ESOC 2014 Introduction to Data Science

Fall 2020

Adriana Picoral, PhD (she/her) adrianaps@email.arizona.edu

2020-10-09

Contents

Chapter 1

Syllabus

The University of Arizona sits on the homelands of the Tohono O'odham and Pascua Yaqui, whose care and keeping of these lands allows us to be here today. Territory acknowledgements are one small part of disrupting and dismantling colonial structures.

This syllabus is subject to change if need arises.

There are two sections of this course

Tuesday & Thursday 12:30pm - 1:45pm

- Final Exam Date: December 16 (Wednesday) 1:00pm - 3:00pm

Tuesday & Thursday 2:00pm - 3:15pm

- Final Exam Date: December 14 (Monday) 3:30pm - 5:30pm

Office Hours/Free help session/Work time

- Tuesday 9:30am - 11:00am & Wednesday 1:00pm - 2:30pm

1.1 Course Description

This course provides an introduction to the various skills and considerations required for data management and analysis in business, education, and science. Particular attention will be given to learning how to use the free and open-source computing environment R, focusing on the `tidyverse` package for data science. This course is designed to be interactive and hands-on.

1.2 Course Objectives

This course aims at providing students with an understanding of the various steps in the data science workflow. Students will engage in data wrangling and exploration to provide answers to questions about the data, using the R programming language. During the semester students will work on an individual data science project to be presented to the class.

1.3 Learning Outcomes

At the end of this course, students will be able to:

1. Apply the different steps of data science as a process to derive knowledge from data
   1.1. form the question to be answered
   1.2. acquire the data to answer question
   1.3. transform and tidy data so that data analysis is possible
   1.4. explore data with understanding as the goal, which includes data visualization
   1.5. communicate data analysis results
2. Demonstrate proficiency of the steps 1.3 - 1.5 above in the R programming language and R Markdown
3. Identify and apply professional standards regarding all aspects of data ethics and privacy, including how data are stored, used, managed, analyzed, and presented
4. Demonstrate knowledge of what a data scientist is and what a career in data science requires in terms of education, and set goals and make plans in case they want to pursue data science beyond the completion of this course

Please refer to the department's undergraduate student competencies to find out how this course's learning outcomes fit into your broad education goals.

1.4 A Few Words on R and Coding

This course will be based around the programming language R which we will use within the integrated development environment (IDE) R Studio. For many of you this will be the first time programming, **AND THAT'S OK! This course is intended for beginners, and we will actively focus on building up**

your R skills over the course of the semester. Of course, there will still be challenges along the way, but you will rapidly figure out how to solve your own problems as well as to apply your current knowledge to new and exciting questions. If you are struggling I highly encourage you to take advantage of my free help sessions (see times above). Of course, Google is always a super helpful way to get insight into coding problems. Our class Slack channel will also be there so you can help each other out. You might want to watch Roger Peng's video on how to get help, which contains guidelines on what information to provide when asking a question in a public forum.

I also want to note that I highly encourage you to help each other, as data scientists are rarely working in isolation. **This does not mean you can directly share code associated with an assignment (this is a violation of UA's Code of Academic Integrity)**. What it does mean is that it is helpful to talk to each other about problems you encountered, resources you found, or provide helpful tips.

Learning to code nowadays is much easier, since a simple Google search will research in a huge amount of code that can solve any number of problems. You may use online resources (e.g., StackOverflow), but we will go over the syntax needed to solve all assignments in class. If you do use any external resources, you must explicitly cite where the code was obtained in your comments (add a direct link to the resource). I'll be checking for recycled code, and any code you re-used without a proper citation will be treated as plagiarism.

1.5 A Few Words on Technology

1. YOU MUST HAVE ACCESS TO A COMPUTER YOU CAN CODE WITH IN EVERY CLASS! We will be actively coding in R on a daily basis, and not being able to follow along will severely hamper your learning. If you do not have a laptop or yours had troubles at some point during the semester, the library offers fast and free rentals of both macs and PCs: ht tps://new.library.arizona.edu/tech/borrow. You can also take advantage of the multiple computer labs on campus: https://it.arizona.edu/service /oscr-computer-labs
2. You will have access to and will be required to retrieve all course materials from the course page on GitHub.
3. You will need to have R and R Studio installed and functioning by the second day of class. We will go over what these programs are and how to install them in the first week of class.
4. Slack participation is critical! If you are having a coding issue, first try and solve it on your own. If you're still struggling, then post it to our Slack. Essentially, if you are about to email me with a homework/class/coding

question, post it to Slack first. I'm not doing this to save me time, but rather because virtually all programmers/coders solve problems by helping each other, and thus I want you to do the same! Please register for our Slack channel.

1.6 Readings

There is no required textbook for this class. A few times we will use the book "R for Data Science" by Hadley Wickham and Garret Grolemund. This book covers how to create full data science pipelines in R (more than we'll be doing here) and is available free here: https://r4ds.had.co.nz/.

Aside from this book, there will be other required readings. I will link these readings for you on this bookdown. Some come from academic journals, and others are news articles that appear in many of the newspapers you read in print and online. For each reading, a word count and an approximate reading time will be provided. Please adjust these approximations to your own reading time, so you can plan accordingly.

It is crucial that you read all assigned readings to do well in this class. Anyone who has not done the reading will simply not be able to participate.

1.7 Assignments with Grade Breakdown

Activity	Total Percent	Unit Percent	Description
Final Project	30%	5% Project Proposal 15% Write-up 10% Oral presentation	This will be a full data science project, complete with formal write-up and presentation to the class
Midterm	20%	20%	
Sharing Code during Zoom sections (5)	10%	2%	

Activity	Total Percent	Unit Percent	Description
Data Challenges (9)	28%	3.5%	Lowest will be dropped. All assignments must completed by the date and time provided in the assignment instructions
Class Participation	10%		Participation includes both in-class and message board questions, engagement. To get full credit I should see your name or hear you in class once a week.
Intro and exit surveys	2%	1%	

Late assignments within 24 hours of due date and time will get a 20% grade penalty. Assignments submitted 24 hours after the due date and time will not get any credit.

If you are unable to complete an assignment on the due date due to an illness or another personal problem, please contact me as soon as possible so we can talk about ways to help you complete that assignment.

Any work turned in for this class needs to be distinctly developed for this class, and not work turned in for other classes.

Grade Distribution:

90-100% = A “exemplary, far beyond reqs/expectations”

80-89% = B “exceeds requirements/expectations”

70-79% = C “meets requirements/expectations”

60-69% = D “falls short of requirements/expectations”

$<$ 60% = E “repeat of course needed”

A Note About Final Grades

I do not modify final grades. I have designed this course to be highly passable for the new learner assuming they do the modest homework assignments, come to class, and participate. I'm not a difficult grader, and I build in extensive opportunities for 'easy points.' Given all this, please do not try and ask for a higher grade when end of semester rolls around.

1.8 Requirements for the Course

To succeed in this course, 2-3 hours of study time per hour of formal class time (or per unit) are required. This means that in addition to our three hours of formal class meeting time, 6-9 hours a week of study time are needed in order to meet course expectations. These hours should be spent on reading texts, working on your data challenges, researching for new information, or thinking about course content.

It's important to mention that each lesson builds upon the previous, and thus staying on top of the material is critical to your success. As mentioned before, this class is built specifically for beginners, and plenty of students who have never coded before have done extremely well. But, the reason they did is that they came to class consistently, asked questions when they had an issue, and completed their data challenges. If you miss a class, come to office hours to make up what you missed. I will do everything possible to make sure you succeeded assuming you're willing to put in the work!

1.9 Course Schedule

Here is the tentative course schedule. **Data challenges are always due before the start of class on the associated due date**. There will sometimes be other short readings and assignments. These will be posted on D2L directly after the class period in which they are assigned.

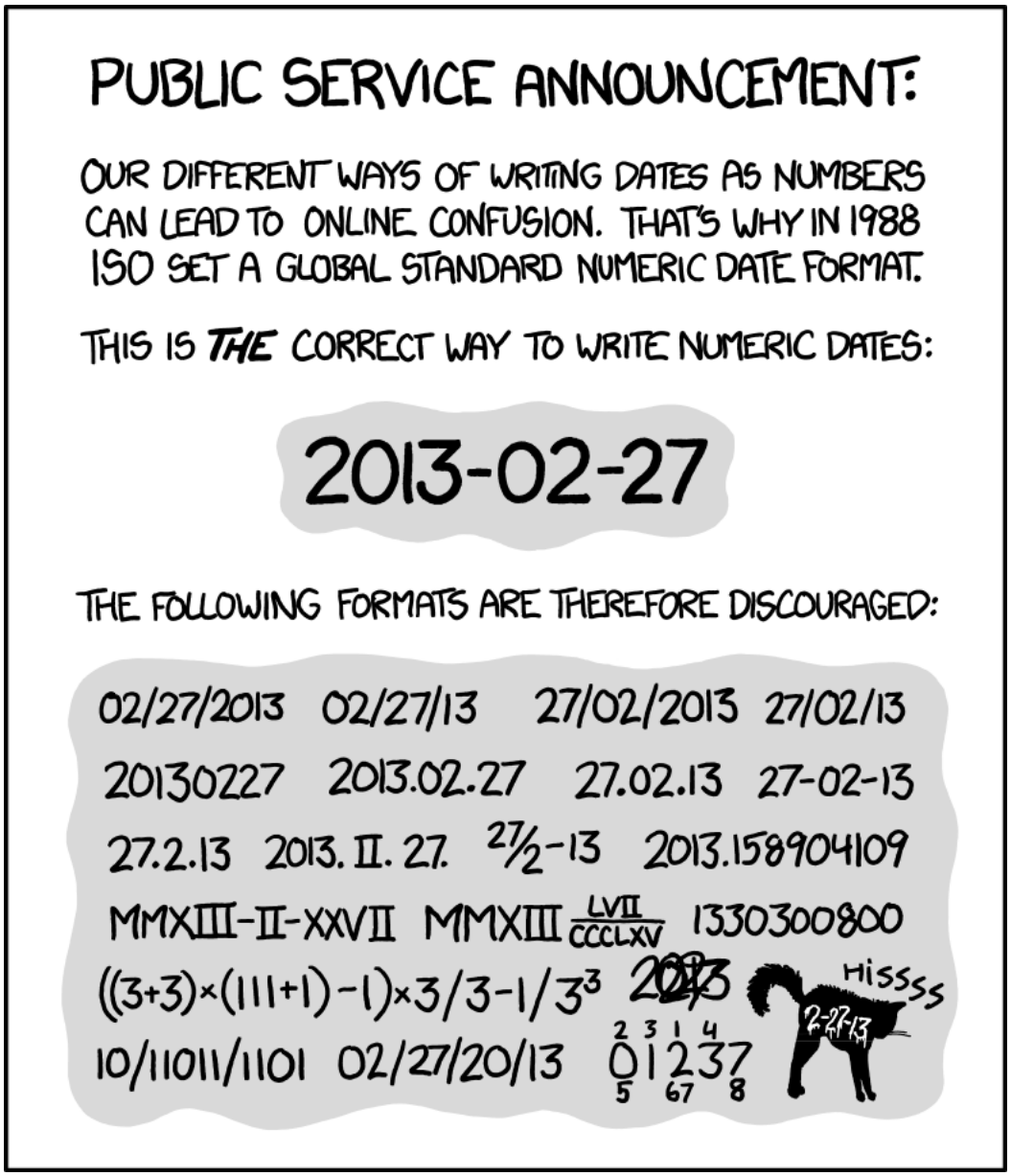
Week	Date	Goals	Assignment
Week 01	2020-08-25	Introductions Syllabus	
	2020-08-27	Intro do Data Science Data Science workflow	Reading: What's data science? (20 min) YouTube video Angry Hiring Manager Panel (6.5 min) Survey 1 (10 min)

Week	Date	Goals	Assignment
Week 02	2020-09-01	What's Data? What does data analysis look like? IDE overview How and Why to Start a Project Basics of R	Reading: Data Science examples (8 min), Data Intake (12 min) Install R and RStudio
	2020-09-03	Basics of R - basic operations - objects - data types	
Week 03	2020-09-08	Basics of R - data frames - inspecting data - slicing your data	Read A Million Lines of Bad Code (5 min) What is Statistics Good For? (3 min)
	2020-09-10	Submitting assignments through GitHub	Join our GitHub classroom
Week 04	2020-09-15	Installing R Packages Intro to Tidyverse	Read Advice to Young (and Old) Programmers: A Conversation with Hadley Wickham (10 min) Submit test assignment
	2020-09-17	Tidyverse	
Week 05	2020-09-22	Data Wrangling	Data Challenge 1
	2020-09-24	Data Wrangling	
Week 06	2020-09-29	Intro to Data Visualization	Data Challenge 2
	2020-10-01	Data Visualization	
Week 07	2020-10-06	Data Visualization	Data Challenge 3
	2020-10-08	Data Visualization	
Week 08	2020-10-13	Data analysis case study 1	Data Challenge 4
	2020-10-15	Data analysis case study 1	

Week	Date	Goals	Assignment
Week 09	2020-10-20	MIDTERM	Data Challenge 5
	2020-10-22	Data analysis case study 2	
Week 10	2020-10-27	Data analysis case study 2	
	2020-10-29	Getting Data	Data Challenge 6
Week 11	2020-11-03	Getting Data	Deadline to meet about final project
	2020-11-05	Data analysis case study 3	Project Proposal
Week 12	2020-11-10	Data analysis case study 3	
	2020-11-12	Markdown	Data Challenge 7
Week 13	2020-11-17	Markdown	
	2020-11-19	Full data analysis case study 4	Data Challenge 8
Week 14	2020-11-24	Full data analysis case study 4	
	2020-11-26	Happy Thanksgiving!	
Week 15	2020-12-01	Written and Oral Communication in Data Science	Data Challenge 9
	2020-12-03	Written and Oral Communication in Data Science	
Week 16	2020-12-08	Preparing for Final Presentations Wrap-up	

For more information about dates including holidays, check UArizona's Academic Calendar.

Why am I using YYYY-MM-DD date format?



ISO 8601 – xkcd

1.10 Final Project

There is a final project in place of a final exam for this class. You will find your own dataset that helps you answer a question that you're interested in. You'll bring these data into R, explore it, clean it, make features, and run an analysis that allows you to answer your question. You will be graded on the completed R script as well as your presentation of the data.

The presentation will last 3-4 minutes, and will take place on the day of the final exam (in place of the exam). University policy on final examinations can be found here: https://www.registrar.arizona.edu/courses/final-examination-regulations-and-information

1.11 Honors Students' Requirements

Students wishing to take this course for Honors Credit should email me to set up an appointment to discuss the terms of the contact and to sign the Honors Course Contract Request Form. The form is available at https://honors.arizona.edu/academics/honors-contracts. Students earning credit with the University of Arizona Honors College will be held to the following enhancements:

1. Honors students will be required to create an academic poster based on their final project, and then present this poster at the iSchool's iShowcase at the end of the semester. Creating a poster will require extra work to ensure clarity of logic, having a well-defined question and approach, and the creation of quality visuals. Guidelines on how to create an engaging academic poster can be found here: https://guides.nyu.edu/posters. Note: The iShowcase is at the end of the semester, but before finals when the regular class will have the project due. Thus, you will have to be ahead of schedule a bit to meet your honors requirement.
2. Honors students will also be expected to informally 'journal' about the course each week. Each week, that is, students will be required to write a five-sentence paragraph reflecting on some issue or moment that has arisen in our readings or discussions (e.g., the problem with particular terms or some philosophical or practical dilemma). Ultimately, if offering a paragraph each week, honors students will have written roughly 15 reflective paragraphs for the semester. This must be emailed directly to me by Sunday 5pm each week.

1.12 Student Accommodations

It is the University's goal that learning experiences be as accessible as possible. If you anticipate or experience physical or academic barriers based on disability

or pregnancy, please let me know immediately so that we can discuss options. You are also welcome to contact Disability Resources (520-621-3268) to establish reasonable accommodations. For additional information on Disability Resources and reasonable accommodations, please visit http://drc.arizona.edu/.

1.13 Attendance, Due Dates, and Missing Work

1. Missed class assignments or exams cannot be made up without a well-documented, verifiable, excuse (for example, a physician's medical excuse). Indeed, due dates are firm, and late work will be accepted only with a verifiable and valid excuse.
2. The UA policy regarding absences for any sincerely held religious belief, observance or practice will be accommodated where reasonable, http://policy.arizona.edu/human-resources/religious-accommodation-policy.
3. Absences pre-approved by the UA Dean of Students (or Dean designee) will be honored. https://deanofstudents.arizona.edu/absences

4. Arriving late and leaving early is extremely disruptive to others in the class. Please avoid this kind of disruption.
5. The UA's policy concerning Class Attendance and Administrative Drops is available at: https://catalog.arizona.edu/policy/class-attendance-participation-and-administrative-drop

1.14 Course Conduct and Campus Policies

It's important to be familiar with all campus policies.

1. Students are encouraged to share intellectual views and discuss freely the principles and applications of course materials. However, graded work/exercises must be the product of independent effort unless otherwise instructed. Students are expected to adhere to the UA Code of Academic Integrity as described in the UA General Catalog. See: http://deanofstudents.arizona.edu/academic-integrity/students/academic-integrity.

2. It is the University's goal that learning experiences be as accessible as possible. If you anticipate or experience physical or academic barriers based on disability or pregnancy, please let me know immediately so that we can discuss options. You are also welcome to contact Disability Resources (520-621-3268) to establish reasonable accommodations. For additional information on Disability Resources and reasonable accommodations, please visit http://drc.arizona.edu/.

3. The UA Threatening Behavior by Students Policy prohibits threats of physical harm to any member of the University community, including to

oneself. See http://policy.arizona.edu/education-and-student-affairs/threatening-behavior-students.

4. All student records will be managed and held confidentially. http://www.registrar.arizona.edu/personal-information/family-educational-rights-and-privacy-act-1974-ferpa?topic=ferpa
5. The University is committed to creating and maintaining an environment free of discrimination; see http://policy.arizona.edu/human-resources/nondiscrimination-and-anti-harassment-policy.
6. Information contained in this syllabus, other than the grade and absence policy, may be subject to change without advance notice as deemed appropriate by the instructor.

1.15 Code of Conduct

This code of conduct is based on GitHub Community Guidelines. One of the goals of this course is to get you familiar with the data science community, and how people work and learn better together. This is a community we build together, and we need everybody's help to make it better each day.

1.15.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as instructor and students pledge to making participation in our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

- **Be welcoming and open-minded.** Although this is an intro course, like in any other learning setting, we have people at different levels of experience. Other people may not have the same experience level or background as you, but that doesn't mean they don't have good ideas to contribute. I encourage you to be welcoming to everyone, from more advanced coders to those just getting started. We can all learn from each other.
- **Respect each other.** Nothing sabotages healthy conversation like rudeness. Be civil and professional, and don't post or say anything that a reasonable person would consider offensive, abusive, or hate speech. Don't harass or grief anyone. Treat each other with dignity and consideration in all interactions.

You may wish to respond to something by disagreeing with it. That's fine. But remember to criticize ideas, not people. Avoid name-calling, ad hominem

attacks, responding to a post's tone instead of its actual content, and knee-jerk reactions. Instead, provide reasoned counter-arguments that improve the conversation.

- **Communicate with empathy.** Disagreements or differences of opinion are a fact of life. Being part of a community means interacting with people from a variety of backgrounds and perspectives (and we are all better because of this variety), many of which may not be your own. If you disagree with someone, try to understand and share their feelings before you address them. This will promote a respectful and friendly atmosphere where people feel comfortable asking questions, participating in discussions, and making contributions.
- **Be clear and stay on topic.** The goal of this course is to learn about data science and how to do data science with R. Off-topic comments are a distraction (sometimes welcome, but usually not) from getting work done and being productive. Staying on topic helps produce positive and productive discussions.

Additionally, as this class will be conducted online, you might not have met each other in person. Communicating on the internet can be awkward, even when you already know people. It's hard to convey or read tone, and sarcasm is frequently misunderstood. Try to use clear language, and think about how it will be received by the other person.

1.15.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission

- Other conduct which could reasonably be considered inappropriate in a professional setting

1.15.3 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting your instructor at adrianaps@email.arizona. Your instructor will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. Your instructor is obligated to maintain confidentiality with regard to the reporter of an incident.

1.15.4 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4

1.16 How to Ask For Help

We'll see in this course that a key skill that you should develop as a data science is the ability to find solutions to problems. Knowing how to get help is part of that skill.

1.16.1 Before You ask for help

- **Check for typos.** One of the most common causes of errors are typos, which usually throw an error such as Error in _____ : could not find function "_____" due to a function being misspelled
- **Check loaded packages.** You also get errors like Error in data %>% summary() : could not find function "%>%" when you failed to load a package.
- **Read the error message.** Don't ignore what R is telling you. Be aware that red text that appears in your console is not alwayws indication of errors. Sometimes it's just a warning.
- **Google is your friend.** Copy and paste the exact error message on a Google search. (this step also includes **read the documentation** on the package you're trying to use).
- If you are still stuck, you an always try **rubber duck debugging**. Describe the problem aloud, explaining it line-by-line, to a rubber duck or another person (who might not have any experience with programming of

data science). This is also a good preparation step to asking other people for help (next section).

1.16.2 Ask other people for help

Like mentioned before, you should ask your peers for help before you ask your instructor. Relying on a single person to solve all of your problems is dangerous, because that person won't be available throughout your career as a data scientist.

- Check **our Slack** to see if someone else has asked a question similar to yours, and whether there's a solution posted for it.
- **Be precise and informative**. The more context you can provide about what you're trying to do and what errors you're getting, the better. Also describe the steps you took to try to solve the problem yourself.

