

# About this Book

## Contents

### Syllabus

- About
- Tools and Resources
- Data Science Achievements
- Grading
- Grading Policies
- Support
- General URI Policies
- Course Communications

### Notes

- 1. Welcome and Introduction
- 2. Syllabus and Python Review
- 3. Grading review, Pandas, and Iterables
- 4. Pandas and Indexing
- 5. Exploratory Data Analysis (EDA)
- 6. Visualization
- 7. Tidy Data and Reshaping Datasets
- 8. Reparing values
- 9. Merging and Databases
- 10. Web Scraping
- 11. Evaluating ML Algorithms
- 12. Auditing with AIF360
- 13. Intro to ML & Naive Bayes
- 14. Predictions with niave Bayes and Decision Trees
- 15. Clustering
- 16. Evaluating Clustering Solutions
- 17. Comparing Classification and Clustering Data
- 18. Regression Preview
- 19. Regression
- 20. ML Task Review and Cross Validation
- 21. SVM and Model Optimization
- 22. Model Comparison
- 23. Intro to NLP- representing text data

- 1. Assignment 1: Portfolio Setup, Data Science, and Python
- 2. Assignment 2: Practicing Python and Accessing Data
- 3. Assignment 3: Exploratory Data Analysis
- 4. Assignment 4:
- 5. Assignment 5: Constructing Datasets and Using Databases
- 6. Assignment 6: Auditing Algorithms
- 7. Assignment 7
- 8. Assignment 8: Clustering
- 9. Assignment 9: Linear Regression
- 10. Assignment 10: Tuning Model Parameters
- 11. Assignment 11: Model Comparison
- 12. Assignment 12: Fake News

## Portfolio

- Portfolio
- Formatting Tips
- Portfolio Check 1 Ideas
- Check 2 Ideas

## 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
- Letters to Future students

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

This class meets TTh 5-6:15pm in Tyler 108.

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

[Skip to main content](#)

 Tip

subscribe to that calendar in your favorite calendar application

## 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 machine learning, where you learn how machine learning algorithms work. In this course, we will start with 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 either way by the end of the semester.

## About this syllabus

This syllabus is a *living* document and accessible from BrightSpace, as a pdf for download directly online at [rhodyprog4ds.github.io/BrownFall20/syllabus](https://rhodyprog4ds.github.io/BrownFall20/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

[Skip to main content](#)

# About your instructor

Name: Dr. Sarah Brown Office hours: TBA via zoom, link in BrightSpace

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.

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 all 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

# 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 [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.
```

[Skip to main content](#)

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.
8. Use the `bash` command followed by the file name of the installer you just downloaded to start the installation.
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](#).

---

<sup>11</sup> Too long: didn't read.

# Data Science Achievements

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 TTh 5-6:15pm in Tyler 108. 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 page. Portfolio deadlines will be announced at least 2 weeks in advance.

1 Not

On the  
Bright  
link to  
choice  
(star)  
page  
becau  
integr  
calen

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

## Achievement Definitions

The table below describes how your participation, assignments, and portfolios 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 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 optimized	Optimize basic model parameters such as model order	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
<b>representation</b>	Choose representations and transform data	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
<b>workflow</b>	use industry standard data science tools and workflows to solve data science problems	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

## 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.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	# Assignments
keyword														
<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	1	1	1	2
<b>workflow</b>	0	0	0	0	0	0	0	0	0	1	1	1	1	4

## ⚠ Warning

**process** achievements are accumulated a little slower. Prior to portfolio check 1, only level 1 can be earned. Portfolio check 1 is the first chance to earn level 2 for process, then level 3 can be earned on portfolio check 2 or later.

## Portfolios and Skills

The objective of your portfolio submissions is to earn Level 3 achievements. The following table shows what Level 3 looks like for each skill and identifies which portfolio submissions you can earn that Level 3 in that skill.

keyword		Level 3	P1	P2	P3	P4
<b>python</b>	reliable, efficient, pythonic code that consistently adheres to pep8	1	1	0	1	
<b>process</b>	Compare different ways that data science can facilitate decision making	0	1	1	1	
<b>access</b>	access data from both common and uncommon formats and identify best practices for formats in different contexts	1	1	0	1	
<b>construct</b>	merge data that is not automatically aligned	1	1	0	1	
<b>summarize</b>	Compute and interpret various summary statistics of subsets of data	1	1	0	1	
<b>visualize</b>	generate complex plots with pandas and plotting libraries and customize with matplotlib or additional parameters	1	1	0	1	
<b>prepare</b>	apply data reshaping, cleaning, and filtering manipulations reliably and correctly by assessing data as received	1	1	0	1	
<b>evaluate</b>	Evaluate a model with multiple metrics and cross validation	0	1	1	1	
<b>classification</b>	fit and apply classification models and select appropriate classification models for different contexts	0	1	1	1	
<b>regression</b>	fit and explain regularized or nonlinear regression	0	1	1	1	
<b>clustering</b>	apply multiple clustering techniques, and interpret results	0	1	1	1	
<b>optimize</b>	Select optimal parameters based of mutiple quantitateve criteria and automate parameter tuning	0	0	1	1	
<b>compare</b>	Evaluate tradeoffs between different model comparison types	0	0	1	1	
<b>representation</b>	apply transformations in different contexts OR compare and contrast multiple representations a single type of data in terms of model performance	0	0	1	1	
<b>workflow</b>	Independently scope and solve realistic data science problems OR independently learn releted tools and describe strengths and weaknesses of common tools	0	0	1	1	

## Detailed Checklists

### python-level1

*python code that mostly runs, occasional pep8 adherrance*

- [ ] logical use of control structures
- [ ] callable functions

[Skip to main content](#)

- [ ] correct use of variables
- [ ] use of logical operators

## python-level2

*python code that reliably runs, frequent pep8 adherence*

- [ ] descriptive variable names
- [ ] pythonic loops
- [ ] efficient use of return vs side effects in functions
- [ ] correct, effective use of builtin python iterable types (lists & dictionaries)

## python-level3

*reliable, efficient, pythonic code that consistently adheres to pep8*

- [ ] pep8 adherant variable, file, class, and function names
- [ ] effective use of multi-paradigm abilities for efficiency gains
- [ ] easy to read code that adheres to readability over other rules

## process-level1

*Identify basic components of data science*

- [ ] identify component disciplines OR
- [ ] idenitfy phases

## process-level2

*Describe and define each stage of the data science process*

- [ ] correctly defines stages
- [ ] identifies stages in use
- [ ] describes general goals as well as a specific processes

## process-level3

*Compare different ways that data science can facilitate decision making*

- [ ] describes exceptions to process and iteration in process
- [ ] connects choices at one phase to impacts in other phases
- [ ] connects data science steps to real world decisions

## access-level1

- [ ] use at least one pandas `read_` function correctly
- [ ] name common types
- [ ] describe the structure of common types

## access-level2

*Load data for processing from the most common formats; Compare and contrast most common formats*

- [ ] load data from at least two of (.csv, .tsv, .dat, database, .json)
- [ ] describe advantages and disadvantages of most common types
- [ ] describe how most common types are different

## access-level3

*Access data from both common and uncommon formats and identify best practices for formats in different contexts*

- [ ] load data from at least 1 uncommon format
- [ ] describe when one format is better than another

## construct-level1

*Identify what should happen to merge datasets or when they can be merged*

- [ ] identify what the structure of a merged dataset should be (size, shape, columns)
- [ ] identify when datasets can or cannot be merged

## construct-level2

*Apply basic merges*

- [ ] use 3 different types of merges
- [ ] choose the right type of merge for realistic scenarios

## construct-level3

*Merge data that is not automatically aligned*

- [ ] manipulate data to make it mergable
- [ ] identify how to combine data from many sources to answer a question
- [ ] implement steps to combine data from multiple sources

## summarize-level1

*Describe the shape and structure of a dataset in basic terms*

- [ ] use attributes to produce a description of a dataset

## **summarize-level2**

*compute and interpret summary standard statistics of a whole dataset and grouped data*

- [ ] compute descriptive statistics on whole datasets
- [ ] apply individual statistics to datasets
- [ ] group data by a categorical variable for analysis
- [ ] apply split-apply-combine paradigm to analyze data
- [ ] interprete statistics on whole datasets
- [ ] interpret statistics on subsets of data

## **summarize-level3**

*Compute and interpret various summary statistics of subsets of data*

- [ ] produce custom aggregation tables to summarize datasets
- [ ] compute multivariate summary statistics by grouping
- [ ] compute custom cacluations on datasets

## **visualize-level1**

*identify plot types, generate basic plots from pandas*

- [ ] generate at least two types of plots with pandas
- [ ] identify plot types by name
- [ ] interpret basic information from plots

## **visualize-level2**

*generate multiple plot types with complete labeling with pandas and seaborn*

- [ ] generate at least 3 types of plots
- [ ] use correct, complete, legible labeling on plots
- [ ] plot using both pandas and seaborn
- [ ] interpret multiple types of plots to draw conclusions

## **visualize-level3**

*generate complex plots with pandas and plotting libraries and customize with matplotlib or additional parameters*

- [ ] use at least two libraries to plot
- [ ] generate figures with subplots
- [ ] customize the display of a plot to be publication ready
- [ ] interpret plot types and explain them for novices
- [ ] choose appopriate plot types to convey information

## **prepare-level1**

*identify if data is or is not ready for analysis, potential problems with data*

- [ ] identify problems in a dataset
- [ ] anticipate how potential data setups will interfere with analysis
- [ ] describe the structure of tidy data
- [ ] label data as tidy or not

## **prepare-level2**

*apply data reshaping, cleaning, and filtering as directed*

- [ ] reshape data to be analyzable as directed
- [ ] filter data as directed
- [ ] rename columns as directed
- [ ] rename values to make data more analyzable
- [ ] handle missing values in at least two ways
- [ ] transform data to tidy format

## **prepare-level3**

*apply data reshaping, cleaning, and filtering manipulations reliably and correctly by assessing data as received*

- [ ] identify issues in a dataset and correctly implement solutions
- [ ] convert variable representation by changing types
- [ ] change variable representation using one hot encoding

## **evaluate-level1**

*Explain basic performance metrics for different data science tasks*

- [ ] define at least two performance metrics
- [ ] describe how those metrics compare or compete

## **evaluate-level2**

*Apply and interpret basic model evaluation metrics to a held out test set*

- [ ] apply at least three performance metrics to models
- [ ] apply metrics to subsets of data
- [ ] apply disparity metrics
- [ ] interpret at least three metrics

## **evaluate-level3**

- [ ] explain cross validation
- [ ] explain importance of held out test and validation data
- [ ] describe why cross validation is important
- [ ] identify appropriate metrics for different types of modeling tasks
- [ ] use multiple metrics together to create a more complete description of a model's performance

## classification-level1

*identify and describe what classification is, apply pre-fit classification models*

- [ ] describe what classification is
- [ ] describe what a dataset must look like for classification
- [ ] identify applications of classification in the real world
- [ ] describe set up for a classification problem (test,train)

## classification-level2

*fit, apply, and interpret preselected classification model to a dataset*

- [ ] split data for training and testing
- [ ] fit a classification model
- [ ] apply a classification model to obtain predictions
- [ ] interpret the predictions of a classification model
- [ ] examine parameters of at least one fit classifier to explain how the prediction is made
- [ ] differentiate between model fitting and generating predictions
- [ ] evaluate how model parameters impact model performance

## classification-level3

*fit and apply classification models and select appropriate classification models for different contexts*

- [ ] choose appropriate classifiers based on application context
- [ ] explain how at least 3 different classifiers make predictions
- [ ] evaluate how model parameters impact model performance and justify choices when tradeoffs are necessary

## regression-level1

*identify what data that can be used for regression looks like*

- [ ] identify data that is/not appropriate for regression
- [ ] describe univariate linear regression
- [ ] identify applications of regression in the real world

## regression-level2

*fit and interpret linear regression models*

- [ ] split data for training and testing
- [ ] fit univariate linear regression models
- [ ] interpret linear regression models
- [ ] fit multivariate linear regression models

## **regression-level3**

*fit and explain regularized or nonlinear regression*

- [ ] fit nonlinear or regularized regression models
- [ ] interpret and explain nonlinear or regularized regression models

## **clustering-level1**

*describe what clustering is*

- [ ] differentiate clustering from classification and regression
- [ ] identify applications of clustering in the real world

## **clustering-level2**

*apply basic clustering*

- [ ] fit Kmeans
- [ ] interpret kmeans
- [ ] evaluate clustering models

## **clustering-level3**

*apply multiple clustering techniques, and interpret results*

- [ ] apply at least two clustering techniques
- [ ] explain the differences between two clustering models

## **optimize-level1**

*Identify when model parameters need to be optimized*

- [ ] identify when parameters might impact model performance

## **optimize-level2**

*Optimize basic model parameters such as model order*

- [ ] evaluate potential tradeoffs
- [ ] interpret optimization results in context

## optimize-level3

*Select optimal parameters based of mutiple quanttiateve criteria and automate parameter tuning*

- [ ] optimize models based on multiple metrics
- [ ] describe when one model vs another is most appropriate

## compare-level1

*Qualitatively compare model classes*

- [ ] compare models within the same task on complexity

## compare-level2

*Compare model classes in specific terms and fit models in terms of traditional model performance metrics*

- [ ] compare models in multiple terms
- [ ] interpret cross model comparisons in context

## compare-level3

*Evaluate tradeoffs between different model comparison types*

- [ ] compare models on multiple criteria
- [ ] compare optimized models
- [ ] jointly interpret optimization result and compare models
- [ ] compare models on quanttiateve and qualitative measures

## representation-level1

*Identify options for representing text and categorical data in many contexts*

- [ ] describe the basic goals for changing the representation of data

## representation-level2

*Apply at least one representation to transform unstructured or inappropriately data for model fitting or summarizing*

- [ ] transform text or image data for use with ML

## representation-level3

*apply transformations in different contexts OR compare and contrast multiple representations a single type of data in terms of model performance*

- [ ] transform both text and image data for use in ml
- [ ] evaluate the impact of representation on model performance

## **workflow-level1**

*Solve well strucutred fully specified problems with a single tool pipeline*

- [ ] pseudocode out the steps to answer basic data science questions

## **workflow-level2**

*Solve well-structured, open-ended problems, apply common structure to learn new features of standard tools*

- [ ] plan and execute answering real questions to an open ended question
- [ ] describe the necessary steps and tools

## **workflow-level3**

*Independently scope and solve realistic data science problems OR independently learn releted tools and describe strengths and weaknesses of common tools*

- [ ] scope and solve realistic data science problems
- [ ] compare different data science tool stacks

# **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 problems as instructed by, for example, a supervisor in an internship or entry level job. This is a very accessible goal, it does

- 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

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. 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. You will have three *types* of opportunities to demonstrate your current skill level: participation, assignments, and a portfolio.

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 earn level 1 achievements in class, assignments, or portfolio submissions.
- 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. You can earn level 2 achievements in assignments or portfolio submissions.
- 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. You can earn level 3 achievements only through your portfolio submissions.

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

## 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 can earn level 1 achievements. In class questions will be administered through the classroom chat platform Prismia.chat; these records will be used to update your skill progression. You can also earn level 1 achievements from adding annotation to a section of the class notes.

## Assignments

For your learning to progress and earn level 2 achievements, you must practice with the skills outside of class time.

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.



If you accept then with a make need

## Portfolio Checks

To earn level 3 achievements, you will build a portfolio consisting of reflections, challenge problems, and longer analyses over the course of the semester. You will submit your portfolio for review 4 times. The first two will cover the skills taught up until 1 week before the submission deadline.

The third and fourth portfolio checks will cover all of the skills. The fourth will be due during finals. This means that, if you have achieved mastery of all of the skills by the 3rd portfolio check, you do not need to submit the fourth one.

Portfolio prompts will be given throughout the class, some will be structured questions, others may be questions that arise in class, for which there is not time to answer.

## TLDR

You *could* earn a C through in class participation alone, if you make nearly zero mistakes. To earn a B, you must complete assignments and participate in class. To earn an A you must participate, complete assignments, and build a portfolio.

## Detailed mechanics

On Brightspace there are 45 Grade items that you will get a 0 or a 1 grade for. These will be revealed, so that you can view them as you have an opportunity to demonstrate each one. The table below shows the minimum number of skills at each level to earn each letter grade.

letter grade	Level 3	Level 2	Level 1
<b>A</b>	15	15	15
<b>A-</b>	10	15	15
<b>B+</b>	5	15	15
<b>B</b>	0	15	15
<b>B-</b>	0	10	15
<b>C+</b>	0	5	15
<b>C</b>	0	0	15
<b>C-</b>	0	0	10
<b>D+</b>	0	0	5
<b>D</b>	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.



In this achieve because

The letter grade can be computed as follows

### ! Important

this will be revealed after assignment 1

For example you can run the code like this in a cell to see the output

```
compute_grade(15, 15, 15)
```

```
'A'
```

```
compute_grade(14, 14, 14)
```

```
'C-'
```

Or use `assert` to test it formally

```
assert compute_grade(14, 14, 14) == 'C-'
```

```
assert compute_grade(15, 15, 15) == 'A'
```

```
assert compute_grade(15, 15, 11) == 'A-'
```

## Late work

Late assignments will not be graded. Every skill will be assessed through more than one assignment, so missing assignments occasionally not necessarily hurt 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 will not be able to earn the level 3 achievement in that skill. If you submit work that is not complete, however, it will be assessed and receive feedback. Submitting pseudocode or code with errors and comments about what you have tried could earn a level 1 achievement. Additionally, most 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.

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.

In this issue, include:

1. A new deadline proposal
2. What additional work you plan to add
3. Why the extension is important to your learning



You n  
assig  
comp  
check  
they v  
demo

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.

## Grading Examples

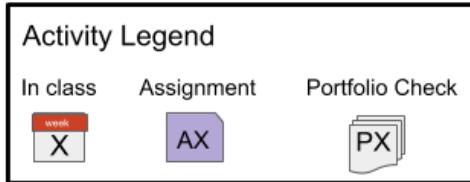
If you always attend and get everything correct, you will earn an A and you won't need to submit the 4th portfolio check.

### Getting an A Without Perfection

#### 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



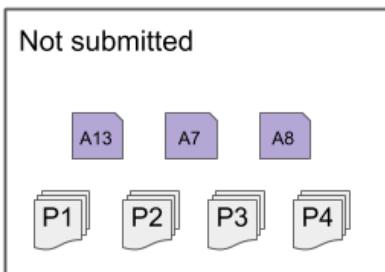
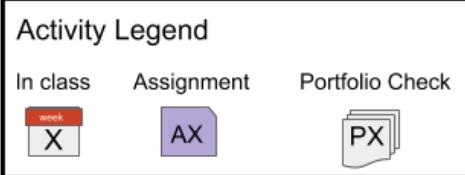
#### Other Activities

- |  |                                    |
|--|------------------------------------|
|  | 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.

# Map to a B easily

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

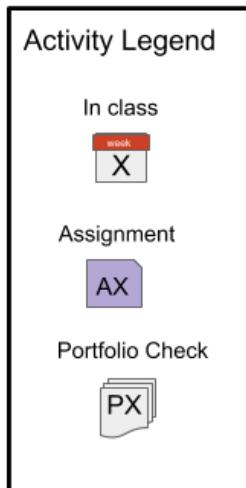


In this example, the student earned all level 1 achievements in class and all level 2 on assignments. This student was content with getting a B and chose to not submit a portfolio.

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.

## Academic Dishonesty

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 two for those skills requires meeting the standard for the level 3.

For example, if you are caught violating academic honesty 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 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 of those skills becomes ineligible.

## Grading Policies

### Late Work

Late assignments will not be graded. Every skill will be assessed through more than one assignment, so missing assignments

2 achievement in that skill through a portfolio check, but you will not be able to earn the level 3 achievement in that skill. If you submit work that is not complete, however, it will be assessed and receive feedback. Submitting pseudocode or code with errors and comments about what you have tried could earn a level 1 achievement. Additionally, most 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.

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.

In this issue, include:

1. A new deadline proposal
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 assignment on time.

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.

## Regrading

Re-request a review on your Feedback Pull request.

For general questions, post on the conversation tab of your Feedback PR with your request.

For specific questions, reply to a specific comment.

If you think we missed *where* you did something, add a comment on that line (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

## 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

- **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. Appointments and locations will be visible in the TutorTrac system on September 14th, 2020. 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-minute, 1 to 1 appointments that students can schedule on Starfish with Dr. David Hayes to address individual academic 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.

### 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 regarded as assurance that the work is the result of the student's own independent thought and study. Work 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
- Facilitating or aiding another's academic dishonesty
- Submitting the same paper for more than one course without prior approval from the instructors

## URI COVID-19 Statement

The University is committed to delivering its educational mission while protecting the health and safety of our community. While the university has worked to create a healthy learning environment for all, it is up to all of us to ensure our campus stays that way.

As members of the URI community, students are required to comply with standards of conduct and take precautions to keep themselves and others safe. Visit [web.uri.edu/coronavirus/](http://web.uri.edu/coronavirus/) for the latest information about the URI COVID-19 response.

- **Universal indoor masking** is required by all community members, on all campuses, regardless of vaccination status. If the universal mask mandate is discontinued during the semester, students who have an approved exemption and are not fully vaccinated will need to continue to wear a mask indoors and maintain physical distance.
- Students who are experiencing symptoms of illness should not come to class. Please stay in your home/room and notify URI Health Services via phone at 401-874-2246.
- If you are already on campus and start to feel ill, go home/back to your room and self-isolate. Notify URI Health Services via phone immediately at 401-874-2246.

If you are unable to attend class, please notify me at [brownsarahm@uri.edu](mailto:brownsarahm@uri.edu). We will work together to ensure that course instruction and work is completed for the semester.

## Course Communications

### 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 notification from the [notification settings page](#)

# Help Hours

Day	Time	Location	Host
Mon	11am-1pm	Tyler 139 and zoom	Kyle
Wed	7-8:30pm	Zoom	Dr. Brown
Fri	3-6pm	Zoom	Kyle

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

## To reach out, By usage

```
-----
TypeError                                         Traceback (most recent call last)
cell In[3], line 2
    1 df = df[['usage','platform','area','note']]
--> 2 display(HTML(df.style.hide()))

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/IPython/core/display.py:430, in HTML
    427     suffix = data[-10:].lower()
    428     return prefix.startswith("<iframe ") and suffix.endswith("</iframe>")
--> 430 if warn():
    431     warnings.warn("Consider using IPython.display.IFrame instead")
    432 super(HTML, self).__init__(data=data, url=url, filename=filename, metadata=metadata)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/IPython/core/display.py:426, in HTML
    420     return False
    422 #
    423 # Avoid calling lower() on the entire data, because it could be a
    424 # long string and we're only interested in its beginning and end.
    425 #
--> 426 prefix = data[:10].lower()
    427 suffix = data[-10:].lower()
    428 return prefix.startswith("<iframe ") and suffix.endswith("</iframe>")

TypeError: 'Styler' object is not subscriptable
```

### 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.

## By Platform

# Use e-mail for

```
-----  
AttributeError                                 Traceback (most recent call last)  
cell In[4], line 3  
  1 for platform, data in df.groupby('platform'):  
  2     display(HTML('<h3> Use '+ platform + ' for </h3>'))  
--> 3     display(HTML(data.drop(columns='platform').style.hide_index()._repr_html_()))  
  
