

# About this Book

## Contents

### Syllabus

- About
- Tools and Resources
- Data Science Achievements
- Grading
- Grading Policies
- Course Style Guide
- Support
- General URI Policies
- Communications & Office Hours

### Notes

- 1. Welcome & What is Data Science
- 2. Iterables and Pandas Data Frames
- 3. Pandas Data Frames and More Iterable Types
- 4. Exploratory Data Analysis
- 5. Visualization
- 6. Cleaning Data - Structure
- 7. Transforming the Coffee data
- 8. Fixing Values
- 9. Webscraping
- 10. Merging Data & Databases
- 11. Evaluating ML Algorithms
- 12. ML Models and Auditing with AIF360
- 13. Classification Models
- 14. Clustering
- 15. Kmeans algorithm
- 16. Regression
- 17. More regression
- 18. Task Review and Cross Validation
- 19. Model Optimization
- 20. Model Comparison

^ ^ . . .

[Skip to main content](#)

- 22. Which model should we deploy?

## Assignments

- 1. Assignment 1: Setup, Syllabus, and Review
- 2. Assignment 2: Practicing Python and Accessing Data
- 3. Assignment 3: Exploratory Data Analysis
- 4. Assignment 4: Cleaning Data
- 5. Assignment 5: Constructing Datasets and Using Databases
- 6. Assignment 6: Auditing Algorithms
- 7. Assignment 7: Classification
- 8. Assignment 8: Clustering
- 9. Assignment 9: Regression and Optimization
- 10. Assignment 11: Model Comparison

## Portfolio

- Deepening your knowledge
- Formatting Tips
- Generic Extension Ideas
- Alternatives to Extending Assignments for Level 3
- Process Level 3

## FAQ

- FAQ
- Syllabus and Grading FAQ
- Git and GitHub
- Code Errors

## Resources

- Glossary
- References on Python
- Cheatsheet
- Data Sources
- General Tips and Resources
- How to Study in this class
- Getting Help with Programming
- Terminals and Environments
- Getting Organized for class
- Advice from FA2020 Students
- Advice from FA2021 Students

Welcome to the course manual for CSC310 at URI with Professor Brown.

This class meets TTh 3:30-4:45 in Pastore 259.

[Skip to main content](#)

---

This website will contain the syllabus, class notes, and other reference material for the class.

# Land University of Rhode Island Land Acknowledgment

## Note

The University of Rhode Island land acknowledgment is a statement written by members of the University community in close partnership with members of the Narragansett Tribe. The statement recognizes and pays tribute to the people who lived on and stewarded the land on which the University now resides. The statement seeks to show gratitude and respect to Indigenous people and cultures and build community with the Narragansett Nation and other Native American tribes.

The University of Rhode Island occupies the traditional stomping ground of the Narragansett Nation and the Niantic People. We honor and respect the enduring and continuing relationship between the Indigenous people and this land by teaching and learning more about their history and present-day communities, and by becoming stewards of the land we, too, inhabit.

## Navigating the Sections

The Syllabus section has logistical operations for the course broken down into sections. You can also read straight through by starting in the first one and navigating to the next section using the arrow navigation at the end of the page.

This site is a resource for the course. We do not follow a text book for this course, but all notes from class are posted in the notes section, accessible on the left hand side menu, visible on large screens and in the menu on mobile.

The resources section has links and short posts that provide more context and explanation. Content in this section is for the most part not strictly the material that you'll be graded on, but it is often material that will help you understand and grow as a programmer and data scientist.

## Reading each page

All class notes can be downloaded in multiple formats, including as a notebook. Some pages of the syllabus and resources are also notebooks, if you want to see behind the curtain of how I manage the course information.

## Try it Yourself

Notes will have exercises marked like this

## Question from Class

Questions that are asked in class, but unanswered at that time will be answered in the notes and marked with a box like this. Long answers will be in the main notes

### Further reading

Notes that are mostly links to background and context will be highlighted like this. These are optional, but will mostly help you understand code excerpts they relate to.

### Hint

Both notes and assignment pages will have hints from time to time. Pay attention to these on the notes, they'll typically relate to things that will appear in the assignment.

### Think Ahead

Think ahead boxes will guide you to start thinking about what can go into your portfolio to build on the material at hand.

### Click here!



Special tips will be formatted like this

### Check your Comprehension

Questions to use to check your comprehension will looklike this

## About

### About the topic

Data science exists at the intersection of computer science, statistics, and domain expertise. That means writing programs to access and manipulate data so that it becomes available for analysis using statistical and machine learning techniques is at the core of data science. Data scientists use their data and analytical ability to find and interpret rich data sources; manage large amounts of data despite hardware, software, and bandwidth constraints; merge data sources; ensure consistency of datasets; create visualizations to aid in understanding data; build mathematical models using the data; and present and communicate the data insights/findings.

### About the goals and preparation

This course provides a survey of data science. Topics include data driven programming in Python; data sets, file formats and meta-data; descriptive statistics, data visualization, and foundations of predictive data modeling and machine learning; accessing web data and databases; distributed data management. You will work on weekly programming problems such as accessing data in database and visualize it or build machine learning models of a given data set.

Basic programming skills (CSC201 or CSC211) are a prerequisite to this course. This course is a prerequisite course to

[Skip to main content](#)

a very fast review

of basic programming ideas, since you've already done that before. We will learn how to *use* machine learning algorithms to do data science, but not how to *build* machine learning algorithms, we'll use packages that implement the algorithms for us.

## About the course

This course is designed to make you a better programmer while learning data science. You may be stronger in one of those areas than the other at the beginning, but you should grow in both areas by the end of the semester.

## About this semester

This semester, I will be trying some new ways to update the course to reflect the reality that in a job, a lot of your work may be done with an AI assistant to help. That said, *learning* the basic material still has to happen, you cannot supervise an AI if you do not know what correct looks like.

Each assignment will have specific AI use guidelines that you must follow in addition to general overall course style guide and requirements.

Additionally, the grading that we will use this semester will be new, if something does not make sense, ask! I will never change a policy in a way that could hurt a student who was acting in good faith (if you were trying to game thigns, I may close loopholes in ways that do not benefit you.)

## About this syllabus

This syllabus is a *living* document and accessible from BrightSpace, as a pdf for download directly, and online at [rhodyprog4ds.github.io/BrownFall24/syllabus](https://rhodyprog4ds.github.io/BrownFall24/syllabus). If you choose to download a copy of it, note that it is only a copy. You can get notification of changes from GitHub by “watching” the repository. You can view the date of changes and exactly what changes were made on the Github commits page.

Creating an issue on the repository is also a good way to ask questions about anything in the course it will prompt additions and expand the FAQ section.

## About your instructor

Name: Dr. Sarah Brown Office hours: TBA via zoom, link on GitHub Org Page

Dr. Brown is an Assistant Professor of Computer Science, who does research on how social context changes machine learning. Dr. Brown earned a PhD in Electrical Engineering from Northeastern University, completed a postdoctoral fellowship at University of California Berkeley, and worked as a postdoctoral research associate at Brown University before joining URI. At Brown University, Dr. Brown taught the Data and Society course for the Master's in Data Science Program.

### Important

For assignment or notes specific issues, a comment on the corresponding repository is the best. I cannot help you with code issues from screenshots.

[Skip to main content](#)

The best way to contact me for general questions is e-mail or by dropping into my office hours. Please include [\[CSC310\]](#) or [\[DSP310\]](#) in the subject line of your email along with the topic of your message. This is important, because your messages are important, but I also get a lot of e-mail. Consider these a cheat code to my inbox: I have setup a filter that will flag your e-mail if you use one of those in the subject to ensure that I see it. I rarely check e-mail between 6pm and 9am, on weekends or holidays. You might see me post or send things during these hours, but I will not reliably see emails that arrive during those hours.

## Tools and Resources

We will use a variety of tools to conduct class and to facilitate your programming. You will need a computer with Linux, MacOS, or Windows. It is unlikely that a tablet will be able to do all of the things required in this course. A Chromebook may work, especially with developer tools turned on. Ask Dr. Brown if you need help getting access to an adequate computer.

All of the tools and resources below are either:

- paid for by URI **OR**
- freely available online.

### BrightSpace

This will be the central location from which you can access links to other materials. Any links that are for private discussion among those enrolled in the course will be available only from our course Brightspace site.

### Prismia chat

Our class link for Prismia chat is available on Brightspace. We will use this for chatting and in-class understanding checks.

On Prismia, all students see the instructor's messages, but only the Instructor and TA see student responses.

### Course website

The course manual will have content including the class policies, scheduling, class notes, assignment information, and additional resources. This will be linked from Brightspace and available publicly online at [rhodyprog4ds.github.io/BrownSpring23/](https://rhodyprog4ds.github.io/BrownSpring23/). Links to the course reference text and code documentation will also be included here in the assignments and class notes.

### GitHub

You will need a GitHub Account. If you do not already have one, please [create one](#) by the first day of class. If you have one, but have not used it recently, you may need to update your password and login credentials as the [Authentication rules](#) changed over the summer. In order to use the command line with https, you will need to us the [GitHub CLI](#) or create a [Personal Access Token](#) for each device you use. In order to use the command line with SSH, set up your public key.

# Programming Environment

This a programming course, so you will need a programming environment. In order to complete assignments you need the items listed in the requirements list. The easiest way to meet these requirements is to follow the recommendations below. I will provide instruction assuming that you have followed the recommendations.

## Requirements:

- Python with scientific computing packages (numpy, scipy, jupyter, pandas, seaborn, sklearn)
- Git
- A web browser compatible with [Jupyter Notebooks](#)

### ⚠ Warning

Everything in this class will be tested with the up to date (or otherwise specified) version of Jupyter Notebooks. Google Colab is similar, but not the same, and some things may not work there. It is an okay backup, but should not be your primary work environment.

## Recommendation:

- Install python via [Anaconda](#)
- if you use Windows, install Git with [GitBash](#) ([video instructions](#)).
- if you use MacOS, install Git with the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this by trying to run git from the Terminal the very first time. `git --version`
- if you use Chrome OS, follow these instructions:
  1. Find Linux (Beta) in your settings and turn that on.
  2. Once the download finishes a Linux terminal will open, then enter the commands: sudo apt-get update and sudo apt-get upgrade. These commands will ensure you are up to date.
  3. Install tmux with:

```
sudo apt -t stretch-backports install tmux
```

4. Next you will install nodejs, to do this, use the following commands:

```
curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash  
sudo apt-get install -y nodejs  
sudo apt-get install -y build-essential.
```

5. Next install Anaconda's Python from the website provided by the instructor and use the top download link under the Linux options.
6. You will then see a .sh file in your downloads, move this into your Linux files.
7. Make sure you are in your home directory (something like home/YOURUSERNAME), do this by using the `pwd` command.

[Skip to main content](#)

9. Next you will add Anaconda to your Linux PATH, do this by using the `vim .bashrc` command to enter the .bashrc file, then add the `export PATH=/home/YOURUSERNAME/anaconda3/bin/:$PATH` line. This can be placed at the end of the file.
10. Once that is inserted you may close and save the file, to do this hold escape and type `:x`, then press enter. After doing that you will be returned to the terminal where you will then type the source .bashrc command.
11. Next, use the `jupyter notebook --generate-config` command to generate a Jupyter Notebook.
12. Then just type `jupyter lab` and a Jupyter Notebook should open up.

Optional:

- Text Editor: you may want a text editor outside of the Jupyter environment. Jupyter can edit markdown files (that you'll need for your portfolio), in browser, but it is more common to use a text editor like Atom or Sublime for this purpose.

Video install instructions for Anaconda:

- [Windows](#)
- [Mac](#)

On Mac, to install python via environment, this article may be helpful

- I don't have a video for linux, but it's a little more straight forward.

## Textbook

The text for this class is a reference book and will not be a source of assignments. It will be a helpful reference and you may be directed there for answers to questions or alternate explanations of topics.

Python for Data Science is available free [online](#):

## Zoom (backup and office hours only)

This is where we will meet if for any reason we cannot be in person. You will find the link to class zoom sessions on Brightspace.

URI provides all faculty, staff, and students with a paid Zoom account. It *can* run in your browser or on a mobile device, but you will be able to participate in class best if you download the [Zoom client](#) on your computer. Please [log in](#) and [configure your account](#). Please add a photo of yourself to your account so that we can still see your likeness in some form when your camera is off. You may also wish to use a virtual background and you are welcome to do so.

Class will be interactive, so if you cannot be in a quiet place at class time, headphones with a built in microphone are strongly recommended.

For help, you can access the [instructions provided by IT](#).

---

[1] Too long; didn't read.

In this course there are 5 learning outcomes that I expect you to achieve by the end of the semester. To get there, you'll focus on 15 smaller achievements that will be the basis of your grade. This section will describe how the topics covered, the learning outcomes, and the achievements are covered over time. In the next section, you'll see how these achievements turn into grades.

## Learning Outcomes

By the end of the semester

1. (process) Describe the process of data science, define each phase, and identify standard tools
2. (data) Access and combine data in multiple formats for analysis
3. (exploratory) Perform exploratory data analyses including descriptive statistics and visualization
4. (modeling) Select models for data by applying and evaluating multiple models to a single dataset
5. (communicate) Communicate solutions to problems with data in common industry formats

We will build your skill in the **process** and **communicate** outcomes over the whole semester. The middle three skills will correspond roughly to the content taught for each of the first three portfolio checks.

## Schedule

The course will meet in . Every class will include participatory live coding (instructor types code while explaining, students follow along) instruction and small exercises for you to progress toward level 1 achievements of the new skills introduced in class that day.

Each Assignment will have a deadline posted on the assignment page, typically the same day each week. Portfolio deadlines will be announced at least 2 weeks in advance.

week	topics	skills
1	[admin, python review]	process
2	Loading data, Python review	[access, prepare, summarize]
3	Exploratory Data Analysis	[summarize, visualize]
4	Data Cleaning	[prepare, summarize, visualize]
5	Databases, Merging DataFrames	[access, construct, summarize]
6	Modeling, classification performance metrics, cross validation	[evaluate]
7	Naive Bayes, decision trees	[classification, evaluate]
8	Regression	[regression, evaluate]
9	Clustering	[clustering, evaluate]
10	SVM, parameter tuning	[optimize, tools]
11	KNN, Model comparison	[compare, tools]
12	Text Analysis	[unstructured]

week	topics	skills
13	Images Analysis	[unstructured, tools]
14	Deep Learning	[tools, compare]

## Achievement Definitions

The table below describes how your work will be assessed to earn each achievement. The keyword for each skill is a short name that will be used to refer to skills throughout the course materials; the full description of the skill is in this table.

	skill	Level 1	Level 2	Level 3
keyword				
<b>python</b>	pythonic code writing	python code that mostly runs, occasional pep8 adherence	python code that reliably runs, frequent pep8 adherence	reliable, efficient, pythonic code that consistently adheres to pep8
<b>process</b>	describe data science as a process	Identify basic components of data science	Describe and define each stage of the data science process	Compare different ways that data science can facilitate decision making
<b>access</b>	access data in multiple formats	load data from at least one format; identify the most common data formats	Load data for processing from the most common formats; Compare and contrast most common formats	access data from both common and uncommon formats and identify best practices for formats in different contexts
<b>construct</b>	construct datasets from multiple sources	identify what should happen to merge datasets or when they can be merged	apply basic merges	merge data that is not automatically aligned
<b>summarize</b>	Summarize and describe data	Describe the shape and structure of a dataset in basic terms	compute summary standard statistics of a whole dataset and grouped data	Compute and interpret various summary statistics of subsets of data
<b>visualize</b>	Visualize data	identify plot types, generate basic plots from pandas	generate multiple plot types with complete labeling with pandas and seaborn	generate complex plots with pandas and plotting libraries and customize with matplotlib or additional parameters
<b>prepare</b>	prepare data for analysis	identify if data is or is not ready for analysis, potential problems with data	apply data reshaping, cleaning, and filtering as directed	apply data reshaping, cleaning, and filtering manipulations reliably and correctly by assessing data as received
<b>evaluate</b>	Evaluate model performance	Explain and compute basic performance metrics for different data science tasks	Apply and interpret basic model evaluation metrics to a held out test set	Evaluate a model with multiple metrics and cross validation
<b>classification</b>	Apply classification	identify and describe what classification is, apply pre-fit classification models	fit, apply, and interpret preselected classification model to a dataset	fit and apply classification models and select appropriate classification models for different contexts
<b>regression</b>	Apply Regression	identify what data that can be used for regression looks like	fit and interpret linear regression models	fit and explain regularized or nonlinear regression
<b>clustering</b>	Clustering	describe what clustering is	apply basic clustering	apply multiple clustering techniques, and interpret results
<b>optimize</b>	Optimize model parameters	Identify when model parameters need to be	Optimize basic model parameters such as	Select optimal parameters based of mutiple quantitave criteria and automate parameter tuning

[Skip to main content](#)

	skill	Level 1	Level 2	Level 3
keyword				
<b>compare</b>	compare models	Qualitatively compare model classes	Compare model classes in specific terms and fit models in terms of traditional model performance metrics	Evaluate tradeoffs between different model comparison types
		Identify options for representing text and categorical data in many contexts	Apply at least one representation to transform unstructured or inappropriate data for model fitting or summarizing	apply transformations in different contexts OR compare and contrast multiple representations a single type of data in terms of model performance
		Solve well structured fully specified problems with a single tool pipeline	Solve well-structured, open-ended problems, apply common structure to learn new features of standard tools	Independently scope and solve realistic data science problems OR independently learn related tools and describe strengths and weaknesses of common tools
<b>representation</b>	Choose representations and transform data			
<b>workflow</b>	use industry standard data science tools and workflows to solve data science problems			

## Assignments and Skills

Using the keywords from the table above, this table shows which assignments you will be able to demonstrate which skills and the total number of assignments that assess each skill. This is the number of opportunities you have to earn Level 2 and still preserve 2 chances to earn Level 3 for each skill.

keyword	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	# Assignments
<b>python</b>	1	1	0	1	1	0	0	0	0	0	0	0	0	4
<b>process</b>	1	0	0	0	0	1	1	1	1	1	1	0	0	7
<b>access</b>	0	1	1	1	1	0	0	0	0	0	0	0	0	4
<b>construct</b>	0	0	0	0	1	0	1	1	0	0	0	0	0	3
<b>summarize</b>	0	0	1	1	1	1	1	1	1	1	1	1	1	11
<b>visualize</b>	0	0	1	1	0	1	1	1	1	1	1	1	1	10
<b>prepare</b>	0	0	0	1	1	0	0	0	0	0	0	0	0	2
<b>evaluate</b>	0	0	0	0	0	1	1	1	0	1	1	0	0	5
<b>classification</b>	0	0	0	0	0	0	1	0	0	1	0	0	0	2
<b>regression</b>	0	0	0	0	0	0	0	1	0	0	1	0	0	2
<b>clustering</b>	0	0	0	0	0	0	0	0	1	0	1	0	0	2
<b>optimize</b>	0	0	0	0	0	0	0	0	0	1	1	0	0	2
<b>compare</b>	0	0	0	0	0	0	0	0	0	0	1	0	1	2
<b>representation</b>	0	0	0	0	0	0	0	0	0	0	0	1	1	2
<b>workflow</b>	0	0	0	0	0	0	0	0	0	1	1	1	1	4

[Skip to main content](#)

### **⚠ Warning**

**process** achievements are accumulated a little slower; details will follow.

## Extensions

### **⚠ Warning**

this rolling deadline is new for Fall 2024 and aims to let students distribute work in a better way for yourself. After A2 feedback is posted, I will give more explanation about how to do this, in concrete terms.

There are no extensions applicable to assignment 1, but starting after assignment 2's feedback you can start working on level 3 achievements. You can add on and extend each analysis, once you have earned level 2 for a skill to earn level 3. You can also add new analyses that instead combine different sets of skills.

Extensions will all be graded by Dr. Brown (and most assignments will be graded by the TA Surbhi). You will make separate PRs for your attempts at level 3 from level 2.

While assignments have fixed grades, you can submit extensions as you complete them. I recommend planning to work on them consistently throughout the semester.

### **⚠ Warning**

In previous semesters, there were checklists, but they are removed because they distracted students from learning the important things

## Grading

This section of the syllabus describes the principles and mechanics of the grading for the course. This course will be graded on a basis of a set of *skills* (described in detail the next section of the syllabus). This is in contrast to more common grading on a basis of points earned through assignments.

## Principles of Grading

Learning happens through practice and feedback. My goal as a teacher is for you to learn. The grading in this course is based on your learning of the material, rather than your completion of the activities that are assigned.

This course is designed to encourage you to work steadily at learning the material and demonstrating your new knowledge. There are no single points of failure, where you lose points that cannot be recovered. Also, you cannot cram anything one time and then forget it. The material will build and you have to demonstrate that you retained things.

- Earning a C in this class means you have a general understanding of Data Science and could participate in a basic conversation about all of the topics we cover. I expect everyone to reach this level.
- Earning a B means that you could solve simple data science problems on your own and complete parts of more complex accessible goal, it

does not require you to get anything on the first try or to explore topics on your own. I expect most students to reach this level.

- Earning an A means that you could solve moderately complex problems independently and discuss the quality of others' data science solutions. This class will be challenging, it requires you to explore topics a little deeper than we cover them in class, but unlike typical grading it does not require all of your assignments to be near perfect.

Grading this way also is more amenable to the fact that there are correct and incorrect ways to do things, but there is not always a single correct answer to a realistic data science problem. Your work will be assessed on whether or not it demonstrates your learning of the targeted skills. You will also receive feedback on how to improve.

## How it works

### ⚠ Warning

This is going to change; you will get a notification when it is final

There are 15 skills that you will be graded on in this course. While learning these skills, you will work through a progression of learning, first getting the basic idea, then applying it, then exploring advanced usage. You will have to demonstrate each skill area on 3 separate occasions to get level 3 credit for it. Your grade will be based on earning 45 achievements that are organized into 15 skill groups with 3 levels for each.

These map onto letter grades roughly as follows:

- If you achieve level 1 in all of the skills, you will earn at least a C in the course.
- To earn a B, you must earn all of the level 1 and level 2 achievements.
- To earn an A, you must earn all of the achievements.

You will have at least three opportunities to earn every level 2 achievement. You will have at least two opportunities to earn every level 3 achievement.

Each level of achievement corresponds to a phase in your learning of the skill:

- To earn level 1 achievements, you will need to demonstrate basic awareness of the required concepts and know approximately what to do, but you may need specific instructions of which things to do or to look up examples to modify every step of the way. You can learn level 1 in any assignment (even without completing it completely correct) or in office hours.
- To earn level 2 achievements you will need to demonstrate understanding of the concepts and the ability to apply them with instruction after earning the level 1 achievement for that skill. Weekly assignments will guide you to level 2, if you complete it well.
- To earn level 3 achievements you will be required to consistently execute each skill and demonstrate deep understanding of the course material, after achieving level 2 in that skill. This will happen mostly extending previous assignments in your portfolio.

For each skill these are defined in the [Achievement Definition Table](#)

This is a  
student v  
up below  
level 3s a  
any work

## Participation

While attending synchronous class sessions, there will be understanding checks and in class exercises. Completing in class exercises and correctly answering questions in class is approximately the level of understanding required for level 1.

We will not directly add these to your grade, but you can always visit office hours to earn level 1, so that your assignment can go straight to 2 in most skills.

## Assignments

For your learning to progress and earn level 2 achievements, you must practice with the skills outside of class time. You will submit all of your assignments in your portfolio repository. Sometimes you will need to sync your repo to get templates for the assignment and some will be open. All will be posted on this site.

Assignments will each evaluate certain skills. After your assignment is reviewed, you will get qualitative feedback on your work, and an assessment of your demonstration of the targeted skills. Your feedback will recap all of your earned achievements to that point, not only what was earned in that assignment.

You can revise assignments if you do not earn achievements by also adding reflections to them while you edit, but since each skill is available in multiple assignments you do not have to.

You can revise what you submitted and resubmit it, with reflections and explanation of what you were confused about, what you tried initially, how you eventually figured it out, and explains the correct answer.

## Extensions

### Warning

the logistics of this are changing, but tbd, will update soon

To earn level 3 achievements, you will extend your prior work. Starting with assignment 2, there will be extension ideas in the assignment.

## TLDR

You *could* earn a C through oral exams in office hours alone. To earn a B, you must complete assignments. To earn an A you must complete assignments and add extensions that demonstrate deeper understanding (tips will be provided).

## Detailed mechanics

The table below shows the minimum number of skills at each level to earn each letter grade.

### Level 3   Level 2   Level 1

letter grade	Level 3	Level 2	Level 1
A	15	15	15
A-	10	15	15
B+	5	15	15
B	0	15	15
B-	0	10	15
C+	0	5	15
C	0	0	15
C-	0	0	10
D+	0	0	5
D	0	0	3

For example, if you achieve level 2 on all of the skills and level 3 on 7 skills, that will be a B+.

If you achieve level 3 on 14 of the skills, but only level 1 on one of the skills, that will be a B-, because the minimum number of level 2 achievements for a B is 15. In this scenario the total number of achievements is 14 at level 3, 14 at level 2 and 15 at level 3, because you have to earn achievements within a skill in sequence.

The letter grade can be computed as follows

#### ⚠ Important

this will be revealed after assignment 1

## Grading Examples

### Getting an A Without Perfection

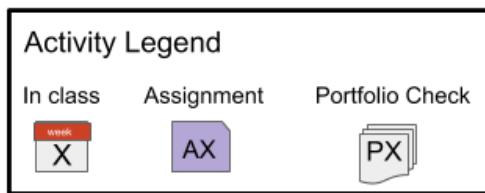
#### ⚠ Warning

achievements are no longer awarded in class in fall 24; images to be updated and portfolio is rolling instead of discrete check points.

# Map to an A

## How Achievements were earned

	Level 1	Level 2	Level 3
python	A1	A3	P1
process	A1	P1	P2
access	week 2	A2	P1
construct	week 5	A5	P1
summarize	week 3	A3	P1
visualize	week 3	A3	P2
prepare	week 4	A5	P2
classification	A10	P2	P3
regression	week 8	A11	P2
clustering	week 9	A9	P3
evaluate	week 7	A11	P3
optimize	week 10	A11	P4
compare	week 11	A13	P3
unstructured	week 12	A13	P4
tools	week 11	A13	P3



## Other Activities

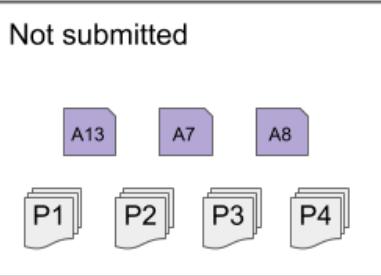
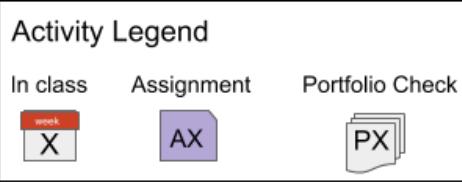
- Attended, but did not understand
- Submitted, but incorrect
- Missed class
- Not submitted
- Submitted, but incorrect
- Not submitted
- Not submitted
- Attended, but all level 1 complete
- Attended, but all level 1 complete

In this example the student made several mistakes, but still earned an A. This is the advantage to this grading scheme. For the `python`, `process`, and `classification` skills, the level 1 achievements were earned on assignments, not in class. For the `process` and `classification` skills, the level 2 achievements were not earned on assignments, only on portfolio checks, but they were earned on the first portfolio of those skills, so the level 3 achievements were earned on the second portfolio check for that skill. This student's fourth portfolio only demonstrated two skills: `optimize` and `unstructured`. It included only 1 analysis, a text analysis with optimizing the parameters of the model. Assignments 4 and 7 were both submitted, but didn't earn any achievements, the student got feedback though, that they were able to apply in later assignments to earn the achievements. The student missed class week 6 and chose to not submit assignment 6 and use week 7 to catch up. The student had too much work in another class and chose to skip assignment 8. The student tried assignment 12, but didn't finish it on time, so it was not graded, but the student visited office hours to understand and be sure to earn the level 2 `unstructured` achievement on assignment 13.

## Getting a B with minimal work

# Map to a B easily

	Level 1	Level 2	Level 3
python	week 1	A3	
process	week 1	A1	
access	week 2	A2	
construct	week 5	A5	
summarize	week 3	A3	
visualize	week 3	A3	
prepare	week 4	A4	
classification	week 10	A6	
regression	week 8	A11	
clustering	week 9	A9	
evaluate	week 7	A10	
optimize	week 10	A10	
compare	week 11	A11	
unstructured	week 12	A12	
tools	week 11	A12	

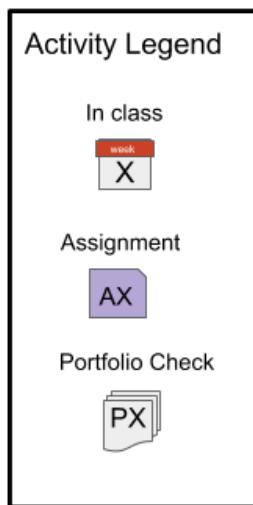


In this example, the student earned all level 1 achievements in class and all level 2 on assignments. This student was content with getting a B and chose to not submit a portfolio.

Getting a B while having trouble

# Map to a B, having trouble

	Level 1	Level 2	Level 3
python	A1	P1	
process	A1	P2	
access	A2	P1	
construct	A5	P1	
summarize	A3	P1	
visualize	A3	P2	
prepare	A5	P2	
classification	A10	P3	
regression	A11	P2	
clustering	A9	P3	
evaluate	A11	P3	
optimize	A11	P4	
compare	A13	P3	
unstructured	A13	P4	
tools	A13	P3	



In this example, the student struggled to understand in class and on assignments. Assignments were submitted that showed some understanding, but all had some serious mistakes, so only level 1 achievements were earned from assignments. The student wanted to get a B and worked hard to get the level 2 achievements on the portfolio checks.

## Grading Policies

### Attendance

Attendance and active participation is expected. You earn level 1 achievements in class and all class sessions are active learning.

If you miss class, you can make it up by reading the posted notes and the prismia transcript. Best practice is to download them as a notebook and run them to make sure you understand each step. If you miss both class sessions in a week, the level one achievements can be made up through annotation or in your assignment.

Absences do not require notification.

### Assignment Deadlines and Late Work

**Late assignments will not be graded.** Extensions will not be granted for assignments. Every skill will be assessed through more than one assignment, so missing assignments occasionally will not necessarily impact your grade. If you do not submit any assignments that cover a given skill, you may earn the level 2 achievement in that skill through a portfolio check, but you

[Skip to main content](#)

If you submit work that is **not complete**, it will be assessed and receive feedback. Submitting pseudocode or code with errors and comments about what you have tried could even be enough to earn a level 1 achievement. Assignments cover multiple skills, so partially completing the assignment may earn level 2 for one, but not all. Submitting *something* even if it is not perfect is important to keeping conversation open and getting feedback and help continuously.

### Important

If you have a serious issue during the semester, that prevents you from submitting an assignment, email Dr. Brown to make a plan. Extensions will still not be granted because they do not help you in the long run, instead an alternate plan of how to earn the target grade.

## Portfolio Deadlines and Extensions

Building your Data Science Portfolio should be an ongoing process, where you commit work to your portfolio frequently. If something comes up and you cannot finish all that you would like assessed by the deadline, open an [Extension Request](#) issue on your repository at least **24 hours** before the deadline.

In this issue, include:

1. A proposed new deadline
2. What additional work you plan to add
3. Why the extension is important to your learning
4. Why the extension will not hinder your ability to complete the next assignments and portfolio check on time.
5. (if less than 24 hours before the deadline) why you need an emergency request

### Important

Your request should not include a reason why you are asking, unless you are asking for an emergency extension. Emergency requests can be submitted at any time, even after the deadline.

This request should be no more than 7 sentences.

Portfolio due dates will be announced well in advance and prompts for it will be released weekly. You should spend some time working on it each week, applying what you've learned so far, from the feedback on previous assignments.

## Academic Dishonesty

All work must represent your own understanding of both the data science practices and the related programming concepts. Submitting code or prose that was generated by a generative model or another person is not allowed.

If you are found to have submitted work that does not constitute your own work, the following penalties apply:

- in a portfolio, all achievements attempted in the dishonest component are permanently ineligible.
- in an assignment the level three achievements for the skills of focus in the assignment are ineligible, and the relevant level

For example, if you violate the academic honesty policy in assignment 4, Prepare level 3 becomes ineligible and you must meet the requirements for prepare level 3 in a portfolio in order to earn prepare level 2.

If you violate academic honesty policy in portfolio 1 while attempting level 3 at Python, access, prepare, summarize and visualize and process level 2, then your maximum grade becomes a B+, because level 3 in all five of those skills becomes ineligible.

## Regrading

1. Add comments:
  - For general questions, post on the conversation tab of your Feedback PR with your request.
  - For specific questions, reply to a specific comment.
2. Re-request a review from Dr. Brown on your Feedback Pull request.

If you think we missed where you did something, add a comment on that line to help us find it (on the code tab of the PR, click the plus (+) next to the line) and then post on the conversation tab with an overview of what you're requesting and tag @brownsarahm

## Course Style Guide

Following a style guide is a common requirement in companies to make it so that code written by different people stays easy to read for everyone. Consistent style also makes it easier to onboard new developers join a project and contribute faster.

The following style guide serves as practice for you following a style guide, makes your work easier to read for grading purposes, and holds you accountable to learning deeply and demonstrating that you have learned well.

## Hard Requirements

### Warning

All work must adhere to these requirements or it may receive no feedback or credit. Minor misses may receive warnings, but if submitted work does not appear to represent a good faith effort at adhering to this style guide, the only comment will be, "Follow the style guide on the next assignment"

1. All code must be submitted in a notebook file (.ipynb or myst)
2. Code must run or have explicit questions and comments about what was done about the errors
3. Python comments (`# comment text`) inside code cells should **only** be used to explain complex code that is not explained in the course notes. Using such code requires a citation for the source.
4. Each code cell should exactly one conceptually complete step in terms of the analysis.
5. Nearly all code cells should have output in some form
6. Every code cell must be motivated by text in markdown before it
7. Every code cell's output must be interpreted in a markdown cell (may be combined in one cell that explains the above

8. the `print` function can only be used when it improves the readability over using jupyter's display, must be justified
9. No deprecated or dangerous code constructs without justification
10. All assignment questions must be answered in markdown cells
11. Notebook files may not have extraneous metadata in them
12. Alternative libraries not taught in class can only be used when attempting level 3.

## Additional Style

### Important

Mistakes on these will get detailed feedback once and a “see previous feedback” a second time before the whole assignment receives no feedback.

1. Code should adhere to PEP8
2. Markdown syntax should be used to enhance the readability of the text (eg not all headings, bullets where they make sense)
3. Best practices that are highlighted in class should be followed (this list will expand over the semester)

## Support

### Warning

URI changed some links and this page is not yet up to date

## Academic Enhancement Center

Academic Enhancement Center (for undergraduate courses): Located in Roosevelt Hall, the AEC offers free face-to-face and web-based services to undergraduate students seeking academic support. Peer tutoring is available for STEM-related courses by appointment online and in-person. The Writing Center offers peer tutoring focused on supporting undergraduate writers at any stage of a writing assignment. The UCS160 course and academic skills consultations offer students strategies and activities aimed at improving their studying and test-taking skills. Complete details about each of these programs, up-to-date schedules, contact information and self-service study resources are all available on the [AEC website](#).

- **STEM Tutoring** helps students navigate 100 and 200 level math, chemistry, physics, biology, and other select STEM courses. The STEM Tutoring program offers free online and limited in-person peer-tutoring this fall. Undergraduates in introductory STEM courses have a variety of small group times to choose from and can select occasional or weekly appointments. The TutorTrac application is available through [URI Microsoft 365 single sign-on](#) and by visiting [aec.uri.edu](#). More detailed information and instructions can be found on the [AEC tutoring page](#).
- **Academic Skills Development** resources helps students plan work, manage time, and study more effectively. In Fall 2020, all Academic Skills and Strategies programming are offered both online and in-person. UCS160: Success in Higher Education is a one-credit course on developing a more effective approach to studying. Academic Consultations are 30-

issues. Study Your Way to Success is a self-guided web portal connecting students to tips and strategies on studying and time management related topics. For more information on these programs, visit the [Academic Skills Page](#) or contact Dr. Hayes directly at [davidhayes@uri.edu](mailto:davidhayes@uri.edu).

- The **Undergraduate Writing Center** provides free writing support to students in any class, at any stage of the writing process: from understanding an assignment and brainstorming ideas, to developing, organizing, and revising a draft. Fall 2020 services are offered through two online options: 1) real-time synchronous appointments with a peer consultant (25- and 50-minute slots, available Sunday - Friday), and 2) written asynchronous consultations with a 24-hour turn-around response time (available Monday - Friday). Synchronous appointments are video-based, with audio, chat, document-sharing, and live captioning capabilities, to meet a range of accessibility needs. View the synchronous and asynchronous schedules and book online, visit [uri.mywconline.com](http://uri.mywconline.com).

## General URI Policies

### Warning

URI changed some links and this page is not yet up to date

### Anti-Bias Statement:

We respect the rights and dignity of each individual and group. We reject prejudice and intolerance, and we work to understand differences. We believe that equity and inclusion are critical components for campus community members to thrive. If you are a target or a witness of a bias incident, you are encouraged to submit a report to the URI Bias Response Team at [www.uri.edu/brt](http://www.uri.edu/brt). There you will also find people and resources to help.

### Mental Health and Wellness

We understand that college comes with challenges and stress associated with your courses, job/family responsibilities and personal life. URI offers students a range of services to support your [mental health and wellbeing](#), including the URI Counseling Center, MySSP (Student Support Program) App, the Wellness Resource Center, and Well-being Coaching.

### Disability Services for Students Statement:

Your access in this course is important. Please send me your Disability Services for Students (DSS) accommodation letter early in the semester so that we have adequate time to discuss and arrange your approved academic accommodations. If you have not yet established services through DSS, please contact them to engage in a confidential conversation about the process for requesting reasonable accommodations in the classroom. DSS can be reached by calling: 401-874-2098, visiting: [web.uri.edu/disability](http://web.uri.edu/disability), or emailing: [dss@etal.uri.edu](mailto:dss@etal.uri.edu). We are available to meet with students enrolled in Kingston as well as Providence courses.

### Academic Honesty

Students are expected to be honest in all academic work. A student's name on any written work, quiz or exam shall be

[Skip to main content](#)

k should be stated

in the student's own words, properly attributed to its source. Students have an obligation to know how to quote, paraphrase, summarize, cite and reference the work of others with integrity. The following are examples of academic dishonesty.

- Using material, directly or paraphrasing, from published sources (print or electronic) without appropriate citation
- Claiming disproportionate credit for work not done independently
- Unauthorized possession or access to exams
- Unauthorized communication during exams
- Unauthorized use of another's work or preparing work for another student
- Taking an exam for another student
- Altering or attempting to alter grades
- The use of notes or electronic devices to gain an unauthorized advantage during exams
- Fabricating or falsifying facts, data or references directly or indirectly through the use of generative AI
- Facilitating or aiding another's academic dishonesty
- Submitting the same paper for more than one course without prior approval from the instructors

## Communications & Office Hours

### Warning

Due to Dept Seminar Office hours on 11/10 will be at 5pm instead of 4pm.

## Announcements

Announcements will be made via GitHub Release. You can view them online in the releases page or you can get notifications by watching the repository, choosing "Releases" under custom see GitHub docs for instructions with screenshots. You can choose GitHub only or e-mail notificaiton from the notification settings page

## Help Hours

Day	Time	Location	Host
Friday	4-6pm	Zoom	Dr. Brown
TBA	TBA	134 Tyler	Dr. Brown
Tuesday	11am-1pm	Zoom	Surbhi
Wednesday	1-3pm	Zoom	Surbhi

Zoom links are on the course organization page of GitHub

## To reach out, By usage

We have several different ways to communicate in this course. This section summarizes them

[Skip to main content](#)

	usage	platform	area	note
	in class	prismia	chat	outside of class time this is not monitored closely
	any time	prismia	download transcript	use after class to get preliminary notes eg if you miss a class
private questions to your assignment		github	issue on assignment repo	eg bugs in your code"
for general questions that can help others		github	issue on course website	eg what the instructions of an assignment mean or questions about the syllabus
to share resources or ask general questions in a semi-private forum		github	discussion on community repo	include links in your portfolio
matters that don't fit into another category	e-mail	to brownsarahm@uri.edu		remember to include '[CSC310]' or '[DSP310]' (note `verbatim` no space)

### Note

e-mail is last because it's not collaborative; other platforms allow us (Professor + TA) to collaborate on who responds to things more easily.

## Tips

### For assignment help

- **send in advance, leave time for a response** I check e-mail/github a small number of times per day, during work hours, almost exclusively. You might see me post to this site, post to BrightSpace, or comment on your assignments outside of my normal working hours, but I will not reliably see emails that arrive during those hours. This means that it is important to start assignments early.

### Using issues

- use issues for content directly related to assignments. If you push your code to the repository and then open an issue, I can see your code and your question at the same time and download it to run it if I need to debug it
- use issues for questions about this syllabus or class notes. At the top right there's a GitHub logo  that allows you to open a issue (for a question) or suggest an edit (eg if you think there's a typo or you find an additional helpful resource related to something)

### For E-mail

- use e-mail for general inquiries or notifications
- Please include **[CSC310]** or **[DSP310]** in the subject line of your email along with the topic of your message. This is important, because your messages are important, but I also get a lot of e-mail. Consider these a cheat code to my inbox: I have setup a filter that will flag your e-mail if you use one of those in the subject to ensure that I see it.

 No

Wh  
not

# 1. Welcome & What is Data Science

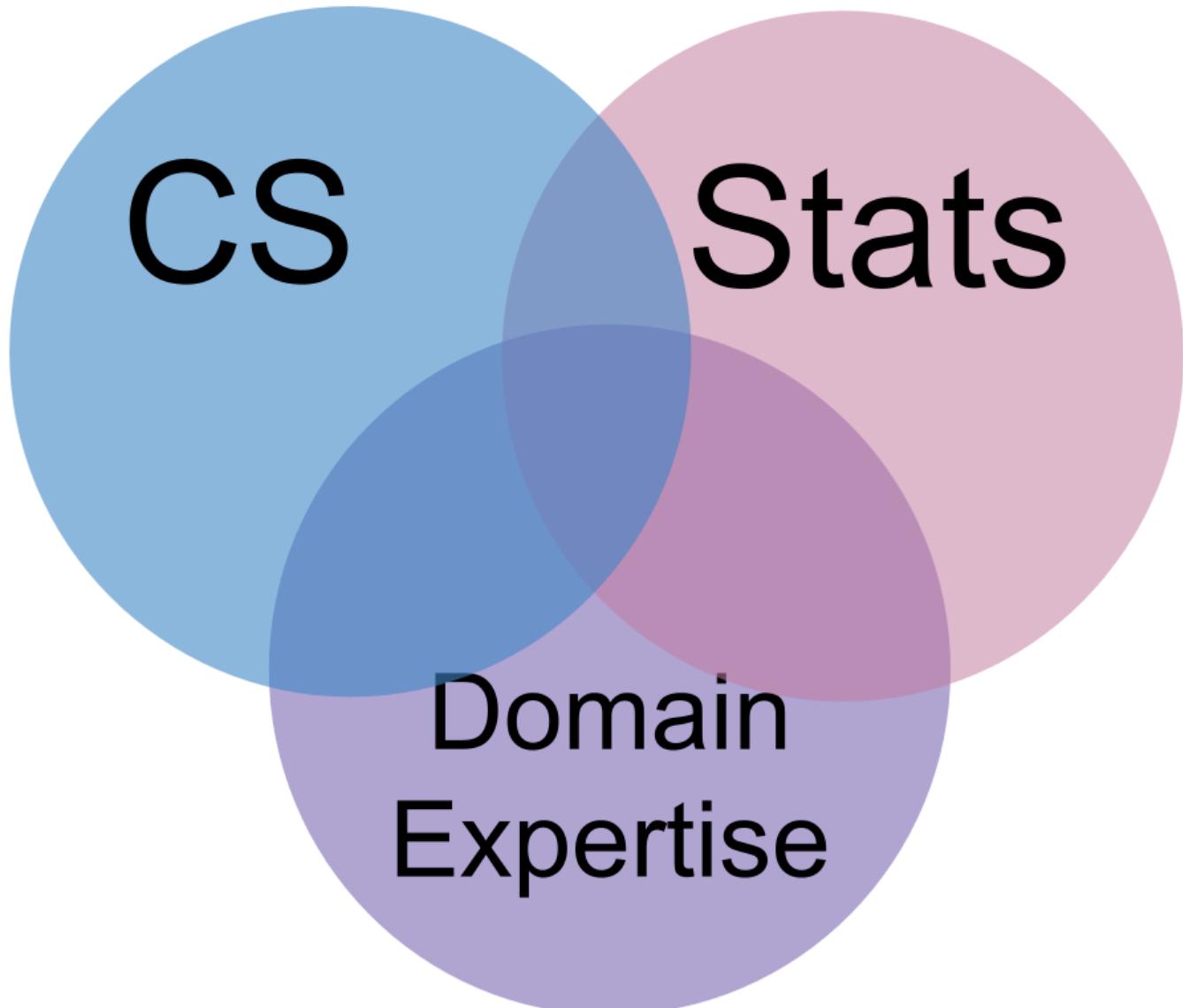
## 1.1. Prismia Chat

We will use these to monitor your participation in class and to gather information. Features:

- instructor only
- reply to you directly
- share responses for all

## 1.2. What is Data Science?

Data Science is the combination of

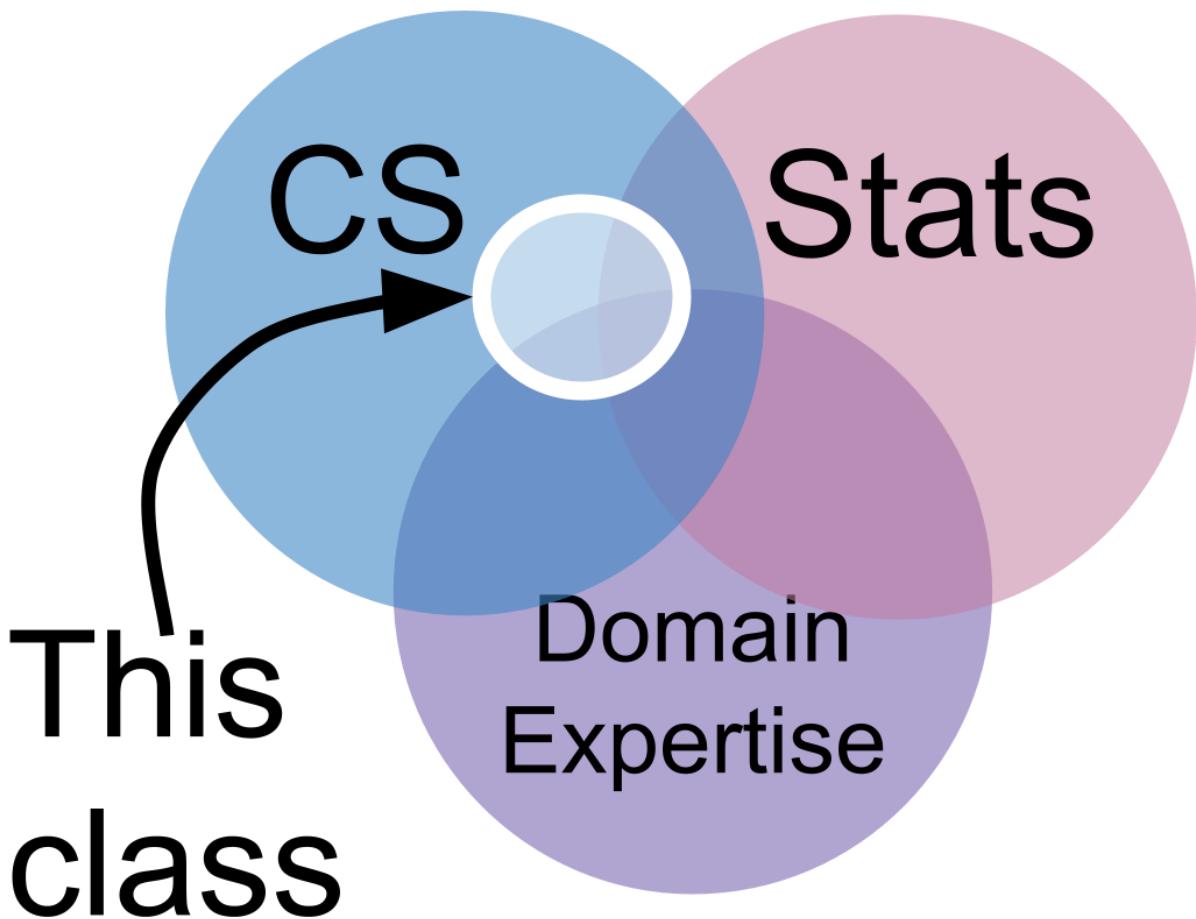


**statistics** is the type of math we use to make sense of data. Formally, a statistic is just a function of data.

**computer science** is so that we can manipulate visualize and automate the inferences we make.

**domain expertise** helps us have the intuition to know if what we did worked right. A statistic must be interpreted in context; the relevant context determines what they mean and which are valid. The context will say whether automating something is safe or not, it can help us tell whether our code actually worked right or not.

### 1.2.1. In this class,

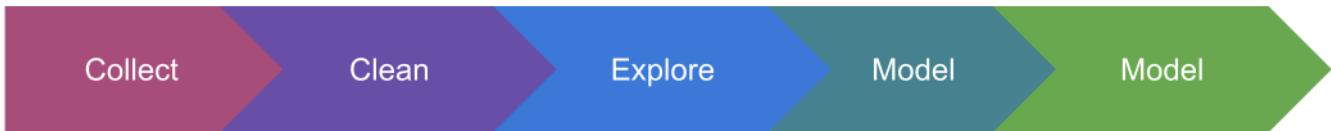


We'll focus on the programming as our main means of studying data science, but we will use bits of the other parts. In particular, you're encouraged to choose datasets that you have domain expertise about, or that you want to learn about.

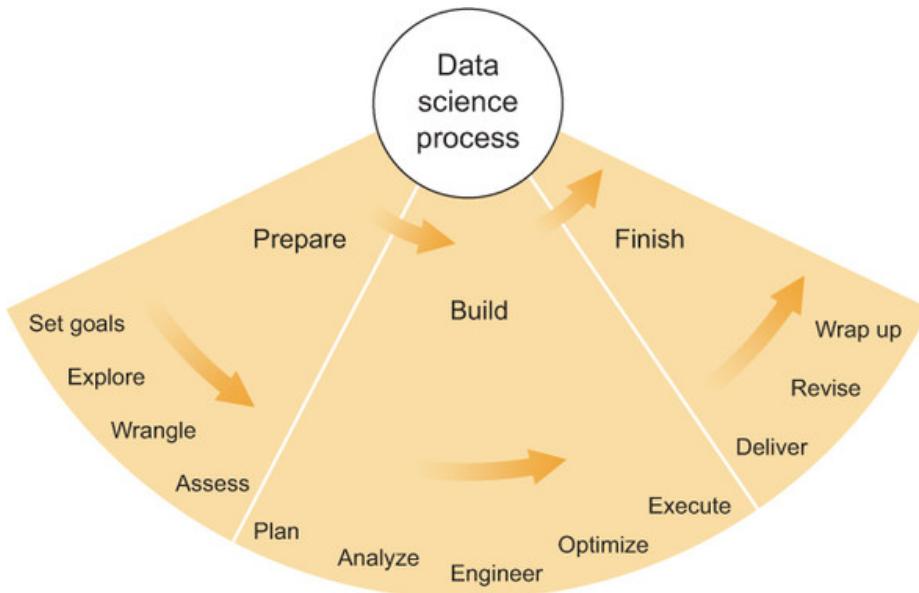
But there are many definitions. We'll use this one, but you may come across others.

### 1.2.2. How does data science happen?

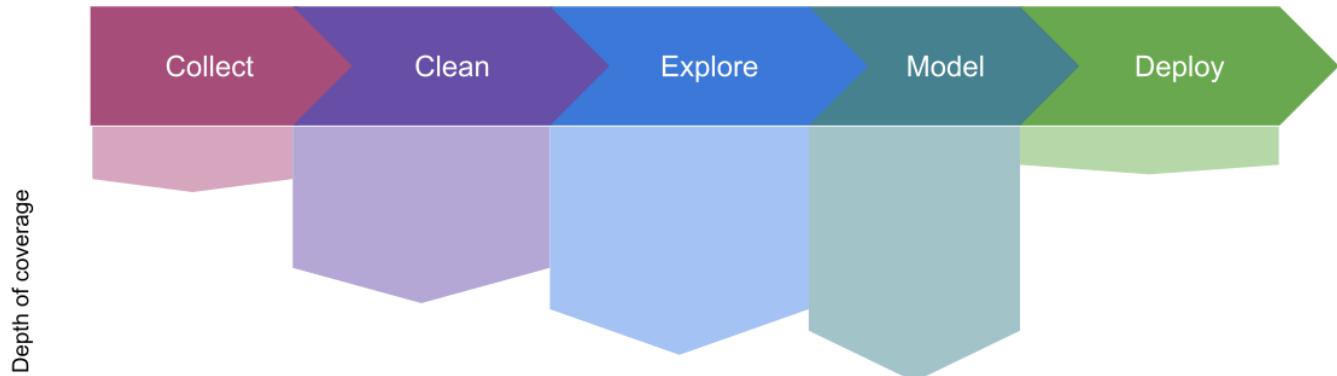
The most common way to think about what doing data science means is to think of this pipeline. It is in the perspective of the data, these are all of the things that happen to the data.



Another way to think about it

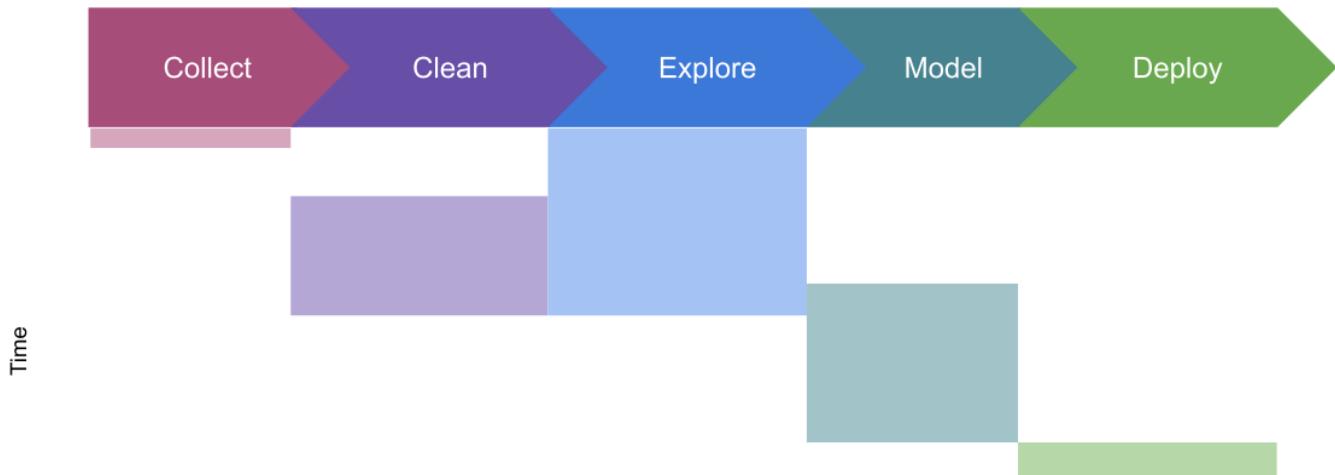


### 1.2.3. how we'll cover Data Science, in depth



- *collect*: Discuss only a little; Minimal programming involved
- *clean*: Cover the main programming techniques; Some requires domain knowledge beyond scope of course
- *explore*: Cover the main programming techniques; Some requires domain knowledge beyond scope of course
- *model*: Cover the main programming, basic idea of models; How to use models, not how learning algorithms work

#### 1.2.4. how we'll cover it in, time



We'll cover exploratory data analysis before cleaning because those tools will help us check how we've cleaned the data.

### 1.3. How this class will work

#### Participatory Live Coding

What is a topic you want to use data to learn about?

#### Note

Here I changed this title from the title of the notebook, to a subsection but the rest of this is what was actually done in class, with explanation.

### 1.4. Intro to Jupyter Notebooks

### 1.5. Programming for Data Science vs other Programming

The audience is different, so the form is different.

In Data Science our product is more often a report than a program.

Sometimes there will be points in the notes that were not made in class due to time or in response questions that came at the end of class.

Also, in data science we are *using code* to interact with data, instead of having a plan in advance

So programming for data science is more like *writing* it has a narrative flow and is made to be seen more than some other

[Skip to main content](#)

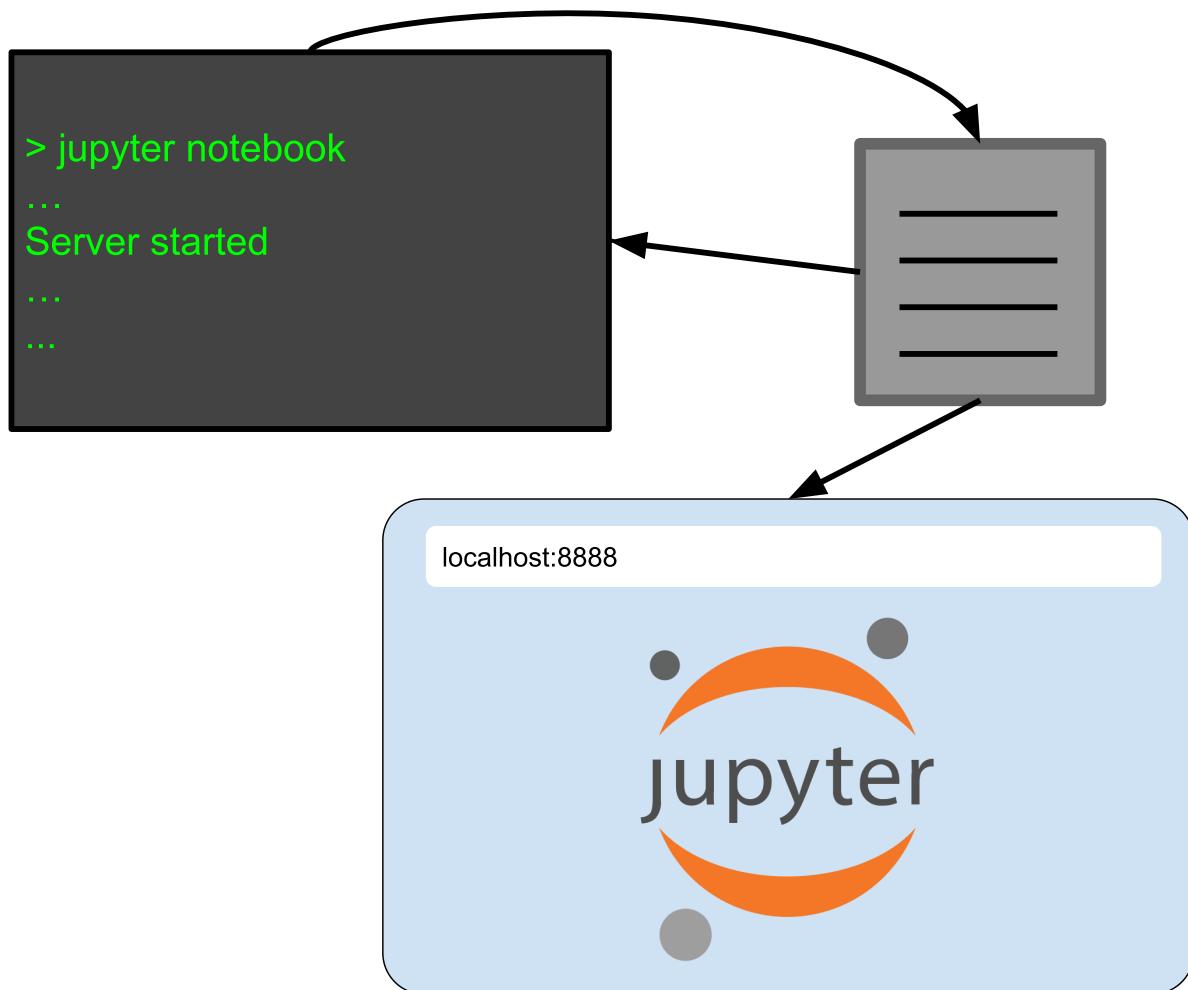
## 1.6. Jupyter Lab and Jupyter notebooks

Launch a `jupyter lab` server:

- on Windows, use anaconda terminal
- on Mac/Linux, use terminal
- `cd path/to/where/you/save/notes`
- enter `jupyter lab`

### 1.6.1. What just happened?

- launched a local web server
- opened a new browser tab pointed to it



## 1.6.2. A jupyter notebook tour

A Jupyter notebook has two modes. When you first open, it is in command mode. It says the mode in the bottom right of the screen. Each box is a cell, the highlighted cell is gray when in command mode.

When you press a key in command mode it works like a shortcut. For example `p` shows the command search menu.

If you press `enter` (or `return`) or click on the highlighted cell, which is the boxes we can type in, it changes to edit mode.

There are two type of cells that we will used: code and markdown. You can change that in command mode with `y` for code and `m` for markdown or on the cell type menu at the top of the notebook.

This is a markdown cell

- we can make
  - itemized lists of
  - bullet points
1. and we can make numbered
  2. lists, and not have to worry
  3. about renumbering them
  4. if we add a step in the middle later

```
3+4
```

```
7
```

the output here is the value returned by the python interpreter for the last line of the cell

We can set variables

```
name = 'Sarah'
```

The notebook displays nothing when we do an assignment, because it returns nothing

we can put a variable there to see it

```
name
```

```
'Sarah'
```

```
name_list = ['Sarah', 'Adam', 'Alex']
```

```
name_list
```

```
['Sarah', 'Adam', 'Alex']
```

### Note

built in functions turn green in jupyter

```
print(name_list)
```

```
['Sarah', 'Adam', 'Alex']
```

Common command mode actions:

- m: switch cell to markdown
- y: switch cell to code
- a: add a cell above
- b: add a cell below
- c: copy cell
- v: paste the cell
- 0 + 0: restart kernel
- p: command menu

use enter/return to get to edit mode

## 1.7. Getting Help in Jupyter

Getting help is important in programming

When your cursor is inside the  of a function if you hold the shift key and press tab it will open a popup with information. If you press tab twice, it gets bigger and three times will make a popup window.

Python has a `print` function and we can use the help in jupyter to learn about how to use it in different ways.

```
year = '2020'
```

```
print(name, year)
```

```
Sarah 2020
```



### Tip

This is an added tip for the notes that I did not show in class, it's less useful while working, but is helpful for the notes

We can also use the `help` function

```
help(print)
```

Help on built-in function print in module builtins:

```
print(*args, **kwargs)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
        file: a file-like object (stream); defaults to the current sys.stdout.
        sep:   string inserted between values, default a space.
        end:   string appended after the last value, default a newline.
        flush: whether to forcibly flush the stream.
```

The first line says that it can take multiple values, because it says `args*`. The `*` means multiple.

It also has a keyword argument (must be used like `argument=value` and has a default) described as `sep=' '`. This means that by default it adds a space as above.

The help also tells us about other parameters, like the `sep` one

```
print(name, year, 'sdkjfdsk', 'ksdjidf', sep='\n')
```

```
Sarah
2020
sdkjfdsk
ksdjidf
```

Basic programming is a prereq and we will go faster soon, but the goal of this review was to understand notebooks, getting help, and reading docstrings



### Important

More extra tips to help you review/refresh your Python that we did not cover in class. In general, I will not require things that we do not at least get very close to in assignments, but since this is supposed to be in the prerequisite, I am providing resources.

## 1.8. Python Review

Official source on python:

- [PEP8 official style](#)

[Skip to main content](#)

We will go quickly through these focusing on pythonic style, because the prerequisite is a programming course.

## 1.9. Functions

```
def greeting(name):
    ...
    say hi to a person

    Parameters
    -----
    name : string
        the name of the person to greet
    ...
    return 'hi ' + name
```

A few things to note:

- the `def` keyword starts a function
- then the name of the function
- parameters in `()` then `:`
- the body is indented
- the first thing in the body should be a docstring, denoted in `'''` which is a multiline comment
- returning is more reliable than printing in a function

In python, [PEP 257](#) says how to write a docstring, but it is very broad.

In Data Science, [numpydoc](#) style docstrings are popular.

- Pandas follows numpydoc
- [Numpy uses it]
- Scipy follows numpydoc

Once the cell with the function definition is run, we can use the function

```
greeting(name)
```

```
'hi Sarah'
```

```
print(greeting('surbhi'))
```

```
hi surbhi
```

```
assert greeting('sarah') == 'hi sarah'
```

With a return this works to check that it does the right thing.

[Skip to main content](#)

when assert is true, it returns nothing, it throws an error on failure

## 1.10. Conditionals

```
def greeting2(name, formal=False):
    ...
    say hi to a person

    Parameters
    ++++++
    name : string
        the name of the person to greet
    formal: bool
        if the greeting should formal (hello) or not (hi)
    ...
    if formal:
        message = 'hello  ' + name
    else:
        message = 'hi  ' + name
    return message
```

key points in this function:

- an `if` also has the conditional part indented
- for a `bool` variable we can just use the variable
- we can set a default value

because of the default value we do not have to pass the second variable:

```
greeting2(name)
```

```
'hi Sarah'
```

```
greeting2(name, True)
```

```
'hello  Sarah'
```

## 1.11. More Reading

Reading [chapter 1 of think like a data scientist](#) will help you with the data science definition part of the assignment.

Think like a data scientist is written for practitioners; not as a text book for a class. It does not have a lot of prerequisite background, but the sections of it that I assign will help you build a better mental picture of what doing Data Science about.

Only the first assignment will be due this fast, it's a short review and setup assignment. It's due quickly so that we know that you have everything set up and the prerequisite material before we start new material next week.

## 1.12. Questions

1.12.1. Do I need anaconda, or can I just install each package? ie. I just installed jupyter labs using an external terminal command, can I do that for each package we use?"

Yes, and even better, I made a `requirements.txt` file with all of what you should need.

## 2. Iterables and Pandas Data Frames

### 2.1. House Keeping

### 2.2. Assignment 1

You can revise it to fix anything you learned today before I give feedback.

### 2.3. Closing Jupyter server.

In the terminal use Ctrl+C (actually control, not command on mac).

It will ask you a question and give options, read and follow

or

do ctrl+C a second time.

A jupyter server typically runs at `localhost:8888`, but if you have multiple servers running the count increases.

Once I saw a student in office hours working on `localhost:8894` asking why their code kept crashing.

#### ! Important

Remember to close your jupyter server

### 2.4. Using Pandas

We will use data with a library called `pandas`. By convention, we import it like:

```
import pandas as pd
```

- `pandas` is the name of the package that is installed
- `as` keyword allows us to assign an alias (nickname)
- `pd` is the typical alias for pandas

## 2.5. Everything is Data

Data we will see:

- tabular data
- websites as data
- activity logs on websites
- images
- text

## 2.6. Why inspection in code?

Some IDEs give you GUI based tools to inspect objects. We are going to do it programmatically inline with our analyses for two reasons.

- (minor, logistical) it helps make for good notes
- (most importantly) it helps build habits of data science

In data science, our code will be aiming to tell a story.

If you're curious about something, try it out, see what happens. We're going to use a lot of code inspection tools during class. These are helpful both for understanding what's going on, but the advantage to knowing how to get this information programmatically even though a different IDE would give you inspection tools is that it helps you treat your code as data.

## 2.7. everything is an object

let's examine the `type` of some variables:

```
a=4
b = 'monday'
c =5.3
d = print
```

```
type(a)
```

```
int
```

` ints are a base python type, like they appear in other languages

strings are iterable types, meaning that they can be indexed into, or their elements iterated over. For a more technical definition,

[Skip to main content](#)

```
type(b)
```

```
str
```

we can select one element

```
b[0]
```

```
'm'
```

or multiple, this is called slicing.

```
b[0:3]
```

```
'mon'
```

negative numbers count from the right.

```
b[-1]
```

```
'y'
```

```
type(c)
```

```
float
```

a variable can hold a whole function.

```
type(d)
```

```
builtin_function_or_method
```

functions are also objects like any other type in python

we can use the variable just like the function itself

```
d('hello')
```

```
hello
```

1 Im

This  
go  
com

```
print(b)
```

```
monday
```

## 2.8. Tabular Data

Structured data is easier to work with than other data.

We're going to focus on tabular data for now. At the end of the course, we'll examine images, which are structured, but more complex and text, which is much less structured.

## 2.9. Getting familiar with the dataset

We're going to use a dataset about [coffee quality](#) today.

How was this dataset collected?

- reviews added to DB
- then scraped

Where did it come from?

- coffee Quality Institute's trained reviewers.

What format is it provided in?

- csv (Comma Separated Values)

What other information is in this repository?

- the code to scrape and clean the data
- the data before cleaning

It's important to always know where data came from and how it was collected.

This helps you know what is useful for and what its limitations are.

### Further Reading

An important research article on documenting datasets for machine learning is called [Datasheets for Datasets](#) these researchers also did a [follow up study](#) to better understand how practitioner use datasheets and decide how to use data.

If topics like this are interesting to you, let me know! my research is related to this and I have a lot of students who complete 310 do research in my lab.

## 2.10. Loading the Coffee Data

Get raw url for the dataset click on the raw button on the [csv page](#), then copy the url.

```
coffee_data_url = 'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/robusta_d
```

### ⚠️ Warning

This did not work in class, so I downloaded the data and dragged it to the same folder as my notebook

```
pd.read_csv(coffee_data_url)
```

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Numbe
0	1 Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	
1	2 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/2
2	3 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	000
3	4 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	
4	5 Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	
5	6 Robusta	andrew hetzel	India	NaN	NaN	(self)	Nal
6	7 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	Nal
7	8 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	7	sethuraman estate	14/1148/2017/1
8	9 Robusta	nishant gurjer	India	sethuraman estate	RKR	sethuraman estate	14/1148/2016/1
9	10 Robusta	ugacof	Uganda	ishaka	NaN	nsubuga umar	
10	11 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	
11	12 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	RC AB	sethuraman estate	14/1148/2016/1
12	13 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	Nal
13	14 Robusta	kasozi coffee farmers association	Uganda	kasozi coffee farmers	NaN	NaN	
14	15 Robusta	ankole coffee producers coop	Uganda	kyangundu coop society	NaN	ankole coffee producers coop union ltd	
15	16 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	000
16	17 Robusta	andrew	India	sethuraman	NaN	sethuraman es	Nal

[Skip to main content](#)

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Numbe
17	18 Robusta	kawacom uganda ltd	Uganda	bushenyi	NaN	kawacom	
18	19 Robusta	nitubaasa ltd	Uganda	kigezi coffee farmers association	NaN	nitubaasa	
19	20 Robusta	mannya coffee project	Uganda	mannya coffee project	NaN	mannya coffee project	
20	21 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaI
21	22 Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaI
22	23 Robusta	andrew hetzel	United States	sethuraman estates	NaN	sethuraman estates	NaI
23	24 Robusta	luis robles	Ecuador	robustasa	Lavado 1	our own lab	NaI
24	25 Robusta	luis robles	Ecuador	robustasa	Lavado 3	own laboratory	NaI
25	26 Robusta	james moore	United States	fazenda cazengo	NaN	cafe cazengo	NaI
26	27 Robusta	cafe politico	India	NaN	NaN	NaN	14-1118-2014 008
27	28 Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaI

28 rows × 44 columns

If the file is local, you can load it this way. The parameter of the function is the **path** to the dataset, that can be relative, like below, absolute (a full address on your computer) or a URL like above.

```
pd.read_csv('robusta_data_cleaned.csv')
```

This read in the data and printed it out because it is the last line on the cell. If we do something else after, it will read it in, but not print it out.