1.16.3 List of Resources

- Getting Help with R
- Roger Peng's How To Get Help video
- Rubber Duck Debugging

Chapter 2

What's data science?

2.1 Before class #1

Required external reading for this module: What's data science? (4,660 words, approx. 20 minutes of reading time)

Watch the YouTube video Angry Hiring Manager Panel from 10:18 to 16:48 (6.5 minutes) and list the skills they mention as important to have in a data science position.

Fill out Survey 1 (10 min)

2.2 What's data science?

Data science is one of the fields with the highest demand, with prospects of increased demand for the next decade (Kross et al., 2020; Hadavand et al., 2018). Interestingly, the data scientist title was invented in 2008, and the median base salary for a data scientist surpassed $100,000 in the United States in 2019 (Robinson and Nolis, 2020).

CHALLENGE

Based on your own experience and on your reading for this module, in your groups discuss the following question:

- What is data science?

2.3 What does a data scientist do?

Data science is an interdisciplinary field, and as such data scientists hold jobs with a broad range of skills, from statistics to communication. A quick search for data science jobs reveals this long list of skills. However, no single data scientist has all skills listed for different data science jobs. Instead, each data scientist specializes in different skills (Robinson and Nolis, 2020).

CHALLENGE

Make a list of skills listed on data science job announcements and in the video you just watched. Based on these, discuss the following questions in your group:

1) Which skills do you already have? At what level of proficiency?
2) Which skills are you interested in developing further?
3) Based on the skills you already have, and the skills you want to acquire, what type of job in data science would you be interested in?

2.4 Data Science Workflow

The basic data science workflow involve three main parts:

1) The Question: Form the question you want to answer. Many times you will be given a question, and you have to "translate" it so you can answer it with your data analysis.
2) Data Acquisition: data file, database, or web API
3) Data Wrangling: import + tidy data + transform (Grolemund and Wickham, 2018)
4) Data Exploration: transform + visualize + model + repeat (Grolemund and Wickham, 2018)
5) Results Communication: visualize + write + knit (Grolemund and Wickham, 2018)

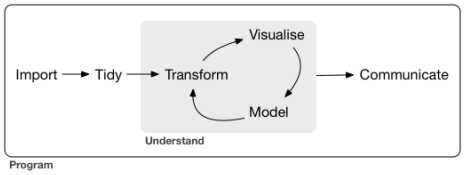


Figure 2.1: typical data science project model (Grolemund and Wickham, 2018)

CHALLENGE

In your groups, based on your own intuition and experience, and based on the Introduction to R for Data Science book (Grolemund and Wickham, 2018), summarize what each of the following steps means:

1) Tidy
2) Transform
3) Visualize
4) Model
5) Communicate

We will approach the entire data science workflow in this course (but not necessarily every step listed), not in this order. We start with step 3 (Data Wrangling) and 4 (Data Exploration), before we address step 2 (Data Acquisition) and step 5 (Communication)

CHALLENGE

Go back to the list of skills and job positions we discussed (based on the reading and the video):

1) Which steps in the data science workflow correspond to the job skills we talked about?

2.5 Before class #2

Please fill out Survey 1 (10 min).

Reading: Data Science examples (1,0333 words, 8 min)

Reading: Data Intake (1,686 words, 12 min)

2.6 What's data?

CHALLENGE

In your small group, discuss the examples provided in the excerpt from "Executive Data Science" (Caffo et al., 2016).

1) Is data science about "data"? Why or why not?
2) Why did Netflix end up not implementing the best solution from the Netflix prize challenge?
3) What data was used in each of the examples provided in the reading?
4) What is data? (come up with a definition).

Examples of what data might look like.

- Structured data (rare):

State

School Year

Average Tuition

Nevada

2004-05

3621.392

Nevada

2005-06

3687.290

Florida

2004-05

3848.201

Florida

2007-08

3879.416

Florida

2006-07

3887.656

Florida

2005-06

3924.234

Wyoming

2008-09

3928.671

Wyoming

2007-08

4071.898

Wyoming

2004-05

4086.351

Wyoming

2006-07

4122.205

CHALLENGE

Which of the columns (or variables) in the data frame above are **categorical**, which are **quantitative**?

- Structured, but messy data (more common):

State

2004-05

2005-06

2006-07

2007-08

2008-09

2009-10

2010-11

2011-12

2012-13

2013-14

2014-15

2015-16

Alabama

5682.838

5840.550

5753.496

6008.169

6475.092

7188.954

8071.134

8451.902

9098.069

9358.929

9496.084

9751.101

Alaska

4328.281

4632.623

4918.501

5069.822

5075.482

5454.607

5759.153

5762.421

6026.143

6012.445

6148.808

6571.340

Arizona

5138.495

5415.516

5481.419

5681.638

6058.464

7263.204

8839.605

9966.716

10133.503

10296.200

10413.844

10646.278

Arkansas
5772.302
6082.379
6231.977
6414.900
6416.503
6627.092
6900.912
7028.991
7286.580
7408.495
7606.410
7867.297
California
5285.921
5527.881
5334.826
5672.472
5897.888
7258.771
8193.739
9436.426
9360.574
9274.193
9186.824
9269.844
Colorado
4703.777
5406.967
5596.348
6227.002

6284.137

6948.473

7748.201

8315.632

8792.856

9292.954

9298.599

9748.188

Connecticut

7983.695

8249.074

8367.549

8677.702

8720.976

9371.019

9827.013

9736.431

10036.627

10453.110

10663.995

11397.337

Delaware

8352.890

8610.597

8681.846

8945.801

8995.473

9987.183

10534.181

11026.241

11362.690

11502.524

11514.660

11676.216

Florida

3848.201

3924.234

3887.656

3879.416

4150.004

4783.032

5510.659

5940.945

6494.901

6451.664

6345.000

6360.159

Georgia

4298.040

4492.167

4584.268

4790.266

4831.365

5549.913

6428.007

7709.284

7853.257

7992.390

8063.014

8446.961

user_id

screen_name

text

reply_to_screen_name

6.331283e+07

blagogirl

? The illiterate calling Iran out? 80 million bounty on Trumps head?

realTuckFrumper

6.331283e+07

blagogirl

? Iran does NOT fear Trump. They realize what OUR country is dealing with. "The White House is inflicted with mental retardation"

JonHutson

1.125104e+18

dl_kirkwood

I'm afraid 11 soldiers had to be shipped out from the Iran hit after all with traumatic brain injuries. Seems the Military does not notify homeland unless a soldiers is shipped out for the injury. So, Trump did not know for a week. https://t.co/HdBNbKClBl

NA

2.820552e+07

djbarro

? Are you going to carry a sign supporting the women in Iran brave enough to remove their hijabs and go to prison?

GloriaAllred

1.506314e+08

kizu91

US…Special…Representative…Hold…Press…Briefing…Situation in…Iran…Video…first…week…January…saw. Trump…order…assassination…elite…Quds…Force…commander…Qasem…Soleimani…Iraq

NA

1.506314e+08

kizu91

crash…land…collide…plane…aircraft…all…176…people…on board…Iran…missile…attack…US…base…Iraq…roc

NA

1.506314e+08

kizu91

Iran…MP…Urge…Gov't…Expel…UK…Envoy…Consider…Downgrading…Diplomatic…Ties…Alleged…Meddling…envoy…R
Macaire…detained…days…ago…alleged…participation…unsanctioned…protest…Tehran…down…Ukraine…Boeing…737…re

NA

1.506314e+08

kizu91

Government…Supporter…Gather…Tehran…13….Friday…Prayer…Video…Iran…gather…rally…commemorate…kill…fatal…

NA

1.506314e+08

kizu91

British…Treasury…Expand…Hezbollah…Asset…Freeze…UK…government…approved…measure…follow…heat…conflict…U
States…Islamic…Republic…Iran…Trump…Administration…target…assassination…high-
profile…military…general…early…January…film

NA

7.297365e+17

SwmpladySH

Hackers Are Coming for the 2020 Election — And We're Not Ready https:
//t.co/q82kNu9gMd via **?**

NA

- Textual Data (always messy):

```
##  [1] "CHAPTER I"
##  [2] ""
##  [3] ""
##  [4] "Emma Woodhouse, handsome, clever, and rich, with a comfortable home"
##  [5] "and happy disposition, seemed to unite some of the best blessings of"
##  [6] "existence; and had lived nearly twenty-one years in the world with very"
##  [7] "little to distress or vex her."
##  [8] ""
##  [9] "She was the youngest of the two daughters of a most affectionate,"
## [10] "indulgent father; and had, in consequence of her sister's marriage, been"
```

** CHALLENGE **

What data formats are out there in the world. Create a list based on your experience and the excerpt from "Modern Data Science with R" (Baumer et al., 2017).

2.7 What does data analysis look like?

The way you communicate your data analysis will depend on what question you're trying to answer and who your audience is. Here are some of my favorite data analysis reports:

- Whose (coffee) beans reign supreme? A #tidytuesday static image
- Women in Space A #tidytuesday static image
- Which city is faster? A City Cycle Race Shinny app
- The Physical Traits that Define Men and Women in Literature An interactice website

Chapter 3

Exploring our IDE (Rstudio)

3.1 Before class #3

Install R and RStudio.

We are using RStudio as our IDE for this course. If you are running your R code in your computer, you need to install both R and RStudio. Alternatively, you can create a free account at http://rstudio.cloud and run your R code in the cloud. Either way, we will be using the same IDE (i.e., RStudio).

What's an **IDE**? IDE stands for **i**ntegrated **d**evelopment **e**nvironment, and its goal is to facilitate coding by integrating a **text editor**, a **console** and other tools into one window.

3.1.1 I've never installed R and RStudio in my computer OR I'm not sure I have R and RStudio installed in my computer

1. Download and install R from https://cran.r-project.org (If you are a Windows user, first determine if you are running the 32 or the 64 bit version)
2. Download and install RStudio from https://rstudio.com/products/rstudio/download/#download

Here's a video on how to install R and RStudio on a mac.

3.1.2 I already have R and RStudio installed

1. Open RStudio
2. Check your R version by entering `sessionInfo()` on your console.
3. The latest release for R was June 22, 2020 (R version 4.0.2 Taking Off Again). If your R version is older than the most recent version, please follow step 1 in the previous section to update R.
4. Check your RStudio version, if your version is older than Version 1.3.x, please follow step 2 in the previous section to update RStudio.

How often should I update R and RStudio? Always make sure that you have the latest version of R, RStudio, and the packages you're using in your code to ensure you are not running into bugs that are caused by having older versions installed in your computer.

When asked, Jenny Bryan summarizes the importance of keeping your system up-to-date saying that "You will always eventually have a reason that you must update. So you can either do that very infrequently, suffer with old versions in the middle, and experience great pain at update. Or admit that maintaining your system is a normal ongoing activity, and do it more often."

You can ensure your packages are also up-to-date by clicking on "Tools" on your RStudio top menu bar, and selecting "Check for Packages Updates…"

3.2 Why learn R?

R is both a programming language and a free software environment for statistical computing and graphics. In addition to being free, here are other reasons to learn R:

- **R is popular.** According to Robert A. Muenchen's post on the popularity of data science software (which is updated frequently), R is among the top 5 technologies that are mentioned in data science job ads on indeed.com.
- **R is very powerful and versatile**. From creating websites (like this bookdown you're reading right now) to building machine learning models, R has it all.
- **The R community is active and very supportive**. Because R is so popular, there are a number of forums on R. A good way to get a glimpse on how active the R community is to follow `#rstats` on twitter.

3.3 Why use RStudio?

You can just use R, but RStudio is an IDE that makes using R easier and more fun. Some features that make RStudio the IDE that many data scientists use:

- RStudio is **free** and **open source**.
- RStudio contains a full-feature integrated text editor, with tab-completion, spellcheck, etc.
- RStudio is a cross-platform interface that looks the same across platforms.
- RStudio allows you to organize your data science projects so you're not always hunting for the right script that goes with the data you want to analyze. (also, it integrates nicely with `rmarkdown` and `knitr`)

3.4 Create an R Project

In today's class, we will focus on situating ourselves around our IDE. For every lesson, we will either start a new R project or open an R project we've been working on.

Why create a RStudio project? RStudio projects make it easier to keep your projects organized, since each project has their own working directory, workspace, history, and source documents. In other words, it's much easier to open an R project and not have to worry about setting your working directory than to try to hunt down your files.

Here are the steps we are starting with today:

1. Start a new R project
2. Create a new R script
3. Save that R script as 01-class_one

CHALLENGE

Take a moment to look around your IDE. What are the main panes on the RStudio interface. What are the 4 main areas of the interface? Can you guess what each area is for?

3.5 Operations and Objects

Let's start by using R as a calculator. On your **console** type `3 + 3` and hit enter.

```
3 + 3
```

```
## [1] 6
```

What symbols do we use for all basic operations (addition, subtraction, multiplication, and division)? What happens if you type `3 +`?

Let's save our calculation into an object, by using the assignment symbol `<-`.

```
sum_result <- 3 + 3
```

Take a moment to look around your IDE once again. What has changed?

Now, let's use this new object in our calculation

```
sum_result + 3
```

```
## [1] 9
```

Take a moment to look around your IDE once again. Has anything changed?

What else can we do with an object?

```
class(sum_result)
```

```
## [1] "numeric"
```

R is primarily a functional programming language. That means that there pre-programmed functions in base R such as `class()` and that you can also write your own functions (more on that later).

Type `?class` in your console and hit enter to get more information about this function.

CHALLENGE

Create an object called `daisys_age` that holds the number 8. Multiply `daisys_age` by 4 and save the results in another object called `daisys_human_age`

Imagine I had multiple pets (unfortunately, that is not true, Daisy is my only pet). I can create a **vector** to hold multiple numbers representing the age of each of my pets.

```
my_pets_ages <- c(8, 2, 6, 3, 1)
```

Take a moment to look around your IDE once again. What has changed?

What is the class of the object `my_pets_ages`?

Now let's multiply this vector by 4.

```
my_pets_ages * 4
```

```
## [1] 32  8 24 12  4
```

Errors are pretty common when writing code in any programming language, so be ready to read error messages and debug your code. Let's insert a typing error in our previous code:

```
my_pets_ages <- c(8, 2, 6, '3', 1)
```

CHALLENGE

Try to multiply `my_pets_ages` by 4. What happens? How can we debug our code to find out what is causing the error?

Chapter 4

R Basics

4.1 Before Class #4

Read A Million Lines of Bad Code a blog post by David Robinson. (549 words, 5 minutes)

Read What is Statistics Good For? (398 words, 3 min)

4.2 Dataframes

You will rarely work with individual numeric values, or even individual numeric vectors. Often, we have information organized in dataframes, which is R's version of a spreadsheet.

Let's go back to my imaginary pet's ages (make sure you have the correct vector in your global environment).

```
my_pets_ages <- c(8, 2, 6, '3', 1)
my_pets_ages <- as.numeric(my_pets_ages)
```

We will now create a vector of strings or characters that holds my imaginary pets' names (we have to be careful to keep the same order then the `my_pets_ages` vector).

```
my_pets_names <- c('Daisy', 'Violet', 'Lily', 'Iris', 'Poppy')
```

Let's now create a dataframe that contains info about my pets.

```
# create dataframe
my_pets <- data.frame(name = my_pets_names, age = my_pets_ages)
```

```
# print out dataframe
my_pets
```

```
##     name age
## 1  Daisy   8
## 2 Violet   2
## 3   Lily   6
## 4   Iris   3
## 5  Poppy   1
```

CHALLENGE

There's a number of functions you can run on dataframes. Try running the following functions on `my_pets`:

- summary()
- nrow()
- ncol()
- dim()

What other functions can/do you think/know of?

4.3 Slicing your dataframe

There are different ways you can slice or subset your dataframe.

You can use indices for rows and columns.

```
my_pets[1,]
```

```
##    name age
## 1 Daisy   8
```

```
my_pets[, 1]
```

```
## [1] "Daisy"  "Violet" "Lily"   "Iris"   "Poppy"
```

```
my_pets[1, 1]
```

```
## [1] "Daisy"
```

You can use a column name or a row name instead of an index.

```
my_pets[, 'age']
```

```
## [1] 8 2 6 3 1
```

```
my_pets['1', ]
```

```
##    name age
## 1 Daisy   8
```

```
my_pets['1', 'age']
```

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

Or you can use `$` to retrieve values from a column.

```
my_pets$age
```

```
## [1] 8 2 6 3 1
```

```
my_pets$age[1]
```

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

You can also use comparisons to filter your dataframe

```
# get index with which() function
which(my_pets$age == 8)
```

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

```
# use which() inside dataframe indexing my_pets[row_number, column_number]
my_pets[which(my_pets$age == 8),]
```

```
##    name age
## 1 Daisy   8
```

```
my_pets[which(my_pets$age == 8), 1]
```

```
## [1] "Daisy"
```

```
my_pets[which(my_pets$age == 8), 'name']
```

```
## [1] "Daisy"
```

```
my_pets[which(my_pets$age == 8),]$name
```

```
## [1] "Daisy"
```

CHALLENGE

Print out a list of pet names that are older than 3.

4.4 Adding new variables (i.e., columns) to your dataframe

So far the `my_pets` dataframe has two columns: name and age.

Let's add a third column with the pets' ages in human years. For that, we are going to use `$` on with a variable (or column) name that does not exist in our dataframe yet. We will then assign to this variable the value in the `age` column multiplied by 4.

```
# create new column called human_years
my_pets$human_years <- my_pets$age * 4

# print dataframe
my_pets
```

```
##     name age human_years
## 1  Daisy   8          32
## 2 Violet   2           8
## 3   Lily   6          24
## 4   Iris   3          12
## 5  Poppy   1           4
```

Inspect the new `my_pets` dataframe. What dimensions does it have now? How could you get a list of just the human years values in the data frame?

4.5 Descriptive stats on dataframes

Let's explore some functions for descriptive statistics.

CHALLENGE

Try running the following functions on `my_pets$age` and `my_pets$human_years`:

- mean()
- sd()
- median()
- max()
- min()
- range()

What other functions can/do you think/know of?

4.6 Note on coding style

Coding style refers to how you name your objects and functions, how you comment your code, how you use spacing throughout your code, etc. If your coding

style is consistent, your code is easier to read and easier to debug as a result. Here's some guides, so you can develop your own coding style:

- The tidyverse style guide
- Hadley Wickham's Advance R coding style
- Google's R Style Guide

Chapter 5

Version Control

5.1 Before Class #5

5.1.1 Install git on your computer

Access the git download page and download the appropriate version for your machine.

If you have a Windows 10 machine, you can watch this video that shows you how to install Git on windows. When installing, note where it's installed (on the "Select Destination Location" window) so you can check if you have the correct path to Git set up in RStudio (it's usually `C:\Program\Git`).

If you have a Mac, you can watch this video that shows you how to install Git on a Mac.

5.1.2 Create a GitHub account

1. Access the GitHub page.
2. Click on "Sign Up for GitHub."
3. Fill out the "Create your account" forms.
4. A verification will be sent to your email address, check your inbox for a "Please verify your email address" message. Click on "Verify email address" button.

If you already have a GitHub account, confirm you know your username and password by logging in at GitHub.

5.1.3 Join our GitHub classroom

1. Access our join our GitHub classroom page.
2. A window with information about what GitHub Classroom wants to access from your GitHub profile will appear. Click on "Authorize github".
3. Access our first assignment and click on "Accept this assignment"

5.1.4 Link RStudio to GitHub

1. Open "preferences" in RStudio.
2. Click on "Git/SVN" in the menu on the left.
3. Under "SSH RSA Key:" click on "Create RSA Key…"
4. A window will pop up, click on "Create"
5. A new window will pop up, click "Close"
6. Now there's a "View public key" link next to "SSH RSA Key:"; click on it
7. Copy key and close the window
8. Go to your GitHub account settings
9. On the menu on the left, click on "SSH and GPC keys"
10. Click on the "New SSH Key" button
11. Choose a title (e.g., RStudio Connection) and copy the key to the "key" field
12. Click "Add SSH Key"

5.2 What is version control?

Version control is a best practice for reproducible analyses, and widely used in industry and research (i.e., you will need to know how to use version control in your future job).

The purpose of version control is to keep track of changes to your files over time, so that you can recall specific versions at any point in your project.

Git is an open source version control software system that is very popular – 58% of data scientist use Git (Beckman et al., 2020). There are a number of other version control software available (e.g., Perforce).

5.3 Submitting assignments

5.3.1 Clone assignment repository

1. Go to our first assignment GitHub repository
2. Click on the "Code" button and copy the git url (e.g., https://github.com/esoc214/test-assignment-yournamehere.git)
3. Open RStudio
4. Go "File" > "New Project…"
5. In the pop-up window, select "Version Control"
6. Then choose "Git"
7. In the "Repository URL:" field enter the link to the first assignment repository from your GitHub account.
8. Click "Create"

5.3.2 Modify files

For the first assignment, which is a test assignment so you're all set up to submitting all of your assignments for this class, you need to modify `READM.md` only. For other assignments you will need to edit .R scripts.

5.3.3 Commit changes

1. On the top right panel in RStudio (i.e., Environment quadrant), click on the "Git" tab
2. You will see a list of files, indicating which files have been modified (a blue "M" shows next to modified file).
3. Click on "Commit" on the top of this tab
4. A new window will pop-up. Stage the files you want to commit (click on the check box next to file) and enter a commit message.
5. Press "Commit" and if everything looks good, close the commit window.
6. Click "Push" on top right

Chapter 6

Intro to Tidyverse

6.1 Before Class #6

Read Advice to Young (and Old) Programmers: A Conversation with Hadley Wickham by Philip Waggoner (2,599 words, 10 minutes)

6.2 What are R Packages?

An R package contains functions, and it might contain data. There are a lot of R packages out here (check the Comprehensive R Archive Network, i.e., CRAN, for a full list). That is one of the beautiful things about R, anyone can create an R package to share their code.