AttributeError: 'Styler' object has no attribute 'hide_index'
```

## 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.

Not

What  
not m

## 1. Welcome and Introduction

### 1.1. Prismia Chat

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

- instructor only

[Skip to main content](#)

- share responses for all

## 1.2. How this class will work

### Participatory Live Coding

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

Debugging is both technical and a soft skill

## 1.3. 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.

#### Note

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 programming that you may have done.

## 1.4. Jupyter Notebooks

Launch a `jupyter notebook` server:

- on Windows, use anaconda terminal
- on Mac/Linux, use terminal

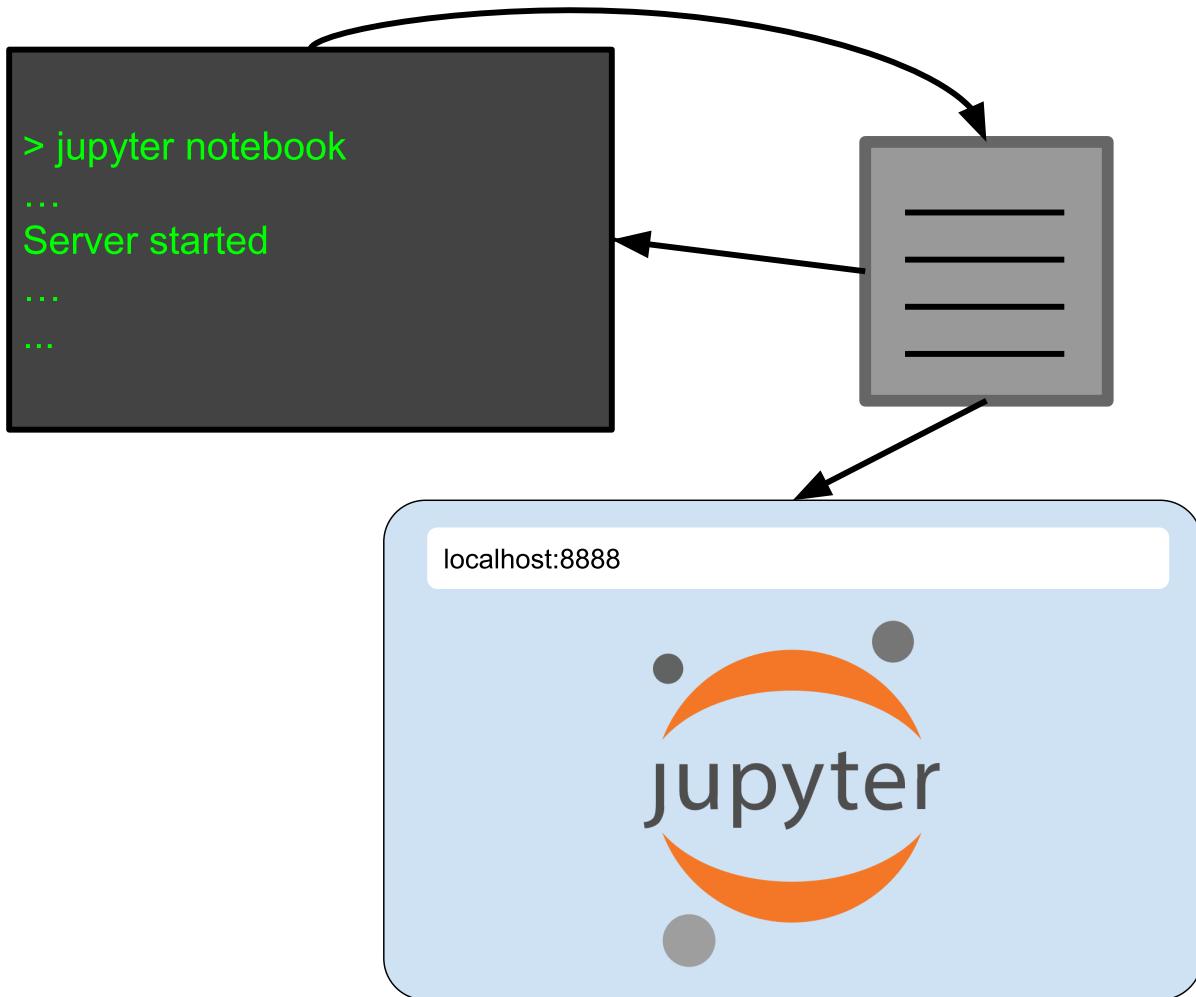
```
cd path/to/where/you/save/notes  
jupyter notebook
```

### 1.4.1. What just happened?

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

#### Warn

Some notes due to that c



#### 1.4.2. Start a Notebook

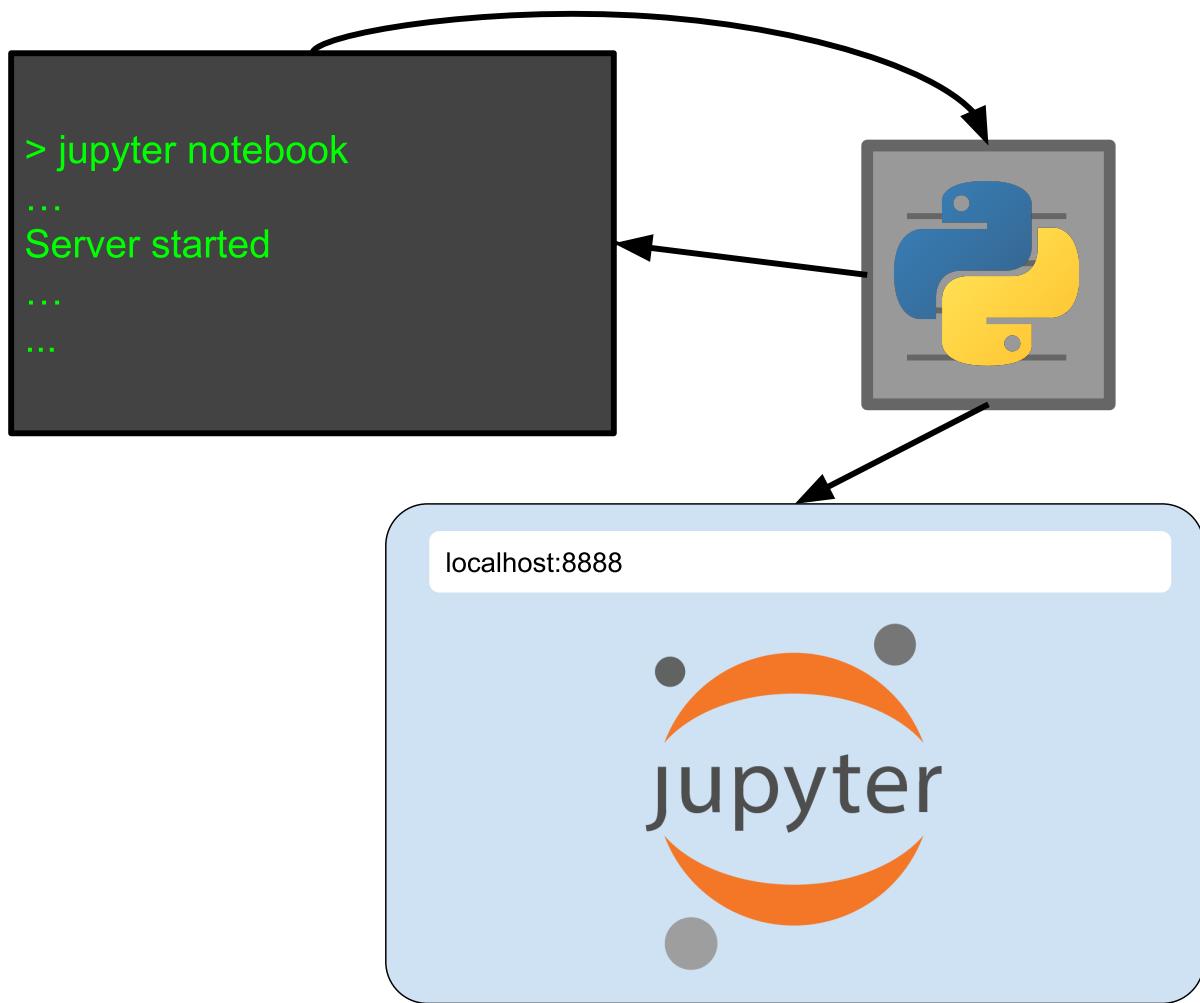
Go to the new menu in the top right and choose Python 3

The screenshot shows the Jupyter Notebook interface with the following elements:

- Header:** Jupyter logo, Quit, Logout buttons.
- Navigation:** Files, Running (selected), Clusters tabs.
- Message:** Select items to perform actions on them.
- Sidebar:** An empty list of notebooks with the message "The notebook list is empty."
- Actions:** Upload, New, Refresh buttons.
- Dropdown Menu:** Notebook section set to "Python 3". Other options include Text File, Folder, and Terminal.

[Skip to main content](#)

Now, it starts a python kernel on the webserver

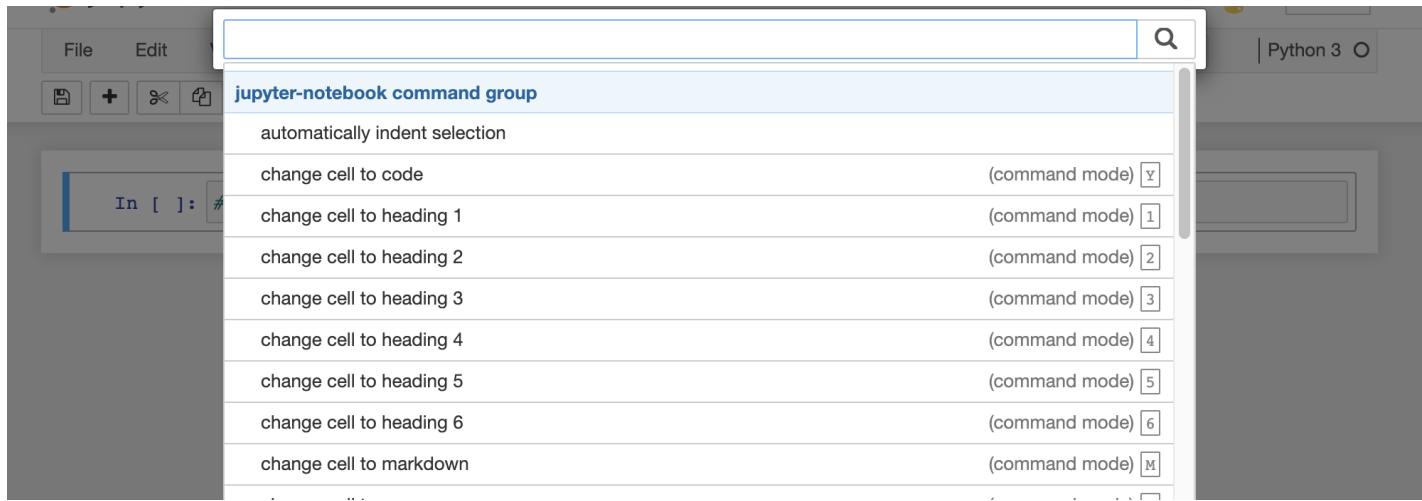


#### 1.4.3. A jupyter notebook tour

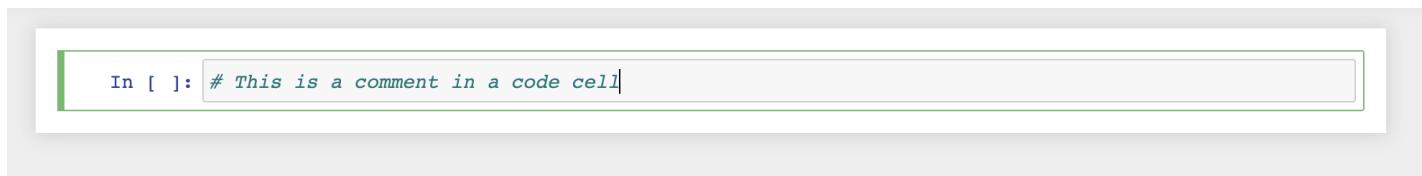
A Jupyter notebook has two modes. When you first open, it is in command mode. The border is blue in command mode.

A screenshot of a Jupyter notebook cell in command mode. The cell has a blue header bar. The text "In [ ]: # This is a comment in a code cell" is visible in the cell area.

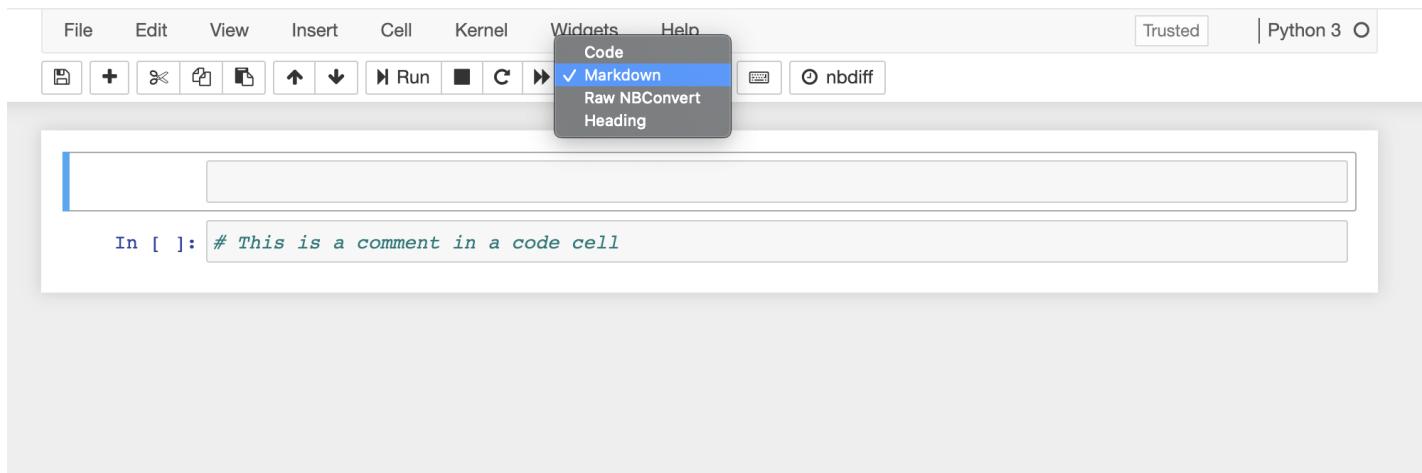
When you press a key in command mode it works like a shortcut. For example `p` shows the command search menu.



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. The border is green in 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

[Skip to main content](#)

```
# this is a comment in a code cell , we can run this to see output  
3+4
```

7

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

We set variables

```
name = 'sarah'
```

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

```
name
```

```
'sarah'
```

we can put a variable there to see it

```
course = 'csc310'  
course  
name
```

```
'sarah'
```

Note that this version doesn't show use the value for `course`

```
name = 'Sarah'
```

### ! Important

In class, we ran these cells out of order and noticed how the value does not update unless we run the new version

```
name
```

```
'Sarah'
```

```
course
```

```
'csc310'
```

Blue border is command mode, green border is edit mode

use Escape to get to command mode

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

In code cells, we can use a python interpreter, for example as a calculator.

```
4+6
```

```
10
```

It prints out the last line of code that it ran, even though it executes all of them

```
name = 'sarah'  
4+5  
name *3
```

```
'sarahsarahsarah'
```

## 1.5. 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.

We can print the docstring out, as a whole instead of using the shft + tab to view it.

```
help(print)
```

```
Help on built-in function print in module builtins:
```

```
print(...)  
    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 `value, ..., sep`

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.

```
print(name)
```

```
sarah
```

How do you use the `print` function to output: `Sarah_csc310`?

```
print(name, course, sep='_')
```

```
sarah_csc310
```

```
help(print)
```

```
Help on built-in function print in module builtins:
```

```
print(...)  
    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.
```

We can put as many values as we want there. Thats what the `...` in the function signature means

```
print(name, course, 'hello', 'bye', sep='_')
```

```
sarah_csc310_hello_bye
```

```
print(name, course, 'hello', 'bye', sep='\n')
```

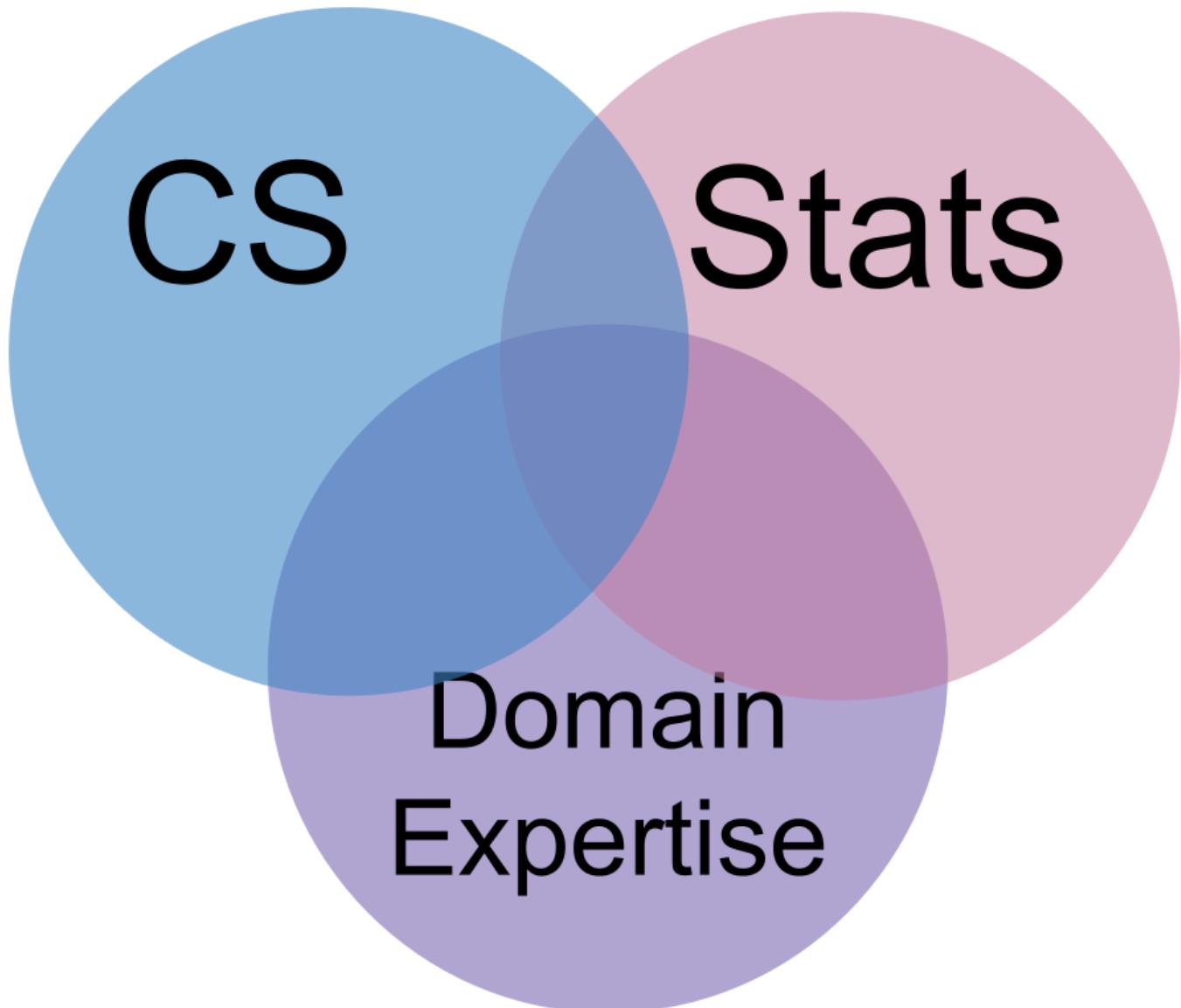
```
sarah  
csc310  
hello  
bye
```

### ⚠ Important

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

## 1.6. What is Data Science?

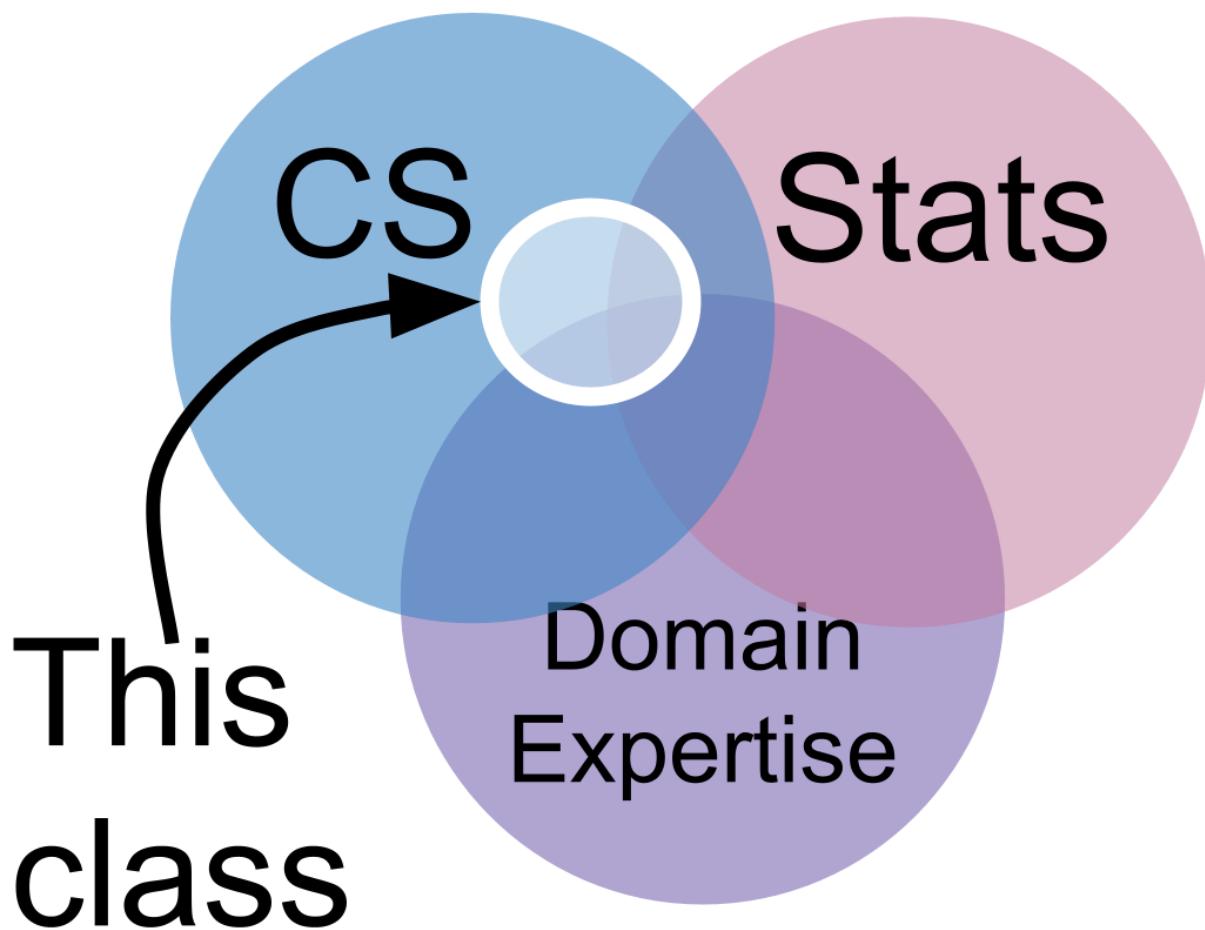
Data Science is the combination of



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

**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.

### 1.6.1. In this class,

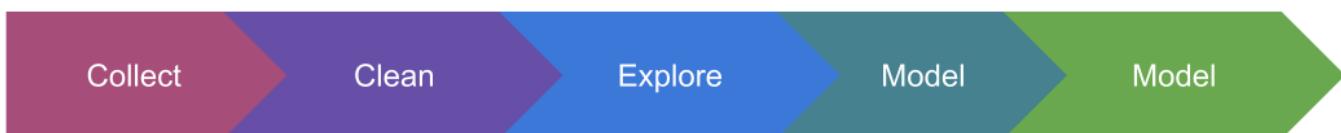


We'll focus on the programming as our main means of studying data science, but we will use bits of the other parts. In particular, you're encouraged to choose datasets that you have domain expertise about, or that you want to learn about.

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

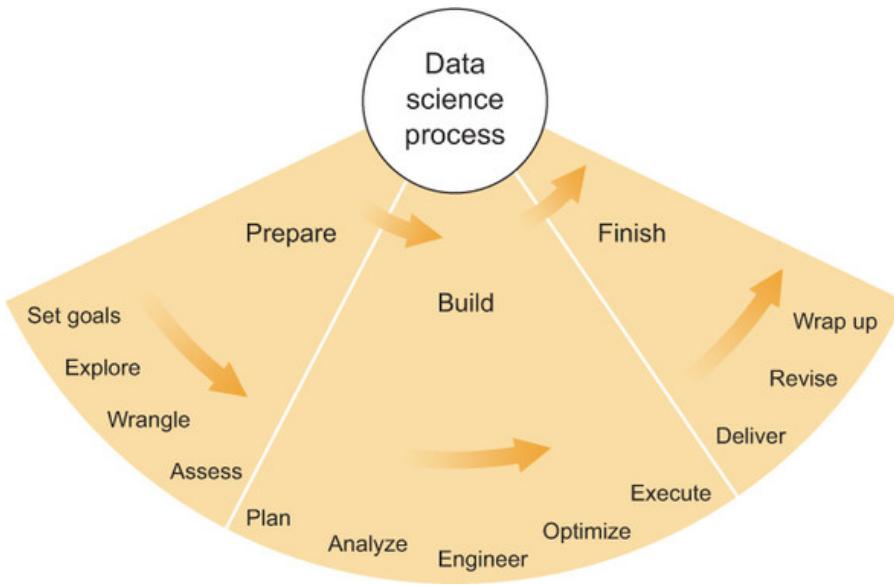
### 1.6.2. How does data science happen?

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.

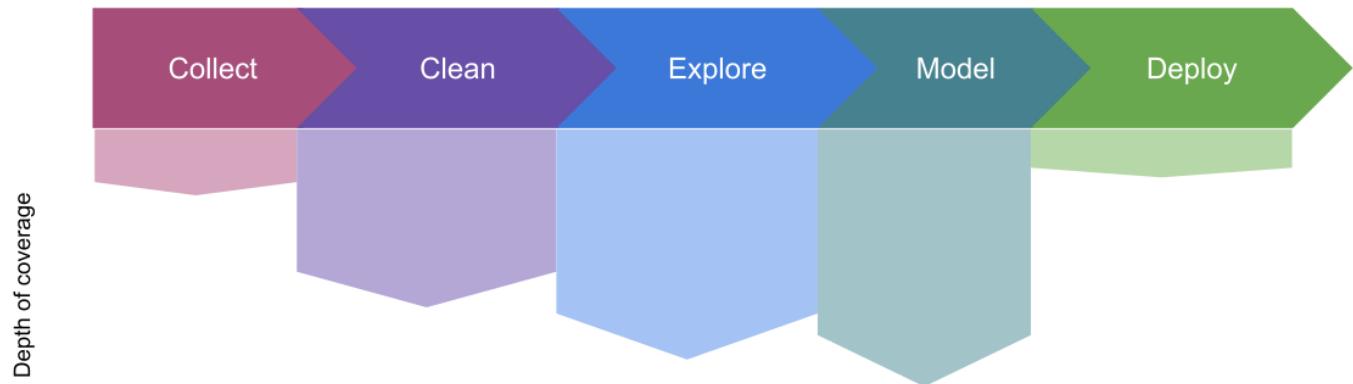


[Skip to main content](#)

Another way to think about it

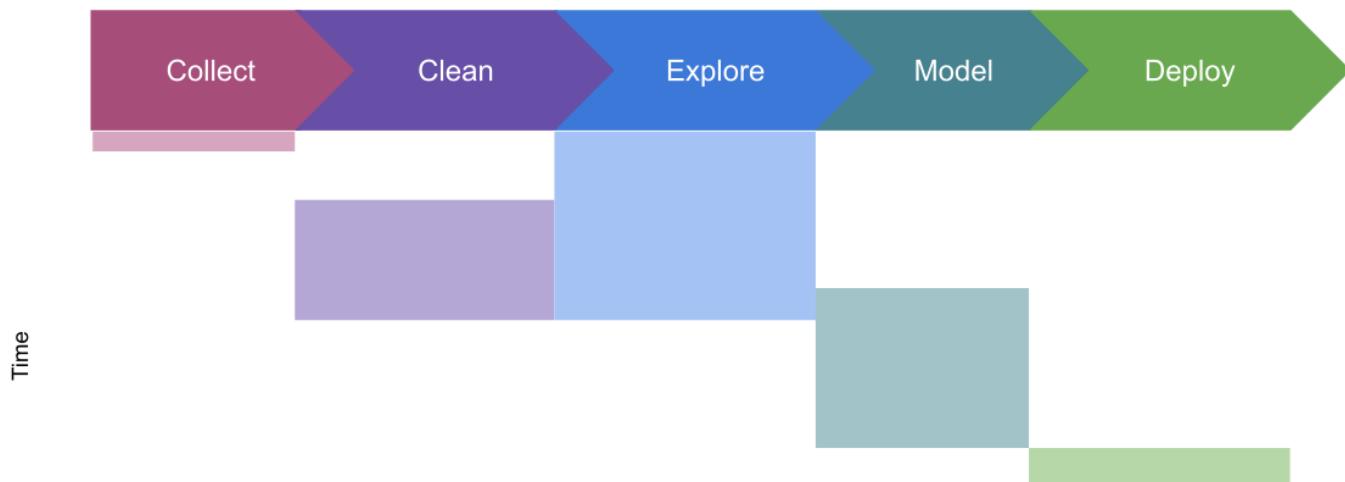


### 1.6.3. how we'll cover Data Science, in depth



- *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
- *deploy*: A little bit at the end, but a lot of preparation for decision making around deployment

### 1.6.4. 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.

## 1.7. Prepare for the next class

- Read carefully the syllabus section of the course website
- skim the rest of the course website
- Bring questions about how the class will work to class on Thursday.
- Review Git & GitHub Fundamentals
- Bring git/github questions on Thursday.
- Begin reading chapter 1 of think like a data scientist (finish in time for it to help you with the assignment due Monday night)

On Thursday we will start with a review of the syllabus. You will answer an ungraded quiz to confirm that you understand and I'll answer all of your questions. Then we will do a little bit with Git/GitHub and start your first assignment in class.