In order to use it, we save the output to a variable.

```
coffee_df = pd.read_csv(coffee_data_url)
```

we choose this name so that related variables will all use **coffee** and then have other parts after **\_** to describe them in terms of type and content. In Python, for variables, the typical convention is to use **\_** to join words, not CamelCase, which is used for classes, like **DataFrame**

coffee\_df

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Numbe
0	1 Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	
1	2 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/2
2	3 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	000
3	4 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	
4	5 Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	
5	6 Robusta	andrew hetzel	India	NaN	NaN	(self)	Nal
6	7 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	Nal
7	8 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	7	sethuraman estate	14/1148/2017/1
8	9 Robusta	nishant gurjer	India	sethuraman estate	RKR	sethuraman estate	14/1148/2016/1
9	10 Robusta	ugacof	Uganda	ishaka	NaN	nsubuga umar	
10	11 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	
11	12 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	RC AB	sethuraman estate	14/1148/2016/1
12	13 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	Nal
13	14 Robusta	kasozi coffee farmers association	Uganda	kasozi coffee farmers	NaN	NaN	
14	15 Robusta	ankole coffee producers coop	Uganda	kyangundu coop society	NaN	ankole coffee producers coop union ltd	
15	16 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	000
16	17 Robusta	andrew	India	sethuraman	NaN	sethuraman es	Nal

[Skip to main content](#)

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Numbe
17	18 Robusta	kawacom uganda ltd	Uganda	bushenyi	NaN	kawacom	
18	19 Robusta	nitubaasa ltd	Uganda	kigezi coffee farmers association	NaN	nitubaasa	
19	20 Robusta	mannya coffee project	Uganda	mannya coffee project	NaN	mannya coffee project	
20	21 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaI
21	22 Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaI
22	23 Robusta	andrew hetzel	United States	sethuraman estates	NaN	sethuraman estates	NaI
23	24 Robusta	luis robles	Ecuador	robustasa	Lavado 1	our own lab	NaI
24	25 Robusta	luis robles	Ecuador	robustasa	Lavado 3	own laboratory	NaI
25	26 Robusta	james moore	United States	fazenda cazengo	NaN	cafe cazengo	NaI
26	27 Robusta	cafe politico	India	NaN	NaN	NaN	14-1118-2014 008
27	28 Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaI

28 rows × 44 columns

Next we examine the type

```
type(coffee_df)
```

```
pandas.core.frame.DataFrame
```

This is a new type provided by the `pandas` library, called a `dataframe`

We can also examine its parts. It consists of several; first the column headings

```
coffee_df.columns
```

```
Index(['Unnamed: 0', 'Species', 'Owner', 'Country.of-Origin', 'Farm.Name',
       'Lot.Number', 'Mill', 'ICO.Number', 'Company', 'Altitude', 'Region',
       'Producer', 'Number.of.Bags', 'Bag.Weight', 'In.Country.Partner',
       'Harvest.Year', 'Grading.Date', 'Owner.1', 'Variety',
       'Processing.Method', 'Fragrance...Aroma', 'Flavor', 'Aftertaste',
       'Salt...Acid', 'Bitter...Sweet', 'Mouthfeel', 'Uniform.Cup',
       'Clean.Cup', 'Balance', 'Cupper.Points', 'Total.Cup.Points', 'Moisture',
       'Category.One.Defects', 'Quakers', 'Color', 'Category.Two.Defects',
       'Expiration', 'Certification.Body', 'Certification.Address',
       'Certification.Contact', 'unit_of_measurement', 'altitude_low_meters',
       'altitude_high_meters', 'altitude_mean_meters'],
      dtype='object')
```

These are a special type called `Index` that is also provided by pandas.

It also tells us that the actual headings are of `dtype object`. `object` is used for strings or columns with `mixed types`

the `dtype` is slightly different from base Python types and is how pandas classifies but roughly is the same idea as a type.

```
type(coffee_df.columns)
```

```
pandas.core.indexes.base.Index
```

We can look at the first 5 rows with `head`

```
coffee_df.head()
```

	Unnamed: 0	Species	Owner	Country.of-Origin	Farm.Name	Lot.Number	Mill	ICO.Number
0	1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0
1	2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21
2	3	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000
3	4	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0
4	5	Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0

5 rows × 44 columns

Some notes:

[Skip to main content](#)

- `head` is a method, it does something to the content and can rely on parameters (here `n=5` can be changed to show different numbers of rows)

If we forget the `( )` on a method, it looks weird in the output

```
coffee_df.head
```

			Species	Owner	Country.of.Origin
0	1	Robusta	ankole coffee	producers coop	Uganda
1	2	Robusta		nishant gurjer	India
2	3	Robusta		andrew hetzel	India
3	4	Robusta		ugacof	Uganda
4	5	Robusta	katuka development trust ltd		Uganda
5	6	Robusta		andrew hetzel	India
6	7	Robusta		andrew hetzel	India
7	8	Robusta		nishant gurjer	India
8	9	Robusta		nishant gurjer	India
9	10	Robusta		ugacof	Uganda
10	11	Robusta		ugacof	Uganda
11	12	Robusta		nishant gurjer	India
12	13	Robusta		andrew hetzel	India
13	14	Robusta	kasozi coffee farmers association		Uganda
14	15	Robusta	ankole coffee	producers coop	Uganda
15	16	Robusta		andrew hetzel	India
16	17	Robusta		andrew hetzel	India
17	18	Robusta	kawacom uganda ltd		Uganda
18	19	Robusta		nitubaasa ltd	Uganda

We can see more about why this happens with `type`.

```
type(coffee_df.head)
```

```
method
```

Without the parenthesis, it is the literal function object.

```
type(coffee_df.head())
```

```
pandas.core.frame.DataFrame
```

With the parenthesis, it runs the function and `type` examines what it returns, the `DataFrame` object.

## 2.11. Assignment 1 tips

### 2.11.1. Pythonic Loops

In Python, we call good style ‘pythonic’, for loops that means making a sensible loop variable. Let’s first make a list object we can iterate over

[Skip to main content](#)

```
coffee_cols_list = list(coffee_df.columns)
coffee_cols_list
```

```
['Unnamed: 0',
 'Species',
 'Owner',
 'Country.of.Origin',
 'Farm.Name',
 'Lot.Number',
 'Mill',
 'ICO.Number',
 'Company',
 'Altitude',
 'Region',
 'Producer',
 'Number.of.Bags',
 'Bag.Weight',
 'In.Country.Partner',
 'Harvest.Year',
 'Grading.Date',
 'Owner.1',
 'Variety',
 'Processing.Method',
 'Fragrance...Aroma',
 'Flavor',
 'Aftertaste',
 'Salt...Acid',
 'Bitter...Sweet',
 'Mouthfeel',
 'Uniform.Cup',
 'Clean.Cup',
 'Balance',
 'Cupper.Points',
 'Total.Cup.Points',
 'Moisture',
 'Category.One.Defects',
 'Quakers',
 'Color',
 'Category.Two.Defects',
 'Expiration',
 'Certification.Body',
 'Certification.Address',
 'Certification.Contact',
 'unit_of_measurement',
 'altitude_low_meters',
 'altitude_high_meters',
 'altitude_mean_meters']
```

Now we will write a loop to clean up the 

```
clean_cols = []
for col in coffee_cols_list:
    clean_cols.append(col.replace('.', '_'))

clean_cols
```

```
['Unnamed: 0',
 'Species',
 'Owner',
 'Country_of_Origin',
 'Farm_Name',
 'Lot_Number',
 'Mill',
 'ICO_Number',
 'Company',
 'Altitude',
 'Region',
 'Producer',
 'Number_of_Bags',
 'Bag_Weight',
 'In_Country_Partner',
 'Harvest_Year',
 'Grading_Date',
 'Owner_1',
 'Variety',
 'Processing_Method',
 'Fragrance__Aroma',
 'Flavor',
 'Aftertaste',
 'Salt__Acid',
 'Bitter__Sweet',
 'Mouthfeel',
 'Uniform_Cup',
 'Clean_Cup',
 'Balance',
 'Cupper_Points',
 'Total_Cup_Points',
 'Moisture',
 'Category_One_Defects',
 'Quakers',
 'Color',
 'Category_Two_Defects',
 'Expiration',
 'Certification_Body',
 'Certification_Address',
 'Certification_Contact',
 'unit_of_measurement',
 'altitude_low_meters',
 'altitude_high_meters',
 'altitude_mean_meters']
```

This is equivalent, but easier to read than:

```
clean_cols = []
for i in range(len(coffee_cols_list)):
    clean_cols.append(coffee_cols_list[i].replace('.','_'))

clean_cols
```

```
[ 'Unnamed: 0',
  'Species',
  'Owner',
  'Country_of_Origin',
  'Farm_Name',
  'Lot_Number',
  'Mill',
  'ICO_Number',
  'Company',
  'Altitude',
  'Region',
  'Producer',
  'Number_of_Bags',
  'Bag_Weight',
  'In_Country_Partner',
  'Harvest_Year',
  'Grading_Date',
  'Owner_1',
  'Variety',
  'Processing_Method',
  'Fragrance__Aroma',
  'Flavor',
  'Aftertaste',
  'Salt__Acid',
  'Bitter__Sweet',
  'Mouthfeel',
  'Uniform_Cup',
  'Clean_Cup',
  'Balance',
  'Cupper_Points',
  'Total_Cup_Points',
  'Moisture',
  'Category_One_Defects',
  'Quakers',
  'Color',
  'Category_Two_Defects',
  'Expiration',
  'Certification_Body',
  'Certification_Address',
  'Certification_Contact',
  'unit_of_measurement',
  'altitude_low_meters',
  'altitude_high_meters',
  'altitude_mean_meters']
```

In this version the loop variable `i` is a number we have to use to access what we want, where in the first one the `col` loop variable is the thing we want. Simpler and easier to read, which is better by definition in Python.

To make it better than in class, without a lot of extra logic we can do the `...` first then the single ones:

```
clean_cols = []
for col in coffee_cols_list:
    clean_cols.append(col.replace('...', '_').replace('.', '_'))
clean_cols
```

```
[ 'Unnamed: 0',
  'Species',
  'Owner',
  'Country_of_Origin',
  'Farm_Name',
  'Lot_Number',
  'Mill',
  'ICO_Number',
  'Company',
  'Altitude',
  'Region',
  'Producer',
  'Number_of_Bags',
  'Bag_Weight',
  'In_Country_Partner',
  'Harvest_Year',
  'Grading_Date',
  'Owner_1',
  'Variety',
  'Processing_Method',
  'Fragrance_Aroma',
  'Flavor',
  'Aftertaste',
  'Salt_Acid',
  'Bitter_Sweet',
  'Mouthfeel',
  'Uniform_Cup',
  'Clean_Cup',
  'Balance',
  'Cupper_Points',
  'Total_Cup_Points',
  'Moisture',
  'Category_One_Defects',
  'Quakers',
  'Color',
  'Category_Two_Defects',
  'Expiration',
  'Certification_Body',
  'Certification_Address',
  'Certification_Contact',
  'unit_of_measurement',
  'altitude_low_meters',
  'altitude_high_meters',
  'altitude_mean_meters']
```

This shows that we can chain string operations (this will come in handy at other times).

The above is a good form for all `for` loops in Python, but since it was specifically making a list with append, we could make it more concise with a [list comprehension](#).

```
clean_cols_alt = [clean_cols.append(col.replace('...', '_').replace('.','_')) for col in coffee_cols_list]
```

these two ways are the same

```
clean_cols_alt == clean_cols
```

```
False
```

## 2.12. Conditionals Evaluate in order

recall we set this variable

```
a
```

```
4
```

If we write conditions where they can be both true, but we want the larger one to act, if we put them in this order it never sees the second, because the first is true.

```
if a >1:  
    print('greater 1')  
elif a >2:  
    print('greater 2')
```

```
greater 1
```

This one works.

```
if a >2:  
    print('greater 2')  
elif a >1:  
    print('greater 1')
```

```
greater 2
```

## 2.13. Questions After Class

### 2.13.1. What is the name of the inline iteration/loop again in Python?

list comprehension

### 2.13.2. I just want to know more about github In general as to me, although it's new so it will take some time to get used to, it's still pretty confusing to me

I will hold an optional session for a bit more GitHub. You can also take CSC311 for a lot more detail

### 2.13.3. when will we learn about the portfolio?

After A2 feedback which will be the first time it makes sense for you to work on it

[Skip to main content](#)

## 2.13.4. How would you attain a level 3 on any given skill?

There are [example ideas](#) in the Portfolio section of the website, but it will make more sense after Assignment 2 and then I'll spend more time on it in class again.

## 2.13.5. Im confused on what this “pandas.core.frame.DataFrame” is

It is the main data type provided by pandas, that represents a table of data. We will keep working with and inspecting them. For a technical description see the [api docs](#), for a high level description, see the [getting started tutorial](#)

## 2.13.6. would like to get feedback on my homework so I can fix any errors I have

You will get feedback and have a chance to revise later.

## 2.13.7. Can you use any dataset from github using the raw URL and importing it? Can you use any dataset URL or only github?

You can use any URL that has a compatible type of data.

## 2.13.8. if level 1 is determined by attendence and participation, how can I assure I am getting my lesson 1s fulfilled every class

I am removing the prismia grading this semester, but it seems I missed one reference of that in the syllabus.

## 2.13.9. If we get an achievement are we gonna see those on github or do we have to keep track of all the achievements we have to see our grade?

in your feedback you will get a table with your current standing each time work is assessed

## 2.13.10. Can you slow down a little, some times it gets hard to follow along

I will try a little, but also please either message on prismia or raise your hand if you ever fall behind.

# 3. Pandas Data Frames and More Iterable Types

### Note

I was able to fix my certificate problem by running [this script](#) from the python install.

We will import pandas

[Skip to main content](#)

```
import pandas as pd
```

We want to import the coffee data and we will keep working with it over several classes. Sometimes we want to put functions we have written or constants like this in a module of our own that we can import. For today we will put the following in a file called `coffee.py`

```
robusta_data_url = 'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/robusta_c  
robusta_data_file = 'robusta_data_cleaned.csv'
```

Now we can import it

```
import coffee
```

and use the variable with a `.`, as an attribute of the module.

```
coffee.robusta_data_url
```

```
'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/robusta_data_cleaned.csv'
```

We could also import the variables directly so that we can use them without the `.`

```
from coffee import robusta_data_url, robusta_data_file
```

like follows

```
ro
```

```
-----  
NameError                                                 Traceback (most recent call last)  
Cell In[5], line 1  
----> 1 ro  
  
NameError: name 'ro' is not defined
```

This is a `NameError` because the variable does not exist.

When we spell it correctly, it works as desired.

```
robusta_data_url
```

```
'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/robusta_data_cleaned.csv'
```

Then we can use the variable.

```
coffee_df = pd.read_csv(robusta_data_url)
```

### 3.1. DataFrame parts

first we will check the type

```
type(coffee_df)
```

```
pandas.core.frame.DataFrame
```

we saw that we can check the first 5 rows with `head`

```
coffee_df.head()
```

	Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number
0	1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0
1	2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21
2	3	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000
3	4	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0
4	5	Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0

5 rows × 44 columns

and `tail` to see the last ones.

```
coffee_df.tail(3)
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company
25	26 Robusta	james moore	United States	fazenda cazengo	NaN	cafe cazengo	NaN	global opportunity func
26	27 Robusta	cafe politico	India	NaN	NaN	NaN	14-1118-2014-0087	cafe politicc
27	28 Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaN	cafe politicc

3 rows × 44 columns

We note that one of the columns is named `Unnamed 0:` because it does not have a name in the file, for that column.

We can use `shift + tab` to see help. We see that the `index_col` parameter to treat that column as the index.

```
coffee_df = pd.read_csv(robusta_data_url, index_col=0)
coffee_df.head(1)
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude
1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ankole coffee producers coop	1488

1 rows × 43 columns

### 3.1.1. Data Frame parts

Now it looks neater and the columns is the index.

We saw before the columns

```
coffee_df.columns
```

```
Index(['Species', 'Owner', 'Country.of.Origin', 'Farm.Name', 'Lot.Number',
       'Mill', 'ICO.Number', 'Company', 'Altitude', 'Region', 'Producer',
       'Number.of.Bags', 'Bag.Weight', 'In.Country.Partner', 'Harvest.Year',
       'Grading.Date', 'Owner.1', 'Variety', 'Processing.Method',
       'Fragrance...Aroma', 'Flavor', 'Aftertaste', 'Salt...Acid',
       'Bitter...Sweet', 'Mouthfeel', 'Uniform.Cup', 'Clean.Cup', 'Balance',
       'Copper.Points', 'Total.Cup.Points', 'Moisture', 'Category.One.Defects',
       'Quakers', 'Color', 'Category.Two.Defects', 'Expiration',
       'Certification.Body', 'Certification.Address', 'Certification.Contact',
       'unit_of_measurement', 'altitude_low_meters', 'altitude_high_meters',
       'altitude_mean_meters'],
      dtype='object')
```

The rows are named by the `index` attribute

```
coffee_df.index
```

```
Index([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28],
      dtype='int64')
```

Both of these are pandas `Index` objects

We can also take out only the values without the index or the column names.

```
coffee_df.values
```

```
array([['Robusta', 'ankole coffee producers coop', 'Uganda', ..., 1488.0,
       1488.0, 1488.0],
      ['Robusta', 'nishant gurjer', 'India', ..., 3170.0, 3170.0,
       3170.0],
      ['Robusta', 'andrew hetzel', 'India', ..., 1000.0, 1000.0, 1000.0],
      ...,
      ['Robusta', 'james moore', 'United States', ..., 795.0, 795.0,
       795.0],
      ['Robusta', 'cafe politico', 'India', ..., nan, nan, nan],
      ['Robusta', 'cafe politico', 'Vietnam', ..., nan, nan, nan]],
     dtype=object)
```

this is a `numpy.array` type

```
type(coffee_df.values)
```

```
numpy.ndarray
```

### 3.1.2. Segmenting rows or columns

We can pick out a column using it's name and `[]`

```
coffee_df['Flavor']
```

```
1    8.08
2    7.75
3    7.83
4    7.92
5    7.83
6    7.92
7    7.75
8    7.75
9    7.75
10   7.83
11   7.92
12   7.83
13   7.58
14   7.75
15   7.67
16   7.75
17   7.50
18   7.58
19   7.58
20   7.42
21   7.67
22   7.42
23   7.50
24   7.58
25   7.67
26   7.33
27   6.83
28   6.67
Name: Flavor, dtype: float64
```

this is a pandas Series

```
type(coffee_df['Flavor'])
```

```
pandas.core.series.Series
```

We can pick a row out with `loc`, the property that grabs a row by its name, a value from the index.

```
coffee_df.loc[7]
```

```

Species                               Robusta
Owner                                 andrew hetzel
Country.of-Origin                      India
Farm.Name                            sethuraman estates
Lot.Number                           NaN
Mill                                  NaN
ICO.Number                           NaN
Company                               cafemakers
Altitude                             750m
Region                                chikmagalur
Producer                              Nishant Gurjer
Number.of.Bags                       320
Bag.Weight                           2 kg
In.Country.Partner                   Specialty Coffee Association
Harvest.Year                          2014
Grading.Date                         May 15th, 2014
Owner.1                               Andrew Hetzel
Variety                               NaN
Processing.Method                    NaN
Fragrance...Aroma                     7.67
Flavor                                7.75
Aftertaste                            7.83
Salt...Acid                           7.83
Bitter...Sweet                         8.0
Mouthfeel                            7.92
Uniform.Cup                           10.0
Clean.Cup                             10.0
Balance                               7.75
Copper.Points                        7.83
Total.Cup.Points                     82.58
Moisture                             0.0
Category.One.Defects                0
Quakers                             0
Color                                 Green
Category.Two.Defects                0
Expiration                           May 15th, 2015
Certification.Body                  Specialty Coffee Association
Certification.Address               ff7c18ad303d4b603ac3f8cff7e611ffc735e720
Certification.Contact              352d0cf7f3e9be14dad7df644ad65efc27605ae2
unit_of_measurement                 m
altitude_low_meters                750.0
altitude_high_meters               750.0
altitude_mean_meters                750.0
Name: 7, dtype: object

```

this is also a series.

### 3.1.3. Masking to Select Subsets

We can use a boolean mask or Series with boolean values, to pick out a subset of rows.

To build up to that, we will first evaluate a boolean expression on a column.

```
coffee_df['Flavor'] > 7.8
```

```
1      True
2     False
3      True
4      True
5      True
6      True
7     False
8     False
9     False
10     True
11     True
12     True
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
27    False
28    False
Name: Flavor, dtype: bool
```

Now, we can save it to a variable, and then use it to mask, by treating it as an index.

```
high_flavor = coffee_df['Flavor']>7.8
coffee_df[high_flavor]
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Compa
1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ank cof produc cc
3	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuran est
4	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof
5	Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0	katu developm trust
6	Robusta	andrew hetzel	India	NaN	NaN	(self)	NaN	cafemake
10	Robusta	ugacof	Uganda	ishaka	NaN	nsubuga umar	0	ugacof
11	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof
12	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	RC AB	sethuraman estate	14/1148/2016/12	kaapi roy

8 rows × 43 columns

this segments out only the selected rows.

The values are a numpy array which also has the same `shape` as the dataframe, but it does not include the index or the column headers (which do not count as rows or columns anyway)

```
coffee_vals = coffee_df.values
```

We can see this with the `shape` attribute

```
coffee_vals.shape
```

```
(28, 43)
```

```
coffee_df.shape
```

```
(28, 43)
```

## 3.2. Reading from a website

We saw in the help that pandas has a lot of `read_` methods. You can also see a list on the [IO page of the docs](#)

```
achievements_url = 'https://rhodyprog4ds.github.io/BrownFall24/syllabus/achievements.html'
```

We can read from html files (websites)

```
pd.read_html(achievements_url, )
```

```

[ Unnamed: 0_level_0                                topics \
  week
0      1                               Unnamed: 1_level_1
1      2                               [admin, python review]
2      3                               Loading data, Python review
3      4                               Exploratory Data Analysis
4      5                               Data Cleaning
5      6   Databases, Merging DataFrames
6      7   Modeling, classification performance metrics, ...
7      8                               Naive Bayes, decision trees
8      9                               Regression
9      10                          Clustering
10     11                          SVM, parameter tuning
11     12                          KNN, Model comparison
12     13                          Text Analysis
13     14                          Images Analysis
14                               Deep Learning

          skills
Unnamed: 2_level_1
0      process
1  [access, prepare, summarize]
2  [summarize, visualize]
3  [prepare, summarize, visualize]
4  [access, construct, summarize]
5  [evaluate]
6  [classification, evaluate]
7  [regression, evaluate]
8  [clustering, evaluate]
9  [optimize, tools]
10  [compare, tools]
11  [unstructured]
12  [unstructured, tools]
13  [tools, compare] , skill \
Unnamed: 0_level_0
  keyword
0  python
1  process
2  access
3  construct
4  summarize
5  visualize
6  prepare
7  evaluate
8  classification
9  regression
10 clustering
11 optimize
12 compare
13 representation
14 workflow      Choose representations and transform data
                 use industry standard data science tools and w...
                 skill \
  Unnamed: 1_level_1
0  keyword
1  process
2  access
3  construct
4  summarize
5  visualize
6  prepare
7  evaluate
8  classification
9  regression
10 clustering
11 optimize
12 compare
13 representation
14 workflow      Choose representations and transform data
                 use industry standard data science tools and w...
                 skill \
  Unnamed: 2_level_1
0  python code that mostly runs, occasional pep8 ...
1  Identify basic components of data science
2  load data from at least one format; identify t...
3  identify what should happen to merge datasets ...
4  Describe the shape and structure of a dataset ...
5  identify plot types, generate basic plots from...
6  identify if data is or is not ready for analys...
7  Explain and compute basic performance metrics ...
8  identify and describe what classification is, ...
9  identify what data that can be used for regres...
10  describe what clustering is
11 Identify when model parameters need to be opti...
12  Qualitatively compare model classes

```

[Skip to main content](#)

```

          Level 2 \
          Unnamed: 3_level_1
0 python code that reliably runs, frequent pep8 ...
1 Describe and define each stage of the data sci...
2 Load data for processing from the most common ...
3 apply basic merges
4 compute summary statndard statistics of a whol...
5 generate multiple plot types with complete lab...
6 apply data reshaping, cleaning, and filtering ...
7 Apply and interpret basic model evaluation met...
8 fit, apply, and interpret preselected classifi...
9     fit and interpret linear regression models
10         apply basic clustering
11 Optimize basic model parameters such as model ...
12 Compare model classes in specific terms and fi...
13 Apply at least one representation to transform...
14 Solve well-strucutred, open-ended problems, ap...

```

```

          Level 3
          Unnamed: 4_level_1
0 reliable, efficient, pythonic code that consis...
1 Compare different ways that data science can f...
2 access data from both common and uncommon form...
3 merge data that is not automatically aligned
4 Compute and interpret various summary statisti...
5 generate complex plots with pandas and plottin...
6 apply data reshaping, cleaning, and filtering ...
7 Evaluate a model with multiple metrics and cro...
8 fit and apply classification models and select...
9 fit and explain regurlarized or nonlinear regr...
10 apply multiple clustering techniques, and inte...
11 Select optimal parameters based of mutiple qua...
12 Evaluate tradeoffs between different model com...
13 apply transformations in different contexts OR...
14 Independently scope and solve realistic data s... , A2 \

```

	Unnamed: 0_level_0	A1	A2	\
0	python	1	1	
1	process	1	0	
2	access	0	1	
3	construct	0	0	
4	summarize	0	0	
5	visualize	0	0	
6	prepare	0	0	
7	evaluate	0	0	
8	classification	0	0	
9	regression	0	0	
10	clustering	0	0	
11	optimize	0	0	
12	compare	0	0	
13	representation	0	0	
14	workflow	0	0	

	A3	A4	A5	\
0	0	1	1	
1	0	0	0	
2	1	1	1	
3	0	0	1	
4	1	1	1	
5	1	1	0	
6	0	1	1	
7	0	0	0	
8	0	0	0	
9	0	0	0	
10	0	0	0	
11	0	0	0	

	0	0	0
14	A6	A7	A8 \
	Unnamed: 6_level_1	Unnamed: 7_level_1	Unnamed: 8_level_1
0	0	0	0
1	1	1	1
2	0	0	0
3	0	1	1
4	1	1	1
5	1	1	1
6	0	0	0
7	1	1	1
8	0	1	0
9	0	0	1
10	0	0	0
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
	A9	A10	A11 \
	Unnamed: 9_level_1	Unnamed: 10_level_1	Unnamed: 11_level_1
0	0	0	0
1	1	1	1
2	0	0	0
3	0	0	0
4	1	1	1
5	1	1	1
6	0	0	0
7	0	1	1
8	0	1	0
9	0	0	1
10	1	0	1
11	0	1	1
12	0	0	1
13	0	0	0
14	0	1	1
	A12	A13	# Assignments
	Unnamed: 12_level_1	Unnamed: 13_level_1	Unnamed: 14_level_1
0	0	0	4
1	0	0	7
2	0	0	4
3	0	0	3
4	1	1	11
5	1	1	10
6	0	0	2
7	0	0	5
8	0	0	2
9	0	0	2
10	0	0	2
11	0	0	2
12	0	1	2
13	1	1	2
14	1	1	4 ]

this does not look like the others, it is a list of the DataFrames for all of the tables on the page. Even if there is only one, it will still be in a list.

Now that we know it is a list, we will save it in a variable and then, pick out one DataFrame.

```
ach_df_list = pd.read_html(achievements_url, )
ach_defn = ach_df_list[2]
```

[Skip to main content](#)

```
type(ach_defn)
```

```
pandas.core.frame.DataFrame
```

### 3.3. Slicing

We will use a small list to practice slicing, but it works the same in a DataFrame.

```
toy_list = [3, 4, 5, 6]
```

we can get the last item with `-1`

```
toy_list[-1]
```

```
6
```

We can use `:` to go up to, but not including a particular point.

```
toy_list[:2]
```

```
[3, 4]
```

Note that using `2` picks out the value in position 2, but the above stopped before that.

```
toy_list[2]
```

```
5
```

we can slice from the right, but this time it is inclusive

```
toy_list[-2:]
```

```
[5, 6]
```

we can put numbers on both sides of the `:` to take a segment out

```
toy_list[1:3]
```

```
[4, 5]
```

we can use two to specify an interval at which to pick values.

```
toy_list[::2]
```

```
[3, 5]
```

or we can state the starting point with it

```
toy_list[1::2]
```

```
[4, 6]
```

## 3.4. Built in iterable types

These are four different iterable constructions:

```
a = [char for char in 'abcde']
b = {char:i for i, char in enumerate('abcde')}
c = ('a','b','c','d','e')
d = 'a b c d e'.split(' ')
```

We can see their types

```
type(a), type(b), type(c), type(d)
```

```
(list, dict, tuple, list)
```

Dictionaries are really useful because they consist of key, value pairs. This is really powerful and we will use it a lot to pass complex structures into functions.

we can make another one more manually that is much like the above.

```
sample_dict = {'a':1,'b':2,'c':3}
sample_dict
```

```
{'a': 1, 'b': 2, 'c': 3}
```

we access values of a dictionary using the key

```
sample_dict['a']
```

we can iterate over them using the `items` method to pop them off separately.

```
for char,num in sample_dict.items():
    print('char', char, 'is linked to number', num)
```

```
char a is linked to number 1
char b is linked to number 2
char c is linked to number 3
```

## 3.5. Saving an excerpt

We can find the rows only with a small `Number.of.Bags`, say <100:

```
small_batch_mask = coffee_df['Number.of.Bags']<100
small_batch = coffee_df[small_batch_mask]
```

After we make the mask, then subset the data, we see its a dataframe

```
type(small_batch)
```

```
pandas.core.frame.DataFrame
```

and then we can save it to csv.

```
small_batch.to_csv('robusta_small_batch.csv')
```

# 4. Exploratory Data Analysis

Now we get to start actual data science!

## 4.1. This week: Exploratory Data Analysis

- How to summarize data
- Interpreting summaries
- Visualizing data
- interpreting summaries

### 4.1.1. Summarizing and Visualizing Data are **very** important

- People cannot interpret high dimensional or large samples quickly
- Important in FDA to help you make decisions about the rest of your analysis

[Skip to main content](#)

- Summaries are similar calculations to performance metrics we will see later
- visualizations are often essential in debugging models

## THEREFORE

- You have a lot of chances to earn summarize and visualize
- we will be picky when we assess if you earned them or not

```
import pandas as pd
```

Today we will work with a new dataset about Carbon emissions

```
carbon_data_url = 'https://github.com/rfordatascience/tidytuesday/raw/master/data/2024/2024-05-21/emissions.csv'
```

```
carbon_df = pd.read_csv(carbon_data_url)
```

data readme

```
carbon_df.head()
```

	year	parent_entity	parent_type	commodity	production_value	production_unit	total_emissions_MtCO2e
0	1962	Abu Dhabi National Oil Company	State-owned Entity	Oil & NGL	0.91250	Million bbl/yr	0.363885
1	1962	Abu Dhabi National Oil Company	State-owned Entity	Natural Gas	1.84325	Bcf/yr	0.134355
2	1963	Abu Dhabi National Oil Company	State-owned Entity	Oil & NGL	1.82500	Million bbl/yr	0.727770
3	1963	Abu Dhabi National Oil Company	State-owned Entity	Natural Gas	4.42380	Bcf/yr	0.322453
4	1964	Abu Dhabi National Oil Company	State-owned Entity	Oil & NGL	7.30000	Million bbl/yr	2.911079

In this case, `head` shows that these rows are very similar because it is sorted. Tail would probably be similar, so we will try `sample` to get a random subset.

```
carbon_df.sample(5)
```

	year	parent_entity	parent_type	commodity	production_value	production_unit	total_emissions_MtC
5586	2015	Kazakhstan	Nation State	Metallurgical Coal	11.639030	Million tonnes/yr	34.528
8004	2007	Petoro	State-owned Entity	Natural Gas	1108.527398	Bcf/yr	80.800
11715	2006	Ukraine	Nation State	Sub- Bituminous Coal	0.229453	Million tonnes/yr	0.463
11668	1994	Ukraine	Nation State	Bituminous Coal	38.829776	Million tonnes/yr	105.389
780	1989	Bapco Energies	State-owned Entity	Oil & NGL	15.700000	Million bbl/yr	6.260

## 4.2. Describing a Dataset

So far, we've loaded data in a few different ways and then we've examined DataFrames as a data structure, looking at what different attributes they have and what some of the methods are, and how to get data into them.

The `describe` method provides us with a set of summary statistics that broadly

```
carbon_df.describe()
```

	year	production_value	total_emissions_MtCO2e
<b>count</b>	12551.000000	12551.000000	12551.000000
<b>mean</b>	1987.148116	412.712258	113.219734
<b>std</b>	29.202455	1357.569683	329.812666
<b>min</b>	1854.000000	0.004398	0.000321
<b>25%</b>	1973.000000	10.601353	8.785294
<b>50%</b>	1994.000000	63.203536	33.058688
<b>75%</b>	2009.000000	320.664764	102.154596
<b>max</b>	2022.000000	27192.000000	8646.905949

And these all give us a sense of the values and the distribution or spread fo the data in each column.

### 4.2.1. Individual statistics

We can also extract each of the statistics that the `describe` method calculates individually, by name.

```
carbon_df.max()
```

```
year                      2022
parent_entity                YPF
parent_type      State-owned Entity
commodity        Thermal Coal
production_value       27192.0
production_unit    Million tonnes/yr
total_emissions_MtCO2e     8646.905949
dtype: object
```

## 4.2.2. Understanding Quantiles

The 50% has another more common name: the median. It means 50% of the data are lower (and higher) than this value.

The quantiles are tricky, we cannot just `.25%( )` to get the 25% percentile, we have to use the `quantile` method and pass it a value between 0 and 1.

```
carbon_df['production_value'].quantile(.25)
```

```
10.6013534253315
```

```
carbon_df['production_value'].quantile(.8)
```

```
448.5
```

## 4.3. Individual variable

We can use the descriptive statistics on individual columns as well

## 4.4. What is the average total emissions in this dataset?

```
carbon_df['total_emissions_MtCO2e'].mean()
```

```
113.21973391166483
```

## 4.5. Working with categorical data

There are different columns in the describe than the the whole dataset:

```
carbon_df.columns
```

```
Index(['year', 'parent_entity', 'parent_type', 'commodity', 'production_value',
       'production_unit', 'total_emissions_MtCO2e'],
      dtype='object')
```

So far, the stats above are only for numerical features.

We can get the prevalence of each one with `value_counts`, this can also allow us to answer certain questions. Let's try a few.

#### 4.5.1. What is the most common parent type?

```
carbon_df['parent_type'].value_counts()
```

```
parent_type
Investor-owned Company    6583
State-owned Entity        3914
Nation State              2054
Name: count, dtype: int64
```

▶ Show code cell source

```
'Investor-owned Company'
```

This shows us that the most common one is 'Investor-owned Company'

We also notice that the data is relatively balanced enough that we can make good comparisons from the overall output.

For the specific question, we could instead use the `mode` method (the most frequent value).

```
carbon_df['parent_type'].mode()
```

```
0    Investor-owned Company
Name: parent_type, dtype: object
```

#### 4.5.2. Is it reasonable to compare and use all of the the `commodity` types in the data

To do this, we again look at the value counts:

```
carbon_df['commodity'].value_counts()
```

Hi  
gets  
the  
feat  
cell

```
commodity
Oil & NGL           3733
Natural Gas          3452
Bituminous Coal     1370
Metallurgical Coal   1073
Lignite Coal         1008
Sub-Bituminous Coal  673
Thermal Coal          611
Anthracite Coal       368
Cement                  263
Name: count, dtype: int64
```

Here, we note that the top few are a lot more similar and the smallest one is a lot lower. Here we might drop only the bottom couple or keep them all because for a lot of things 200 measurements is enough.

Another choice is that we could combine the different types of coals together. We can do this by adding a column.

```
carbon_df['commodity_simple'] = carbon_df['commodity'].apply(lambda s: s if not('Coal' in s) else 'Coal')
```

The way this works is:

- `carbon_df['commodity']` picks out the column of interest
- the `.apply` method applies a function (that we pass to it) to all of the elements of a series (or to the rows or columns of a DataFrame)
- `lambda s: s if not('Coal' in s) else 'coal'` is a lambda function that returns the same value if Col is not in it or only `Coal` if it is
- assigns values to a column `'commodity_simple'` `carbon_df['commodity_simple'] =`. It also creates that column since it did not exist

### 4.5.3. understanding lambda functions

To better understand the lambda function that we used above, let's assign it to a variable.

```
coal_strip = lambda s: s if not('Coal' in s) else 'Coal'
```

First we will inspect it a little:

```
type(coal_strip)
```

```
function
```

We can see it is a function here

Now we can make calls to it, like any other functions

```
coal_strip('Metallurgical Coal')
```

[Skip to main content](#)

```
'Coal'
```

this works and now we test the other case:

```
coal_strip('naything else')
```

```
'naything else'
```

#### 4.5.4. How a column is added

Now we can see a sample of the DataFrame again to see how the `apply` method works.

```
carbon_df.sample(5)
```

	year	parent_entity	parent_type	commodity	production_value	production_unit	total_emissions_MtC
3294	2007	Coterra Energy	Investor-owned Company	Oil & NGL	10.402000	Million bbl/yr	4.14€
11666	1993	Ukraine	Nation State	Metallurgical Coal	32.267974	Million tonnes/yr	95.72€
8381	1977	Petroleum Development Oman	State-owned Entity	Oil & NGL	74.495040	Million bbl/yr	29.70€
10575	2014	Singareni Collieries	State-owned Entity	Lignite Coal	3.863218	Million tonnes/yr	5.17€
563	2010	APA Corporation	Investor-owned Company	Natural Gas	689.485000	Bcf/yr	50.25€

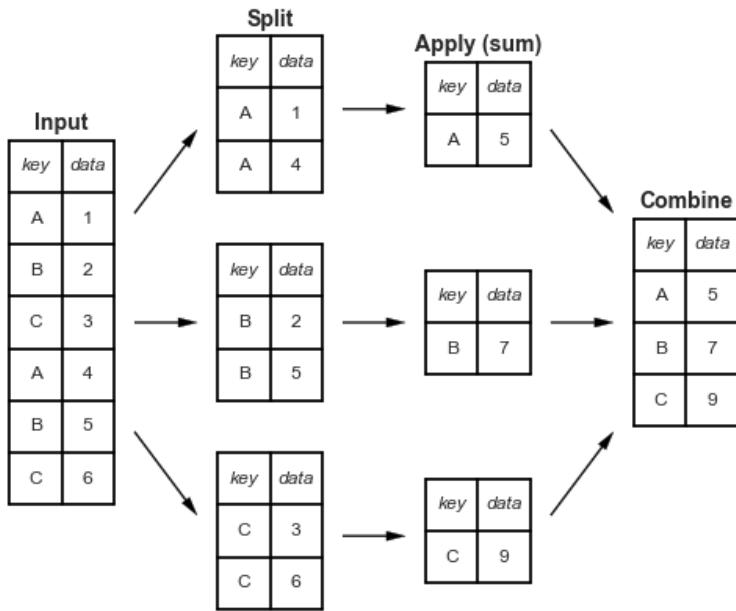
We can further examine this column:

```
carbon_df['commodity_simple'].value_counts()
```

```
commodity_simple
Coal           5103
Oil & NGL      3733
Natural Gas     3452
Cement          263
Name: count, dtype: int64
```

Now the numbers are more similar, we might want to drop the cement, we can do that using a mask that we saw last week.

## 4.6. Split-Apply-Combine



see it in action on [pandas tutor](#)

### 4.6.1. Which commodity type has the highest average emissions?

To do this we group by the categorical value, the new simplified commodity column we created, then we pick out the emissions column, and take the average.

```
carbon_df.groupby('commodity_simple')['total_emissions_MtCO2e'].mean()
```

```
commodity_simple
Cement           112.859242
Coal             118.369779
Natural Gas      75.073875
Oil & NGL        141.479481
Name: total_emissions_MtCO2e, dtype: float64
```

### 4.6.2. How apply works

We can use `apply` with any function, that we write in advance, that comes from a library, or is built in. Above, we used a lambda we defined on the fly, but here we can make it first.

```
def first_chars(s):
    return s[:5]

carbon_df['commodity'].apply(first_chars)
```

[Skip to main content](#)

```
0      Oil &
1      Natur
2      Oil &
3      Natur
4      Oil &
...
12546  Natur
12547  Oil &
12548  Natur
12549  Oil &
12550  Natur
Name: commodity, Length: 12551, dtype: object
```

### 4.6.3. Grouping Multiple times

we can pass a list to group by two values

```
carbon_df.groupby(['commodity_simple', 'parent_type'])['total_emissions_MtCO2e'].mean()
```

```
commodity_simple  parent_type
Cement           Investor-owned Company    38.813769
                  Nation State            243.802816
Coal             Investor-owned Company    45.397065
                  Nation State            228.621342
                  State-owned Entity      94.977580
Natural Gas      Investor-owned Company    58.313678
                  Nation State            741.445688
                  State-owned Entity      80.152170
Oil & NGL        Investor-owned Company    106.583580
                  Nation State            885.428265
                  State-owned Entity      162.042650
Name: total_emissions_MtCO2e, dtype: float64
```

This is one Series, but it has a multi-index. We can check the type

```
em_by_comm_parent = carbon_df.groupby(['commodity_simple', 'parent_type'])['total_emissions_MtCO2e'].mean()
type(em_by_comm_parent)
```

```
pandas.core.series.Series
```

and look at the index to confirm

```
em_by_comm_parent.index
```

```
MultiIndex([(  'Cement', 'Investor-owned Company'),
(  'Cement',          'Nation State'),
(  'Coal', 'Investor-owned Company'),
(  'Coal',          'Nation State'),
(  'Coal', 'State-owned Entity'),
('Natural Gas', 'Investor-owned Company'),
('Natural Gas',          'Nation State'),
('Natural Gas', 'State-owned Entity'),
('Oil & NGL', 'Investor-owned Company'),
('Oil & NGL',          'Nation State'),
('Oil & NGL', 'State-owned Entity')],
names=['commodity_simple', 'parent_type'])
```

We can use `reset_index` to change from the multi-index into a new count from 0 index (and make it a DataFrame in the process)

```
carbon_df.groupby(['commodity_simple', 'parent_type'])['total_emissions_MtCO2e'].mean().reset_index()
```

	commodity_simple	parent_type	total_emissions_MtCO2e
0	Cement	Investor-owned Company	38.813769
1	Cement	Nation State	243.802816
2	Coal	Investor-owned Company	45.397065
3	Coal	Nation State	228.621342
4	Coal	State-owned Entity	94.977580
5	Natural Gas	Investor-owned Company	58.313678
6	Natural Gas	Nation State	741.445688
7	Natural Gas	State-owned Entity	80.152170
8	Oil & NGL	Investor-owned Company	106.583580
9	Oil & NGL	Nation State	885.428265
10	Oil & NGL	State-owned Entity	162.042650

## 4.7. Prepare for next class

Seaborn is a plotting library that gives us *opinionated defaults*

### ⚠ Important

Run this line one time before class on Thursday

```
import seaborn as sns
```

If you get `ModuleNotFoundError` error, open a terminal tab and run `pip install seaborn`.

[Skip to main content](#)

seaborn's alias is `sns` as an inside joke among the developers to the character, Samuel Norman Seaborn from West Wing they named the library after per their FAQ

## 4.8. Questions After Class

### ! Important

Questions about the assignment have been addressed by adding hints and tips to the end of the instructions

### 4.8.1. when putting the two column names `["commodity_simple", "parent_type"]` in the square brackets, what did that do?

We can answer a question like this by inspecting the code further

First we can look directly at it:

```
[ "commodity_simple", "parent_type" ]
```

```
'commodity_simple', 'parent_type'
```

Next, we can check its type:

```
type([ "commodity_simple", "parent_type" ])
```

```
list
```

It makes a `list` which is then compatible with the `groupby` method.

We can look at its help to remember what it can be passed:

```
# the __doc__ attribute is a property of all functions
# it is a string so I used split to break it into lines,
# looked to see how many I needed then joined them back together
# and printed them (to have the newlines render)
print('\n'.join(carbon_df.groupby.__doc__.split('\n')[:18]))
```

Group DataFrame using a mapper or by a Series of columns.

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

#### Parameters

by : mapping, function, label, pd.Grouper or list of such  
Used to determine the groups for the groupby.  
If ``by`` is a function, it's called on each value of the object's index. If a dict or Series is passed, the Series or dict VALUES will be used to determine the groups (the Series' values are first aligned; see ``.align()`` method). If a list or ndarray of length equal to the selected axis is passed (see the `groupby user guide <[https://pandas.pydata.org/pandas-docs/stable/user\\_guide/groupby.html#splitting-an-object-into-groups](https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html#splitting-an-object-into-groups)>

### 4.8.2. How does the apply function work?

I have explained it and linked resources above, plus the [pandas](#) docs have a section on it in their user guide

### 4.8.3. How should we be practicing this material. Its hard to tell whats the specific take away from this class vs memorizing methods.

Review the notes and see the analysis.

What we did is ann example data anlysis. We asked and answered several questions. This serves as example questions and types of questions. How to interpret the statistics.

### 4.8.4. what does the lambda do inside of the function we created to group the coal commodity

`lambda` is the keyword to define a function in line, like `def` is in a full length definition.

### 4.8.5. Is there a way to scrape data from a website and make it a csv file if the website does not allow for exporting?

Sometimes, we will learn webscraping in a few weeks. For now, download locally, or try a different dataset.

### 4.8.6. Will be working on forms of data visualization?

Yes

## 5. Visualization

## 5.1. Plotting in Python

There are several popular plotting libraries:

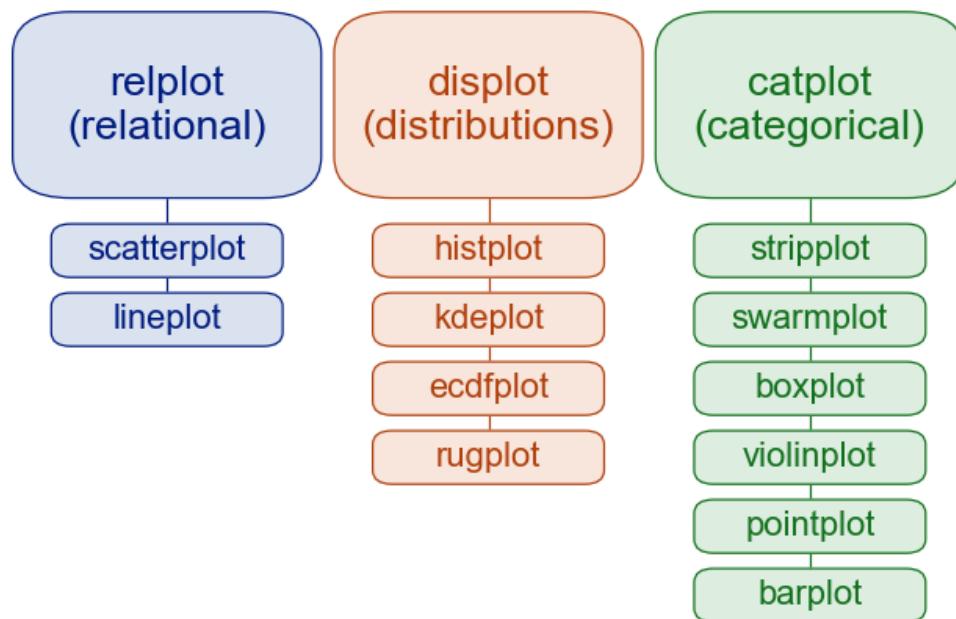
- `matplotlib`: low level plotting tools
- `seaborn`: high level plotting with opinionated defaults
- `ggplot`: plotting based on the `ggplot` library in R.

Plus pandas has a `plot` method

Pandas and seaborn use matplotlib under the hood.

Seaborn and ggplot both assume the data is set up as a DataFrame. Getting started with seaborn is the simplest, so we'll use that.

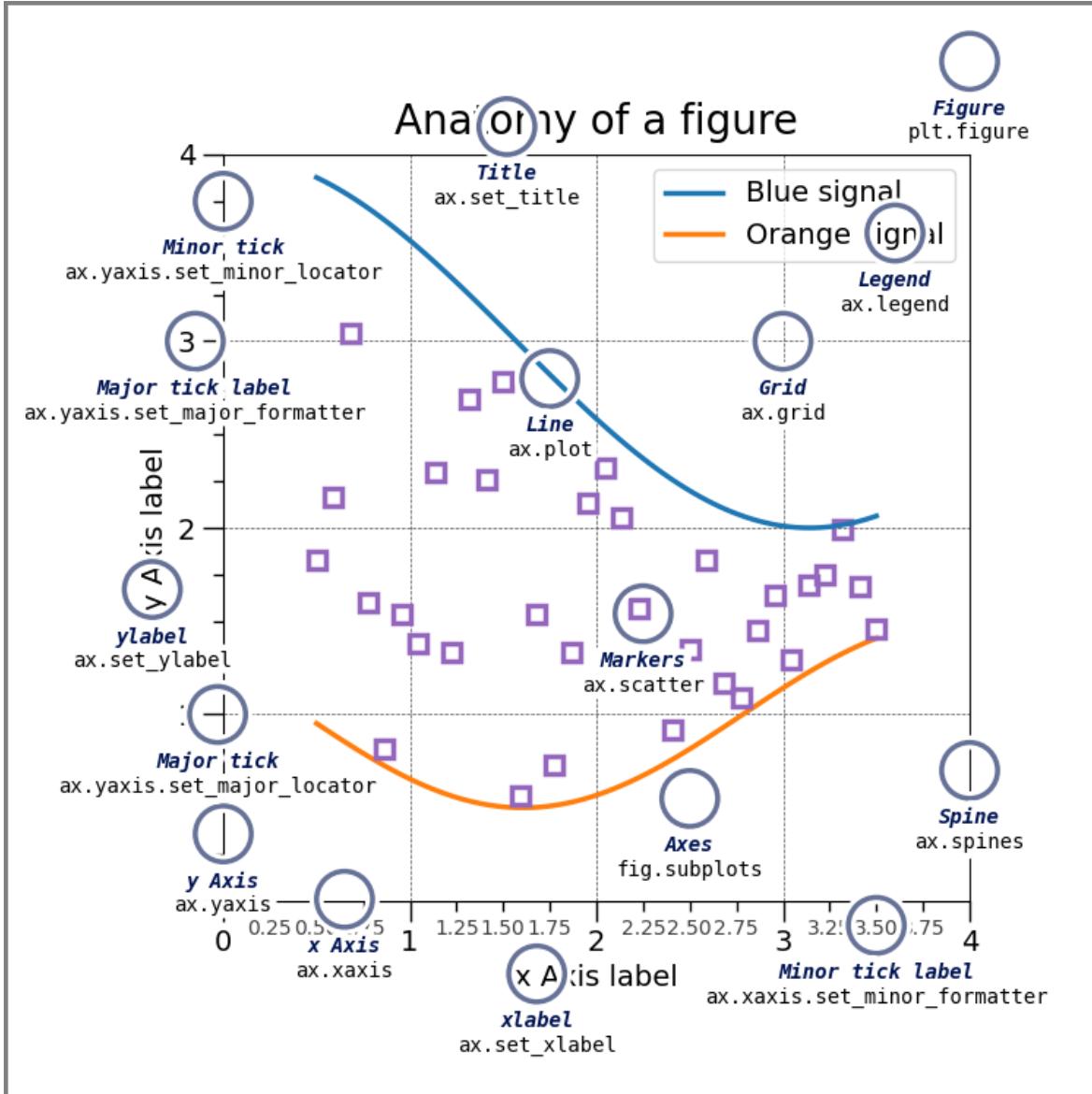
## 5.2. Figure and axis level plots



add the image to your notebook with the following:

```
![summary of plot types](https://seaborn.pydata.org/_images/function_overview_8_0.png)
```

## 5.3. Anatomy of a figure



\*this was drawn with code

add the image to your notebook with the following:

```
![annotated graph](https://matplotlib.org/stable/_images/sphx_glr_anatomy_001.png)
```

figure vs axes

we will load pandas and seaborn

```
import pandas as pd
import seaborn as sns
```

[Skip to main content](#)

```
carbon_data_url = 'https://github.com/rfordatascience/tidytuesday/raw/master/data/2024/2024-05-21/emissions.csv'
```

and the added column we made last class

```
carbon_df = pd.read_csv(carbon_data_url)
carbon_df['commodity_simple'] = carbon_df['commodity'].apply(lambda s: s if not('Coal' in s) else 'Coal')
```

```
carbon_df.head()
```

	year	parent_entity	parent_type	commodity	production_value	production_unit	total_emissions_MtCO2e
0	1962	Abu Dhabi National Oil Company	State-owned Entity	Oil & NGL	0.91250	Million bbl/yr	0.363885
1	1962	Abu Dhabi National Oil Company	State-owned Entity	Natural Gas	1.84325	Bcf/yr	0.134355
2	1963	Abu Dhabi National Oil Company	State-owned Entity	Oil & NGL	1.82500	Million bbl/yr	0.727770
3	1963	Abu Dhabi National Oil Company	State-owned Entity	Natural Gas	4.42380	Bcf/yr	0.322453
4	1964	Abu Dhabi National Oil Company	State-owned Entity	Oil & NGL	7.30000	Million bbl/yr	2.911079

## 5.4. How are the samples distributed in time?

We have not yet worked with the `year` column. An important first step we might want to know is how the measurements are distributed in time.

From last class, we might try `value_counts`

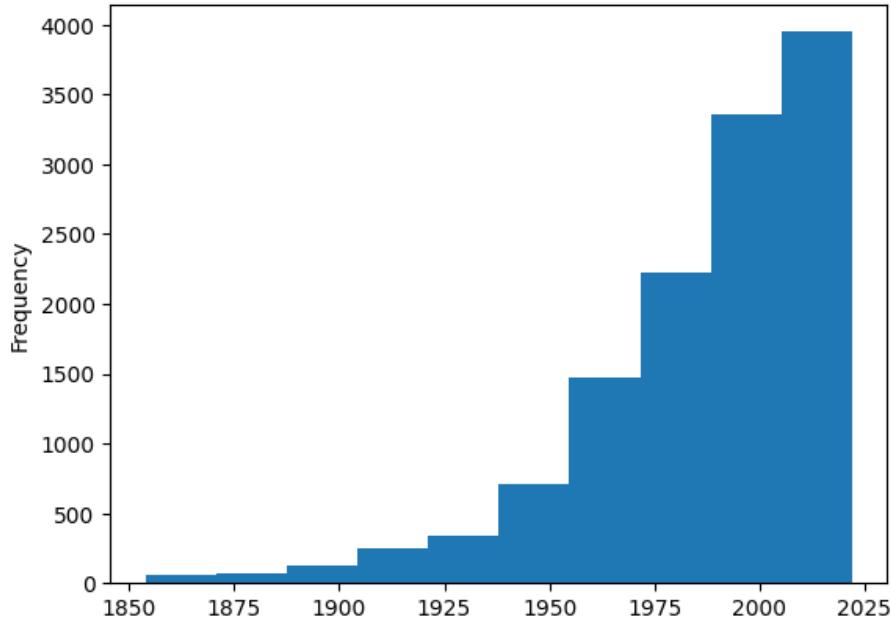
```
carbon_df['year'].value_counts()
```

```
year
2021    238
2022    238
2018    237
2019    236
2020    235
...
1858     3
1857     3
1856     3
1854     3
1863     3
Name: count, Length: 169, dtype: int64
```

but it's a little hard to read. A histogram might better

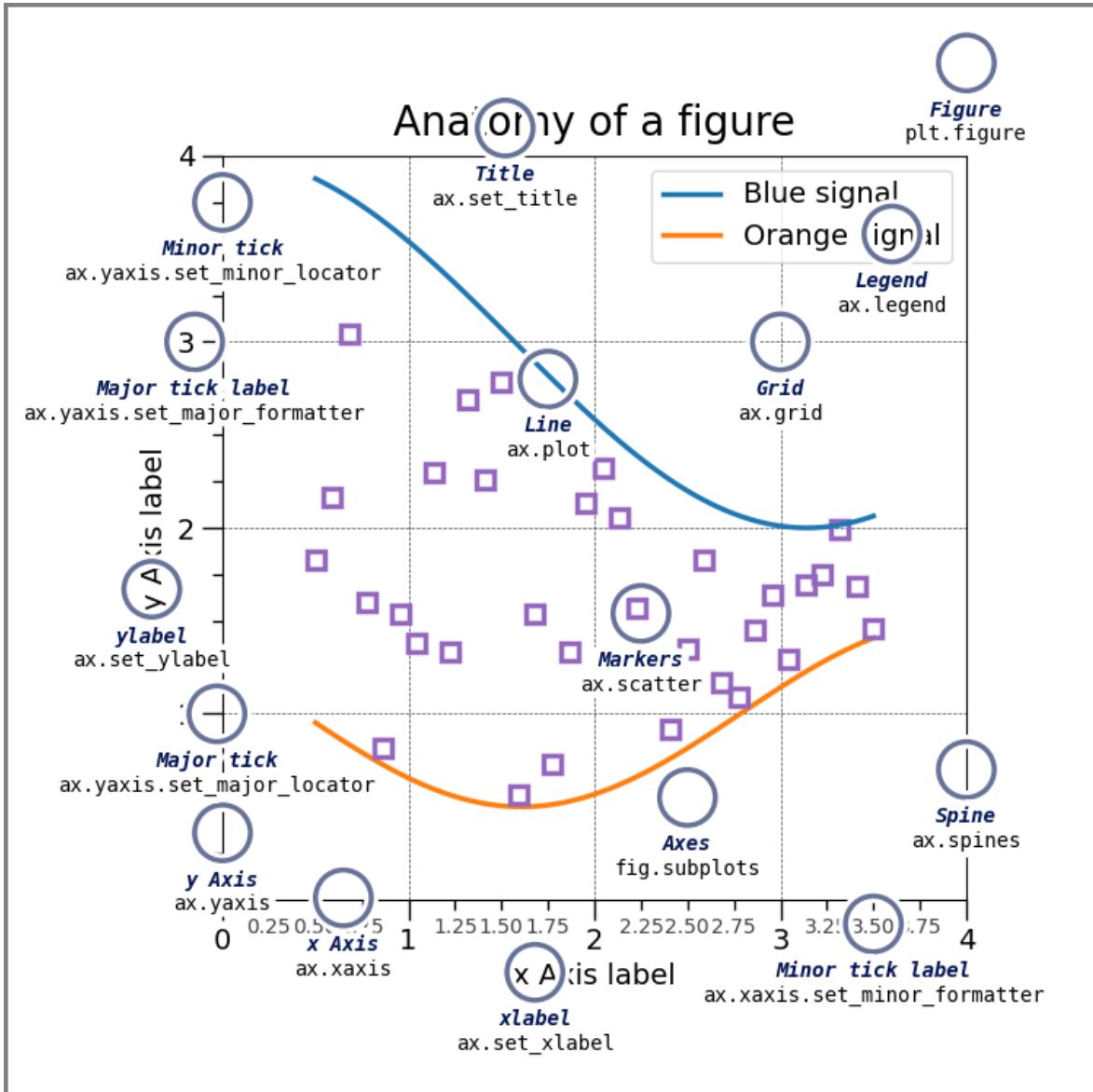
```
carbon_df['year'].plot(kind='hist')
```

```
<Axes: ylabel='Frequency'>
```



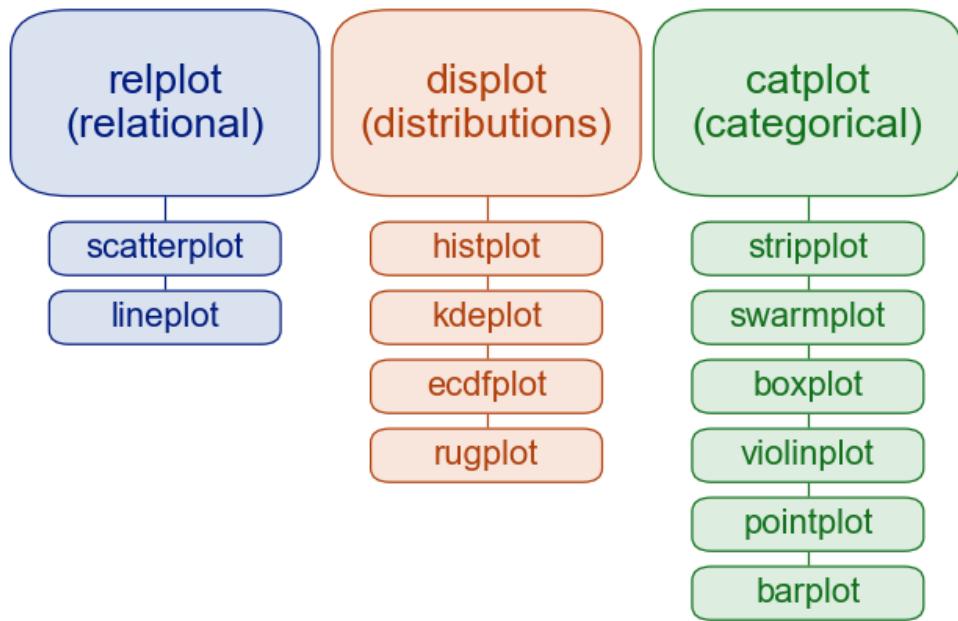
Here we see that there are a lot more samples in more recent years.

## 5.5. Anatomy of a plot



come from matplotlib's Anatomy of a Figure page which includes the code to generate that figure

## 5.6. Figure and axis level plots



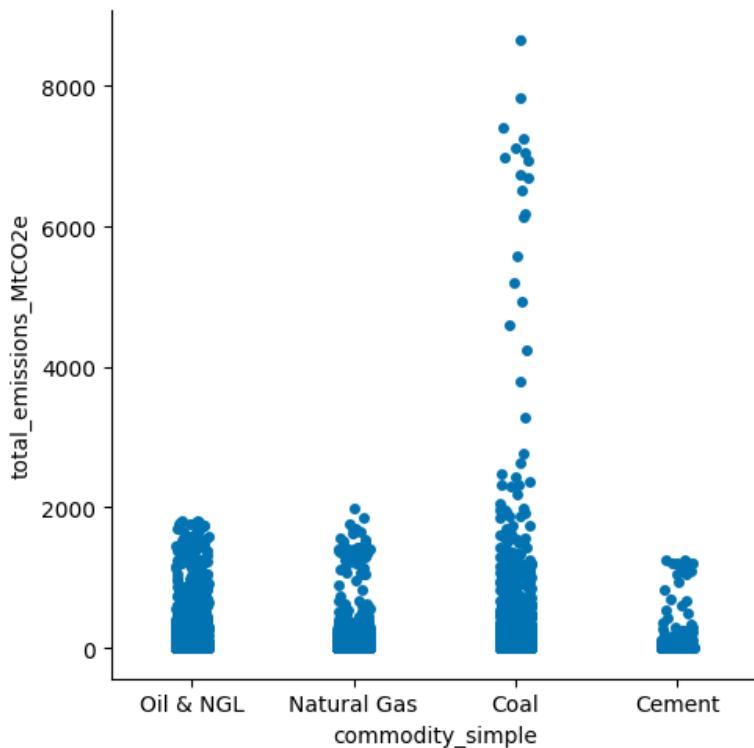
## 5.7. Changing colors

```
sns.set_palette('colorblind')
```

## 5.8. Emissions by type

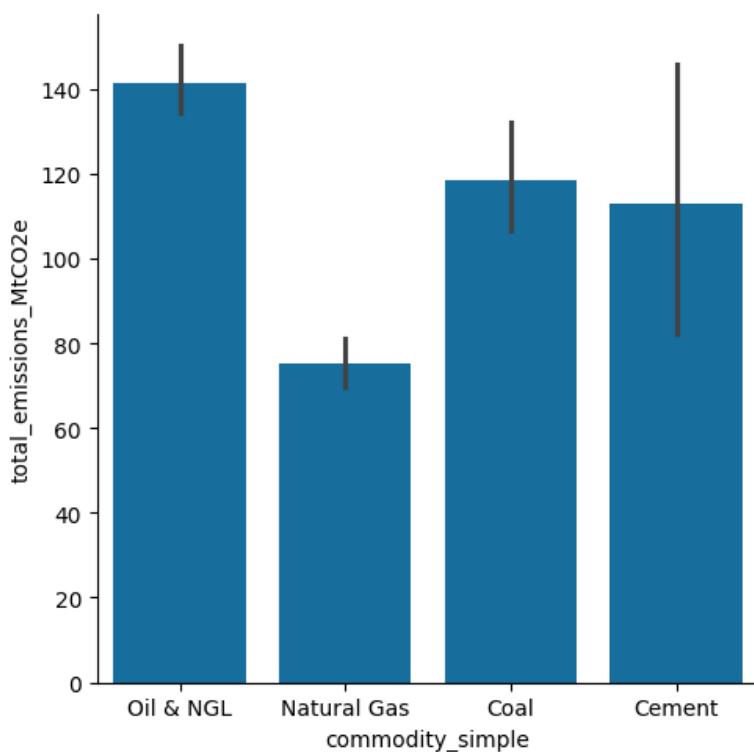
```
sns.catplot(data=carbon_df, x='commodity_simple', y='total_emissions_MtCO2e')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f729a3c6940>
```



```
sns.catplot(data=carbon_df, x='commodity_simple', y='total_emissions_MtCO2e', kind='bar')
```

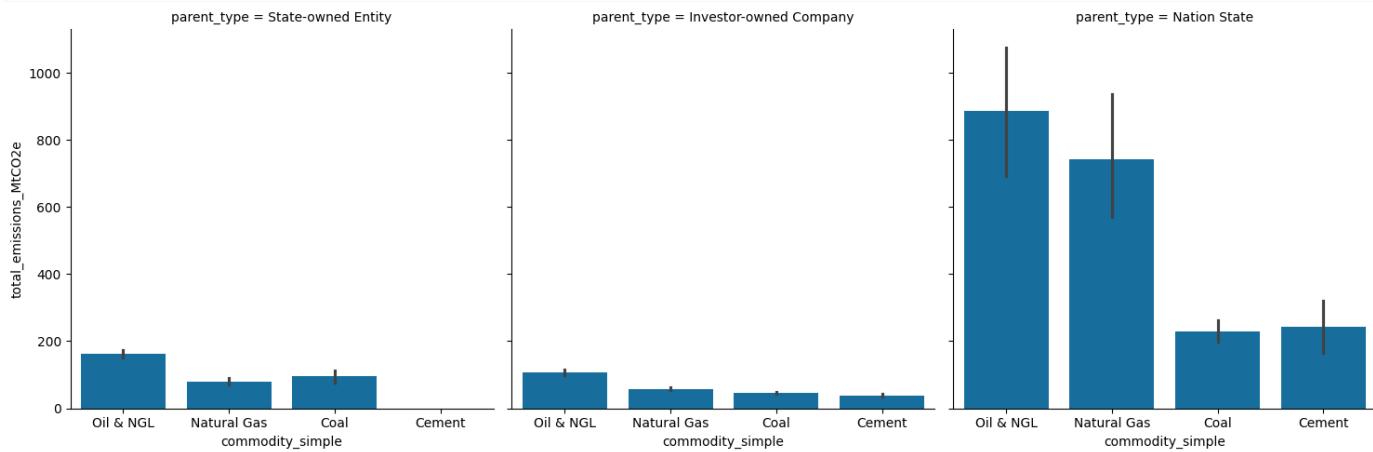
```
<seaborn.axisgrid.FacetGrid at 0x7f729a15b7f0>
```



[Skip to main content](#)

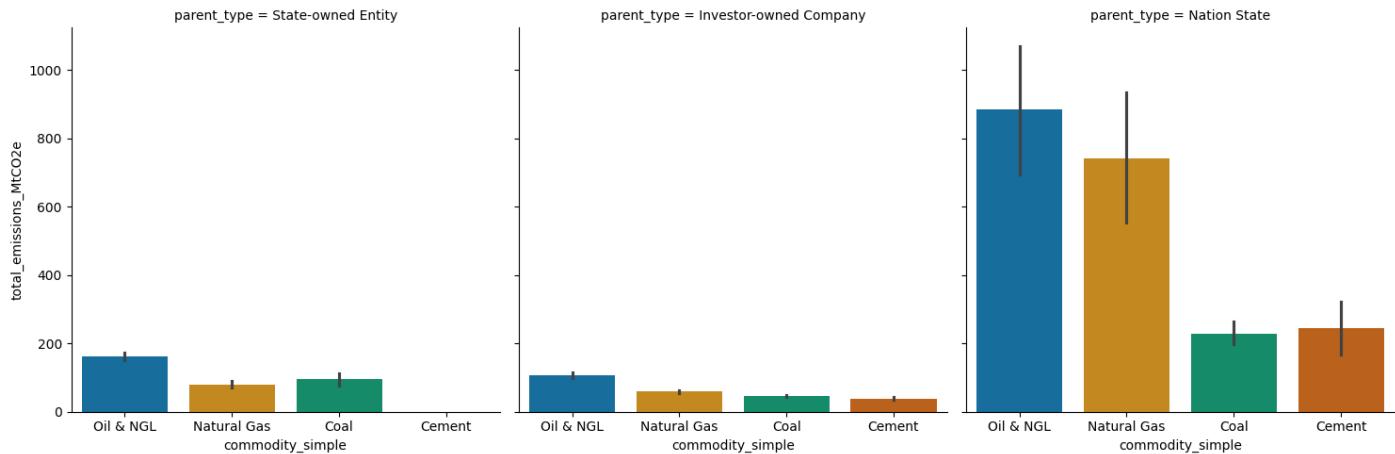
```
sns.catplot(data=carbon_df,x='commodity_simple', y='total_emissions_MtCO2e',kind='bar',
            col = 'parent_type')
```

<seaborn.axisgrid.FacetGrid at 0x7f72e02e79d0>



```
sns.catplot(data=carbon_df,x='commodity_simple', y='total_emissions_MtCO2e',kind='bar',
            col = 'parent_type',hue='commodity_simple')
```

<seaborn.axisgrid.FacetGrid at 0x7f72e02f2af0>



Example okay questions:

- which parent type has the most constant emissions across commodity type?
- which parent type has highest emission?

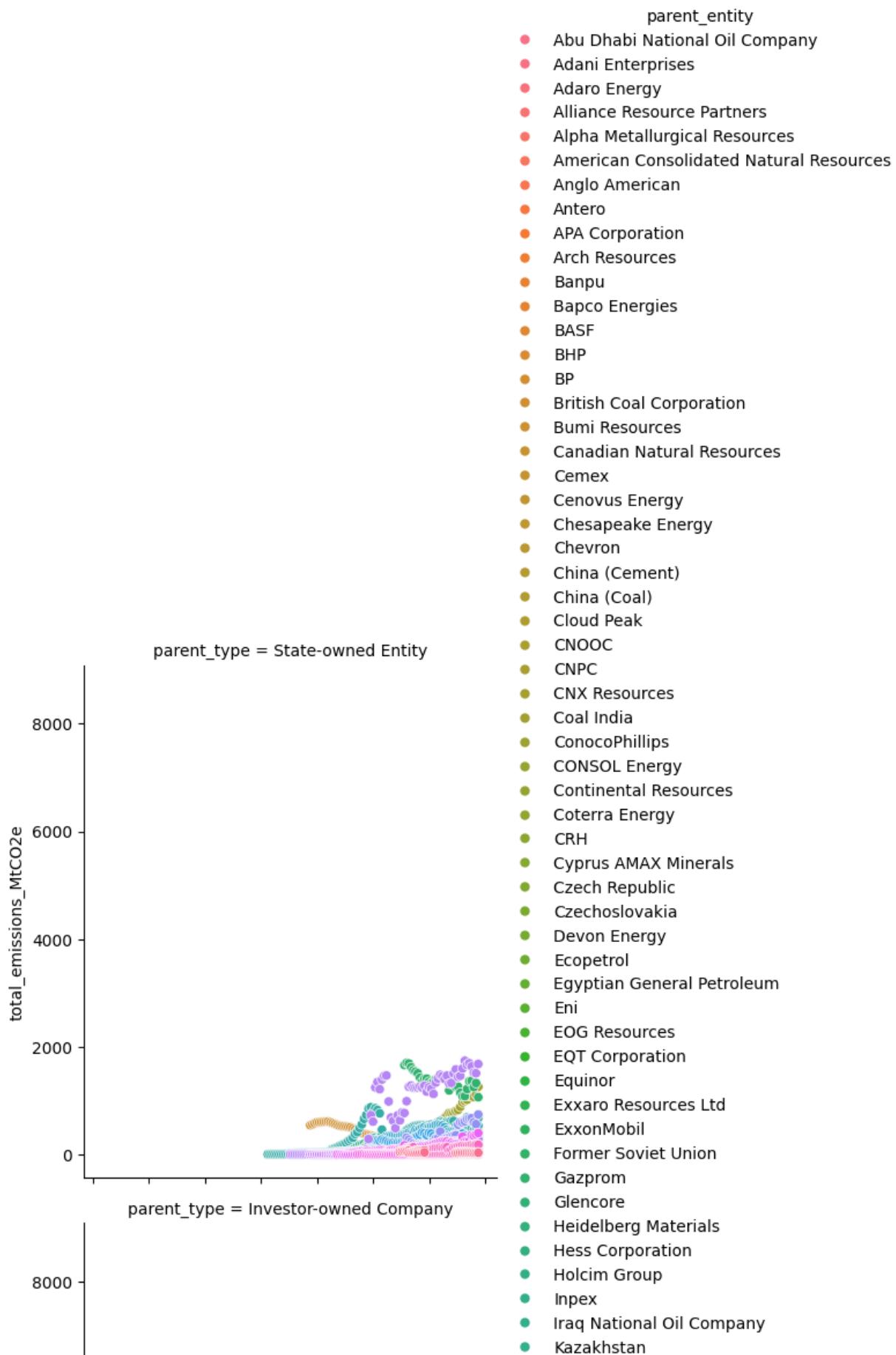
Example good questions

- which type of emissions should be targeted for interventions (the highest)?

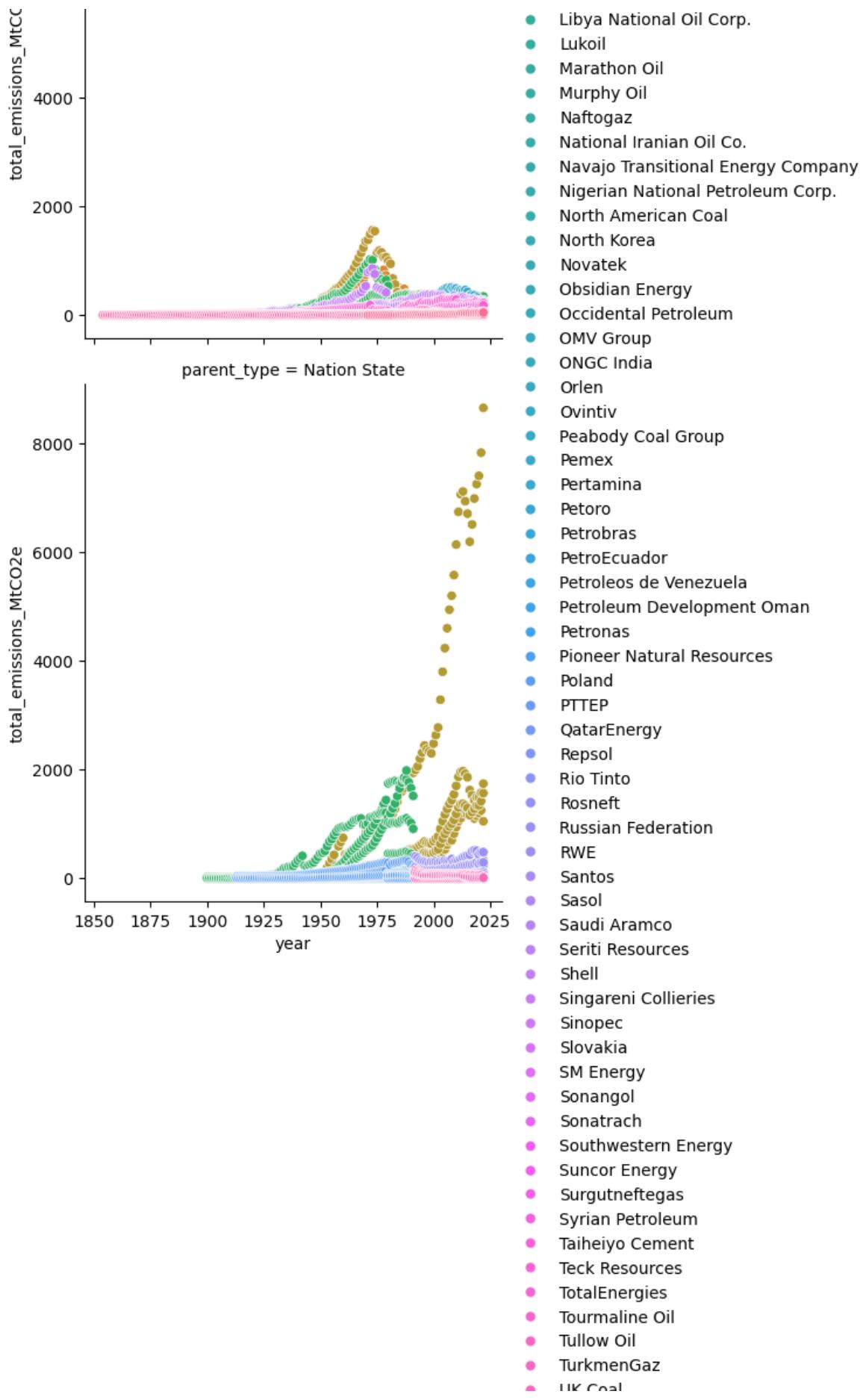
## 5.9. Emissions over time?