6.3 Installing Packages

The function to install packages in R is `install.packages()`. We will be working with TidyVerse extensively in this course, which is a collection of R packages carefully designed for data science.

Open your RStudio. In your console, enter the following to install tidyverse (this may take a while).

```
install.packages("tidyverse")
```

You need to install any package only once (remember to check for new package versions and to keep your packages updated). However, with every new R session, you need to load the packages you are going to use by using the `library()` function (a library is an installed R package in your computer).

```
library(tidyverse)
```

Note that when calling the `install.packages()` function you need to enter the package name between quotation marks (e.g., "tidyverse"). When you call the `library()` function, you don't use quotation marks (e.g., tidyverse).

6.4 Before You Load your Data

Although we are working within an R project, which sets the working directory automatically for you, it's good practice to check what folder you are working from by calling the `getwd()` function.

```
getwd()
```

```
## [1] "/Users/adriana/Desktop/ESOC214/Fall 2020/bookdown/ESOC_214_Fall_2020"
```

You can list the contents of your working directory by using the `dir()` function.

```
dir()
```

We are going to create a `data` folder in our project, to keep things organized. Today we will be working with a data set that contains groundhog day forecasts and temperature. I cleaned up this data set already (no need for data tidying for now).

You can now list the contents of your `data` folder with the `dir()` function with a string that specifies the folder as a parameter.

```
dir("data")
```

```
##  [1] "elnino.csv"                          "GlobalLandTemperaturesByCountry.csv"
##  [3] "groundhog_day.csv"                   "nfl_salary.xlsx"
##  [5] "olympic_history_athlete_events.csv"  "olympic_history_noc_regions.csv"
##  [7] "passwords.csv"                       "spotify_songs_clean.csv"
##  [9] "spotify_songs.csv"                   "tweets.tsv"
## [11] "us_avg_tuition.xlsx"
```

6.5 What's our question again?

Here's what we will focus on answering today, which is an excerpt from the Groundhog Day Forecasts and Temperatures kaggle page.

"Thousands gather at Gobbler's Knob in Punxsutawney, Pennsylvania, on the second day of February to await the spring forecast from a groundhog known as Punxsutawney Phil. According to legend, if Phil sees his shadow the United States is in store for six more weeks of winter weather. But, if Phil doesn't see

his shadow, the country should expect warmer temperatures and the arrival of an early spring."

So, in summary, our question is **How accurate is Punxsutawney Phil's winter weather forecast?**

6.6 Load Data with Tidyverse

We will use the `read_csv()` function from the `readr` package (which is part of `tidyverse`) to read data in. Be careful, there's a similar function that is read.csv() from base R. We do want to use the function with the `_` (i.e., `read_csv()`)

```
groundhog_predictions <- read_csv("data/groundhog_day.csv")
```

```
## Parsed with column specification:
## cols(
##   Year = col_double(),
##   Punxsutawney_Phil = col_character(),
##   February_Average_Temperature = col_double(),
##   February_Average_Temperature_Northeast = col_double(),
##   February_Average_Temperature_Midwest = col_double(),
##   February_Average_Temperature_Pennsylvania = col_double(),
##   March_Average_Temperature = col_double(),
##   March_Average_Temperature_Northeast = col_double(),
##   March_Average_Temperature_Midwest = col_double(),
##   March_Average_Temperature_Pennsylvania = col_double()
## )
```

CHALLENGE

Reading warnings - R often prints out warnings in red (these are not always errors). What information did you get when loading your data?

6.7 Inspect Your Data

As with any other programming language, there are multiple ways to doing anything. As such, there are multiple ways of inspecting your data in R. Here are some of my favorite ways of inspecting my data:

```
# get an overview of the data frame
glimpse(groundhog_predictions)
```

```
## Rows: 122
## Columns: 10
```

```
## $ Year                                      <dbl> 1895, 1896, 1897, 1898, 1...
## $ Punxsutawney_Phil                         <chr> "No Record", "No Record",...
## $ February_Average_Temperature              <dbl> 26.60, 35.04, 33.39, 35.3...
## $ February_Average_Temperature_Northeast    <dbl> 15.6, 22.2, 23.6, 24.8, 1...
## $ February_Average_Temperature_Midwest      <dbl> 21.9, 33.5, 34.7, 33.3, 2...
## $ February_Average_Temperature_Pennsylvania <dbl> 17.0, 26.6, 27.9, 26.7, 2...
## $ March_Average_Temperature                 <dbl> 39.97, 38.03, 38.79, 41.0...
## $ March_Average_Temperature_Northeast       <dbl> 27.6, 25.3, 32.0, 38.0, 2...
## $ March_Average_Temperature_Midwest         <dbl> 40.2, 36.9, 44.0, 46.0, 3...
## $ March_Average_Temperature_Pennsylvania    <dbl> 31.3, 27.8, 36.9, 42.0, 3...
```

```
summary(groundhog_predictions)
```

```
##       Year      Punxsutawney_Phil  February_Average_Temperature
##  Min.   :1895   Length:122         Min.   :25.23
##  1st Qu.:1925   Class :character   1st Qu.:31.78
##  Median :1956   Mode  :character   Median :33.69
##  Mean   :1956                      Mean   :33.80
##  3rd Qu.:1986                      3rd Qu.:36.01
##  Max.   :2016                      Max.   :41.41
##  February_Average_Temperature_Northeast February_Average_Temperature_Midwest
##  Min.   :10.40                         Min.   :20.30
##  1st Qu.:20.02                         1st Qu.:29.62
##  Median :22.95                         Median :33.20
##  Mean   :22.69                         Mean   :32.69
##  3rd Qu.:25.98                         3rd Qu.:36.30
##  Max.   :31.60                         Max.   :41.40
##  February_Average_Temperature_Pennsylvania March_Average_Temperature
##  Min.   :15.20                               Min.   :35.44
##  1st Qu.:23.60                               1st Qu.:39.38
##  Median :26.95                               Median :41.81
##  Mean   :26.52                               Mean   :41.70
##  3rd Qu.:29.80                               3rd Qu.:43.56
##  Max.   :35.80                               Max.   :50.41
##  March_Average_Temperature_Northeast March_Average_Temperature_Midwest
##  Min.   :24.20                       Min.   :28.50
##  1st Qu.:29.70                       1st Qu.:39.08
##  Median :32.55                       Median :42.85
##  Mean   :32.37                       Mean   :42.57
##  3rd Qu.:34.80                       3rd Qu.:45.60
##  Max.   :43.40                       Max.   :56.30
##  March_Average_Temperature_Pennsylvania
##  Min.   :24.50
##  1st Qu.:32.95
##  Median :35.85
##  Mean   :35.91
```

```
##  3rd Qu.:38.55
##  Max.   :47.70
```

```
# get variable names
colnames(groundhog_predictions)
```

```
##  [1] "Year"
##  [2] "Punxsutawney_Phil"
##  [3] "February_Average_Temperature"
##  [4] "February_Average_Temperature_Northeast"
##  [5] "February_Average_Temperature_Midwest"
##  [6] "February_Average_Temperature_Pennsylvania"
##  [7] "March_Average_Temperature"
##  [8] "March_Average_Temperature_Northeast"
##  [9] "March_Average_Temperature_Midwest"
## [10] "March_Average_Temperature_Pennsylvania"
```

```
names(groundhog_predictions)
```

```
##  [1] "Year"
##  [2] "Punxsutawney_Phil"
##  [3] "February_Average_Temperature"
##  [4] "February_Average_Temperature_Northeast"
##  [5] "February_Average_Temperature_Midwest"
##  [6] "February_Average_Temperature_Pennsylvania"
##  [7] "March_Average_Temperature"
##  [8] "March_Average_Temperature_Northeast"
##  [9] "March_Average_Temperature_Midwest"
## [10] "March_Average_Temperature_Pennsylvania"
```

```
# check the categorical variable
unique(groundhog_predictions$Punxsutawney_Phil)
```

```
## [1] "No Record"      "Full Shadow"    "No Shadow"      "Partial Shadow"
```

CHALLENGE

Which variables are numeric? Which are categorical?

6.8 The Pipe

We will be using the package `dplyr` (which is also part of `tidyverse`) to do an exploratory analysis of our data.

The package `dplyr` most used function is `%>%` (called the pipe). The pipe allows you to "pipe" (or redirect) objects into functions. (hint: use ctrl+shift+m or cmd+shift+m as a shortcut for typing `%>%`).

Here's how to pipe the `avocado_data` object into the `summary()` function

```
# get an overview of the data frame
groundhog_predictions %>%
  summary()
```

```
##       Year      Punxsutawney_Phil  February_Average_Temperature
##  Min.   :1895   Length:122         Min.   :25.23
##  1st Qu.:1925   Class :character   1st Qu.:31.78
##  Median :1956   Mode  :character   Median :33.69
##  Mean   :1956                      Mean   :33.80
##  3rd Qu.:1986                      3rd Qu.:36.01
##  Max.   :2016                      Max.   :41.41
##  February_Average_Temperature_Northeast February_Average_Temperature_Midwest
##  Min.   :10.40                          Min.   :20.30
##  1st Qu.:20.02                          1st Qu.:29.62
##  Median :22.95                          Median :33.20
##  Mean   :22.69                          Mean   :32.69
##  3rd Qu.:25.98                          3rd Qu.:36.30
##  Max.   :31.60                          Max.   :41.40
##  February_Average_Temperature_Pennsylvania March_Average_Temperature
##  Min.   :15.20                             Min.   :35.44
##  1st Qu.:23.60                             1st Qu.:39.38
##  Median :26.95                             Median :41.81
##  Mean   :26.52                             Mean   :41.70
##  3rd Qu.:29.80                             3rd Qu.:43.56
##  Max.   :35.80                             Max.   :50.41
##  March_Average_Temperature_Northeast March_Average_Temperature_Midwest
##  Min.   :24.20                       Min.   :28.50
##  1st Qu.:29.70                       1st Qu.:39.08
##  Median :32.55                       Median :42.85
##  Mean   :32.37                       Mean   :42.57
##  3rd Qu.:34.80                       3rd Qu.:45.60
##  Max.   :43.40                       Max.   :56.30
##  March_Average_Temperature_Pennsylvania
##  Min.   :24.50
##  1st Qu.:32.95
##  Median :35.85
##  Mean   :35.91
##  3rd Qu.:38.55
##  Max.   :47.70
```

The pipe allows us to apply multiple functions to the same object.

Let's start by selecting one column in our data.

```
groundhog_predictions %>%
  select(Punxsutawney_Phil)
```

```
## # A tibble: 122 x 1
##    Punxsutawney_Phil
##    <chr>
##  1 No Record
##  2 No Record
##  3 No Record
##  4 Full Shadow
##  5 No Record
##  6 Full Shadow
##  7 Full Shadow
##  8 No Record
##  9 Full Shadow
## 10 Full Shadow
## # ... with 112 more rows
```

Now let's add another pipe to get unique values in this column.

```
groundhog_predictions %>%
  select(Punxsutawney_Phil) %>%
  unique()
```

```
## # A tibble: 4 x 1
##   Punxsutawney_Phil
##   <chr>
## 1 No Record
## 2 Full Shadow
## 3 No Shadow
## 4 Partial Shadow
```

6.9 Counting Categorical Variables

One of the functions I most use when exploring my data is `count()`, which you can combine with `%>%`.

```
groundhog_predictions %>%
  count(Punxsutawney_Phil)
```

```
## # A tibble: 4 x 2
##   Punxsutawney_Phil     n
##   <chr>             <int>
## 1 Full Shadow         100
## 2 No Record             6
## 3 No Shadow            15
## 4 Partial Shadow        1
```

You can do the same adding `group_by()` to your pipeline.

```
groundhog_predictions %>%
  group_by(Punxsutawney_Phil) %>%
  count()
```

```
## # A tibble: 4 x 2
## # Groups:   Punxsutawney_Phil [4]
##   Punxsutawney_Phil     n
##   <chr>             <int>
## 1 Full Shadow         100
## 2 No Record             6
## 3 No Shadow            15
## 4 Partial Shadow        1
```

And instead of `count()` we can use the `summarise()` and `n()` functions.

```
groundhog_predictions %>%
  group_by(Punxsutawney_Phil) %>%
  summarise(total = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 4 x 2
##   Punxsutawney_Phil total
##   <chr>             <int>
## 1 Full Shadow         100
## 2 No Record             6
## 3 No Shadow            15
## 4 Partial Shadow        1
```

CHALLENGE

This last way of counting categorical variables (with `summarise()` and `n()`) outputs a data frame that is slightly different from the previous too. What's the difference?

6.10 group_by + summarise

The combination of the `group_by()` and `summarise()` functions is very powerful. In addition to using the `n()` function to count how many rows per each category in our categorical variable, we can use other functions with numeric (i.e., quantitative) variable such as `sum()` and `mean()`.

CHALLENGE

Take a moment to revisit the question we want to answer.

- What do we want to find out?
- How can we answer our question with this data?

- What function (e.g., `sum()`, `max()`, `mean()`) do we use to answer our question? With what variables/columns?

Complete the code below.

```
groundhog_predictions %>%
  group_by(Punxsutawney_Phil) %>%
  summarise(total = n(),
            ____ = ____(____))
```

Example of output that you might want to get to answer our question:

```
## # A tibble: 4 x 4
##   Punxsutawney_Phil total feb_mean_temp mar_mean_temp
##   <chr>             <int>         <dbl>         <dbl>
## 1 Full Shadow         100          33.7          41.7
## 2 No Record             6          31.4          39.1
## 3 No Shadow            15          35.6          43.0
## 4 Partial Shadow        1          30.7          41.3
```

6.11 group_by + filter

The output above contains six `No Record` observations and only one `Partial Shadow`. We can keep just observations that are `Full Shadow` and `No Shadow` by using the filter() function:

```
groundhog_predictions %>%
  filter(Punxsutawney_Phil == "Full Shadow" |
           Punxsutawney_Phil == "No Shadow") %>%
  count(Punxsutawney_Phil)
```

```
## # A tibble: 2 x 2
##   Punxsutawney_Phil     n
##   <chr>             <int>
## 1 Full Shadow         100
## 2 No Shadow            15
```

CHALLENGE

Add a `filter()` to your solution from the previous challenge.

Example of output that you might want to get:

```
## # A tibble: 2 x 4
##   Punxsutawney_Phil total feb_mean_temp mar_mean_temp
##   <chr>             <int>         <dbl>         <dbl>
## 1 Full Shadow         100          33.7          41.7
## 2 No Shadow            15          35.6          43.0
```

6.12 Pivot Dataframe

Another useful function we will be using a lot during this course is `pivot_longer()`, which pivots (or tilts) some columns in our dataframe so we have one column for each of our variables.

Let's first select the temperatures for individual regions and create a new dataframe.

```
selected_predictions <- groundhog_predictions %>%
  select(Year, Punxsutawney_Phil,
         February_Average_Temperature_Northeast,
         February_Average_Temperature_Midwest,
         February_Average_Temperature_Pennsylvania,
         March_Average_Temperature_Northeast,
         March_Average_Temperature_Midwest,
         March_Average_Temperature_Pennsylvania)

colnames(selected_predictions)
```

```
## [1] "Year"
## [2] "Punxsutawney_Phil"
## [3] "February_Average_Temperature_Northeast"
## [4] "February_Average_Temperature_Midwest"
## [5] "February_Average_Temperature_Pennsylvania"
## [6] "March_Average_Temperature_Northeast"
## [7] "March_Average_Temperature_Midwest"
## [8] "March_Average_Temperature_Pennsylvania"
```

Another way of doing the same thing we just did is by saying the columns we want to eliminate from our selection, using the `-` (i.e, minus) sign.

```
selected_predictions <- groundhog_predictions %>%
  select(-February_Average_Temperature, -March_Average_Temperature)

colnames(selected_predictions)
```

```
## [1] "Year"
## [2] "Punxsutawney_Phil"
## [3] "February_Average_Temperature_Northeast"
## [4] "February_Average_Temperature_Midwest"
## [5] "February_Average_Temperature_Pennsylvania"
## [6] "March_Average_Temperature_Northeast"
## [7] "March_Average_Temperature_Midwest"
## [8] "March_Average_Temperature_Pennsylvania"
```

Now we are ready to `pivot_longer()` our selected_predictions dataframe. Let's examine our selected_predictions dataframe. We want to move all numbers to

our numeric variable called temperature, and we want the column names to be another variable called `month_region` so that are data is tidy.

```
head(selected_predictions)
```

```
## # A tibble: 6 x 8
##    Year Punxsutawney_Ph~ February_Averag~ February_Averag~ February_Averag~
##   <dbl> <chr>                      <dbl>            <dbl>            <dbl>
## 1  1895 No Record                   15.6             21.9             17
## 2  1896 No Record                   22.2             33.5             26.6
## 3  1897 No Record                   23.6             34.7             27.9
## 4  1898 Full Shadow                 24.8             33.3             26.7
## 5  1899 No Record                   18.1             22.2             20
## 6  1900 Full Shadow                 21.4             27.5             24.1
## # ... with 3 more variables: March_Average_Temperature_Northeast <dbl>,
## #   March_Average_Temperature_Midwest <dbl>,
## #   March_Average_Temperature_Pennsylvania <dbl>
```

Again we can list all the columns we want to pivot (i.e., all the columns that are numeric), but we have a smaller number of columns we don't want to pivot, so we use the `-` (minus) symbol with the columns we don't want to pivot.

```
selected_predictions %>%
  pivot_longer(cols = c(-Year, -Punxsutawney_Phil))
```

```
## # A tibble: 732 x 4
##     Year Punxsutawney_Phil name                                    value
##    <dbl> <chr>             <chr>                                   <dbl>
##  1  1895 No Record         February_Average_Temperature_Northeast   15.6
##  2  1895 No Record         February_Average_Temperature_Midwest     21.9
##  3  1895 No Record         February_Average_Temperature_Pennsylvania 17
##  4  1895 No Record         March_Average_Temperature_Northeast      27.6
##  5  1895 No Record         March_Average_Temperature_Midwest        40.2
##  6  1895 No Record         March_Average_Temperature_Pennsylvania   31.3
##  7  1896 No Record         February_Average_Temperature_Northeast   22.2
##  8  1896 No Record         February_Average_Temperature_Midwest     33.5
##  9  1896 No Record         February_Average_Temperature_Pennsylvania 26.6
## 10  1896 No Record         March_Average_Temperature_Northeast      25.3
## # ... with 722 more rows
```

We can specify the column names so they are not just `name` and `value`:

```
selected_predictions %>%
  pivot_longer(cols = c(-Year, -Punxsutawney_Phil),
               names_to = "month_region",
               values_to = "temperature")
```

```
## # A tibble: 732 x 4
##     Year Punxsutawney_Phil month_region                            temperature
```

```
##    <dbl> <chr>            <chr>                                    <dbl>
##  1  1895 No Record        February_Average_Temperature_Northeast    15.6
##  2  1895 No Record        February_Average_Temperature_Midwest      21.9
##  3  1895 No Record        February_Average_Temperature_Pennsylvania 17
##  4  1895 No Record        March_Average_Temperature_Northeast       27.6
##  5  1895 No Record        March_Average_Temperature_Midwest         40.2
##  6  1895 No Record        March_Average_Temperature_Pennsylvania    31.3
##  7  1896 No Record        February_Average_Temperature_Northeast    22.2
##  8  1896 No Record        February_Average_Temperature_Midwest      33.5
##  9  1896 No Record        February_Average_Temperature_Pennsylvania 26.6
## 10  1896 No Record        March_Average_Temperature_Northeast       25.3
## # ... with 722 more rows
```

Let's save the result above in a new dataframe.

```
predictions_tidy <- selected_predictions %>%
  pivot_longer(cols = c(-Year, -Punxsutawney_Phil),
               names_to = "month_region",
               values_to = "temperature")
```

6.13 Separating one categorical column into two

When we look at our `predictions_tidy`, we see that the `month_region` is actually holding two categorical variables. We can separate this column into its two variables with the `separate()` function. There are four parts to each of the values, the two middle parts are not useful so we name the first part `month` the last `region` and the other two not useful parts (the middle two) `trash1` and `trash2`.

```
predictions_tidy_v2 <- predictions_tidy %>%
  separate(col = month_region,
           into = c("month", "trash1", "trash2", "region"))

predictions_tidy_v2
```

```
## # A tibble: 732 x 7
##     Year Punxsutawney_Phil month    trash1  trash2      region       temperature
##    <dbl> <chr>             <chr>    <chr>   <chr>       <chr>              <dbl>
##  1  1895 No Record         February Average Temperature Northeast           15.6
##  2  1895 No Record         February Average Temperature Midwest             21.9
##  3  1895 No Record         February Average Temperature Pennsylvania        17
##  4  1895 No Record         March    Average Temperature Northeast           27.6
##  5  1895 No Record         March    Average Temperature Midwest             40.2
##  6  1895 No Record         March    Average Temperature Pennsylvania        31.3
##  7  1896 No Record         February Average Temperature Northeast           22.2
```

```
##  8  1896 No Record         February Average Temperature Midwest          33.5
##  9  1896 No Record         February Average Temperature Pennsylvania     26.6
## 10  1896 No Record         March    Average Temperature Northeast        25.3
## # ... with 722 more rows
```

We can delete the two not useful columns (i.e., `trash1` and `trash2`) by using the `select()` and `-` (minus) symbol.

```
predictions_tidy_v2 <- predictions_tidy_v2 %>%
  select(-trash1, -trash2)
```

6.14 Example of Plotting

Now that we have our tidy dataframe (i.e., `predictions_tidy_v2`), we can plot temperatures by year:

```
predictions_tidy_v2 %>%
  ggplot(aes(x = Year,
             y = temperature)) +
  geom_point(aes(color = month))
```

6.15 DATA CHALLENGE 01

Accept data challenge 01 assignment

Chapter 7

Data Wrangling

7.1 Load libraries

Load tidyverse using `library()`

Our data for this module is an excel spreadsheet, so we need to install a new package to handle this type of data.

```
install.packages("readxl")
```

7.2 Read your data in

After `readxl` package installation is done:

1. load `readxl` using `library()`
2. check your working environment with `getwd()` and `dir()`
3. load your data

```
nfl_salary <- read_excel("data/nfl_salary.xlsx")
```

4. inspect your data with `summary()`, `glimpse()` and `View()`

```
glimpse(nfl_salary)
```

```
## Rows: 800
## Columns: 11
## $ year                <dbl> 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011,...
## $ Cornerback          <dbl> 11265916, 11000000, 10000000, 10000000, 1000000...
## $ `Defensive Lineman` <dbl> 17818000, 16200000, 12476000, 11904706, 1176278...
## $ Linebacker          <dbl> 16420000, 15623000, 11825000, 10083333, 1002000...
```

```
## $ `Offensive Lineman` <dbl> 15960000, 12800000, 11767500, 10358200, 1000000...
## $ Quarterback         <dbl> 17228125, 16000000, 14400000, 14100000, 1351000...
## $ `Running Back`      <dbl> 12955000, 10873833, 9479000, 7700000, 7500000, ...
## $ Safety              <dbl> 8871428, 8787500, 8282500, 8000000, 7804333, 76...
## $ `Special Teamer`    <dbl> 4300000, 3725000, 3556176, 3500000, 3250000, 32...
## $ `Tight End`         <dbl> 8734375, 8591000, 8290000, 7723333, 6974666, 61...
## $ `Wide Receiver`     <dbl> 16250000, 14175000, 11424000, 11415000, 1080000...
```

5. How many observations are there?
6. What variables are there in the data?

7.3 Summarise data

QUESTIONS:

1) Have salaries for different NFL positions increased between 2011 and 2018?
2) What positions pay more and less?