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.

### ⚠ Warning

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.

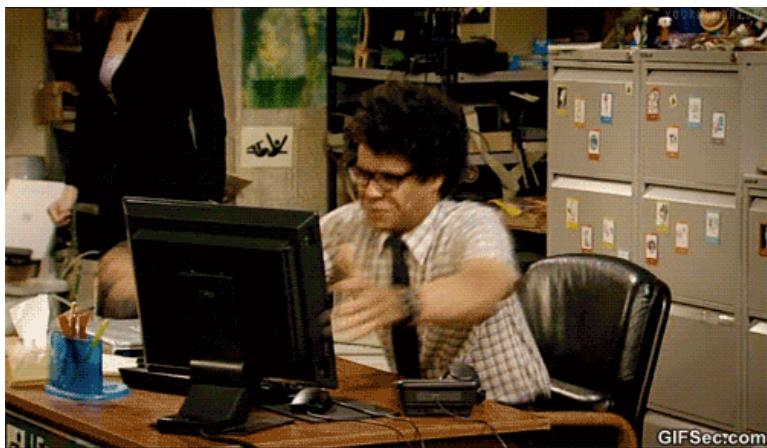
### 💡 Tip

In ch  
1.1, 1

## 2. Syllabus and Python Review

### 2.1. Course Logistics

Class is designed to avoid this:



Julia Evans

@b0rk · Follow

we think about debugging as a technical skill (and it absolutely is!!) but a huge amount of it is managing your feelings so you don't get discouraged and being self-aware so you can recognize your incorrect assumptions

5:35 PM · Jun 11, 2021

•

4K

[Read 79 replies](#)

Read more about how I'm designing this course to help you learn on the [how to learn page](#).

## 2.2. Check your understanding of the syllabus

It's easy when reading something long to lose track of it. Your eyes can go over each word, without actually retaining the information, but it's important to understand the syllabus for the course.

You can find the answers to the following questions on the syllabus. If you've already read it, try answering them to check your understanding. If you haven't read it yet, use these to guide you to get familiar with finding key facts about the course on the syllabus.

1. What do you need to bring to class each day?
2. What is the basis of grading for this course?
3. How do you reference the course text?
4. What is the penalty for missing an assignment?

More information about the course is available throughout the site, the next few questions will help you self-check that you've found the important things. Remember, the goal is not necessarily to memorize all of this, but to be able to find it.

1. When & what are you expected to read for this class?
  - read the text book before class
  - review notes & documentation after class
  - preview the notes & documentation before class

[Skip to main content](#)

- [ ] read documentation and text book after class
1. Your assignment says to find a dataset that has variables of a specific type, which website can you use?
  2. Your assignment says to find a dataset of any type about something you're interested in, which resource would you use?

## 2.3. Python Review

Official source on python:

- pep8 official style
- documentation note that you can change which version you are using

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

### 2.3.1. Functions

Syntax of a function in python:

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

    Parameters
    -----
    name : string
        the name of who 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

#### 💡 Tip

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('sarah')
```

```
'hi sarah'
```

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

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

### 2.3.2. Conditionals

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

    Parameters
    -----
    name : string
        the name of who to greet
    formal : bool
        if the greeting should be 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

```
greeting2('sarah', True)
```

```
'hello sarah'
```

```
greeting2('sarah', False)
```

```
'hi sarah'
```

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

```
greeting2('sarah')
```

```
'hi sarah'
```

```
help(greeting)
```

```
Help on function greeting in module __main__:
```

```
greeting(name)
    say hi to a person
```

```
Parameters
```

```
-----  
name : string  
    the name of who to greet
```

## 2.4. Questions After Class

### 2.4.1. Why is indentation important in python but not other languages like C++?

Python is a newer language than C and C++. Older languages had to contend with the fact that a space character uses the same amount of memory as any other character, so they were not used. However, whitespace is easy to read.

Python was started in 1989, compared to C in 1972, C++ was started in 1985ish, but stuck with a lot of things from C, so tht 1972 is strongly operative.

Python is designed to be easy. It is designed to make complex tasks easier to do. C is designed to be efficient and to compeile well, even if it is hard to learn and do.

### 2.4.2. Why is python so much slower as well?

Python is slower because it is an interpreted language. That means another program called the Python interpreter (which mostly are written in C) is actually running on your computer, that program parses the text of your source code and then executes the code. The interpreter cannot look ahead and change things in how you wrote your code while it runs.

In contrast, C++ (like C) is a compiled language. This means that a program called a compiler parses your code and translates it into assembly then to machine code. During this process it can optimize your code to make sure that it is fast.

### 2.4.3. Are portfolios simply whatever we submit, such as assignments, to Github or are there other things that need to be submitted to the portfolios for level 3?

Assignments are separate from the portfolio checks. It will become more clear what to do in your portfolio after you get feedback on assignment 2, and start working on assignment 3.

### 2.4.4. Will we know the specific criteria to fulfill a level 3 achievement when doing the portfolio?

The evaluation criteria are already listed on the [Detailed Checklists](#).

## 2.4.6. how many large scale programs are we going to write in jupyter?

We won't actually write large scale programs, per se, but we will write some long analyses.

# 3. Grading review, Pandas, and Iterables

## 3.1. Grading Calculation

here is my solution

```
def compute_grade(num_level1, num_level2, num_level3):
    """
    Computes a grade for CSC/DSP310 from numbers of achievements at each level

    Parameters:
    -----
    num_level1 : int
        number of level 1 achievements earned
    num_level2 : int
        number of level 2 achievements earned
    num_level3 : int
        number of level 3 achievements earned

    Returns:
    -----
    letter_grade : string
        letter grade with modifier (+/-)
    """

    if num_level1 == 15:
        if num_level2 == 15:
            if num_level3 == 15:
                grade = 'A'
            elif num_level3 >= 10:
                grade = 'A-'
            elif num_level3 >= 5:
                grade = 'B+'
            else:
                grade = 'B'
        elif num_level2 >= 10:
            grade = 'B-'
        elif num_level2 >= 5:
            grade = 'C+'
        else:
            grade = 'C'
    elif num_level1 >= 10:
        grade = 'C-'
    elif num_level1 >= 5:
        grade = 'D+'
    elif num_level1 >= 3:
        grade = 'D'
    else:
        grade = 'F'

    return grade
```

Note that we can verify it works using `assert`

```
assert compute_grade(15,15,15) =='A'
```

```
assert compute_grade(15,15,9) =='B+'
```

this also means we can assign the value out

```
my_grade = compute_grade(15,11,5)
```

```
my_grade
```

```
'B-'
```

Alternatively if we use a side effect instead, printing the value instead of returning it.

```
def compute_grade_sideeffect(num_level1,num_level2,num_level3):
    """
    Computes a grade for CSC/DSP310 from numbers of achievements at each level

    Parameters:
    -----
    num_level1 : int
        number of level 1 achievements earned
    num_level2 : int
        number of level 2 achievements earned
    num_level3 : int
        number of level 3 achievements earned

    Returns:
    -----
    letter_grade : string
        letter grade with modifier (+/-)
    """

    if num_level1 == 15:
        if num_level2 == 15:
            if num_level3 == 15:
                grade = 'A'
            elif num_level3 >= 10:
                grade = 'A-'
            elif num_level3 >=5:
                grade = 'B+'
            else:
                grade = 'B'
        elif num_level2 >=10:
            grade = 'B-'
        elif num_level2 >=5:
            grade = 'C+'
        else:
            grade = 'C'
    elif num_level1 >= 10:
        grade = 'C-'
    elif num_level1 >= 5:
        grade = 'D+'
    elif num_level1 >=3:
        grade = 'D'
    else:
        grade = 'F'

    print( grade)
```

Look this way it looks similar:

```
compute_grade_sideeffect(15,15,15)
```

```
A
```

```
compute_grade(15,15,15)
```

```
'A'
```

and python lets us assign something

```
my_grade = compute_grade_sideeffect(15,15,15)
```

```
A
```

but the output is nothing

```
type(my_grade)
```

```
NoneType
```

## 3.2. Loading data with Pandas

First we learned about the dataset then we can load it.

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

We will use pandas.

```
import pandas as pd
```

load the data

```
coffee_df = pd.read_csv(coffee_data_url, index_col=0)
```

```
type(coffee_df)
```

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

```
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',  
       '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')
```

```
coffee_df.columns[0]
```

```
'Species'
```

```
len(coffee_df.columns)
```

```
43
```

```
coffee_df.shape
```

```
(28, 43)
```

```
coffee_df.head()
```

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

5 rows × 43 columns

[Skip to main content](#)

```
coffee_df.head
```

		Species	Owner	Country.of.Origin
1	Robusta	ankole coffee producers coop	Uganda	\
2	Robusta	nishant gurjer	India	
3	Robusta	andrew hetzel	India	
4	Robusta	ugacof	Uganda	
5	Robusta	katuka development trust ltd	Uganda	
6	Robusta	andrew hetzel	India	
7	Robusta	andrew hetzel	India	
8	Robusta	nishant gurjer	India	
9	Robusta	nishant gurjer	India	
10	Robusta	ugacof	Uganda	
11	Robusta	ugacof	Uganda	
12	Robusta	nishant gurjer	India	
13	Robusta	andrew hetzel	India	
14	Robusta	kasozi coffee farmers association	Uganda	
15	Robusta	ankole coffee producers coop	Uganda	
16	Robusta	andrew hetzel	India	
17	Robusta	andrew hetzel	India	
18	Robusta	kawacom uganda ltd	Uganda	
19	Robusta	nitubaasa ltd	Uganda	
20	Robusta	mannya coffee project	Uganda	
21	Robusta	andrew hetzel	India	
22	Robusta	andrew hetzel	India	
23	Robusta	andrew hetzel	United States	
24	Robusta	luis robles	Ecuador	
25	Robusta	luis robles	Ecuador	
26	Robusta	james moore	United States	
27	Robusta	cafe politico	India	
28	Robusta	cafe politico	Vietnam	

	Farm.Name	Lot.Number
1	kyangundu cooperative society	Nan \
2	sethuraman estate kaapi royale	25
3	sethuraman estate	Nan
4	ugacof project area	Nan
5	katikamu capca farmers association	Nan
6		Nan
7	sethuraman estates	Nan
8	sethuraman estate kaapi royale	7
9	sethuraman estate	RKR
10	ishaka	Nan
11	ugacof project area	Nan
12	sethuraman estate kaapi royale	RC AB
13	sethuraman estates	Nan
14	kasozi coffee farmers	Nan
15	kyangundu coop society	Nan
16	sethuraman estate	Nan
17	sethuraman estates	Nan
18	bushenyi	Nan
19	kigezi coffee farmers association	Nan
20	mannya coffee project	Nan
21	sethuraman estates	Nan
22	sethuraman estates	Nan
23	sethuraman estates	Nan
24	robustasa	Lavado 1
25	robustasa	Lavado 3
26	fazenda cazengo	Nan
27		Nan
28		Nan

	Mill	ICO.Number
1	ankole coffee producers	0 \
2	sethuraman estate	14/1148/2017/21
3		0000
4	ugacof	0
5	katuka development trust	0
6	(self)	NaN
7		NaN
8	sethuraman estate	14/1148/2017/18
9	sethuraman estate	14/1148/2016/17
10	nsuhuna lumar	A

[Skip to main content](#)

13		NaN	NaN
14		NaN	0
15	ankole coffee producers coop union ltd		0
16		NaN	0000
17	sethuraman estates		NaN
18	kawacom		0
19	nitubaasa		0
20	mannya coffee project		0
21	NaN		NaN
22	sethuraman estates		NaN
23	sethuraman estates		NaN
24	our own lab		NaN
25	own laboratory		NaN
26	cafe cazengo		NaN
27		NaN	14-1118-2014-0087
28		NaN	NaN

	Company	Altitude	
1	ankole coffee producers coop	1488	\
2	kaapi royale	3170	
3	sethuraman estate	1000m	
4	ugacof ltd	1212	
5	katuka development trust ltd	1200-1300	
6	cafemakers, llc	3000'	
7	cafemakers	750m	
8	kaapi royale	3140	
9	kaapi royale	1000	
10	ugacof ltd	900-1300	
11	ugacof ltd	1095	
12	kaapi royale	1000	
13	cafemakers	750m	
14	kasozi coffee farmers association	1367	
15	ankole coffee producers coop	1488	
16	sethuraman estate	1000m	
17	cafemakers, llc	750m	
18	kawacom uganda ltd	1600	
19	nitubaasa ltd	1745	
20	mannya coffee project	1200	
21	cafemakers	750m	
22	cafemakers, llc	750m	
23	cafemakers, llc	3000'	
24	robustasa	NaN	
25	robustasa	40	
26	global opportunity fund	795	meters
27	cafe politico	NaN	
28	cafe politico	NaN	

	Region	...	Color	Category.Two.Defects	
1	sheema south western	...	Green	2	\
2	chikmagalur karnataka india	...	NaN	2	
3	chikmagalur	...	Green	0	
4	central	...	Green	7	
5	luwero central region	...	Green	3	
6	chikmagalur	...	Green	0	
7	chikmagalur	...	Green	0	
8	chikmagalur karnataka india	...	Bluish-Green	0	
9	chikmagalur karnataka	...	Green	0	
10	western	...	Green	6	
11	iganga namadropo eastern	...	Green	1	
12	chikmagalur karnataka	...	Green	0	
13	chikmagalur	...	Green	1	
14	eastern	...	Green	7	
15	south western	...	Green	2	
16	chikmagalur	...	Green	0	
17	chikmagalur	...	Blue-Green	0	
18	western	...	Green	1	
19	western	...	Green	2	
20	southern	...	Green	1	
21	chikmagalur	...	Bluish-Green	1	
22	chikmagalur	...	Green	0	
23	chikmagalur	...	Green	0	
24	san juan plavas		Blue-Green	1	

27 NaN ... Green 1  
28 NaN ... NaN 9

	Expiration	Certification.Body
1	June 26th, 2015	Uganda Coffee Development Authority \
2	October 31st, 2018	Specialty Coffee Association
3	April 29th, 2016	Specialty Coffee Association
4	July 14th, 2015	Uganda Coffee Development Authority
5	June 26th, 2015	Uganda Coffee Development Authority
6	February 28th, 2013	Specialty Coffee Association
7	May 15th, 2015	Specialty Coffee Association
8	October 25th, 2018	Specialty Coffee Association
9	August 17th, 2017	Specialty Coffee Association
10	August 5th, 2015	Uganda Coffee Development Authority
11	June 26th, 2015	Uganda Coffee Development Authority
12	August 23rd, 2017	Specialty Coffee Association
13	May 19th, 2015	Specialty Coffee Association
14	July 14th, 2015	Uganda Coffee Development Authority
15	July 14th, 2015	Uganda Coffee Development Authority
16	April 29th, 2016	Specialty Coffee Association
17	June 3rd, 2014	Specialty Coffee Association
18	June 27th, 2015	Uganda Coffee Development Authority
19	June 27th, 2015	Uganda Coffee Development Authority
20	June 27th, 2015	Uganda Coffee Development Authority
21	May 19th, 2015	Specialty Coffee Association
22	June 20th, 2014	Specialty Coffee Association
23	February 28th, 2013	Specialty Coffee Association
24	January 18th, 2017	Specialty Coffee Association
25	January 18th, 2017	Specialty Coffee Association
26	December 23rd, 2015	Specialty Coffee Association
27	August 25th, 2015	Specialty Coffee Association
28	August 25th, 2015	Specialty Coffee Association

	Certification.Address
1	e36d0270932c3b657e96b7b0278dfd85dc0fe743 \
2	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
3	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
4	e36d0270932c3b657e96b7b0278dfd85dc0fe743
5	e36d0270932c3b657e96b7b0278dfd85dc0fe743
6	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
7	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
8	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
9	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
10	e36d0270932c3b657e96b7b0278dfd85dc0fe743
11	e36d0270932c3b657e96b7b0278dfd85dc0fe743
12	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
13	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
14	e36d0270932c3b657e96b7b0278dfd85dc0fe743
15	e36d0270932c3b657e96b7b0278dfd85dc0fe743
16	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
17	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
18	e36d0270932c3b657e96b7b0278dfd85dc0fe743
19	e36d0270932c3b657e96b7b0278dfd85dc0fe743
20	e36d0270932c3b657e96b7b0278dfd85dc0fe743
21	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
22	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
23	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
24	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
25	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
26	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
27	ff7c18ad303d4b603ac3f8cff7e611ffc735e720
28	ff7c18ad303d4b603ac3f8cff7e611ffc735e720

	Certification.Contact unit_of_measurement
1	03077a1c6bac60e6f514691634a7f6eb5c85aae8 m \
2	352d0cf7f3e9be14dad7df644ad65efc27605ae2 m
3	352d0cf7f3e9be14dad7df644ad65efc27605ae2 m
4	03077a1c6bac60e6f514691634a7f6eb5c85aae8 m
5	03077a1c6bac60e6f514691634a7f6eb5c85aae8 m
6	352d0cf7f3e9be14dad7df644ad65efc27605ae2 m
7	352d0cf7f3e9be14dad7df644ad65efc27605ae2 m
8	352d0cf7f3e9be14dad7df644ad65efc27605ae2 m

[Skip to main content](#)

```

11 03077a1c6bac60e6f514691634a7f6eb5c85aae8      m
12 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
13 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
14 03077a1c6bac60e6f514691634a7f6eb5c85aae8      m
15 03077a1c6bac60e6f514691634a7f6eb5c85aae8      m
16 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
17 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
18 03077a1c6bac60e6f514691634a7f6eb5c85aae8      m
19 03077a1c6bac60e6f514691634a7f6eb5c85aae8      m
20 03077a1c6bac60e6f514691634a7f6eb5c85aae8      m
21 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
22 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
23 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
24 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
25 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
26 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
27 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m
28 352d0cf7f3e9be14dad7df644ad65efc27605ae2      m

  altitude_low_meters altitude_high_meters  altitude_mean_meters
1          1488.0            1488.0           1488.0
2          3170.0            3170.0           3170.0
3          1000.0            1000.0           1000.0
4          1212.0            1212.0           1212.0
5          1200.0            1300.0           1250.0
6          3000.0            3000.0           3000.0
7          750.0             750.0            750.0
8          3140.0            3140.0           3140.0
9          1000.0            1000.0           1000.0
10         900.0             1300.0           1100.0
11         1095.0            1095.0           1095.0
12         1000.0            1000.0           1000.0
13         750.0             750.0            750.0
14         1367.0            1367.0           1367.0
15         1488.0            1488.0           1488.0
16         1000.0            1000.0           1000.0
17         750.0             750.0            750.0
18         1600.0            1600.0           1600.0
19         1745.0            1745.0           1745.0
20         1200.0            1200.0           1200.0
21         750.0             750.0            750.0
22         750.0             750.0            750.0
23         3000.0            3000.0           3000.0
24         NaN               NaN              NaN
25         40.0              40.0             40.0
26         795.0             795.0            795.0
27         NaN               NaN              NaN
28         NaN               NaN              NaN

```

[28 rows x 43 columns]>

`coffee_df.head(2)`

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitud
1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ankole coffee producers coop	1488.0
2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	3170.0

2 rows x 43 columns

[Skip to main content](#)

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	
24	Robusta	luis robles	Ecuador	robustasa	Lavado 1	our own lab	NaN	robustasa	NaN	sa
25	Robusta	luis robles	Ecuador	robustasa	Lavado 3	own laboratory	NaN	robustasa	40	sa
26	Robusta	james moore	United States	fazenda cazengo	NaN	cafe cazengo	NaN	global opportunity fund	795 meters	pr
27	Robusta	cafe politico	India	NaN	NaN	NaN	14-1118-2014-0087	cafe politico	NaN	
28	Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaN	cafe politico	NaN	

5 rows × 43 columns

```
coffee_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 28 entries, 1 to 28
Data columns (total 43 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Species           28 non-null      object  
 1   Owner              28 non-null      object  
 2   Country.of.Origin  28 non-null      object  
 3   Farm.Name          25 non-null      object  
 4   Lot.Number         6 non-null       object  
 5   Mill               20 non-null      object  
 6   ICO.Number         17 non-null      object  
 7   Company            28 non-null      object  
 8   Altitude           25 non-null      object  
 9   Region             26 non-null      object  
 10  Producer           26 non-null      object  
 11  Number.of.Bags    28 non-null      int64  
 12  Bag.Weight         28 non-null      object  
 13  In.Country.Partner 28 non-null      object  
 14  Harvest.Year       28 non-null      int64  
 15  Grading.Date       28 non-null      object  
 16  Owner.1            28 non-null      object  
 17  Variety            3 non-null       object  
 18  Processing.Method  10 non-null      object  
 19  Fragrance...Aroma  28 non-null      float64 
 20  Flavor              28 non-null      float64 
 21  Aftertaste          28 non-null      float64 
 22  Salt...Acid         28 non-null      float64 
 23  Bitter...Sweet       28 non-null      float64 
 24  Mouthfeel           28 non-null      float64 
 25  Uniform.Cup         28 non-null      float64 
 26  Clean.Cup            28 non-null      float64 
 27  Balance              28 non-null      float64 
 28  Copper.Points        28 non-null      float64 
 29  Total.Cup.Points     28 non-null      float64 
 30  Moisture             28 non-null      float64 
 31  Category.One.Defects 28 non-null      int64  
 32  Quakers              28 non-null      int64  
 33  Color                25 non-null      object  
 34  Category.Two.Defects 28 non-null      int64  
 35  Expiration            28 non-null      object  
 36  Certification.Body     28 non-null      object  
 37  Certification.Address   28 non-null      object  
 38  Certification.Contact   28 non-null      object  
 39  unit_of_measurement     28 non-null      object  
 40  altitude_low_meters     25 non-null      float64 
 41  altitude_high_meters     25 non-null      float64 
 42  altitude_mean_meters     25 non-null      float64 
dtypes: float64(15), int64(5), object(23)
memory usage: 9.6+ KB
```

```
coffee_df.columns[42]
```

```
'altitude_mean_meters'
```

```
col_types = coffee_df.dtypes
```

```
col_types[:5]
```

```
Species          object
Owner            object
Country.of.Origin  object
Farm.Name        object
Lot.Number       object
dtype: object
```

```
type(col_types[0])
```

```
numpy.dtype[object_]
```

```
for col_name in coffee_df.columns:
    print(col_name[:3])
```

```
Spe
Own
Cou
Far
Lot
Mil
ICO
Com
Alt
Reg
Pro
Num
Bag
In.
Har
Gra
Own
Var
Pro
Fra
Fla
Aft
Sal
Bit
Mou
Uni
Cle
Bal
Cup
Tot
Moi
Cat
Qua
Col
Cat
Exp
Cer
Cer
Cer
uni
alt
alt
alt
```

```
my_list = ['honda', 'ford', 'nissan']
```

```
type(my_list)
```

```
list
```

```
my_list[-1]
```

```
'nissan'
```

```
short_names = [col_name[:3] for col_name in coffee_df.columns]
```

```
type(short_names)
```

```
list
```

```
short_names
```

```
['Spe',
 'Own',
 'Cou',
 'Far',
 'Lot',
 'Mil',
 'ICO',
 'Com',
 'Alt',
 'Reg',
 'Pro',
 'Num',
 'Bag',
 'In.',
 'Har',
 'Gra',
 'Own',
 'Var',
 'Pro',
 'Fra',
 'Fla',
 'Aft',
 'Sal',
 'Bit',
 'Mou',
 'Uni',
 'Cle',
 'Bal',
 'Cup',
 'Tot',
 'Moi',
 'Cat',
 'Qua',
 'Col',
 'Cat',
 'Exp',
 'Cer',
 'Cer',
 'Cer',
 'uni',
 'alt',
 'alt',
 'alt']
```

### 3.3. Questions After Class

3.3.1. • will we be gathering our own data or will it all be provided for the course?

3.3.2. Will you always give us an answer key for assignments?

No, but we will always give personalized feedback.

3.3.3. will we be cleaning data?

Yes, see the notes from the first class.

3.3.4. Will we do correlations later?

### **3.3.5. will we go over more imports like pandas?**

Yes, we will use other libraries. In A2, you'll even import your own code.

### **3.3.6. How will datasets be used in conjunction with one another?**

We will combine data in a few weeks.

### **3.3.7. How similar of an alternative to modeling in R exists in python?**

They are both compete languages so at some level, they can both do all the same things. Some libraries are easier/better for specific things in one language or the other.

### **3.3.8. Will we be working with databases at all in this class?**

A little. We'll pull from a database into python.

### **3.3.9. Are we going going to be using pandas a lot in this class?**

Yes. We will use pandas for almost every single remaining class session this semester.

### **3.3.10. how do you short-cut fill variable names in jupyter**

Press tab to autocomplete

### **3.3.11. what is the most efficient way to get help on the homework ?**

Accept the assignment and make an issue or go to office hours for general questions. For clarifying questions, you can post an issue on the course website.

If you are stuck with some progress made, upload (or push) your code first and then ask for help so we can see where you are.

### **3.3.12. is there a best or most common way to organize data for use**

csv

### **3.3.13. how would you laod a csv file with python if i was using visual studio instead of jupyter?**

All of the code we are writing will work in any python environment that has the libraries. the Editor you use (jupyter vs VSCode vs PyCharm) does not impact what you can do with python. Which interpreter you use can impact and jupyter does default to ipython instead of the core python kernel, but that does not change how to load data.

Remember that jupyter is not on a cloud service. YOu can use any file on your computer with a relative path.

## 4. Pandas and Indexing

### 4.1. Iterable types