```
sns.relplot(data=carbon_df, x='year', y='total_emissions_MtCO2e',
             hue ='parent_entity',row ='parent_type')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f72977f8eb0>
```



Skip to main content



[Skip to main content](#)

- Vistra
- Westmoreland Mining
- Whitehaven Coal
- Wolverine Fuels
- Woodside Energy
- YPF

## 5.10. Variable types and data types

Related but not the same.

Data types are literal, related to the representation in the computer.

ther can be `int16, int32, int64`

We can also have mathematical types of numbers

- Integers can be positive, 0, or negative.
- Reals are continuous, infinite possibilities.

Variable types are about the meaning in a conceptual sense.

- categorical (can take a discrete number of values, could be used to group data, could be a string or integer; unordered)
- continuous (can take on any possible value, always a number)
- binary (like data type boolean, but could be represented as yes/no, true/false, or 1/0, could be categorical also, but often makes sense to calculate rates)
- ordinal (ordered, but appropriately categorical)

we'll focus on the first two most of the time. Some values that are technically only integers range high enough that we treat them more like continuous most of the time.

carbon\_df.columns

```
Index(['year', 'parent_entity', 'parent_type', 'commodity', 'production_value',
       'production_unit', 'total_emissions_MtCO2e', 'commodity_simple'],
      dtype='object')
```

## 5.11. Questions After Class

Class Response Summary:

### 5.11.1. To what degree should we be familiarizing ourselves with these different kinds of graphs?

There is also a full semester data visualization class, so we will not cover *everything* it is useful to know a few basic ones and

## 5.11.2. Should I upload all parts of A2 today if I plan to go to office hours tomorrow? Or just the finished parts?

All of it with your questions written in the file(s).

## 5.11.3. is there ways to overlap the different parent types into the same graph?

This is called a stacked bar graph, there are examples in the seaborn tutorials for [displot](#) but with an important caveat that that can make some things hard to see and you can also [stack with the low level features](#)

## 5.11.4. is the ggplot option just the same method names as the version in R? or is the syntax updated to be similar also?

I think its mostly matching method names, attribute names, and conceptual ideas. Python libraries all have to use Python syntax.

## 5.11.5. Is the peer review just for assignment 3 or will we have the option to do it for future assignments?

Probably only 3, but possibly a couple more.

## 5.11.6. What is the typical range of sizes for a good dataset for this assignment

hundred to maybe 2000, you do not need more than that and too many can make it slow

## 5.11.7. if we don't get any achievements on an assignment are we able to revise them to get an achievement?

If you are very close, yes, if you are not very close, you will get advice that we recommend you apply on future assignments.

## 5.11.8. What is a numpy array?

A DataType that one of the attributes of a DataFrame takes. See the glossary entry for numpy array and the intro to DataFrames

# 6. Cleaning Data - Structure

## 6.1. Intro

This week we'll be cleaning data

[Skip to main content](#)

Cleaning data is **labor intensive** and requires making *subjective* choices.

We'll focus on, and assess you on, manipulating data correctly, making reasonable choices, and documenting the choices you make carefully.

We'll focus on the programming tools that get used in cleaning data in class this week:

- reshaping data
- handling missing or incorrect values
- renaming columns

```
import pandas as pd
import seaborn as sns

# make plots look nicer and increase font size
sns.set_theme(font_scale=2, palette='colorblind')
```

Note here we set the `theme` and pass the palette and font size as parameters. To learn more about this and the various options, see the [seaborn aesthetics tutorial](#)

## 6.2. What is Tidy Data?

Read in the three csv files described below and store them in a dictionary

```
url_base = 'https://raw.githubusercontent.com/rhodyprog4ds/rhodyds/main/data/'

data_filename_list = ['study_a.csv', 'study_b.csv', 'study_c.csv']
```

Here we can use a dictionary comprehension

```
example_data = {datafile.split('.')[0]:pd.read_csv(url_base+datafile) for datafile in data_filename_list}
```

- this is a *dictionary comprehension* which is like a list comprehension, but builds a dictionary instead.
- `split` is a string method that splits the string at the given character and returns a list
- `[0]` then takes the first item out
- `:` makes the part to the left a key and right the value
- we can add strings to combine them

### 6.2.1. Breakign down that comprehension

#### 💡 Tip

This section also illustrates a way you can work to understand **any** piece of code: *take small sections and run them one at a time*

First a tiny dictionary comprehension

```
{char:ord(char) for char in 'abcdef'}
```

```
{'a': 97, 'b': 98, 'c': 99, 'd': 100, 'e': 101, 'f': 102}
```

this gives us the ascii number for each character in that string with the `char` as key and `ord(char)`'s *output* as the value. Note that it *runs* the code assigned in the value because of the `( )` doing the function call.

Next the split method, we will use this sentence.

```
sentence = 'Next the split method, we will use this sentence.'  
type(sentence), len(sentence)
```

```
(str, 49)
```

this is type `str` so we can use the `string class methods` in Python, such as `split`. As a `str` the `len` function tells us how many *characters* because a string is iterable over characters. We used this fact in the tiny dictionary above.

By default it splits at spaces

```
sentence.split()
```

```
['Next', 'the', 'split', 'method', 'we', 'will', 'use', 'this', 'sentence.']}
```

we can instead tell it on what string to split

```
clauses = sentence.split(',')  
type(clauses)
```

```
list
```

`split` always returns a list, so we can pick the first item with `[0]`

```
clauses[0]
```

```
'Next the split method'
```

We can concatenate `str` objects with `+`

```
title = 'Dr.'  
last = 'Brown'  
title + last
```

[Skip to main content](#)

```
'Dr.Brown'
```

## 6.2.2. the same data 3 ways

We can pick a single one out this way

```
example_data['study_a']
```

	name	treatmenta	treatmentb
0	John Smith	-	2
1	Jane Doe	16	11
2	Mary Johnson	3	1

or with an additional import,

```
from IPython.display import display
```

Then we can iterate over the values in the dictionary (the three dataframes) and see them all:

```
[display(df) for df in example_data.values()];
```

	name	treatmenta	treatmentb
0	John Smith	-	2
1	Jane Doe	16	11
2	Mary Johnson	3	1

	intervention	John Smith	Jane Doe	Mary Johnson
0	treatmenta	-	16	3
1	treatmentb	2	11	1

	person	treatment	result
0	John Smith	a	-
1	Jane Doe	a	16
2	Mary Johnson	a	3
3	John Smith	b	2
4	Jane Doe	b	11
5	Mary Johnson	b	1

These three all show the same data, but let's say we have two goals:

[Skip to main content](#)

- find the average effect per treatment across people

This works differently for these three versions.

For `study_a` we can easily get the average per treatment, but to get the average per person, we have to go across rows, which we can do here, but doesn't work as well with plotting

we can work across rows with the `axis` parameter if needed

For B, we get the average per person, but what about per treatment? again we have to go across rows instead.

For the third one, however, we can use groupby, because this one is tidy.

## 6.3. Encoding Missing values

We can see the impact of a bad choice to represent a missing value with `-`

```
example_data['study_c'].dtypes
```

person	object
treatment	object
result	object
dtype:	object

one non numerical value changed the whole column!

```
example_data['study_c'].describe()
```

	person	treatment	result
<b>count</b>	6	6	6
<b>unique</b>	3	2	6
<b>top</b>	John Smith	a	-
<b>freq</b>	2	3	1

so `describe` treats it all like categorical data

We can use the `na_values` parameter to fix this. Pandas recognizes a lot of different values to mean missing and store as `NaN` but this is not one. Find the full list in the `pd.read_csv` documentation of the `na_values` parameter

```
example_data = {datafile.split('.')[0]:pd.read_csv(url_base+datafile,na_values='-' )}
for datafile in data_filename_list}
```

now we can check again

```
example_data['study_c'].dtypes
```

[Skip to main content](#)

```
person      object
treatment    object
result       float64
dtype: object
```

and we see that it is `float` this is because `NaN` is a float.

```
example_data['study_c']
```

	person	treatment	result
0	John Smith	a	NaN
1	Jane Doe	a	16.0
2	Mary Johnson	a	3.0
3	John Smith	b	2.0
4	Jane Doe	b	11.0
5	Mary Johnson	b	1.0

Now it shows as Nan here

## 6.4. Computing on Tidy Data

So we can see how to compute on this data to compute both ways.

```
example_data['study_c'].groupby('person')['result'].mean()
```

```
person
Jane Doe      13.5
John Smith     2.0
Mary Johnson   2.0
Name: result, dtype: float64
```

```
example_data['study_c'].groupby('treatment')['result'].mean()
```

```
treatment
a      9.500000
b      4.666667
Name: result, dtype: float64
```

The original [Tidy Data](#) paper is worth reading to build a deeper understanding of these ideas.

## 6.5. Tidying Data

Let's reshape the first one to match the tidy one. First, we will save it to a DataFrame, this makes things easier to read and enables us to use the built in help in jupyter, because it can't check types too many levels into a data structure.

Before

```
example_data['study_a']
```

	name	treatmenta	treatmentb
0	John Smith	NaN	2
1	Jane Doe	16.0	11
2	Mary Johnson	3.0	1

After

```
df_a = example_data['study_a']
```

When we melt a dataset:

- the `id_vars` stay as columns
- the data from the `value_vars` columns become the values in the `value` column
- the column names from the `value_vars` become the values in the `variable` column
- we can rename the value and the variable columns.

```
tall_a = df_a.melt(id_vars='name', var_name='treatment', value_name='result')  
tall_a
```

	name	treatment	result
0	John Smith	treatmenta	NaN
1	Jane Doe	treatmenta	16.0
2	Mary Johnson	treatmenta	3.0
3	John Smith	treatmentb	2.0
4	Jane Doe	treatmentb	11.0
5	Mary Johnson	treatmentb	1.0

see visualized on pandas tutor

## 7. Transforming the Coffee data

[Skip to main content](#)

For example, let's only keep the data from the top 10 countries in terms of number of bags.

```
arabica_data_url = 'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/arabica.csv'
# load the data
coffee_df = pd.read_csv(arabica_data_url)
coffee_df.head()
```

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Co
0	1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015 agr devt
1	2	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015 agr devt
2	3	Arabica	grounds for health admin	Guatemala	san marcos barrancas "san cristobal cuch	NaN	NaN	NaN
3	4	Arabica	yidnekachew dabessa	Ethiopia	yidnekachew dabessa coffee plantation	NaN	wolensu	yidnekachew dabessa coffee plantation
4	5	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015 agr devt

5 rows × 44 columns

We have separate countries for the country and number of bags

So we make a table the totals how many bags per country

```
bags_per_country = coffee_df.groupby('Country.of.Origin')['Number.of.Bags'].sum()
bags_per_country.head(15)
```

```
Country.of.Origin
Brazil      30534
Burundi     520
China       55
Colombia    41204
Costa Rica   10354
Cote d'Ivoire  2
Ecuador     1
El Salvador  4449
Ethiopia     11761
Guatemala   36868
Haiti        390
Honduras    13167
India        20
Indonesia   1658
Japan        20
Name: Number.of.Bags, dtype: int64
```

Then we can take only the highest 10, and keep only the index

```
# sort descending, keep only the top 10 and pick out only the country names
top_bags_country_list = bags_per_country.sort_values(ascending=False)[:10].index

# filter the original data for only the countries in the top list
top_coffee_df = coffee_df[coffee_df['Country.of.Origin'].isin(top_bags_country_list)]
```

we can see which countries are in the list

```
top_bags_country_list
```

```
Index(['Colombia', 'Guatemala', 'Brazil', 'Mexico', 'Honduras', 'Ethiopia',
       'Costa Rica', 'Nicaragua', 'El Salvador', 'Kenya'],
      dtype='object', name='Country.of.Origin')
```

and confirm that is all that is in our new dataframe

```
top_coffee_df['Country.of.Origin'].value_counts()
```

```
Country.of.Origin
Mexico      236
Colombia    183
Guatemala  181
Brazil      132
Honduras    53
Costa Rica   51
Ethiopia     44
Nicaragua   26
Kenya       25
El Salvador  21
Name: count, dtype: int64
```

compared to the original, it is far fewer

```
coffee_df['Country.of-Origin'].value_counts()
```

Country.of.Origin	
Mexico	236
Colombia	183
Guatemala	181
Brazil	132
Taiwan	75
United States (Hawaii)	73
Honduras	53
Costa Rica	51
Ethiopia	44
Tanzania, United Republic Of	40
Thailand	32
Uganda	26
Nicaragua	26
Kenya	25
El Salvador	21
Indonesia	20
China	16
Malawi	11
Peru	10

### ⚠ Warning

This section is changed slightly from in class to be easier to read.

This limits the rows, but keeps all of the columns

```
top_coffee_df.columns
```

```
Index(['Unnamed: 0', 'Species', 'Owner', 'Country.of.Origin', 'Farm.Name',
       'Lot.Number', 'Mill', 'ICO.Number', 'Company', 'Altitude', 'Region',
       'Producer', 'Number.of.Bags', 'Bag.Weight', 'In.Country.Partner',
       'Harvest.Year', 'Grading.Date', 'Owner.1', 'Variety',
       'Processing.Method', 'Aroma', 'Flavor', 'Aftertaste', 'Acidity', 'Body',
       'Balance', 'Uniformity', 'Clean.Cup', 'Sweetness', 'Cupper.Points',
       'Total.Cup.Points', 'Moisture', 'Category.One.Defects', 'Quakers',
       'Color', 'Category.Two.Defects', 'Expiration', 'Certification.Body',
       'Certification.Address', 'Certification.Contact', 'unit_of_measurement',
       'altitude_low_meters', 'altitude_high_meters', 'altitude_mean_meters'],
      dtype='object')
```

Since this has a lot of variables, we will make lists of how we want to use different columns in variables before using.

```
value_vars = ['Aroma', 'Flavor', 'Aftertaste', 'Acidity', 'Body',
              'Balance', 'Uniformity', 'Clean.Cup', 'Sweetness', ]
id_vars = ['Species', 'Owner', 'Country.of.Origin']
coffee_tall = top_coffee_df.melt(id_vars=id_vars, value_vars=value_vars,
                                 var_name='rating')
coffee_tall.head(1)
```

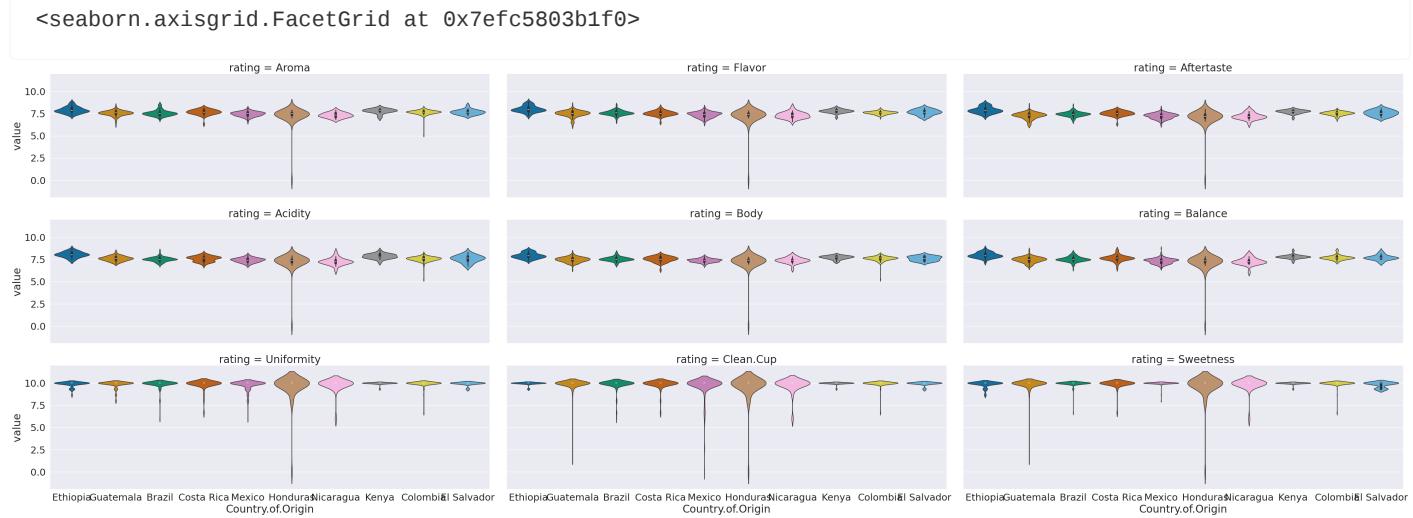
Species	Owner	Country.of.Origin	rating	value
---------	-------	-------------------	--------	-------

[Skip to main content](#)

This data is “tidy” all of the different measurements (ratings) are different rows and we have a few columns that identify each sample

This version plays much better with plots where we want to compare the ratings.

```
sns.catplot(data = coffee_tall, col='rating', y='value',x='Country.of.Origin',  
hue='Country.of.Origin',aspect=3, col_wrap =3, kind='violin')
```



This now we can use the `rating` as a variable to break the data by in our subplots. I did a few new things in this:

- a **violin plot** allows us to compare how different data is distributed.
- I used both `col` and `col_wrap`, `col` alone would have made nine columns, and without `row` all in one row. `col_wrap` says to line wrap after a certain number of columns (here 3)
- `aspect` is an attribute of the **figure level plots** that controls the ratio of width to height (the aspect ratio). so `aspect=3` makes it 3 times as wide as tall in each subplot, this spreads out the x tick labels so can read them

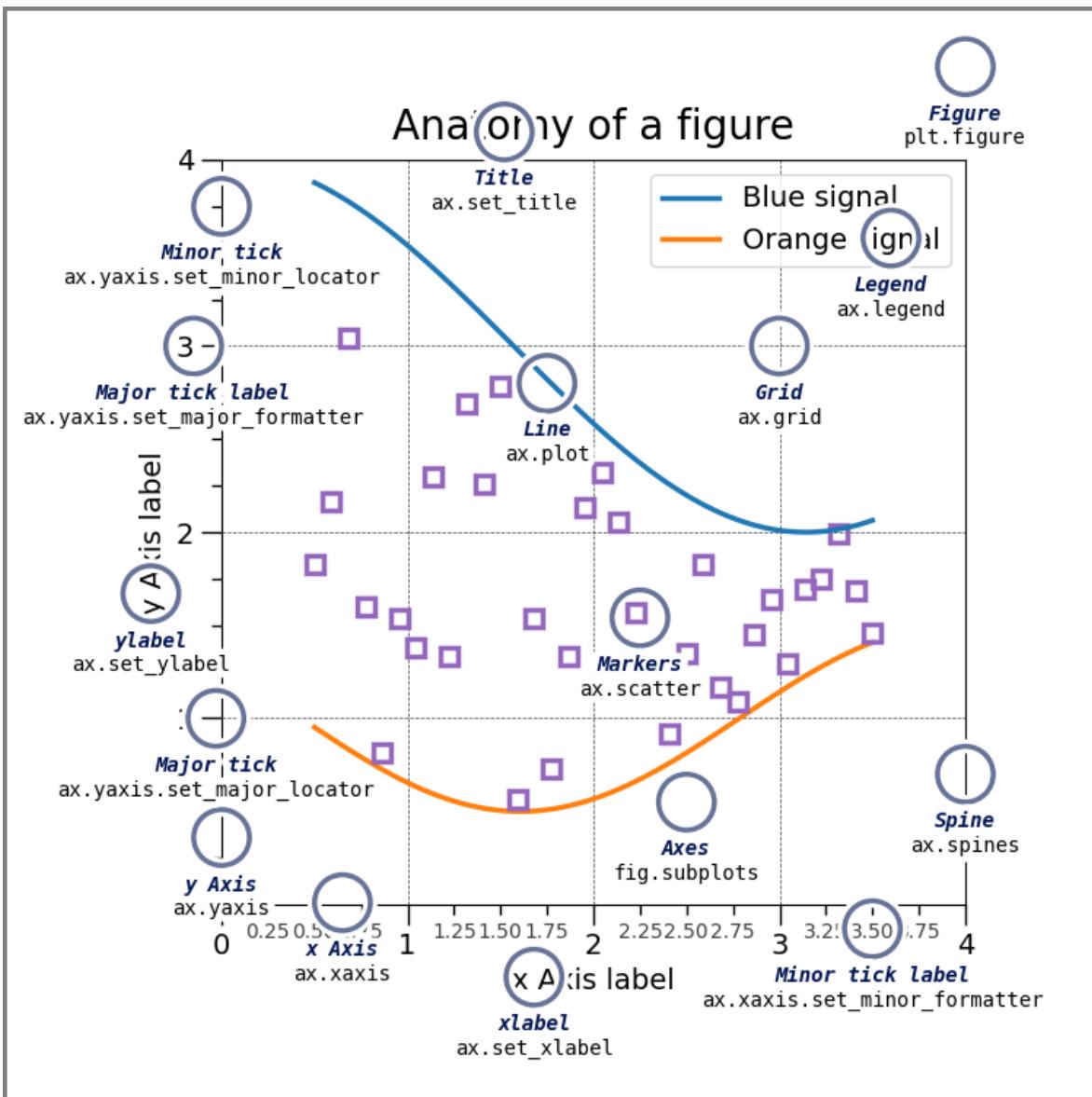
## 7.1. Questions After Class

Most of the questions today were about assignment 3 or asking to explain specific things that are above. Feel free to post an issue if a question comes up later.

### 7.1.1. Can you change the locations of the names next to a plot?

First, see the annotated graph and learn the technical name of the element you want to move.

Tip  
Copy out  
before what



the above figure come from [matplotlib's Anatomy of a Figure page](#) which includes the `code` to generate that figure

You can control each of these but you may need to use [matplotlib](#).

If this means the legend, seaborn can control that location. If it refers to fixing overlapping tick labels, I demonstrated that above, with [aspect](#).

## 8. Fixing Values

so far, we've dealt with structural issues in data. but there's a lot more to cleaning.

Today, we'll deal with how to fix the values within the data.

### 8.1. Cleaning Data review

Instead of more practice with these manipulations, below are more examples of cleaning data to see how these types of

[Skip to main content](#)

Your goal here is not to memorize every possible thing, but to build a general idea of what good data looks like and good habits for cleaning data and keeping it reproducible.

- Cleaning the Adult Dataset
- All Shades Also here are some tips on general data management and organization.

This article is a comprehensive discussion of data cleaning.

### 8.1.1. A Cleaning Data Recipe

**not everything possible, but good enough for this course**

1. Can you use parameters to read the data in better?
2. Fix the index and column headers (making these easier to use makes the rest easier)
3. Is the data structured well?
4. Are there missing values?
5. Do the datatypes match what you expect by looking at the head or a sample?
6. Are categorical variables represented in usable way?
7. Does your analysis require filtering or augmenting the data?

## 8.2. What is clean enough?

This is a great question, without an easy answer.

It depends on what you want to do. This is why it's important to have potential questions in mind if you are cleaning data for others *and* why we often have to do a little bit more preparation after a dataset has been "cleaned"

Dealing with missing data is a whole research area. There isn't one solution.

in 2020 there was a whole workshop on missing

one organizer is the main developer of sci-kit learn the ML package we will use soon. In a 2020 invited talk he listed more automatic handling as an active area of research and a development goal for sklearn.

There are also many classic approaches both when training and when applying models.

example application in breast cancer detection

Even in pandas, dealing with missing values is under experimentation as to how to represent it symbolically

Missing values even causes the datatypes to change

That said, there are a few basic tools:

- dropna
- fillna

- you can approximate with another column
- you can approximate with that column from other rows

Special case, what if we're filling a summary table?

- filling with a symbol for printing can be a good choice, but not for analysis.

**whatever you do, document it**

```
import pandas as pd
import seaborn as sns
import numpy as np
na_toy_df = pd.DataFrame(data = [[1,3,4,5],[2 ,6, np.nan],[np.nan]*4,[np.nan,3,4,5]], columns=['a','b','c'])

# make plots look nicer and increase font size
sns.set_theme(font_scale=2)

# todays data
arabica_data_url = 'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/arabica.csv'

coffee_df = pd.read_csv(arabica_data_url)
```

## 8.3. Missing values

We tend to store missing values as `NaN` or use the constants:

`pd.NA, np.nan`

`(<NA>, nan)`

Pandas makes that a special typed object, but converts the whole *column* to float

Numpy uses *float* value for `NaN` that is defined by IEEE floating point standard

`type(pd.NA), type(np.nan)`

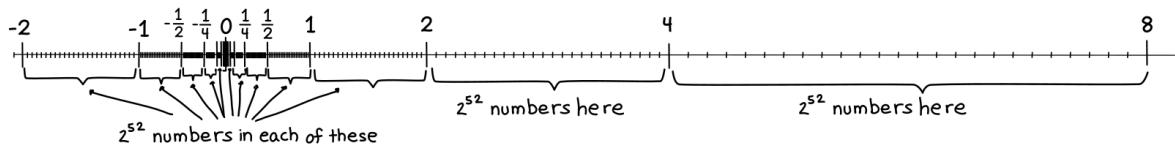
`(pandas._libs.missing.NAType, float)`

# the floating point number line

## the (64-bit) floating point number line

Floating point numbers aren't evenly distributed. Instead, they're organized into windows:  $[0.25, 0.5]$ ,  $[0.5, 1]$ ,  $[1, 2]$ ,  $[2, 4]$ ,  $[4, 8]$ ,  $[8, 16]$ , all the way up to  $[2^{1023}, 2^{1024}]$ .

Every window has  $2^{52}$  floats in it.



the windows go from  
REALLY small to REALLY big

The window closest to 0 is  $[2^{-1023}, 2^{-1022}]$ . This is TINY: a hydrogen atom weighs about  $2^{-76}$  grams. The biggest window is  $[2^{1023}, 2^{1024}]$ . This is HUUUGE: the farthest galaxy we know about is about  $2^{40}$  meters away.

the gaps between floats double with every window

window	gap	1	2
$[1, 2]$	$2^{-52}$	1	2
$[2, 4]$	$2^{-51}$	2	4
$[4, 8]$	$2^{-50}$	4	8
$[8, 16]$	$2^{-49}$	8	

why does  $1000000000000000.0 + 1 = 1000000000000000.0$ ?

- In the window  $[2^n, 2^{n+1}]$ , the gap between floats is  $2^{n-52}$
- $1000000000000000.0$  is in the window  $[2^{53}, 2^{54}]$ , where the gap is  $2^1$  (or 2)
- So the next float after  $1000000000000000.0$  is  $1000000000000002.0$

there are values that cannot be represented.

- see `float.exposed` for infinity
- negative infinity
- see a `nan` (which bits can be changed without making it *not* nan)
- from `9007199254740992.0` the next closest value is `9007199254740994.0`... no values in between can be stored in double precision float

We can see a few in this toy dataset

This data has some missing values

`na_toy_df`

	a	b	c	d
0	1.0	3.0	4.0	5.0
1	2.0	6.0	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	3.0	4.0	5.0

Let's try the default behavior of `dropna`

Skip to main content

```
na_toy_df.dropna()
```

	a	b	c	d
0	1.0	3.0	4.0	5.0

This is the same as

```
na_toy_df.dropna(how='any', subset=na_toy_df.columns, axis=0)
```

	a	b	c	d
0	1.0	3.0	4.0	5.0

by default it drops all of the *rows* where **any** of the elements are missing (1 or more) we can change `how` to its other mode:

```
na_toy_df.dropna(how='all')
```

	a	b	c	d
0	1.0	3.0	4.0	5.0
1	2.0	6.0	NaN	NaN
3	NaN	3.0	4.0	5.0

in `'all'` mode it only drops rows where **all** of the values are missing

we can also change it to work along columns (`axis=1`) instead

```
na_toy_df.dropna(how='all', axis=1)
```

	a	b	c	d
0	1.0	3.0	4.0	5.0
1	2.0	6.0	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	3.0	4.0	5.0

None of the columns are *all* missing so nothing is dropped

Let's say we had an analysis where we needed at least one of column `c` or `d` or else we could not use the row, we can check that this way:

```
na_toy_df.dropna(how='all', subset=['c', 'd'])
```

```
Cell In[9], line 1
na_toy_df.dropna(how='all',subset=['c','d'])^
SyntaxError: unexpected EOF while parsing
```

### 8.3.1. Filling missing values

Let's look at a real dataset now

```
coffee_df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1311 entries, 0 to 1310
Data columns (total 44 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Unnamed: 0          1311 non-null   int64  
 1   Species             1311 non-null   object  
 2   Owner               1304 non-null   object  
 3   Country.of-Origin  1310 non-null   object  
 4   Farm.Name           955 non-null   object  
 5   Lot.Number          270 non-null   object  
 6   Mill                1001 non-null   object  
 7   ICO.Number          1163 non-null   object  
 8   Company              1102 non-null   object  
 9   Altitude             1088 non-null   object  
 10  Region              1254 non-null   object  
 11  Producer             1081 non-null   object  
 12  Number.of.Bags      1311 non-null   int64  
 13  Bag.Weight          1311 non-null   object  
 14  In.Country.Partner  1311 non-null   object  
 15  Harvest.Year         1264 non-null   object  
 16  Grading.Date         1311 non-null   object  
 17  Owner.1              1304 non-null   object  
 18  Variety              1110 non-null   object  
 19  Processing.Method    1159 non-null   object  
 20  Aroma                1311 non-null   float64 
 21  Flavor               1311 non-null   float64 
 22  Aftertaste            1311 non-null   float64 
 23  Acidity              1311 non-null   float64 
 24  Body                 1311 non-null   float64 
 25  Balance               1311 non-null   float64 
 26  Uniformity            1311 non-null   float64 
 27  Clean.Cup             1311 non-null   float64 
 28  Sweetness              1311 non-null   float64 
 29  Cupper.Points          1311 non-null   float64 
 30  Total.Cup.Points       1311 non-null   float64 
 31  Moisture              1311 non-null   float64 
 32  Category.One.Defects  1311 non-null   int64  
 33  Quakers              1310 non-null   float64 
 34  Color                 1044 non-null   object  
 35  Category.Two.Defects  1311 non-null   int64  
 36  Expiration             1311 non-null   object  
 37  Certification.Body      1311 non-null   object  
 38  Certification.Address   1311 non-null   object  
 39  Certification.Contact   1311 non-null   object  
 40  unit_of_measurement     1311 non-null   object  
 41  altitude_low_meters    1084 non-null   float64 
 42  altitude_high_meters   1084 non-null   float64 
 43  altitude_mean_meters   1084 non-null   float64 
dtypes: float64(16), int64(4), object(24)
memory usage: 450.8+ KB

```

The 'Lot.Number' has a lot of NaN values, how can we explore it?

We can look at the type:

```
coffee_df['Lot.Number'].dtype
```

```
dtype('O')
```

```
coffee_df['Lot.Number'].value_counts()
```

```
Lot.Number
1                 18
020/17            6
019/17            5
2                 3
102               3
..
11/23/0696        1
3-59-2318         1
8885              1
5055              1
017-053-0211/ 017-053-0212  1
Name: count, Length: 221, dtype: int64
```

We see that a lot are '1', maybe we know that when the data was collected, if the Farm only has one lot, some people recorded '1' and others left it as missing. So we could fill in with 1:

```
coffee_df['Lot.Number'].fillna('1').head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: Lot.Number, dtype: object
```

```
coffee_df['Lot.Number'].head()
```

```
0    NaN
1    NaN
2    NaN
3    NaN
4    NaN
Name: Lot.Number, dtype: object
```

Note that even after we called `fillna` we display it again and the original data is unchanged. To save the filled in column, *technically* we have a few choices:

- use the `inplace` parameter. This doesn't offer performance advantages, but does it still copies the object, but then reassigns the pointer. Its under discussion to deprecate
- write to a new DataFrame
- add a column

we will add a column

```
coffee_df['lot_number_clean'] = coffee_df['Lot.Number'].fillna('1')
```

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company
0	1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015

1 rows × 45 columns

```
coffee_df.shape
```

```
(1311, 45)
```

## 8.4. Dropping

Dropping is a good choice when you otherwise have a lot of data and the data is missing at random.

Dropping can be risky if it's not missing at random. For example, if we saw in the coffee data that one of the scores was missing for all of the rows from one country, or even just missing more often in one country, that could bias our results.

here will focus on how this impacts how much data we have:

```
coffee_df.dropna().shape
```

```
(130, 45)
```

we lose a lot this way.

We could instead tell it to only drop rows with `NaN` in a subset of the columns.

```
coffee_df.dropna(subset=['altitude_low_meters']).shape
```

```
(1084, 45)
```

Now, it drops any row with one or more `NaN` values in that column.

In the [Open Policing Project Data Summary](#) we saw that they made a summary information that showed which variables had at least 70% not missing values. We can similarly choose to keep only variables that have more than a specific threshold of data, using the `thresh` parameter and `axis=1` to drop along columns.

```
n_rows, _ = coffee_df.shape
coffee_df.dropna(thresh=.7*n_rows, axis=1).shape
```

```
...
```

[Skip to main content](#)

## 8.5. Inconsistent values

This was one of the things that many of you anticipated or had observed. A useful way to investigate for this, is to use `value_counts` and sort them alphabetically by the values from the original data, so that similar ones will be consecutive in the list. Once we have the `value_counts()` Series, the values from the `coffee_df` become the index, so we use `sort_index`.

Let's look at the `in_country_partner` column

```
coffee_df['In.Country.Partner'].value_counts().sort_index()
```

In.Country.Partner	
AMECAFE	205
Africa Fine Coffee Association	49
Almacafé	178
Asociación Nacional Del Café	155
Asociación Mexicana De Cafés y Cafeterías De Especialidad A.C.	6
Asociación de Cafés Especiales de Nicaragua	8
Blossom Valley International	58
Blossom Valley International\n	1
Brazil Specialty Coffee Association	67
Central De Organizaciones Productoras De Café y Cacao Del Perú - Central Café & Cacao	1
Centro Agroecológico del Café A.C.	8
Coffee Quality Institute	7
Ethiopia Commodity Exchange	18
Instituto Hondureño del Café	60
Kenya Coffee Traders Association	22
METAD Agricultural Development plc	15
NUCOFFEE	36
Salvadoran Coffee Council	11
Specialty Coffee Ass	1

We can see there's only one `Blossom Valley International\n` but 58 `Blossom Valley International`, the former is likely a typo, especially since `\n` is a special character for a newline. Similarly, with 'Specialty Coffee Ass' and 'Specialty Coffee Association'.

```
partner_corrections = {'Blossom Valley International\n':'Blossom Valley International',
 'Specialty Coffee Ass':'Specialty Coffee Association'}
coffee_df['in_country_partner_clean'] = coffee_df['In.Country.Partner'].replace(
    to_replace=partner_corrections)
coffee_df['in_country_partner_clean'].value_counts().sort_index()
```

in_country_partner_clean	
AMECAFE	205
Africa Fine Coffee Association	49
Almacafé	178
Asociacion Nacional Del Café	155
Asociación Mexicana De Cafés y Cafeterías De Especialidad A.C.	6
Asociación de Cafés Especiales de Nicaragua	8
Blossom Valley International	59
Brazil Specialty Coffee Association	67
Central De Organizaciones Productoras De Café y Cacao Del Perú - Central Café & Cacao	1
Centro Agroecológico del Café A.C.	8
Coffee Quality Institute	7
Ethiopia Commodity Exchange	18
Instituto Hondureño del Café	60
Kenya Coffee Traders Association	22
METAD Agricultural Development plc	15
NUCOFFEE	36
Salvadoran Coffee Council	11
Specialty Coffee Association	296
Specialty Coffee Association of Costa Rica	42

## 8.6. Multiple values in a single column

Let's look at the column about the bag weights

```
coffee_df['Bag.Weight'].head()
```

```
0    60 kg
1    60 kg
2      1
3    60 kg
4    60 kg
Name: Bag.Weight, dtype: object
```

it has both the *value* and the *units* in a single column, which is not what we want.

This would be better in two separate columns

```
bag_df = coffee_df['Bag.Weight'].str.split(' ').apply(pd.Series).rename({0:'bag_weight_clean',
                                                               1:'bag_weight_unit'},
                                                               axis=1)

bag_df.head()
```

	bag_weight_clean	bag_weight_unit
0	60	kg
1	60	kg
2	1	NaN
3	60	kg
4	60	kg

- picks the column
- treats it as a string with the pandas Series attribute `.str`
- uses base python `str.split` to split at `' '` spaces and makes a list
- casts each list to Series with `.apply(pd.Series)`
- renames the resulting columns from being numbered to usable names `rename({0: 'bag_weight_clean', 1: 'bag_weight_unit'}, axis=1)`

### Tip

The `.apply(pd.Series)` works on dictionaries too (anything the series constructor can take to its `data` parameter) so this is good for json data

The following subsections break down the casting and string methods in more detail

## 8.6.1. String methods

Python has a powerful `string` class. There is also an even more powerful `string` module

we only need the base `str` methods most of the time

```
example_str = 'kjksfjds sklfjsdl'
type(example_str)
```

`str`

Some helpful ones:

```
example_str.split()
```

```
['kjksfjds', 'sklfjsdl']
```

this gives a list

you can also change the separator

```
'phrases-with-hyphens'.split('-')
```

```
['phrases', 'with', 'hyphens']
```

there are also methods for changing the case and other similar things. \*Use these instead of implementing your own string operations!!

```
('KJKSFJDS SKLFJSSDL', 'Kjksfjds sklfjsdl')
```

## 8.6.2. Casting Review

If we have a variable that is not the type we want like this:

```
a='5'
```

we can check type

```
type(a)
```

```
str
```

and we can use the name of the type we want, as a function to *cast* it to the new type.

```
int(5)
```

```
5
```

and check

```
type(int(a))
```

```
int
```

## 8.7. Combining parts of dataframes

```
bag_df.head()
```

	bag_weight_clean	bag_weight_unit
0	60	kg
1	60	kg
2	1	NaN
3	60	kg
4	60	kg

```
pd.concat([coffee_df,bag_df],axis=1)
```

	Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number
0	1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015
1	2	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015
2	3	Arabica	grounds for health admin	Guatemala	san marcos barrancas "san cristobal cuch	NaN	NaN	NaN
3	4	Arabica	yidnekachew dabessa	Ethiopia	yidnekachew dabessa coffee plantation	NaN	wolensu	NaN
4	5	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015
...	...	...	...	...	...	...	...	...
1306	1307	Arabica	juan carlos garcia lopez	Mexico	el centenario	NaN	la esperanza, municipio juchique de ferrer, ve...	1104328663
1307	1308	Arabica	myriam kaplan- pasternak	Haiti	200 farms	NaN	coeb koperativ ekselsyo basen (350 members)	NaN
1308	1309	Arabica	exportadora atlantic, s.a.	Nicaragua	finca las marías	017-053- 0211/ 017- 053-0212	beneficio atlantic condega	017-053- 0211/ 017- 053-0212
1309	1310	Arabica	juan luis alvarado romero	Guatemala	finca el limon	NaN	beneficio serben	11/853/165
1310	1312	Arabica	bismarck castro	Honduras	los hicaques	103	cigrah s.a de c.v.	13-111-053

1311 rows × 48 columns

## 8.8. Quantizing a variable

Sometimes a variable is recorded continuous, or close (like age in years, technically integers are discrete, but for wide enough

[Skip to main content](#)

We can add a new variable that is *calculated* from the original one.

Let's say we want to categorize coffees as small, medium or large batch size based on the quantiles for the `'Number.of.Bags'` column.

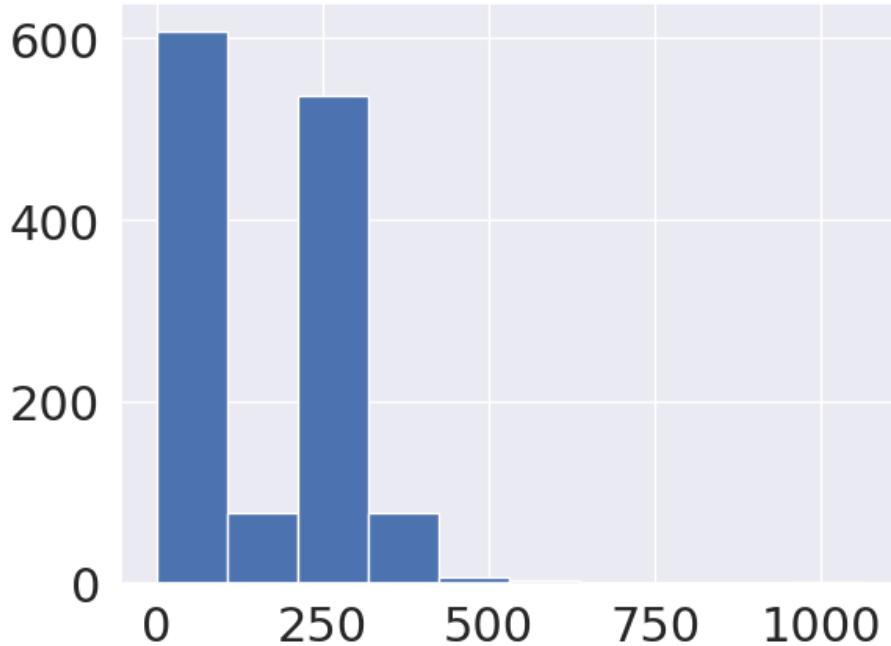
First, we get an idea of the distribution with EDA to make our plan:

```
coffee_df_bags = pd.concat([coffee_df, bag_df], axis=1)
coffee_df_bags['Number.of.Bags'].describe()
```

```
count    1311.000000
mean     153.887872
std      129.733734
min      0.000000
25%     14.500000
50%     175.000000
75%     275.000000
max     1062.000000
Name: Number.of.Bags, dtype: float64
```

```
coffee_df_bags['Number.of.Bags'].hist()
```

```
<Axes: >
```



We see that most are small, but there is at least one major outlier, 75% are below 275, but the max is 1062.

We can use `pd.cut` to make discrete values

```
pd.cut(coffee_df_bags['Number.of.Bags'], bins=3).sample(10)
```

```
1003    (-1.062, 354.0]
840     (-1.062, 354.0]
349     (-1.062, 354.0]
531     (-1.062, 354.0]
1114    (354.0, 708.0]
499     (-1.062, 354.0]
324     (-1.062, 354.0]
1007    (-1.062, 354.0]
995     (-1.062, 354.0]
380     (-1.062, 354.0]
Name: Number.of.Bags, dtype: category
Categories (3, interval[float64, right]): [(-1.062, 354.0] < (354.0, 708.0] < (708.0, 1062.0]]
```

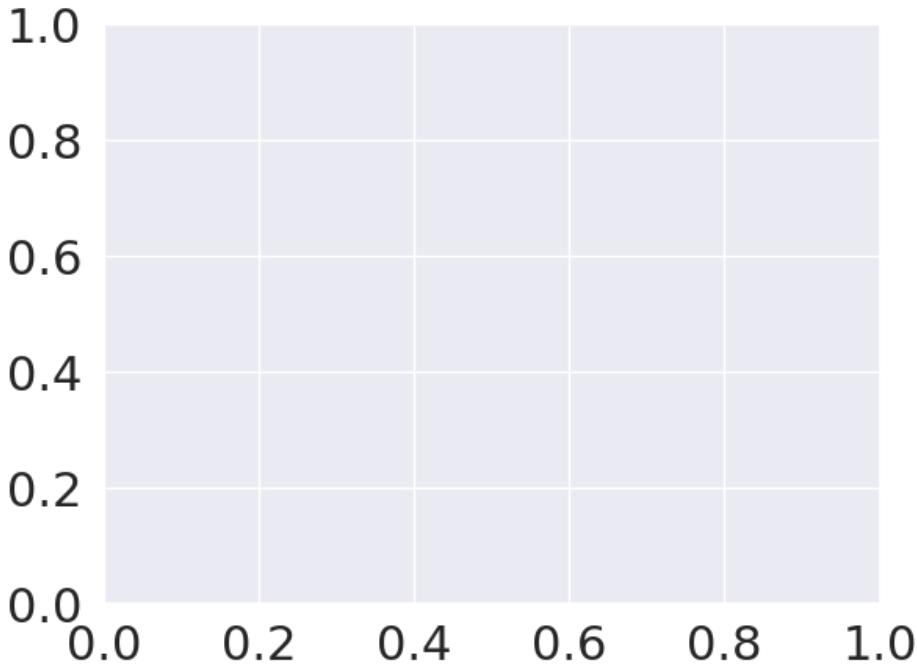
by default, it makes bins of equal size, meaning the range of values. This is not good based on what we noted above. Most will be in one label

### Note

I would like to show a histogram here, but for somereason it broke. The output is hidden for now.

► Show code cell source

```
-----  
TypeError Traceback (most recent call last)  
Cell In[38], line 1  
----> 1 pd.cut(coffee_df_bags['Number.of.Bags'],bins=3).hist()  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/plotting/_core.py:99, in hist(self, bins=10, range=None, density=False, cumulative=False, bottom=False, right=False, align='left', grid=True, log=False, **kwds)  
    50     """  
    51     Draw histogram of the input series using matplotlib.  
    52  
    (...)  
    96     matplotlib.axes.Axes.hist : Plot a histogram using matplotlib.  
    97     """  
    98     plot_backend = _get_plot_backend(backend)  
--> 99     return plot_backend.hist_series(  
    100         self,  
    101         by=by,  
    102         ax=ax,  
    103         grid=grid,  
    104         xlabelsize=xlabelsize,  
    105         xrot=xrot,  
    106         ylabelsize=ylabelsize,  
    107         yrot=yrot,  
    108         figsize=figsize,  
    109         bins=bins,  
    110         legend=legend,  
    111         **kwargs,  
    112     )  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/plotting/_matplotlib/hist.py:423, in hist(self, bins=10, density=False, bottom=False, right=False, align='left', grid=True, log=False, **kwds)  
    423     if legend:  
    424         kwds["label"] = self.name  
--> 425     ax.hist(values, bins=bins, **kwds)  
    426     if legend:  
    427         ax.legend()  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/matplotlib/__init__.py:1446, in __call__(self, func, *args, **kwargs)  
    1443     @functools.wraps(func)  
    1444     def inner(ax, *args, data=None, **kwargs):  
    1445         if data is None:  
-> 1446             return func(ax, *map(sanitize_sequence, args), **kwargs)  
    1447         bound = new_sig.bind(ax, *args, **kwargs)  
    1448         auto_label = (bound.arguments.get(label_namer)  
    1449                         or bound.kwargs.get(label_namer))  
    1450  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/matplotlib/axes/_axes.py:6763, in _get_x_min_max(self, x)  
    6759     for xi in x:  
    6760         if len(xi):  
    6761             # python's min/max ignore nan,  
    6762             # np.nanmin returns nan for all nan input  
-> 6763             xmin = min(xmin, np.nanmin(xi))  
    6764             xmax = max(xmax, np.nanmax(xi))  
    6765     if xmin <= xmax: # Only happens if we have seen a finite value.  
  
TypeError: '<' not supported between instances of 'pandas._libs.interval.Interval' and 'float'
```



To make it better, we can specify the bin edges instead of only the number

```
min_bags = coffee_df_bags['Number.of.Bags'].min()
sm_cutoff = coffee_df_bags['Number.of.Bags'].quantile(.33)
md_cutoff = coffee_df_bags['Number.of.Bags'].quantile(.66)
max_bags = coffee_df_bags['Number.of.Bags'].max()
pd.cut(coffee_df_bags['Number.of.Bags'],
       bins=[min_bags, sm_cutoff, md_cutoff, max_bags]).head()
```

```
0    (250.0, 1062.0]
1    (250.0, 1062.0]
2    (0.0, 28.0]
3    (250.0, 1062.0]
4    (250.0, 1062.0]
Name: Number.of.Bags, dtype: category
Categories (3, interval[float64, right]): [(0.0, 28.0] < (28.0, 250.0] < (250.0, 1062.0]]
```

here, we made cutoffs individually and pass them as a list to `pd.cut`

This is okay for 3 bins, but if we change our mind, it's a lot of work to make more. Better is to make the bins more programmatically:

```
[coffee_df_bags['Number.of.Bags'].quantile(pct) for pct in np.linspace(0,1,4)]
```

```
[0.0, 29.0, 250.0, 1062.0]
```

`np.linspace` returns a numpyarray of evenly (linearly; there is also logspace) spaced numbers. From the start to the end value for the number you specify. Here we said 4 evenly spaced from 0 to 1.

this is the same as we had before (up to rounding error)

[Skip to main content](#)

```
[min_bags, sm_cutoff, md_cutoff, max_bags]
```

```
[0, 28.0, 250.0, 1062]
```

Now we can use these and optionally, change to text labels (which then means we have to update that too if we change the number 4 to another number, but still less work than above)

```
bag_num_bins = [coffee_df_bags['Number.of.Bags'].quantile(pct) for pct in np.linspace(0,1,4)]
pd.cut(coffee_df_bags['Number.of.Bags'],
       bins=bag_num_bins,labels = ['small','medium','large']).head()
```

```
0    large
1    large
2   small
3    large
4    large
Name: Number.of.Bags, dtype: category
Categories (3, object): ['small' < 'medium' < 'large']
```

we could then add this to the dataframe to work with it

## 8.9. Questions

### 8.9.1. How can I rename without a dictionary

Really, best practice is a dictionary or function, that is what `rename` uses.

You can assign to the columns attribute, but then you have to provide all of the column names

### 8.9.2. Why are strings `object`?

it's largely for backwards compatibility

## 9. Webscraping

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
```

### ⚠️ Warning

If it says it cannot load one of the libraries, use pip inside your notebook to install,

```
pip install beautifulsoup4
```

then restart your kernel (Kernel menu, choose restart)

## 9.1. Getting Data From Websites

We have seen that `read_html` can get content from an actual website, not a data file that is hosted somewhere on the internet, that takes tables on a website and returns a list of DataFrames.

```
pd.read_html('https://rhodyprog4ds.github.io/BrownSpring23/syllabus/achievements.html')
```

▶ Show code cell output

This gives us a list of DataFrames that come from the website. `pandas` gets tables by looking in the html for the site and finding the `<table>` tags.

If we store it in a variable , we can see that

```
df_list = pd.read_html('https://rhodyprog4ds.github.io/BrownFall24/syllabus/achievements.html')  
[type(d) for d in df_list]
```

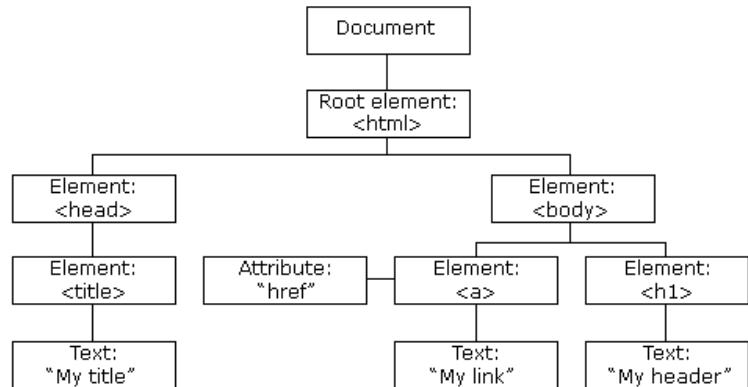
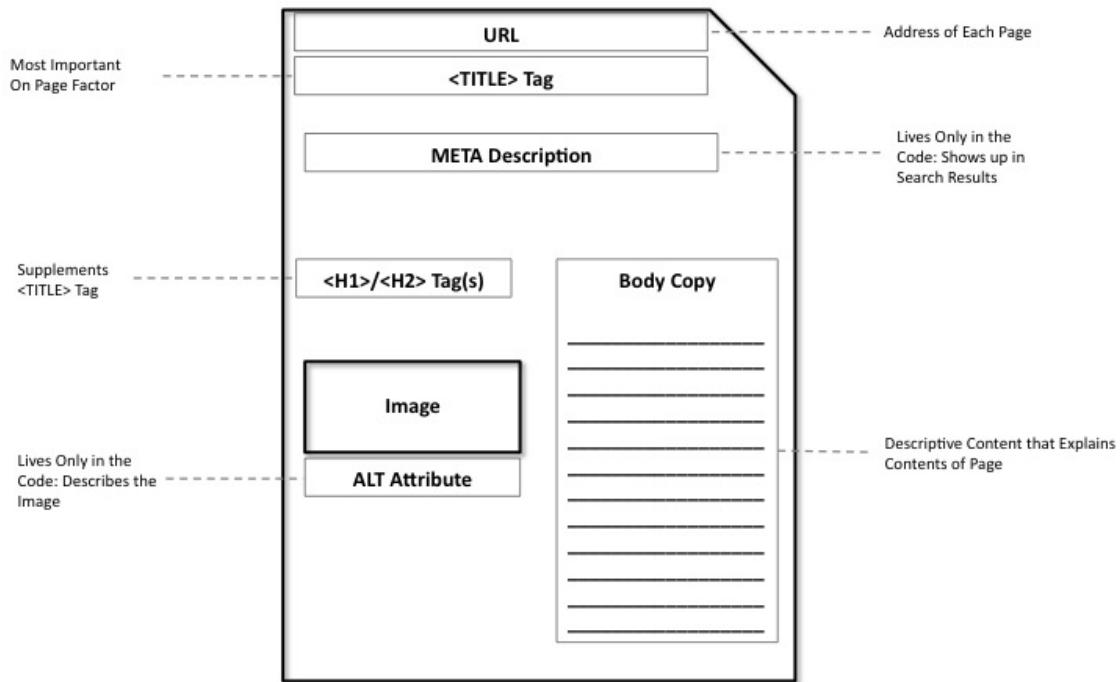
```
[pandas.core.frame.DataFrame,  
 pandas.core.frame.DataFrame,  
 pandas.core.frame.DataFrame]
```

## 9.2. Everything is Data

For the purpose of this class, it is best to think of the content on a web page like a datastructure.

i No

This  
defa



there are tags `<>` that define the structure, and these can be further classified with `classes`

## 9.3. Scraping a URI website

We're going to create a DataFrame about URI CS & Statistics Faculty.

from the people page of the department website.

We can inspect the page to check that it's well structured.

i No

the  
for e

## ⚠️ Warning

With great power comes great responsibility.

- always check the `robots.txt`
- do not do things that the owner says not to do
- government websites are typically safe

We'll save the URL for easy use

```
cs_people_url = 'https://web.uri.edu/cs/people/'
```

Then we can use the `requests` library to make a call to the internet. It actually gets back a `response object` which has a lot of extra information. For today we only need the `content` from the page which is an attribute of that object:

```
cs_people_html = requests.get(cs_people_url).content  
cs_people = BeautifulSoup(cs_people_html, 'html.parser')
```

This is raw:

```
cs_people_html[:100]
```

```
b'\n<!DOCTYPE html>\n<html lang="en-US">\n\t<head>\n\t\t<meta charset="UTF-8">\n\t\t<script type="text/javascript">
```

But we do not need to manually write search tools, that's what `BeautifulSoup` is for.

```
cs_people
```

```
<!DOCTYPE html>  
  
<html lang="en-US">  
<head>  
<meta charset="utf-8"/><script type="text/javascript">(window.NREUM||(NREUM={})).init={privacy:{cookies_e  
(( ))=>{var e,t,r={8122:(e,t,r)=>{"use strict";r.d(t,{a:( )=>i});var n=r(944);function i(e,t){try{if(!e||"ot  
<meta content="width=device-width, initial-scale=1" name="viewport"/>  
<link href="http://gmpg.org/xfn/11" rel="profile"/>  
<title>People - Department of Computer Science and Statistics</title>  
<meta content="max-image-preview:large" name="robots">  
<link href="//web.uri.edu" rel="dns-prefetch">  
<link href="https://web.uri.edu/cs/feed/" rel="alternate" title="Department of Computer Science and Stat  
<link href="https://web.uri.edu/cs/comments/feed/" rel="alternate" title="Department of Computer Science  
<script type="text/javascript">  
/* <![CDATA[ */  
window._wpemojiSettings = {"baseUrl":"https:\/\/s.w.org\/images\/core\/emoji\/15.0.3\/72x72\/","ext":".pr  
/*! This file is auto-generated */  
!function(i,n){var o,s,e;function c(e){try{var t={supportTests:e,timestamp:(new Date).valueOf()};sessions  
/* ]]> */  
</script>
```

```
type(cs_people)
```

```
bs4.BeautifulSoup
```

### 9.3.1. Looking at tags

In this object we can use any tag from the file and get back the first instance

the `<a>` tag in HTML makes a link

```
cs_people.a
```

```
<a class="skip-link screen-reader-text" href="#content">Skip to content</a>
```



Tip

This content is a best practice for accessible web design

We also see `<h3>` in the code so we can get the first one like this:

```
cs_people.h3
```

```
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/gavino-puggioni/">Gavino Puggioni</a></h3>
```

this [cheatsheet](#) shows lots of html tags, but for this purpose you do not really need it. You'll be inspecting the page and then looking for what you want

### 9.3.2. Searching the source

More helpful is the `find_all` method we want to find all `div` tags that are “peopleitem” class. We decided this by inspecting the code on the website.

```
type(cs_people.find_all('div', 'peopleitem'))
```

```
bs4.element.ResultSet
```

```
cs_people.find_all('div', 'peopleitem')
```

```
[<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/gavino-puggioni/">
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/gavino-puggioni/">Gavino Puggioni</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor | Chair</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.4388</span> <br/> <a class="u-email" href="mailto:gpu...@uri.edu">gavino.puggioni@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>,
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
```

this is a long, object and we can see it looks iterable ([ at the start)

```
[ people_items = cs_people.find_all('div', 'peopleitem')
```

We can look at a single item

```
[ people_items[0]
```

```
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/gavino-puggioni/">
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/gavino-puggioni/">Gavino Puggioni</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor | Chair</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.4388</span> <br/> <a class="u-email" href="mailto:gpu...@uri.edu">gavino.puggioni@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
```

and how many it finds

```
[ len(people_items)
```

## ! Important

answer to questions about searching from the docs

We notice that the name is inside a `<h3>` tag with class `p-name` and then inside an a tag after that. We also know from looking at the overall page that there are lots of other a tags, so we do not want to search all of those.

```
people_items[0].find('h3', 'p-name')
```

```
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/gavino-puggioni/">Gavino Puggioni</a></h3>
```

Then we see in this, there is an `<a>` tag, so we can pull that out next, we can use the tag attribute, because the first instance of the tag is exactly what we want.

```
people_items[0].find('h3', 'p-name').a
```

```
<a href="https://web.uri.edu/cs/meet/gavino-puggioni/">Gavino Puggioni</a>
```

inside that there is the text in a string, so we can pull that out

```
people_items[0].find('h3', 'p-name').a.string
```

```
'Gavino Puggioni'
```

Finally, now that we know how to get one out, we can put it all in a list comprehension

```
names = [person.find('h3', 'p-name').a.string for person in people_items]
```

pull out the titles for each person and store in a variable `titles`

## 9.4. Pulling more information

First, we look at the whole person entry again.

```
people_items[0]
```

! Im  
In th  
sim  
rep  
the  
you  
inte  
aud  
the  
stu  
aud

```

<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/gavino-puggioni/"></a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/gavino-puggioni/">Gavino Puggioni</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor | Chair</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.4388</span> <br/> <a class="u-email" href="mailto:gpu...@uri.edu">gpuggioni@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>

```

on one item, the `p` tag with the `people-title` class gets us what we want and then we can

```
title = [person.find('p', 'people-title').string for person in people_items]
```

We can also use the whole page object, but in some cases it might be safer or faster to only work with the chunk we had separate (like above with `people_items`). There are many ways and all are valid, but it is worth thinking about pros and cons. If there were any of the thing we were searching for in a different part that we did not want here, then the subset would be more accurate.

We can pull out two more things, the people-department indicates who is CS & who is Statistics.

```
disciplines = [d.string for d in cs_people.find_all("p", 'people-department')]
emails = [e.string for e in cs_people.find_all("a", 'u-email')]
```

We can finally use the DataFrame constructor to make it a table. I chose to use a dictionary in class

```
css_df = pd.DataFrame({'name':names, 'title':title,
                      'discipline':disciplines, 'email':emails})
css_df.head()
```

	name	title	discipline	email
0	Gavino Puggioni	Associate Professor   Chair	Statistics	gpuggioni@uri.edu
1	Lisa DiPippo	Professor   Director of Undergraduate Studies	Computer Science	ldipippo@uri.edu
2	Natallia Katenka	Associate Professor   Director of Data Science	Statistics	nkatenka@uri.edu
3	Krishna Venkatasubramanian	Associate Professor   Director of Graduate Studies	Computer Science	krish@uri.edu
4	Jing Wu	Associate Professor   Director of Graduate Studies	Statistics	jing_wu@uri.edu

We could also use a list of list and separate list of column names.

```
css_df_b = pd.DataFrame(data=[names,titles,emails,disciplines],  
                         columns=['names','titles','emails','disciplines'])  
css_df_b
```

```
NameError Traceback (most recent call last)  
Cell In[24], line 1  
----> 1 css_df_b = pd.DataFrame(data=[names,titles,emails,disciplines],  
                                columns=['names','titles','emails','disciplines'])  
      2  
      3 css_df_b  
  
NameError: name 'titles' is not defined
```

## 9.5. Crawling and scraping

Remember we pulled the names out of links, when in the browser, we click on the links, we see that they are to a profile page. On these pages, they have the office number. Let's add those to our dataframe.

First, we will do it for one person, then make a loop.

```
people_items[0].find('h3','p-name').a
```

```
<a href="https://web.uri.edu/cs/meet/gavino-puggioni/">Gavino Puggioni</a>
```

We see that the information that we want is in the `href` attribute, to read that, we check the documentation. This tells us there is a `.attrs` attribute of the python object we are working with.

```
people_items[0].find('h3','p-name').a.attrs
```

```
{'href': 'https://web.uri.edu/cs/meet/gavino-puggioni/'}
```

the classes are also attributes in HTML so we could access that information this way:

```
people_items[0].find('h3','p-name').attrs
```

```
{'class': ['p-name']}
```

It's a dictionary and the attribute we want is the key we want.

```
chair_url = people_items[0].find('h3','p-name').a.attrs['href']  
chair_url
```

```
'https://web.uri.edu/cs/meet/gavino-puggioni/'
```

Now, we do the same thing we did above, request, pull the content from the response and then use the parser.

```
chair_html = requests.get(chair_url).content
chair_info = BeautifulSoup(chair_html, 'html.parser')
```

then we find the tag and class we need from inspecting and pull that.

```
chair_info.find('li', 'people-location')
```

```
<li class="people-location"><strong>Office Location:</strong> Tyler Hall 254</li>
```

This structure is different and we tried string without success

Here, we could go to the documentation and look up what the object contains, or, we can use object serialization.

We can use the python `__dict__` to inspect the object and see where it stored what we want.

```
chair_info.find('li', 'people-location').__dict__
```

```
{'parser_class': bs4.BeautifulSoup,
'name': 'li',
'namespace': None,
'_namespaces': {},
'prefix': None,
'sourceline': 408,
'sourcepos': 303,
'known_xml': False,
'attrs': {'class': ['people-location']},
'contents': [<strong>Office Location:</strong>, ' Tyler Hall 254'],
'parent': <ul class="people-list">
<li class="people-title">Associate Professor | Chair</li> <li class="people-department">Statistics</li>
</ul>,
'previous_element': ' ',
'next_element': <strong>Office Location:</strong>,
'next_sibling': '\n',
'previous_sibling': ' ',
'hidden': False,
'can_be_empty_element': False,
'cdata_list_attributes': {'*': ['class', 'accesskey', 'dropzone'],
'a': ['rel', 'rev'],
'link': ['rel', 'rev'],
'td': ['headers'],
'th': ['headers'],
'form': ['accept-charset'],
'object': ['archive'],
'area': ['rel'],
'icon': ['sizes'],
'iframe': ['sandbox'],
'output': ['for']},
'preserve_whitespace_tags': {'pre', 'textarea'},
'interesting_string_types': (bs4.element.NavigableString, bs4.element.CData)}
```

we see its the second element in a list in the `'content'` value

```
chair_info.find('li', 'people-location').contents
```

[Skip to main content](#)

```
[<strong>Office Location:</strong>, ' Tyler Hall 254']
```

so we can pull it out

```
chair_info.find('li', 'people-location').contents[1]
```

```
' Tyler Hall 254'
```

Now that we know how to do it, we can put it in a loop.

```
offices = []
for person in people_items:
    person_url = person.find('h3', 'p-name').a.attrs['href']

    person_html = requests.get(person_url).content
    person_info = BeautifulSoup(person_html, 'html.parser')
    try:
        office_loc = person_info.find('li', 'people-location').contents[1]
        offices.append(office_loc)
    except:
        offices.append(pd.NA)
```

We added the `try` and `except` to handle when there is no office location. This is something in practice you would often think to do due to an error.

Next we add the offices to our dataframe:

```
css_df['offices'] = offices
```

and peek at the head

```
css_df.head()
```

	name	title	discipline	email	offices
0	Gavino Puggioni	Associate Professor   Chair	Statistics	gpuggioni@uri.edu	Tyler Hall 254
1	Lisa DiPippo	Professor   Director of Undergraduate Studies	Computer Science	ldipippo@uri.edu	Tyler 246
2	Natallia Katenka	Associate Professor   Director of Data Science	Statistics	nkatenka@uri.edu	Tyler 247
3	Krishna Venkatasubramanian	Associate Professor   Director of Graduate Stu...	Computer Science	krish@uri.edu	Tyler 131
4	Jing Wu	Associate Professor   Director of Graduate Stu...	Statistics	jing_wu@uri.edu	Tyler 260

Here we check the `info()` and we can see many were missing.

