grid"     [69] "3.3.0"
[70] "grid"     [70] "3.3.0"
[71] "grid"     [71] "3.3.0"
[72] "grid"     [72] "3.3.0"
[73] "grid"     [73] "3.3.0"
[74] "grid"     [74] "3.3.0"
[75] "grid"     [75] "3.3.0"
[76] "grid"     [76] "3.3.0"
[77] "grid"     [77] "3.3.0"
[78] "grid"     [78] "3.3.0"
[79] "grid"     [79] "3.3.0"
[80] "grid"     [80] "3.3.0"
[81] "grid"     [81] "3.3.0"
[82] "grid"     [82] "3.3.0"
[83] "grid"     [83] "3.3.0"
[84] "grid"     [84] "3.3.0"
[85] "grid"     [85] "3.3.0"
[86] "grid"     [86] "3.3.0"
[87] "grid"     [87] "3.3.0"
[88] "grid"     [88] "3.3.0"
[89] "grid"     [89] "3.3.0"
[90] "grid"     [90] "3.3.0"
[91] "grid"     [91] "3.3.0"
[92] "grid"     [92] "3.3.0"
[93] "grid"     [93] "3.3.0"
[94] "grid"     [94] "3.3.0"
[95] "grid"     [95] "3.3.0"
[96] "grid"     [96] "3.3.0"
[97] "grid"     [97] "3.3.0"
[98] "grid"     [98] "3.3.0"
[99] "grid"     [99] "3.3.0"
[100] "grid"    [100] "3.3.0"
[101] "grid"    [101] "3.3.0"
[102] "grid"    [102] "3.3.0"
[103] "grid"    [103] "3.3.0"
[104] "grid"    [104] "3.3.0"
[105] "grid"    [105] "3.3.0"
[106] "grid"    [106] "3.3.0"
[107] "grid"    [107] "3.3.0"
[108] "grid"    [108] "3.3.0"
[109] "grid"    [109] "3.3.0"
[110] "grid"    [110] "3.3.0"
[111] "grid"    [111] "3.3.0"
[112] "grid"    [112] "3.3.0"
[113] "grid"    [113] "3.3.0"
[114] "grid"    [114] "3.3.0"
[115] "grid"    [115] "3.3.0"
[116] "grid"    [116] "3.3.0"
[117] "grid"    [117] "3.3.0"
[118] "grid"    [118] "3.3.0"
[119] "grid"    [119] "3.3.0"
[120] "grid"    [120] "3.3.0"
[121] "grid"    [121] "3.3.0"
[122] "grid"    [122] "3.3.0"
[123] "grid"    [123] "3.3.0"
[124] "grid"    [124] "3.3.0"
[125] "grid"    [125] "3.3.0"
[126] "grid"    [126] "3.3.0"
[127] "grid"    [127] "3.3.0"
[128] "grid"    [128] "3.3.0"
[129] "grid"    [129] "3.3.0"
[130] "grid"    [130] "3.3.0"
[131] "grid"    [131] "3.3.0"
[132] "grid"    [132] "3.3.0"
[133] "grid"    [133] "3.3.0"
[134] "grid"    [134] "3.3.0"
[135] "grid"    [135] "3.3.0"
[136] "grid"    [136] "3.3.0"
[137] "grid"    [137] "3.3.0"
[138] "grid"    [138] "3.3.0"
[139] "grid"    [139] "3.3.0"
[140] "grid"    [140] "3.3.0"
[141] "grid"    [141] "3.3.0"
[142] "grid"    [142] "3.3.0"
[143] "grid"    [143] "3.3.0"
[144] "grid"    [144] "3.3.0"
[145] "grid"    [145] "3.3.0"
[146] "grid"    [146] "3.3.0"
[147] "grid"    [147] "3.3.0"
[148] "grid"    [148] "3.3.0"
[149] "grid"    [149] "3.3.0"
[150] "grid"    [150] "3.3.0"
[151] "grid"    [151] "3.3.0"
[152] "grid"    [152] "3.3.0"
[153] "grid"    [153] "3.3.0"
[154] "grid"    [154] "3.3.0"
[155] "grid"    [155] "3.3.0"
[156] "grid"    [156] "3.3.0"
[157] "grid"    [157] "3.3.0"
[158] "grid"    [158] "3.3.0"
[159] "grid"    [159] "3.3.0"
[160] "grid"    [160] "3.3.0"
[161] "grid"    [161] "3.3.0"
[162] "grid"    [162] "3.3.0"
[163] "grid"    [163] "3.3.0"
[164] "grid"    [164] "3.3.0"
[165] "grid"    [165] "3.3.0"
[166] "grid"    [166] "3.3.0"
[167] "grid"    [167] "3.3.0"
[168] "grid"    [168] "3.3.0"
[169] "grid"    [169] "3.3.0"
[170] "grid"    [170] "3.3.0"
[171] "grid"    [171] "3.3.0"
[172] "grid"    [172] "3.3.0"
[173] "grid"    [173] "3.3.0"
[174] "grid"    [174] "3.3.0"
[175] "grid"    [175] "3.3.0"
[176] "grid"    [176] "3.3.0"
[177] "grid"    [177] "3.3.0"
[178] "grid"    [178] "3.3.0"
[179] "grid"    [179] "3.3.0"
[180] "grid"    [180] "3.3.0"
[181] "grid"    [181] "3.3.0"
[182] "grid"    [182] "3.3.0"
[183] "grid"    [183] "3.3.0"
[184] "grid"    [184] "3.3.0"]

```

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 assignment¹

¹ 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 assignment¹

`←` is normally read aloud as "gets"

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

¹ 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

```
a ← 10 + 5  
a
```

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

An arrow can point in the other direction too (i.e. →), this is used less frequently though

```
10 + 5 → a  
a
```

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

Assignment

You can also use `=` for assignment, but the object must be on the left

```
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 use `=` for both

Which assignment operator to use?