Let's summarise the `mean` salary for Quarterback `by year`.

```
nfl_salary %>%
  group_by(year) %>%
  summarise(quarterback_mean_salary = mean(Quarterback, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 8 x 2
##    year quarterback_mean_salary
##   <dbl>                   <dbl>
## 1  2011                3376113.
## 2  2012                3496408.
## 3  2013                3450185.
## 4  2014                4234160.
## 5  2015                4225789.
## 6  2016                5499939.
## 7  2017                5329727.
## 8  2018                6593769.
```

What would we do to add the mean salary for `Cornerback`?

```
nfl_salary %>%
  group_by(year) %>%
  summarise(quarterback_mean_salary = mean(Quarterback, na.rm = TRUE),
            cornerback_mean_salary = mean(Cornerback, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 8 x 3
##    year quarterback_mean_salary cornerback_mean_salary
##   <dbl>                   <dbl>                  <dbl>
## 1  2011                3376113.               3037766.
## 2  2012                3496408.               3132916.
## 3  2013                3450185.               2901798.
## 4  2014                4234160.               3038278.
## 5  2015                4225789.               3758543.
## 6  2016                5499939.               4201470.
## 7  2017                5329727.               4125692.
## 8  2018                6593769.               4659704.
```

Let's stop and think about how our data is organized. Is our data tidy?

We have columns that mix two type of variables:

a) categorical variable for position

b) numeric variable for salary

7.4 Tidy data

In order to make our data easier to work with, we need to make sure each column in our data represents just one variable. To do that for our `nfl_salary` dataframe, we need to pivot it.

```
nfl_salary_tidy <- nfl_salary %>%
  pivot_longer(cols = -year,
               names_to = "position",
               values_to = "salary")
```

Always inspect your new data frame.

```
glimpse(nfl_salary_tidy)
```

```
## Rows: 8,000
## Columns: 3
## $ year     <dbl> 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011...
## $ position <chr> "Cornerback", "Defensive Lineman", "Linebacker", "Offensiv...
## $ salary   <dbl> 11265916, 17818000, 16420000, 15960000, 17228125, 12955000...
```

How many positions are there in the data? We can now do a `count()` with our categorical variable for `position`

```
nfl_salary_tidy %>%
  count(position)
```

```
## # A tibble: 10 x 2
##    position              n
```

```
##    <chr>             <int>
##  1 Cornerback          800
##  2 Defensive Lineman   800
##  3 Linebacker          800
##  4 Offensive Lineman   800
##  5 Quarterback         800
##  6 Running Back        800
##  7 Safety              800
##  8 Special Teamer      800
##  9 Tight End           800
## 10 Wide Receiver       800
```

We can add `year` to our `group_by` to check how many observations per `position` across `year`

```
nfl_salary_tidy %>%
  count(position, year)
```

```
## # A tibble: 80 x 3
##    position           year     n
##    <chr>             <dbl> <int>
##  1 Cornerback         2011   100
##  2 Cornerback         2012   100
##  3 Cornerback         2013   100
##  4 Cornerback         2014   100
##  5 Cornerback         2015   100
##  6 Cornerback         2016   100
##  7 Cornerback         2017   100
##  8 Cornerback         2018   100
##  9 Defensive Lineman  2011   100
## 10 Defensive Lineman  2012   100
## # ... with 70 more rows
```

Let's check for NAs (i.e., missing data), we can do that by using `is.na()` and `filter()`.

```
nfl_salary_tidy %>%
  filter(is.na(salary)) %>%
  count(position, year)
```

```
## # A tibble: 9 x 3
##   position        year     n
##   <chr>          <dbl> <int>
## 1 Quarterback     2011     3
## 2 Quarterback     2012    12
## 3 Quarterback     2013     7
## 4 Quarterback     2014    11
## 5 Quarterback     2015     3
```

```
## 6 Quarterback     2016     5
## 7 Quarterback     2017     3
## 8 Quarterback     2018    11
## 9 Special Teamer  2011     1
```

We can remove these rows from our data frame.

```
nfl_salary_tidy_clean <- nfl_salary_tidy %>%
  filter(!is.na(salary))
```

Inspect your new data frame.

```
glimpse(nfl_salary_tidy_clean)
```

```
## Rows: 7,944
## Columns: 3
## $ year     <dbl> 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011...
## $ position <chr> "Cornerback", "Defensive Lineman", "Linebacker", "Offensiv...
## $ salary   <dbl> 11265916, 17818000, 16420000, 15960000, 17228125, 12955000...
```

Now we can do our salary `summarise()` in a cleaner way. We are going to do a `mean()` of our numeric variable `salary` by `year` AND `position`.

```
nfl_salary_tidy_clean %>%
  group_by(year, position) %>%
  summarise(mean_salary = mean(salary))
```

```
## `summarise()` regrouping output by 'year' (override with `.groups` argument)
```

```
## # A tibble: 80 x 3
## # Groups:   year [8]
##     year position          mean_salary
##    <dbl> <chr>                   <dbl>
##  1  2011 Cornerback           3037766.
##  2  2011 Defensive Lineman    4306995.
##  3  2011 Linebacker           4016045.
##  4  2011 Offensive Lineman    4662748.
##  5  2011 Quarterback          3376113.
##  6  2011 Running Back         1976341.
##  7  2011 Safety               2241891.
##  8  2011 Special Teamer       1244069.
##  9  2011 Tight End            1608100.
## 10  2011 Wide Receiver        2996590.
## # ... with 70 more rows
```

We can do the group_by both ways (first `year` and then `position` or vice-versa).

```
nfl_salary_tidy_clean %>%
  group_by(position, year) %>%
  summarise(mean_salary = mean(salary)) %>%
```

```
  arrange(mean_salary)
```

```
## `summarise()` regrouping output by 'position' (override with `.groups` argument)
```

```
## # A tibble: 80 x 3
## # Groups:   position [10]
##    position        year mean_salary
##    <chr>          <dbl>       <dbl>
##  1 Special Teamer  2013    1235892.
##  2 Special Teamer  2011    1244069.
##  3 Special Teamer  2014    1264493.
##  4 Special Teamer  2012    1313043.
##  5 Special Teamer  2015    1348637.
##  6 Special Teamer  2016    1394443.
##  7 Special Teamer  2017    1459552.
##  8 Special Teamer  2018    1571447.
##  9 Tight End       2011    1608100.
## 10 Tight End       2012    1664520.
## # ... with 70 more rows
```

Add a - (minus) sign to the argument in `arrange()` to arrange your results by decreasing order of mean_salary.

```
nfl_salary_tidy_clean %>%
  group_by(position, year) %>%
  summarise(mean_salary = mean(salary)) %>%
  arrange(-mean_salary)
```

```
## `summarise()` regrouping output by 'position' (override with `.groups` argument)
```

```
## # A tibble: 80 x 3
## # Groups:   position [10]
##    position           year mean_salary
##    <chr>             <dbl>       <dbl>
##  1 Offensive Lineman  2018    7522647.
##  2 Defensive Lineman  2018    7202360.
##  3 Quarterback        2018    6593769.
##  4 Offensive Lineman  2017    6370947.
##  5 Defensive Lineman  2017    6202601.
##  6 Wide Receiver      2018    5627721.
##  7 Quarterback        2016    5499939.
##  8 Offensive Lineman  2016    5410392.
##  9 Quarterback        2017    5329727.
## 10 Linebacker         2018    5293675.
## # ... with 70 more rows
```

We can also add `arrange()` to our code block.

```
nfl_salary_tidy_clean %>%
  group_by(position, year) %>%
  summarise(mean_salary = mean(salary)) %>%
  arrange()
```

```
## `summarise()` regrouping output by 'position' (override with `.groups` argument)
```

```
## # A tibble: 80 x 3
## # Groups:   position [10]
##    position           year mean_salary
##    <chr>             <dbl>       <dbl>
##  1 Cornerback         2011    3037766.
##  2 Cornerback         2012    3132916.
##  3 Cornerback         2013    2901798.
##  4 Cornerback         2014    3038278.
##  5 Cornerback         2015    3758543.
##  6 Cornerback         2016    4201470.
##  7 Cornerback         2017    4125692.
##  8 Cornerback         2018    4659704.
##  9 Defensive Lineman  2011    4306995.
## 10 Defensive Lineman  2012    4693730.
## # ... with 70 more rows
```

7.4.1 Viz Demo

We can also visualize our data using `ggplot()`.

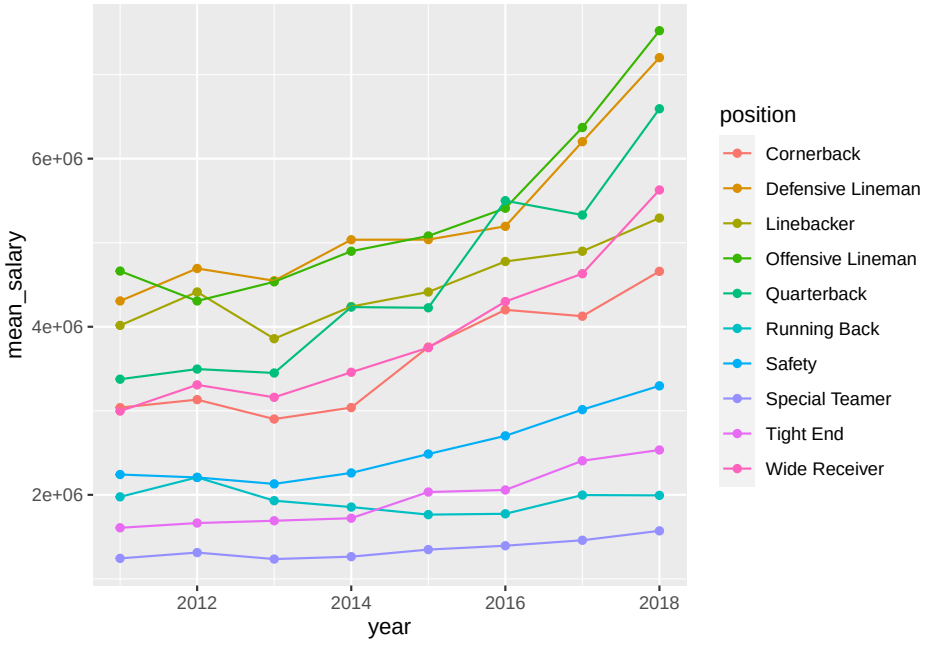
First we save our summary results in a new dataframe called `nfl_salary_summary`.

```
nfl_salary_summary <- nfl_salary_tidy_clean %>%
  group_by(position, year) %>%
  summarise(mean_salary = mean(salary)) %>%
  arrange()
```

```
## `summarise()` regrouping output by 'position' (override with `.groups` argument)
```

Then we plot it.

```
nfl_salary_summary %>%
  ggplot(aes(x = year, y = mean_salary,
             color = position,
             group = position)) +
  geom_point() +
  geom_line()
```



7.5 Transform Data

Now that our data is tidy, we can transform our data by adding new variables/columns to it.

It seems some salaries for certain positions show a higher increase across the years than the salaries for other positions. In other words, the proportion of what position makes in relation to total money spent in salaries for each each.

We can check this is true by creating a `sum()` of salaries for each year and a count of players using `n()`:

```
nfl_salary_tidy_clean %>%
  group_by(year, position) %>%
  summarise(player_count = n(),
            total_per_position = sum(salary))
```

```
## `summarise()` regrouping output by 'year' (override with `.groups` argument)
```

```
## # A tibble: 80 x 4
## # Groups:   year [8]
##     year position          player_count total_per_position
##    <dbl> <chr>                    <int>              <dbl>
##  1  2011 Cornerback                 100          303776605
##  2  2011 Defensive Lineman          100          430699528
```

```
##  3  2011 Linebacker                100          401604548
##  4  2011 Offensive Lineman         100          466274753
##  5  2011 Quarterback                97          327482939
##  6  2011 Running Back              100          197634074
##  7  2011 Safety                    100          224189136
##  8  2011 Special Teamer             99          123162874
##  9  2011 Tight End                 100          160810030
## 10  2011 Wide Receiver             100          299659044
## # ... with 70 more rows
```

We can then add `mutate()` to our code block to calculate `sum()` of all salaries per year.

```
nfl_salary_tidy_clean %>%
  group_by(year, position) %>%
  summarise(player_count = n(),
            total_per_position = sum(salary)) %>%
  mutate(total_per_year = sum(total_per_position))
```

```
## `summarise()` regrouping output by 'year' (override with `.groups` argument)
```

```
## # A tibble: 80 x 5
## # Groups:   year [8]
##     year position          player_count total_per_position total_per_year
##    <dbl> <chr>                    <int>              <dbl>          <dbl>
##  1  2011 Cornerback                 100          303776605     2935293531
##  2  2011 Defensive Lineman          100          430699528     2935293531
##  3  2011 Linebacker                 100          401604548     2935293531
##  4  2011 Offensive Lineman          100          466274753     2935293531
##  5  2011 Quarterback                 97          327482939     2935293531
##  6  2011 Running Back               100          197634074     2935293531
##  7  2011 Safety                     100          224189136     2935293531
##  8  2011 Special Teamer              99          123162874     2935293531
##  9  2011 Tight End                  100          160810030     2935293531
## 10  2011 Wide Receiver              100          299659044     2935293531
## # ... with 70 more rows
```

Now we can calculate the percentage cost of each position by the total salaries for each year, we can do that all in the same `mutate()`.

```
nfl_salary_tidy_clean %>%
  group_by(year, position) %>%
  summarise(player_count = n(),
            total_per_position = sum(salary)) %>%
  mutate(total_per_year = sum(total_per_position),
         percentage_cost = total_per_position/total_per_year)
```

```
## `summarise()` regrouping output by 'year' (override with `.groups` argument)
```

```
## # A tibble: 80 x 6
## # Groups:   year [8]
##     year position   player_count total_per_posit~ total_per_year percentage_cost
##    <dbl> <chr>             <int>            <dbl>          <dbl>           <dbl>
##  1  2011 Cornerback          100        303776605     2935293531           0.103
##  2  2011 Defensive~          100        430699528     2935293531           0.147
##  3  2011 Linebacker          100        401604548     2935293531           0.137
##  4  2011 Offensive~          100        466274753     2935293531           0.159
##  5  2011 Quarterba~           97        327482939     2935293531           0.112
##  6  2011 Running B~          100        197634074     2935293531           0.0673
##  7  2011 Safety              100        224189136     2935293531           0.0764
##  8  2011 Special T~           99        123162874     2935293531           0.0420
##  9  2011 Tight End           100        160810030     2935293531           0.0548
## 10  2011 Wide Rece~          100        299659044     2935293531           0.102
## # ... with 70 more rows
```

Add `arrange()` to see higher percentages at the top.

```
nfl_salary_tidy_clean %>%
  group_by(year, position) %>%
  summarise(player_count = n(),
            total_per_position = sum(salary)) %>%
  mutate(total_per_year = sum(total_per_position),
         percentage_cost = total_per_position/total_per_year) %>%
  arrange(-percentage_cost)
```

```
## `summarise()` regrouping output by 'year' (override with `.groups` argument)
```

```
## # A tibble: 80 x 6
## # Groups:   year [8]
##     year position   player_count total_per_posit~ total_per_year percentage_cost
##    <dbl> <chr>             <int>            <dbl>          <dbl>           <dbl>
##  1  2018 Offensive~          100        752264724     4557047519           0.165
##  2  2014 Defensive~          100        503535499     3154183189           0.160
##  3  2011 Offensive~          100        466274753     2935293531           0.159
##  4  2017 Offensive~          100        637094749     4027571325           0.158
##  5  2018 Defensive~          100        720236012     4557047519           0.158
##  6  2013 Defensive~          100        454787761     2920039442           0.156
##  7  2014 Offensive~          100        489885308     3154183189           0.155
##  8  2013 Offensive~          100        453489965     2920039442           0.155
##  9  2012 Defensive~          100        469373045     3032589536           0.155
## 10  2017 Defensive~          100        620260110     4027571325           0.154
## # ... with 70 more rows
```

7.5.1 Viz Demo

We can also visualize our data using `ggplot()`.

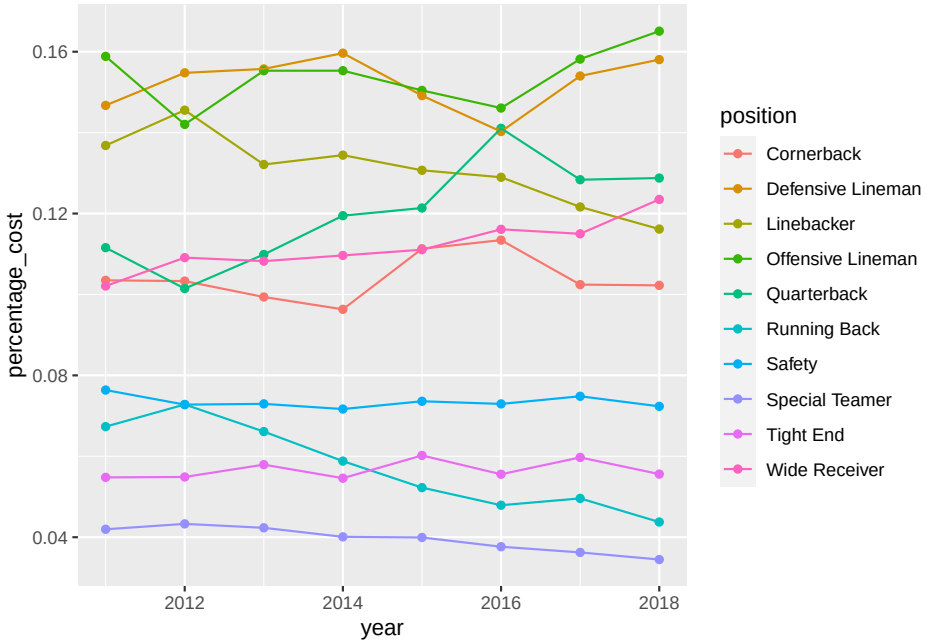
First we save our summary results in a new dataframe called `nfl_salary_summary`.

```
nfl_salary_summary <- nfl_salary_tidy_clean %>%
  group_by(year, position) %>%
  summarise(player_count = n(),
            total_per_position = sum(salary)) %>%
  mutate(total_per_year = sum(total_per_position),
         percentage_cost = total_per_position/total_per_year) %>%
  arrange(-percentage_cost)
```

```
## `summarise()` regrouping output by 'year' (override with `.groups` argument)
```

Then we plot it.

```
nfl_salary_summary %>%
  ggplot(aes(x = year, y = percentage_cost,
             color = position,
             group = position)) +
  geom_point() +
  geom_line()
```



7.6 DATA CHALLENGE 02

Accept data challenge 02 assignment

Chapter 8

Data Visualization

8.1 The layered grammar of graphics

The package we will be using for plotting in this class is called `ggplot2` which is part of `tidyverse`, and it uses as a principle the idea of layered grammar of graphics. That means you can add code to add or change your plot in layers, by using the `+` symbol.

Let's start with some data, so we can created different plots using different layers.