[Skip to main content](#)

```
css_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29 entries, 0 to 28
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   name        29 non-null    object  
 1   title       29 non-null    object  
 2   discipline  29 non-null    object  
 3   email       29 non-null    object  
 4   offices     26 non-null    object  
dtypes: object(5)
memory usage: 1.3+ KB
```

## 9.6. Questions

### Note

Some of these are questions asked in previous semesters

### 9.6.1. what does .a do?

it gives the first instance of the `<a>` tag

### 9.6.2. is it worth it to try and web scrape a page that is poorly written?

If it is important information. In these cases, you might have to do more manual parsing or even some manual fixes.

For this class, no.

### 9.6.3. Is there a way to check robots.txt through BeautifulSoup or must that be done manually in a browser?

it could maybe be read programmaticlaly, but it doesn't necessarily save time to do it that way.

### 9.6.4. What else can I do with inspect?

It lets you view the code. It's most often used to debug websites.

### 9.6.5. when web scraping if the html is not set up well is it possible to change the html to make it easier to parse

Technically you could manually edit a copy of it.

## 9.6.6. Are there instances where you can get data from websites that are not in tabular form?

Web scraping is *for* when the website is not in tabular form. It should be structured, but the structure does not need to come from a single page. It could be that there are many pages structured similarly and you build most of the columns from the other pages, not the starting page.

For example from the teams page of the nba you can get to a page with info about each team that includes all time records and the current rosters. On these individual pages, most info is an actual table, so you can use `pd.read_html` for those, but the crawling part from the first page would count.

▶ Show code cell content

# 10. Merging Data & Databases

## 10.1. Merging Data

```
import pandas as pd
import sqlite3
from urllib import request
```

we're going to work with a set of datasets today that are stored in a repo.

```
course_data_url = 'https://raw.githubusercontent.com/rhodyprog4ds/rhodyds/main/data/'
```

We can load in two data sets of NBA player information.

```
df_18 = pd.read_csv(course_data_url+'2018-players.csv')
df_19 = pd.read_csv(course_data_url+'2019-players.csv')
```

and take a peek at each

```
df_18.head()
```

	TEAM_ID	PLAYER_ID	SEASON
0	1610612761	202695	2018
1	1610612761	1627783	2018
2	1610612761	201188	2018
3	1610612761	201980	2018
4	1610612761	200768	2018

[Skip to main content](#)

```
df_19.head()
```

	PLAYER_NAME	TEAM_ID	PLAYER_ID	SEASON
0	Royce O'Neale	1610612762	1626220	2019
1	Bojan Bogdanovic	1610612762	202711	2019
2	Rudy Gobert	1610612762	203497	2019
3	Donovan Mitchell	1610612762	1628378	2019
4	Mike Conley	1610612762	201144	2019

### ! Important

Remember `shape` is a property, not a method, so it does not need `()`

Let's make note of the shape of each

```
df_18.shape, df_19.shape
```

```
((748, 3), (626, 4))
```

this created a tuple

#### 10.1.1. What if we want to analyze them together?

We can stack them, but this does not make it easy to see , for example, who changed teams.

```
pd.concat([df_18, df_19])
```

	TEAM_ID	PLAYER_ID	SEASON	PLAYER_NAME
0	1610612761	202695	2018	NaN
1	1610612761	1627783	2018	NaN
2	1610612761	201188	2018	NaN
3	1610612761	201980	2018	NaN
4	1610612761	200768	2018	NaN
...	...	...	...	...
621	1610612745	203461	2019	Anthony Bennett
622	1610612737	1629034	2019	Ray Spalding
623	1610612744	203906	2019	Devyn Marble
624	1610612753	1629755	2019	Hassani Gravett
625	1610612754	1629721	2019	JaKeenan Gant

1374 rows × 4 columns

```
pd.concat([df_18, df_19]).shape
```

```
(1374, 4)
```

we can see that this is the total number of rows:

```
748+626
```

```
1374
```

Note that this has the maximum number of columns (because both had some overlapping columns) and the total number of rows.

### 10.1.2. How can we find which players changed teams?

To do this we want to have one player column and a column with each year's team.

We can use a merge to do that.

```
pd.merge(df_18, df_19).head(2)
```

TEAM_ID	PLAYER_ID	SEASON	PLAYER_NAME
---------	-----------	--------	-------------

if we merge them without any parameters, it tries to merge on all shared columns. We want to merge them using the `PLAYER_ID` column though, we would say that we are “merging on player ID” and we use the `on` parameter to do it. In this case we want to merge on the `PLAYER_ID` column and then drop the other shared columns. This will result in a single row.

[Skip to main content](#)

1 into a single row.

```
pd.merge(df_18, df_19, on='PLAYER_ID', suffixes=('_18', '_19')).head(2)
```

	TEAM_ID_18	PLAYER_ID	SEASON_18	PLAYER_NAME	TEAM_ID_19	SEASON_19
0	1610612761	202695	2018	Kawhi Leonard	1610612746	2019
1	1610612761	1627783	2018	Pascal Siakam	1610612761	2019

Since there are other columns that appear in both DataFrames, they get a suffix, which by default is `x` or `y`, we can specify them though.

```
pd.merge(df_18, df_19, on='PLAYER_ID', suffixes=('_18', '_19')).shape
```

```
(538, 6)
```

### 10.1.3. Which players played in 2018, but not 2019?

We have different types of merges, inner is both, out is either. Left and right keep all the rows of one DataFrame. We can use left with `df_18` as the left DataFrame to see which players played only in 18.

```
pd.merge(df_18, df_19, on='PLAYER_ID', how='left', suffixes=('_18', '_19')).shape
```

```
(754, 6)
```

```
pd.merge(df_18, df_19, on='PLAYER_ID', how='left', suffixes=('_18', '_19')).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 754 entries, 0 to 753
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --    
 0   TEAM_ID_18    754 non-null   int64  
 1   PLAYER_ID     754 non-null   int64  
 2   SEASON_18     754 non-null   int64  
 3   PLAYER_NAME    538 non-null   object  
 4   TEAM_ID_19     538 non-null   float64 
 5   SEASON_19     538 non-null   float64 
dtypes: float64(2), int64(3), object(1)
memory usage: 35.5+ KB
```

If we save this to a variable, we can answer our question

```
df_18_only = pd.merge(df_18, df_19, on='PLAYER_ID', suffixes=('_18', '_19'), how='left')
df_18_only.head(2)
```

	TEAM_ID_18	PLAYER_ID	SEASON_18	PLAYER_NAME	TEAM_ID_19	SEASON_19
0	1610612761	202695	2018	Kawhi Leonard	1.610613e+09	2019.0
1	1610612761	1627783	2018	Pascal Siakam	1.610613e+09	2019.0

```
len(df_18_only[df_18_only['TEAM_ID_19'].isna()]['PLAYER_ID'].unique())
```

178

Also, note that this has different types than before. There are some players who only played one season, so they have a NaN value in some columns. pandas always casts a whole column.

## 10.2. Getting Data from Databases

### 10.2.1. What is a Database?

A common attitude in Data Science is:

If your data fits in memory there is no advantage to putting it in a database: it will only be slower and more frustrating. — Hadley Wickham

Businesses and research organizations nearly always have too much data to feasibly work without a database. Instead, they use different tools which are designed to scale to very large amounts of data. These tools are largely databases like Snowflake or Google's BigQuery and distributed computing frameworks like Apache Spark.

#### ⚠ Warning

We are going to focus on the case of getting data out of a Database so that you can use it and making sure you know what a Database is.

You could spend a whole semester on databases:

- CSC436 covers how to implement them in detail (recommended, but requires CSC212)
- BAI456 only how to use them (counts for DS majors, but if you want to understand them deeper, the CSC one is recommended)

For the purpose of this class the key attributes of a database are:

- it is a collection of tables
- the data is accessed live from disk (not RAM)
- you send a query to the database to get the data (or your answer)

Databases can be designed in many different ways. For examples two popular ones.

- SQLite is optimized for transactional workloads, which means a high volume of requests that involve inserting or reading a couple things. This is good for eg a webserver.
- DuckDB is optimized for analytical workloads, which means a small number of requests that each require reading many records in the database. This is better for eg: data science

**Experimenting with DuckDB is a way to earn construct level 3**

### 10.2.2. Accessing a Database from Python

We will use pandas again, as well as the `request` module from the `urllib` package and `sqlite3`.

Off the shelf, pandas cannot read databases by default. We'll use the `sqlite3` library, but there are others, depending on the type of database.

First we need to download the database to work with it.

```
request.urlretrieve('https://github.com/rhodyprog4ds/rhodyds/raw/main/data/nba1819.db',
    'nba1819.db')
```

```
('nba1819.db', <http.client.HTTPMessage at 0x7f31bff18e20>)
```

Next, we set up a connection, that links the notebook to the database. To use it, we add a cursor.

```
conn = sqlite3.connect('nba1819.db')
cursor = conn.cursor()
```

We can use `execute` to pass SQL queries through the cursor to the database.

```
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
```

```
<sqlite3.Cursor at 0x7f31bff30ab0>
```

Then we use `fetchall` to get the results of the query.

```
cursor.fetchall()
```

```
[('teams',),
 ('conferences',),
 ('playerGameStats2018',),
 ('playerGameStats2019',),
 ('teamGameStats2018',),
 ('teamGameStats2019',),
 ('playerTeams2018',),
 ('playerTeams2019',),
 ('teamDailyRankings2018',),
 ('teamDailyRankings2019',),
 ('playerNames',)]
```

If we fetch again, there is nothing to fetch. Fetch pulls what was queued by execute.

```
cursor.fetchall()
```

```
[]
```

```
pd.read_sql("SELECT name FROM sqlite_master WHERE type='table';",conn)
```

	name
0	teams
1	conferences
2	playerGameStats2018
3	playerGameStats2019
4	teamGameStats2018
5	teamGameStats2019
6	playerTeams2018
7	playerTeams2019
8	teamDailyRankings2018
9	teamDailyRankings2019
10	playerNames

### 10.3. Querying with pandas

We can use `pd.read_sql` to send queries, get the result sand transform them into a DataFrame all at once

We can pass the exact same queries if we want.

```
pd.read_sql("SELECT name FROM sqlite_master WHERE type='table';",conn)
```

	name
0	teams
1	conferences
2	playerGameStats2018
3	playerGameStats2019
4	teamGameStats2018
5	teamGameStats2019
6	playerTeams2018
7	playerTeams2019
8	teamDailyRankings2018
9	teamDailyRankings2019
10	playerNames

or we can get all of one of the tables:

```
pd.read_sql('SELECT * FROM teams', conn).head(1)
```

index	LEAGUE_ID	TEAM_ID	MIN_YEAR	MAX_YEAR	ABBREVIATION	NICKNAME	YEARFOUNDED
0	0	1610612737	1949	2019	ATL	Hawks	1949 A

### 10.3.1. Which player was traded the most during the 2018 season? How many times?

There is one row in players per team a played for per season, so if a player was traded (changed teams), they are in there multiple times.

First, we'll check the column names

```
pd.read_sql("SELECT * FROM playerTeams2018 LIMIT 1", conn)
```

index	TEAM_ID	PLAYER_ID
0	1610612761	202695

then get the 2018 players, we only need the `PLAYER_ID` column for this question

```
p18 =pd.read_sql("SELECT PLAYER_ID FROM playerTeams2018 ", conn)
```

Then we can use value counts

```
p18.value_counts().sort_values(ascending=False).head(10)
```

```
PLAYER_ID
1629150      4
202325       3
203092       3
201160       3
202328       3
1626150      3
1628393      3
202083       3
202692       3
203477       3
Name: count, dtype: int64
```

and we can get the player's name from the player name **remember our first query told us all the tables**

```
pd.read_sql("SELECT PLAYER_NAME FROM playerNames WHERE PLAYER_ID = 1629150", conn)
```

PLAYER_NAME	
0	Emanuel Terry

### 10.3.2. Did more players who changed teams from the 2018 season to the 2019 season stay in the same conferences or switch conferences?

In the NBA, there are 30 teams organized into two conferences: East and West; the `conferences` table has the columns `TEAM_ID` and `CONFERENCE`

Let's build a Dataframe that could answer the question.

I first pulled 1 row from each table I needed to see the columns.

```
pd.read_sql('SELECT * FROM conferences LIMIT 1', conn)
```

index	TEAM_ID	CONFERENCE
0	1610612744	West

```
pd.read_sql('SELECT * FROM playerTeams2018 LIMIT 1', conn)
```

index	TEAM_ID	PLAYER_ID
0	1610612761	202695

```
pd.read_sql('SELECT * FROM playerTeams2019 LIMIT 1', conn)
```

index	TEAM_ID	PLAYER_ID
Skip to main content		

Then I pulled the columns I needed from each of the 3 tables into a separate DataFrame.

```
conf_df = pd.read_sql('SELECT TEAM_ID, CONFERENCE FROM conferences', conn)
df18 = pd.read_sql('SELECT TEAM_ID, PLAYER_ID FROM playerTeams2018', conn)
df19 = pd.read_sql('SELECT TEAM_ID, PLAYER_ID FROM playerTeams2019', conn)
df18_c = pd.merge(df18, conf_df, on='TEAM_ID')
df19_c = pd.merge(df19, conf_df, on='TEAM_ID')
df1819_conf = pd.merge(df18_c, df19_c, on='PLAYER_ID', suffixes=('_2018', '_2019'))
df1819_conf
```

	TEAM_ID_2018	PLAYER_ID	CONFERENCE_2018	TEAM_ID_2019	CONFERENCE_2019
0	1610612761	202695	East	1610612746	West
1	1610612761	1627783	East	1610612761	East
2	1610612761	201188	East	1610612761	East
3	1610612763	201188	West	1610612761	East
4	1610612761	201980	East	1610612747	West
...	...	...	...	...	...
533	1610612739	1628021	East	1610612751	East
534	1610612739	201567	East	1610612739	East
535	1610612739	202684	East	1610612739	East
536	1610612739	1628424	East	1610612766	East
537	1610612739	1627819	East	1610612761	East

538 rows × 5 columns

Then I merged the conference with each set of player informationon the teams. Then I merged the two expanded single year DataFrames together.

Now, to answer the question, we have a bit more work to do. I'm going to use a `lambda` and `apply` to make a column that says same or new for the relative conference of the two seasons.

```
labels = {False:'new',True:'same'}
change_conf = lambda row: labels[row['CONFERENCE_2018']==row['CONFERENCE_2019']]
df1819_conf['conference_1819']= df1819_conf.apply(change_conf, axis=1)
df1819_conf.head()
```

	TEAM_ID_2018	PLAYER_ID	CONFERENCE_2018	TEAM_ID_2019	CONFERENCE_2019	conference_1819
0	1610612761	202695	East	1610612746	West	new
1	1610612761	1627783	East	1610612761	East	same
2	1610612761	201188	East	1610612761	East	same
3	1610612763	201188	West	1610612761	East	new
4	1610612761	201980	East	1610612747	West	new

Then I can use this DataFrame grouped by my new column to get the unique players in each situation new or same

[Skip to main content](#)

```
df1819_conf.groupby('conference_1819')['PLAYER_ID'].apply(pd.unique)
```

```
conference_1819
new      [202695, 201188, 201980, 203961, 1626153, 1011...
same     [1627783, 201188, 200768, 1627832, 201586, 162...
Name: PLAYER_ID, dtype: object
```

And finally, get the length of each of those lists.

```
df1819_conf.groupby('conference_1819')['PLAYER_ID'].apply(pd.unique).apply(len)
```

```
conference_1819
new      119
same     385
Name: PLAYER_ID, dtype: int64
```

This, however, includes players who stayed on the same team, so we also need to split for who changed teams. First we add the team comparison column, then groupby by both and count unique players.

```
new_team = lambda row: labels[row['TEAM_ID_2018']] == row['TEAM_ID_2019']
df1819_conf['team_1819'] = df1819_conf.apply(new_team, axis=1)
df1819_conf.groupby(['conference_1819', 'team_1819'])['PLAYER_ID'].apply(pd.unique).apply(len)
```

```
conference_1819  team_1819
new              new      119
same             new      135
                  same     263
Name: PLAYER_ID, dtype: int64
```

This is good, we could read the answer from here. It's good practice, though, to be able to pull that value out programmatically.

```
player_counts_1819_team = df1819_conf.groupby(['conference_1819', 'team_1819'])['PLAYER_ID'].apply(pd.uniq...
```

```
('same', 'same')
```

This tells us that the largest number of players stayed on the same team (and therefore same conference). We're not interested in this though, we're interested in those that changed teams, so we can drop the `(same, same)` value and then do this again.

```
player_counts_1819_team.drop(('same', 'same')).idxmax()
```

```
('same', 'new')
```

This tells us that more players changed teams within the same conference than changed teams and conferences. We can

[Skip to main content](#)

```
player_counts_1819_team['new', 'new'], player_counts_1819_team['same', 'new']
```

```
(119, 135)
```

Again 135 is more than 119.

We can also make this a little neater to print it as a DataFrame. If we use `reset_index` it will make a DataFrame, but the count column will still be named `PLAYER_ID` so we can rename it.

```
player_counts_1819_team.reset_index().rename(columns={'PLAYER_ID': 'num_players'})
```

	conference_1819	team_1819	num_players
0	new	new	119
1	same	new	135
2	same	same	263

All in all, this gives us a good answer that we can get with data and display answers and this is one way that using multiple data sources can help answer richer questions.

```
conn.close()
```

## 10.4. Questions

### 10.4.1. do the level 3 attempts have a due date?

End of the semester basically is it.

### 10.4.2. how long is assignment 5?

It can be a little long, web scraping is relatively open-ended.

## 11. Evaluating ML Algorithms

This week we are going to start learning about machine learning.

We are going to do this by looking at how to tell if machine learning has worked.

This is because:

- you have to check if one worked after you build one
- if you do not check carefully, it might only sometimes work

[Skip to main content](#)

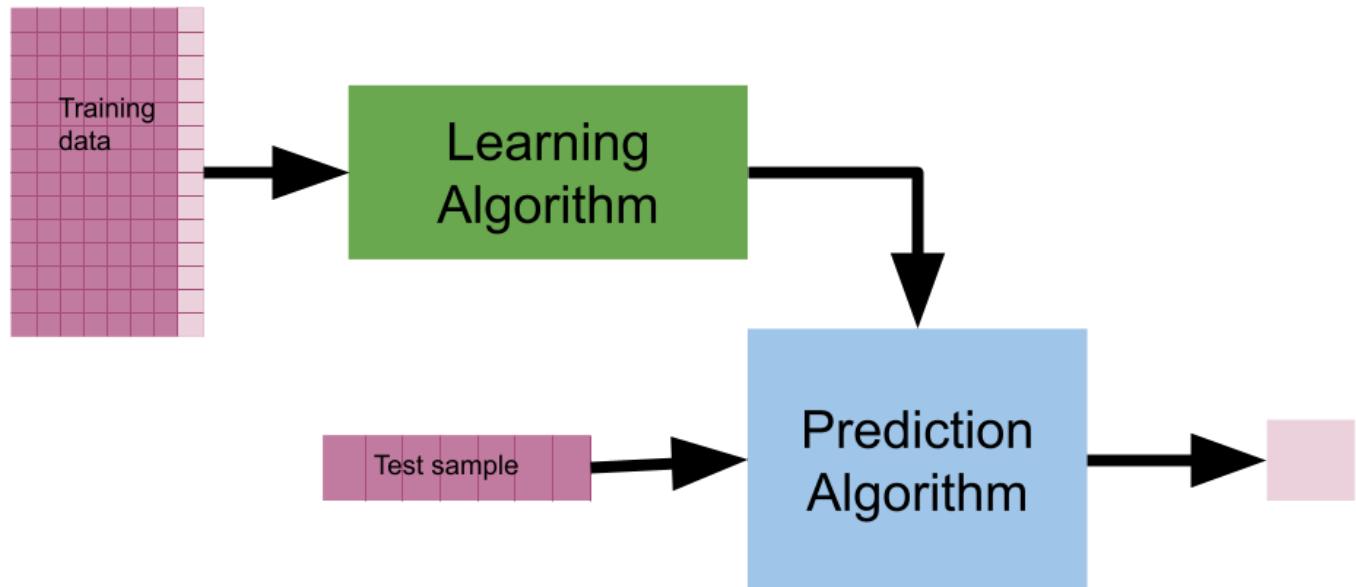
## 11.1. What is a Machine Learning Algorithm?

First, what is an Algorithm?

An algorithm is a set of ordered steps to complete a task.

Note that when people outside of CS talk about algorithms that impact people's lives these are often *not written directly by people* anymore. They are often the result of machine learning.

In machine learning, people write an algorithm for how to write an algorithm based on data. This often comes in the form of a statistical model of some sort.



When we *do* machine learning, this can also be called:

- data mining
- pattern recognition
- modeling

because we are looking for patterns in the data and typically then planning to use those patterns to make predictions or automate a task.

Each of these terms does have slightly different meanings and usage, but sometimes they're used close to exchangeably.

## 11.2. How can we tell if ML is working?

We measure the performance of the prediction algorithm, to determine if the learning algorithm worked.

## 11.3. Replicating the COMPAS Audit

We are going to replicate the audit from ProPublica Machine Bias

### 11.3.1. Why COMPAS?

Propublica started the COMPAS Debate with the article [Machine Bias](#). With their article, they also released details of their methodology and their [data and code](#). This presents a real data set that can be used for research on how data is used in a criminal justice setting without researchers having to perform their own requests for information, so it has been used and reused a lot of times.

### 11.3.2. Propublica COMPAS Data

The dataset consists of COMPAS scores assigned to defendants over two years 2013-2014 in Broward County, Florida, it was released by Propublica in a [GitHub Repository](#). These scores are determined by a proprietary algorithm designed to evaluate a persons recidivism risk - the likelihood that they will reoffend. Risk scoring algorithms are widely used by judges to inform their sentencing and bail decisions in the criminal justice system in the United States.

The journalists collected, for each person arrested in 2013 and 2014:

- basic demographics
- details about what they were charged with and priors
- the COMPAS score assigned to them
- if they had actually been re-arrested within 2 years of their arrest

This means that we have what the COMPAS algorithm predicted (in the form of a score from 1-10) and what actually happened (re-arrested or not). We can then measure how well the algorithm worked, in practice, in the real world.

Now we will start our notebook

```
import pandas as pd
from sklearn import metrics
import seaborn as sns
```

We're going to work with a cleaned copy of the data released by Propublica that also has a minimal subset of features.

- `c_charge_degree`: degree charged (Misdemeanor or Felony)
- `race`: defendant's race
- `age_cat`: defendant's age quantized in "less than 25", "25-45", or "over 45"
- `score_text`: COMPAS score: 'low'(1 to 5), 'medium' (5 to 7), and 'high' (8 to 10).
- `sex`: defendant's gender
- `priors_count`: number of prior charges
- `days_b_screening_arrest`: number of days between charge date and arrest where defendant was screened for compas score
- `decile_score`: COMPAS score from 1 to 10 (low risk to high risk)
- `is_recid`: if the defendant recidivized
- `two_year_recid`: if the defendant within two years
- `c_jail_in`: date defendant was imprisoned
- `c_jail_out`: date defendant was released from jail
- `length_of_stay`: length of jail stay

```
compas_clean_url = 'https://raw.githubusercontent.com/ml4sts/outreach-compas/main/data/compas_c.csv'
compas_df = pd.read_csv(compas_clean_url)
compas_df.head()
```

	<b>id</b>	<b>age</b>	<b>c_charge_degree</b>	<b>race</b>	<b>age_cat</b>	<b>score_text</b>	<b>sex</b>	<b>priors_count</b>	<b>days_b_screening_arrest</b>	
0	3	34		F	African-American	25 - 45	Low	Male	0	-1.0
1	4	24		F	African-American	Less than 25	Low	Male	4	-1.0
2	8	41		F	Caucasian	25 - 45	Medium	Male	14	-1.0
3	10	39		M	Caucasian	25 - 45	Low	Female	0	-1.0
4	14	27		F	Caucasian	25 - 45	Low	Male	0	-1.0

## 11.4. One-hot Encoding

We will audit first to see how good the algorithm is by treating the predictions as either high or not high. One way we can get to that point is to transform the `score_text` column from one column with three values, to 3 binary columns.

First lets understand the `score_text`

```
compas_df.groupby('score_text')['decile_score'].agg(['min', 'max'])
```

min max

score_text		min	max
High	8	10	
Low	1	4	
Medium	5	7	

`agg` is short for aggregate. It allows us to apply one or more functions to a dataframe (or groupby) along a single axis (rows or columns). It is similar to `apply` but allows us to pass a list of functions or function names and similar to `describe` but allows us to make custom summary tables.

You can learn more about `agg` from its [docs](#) or with visuals on [pandas tutor](#)

To see what one hot encoding looks like, we will apply it first to just the one column.

```
pd.get_dummies(compas_df['score_text'])
```

	High	Low	Medium
0	False	True	False
1	False	True	False
2	False	False	True
3	False	True	False
4	False	True	False
...	...	...	...
5273	False	True	False
5274	True	False	False
5275	False	False	True
5276	False	True	False
5277	False	True	False

5278 rows × 3 columns

Since we actually want all of the columns with that one expanded, we will apply it a different way to save it to a variable.

```
compas_onehot = pd.get_dummies(compas_df, columns=['score_text'])
```

We will also audit with respect to a second threshold.

```
compas_onehot['MedHigh'] = compas_onehot['High'] + compas_onehot['Medium']
```

```

-----
KeyError                                     Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'High'

The above exception was the direct cause of the following exception:

KeyError                                     Traceback (most recent call last)
Cell In[6], line 1
----> 1 compas_onehot['MedHigh'] = compas_onehot['High'] + compas_onehot['Medium']

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat
  3759     if self.columns.nlevels > 1:
  3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
  3762     if is_integer(indexer):
  3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,
  3653     return self._engine.get_loc(casted_key)
  3654     except KeyError as err:
-> 3655         raise KeyError(key) from err
  3656     except TypeError:
  3657         # If we have a listlike key, _check_indexing_error will raise
  3658         # InvalidIndexError. Otherwise we fall through and re-raise
  3659         # the TypeError.
  3660         self._check_indexing_error(key)

KeyError: 'High'

```

## 11.5. Sklearn Performance metrics

The first thing we usually check is the accuracy: the percentage of all samples that are correct.

```

metrics.accuracy_score(compas_onehot['two_year_recid'],
                      compas_onehot['MedHigh'])

```

```

-----
KeyError                                         Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'MedHigh'

The above exception was the direct cause of the following exception:

KeyError                                         Traceback (most recent call last)
Cell In[7], line 2
      1 metrics.accuracy_score(compas_onehot['two_year_recid'],
-> 2                         compas_onehot['MedHigh'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat
  3759     if self.columns.nlevels > 1:
  3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
  3762     if is_integer(indexer):
  3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,
  3653     return self._engine.get_loc(casted_key)
  3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
  3656 except TypeError:
  3657     # If we have a listlike key, _check_indexing_error will raise
  3658     # InvalidIndexError. Otherwise we fall through and re-raise
  3659     # the TypeError.
  3660     self._check_indexing_error(key)

KeyError: 'MedHigh'

```

## ! Important

all `sklearn.metrics` functions have the same first two parameters:

- `y_true` the *real world* outcome our model tries to predict
- `y_pred` the \*model's predictions

```

metrics.accuracy_score(compas_onehot['two_year_recid'],
                      compas_onehot['High'])

```

```

-----
KeyError                                     Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'High'

The above exception was the direct cause of the following exception:

KeyError                                     Traceback (most recent call last)
Cell In[8], line 2
      1 metrics.accuracy_score(compas_onehot['two_year_recid'],
-> 2                   compas_onehot['High'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat
  3759     if self.columns.nlevels > 1:
  3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
  3762     if is_integer(indexer):
  3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,
  3653     return self._engine.get_loc(casted_key)
  3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
  3656 except TypeError:
  3657     # If we have a listlike key, _check_indexing_error will raise
  3658     # InvalidIndexError. Otherwise we fall through and re-raise
  3659     # the TypeError.
  3660     self._check_indexing_error(key)

KeyError: 'High'

```

the accuracy is not great either way

However this does not tell us anything about what *types* of mistakes the algorithm made. The type of mistake often matters in terms of how we trust or deploy an algorithm. We use a [confusion matrix](#) to describe the performance in more detail.

A confusion matrix counts the number of samples of each *true* category that were predicted to be in each category. In this case we have a binary prediction problem: people either are re-arrested (truth) or not and were given a high score or not(prediction). In binary problems we adopt a common language of labeling one outcome/predicted value positive and the other negative. We do this not based on the social value of the outcome, but on the numerical encoding.

In this data, being re-arrested is indicated by a 1 in the `two_year_recid` column, so this is the *positive class* and not being re-arrested is 0, so the *negative class*. Similarly a high score is 1, so that's the *positive prediction* and not high is 0, so that is the a *negative prediction*.

[docs](#)

```
metrics.confusion_matrix(compas_onehot['two_year_recid'],
                         compas_onehot['MedHigh'])
```

```
-----
KeyError Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
      3652 try:
-> 3653     return self._engine.get_loc(casted_key)
      3654 except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'MedHigh'
```

The above exception was the direct cause of the following exception:

```
KeyError Traceback (most recent call last)
Cell In[9], line 2
      1 metrics.confusion_matrix(compas_onehot['two_year_recid'],
-> 2                     compas_onehot['MedHigh'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat
 3759 if self.columns.nlevels > 1:
 3760     return self._getitem_multilevel(key)
-> 3761 indexer = self.columns.get_loc(key)
 3762 if is_integer(indexer):
 3763     indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,
 3653     return self._engine.get_loc(casted_key)
 3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
 3656 except TypeError:
 3657     # If we have a listlike key, _check_indexing_error will raise
 3658     # InvalidIndexError. Otherwise we fall through and re-raise
 3659     # the TypeError.
 3660     self._check_indexing_error(key)

KeyError: 'MedHigh'
```

Using the help above, we can label the axes and put this in a DataFrame to make it easier to read.

```
cf = metrics.confusion_matrix(compas_onehot['two_year_recid'],
                             compas_onehot['MedHigh'])
pd.DataFrame(data = cf, columns = ['low score','medium or high score'],
              index = ['not rearrested','rearrested'])
```

```

-----
KeyError                                     Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'MedHigh'

The above exception was the direct cause of the following exception:

KeyError                                     Traceback (most recent call last)
Cell In[10], line 2
  1 cf = metrics.confusion_matrix(compas_onehot['two_year_recid'],
----> 2                               compas_onehot['MedHigh'])
  3 pd.DataFrame(data =cf,columns = ['low score','medium or high score'],
  4                 index = ['not rearrested','rearrested'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat
  3759 if self.columns.nlevels > 1:
  3760     return self._getitem_multilevel(key)
-> 3761 indexer = self.columns.get_loc(key)
  3762 if is_integer(indexer):
  3763     indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,
  3653     return self._engine.get_loc(casted_key)
  3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
  3656 except TypeError:
  3657     # If we have a listlike key, _check_indexing_error will raise
  3658     # InvalidIndexError. Otherwise we fall through and re-raise
  3659     # the TypeError.
  3660     self._check_indexing_error(key)

KeyError: 'MedHigh'

```

### Note

these terms can be used in any sort of detection problem, whether machine learning is used or not

`sklearn.metrics` provides a `[confusion matrix](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)` function that we can use.

Since this is binary problem we have 4 possible outcomes:

- true negatives( $C_{0,0}$ ): did not get a high score and were not re-arrested
- false negatives( $C_{1,0}$ ): did not get a high score and were re-arrested
- false positives( $C_{0,1}$ ): got a high score and were not re-arrested

[Skip to main content](#)

With these we can revisit accuracy:

$$A = \frac{C_{0,0} + C_{1,1}}{C_{0,0} + C_{1,0} + C_{0,1} + C_{1,1}}$$

and we can define new scores.

## 11.6. Precision and Recall

Two common ones in CS are recall and precision.

Recall is:

$$R = \frac{C_{1,1}}{C_{1,0} + C_{1,1}}$$

```
metrics.recall_score(compas_onehot['two_year_recid'],
                      compas_onehot['MedHigh'])
```

```

-----
KeyError                                     Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'MedHigh'

The above exception was the direct cause of the following exception:

KeyError                                     Traceback (most recent call last)
Cell In[11], line 2
      1 metrics.recall_score(compas_onehot['two_year_recid'],
-> 2                   compas_onehot['MedHigh'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat
  3759     if self.columns.nlevels > 1:
  3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
  3762     if is_integer(indexer):
  3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,
  3653     return self._engine.get_loc(casted_key)
  3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
  3656 except TypeError:
  3657     # If we have a listlike key, _check_indexing_error will raise
  3658     # InvalidIndexError. Otherwise we fall through and re-raise
  3659     # the TypeError.
  3660     self._check_indexing_error(key)

KeyError: 'MedHigh'

```

That is, among the truly positive class how many were correctly predicted? In COMPAS, it's the percentage of the re-arrested people who got a high score.

$$\text{Precision is } P = \frac{C_{1,1}}{C_{0,1} + C_{1,1}}$$

```

metrics.precision_score(compas_onehot['two_year_recid'],
                      compas_onehot['MedHigh'])

```

```

-----
KeyError                                     Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'MedHigh'

The above exception was the direct cause of the following exception:

KeyError                                     Traceback (most recent call last)
Cell In[12], line 2
      1 metrics.precision_score(compas_onehot['two_year_recid'],
----> 2                         compas_onehot['MedHigh'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat
  3759     if self.columns.nlevels > 1:
  3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
  3762     if is_integer(indexer):
  3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,
  3653     return self._engine.get_loc(casted_key)
  3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
  3656 except TypeError:
  3657     # If we have a listlike key, _check_indexing_error will raise
  3658     # InvalidIndexError. Otherwise we fall through and re-raise
  3659     # the TypeError.
  3660     self._check_indexing_error(key)

KeyError: 'MedHigh'

```

## 11.7. Per Group Scores

To groupby and then do the score, we can use a lambda, with apply

```

acc_fx = lambda d: metrics.accuracy_score(d['two_year_recid'], d['MedHigh'])
compas_onehot.groupby('race').apply(acc_fx)

```

```
-----
KeyError                                         Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:
  3655
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa...
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa...
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()
KeyError: 'MedHigh'

The above exception was the direct cause of the following exception:

KeyError                                         Traceback (most recent call last)
Cell In[13], line 2
      1 acc_fx = lambda d: metrics.accuracy_score(d['two_year_recid'], d['MedHigh'])
----> 2 compas_onehot.groupby('race').apply(acc_fx)
  3
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1351
  1351     with option_context("mode.chained_assignment", None):
  1352         try:
-> 1353             result = self._python_apply_general(f, self._selected_obj)
  1354         except TypeError:
  1355             # gh-20949
  1356             # try again, with .apply acting as a filtering
  (...).
  1360             # fails on *some* columns, e.g. a numeric operation
  1361             # on a string grouper column
  1363             return self._python_apply_general(f, self._obj_with_exclusions)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1367
  1367 @final
  1368 def _python_apply_general(
  1369     self,
  (...).
  1374     is_agg: bool = False,
  1375 ) -> NDFrameT:
  1376     """
  1377     Apply function f in python space
  1378
  (...).
  1400     data after applying f
  1401     """
-> 1402     values, mutated = self.grouper.apply(f, data, self.axis)
  1403     if not_indexed_same is None:
  1404         not_indexed_same = mutated

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:767, i...
  765 # group might be modified
  766 group_axes = group.axes
--> 767 res = f(group)
  768 if not mutated and not _is_indexed_like(res, group_axes, axis):
  769     mutated = True

Cell In[13], line 1, in <lambda>(d)
----> 1 acc_fx = lambda d: metrics.accuracy_score(d['two_year_recid'], d['MedHigh'])
      2 compas_onehot.groupby('race').apply(acc_fx)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat...
  3759 if self.columns.nlevels > 1:
```

[Skip to main content](#)

```

3762 if is_integer(indexer):
3763     indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,
3653     return self._engine.get_loc(casted_key)
3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
3656 except TypeError:
3657     # If we have a listlike key, _check_indexing_error will raise
3658     # InvalidIndexError. Otherwise we fall through and re-raise
3659     # the TypeError.
3660     self._check_indexing_error(key)

KeyError: 'MedHigh'

```

That lambda + apply is equivalent to:

```

race_acc = []
for race, rdf in compas_race:
    acc = skmetrics.accuracy_score(rdf['two_year_recid'],
                                    rdf['score_text_MedHigh'])
    race_acc.append([race, acc])

pd.DataFrame(race_acc, columns=['race', 'accuracy'])

```

```

-----
NameError                                                 Traceback (most recent call last)
Cell In[14], line 2
      1 race_acc = []
----> 2 for race, rdf in compas_race:
      3     acc = skmetrics.accuracy_score(rdf['two_year_recid'],
      4                                     rdf['score_text_MedHigh'])
      5     race_acc.append([race, acc])

NameError: name 'compas_race' is not defined

```

then we can do the same thing for recall and precision.

```

recall_fx = lambda d: metrics.recall_score(d['two_year_recid'], d['MedHigh'])
compas_onehot.groupby('race').apply(recall_fx)

```

```
-----
KeyError                                         Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:
  3655
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa...
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa...
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()
KeyError: 'MedHigh'

The above exception was the direct cause of the following exception:

KeyError                                         Traceback (most recent call last)
Cell In[15], line 2
    1 recall_fx = lambda d: metrics.recall_score(d['two_year_recid'],d['MedHigh'])
-> 2 compas_onehot.groupby('race').apply(recall_fx)
  3
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1351
  1351     with option_context("mode.chained_assignment", None):
  1352         try:
-> 1353             result = self._python_apply_general(f, self._selected_obj)
  1354         except TypeError:
  1355             # gh-20949
  1356             # try again, with .apply acting as a filtering
  (...).
  1360             # fails on *some* columns, e.g. a numeric operation
  1361             # on a string grouper column
  1363             return self._python_apply_general(f, self._obj_with_exclusions)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1367
  1367 @final
  1368 def _python_apply_general(
  1369     self,
  (...).
  1374     is_agg: bool = False,
  1375 ) -> NDFrameT:
  1376     """
  1377     Apply function f in python space
  1378
  (...).
  1400     data after applying f
  1401     """
-> 1402     values, mutated = self.grouper.apply(f, data, self.axis)
  1403     if not_indexed_same is None:
  1404         not_indexed_same = mutated

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:767, i...
  765 # group might be modified
  766 group_axes = group.axes
--> 767 res = f(group)
  768 if not mutated and not _is_indexed_like(res, group_axes, axis):
  769     mutated = True

Cell In[15], line 1, in <lambda>(d)
-> 1 recall_fx = lambda d: metrics.recall_score(d['two_year_recid'],d['MedHigh'])
  2 compas_onehot.groupby('race').apply(recall_fx)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat...
  3759 if self.columns.nlevels > 1:
```

[Skip to main content](#)

```
3762 if is_integer(indexer):  
3763     indexer = [indexer]  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,  
3653     return self._engine.get_loc(casted_key)  
3654 except KeyError as err:  
-> 3655     raise KeyError(key) from err  
3656 except TypeError:  
3657     # If we have a listlike key, _check_indexing_error will raise  
3658     # InvalidIndexError. Otherwise we fall through and re-raise  
3659     # the TypeError.  
3660     self._check_indexing_error(key)  
  
KeyError: 'MedHigh'
```

The recall tells us that the model has very different impact on people. On the other hand the precision tells us the scores mean about the same thing for Black and White people.

Researchers established that these are mutually exclusive, provably. We cannot have both, so it is very important to think about what the performance metrics mean and how your algorithm will be used in order to choose how to prepare a model. We will train models starting next week, but knowing these goals in advance is essential.

Importantly, this is not a statistical, computational choice that data can answer for us. This is about *human* values (and to some extent the law; certain domains have legal protections that require a specific condition).

The Fair Machine Learning book's classificaiton Chapter has a section on relationships between criteria with the proofs.

Looking at this gives a larger seeming difference to make it more clear, the error rate is almost twice as high.

```
1-compas_onehot.groupby('race').apply(recall_fx)
```

```
-----
KeyError Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'MedHigh'

The above exception was the direct cause of the following exception:

KeyError Traceback (most recent call last)
Cell In[16], line 1
----> 1 1> compas_onehot.groupby('race').apply(recall_fx)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:13
  1351     with option_context("mode.chained_assignment", None):
  1352         try:
-> 1353             result = self._python_apply_general(f, self._selected_obj)
  1354         except TypeError:
  1355             # gh-20949
  1356             # try again, with .apply acting as a filtering
  (...):
  1360             # fails on *some* columns, e.g. a numeric operation
  1361             # on a string grouper column
  1363             return self._python_apply_general(f, self._obj_with_exclusions)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:14
  1367 @final
  1368 def _python_apply_general(
  1369     self,
  (...):
  1374     is_agg: bool = False,
  1375 ) -> NDFrameT:
  1376     """
  1377         Apply function f in python space
  1378
  (...):
  1400         data after applying f
  1401     """
-> 1402     values, mutated = self.grouper.apply(f, data, self.axis)
  1403     if not_indexed_same is None:
  1404         not_indexed_same = mutated

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:767, i
  765 # group might be modified
  766 group_axes = group.axes
--> 767 res = f(group)
  768 if not mutated and not _is_indexed_like(res, group_axes, axis):
  769     mutated = True

Cell In[15], line 1, in <lambda>(d)
----> 1 recall_fx = lambda d: metrics.recall_score(d['two_year_recid'], d['MedHigh'])
  2 compas_onehot.groupby('race').apply(recall_fx)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat
  3759 if self.columns.nlevels > 1:
  3760     return self._getitem_multilevel(key)
```

[Skip to main content](#)

```
3763     indexer = [indexer]
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,
3653     return self._engine.get_loc(casted_key)
3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
3656 except TypeError:
3657     # If we have a listlike key, _check_indexing_error will raise
3658     # InvalidIndexError. Otherwise we fall through and re-raise
3659     # the TypeError.
3660     self._check_indexing_error(key)

KeyError: 'MedHigh'
```

Note this is almost double the error rate

```
precision_fx = lambda d: metrics.precision_score(d['two_year_recid'], d['MedHigh'])
compas_onehot.groupby('race').apply(precision_fx)
```

```
-----
KeyError                                         Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653,
  3652     try:
-> 3653         return self._engine.get_loc(casted_key)
  3654     except KeyError as err:
  3655
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pa...
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pa...
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()
KeyError: 'MedHigh'

The above exception was the direct cause of the following exception:

KeyError                                         Traceback (most recent call last)
Cell In[17], line 2
      1 precision_fx = lambda d: metrics.precision_score(d['two_year_recid'],d['MedHigh'])
-> 2 compas_onehot.groupby('race').apply(precision_fx)
  2
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1351
  1351     with option_context("mode.chained_assignment", None):
  1352         try:
-> 1353             result = self._python_apply_general(f, self._selected_obj)
  1354         except TypeError:
  1355             # gh-20949
  1356             # try again, with .apply acting as a filtering
  (...).
  1360             # fails on *some* columns, e.g. a numeric operation
  1361             # on a string grouper column
  1363             return self._python_apply_general(f, self._obj_with_exclusions)
  1363
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1367
  1367 @final
  1368 def _python_apply_general(
  1369     self,
  (...).
  1374     is_agg: bool = False,
  1375 ) -> NDFrameT:
  1376     """
  1377     Apply function f in python space
  1378
  (...).
  1400     data after applying f
  1401     """
-> 1402     values, mutated = self.grouper.apply(f, data, self.axis)
  1403     if not_indexed_same is None:
  1404         not_indexed_same = mutated
  1404
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:767, i...
  765 # group might be modified
  766 group_axes = group.axes
--> 767 res = f(group)
  768 if not mutated and not _is_indexed_like(res, group_axes, axis):
  769     mutated = True
  769
Cell In[17], line 1, in <lambda>(d)
----> 1 precision_fx = lambda d: metrics.precision_score(d['two_year_recid'],d['MedHigh'])
      2 compas_onehot.groupby('race').apply(precision_fx)
  2
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in Dat...
  3759 if self.columns.nlevels > 1:
```

[Skip to main content](#)

```
3762 if is_integer(indexer):  
3763     indexer = [indexer]  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655,  
3653     return self._engine.get_loc(casted_key)  
3654 except KeyError as err:  
-> 3655     raise KeyError(key) from err  
3656 except TypeError:  
3657     # If we have a listlike key, _check_indexing_error will raise  
3658     # InvalidIndexError. Otherwise we fall through and re-raise  
3659     # the TypeError.  
3660     self._check_indexing_error(key)  
  
KeyError: 'MedHigh'
```

### ! Important

We used ProPublica's COMPAS dataset to replicate (parts of, with different tools) their analysis. That is, they collected the dataset in order to audit the COMPAS algorithm and we used it for the same purpose (and to learn model evaluation). This dataset is not designed for *training* models, even though it has been used as such many times. This is [not the best way](#) to use this dataset and for future assignments I do not recommend using this dataset.

## 11.8. Questions

### 11.8.1. Does ChatGPT have a dataset similar to Compas? How accurate are the AI's that students are using instead of learning?

We do not know *exactly* what the training data for ChatGPT is, it uses OpenAI's model GPT3.5 and beyond. I also want to clarify that the dataset we used today was some journalists' data to *audit* COMPAS. It was trained on *different* data.

However for GPT3, OpenAI published a [paper](#) in section 2.2 they state that their training data includes (links are mine; they had citations):

- filtered version of [CommonCrawl1](#)
- an expanded version of the [WebText](#) dataset
- two sets of books
- English Language Wikipedia

### 11.8.2. There was a lot in this lecture today. What generally should be the take away from today's class?

The performance metrics and what ML *is* roughly, though we will come back to that.

### 11.8.3. What's the difference between precision score and recall score?

Conceptually, the precision is a perfect score (1) if there are no false *positives* but the precision score is not impacted by there being false negatives. Precision is a measure of how many of the high scores were actually re-arrested, it only cares about when the algorithm predicts the positive class.

In contrast, a perfect recall only requires no false *negatives* and is not impacted by false positives at all. Recall only cares about the actually positive class, here the people who *actually* got re-arrested and how well the model performed on those people. Recall is how many of the the thing we want were found in a radar context.

#### 11.8.4. how to combat biased data

This is still a really open area of research and it is context dependent. The COMPAS audit showed that there are different ways that we can write euqations to try to represent fairness and researchers proved that these cannot all be true at the same time.

In this case, I would actually not say that the “data is biased” because in statistics, biased data refers to a type of not correct measurement. This data reflects the way the actual **world** is biased.

#### 11.8.5. I guess when should and shouldn't you use one-hot?

We use one hot encoding when we want to numerically represent categorical data and if we want to do something that is like binary for one of the values for a variable with more than 2 values.

#### 11.8.6. What are some good intro resources you suggest for looking more into some of the math that make up these machine learning models that we will use in this class?

For the purposes of this course, the `sklearn` documentation is a good starting point. It has the specific equations that are implemented in the package, which is relevant for some models that have multiple ways they can be implemented.

## 12. ML Models and Auditing with AIF360

Let's start where we left off, plus some additional imports

```
import pandas as pd
from sklearn import metrics as skmetrics
from aif360 import metrics as fairmetrics
from aif360.datasets import BinaryLabelDataset
import seaborn as sns

compas_clean_url = 'https://raw.githubusercontent.com/ml4sts/outreach-compas/main/data/compas_c.csv'
compas_df = pd.read_csv(compas_clean_url, index_col = 'id')

compas_df = pd.get_dummies(compas_df, columns=['score_text'],)
```

```
WARNING:root:No module named 'tensorflow': AdversarialDebiasing will be unavailable. To install, run:  
pip install 'aif360[AdversarialDebiasing]'
```

```
WARNING:root:No module named 'tensorflow': AdversarialDebiasing will be unavailable. To install, run:  
pip install 'aif360[AdversarialDebiasing]'
```

```
WARNING:root:No module named 'fairlearn': ExponentiatedGradientReduction will be unavailable. To install,  
pip install 'aif360[Reductions]'
```

```
WARNING:root:No module named 'fairlearn': GridSearchReduction will be unavailable. To install, run:  
pip install 'aif360[Reductions]'
```

```
WARNING:root:No module named 'inFairness': SenSeI and SenSR will be unavailable. To install, run:  
pip install 'aif360[inFairness]'
```

```
WARNING:root:No module named 'fairlearn': GridSearchReduction will be unavailable. To install, run:  
pip install 'aif360[Reductions]'
```

We may get a warning which is **okay**. If you run the cell again it will go away.

We are going to continue with the ProPublica COMPAS audit data. Remember it contains:

- `age`: defendant's age
- `c_charge_degree`: degree charged (Misdemeanor or Felony)
- `race`: defendant's race
- `age_cat`: defendant's age quantized in "less than 25", "25-45", or "over 45"
- `score_text`: COMPAS score: 'low'(1 to 5), 'medium' (5 to 7), and 'high' (8 to 10).
- `sex`: defendant's gender
- `priors_count`: number of prior charges
- `days_b_screening_arrest`: number of days between charge date and arrest where defendant was screened for compas score
- `decile_score`: COMPAS score from 1 to 10 (low risk to high risk)
- `is_recid`: if the defendant recidivized
- `two_year_recid`: if the defendant within two years
- `c_jail_in`: date defendant was imprisoned
- `c_jail_out`: date defendant was released from jail
- `length_of_stay`: length of jail stay

```
compas_df.head()
```

	age	c_charge_degree	race	age_cat	sex	priors_count	days_b_screening_arrest	decile_score
id								
3	34	F	African-American	25 - 45	Male	0	-1.0	3
4	24	F	African-American	Less than 25	Male	4	-1.0	4
8	41	F	Caucasian	25 - 45	Male	14	-1.0	6
10	39	M	Caucasian	25 - 45	Female	0	-1.0	1
14	27	F	Caucasian	25 - 45	Male	0	-1.0	4

Notice the last three columns. When we use `pd.getdummies` with its `columns` parameter, then we can append the columns all at once and they get the original column name prepended to the value in the new column name.

Most common is to use medium or high to check accuracy (or not low) we can calculate this by either summing two or inverting

let's do it by inverting here

```
int_not = lambda a:int(not(a))
compas_df['score_text_MedHigh'] = compas_df['score_text_Low'].apply(int_not)
```

the st

We use the two year recid column, because that is what actually happened in the real world, if the person was re-arrested or not within two years

```
skmetrics.accuracy_score(compas_df['two_year_recid'],
                        compas_df['score_text_MedHigh'])
```

0.6582038651004168

```
skmetrics.accuracy_score(compas_df['two_year_recid'],
                        compas_df['score_text_High'])
```

0.6288366805608185

## 12.1. Using AIF 360

## 12.2. Using AIF360

The AIF360 package implements fairness metrics, some of which are derived from metrics we have seen and some others. [the documentation](#) has the full list in a summary table with English explanations and details with most equations.

However, it has a few requirements:

- its constructor takes two `BinaryLabelDataset` objects
- these objects must be the same except for the label column
- the constructor for `BinaryLabelDataset` only accepts all numerical DataFrames

So, we have some preparation to do.

First, we'll make a numerical copy of the `compas_df` columns that we need. The only nonnumerical column that we need is race, so we'll make a `dict` to replace that/

We need to use numerical values for the protected attribute, so let's make a mapping value

one way to get the values for race is:

```
compas_df['race'].value_counts().index
```

```
Index(['African-American', 'Caucasian'], dtype='object', name='race')
```

then we want to number them, so we can replace each string with a number. Python has `enumerate` for that, [enumerate doc](#). We can combine that with a dictionary comprehension.

`enumerate` is [generator](#) which is a special sort of function. From the docs, the following is equivalent:

```
def enumerate(iterable, start=0):
    n = start
    for elem in iterable:
        yield n, elem
        n += 1
```

`yield` works sort of like `return` but it doesn't end the function, it gives the one value back and then waits for it to be used again. The [core python docs for yeild expressions](#) explain in detail.

```
race_num_map = {r:i for i,r, in enumerate(compas_df['race'].value_counts().index)}
```

```
{'African-American': 0, 'Caucasian': 1}
```

so we have this mapper that works with replace:

```
id
3      0
4      0
8      1
10     1
14     1
...
10994   0
10995   0
10996   0
10997   0
11000   0
Name: race, Length: 5278, dtype: int64
```

Now we will make our smaller dataframe.

We will also only use a few of the variables.

```
required_cols = ['race', 'two_year_recid', 'score_text_MedHigh']
num_compas = compas_df[required_cols].replace(race_num_map)
num_compas.head(2)
```

	race	two_year_recid	score_text_MedHigh
id			
3	0	1	0
4	0	1	0

The scoring object requires that we have special data structures that wrap a DataFrame.

We need one aif360 binary labeled dataset for the true values and one for the predictions.

Next we will make two versions, one with race & the ground truth and ht ether with race & the predictions. It's easiest to drop the column we don't want.

The difference between the two datasets needs to be only the label column, so we drop the other variable from each small dataframe that we create.

```
num_compas_true = num_compas.drop(columns=['score_text_MedHigh'])
num_compas_pred = num_compas.drop(columns=['two_year_recid'])
```

```
num_compas_true.head()
```

	<b>race</b>	<b>two_year_recid</b>
<b>id</b>		
<b>3</b>	0	1
<b>4</b>	0	1
<b>8</b>	1	1

	<b>race</b>	<b>two_year_recid</b>
<b>id</b>		
<b>10</b>	1	0
<b>14</b>	1	0

Now we make the `BinaryLabelDataset` objects, this type comes from AIF360 too. Basically, it is a DataFrame with extra attributes; some specific and some inherited from `StructuredDataset`.

```
broward_true = BinaryLabelDataset(favorable_label=0, unfavorable_label=1,
                                 df = num_compas_true,
                                 label_names= ['two_year_recid'],
                                 protected_attribute_names=['race'])
compas_predictions = BinaryLabelDataset(favorable_label=0, unfavorable_label=1,
                                         df = num_compas_pred,
                                         label_names= ['score_text_MedHigh'],
                                         protected_attribute_names=['race'])
```

This type also has an `ignore_fields` column for when comparisons are made, since the requirement is that **only** the *content* of the label column is different, but in our case the label names are also different, we have to tell it that that's okay.

```
# because our columns are named differently, we have to ignore that
compas_predictions.ignore_fields.add('label_names')
broward_true.ignore_fields.add('label_names')
compas_fair_scorer = fairmetrics.ClassificationMetric(broward_true,
                                                       compas_predictions,
                                                       unprivileged_groups=[{'race':0}],
                                                       privileged_groups = [{'race':1}])
```

Now we can use the scores

```
compas_fair_scorer.accuracy()
```

```
0.6582038651004168
```

By default, we get the overall accuracy. This calculation matches what we got using sklearn.

For the aif360 metrics, they have one parameter, `privileged` with a default value of `None` when it's none it computes the whole dataset. When `True` it computes only the privileged group.

```
compas_fair_scorer.accuracy(True)
```

```
0.6718972895863052
```

Here that is Caucasian people.

When `False` it's the unprivileged group, here African American

```
compas_fair_scorer.accuracy(False)
```

```
0.6491338582677165
```

These again match what we calculated before, the advantaged group (White) for True and disadvantaged group (Black) for False

```
compas_fair_scorer.error_rate_difference()
```

```
0.02276343131858871
```

the error rate alone does not tell the whole story because there are two types of errors. Plus there are even more ways we can think about if something is fair or not.

#### Important

This is extra detail

### 12.2.1. Disparate Impact

One way we might want to be fair is if the same % of each group of people (Black,  $A = 0$  and White,  $A = 1$ ) get the favorable outcome (a low score).

In Disparate Impact the ratio is of the positive outcome, independent of the predictor. So this is the ratio of the % of Black people not rearrested to % of white people rearrested.

$$D = \frac{\Pr(\hat{Y} = 1|A = 0)}{\Pr(\hat{Y} = 1|A = 1)}$$

This is equivalent to saying that the score is unrelated to race.

This type of fair is often the kind that most people think of intuitively.

```
compas_fair_scorer.disparate_impact()
```

```
0.6336457196581771
```

US court doctrine says that this quantity has to be above .8 for employment decisions.

## 12.3. Equalized Odds Fairness

The journalists were concerned with the types of errors. They accepted that it is not the creators of COMPAS fault that Black people get arrested at higher rates (though actual crime rates are equal; Black neighborhoods tend to be overpoliced). They wanted to consider what actually happened and then see how COMPAS did within each group.

```
compas_fair_scorer.false_positive_rate(True)
```

```
0.49635036496350365
```

```
compas_fair_scorer.false_positive_rate(False)
```

```
0.2847682119205298
```

false positives are incorrectly got a low score.

This is different from how the problem was setup when we used sklearn because sklearn assumes tht 0 is the negative class and 1 is the “positive” class, but AIF360 lets us declare the favorable outcome(positive class) and unfavorable outcome (negative class)

White people were given a low score and then re-arrested almost twice as often as Black people.

To make a single metric, we might take a ratio. This is where the journalists found bias.

```
compas_fair_scorer.false_positive_rate_ratio()
```

```
0.5737241916634204
```

We can look at the other type of error

```
compas_fair_scorer.false_negative_rate(True)
```

```
0.22014051522248243
```

```
compas_fair_scorer.false_negative_rate(False)
```

```
0.4233817701453104
```

```
compas_fair_scorer.false_negative_rate_ratio()
```

```
1.9232342111919953
```

Black people were given a high score and not rearrested almost twice as often as white people.

So while the accuracy was similar (see error rate ratio) for Black and White people; the algorithm makes the opposite types of errors.

### 12.3.1. Average Odds Difference

This is a combines the two errors we looked at separately into a single metric.

$$\frac{1}{2}[(FPR_{A=\text{unprivileged}} - FPR_{A=\text{privileged}}) + (TPR_{A=\text{unprivileged}} - TPR_{A=\text{privileged}})]$$

```
compas_fair_scorer.average_odds_difference()
```

```
-0.2074117039829009
```

After the journalists published the piece, the people who made COMPAS countered with a technical report, arguing that that the journalists had measured fairness incorrectly.

The journalists two measures false positive rate and false negative rate use the true outcomes as the denominator.

The COMPAS creators argued that the model should be evaluated in terms of if a given score means the same thing across races; using the prediction as the denominator.

We can look at their preferred metrics too

```
compas_fair_scorer.false_omission_rate(True)
```

```
0.4051724137931034
```

```
compas_fair_scorer.false_omission_rate(False)
```

```
0.35046473482777474
```

```
compas_fair_scorer.false_omission_rate_ratio()
```

```
0.8649767923408909
```

```
compas_fair_scorer.false_discovery_rate_ratio()
```

[Skip to main content](#)

On these two metrics, the ratio is closer to 1 and much less disparate.

The creators thought it was important for the score to mean the same thing for every person assigned a score. The journalists thought it was more important for the algorithm to have the same impact of different groups of people.

Ideally, we would like the score to both mean the same thing for different people and to have the same impact.

Researchers established that these are mutually exclusive, provably. We cannot have both, so it is very important to think about what the performance metrics mean and how your algorithm will be used in order to choose how to prepare a model. We will train models starting next week, but knowing these goals in advance is essential.

Importantly, this is not a statistical, computational choice that data can answer for us. This is about *human* values (and to some extent the law; certain domains have legal protections that require a specific condition).

The Fair Machine Learning book's classification Chapter has a section on relationships between criteria with the proofs.

## 12.4. ML Modeling

We're going to approach machine learning from the perspective of *modeling* for a few reasons:

- model based machine learning streamlines understanding the big picture
- the model way of interpreting it aligns well with using sklearn
- thinking in terms of models aligns with incorporating domain expertise, as in our data science definition

this [paper](#) by Christopher M. Bishop, a pioneering ML researcher who also wrote one of the widely preferred graduate level ML textbooks, details advantages of a model based perspective and a more mathematical version of a model based approach to machine learning. He is a co-author on an introductory [model based ML](#)

In CSC461: Machine Learning, you can encounter an *algorithm* focused approach to machine learning, but I think having the model based perspective first helps you avoid common pitfalls.

## 12.5. What is a Model?

A model is a simplified representation of some part of the world. A famous quote about models is:

All models are wrong, but some are useful –[George Box](#)[^wiki]

In machine learning, we use models, that are generally *statistical* models.

A statistical model is a mathematical model that embodies a set of statistical assumptions concerning the generation of sample data (and similar data from a larger population). A statistical model represents, often in considerably idealized form, the data-generating process [wikipedia](#)

read more in the [Model Based Machine Learning Book](#)

## 12.6. Models in Machine Learning

Starting from a dataset, we first make an additional designation about how we will use the different variables (columns). We will call most of them the *features*, which we denote mathematically with  $\mathbf{X}$  and we'll choose one to be the *target* or *labels*, denoted by  $\mathbf{y}$ .

The core assumption for just about all machine learning is that there exists some function  $f$  so that for the  $i$ th sample

$$y_i = f(\mathbf{x}_i)$$

$i$  would be the index of a DataFrame

## 12.7. Types of Machine Learning

Then with different additional assumptions we get different types of machine learning:

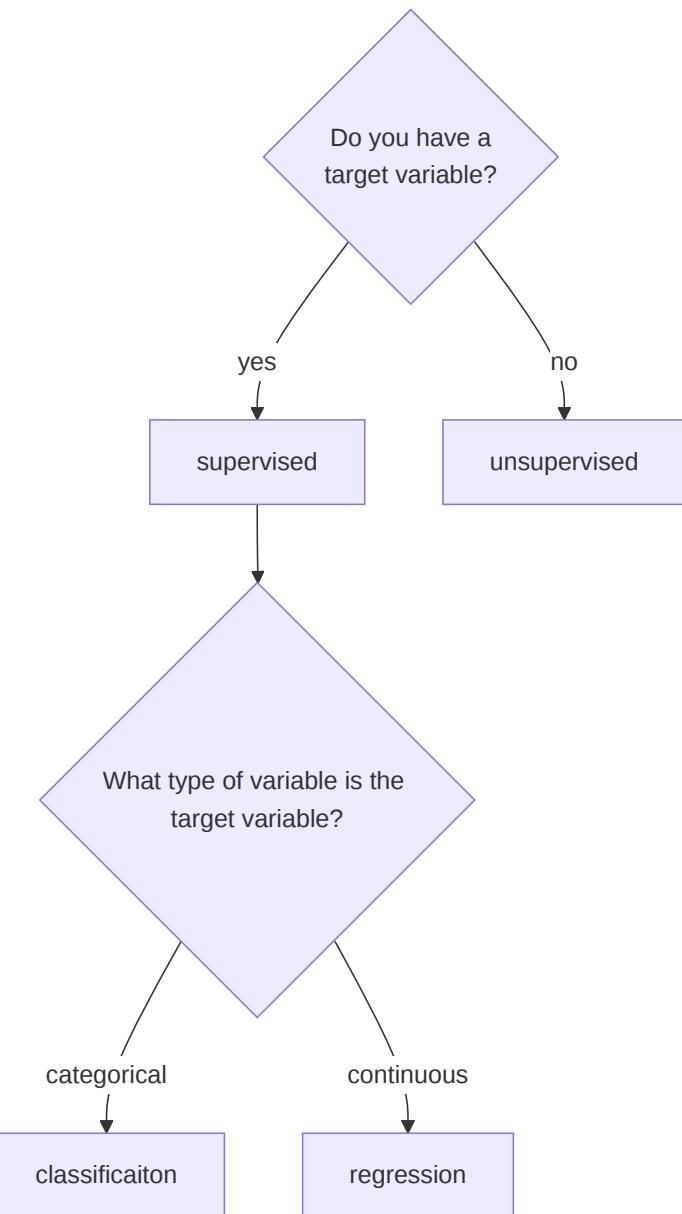
- if both features ( $\mathbf{X}$ ) and target ( $\mathbf{y}$ ) are observed (contained in our dataset) it's **supervised learning** code
- if only the features ( $\mathbf{X}$ ) are observed, it's **unsupervised learning** code

## 12.8. Supervised Learning

we'll focus on supervised learning first. we can take that same core assumption and use it with additional information about our target variable to determine learning **task** we are working to do.

$$y_i = f(\mathbf{x}_i)$$

- if  $y_i$  are discrete (eg flower species) we are doing **classification**
- if  $y_i$  are continuous (eg height) we are doing **regression**



## 12.9. Machine Learning Pipeline

To do machine learning we start with **training data** which we put as input to the **learning algorithm**. A learning algorithm might be a generic optimization procedure or a specialized procedure for a specific model. The learning algorithm outputs a trained **model** or the parameters of the model. When we deploy a model we pair the **fit model** with a **prediction algorithm** or **decision** algorithm to evaluate a new sample in the world.

In experimenting and design, we need **testing data** to evaluate how well our learning algorithm understood the world. We need to use previously unseen data, because if we don't we can't tell if the prediction algorithm is using a rule that the learning algorithm produced or just looking up from a lookup table the result. This can be thought of like the difference between memorization and understanding.

When the model does well on the training data, but not on test data, we say that it does not **generalize** well.

# 13. Classification Models

First machine learning model: Naive bayes

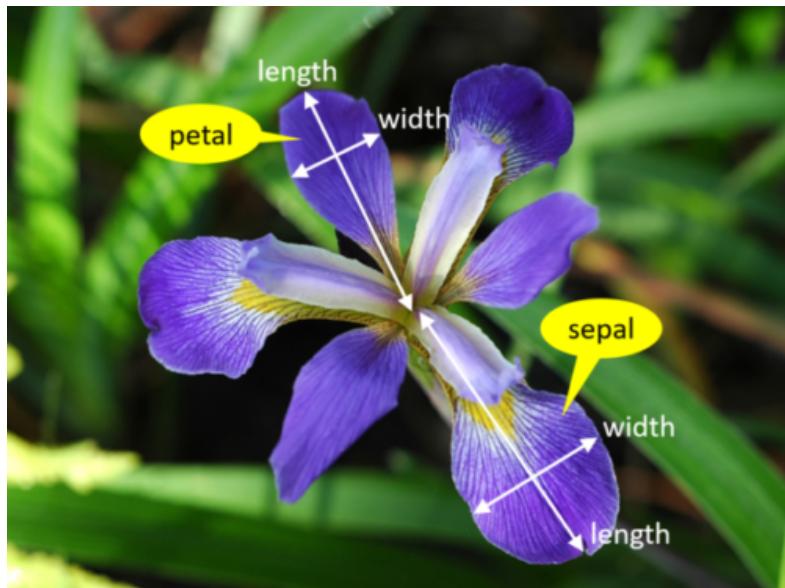
```
import pandas as pd
import seaborn as sns
from sklearn import tree
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
import matplotlib.pyplot as plt

iris_df = sns.load_dataset('iris')
```

To start we will look at the data

```
iris_df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa



We're trying to build an automatic flower classifier that, for measurements of a new flower returns the predicted species. To do this, we have a DataFrame with columns for species, petal width, petal length, sepal length, and sepal width. The species is what type of flower it is the petal and sepal are parts of the flower.

[Skip to main content](#)

The species will be the target and the measurements will be the features. We want to predict the target from the features, the species from the measurements.

```
feature_vars = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
target_var = 'species'
```

## 13.1. What does Naive Bayes do?

- Naive = independent features
- Bayes = most probable

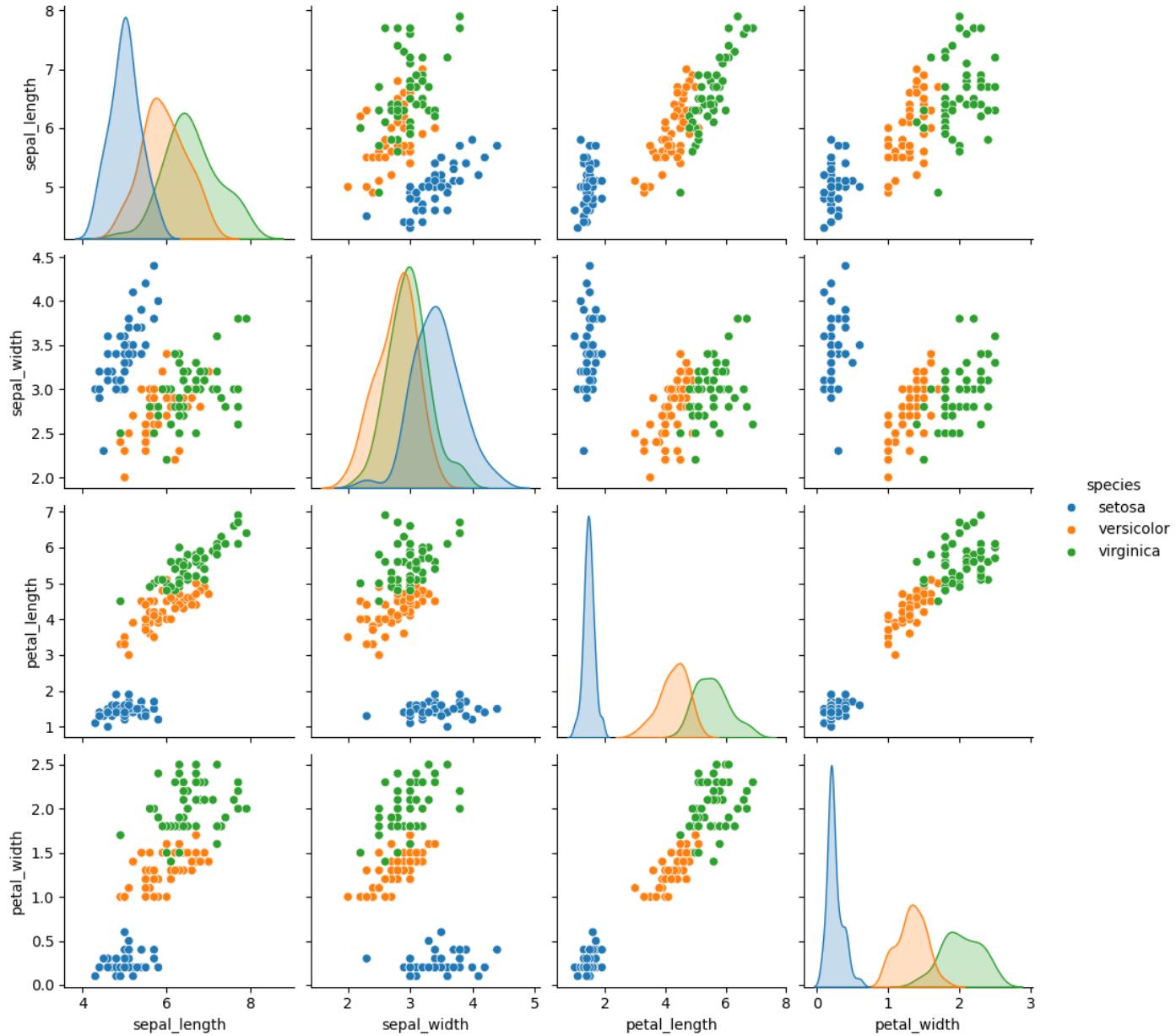
More resources:

- [Bayes Estimator](#)
- [docs](#)

We can look at this data using a pair plot. It plots each pair of numerical variables in a grid of scatterplots and on the diagonal (where it would be a variable with itself) shows the distribution of that variable.

```
sns.pairplot(data =iris_df,hue=target_var)
```

```
<seaborn.axisgrid.PairGrid at 0x7f0f47376970>
```



This data is reasonably **separable** because the different species (indicated with colors in the plot) do not overlap much. We see that the features are distributed sort of like a normal, or Gaussian, distribution. In 2D a Gaussian distribution is like a hill, so we expect to see more points near the center and fewer on the edge of circle-ish blobs. These blobs are slightly like ovals, but not too skewed.

This means that the assumptions of the Gaussian Naive Bayes model are met well enough we can expect the classifier to do well.

## 13.2. Separating Training and Test Data

To do machine learning, we split the data both sample wise (rows if tidy) and variable-wise (columns if tidy). First, we'll

The features are the input that we wish to use to predict the target.

Next, we'll use a sklearn function to split the data randomly into test and train portions.