```
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()
```

a

```
['a', 'b', 'c', 'd', 'e']
```

```
type(a)
```

list

b

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4}
```

```
type(b)
```

dict

c

```
('a', 'b', 'c', 'd', 'e')
```

```
type(c)
```

tuple

d

```
['a', 'b', 'c', 'd', 'e']
```

```
type(d)
```

```
list
```

## 4.2. Reading data other ways

```
import pandas as pd
```

```
course_comms_url = 'https://rhodyprog4ds.github.io/BrownSpring23/syllabus/communication.html'
```

This reads in from the html directly.

```
pd.read_html(course_comms_url)
```

```
[   Day      Time           Location     Host
0  Mon  11am-1pm  Tyler 139 and zoom    Kyle
1  Wed  7-8:30pm        Zoom  Dr. Brown
2  Fri      3-6pm        Zoom      Kyle]
```

```
html_list = pd.read_html(course_comms_url)
```

```
type(html_list)
```

```
list
```

```
type(html_list[0])
```

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

```
[type(h) for h in html_list]
```

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

```
achievements_url = 'https://rhodyprog4ds.github.io/BrownSpring23/syllabus/achievements.html'
```

get the tables

```
achievements_df_list = pd.read_html(achievements_url)
```

make a list means use a list comprehension

```
[(14, 3), (15, 5), (15, 15), (15, 6)]
```

```
achievements_df_list.shape
```

```
-----  
AttributeError                                 Traceback (most recent call last)  
Cell In[20], line 1  
----> 1 achievements_df_list.shape  
  
AttributeError: 'list' object has no attribute 'shape'
```

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

```
coffee_df = pd.read_csv(coffee_data_url, index_col=0)
```

```
coffee_df.head(1)
```

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

1 rows × 43 columns

```
coffee_df['Species'].head()
```

```
1    Robusta  
2    Robusta  
3    Robusta  
4    Robusta  
5    Robusta  
Name: Species, dtype: object
```

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

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

```
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',
       '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')
```

```
coffee_df['Number.of.Bags']
```

```
1    300
2    320
3    300
4    320
5     1
6    200
7    320
8    320
9    320
10   320
11   320
12   320
13   100
14    1
15   320
16   300
17   140
18    1
19    20
20     6
21   100
22   250
23   100
24    1
25    1
26    1
27    1
28    1
Name: Number.of.Bags, dtype: int64
```

```
new_values = {0: '<100', 1: '100-199', 2: '200-299', 3: '300+'}
```

```
[new_values[int(num/100)] for num in coffee_df['Number.of.Bags']]
```

```
[ '300+',
  '300+',
  '300+',
  '300+',
  '<100',
  '200-299',
  '300+',
  '300+',
  '300+',
  '300+',
  '300+',
  '300+',
  '300+',
  '300+',
  '100-199',
  '<100',
  '300+',
  '300+',
  '100-199',
  '<100',
  '<100',
  '<100',
  '100-199',
  '200-299',
  '100-199',
  '<100',
  '<100',
  '<100',
  '<100',
  '<100']
```

```
bags_bin = lambda num: int(num/100)
[new_values[bags_bin(num)] for num in coffee_df['Number.of.Bags']]
```

```
[ '300+',
  '300+',
  '300+',
  '300+',
  '<100',
  '200-299',
  '300+',
  '300+',
  '300+',
  '300+',
  '300+',
  '300+',
  '300+',
  '300+',
  '100-199',
  '<100',
  '300+',
  '300+',
  '100-199',
  '<100',
  '<100',
  '<100',
  '100-199',
  '200-299',
  '100-199',
  '<100',
  '<100',
  '<100',
  '<100']
```

```
type(pd.read_csv)
```

```
type(bags_bin)
```

```
function
```

## 4.3. Importing locally

If I make a file in the same folder as my notebook called `example.py` and then put

▶ Show code cell source

```
name = 'sarah'
```

in the file, we can use that file like:

```
from example import name
```

```
name
```

```
'sarah'
```

```
import example
```

```
example.name
```

```
'sarah'
```

## 4.4. Questions After Class

4.4.1. why does casting the int over the `(num/100)` give you the right number? Is it because of floor division?

First let's look at an interim value, lets pick a value for `num`

```
num = 307
```

Then do the calculation without casting to int

```
num/100
```

Remember that `int` type is an integer or whole number, no fraction. So, casting drops the decimal part.

#### 4.4.2. How would adding 2 DataFrames together of separate types affect the type command?

It depends what “add” means. If addition it might error, but if it worked, then it would still be a DataFrame. If stacking with `pd.concat` it would also be a DataFrame.

If you make them into a list, then the would be a list.

#### 4.4.3. what keys to use in the dictionaries?

In the assignment the instruction say

#### 4.4.4. how to save as a local csv file?

`pandas.DataFrame.to_csv`

#### 4.4.5. how to create a DataFrame?

Use the `constructor`

#### 4.4.6. how to read using relative path?

A relative path can work just like a URL. [read about them here](#)

#### 4.4.7. I would like to know about other common forms of data files.

The pandas documentation's [I/O](#) page is where I recommend starting

### 4.5. What other libraries do we end up using?

Next week we will use `seaborn` for plotting. Later in the semester we will use `sklearn` for machine learning. We will use a few other libraries for a few features, but these three are the main ones.

## 5. Exploratory Data Analysis (EDA)

Again, we import pandas as usual

```
import pandas as pd
```

```
coffee_data_url = 'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/robusta_data_.csv'
coffee_df = pd.read_csv(coffee_data_url, index_col=0)
```

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

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

## 5.2. 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.

We can also get more structural information with the `info` method.

```
coffee_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 28 entries, 1 to 28
Data columns (total 43 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Species           28 non-null      object  
 1   Owner              28 non-null      object  
 2   Country.of.Origin  28 non-null      object  
 3   Farm.Name          25 non-null      object  
 4   Lot.Number         6 non-null       object  
 5   Mill               20 non-null      object  
 6   ICO.Number         17 non-null      object  
 7   Company            28 non-null      object  
 8   Altitude           25 non-null      object  
 9   Region             26 non-null      object  
 10  Producer           26 non-null      object  
 11  Number.of.Bags    28 non-null      int64  
 12  Bag.Weight         28 non-null      object  
 13  In.Country.Partner 28 non-null      object  
 14  Harvest.Year       28 non-null      int64  
 15  Grading.Date       28 non-null      object  
 16  Owner.1            28 non-null      object  
 17  Variety            3 non-null       object  
 18  Processing.Method  10 non-null      object  
 19  Fragrance...Aroma  28 non-null      float64 
 20  Flavor              28 non-null      float64 
 21  Aftertaste          28 non-null      float64 
 22  Salt...Acid         28 non-null      float64 
 23  Bitter...Sweet       28 non-null      float64 
 24  Mouthfeel           28 non-null      float64 
 25  Uniform.Cup         28 non-null      float64 
 26  Clean.Cup           28 non-null      float64 
 27  Balance              28 non-null      float64 
 28  Copper.Points       28 non-null      float64 
 29  Total.Cup.Points    28 non-null      float64 
 30  Moisture             28 non-null      float64 
 31  Category.One.Defects 28 non-null      int64  
 32  Quakers             28 non-null      int64  
 33  Color                25 non-null      object  
 34  Category.Two.Defects 28 non-null      int64  
 35  Expiration           28 non-null      object  
 36  Certification.Body    28 non-null      object  
 37  Certification.Address 28 non-null      object  
 38  Certification.Contact 28 non-null      object  
 39  unit_of_measurement    28 non-null      object  
 40  altitude_low_meters   25 non-null      float64 
 41  altitude_high_meters  25 non-null      float64 
 42  altitude_mean_meters  25 non-null      float64 
dtypes: float64(15), int64(5), object(23)
memory usage: 9.6+ KB
```

Now, we can actually start to analyze the data itself.

The `describe` method provides us with a set of summary statistics that broadly describe the data overall.

```
coffee_df.describe()
```

	Number.of.Bags	Harvest.Year	Fragrance...Aroma	Flavor	Aftertaste	Salt...Acid	Bitter...Sweet	Mouthf
<b>count</b>	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000
<b>mean</b>	168.000000	2013.964286	7.702500	7.630714	7.559643	7.657143	7.675714	7.5067
<b>std</b>	143.226317	1.346660	0.296156	0.303656	0.342469	0.261773	0.317063	0.7251
<b>min</b>	1.000000	2012.000000	6.750000	6.670000	6.500000	6.830000	6.670000	5.0800
<b>25%</b>	1.000000	2013.000000	7.580000	7.560000	7.397500	7.560000	7.580000	7.5000
<b>50%</b>	170.000000	2014.000000	7.670000	7.710000	7.670000	7.710000	7.750000	7.6700
<b>75%</b>	320.000000	2015.000000	7.920000	7.830000	7.770000	7.830000	7.830000	7.8300
<b>max</b>	320.000000	2017.000000	8.330000	8.080000	7.920000	8.000000	8.420000	8.2500

From this, we can draw several conclusions. For example straightforward ones like:

- the smallest number of bags rated is 1 and at least 25% of the coffees rates only had 1 bag
- the first ratings included were 2012 and last in 2017 (min & max)
- the mean Mouthfeel was 7.5
- Category One defects are not very common (the 75th% is 0)

Or more nuanced ones that compare across variables like

- the raters scored coffee higher on Uniformity.Cup and Clean.Cup than other scores (mean score; only on the ones that seem to have a scale of up to 8/10)
- the coffee varied more in Mouthfeel and Balance than most other scores (the std; only on the ones that seem to have a scale of up to 8/10)
- there are 3 ratings with no altitude (count of other variables is 28; alt is 25)

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

We can use the descriptive statistics on individual columns as well.

### 5.2.1. Understanding Quantiles

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

We can use the descriptive statistics on individual columns as well.

```
coffee_df['Uniform.Cup'].describe()
```

count	28.000000
mean	9.904286
std	0.238753
min	9.330000
25%	10.000000
50%	10.000000
75%	10.000000
max	10.000000
Name:	Uniform.Cup, dtype: float64

```
coffee_df[['Uniform.Cup', 'Mouthfeel']].describe()
```

	Uniform.Cup	Mouthfeel
count	28.000000	28.000000
mean	9.904286	7.506786
std	0.238753	0.725152
min	9.330000	5.080000
25%	10.000000	7.500000
50%	10.000000	7.670000
75%	10.000000	7.830000
max	10.000000	8.250000

## 5.3. Individual statistics

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

```
coffee_df.mean(numeric_only=True)
```

```
Number.of.Bags      168.000000
Harvest.Year       2013.964286
Fragrance...Aroma    7.702500
Flavor              7.630714
Aftertaste          7.559643
Salt...Acid         7.657143
Bitter...Sweet       7.675714
Mouthfeel           7.506786
Uniform.Cup         9.904286
Clean.Cup            9.928214
Balance              7.541786
Cupper.Points        7.761429
Total.Cup.Points     80.868929
Moisture             0.065714
Category.One.Defects 2.964286
Quakers              0.000000
Category.Two.Defects 1.892857
altitude_low_meters 1367.600000
altitude_high_meters 1387.600000
altitude_mean_meters 1377.600000
dtype: float64
```

```
coffee_df['Flavor'].quantile(.8)
```

```
7.83
```

```
coffee_df['Aftertaste'].mean()
```

```
7.559642857142856
```

```
coffee_df.head(2)
```

	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	1480
2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	3100

2 rows × 43 columns

## 5.4. Working with categorical data

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

```
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')
```

```
coffee_df.describe().columns
```

```
Index(['Number.of.Bags', 'Harvest.Year', 'Fragrance...Aroma', 'Flavor',
       'Aftertaste', 'Salt...Acid', 'Bitter...Sweet', 'Mouthfeel',
       'Uniform.Cup', 'Clean.Cup', 'Balance', 'Copper.Points',
       'Total.Cup.Points', 'Moisture', 'Category.One.Defects', 'Quakers',
       'Category.Two.Defects', 'altitude_low_meters', 'altitude_high_meters',
       'altitude_mean_meters'],
      dtype='object')
```

We can get the prevalence of each one with `value_counts`

```
coffee_df['Color'].value_counts()
```

```
Color
Green      20
Blue-Green   3
Bluish-Green  2
Name: count, dtype: int64
```

### Try it Yourself

Note `value_counts` does not count the `NaN` values, but `count` counts all of the not missing values and the shape of the DataFrame is the total number of rows. How can you get the number of missing Colors?

Describe only operates on the numerical columns, but we might want to know about the others. We can get the number of each value with `value_counts`

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

```
Country.of.Origin
India          13
Uganda         10
United States   2
Ecuador         2
Vietnam         1
Name: count, dtype: int64
```

Value counts returns a pandas Series that has two parts: values and index

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

```
array([13, 10, 2, 2, 1])
```

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

```
Index(['India', 'Uganda', 'United States', 'Ecuador', 'Vietnam'], dtype='object', name='Country.of.Origin')
```

The max takes the max of the values.

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

```
13
```

We can get the name of the most common country out of this Series using `idxmax`

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

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

[Skip to main content](#)

Or see only how many different values with the related:

```
coffee_df['Country.of.Origin'].nunique()
```

5

## 5.5. Split-Apply-Combine

So, we can summarize data now, but the summaries we have done so far have treated each variable one at a time. The most interesting patterns are often in how multiple variables interact. We'll do some modeling that looks at multivariate functions of data in a few weeks, but for now, we do a little more with summary statistics.

For example, how does the flavor ratings relate to the country?

```
coffee_df.groupby('Country.of.Origin')['Flavor'].describe()
```

	count	mean	std	min	25%	50%	75%	max
<b>Country.of.Origin</b>								
Ecuador	2.0	7.625000	0.063640	7.58	7.6025	7.625	7.6475	7.67
India	13.0	7.640769	0.279835	6.83	7.5800	7.750	7.7500	7.92
Uganda	10.0	7.758000	0.197754	7.42	7.6025	7.790	7.8975	8.08
United States	2.0	7.415000	0.120208	7.33	7.3725	7.415	7.4575	7.50
Vietnam	1.0	6.670000		NaN	6.67	6.6700	6.670	6.6700

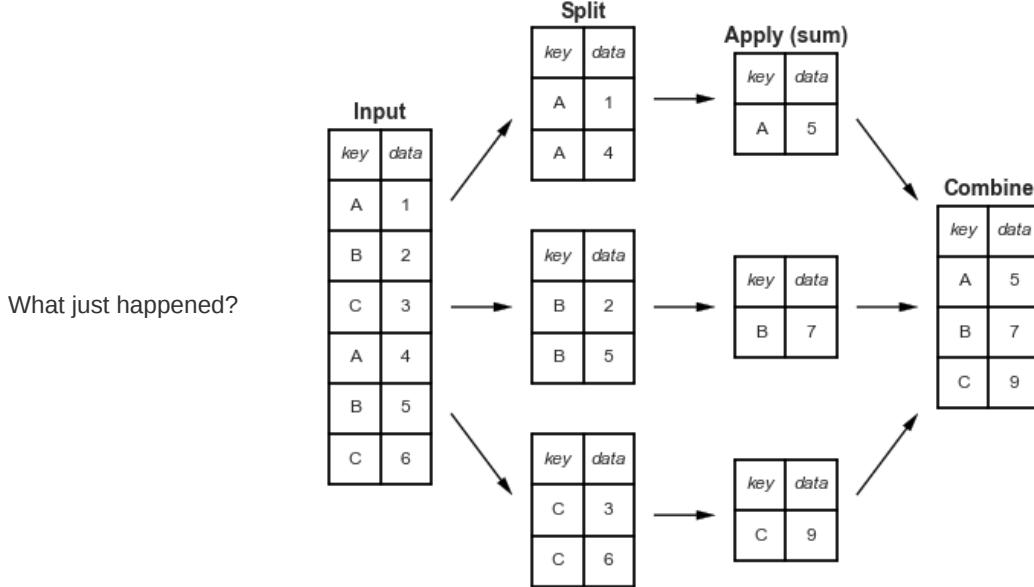
Above we saw which country had the most ratings (remember one row is one rating), but what if we wanted to see the mean number of bags per country?

```
coffee_df.groupby('Country.of.Origin')['Number.of.Bags'].mean()
```

```
Country.of.Origin
Ecuador           1.000000
India             230.076923
Uganda            160.900000
United States     50.500000
Vietnam           1.000000
Name: Number.of.Bags, dtype: float64
```

### Important

This data is only about coffee that was rated by a particular agency it is not economic data, so we cannot, for example conclude which country produces the amount of data. If we had economic dataset, a `Number.of.Bags` column's mean would tell us exactly that, but the context of the dataset defines what a row means and therefore how we can interpret the **every single statistic** we calculate.



Groupby splits the whole dataframe into parts where each part has the same value for `Country.of-Origin` and then after that, we extracted the `Number.of.Bags` column, took the sum (within each separate group) and then put it all back together in one table (in this case, a `Series` because we picked one variable out)

### 5.5.1. How does Groupby Work?

#### ! Important

This is more details with code examples on how the groupby works. If you want to run this code for yourself, use the download icon at the top right to download these notes as a notebook.

We can view this by saving the groupby object as a variable and exploring it.

```
country_grouped = coffee_df.groupby('Country.of.Origin')
country_grouped
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f8415fa0190>
```

Trying to look at it without applying additional functions, just tells us the type. But, it's iterable, so we can loop over.

```
for country, df in country_grouped:
    print(type(country), type(df))
```

```
<class 'str'> <class 'pandas.core.frame.DataFrame'>
```

We could manually compute things using the data structure, if needed, though using pandas functionality will usually do what we want. For example:

```
bag_total_dict = {}

for country,df in country_grouped:
    tot_bags = df['Number.of.Bags'].sum()
    bag_total_dict[country] = tot_bags

pd.DataFrame.from_dict(bag_total_dict, orient='index',
                       columns = ['Number.of.Bags.Sum'])
```

Number.of.Bags.Sum	
Ecuador	2
India	2991
Uganda	1609
United States	101
Vietnam	1

is the same as what we did before

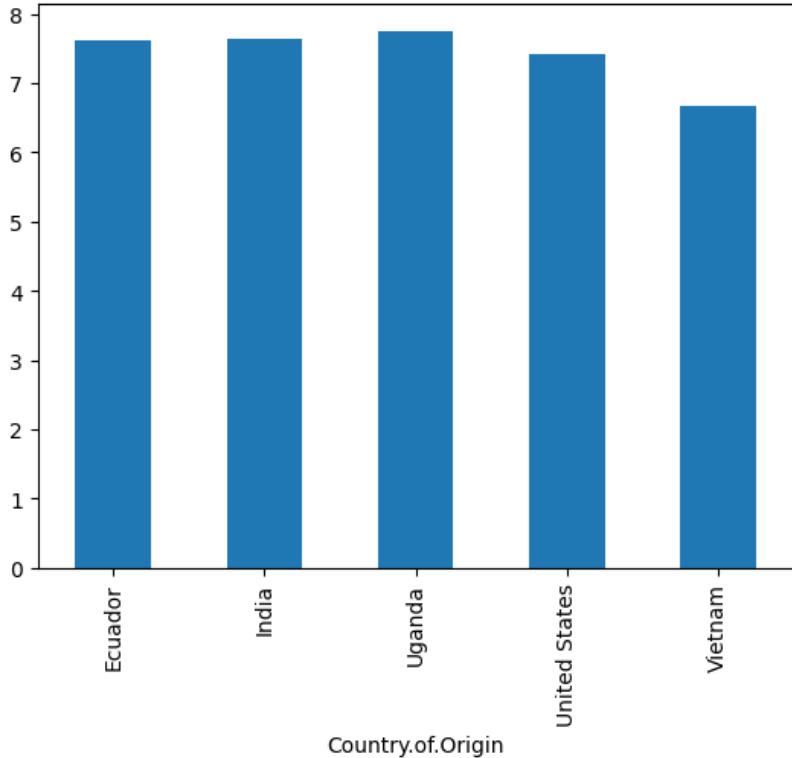
## 5.6. Plotting with Pandas

Pandas allows us to do basic plots on a `DataFrame` or `Series` with the `plot` method.

We want bars so we will use the `kind` parameter to switch it.

```
coffee_df.groupby('Country.of.Origin')['Flavor'].mean().plot(kind='bar')
```

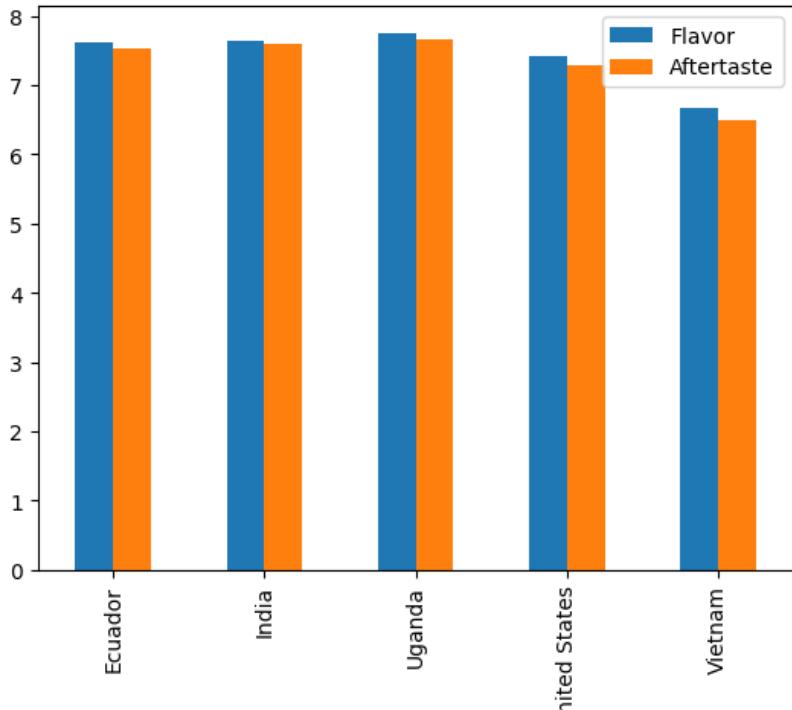
```
<Axes: xlabel='Country.of.Origin'>
```



It can also be done on a dataframe like this

```
coffee_df.groupby('Country.of.Origin')[['Flavor', 'Aftertaste']].mean().plot(kind='bar')
```

```
<Axes: xlabel='Country.of.Origin'>
```



[Skip to main content](#)

Note that it adds a legend for us and uses two colors.

#### 🔔 What is the default plot type

Try removing the `kind=bar` and see what it does

## 5.7. Questions after Class

### 5.7.1. Are there any calls for examples such as .plot or .describe that you do not want us to use?

Everything in pandas is welcome, unless it is deprecated or not recommended by pandas. Pandas will tell, like the FutureWarning that we saw today. Work will be accepted with warnings, most of the time but it is not best practice and some that I specifically tell you to avoid as we encounter them will not be accepted.

### 5.7.2. What keyboard key do you use to run the code so I don't have to use the mouse?

hold shift, press enter

### 5.7.3. How do we take the plots and save them to a separate file?

We will need an additional library to do this, we will do that on Thursday.

### 5.7.4. how do you display different types of charts

the `kind` attribute can change to differen types

### 5.7.5. When you give feedback on an assignment is there a way to fix it to get the points you said it did not meet?

No, you can attempt in the next assignment or portfolio check.

### 5.7.6. I want to know more about the limitations of pandas

pandas is relatively slow and cannot use accelerated hardware very well. However it is still good to learn because it has gained a lot of traction. So much so that in `modin` you can change one line of code to get those advantages.

### 5.7.7. Can Jupyter use other graphics software?

Jupyter notebooks are a file(roughly a json). You can edit it using any text editor. You can also convert to a plain text files using `jupytext` that is still runnable.

## 5.7.8. How do we generate different models as done in r, also is there a supernova function?

R is designed by and for statisticians. Most of the calculations can be done. They may be slightly more clunky in Python than R.

## 5.7.9. I had a question on the assignment, in the `datasets.py` file were we were supposed to save a function handle, should it be a function object or a string?

function object

## 5.7.10. Besides accepting the invite, is there any more setup we were supposed to do with the achievement tracker repository?

No, that's it.

# 6. Visualization

If your plots do not show, include this in any cell. The `%` signals that this is an ipython [magic](#). This one controls `matplotlib`. Jupyter uses the [IPython](#) python kernel.

```
%matplotlib inline
```

Today's imports

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

## 6.1. Summarizing Review

We will start with the same dataset we have been working with

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

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

Is the robust coffee's `Mouthfeel` or the `Aftertaste` more consistently scored in this dataset?

Why?

```
robusta_df[['Mouthfeel', 'Aftertaste']].describe()
```

	Mouthfeel	Aftertaste
count	28.000000	28.000000
mean	7.506786	7.559643
std	0.725152	0.342469
min	5.080000	6.500000
25%	7.500000	7.397500
50%	7.670000	7.670000
75%	7.830000	7.770000
max	8.250000	7.920000

from the lower `std` we can see that Aftertaste is more consistently rated.