8.2 Load data

Get data directly from tidy tuesday github.

```
## Rows: 32,833
## Columns: 23
## $ track_id                <chr> "6f807x0ima9a1j3VPbc7VN", "0r7CVbZTWZgbTCY...
## $ track_name              <chr> "I Don't Care (with Justin Bieber) - Loud ...
## $ track_artist            <chr> "Ed Sheeran", "Maroon 5", "Zara Larsson", ...
## $ track_popularity        <dbl> 66, 67, 70, 60, 69, 67, 62, 69, 68, 67, 58...
## $ track_album_id          <chr> "2oCs0DGTsRO98Gh5ZSl2Cx", "63rPSO264uRjW1X...
## $ track_album_name        <chr> "I Don't Care (with Justin Bieber) [Loud L...
## $ track_album_release_date <chr> "2019-06-14", "2019-12-13", "2019-07-05", ...
## $ playlist_name           <chr> "Pop Remix", "Pop Remix", "Pop Remix", "Po...
## $ playlist_id             <chr> "37i9dQZF1DXcZDD7cfEKhW", "37i9dQZF1DXcZDD...
## $ playlist_genre          <chr> "pop", "pop", "pop", "pop", "pop", "pop", ...
## $ playlist_subgenre       <chr> "dance pop", "dance pop", "dance pop", "da...
## $ danceability            <dbl> 0.748, 0.726, 0.675, 0.718, 0.650, 0.675, ...
```

```
## $ energy                <dbl> 0.916, 0.815, 0.931, 0.930, 0.833, 0.919, ...
## $ key                   <dbl> 6, 11, 1, 7, 1, 8, 5, 4, 8, 2, 6, 8, 1, 5,...
## $ loudness              <dbl> -2.634, -4.969, -3.432, -3.778, -4.672, -5...
## $ mode                  <dbl> 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, ...
## $ speechiness           <dbl> 0.0583, 0.0373, 0.0742, 0.1020, 0.0359, 0....
## $ acousticness          <dbl> 0.10200, 0.07240, 0.07940, 0.02870, 0.0803...
## $ instrumentalness      <dbl> 0.00e+00, 4.21e-03, 2.33e-05, 9.43e-06, 0....
## $ liveness              <dbl> 0.0653, 0.3570, 0.1100, 0.2040, 0.0833, 0....
## $ valence               <dbl> 0.518, 0.693, 0.613, 0.277, 0.725, 0.585, ...
## $ tempo                 <dbl> 122.036, 99.972, 124.008, 121.956, 123.976...
## $ duration_ms           <dbl> 194754, 162600, 176616, 169093, 189052, 16...
```

EXERCISE

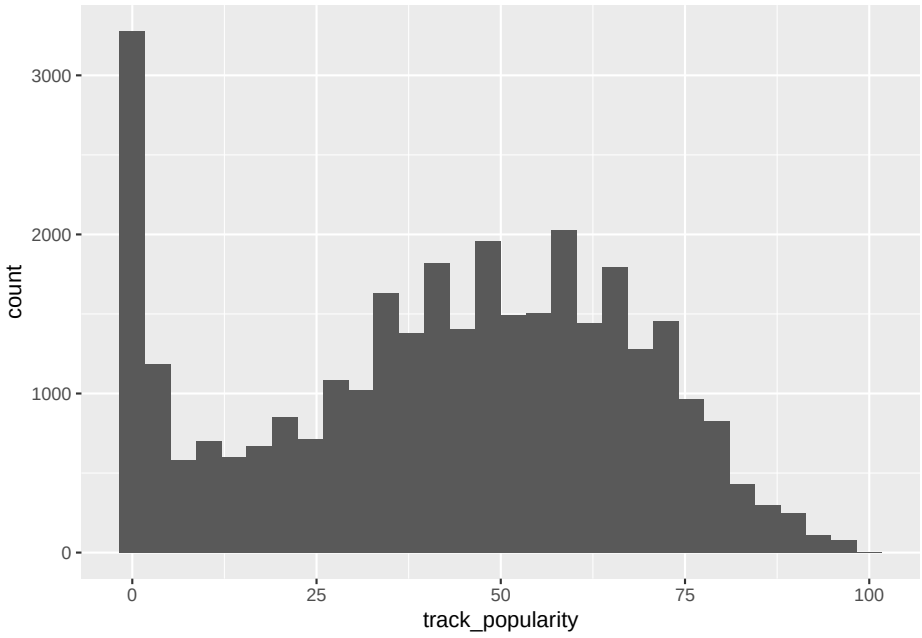
1) What variables do we have in this data?

2) What questions can you ask about this data?

8.3 What to plot?

The first thing you need to do is define what you want to plot. If you've never plotted data before, you might not be familiar with the different types of charts you can create.

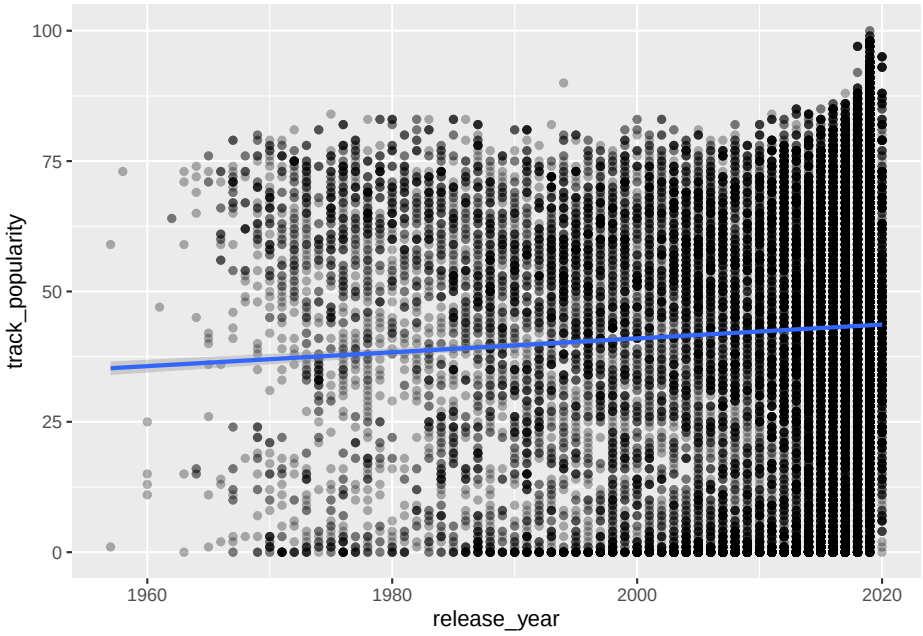
Here's a few (can you tell what type of plot these are?):



1) What is the plot above called?

2) What variable(s) are we plotting?

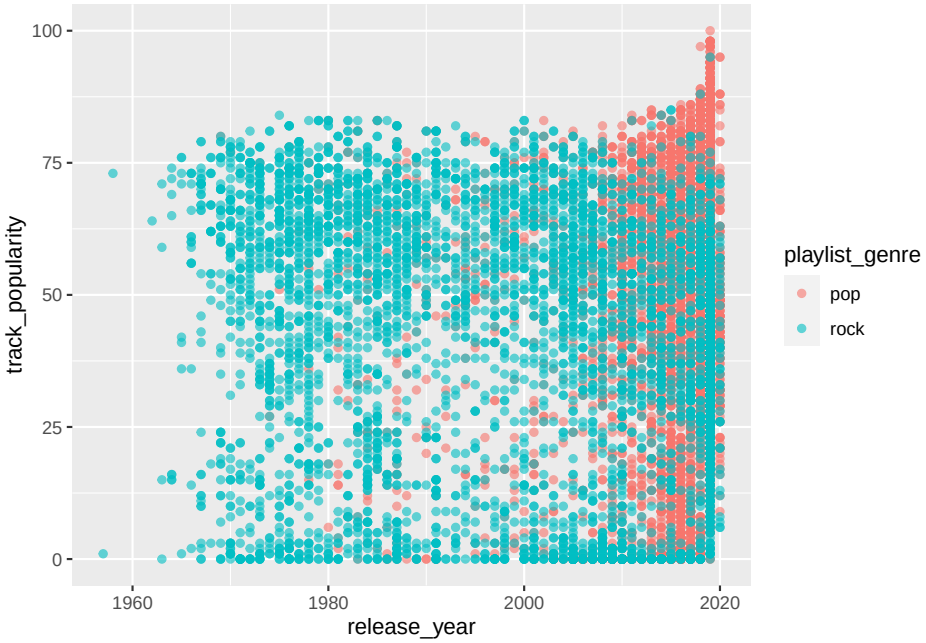
3) What can we conclude based on this plot?



1) What is the plot above called?

2) What variable(s) are we plotting?

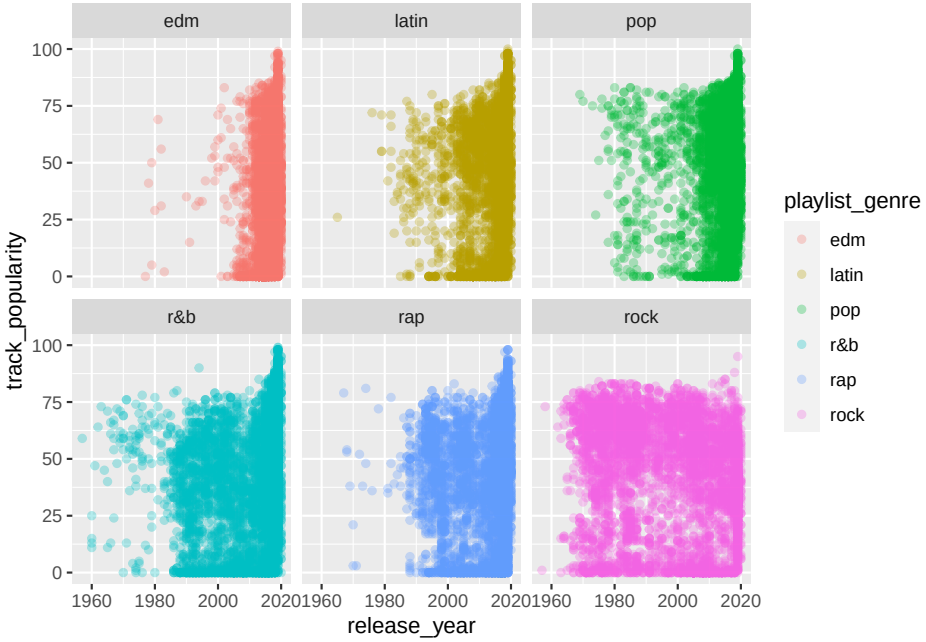
3) What can we conclude based on this plot?



1) What is the plot above called?

2) What variable(s) are we plotting?

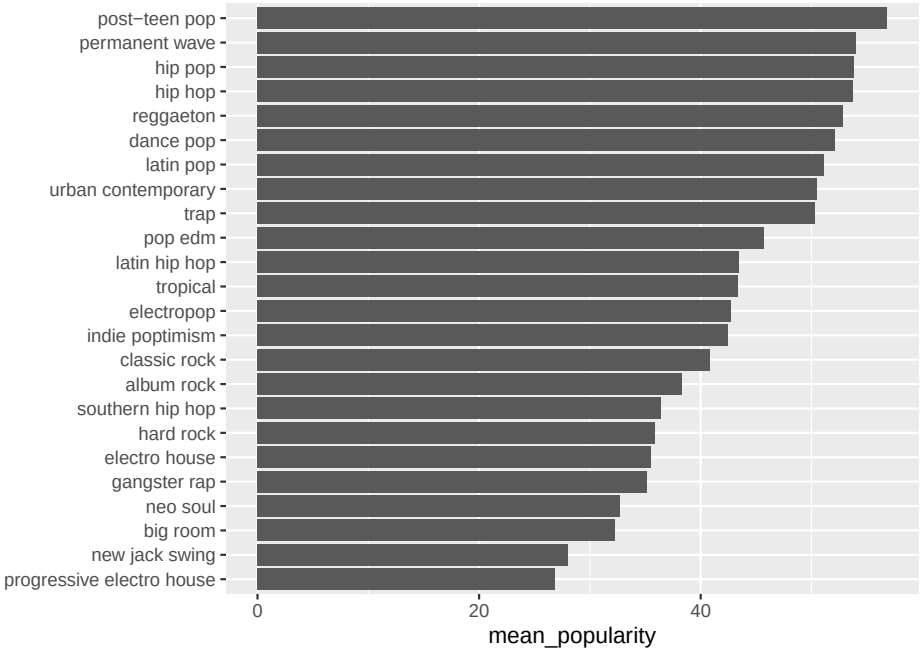
3) What can we conclude based on this plot?



1) What is the plot above called?

2) What variable(s) are we plotting?

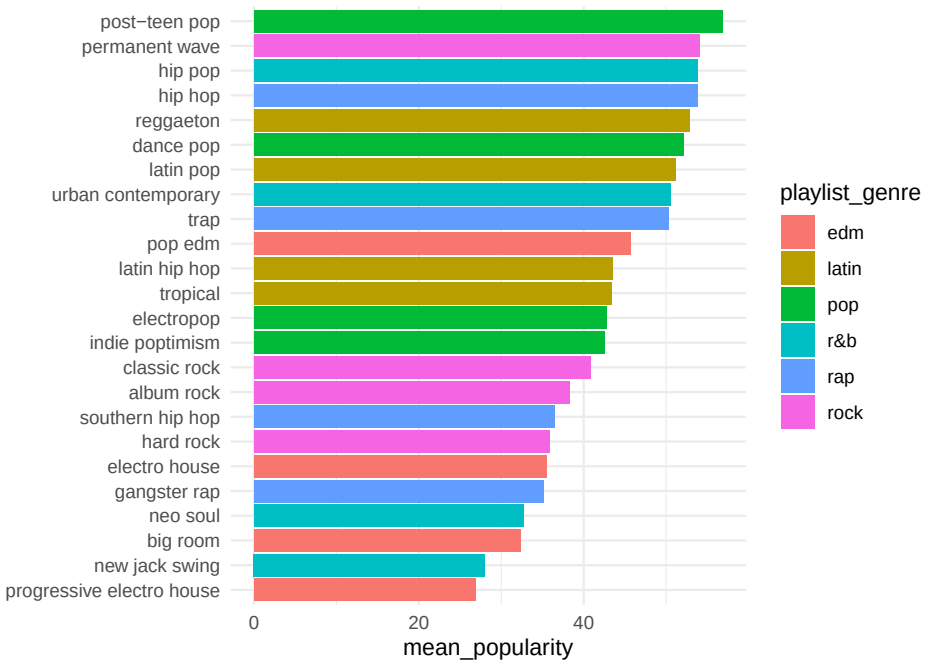
3) What can we conclude based on this plot?



1) What is the plot above called?

2) What variable(s) are we plotting?

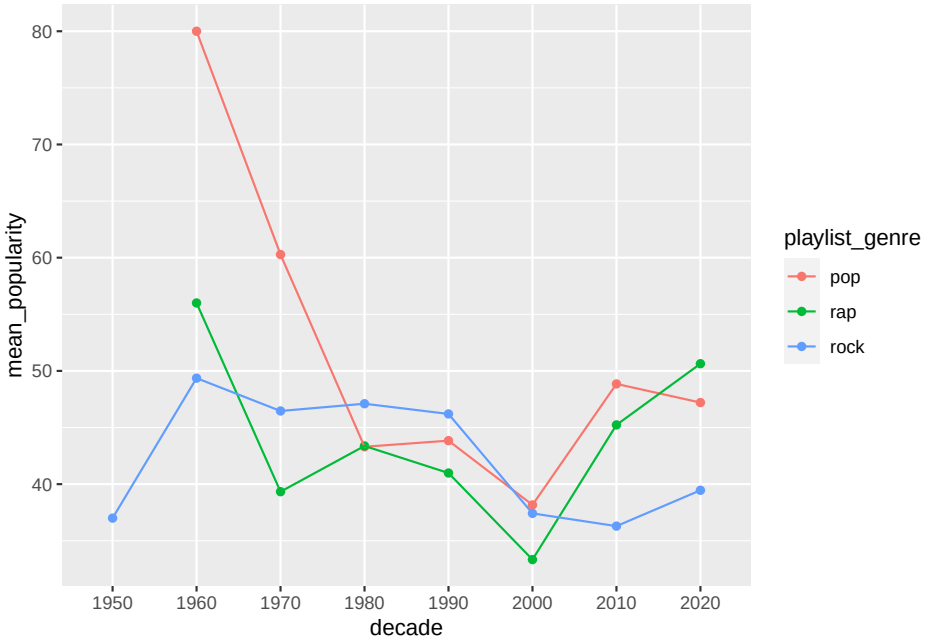
3) What can we conclude based on this plot?



1) What is the plot above called?

2) What variable(s) are we plotting?

3) What can we conclude based on this plot?



8.4 Aesthetic Mappings

You map your aesthetics using the `aes()` function, which can be place inside of the `ggplot()` function.

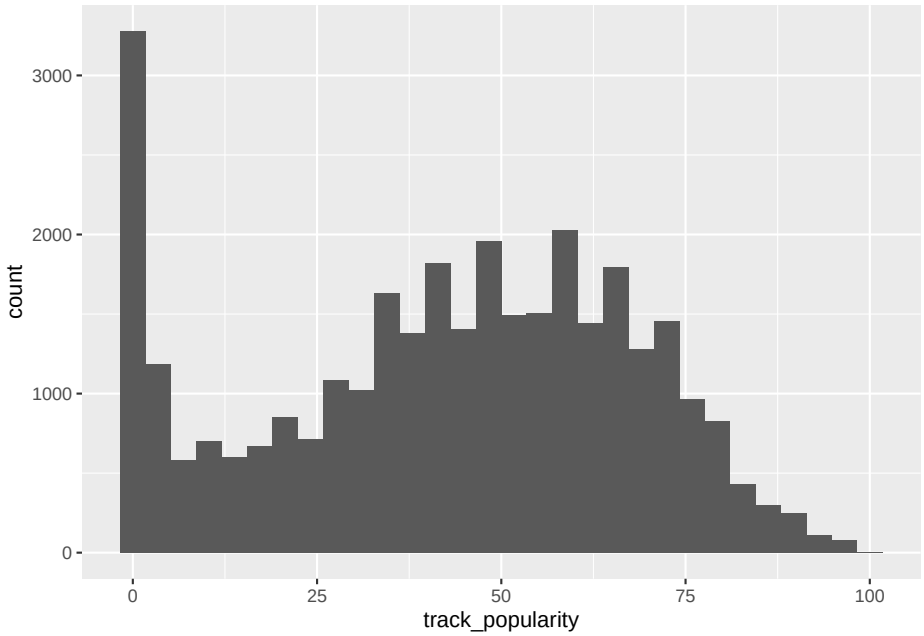
8.4.1 One continuous variable

You need to map at least one variable when you are plotting. Check the help for `geom_histogram` to see what kind of variable you can plot in a histogram.

The variable `track_popularity` is continuous.

```
spotify_songs %>%
  ggplot(aes(x = track_popularity)) +
  geom_histogram()
```

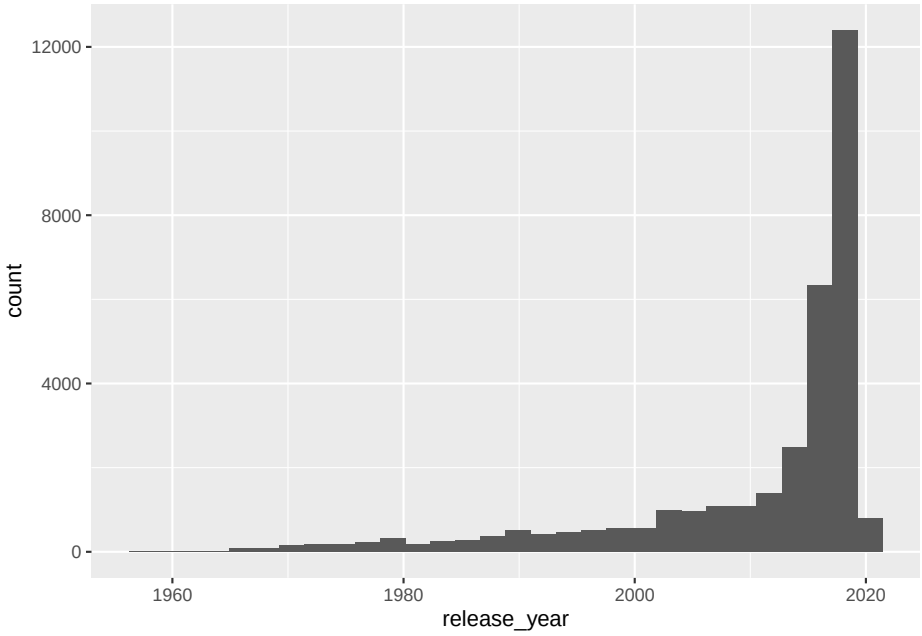
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The variable `release_year` is also continuous.

```
spotify_songs %>%
  ggplot(aes(x = release_year)) +
  geom_histogram()
```

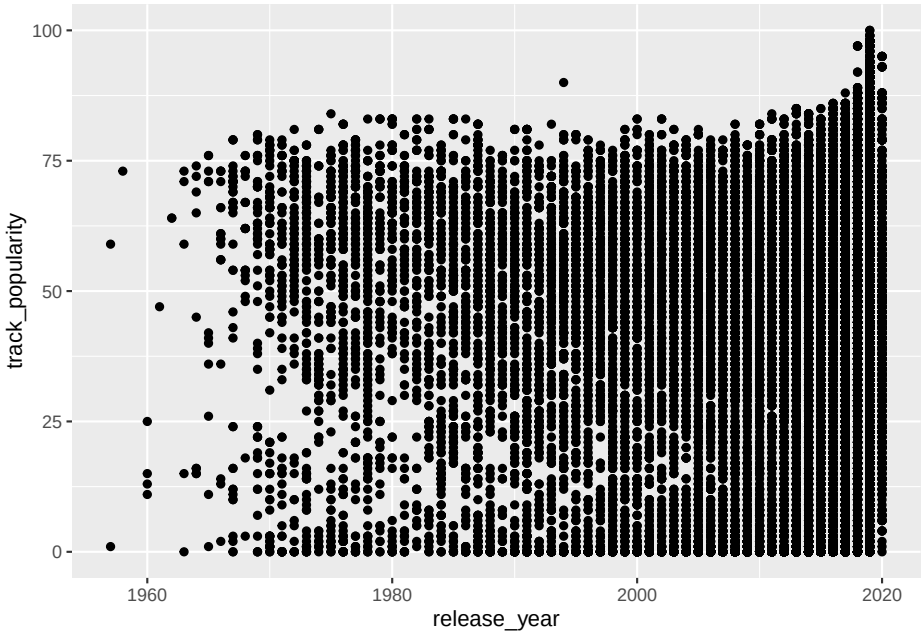
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



8.4.2 Two numeric variables

Two-dimensional plots have two axis, `x` (horizontal) and `y` (vertical).

```
spotify_songs %>%
  ggplot(aes(x = release_year,
             y = track_popularity)) +
  geom_point()
```

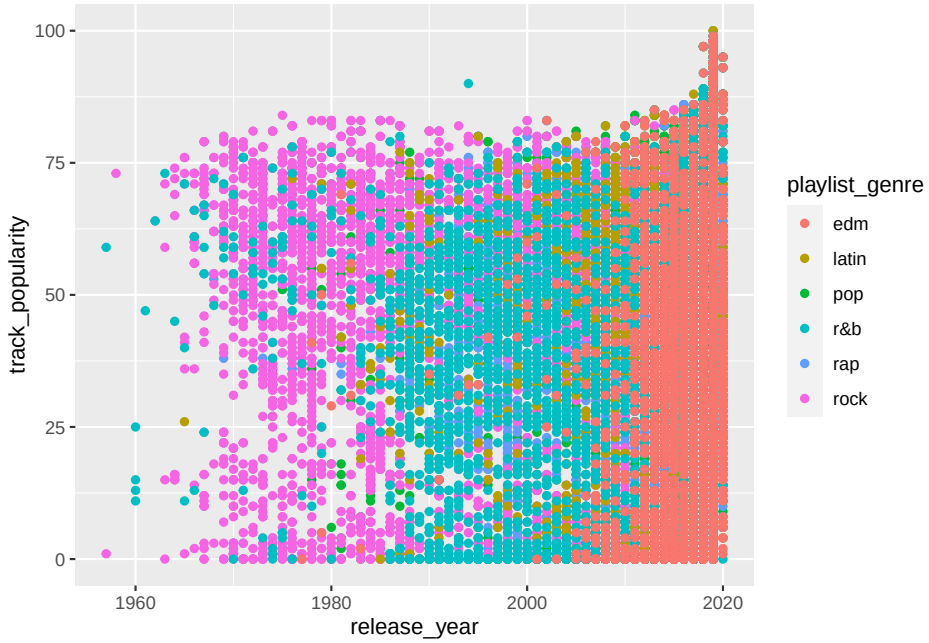


8.5 geom_ (i.e., Geometric Objects)

Functions such as `geom_histogram()`, `geom_point()`, and `geom_col()` are geometric objects and determine what type of plot R draws.