```
X_train, X_test, y_train, y_test = train_test_split(iris_df[feature_vars],  
                                                iris_df[target_var],  
                                                random_state=5)
```

This function returns multiple values, the docs say that it returns **twice as many** as it is passed. We passed two separate things, the features and the labels separated, so we get train and test each for both.

#### **Note**

If you get different numbers for the index than I do here or run the train test split multiple times and see things change, you have a different random seed above.

```
X_train.head()
```

	sepal_length	sepal_width	petal_length	petal_width
<b>40</b>	5.0	3.5	1.3	0.3
<b>115</b>	6.4	3.2	5.3	2.3
<b>142</b>	5.8	2.7	5.1	1.9
<b>69</b>	5.6	2.5	3.9	1.1
<b>17</b>	5.1	3.5	1.4	0.3

```
X_train.shape, X_test.shape
```

```
((112, 4), (38, 4))
```

We can see by default how many samples it puts the training set:

```
len(X_train)/len(iris_df)
```

```
0.7466666666666667
```

So by default we get a 75-25 split.

### 13.3. Instantiating our Model Object

Next we will instantiate the object for our *model*. In `sklearn` they call these objects **estimator**. All estimators have a similar usage. First we instantiate the object and set any *hyperparameters*.

[Skip to main content](#)

Instantiating the object says we are assuming a particular type of model. In this case Gaussian Naive Bayes. This sets several assumptions in one form:

- we assume data are Gaussian (normally) distributed
- the features are uncorrelated/independent (Naive)
- the best way to predict is to find the highest probability (Bayes)

this is one example of a Bayes Estimator

```
gnb = GaussianNB()
```

At this point the object is not very interesting

```
gnb.__dict__
```

```
{'priors': None, 'var_smoothing': 1e-09}
```

The fit method uses the data to learn the model's parameters. In this case, a Gaussian distribution is characterized by a mean and variance; so the GNB classifier is characterized by one mean and one variance for each class (in 4d, like our data)

```
gnb.fit(X_train,y_train)
```

▼ GaussianNB  
GaussianNB()

The attributes of the estimator object (`gnb`) describe the data (eg the class list) and the model's parameters. The `theta_` (often in math as  $\theta$  or  $\mu$ ) represents the mean and the `var_` ( $\sigma$ ) represents the variance of the distributions.

```
gnb.__dict__
```

```
{'priors': None,
 'var_smoothing': 1e-09,
 'classes_': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'feature_names_in_': array(['sepal_length', 'sepal_width', 'petal_length', 'petal_width'],
    dtype=object),
 'n_features_in_': 4,
 'epsilon_': 3.1373206313775512e-09,
 'theta_': array([[5.00526316, 3.45263158, 1.45       , 0.23421053],
    [5.96666667, 2.75277778, 4.26666667, 1.31944444],
    [6.55789474, 2.98684211, 5.52105263, 2.02631579]]),
 'var_': array([[0.14207757, 0.16196676, 0.03460527, 0.00698754],
    [0.26611111, 0.10471451, 0.21722223, 0.03767747],
    [0.40770083, 0.1090374 , 0.32745153, 0.06193906]]),
 'class_count_': array([38., 36., 38.]),
 'class_prior_': array([0.33928571, 0.32142857, 0.33928571])}
```

### 13.3.1. Scoring a model

Estimator objects also have a score method. If the estimator is a classifier, that score is accuracy. We will see that for other types of estimators it is different types.

```
gnb.score(X_test,y_test)
```

```
0.9210526315789473
```

## 13.4. Making model predictions

we can predict for each sample as well:

```
y_pred = gnb.predict(X_test)
```

### ! Important

in the end of class I tried to demo this and got an error

We can also do one single sample, the `iloc` attribute lets us pick out rows by integer index even if that is not the actual index of the DataFrame

```
X_test.iloc[0]
```

```
sepal_length    5.8
sepal_width     2.7
petal_length    3.9
petal_width     1.2
Name: 82, dtype: float64
```

but if we pick one row, it returns a series, which is incompatible with the predict method.

```
gnb.predict(X_test.iloc[0])
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have numerical features: [u'petal length', u'petal width']
```

```
-----  
ValueError                                Traceback (most recent call last)  
Cell In[17], line 1  
----> 1 gnb.predict(X_test.iloc[0])  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/naive_bayes.py:101, in _E  
    87 """  
    88 Perform classification on an array of test vectors X.  
    89  
(...)  
    98     Predicted target values for X.  
    99 """  
100 check_is_fitted(self)  
--> 101 X = self._check_X(X)  
102 jll = self._joint_log_likelihood(X)  
103 return self.classes_[np.argmax(jll, axis=1)]  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/naive_bayes.py:269, in Ga  
    267 def _check_X(self, X):  
    268     """Validate X, used only in predict* methods."""  
--> 269     return self._validate_data(X, reset=False)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/base.py:605, in BaseEstim  
    603     out = X, y  
    604 elif not no_val_X and no_val_y:  
--> 605     out = check_array(X, input_name="X", **check_params)  
    606 elif no_val_X and not no_val_y:  
    607     out = _check_y(y, **check_params)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/utils/validation.py:938,  
    936     # If input is 1D raise error  
    937     if array.ndim == 1:  
--> 938         raise ValueError(  
    939             "Expected 2D array, got 1D array instead:\narray={}.\\n"  
    940             "Reshape your data either using array.reshape(-1, 1) if "  
    941             "your data has a single feature or array.reshape(1, -1) "  
    942             "if it contains a single sample.".format(array)  
    943         )  
    945 if dtype_numeric and hasattr(array.dtype, "kind") and array.dtype.kind in "USV":  
    946     raise ValueError(  
    947         "dtype='numeric' is not compatible with arrays of bytes/strings."  
    948         "Convert your data to numeric values explicitly instead."  
    949     )  
  
ValueError: Expected 2D array, got 1D array instead:  
array=[5.8 2.7 3.9 1.2].  
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape(1,
```

If we select with a range, that only includes 1, it still returns a DataFrame

```
X_test.iloc[0:1]
```

	sepal_length	sepal_width	petal_length	petal_width
	82	5.8	2.7	3.9

which we can get a prediction for:

[Skip to main content](#)

```
gnb.predict(X_test.iloc[0:1])
```

```
array(['versicolor'], dtype='<U10')
```

We could also transform with `to_frame` and then transpose with `T` or (`transpose`)

```
gnb.predict(X_test.iloc[0].to_frame().T)
```

```
array(['versicolor'], dtype='<U10')
```

We can also pass a 2D array (list of lists) with values in it (here I typed in values similar to the mean for setosa above)

```
gnb.predict([[5.1, 3.6, 1.5, 0.25]])
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, so we cannot determine class weights. Set n_classes=3 or set class_weight='balanced' to avoid this warning.
```

```
array(['setosa'], dtype='<U10')
```

This way it warns us that the feature names are missing, but it still gives a prediction.

### 13.4.1. Evaluating Performance in more detail

```
confusion_matrix(y_test,y_pred)
```

```
array([[12,  0,  0],
       [ 0, 13,  1],
       [ 0,  2, 10]])
```

This is a little harder to read than the 2D version but we can make it a dataframe to read it better.

```
n_classes = len(gnb.classes_)
prediction_labels = [['predicted class']*n_classes, gnb.classes_]
actual_labels = [['true class']*n_classes, gnb.classes_]
conf_mat = confusion_matrix(y_test,y_pred)
conf_df = pd.DataFrame(data = conf_mat, index=actual_labels, columns=prediction_labels)
```

▶ Show code cell content

1 flowers were mistakenly classified as ‘versicolor’ when they were really ‘virginica’

This report is also available:

[Skip to main content](#)

```
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	12
versicolor	0.87	0.93	0.90	14
virginica	0.91	0.83	0.87	12
accuracy			0.92	38
macro avg	0.93	0.92	0.92	38
weighted avg	0.92	0.92	0.92	38

We can also get a report with a few metrics.

- Recall is the percent of each species that were predicted correctly.
- Precision is the percent of the ones predicted to be in a species that are truly that species.
- the F1 score is combination of the two

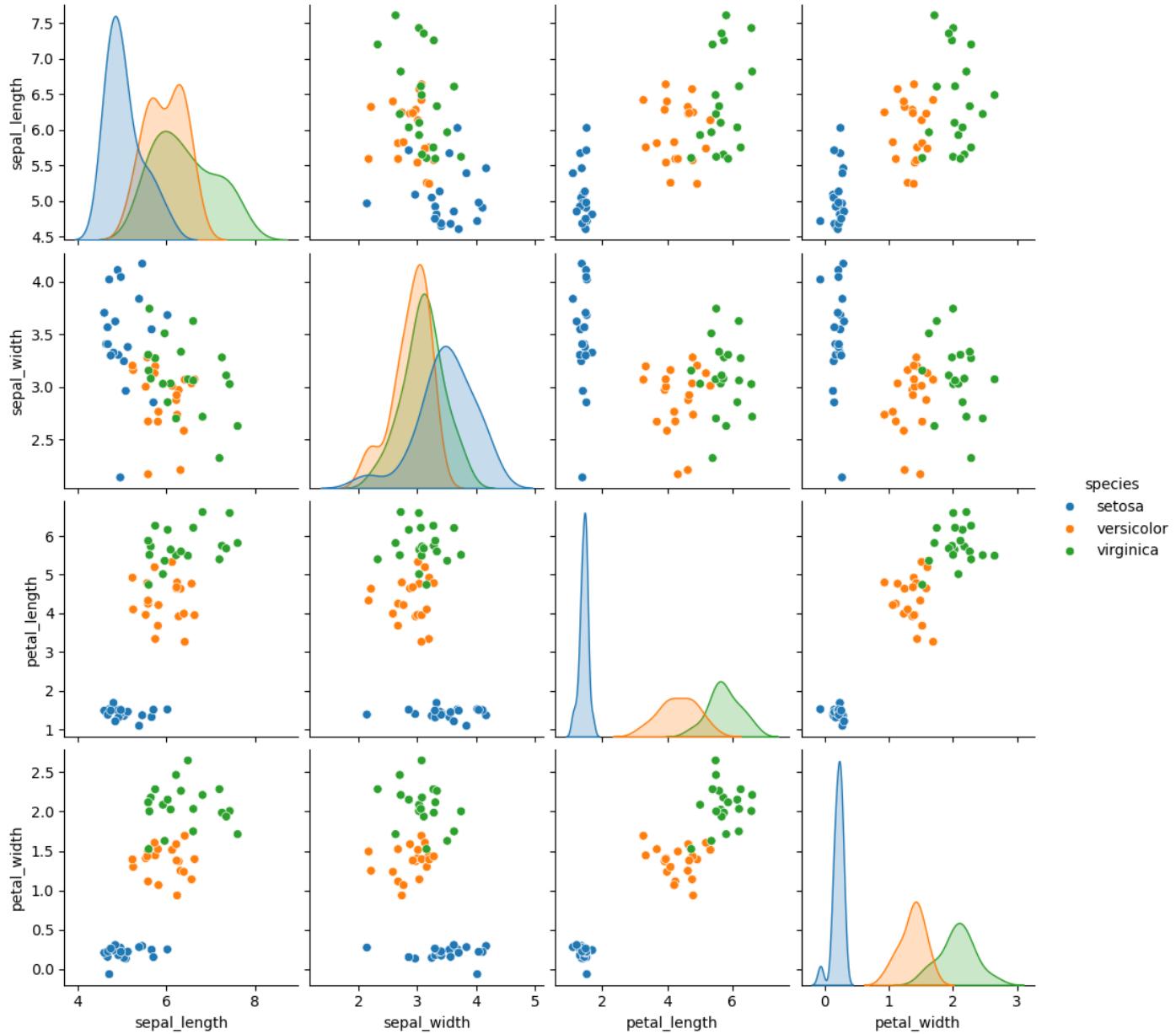
We see we have perfect recall and precision for setosa, as above, but we have lower for the other two because there were mistakes where versicolor and virginica were mixed up.

## 13.5. What does a generative model mean?

Gaussian Naive Bayes is a very simple model, but it is a **generative** model (in contrast to a discriminative model) so we can use it to generate synthetic data that looks like the real data, based on what the model learned.

```
N = 20
n_features = len(feature_vars)
gnb_df = pd.DataFrame(np.concatenate([np.random.multivariate_normal(theta, np.eye(n_features), N)
                                         for theta in zip(gnb.theta_, gnb.var_)]),
                      columns=gnb.feature_names_in_)
gnb_df['species'] = [ci for cl in [[c]*N for c in gnb.classes_] for ci in cl]
sns.pairplot(data=gnb_df, hue='species')
```

```
<seaborn.axisgrid.PairGrid at 0x7f0f47376580>
```



To break this code down:

To do this, we extract the mean and variance parameters from the model (`gnb.theta_`, `gnb.sigma_`) and `zip` them together to create an iterable object that in each iteration returns one value from each list (`for th, sig in zip(gnb.theta_, gnb.sigma_)`). We do this inside of a list comprehension and for each `th, sig` where `th` is from `gnb.theta_` and `sig` is from `gnb.sigma_` we use `np.random.multivariate_normal` to get 20 samples. In a general multivariate normal distribution the second parameter is actually a covariance matrix. This describes both the variance of each individual feature and the correlation of the features. Since Naive Bayes is Naive it assumes the features are independent or have 0 correlation. So, to create the matrix from the vector of variances we multiply by `np.eye(4)` which is the identity matrix or a matrix with 1 on the diagonal and 0 elsewhere. Finally we stack the groups for each species together with `np.concatenate` (like `pd.concat` but works on numpy objects and `np.random.multivariate_normal` returns numpy arrays not data frames) and put all of that in a DataFrame using the feature names as the columns.

[Skip to main content](#)

Then we add a species column, by repeating each species 20 times `[c]*N for c in gnb.classes_` and then unpack that into a single list instead of as list of lists.

## 13.6. How does it make the predictions?

It computes the probability for each class and then predicts the highest one:

```
gnb.predict_proba(X_test)
```

```
array([[4.98983163e-068, 9.99976886e-001, 2.31140535e-005],
       [2.38605399e-151, 6.05788555e-001, 3.94211445e-001],
       [1.52930285e-229, 6.40965644e-007, 9.99999359e-001],
       [1.00000000e+000, 3.59116140e-019, 4.35301434e-027],
       [2.95380506e-300, 1.34947916e-012, 1.00000000e+000],
       [2.16647395e-038, 9.99999791e-001, 2.08857492e-007],
       [1.00000000e+000, 1.05055556e-017, 6.59080633e-026],
       [9.55629438e-147, 1.08807974e-001, 8.91192026e-001],
       [1.00000000e+000, 2.52482409e-016, 1.46364748e-024],
       [6.02158895e-108, 9.90558853e-001, 9.44114652e-003],
       [2.10951969e-108, 9.88848490e-001, 1.11515105e-002],
       [1.45542701e-135, 7.56061808e-001, 2.43938192e-001],
       [8.96514127e-239, 1.55965373e-007, 9.99999844e-001],
       [5.55336438e-231, 1.60582831e-006, 9.99998394e-001],
       [1.00000000e+000, 3.60824506e-014, 6.21633393e-022],
       [1.00000000e+000, 1.29808311e-010, 2.24314344e-018],
       [3.09565382e-153, 1.07603601e-001, 8.92396399e-001],
       [3.81317125e-230, 1.03976329e-006, 9.99998960e-001],
       [1.00000000e+000, 1.37507763e-015, 5.26100959e-023],
       [1.00000000e+000, 1.15359632e-018, 7.80287600e-027],
       [2.60921281e-078, 9.99821175e-001, 1.78824945e-004],
       [2.59248282e-219, 8.60114059e-006, 9.99991399e-001],
       [1.00000000e+000, 2.18609042e-018, 7.77596623e-026],
       [2.27597506e-127, 9.40305572e-001, 5.96944278e-002],
       [8.22137518e-080, 9.99794097e-001, 2.05903463e-004],
       [1.19322368e-231, 1.64825193e-007, 9.99999835e-001],
       [2.39789249e-089, 9.98853183e-001, 1.14681677e-003],
       [1.09087833e-083, 9.99736540e-001, 2.63460309e-004],
       [2.25963865e-110, 9.88633468e-001, 1.13665324e-002],
       [3.92002893e-231, 1.15859357e-006, 9.99998841e-001],
       [1.00000000e+000, 7.65551108e-018, 1.19398607e-025],
       [1.76892853e-077, 9.99969673e-001, 3.03269074e-005],
       [6.37930105e-045, 9.99999177e-001, 8.23423901e-007],
       [1.00000000e+000, 3.51305986e-018, 1.33290716e-025],
       [3.91957440e-113, 8.57857810e-001, 1.42142190e-001],
       [1.00000000e+000, 2.70877085e-018, 3.42709168e-026],
       [9.9999997e-001, 2.88681309e-009, 1.12890422e-016],
       [4.10240137e-278, 9.44646226e-011, 1.00000000e+000]])
```

we can also plot these

```

# make the probabilities into a dataframe labeled with classes & make the index a separate column
prob_df = pd.DataFrame(data = gnb.predict_proba(X_test), columns = gnb.classes_).reset_index()
# add the predictions
prob_df['predicted_species'] = y_pred
prob_df['true_species'] = y_test.values
# for plotting, make a column that combines the index & prediction
pred_text = lambda r: str(r['index']) + ',' + r['predicted_species']
prob_df['i,pred'] = prob_df.apply(pred_text, axis=1)
# same for ground truth
true_text = lambda r: str(r['index']) + ',' + r['true_species']
prob_df['correct'] = prob_df['predicted_species'] == prob_df['true_species']
# add a column for which are correct
prob_df['i,true'] = prob_df.apply(true_text, axis=1)
prob_df_melted = prob_df.melt(id_vars =[ 'index', 'predicted_species','true_species','i,pred','i,true','i,true'],
                               var_name = target_var, value_name = 'probability')
prob_df_melted.head()

```

	index	predicted_species	true_species	i,pred	i,true	correct	species	probability
0	0	versicolor	versicolor	0,versicolor	0,versicolor	True	setosa	4.989832e-68
1	1	versicolor	virginica	1,versicolor	1,virginica	False	setosa	2.386054e-151
2	2	virginica	virginica	2,virginica	2,virginica	True	setosa	1.529303e-229
3	3	setosa	setosa	3,setosa	3,setosa	True	setosa	1.000000e+00
4	4	virginica	virginica	4,virginica	4,virginica	True	setosa	2.953805e-300

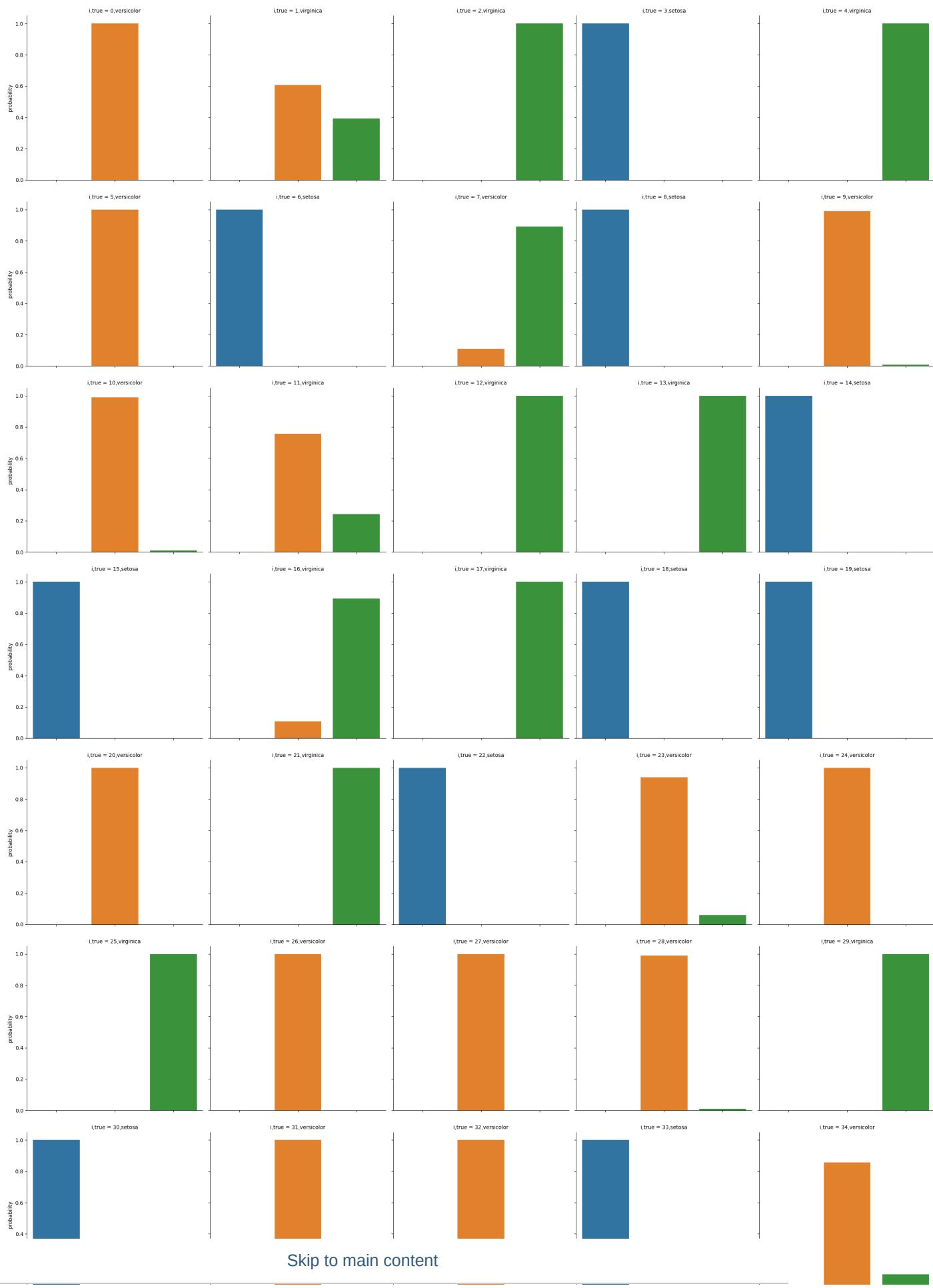
and then we can plot this:

```

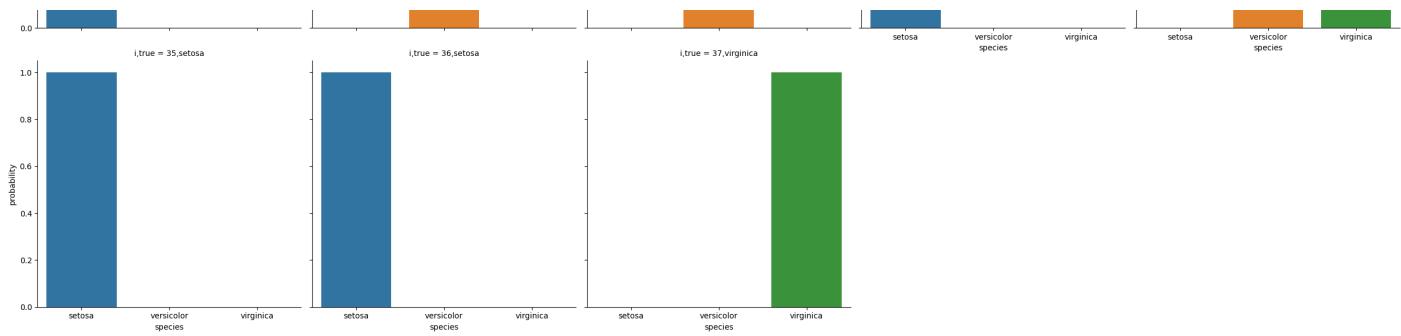
# plot a bar graph for each point labeled with the prediction
sns.catplot(data =prob_df_melted, x = 'species', y='probability' ,col ='i,true',
            col_wrap=5,kind='bar', hue='species')

```

```
<seaborn.axisgrid.FacetGrid at 0x7f0f39f47e80>
```



Skip to main content

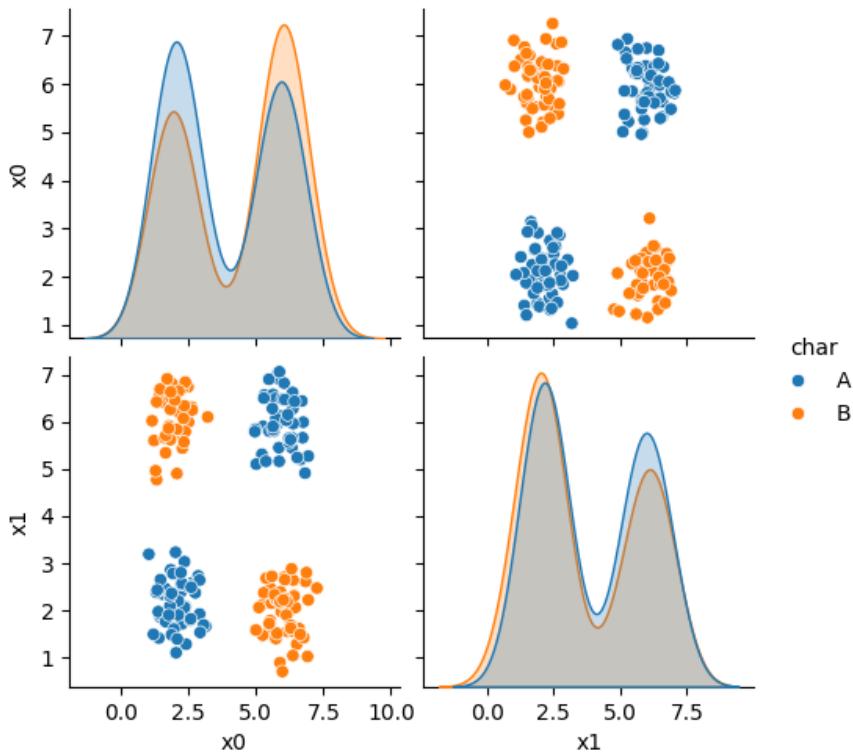


## 13.7. What if the assumptions are not met?

Using a toy dataset here shows an easy to see challenge for the classifier that we have seen so far. Real datasets will be hard in different ways, and since they're higher dimensional, it's harder to visualize the cause.

```
corner_data = 'https://raw.githubusercontent.com/rhodyprog4ds/06-naive-bayes/f425ba121cc0c4dd8bcaa7ebb2f'
df6= pd.read_csv(corner_data,usecols=[1,2,3])
sns.pairplot(data=df6, hue='char',hue_order=['A', 'B'])
```

<seaborn.axisgrid.PairGrid at 0x7f0f4176efa0>



As we can see in this dataset, these classes are quite separated.

```
X_train, X_test, y_train, y_test = train_test_split(df6[['x0', 'x1']],
df6['char'],
random_state = 4)
```

[Skip to main content](#)

```
gnb_corners = GaussianNB()
gnb_corners.fit(X_train,y_train)
gnb_corners.score(X_test, y_test)
```

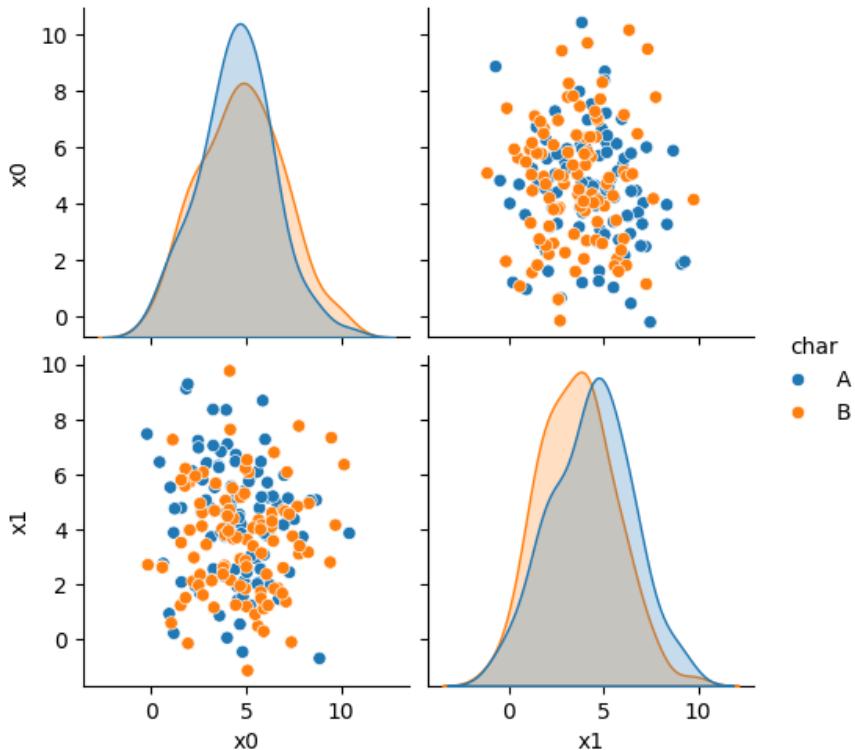
```
0.72
```

But we do not get a very good classification score.

To see why, we can look at what it learned.

```
N = 100
gnb_df = pd.DataFrame(np.concatenate([np.random.multivariate_normal(theta, sig*np.eye(2), N)
    for theta, sig in zip(gnb_corners.theta_, gnb_corners.var_)]),
    columns = ['x0', 'x1'])
gnb_df['char'] = [ci for cl in [[c]*N for c in gnb_corners.classes_] for ci in cl]
sns.pairplot(data = gnb_df, hue='char', hue_order=['A', 'B'])
```

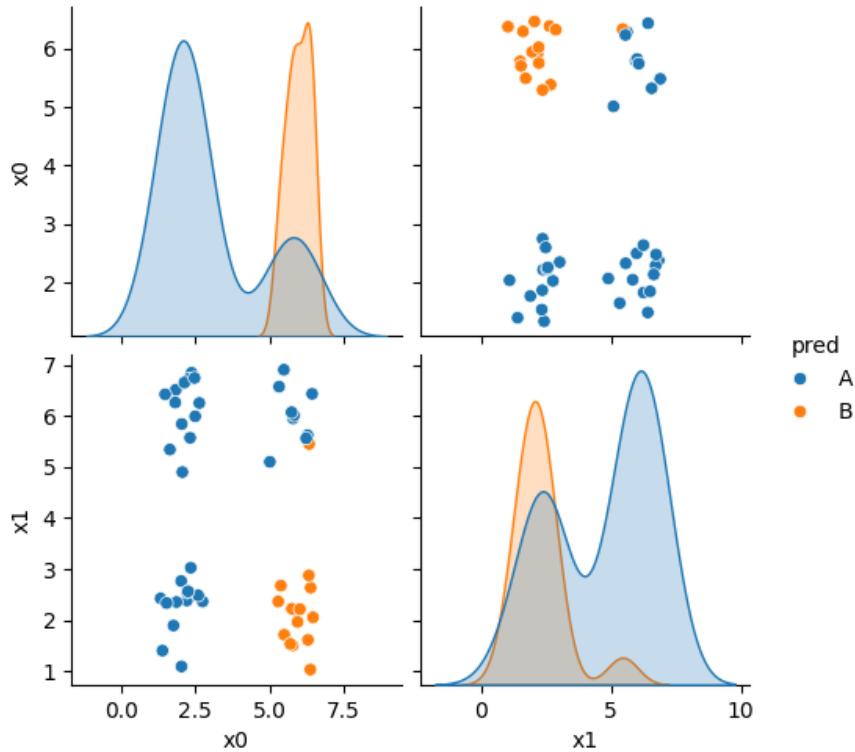
```
<seaborn.axisgrid.PairGrid at 0x7f0f33ee9070>
```



```
df6_pred = X_test.copy()
df6_pred['pred'] = gnb_corners.predict(X_test)
```

```
sns.pairplot(data = df6_pred, hue = 'pred', hue_order = ['A', 'B'])
```

```
<seaborn.axisgrid.PairGrid at 0x7f0f33c88910>
```



This does not look much like the data and it's hard to tell which is higher at any given point in the 2D space. We know though, that it has missed the mark. We can also look at the actual predictions.

If you try this again, split, fit, plot, it will learn different decisions, but always at least about 25% of the data will have to be classified incorrectly.

## 13.8. Decision Trees

This data does not fit the assumptions of the Naive Bayes model, but a decision tree has a different rule. It can be more complex, but for the scikit learn one relies on splitting the data at a series of points along one feature at a time, sequentially. It basically learns a flowchart for deciding what class the sample belongs to at test time

It is a discriminative model, because it describes how to discriminate (in the sense of differentiate) between the classes. This is in contrast to the **generative** model that describes how the data is distributed.

That said, sklearn makes using new classifiers *really* easy once have learned one. All of the classifiers have the same API (same methods and attributes).

```
dt = tree.DecisionTreeClassifier()  
dt.fit(X_train,y_train)  
dt.score(X_test,y_test)
```

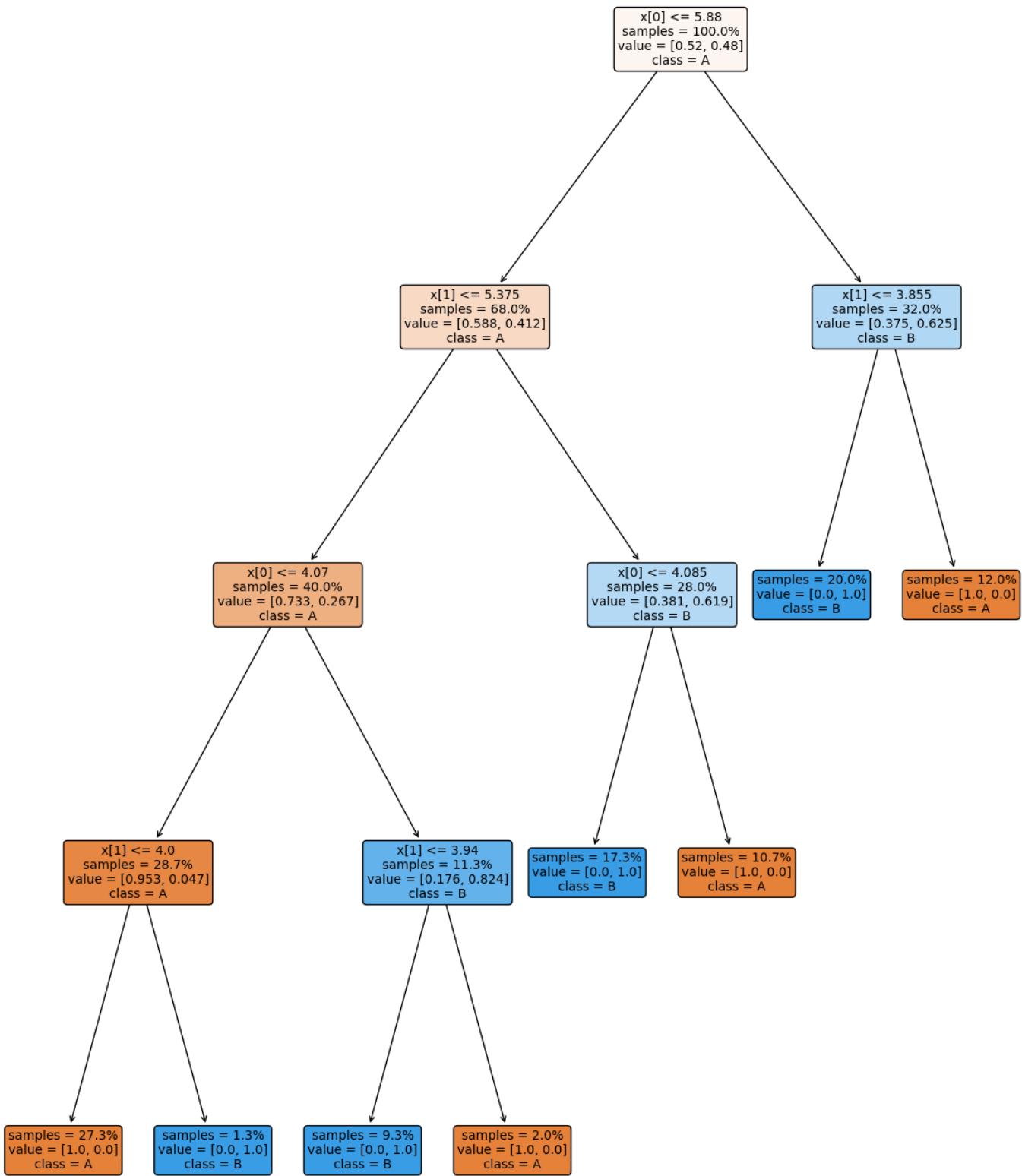
1.0

[Skip to main content](#)

The sklearn estimator objects (that correspond to different models) all have the same API, so the `fit`, `predict`, and `score` methods are the same as above. We will see this also in regression and clustering. What each method does in terms of the specific calculations will vary depending on the model, but they're always there.

the `tree` module also allows you to plot the tree to examine it.

```
plt.figure(figsize=(15,20))
tree.plot_tree(dt, rounded =True, class_names = ['A','B'],
               proportion=True, filled =True, impurity=False, fontsize=10);
```



On the iris dataset, the sklearn docs include a diagram showing the decision boundary. You should be able to modify this for another classifier.

## 13.9. Setting Classifier Parameters

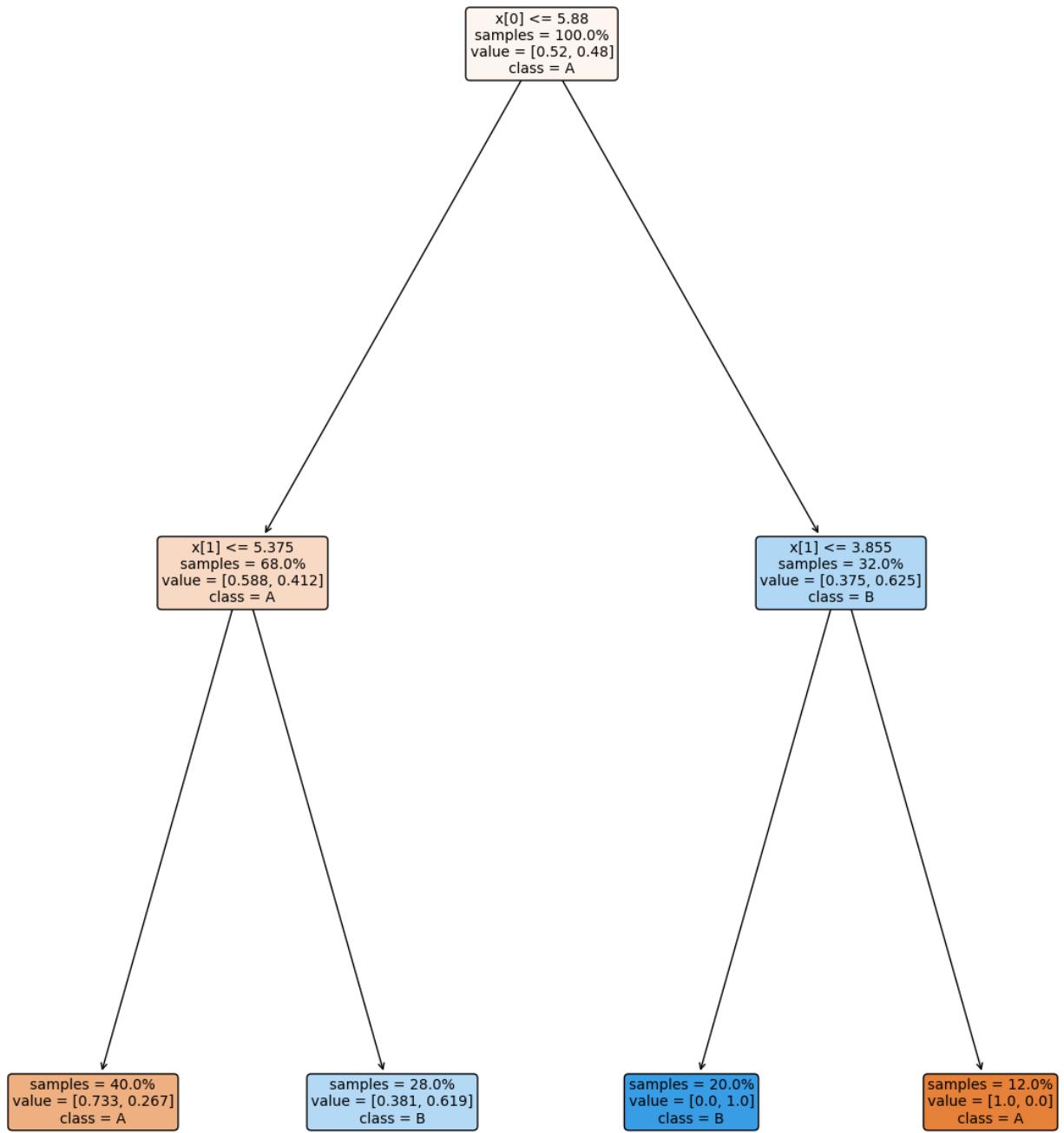
The decision tree we had above has a lot more layers than we would expect. This is really simple data so we still got perfect classification. However, the more complex the model, the more risk that it will learn something noisy about the training data that doesn't hold up in the test set.

Fortunately, we can control the parameters to make it find a simpler decision boundary.

```
dt2 = tree.DecisionTreeClassifier(max_depth=2)
dt2.fit(X_train,y_train)
dt2.score(X_test,y_test)
```

0.74

```
plt.figure(figsize=(15,20))
tree.plot_tree(dt2, rounded =True, class_names = ['A','B'],
               proportion=True, filled =True, impurity=False, fontsize=10);
```



## 13.10. Questions

### Note

I added in some questions from previous semesters because few questions were asked.

### 13.10.1. Are there any good introductions to ScikitLearn that you are aware of?

Scikit Learn [User Guide](#) is the best one and they have a large example gallery.

### 13.10.2. Are there any popular machine learning models that use decision trees?

Yes, a lot of medical applications do, because since they are easy to understand, it is easier for healthcare providers to trust them.

### 13.10.3. Do predictive algorithms have pros and cons? Or is there a standard?

Each algorithm has different properties and strengths and weaknesses. Some are more popular than others, but they all do have weaknesses.

### 13.10.4. how often should we use confusion matrixes? Would it be better just to check the accuracy without one?

A confusion matrix gives *more detail* on the performance than accuracy alone. If the accuracy was like 99.99 maybe the confusion matrix is not informative, but otherwise, it is generally useful to understand what types of mistakes as context for how you might use/not use/ trust the model.

### 13.10.5. Due to the initial ‘shuffling’ of data: Is it possible to get a seed/shuffle of data split so that it that does much worse in a model?

Yes you can get a bad split, we will see next week a statistical technique that helps us improve this. However, the larger your dataset, the less likely this will happen, so we mostly now just get bigger and bigger datasets.

### 13.10.6. How does `gnb = GaussianNB()` work?

this line calls the `GaussianNB` class constructor method with all default parameters.

This means it creates an object of that type, we can see that like follows

```
gnb_ex = GaussianNB()  
type(gnb_ex)
```

[Skip to main content](#)

```
sklearn.naive_bayes.GaussianNB
```

### 13.10.7. how will level 3s work?

I will update on level 3s in class on Tuesday (and the site before then)

### 13.10.8. Could we use strata to better identify the the data in the toy dataset?

The dataset didn't have any additional columns we could use to stratify this data, this data is simply not a good fit for `GaussianNB` because it does not fit the assumptions. Above I included the Decision Tree example, so you can see a classifier that does work for it.

### 13.10.9. is it possible after training the model to add more data to it ?

The models we will see in class, no. However, there is a thing called "online learning" that involves getting more data on a regular basis to improve its performance.

### 13.10.10. Can you use machine learning for any type of data?

Yes the features for example could be an image instead of four numbers. It could also be text. The basic ideas are the same for more complex data, so we are going to spend a lot of time building your understanding of what ML *is* on simple data. Past students have successfully applied ML in more complex data after this course because once you have a good understanding of the core ideas, applying it to other forms of data is easier to learn on your own.

### 13.10.11. Can we check how well the model did using the `y_test` df?

we could compare them directly or using `score` that does.

```
y_pred == y_test
```

```
/tmp/ipykernel_2036/1082399451.py:1: FutureWarning: elementwise comparison failed; returning scalar instead of array for consistency: y_pred == y_test
```

```
-----  
ValueError                                Traceback (most recent call last)  
Cell In[41], line 1  
----> 1 y_pred == y_test  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/ops/common.py:81, in  
    77         return NotImplemented  
    79 other = item_from_zerodim(other)  
---> 81 return method(self, other)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/arraylike.py:40, in  
    38 @unpack_zerodim_and_defer("__eq__")  
    39 def __eq__(self, other):  
---> 40     return self._cmp_method(other, operator.eq)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/series.py:6096, in Series  
    6093 rvalues = extract_array(other, extract_numpy=True, extract_range=True)  
    6095 with np.errstate(all="ignore"):  
-> 6096     res_values = ops.comparison_op(lvalues, rvalues, op)  
    6098 return self._construct_result(res_values, name=res_name)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/ops/array_ops.py:270,  
    265 if isinstance(rvalues, (np.ndarray, ABCExtensionArray)):  
    266     # TODO: make this treatment consistent across ops and classes.  
    267     # We are not catching all listlikes here (e.g. frozenset, tuple)  
    268     # The ambiguous case is object-dtype. See GH#27803  
    269     if len(lvalues) != len(rvalues):  
---> 270         raise ValueError(  
    271             "Lengths must match to compare", lvalues.shape, rvalues.shape  
    272         )  
    274 if should_extension_dispatch(lvalues, rvalues) or (  
    275     (isinstance(rvalues, (Timedelta, BaseOffset, Timestamp)) or right is NaT)  
    276     and not is_object_dtype(lvalues.dtype)  
    277 ):  
    278     # Call the method on lvalues  
    279     res_values = op(lvalues, rvalues)  
  
ValueError: ('Lengths must match to compare', (50,), (38,))
```

```
sum(y_pred == y_test)/len(y_test)
```

```
/tmp/ipykernel_2036/369106620.py:1: FutureWarning: elementwise comparison failed; returning scalar instead
sum(y_pred == y_test)/len(y_test)
```

```
-----
ValueError                                Traceback (most recent call last)
Cell In[42], line 1
----> 1 sum(y_pred == y_test)/len(y_test)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/ops/common.py:81, in 
    77         return NotImplemented
    79 other = item_from_zerodim(other)
--> 81 return method(self, other)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/arraylike.py:40, in 
    38 @unpack_zerodim_and_defer("__eq__")
    39 def __eq__(self, other):
--> 40     return self._cmp_method(other, operator.eq)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/series.py:6096, in Se
    6093 rvalues = extract_array(other, extract_numpy=True, extract_range=True)
    6095 with np.errstate(all="ignore"):
-> 6096     res_values = ops.comparison_op(lvalues, rvalues, op)
    6098 return self._construct_result(res_values, name=res_name)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/ops/array_ops.py:270,
    265 if isinstance(rvalues, (np.ndarray, ABCExtensionArray)):
    266     # TODO: make this treatment consistent across ops and classes.
    267     # We are not catching all listlikes here (e.g. frozenset, tuple)
    268     # The ambiguous case is object-dtype. See GH#27803
    269     if len(lvalues) != len(rvalues):
--> 270         raise ValueError(
    271             "Lengths must match to compare", lvalues.shape, rvalues.shape
    272         )
    274 if should_extension_dispatch(lvalues, rvalues) or (
    275     (isinstance(rvalues, (Timedelta, BaseOffset, Timestamp)) or right is NaT)
    276     and not is_object_dtype(lvalues.dtype)
    277 ):
    278     # Call the method on lvalues
    279     res_values = op(lvalues, rvalues)

ValueError: ('Lengths must match to compare', (50,), (38,))
```

```
gnb.score(X_test,y_test)
```

```
-----
ValueError                                Traceback (most recent call last)
Cell In[43], line 1
----> 1 gnb.score(X_test,y_test)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/base.py:706, in ClassifierMixin.score(self, X, y)
    681 """
    682 Return the mean accuracy on the given test data and labels.
    683
(...):
    702     Mean accuracy of ``self.predict(X)`` w.r.t. `y`.
    703 """
    704 from .metrics import accuracy_score
--> 706 return accuracy_score(y, self.predict(X), sample_weight=sample_weight)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/naive_bayes.py:101, in _E
    87 """
    88 Perform classification on an array of test vectors X.
    89
(...):
    98     Predicted target values for X.
    99 """
100 check_is_fitted(self)
--> 101 X = self._check_X(X)
102 jll = self._joint_log_likelihood(X)
103 return self.classes_[np.argmax(jll, axis=1)]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/naive_bayes.py:269, in GaussianNB._check_X(self, X):
    267 def _check_X(self, X):
    268     """Validate X, used only in predict* methods."""
--> 269     return self._validate_data(X, reset=False)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/base.py:580, in BaseEstimator._validate_data(self, X, y, X_name='X', y_name='y', validate_params=True, reset=False)
    509 def _validate_data(
    510     self,
    511     X="no_validation",
(...):
    516     **check_params,
    517 ):
    518     """Validate input data and set or check the `n_features_in_` attribute.
    519
    520     Parameters
(...):
    578         validated.
    579     """
--> 580     self._check_feature_names(X, reset=reset)
582     if y is None and self._get_tags()["requires_y"]:
583         raise ValueError(
584             f"This {self.__class__.__name__} estimator "
585             "requires y to be passed, but the target y is None."
586         )

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/base.py:507, in BaseEstimator._check_feature_names(self, X, reset=False)
    502 if not missing_names and not unexpected_names:
    503     message += (
    504         "Feature names must be in the same order as they were in fit.\n"
    505     )
--> 507 raise ValueError(message)

ValueError: The feature names should match those that were passed during fit.
Feature names unseen at fit time:
- x0
- x1
Feature names seen at fit time, yet now missing:
- petal_length
- petal width
```

[Skip to main content](#)

- sepal\_length
- sepal\_width

We can also use any of the other metrics we saw, we'll practice more on Wednesday

### 13.10.12. I want to know more about the test\_train\_split() function

the docs are a good place to start.

## 14. Clustering

Clustering is unsupervised learning. That means we do not have the labels to learn from. We aim to learn both the labels for each point and some way of characterizing the classes at the same time.

Computationally, this is a harder problem. Mathematically, we can typically solve problems when we have a number of equations equal to or greater than the number of unknowns. For  $N$  data points in  $d$  dimensions and  $K$  clusters, we have  $N$  equations and  $N + K * d$  unknowns. This means we have a harder problem to solve.

For today, we'll see K-means clustering which is defined by  $K$  a number of clusters and a mean (center) for each one. There are other K-centers algorithms for other types of centers.

Clustering is a stochastic (random) algorithm, so it can be a little harder to debug the models and measure performance. For this reason, we are going to look a little more closely at what it actually does than we did with classification.

### 14.1. KMeans

- clustering goal: find groups of samples that are similar
- k-means assumption: a fixed number ( $k$ ) of means will describe the data enough to find the groups

#### Note

This is also a naive and gaussian assumption, but it is one step stronger in that it assumes that every feature has the same covariance

### 14.2. Clustering with Sci-kit Learn

```
import seaborn as sns
import numpy as np
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn import metrics
import pandas as pd
import matplotlib.pyplot as plt
sns.set_theme(palette='colorblind')

# set global random seed so that the notes are the same each time the site builds
np.random.seed(1103)
```

[Skip to main content](#)

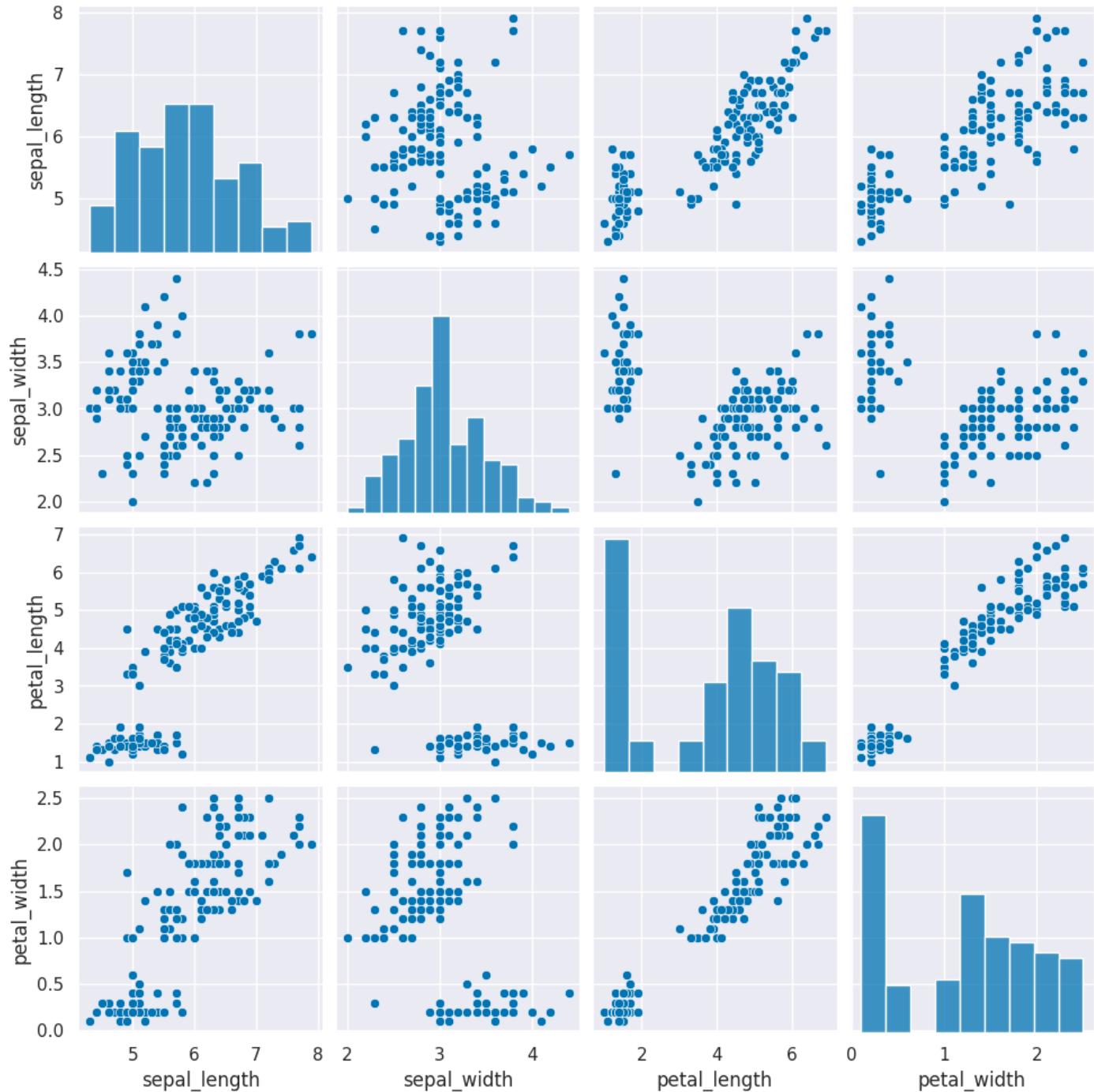
Today we will load the iris data from seaborn:

```
iris_df = sns.load_dataset('iris')
```

this is how the clustering algorithm sees the data, with no labels:

```
sns.pairplot(iris_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7ff4751fef40>
```



[Skip to main content](#)

Next we need to create a copy of the data that's appropriate for clustering. Remember that clustering is *unsupervised* so it doesn't have a target variable. We also can do clustering on the data with or without splitting into test/train splits, since it doesn't use a target variable, we can evaluate how good the clusters it finds are on the actual data that it learned from.

We can either pick the measurements out or drop the species column. remember most data frame operations return a copy of the dataframe.

We will drop here:

```
iris_X = iris_df.drop(columns=['species'])
```

and inspect to see it:

```
iris_X.head()
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Next, we create a Kmeans estimator object with 3 clusters, since we know that the iris data has 3 species of flowers. We refer to these three groups as classes in classification (the goal is to label the classes...) and in clustering we sometimes borrow that word. Sometimes, clustering literature will be more abstract and refer to partitions, this is especially common in more mathematical/statistical work as opposed to algorithmic work on clustering.

```
km = KMeans(n_clusters=3)
```

We dropped the *column* that tells us which of the three classes that each sample(row) belongs to. We still have data from three species of flows.

We can use fit again, but this time it only requires the features, no labels.

```
km.fit(iris_X)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: super().__check_params_vs_input(X, default_n_init=10)
```

▼ KMeans

KMeans(n\_clusters=3)

We see it learns similar, but fewer, parameters:

[Skip to main content](#)

```
km.__dict__
```

```
{'n_clusters': 3,
 'init': 'k-means++',
 'max_iter': 300,
 'tol': 0.0001,
 'n_init': 'warn',
 'verbose': 0,
 'random_state': None,
 'copy_X': True,
 'algorithm': 'lloyd',
 'feature_names_in_': array(['sepal_length', 'sepal_width', 'petal_length', 'petal_width'],
    dtype=object),
 'n_features_in_': 4,
 '_tol': 0.0001135617666666667,
 '_n_init': 10,
 '_algorithm': 'lloyd',
 '_n_threads': 2,
 'cluster_centers_': array([[5.006      , 3.428      , 1.462      , 0.246      ],
 [5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
 [6.85       , 3.07368421, 5.74210526, 2.07105263]]),
 '_n_features_out': 3,
 'labels_': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 2, 2, 2, 1, 1, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 2, 2, 2,
 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1], dtype=int32),
 'inertia_': 78.851441426146,
 'n_iter_': 6}
```

### 💡 Hint

use shift+tab or another jupyter help to figure out what the parameter names are for any function or class you're working with.

Since we don't have separate test and train data, we can use the `fit_predict` method. This is what the kmeans algorithm always does anyway, it both learns the means and the assignment (or prediction) for each sample at the same time.

This gives the labeled cluster by index, or the assignment, of each point.

If we run that a few times, we will see different solutions each time because the algorithm is random, or stochastic.

These are similar to the outputs in classification, except that in classification, it's able to tell us a specific species for each. Here it can only say clust 0, 1, or 2. It can't match those groups to the species of flower.

## 14.3. Visualizing the outputs

First we'll save it in the dataframe

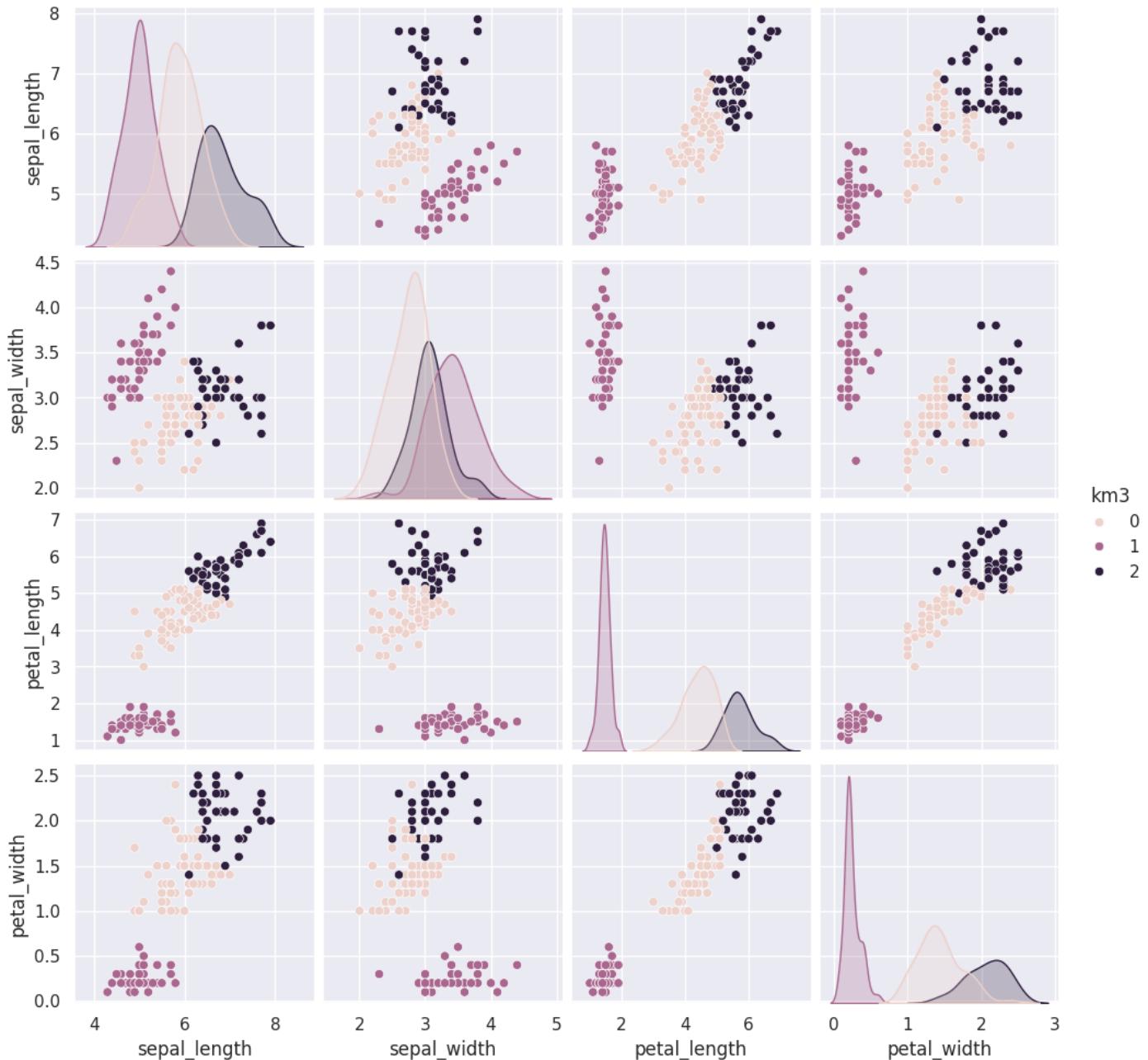
```
iris_df['km3'] = km.fit_predict(iris_X)
```

[Skip to main content](#)

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: super().__check_params_vs_input(X, default_n_init=10)
```

```
    sns.pairplot(data=iris_df, hue='km3')
```

```
<seaborn.axisgrid.PairGrid at 0x7ff471f1b400>
```

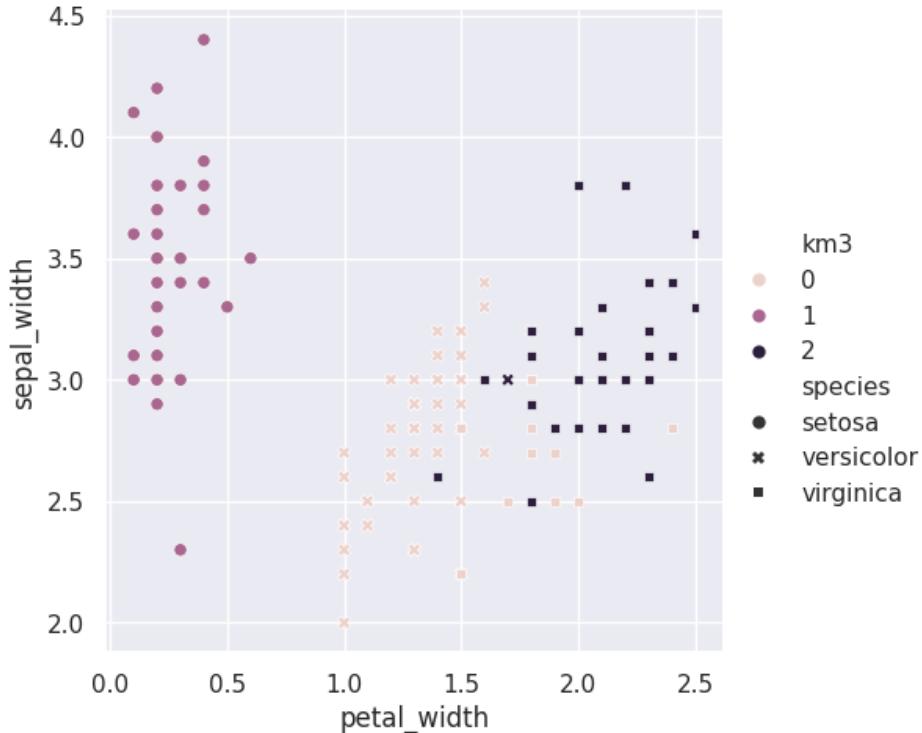


For one pair of features we can look in more detail:

```
    sns.relplot(data=iris_df, x = 'petal_width', y='sepal_width',
                 hue='km3', style='species')
```

[Skip to main content](#)

```
<seaborn.axisgrid.FacetGrid at 0x7ff466bbcd0>
```



here i used the `style` to set the shape and the `hue` to set th color of the markers so that we can see where the ground truth and learned groups agree and disagree.

## 14.4. Clustering Evaluation

$$s = \frac{b - a}{\max(a, b)}$$

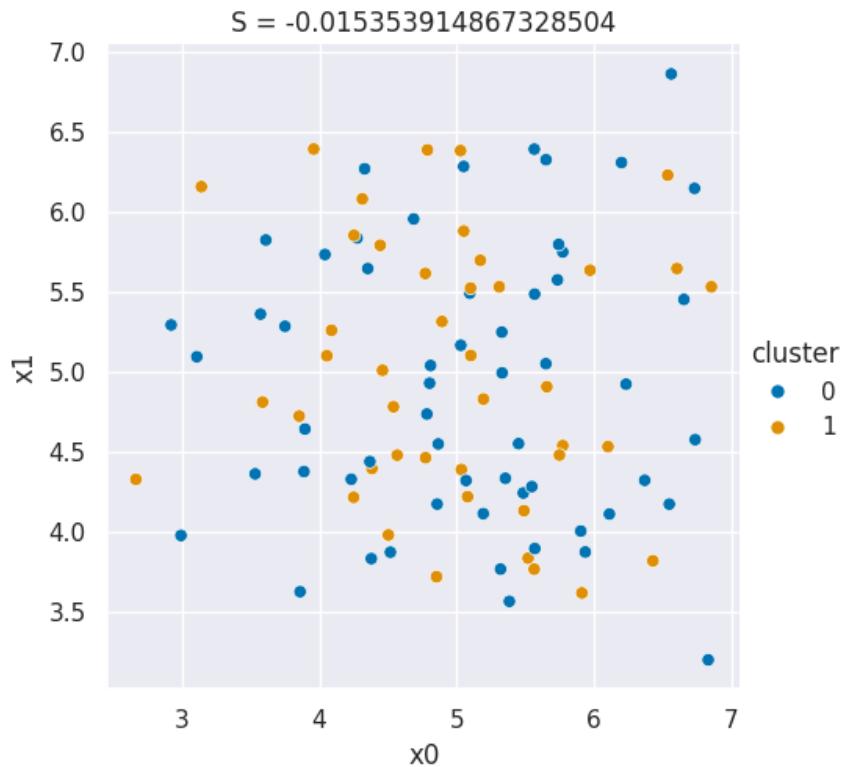
a: The mean distance between a sample and all other points in the same class.

b: The mean distance between a sample and all other points in the next nearest cluster.

This score computes a ratio of how close points are to points in the same cluster vs other clusters

In class, we drew pictures in prismia, but we can also generate samples to make a plot that shows a bad silhouette score:

▶ Show code cell source

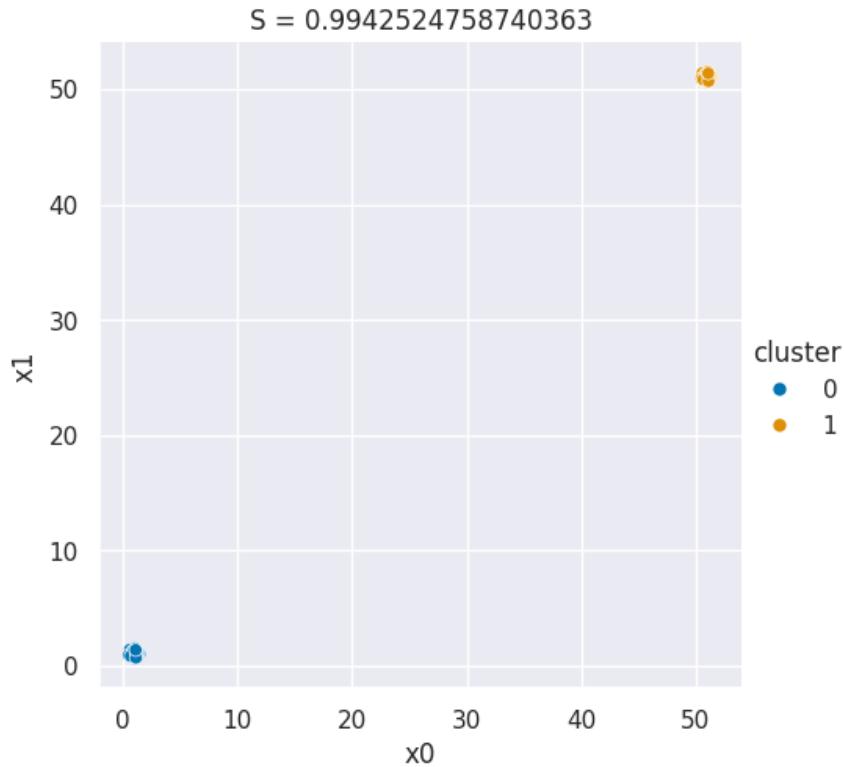


Here, since I generated the samples from one cluster and then assigned the cluster labels independently, randomly, for each point with no regard for its position the clustering score is pretty close to 0.

If, instead, I make more intentional clusters, we can see a better score:

And we can make a plot for a good silhouette score

▶ Show code cell source



If you change the distance in the code above running it you can increase or decrease the score value

Then we can use the silhouette score on our existing clustering solution.

```
metrics.silhouette_score(iris_X,iris_df['km3'])
```

```
0.5528190123564102
```

We can also compare different numbers of clusters, we noted that in class it looked like maybe 2 might be better than 3, so lets look at that:

```
km2 = KMeans(n_clusters=2)
iris_df['km2'] = km2.fit_predict(iris_X)
metrics.silhouette_score(iris_X,iris_df['km2'])
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: super().__check_params_vs_input(X, default_n_init=10)
```

```
0.6810461692117465
```

it is higher, as expected

we can also check 4:

```
km4 = KMeans(n_clusters=4)
iris_df['km4'] = km4.fit_predict(iris_X)
metrics.silhouette_score(iris_X, iris_df['km4'])
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: super().__check_params_vs_input(X, default_n_init=10)
```

```
0.49805050499728815
```

it is not better, we can check more:

```
km10 = KMeans(n_clusters=10)
iris_df['km10'] = km10.fit_predict(iris_X)
metrics.silhouette_score(iris_X, iris_df['km10'])
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: super().__check_params_vs_input(X, default_n_init=10)
```

```
0.32802357624839684
```

also not, so we would say that 2 clusters best describes this data.

## 14.5. Questions after class

### 14.5.1. Is the pairplot slices of a 3d plot?

Sort of, it's actually 2D slices of a 4D plot for the iris data because it has 4 features. We can think of the features like dimensions.

### 14.5.2. Why does the silhouette score not give a score for each predicted group? Since the km3 columns in metrics.silhouette\_score(iris\_X, iris\_df['km3']) is the predicted categories, why is the silhouette score not separated for each km3?

There is not an equivalent of confusion matrix in this concept, but we can get a score per sample and groupby to create a per group average:

```
iris_df['km3_silhouette'] = metrics.silhouette_samples(iris_X, iris_df['km3'])
iris_df.groupby('km3')['km3_silhouette'].mean()
```

```
km3
0    0.417320
1    0.798140
2    0.451105
```

[Skip to main content](#)

# 15. Kmeans algorithm

Computationally, this is a harder problem. Mathematically, we can typically solve problems when we have a number of equations equal to or greater than the number of unknowns. For  $N$  data points in  $d$  dimensions and  $K$  clusters, we have  $N$  equations and  $N + K * d$  unknowns. This means we have a harder problem to solve.