We can also save this subset into a smaller dataframe to work with it more and plot it.

```
rob_ma_df = robusta_df[['Mouthfeel', 'Aftertaste']]
rob_ma_df.head(1)
```

	Mouthfeel	Aftertaste
0	8.25	7.75

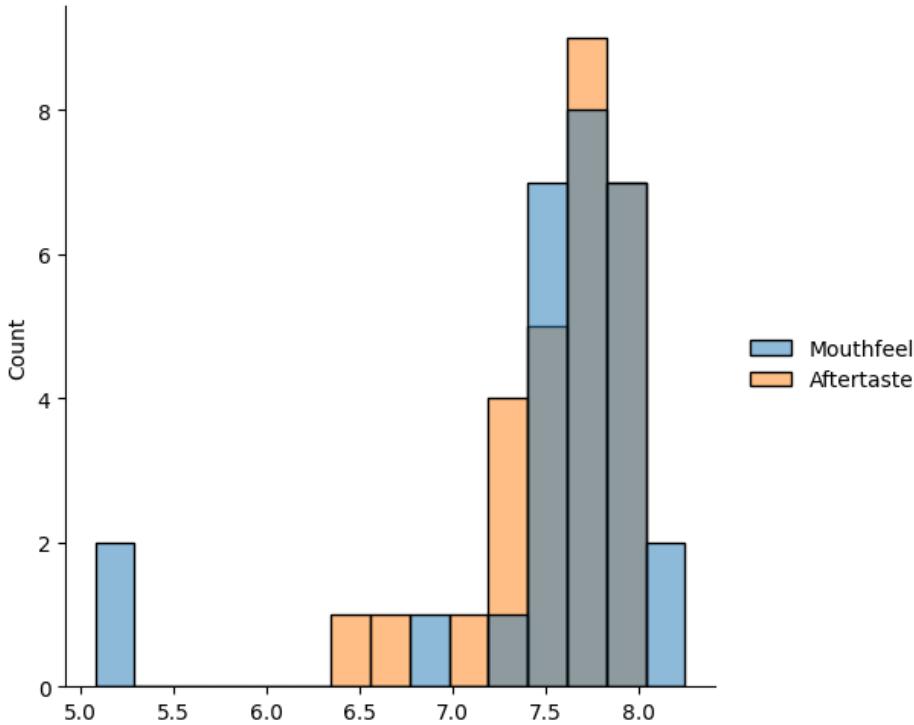
We will use `sns.displot` to look at how the data is distributed.

### ! Important

For `seaborn` the online documentation is **immensely** valuable. Every function's page has basic documentation and lots of examples, so you can see how they use different parameters to modify plots visually. I **strongly recommend reading it often**. I recommend reading [their tutorial](#) too

```
sns.displot(rob_ma_df)
```

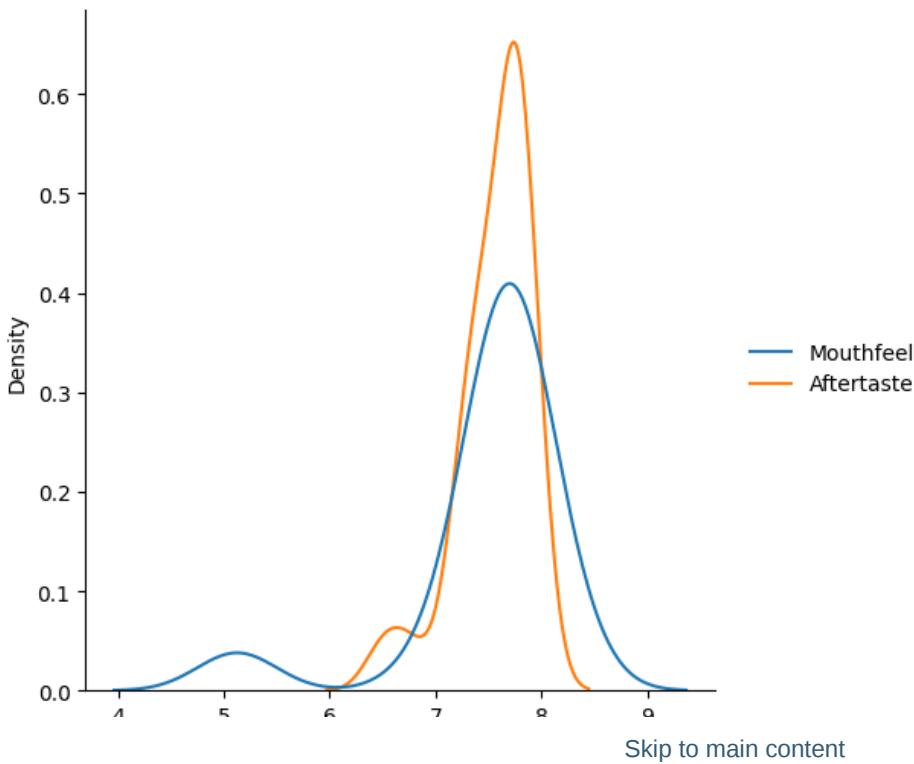
```
<seaborn.axisgrid.FacetGrid at 0x7f34b9c54c40>
```



We can change the kind, for example to a Kernel Density Estimate. This approximates the distribution of the data, you can think of it roughly like a smoothed out histogram.

```
sns.displot(rob_ma_df, kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f34b7b2a0d0>
```



[Skip to main content](#)

This version makes it more visually clear that the the Aftertaste is more consistently, but it also helps us see that that might not be the whole story. Both have a second smaller bump, so the overall std might not be the best measure.

### Question from class

Why do we need two sets of brackets?

It tries to use them to index in multiple ways instead.

```
robusta_df['Aftertaste', 'Mouthfeel']
```

```
-----
KeyError                                 Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3652, in 
    3651     try:
-> 3652         return self._engine.get_loc(casted_key)
    3653     except KeyError as err:
    3654
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pandas._libs.index.IndexEngine.get_loc
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: ('Aftertaste', 'Mouthfeel')

The above exception was the direct cause of the following exception:

KeyError                                 Traceback (most recent call last)
cell In[9], line 1
----> 1 robusta_df['Aftertaste', 'Mouthfeel']

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/frame.py:3760, in DataFrame.__getitem__
    3758     if self.columns.nlevels > 1:
    3759         return self._getitem_multilevel(key)
-> 3760     indexer = self.columns.get_loc(key)
    3761     if is_integer(indexer):
    3762         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3654, in 
    3652     return self._engine.get_loc(casted_key)
    3653     except KeyError as err:
-> 3654         raise KeyError(key) from err
    3655     except TypeError:
    3656         # If we have a listlike key, _check_indexing_error will raise
    3657         # InvalidIndexError. Otherwise we fall through and re-raise
    3658         # the TypeError.
    3659         self._check_indexing_error(key)

KeyError: ('Aftertaste', 'Mouthfeel')
```

It tries to look for a `multiindex`, but we do not have one so it fails. The second square brackets, makes it a list of names to use and pandas looks for them sequentially.

We will use a larger dataset for more interesting plots.

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

[Skip to main content](#)

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

## 6.2. Plotting in Python

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

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.

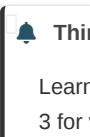
There are lots of type of plots, we saw the basic patterns of how to use them and we've used a few types, but we cannot (and do not need to) go through every single type. There are general patterns that you can use that will help you think about what type of plot you might want and help you understand them to be able to customize plots.

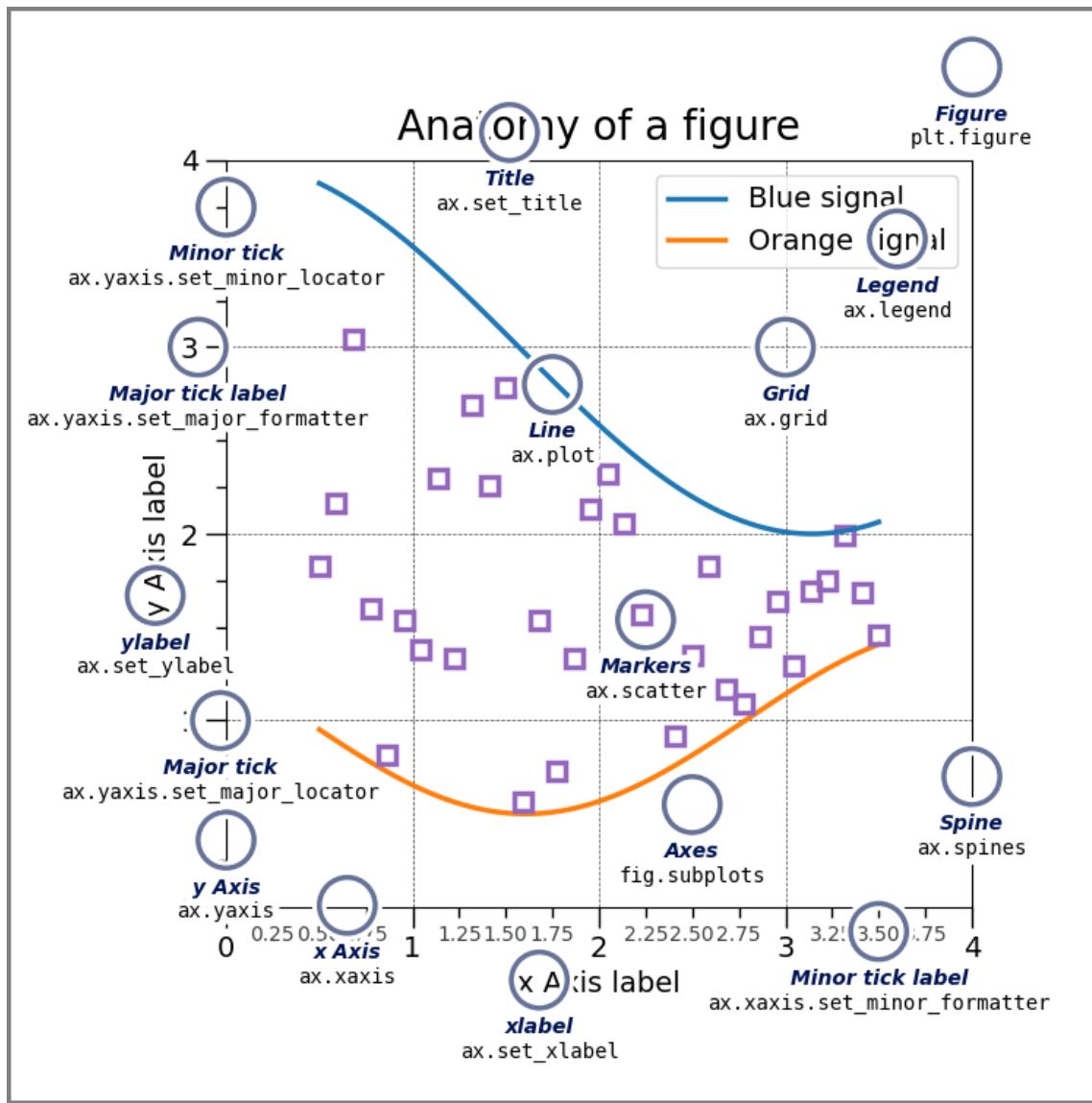
[Seaborn's main goal is opinionated defaults and flexible customization]

(<https://seaborn.pydata.org/tutorial/introduction.html#opinionated-defaults-and-flexible-customization>)

### 6.2.1. Anatomy of a figure

First is the **matplotlib** structure of a figure. BOth pandas and seaborn and other plotting libraries use matplotlib. Matplotlib was used in visualizing the first Black hole.





This is a lot of information, but these are good to know things. THe most important is the figure and the axes.

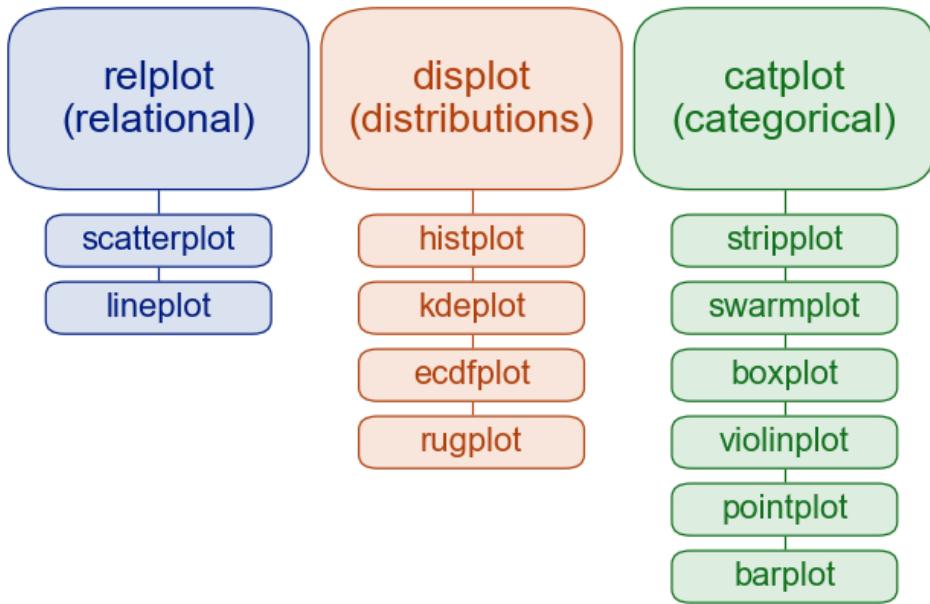
#### Try it Yourself

Make sure you can explain what is a figure and what are axes in your own words and why that distinction matters. Discuss in office hours if you are unsure.

that image was *drawn with code* and that page explains more.

## 6.2.2. Plotting Function types in Seaborn

Seaborn has two *levels* or groups of plotting functions. Figure and axes. Figure level fucntions can plot with subplots.



This is from the overview section of the official seaborn tutorial. It also includes a comparison of [figure vs axes plotting](#).

The [official introduction](#) is also a good read.

### 6.2.3. More

The [seaborn gallery](#) and [matplotlib gallery](#) are nice to look at too.

### 6.2.4. Styling in Seaborn

Seaborn also lets us set a theme for visual styling. This by default styles the plots to be more visually appealing

```
sns.set_theme(palette='colorblind')
```

the `colorblind` palette is more distinguishable under a variety of colorblindness types. [for more](#). Colorblind is a good default, but you can choose others that you like more too.

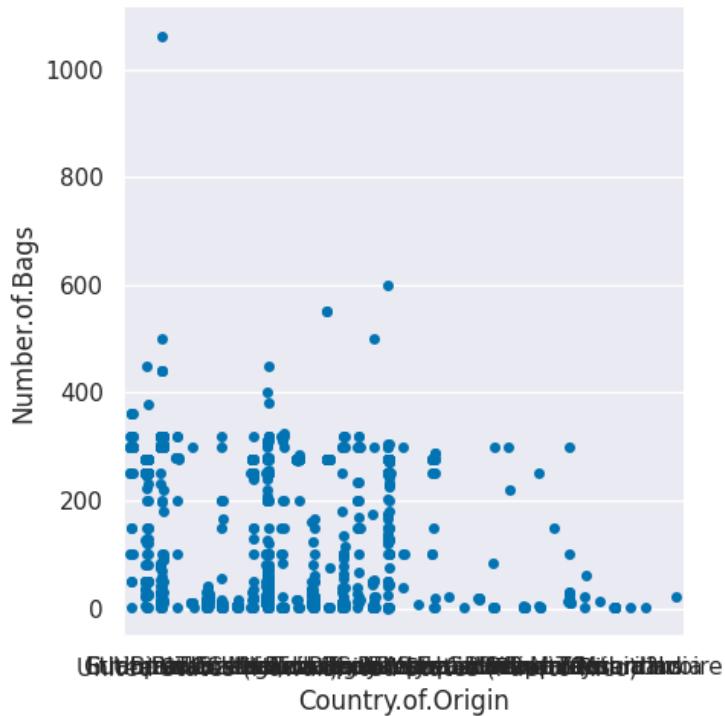
[more on colors](#)

## 6.3. Bags by country

the `catplot` lets us plot vs categorical variables.

```
sns.catplot(data=coffee_df, y='Number.of.Bags', x='Country.of.Origin')
```

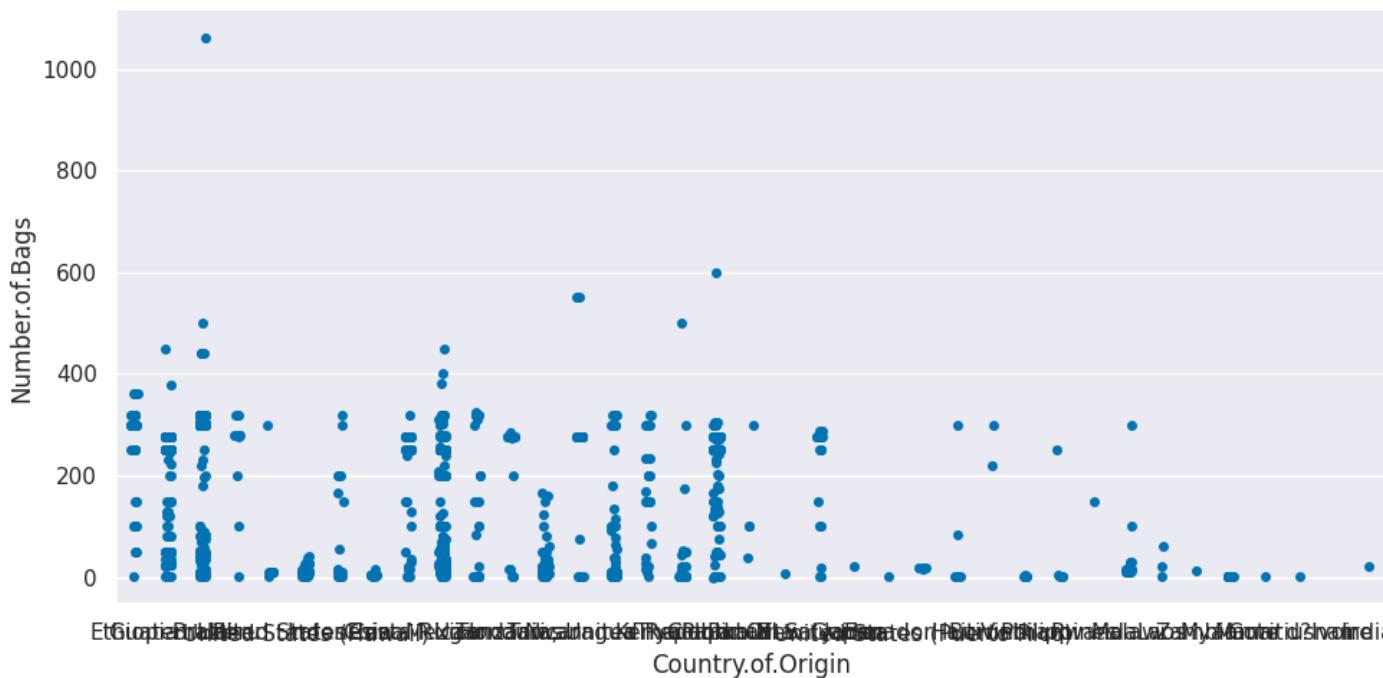
```
<seaborn.axisgrid.FacetGrid at 0x7f34b448a7c0>
```



This is hard to read, we could try stretching it out to make it better

```
sns.catplot(data=coffee_df, y='Number.of.Bags', x='Country.of.Origin', aspect=2)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f34b9bbba90>
```



A better way might be to filter only the top countries. We'll find those by grouping by country then summing each smaller dataframe

[Skip to main content](#)

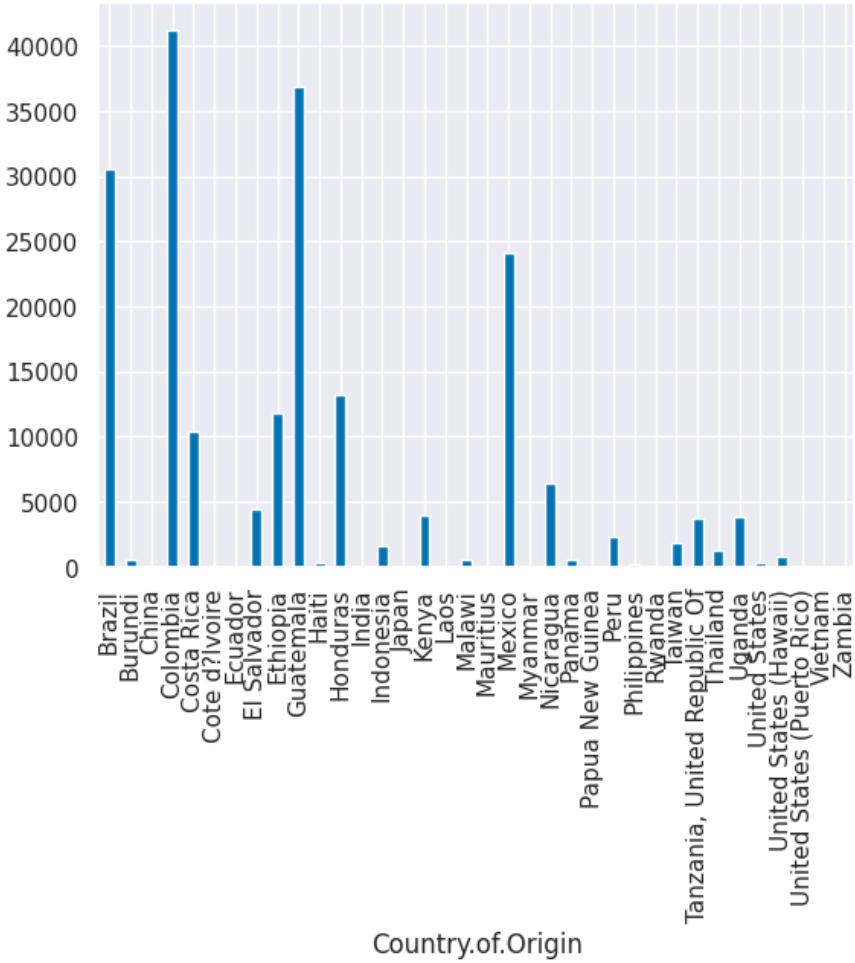
```
tot_per_country = coffee_df.groupby('Country.of-Origin')['Number.of.Bags'].sum()
tot_per_country.head()
```

```
Country.of.Origin
Brazil      30534
Burundi     520
China       55
Colombia    41204
Costa Rica   10354
Name: Number.of.Bags, dtype: int64
```

We can plot this now this way

```
tot_per_country.plot(kind='bar')
```

```
<Axes: xlabel='Country.of.Origin'>
```



What if we take out only the top 10 countries? First we have to sort it. The default is to sort ascending so we use `ascending=False` to switch. pandas doesn't have a plain `sort` method, we have to say if we want to sort by the values or the index. In this Series, the total number per for each country are the values and the country names are the index.

```
tot_per_country.sort_values(ascending=False)[:10]
```

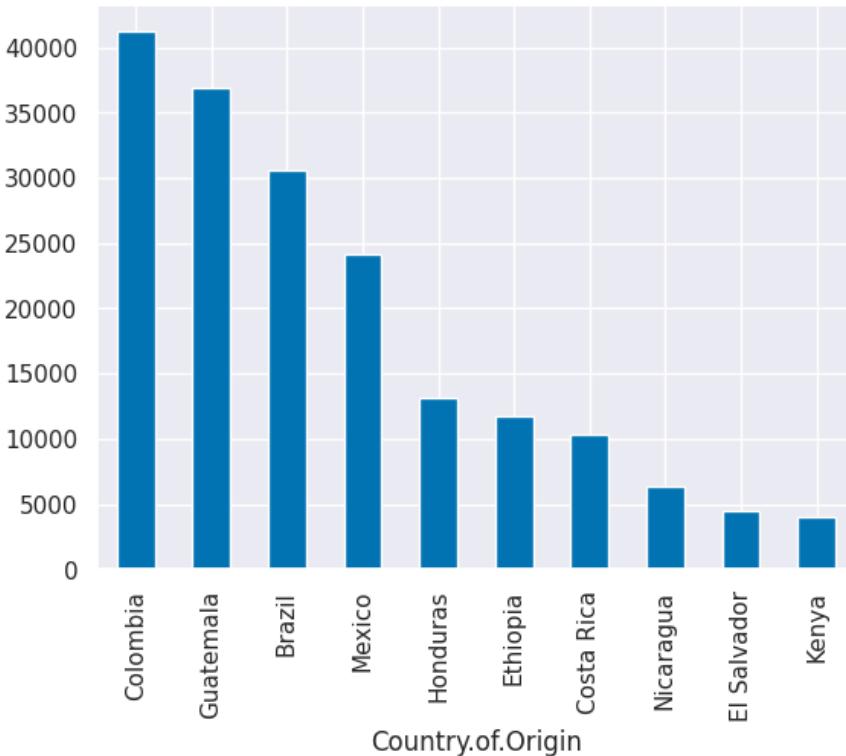
[Skip to main content](#)

```
Country.of.Origin
Colombia      41204
Guatemala    36868
Brazil        30534
Mexico        24140
Honduras      13167
Ethiopia      11761
Costa Rica    10354
Nicaragua     6406
El Salvador   4449
Kenya         3971
Name: Number.of.Bags, dtype: int64
```

We can also plot this

```
tot_per_country.sort_values(ascending=False)[:10].plot(kind='bar')
```

```
<Axes: xlabel='Country.of.Origin'>
```



## 6.4. Filtering a DataFrame

Now, we'll take just the country names out

```
top_countries = tot_per_country.sort_values(ascending=False)[:10].index
top_countries
```

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

[Skip to main content](#)

and we can use that to filter the original `DataFrame`. To do this, we use `isin` to check each element in the `'Country.of.Origin'` column is in that list.

```
coffee_df['Country.of.Origin'].isin(top_countries)
```

```
0      True
1      True
2      True
3      True
4      True
...
1306    True
1307    False
1308    True
1309    True
1310    True
Name: Country.of.Origin, Length: 1311, dtype: bool
```

This is roughly equivalent to:

```
[country in top_countries for country in coffee_df['Country.of.Origin']]
```

```
[True,  
 True,  
 True,  
 True,  
 True,  
 True,  
 False,  
 True,  
 True,  
 True,  
 True,  
 False,  
 False,  
 False,  
 True,  
 False,  
 False,  
 True,  
 False,  
 True,  
 True,  
 True,  
 True,  
 False,  
 True,  
 True,  
 True,  
 True,  
 False,  
 False,  
 True,  
 True,  
 True,  
 True,  
 False,  
 False,  
 True,  
 True]
```

True,  
True,  
False,  
True,  
False,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
False,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
True,

True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
True,  
False,  
True,  
False,  
True,  
False,  
False,  
True,  
True,  
True,  
False,  
True,  
True,

[Skip to main content](#)

False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
False,  
True,  
True,  
False,  
True,  
False,  
True,  
False

[Skip to main content](#)

```
True,  
True,  
True,  
False,  
True,  
False,  
True,  
False,  
False,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
True
```

```
True,  
True,  
True,  
True,  
False,  
False,  
True,  
True,  
True,  
True,  
False,  
False,  
False,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
False,  
False,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
True
```

```
False,  
False,  
True,  
True,  
False,  
False,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
False,  
True,  
True,  
True,  
False,  
False,  
True,  
False,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
False,  
True,  
True,  
True,  
True,  
False,  
False,  
False,  
True,  
True,
```

True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
False,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,

```
False,
True,
False,
True,
True,
False,
False,
True,
True,
True,
True,
True,
True,
True,
False,
False,
False,
True,
True,
True,
False,
False,
False,
True,
True,
True,
True,
True,
True,
True,
False,
True,
True,
True,
True,
False,
True,
True,
True,
True,
False,
False,
False,
False,
False,
True,
False,
True,
True,
True,
False,
False,
False,
False,
True,
True,
```

True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
False,  
False,  
True,  
False,  
True

```
True,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
False,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
False,  
False,  
False,  
False,  
False,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,
```

True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
True,  
True,  
False,  
False,  
False,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
False,  
True,  
True,  
True

True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
False,  
True,  
False,  
False,  
False,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
True

```
True,  
False,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
False,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
False,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
True,  
...]
```

except this builds a list and the pandas way makes a `pd.Series` object. The Python `in` operator is really helpful to know and pandas offers us an `isin` method to get that type of pattern.