You can also map other elements of your chart in addition to position (i.e., `x` and `y`), such as color, size, and shape.

```
spotify_songs %>%
  ggplot(aes(x = release_year,
             y = track_popularity,
             color = playlist_genre)) +
  geom_point()
```



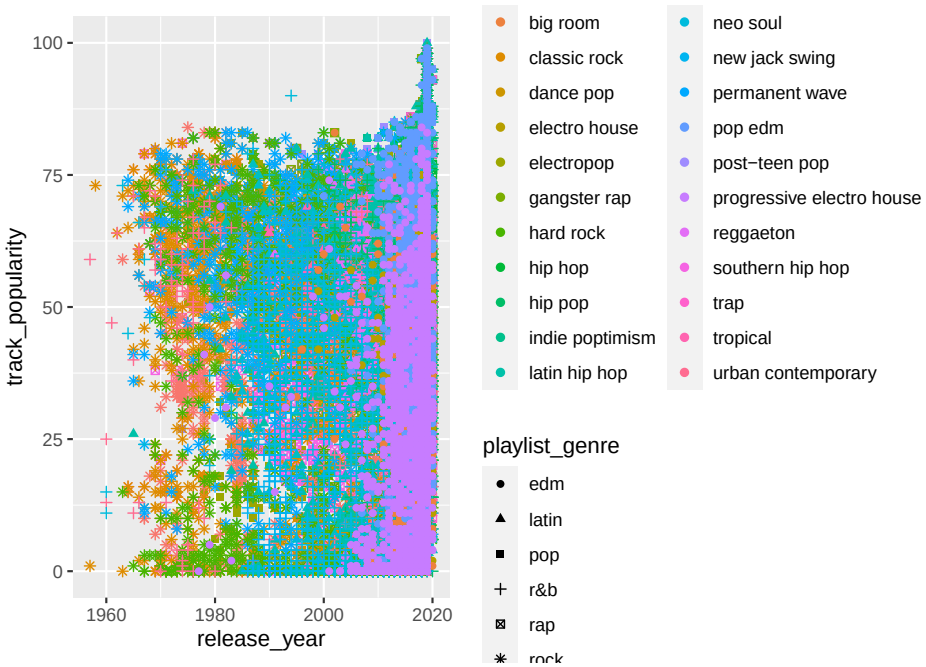
EXERCISE

Check the help page for `geom_point` (enter `?geom_point` in your console). Change `geom_point()` to the suggested variations in its help page.

8.6 More mappings with aes()

In addition to `color` you can also add `size` and `shape` to `aes()`.

```
spotify_songs %>%
  ggplot(aes(x = release_year,
             y = track_popularity,
             color = playlist_subgenre,
             shape = playlist_genre)) +
  geom_point()
```



The plot above is too messy, there's too much information. You often need to transform your data before plotting it.

EXERCISE

Summarize mean `track_popularity` by `release_year`, `playlist_genre` and `playlist_subgenre`.

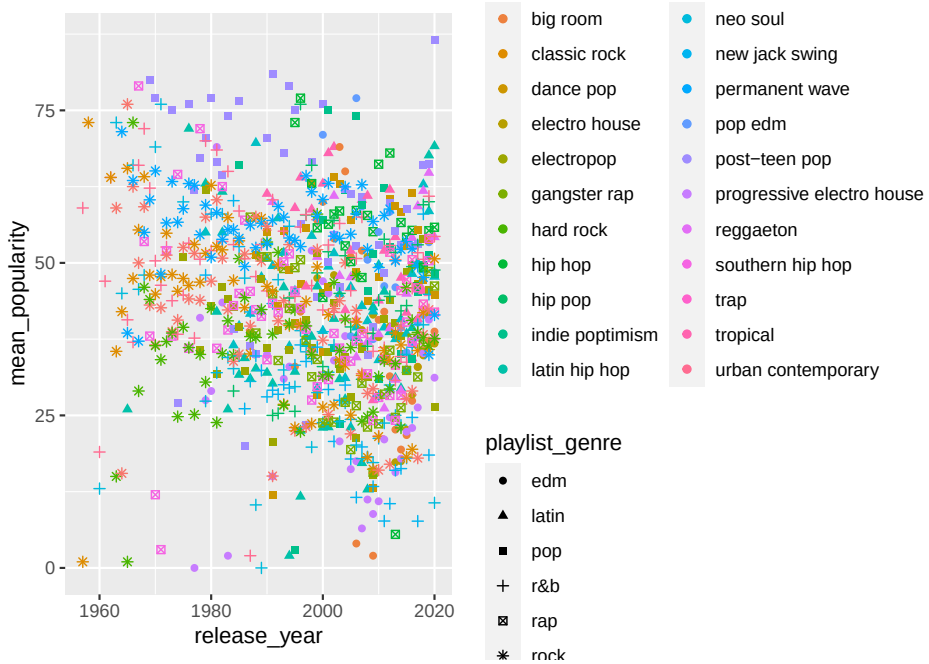
Your summarized data frame should look something like this:

```
## `summarise()` regrouping output by 'release_year', 'playlist_genre' (override with `.groups` a
```

```
## # A tibble: 883 x 4
## # Groups:   release_year, playlist_genre [302]
##    release_year playlist_genre playlist_subgenre  mean_popularity
##           <dbl> <chr>          <chr>                        <dbl>
##  1         1957 r&b            urban contemporary              59
##  2         1957 rock           classic rock                     1
##  3         1958 rock           classic rock                    73
##  4         1960 r&b            neo soul                        13
##  5         1960 r&b            urban contemporary              19
##  6         1961 r&b            urban contemporary              47
##  7         1962 r&b            urban contemporary              64
##  8         1962 rock           classic rock                    64
##  9         1963 r&b            neo soul                        73
## 10         1963 rock           album rock                      59
## # ... with 873 more rows
```

Now you can plot your summarized data.

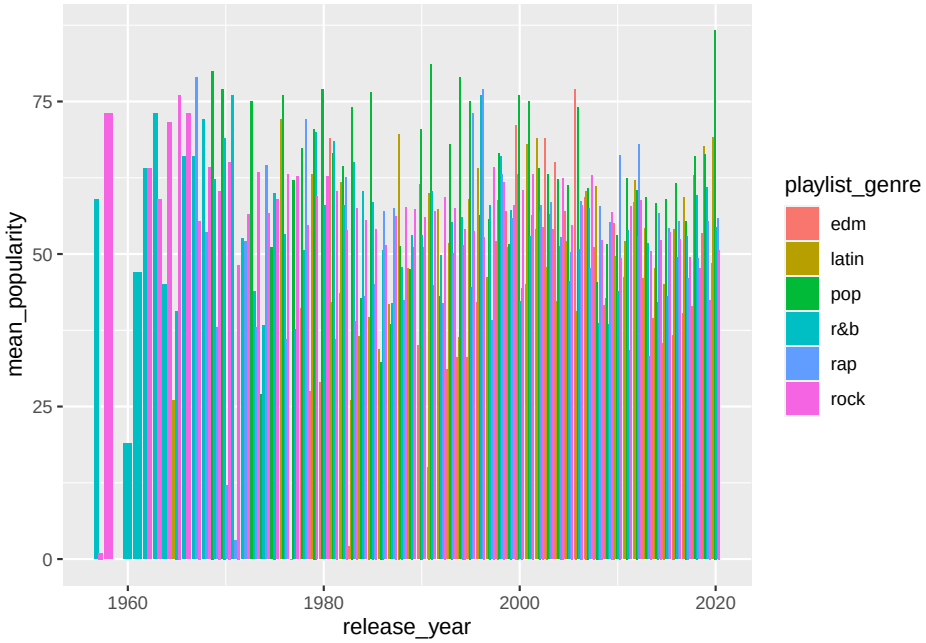
```
spotify_summary %>%
  ggplot(aes(x = release_year,
             y = mean_popularity,
             color = playlist_subgenre,
             shape = playlist_genre)) +
  geom_point()
```



Still super messy. Shape is not really a good way to do this.

Let's try a bar plot.

```
spotify_summary %>%
  ggplot(aes(x = release_year,
             y = mean_popularity,
             fill = playlist_genre)) +
  geom_col(position = "dodge")
```



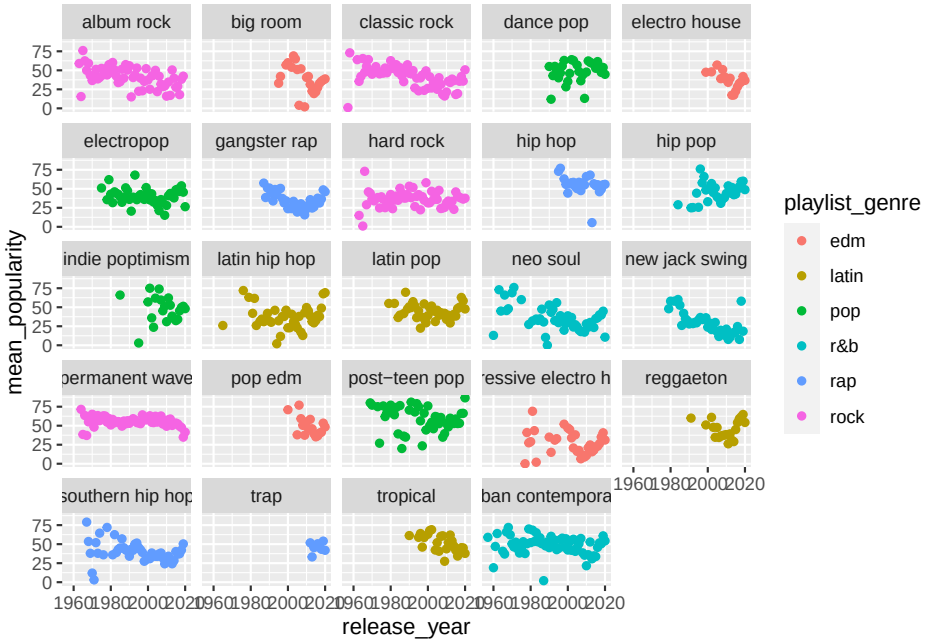
Not great either, too much going on.

8.7 Facets

You can use the `facet_wrap()` function to split your plotting into several smaller plots, usually by a categorical variable.

Let's try the scatterplot first.

```
spotify_summary %>%
  ggplot(aes(x = release_year,
             y = mean_popularity,
             color = playlist_genre)) +
  geom_point() +
  facet_wrap(~playlist_subgenre)
```



What about a bar plot?

```
spotify_summary %>%
  ggplot(aes(x = release_year,
             y = mean_popularity,
             fill = playlist_genre)) +
  geom_col(position = "dodge") +
  facet_wrap(~playlist_subgenre)
```

8.8 More Summarize

We can also plot categorical variables on the `x` axis.

EXERCISE

Summarize mean `track_popularity` by `playlist_genre`.

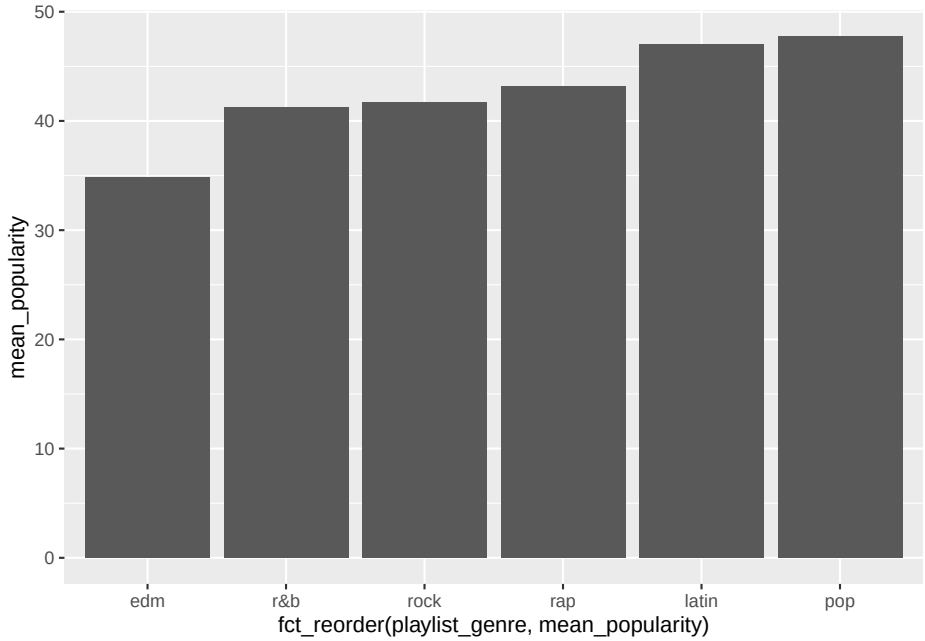
Your summarized data frame should look something like this:

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 6 x 2
##   playlist_genre mean_popularity
##   <chr>                    <dbl>
## 1 edm                       34.8
## 2 latin                     47.0
## 3 pop                       47.7
## 4 r&b                       41.2
## 5 rap                       43.2
## 6 rock                      41.7
```

Now plot a bar chart mapping `x` to `playlist_genre`, `y` to `mean_popularity`.

Your plot should look something like this:



8.9 DATA CHALLENGE 03

Accept data challenge 03 assignment

Chapter 9

Data Visualization II

We will continue working with the spotify data set we worked with last week. The objectives of this module are as follows: by the end of this module you will be able to …

1) Explore a large data frame to decide what part of the data you want to focus on
2) Create subsets of your original data frame
3) Create summarizations of your data based on different variables
4) Plot these summarizations

```
## Rows: 32,833
## Columns: 25
## $ track_id                <chr> "6f807x0ima9a1j3VPbc7VN", "0r7CVbZTWZgbTCY...
## $ track_name              <chr> "I Don't Care (with Justin Bieber) - Loud ...
## $ track_artist            <chr> "Ed Sheeran", "Maroon 5", "Zara Larsson", ...
## $ track_popularity        <dbl> 66, 67, 70, 60, 69, 67, 62, 69, 68, 67, 58...
## $ track_album_id          <chr> "2oCs0DGTsRO98Gh5ZSl2Cx", "63rPSO264uRjW1X...
## $ track_album_name        <chr> "I Don't Care (with Justin Bieber) [Loud L...
## $ track_album_release_date <chr> "2019-06-14", "2019-12-13", "2019-07-05", ...
## $ playlist_name           <chr> "Pop Remix", "Pop Remix", "Pop Remix", "Po...
## $ playlist_id             <chr> "37i9dQZF1DXcZDD7cfEKhW", "37i9dQZF1DXcZDD...
## $ playlist_genre          <chr> "pop", "pop", "pop", "pop", "pop", "pop", ...
## $ playlist_subgenre       <chr> "dance pop", "dance pop", "dance pop", "da...
## $ danceability            <dbl> 0.748, 0.726, 0.675, 0.718, 0.650, 0.675, ...
## $ energy                  <dbl> 0.916, 0.815, 0.931, 0.930, 0.833, 0.919, ...
## $ key                     <dbl> 6, 11, 1, 7, 1, 8, 5, 4, 8, 2, 6, 8, 1, 5,...
## $ loudness                <dbl> -2.634, -4.969, -3.432, -3.778, -4.672, -5...
## $ mode                    <dbl> 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, ...
## $ speechiness             <dbl> 0.0583, 0.0373, 0.0742, 0.1020, 0.0359, 0....
```

```
## $ acousticness            <dbl> 0.10200, 0.07240, 0.07940, 0.02870, 0.0803...
## $ instrumentalness        <dbl> 0.00e+00, 4.21e-03, 2.33e-05, 9.43e-06, 0....
## $ liveness                <dbl> 0.0653, 0.3570, 0.1100, 0.2040, 0.0833, 0....
## $ valence                 <dbl> 0.518, 0.693, 0.613, 0.277, 0.725, 0.585, ...
## $ tempo                   <dbl> 122.036, 99.972, 124.008, 121.956, 123.976...
## $ duration_ms             <dbl> 194754, 162600, 176616, 169093, 189052, 16...
## $ release_year            <dbl> 2019, 2019, 2019, 2019, 2019, 2019, 2019, ...
## $ decade                  <dbl> 2010, 2010, 2010, 2010, 2010, 2010, 2010, ...
```

9.1 Data Viz by Artist

QUESTION: Which artist (from just a few) is most popular? Does that change across different decades?

9.1.1 Explore Artist Info

Let's check who the artists are in this data set. Check what the unique values are for the `track_artist` variable using `select()` and `unique()`.

```
## # A tibble: 10,693 x 1
##    track_artist
##    <chr>
##  1 Ed Sheeran
##  2 Maroon 5
##  3 Zara Larsson
##  4 The Chainsmokers
##  5 Lewis Capaldi
##  6 Katy Perry
##  7 Sam Feldt
##  8 Avicii
##  9 Shawn Mendes
## 10 Ellie Goulding
## # ... with 10,683 more rows
```

Who's the artist with the most songs? Use `count()` and `arrange()` to find out.

```
## # A tibble: 10,693 x 2
##    track_artist                n
##    <chr>                   <int>
##  1 Martin Garrix             161
##  2 Queen                     136
##  3 The Chainsmokers          123
##  4 David Guetta              110
##  5 Don Omar                  102
##  6 Drake                     100
```

```
##  7 Dimitri Vegas & Like Mike    93
##  8 Calvin Harris                91
##  9 Hardwell                     84
## 10 Kygo                         83
## # ... with 10,683 more rows
```

What genre are these artists classified as?

```
## # A tibble: 13,175 x 3
##    track_artist              playlist_genre     n
##    <chr>                     <chr>          <int>
##  1 Queen                     rock             134
##  2 Martin Garrix             edm              125
##  3 Don Omar                  latin            100
##  4 Dimitri Vegas & Like Mike edm               79
##  5 Guns N' Roses             rock              76
##  6 Hardwell                  edm               76
##  7 Logic                     rap               65
##  8 Daddy Yankee              latin             61
##  9 David Guetta              edm               60
## 10 Wisin & Yandel            latin             60
## # ... with 13,165 more rows
```

What can we conclude about artist tracks and `playlist_genre`?

Let's look at specific artist of our choosing. I'm looking at `The Cranberries`, `The Beatles` and `Queen`. What genres are their songs classfied as?

```
## # A tibble: 4 x 3
##   track_artist    playlist_genre     n
##   <chr>           <chr>          <int>
## 1 Queen           pop                2
## 2 Queen           rock             134
## 3 The Beatles     rock              19
## 4 The Cranberries rock              45
```

What are the two pop songs by Queen? Use `filter()` and `select()` to find out.