Most R folks prefer `←` for assignment

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

Use whatever you prefer, just be consistent

Help

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

```
?Negate
```

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 object details but here are some objects that we'll be working with regularly:

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

What are objects?

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

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

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

Global environment

```
## 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
```

Global environment

```
## 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`

Global environment

```
## 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 = runif(10))  
df
```

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

```
df2
```

```
##           x          y  
## 1 -1.43614571 0.8393892  
## 2 -0.62925965 0.2673705  
## 3  0.24352177 0.1951011  
## 4  1.05836223 0.2513002  
## 5  0.83134882 0.3891329  
## 6  0.10521182 0.8435996  
## 7 -1.71712226 0.2225674  
## 8  0.52052052 0.52052052  
## 9  0.12345679 0.12345679  
## 10 0.34567901 0.34567901
```

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
```

Other ones are `pi`, `e`, etc

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.¹

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.¹

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)
```

```
## [1] 0.31561282 0.66029338 -1.72220241 -2.13462605 0.06894560 0.86782174 -2.29004418 -0.1501902  
## [10] 1.79133204 0.67226804 -0.20930114 0.01218251 1.53411686 0.07729182 0.07843753 -0.7792610  
## [19] 0.26532457 0.89078071 -0.46788837 0.75837456 -0.64173636 0.62767182 0.24833013 -0.7000758  
## [28] -0.26139393 -1.06388504 -0.10636865 0.77110374 2.74740354 -0.08393476 0.54356763 0.7528612  
## [37] 1.00111985 0.45605250 -1.43425031 -0.26530482 0.64176916 -0.41502103 -0.45957568 -0.7924940  
## [46] 0.71089000 1.26760175 -0.14315106 -0.51502891 1.48289118 -0.16258891 0.04170917 0.4830399  
## [55] -0.66357374 -0.63464989 -0.70196303 0.57685038 -2.11308037 0.26090968 1.14712719 0.0147936  
## [64] -0.95619611 0.47341376 -1.51386408 0.16428100 -0.87086522 1.59332899 0.64659752 0.3573696  
## [73] -0.67526683 0.97208502 0.75586993 -0.42828562 -0.71392476 -0.19038407 0.39986481 -0.9778449  
## [82] -2.15031053 -0.62296653 -0.76543932 0.46430942 0.52228217 0.00979376 -0.44052620 1.1994895  
## [91] 0.03820979 1.19480563 0.34395835 -0.32907297 1.67085792 -0.91805820 -0.08780733 1.3202937  
## [100] 2.16259608
```

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')`

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

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>
```

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

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

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

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)`
- You could also use a combination of `mutate`, `group_by`, and `n()`, e.g.
`starwars %>% group_by(species) %>% mutate(num = n())`.

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

We merge data with **join operations**:

- inner_join(df1, df2)
- left_join(df1, df2)
- right_join(df1, df2)
- full_join(df1, df2)
- semi_join(df1, df2)
- anti_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>, distance <dbl>
## #   hour <dbl>, minute <dbl>, time_hour <dbl>, tailnum <chr>
```

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) tidy::pivot_longer

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

```
##          time        X        Y        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 X      -0.658
## 2 2009-01-01 Y       0.368
## 3 2009-01-01 Z       4.38
## 4 2009-01-02 X      -1.05
## 5 2009-01-02 Y      -0.549
## 6 2009-01-02 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           X     Y     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 X         -0.658    -1.05
## 2 Y          0.368   -0.549
## 3 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           X     Y     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 X         -0.658    -1.05
## 2 Y          0.368   -0.549
## 3 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=".")`

3) `tidyr::separate`

A related function is `separate_rows`, for splitting up cells that contain multiple fields or observations (a frustratingly common occurrence with survey data)

```
jobs ← data.frame(  
  name = c("Jack", "Jill"),  
  occupation = c("Homemaker", "Philosopher, Philanthropist, Troublemaker")  
)  
jobs
```

```
##      name          occupation  
## 1 Jack           Homemaker  
## 2 Jill Philosopher, Philanthropist, Troublemaker
```

3) `tidyr::separate`

A related function is `separate_rows`, for splitting up cells that contain multiple fields or observations (a frustratingly common occurrence with survey data)

```
## Now split out Jill's various occupations into different rows
jobs %>% separate_rows(occupation)
```

```
## # A tibble: 4 x 2
##   name  occupation
##   <chr> <chr>
## 1 Jack   Homemaker
## 2 Jill   Philosopher
## 3 Jill   Philanthropist
## 4 Jill   Troublemaker
```

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
```

Other tidyverse goodies

Use `crossing` to get the full combination of a group of variables

```
crossing(side=c("left", "right"), height=c("top", "bottom"))
```

```
## # A tibble: 4 x 2
##   side  height
##   <chr> <chr>
## 1 left   bottom
## 2 left   top
## 3 right  bottom
## 4 right  top
```

Other tidyverse goodies

Use `crossing` to get the full combination of a group of variables

```
crossing(side=c("left", "right"), height=c("top", "bottom"))
```

```
## # A tibble: 4 x 2
##   side  height
##   <chr> <chr>
## 1 left   bottom
## 2 left   top
## 3 right  bottom
## 4 right  top
```

See `?expand` and `?complete` for more specialized functions that allow you to fill in (implicit) missing data or variable combinations in existing data frames

Regression

Regression

Regression

Regression

Regression

Regression

Regression

Regression

Regression

Regression

Regression