In a more basic programming format this process would be two separate loops worth of work.

```
c_in = []  
# iterate over the country of each rating  
for country in coffee_df['Country.of.Origin']:  
    # make a false temp value  
    cur_search = False  
    # iterate over top countries  
    for tc in top_countries:  
        # flip the value if the current top & rating cofee match  
        if tc==country:  
            cur_search = True  
    # save the result of the search  
    c_in.append(cur_search)
```

### Try it yourself

Run these versions and confirm for yourself that they are the same.

With that list of booleans, we can then mask the original DataFrame. This keeps only the value where the inner quantity is `True`

[Skip to main content](#)

```
top_coffee_df = coffee_df[coffee_df['Country.of.Origin'].isin(top_countries)]
top_coffee_df.head(1)
```

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

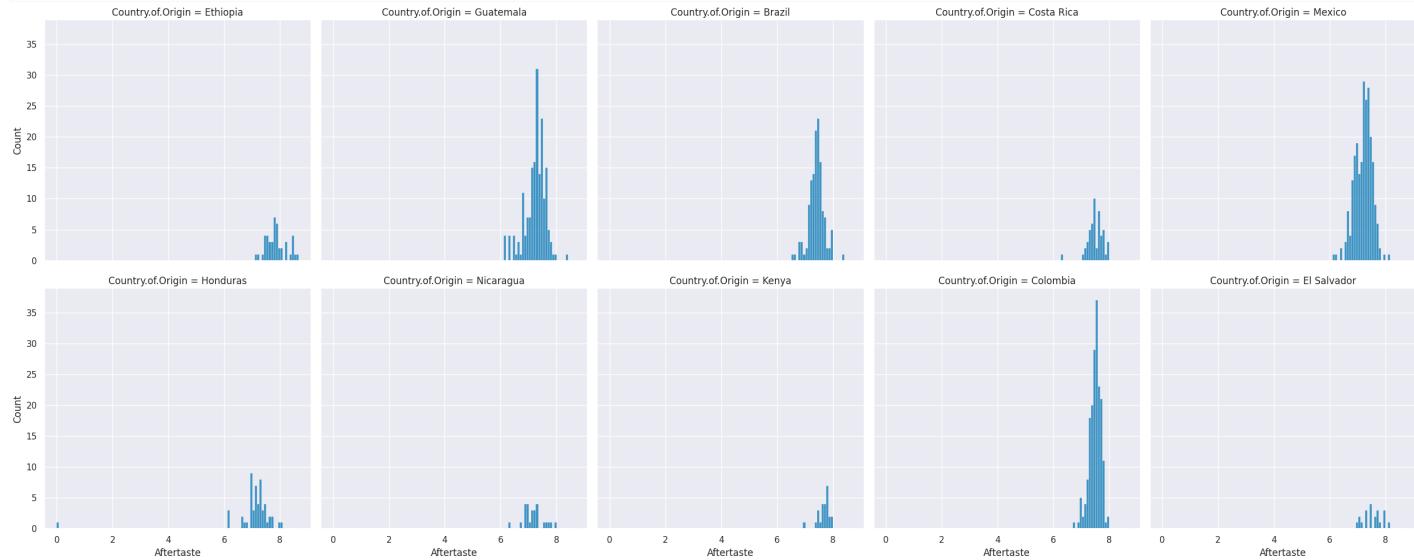
1 rows × 44 columns

```
top_coffee_df.shape, coffee_df.shape
```

```
((952, 44), (1311, 44))
```

```
sns.displot(data=top_coffee_df,x='Aftertaste', col='Country.of.Origin',col_wrap=5)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f34b3ee9670>
```



## 6.5. 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

- 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.

## 6.6. Questions After Class

### 6.6.1. Do we earn level 3's the same way level 1 and 2 are or are there more steps required?

You earn level 3s from your portfolio. The portfolio makes more sense after you have completed assignment 3, so we will follow up on it next week after you all get a3 feedback.

### 6.6.2. How can I check what parameters can go into a method?

You can use the documentation online, or in jupyter, you can get help from the docstring. I usually use shift+tab to read the docstring but you can also use the `help()` function or the `?` in jupyter.

### 6.6.3. How do you know you can put kind = “bar” into the method?

I happen to remember this now, but to know what values you can read the docstring as above.

### 6.6.4. Do companies use things like “sns” for more in depth/graphical plots?

It depends on your role within the company. If you are a data scientist in a more research role you might use seaborn more, but if you build customer facing visualizations, you might use something else.

For more interactive visualization, you could use `plotly` or `bokeh` that generate more javascript for you. Plotly as a company now also has a product called `dash` for building data dashboard apps.

### 6.6.5. Does “component disciplines” mean statistics, computer science and domain expertise, and does “phases” mean collect, clean, explore, model and deploy?

Yes.

## ! Important

I updated the assignment text to clarify in response to some questions

## 7. Tidy Data and Reshaping Datasets

```
import pandas as pd
import seaborn as sns

sns.set_theme(palette='colorblind', font_scale=2)
```

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

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

```
list_of_df = [pd.read_csv(url_base + dataset, na_values='--') for dataset in datasets]
```

```
list_of_df[0]
```

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

```
list_of_df[1]
```

	intervention	John Smith	Jane Doe	Mary Johnson
0	treatmenta	NaN	16	3
1	treatmentsb	2.0	11	1

```
list_of_df[2]
```

	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

```
-----  
ValueError Traceback (most recent call last)  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:1680, in _ensure_numeric()  
1679     try:  
-> 1680         x = x.astype(np.complex128)  
1681     except (TypeError, ValueError):  
  
ValueError: complex() arg is a malformed string  
  
During handling of the above exception, another exception occurred:  
  
ValueError Traceback (most recent call last)  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:1683, in _ensure_float64()  
1682     try:  
-> 1683         x = x.astype(np.float64)  
1684     except ValueError as err:  
1685         # GH#29941 we get here with object arrays containing strs  
  
ValueError: could not convert string to float: 'John SmithJane DoeMary JohnsonJohn SmithJane DoeMary Johnson  
  
The above exception was the direct cause of the following exception:  
  
TypeError Traceback (most recent call last)  
Cell In[7], line 1  
----> 1 list_of_df[2].mean()  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:11563, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)  
11546     @doc()  
11547         _num_doc,  
11548         desc="Return the mean of the values over the requested axis.",  
(...)  
11561         **kwargs,  
11562     ):  
> 11563         return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:11208, in NDFrame._stat_function(self, name, axis, skipna, numeric_only, **kwargs)  
11201     def mean(  
11202         self,  
11203         axis: Axis | None = 0,  
(...)  
11206         **kwargs,  
11207     ) -> Series | float:  
> 11208         return self._stat_function(  
11209             "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs  
11210         )  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:11165, in NDFrame._reduce(self, func, name, axis, skipna, numeric_only)  
11161     nv.validate_stat_func(func, kwargs, fname=name)  
11163 validate_bool_kwarg(skipna, "skipna", none_allowed=False)  
> 11165     return self._reduce(  
11166         func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_only  
11167     )  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/frame.py:10519, in DataFrame._reduce(self, func, name, axis, skipna, numeric_only)  
10515     df = df.T  
10517 # After possibly _get_data and transposing, we are now in the  
10518 # simple case where we can use BlockManager.reduce  
> 10519     res = df._mgr.reduce(bla_func)  
10520     out = df._constructor(res).iloc[0]  
10521     if out.dtype is not None:  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/internals/managers.py:1532, in Manager._reduce(self, func, name, axis, skipna, numeric_only)  
1532     res_blocks: list[Block] = []  
1533     for blk in self.blocks:  
-> 1534         nbs = blk.reduce(func)  
1535         res_blocks.extend(nbs)  
1537     index = Index([None]) # placeholder  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/internals/blocks.py:339,  
333     @final  
334     def reduce(self, func) -> list[Block]:
```

[Skip to main content](#)

```

337     assert self.ndim == 2
--> 339     result = func(self.values)
341     if self.values.ndim == 1:
342         # TODO(EA2D): special case not needed with 2D EAs
343         res_values = np.array([[result]])
344
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/frame.py:10482, in DataFi
10480     return values._reduce(name, skipna=skipna, **kwds)
10481 else:
-> 10482     return op(values, axis=axis, skipna=skipna, **kwds)
10483
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:96, in disallow
94     try:
95         with np.errstate(invalid="ignore"):
--> 96             return f(*args, **kwargs)
97     except ValueError as e:
98         # we want to transform an object array
99         # ValueError message to the more typical TypeError
100        # e.g. this is normally a disallowed function on
101        # object arrays that contain strings
102        if is_object_dtype(args[0]):
102
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:158, in bottler
156     result = alt(values, axis=axis, skipna=skipna, **kwds)
157 else:
--> 158     result = alt(values, axis=axis, skipna=skipna, **kwds)
159
160 return result
161
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:421, in _datetim
418 if datetimelike and mask is None:
419     mask = isna(values)
--> 420 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
421 if datetimelike:
422     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)
423
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:727, in nanmean
724     dtype_count = dtype
725 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
--> 726 the_sum = _ensure_numeric(values.sum(axis, dtype=dtype_sum))
727 if axis is not None and getattr(the_sum, "ndim", False):
728     count = cast(np.ndarray, count)
729
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:1686, in _ensur
1683     x = x.astype(np.float64)
1684 except ValueError as err:
1685     # GH#29941 we get here with object arrays containing strs
-> 1686     raise TypeError(f"Could not convert {x} to numeric") from err
1687 else:
1688     if not np.any(np.imag(x)):
1688
TypeError: Could not convert ['John SmithJane DoeMary JohnsonJohn SmithJane DoeMary Johnson' 'aaabbb'] to num

```

sum([16,3,2,11,1])/5

6.6

sum([16,3,2,11,1,0])/6

5.5

list\_of\_df[2].groupby('treatment').mean()

[Skip to main content](#)

```

-----
NotImplementedError                                Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1490,
1489 try:
-> 1490     result = self.grouper._cython_operation(
1491         "aggregate",
1492         values,
1493         how,
1494         axis=data.ndim - 1,
1495         min_count=min_count,
1496         **kwargs,
1497     )
1498 except NotImplemented:
1499     # generally if we have numeric_only=False
1500     # and non-applicable functions
1501     # try to python agg
1502     # TODO: shouldn't min_count matter?

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:959, in Ba
958 ngroups = self.ngroups
--> 959 return cy_op.cython_operation(
960     values=values,
961     axis=axis,
962     min_count=min_count,
963     comp_ids=ids,
964     ngroups=ngroups,
965     **kwargs,
966 )

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:657, in Wi
649     return self._ea_wrap_cython_operation(
650         values,
651         min_count=min_count,
(...),
654         **kwargs,
655     )
--> 657 return self._cython_op_ndim_compat(
658     values,
659     min_count=min_count,
660     ngroups=ngroups,
661     comp_ids=comp_ids,
662     mask=None,
663     **kwargs,
664 )

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:497, in Wi
495     return res.T
--> 497 return self._call_cython_op(
498     values,
499     min_count=min_count,
500     ngroups=ngroups,
501     comp_ids=comp_ids,
502     mask=mask,
503     result_mask=result_mask,
504     **kwargs,
505 )

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:541, in Wi
540 out_shape = self._get_output_shape(ngroups, values)
--> 541 func = self._get_cython_function(self.kind, self.how, values.dtype, is_numeric)
542 values = self._get_cython_vals(values)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:173, in Wi
171 if "object" not in f.__signatures__:
172     # raise NotImplementedError here rather than TypeError later
--> 173     raise NotImplementedError(
174         f"function is not implemented for this dtype: "
175         f"[{how}-{>{len(how)}}, {dtype}-{>{len(dtype)}}]"
176     )
177 return f

```

During handling of the above exception, another exception occurred:

```
ValueError                                Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:1692, in _ensure_float
1691     try:
-> 1692         x = float(x)
1693     except (TypeError, ValueError):
1694         # e.g. "1+1j" or "foo"
```

ValueError: could not convert string to float: 'John SmithJane DoeMary Johnson'

During handling of the above exception, another exception occurred:

```
ValueError                                Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:1696, in _ensure_complex
1695     try:
-> 1696         x = complex(x)
1697     except ValueError as err:
1698         # e.g. "foo"
```

ValueError: complex() arg is a malformed string

The above exception was the direct cause of the following exception:

```
TypeError                                Traceback (most recent call last)
Cell In[10], line 1
----> 1 list_of_df[2].groupby('treatment').mean()

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1855,
1853     return self._numba_agg_general(sliding_mean, engine_kwargs)
1854 else:
-> 1855     result = self._cython_agg_general(
1856         "mean",
1857         alt=lambda x: Series(x).mean(numeric_only=numeric_only),
1858         numeric_only=numeric_only,
1859     )
1860     return result.__finalize__(self.obj, method="groupby")
```

```
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1507,
1503     result = self._agg_py_fallback(values, ndim=data.ndim, alt=alt)
1505     return result
-> 1507 new_mgr = data.groupby_reduce(array_func)
1508 res = self._wrap_agged_manager(new_mgr)
1509 out = self._wrap_aggregated_output(res)
```

```
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/internals/managers.py:1500,
1499 if blk.is_object:
1500     # split on object-dtype blocks bc some columns may raise
1501     # while others do not.
1502     for sb in blk._split():
-> 1503         applied = sb.apply(func)
1504         result_blocks = extend_blocks(applied, result_blocks)
1505 else:
```

```
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/internals/blocks.py:329,
323 @final
324 def apply(self, func, **kwargs) -> list[Block]:
325     """
326     apply the function to my values; return a block if we are not
327     one
328     """
-> 329     result = func(self.values, **kwargs)
331     return self._split_op_result(result)
```

```
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1503,
1490     result = self.grouper._cython_operation(
1491         "aggregate",
1492         values,
1493         **kwargs,
1494     )
1498 except NotImplementedError:
```

[Skip to main content](#)

```

1501     # try to python agg
1502     # TODO: shouldn't min_count matter?
-> 1503     result = self._agg_py_fallback(values, ndim=data.ndim, alt=alt)
1505 return result

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1457,
1452     ser = df.iloc[:, 0]
1454 # We do not get here with UDFs, so we know that our dtype
1455 # should always be preserved by the implemented aggregations
1456 # TODO: Is this exactly right; see WrappedCythonOp get_result_dtype?
-> 1457 res_values = self.grouper.agg_series(ser, alt, preserve_dtype=True)
1459 if isinstance(values, Categorical):
1460     # Because we only get here with known dtype-preserving
1461     # reductions, we cast back to Categorical.
1462     # TODO: if we ever get "rank" working, exclude it here.
1463     res_values = type(values).from_sequence(res_values, dtype=values.dtype)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:994, in BaseGrouper.aggregate(self, func, *args, **kwargs)
987 if len(obj) > 0 and not isinstance(obj._values, np.ndarray):
988     # we can preserve a little bit more aggressively with EA dtype
989     # because maybe_cast_pointwise_result will do a try/except
990     # with _from_sequence. NB we are assuming here that _from_sequence
991     # is sufficiently strict that it casts appropriately.
992     preserve_dtype = True
--> 994 result = self._aggregate_series_pure_python(obj, func)
996 npvalues = lib.maybe_convert_objects(result, try_float=False)
997 if preserve_dtype:

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:1015, in BaseGrouper._aggregate_series_pure_python(self, obj, func)
1012 splitter = self._get_splitter(obj, axis=0)
1014 for i, group in enumerate(splitter):
-> 1015     res = func(group)
1016     res = lib.reduction.extract_result(res)
1018     if not initialized:
1019         # We only do this validation on the first iteration
1020     validate_result(res, obj)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1857,
1853     return self._numba_agg_general(sliding_mean, engine_kwargs)
1854 else:
1855     result = self._cython_agg_general(
1856         "mean",
-> 1857         alt=lambda x: Series(x).mean(numeric_only=numeric_only),
1858         numeric_only=numeric_only,
1859     )
1860     return result.__finalize__(self.obj, method="groupby")

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:11563, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
11546 @doc(
11547     _num_doc,
11548     desc="Return the mean of the values over the requested axis.",
11549     (...)
11561     **kwargs,
11562 ):
-> 11563     return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:11208, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
11201 def mean(
11202     self,
11203     axis: Axis | None = 0,
11204     (...)
11206     **kwargs,
11207 ) -> Series | float:
-> 11208     return self._stat_function(
11209         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
11210     )

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:11165, in NDFrame._stat_function(self, name, func, args, kwargs)
11161     nv.validate_stat_func(func, kwargs, fname=name)
11163 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
-> 11165 return self._reduce(
11166     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_only
11167 )

```

[Skip to main content](#)

```

4666     raise TypeError(
4667         f"Series.{name} does not allow {kwd_name}={numeric_only} "
4668         "with non-numeric dtypes."
4669     )
4670 with np.errstate(all="ignore"):
-> 4671     return op(delegate, skipna=skipna, **kwds)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:96, in disallow_object_array_if_numeric
94     try:
95         with np.errstate(invalid="ignore"):
--> 96             return f(*args, **kwargs)
97     except ValueError as e:
98         # we want to transform an object array
99         # ValueError message to the more typical TypeError
100        # e.g. this is normally a disallowed function on
101        # object arrays that contain strings
102        if is_object_dtype(args[0]):


File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:158, in bottleneck
156     result = alt(values, axis=axis, skipna=skipna, **kwds)
157 else:
--> 158     result = alt(values, axis=axis, skipna=skipna, **kwds)
160 return result


File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:421, in _datetimelike_functions
418 if datetimelike and mask is None:
419     mask = isna(values)
--> 421 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
423 if datetimelike:
424     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)


File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:727, in nanmean
724     dtype_count = dtype
726 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
--> 727 the_sum = _ensure_numeric(values.sum(axis, dtype=dtype_sum))
729 if axis is not None and getattr(the_sum, "ndim", False):
730     count = cast(np.ndarray, count)


File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:1699, in _ensure_numeric
1696         x = complex(x)
1697     except ValueError as err:
1698         # e.g. "foo"
--> 1699         raise TypeError(f"Could not convert {x} to numeric") from err
1700 return x


TypeError: Could not convert John SmithJane DoeMary Johnson to numeric

```

```
list_of_df[2].groupby('person').mean()
```

```

-----
NotImplementedError                                Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1490,
1489 try:
-> 1490     result = self.grouper._cython_operation(
1491         "aggregate",
1492         values,
1493         how,
1494         axis=data.ndim - 1,
1495         min_count=min_count,
1496         **kwargs,
1497     )
1498 except NotImplemented:
1499     # generally if we have numeric_only=False
1500     # and non-applicable functions
1501     # try to python agg
1502     # TODO: shouldn't min_count matter?

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:959, in Ba
958 ngroups = self.ngroups
--> 959 return cy_op.cython_operation(
960     values=values,
961     axis=axis,
962     min_count=min_count,
963     comp_ids=ids,
964     ngroups=ngroups,
965     **kwargs,
966 )

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:657, in Wi
649     return self._ea_wrap_cython_operation(
650         values,
651         min_count=min_count,
(...),
654         **kwargs,
655     )
--> 657 return self._cython_op_ndim_compat(
658     values,
659     min_count=min_count,
660     ngroups=ngroups,
661     comp_ids=comp_ids,
662     mask=None,
663     **kwargs,
664 )

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:497, in Wi
495     return res.T
--> 497 return self._call_cython_op(
498     values,
499     min_count=min_count,
500     ngroups=ngroups,
501     comp_ids=comp_ids,
502     mask=mask,
503     result_mask=result_mask,
504     **kwargs,
505 )

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:541, in Wi
540 out_shape = self._get_output_shape(ngroups, values)
--> 541 func = self._get_cython_function(self.kind, self.how, values.dtype, is_numeric)
542 values = self._get_cython_vals(values)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:173, in Wi
171 if "object" not in f.__signatures__:
172     # raise NotImplementedError here rather than TypeError later
--> 173     raise NotImplementedError(
174         f"function is not implemented for this dtype: "
175         f"[{how}-{>{len(how)}}, {dtype}-{>{len(dtype)}}]"
176     )
177 return f

```

During handling of the above exception, another exception occurred:

```
ValueError                                Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:1692, in _ensure_float
1691     try:
-> 1692         x = float(x)
1693     except (TypeError, ValueError):
1694         # e.g. "1+1j" or "foo"
```

ValueError: could not convert string to float: 'ab'

During handling of the above exception, another exception occurred:

```
ValueError                                Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:1696, in _ensure_complex
1695     try:
-> 1696         x = complex(x)
1697     except ValueError as err:
1698         # e.g. "foo"
```

ValueError: complex() arg is a malformed string

The above exception was the direct cause of the following exception:

```
TypeError                                Traceback (most recent call last)
Cell In[11], line 1
----> 1 list_of_df[2].groupby('person').mean()

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1855,
1853     return self._numba_agg_general(sliding_mean, engine_kwargs)
1854 else:
-> 1855     result = self._cython_agg_general(
1856         "mean",
1857         alt=lambda x: Series(x).mean(numeric_only=numeric_only),
1858         numeric_only=numeric_only,
1859     )
1860     return result.__finalize__(self.obj, method="groupby")
```

```
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1507,
1503     result = self._agg_py_fallback(values, ndim=data.ndim, alt=alt)
1505     return result
-> 1507 new_mgr = data.groupby_reduce(array_func)
1508 res = self._wrap_agged_manager(new_mgr)
1509 out = self._wrap_aggregated_output(res)
```

```
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/internals/managers.py:1500,
1499 if blk.is_object:
1500     # split on object-dtype blocks bc some columns may raise
1501     # while others do not.
1502     for sb in blk._split():
-> 1503         applied = sb.apply(func)
1504         result_blocks = extend_blocks(applied, result_blocks)
1505 else:
```

```
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/internals/blocks.py:329,
323 @final
324 def apply(self, func, **kwargs) -> list[Block]:
325     """
326     apply the function to my values; return a block if we are not
327     one
328     """
-> 329     result = func(self.values, **kwargs)
331     return self._split_op_result(result)
```

```
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1503,
1490     result = self.grouper._cython_operation(
1491         "aggregate",
1492         values,
1493         **kwargs,
1494     )
1498 except NotImplementedError:
```

```

1501     # try to python agg
1502     # TODO: shouldn't min_count matter?
-> 1503     result = self._agg_py_fallback(values, ndim=data.ndim, alt=alt)
1505 return result

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1457,
1452     ser = df.iloc[:, 0]
1454 # We do not get here with UDFs, so we know that our dtype
1455 # should always be preserved by the implemented aggregations
1456 # TODO: Is this exactly right; see WrappedCythonOp get_result_dtype?
-> 1457 res_values = self.grouper.agg_series(ser, alt, preserve_dtype=True)
1459 if isinstance(values, Categorical):
1460     # Because we only get here with known dtype-preserving
1461     # reductions, we cast back to Categorical.
1462     # TODO: if we ever get "rank" working, exclude it here.
1463     res_values = type(values).from_sequence(res_values, dtype=values.dtype)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:994, in BaseGrouper.aggregate(self, func, *args, **kwargs)
987 if len(obj) > 0 and not isinstance(obj._values, np.ndarray):
988     # we can preserve a little bit more aggressively with EA dtype
989     # because maybe_cast_pointwise_result will do a try/except
990     # with _from_sequence. NB we are assuming here that _from_sequence
991     # is sufficiently strict that it casts appropriately.
992     preserve_dtype = True
--> 994 result = self._aggregate_series_pure_python(obj, func)
996 npvalues = lib.maybe_convert_objects(result, try_float=False)
997 if preserve_dtype:

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:1015, in BaseGrouper._aggregate_series_pure_python(self, obj, func)
1012 splitter = self._get_splitter(obj, axis=0)
1014 for i, group in enumerate(splitter):
-> 1015     res = func(group)
1016     res = lib.reduction.extract_result(res)
1018     if not initialized:
1019         # We only do this validation on the first iteration
1020     validate_result(res, obj)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1857,
1853     return self._numba_agg_general(sliding_mean, engine_kwargs)
1854 else:
1855     result = self._cython_agg_general(
1856         "mean",
-> 1857         alt=lambda x: Series(x).mean(numeric_only=numeric_only),
1858         numeric_only=numeric_only,
1859     )
1860     return result.__finalize__(self.obj, method="groupby")

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:11563, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
11546 @doc(
11547     _num_doc,
11548     desc="Return the mean of the values over the requested axis.",
11549     (...)
11561     **kwargs,
11562 ):
-> 11563     return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:11208, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
11201 def mean(
11202     self,
11203     axis: Axis | None = 0,
11204     (...)
11206     **kwargs,
11207 ) -> Series | float:
-> 11208     return self._stat_function(
11209         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
11210     )

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:11165, in NDFrame._stat_function(self, name, func, args, kwargs)
11161     nv.validate_stat_func(func, kwargs, fname=name)
11163 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
-> 11165 return self._reduce(
11166     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_only
11167 )