```
## # A tibble: 2 x 1
##   track_name
##   <chr>
## 1 Don't Stop Me Now - 2011 Mix
## 2 Radio Ga Ga
```

9.1.2 Create new data frame with selected artists

Create another data frame that is a subset of the original `spotify_songs` data frame to start visualizing info about the artists you chose.

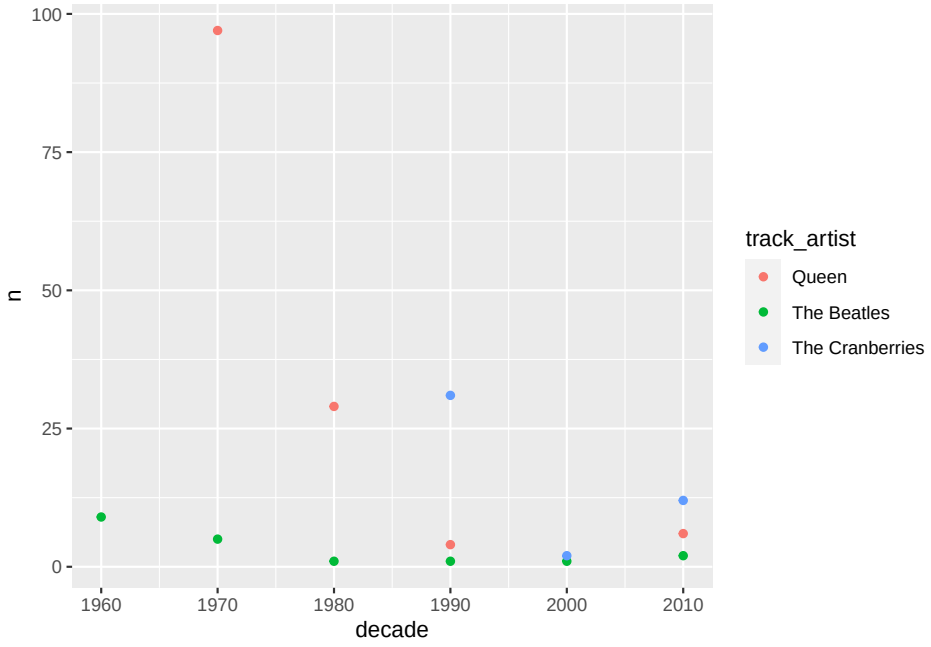
```
# filter original data frame to create new data frame with selected artists
spotify_tc_tv_q <- spotify_songs %>%
  filter(track_artist %in% c('The Cranberries', 'The Beatles', 'Queen'))

# inspect new data frame
glimpse(spotify_tc_tv_q)
```

```
## Rows: 200
## Columns: 25
## $ track_id                <chr> "7hQJA50XrCWABAu5v6QZ4i", "1lpFXKKckqVkyAN...
## $ track_name              <chr> "Don't Stop Me Now - 2011 Mix", "Radio Ga ...
## $ track_artist            <chr> "Queen", "Queen", "The Beatles", "The Cran...
## $ track_popularity        <dbl> 75, 3, 1, 43, 42, 44, 40, 40, 38, 37, 37, ...
## $ track_album_id          <chr> "21HMAUrbbYSj9NiPPlGumy", "39MMaY4ampwjkSO...
## $ track_album_name        <chr> "Jazz (Deluxe Remastered Version)", "The W...
## $ track_album_release_date <chr> "1978-11-10", "1984-02-27", "1996-03-18", ...
## $ playlist_name           <chr> "Dr. Q's Prescription Playlist\U0001f48a",...
## $ playlist_id             <chr> "6jAPdgY9XmxC9cgkXAVmVv", "65HtIbyFkaQPflC...
## $ playlist_genre          <chr> "pop", "pop", "rock", "rock", "rock", "roc...
## $ playlist_subgenre       <chr> "post-teen pop", "electropop", "album rock...
## $ danceability            <dbl> 0.563, 0.762, 0.388, 0.529, 0.473, 0.437, ...
## $ energy                  <dbl> 0.865, 0.414, 0.677, 0.845, 0.598, 0.785, ...
## $ key                     <dbl> 5, 5, 8, 0, 6, 4, 0, 9, 7, 9, 7, 0, 0, 9, ...
## $ loudness                <dbl> -5.277, -12.036, -7.262, -5.432, -5.101, -...
## $ mode                    <dbl> 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, ...
## $ speechiness             <dbl> 0.1600, 0.0379, 0.0301, 0.0294, 0.0268, 0....
## $ acousticness            <dbl> 0.047200, 0.173000, 0.052700, 0.000199, 0....
## $ instrumentalness        <dbl> 1.91e-04, 1.11e-04, 1.07e-02, 1.74e-01, 7....
## $ liveness                <dbl> 0.7700, 0.0942, 0.2210, 0.2270, 0.1250, 0....
## $ valence                 <dbl> 0.6010, 0.7310, 0.4240, 0.5710, 0.0565, 0....
## $ tempo                   <dbl> 156.271, 112.398, 175.818, 109.093, 93.022...
## $ duration_ms             <dbl> 209413, 349133, 234053, 256387, 239947, 25...
## $ release_year            <dbl> 1978, 1984, 1996, 2019, 2019, 2019, 2019, ...
## $ decade                  <dbl> 1970, 1980, 1990, 2010, 2010, 2010, 2010, ...
```

9.1.3 Plotting

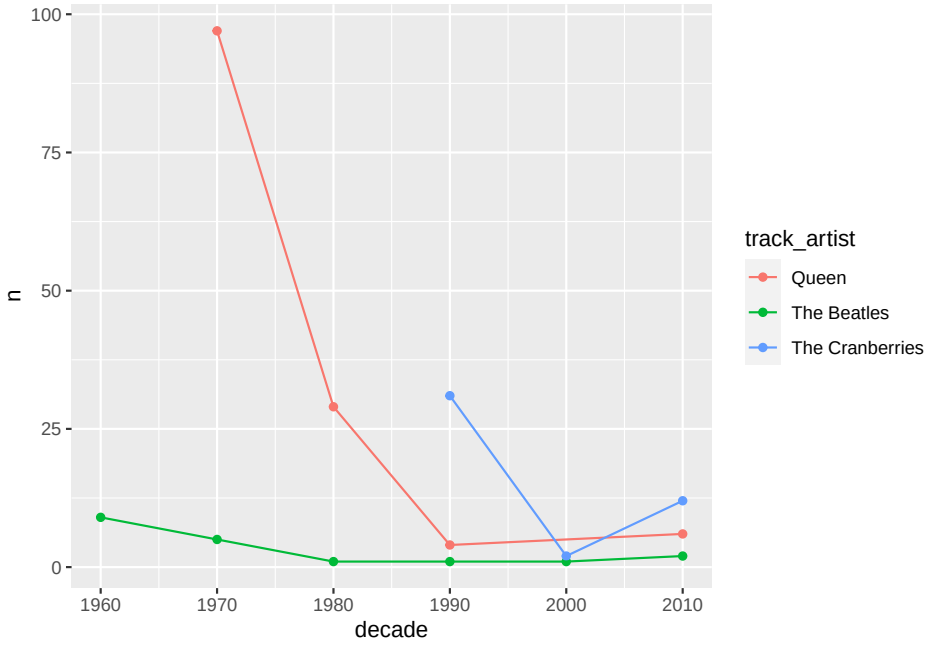
Plot song count (x) by `decade` (y) the songs were release across `track_artist` (color). You need a `count` of `track_artist` and `decade` for this plot.

```
spotify_tc_tv_q %>%
  count(track_artist, decade) %>%
  ggplot(aes(x = decade, y = n, color = track_artist)) +
  geom_point()
```



To make tendencies clearer, we can add `geom_line` to our plot. We need a new aesthetics for the lines to connect the right points, called `group`. In this case, `group` takes the same variable as the `color` mapping.

```
spotify_tc_tv_q %>%
  count(track_artist, decade) %>%
  ggplot(aes(x = decade, y = n, color = track_artist)) +
  geom_point() +
  geom_line(aes(group = track_artist))
```

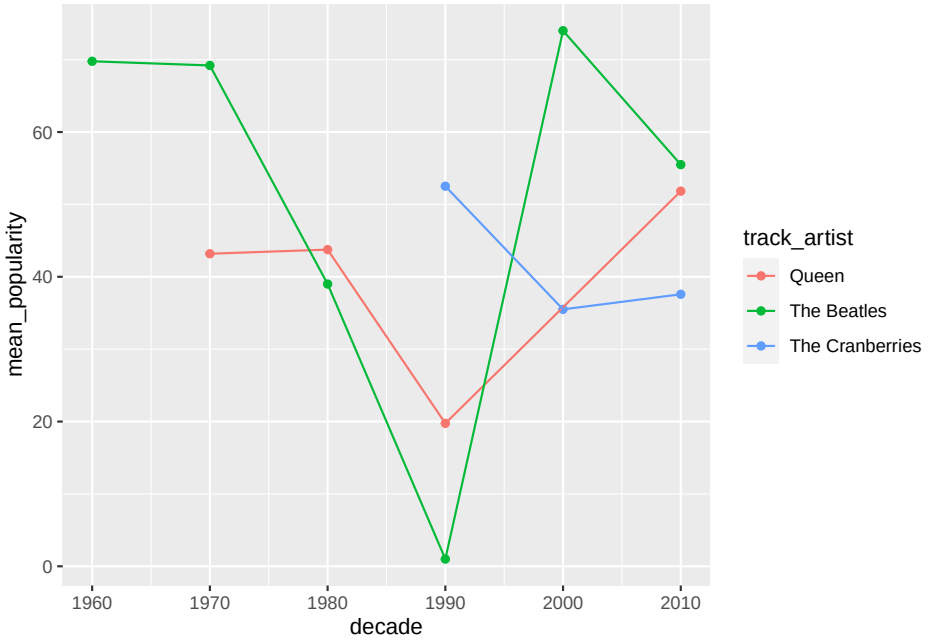


From the plot above, what can we conclude about the selected artists? When did they start releasing songs?

Let's look at `track_popularity` by artist across decade. For this, we need `group_by` and `summarise` before we can build our plot.

```
spotify_tc_tv_q %>%
  group_by(track_artist, decade) %>%
  summarise(mean_popularity = mean(track_popularity)) %>%
  ggplot(aes(x = decade, y = mean_popularity, color = track_artist)) +
  geom_point() +
  geom_line(aes(group = track_artist))
```

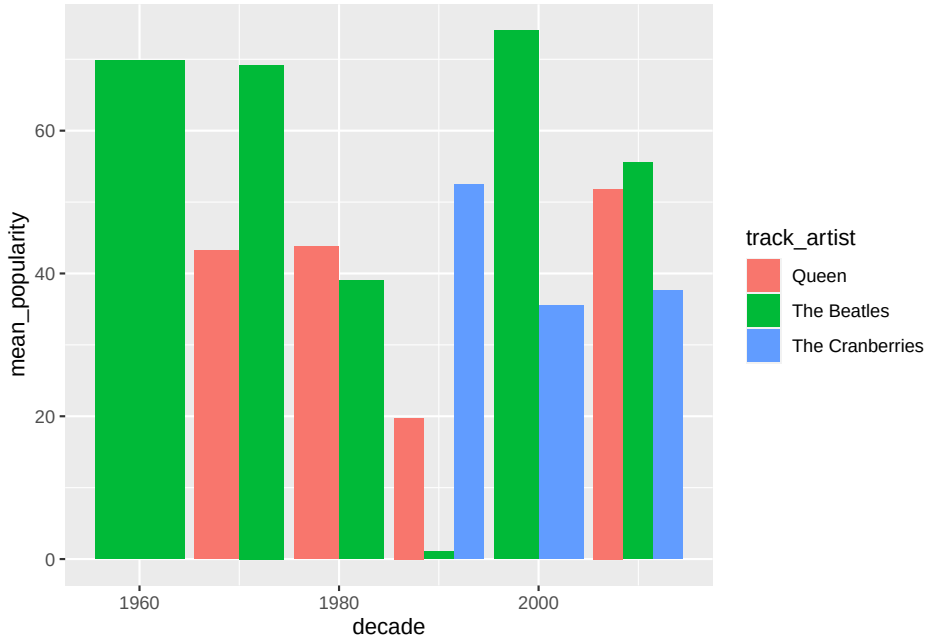
```
## `summarise()` regrouping output by 'track_artist' (override with `.groups` argument)
```



How would this plot look like as a bar plot?

```
spotify_tc_tv_q %>%
  group_by(track_artist, decade) %>%
  summarise(mean_popularity = mean(track_popularity)) %>%
  ggplot(aes(x = decade, y = mean_popularity, fill = track_artist)) +
  geom_col(position = "dodge")
```

```
## `summarise()` regrouping output by 'track_artist' (override with `.groups` argument)
```



Which chart do you think is easier to read? Why?

We have multiple songs per artists, so we can include standard deviation in our `summarise`.

```
spotify_tc_tv_q %>%
  group_by(track_artist, decade) %>%
  summarise(n = n(),
            mean_popularity = mean(track_popularity),
            sd_popularity = sd(track_popularity))
```

```
## `summarise()` regrouping output by 'track_artist' (override with `.groups` argument)
```

```
## # A tibble: 13 x 5
## # Groups:   track_artist [3]
##    track_artist    decade     n mean_popularity sd_popularity
##    <chr>            <dbl> <int>           <dbl>         <dbl>
##  1 Queen             1970    97            43.2         15.1
##  2 Queen             1980    29            43.8         22.5
##  3 Queen             1990     4            19.8         27.6
##  4 Queen             2010     6            51.8         11.7
##  5 The Beatles       1960     9            69.8          6.53
##  6 The Beatles       1970     5            69.2          5.89
##  7 The Beatles       1980     1            39           NA
##  8 The Beatles       1990     1             1           NA
##  9 The Beatles       2000     1            74           NA
## 10 The Beatles       2010     2            55.5          4.95
```

```
## 11 The Cranberries   1990    31            52.5         13.3
## 12 The Cranberries   2000     2            35.5          3.54
## 13 The Cranberries   2010    12            37.6         12.3
```

NAs in our data frame is a problem. We can add `mutate` with `replace_na` to replace these NAs with zero.

```
spotify_tc_tv_q %>%
  group_by(track_artist, decade) %>%
  summarise(n = n(),
            mean_popularity = mean(track_popularity),
            sd_popularity = sd(track_popularity)) %>%
  mutate(sd_popularity = replace_na(sd_popularity, 0))
```

```
## `summarise()` regrouping output by 'track_artist' (override with `.groups` argument)
```

```
## # A tibble: 13 x 5
## # Groups:   track_artist [3]
##    track_artist    decade     n mean_popularity sd_popularity
##    <chr>            <dbl> <int>           <dbl>         <dbl>
##  1 Queen             1970    97            43.2         15.1
##  2 Queen             1980    29            43.8         22.5
##  3 Queen             1990     4            19.8         27.6
##  4 Queen             2010     6            51.8         11.7
##  5 The Beatles       1960     9            69.8          6.53
##  6 The Beatles       1970     5            69.2          5.89
##  7 The Beatles       1980     1            39            0
##  8 The Beatles       1990     1             1            0
##  9 The Beatles       2000     1            74            0
## 10 The Beatles       2010     2            55.5          4.95
## 11 The Cranberries   1990    31            52.5         13.3
## 12 The Cranberries   2000     2            35.5          3.54
## 13 The Cranberries   2010    12            37.6         12.3
```

The data frame looks good, let's add the plot code lines to the block of code above. This time, let's do a bar chart faceted by `track_artist`.

```
spotify_tc_tv_q %>%
  group_by(track_artist, decade) %>%
  summarise(n = n(),
            mean_popularity = mean(track_popularity),
            sd_popularity = sd(track_popularity)) %>%
  mutate(sd_popularity = replace_na(sd_popularity, 0),
         lower = mean_popularity - sd_popularity,
         upper = mean_popularity + sd_popularity)  %>%
  ggplot(aes(x = decade, y = mean_popularity, fill = track_artist)) +
  geom_col() +
  facet_wrap(~track_artist)
```

```
## `summarise()` regrouping output by 'track_artist' (override with `.groups` argument)
```

It looks the same as before. Let's add `geom_errorbar` to it with `ymin` and `ymax` mappings. For that, we need to transform our data frame with `mutate` to calculate `lower` and `upper` variables, which represent the mean **minus** the standard deviation for the lower value of the range, and mean **plus** standard deviation for the upper value of the range.

```
spotify_tc_tv_q %>%
  group_by(track_artist, decade) %>%
  summarise(n = n(),
            mean_popularity = mean(track_popularity),
            sd_popularity = sd(track_popularity)) %>%
  mutate(sd_popularity = replace_na(sd_popularity, 0),
         lower = mean_popularity - sd_popularity,
         upper = mean_popularity + sd_popularity)
```

```
## `summarise()` regrouping output by 'track_artist' (override with `.groups` argument)
```

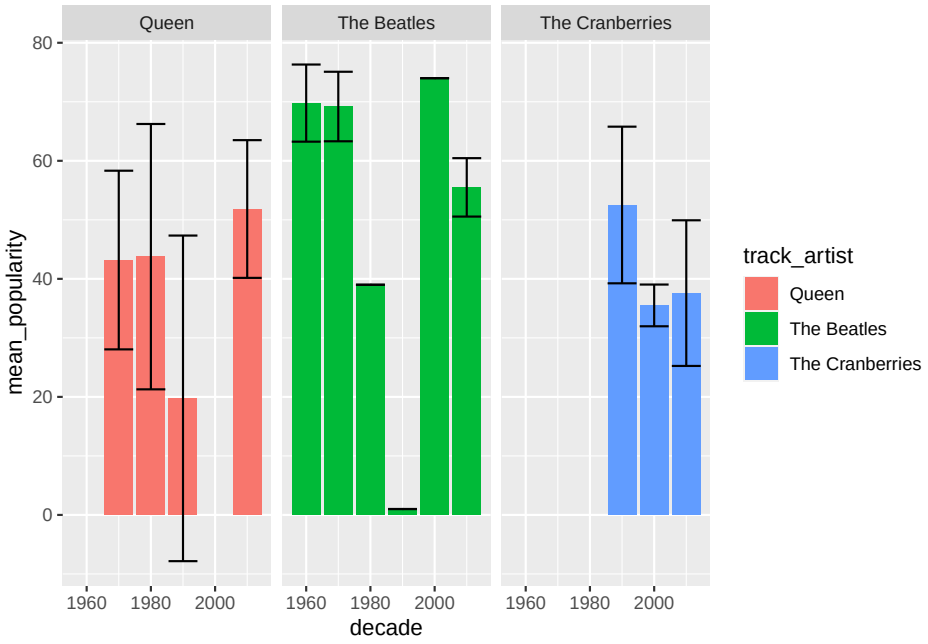
```
## # A tibble: 13 x 7
## # Groups:   track_artist [3]
##    track_artist    decade     n mean_popularity sd_popularity lower upper
##    <chr>            <dbl> <int>           <dbl>         <dbl> <dbl> <dbl>
##  1 Queen             1970    97            43.2          15.1  28.0   58.3
##  2 Queen             1980    29            43.8          22.5  21.3   66.2
##  3 Queen             1990     4            19.8          27.6  -7.83  47.3
##  4 Queen             2010     6            51.8          11.7  40.2   63.5
```

```
##  5 The Beatles       1960     9          69.8          6.53 63.2   76.3
##  6 The Beatles       1970     5          69.2          5.89 63.3   75.1
##  7 The Beatles       1980     1          39            0    39     39
##  8 The Beatles       1990     1           1            0     1      1
##  9 The Beatles       2000     1          74            0    74     74
## 10 The Beatles       2010     2          55.5          4.95 50.6   60.4
## 11 The Cranberries   1990    31          52.5         13.3  39.2   65.8
## 12 The Cranberries   2000     2          35.5          3.54 32.0   39.0
## 13 The Cranberries   2010    12          37.6         12.3  25.2   49.9
```

Now we can use `geom_errorbar`.

```
spotify_tc_tv_q %>%
  group_by(track_artist, decade) %>%
  summarise(n = n(),
            mean_popularity = mean(track_popularity),
            sd_popularity = sd(track_popularity)) %>%
  mutate(sd_popularity = replace_na(sd_popularity, 0),
         lower = mean_popularity - sd_popularity,
         upper = mean_popularity + sd_popularity)  %>%
  ggplot(aes(x = decade, y = mean_popularity, fill = track_artist)) +
  geom_col() +
  geom_errorbar(aes(ymin = lower, ymax = upper)) +
  facet_wrap(~track_artist)
```

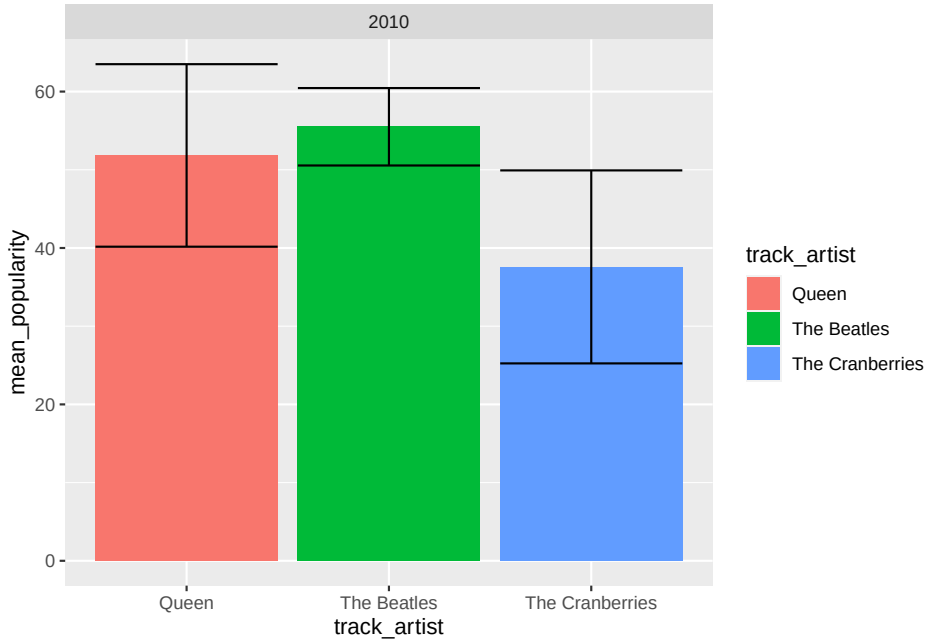
```
## `summarise()` regrouping output by 'track_artist' (override with `.groups` argument)
```



We can do a similar chart but look at the 2010 decade only.