Clustering is a stochastic (random) algorithm, so it can be a little harder to debug the models and measure performance. For this reason, we are going to look a little more closely at what it actually does than we did with classification.

```
import matplotlib.pyplot as plt
import numpy as np
import itertools
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn import metrics
import string
import itertools as it
import imageio.v3 as iio

# set global random seed so that the notes are the same each time the site builds
rand_seed = 398
np.random.seed(rand_seed)
```

## 15.1. How does Kmeans work?

We will start with some synthetic data and then see how the clustering works.

This code should be fairly readable, but is not strictly required. One tricky part is `zip` which is a built-in function for iterating over groups of things together.

And `itertools` is a core library for related more iterating.

One way to understand this block of code is to try changing the parameters (the top section) and see how they effect the plots. The `var` and `spacing` will have the most impact on the clustering performance.

```

# ----- Set parameters, change these to adjust the sample
# number of classes (groups)
C = 4
# number of dimensions of the features
D=2
# number of samples
N = 200
# minimum of the means
offset = 2
# distance between means
spacing = 2
# set the variance of the blobs
var = .25

# ----- Get the class labels
# choose the first C uppercase letters using the builtin string class
classes = list(string.ascii_uppercase[:C])

# ----- Pick some means
# get the number of grid locations needed
G = int(np.ceil(np.sqrt(C)))

# get the locations for each axis
grid_locs = a = np.linspace(offset, offset+G*spacing,G)

# compute grid (i,j) for each combination of values above & keep C values
means = [(i,j) for i, j in it.product(grid_locs,grid_locs)][:C]

# store in dictionary with class labels
mu = {c: i for c, i in zip(classes,means)}

# ----- Sample the data
#randomly choose a class for each point, with equal probability
clusters_true = np.random.choice(classes,N)
# draw a random point according to the means from above for each point
data = [np.random.multivariate_normal(mu[c], var*np.eye(D)) for c in clusters_true]

# ----- Store in a dataframe
# rounding to make display neater later
df = pd.DataFrame(data = data,columns = ['x' + str(i) for i in range(D)]).round(2)

# add true cluster
df['true_cluster'] = clusters_true

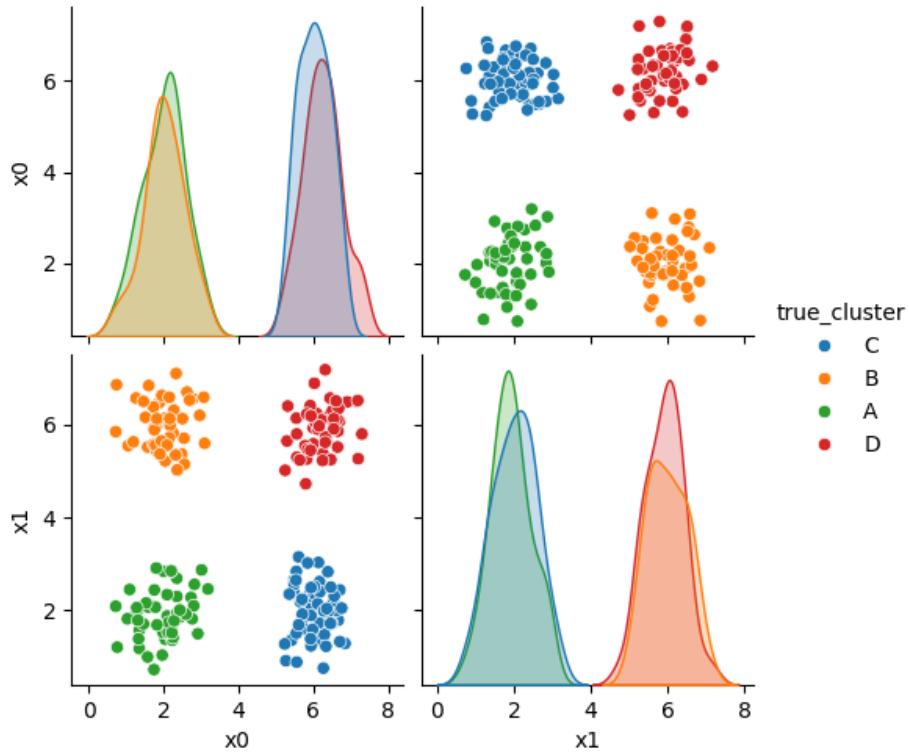
```

This gives us a small dataframe with 2 features (`D=2`) and four clusters (`C=4`).

We can see the data with the labels

```
sns.pairplot(data=df,hue='true_cluster')
```

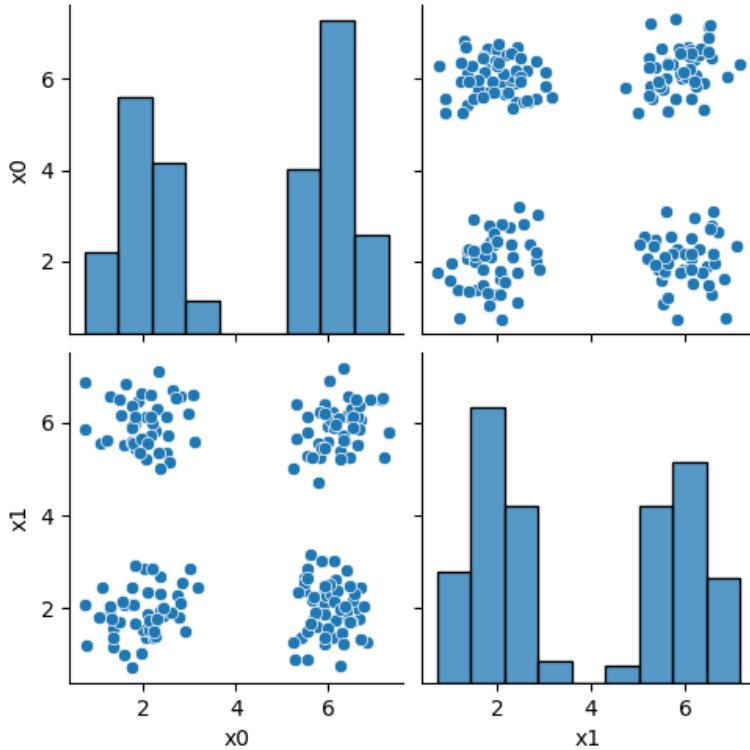
```
<seaborn.axisgrid.PairGrid at 0x7f0aa395cc10>
```



this is what the clustering algorithm will see:

```
{ sns.pairplot(data=df)
```

```
<seaborn.axisgrid.PairGrid at 0x7f0a9f3e2ee0>
```



## 15.2. Setup

First we will make a variable that we can use to pick out the feature columns

```
data_cols = ['x0', 'x1']
```

and we will set a constant as our iteration number

```
i = 0
```

Here, we'll set up some helper code.

For more on color palettes see the seaborn docs

```
def mu_to_df(mu,i):
    mu_df = pd.DataFrame(mu,columns=['x0','x1'])
    mu_df['iteration'] = str(i)
    mu_df['class'] = ['M'+str(i) for i in range(K)]
    mu_df['type'] = 'mu'
    return mu_df

cmap_pt = sns.color_palette('tab20',8)[1::2]
cmap_mu = sns.color_palette('tab20',8)[0::2]
```

You can see that this whole color palette is paired colors, and what the `cmap_pt` and `cmap_mu` do is take odd and even subsets of the palette into two separate palettes.

```
sns.color_palette('tab20', 8)
```



## 15.3. Initializing the algorithm

Kmeans starts by setting `K`, we will use 4, and selecting hat number of random points to be the starting points as the means.

```
K = 4  
mu_0 = df[data_cols].sample(n=K).values  
mu_0
```

```
array([[5.56, 5.79],  
       [5.3 , 5.64],  
       [2.26, 1.42],  
       [6.35, 5.63]])
```

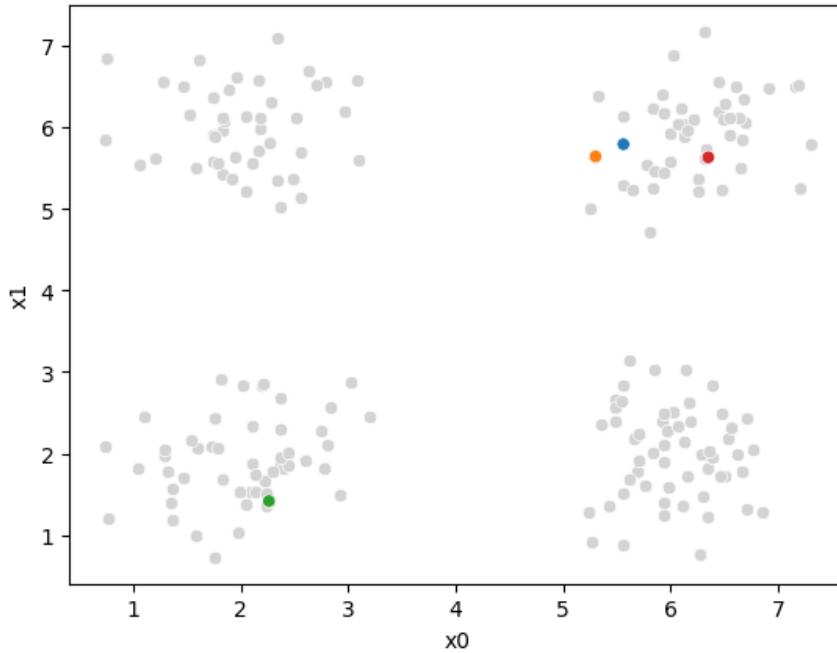
then we will use our helper function to make this work

```
mu_df = mu_to_df(mu_0,i)  
mu_df
```

	x0	x1	iteration	class	type
0	5.56	5.79	0	M0	mu
1	5.30	5.64	0	M1	mu
2	2.26	1.42	0	M2	mu
3	6.35	5.63	0	M3	mu

We can see what our algorithm knows:

```
sfig = sns.scatterplot(data =df,x='x0',y='x1',color='lightgray',legend=False)  
# plt.plot(mu[:,0],mu[:,1],marker='s',linewidth=0)  
mu_df = mu_to_df(mu_0,i)  
sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)  
# save the figure for later use:  
sfig.get_figure().savefig('kmeans00.png')
```



we have all the points, but no labels for them, and we have these 4 points as our means.

## 15.4. Assignment step

Now, we will compute, for each sample which of those four points it is closest to first by taking the difference, squaring it, then summing along each row.

To compute the distance we will first subtract from the mean:

```
(df[data_cols].loc[0] - mu_0[0])
```

```
x0      0.27
x1     -3.78
Name: 0, dtype: float64
```

then square each

```
((df[data_cols].loc[0] - mu_0[0])**2)
```

```
x0      0.0729
x1    14.2884
Name: 0, dtype: float64
```

and sum

```
((df[data_cols].loc[0] - mu_0[0])**2).sum()
```

```
14.361300000000002
```

If we needed the right units on these distances, we would also `sqrt` but, we actually will just use the distances to figure out which mean is closer to the each point. If  $a > b > c$  then  $\sqrt{a} > \sqrt{b} > \sqrt{c}$  so we do not need the `sqrt` to get the right end answer so we can save that step, which saves computation.

we can do this for all of the rows

```
((df[data_cols] - mu_0[0])**2).sum(axis=1)
```

```
0      14.3613
1      20.3125
2      19.9274
3      9.0682
4      15.9188
...
195    32.1458
196    8.7125
197    21.2141
198    0.3200
199    31.0472
Length: 200, dtype: float64
```

then we can do this for all four means:

```
[((df[data_cols] - mu_i)**2).sum(axis=1) for mu_i in mu_0]
```

```
[0    14.3613
1    20.3125
2    19.9274
3     9.0682
4    15.9188
...
195   32.1458
196   8.7125
197  21.2141
198   0.3200
199  31.0472
Length: 200, dtype: float64,
0    13.4578
1    17.9876
2    18.9745
3     7.5661
4    13.8553
...
195   28.8965
196   7.5344
197  20.5642
198   0.2837
199  27.8161
Length: 200, dtype: float64,
0    13.0930
1    18.4144
2    14.8261
3    18.4025
4    17.1905
...
195   1.3505
196  26.2036
197  19.8106
198  25.9405
199   2.4065
Length: 200, dtype: float64,
0    13.3748
1    27.9922
2    18.2905
3    14.4449
4    22.7673
...
195   38.3485
196  14.1146
197  18.6196
198   0.6641
199  37.6749
Length: 200, dtype: float64]
```

This gives us a list of 4 data DataFrames, one for each mean ( $\mu_i$ ), with one row for each point in the dataset with the distance from that point to the corresponding mean. We can concatenate these horizontally (`axis=1`) these into one DataFrame.

```
pd.concat([(df[data_cols]- mu_i)**2].sum(axis=1) for mu_i in mu_0], axis=1).head()
```

	0	1	2	3
0	14.3613	13.4578	13.0930	13.3748
1	20.3125	17.9876	18.4144	27.9922
2	19.9274	18.9745	14.8261	18.2905
3	9.0682	7.5661	18.4025	14.4449
4	15.9188	13.8553	17.1905	22.7673

Now we have one row per sample and one column per mean, with the distance from that point to the mean. What we want is to calculate the assignment, which mean is closest, for each point. Using `idxmin` with `axis=1` we take the minimum across each row and returns the index (location) of that minimum.

```
pd.concat([(df[data_cols]- mu_i)**2).sum(axis=1) for mu_i in mu_0],axis=1).idxmin(axis=1)
```

```
0      2
1      1
2      2
3      1
4      1
..
195     2
196     1
197     3
198     1
199     2
Length: 200, dtype: int64
```

We'll save all of this in a column named `'0'`. Since it is our 0th iteration.

This is called the **assignment** step.

```
df[str(i)] = pd.concat([(df[data_cols]- mu_i)**2).sum(axis=1) for mu_i in mu_0],axis=1).idxmin(axis=1)
```

to see what we have:

```
df.head()
```

	x0	x1	true_cluster	0
0	5.83	2.01	C	2
1	1.06	5.54	B	1
2	6.11	1.36	C	2
3	2.55	5.70	B	1
4	1.58	5.51	B	1

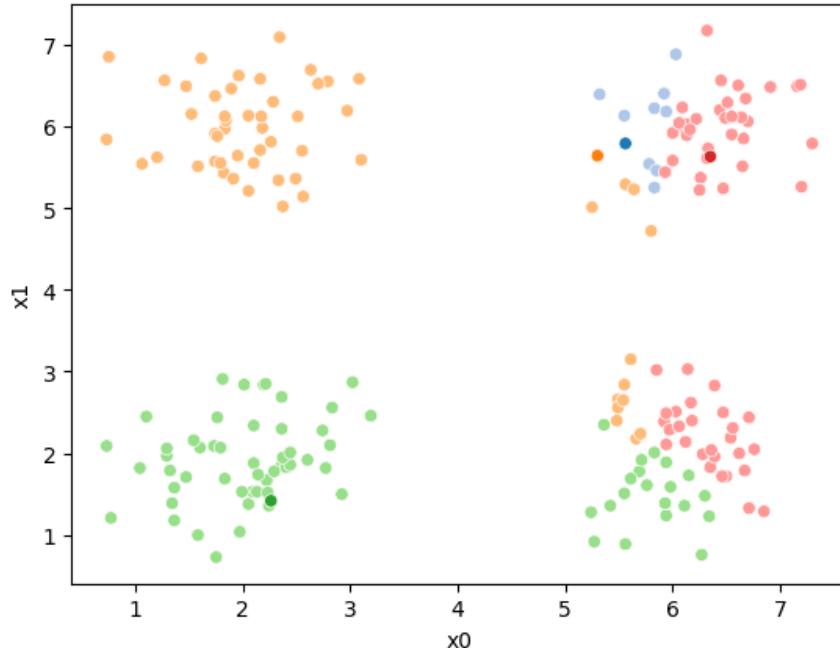
and now we can re-plot, we still have the same means, but now we use the `'0'` column to give the points each a color.

[Skip to main content](#)

Now we can plot the data, save the axis, and plot the means on top of that. Seaborn plotting functions return an axis, by saving that to a variable, we can pass it to the `ax` parameter of another plotting function so that both plotting functions go on the same figure.

```
sfig = sns.scatterplot(data =df,x='x0',y='x1',hue='0',palette=cmap_pt,legend=False)
# plt.plot(mu[:,0],mu[:,1],marker='s',linewidth=0)
mu_df = mu_to_df(mu_0,i)
sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,
                 ax=sfig,legend=False)

sfig.get_figure().savefig('kmeans01.png')
```



We see that each point is assigned to the lighter shade of its matching mean. These points are the one that is closest to each point, but they're not the centers of the point clouds. Now, we can compute new means of the points assigned to each cluster, using groupby.

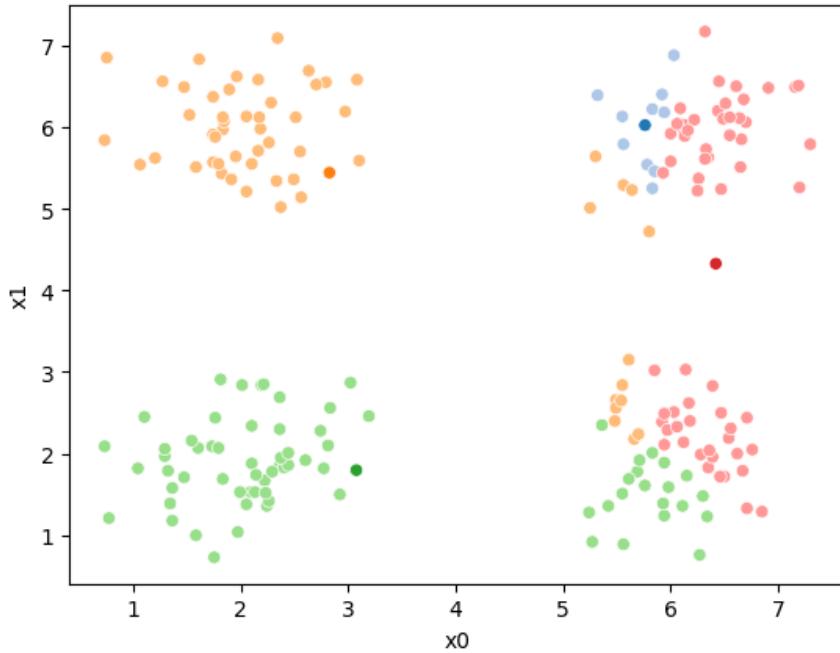
## 15.5. Updating the means

```
mu_1 = df.groupby('0')[data_cols].mean().values
```

We can plot these again, the same data, but with the new means.

```
fig = plt.figure()
mu_df = mu_to_df(mu_1,1)
sfig = sns.scatterplot(data =df,x='x0',y='x1',hue='0',palette=cmap_pt,legend=False)
sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)

sfig.get_figure().savefig('kmeans02.png')
```



We see that now the means are in the center of each cluster, but that there are now points in one color that are assigned to other clusters.

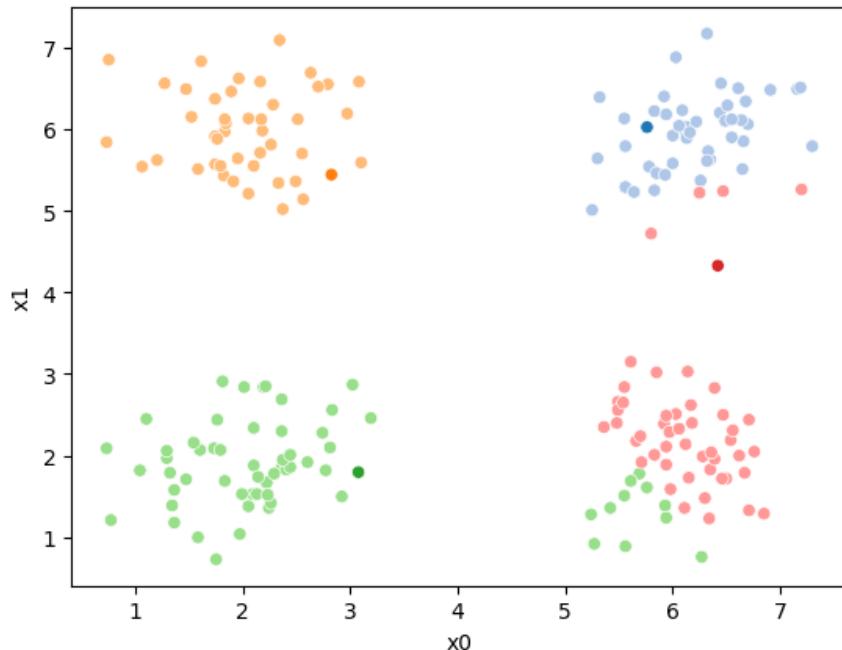
So, again we can update the assignments.

```
i=1 #increment
df[str(i)] = pd.concat([(df[data_cols]-mu_i)**2).sum(axis=1) for mu_i in mu_1],axis=1).idxmin(axis=1)
df.head()
```

	x0	x1	true_cluster	0	1
<b>0</b>	5.83	2.01	C	2	3
<b>1</b>	1.06	5.54	B	1	1
<b>2</b>	6.11	1.36	C	2	3
<b>3</b>	2.55	5.70	B	1	1
<b>4</b>	1.58	5.51	B	1	1

And plot again:

```
sfig = sns.scatterplot(data =df,x='x0',y='x1',hue=str(i),palette=cmap_pt,legend=False)
# plt.plot(mu[:,0],mu[:,1],marker='s',linewidth=0)
mu_df = mu_to_df(mu_1,i)
sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
sfig.get_figure().savefig('kmeans03.png')
```



we see it improves by the means moved to the center of each color.

## 15.6. Iterating to completion

If we keep going back and forth like this, eventually, the assignment step will not change any assignments. We call this condition convergence. We can implement the algorithm with a while loop.

```

mu_list = [mu_to_df(mu_0,0),mu_to_df(mu_1,1)]
cur_old = str(i-1)
cur_new = str(i)
mu = mu_1
number_changed_assignments = sum(df[cur_old] !=df[cur_new])
while number_changed_assignments >0:
    cur_old = cur_new
    i +=1
    cur_new = str(i)
    #     update the means and plot with current generating assignments
    mu = df.groupby(cur_old)[data_cols].mean().values
    mu_df = mu_to_df(mu,i)
    mu_list.append(mu_df)

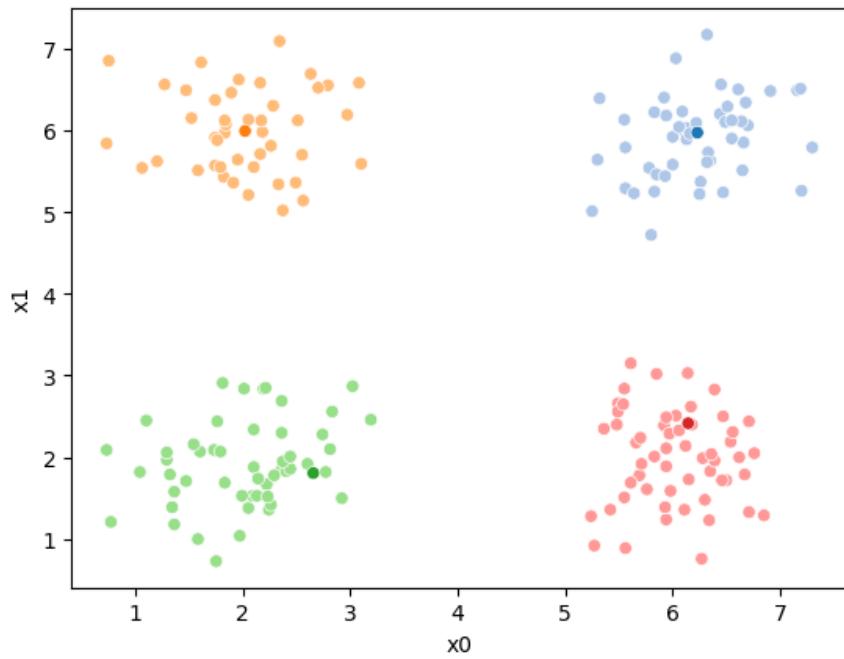
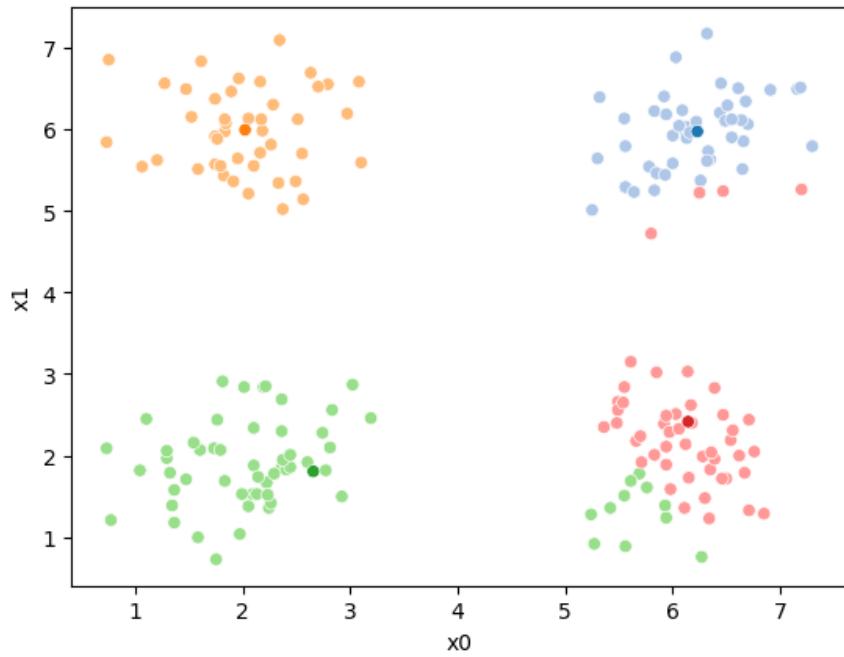
    fig = plt.figure()
    # plot with old assignments
    sfig = sns.scatterplot(data =df,x='x0',y='x1',hue=cur_old,palette=cmap_pt,legend=False)
    sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
    # save image to combine into a gif for the notes (do not need to do this)
    file_num = str(i*2 ).zfill(2)
    sfig.get_figure().savefig('kmeans' +file_num + '.png')

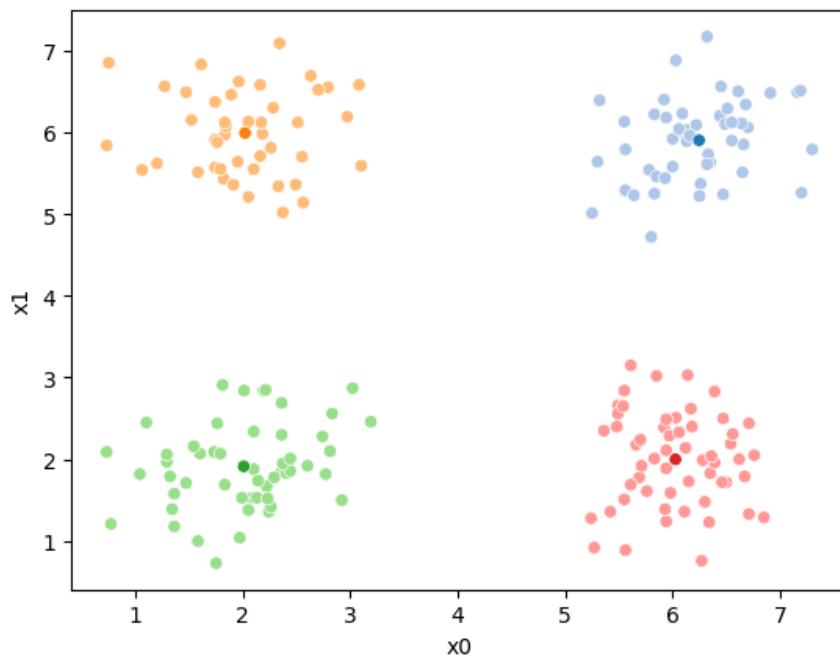
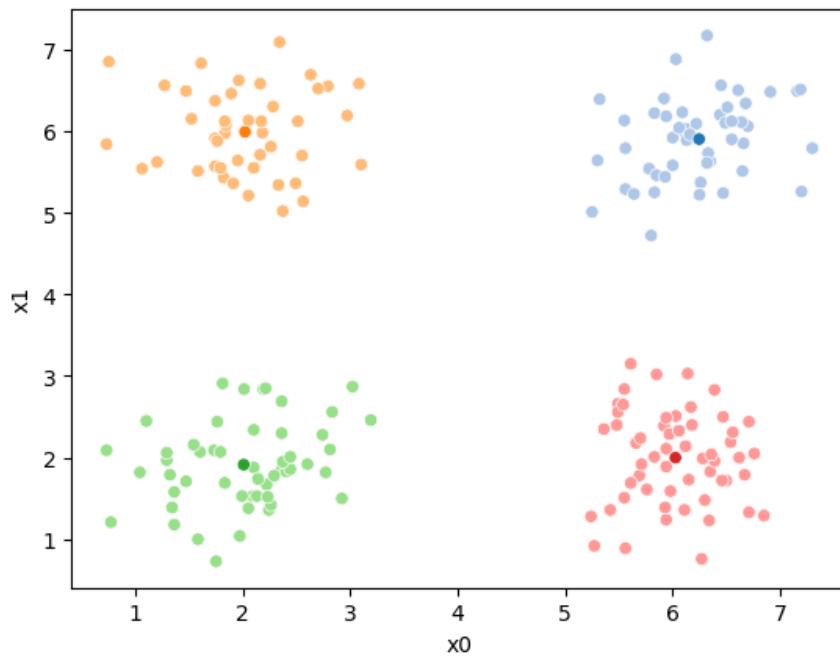
#     update the assigments and plot with the associated means
df[cur_new] = pd.concat([(df[data_cols]-mu_i)**2).sum(axis=1) for mu_i in mu],axis=1).idxmin(axis=1)

fig = plt.figure()
sfig = sns.scatterplot(data =df,x='x0',y='x1',hue=cur_new,palette=cmap_pt,legend=False)
sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
# save image to combine into a gif for the notes (do not need to do this)
# we are making 2 images per iteration
file_num = str(i*2+1).zfill(2)
sfig.get_figure().savefig('kmeans' +file_num + '.png')
number_changed_assignments = sum(df[cur_old] !=df[cur_new])
print( 'iteration ' + str(i))

```

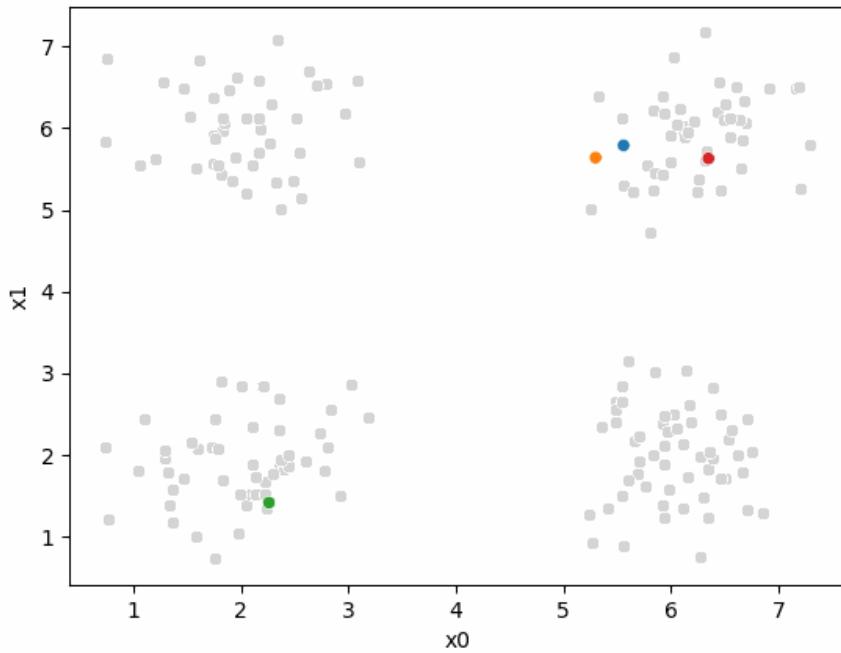
iteration 2  
iteration 3



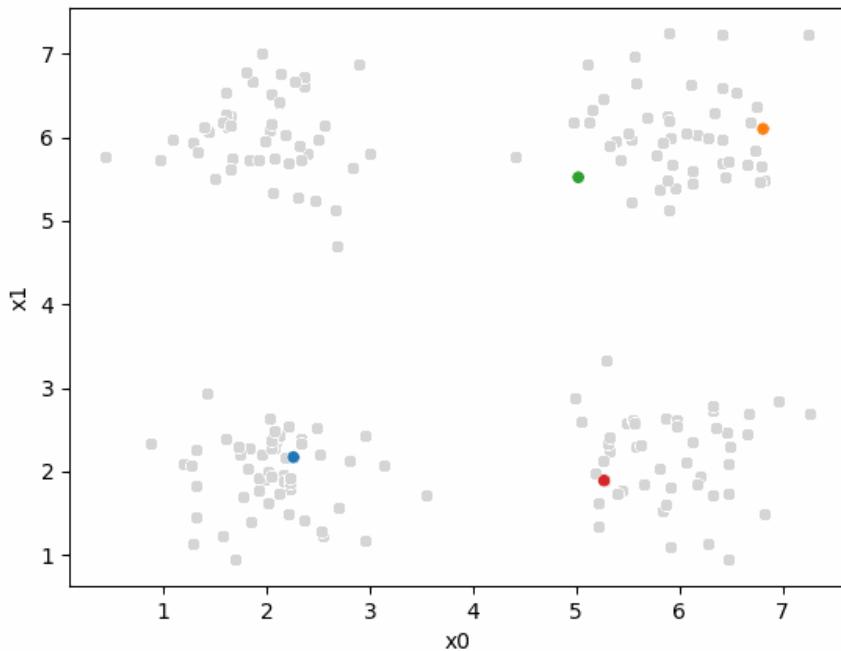


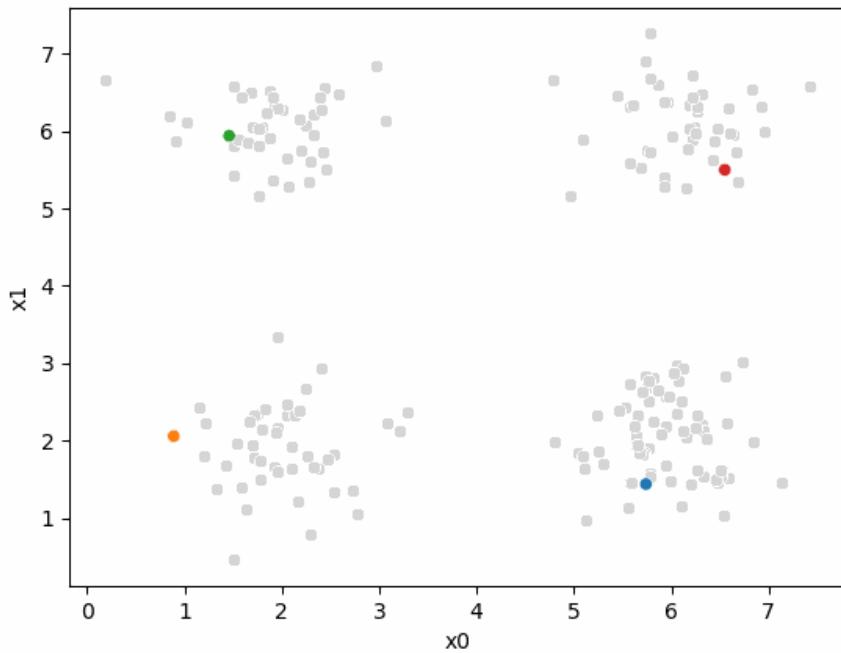
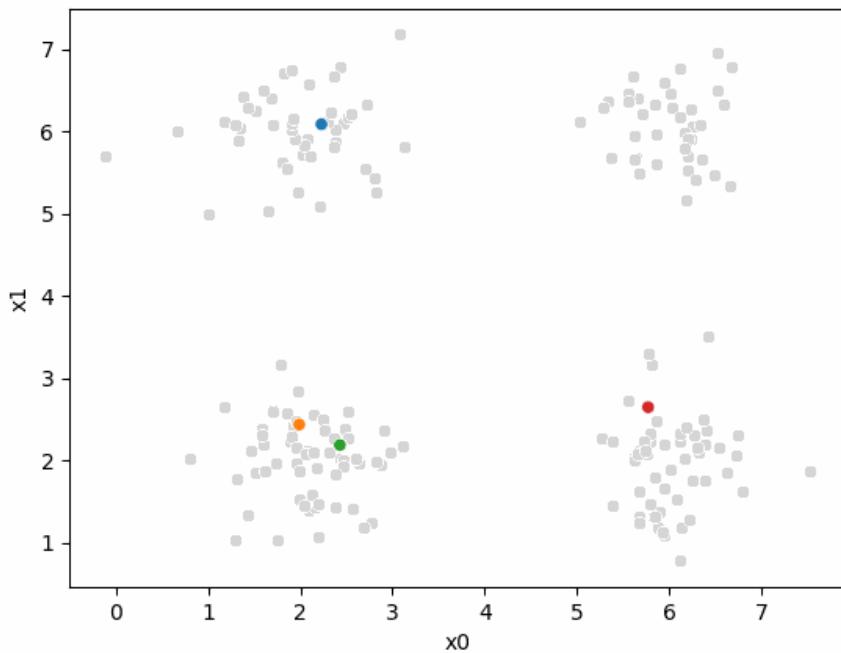
### 15.6.1. animated

▶ Show code cell content

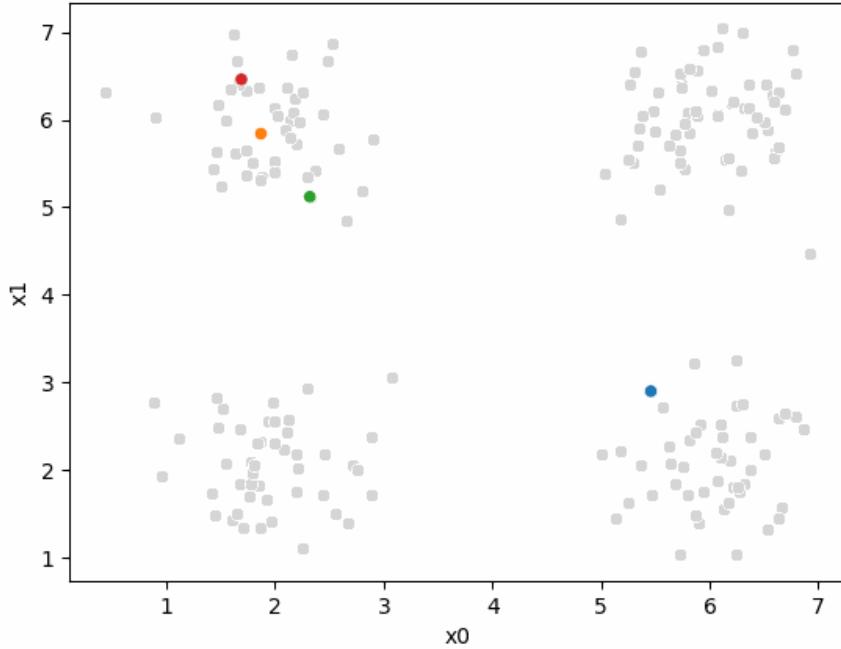


Here are some other runs that I saved. I updated this notebook so that it saved the final gif as a different name based on the random seed, ran it a few times with different seeds then kept one I thought was a good visualization above.





Including this on were it got stuck:



## 15.6.2. storyboard

We can also make them a grid:

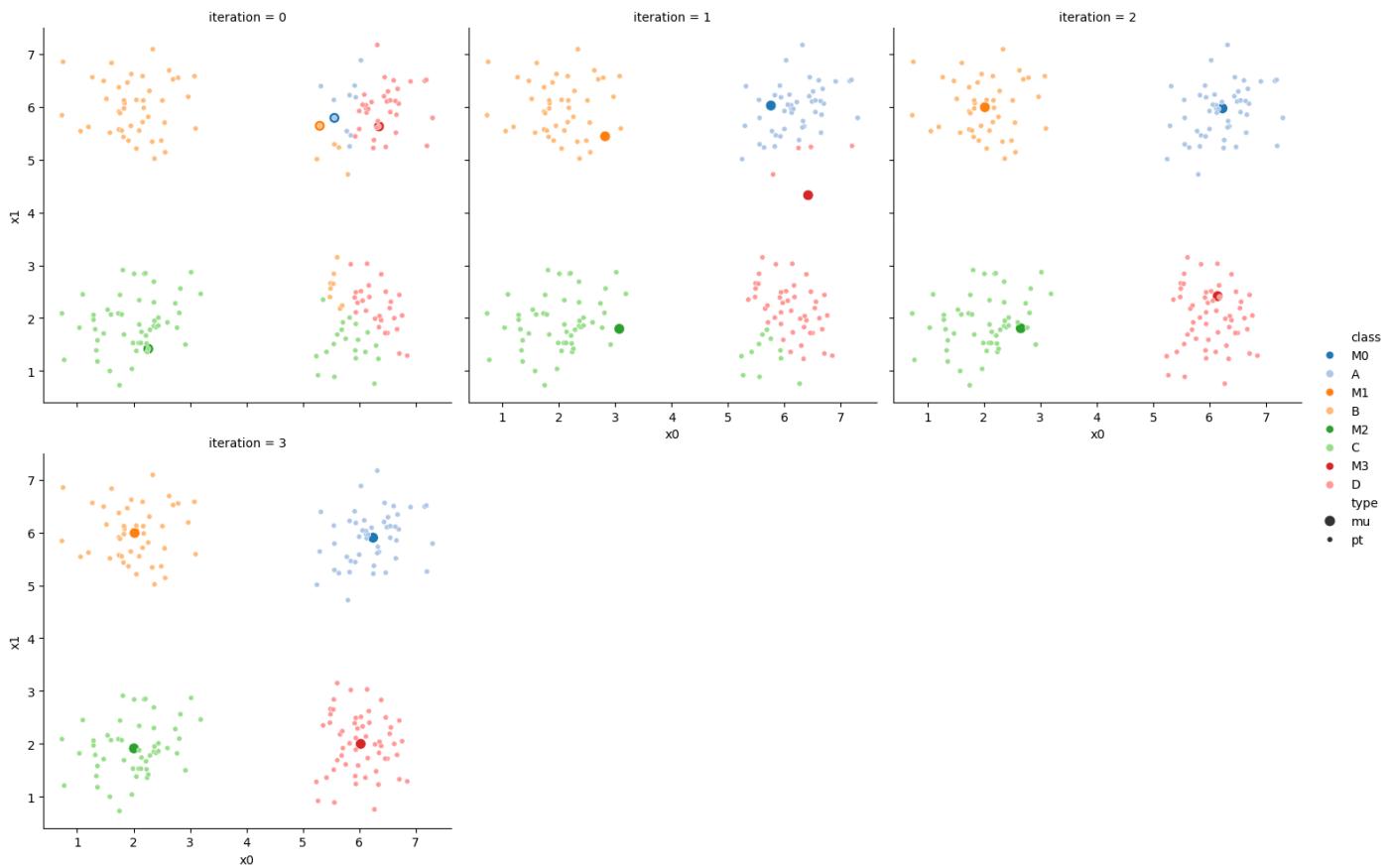
```

df_vis = df.melt(id_vars = ['x0','x1'], var_name ='iteration',value_name='class')
df_vis.replace({'class':{i:c for i,c in enumerate(string.ascii_uppercase[:C])}},inplace=True)

df_vis['type'] = 'pt'
df_mu_vis = pd.concat([pd.concat(mu_list),df_vis])
cmap = sns.color_palette('tab20',8)
n_iter = i

sfig = sns.relplot(data=df_mu_vis,x='x0',y='x1',hue='class',col='iteration',
    col_wrap=3,hue_order = ['M0','A','M1','B','M2','C','M3','D'],
    palette = cmap,size='type',col_order=[str(i) for i in range(n_iter+1)])

```

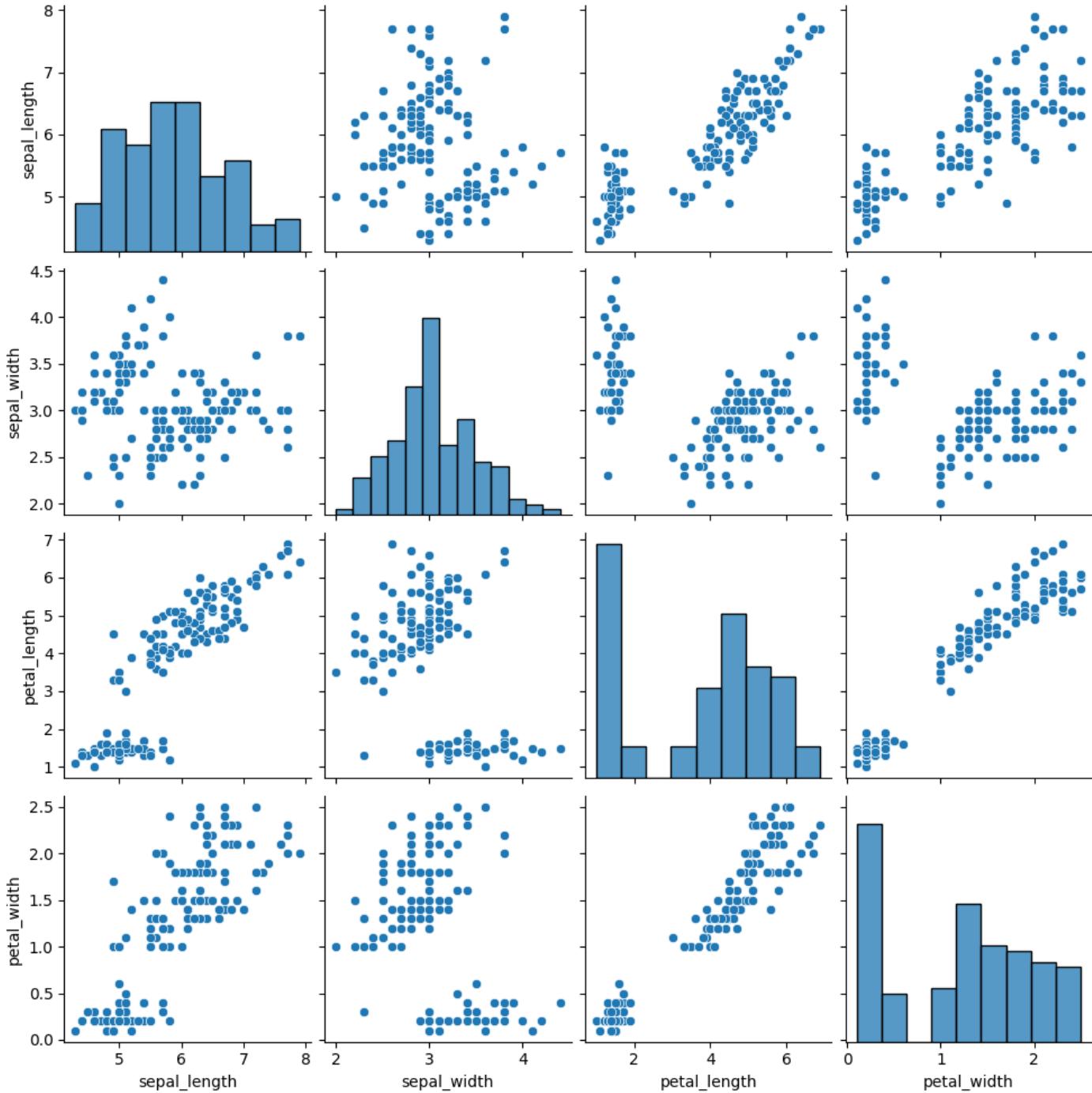


## 15.7. More metrics

We'll go back to the iris data

```
iris_df =sns.load_dataset('iris')
sns.pairplot(iris_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7f0a9693f8e0>
```



first let's cluster it again:

```
{ measurement_cols = ['sepal_length', 'petal_length', 'sepal_width', 'petal_width']  
iris_X = iris_df[measurement_cols]
```

we'll create our object

```
| km3 = KMeans(n_clusters=3)
```

[Skip to main content](#)

and save the predictions

```
y_pred = km3.fit_predict(iris_X)  
y_pred[:5]
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning:  
super().__check_params_vs_input(X, default_n_init=10)
```

```
array([0, 0, 0, 0, 0], dtype=int32)
```

### ! Important

In class, we ran the algorithm a few times and saw that the number changed

We saw the silhouette score last class, which is a true clustering method, so we can use that to see if a clustering algorithm has succeeded or in a real use case where we actually do not have the labels. However if we were checking if our implementation of a clustering algorithm worked or if our new clustering algorithm worked, we would test with the actual labels that we do have to know if the algorithm found the same groups we had. We might also want to in a real clustering context see if two different runs of a clustering algorithm were the same, or if two different algorithms on the same data found the same solution. In this case we can use other metrics, like mutual information.

Mutual information measures how much knowing one variable tells about another. It can be interpreted like a correlation for categorical variables.

```
metrics.adjusted_mutual_info_score(iris_df['species'],y_pred)
```

```
0.7551191675800484
```

This score is pretty good, but not perfect, but the classification with the labels was not either so that makes sense

A variable has 1 AMI with itself:

```
metrics.adjusted_mutual_info_score(iris_df['species'],iris_df['species'])
```

```
1.0
```

and uncorrelated things have AMI near 0

```
metrics.adjusted_mutual_info_score(np.random.choice(4,size=100),  
np.random.choice(4,size=100))
```

```
0.012048070770028944
```

## 15.8. Questions

### 15.8.1. what happens if we do not know how many clusters there are?

There are some clustering algorithms that can also learn the number of clusters, for example the [Dirichlet Process Gaussian Mixture Model](#).

Alternatively, you can try the algorithm with different numbers of clusters and evaluate to determine which one is the best fit.

### 15.8.2. what's the difference between synthetic data and data from the gaussian machine learning model.

They're both Gaussian data. This data is actually a little bit simpler than the GNB data, because all of the variances are the same.

### 15.8.3. How do clustering algorithms relate the data to the desired output. is it pureley for humans to be able to understand what the data is or is there some deeper systems going on.

Clustering can help you discover groups in the data even if they are not available in the real data. It is a way to discover something else that is going on that was not known in advance.

### 15.8.4. What is a real world example of this algorithm?

This one is a bit simpler, but slightly more complicated ones (basically changing the distance calculated for the assignment step) are used for real things in production and in research

Some example applications:

- one of my lab mates in grad school, used unsupervised learning in collaboration with medical doctors to discover subtypes of COPD (Chronic Obstructive Pulmonary Disorder).
- Etsy Data Scientists used clustering to discover “styles” of products like geometric, farmhouse, goth, etc.
- My Dissertation research was a different type of unsupervised learning but similar, to use brain imaging data and discover what images different people in a study viewed which images as similar.
- a data scientist at ESPN might think there are “types” of players in a sport, they could use data on the players to discover it
- a data scientist at a retail company like Walmart or Target might think there are “types” of customers and what to discover them from the data on what people have purchased.

### 15.8.5. is there ever a case where the elbow of the score and cluster graph is not

# the best number of clusters for a dataset

The elbo of the graph is one *definition* of best. Of course you could have a different definition of best and they may not be the same number.

## 16. Regression

### 16.1. Setting up for regression

We're going to predict **tip** from **total\_bill**. This is a regression problem because the target, *tip* is:

1. available in the data (makes it supervised) and
2. a continuous value. The problems we've seen so far were all classification, species of iris and the character in that corners data were both categorical.

Using linear regression is also a good choice because it makes sense that the tip would be approximately linearly related to the total bill, most people pick some percentage of the total bill. If we our prior knowledge was that people typically tipped with some more complicated function, this would not be a good model.

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import pandas as pd
sns.set_theme(font_scale=2, palette='colorblind')
```

We wil load this data from `seaborn`

```
tips_df = sns.load_dataset('tips')
tips_df.head()
```

	total_bill	tip	sex	smoker	day	time	size	
0	16.99	1.01	Female		No	Sun	Dinner	2
1	10.34	1.66	Male		No	Sun	Dinner	3
2	21.01	3.50	Male		No	Sun	Dinner	3
3	23.68	3.31	Male		No	Sun	Dinner	2
4	24.59	3.61	Female		No	Sun	Dinner	4

Since we will use only one variable, we have to do some extra work. When we use sklearn with a DataFrame, it picks out the values, like this:

```
tips_df['total_bill'].values
```

```
array([16.99, 10.34, 21.01, 23.68, 24.59, 25.29, 8.77, 26.88, 15.04,
       14.78, 10.27, 35.26, 15.42, 18.43, 14.83, 21.58, 10.33, 16.29,
       16.97, 20.65, 17.92, 20.29, 15.77, 39.42, 19.82, 17.81, 13.37,
       12.69, 21.7 , 19.65, 9.55, 18.35, 15.06, 20.69, 17.78, 24.06,
       16.31, 16.93, 18.69, 31.27, 16.04, 17.46, 13.94, 9.68, 30.4 ,
       18.29, 22.23, 32.4 , 28.55, 18.04, 12.54, 10.29, 34.81, 9.94,
       25.56, 19.49, 38.01, 26.41, 11.24, 48.27, 20.29, 13.81, 11.02,
       18.29, 17.59, 20.08, 16.45, 3.07, 20.23, 15.01, 12.02, 17.07,
       26.86, 25.28, 14.73, 10.51, 17.92, 27.2 , 22.76, 17.29, 19.44,
       16.66, 10.07, 32.68, 15.98, 34.83, 13.03, 18.28, 24.71, 21.16,
       28.97, 22.49, 5.75, 16.32, 22.75, 40.17, 27.28, 12.03, 21.01,
       12.46, 11.35, 15.38, 44.3 , 22.42, 20.92, 15.36, 20.49, 25.21,
       18.24, 14.31, 14. , 7.25, 38.07, 23.95, 25.71, 17.31, 29.93,
       10.65, 12.43, 24.08, 11.69, 13.42, 14.26, 15.95, 12.48, 29.8 ,
       8.52, 14.52, 11.38, 22.82, 19.08, 20.27, 11.17, 12.26, 18.26,
       8.51, 10.33, 14.15, 16. , 13.16, 17.47, 34.3 , 41.19, 27.05,
       16.43, 8.35, 18.64, 11.87, 9.78, 7.51, 14.07, 13.13, 17.26,
       24.55, 19.77, 29.85, 48.17, 25. , 13.39, 16.49, 21.5 , 12.66,
       16.21, 13.81, 17.51, 24.52, 20.76, 31.71, 10.59, 10.63, 50.81,
       15.81, 7.25, 31.85, 16.82, 32.9 , 17.89, 14.48, 9.6 , 34.63,
       34.65, 23.33, 45.35, 23.17, 40.55, 20.69, 20.9 , 30.46, 18.15,
       23.1 , 15.69, 19.81, 28.44, 15.48, 16.58, 7.56, 10.34, 43.11,
       13. , 13.51, 18.71, 12.74, 13. , 16.4 , 20.53, 16.47, 26.59,
       38.73, 24.27, 12.76, 30.06, 25.89, 48.33, 13.27, 28.17, 12.9 ,
       28.15, 11.59, 7.74, 30.14, 12.16, 13.42, 8.58, 15.98, 13.42,
       16.27, 10.09, 20.45, 13.28, 22.12, 24.01, 15.69, 11.61, 10.77,
       15.53, 10.07, 12.6 , 32.83, 35.83, 29.03, 27.18, 22.67, 17.82,
       18.78])
```

This is because originally sklearn actually only worked on `numpy` arrays and so what they do now is pull it out of the DataFrame, it gives us a numpy array.

We can change this by adding an axis. It doesn't change values, but changes the way it is represented enough that it has the properties we need.

```
tips_df['total_bill'].values[:, np.newaxis]
```

```
array([[16.99],  
       [10.34],  
       [21.01],  
       [23.68],  
       [24.59],  
       [25.29],  
       [ 8.77],  
       [26.88],  
       [15.04],  
       [14.78],  
       [10.27],  
       [35.26],  
       [15.42],  
       [18.43],  
       [14.83],  
       [21.58],  
       [10.33],  
       [16.29],  
       [16.97],  
       [20.65],  
       [17.92],  
       [20.29],  
       [15.77],  
       [39.42],  
       [19.82],  
       [17.81],  
       [13.37],  
       [12.69],  
       [21.7 ],  
       [19.65],  
       [ 9.55],  
       [18.35],  
       [15.06],  
       [20.69],  
       [17.78],  
       [24.06],  
       [16.31],  
       [16.93],  
       [18.69],  
       [31.27],  
       [16.04],  
       [17.46],  
       [13.94],  
       [ 9.68],  
       [30.4 ],  
       [18.29],  
       [22.23],  
       [32.4 ],  
       [28.55],  
       [18.04],  
       [12.54],  
       [10.29],  
       [34.81],  
       [ 9.94],  
       [25.56],  
       [19.49],  
       [38.01],  
       [26.41],  
       [11.24],  
       [48.27],  
       [20.29],  
       [13.81],  
       [11.02],  
       [18.29],  
       [17.59],  
       [20.08].
```

[Skip to main content](#)

[20.23],  
[15.01],  
[12.02],  
[17.07],  
[26.86],  
[25.28],  
[14.73],  
[10.51],  
[17.92],  
[27.2 ],  
[22.76],  
[17.29],  
[19.44],  
[16.66],  
[10.07],  
[32.68],  
[15.98],  
[34.83],  
[13.03],  
[18.28],  
[24.71],  
[21.16],  
[28.97],  
[22.49],  
[ 5.75],  
[16.32],  
[22.75],  
[40.17],  
[27.28],  
[12.03],  
[21.01],  
[12.46],  
[11.35],  
[15.38],  
[44.3 ],  
[22.42],  
[20.92],  
[15.36],  
[20.49],  
[25.21],  
[18.24],  
[14.31],  
[14. ],  
[ 7.25],  
[38.07],  
[23.95],  
[25.71],  
[17.31],  
[29.93],  
[10.65],  
[12.43],  
[24.08],  
[11.69],  
[13.42],  
[14.26],  
[15.95],  
[12.48],  
[29.8 ],  
[ 8.52],  
[14.52],  
[11.38],  
[22.82],  
[19.08],  
[20.27],  
[11.17],  
[12.26],  
[18.26],  
[ 8.51]

[16. ],  
[13.16],  
[17.47],  
[34.3 ],  
[41.19],  
[27.05],  
[16.43],  
[ 8.35],  
[18.64],  
[11.87],  
[ 9.78],  
[ 7.51],  
[14.07],  
[13.13],  
[17.26],  
[24.55],  
[19.77],  
[29.85],  
[48.17],  
[25. ],  
[13.39],  
[16.49],  
[21.5 ],  
[12.66],  
[16.21],  
[13.81],  
[17.51],  
[24.52],  
[20.76],  
[31.71],  
[10.59],  
[10.63],  
[50.81],  
[15.81],  
[ 7.25],  
[31.85],  
[16.82],  
[32.9 ],  
[17.89],  
[14.48],  
[ 9.6 ],  
[34.63],  
[34.65],  
[23.33],  
[45.35],  
[23.17],  
[40.55],  
[20.69],  
[20.9 ],  
[30.46],  
[18.15],  
[23.1 ],  
[15.69],  
[19.81],  
[28.44],  
[15.48],  
[16.58],  
[ 7.56],  
[10.34],  
[43.11],  
[13. ],  
[13.51],  
[18.71],  
[12.74],  
[13. ],  
[16.4 ],  
[20.53],  
[16.47]

```
[24.27],  
[12.76],  
[30.06],  
[25.89],  
[48.33],  
[13.27],  
[28.17],  
[12.9 ],  
[28.15],  
[11.59],  
[ 7.74],  
[30.14],  
[12.16],  
[13.42],  
[ 8.58],  
[15.98],  
[13.42],  
[16.27],  
[10.09],  
[20.45],  
[13.28],  
[22.12],  
[24.01],  
[15.69],  
[11.61],  
[10.77],  
[15.53],  
[10.07],  
[12.6 ],  
[32.83],  
[35.83],  
[29.03],  
[27.18],  
[22.67],  
[17.82],  
[18.78]))
```

All sklearn estimator objects are designed to work on multiple features, which mean they expect that the features have a 2D shape, but when we pick out a single column's (pandas Series) values, we get a 1D shape but after we get 2d, though the second dimension the size is 1

```
tips_df['total_bill'].values.shape, tips_df['total_bill'].values[:,np.newaxis].shape
```

```
((244,), (244, 1))
```

The target is expected to be a single feature, so we do not need to do anything special.

```
tips_X = tips_df['total_bill'].values[:,np.newaxis]  
tips_y = tips_df['tip']
```

Next, we split the data

```
tips_X_train, tips_X_test, tips_y_train, tips_y_test = train_test_split(tips_X,  
                                                               tips_y)
```

## 16.2. Fitting a regression model

We instantiate the object as we do with all other sklearn estimator objects.

```
regr = linear_model.LinearRegression()
```

we can inspect it to get a before

```
regr.__dict__
```

```
{'fit_intercept': True, 'copy_X': True, 'n_jobs': None, 'positive': False}
```

and `fit` like the others too.

```
regr.fit(tips_X_train,tips_y_train)
```

▼ LinearRegression

LinearRegression()

and look again:

```
regr.__dict__
```

```
{'fit_intercept': True,
 'copy_X': True,
 'n_jobs': None,
 'positive': False,
 'n_features_in_': 1,
 'coef_': array([0.10526973]),
 'rank_': 1,
 'singular_': array([121.53480582]),
 'intercept_': 0.9302105911804008}
```

we also can predict as before

```
tips_y_pred = regr.predict(tips_X_test)
```

## 16.3. Regression Predictions

Linear regression is making the assumption that the target is a linear function of the features so that we can use the equation for a line(for scalar variables):

becomes equivalent to the following code assuming they were all vectors/matrices:

```
i = 1 # could be any number from 0 ... len(target)
target[i] = regr.coef_*features[i]+ regr.intercept_
```

You can match these up one for one.

- `target` is  $y$  (this is why we usually put the `_y_` in the variable name)
- `regr.coef_` is the slope,  $m$
- `features` are the  $x$  (like for  $y$ , we label that way)
- and `regr.intercept_` is the  $b$  or the  $y$  intercept.

### ! Important

I have tried to expand the detail here substantively. Use the link at the top of the page to make an issue if you have specific questions

We will look at each of them and store them to variables here

```
slope = regr.coef_
intercept = regr.intercept_
```

▶ Show code cell content

### ! Important

This is what our model *predicts* the tip will be based on the past data. It is important to note that this is not what the tip *should* be by any sort of virtues. For example, a typical normative rule for tipping is to tip 15% or 20%. the model we learned, from this data, however is `array([10.52697329])% + $0.9302105911804008`.

now we can pull out our first  $x_i$  and calculate a predicted value of  $y$  manually

```
slope * tips_X_test[0] + intercept
```

```
array([2.72716493])
```

and see that this matches exactly the prediction from the `predict` method.

```
tips_y_pred[0]
```

```
2.7271649312146087
```

```
tips_X_test[0]
```

Skip to main content

```
numpy.ndarray
```

### ! Important

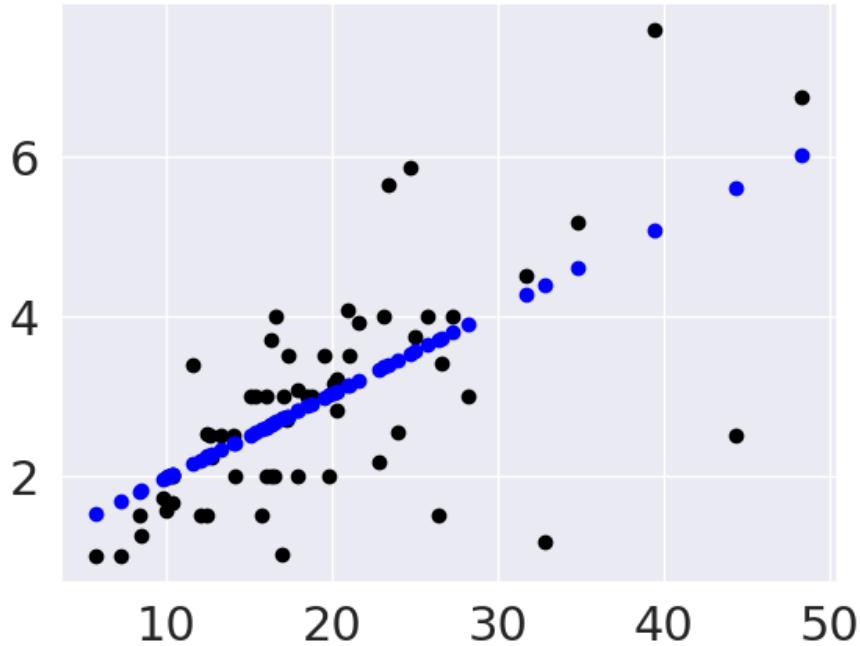
Note that my random seed is not fixed, so the values there are going to change each time the website is updated, but that these values will *always* be true

## 16.4. Visualizing Regression

Since we only have one feature, we can visualize what was learned here. We will use plain `matplotlib` plots because we are plotting from numpy arrays not data frames.

```
plt.scatter(tips_X_test, tips_y_test, color = 'black')
plt.scatter(tips_X_test, tips_y_pred, color='blue')
```

```
<matplotlib.collections.PathCollection at 0x7f905d078220>
```



This plots the predictions in blue vs the real data in black. The above uses them both as scatter plots to make it more clear, below, like in class, I'll use a normally inconvenient aspect of a line plot to show all the predictions the model could make in this range.

## 16.5. Evaluating Regression - Mean Squared Error

From the plot, we can see that there is some error for each point, so accuracy that we've been using, won't work. One idea is to look at how much error there is in each prediction, we can look at that visually first.

[Skip to main content](#)

In this code block, I use `zip` a builtin function in Python to iterate over all of the test samples and predictions (they're all the same length) and plot each red line in a list comprehension. This could have been a for loop, but the comprehension is slightly more compact visually. The zip allows us to have a Pythonic, easy to read loop that iterates over multiple variables.

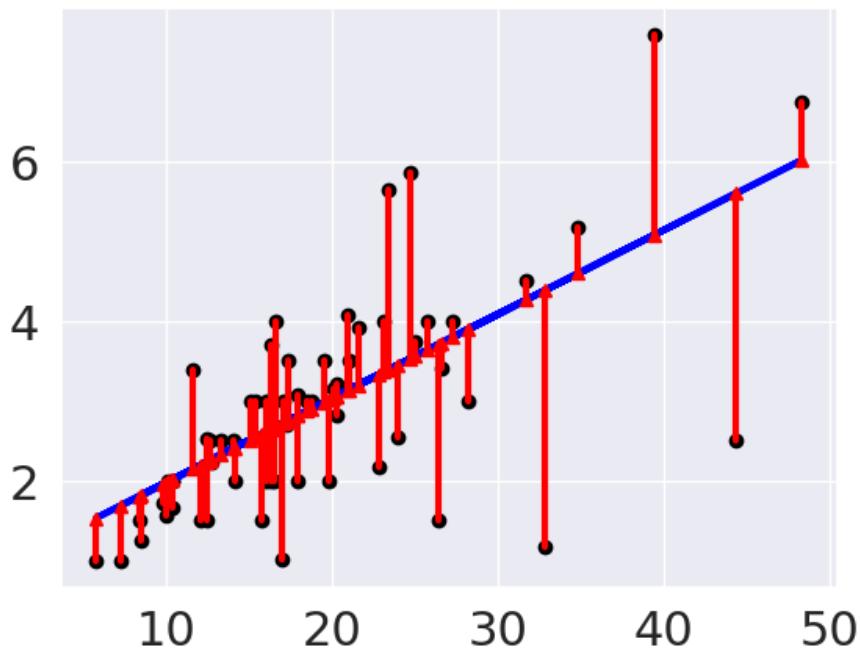
To make a vertical line, I make a line plot with just two values. My plotted data is: `[x, x], [yp, yt]` so the first point is `(x, yp)` and the second is `x, yt`.

The `;` at the end of each line suppressed the text output.

```
# plot line prediction
plt.plot(tips_X_test, tips_y_pred, color='blue', linewidth=3);

# draw vertical lines from each data point to its predict value
[plt.plot([x,x],[yp,yt], color='red', linewidth=3, markevery=[0], marker ='^')
 for x, yp, yt in zip(tips_X_test, tips_y_pred,tips_y_test)];

# plot these last so they are visually on top
plt.scatter(tips_X_test, tips_y_test, color='black');
```



We can see that these are pretty small errors mostly, but now we can get some metrics.

To see more about how the code above works:

▶ Show code cell content

We can use the average length of these red lines to capture the error. To get the length, we can take the difference between the prediction and the data for each point. Some would be positive and others negative, so we will square each one then take the average. This will be the mean squared error

```
mean_squared_error(tips_y_test,tips_y_pred)
```

Which is equivalent to:

```
np.mean((tips_y_test - tips_y_pred)**2)
```

```
1.0441826161075698
```

To interpret this, we can take the square root to get it back into dollars. This becomes equivalent to taking the mean absolute value of the error.