```

[Skip to main content](#)

```

4666     raise TypeError(
4667         f"Series.{name} does not allow {kwd_name}={numeric_only} "
4668         "with non-numeric dtypes."
4669     )
4670 with np.errstate(all="ignore"):
-> 4671     return op(delegate, skipna=skipna, **kwds)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:96, in disallow_object_array_if_numeric
94     try:
95         with np.errstate(invalid="ignore"):
--> 96             return f(*args, **kwargs)
97     except ValueError as e:
98         # we want to transform an object array
99         # ValueError message to the more typical TypeError
100        # e.g. this is normally a disallowed function on
101        # object arrays that contain strings
102        if is_object_dtype(args[0]):
```

```

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:158, in bottleneck
156     result = alt(values, axis=axis, skipna=skipna, **kwds)
157 else:
--> 158     result = alt(values, axis=axis, skipna=skipna, **kwds)
160 return result
```

```

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:421, in _datetimelike
418 if datetimelike and mask is None:
419     mask = isna(values)
--> 421 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
423 if datetimelike:
424     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)
```

```

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:727, in nanmean
724     dtype_count = dtype
726 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
--> 727 the_sum = _ensure_numeric(values.sum(axis, dtype=dtype_sum))
729 if axis is not None and getattr(the_sum, "ndim", False):
730     count = cast(np.ndarray, count)
```

```

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/nanops.py:1699, in _ensure_numeric
1696     x = complex(x)
1697     except ValueError as err:
1698         # e.g. "foo"
--> 1699         raise TypeError(f"Could not convert {x} to numeric") from err
1700 return x
```

TypeError: Could not convert ab to numeric

```
dfa = list_of_df[0]
dfa
```

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

```
dfa.melt(id_vars=['name'], var_name='treatment', value_name='result')
```

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

```
arabica_data_url = 'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/arabica_data.csv'
# load the data
coffee_df = pd.read_csv(arabica_data_url)
# get total bags per country
bags_per_country = coffee_df.groupby('Country.of-Origin')['Number.of.Bags'].sum()

# 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)]
```

bags\_per\_country

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
Kenya	3971
Laos	81
Malawi	557
Mauritius	1
Mexico	24140
Myanmar	10
Nicaragua	6406
Panama	537
Papua New Guinea	7
Peru	2336
Philippines	259
Rwanda	150
Taiwan	1914
Tanzania, United Republic Of	3760
Thailand	1310
Uganda	3868
United States	361
United States (Hawaii)	833
United States (Puerto Rico)	71
Vietnam	10
Zambia	13
Name: Number.of.Bags	dtype: int64

[Skip to main content](#)

```
top_bags_country_list
```

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

```
top_coffee_df.head(1)
```

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

1 rows × 44 columns

```
coffee_df.head(1)
```

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

1 rows × 44 columns

```
coffee_df.shape, top_coffee_df.shape
```

```
((1311, 44), (952, 44))
```

```
top_coffee_df.describe()
```

Unnamed: 0	Number.of.Bags	Aroma	Flavor	Aftertaste	Acidity	Body	Balance
count	952.000000	952.000000	952.000000	952.000000	952.000000	952.000000	952.000000
mean	653.811975	192.073529	7.557468	7.513330	7.379338	7.533172	7.505662
std	378.427772	120.682457	0.400004	0.418425	0.430553	0.403558	0.383316
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	323.750000	50.000000	7.420000	7.330000	7.170000	7.330000	7.330000
50%	659.500000	250.000000	7.580000	7.580000	7.420000	7.500000	7.500000
75%	972.500000	275.000000	7.750000	7.750000	7.580000	7.750000	7.670000
max	1312.000000	1062.000000	8.750000	8.830000	8.670000	8.750000	8.580000

[Skip to main content](#)

```
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')
```

```
ratings_of_interest = ['Aroma', 'Flavor', 'Aftertaste', 'Acidity', 'Body',
                      'Balance', ]
coffe_scores_df = top_coffee_df.melt(id_vars='Country.of-Origin', value_vars=ratings_of_interest,
                                      var_name='rating', value_name='score')
coffe_scores_df.head(1)
```

	Country.of-Origin	rating	score
0	Ethiopia	Aroma	8.67

```
top_coffee_df.melt(id_vars='Country.of-Origin')['variable'].unique()
```

```
array(['Unnamed: 0', 'Species', 'Owner', '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)
```

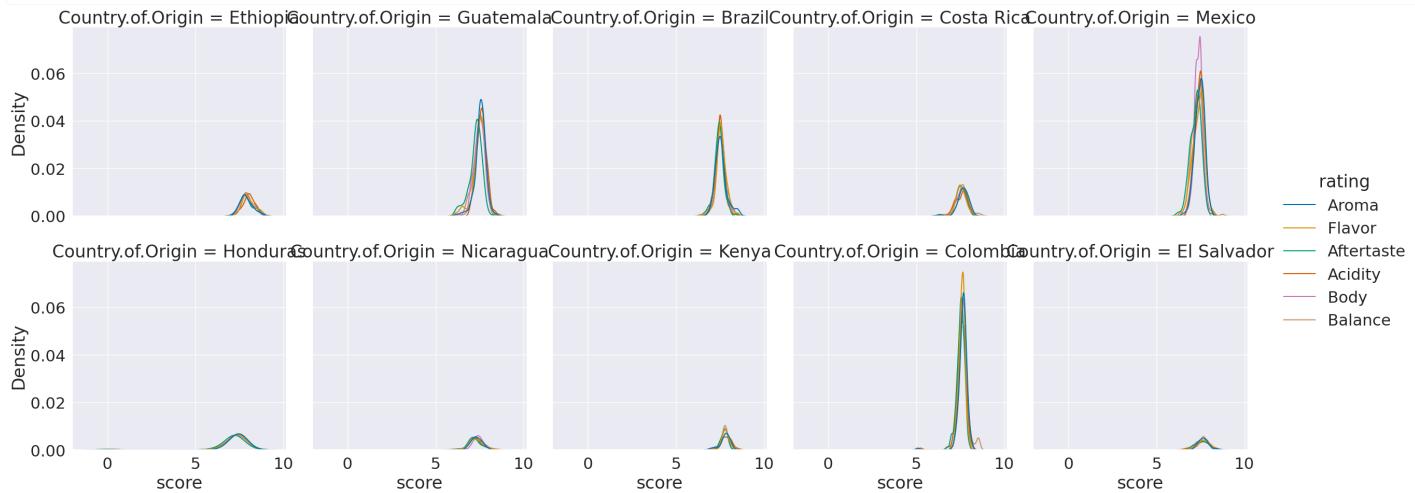
```
top_coffee_df.melt(id_vars='Country.of-Origin', value_vars=ratings_of_interest,)['variable'].unique()
```

```
array(['Aroma', 'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance'],
      dtype=object)
```

```
%matplotlib inline
```

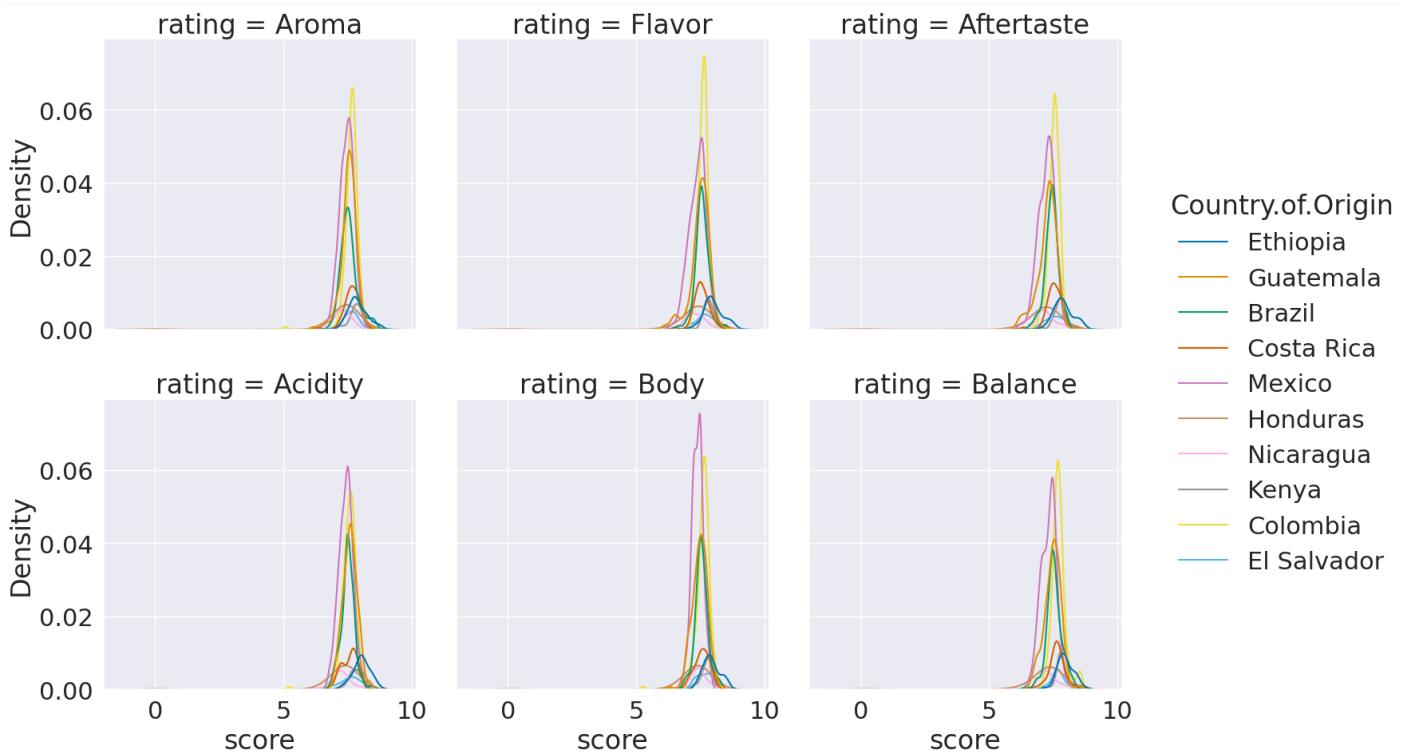
```
sns.displot(data=coffe_scores_df, x='score', col='Country.of-Origin',
             hue = 'rating', col_wrap=5, kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fdf38ea54c0>
```



```
sns.displot(data=coffee_scores_df, x='score', hue='Country.of-Origin',
             col = 'rating', col_wrap=3, kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fdefd417520>
```



```
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', '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')
```

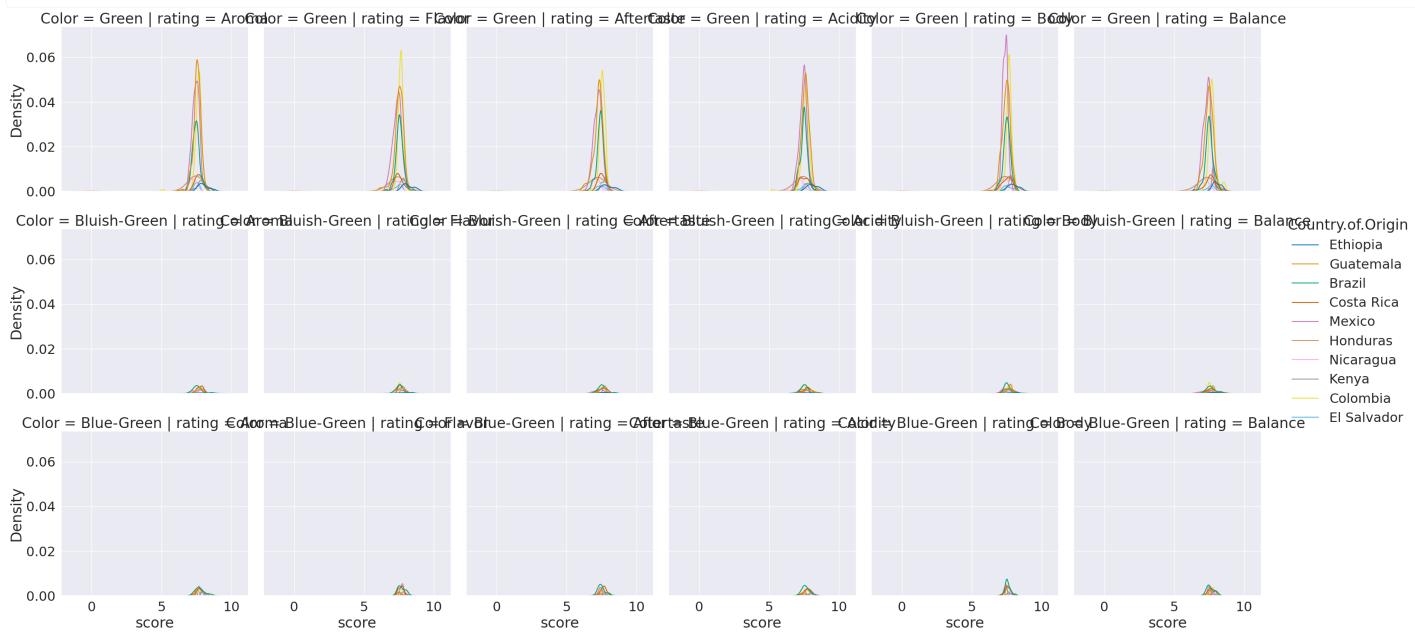
```
coffe_scores_df2= top_coffee_df.melt(id_vars=['Country.of-Origin','Color'],value_vars=ratings_of_interest,  
var_name='rating',value_name='score')  
coffe_scores_df2.head(1)
```

	Country.of-Origin	Color	rating	score
0	Ethiopia	Green	Aroma	8.67

```
sns.displot(data=coffe_scores_df2, x='score',hue='Country.of-Origin',  
col = 'rating',row='Color',kind='kde')
```

```
/tmp/ipykernel_2010/3482930274.py:1: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `w  
sns.displot(data=coffe_scores_df2, x='score',hue='Country.of-Origin',
```

```
<seaborn.axisgrid.FacetGrid at 0x7fdfef822b760>
```



```
coffee_df.describe()
```

	Unnamed: 0	Number.of.Bags	Aroma	Flavor	Aftertaste	Acidity	Body	Bal
<b>count</b>	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000
<b>mean</b>	656.000763	153.887872	7.563806	7.518070	7.397696	7.533112	7.517727	7.517727
<b>std</b>	378.598733	129.733734	0.378666	0.399979	0.405119	0.381599	0.359213	0.400000
<b>min</b>	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
<b>25%</b>	328.500000	14.500000	7.420000	7.330000	7.250000	7.330000	7.330000	7.330000
<b>50%</b>	656.000000	175.000000	7.580000	7.580000	7.420000	7.500000	7.500000	7.500000
<b>75%</b>	983.500000	275.000000	7.750000	7.750000	7.580000	7.750000	7.670000	7.750000
<b>max</b>	1312.000000	1062.000000	8.750000	8.830000	8.670000	8.750000	8.580000	8.750000

## 7.1. More manipulations

Here, we will make a tiny `DataFrame` from scratch to illustrate a couple of points

```
large_num_df = pd.DataFrame(data= [[730000000, 392000000, 580200000],
                                    [315040009, 580000000, 967290000]],
                            columns = ['a', 'b', 'c'])
large_num_df
```

	a	b	c
<b>0</b>	730000000	392000000	580200000
<b>1</b>	315040009	580000000	967290000

This dataet is not tidy, but making it this way was faster to set it up. We could make it tidy using melt as is.

```
large_num_df.melt()
```

	variable	value
<b>0</b>	a	730000000
<b>1</b>	a	315040009
<b>2</b>	b	392000000
<b>3</b>	b	580000000
<b>4</b>	c	580200000
<b>5</b>	c	967290000

However, I want an additional variable, so I wil reset the index, which adds an index column for the original index and adds a new index that is numerical. In this case they're the same.

```
large_num_df.reset_index()
```

	index	a	b	c
0	0	730000000	392000000	580200000
1	1	315040009	580000000	967290000

If I melt this one, using the index as the `id`, then I get a reasonable tidy DataFrame

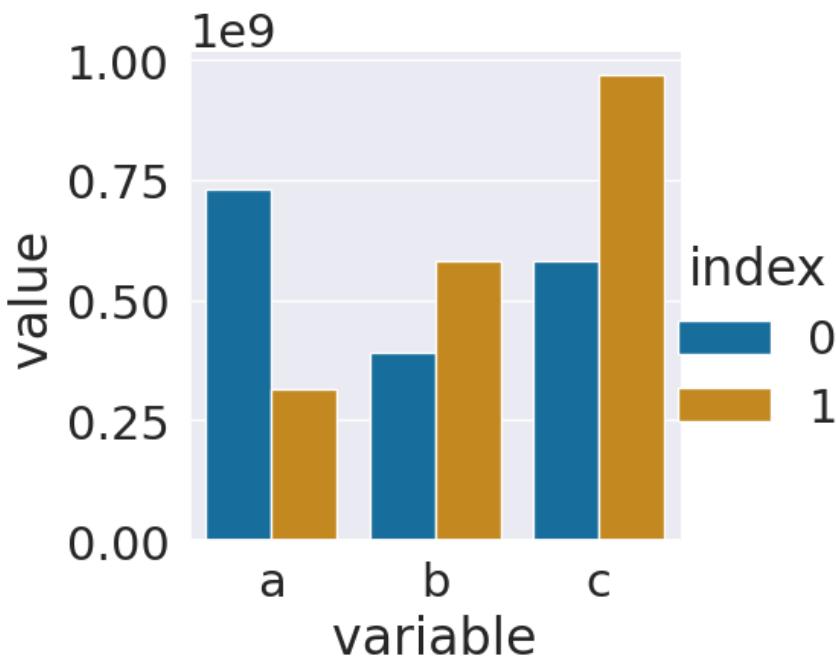
```
ls_tall_df = large_num_df.reset_index().melt(id_vars='index')
ls_tall_df
```

	index	variable	value
0	0	a	730000000
1	1	a	315040009
2	0	b	392000000
3	1	b	580000000
4	0	c	580200000
5	1	c	967290000

Now, we can plot.

```
sns.catplot(data = ls_tall_df,x='variable',y='value',
            hue='index',kind='bar')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fdef7b44a60>
```



Since the numbers are so big, this might be hard to interpret. Displaying it with all the 0s would not be easier to read. The best thing to do is to add a new column with adjusted values and a corresponding title.

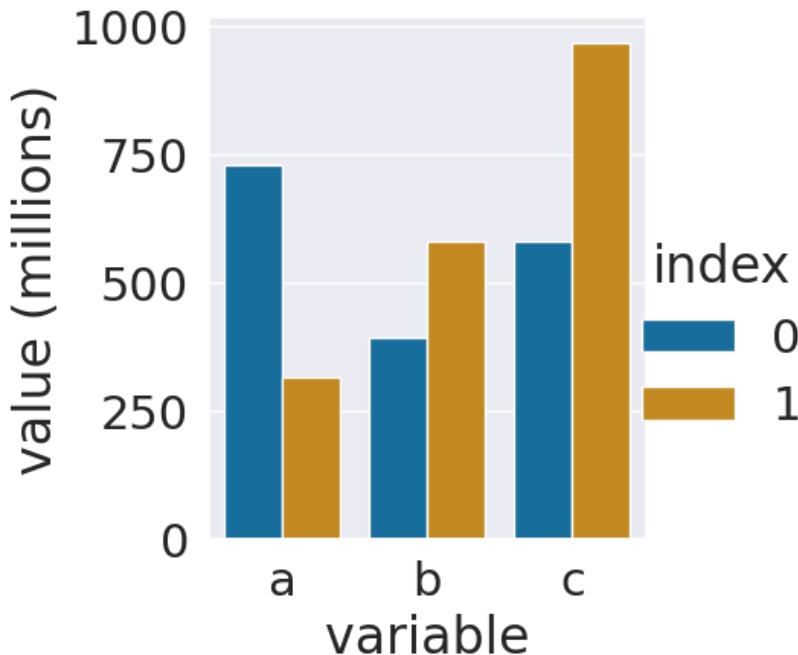
```
ls_tall_df['value (millions)'] = ls_tall_df['value']/1000000
ls_tall_df.head()
```

	index	variable	value	value (millions)
0	0	a	730000000	730.000000
1	1	a	315040009	315.040009
2	0	b	392000000	392.000000
3	1	b	580000000	580.000000
4	0	c	580200000	580.200000

Now we can plot again, with the smaller values and an updated axis label. Adding a column with the adjusted title is good practice because it does not lose any data and since we set the value and the title at the same time it keeps it clear what the values are.

```
sns.catplot(data = ls_tall_df,x='variable',y='value (millions)',
hue='index',kind='bar')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fdef8262040>
```



## 8. Reparing 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

[Skip to main content](#)

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

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 strucutured 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?

```
import pandas as pd
import seaborn as sns
import numpy as np #
na_toy_df_np = pd.DataFrame(data = [[1,3,4,5],[2 ,6, np.nan]])
na_toy_df_pd = pd.DataFrame(data = [[1,3,4,5],[2 ,6, pd.NA]])

# make plots look nicer and increase font size
sns.set_theme(font_scale=2)
arabica_data_url = 'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/arabica_data.csv'

coffee_df = pd.read_csv(arabica_data_url,index_col=0)

rhodyprog4ds_gh_events_url = 'https://api.github.com/orgs/rhodyprog4ds/events'
course_gh_df = pd.read_json(rhodyprog4ds_gh_events_url)
```

## 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"

## 8.3. Fixing Column names

```
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', 'Aroma',
       'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance', 'Uniformity',
       'Clean.Cup', 'Sweetness', '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')
```

```
col_name_mapper = {col_name:col_name.lower().replace('.','_') for col_name in coffee_df.columns}
```

```
coffee_df.rename(columns=col_name_mapper).head(1)
```

	species	owner	country_of_origin	farm_name	lot_number	mill	ico_number	company	altitude	region
1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	guji hambelk

1 rows × 43 columns

```
coffee_df.head(1)
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	Regic
1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	gu hambe

1 rows × 43 columns

```
coffee_df_fixedcols = coffee_df.rename(columns=col_name_mapper)
coffee_df_fixedcols.head(1)
```

	species	owner	country_of_origin	farm_name	lot_number	mill	ico_number	company	altitude	regior
1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	guji hambelk

1 rows × 43 columns

```
coffee_df_fixedcols['unit_of_measurement'].value_counts()
```

```
unit_of_measurement
m      1129
ft      182
Name: count  dtype: int64
```

[Skip to main content](#)

```
coffee_df_fixedcols['unit_of_measurement'].replace({'m':'meters', 'ft':'feet'})
```

```
1      meters
2      meters
3      meters
4      meters
5      meters
...
1307    meters
1308    meters
1309    meters
1310      feet
1312    meters
Name: unit_of_measurement, Length: 1311, dtype: object
```

```
coffee_df_fixedcols['unit_of_measurement_long'] = coffee_df_fixedcols['unit_of_measurement'].replace(
    {'m':'meters', 'ft':'feet'})
coffee_df_fixedcols.head(1)
```

	species	owner	country_of_origin	farm_name	lot_number	mill	ico_number	company	altitude	region
1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural development plc	1950- 2200	guji hambelk

1 rows × 11 columns

## 8.4. Missing Values

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 are om Pandas gives a few basic tools:

- dropna
- fillna

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 coffee from one country, or even just missing more often in one country, that could bias our results.

[Skip to main content](#)

Filling can be good if you know how to fill reasonably, but don't have data to spare by dropping. For example

- 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**

```
coffee_df_fixedcols.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1311 entries, 1 to 1312
Data columns (total 44 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   species          1311 non-null   object  
 1   owner             1304 non-null   object  
 2   country_of_origin 1310 non-null   object  
 3   farm_name         955 non-null   object  
 4   lot_number        270 non-null   object  
 5   mill              1001 non-null   object  
 6   ico_number        1163 non-null   object  
 7   company            1102 non-null   object  
 8   altitude           1088 non-null   object  
 9   region             1254 non-null   object  
 10  producer           1081 non-null   object  
 11  number_of_bags    1311 non-null   int64  
 12  bag_weight         1311 non-null   object  
 13  in_country_partner 1311 non-null   object  
 14  harvest_year       1264 non-null   object  
 15  grading_date       1311 non-null   object  
 16  owner_1             1304 non-null   object  
 17  variety             1110 non-null   object  
 18  processing_method  1159 non-null   object  
 19  aroma               1311 non-null   float64
 20  flavor              1311 non-null   float64
 21  aftertaste          1311 non-null   float64
 22  acidity             1311 non-null   float64
 23  body                1311 non-null   float64
 24  balance             1311 non-null   float64
 25  uniformity          1311 non-null   float64
 26  clean_cup            1311 non-null   float64
 27  sweetness            1311 non-null   float64
 28  copper_points        1311 non-null   float64
 29  total_cup_points     1311 non-null   float64
 30  moisture             1311 non-null   float64
 31  category_one_defects 1311 non-null   int64  
 32  quakers             1310 non-null   float64
 33  color                1044 non-null   object  
 34  category_two_defects 1311 non-null   int64  
 35  expiration            1311 non-null   object  
 36  certification_body    1311 non-null   object  
 37  certification_address 1311 non-null   object  
 38  certification_contact 1311 non-null   object  
 39  unit_of_measurement    1311 non-null   object  
 40  altitude_low_meters   1084 non-null   float64
 41  altitude_high_meters 1084 non-null   float64
 42  altitude_mean_meters 1084 non-null   float64
 43  unit_of_measurement_long 1311 non-null   object  
dtypes: float64(16), int64(3), object(25)
memory usage: 460.9+ KB
```

### 8.4.1. Filling missing values

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

We can look at the type:

```
coffee_df_fixedcols['lot_number'].dtype
```

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

And we can look at the value counts.