```
spotify_tc_tv_q %>%
  filter(decade == 2010) %>%
  group_by(track_artist, decade) %>%
  summarise(n = n(),
            mean_popularity = mean(track_popularity),
            sd_popularity = sd(track_popularity)) %>%
  mutate(sd_popularity = replace_na(sd_popularity, 0),
         lower = mean_popularity - sd_popularity,
         upper = mean_popularity + sd_popularity)  %>%
  ggplot(aes(x = track_artist, y = mean_popularity, fill = track_artist)) +
  geom_col() +
  geom_errorbar(aes(ymin = lower, ymax = upper)) +
  facet_wrap(~decade)
```

```
## `summarise()` regrouping output by 'track_artist' (override with `.groups` argument)
```

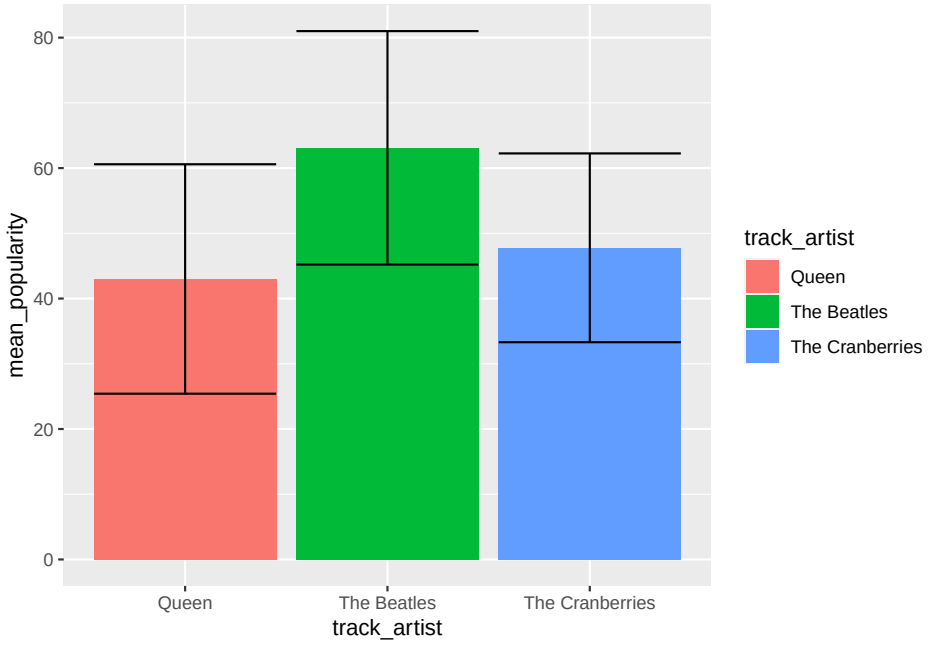


We can also collapse decade, and just look at popularity overall.

```
spotify_tc_tv_q %>%
  group_by(track_artist) %>%
  summarise(n = n(),
            mean_popularity = mean(track_popularity),
            sd_popularity = sd(track_popularity)) %>%
  mutate(sd_popularity = replace_na(sd_popularity, 0),
```

```
        lower = mean_popularity - sd_popularity,
        upper = mean_popularity + sd_popularity)  %>%
  ggplot(aes(x = track_artist, y = mean_popularity, fill = track_artist)) +
  geom_col() +
  geom_errorbar(aes(ymin = lower, ymax = upper))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



9.2 Data Viz by Album

QUESTION: Which Drake album is the most popular?

Let's review the steps to answer our question:

1) Create a new data frame that is a subset of our original data frame
2) Summarize and transform our new data frame to create the variables we need to plot the info we need
3) Try different plots until we find a plot that looks clear

9.2.1 Create new data frame

We first filter our data frame by artist.

```
# filter original data frame to create new data frame with selected artists
spotify_drake <- spotify_songs %>%
  filter(track_artist == 'Drake')

# inspect new data frame
glimpse(spotify_drake)
```

```
## Rows: 100
## Columns: 25
## $ track_id                <chr> "76P07ei8drjrenqtvDbefy", "1xznGGDReH1oQq0...
## $ track_name              <chr> "Hotline Bling", "One Dance", "Too Good", ...
## $ track_artist            <chr> "Drake", "Drake", "Drake", "Drake", "Drake...
## $ track_popularity        <dbl> 0, 20, 12, 72, 12, 10, 83, 83, 86, 68, 15,...
## $ track_album_id          <chr> "2e42oY2oFArkkTENT8UVXD", "3hARKC8cinq3mZL...
## $ track_album_name        <chr> "Views", "Views", "Views", "Thank Me Later...
## $ track_album_release_date <chr> "2016-05-06", "2016-05-06", "2016-05-06", ...
## $ playlist_name           <chr> "BALLARE - "  , "Electropop Hits  2017-20...
## $ playlist_id             <chr> "1CMvQ4Yr5DlYvYzI0Vc2UE", "7kyvBmlc1uSqsTL...
## $ playlist_genre          <chr> "pop", "pop", "pop", "pop", "pop", "pop", ...
## $ playlist_subgenre       <chr> "post-teen pop", "electropop", "electropop...
## $ danceability            <dbl> 0.905, 0.791, 0.804, 0.431, 0.771, 0.893, ...
## $ energy                  <dbl> 0.617, 0.619, 0.648, 0.894, 0.629, 0.639, ...
## $ key                     <dbl> 2, 1, 7, 5, 1, 2, 1, 1, 7, 1, 11, 10, 2, 1...
## $ loudness                <dbl> -8.039, -5.886, -7.805, -2.673, -5.790, -7...
## $ mode                    <dbl> 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, ...
## $ speechiness             <dbl> 0.0596, 0.0532, 0.1170, 0.3300, 0.0511, 0....
## $ acousticness            <dbl> 0.00287, 0.00784, 0.05730, 0.09510, 0.0080...
## $ instrumentalness        <dbl> 4.40e-04, 4.23e-03, 3.49e-05, 0.00e+00, 2....
## $ liveness                <dbl> 0.0484, 0.3510, 0.1020, 0.1880, 0.3560, 0....
## $ valence                 <dbl> 0.572, 0.371, 0.392, 0.604, 0.362, 0.579, ...
## $ tempo                   <dbl> 134.972, 103.989, 117.983, 162.193, 103.91...
## $ duration_ms             <dbl> 267187, 173987, 263373, 258760, 173975, 26...
## $ release_year            <dbl> 2016, 2016, 2016, 2010, 2016, 2015, 2016, ...
## $ decade                  <dbl> 2010, 2010, 2010, 2010, 2010, 2010, 2010, ...
```

What albums are there in this new data frame?

```
spotify_drake %>%
  count(track_album_name) %>%
  arrange(-n)
```

```
## # A tibble: 27 x 2
##    track_album_name                                        n
##    <chr>                                               <int>
##  1 Views                                                  23
##  2 Scorpion                                               16
##  3 More Life                                               9
```

```
##  4 The Best In The World Pack                                    7
##  5 Take Care (Deluxe)                                            5
##  6 What A Time To Be Alive                                       5
##  7 If You're Reading This It's Too Late                          4
##  8 Care Package                                                  3
##  9 Hotline Bling                                                 3
## 10 Top Boy (A Selection of Music Inspired by the Series)         3
## # ... with 17 more rows
```

9.2.2 Summarize data

Now we summarize our data for mean popularity per album.

```
spotify_drake %>%
  group_by(track_album_name) %>%
  summarise(mean_popularity = mean(track_popularity))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

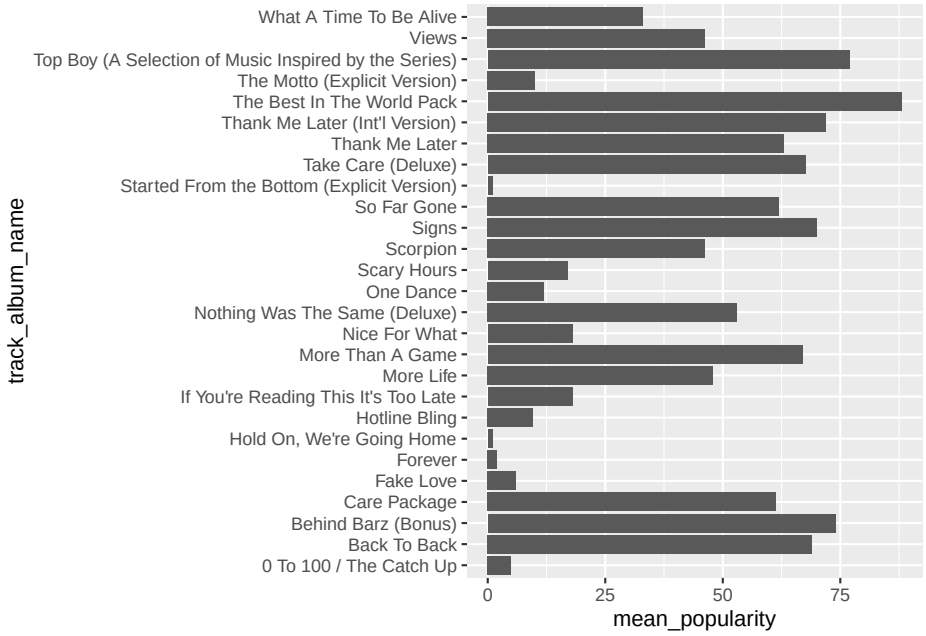
```
## # A tibble: 27 x 2
##    track_album_name                      mean_popularity
##    <chr>                                           <dbl>
##  1 0 To 100 / The Catch Up                          5
##  2 Back To Back                                    69
##  3 Behind Barz (Bonus)                             74
##  4 Care Package                                    61.3
##  5 Fake Love                                        6
##  6 Forever                                          2
##  7 Hold On, We're Going Home                        1
##  8 Hotline Bling                                    9.67
##  9 If You're Reading This It's Too Late            18
## 10 More Life                                       47.9
## # ... with 17 more rows
```

9.2.3 Plot summarized data

We now add the `ggplot` code lines to our summarized data frame.

```
spotify_drake %>%
  group_by(track_album_name) %>%
  summarise(mean_popularity = mean(track_popularity)) %>%
  ggplot(aes(y = track_album_name, x = mean_popularity)) +
  geom_col()
```

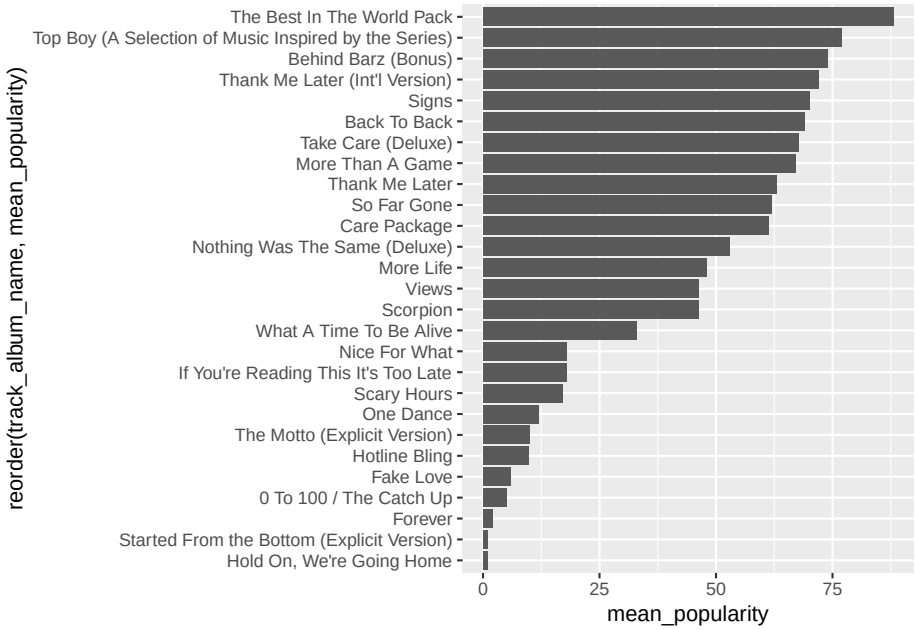
```
## `summarise()` ungrouping output (override with `.groups` argument)
```



Let's order the songs by `mean_popularity`.

```
spotify_drake %>%
  group_by(track_album_name) %>%
  summarise(mean_popularity = mean(track_popularity)) %>%
  ggplot(aes(y = reorder(track_album_name, mean_popularity), x = mean_popularity)) +
  geom_col()
```

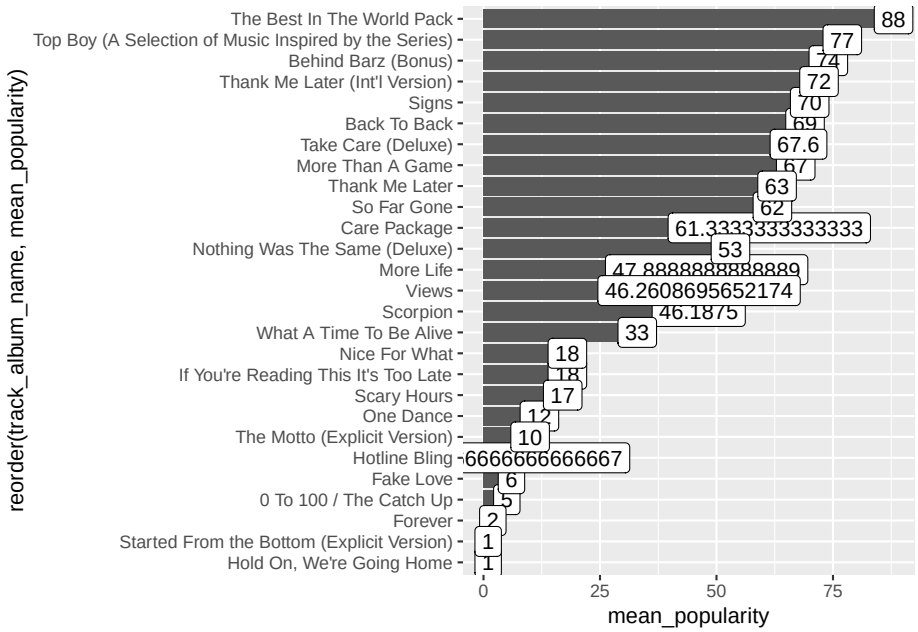
```
## `summarise()` ungrouping output (override with `.groups` argument)
```



We can add labels to the bars, with the mean_popularity for each album using `geom_label`. A new mapping is needed for `label`, which is the same as the x mapping in this case.

```
spotify_drake %>%
  group_by(track_album_name) %>%
  summarise(mean_popularity = mean(track_popularity)) %>%
  ggplot(aes(y = reorder(track_album_name, mean_popularity), x = mean_popularity)) +
  geom_col() +
  geom_label(aes(label = mean_popularity))
```

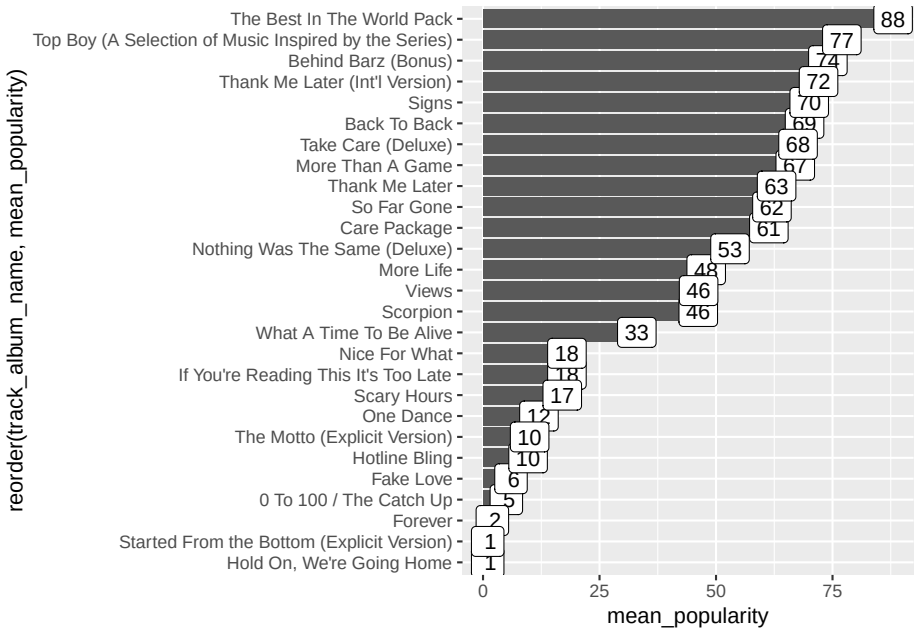
```
## `summarise()` ungrouping output (override with `.groups` argument)
```



We need to clean up the means. We can do that using `format`.

```
spotify_drake %>%
  group_by(track_album_name) %>%
  summarise(mean_popularity = mean(track_popularity)) %>%
  ggplot(aes(y = reorder(track_album_name, mean_popularity), x = mean_popularity)) +
  geom_col() +
  geom_label(aes(label = format(mean_popularity, digits = 1)))
```

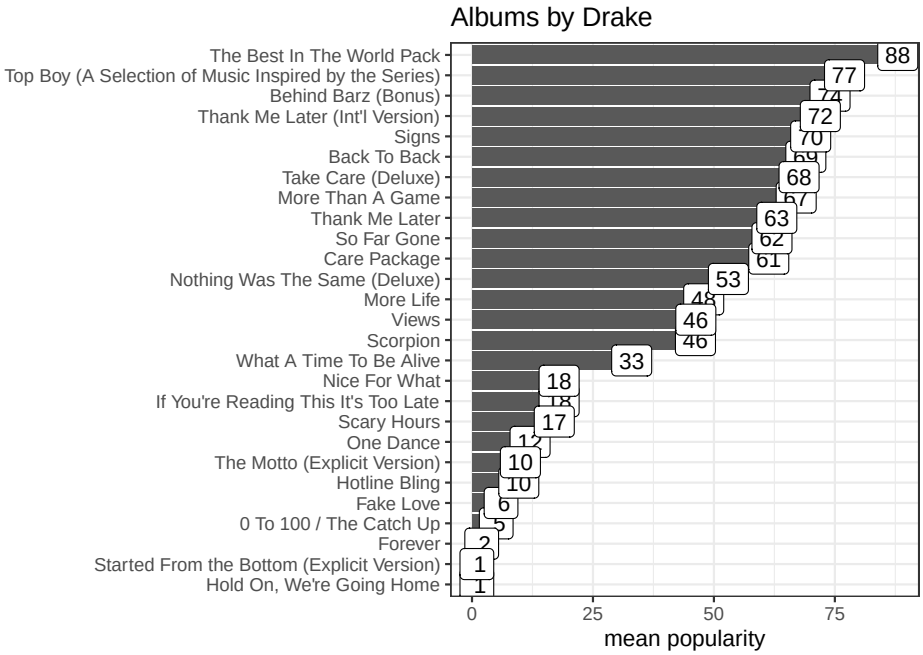
```
## `summarise()` ungrouping output (override with `.groups` argument)
```



We can clean up our chart even more.

```
spotify_drake %>%
  group_by(track_album_name) %>%
  summarise(mean_popularity = mean(track_popularity)) %>%
  ggplot(aes(y = reorder(track_album_name, mean_popularity), x = mean_popularity)) +
  geom_col() +
  geom_label(aes(label = format(mean_popularity, digits = 1))) +
  xlab("mean popularity") +
  ylab("") +
  theme_bw() +
  ggtitle("Albums by Drake")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



9.3 DATA CHALLENGE 04

Accept data challenge 04 assignment

Chapter 10

Data Case Study 1

Chapter 11

Data Case Study 2

Chapter 12

Getting Data

Chapter 13

Data Case Study 3

Chapter 14

Markdown

Chapter 15

Data Case Study 4

Chapter 16

Analysis Reporting

Bibliography

Baumer, B. S., Kaplan, D. T., and Horton, N. J. (2017). *Modern data science with R*. CRC Press.

Beckman, M. D., Çetinkaya-Rundel, M., Horton, N. J., Rundel, C. W., Sullivan, A. J., and Tackett, M. (2020). Implementing version control with git as a learning objective in statistics courses. *arXiv preprint arXiv:2001.01988*.

Caffo, B., Peng, R. D., and Leek, R. H. (2016). *Executive data science: A guide to training and managing the best data scientists*. Leanpub.

Grolemund, G. and Wickham, H. (2018). *R for data science*. O'Reilly.

Hadavand, A., Gooding, I., and Leek, J. T. (2018). Can mooc programs improve student employment prospects? *Available at SSRN 3260695*.

Kross, S., Peng, R. D., Caffo, B. S., Gooding, I., and Leek, J. T. (2020). The democratization of data science education. *The American Statistician*, 74(1):1–7.

Robinson, E. and Nolis, J. (2020). *Build a career in data science*. Manning.