```
mean_abs_error = np.sqrt(mean_squared_error(tips_y_test, tips_y_pred))  
mean_abs_error
```

```
1.021852541273725
```

which is equivalent to:

```
np.mean(np.abs(tips_y_test - tips_y_pred))
```

```
0.7162692166427864
```

Still, to know if this is a big or small error, we have to compare it to the values we were predicting

```
tips_y_test.describe()
```

```
count    61.000000  
mean     2.878689  
std      1.361472  
min      1.000000  
25%     2.000000  
50%     2.710000  
75%     3.500000  
max      7.580000  
Name: tip, dtype: float64
```

```
avg_tip = tips_y_test.mean()  
tip_var = tips_y_test.var()  
avg_tip, tip_var
```

```
(2.8786885245901632, 1.853604918032787)
```

▶ Show code cell content

So we can see that the average error is *mae* which is relatively large compared to the average tip size of but is smaller than most of them.

Skip to main content

## 16.6. Evaluating Regression - R2

We can also use the  $R^2$  score, the coefficient of determination.

If we have the following:

- $n = \text{len}(y_{\text{test}})$
- $y = y_{\text{test}}$
- $y_i = y_{\text{test}}[i]$
- $\hat{y} = y_{\text{pred}}$
- $\bar{y} = \frac{1}{n} \sum_{i=0}^n y_i = \text{sum}(y_{\text{test}}) / \text{len}(y_{\text{test}})$

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^n (y_i - \hat{y}_i)^2}{\sum_{i=0}^n (y_i - \bar{y}_i)^2}$$

```
r2_score(tips_y_test, tips_y_pred)
```

```
0.42728590683929923
```

This is a bit harder to interpret, but we can use some additional plots to visualize. This code simulates data by randomly picking 20 points, spreading them out and makes the “predicted” y values by picking a slope of 3. Then I simulated various levels of noise, by sampling noise and multiplying the same noise vector by different scales and adding all of those to a data frame with the column name the r score for if that column of target values was the truth.

Then I added some columns of y values that were with different slopes and different functions of x. These all have the small amount of noise.

Tip  
Fac  
cus  
plot  
oth  
com  
part

```

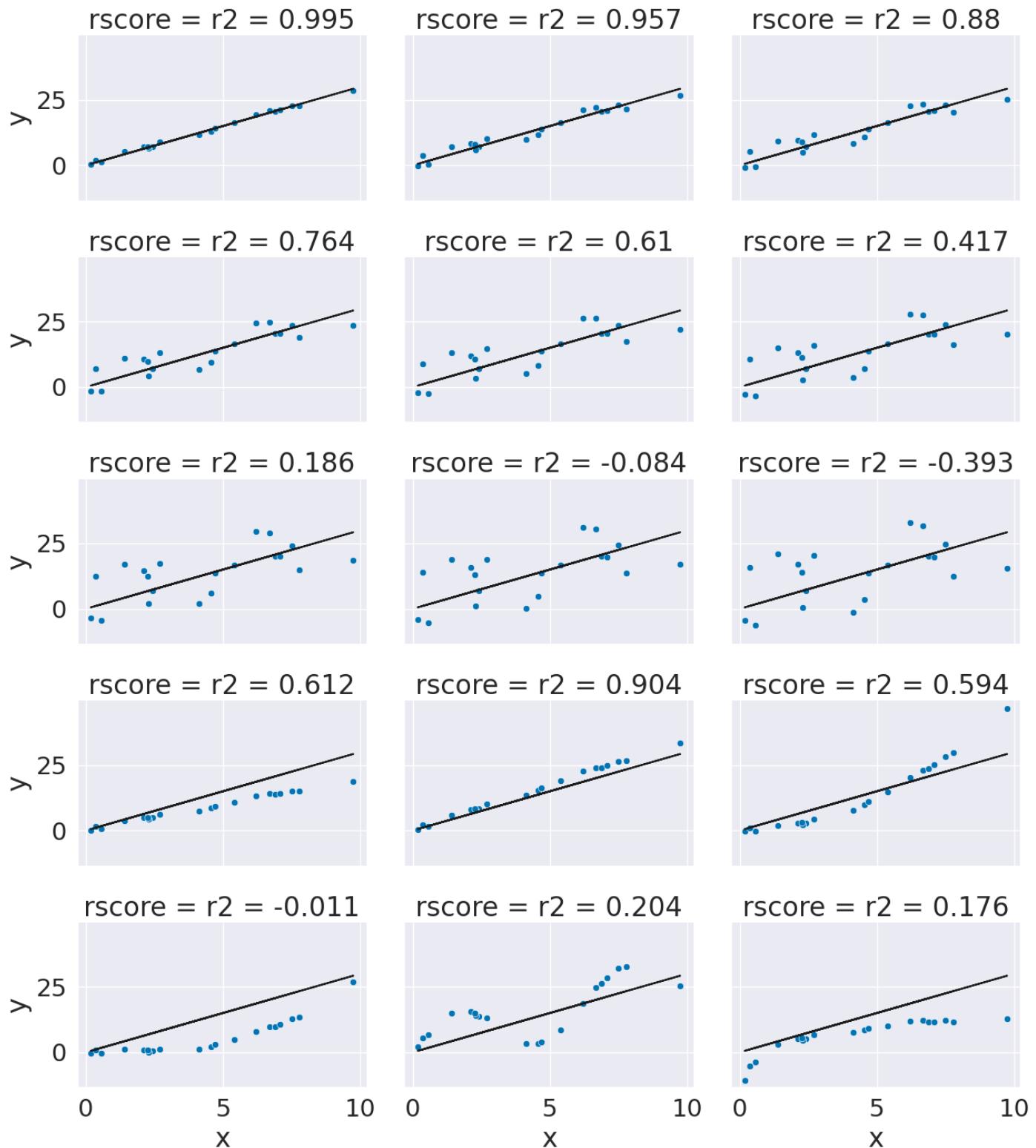
# decide a number of points
N = 20
# pick some random points
x = 10*np.random.random(N)
# make a line with a fixed slope
y_pred = 3*x
# put it in a data frame
ex_df = pd.DataFrame(data = x,columns = ['x'])
ex_df['y_pred'] = y_pred
# set some amounts of noise
n_levels = range(1,18,2)
# sample random values between 0 and 1, then make the spread between -1 and +1
noise = (np.random.random(N)-.5)*2
# add the noise and put it in the data frame with col title the r2
for n in n_levels:
    y_true = y_pred + n* noise
    ex_df['r2 = '+ str(np.round(r2_score(y_pred,y_true),3))] = y_true

# set some other functions
f_x_list = [2*x,3.5*x,.5*x**2, .03*x**3, 10*np.sin(x)+x*3,3*np.log(x**2)]
# add them to the noise and store with r2 col title
for fx in f_x_list:
    y_true = fx + noise
    ex_df['r2 = '+ str(np.round(r2_score(y_pred,y_true),3))] = y_true

# make the data tidy for plotting
xy_df = ex_df.melt(id_vars=['x','y_pred'],var_name='rscore',value_name='y')
# make a custom grid by using facet grid directly
g = sns.FacetGrid(data = xy_df,col='rscore',col_wrap=3,aspect=1.5,height=3)
# plot the lin
g.map(plt.plot, 'x','y_pred',color='k')
# plot the dta
g.map(sns.scatterplot, "x", "y", )

```

```
<seaborn.axisgrid.FacetGrid at 0x7f905a59cb80>
```



## 16.7. Regression auto scoring

as all sklearn objects, it has a built in score method

[Skip to main content](#)

```
regr.score(tips_X_test, tips_y_test)
```

```
0.42728590683929923
```

this matches the  $R^2$  score:

```
r2_score(tips_y_test, tips_y_pred)
```

```
0.42728590683929923
```

and is different from mse

```
mean_squared_error(tips_y_test, tips_y_pred)
```

```
1.0441826161075698
```

```
tips_df.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

## 16.8. Multivariate Regression

Recall the equation for a line:

$$\hat{y} = mx + b$$

When we have multiple variables instead of a scalar  $x$  we can have a vector  $\mathbf{x}$  and instead of a single slope, we have a vector of coefficients  $\beta$

$$\hat{y} = \beta^T \mathbf{x} + \beta_0$$

where  $\beta$  is the `regr_db.coef_` and  $\beta_0$  is `regr_db.intercept_` and that's a vector multiplication and  $\hat{y}$  is `y_pred` and  $y$  is `y_test`.

In scalar form, a vector multiplication can be written like

[Skip to main content](#)

$$\hat{y} = \sum_{k=0}^d (x_k * \beta_k) + \beta_0$$

where there are  $d$  features, that is  $d = \text{len}(X_{\text{test}}[k])$  and  $k$  indexed into it.

We are going to use 2 variables so it would be like:

$$\hat{y} = \beta_1 * x_1 + \beta_2 * x_2 + \beta_0$$

```
tips_X_size = tips_df[['total_bill', 'size']]
```

We do not need the newaxis once we have more than one feature

```
tips_X_size_train, tips_X_size_test, tips_y_size_train, tips_y_size_test = train_test_split(tips_X_size,
```

```
regr2 = linear_model.LinearRegression()
```

```
regr2.fit(tips_X_size_train, tips_y_size_train).score(tips_X_size_test, tips_y_size_test)
```

```
0.27350223555370845
```

this does a little bit worse:

```
tips_y_size_pred = regr2.predict(tips_X_size_test)
```

we can also check in terms of the absolute values

```
np.sqrt(mean_squared_error(tips_y_size_test, tips_y_size_pred))
```

```
1.0888035073416547
```

We can view the coefficients

```
regr2.coef_
```

```
array([0.10459153, 0.16443537])
```

It still computes the same way:

```
tips_X_size_test.values[0][0]*regr.coef_[0] + tips_X_size_test.values[0][1]*regr.coef_[1] + regr.intercept_
```

```
-----  
IndexError Traceback (most recent call last)  
Cell In[41], line 1  
----> 1 tips_X_size_test.values[0][0]*regr.coef_[0] + tips_X_size_test.values[0][1]*regr.coef_[1] +regr.  
  
IndexError: index 1 is out of bounds for axis 0 with size 1
```

## 16.9. Questions after class

### 16.9.1. What key points in data we should look for that can help us to chose the right model to work with?

Honestly after choosing the right task, (eg classification, regression, or clustering) a good path is to start with a simple model and then evaluate how it does or does not fit, then you can go more complex within a task.

Also, you want to check the assumptions of the model, like from the documentation or the math underlying it and then check if your data seems to fit those assumptions. That said, sometimes they are hard to check. For example, it is hard to view data and see if it is linear in a lot of dimensions, but you can check residuals after with respect to each feature.

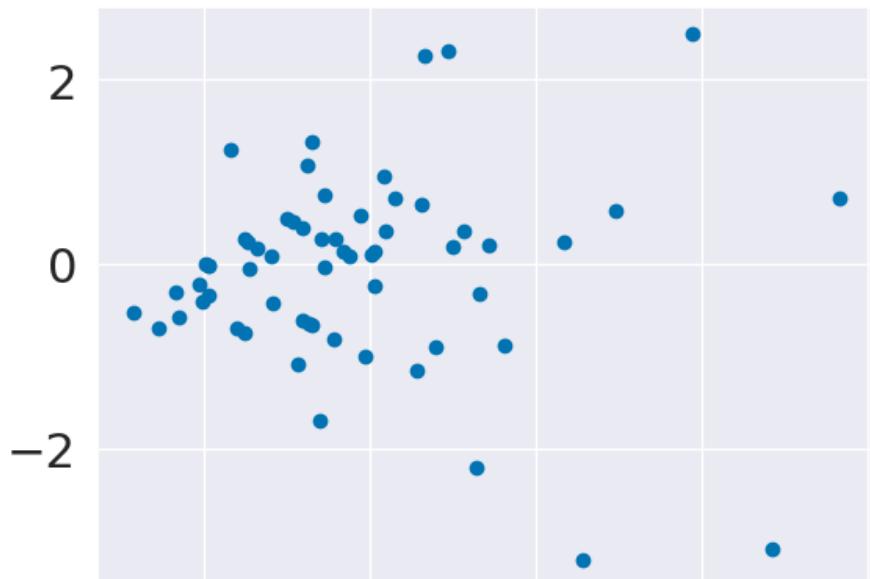
### 16.9.2. Im confused on how residuals are being factored into these models

The residuals are the point by point errors in the model. The errors

We can plot the residuals, or errors directly versus the input.

```
plt.scatter(tips_X_test,tips_y_test-tips_y_pred,)
```

```
<matplotlib.collections.PathCollection at 0x7f905828dee0>
```



If our model was predicting uniformly well, these would be evenly distributed left to right.

### 16.9.3. How do you know if a regression model is good or works for a dataset?

Let's break this down. Regression is supervised learning, so there needs to be variables that you can use as features and one to use as the target. To test if supervised learning makes sense, try filling column names in place of `features` and `target` in the following sentence, that describes a prediction task: we want to predict `target` from `features`.

Regression, specifically means that the target variable needs to be continuous, not discrete. For example, in today's class, we predicted the tip in dollars, which is continuous in this sense. This is not strictly continuous in the mathematical sense, but anything that is most useful to consider as continuous.

*logistic regression is actually a classification model despite the name*

### 16.9.4. So is regression is just used with numeric values in terms of predicting?

Yes linear regression requires numeric values for the features as well as the target.

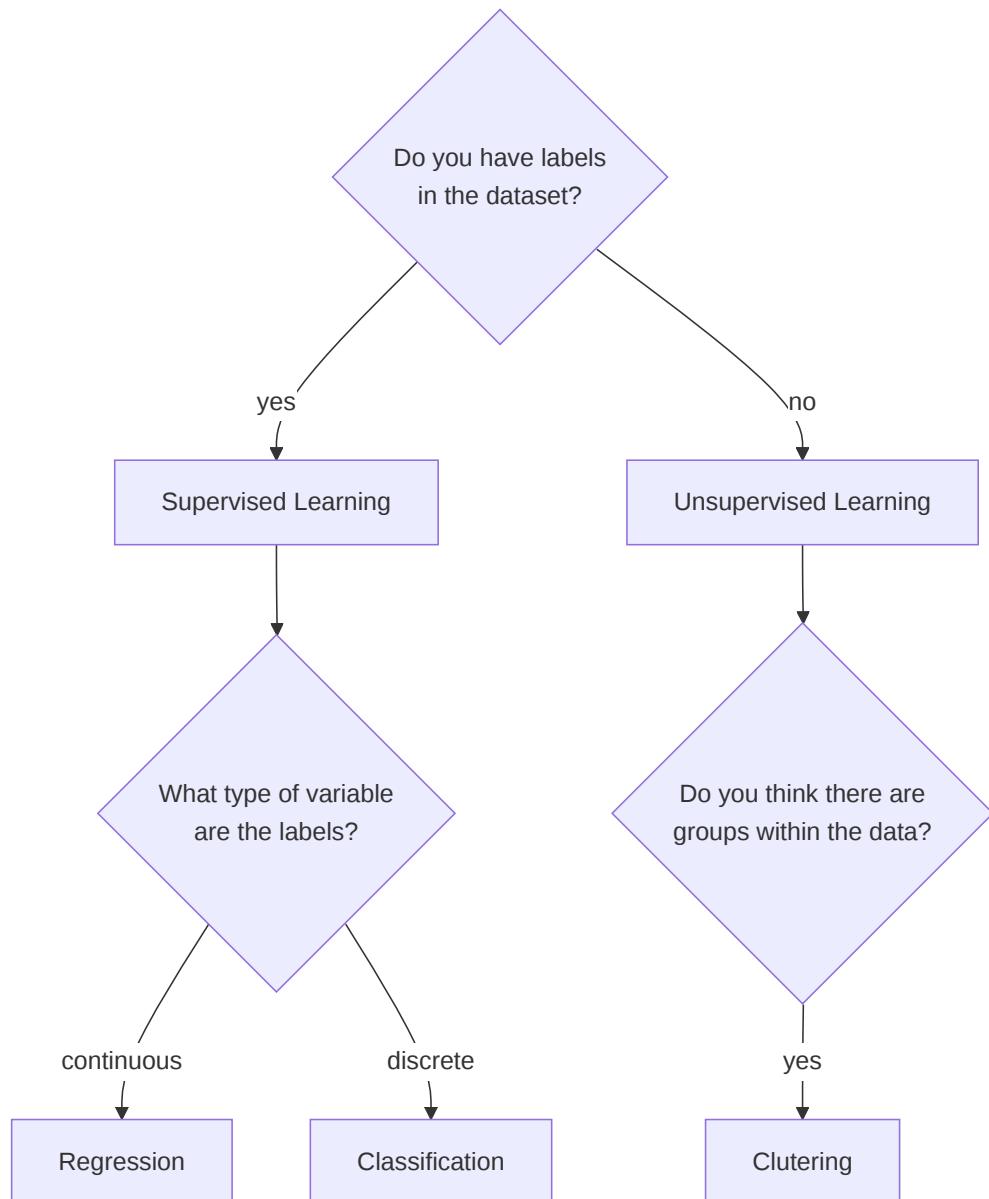
Other types of regression can use other types of features.

All regression is for a continuous target.

### 16.9.5. What are the strengths of a regression model as opposed to the previous models we've seen?

Using regression, classification, or clustering is based on what type of problem you are solving.

Our simple flowchart for what task you are doing is:



the sklearn docs include a more detailed guide

#### 16.9.6. Are linear models exclusively used for regression problems?

No, there are also linear classification models.

#### 16.9.7. What does a negative R2 score mean?

It's a bad fit.

#### 16.9.8. What is the difference between a good and bad rscore?

1 is perfect, low is bad. What is “good enough” is context dependent.

## 16.9.9. why were your values coming out negative?

Mine was not a very good fit. We'll see why on Thursday. Yours may have been better

# 17. More regression

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

import pandas as pd
sns.set_theme(font_scale=2, palette='colorblind')
```

## 17.1. Multivariate Regression

Recall the equation for a line:

$$\hat{y} = mx + b$$

When we have multiple variables instead of a scalar  $x$  we can have a vector  $\mathbf{x}$  and instead of a single slope, we have a vector of coefficients  $\beta$

$$\hat{y} = \beta^T \mathbf{x} + \beta_0$$

where  $\beta$  is the `regr_db.coef_` and  $\beta_0$  is `regr_db.intercept_` and that's a vector multiplication and  $\hat{y}$  is `y_pred` and  $y$  is `y_test`.

In scalar form, a vector multiplication can be written like

$$\hat{y} = \sum_{k=0}^d (x_k * \beta_k) + \beta_0$$

where there are  $d$  features, that is  $d = \text{len}(X\_test[k])$  and  $k$  indexed into it.

We can also load data from Scikit learn.

This dataset includes 10 features measured on a given date and a measure of diabetes disease progression measured one year later. The predictor we can train with this data might be something a doctor uses to calculate a patient's risk.

```
# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
X_train,X_test, y_train,y_test = train_test_split(diabetes_X, diabetes_y ,
                                                test_size=20,random_state=0)
```

by default it would have returned a very different object

```
db_bunch = datasets.load_diabetes()
```

```
type(db_bunch)
```

```
sklearn.utils._bunch.Bunch
```

it has a very useful attribute, the data description:

```
print(db_bunch.DESCR)
```

```
.. _diabetes_dataset:
```

```
Diabetes dataset
```

```
-----  
Ten baseline variables, age, sex, body mass index, average blood  
pressure, and six blood serum measurements were obtained for each of n =  
442 diabetes patients, as well as the response of interest, a  
quantitative measure of disease progression one year after baseline.
```

```
**Data Set Characteristics:**
```

```
:Number of Instances: 442
```

```
:Number of Attributes: First 10 columns are numeric predictive values
```

```
:Target: Column 11 is a quantitative measure of disease progression one year after baseline
```

```
:Attribute Information:
```

- age age in years
- sex
- bmi body mass index
- bp average blood pressure
- s1 tc, total serum cholesterol
- s2 ldl, low-density lipoproteins
- s3 hdl, high-density lipoproteins
- s4 tch, total cholesterol / HDL
- s5 ltg, possibly log of serum triglycerides level
- s6 glu, blood sugar level

```
Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation tim
```

```
Source URL:
```

```
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html
```

```
For more information see:
```

```
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics  
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle\_2002.pdf)
```

This model predicts what lab measure a patient will have one year in the future based on lab measures in a given day. Since we see that this is not a very high r2, we can say that this is not a perfect predictor, but a Doctor, who better understands the score would have to help interpret the core.

```
regr_db = linear_model.LinearRegression()
regr_db.fit(X_train, y_train)
```

#### ▼ LinearRegression

```
LinearRegression()
```

We can look at the estimator again and see what it learned. It describes the model like a line:

$$\hat{y} = mx + b$$

except in this case it's multivariate, so we can write it like:

$$\hat{y} = \beta^T x + \beta_0$$

where  $\beta$  is the `regr_db.coef_` and  $\beta_0$  is `regr_db.intercept_` and that's a vector multiplication and  $\hat{y}$  is `y_pred` and  $y$  is `y_test`.

In scalar form it can be written like

$$\hat{y} = \sum_{k=0}^d (x_k * \beta_k) + \beta_0$$

where there are  $d$  features, that is  $d = \text{len}(X\_test[k])$  and  $k$  indexed into it. For example in the below  $k = 0$

```
y_pred = regr_db.predict(X_test)
r2_score(y_test,y_pred)
```

```
0.5195333332288746
```

```
np.sqrt(mean_squared_error(y_test,y_pred))
```

```
53.387672540450204
```

```
np.mean(y_test)
```

```
159.4
```

```
... and so on
```

[Skip to main content](#)

```
77.02103608755208
```

```
X_train.shape
```

```
(422, 10)
```

## 17.2. LASSO

lasso allows us to pick a subset of the features at the same time we learn the weights

```
lasso = linear_model.Lasso()  
lasso.fit(X_train, y_train)  
lasso.score(X_test, y_test)
```

```
0.42249260665177746
```

We see it learns a model with most of the coefficients set to 0.

```
lasso.coef_
```

```
array([ 0.          , -0.          ,  358.27705585,  9.71139556,  
       0.          ,  0.          , -0.          ,  0.          ,  
     309.50698469,  0.          ])
```

```
lasso = linear_model.Lasso(.25)  
lasso.fit(X_train, y_train)  
lasso.score(X_test, y_test)
```

```
0.5501992246861973
```

```
lasso.coef_
```

```
array([-0.          , -52.1429064 ,  503.60281992,  219.24141192,  
      -0.          , -0.          , -147.83111493,   0.          ,  
    441.7507385 ,  0.          ])
```

## 17.3. Polynomial Regression

Polynomial regression is still a linear problem. Linear regression solves for the  $\beta_i$  for a  $d$  dimensional problem.

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d = \sum_i^d \beta_i x_i$$

Quadratic regression solves for

$$y = \beta_0 + \sum_i^d \beta_i x_i + \sum_j^d \sum_i^d \beta_{d+i} x_i x_j + \sum_i^d x_i^2$$

This is still a linear problem, we can create a new  $X$  matrix that has the polynomial values of each feature and solve for more  $\beta$  values.

We use a transformer object, which works similarly to the estimators, but does not use targets. First, we instantiate.

```
pol = PolynomialFeatures()
```

```
x_train2 = pol.fit_transform(X_train)
X_test2 = pol.fit_transform(X_test)
```

This changes the shape a lot, now we have a lot more features

```
X_train2.shape
```

```
(422, 66)
```

We can break down this total into different types, the original ones ( $x_0, x_1, \dots, x_9$ ), those squared, ( $x_0^2, x_1^2, \dots, x_9^2$ ), every pair ( $x_0x_1, x_0x_1, \dots, x_7x_8, x_8x_9$ ) and a constant (so we do not need the intercept separately)

```
original_feats = 10
sq_feats = 10
pair_prod_feats = sum(range(10))
const_feats = 1
original_feats + sq_feats + pair_prod_feats + const_feats
```

```
66
```

Now we can try lasso again (and different `alpha` values, but we changed it in place and I am leaving a good one)

```
lasso2 = linear_model.Lasso(.27)
lasso2.fit(X_train2, y_train)
lasso2.score(X_test2, y_test)
```

```
0.5501767795382546
```

and see this can improve some by using a subset of the new features and the old features.

[Skip to main content](#)

```
lasso2.coef_
```

```
array([ 0.          ,  0.          , -36.0627981 ,  502.37020182,
       210.0069951 , -0.          , -0.          , -136.1195633 ,
       0.          , 439.39268658,  0.          ,  0.          ,
       0.          ,  0.          ,  0.          ,  0.          ,
      -0.          , -0.          ,  0.          ,  0.          ,
       0.          , -0.          ,  0.          ,  0.          ,
       0.          , -0.          ,  0.          , -0.          ,
       0.          ,  0.          ,  0.          ,  0.          ,
       0.          , -0.          , -0.          ,  0.          ,
       0.          ,  0.          ,  0.          ,  0.          ,
       0.          ,  0.          ,  0.          ,  0.          ,
       0.          , -0.          , -0.          ,  0.          ,
      -0.          ,  0.          ,  0.          , -0.          ,
       0.          , -0.          ,  0.          ,  0.          ,
      -0.          , -0.          ,  0.          , -0.          ,
       0.          ,  0.          ,  0.          ,  0.          ,
       0.          ,  0.          ,  0.          ,  0.        ])
```

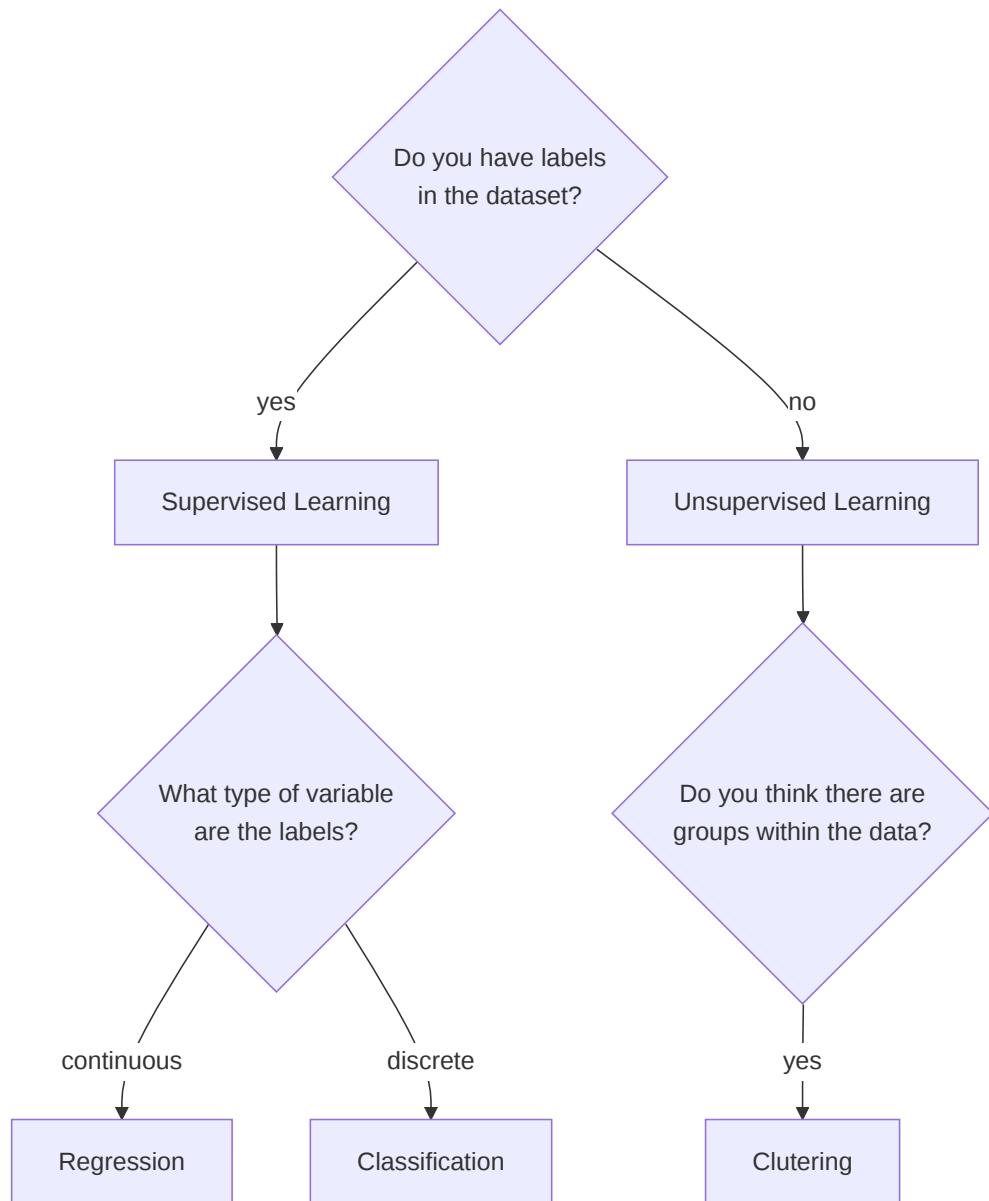
## 18. Task Review and Cross Validation

### 18.1. ML Tasks

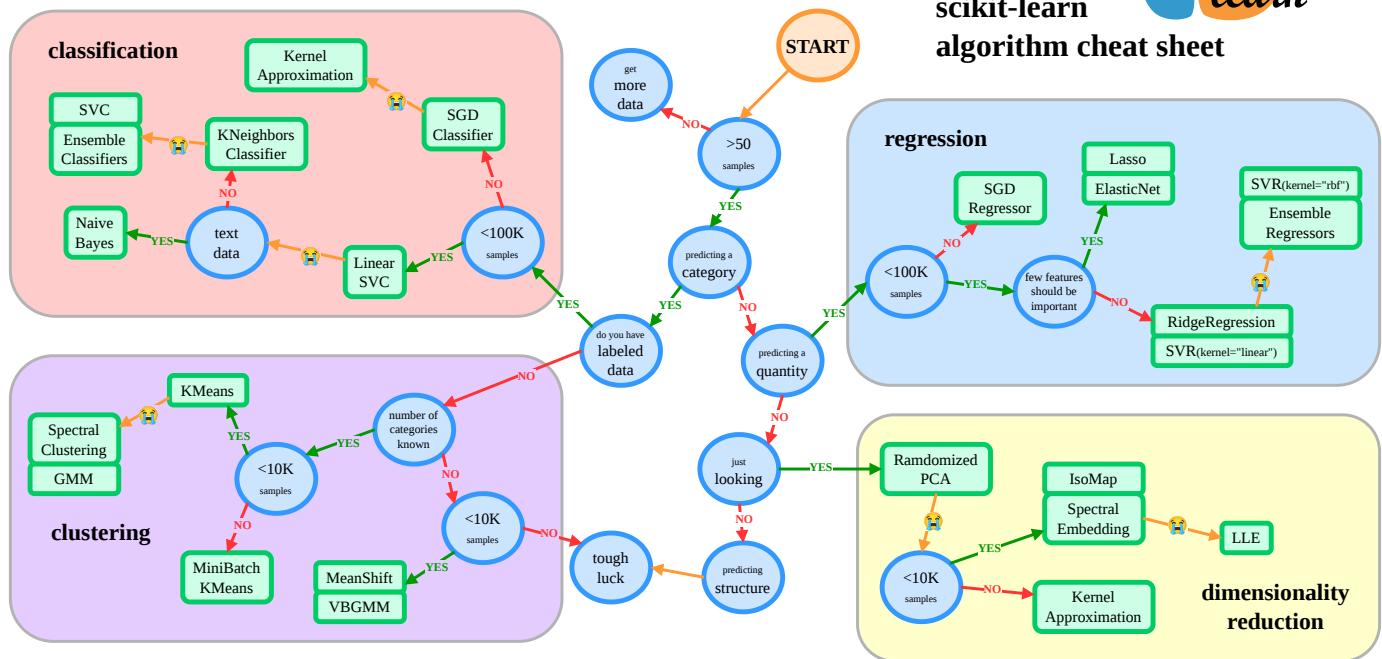
We learned classification first, because it shares similarities with each regression and clustering, while regression and clustering have less in common.

- Classification is supervised learning for a categorical target.
- Regression is supervised learning for a continuous target.
- Clustering is unsupervised learning for a categorical target.

We have used a small flowchart for the tasks:



Sklearn provides a nice [flow chart](#) for thinking through both tasks (big blocks) and models (the small green boxes)



Predicting a category is another way of saying categorical target. Predicting a quantity is another way of saying continuous target. Having labels or not is the difference between

The flowchart assumes you know what you want to do with data and that is the ideal scenario. You have a dataset and you have a goal. For the purpose of getting to practice with a variety of things, in this course we ask you to start with a task and then find a dataset. Assignment 9 is the last time that's true however. Starting with Assignment 10, you can choose and focus on a specific application domain and then choose the right task from there.

Thinking about this, however, you use this information to move between the tasks within a given type of data. For example, you can use the same data for clustering as you did for classification. Switching the task changes the questions though: classification evaluation tells us how separable the classes are given that classifiers decision rule. Clustering can find other subgroups or the same ones, so the evaluation we choose allows us to explore this in more ways.

Regression requires a continuous target, so we need a dataset to be suitable for that, we can't transform from the classification dataset to a regression one.

However, we can go the other way and that's how some classification datasets are created.

The UCI [adult](#) Dataset is a popular ML dataset that was derived from census data. The goal is to use a variety of features to predict if a person makes more than  $50k$  per year or not. While income is a continuous value, they applied a threshold ( $50k$ ) to it to make a binary variable. The dataset does not include income in dollars, only the binary indicator.

Recent work reconstructed the dataset with the continuous valued income. Their [repository](#) contains the data as well as links to their paper and a video of their talk on it.

## 18.2. Cross Validation

This week our goal is to learn how to optimize models. The first step in that is to get a good estimate of its performance.

[Skip to main content](#)

We have seen that the test train splits, which are random, influence the performance.

In order to find the best [hyperparameters](#) for the model on a given dataset, we need some better evaluation techniques.

```
# basic libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# models classes
from sklearn import tree
from sklearn import cluster
from sklearn import svm
# datasets
from sklearn import datasets
# model selection tools
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold, ShuffleSplit
from sklearn import metrics
```

This week and next week together are about model selection or choosing what model we might want to actually *use*

### 18.2.1. Loading Data with sklearn

#### Note

in class we tried to use `as_frame=True` alone, but I had misread. To figure it out, I looked at the source code to see what it does, I saw that it does not return the structure for that case, but it does call it. This is also in the git blame view where I could see it had not changed recently. What it does is change the type of those attributes in the bunch or the two items returned if in the `return_X_y`, so here.

It does say that but is an easy thing to misinterpret. This is the advantage of open source code though, worst case, even if the documentation confuses you, you can see the actual code and see it.

This way we can get dataFrames:

```
iris_X, iris_y = datasets.load_iris(return_X_y=True, as_frame=True)
type(iris_X)
```

```
pandas.core.frame.DataFrame
```

or in the bunch

```
iris_bunch_frame= datasets.load_iris(as_frame=True)
type(iris_bunch.data)
```

```
NameError                                 Traceback (most recent call last)
Cell In[3], line 2
      1 iris_bunch_frame= datasets.load_iris(as_frame=True)
----> 2 type(iris_bunch.data)

NameError: name 'iris_bunch' is not defined
```

otherwise the bunch contains numpy arrays.

```
iris_bunch= datasets.load_iris()
type(iris_bunch.data)
```

```
numpy.ndarray
```

### 18.2.2. K-fold Cross Validation

We'll use the Iris data with a decision tree.

We are going to use it this way for class:

```
iris_X, iris_y = datasets.load_iris(return_X_y=True)
```

and instantiate a Decision tree see classification notes about dt

```
dt = tree.DecisionTreeClassifier()
```

We can split the data, fit the model, then compute a score, but since the splitting is a randomized step, the score is a random variable.

For example, if we have a coin that we want to see if it's fair or not. We would flip it to test. One flip doesn't tell us, but if we flip it a few times, we can estimate the probability it is heads by counting how many of the flips are heads and dividing by how many flips.

We can do something similar with our model performance. We can split the data a bunch of times and compute the score each time.

`cross_val_score` does this all for us.

It takes an estimator object and the data.

By default it uses 5-fold cross validation. It splits the data into 5 sections, then uses 4 of them to train and one to test. It then iterates through so that each section gets used for testing.

```
cross_val_score(dt, iris_X, iris_y)
```

[Skip to main content](#)

```
array([0.96666667, 0.96666667, 0.9       , 1.       , 1.       , 1.       ])
```

We get back a score for each section or “fold” of the data. We can average those to get a single estimate.

To actually report, we would take the mean

```
np.mean(cross_val_score(dt, iris_X, iris_y))
```

```
0.9600000000000002
```

We can change from 5-fold to 10-fold by setting the `cv` parameter

```
cross_val_score(dt, iris_X, iris_y, cv=10)
```

```
array([1.       , 0.93333333, 1.       , 0.93333333, 0.93333333,
       0.86666667, 0.93333333, 0.93333333, 1.       , 1.       , 1.       ])
```

In K-fold cross validation, we split the data in `K` sections and train on `K-1` and test one 1 section. So the percentages are like:

```
# K is the cv
K = 10
train_pct =(K-1)/K
test_pct = 1/K
train_pct, test_pct
```

```
(0.9, 0.1)
```

## 18.3. Cross validation is model-agnostic

We can use *any* estimator object here.

For example, we can apply it to clustering too:

```
km = cluster.KMeans(n_clusters = 3)
```

```
cross_val_score(km, iris_X)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: super().__check_params_vs_input(X, default_n_init=10)
```

```
array([-9.062      , -14.93195873, -18.93234207, -23.61265827,
       -19.55457726])
```

## 18.4. Details and types of Cross validation

First, let's look at the Kfold cross validation object. Under the hood, this is the default thing that `cross_val_score` uses [1]

Cross validation objects are all examples of the `splitter` type in `sklearn`

we can instantiate one of these too:

```
kf = KFold(n_splits=10)
```

it returns the splits as a generator

```
splits = kf.split(iris_X, iris_y)
splits
```

```
<generator object _BaseKFold.split at 0x7f2406a8a970>
```

Jupyter does not know how to display it, so it just says the type.

```
type(splits)
```

```
generator
```

```
type(range(3))
```

```
range
```

We can use this in a loop to get the list of indices that will be used to get the test and train data for each fold. To visualize what this is doing, see below.

### ⚠️ Warning

I changed this to create an empty list and append a dataframe to it each time through the list then create a combined dataframe after to remove the

```
N_samples = len(iris_y)
kf_tt_list = []
i = 1
for train_idx, test_idx in splits:

    # make a list of "unused"
    col_data = np.asarray(['unused']*N_samples)
    # fill in train and test by the indices
    col_data[train_idx] = 'Train'
    col_data[test_idx] = 'Test'
    kf_tt_list.append(pd.DataFrame(data = col_data, index=list(range(N_samples)), columns = ['split ' + str(i) + 1]))
    i += 1

kf_tt_df = pd.concat(kf_tt_list, axis=1)
```

Note that unlike `test_train_split` this does not always randomize and shuffle the data before splitting.

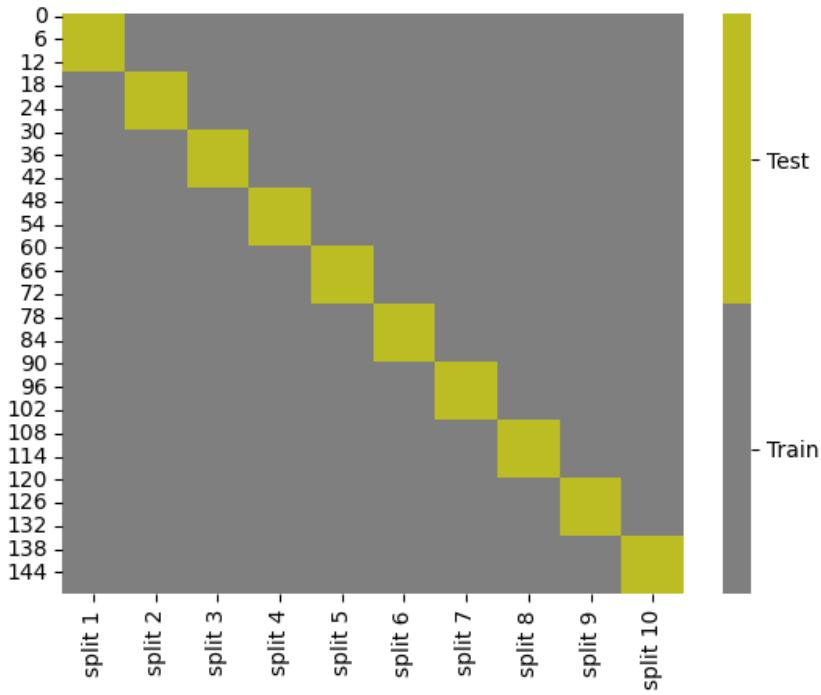
`kf_tt_df`

	split 1	split 2	split 3	split 4	split 5	split 6	split 7	split 8	split 9	split 10
<b>0</b>	Test	Train								
<b>1</b>	Test	Train								
<b>2</b>	Test	Train								
<b>3</b>	Test	Train								
<b>4</b>	Test	Train								
...	...	...	...	...	...	...	...	...	...	...
<b>145</b>	Train	Test								
<b>146</b>	Train	Test								
<b>147</b>	Train	Test								
<b>148</b>	Train	Test								
<b>149</b>	Train	Test								

150 rows × 10 columns

We can also visualize:

```
cmap = sns.color_palette("tab10", 10)
g = sns.heatmap(kf_tt_df.replace({'Test':1, 'Train':0}), cmap=cmap[7:9], cbar_kws={'ticks':[.25, .75]}, linewidths=.5, linecolor='gray')
colorbar = g.collections[0].colorbar
colorbar.set_ticklabels(['Train', 'Test'])
```



We can also see another splitter(this is what `train_test_split` uses)

```

skf = ShuffleSplit(10)
N_samples = len(iris_y)
ss_tt_list = []
i = 1
for train_idx, test_idx in skf.split(iris_X, iris_y):
    col_data = np.asarray(['unused']*N_samples)
    col_data[train_idx] = 'Train'
    col_data[test_idx] = 'Test'
    ss_tt_list.append(pd.DataFrame(data = col_data, index=list(range(N_samples))), columns=['split' + str(i)))
    i +=1

ss_tt_df = pd.concat(ss_tt_list, axis=1)
ss_tt_df

```

```

-----
TypeError                                 Traceback (most recent call last)
Cell In[20], line 9
      7     col_data[train_idx] = 'Train'
      8     col_data[test_idx] = 'Test'
--> 9     ss_tt_list.append(pd.DataFrame(data = col_data, index=list(range(N_samples))), columns=['split' + str(i)))
     10     i +=1
     12 ss_tt_df = pd.concat(ss_tt_list, axis=1)

TypeError: append() takes no keyword arguments

```

```

cmap = sns.color_palette("tab10",10)
g = sns.heatmap(ss_tt_df.replace({'Test':1,'Train':0}),cmap=cmap[7:9],cbar_kws={'ticks':[.25,.75]},linew=1,linewidths=1, linecolor='gray')
colorbar = g.collections[0].colorbar
colorbar.set_ticklabels(['Train','Test'])

```

```

-----
NameError                                 Traceback (most recent call last)
Cell In[21], line 2
  1 cmap = sns.color_palette("tab10", 10)
----> 2 g = sns.heatmap(ss_tt_df.replace({'Test':1,'Train':0}),cmap=cmap[7:9],cbar_kws={'ticks':[.25,.75]
  3     linecolor='gray')
  4 colorbar = g.collections[0].colorbar
  5 colorbar.set_ticklabels(['Train','Test'])

NameError: name 'ss_tt_df' is not defined

```

we can see this is different and more random.

We can also use it in `cross_val_score`

```
cross_val_score(dt,iris_X,iris_y,cv=ShuffleSplit(n_splits=100,test_size=.2))
```

```
array([1.        , 0.93333333, 0.93333333, 0.96666667, 0.93333333,
       0.96666667, 0.96666667, 0.9        , 0.96666667, 0.96666667,
       0.96666667, 0.93333333, 1.        , 0.96666667, 0.96666667,
       0.96666667, 1.        , 0.9        , 0.93333333, 0.96666667,
       0.9        , 1.        , 0.96666667, 0.9        , 0.93333333,
       0.86666667, 0.9        , 0.83333333, 0.96666667, 0.96666667,
       0.96666667, 0.96666667, 0.9        , 0.9        , 0.9        ,
       0.96666667, 0.9        , 0.93333333, 0.93333333, 0.96666667,
       0.96666667, 0.93333333, 0.93333333, 1.        , 0.93333333,
       0.93333333, 0.86666667, 0.96666667, 0.96666667, 0.96666667,
       0.9        , 0.93333333, 0.9        , 0.96666667, 0.93333333,
       0.93333333, 0.96666667, 0.93333333, 0.96666667, 0.9        ,
       0.93333333, 1.        , 1.        , 0.93333333, 0.96666667,
       0.86666667, 1.        , 0.86666667, 0.9        , 0.96666667,
       0.96666667, 0.93333333, 0.93333333, 0.96666667, 0.96666667,
       0.96666667, 0.93333333, 0.93333333, 0.93333333, 0.96666667,
       0.93333333, 0.93333333, 0.96666667, 0.93333333, 0.96666667,
       0.83333333, 0.93333333, 1.        , 0.9        , 1.        ,
       0.96666667, 0.96666667, 1.        , 1.        , 0.93333333,
       0.96666667, 0.9        , 0.93333333, 0.93333333, 0.96666667])
```

## 18.5. Questions after class

### 18.5.1. When should you change test and train sizes rather than just using the default?

Having more training data often gets you a better model, having more test samples gives you a better idea of how well the model will work on new data. You have to make that choice.

I think sometimes trying a few and seeing if the performance changes much or not, can also help you decide, if it doesn't matter much, then use the default. If it changes a lot, then maybe use as much data as possible for training and you can go as far as KFold cross validation with `K = len(X)` which is called leave-one-out cross validation.

[1] actually above, since we used a classifier it actually used stratified cross validation which solves the problem we

the same percentage of each class for the classifier. It only does this if the estimator is a classifier, for clustering and regression it always does regular Kfold.

## 19. Model Optimization

### ! Important

Remember that best is context dependent and relative. The best accuracy might not be the best overall. Automatic optimization can only find the best thing in terms of a single score.

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn import cluster
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import tree
```

### 19.1. Model validation

We will work with the iris data again.

```
iris_df = sns.load_dataset('iris')

iris_X = iris_df.drop(columns=['species'])

iris_y = iris_df['species']
```

We will still use the test train split to keep our test data separate from the data that we use to find our preferred parameters.

```
iris_X_train, iris_X_test, iris_y_train, iris_y_test = train_test_split(iris_X,iris_y,
random_state=0)
```

Today we will optimize a decision tree over three parameters. One is the criterion, which is how it decides where to create thresholds in parameters. Gini is the default and it computes how concentrated each class is at that node, another is entropy, entropy is, generally how random something is. Intuitively these do similar things, which makes sense because they are two ways to make the same choice, but they have slightly different calculations.

The other two parameters relate to the structure of the decision tree that is produced.

- Max depth is the height of the tree
- min samples per leaf makes it keeps the leaf sizes small.

the  
mat  
ske

```
dt = tree.DecisionTreeClassifier()
params_dt = {'criterion':['gini','entropy'],'max_depth':[2,3,4],
             'min_samples_leaf':list(range(2,20,2))}
```

We will fit it with default CV settings.

```
dt_opt = GridSearchCV(dt,params_dt)
```

then we can fit

```
dt_opt.fit(iris_X_train,iris_y_train)
```

```
►      GridSearchCV
► estimator: DecisionTreeClassifier
    ► DecisionTreeClassifier
```

we can use `ti` to get predictions

```
y_pred = dt_opt.predict(iris_X_test)
```

we can also score it as regular.

```
dt_opt.score(iris_X_test,iris_y_test)
```

```
0.9473684210526315
```

we can also see the best parameters.

```
dt_opt.best_params_
```

```
{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 2}
```

we can look at the overall results this way:

```
dt_5cv_df = pd.DataFrame(dt_opt.cv_results_)
dt_5cv_df.head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_
0	0.001635	0.000207	0.001413	0.000297	gini	2	
1	0.001498	0.000017	0.001244	0.000018	gini	2	
2	0.001503	0.000033	0.001252	0.000021	gini	2	
3	0.001482	0.000009	0.001237	0.000016	gini	2	
4	0.001491	0.000014	0.001239	0.000017	gini	2	

To see more carefully what it does, we can look at its shape.

```
dt_5cv_df.shape
```

```
(54, 16)
```

we can see that the total number of rows matched the product of the length of each list of parameters to try.

```
np.product([len(param_list) for param_list in params_dt.values()])
```

```
54
```

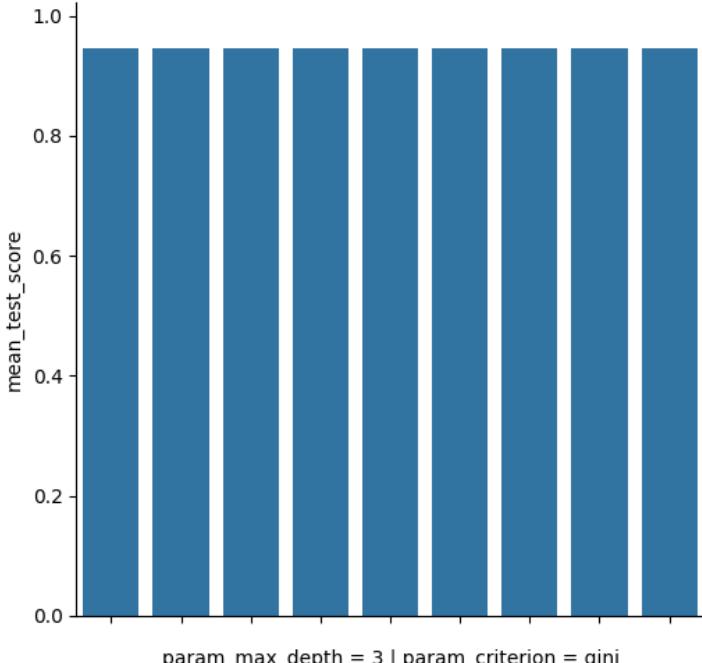
This means that it tests every combination of features– without us writing a bunch of nested loops.

We can also plot the data and look at the performance.

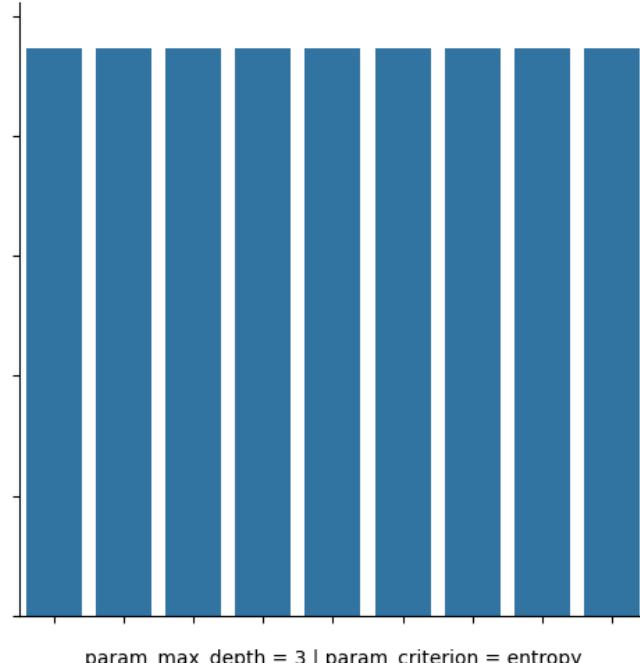
```
sns.catplot(data=dt_5cv_df,x='param_min_samples_leaf',y='mean_test_score',
            col='param_criterion', row= 'param_max_depth', kind='bar',)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb45e33be0>
```

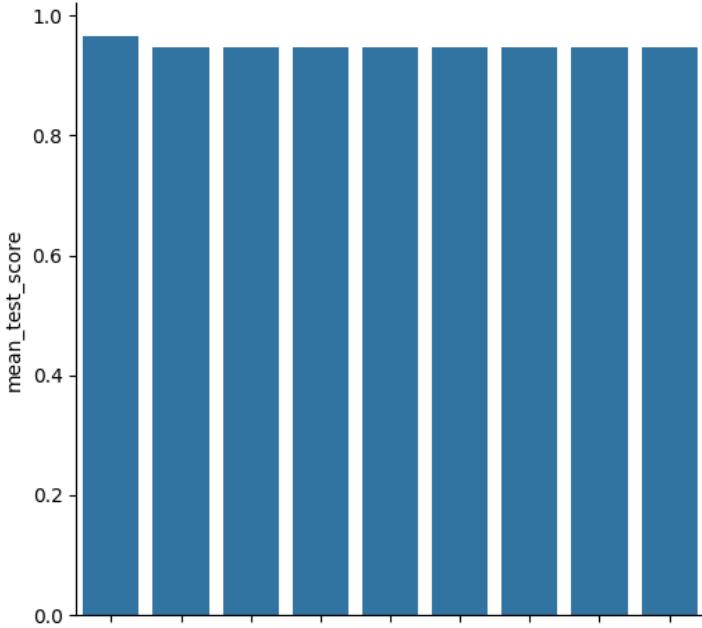
param\_max\_depth = 2 | param\_criterion = gini



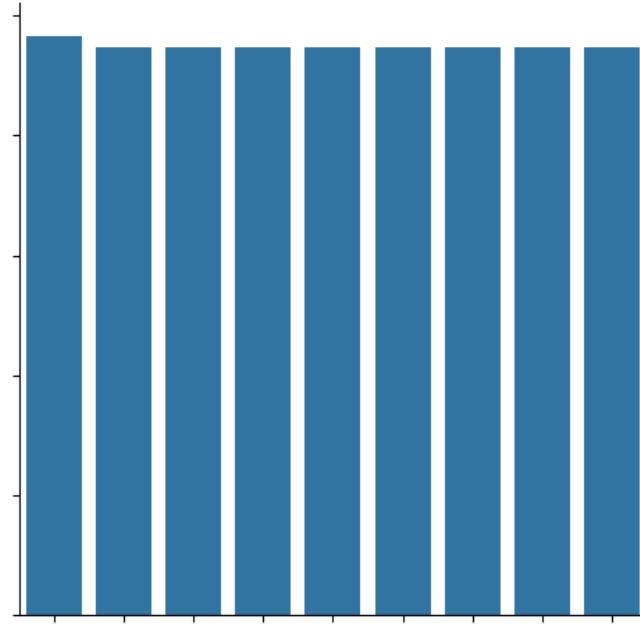
param\_max\_depth = 2 | param\_criterion = entropy



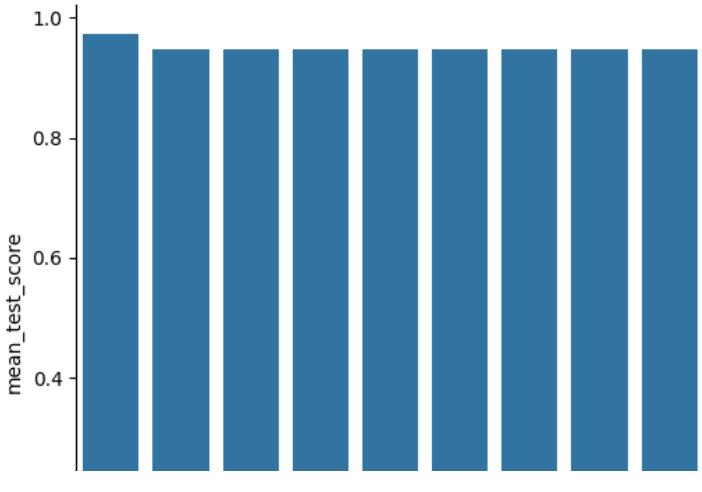
param\_max\_depth = 3 | param\_criterion = gini



param\_max\_depth = 3 | param\_criterion = entropy



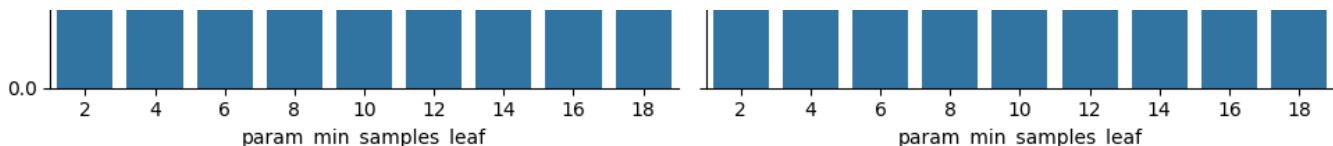
param\_max\_depth = 4 | param\_criterion = gini



param\_max\_depth = 4 | param\_criterion = entropy



Skip to main content



this makes it clear that none of these stick out much in terms of performance.

## 19.2. Impact of CV parameters

```
dt_opt10 = GridSearchCV(dt, params_dt, cv=10)
dt_opt10.fit(iris_X_train, iris_y_train)
```

```
►      GridSearchCV
► estimator: DecisionTreeClassifier
    ► DecisionTreeClassifier
```

```
dt_10cv_df = pd.DataFrame(dt_opt10.cv_results_)
```

We can stack the columns we want from the two results together with a new indicator column `cv`

```
plot_cols = ['param_min_samples_leaf', 'std_test_score', 'mean_test_score',
            'param_criterion', 'param_max_depth', 'cv']
dt_10cv_df['cv'] = 10
dt_5cv_df['cv'] = 5

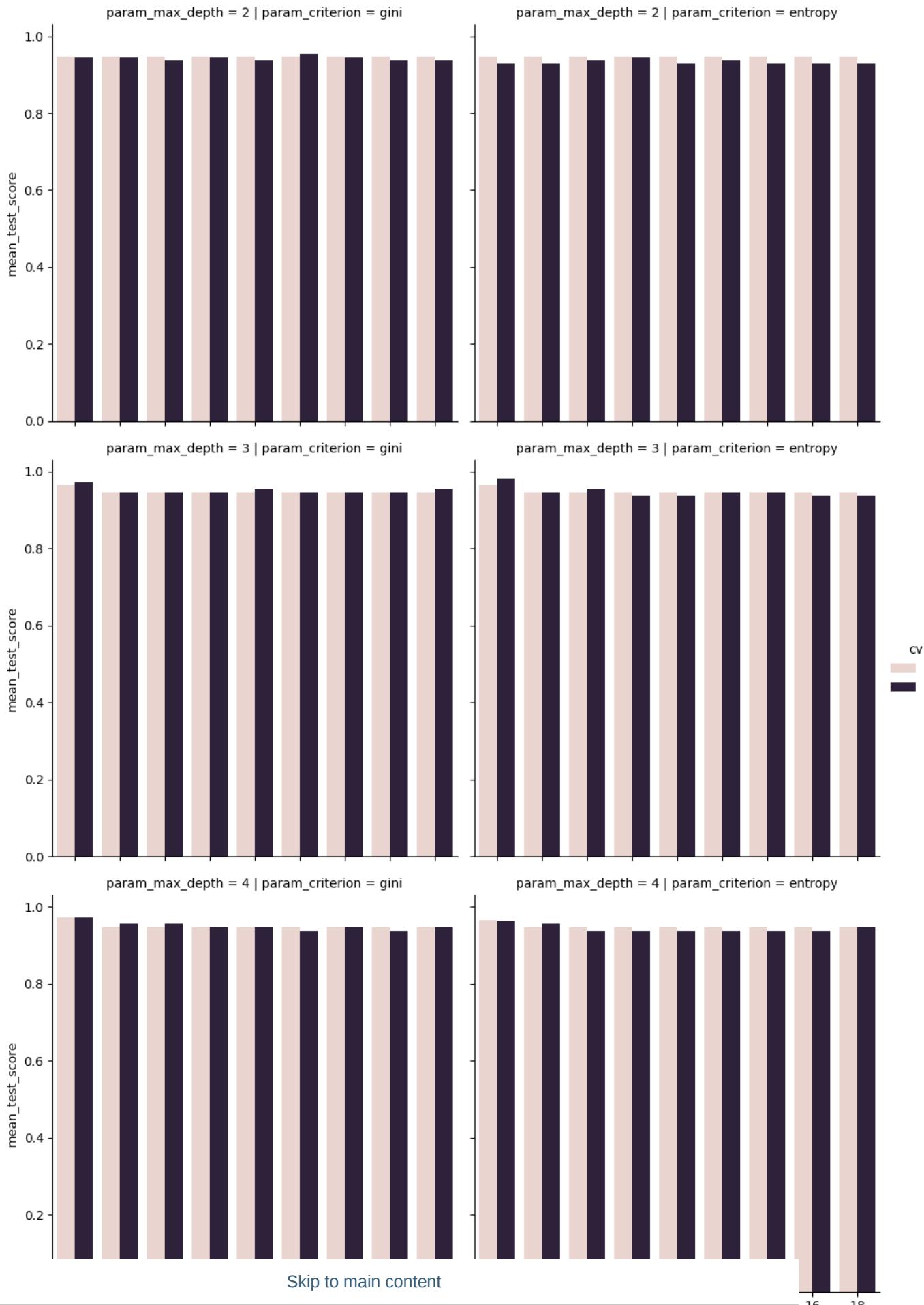
dt_cv_df = pd.concat([dt_5cv_df[plot_cols], dt_10cv_df[plot_cols]])
dt_cv_df.head()
```

	param_min_samples_leaf	std_test_score	mean_test_score	param_criterion	param_max_depth	cv
0	2	0.033305	0.94664	gini	2	5
1	4	0.033305	0.94664	gini	2	5
2	6	0.033305	0.94664	gini	2	5
3	8	0.033305	0.94664	gini	2	5
4	10	0.033305	0.94664	gini	2	5

this can be used to plot.

```
sns.catplot(data=dt_cv_df, x='param_min_samples_leaf', y='mean_test_score',
            col='param_criterion', row='param_max_depth', kind='bar',
            hue = 'cv')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb459d1b20>
```

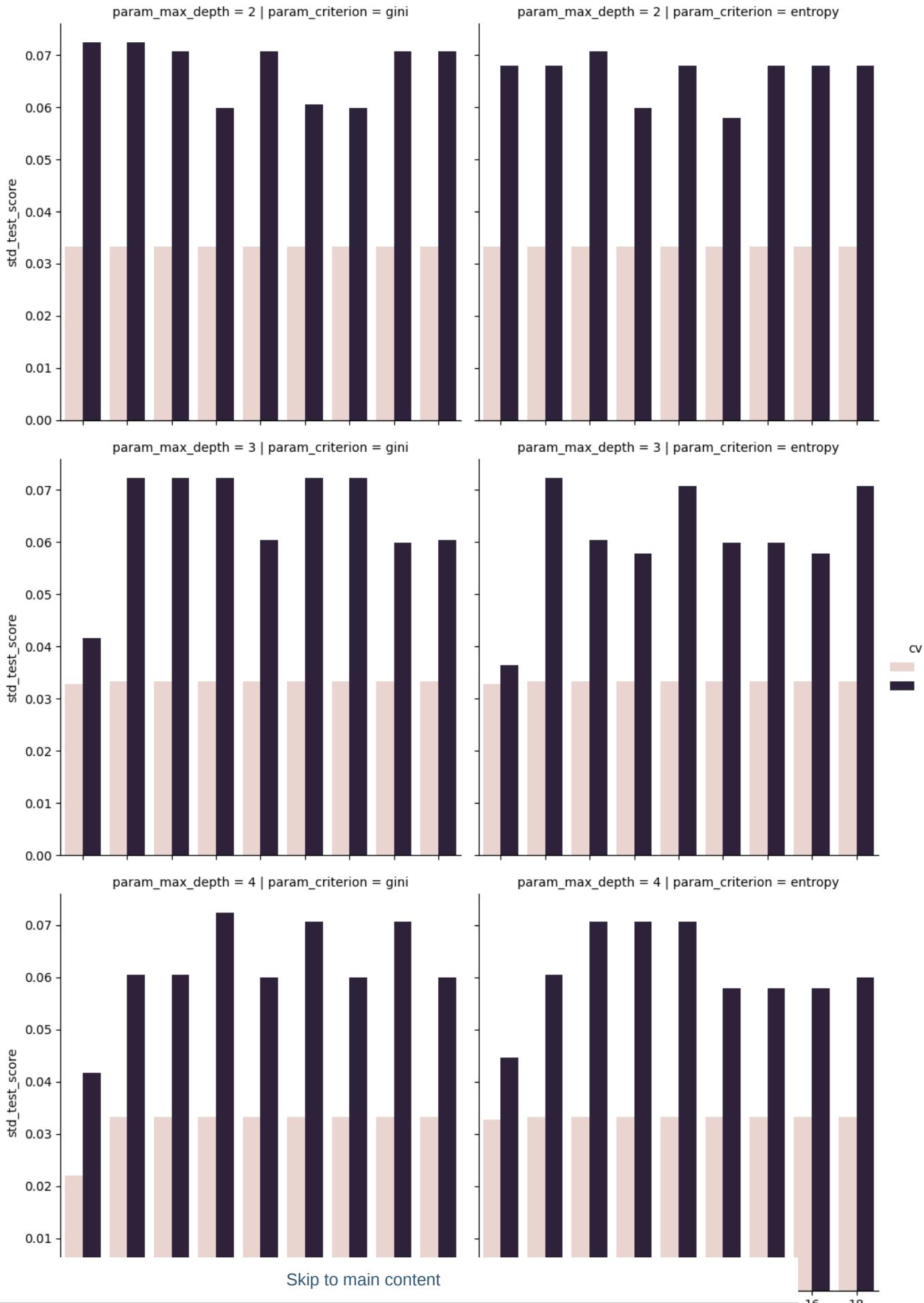


```
param_min_samples_leaf param_min_samples_leaf
```

we see that the mean scores are not very different, but that 10 is a little higher in some cases. This makes sense, it has more data to learn from, so it found something that applied better, on average, to the test set.

```
sns.catplot(data=dt_cv_df,x='param_min_samples_leaf',y='std_test_score',
            col='param_criterion', row= 'param_max_depth', kind='bar',
            hue = 'cv')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb4326dbb0>
```



```
<   <   o   o   10   12   14   10   10   <   <   o   o   10   12   14   10   10  
      param_min_samples_leaf
```

However here we see that the variability in those scores is much higher, so maybe the 5 is better.

We can compare to see if it finds the same model as best:

```
dt_opt.best_params_
```

```
{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 2}
```

```
dt_opt10.best_params_
```

```
{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 2}
```

In some cases they will and others they will not.

```
dt_opt.score(iris_X_test,iris_y_test)
```

```
0.9473684210526315
```

```
dt_opt10.score(iris_X_test,iris_y_test)
```

```
0.9736842105263158
```

In some cases they will find the same model and score the same, but at other times they will not.

The takeaway is that the cross-validation parameters impact our ability to measure the score and possibly how close that cross-validation mean score will match the true test score. Mostly it will change the variability in the estimate of the score. It does not change necessarily which model is best, that is up to the data itself (the original test/train split would impact this).

### 💡 Try it yourself

Does this vary if you repeat this with different test sets? How much does it depend on that? Does repeating it produce the same scores? Does one take longer to fit or score?

## 19.3. Other searches

```
from sklearn import model_selection  
from sklearn.model_selection import LeaveOneOut
```

```
rand_opt = model_selection.RandomizedSearchCV(dt,params_dt).fit(iris_X_train,
```

[Skip to main content](#)

```
rand_opt.score(iris_X_test,iris_y_test)
```

```
0.8947368421052632
```

```
rand_opt.best_params_
```

```
{'min_samples_leaf': 18, 'max_depth': 3, 'criterion': 'gini'}
```

It might find the same solution, but it also might not. If you do some and see that the parameters overall do not impact the scores much, then you can trust whichever one, or consider other criteria to choose the best model to use.

## 20. Model Comparison

To compare models, we will first optimize the parameters of two different models and look at how the different parameters settings impact the model comparison. Later, we'll see how to compare across models of different classes.

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn import cluster
from sklearn import svm
from sklearn import tree
# import the whole model selection module
from sklearn import model_selection
sns.set_theme(palette='colorblind')
```

We'll use the iris data again.

Remember, we need to split the data into training and test. The cross validation step will help us optimize the parameters, but we don't want **data leakage** where the model has seen the test data multiple times. So, we split the data here for train and test and the cross validation splits the training data into train and "test" again, but that "test" is better termed validation.

```
# load and split the data
iris_X, iris_y = datasets.load_iris(return_X_y=True)
iris_X_train, iris_X_test, iris_y_train, iris_y_test = model_selection.train_test_split(
    iris_X,iris_y, test_size=.2)
```

Then we can make the object, the parameter grid dictionary and the Grid Search object. We split these into separate cells, so that we can use the built in help to see more detail.

```

# create dt,
dt = tree.DecisionTreeClassifier()

# set param grid
params_dt = {'criterion':['gini', 'entropy'],
             'max_depth':[2,3,4,5,6],
             'min_samples_leaf':list(range(2,20,2))}

# create optimizer
dt_opt = model_selection.GridSearchCV(dt,params_dt,cv=10)

# optimize the dt parameters
dt_opt.fit(iris_X_train,iris_y_train)

# store the results in a dataframe
dt_df = pd.DataFrame(dt_opt.cv_results_)

# create svm, its parameter grid and optimizer
svm_clf = svm.SVC()
param_grid = {'kernel':['linear','rbf'], 'C':[.5, .75,1,2,5,7, 10]}
svm_opt = model_selection.GridSearchCV(svm_clf,param_grid,cv=10)

# optimize the svm put the CV results in a dataframe
svm_opt.fit(iris_X_train,iris_y_train)
sv_df = pd.DataFrame(svm_opt.cv_results_)

```

Then we fit the Grid search using the training data, and remember this actually resets the parameters and then cross validates multiple times.

We fit both a decision tree and an SVM for this case.

## 20.1. What does the grid search do?

### ! Important

The notes from the last class include more detail than was covered in class time on November 7, o make it up.  
Please see that, it includes that there are *other* types of searches that might be better in some applications

It fits and scores the model with each set of parameters on each fold of the data.

It saves results for each set of parameters and each fold of cross validaiton.

dt\_df.shape

(90, 21)

We can confirm this matches what our parameter grid said, here we pick out the values from the parameter dictionary then store those in a list (via list comprehension) then we multiply all of those together:

np.product([len(param\_list) for param\_list in params\_dt.values()])

[Skip to main content](#)

this matches the number of rows above

we can see how much it stores for each of these parameter settings

```
dt_df.head(2)
```

	<code>mean_fit_time</code>	<code>std_fit_time</code>	<code>mean_score_time</code>	<code>std_score_time</code>	<code>param_criterion</code>	<code>param_max_depth</code>	<code>param_</code>
<b>0</b>	0.000569	0.000166	0.000440	0.000037	gini		2
<b>1</b>	0.000497	0.000015	0.000425	0.000016	gini		2

2 rows × 21 columns

The `GridSearchCV` object also reveals the model with the best test score to be used with `score` and `predict`:

```
dt_opt.score(iris_X_test,iris_y_test), svm_opt.score(iris_X_test,iris_y_test)
```

```
(0.9666666666666667, 1.0)
```

## 21. Comparing performance

We can compare how much time each model takes to score (make predictions) for each model.

```
dt_df['mean_score_time'].describe()
```

```
count    90.000000
mean     0.000415
std      0.000010
min     0.000406
25%     0.000410
50%     0.000412
75%     0.000416
max     0.000485
Name: mean_score_time, dtype: float64
```

```
sv_df['mean_score_time'].describe()
```

```
count      14.000000
mean       0.000449
std        0.000009
min        0.000437
25%        0.000442
50%        0.000448
75%        0.000455
max        0.000465
Name: mean_score_time, dtype: float64
```

In these, we can look at the means but also the standard deviation, it is important to understand how sensitive each classifier is to the parameters. If a model is really sensitive to the parameters, that can mean it might not transfer as well, or at least warrant further optimization.

Which model's performance is more sensitive to the parameters?

```
dt_df['mean_test_score'].describe()
```

```
count      90.000000
mean       0.915370
std        0.014986
min        0.900000
25%        0.908333
50%        0.908333
75%        0.916667
max        0.966667
Name: mean_test_score, dtype: float64
```

```
sv_df['mean_test_score'].describe()
```

```
count      14.000000
mean       0.961310
std        0.014099
min        0.933333
25%        0.958333
50%        0.966667
75%        0.966667
max        0.983333
Name: mean_test_score, dtype: float64
```

Again here, the better fit model was similar, but this is something that we can look at the broader statistics

## 21.1. How does the best model do?

We can get the best ranked model

```
dt_df['rank_test_score'].idxmin()
```

and look at its performance to compare the best parameter values of each model

```
dt_df.loc[dt_df['rank_test_score'].idxmin()]
```

```
mean_fit_time          0.000501
std_fit_time           0.000011
mean_score_time        0.000415
std_score_time         0.000009
param_criterion        gini
param_max_depth        3
param_min_samples_leaf 2
params                 {'criterion': 'gini', 'max_depth': 3, 'min_sam...
split0_test_score      1.0
split1_test_score      1.0
split2_test_score      0.916667
split3_test_score      1.0
split4_test_score      0.916667
split5_test_score      1.0
split6_test_score      0.916667
split7_test_score      1.0
split8_test_score      1.0
split9_test_score      0.916667
mean_test_score        0.966667
std_test_score         0.040825
rank_test_score        1
Name: 9, dtype: object
```

```
sv_df.loc[sv_df['rank_test_score'].idxmin()]
```

```
mean_fit_time          0.000645
std_fit_time           0.000011
mean_score_time        0.000445
std_score_time         0.000013
param_C                1
param_kernel            linear
params                 {'C': 1, 'kernel': 'linear'}
split0_test_score      1.0
split1_test_score      1.0
split2_test_score      1.0
split3_test_score      1.0
split4_test_score      1.0
split5_test_score      1.0
split6_test_score      0.916667
split7_test_score      1.0
split8_test_score      1.0
split9_test_score      0.916667
mean_test_score        0.983333
std_test_score         0.033333
rank_test_score        1
Name: 4, dtype: object
```

## 21.2. What about training time

In some contexts it might be important to also check how long it takes to train, for example if you will train and retrain a lot on changing data.

[Skip to main content](#)

```
dt_df['mean_fit_time'].describe()
```

```
count    90.000000
mean     0.000499
std      0.000012
min     0.000479
25%     0.000494
50%     0.000499
75%     0.000503
max     0.000569
Name: mean_fit_time, dtype: float64
```

```
sv_df['mean_fit_time'].describe()
```

```
count    14.000000
mean     0.000679
std      0.000033
min     0.000640
25%     0.000657
50%     0.000673
75%     0.000699
max     0.000745
Name: mean_fit_time, dtype: float64
```

## 21.3. How sensitive to the parameter values is each model?

We can look at a lot of different things to check this. First, we can look back above at the `std` in the describe tables above. That is how much the parameter values changed the different measurements we took.

Another thing we can look at is the ranks. If most of the models are ranked the same, that means they all got the same score, so the parameter values did not change the accuracy. That is not sensitive. This helps us see a bit more detail in how sensitive when the std are small already.

If there are no ties, that shows some sensitivity, and then it would be important to look at the std to see if it is small or not.

### ! Important

the below may be different due to random splits

## 21.4. More Complete comparison

A coomplete comparison will compare across multiple metrics and consider the relative important of those metrics on the particular context.

We can look at the performance, here the score is the accuracy and we could also look at other performance metrics.

We can compare them on time: the training time or the test time (more important).

## 21.5. Interpretability

We can also compare models on their interpretability: a decision tree is easy to explain how it makes a decision.

An SVM is somewhat harder and a neural network that we will see after Thanksgiving is even more complex.

If the model is going in a webapp, it may not matter if it is interpretable, but if it is being used with experts of some sort, they will likely care about its interpretability.

For example, in healthcare, interpretability is often valued. I shared a [story](#) about a model that performed well on one metric, but looking at how it worked raised concerns. That link leads to the person who trained that model telling the story.

## 21.6. Model type

For classifiers, we can compare the type of classifier, or what it learns. We can compare if it is generative (describes the data, could generate synthetic data) or discriminative (describes the decision rule). A generative model might be preferred even with lower accuracy by a scientist who wants to understand the data. It can also help to give ideas about what you might do to improve the model. If you just need the most accuracy, like if you are placing ads, you would typically use a discriminative model because it is complex, and you just want accuracy you don't need to understand.

```
dt_opt.score(iris_X_test,iris_y_test), svm_opt.score(iris_X_test,iris_y_test)
```

```
(0.9666666666666667, 1.0)
```

```
len(iris_y_test)
```

```
30
```

## 21.7. Questions

### 21.7.1. Is there a sample size restriction to compare models?

No, but you should compare them on the *same* data (for training and testing)

### 21.7.2. Does GNB also have these mean\_score, mean\_time parameters that DT and SVC?

These parameters are of the `GridSearchCV` object, not the `DecisionTreeClassifier` or the `SVC`. Therefore, they exist for any `GridSearchCV` no matter what estimator object is used to instantiate it. So, yes, for a `GridSearchCV` over a `GaussianNB` these values would exist.

### 21.7.3. how does the size of the parameter grid impact the requirements of the grid search?

It impacts how long it will take to run.

If this answer is not enough for you, please write to clarify your question and I will expand.

### 21.7.4. are they still using the ai to see patient priority in hospitals?

The model I told the [story](#) about never got deployed. ML is used to varying degrees in hospitals. There is a lot of research on machine learning for healthcare see for example the list of [2023 papers](#)

this is an example of a well designed successfully deployed model. You can read it through URI, let me know if you do not know how to use the library to get access.

### 21.7.5. Why does it matter how long it takes to fit data when the time is almost instantaneous?

When it is this low, we likely would not care. The minor difference in score time could add up because even if it is .0001s but it is called 3 million times per day, then the time(and associated compute time, and energy costs, etc) adds up.

### 21.7.6. when models are trained at an enterprise level / for an enterprise applications is pandas & scikit-learn still used or is a lower-level language/package used (like for netflix)

What data scientists do is typically still using pandas and scikit-learn. For time-critical or low memory location deployed model, the fit model may be re-implemented in a faster language.

## 22. Which model should we deploy?

Here, we use the [ucimlrepo](#) library to load a dataset.

```
from ucimlrepo import fetch_ucirepo
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn import cluster
from sklearn import svm
from sklearn import tree
# import the whole model selection module
from sklearn import model_selection
from myst_nb import glue
from IPython.display import Markdown
sns.set_theme(palette='colorblind')

# fetch dataset
phishing_websites = fetch_ucirepo(id=327)
```

First, let's learn about the data a little:

▶ Show code cell source

**Uci\_Id**

---

327

**Name**

---

Phishing Websites

**Repository\_Url**

---

<https://archive.ics.uci.edu/dataset/327/phishing+websites>

**Data.Url**

---

<https://archive.ics.uci.edu/static/public/327/data.csv>

**Abstract**

---

This dataset collected mainly from: PhishTank archive, MillerSmiles archive, Googleâ€™s searching operators.

**Area**

---

Computer Science

**Tasks**

---

['Classification']

**Characteristics**

---

['Tabular']

**Num\_Instances**

---

11055

**Num\_Features**

---

30

**Feature\_Types**

---

['Integer']

**Demographics**

---

[Skip to main content](#)

**Target\_Col**

---

['result']

**Index\_Col**

---

None

**Has\_Missing\_Values**

---

no

**Missing\_Values\_Symbol**

---

None

**Year\_Of\_Dataset\_Creation**

---

2012

**Last\_Updated**

---

Tue Mar 05 2024

**Dataset\_Doi**

---

10.24432/C51W2X

**Creators**

---

['Rami Mohammad', 'Lee McCluskey']

**Intro\_Paper**

---

{'ID': 396, 'type': 'NATIVE', 'title': 'An assessment of features related to phishing websites using an automated technique', 'authors': 'R. Mohammad, F. Thabtah, L. McCluskey', 'venue': 'International Conference for Internet Technology and Secured Transactions', 'year': 2012, 'journal': None, 'DOI': None, 'URL': 'https://www.semanticscholar.org/paper/An-assessment-of-features-related-to-phishing-using-Mohammad-Thabtah/0c0ff58063f4e078714ea74f112bc709ba9fed06', 'sha': None, 'corpus': None, 'arxiv': None, 'mag': None, 'acl': None, 'pmid': None, 'pmcid': None}

**Additional\_Info**

---

{'summary': 'One of the challenges faced by our research was the unavailability of reliable training datasets. In fact this challenge faces any researcher in the field. However, although plenty of articles about predicting phishing websites have been disseminated these days, no reliable training dataset has been published publically, may be because there is no agreement in literature on the definitive features that characterize phishing webpages, hence it is difficult to shape a dataset that covers all and effective in'}

[Skip to main content](#)

predicting phishing websites. In addition, we propose some new features.', 'purpose': None, 'funded\_by': None, 'instances\_represent': None, 'recommended\_data\_splits': None, 'sensitive\_data': None, 'preprocessing\_description': None, 'variable\_info': 'For Further information about the features see the features file in the data folder.', 'citation': None}

```
# variable information
print(phishing_websites.variables)
```

		name	role	type	demographic	description	\
0		having_ip_address	Feature	Integer	None	None	
1		url_length	Feature	Integer	None	None	
2		shortining_service	Feature	Integer	None	None	
3		having_at_symbol	Feature	Integer	None	None	
4		double_slash_redirecting	Feature	Integer	None	None	
5		prefix_suffix	Feature	Integer	None	None	
6		having_sub_domain	Feature	Integer	None	None	
7		sslfinal_state	Feature	Integer	None	None	
8		domain_registration_length	Feature	Integer	None	None	
9		favicon	Feature	Integer	None	None	
10		port	Feature	Integer	None	None	
11		https_token	Feature	Integer	None	None	
12		request_url	Feature	Integer	None	None	
13		url_of_anchor	Feature	Integer	None	None	
14		links_in_tags	Feature	Integer	None	None	
15		sfh	Feature	Integer	None	None	
16		submitting_to_email	Feature	Integer	None	None	
17		abnormal_url	Feature	Integer	None	None	
18		redirect	Feature	Integer	None	None	
19		on_mouseover	Feature	Integer	None	None	
20		rightclick	Feature	Integer	None	None	
21		popupwindow	Feature	Integer	None	None	
22		iframe	Feature	Integer	None	None	
23		age_of_domain	Feature	Integer	None	None	
24		dnsrecord	Feature	Integer	None	None	
25		web_traffic	Feature	Integer	None	None	
26		page_rank	Feature	Integer	None	None	
27		google_index	Feature	Integer	None	None	
28		links_pointing_to_page	Feature	Integer	None	None	
29		statistical_report	Feature	Integer	None	None	
30		result	Target	Integer	None	None	
		units missing_values					
0	None	no					
1	None	no					
2	None	no					
3	None	no					
4	None	no					
5	None	no					
6	None	no					
7	None	no					
8	None	no					
9	None	no					
10	None	no					
11	None	no					
12	None	no					
13	None	no					
14	None	no					
15	None	no					
16	None	no					
17	None	no					
18	None	no					
19	None	no					
20	None	no					
21	None	no					
22	None	no					
23	None	no					
24	None	no					
25	None	no					
26	None	no					
27	None	no					
28	None	no					
29	None	no					
30	None	no					

[Skip to main content](#)

Next, we can pull out the features and targets:

```
# data (as pandas dataframes)
phishing_X = phishing_websites.data.features[:500]
phishing_y = phishing_websites.data.targets [:500]
```

```
rows,cols = phishing_X.shape
glue("num_rows", rows, display=False)
glue("num_cols", cols, display=False)
```

we imported only 500 rows of the data which has 30 variables

Now the target

```
phishing_y.shape
```

```
(500, 1)
```

```
phishing_X.head()
```

	having_ip_address	url_length	shortining_service	having_at_symbol	double_slash_redirecting	prefix_suffi
0	-1	1		1	1	-1
1	1	1		1	1	1
2	1	0		1	1	1
3	1	0		1	1	1
4	1	0		-1	1	1

5 rows × 30 columns

Now we can split into train and test data

```
phishing_X_train, phishing_X_test, phishing_y_train, phishing_y_test = model_selection.train_test_split(
    phishing_X,phishing_y, test_size =.2)
```

and then we can fit and optimize models

```
# create dt,
dt = tree.DecisionTreeClassifier()

# set param grid
params_dt = {'criterion':['gini','entropy'],
             'max_depth':list(range(4,45,3)),
             'min_samples_leaf':list(range(2,20,2))}
# create optimizer
dt_opt = model_selection.GridSearchCV(dt,params_dt,cv=10)

# optimize the dt parameters
dt_opt.fit(phishing_X_train,phishing_y_train)

# store the results in a dataframe
dt_df = pd.DataFrame(dt_opt.cv_results_)

# create svm, its parameter grid and optimizer
svm_clf = svm.SVC()
param_grid = {'kernel':['linear','rbf'], 'C':[.5, .75,1,2,5,7, 10]}
svm_opt = model_selection.GridSearchCV(svm_clf,param_grid,cv=10)

# optmize the svm put the CV results in a dataframe
svm_opt.fit(phishing_X_train,phishing_y_train)
sv_df = pd.DataFrame(svm_opt.cv_results_)
```





```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/utils/validation.py:1183: DataWarning: y = column_or_1d(y, warn=True)
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/utils/validation.py:1183: DataWarning: y = column_or_1d(y, warn=True)
```



```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/utils/validation.py:1183: DataWarning: y = column_or_1d(y, warn=True)
```

and then we can start to design our comparison analysis.

## 22.1. Overall Performance

```
acc_dt = dt_opt.score(phishing_X_test, phishing_y_test) *100
acc_sv = svm_opt.score(phishing_X_test, phishing_y_test) *100

two_scores = pd.Series({'decision tree':acc_dt,
                       'support vector machine':acc_sv})

winner = two_scores.idxmax()

glue("dt_test", acc_dt, display=False)
glue("sv_test", acc_sv, display=False)
glue("best_model", winner, display=False)
```

The decisions tree test score was: 86.0 and the SVM test score was 86.0, so on performance alone, ‘decision tree’ is the better one.

## 22.2. Reliability

```
dt_df.loc[dt_opt.best_index_]
```

```
mean_fit_time          0.002554
std_fit_time           0.000029
mean_score_time        0.003054
std_score_time         0.000031
param_criterion        gini
param_max_depth        34
param_min_samples_leaf 2
params                 {'criterion': 'gini', 'max_depth': 34, 'min_sa...
split0_test_score      0.925
split1_test_score       0.9
split2_test_score       0.925
split3_test_score       0.95
split4_test_score       0.95
split5_test_score       0.95
split6_test_score       0.9
split7_test_score       0.95
split8_test_score       0.95
split9_test_score       0.9
mean_test_score         0.93
std_test_score          0.021794
rank_test_score         1
Name: 90, dtype: object
```

```
dt_df['mean_test_score'].describe()
```

```
count    252.000000
mean     0.901319
std      0.009659
min      0.882500
25%     0.892500
50%     0.902500
75%     0.905000
max      0.930000
Name: mean_test_score, dtype: float64
```

### i Note

we discussed the ways to compare a model further, mostly reviewing and reinforcing what was covered in the last class

## 1. Assignment 1: Setup, Syllabus, and Review

Due: 2024-09-10 2:00pm

### ⚠ Warning

You must *complete* it by 2pm on Tuesday, but if you are confused on anything from the syllabus put `question: <your question here>`, replacing the `<>` part with our actual question in that section and then ask in class. There will be time in class to make revisions to your work before it is officially graded.

I will be reading everything before class (and I can use GitHub timestamps to see what was done before and later)

[Skip to main content](#)

## 1.1. Evaluation

Eligible skills:

- Python
- Process

## 1.2. Related notes

- Welcome & What is Data Science

## 1.3. Instructions

### ! Important

If you have trouble, check the GitHub FAQ on the left first

Your task is to:

1. Install required software from the Tools & Resource page (should have been done before the first class)
2. Create your portfolio, using the link on Brightspace
3. Learn about your portfolio from the README file on your repository.
4. Follow instructions in the README to make your portfolio your own with information about yourself(completeness only) and your own definition of data science (graded for **level 1 process**)
5. complete the `success.md` file as per the instructions in the comments in that file (it is a syllabus quiz)
6. Create a Jupyter notebook called `grading.ipynb` and write a function that computes a grade for this course, with the docstring below. Your notebook will need to follow the [course style guide](#), following style from other courses will not earn credit.
7. In the same notebook, iterate over the example anonymized gradebook below and compute a grade for each one.
8. Upload your `grading.ipynb` file to your portfolio (upload to main branch).

### ⚠ Warning

Do not merge your “Feedback” Pull Request

### 1.3.1. Docstring

...

Computes a grade for CSC/DSP310 from numbers of achievements at each level

Parameters:

[Skip to main content](#)

```
level2_acheivements : int
    number of level 2 achievements earned
level3_acheivements : int
    number of level 3 achievements earned

>Returns:
-----
letter_grade : string
    letter grade with possible modifier (+/-)
'''
```

## 1.3.2. Students to check

Treat this list of lists as a gradebook and use your function to compute a grade for each one and store them all to a list.

```
acheivements_only = [[15,14,1],[15,15,10],[12,12,12]]
```

## 1.4. Help

(optional, but useful information)

### 1.4.1. Sample tests

Here are some sample tests you could run to confirm that your function works correctly, they are in one block here, but you should run them one at a time and with text between:

```
assert compute_grade(15,15,15) == 'A'
assert compute_grade(15,15,13) == 'A-'
assert compute_grade(15,14,14) == 'B-'
assert compute_grade(14,14,14) == 'C-'
assert compute_grade(4,3,1) == 'D'
assert compute_grade(15,15,6) == 'B+'
```

### 1.4.2. Notebook Checklist

- a Markdown cell with a heading
- your function called `compute_grade`
- three calls to your function that verify it returns the correct value for different number of badges that produce at three different letter grades.
- markdown cells with explanation of the calls and anything else so that the notebook reads like a report.

## 2. Assignment 2: Practicing Python and Accessing Data

### 2.1. Deadline

2024-09-19 end of day

However, if you upload an attempt and specific questions on time and then attend office hours on 9/20 then any revisions made based on our conversation will count as on time

### 2.2. Submission and Template

Complete your work on the `assignment2` branch in your portfolio, a template file for `datasets.py` is located there, as well as the `data` folder with two files.

In your repo you can see a menu on the left hand side of the page, select `assignment2` from the list and work from there.

The screenshot shows a GitHub repository page for 'portfolio-fall24-brownsarahm'. The 'Code' tab is selected in the top navigation bar. A yellow banner at the top left indicates that the 'assignment2' branch has recent pushes 6 minutes ago. On the right, there's an 'About' section with details about the repository, including its name, description ('csc-dsp310-fall-24-portfolio-fall24-portfolio-template'), and it being created by GitHub Classroom. Below the banner, a message says 'Your main branch isn't protected' and provides options to 'Protect this branch' or 'View documentation'. The main content area shows a list of branches and tags. The 'main' branch is selected, showing recent commits like 'convert to md' and 'Hub Classroom Feedback'. Other branches listed include 'assignment2' and 'feedback'. The 'Tags' tab is also visible. On the right, sections for 'Releases' and 'Packages' are shown, both indicating no activity.

Skip to main content

## 2.3. Objective & Evaluation

This assignment is an opportunity to earn level 1 and 2 achievements in `python` and `access` and begin working toward level 1 for `summarize`. You can also earn level 1 for `process`.

Eligible skills: (links to checklists)

- process level 1
- python level 1 or 2
- access level 1
- summarize level 1

The goal of this assignment is for you to practice

1. finding datasets
2. getting familiar with different critieria we describe datasets with
3. practice loading data
4. practice getting basic facts about a dataset
5. manipulating dictionaries do do things efficiently
6. writing pythonic loops
7. creating your own DataFrame from other Python objects
8. saving a dataframe to csv (preview for cleaning)

These are all important skills that help you prepare to think in the right way to do more typical data analyses starting with assignment 3.

## 2.4. Related notes

- Iterables and Pandas Data Frames
- 

## 2.5. Store info about data for loading

Find 3 datasets of interest to you that are provided in at least two different file formats. Choose datasets that are not too big, so that they do not take more than a few second to load. At least one dataset, must have non numerical (eg string or boolean) data in at least 1 column.

### Important

So that we can use this in class next week, this part is due early

Create a list of dictionaries in file called `datasets.py`, so that there is one dictionary for each dataset. Each dictionary should

[Skip to main content](#)

Table 2.1 Meta data of the dictionaries

<code>url</code>	the full url of the dataset
<code>short_name</code>	a short name
<code>load_function</code>	(the actual function handle) what function should be used to load the data into a <code>pandas.DataFrame</code> .

### 💡 Hint

See below for how you will use the dictionary as help for how you should construct it

## 2.6. Analyze Your Datasets

Do the following in a notebook called `loading_data.ipynb`.

### 2.6.1. Document your data

In your notebook, create a markdown cell for each dataset that includes:

- heading 2 with the dataset's name
- a 1-2 sentence summary of what the dataset contains and why it was collected
- a "more info" link to where someone can learn about the dataset
- 2-3 questions you would like to answer with that dataset in a bulleted ( - ) or numbered list

### 💡 Hint

markdown links are written like: `[text to show](url/to/go/to)`

### 2.6.2. Make a dataset about your datasets

In a notebook called `dataset_of_datasets.ipynb`, import the list of dictionaries from the `datasets` module you created in the step above. Then iterate over the list of dictionaries, and:

1. load each dataset like `dataset_dict['load_function'](dataset_dict['url'])`
2. save it to a local csv using the short name you provided for the dataset as the file name, without writing the index column to the file.
3. record attributes about the dataset as in the table below in a list or dictionary of lists
4. Use that to [create a DataFrame](#) with columns that match the rows of the following table.

Table 2.2 Meta Data Description of the DataFrame to build

name	a short name for the dataset
source	a url to where you found the data
num_rows	number of rows in the dataset
num_columns	number of columns in the dataset
num_numerical	number of numerical variables in the dataset

### 2.6.3. Explore Your Datasets

Create a second notebook file called `exploration.ipynb`:

For one dataset that includes nonnumerical data:

- read it in from your local csv using a relative path
- display the heading and the first 4 rows
- make a numpy array of only the numerical data and save it to a new variable (select these programmatically)
- was the format that the data was provided in a good format? why or why not?

For any other dataset:

- read it in from your local csv using a relative path
- display the heading with the last 3 rows
- display the pandas datatype for each column
- Are there any variables where pandas may have read in the data as a datatype that's not what you expect (eg a numerical column mistaken for strings)? If so, investigate and try to figure out why.

For the third dataset:

- read it in from your local csv using a relative path
- save every third row (3,6,9,...) of the data for two columns of your choice into a new DataFrame and display that

### 2.6.4. Exploring data files

There are two files in the data folder, both can be read in with `read_csv` but need some options or fixing.

- try to read in the `german.data` file, what happens with the default settings? What option do you need to use to make it look right?
- try to read in the `.csv` file that's included in the template repository, use the error messages you get to try to fix the file manually (any text editor, including jupyter can edit a `.csv`), making notes about what changes you made in a markdown cell.

### 💡 Hint

For the csv file in the template's data folder, in Jupyter Lab, it will not let you edit a .csv file, but you can change the file name to txt (in your code too) and then it will work.

## 2.7. Tips and hints

### 2.7.1. the goal

- I am not looking for “an answer”
- You should not be either
- I am looking for evidence that you **understand** the material (thus far including prereqs)
- You should be trying to **understand** material
- Review all of the methods the `DataFrame` has

This means that in office hours, I am going to:

- ask you questions to help you think about the problem and the material
- help direct your attention to the right part of the error message to figure out what is wrong
- `dtype` shows you the type of each column [read more](#)
- there is also a `select dtypes` method

## 2.8. Thinking ahead

### ❗ Important

This section is not required, but is intended to help you get started thinking about how you could extend this assignment. If you complete it, we'll give your feedback to help shape your ideas to get to level 3 achievements. If you want to focus only on level 2 at this moment in time, feel free to skip this part. You could also think about these after submitting the assignment. If you want, you could discuss these ideas in office hours.

1. When might you prefer one datatype over another?
2. How does PEP 8 standard code help you be collaborative?
3. Learn about [Datasheets for Datasets](#) and find some examples, (eg this [google scholar result](#)) How could something like this impact your work as a data scientist?

## 3. Assignment 3: Exploratory Data Analysis

Due:2023-10-01 end of day

## 3.1. Submission

### ! Important

You have the option to work with a partner. You must plan this in advance so that you have access to collaborate.

### 3.1.1. Solo

Add your work to the assignment3 branch in your portfolio and you do not need to edit the `a3_location` file

### 3.1.2. Group

1. coordinate so that the first person makes the team when they accept the assignment
2. the second (and third) joins the same team when they accept the assignment.
3. Each person should upload their work to a branch named `d1`, `d2` or `d3` for which dataset checklist you followed from below and open a PR. (each person should do a different dataset)
4. In your portfolio, replace the contents of the `a3_location.md` file on the assignment3 branch with your team name. We will use that to create a PR to give you your individualized achievements update.

## 3.2. Objective & Evaluation

This week your goal is to do a small exploratory data analysis for two datasets (or one if in a group) of your choice.

Eligible skills: (links to checklists)

- process 1
- access 1 and 2
- summarize 1 and 2
- visualize 1

## 3.3. Related notes

- Exploratory Data Analysis
- Visualization

## 3.4. Choose Datasets

Each Dataset must have at least three variables, but can have more. Both datasets must have multiple types of variables. These **can** be datasets you used for Assignment 2, if they meet the criteria below. All datasets must be different datasets even in a group

### 3.4.1. Dataset 1 (d1)

must include at least:

- two continuous valued variables **and**
- one categorical variable.

#### 💡 Hint

a dataset from the UCI data repository that's for classification and has continuous features would work for this

### 3.4.2. Dataset 2 (d2)

must include at least:

- two categorical variables **and**
- one continuous valued variable

### 3.4.3. Dataset 3 (d3)

#### ⚠️ Warning

This is only for groups of 3

must include at least:

- two continuous valued variables **and**
- one categorical variable.

## 3.5. EDA

Use a separate notebook for each dataset, name them `dataset_0x.ipynb` where `x` is the number checklist you are following.

For **each** dataset, in the corresponding notebook complete the following:

1. Load the data to a notebook as a `DataFrame` from url or local path, if local, include the data file in your repository.
2. Write a short description of what the data contains and what it could be used for
3. Explore the dataset in a notebook enough to describe its structure. Use the heading `## Description` and include at least the following with interpretation. *What does the structure imply about the conclusions you can draw from this data? Are there limitations in how to safely interpret the data that the summary helps you see? are the variables what you expect?*

- columns
  - variable types
  - overall summary statistics
4. Ask and answer **at least** 3 questions by using and interpreting statistics and visualizations as appropriate. Include a heading for each question using a markdown cell and H2: `##`. **Make sure your analyses meet the criteria in the check lists below.** Use the checklists to think of what kinds of questions would use those type of analyses and help shape your questions. Your questions can be related or different levels of detail or views on a big picture question as long as the analysis addresses the checklist.
5. Describe what, if anything might need to be done to clean or prepare this data for further analysis in a finale `## Future analysis` markdown cell in your notebook.

### 3.5.1. (overall) Question checklist

be sure that every question (all six, 3 per dataset) has:

- a heading
- at least 1 statistic or plot
- interpretation that answers the question
- the question does not include the name of the statistic or plot in it

### 3.5.2. Dataset 1 Checklist

make sure that your `dataset_01.ipynb` has:

- Overall summary statistics grouped by a categorical variable
- A single statistic grouped by a categorical variable
- at least one plot that uses 3 total variables
- a plot and summary table that convey the same information. This can be one statistic or many.

### 3.5.3. Dataset 2 Checklist

make sure that your `dataset_02.ipynb` has:

- overall summary statistics
- two individual summary statistics for one variable
- one summary statistic grouped by two categorical variables
- a figure with a grid of subplots that correspond to two categorical variables

### 3.5.4. Dataset 3 Checklist

#### ⚠ Warning

This is only for groups of 2

[Skip to main content](#)

Hi  
Try  
the  
as i  
it lo  
earl

make sure that your `dataset_03.ipynb` has:

- overall summary statistics
- two individual summary statistics for one variable
- at least one plot that uses 3 total variables
- a plot and summary table that convey the same information. This can be one statistic or many.

## 3.6. Peer Review

### Note

If you work alone and complete 2 analyses you do not need to do this, but you might review these questions because they are similar to how we will grade.

With a partner (or group of 3 where person 1 reviews 2 work, 2 reviews 3, and 3 reviews 1) read your partner's notebook and complete a peer review on their pull request. You can do peer review when you have done most of your analysis, and explanation, even if some parts of the code do not work.

You will complete your review on a PR, by reviewing it. If you want a big picture overview on that, the [github PR review "course"](#) is a good place to go, it is designed to take <30 minutes.

1. start a review
2. (optional) Use [PR comments](#) to denote places that are confusing or if you see solutions to problems your classmate could not solve *this is hard on notebook files, so it is okay to skip*
3. Prepare to [submit your review](#)
4. Use the list of questions below for your summary review (copy the template into the box and fill in )

### Important

Your review should use the **template** for organization, but the **questions** guide what sorts of aspects to consider across the sections.

### 3.6.1. Review Questions

1. Describe overall how it was to read the analysis overall to read. Was it easy? hard? cohesive? jumpy?
2. How did the data summaries help prepare you to read the rest of the analysis? What do you think might be missing?
3. For each question, consider the following and write any tips for improvement
  1. Does the question make sense based on the data? How does it relate to the real world is there a reasonable audience? How could the question be improved
  2. How well do the statistics and plots match the question?
  3. Are the interpretations complete, clear, and consistent with the statistics and plots?
  4. What could be done to make the explanations more clear and complete?

### 3.6.1.1. Template

```
## Overall
<!-- Describe overall how it was to read the analysis overall to read. Was it easy? hard? cohesive? jump
## Intro
## Question 1
## Question 2
## Question 3
```

### 3.6.2. Response

Respond to the review on your notebook either with inline comments, replies, or by updating your analysis accordingly.

## 3.7. Tips and Hints

- Remember you can also use [masking](#) in your EDA even though we did not do any in class
- To ensure you understand the checklist you can [optionally](#) make an issue using the appropriate issue type from your repo and fill in what it should be to get early feedback that you are on track
- variable types are in the notes
- the [DataFrame API](#) reference shows all the methods (and more) grouped by high level concepts.

## 3.8. Think Ahead

This can be added to any or all of the datasets

```
## Thinking Ahead
1. How could you make more customized summary tables?
1. Could you use any of the variables in this dataset to add more variables that would make interesting v
1. Are there multiple ways to answer your big picture question (like different thresholding or subsets of
1. Could any cleaning improve your analysis?
```

## 4. Assignment 4: Cleaning Data

Due: 2023-10-03

Eligible skills:

- prepare 1
- access 2
- python 1 ?

[Skip to main content](#)

## 4.1. Submission

Work on your assignment4 branch. When you are done, [open a PR with:](#)

base: feedback, compare: assignment4

and request a review from @surbhir08

## 4.2. Related notes

- Cleaning Data - Structure
- Fixing Values

## 4.3. Check the Datasets you have worked with already

In the datasets you have used or come across but decided you could not work with in your past assignments identify at least one thing you could not do because the data was not in an appropriate format.

In a notebook file called `dataset_fix.ipynb` apply one fix and show one summary statistic or plot that was not possible before to show that it works.

Some examples:

- a column that was a list or dictionary
- missing values
- a column that was continuous, but more interesting as a categorical
- too many header rows
- a data set that was wide, but tall would be better for plotting

## 4.4. CS Degrees

See the notebook on your assignment4 branch and complete the instructions there.

## 4.5. Study Cleaned Datasets

### Tip

there is a dedicated section in the Data Sources page

Read example data cleaning notes or scripts. To do this find at least one dataset for which the messy version, clean version, and a script or notes about how it was cleaned are available, answer the following questions in a markdown file, named

`cleaning_notes.md` (some example datasets are on the datasets page and one is in the notes are added to the course)

[Skip to main content](#)

1. What are 3 common problems to look for in a dataset? Describe them with examples.
2. Using one of the examples you found of cleaned data, give an example of a question or context that would require making different choices for cleaning than were made. Include a bit about the data, what was done, the question, what would need to be done instead and justification.
3. Explain in your own words, with a concrete example, how domain expertise can help you when cleaning data. Use either a made up example or one that you read about.

#### Warning

Some of these examples have both the clean and messy data files and an R script to do the cleaning. You are not required to *know* R, but looking at their R cleaning script could give hints of what things they fixed or changed. You could also compare the clean and messy versions by looking at them with a tool of your choice.

## 5. Assignment 5: Constructing Datasets and Using Databases

due date : 2023-10-10

#### Note

I encourage you to get it done and get rest over the long weekend. However, the TA and I are also going to get rest this weekend so grading will not begin until Tuesday and we will grade any submissions in at that time

Skills:

- prepare level 1
- summarize 1,2
- visualize 1,2
- python 1,2

#### Warning

sumbit to the assignment5 branch, but there are no template files.

### 5.1. Related notes

- [Webscraping](#)
- [Merging Data & Databases](#)

### 5.2. Constructing Datasets

Your goal is to programmatically construct a ready to analyze dataset that combines information from multiple sources. This can be:

[Skip to main content](#)

- combining two tables with a merge.

If you use a merge to meet the multiple sources criterion, only one source must be scraped, the second can be provided as tabular data.

If you crawl, all the data could come from tables(files or [read\\_html](#)), as long as you pull the urls programmatically.

The notebook you submit should include:

- a motivating question for why your are building the dataset you are building
- code and description of how you built and prepared your dataset. For each step, describe what you're about to do, the code with output, interpretation that leads into the next step.
- exploratory data analysis that shows why you built the data and confirms that is prepared enough to analyze.
- also save your dataset to csv

For construct only, this can be very minimal EDA.

## 5.3. Additional achievements

To earn additional achievements, you must do more cleaning and/or exploratory data analysis.

### ! Important

Make sure everything is well explained

### 5.3.1. Prepare level 2

To earn level 2 for prepare, you must manipulate either a component table or the final dataset. Sample manipulations include:

- transform into a tidy format
- add a new column by computing from others
- handle NaN values by dropping or filling
- drop a column, row, or duplicates in another way
- change a continuous value to categorical (there is an added section in the notes on [quantizing](#) that we did not do in class, but should be easy to follow)

### 5.3.2. Summarize and Visualize level 2

To earn level 2 for summarize and/or visualize, include additional analyses after building the datasets.

Connect your EDA to questions, and demonstrate items from one of the checklists in A3.

### 5.3.3. Python Level 2

- Use pythonic loops and a list or dictionary OR
- use a list or dictionary comprehension

this can be in your cleanup or your EDA

#### Thinking Ahead

Compare the level 2 skill definitions to level 3, how could you extend and adapt what you've done to meet level 3?

#### Thinking Ahead

You could also demonstrate understanding of how merges work by converting a dataset that is provided as a single table with redundant information into a number of smaller tables in a database.

## 6. Assignment 6: Auditing Algorithms

Due: 2023-10-21

Eligible skills:

- evaluate level 1
- construct level 2
- summarize, 1,2
- visualise 1,2

### 6.1. Related notes

- ML Models and Auditing with AIF360

### 6.2. About the data

We have provided a copy of [reconstructed version](#) of the [Adult Dataset](#), which is a popular benchmark dataset for training machine learning models that comes from a [recent paper](#) about the risks of that dataset. The classic [Adult](#) dataset tries to predict if a person makes more or less than 50k.

Researchers reconstructed the Adult dataset with the actual value of the income. We trained models to predict

`income>=$10k` , `income>=$20k` , etc. We used three different learning algorithms, nicknamed 'LR', 'GPR', and 'RPR' for each target (`>10k` , `>20k` ,..., `>90k` ).

- `adult_models_only.csv` has the model's predictions
- `adult_reconstruction_bin.csv` has the data.

Both data files have a unique identifier column included.

### 💡 Hint

Why might the dataset have more samples in it than the model predictions one?

## 6.3. Complete an audit

Thoroughly audit two randomly selected models. If you load the `adult_models_only.csv` to `models_df` then the following will give you replicable, but random, two columns

```
import numpy as np
my_num = # pick a number
np.random.seed(my_num)
models_to_audit = np.random.choice( models_df.columns, 2)
```

In your audit, use at least three different performance metrics. Compare and contrast performance in those metrics across racial or gender groups.

Include easy to read tables with your performance metrics and interpretations of the model's overall performance and any disparities that could be understood by a general audience.

Which of the two would you think is better to deploy?

## 6.4. Extend your Audit

### ℹ️ Note

optional (for more Achievements or deeper understanding/more practice)

Use functions and loops to build a dataset about the performance of the different models so that you can answer the following questions:

1. Which model (target and learning algorithm) has the best accuracy?
2. Which target value has the least average disparity by race? by gender?
3. Which learning algorithm has the least average disparity by race? by gender?
4. Which model (target and learning algorithm) do you think is overall the best?

Table 6.1 Example table format

y	model	score	value	subset
=10k	LR	accuracy	.873	overall
=20k	RPR	false_pos_rate	.873	men

[Skip to main content](#)

This table is not real data, just headers with one example value to help illustrate what the column name means.

### 💡 Hint

This step you should make separate data frames and then merge them together for construct. If you don't need construct you can build it as one, for visualize you should use appropriate groupings

## 7. Assignment 7: Classification

accept the assignment

**Due:** 2024-10-28

Eligible skills:

- evaluate level 2
- classification level 1,2
- summarize, 1,2
- visualuze 1,2

### 7.1. Related notes

- Classification Models

### ❗ Important

There is a large extra section in the notes, that should be of use for this assignment.

You can use Gaussian Naive Bayes **or** a Decision tree for the assignment.

### 7.2. Dataset and EDA

Choose a dataset that is well suited for classification and that has *all numerical features*. If you want to use a dataset with nonnumerical features you will have to convert the categorical features to numerical with one hot encoding.

### 💡 Hint

Use the [UCI ML repository](#), it will let you filter data by the attributes of it you need.

1. Include a basic description of the data(what the features are)
2. Describe the classification task in your own words
3. Use EDA to determine if you expect the classification to get a high accuracy or not. What types of mistakes do you think will happen most (think about the confusion matrix)?

[Skip to main content](#)

4. Hypothesize which classifier from the notes will do better and why you think that. Does the data meet the assumptions of Naive Bayes? What is important about this classifier for this application?

**!** **Important**

You will get to reuse the above, and this dataset, for the clustering assignment *and* optionally one or both of A10 and A11.

## 7.3. Basic Classification

1. Fit your chosen classifier with the default parameters on 80% of the data
2. Inspect the model to answer the questions appropriate to your model.
  - Does this model make sense?
  - (if DT) Are there any leaves that are very small?
  - (if DT) Is this an interpretable number of levels?
  - (if GNB) do the parameters fit the data well? or do the paramters generate similar synthetic data (you can answer statistically only or with synthetic data & a plot)
3. Test it on 20% held out data and generate a classification report
4. Interpret the model and its performance in terms of the application in order to give a recommendation, “would you deploy this model” . Example questions to consider in your response include
  - do you think this model is good enough to use for real?
  - is this a model you would trust?
  - do you think that a more complex model should be used?
  - do you think that maybe this task cannot be done with machine learning?

**i** **Note**

You need to give a thorough answer to the deployment question and these bulleted questions will help you create a thorough response.

## 7.4. Exploring Problem Setups

**!** **Important**

Understanding the impact of test/train size is a part of classification and helps with evaluation. This exercise is *also* a chance at python level 2.

Do an experiment to compare test set size vs performance:

1. Use a loop to train a model on 10%, 30%, … , 90% of the data. Compute the training accuracy and test accuracy for each size training data. Create a DataFrame with columns ['train\_pct','n\_train\_samples','n\_test\_samples','train\_acc','test\_acc']

[Skip to main content](#)

2. Use EDA on this data frame to interpret the results of your experiment. How does training vs test size impact the model's performance? Does it impact training and test accuracy the same way?

### ⚠ Warning

Please make sure you complete the above and get feedback before you go on to the following, because if you attempt the following with an error in the above you may spend a lot of time on something that might not earn credit

## 8. Assignment 8: Clustering

Due: 2024-11-04

### 8.1. Evaluation

Eligible skills: (links to checklists)

- clustering 1
- evaution 1,2
- visualize 1,2
- summarize 1,2

for some of these you will need to add analysis that is not described in the instructions below, but is related to this and that skill

### 8.2. Related notes

- [Clustering](#)
- [Kmeans algorithm](#)

### 8.3. Instructions

Use the same dataset you used for assignment 7, unless there was a problem. If you skipped assignment 7, choose a dataset well suited for classification. See A7 for tips.

1. Describe what question you would be asking in applying clustering to this dataset. What does it mean if clustering does not work well?
2. How does this task compare to what the classification task on this dataset?
3. Apply Kmeans using the known, correct number of clusters,  $K$ .
4. Evaluate how well clustering worked on the data:
  - using a true clustering metric and
  - using visualization and
  - using a clustering metric that uses the ground truth labels
5. Include a discussion of your results that addresses the following:

[Skip to main content](#)

- what the metrics show
  - Does this clustering work better or worse than expected based on the classification performance (if you didn't complete assignment 7, also apply a classifier)
6. Repeat your analysis using a 2 different numbers (1 higher, one lower) of clusters:
- can you interpret the new clusters?
  - how do they relate to the original clusters? are they completely different, did one split?
  - is there a reasonable explanation for more or less clusters than there are classes in this dataset?

## 9. Assignment 9: Regression and Optimization

Due: 2024-11-15

### Important

Since this is posted late you can do regression and optimize together or you can do only the regression part of this assignment and focus on optimization for classification or clustering in A10 while **also** doing model comparison.

If you optimize a regression model other than LASSO, we will evaluate it for regresion level 3.

### 9.1. Assessment

Eligible skills: (links to checklists)

- process
- regression
- evaluate
- optimize

### 9.2. Related notes

- [Regression](#)
- [More regression](#)
- [Task Review and Cross Validation](#)

### 9.3. Instructions

Find a dataset suitable for regression. We recommend a dataset from the UCI repository. Remember that you can use the filters to choose a dataset that is well-suited for regression.

#### 9.3.1. Linear Regression Basics

the result.

[Skip to main content](#)

1. Include a basic description of the data(what the features are, units, size of dataset, etc)
  2. Write your own description of what the prediction task it, why regression is appropriate.
  3. Fit a linear model on all numerical features with 75% training data.
  4. Test it on 25% held out test data and measure the fit with two metrics and one plot
  5. Inspect the model to answer:
    - What to the coefficients tell you?
    - What to the residuals tell you?
  6. Repeat the split, train, and test steps 5 times.
    - Is the performance consistent enough you trust it?
  7. Interpret the model and its performance in terms of the application. Some questions you might want to answer in order to do this include:
    - do you think this model is good enough to use for real?
    - is this a model you would trust?
    - do you think that a more complex model should be used?
    - do you think that maybe this task cannot be done with machine learning?
1. Try fitting the model only on one feature. Justify your choice of feature based on the results above. Plot this result.

### 9.3.2. Optimize a more complex regression model

1. Choose a different regression model based on the structure of your data and the performance of the basic linear regression on both all features and a single feature. This could be LASSO, [DecisionTreeRegression](#), [SVR](#) regression (note that at these links there are examples so you can learn about them by looking at data they excel on)
2. Fit and score this model with the default parameters
3. Choose reasonable model parameter values for your parameter grid by assessing how well the model fit with the default values.
4. Use grid search to find the best performing model parameters.
5. Examine the best fit model, how different is it from the default
6. Score the best fit model on a held out test set.
7. Examine and interpret the cross validation results. How do they vary in terms of time? Is the performance meaningfully different or just a little?
8. Try varying the cross validation parameters (eg the number of folds and/or type of cross validation). Does this change your conclusions?

## 10. Assignment 11: Model Comparison

### 10.1. Quick Facts

- Due: 2024-11-22

## 10.2. Related notes

- Model Comparison
- Which model should we deploy?

## 10.3. Assessment

Eligible skills:

- compare
- optimize
- evaluate
- (one of) regression, classification, clustering
- summarize
- visualize

### 💡 Tip

The next assignment will focus on a categorical outcome, so if you did not complete A9, use regression for this assignment.

The next assignment can use either, but classification will be less for for A11, so clustering might be a good choice for this assignment if you have not earned that yet.

## 10.4. Instructions

### 💡 Tip

this is best for regression or classification, but if you use clustering use the `scoring` parameter to pass better metrics than the default of the score method.

### 💡 Hint

See the questions to consider in A9 for more detail on what to consider when optimizing each model.

If you are skipping submitting A9, follow those instructions for each model.

Choose a dataset, it can be appropriate for classification, regression, or clustering. Optimize at least two models for the same task and choose the appropriate metrics to compare the fit. Decide which model you would recommend based on a realistic setting for that dataset and include evidence justifying that choice. Summarize your findings with plots and tables as appropriate.

### Think Ahead

How would this decision making compare for a more complex model or in more realistic setting.

## 10.5. Collaborative Version

Use the [template repo](#) and join a team with your selected partner

First in either a notebook or a python file, split the data into train and test sets and save each to a separate csv.

In the shared repo, each student should use the training data to optimize one model, evaluating the result somewhat and explaining its general strengths and weaknesses. Then save the results of the optimization with at least 50 total parameter options to a csv and [export the best model](#)

In a third notebook, load both optimization datasets and models and then compare the two thoroughly.

## Deepening your knowledge

Completing the basic assignments that demonstrate you can apply what we cover in class to your own data, is enough for a B, to earn an A you need to extend your knowledge.

You have some different pathways to earning an A.

## Extending Assignments

Starting in week 3 it is recommended that you spend some time each week working on extensions to earn level 3 on the skills.

Use the feedback you get on assignments to inspire your extensions.

To submit these, submit the work to a separate `<assignment>extended` branch so for assignment 2 extension, submit to `assignment2extended`.

You should **not** extend every assignment since skills overlap and relate to one another.

I recommend making an issue that is a plan and asking for feedback before you work on extensions, so that you do not do too much extra work.

Some ideas:

- extend A5 to add in a database source or combine the data in new ways, plus do extended EDA to earn level 3 for: access, construct, prepare, visualize and summarize
- extend one of A7-9 to have experiments that more carefully analyze and compare different models at a task for one of the ml tasks (classification, regression, clustering) and visualize and summarize

# Deployment and Distribution

## ⚠️ Warning

this is currently a draft, will be updated by 10/24

Instead of earning all 15 level 3s you can earn any 10 plus the following:

- transform your portfolio to a publishable portfolio by making it a jupyter book (mostly set up already, just need to clean up)
- share a model and use a classmate's model with [huggingface](#) (making the model card would count for process and fitting it could count for one of the 3 tasks)

if you go this route, I recommend level 3 for:

- access (loading models from [huggingface](#) will count)
- python
- process (see [Process Level 3](#))
- summarize
- visualize
- evaluate
- one of: classification, regression, clustering
- optimize
- compare
- workflow

If your dataset of interest is hard to work with, images, or text, you might swap in representation instead of access

this means skipping level 3 for:

- prepare
- construct
- 2 of: classification, regression, clustering
- representation

# Formatting Tips

## ⚠️ Warning

This is all based on you having accepted the portfolio assignment on github and having a cloned copy of the template. If you are not enrolled or the initial assignment has not been issued, you can view [the template on GitHub](#)

- it uses myst markdown
- it will run and compile Jupyter notebooks

This page will cover a few basic tips.

## Managing Files and versions

### ! Important

This will be updated; new gitbut features change how it works

## Organization

The summary of for the `part` or whole submission, should match the skills to the chapters. Which prompt you're addressing is not important, the prompts are a *starting point* not the end goal of your portfolio.

## Data Files

Also note that for your portfolio to build, you will have to:

- include the data files in the repository and use a relative path OR
- load via url

using a full local path(eg that starts with `///file:`) **will not work** and will render your portfolio unreadable.

## Structure of plain markdown

Use a heading like this:

```
# Heading of page
## Heading 2
### Heading 3
```

in the file and it will appear in the sidebar.

You can also make text **italic** or **bold** with either `*asterics*` or `__underscores__` with `_one for italic_` or `**two for bold**` in either case

## File Naming

It is best practice to name files *without* spaces, underscores `_` or hyphens `-` are both good. Each `chapter` or file should have a descriptive file name (`with_no_spaces`) and descriptive title for it.

# Adding annotations with formatting or margin notes

You can either install `jupytext` and convert locally or upload /push a notebook to your repository and let GitHub convert. Then edit the .md file with a [text editor](#) of your choice. You can run by uploading if you don't have `jupytext` installed, or locally if you have installed `jupytext` or `jupyterbook`.

In your .md file use backticks to mark special content blocks

```
```{note}
Here is a note!
````
```

```
```{warning}
Here is a warning!
````
```

```
```{tip}
Here is a tip!
````
```

```
```{margin}
Here is a margin note!
````
```

For a complete list of options, see the [sphinx-book-theme](#) documentation.

## Links

Markdown syntax for links

```
[text to show](path/or/url)
```

## Configurations

Things like the menus and links at the top are controlled as settings, in [\\_config.yml](#). The following are some things that you might change in your configuration file.

## Show errors and continue

To show errors and continue running the rest, add the following to your configuration file:

```
# Execution settings
execute:
```

[Skip to main content](#)

# Using additional packages

You'll have to add any additional packages you use (beyond pandas and seaborn) to the `requirements.txt` file in your portfolio.

## Generic Extension Ideas

### Extension

If there were parts of your previous assignments that you thought were interesting and you want to work with those data more, you can. You need to do *more complex* analyses of them, but you can build off of what you already have done, especially for assignments 2, 3, and 5.

### Extend Assignment 7, 8, or 9

Assignments 7-9 help you think through what machine learning tasks are. Extend those ideas by adding additional experiments based on your own questions or the questions in your feedback.

### Learn a new model

Repeat what you did in 7, 8, or 9, with a different model.

### Alternatives to Extending Assignments for Level 3

These are other ways you can earn level 3 achievements besides adding onto previous assignments.

If your goal is, for example, a B+ (you need 5 level 3s) you could only do 1-2 skills per portfolio check (there are 4 checks).

### Tutorial

Write a notebook that explains a concept related to a skill with examples in a real dataset and with visuals or a toy dataset (minimal number of columns rows)

### Cheatsheet

Make a detailed reference with code outputs on a topic or a few topics.

### Blog post

Tip  
If you achieve  
completing  
work

- how do different ways of loading data compare
- describe best practices you've learned and show why they're good with examples

## Practice Problems and Solutions

Based on the level 3 rubric descriptions, write practice problems that build off of the lecture notes. Include solutions and descriptions for each. These can be open ended or multiple choice questions with plausible distractors. A plausible distractor is an incorrect answer that represents a way that you think someone could misunderstand.

For example if the question is  $37 + 15 = ?$ , MCQ with plausible distractors might be:

- 52 (correct)
- 412 (didn't carry the one, correctly:  $7+5 = 12$ ,  $3+1 = 4$ )
- 42 (dropped the one  $7+5 = 12$ , ones place is 2,  $3+1 = 4$ )
- 43 (carried one into wrong column,  $7 + 5 = 12$ ,  $1+2 = 3$ ,  $3+1 = 3$ )

## Long single analysis

Collect data from multiple sources, prepare each for analysis, and merge them together then do some exploratory data analysis. Describe each step, interpret all outputs, and put the analysis in context of the Data Science Process.

This would be one long notebook that covers many skills at once.

## Create datasets that fail

Create datasets that violate assumptions of a model we have learned. The `sklearn` data generators are a good place to start.

## Build a data set for Prediction

Build a dataset that works for prediction (classification, regression, or clustering) from other sources.

## Organize your knowledge

Develop some sort of visual aid that demonstrates how you understand some aspect(s) of data science working. Think of this as something that future students could use to help them learning, so assume prior knowledge topics covered earlier than the one you are demonstrating.

This could be a concept map, a table that shows how you've traced how something works or any other sort of conceptual tool that helps convey your understanding.

## Try alternative libraries/ tools

One option for workflow level 3 is to use other data science skills and reflect on how what we have learned so far helped you learn a new set of tool as an alternative way to do things.

## Try feature engineering or representation learning

Try different transformations and see how they impact how well a model performs. This could be using `sklearn.feature_extraction` tools or trying different types of neural network layers at the beginning.

## Process Level 3

### Process level 3

Process level 3 is a little different than most of the others. You may be able to work it into an analysis notebook, but likely, you'll need to do one of the following.

### Data Science Pipeline Comparisons

Find two different sources that describe the data science pipeline or lifecycle. Write a blog style post that discusses their differences and hypothesizes about why they may be different? Are they for different audiences? Is one domain specific? How do they emphasize different modeling tasks? Include a Recommendation for when you think each one is better

### Write a short story

Write a short story that explains the concepts of data science to demonstrate your understanding of process.

### Media Review

Watch/listen/read to an episode of a high quality<sup>[1]</sup> podcast or other type of media and write a blog style summary and review. Highlight what you learned and how it relates to topics covered in class.

Approved Media:

- [Pod of Asclepius, Fall Series: The Philosophy of Data Science](#)
- Chapter 1 & 2 of [Think like a Data Scientist](#) in particular, if you think these would be helpful to assign as reading or teach from at the beginning of the semester next year.
- [Algorithms of Oppression](#) (book)
- [Weapons of Math Destruction](#) (book)
- [Coded Bias](#) (film, available on netflix & PBS)

[1] approved Dr. Brown by creating a pull request to add it to the list on this page that is successfully merged. To create a PR, use the suggest an edit button at the top of this page.

## FAQ

This section will grow as questions are asked and new content is introduced to the site. You can submit questions:

- via e-mail to Dr. Brown (brownsarahm) or TA
- via Prismia.chat during class
- by creating an [issue](#)

## Syllabus and Grading FAQ

How much does assignment x, class participation, or a portfolio check weigh in my grade?

What time are assignments due?

Can I submit this assignment late if ...?

I don't understand my grade on this assignment

## Git and GitHub

I can't push to my repository, I get an error that updates were rejected

The content I added to my portfolio isn't in the pdf

My command line says I cannot use a password

My .ipynb file isn't showing in the staging area or didn't push

My portfolio won't compile

Help! I accidentally merged the Feedback Pull Request before my assignment was graded

[Skip to main content](#)

---

# Code Errors

## Key Error

<bound method

# Glossary

## **aggregate**

to combine data in some way, a function that can produce a customized summary table

[Skip to main content](#)

---

a function that's defined on the fly, typically to lighten syntax or return a function within a function. In python, they're defined with the `lambda` keyword.

## **BeautifulSoup**

a python library used to assist in web scraping, it pulls data from html and xml files that can be parsed in a variety of different ways using different methods.

## **conditional**

a logical control to do something, conditioned on something else, for example the `if`, `elif` `else`

## **corpus**

(NLP) a set of documents for analysis

## **DataFrame**

a data structure provided by pandas for tabular data in python.

## **data leakage**

data from the test set is used in training and then falsely improves the test performance

## **dictionary**

(data type) a mapping array that matches keys to values. (in NLP) all of the possible tokens a model knows

## **discriminative**

a model that describes the decision rule for labeling a sample as one class or another

## **document**

unit of text for analysis (one sample). Could be one sentence, one paragraph, or an article, depending on the goal

## **error bars**

typically vertical, but sometimes also horizontal lines on a point in a line graph or bar in bar chart that indicate the spread of the samples used to create that point or bar height

## **generative**

a model that describes the data and therefore can also be used to generate new data that looks like the training data.

## **gh**

GitHub's command line tools

## **git**

a version control tool; it's a fully open source and always free tool, that can be hosted by anyone or used without a host, locally only.

## **GitHub**

a hosting service for git repositories

## **hyperparameter**

parameters of the learning *algorithm* that are set by the user and possibly optimized over in an outer loop; in `sklearn` these are set when instantiating the estimator object

## index

(verb) to index into a data structure means to pick out specified items, for example index into a list or a index into a data frame. Indexing usually invovles square brackets `[]` (noun) the index of a dataframe is like a column, but it can be used to refer to the rows. It's the list of names for the rows.

## interpreter

the translator from human readable python code to something the computer can run. An interpreted language means you can work with python interactively

## iterate

To do the same thing to each item in an `iterable` data structure, typically, an iterable type. Iterating is usually described as iterate over some data structure and typically uses the `for` keyword

## iterable

any object in python that can return its members one at a time. The most common example is a list, but there are others.

## kernel

in the jupyter environment, the kernel is a language specific computational engine

## lambda

they keyword used to define an anonymous function; lambda functions are defined with a compact syntax `<name> = lambda <parameters>: <body>`

## numpy array

a type provided by numpy to represent matrices, used by `pd.DataFrame.values` doc and accessed by `pd.DataFrame.to_numpy` doc

## PEP 8

Python Enhancement Proposal 8, the Style Guide for Python Code.

## repository

a project folder with tracking information in it in the form of a .git file

## Series

a data structure provided by pandas for single columnar data with an index. Subsetting a Dataframe or applying a function to one will often produce a Series

## shape

of a dataframe, or matrix is the number of rows and columns.

## Split Apply Combine

a paradigm for splitting data into groups using a column, applying some function(aggregation, transformation, or filtration) to each piece and combinging in the individual pieces back together to a single table

## **stop words**

Words that do not convey important meaning, we don't need them (like a, the, an,). Note that this is context dependent. These words are removed when transforming text to numerical representation

## **suffix**

additional part of the name that gets added to end of a name in a merge operation

## **test accuracy**

percentage of predictions that the model predict correctly, based on held-out (previously unseen) test data

## **Tidy Data Format**

Tidy data is a database format that ensures data is easy to manipulate, model and visualize. The specific rules of Tidy Data are as follows: Each variable is a column, each row is an observation, and each observable unit is a table.

## **token**

a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing (typically a word, but more general)

## **TraceBack**

an error message in python that traces back from the line of code that had caused the exception back through all of the functions that called other functions to reach that line. This is sometimes called tracing back through the stack

## **training accuracy**

percentage of predictions that the model predict correctly, based on the training data

## **transpose**

swap the rows and columns of a matrix or dataframe

## **Web Scraping**

the process of extracting data from a website. In the context of this class, this is usually done using the python library beautiful soup and a html parser to retrieve specific data.

# References on Python

## Official Documentation

- [Python](#)
- [Pandas](#)
- [Matplotlib](#)
- [Seaborn](#)

# Key Resources

- [Course Text](#) this book roughly covers things that we cover in the course, but since things change quickly, we don't rely on it too closely
- [Real Python](#) this site includes high quality tutorials
- [Towards Data Science](#) this blog has some good tutorials, but old ones are not always updated, so always check the date and don't rely too much on posts more than 2 years old.

## Ram Token Opportunity

If you find other high quality, reliable sources that you want to share, you can earn ram tokens.

# Cheatsheet

Patterns and examples of how to use common tips in class

## How to use brackets

| symbol                 | use   |
|------------------------|---|
| [val]                  | indexing item val from an object; <code>val</code> is int for iterables, or any for mapping |
| [val : val2]           | slicing elements val to val2-1 from a listlike object                                       |
| [ item1, item2 ]       | creating a list consisting of <code>item1</code> and <code>item2</code>                     |
| (param)                | function calls  |
| (item1, item2)         | defining a tuple of <code>item1</code> and <code>item2</code>                               |
| {item1, item2}         | defining a set of <code>item1</code> and <code>item2</code>                                 |
| {key:val1, key2: val2} | defining a dictionary where key1 indexes to val2  |

# Axes

First build a small dataset that's just enough to display

```
data = [[1,0],[5,4],[1,4]]  
df = pd.DataFrame(data = data,  
                  columns = ['A','B'])  
  
df
```

[Skip to main content](#)

|   | A | B |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 5 | 4 |
| 2 | 1 | 4 |

This data frame is originally 3 rows, 2 columns. So summing across rows will give us a [Series](#) of length 3 (one per row) and long columns will give length 2, (one per column). Setting up our toy dataset to not be a square was important so that we can use it to check which way is which.

```
df.sum(axis=0)
```

```
A    7
B    8
dtype: int64
```

```
df.sum(axis=1)
```

```
0    1
1    9
2    5
dtype: int64
```

```
df.apply(sum, axis=0)
```

```
A    7
B    8
dtype: int64
```

```
df.apply(sum, axis=1)
```

```
0    1
1    9
2    5
dtype: int64
```

## Indexing

```
df['A'][1]
```

5

```
df.iloc[0][1]
```

```
0
```

## Data Sources

This page is a semi-curated source of datasets for use in assignments. The different sections have datasets that are good for different assignments.

### Best for loading directly into a notebook

- Tidy Tuesday inside the folder for each year there is a README file with list of the datasets. These are .csv files
- Json Datasets
- National Center for Education Statistics Digest 2019 These data tables are available for download as excel and visible on the page.
- Lots of wikipedia pages have tables in them.

### Cleaning Examples

- Messy Artists .csv file, that needs to be cleaned, containing data on artists
- Messy Wheels .csv file, that needs to be cleaned, containing data on various wheel attractions around the globe
- Clean Artists .csv file, already cleaned, containing data on artists
- Clean Wheels, .csv file, already cleaned, containing data on various wheel attractions around the globe
- Women's Rugby
- Web page metrics
- data cleaning with open refine on survey data this is a tutorial for cleaning data with another tool, but it demonstrates common problems with data well.
- data clearing for ecology this is a tutorial for cleaning data with another tool, but it demonstrates common problems with data well.
- us solar data
- NYT Data Preparation document
- Corporate Reputation Rankings

### General Sources

These may require some more work

- Stackoverflow Developer Survey This data comes with readme info all packaged together in a .zip. You'll need to unzip it first.

- Kaggle most Kaggle datasets will require you to download and unzip them first and then you can copy them into your repo folder.
- UCI Data Repository Machine Learning focused datasets, can filter by task
- A curated list of datasets by task It includes datasets for cleaning, visualization, machine learning, and “data analysis” which would align with EDA in this course.
- Hugging Face NLP Datasets lots of text datasets

## Datasets in many parts

- Makeup Shades
- Kenya Census
- Wealth and Income over time
- UN Votes
- Deforestation
- Survivor
- Billboard
- Caribou Tracking
- Video games from steam 2021 and from 2019
- BBC Rap Artists
- character psychometrics
- weather forecast accuracy

## Datasets with time

- Superbowl commercials

## Databases

- SQLite Databases

If you have others please share by creating a pull request or issue on this repo (from the GitHub logo at the top right, [suggest edit](#)).

## General Tips and Resources

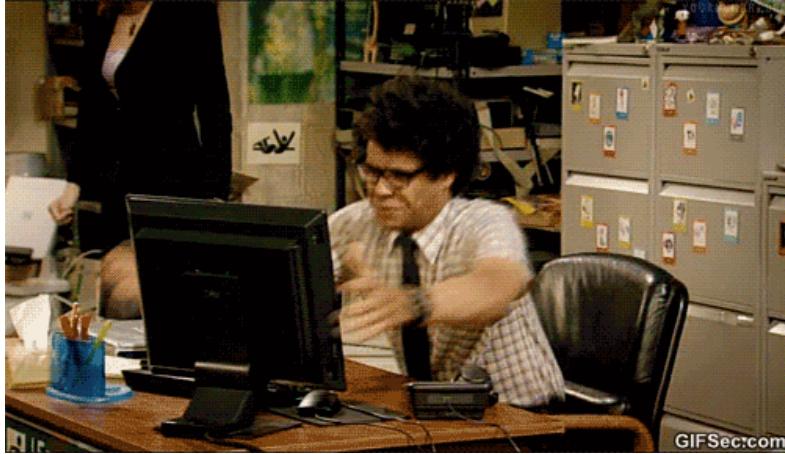
This section is for materials that are not specific to this course, but are likely useful. They are not generally required readings or installs, but are options or advice I provide frequently.

## on email

# How to Study in this class

This is a programming intensive course and it's about data science. This course is designed to help you learn how to program for data science and in the process build general skills in both programming and using data to understand the world. Learning two things at once is more complex. In this page, I break down how I expect learning to work for this class.

Remember the goal is to avoid this:



## Why this way?

Learning to program requires iterative practice. It does not require memorizing all of the specific commands, but instead learning the basic patterns.

Using reference materials frequently is a built in part of programming, most languages have built in help as a part of the language for this reason. This course is designed to have you not only learn the material, but also to build skill in learning to program. Following these guidelines will help you build habits to not only be successful in this class, but also in future programming.

### 🔔 Where are your help tools?

In Python and Jupyter notebooks, what help tools do you have?

## Learning in class

### ❗ Important

My goal is to use class time so that you can be successful with *minimal frustration* while working outside of class time.

Programming requires both practical skills and abstract concepts. During class time, we will cover the practical aspects and introduce the basic concepts. You will get to see the basic practical details and real examples of debugging during class sessions. Learning to debug something you've never encountered before and setting up your programming environment, for example, are *high frustration* activities, when you're learning, because you don't know what you don't know. On the other

something I'm confident that you can all be successful at with minimal frustration once you've seen basic ideas in class. My goal is that you can repeat the patterns and processes we use in class outside of class to complete assignments, while acknowledging that you will definitely have to look things up and read documentation outside of class.

Each class will open with some time to review what was covered in the last session before adding new material.

To get the most out of class sessions, you should have a laptop with you. During class you should be following along with Dr. Brown, typing and running the same code. You'll answer questions on Prismia chat, when you do so, you should try running necessary code to answer those questions. If you encounter errors, share them via prismia chat so that we can see and help you.

## After class

After class, you should practice with the concepts introduced.

This means reviewing the notes: both yours from class and the annotated notes posted to the course website.

When you review the notes, you should be adding comments on tricky aspects of the code and narrative text between code blocks in markdown cells. While you review your notes and the annotated course notes, you should also read the documentation for new modules, libraries, or functions introduced that day.

In the annotated notes, there will often be extra questions or ideas on how to extend and practice the concepts. Try these out.

If you find anything hard to understand or unclear, write it down to bring to class the next day.

## Assignments

In assignments, you will be asked to practice with specific concepts at an intermediate level. Assignments will apply the concepts from class with minimal extensions. You will probably need to use help functions and read documentation to complete assignments, but mostly to look up things you saw in class and make minor variations. Most of what you need for assignments will be in the class notes, which is another reason to read them after class.

## Portfolios

In portfolios, your goal is to extend and apply the concepts taught in class and practiced in assignments to solve more realistic problems. You may also reflect on your learning in order to demonstrate deep understanding. These will require significant reading beyond what we cover in class.

## Getting Help with Programming

## Screenshots

Sending me a screenshot is almost guaranteed to *not* get you help. Not because I do not want to, but because I literally do not have the information to get you an answer.

Typically when someone does not know how to fix something from the error message, it is because they are reading the wrong part of the error message or looking at the wrong part of the code trying to find the problem.

This means they end up screenshotting that wrong thing, so I literally **cannot** tell what is wrong from the screenshot.

I am not being stubborn, but I do need the right information to be able to tell what is wrong. Debugging code requires context, if you deprive me that, then I cannot help.

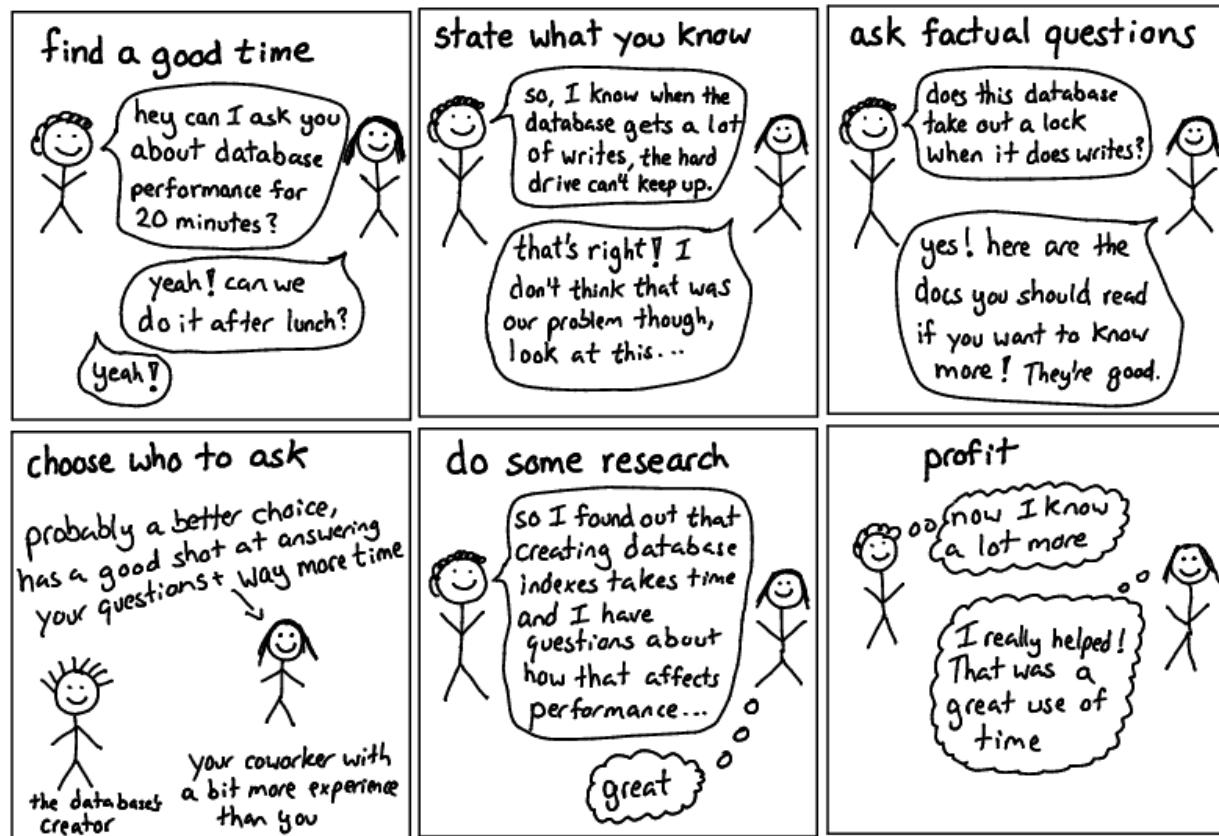
To get asynchronous help:

- upload your whole notebook, errors and all
- create an issue on that repo

## Asking Questions

JULIA EVANS  
@b0rk

### asking good questions



One of my favorite resources that describes how to ask good questions is [this blog post](#) by Julia Evans, a developer who writes comics about the things she learns in the course of her work and publisher of wizard zines.

## Describing what you have so far

Stackoverflow is a common place for programmers to post and answer questions.

[Skip to main content](#)

Creating a minimal reproducible example may even help you debug your own code, but if it does not, it will definitely make it easier for another person to understand what you have, what your goal is, and what's working.

## Understanding Errors

Error messages from the compiler are not always straight forward.

The [TraceBack](#) can be a really long list of errors that seem like they are not even from your code. It will trace back to all of the places that the error occurred. It is often about how you called the functions from a library, but the compiler cannot tell that.

To understand what the traceback is, how to read one, and common examples, see [this post on Real Python](#).

One thing to try, is [friendly traceback](#) a python package that is designed to make that error message text more clear and help you figure out what to do next.

### Ram Token Opportunity

If you try out friendly traceback and find it helpful, add a testimonial here. using

```
```{epigraph}
```

```

## Terminals and Environments

### Why all this work?

Managing environments is **one of the hardest parts of programming** so, as instructors, we often design our courses around not having to do it. In this class, however, I'm choosing to take the risk and help you all through beginning to manage your own environments.

These issues will be the most painful in the course, I promise.

I think it's worth this type of pain though, because all fo the code you ever run must run in *some* sort of environment. By giving you control, I'm hoping to increase your indepence as a programmer. This also means responsibility and some messy debugging, but I think this is a good tradeoff. This is an upper level (300+) level course, so increasing some complexity is expected and I want as much as possible to keep you close to realisitc programming environments; so that what you see in this course is **directly, and immediately**, applicable in real world contexts. You should be able to pick up data science side projects or an internship with ease after this course.

I know some of these things will be frustrating at times, but I want you to feel supported in that and know that your grade will not be blocked by you having environment issues, as long as you ask for help in a timely matter.

## Windows

[Skip to main content](#)

The least setup required involves using anaconda prompt and `conda` to manage your python environment and GitBash to work with git (and it can also do other bash related things).

Instead of managing two terminals, you may [configure your path in GitBash to make Anaconda work](#)

will be here  
behind a  
often.

## MacOS

MacOS has one terminal app, but it can run different shells.

On MacOS You may want to switch to bash (using the `bash` command or make it your default and [update bash](#).

## Getting Organized for class

The only **required** things are in the Tools section of the syllabus, but this organizational structure will help keep you on top of what is going on.

Your username will be appended to the end of the repository name for each of your assignments in class.

## File structure

I recommend the following organization structure for the course:

```
CSC310
|- notes
|- portfolio-username
|- 02-accessing-data-username
|- ...
```

This is one top level folder with all materials in it. A folder inside that for in class notes, and one folder per repository.

Please **do not** include all of your notes or your other assignments all inside your portfolio, it will make it harder to grade.

## Finding repositories on github

Each assignment repository will be created on GitHub with the `rhodyprog4ds` organization as the owner, not your personal account. Since your account is not the owner, they do not show on your profile.

Your assignment repositories are all private during the semester. At the end, you may take ownership of your portfolio[^pttrans] if you would like.

If you go to the main page of the [organization](#) you can search by your username (or the first few characters of it) and see only your repositories.

 **Warning**

Don't try to work on a repository that does not end in your username; those are the template repositories for the course and you don't have edit permission on them.