```
coffee_df_fixedcols['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_fixedcols['lot_number'].fillna('1')
```

```
1                  1
2                  1
3                  1
4                  1
5                  1
.
1307                1
1308                1
1309    017-053-0211/ 017-053-0212
1310                1
1312                103
Name: lot_number, Length: 1311, 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 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. It's under discussion to deprecate
- write to a new DataFrame
- add a column

We'll use adding a column:

```
coffee_df_fixedcols['lot_number_clean'] = coffee_df_fixedcols['lot_number'].fillna('1')
```

```
coffee_df_fixedcols['lot_number_clean'].value_counts()
```

```
lot_number_clean
1                    1059
020/17                  6
019/17                  5
102                     3
103                     3
...
3-59-2318                1
8885                     1
5055                     1
MCCFWXA15/16                1
017-053-0211/ 017-053-0212    1
Name: count, Length: 221, dtype: int64
```

## 8.4.2. Dropping missing values

To illustrate how `dropna` works, we'll use the `shape` method:

```
coffee_df_fixedcols.shape
```

```
(1311, 45)
```

```
coffee_df_fixedcols.dropna().shape
```

```
(130, 45)
```

By default, it drops any row with one or more `NaN` values.

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

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

```
(1084, 45)
```

```
coffee_alt_df = coffee_df_fixedcols.dropna(subset=['altitude_low_meters'])
```

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, n_cols = coffee_df_fixedcols.shape  
coffee_df_fixedcols.dropna(thresh = .7*n_rows, axis=1).shape
```

```
(1311, 44)
```

This dataset is actually in pretty good shape, but if we use a more stringent threshold it drops more columns.

```
coffee_df_fixedcols.dropna(thresh = .85*n_rows, axis=1).shape
```

```
(1311, 34)
```

## 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_fixedcols['in_country_partner'].value_counts().sort_index()
```

in_country_partner	
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	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
Specialty Coffee Association	295
Specialty Coffee Association of Costa Rica	42
Specialty Coffee Association of Indonesia	10
Specialty Coffee Institute of Asia	16
Tanzanian Coffee Board	6
Torch Coffee Lab Yunnan	2
Uganda Coffee Development Authority	22
Yunnan Coffee Exchange	12
Name: count, dtype: int64	

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

[Skip to main content](#)

```
partner_corrections = {'Blossom Valley International\n':'Blossom Valley International',
 'Specialty Coffee Ass':'Specialty Coffee Association'}
```

```
coffee_df_clean = coffee_df_fixedcols.replace(partner_corrections)
```

## 8.6. Example: Unpacking Jsons

```
rhodyprog4ds_gh_events_url
```

```
'https://api.github.com/orgs/rhodyprog4ds/events'
```

```
gh_df = pd.read_json(rhodyprog4ds_gh_events_url)
gh_df.head()
```

		<b>id</b>	<b>type</b>	<b>actor</b>	<b>repo</b>	<b>payload</b>	<b>public</b>	<b>created_at</b>	
0	28508702433	PushEvent		{'id': 10656079, 'login': 'brownsarahm', 'disp...}	{'id': 592944632, 'name': 'rhodyprog4ds/BrownS...}	{'repository_id': 592944632, 'push_id': 133481...}	True	2023-04-19 01:02:22+00:00	{'id': 'rhod
1	28508694580	ReleaseEvent		{'id': 10656079, 'login': 'brownsarahm', 'disp...}	{'id': 592944632, 'name': 'rhodyprog4ds/BrownS...}	{'action': 'published', 'release': {'url': 'ht...}}	True	2023-04-19 01:01:45+00:00	{'id': 'rhod
2	28508680588	CreateEvent		{'id': 10656079, 'login': 'brownsarahm', 'disp...}	{'id': 592944632, 'name': 'rhodyprog4ds/BrownS...}	{'ref': 'c21', 'ref_type': 'tag', 'master_branch': null}	True	2023-04-19 01:00:40+00:00	{'id': 'rhod
3	28508676304	PushEvent		{'id': 10656079, 'login': 'brownsarahm', 'disp...}	{'id': 592944632, 'name': 'rhodyprog4ds/BrownS...}	{'repository_id': 592944632, 'push_id': 133481...}	True	2023-04-19 01:00:20+00:00	{'id': 'rhod
4	28380142574	PushEvent		{'id': 41898282, 'login': 'github-actions[bot]...')}	{'id': 592944632, 'name': 'rhodyprog4ds/BrownS...}	{'repository_id': 592944632, 'push_id': 132810...}	True	2023-04-13 01:55:52+00:00	{'id': 'rhod

Some datasets have a nested structure

We want to transform each one of those from a dictionary like thing into a row in a data frame.

We can see each row is a Series type.

```
type(gh_df.loc[0])
```

```
a= '1'  
type(a)
```

```
str
```

Recall, that base python types can be used as function, to cast an object from type to another.

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

```
int
```

This works with Pandas Series too

```
pd.Series(gh_df.loc[0]['actor'])
```

```
id                               10656079  
login                            brownsarahm  
display_login                     brownsarahm  
gravatar_id                      url  
url                           https://api.github.com/users/brownsarahm  
avatar_url           https://avatars.githubusercontent.com/u/10656079?  
dtype: object
```

We can use pandas `apply` to do the same thing to every item in a dataset (over rows or columns as items ) For example

```
gh_df['actor'].apply(pd.Series).head()
```

	<b>id</b>	<b>login</b>	<b>display_login</b>	<b>gravatar_id</b>	<b>url</b>
<b>0</b>	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?
<b>1</b>	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?
<b>2</b>	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?
<b>3</b>	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?
<b>4</b>	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot] https://avatars.githubusercontent.com/u/41898282?

compared to the original:

```
gh_df.head(1)
```

	<b>id</b>	<b>type</b>	<b>actor</b>	<b>repo</b>	<b>payload</b>	<b>public</b>	<b>created_at</b>
<b>0</b>	28508702433	PushEvent	{'id': 10656079, 'login': 'brownsarahm', 'disp...}	{'id': 592944632, 'name': 'rhodyprog4ds/BrownS...'} {'repository_id': 592944632, 'push_id': 133481...}	True	2023-04-19 01:02:22+00:00	{'id': 695 'rhodypr...

[Skip to main content](#)

```
js_cols = ['actor', 'repo', 'payload', 'org']
```

`pd.concat` takes a list of dataframes and puts the together in one DataFrame.

```
pd.concat([gh_df[col].apply(pd.Series) for col in js_cols], axis=1).head()
```

	<b>id</b>	<b>login</b>	<b>display_login</b>	<b>gravatar_id</b>	<b>url</b>
<b>0</b>	10656079	brownsarahm	brownsarahm		<a href="https://api.github.com/users/brownsarahm">https://api.github.com/users/brownsarahm</a> <a href="https://avatars.githubusercontent.com/u/10656079?v=4">https://avatars.githubusercontent.com/u/10656079?v=4</a>
<b>1</b>	10656079	brownsarahm	brownsarahm		<a href="https://api.github.com/users/brownsarahm">https://api.github.com/users/brownsarahm</a> <a href="https://avatars.githubusercontent.com/u/10656079?v=4">https://avatars.githubusercontent.com/u/10656079?v=4</a>
<b>2</b>	10656079	brownsarahm	brownsarahm		<a href="https://api.github.com/users/brownsarahm">https://api.github.com/users/brownsarahm</a> <a href="https://avatars.githubusercontent.com/u/10656079?v=4">https://avatars.githubusercontent.com/u/10656079?v=4</a>
<b>3</b>	10656079	brownsarahm	brownsarahm		<a href="https://api.github.com/users/brownsarahm">https://api.github.com/users/brownsarahm</a> <a href="https://avatars.githubusercontent.com/u/10656079?v=4">https://avatars.githubusercontent.com/u/10656079?v=4</a>
<b>4</b>	41898282	github-actions[bot]	github-actions		<a href="https://api.github.com/users/github-actions[bot]">https://api.github.com/users/github-actions[bot]</a> <a href="https://avatars.githubusercontent.com/u/41898282?v=4">https://avatars.githubusercontent.com/u/41898282?v=4</a>

5 rows × 29 columns

This is close, but a lot of columns have the same name. To fix this we will rename the new columns so that they have the original column name prepended to the new name.

pandas has a rename method for this.

and this is another job for lambdas.

```
pd.concat([gh_df[col].apply(pd.Series).rename(lambda c: '_' .join([c,col])) for col in js_cols], axis=1).head()
```

```

-----
TypeError                                         Traceback (most recent call last)
Cell In[35], line 1
----> 1 pd.concat([gh_df[col].apply(pd.Series).rename(lambda c: '_'.join([c,col])) for col in js_cols],axis=1)

Cell In[35], line 1, in <listcomp>(.0)
----> 1 pd.concat([gh_df[col].apply(pd.Series).rename(lambda c: '_'.join([c,col])) for col in js_cols],axis=1)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/frame.py:5440, in DataFrame.rename(self, mapper, index, columns, axis, copy, inplace, level, errors)
5321     def rename(
5322         self,
5323         mapper: Renamer | None = None,
5324         ...) ...
5325         errors: IgnoreRaise = "ignore",
5326     ) -> DataFrame | None:
5327         """
5328         Rename columns or index labels.
5329
5330     (...)

5331     4 3 6
5332     """
5333
5334     return super().__rename__(
5335         mapper=mapper,
5336         index=index,
5337         columns=columns,
5338         axis=axis,
5339         copy=copy,
5340         inplace=inplace,
5341         level=level,
5342         errors=errors,
5343     )
5344

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/generic.py:1034, in NDFrame._set_axis_nocheck(self, new_index, axis_no, inplace=True, copy=False)
1027     missing_labels = [
1028         label
1029         for index, label in enumerate(replacements)
1030         if indexer[index] == -1
1031     ]
1032     raise KeyError(f"{missing_labels} not found in axis")
-> 1034 new_index = ax._transform_index(f, level=level)
1035 result._set_axis_nocheck(new_index, axis=axis_no, inplace=True, copy=False)
1036 result._clear_item_cache()

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:6204, in Index.__new__(cls, values, name, tupleize_cols, copy)
6202     return type(self).from_arrays(values)
6203 else:
-> 6204     items = [func(x) for x in self]
6205     return Index(items, name=self.name, tupleize_cols=False)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:6204, in Index.__new__(cls, values, name, tupleize_cols, copy)
6202     return type(self).from_arrays(values)
6203 else:
-> 6204     items = [func(x) for x in self]
6205     return Index(items, name=self.name, tupleize_cols=False)

Cell In[35], line 1, in <lambda>(c)
----> 1 pd.concat([gh_df[col].apply(pd.Series).rename(lambda c: '_'.join([c,'actor'])) for col in js_cols],axis=1)

TypeError: sequence item 0: expected str instance, int found

```

```
gh_df['actor'].apply(pd.Series).rename(columns=lambda c: '_'.join([c, 'actor']))
```



```
json_cols_df = pd.concat([gh_df[col].apply(pd.Series).rename(columns=lambda c: '_' .join([c,col])) for col in
```

```
gh_df.columns
```

```
Index(['id', 'type', 'actor', 'repo', 'payload', 'public', 'created_at',  
       'org'],  
      dtype='object')
```

```
json_cols_df.columns
```

```
Index(['id_actor', 'login_actor', 'display_login_actor', 'gravatar_id_actor',  
       'url_actor', 'avatar_url_actor', 'id_repo', 'name_repo', 'url_repo',  
       'repository_id_payload', 'push_id_payload', 'size_payload',  
       'distinct_size_payload', 'ref_payload', 'head_payload',  
       'before_payload', 'commits_payload', 'action_payload',  
       'release_payload', 'ref_type_payload', 'master_branch_payload',  
       'description_payload', 'pusher_type_payload', 'issue_payload', 'id_org',  
       'login_org', 'gravatar_id_org', 'url_org', 'avatar_url_org'],  
      dtype='object')
```

Then we can put the two parts of the data together

```
pd.concat([gh_df[['id', 'type', 'public', 'created_at']], json_cols_df],)
```

	<a href="#">id</a>	<a href="#">type</a>	<a href="#">public</a>	<a href="#">created_at</a>	<a href="#">id_actor</a>	<a href="#">login_actor</a>	<a href="#">display_login_actor</a>	<a href="#">gravatar_id</a>
0	2.850870e+10	PushEvent	True	2023-04-19 01:02:22+00:00	NaN	NaN	NaN	NaN
1	2.850869e+10	ReleaseEvent	True	2023-04-19 01:01:45+00:00	NaN	NaN	NaN	NaN
2	2.850868e+10	CreateEvent	True	2023-04-19 01:00:40+00:00	NaN	NaN	NaN	NaN
3	2.850868e+10	PushEvent	True	2023-04-19 01:00:20+00:00	NaN	NaN	NaN	NaN
4	2.838014e+10	PushEvent	True	2023-04-13 01:55:52+00:00	NaN	NaN	NaN	NaN
5	2.838009e+10	ReleaseEvent	True	2023-04-13 01:51:24+00:00	NaN	NaN	NaN	NaN
6	2.838007e+10	CreateEvent	True	2023-04-13 01:49:36+00:00	NaN	NaN	NaN	NaN
7	2.838006e+10	PushEvent	True	2023-04-13 01:49:22+00:00	NaN	NaN	NaN	NaN
8	2.835099e+10	PushEvent	True	2023-04-12 02:28:24+00:00	NaN	NaN	NaN	NaN
9	2.835091e+10	ReleaseEvent	True	2023-04-12 02:22:35+00:00	NaN	NaN	NaN	NaN
10	2.835088e+10	CreateEvent	True	2023-04-12 02:21:10+00:00	NaN	NaN	NaN	NaN
11	2.835088e+10	PushEvent	True	2023-04-12 02:21:12+00:00	NaN	NaN	NaN	NaN
12	2.834636e+10	PushEvent	True	2023-04-11 21:18:20+00:00	NaN	NaN	NaN	NaN
13	2.834622e+10	PushEvent	True	2023-04-11 21:10:21+00:00	NaN	NaN	NaN	NaN
14	2.833450e+10	PushEvent	True	2023-04-11 13:02:29+00:00	NaN	NaN	NaN	NaN
15	2.833432e+10	PushEvent	True	2023-04-11 12:56:32+00:00	NaN	NaN	NaN	NaN
16	2.826148e+10	PushEvent	True	2023-04-06 23:43:27+00:00	NaN	NaN	NaN	NaN
17	2.826142e+10	PushEvent	True	2023-04-06 23:36:57+00:00	NaN	NaN	NaN	NaN
18	2.820908e+10	PushEvent	True	2023-04-05 01:28:11+00:00	NaN	NaN	NaN	NaN
19	2.820905e+10	ReleaseEvent	True	2023-04-05 01:26:09+00:00	NaN	NaN	NaN	NaN
20	2.820902e+10	CreateEvent	True	2023-04-05 01:22:48+00:00	NaN	NaN	NaN	NaN
21	2.820901e+10	PushEvent	True	2023-04-05 01:22:25+00:00	NaN	NaN	NaN	NaN
22	2.820560e+10	IssuesEvent	True	2023-04-04 21:04:21+00:00	NaN	NaN	NaN	NaN

[Skip to main content](#)

	<b>id</b>	<b>type</b>	<b>public</b>	<b>created_at</b>	<b>id_actor</b>	<b>login_actor</b>	<b>display_login_actor</b>	<b>gravatar_id</b>
23	2.817440e+10	PushEvent	True	2023-04-03 19:24:10+00:00	NaN	NaN	NaN	NaN
24	2.817424e+10	PushEvent	True	2023-04-03 19:16:08+00:00	NaN	NaN	NaN	NaN
25	2.810662e+10	ReleaseEvent	True	2023-03-31 01:37:29+00:00	NaN	NaN	NaN	NaN
26	2.810659e+10	CreateEvent	True	2023-03-31 01:35:28+00:00	NaN	NaN	NaN	NaN
27	2.810659e+10	PushEvent	True	2023-03-31 01:35:09+00:00	NaN	NaN	NaN	NaN
28	2.804051e+10	PushEvent	True	2023-03-28 18:07:55+00:00	NaN	NaN	NaN	NaN
29	2.804047e+10	ReleaseEvent	True	2023-03-28 18:06:10+00:00	NaN	NaN	NaN	NaN
0	NaN	NaN	NaN	NaT	10656079.0	brownsarahm	brownsarahm	
1	NaN	NaN	NaN	NaT	10656079.0	brownsarahm	brownsarahm	
2	NaN	NaN	NaN	NaT	10656079.0	brownsarahm	brownsarahm	
3	NaN	NaN	NaN	NaT	10656079.0	brownsarahm	brownsarahm	
4	NaN	NaN	NaN	NaT	41898282.0	github-actions[bot]	github-actions	

35 rows × 33 columns

and finally save this

```
gh_df_clean = pd.concat([gh_df[['id', 'type', 'public', 'created_at']], json_cols_df], axis=1)
gh_df_clean.head()
```

	<b>id</b>	<b>type</b>	<b>public</b>	<b>created_at</b>	<b>id_actor</b>	<b>login_actor</b>	<b>display_login_actor</b>	<b>gravatar_id</b>
0	28508702433	PushEvent	True	2023-04-19 01:02:22+00:00	10656079.0	brownsarahm	brownsarahm	
1	28508694580	ReleaseEvent	True	2023-04-19 01:01:45+00:00	10656079.0	brownsarahm	brownsarahm	
2	28508680588	CreateEvent	True	2023-04-19 01:00:40+00:00	10656079.0	brownsarahm	brownsarahm	
3	28508676304	PushEvent	True	2023-04-19 01:00:20+00:00	10656079.0	brownsarahm	brownsarahm	
4	28380142574	PushEvent	True	2023-04-13 01:55:52+00:00	41898282.0	github-actions[bot]	github-actions	

5 rows × 33 columns

If we want to analyze this data, this is a good place to save it to disk and start an analysis in separate notebook.

[Skip to main content](#)

```
gh_df_clean.to_csv('gh_events_unpacked.csv')
```

## 8.7. Questions After Class

### 8.7.1. How the apply function works/use cases?

A4 will give you some examples, especially the airline dataset. We will also keep seeing it come up as we manipulate data more.

The `apply` docs have tiny examples that help illustrate what it does and some of how it works. The pandas faq has a section on [apply and similar methods](#) that gives some more use cases.

### 8.7.2. Is there a better way to see how many missing values?

There are lots of ways. All are fine. We used `info` in class because I was trying to use the way we had already seen. Info focuses on how many values are *present* instead of what is missing because it makes more sense in most cases. The more common question is: are there enough values to make decisions with?

If you wanted to get counts of the missing values, you can use the pandas `isna` function. It is a pandas function, the docs say `pandas.isna` not a DataFrame method (which would be described like `pandas.DataFrame.methodname`). This means we use it like

```
value_to_test = 4
pd.isna(value_to_test)
```

False

#### Try it Yourself

pass different values like: `False`, `np.nan` (also `import numpy as np`) and, `pd.NA`, `hello` to this function

```
help(pd.isna)
```

```

Help on function isna in module pandas.core.dtypes.missing:

isna(obj: 'object') -> 'bool | npt.NDArray[np.bool_] | NDFrame'
    Detect missing values for an array-like object.

This function takes a scalar or array-like object and indicates
whether values are missing ('`NaN`' in numeric arrays, '`None`' or '`NaN`'
in object arrays, '`NaT`' in datetimelike).

Parameters
-----
obj : scalar or array-like
    Object to check for null or missing values.

Returns
-----
bool or array-like of bool
    For scalar input, returns a scalar boolean.
    For array input, returns an array of boolean indicating whether each
    corresponding element is missing.

See Also
-----
notna : Boolean inverse of pandas.isna.
Series.isna : Detect missing values in a Series.
DataFrame.isna : Detect missing values in a DataFrame.
Index.isna : Detect missing values in an Index.

Examples
-----
Scalar arguments (including strings) result in a scalar boolean.

>>> pd.isna('dog')
False

>>> pd.isna(pd.NA)
True

>>> pd.isna(np.nan)
True

ndarrays result in an ndarray of booleans.

>>> array = np.array([[1, np.nan, 3], [4, 5, np.nan]])
>>> array
array([[ 1., nan,  3.],
       [ 4.,  5., nan]])
>>> pd.isna(array)
array([[False,  True, False],
       [False, False,  True]])

For indexes, an ndarray of booleans is returned.

>>> index = pd.DatetimeIndex(["2017-07-05", "2017-07-06", None,
...                            "2017-07-08"])
>>> index
DatetimeIndex(['2017-07-05', '2017-07-06', 'NaT', '2017-07-08'],
              dtype='datetime64[ns]', freq=None)
>>> pd.isna(index)
array([False, False,  True, False])

For Series and DataFrame, the same type is returned, containing booleans.

>>> df = pd.DataFrame([['ant', 'bee', 'cat'], ['dog', None, 'fly']])
>>> df
   0      1      2
0  ant    bee   cat
1  dog    None   fly
>>> pd.isna(df)
   0      1      2
0  False  False  False

```

```
>>> pd.isna(df[1])
0    False
1    True
Name: 1, dtype: bool
```

The docstring says that it returns “bool or array-like of bool” but if we go to the website docs that have more examples, we can find out what that it will return a DataFrame if we pass it a DataFrame. Then we can use the [pandas.DataFrame.sum](#) method.

```
pd.isna(coffee_df_clean).sum()
```

```
species                  0
owner                   7
country_of_origin        1
farm_name                356
lot_number              1041
mill                     310
ico_number               148
company                 209
altitude                 223
region                   57
producer                 230
number_of_bags            0
bag_weight                0
in_country_partner         0
harvest_year              47
grading_date                0
owner_1                   7
variety                  201
processing_method          152
aroma                      0
flavor                      0
aftertaste                  0
acidity                      0
body                        0
balance                      0
uniformity                  0
clean_cup                    0
sweetness                    0
cupper_points                0
total_cup_points                0
moisture                      0
category_one_defects          0
quakers                      1
color                      267
category_two_defects          0
expiration                  0
certification_body             0
certification_address           0
certification_contact           0
unit_of_measurement              0
altitude_low_meters            227
altitude_high_meters            227
altitude_mean_meters            227
unit_of_measurement_long           0
lot_number_clean                0
dtype: int64
```

### 8.7.3. in `col_name_mapper = {col_name:col_name.lower().replace('.','_') for col_name in coffee_df.columns}` what is the `{}` for?

This is called a dictionary comprehension. It is very similar to a list comprehension. It is one of the defined ways to build a [dict](#) type object

[Skip to main content](#)

We also saw one when we looked at different types in a previous class.

```
{char:i for i,char in enumerate('abcde')}
```

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4}
```

`enumerate` is a built in function that iterates over items in an iterable type(list-like) and pops the each value paired with its index within the structure.

This way we get each character and it's position. We could use this as follows

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

```
Cell In[47], line 2
  alpha_data = ['a','d','e','c','b',']
               ^
SyntaxError: EOL while scanning string literal
```

## 9. Merging and Databases

### 9.1. Announcements

Assignment 3 is graded, but only today, so A4 is extended until tomorrow so that you can incorporate your A3 feedback into A4 (in particular, extending your EDA if you did not earn level 2 for summarize and visualize in A3).

Your achievement trackers are updated.

Take a few minutes to think about the following questions and make a few notes for yourself wherever you need them to be (a planner, calendar, etc.).

1. What achievements have you earned?
2. Does your tracking repo seem accurate to what you've done? If not, make an issue to ask about it.
3. Are you on track to earn the grade you want in this class?
4. If not, what will you need to do (respond more in class, submit more assignments, use your portfolio to catch up) to get back on track?
5. If you are on track and you want to earn above a B, take a minute to think about your portfolio. (tip: post an idea as an issue to get early feedback and help shaping your idea)

### 9.2. Merging Data

Focus this week is on how to programmatically combine sources of data

We will start by looking at combining multiple tabular data formats and see how to get data from other sources.

```
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 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

```
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 `columns` is an attribute, so it does not need `()`

```
df_19.columns
```

```
Index(['PLAYER_NAME', 'TEAM_ID', 'PLAYER_ID', 'SEASON'], dtype='object')
```

```
df_18.shape, df_19.shape
```

```
((748, 3), (626, 4))
```

### 9.2.1. 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)
```

Note that this has the maximum number of columns (because both had some overlapping columns) and the total number of rows.

### 9.2.2. 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

[Skip to main content](#)

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, it looks for the values in the `PLAYER_ID` column that appear in both DataFrames and combines them into a single row.

```
pd.merge(df_18, df_19, on='PLAYER_ID')
```

	TEAM_ID_x	PLAYER_ID	SEASON_x	PLAYER_NAME	TEAM_ID_y	SEASON_y
0	1610612761	202695	2018	Kawhi Leonard	1610612746	2019
1	1610612761	1627783	2018	Pascal Siakam	1610612761	2019
2	1610612761	201188	2018	Marc Gasol	1610612761	2019
3	1610612763	201188	2018	Marc Gasol	1610612761	2019
4	1610612761	201980	2018	Danny Green	1610612747	2019
...	...	...	...	...	...	...
533	1610612760	203583	2018	Abdul Gaddy	1610612760	2019
534	1610612760	203460	2018	Andre Roberson	1610612760	2019
535	1610612755	203658	2018	Norvel Pelle	1610612755	2019
536	1610612741	1627756	2018	Denzel Valentine	1610612741	2019
537	1610612754	203912	2018	C.J. Wilcox	1610612754	2019

538 rows × 6 columns

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.

```
df_1819_inner = pd.merge(df_18, df_19, on='PLAYER_ID', suffixes=('_2018', '_2019'))
df_1819_inner.shape
```

```
(538, 6)
```

By default, this uses an *inner* merge, so we get the players that are in both datasets only. If we want to see differences, we need another type of merge.

Some players still appear twice, because they were in one of the datasets twice, this happens when a player plays for two teams in one season.

### 9.2.3. Which players played in 2018, but not 2019?

We have different types of merges, inner is both, outer 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.

```
df_1819_outer = pd.merge(df_18, df_19, how='outer', on='PLAYER_ID', suffixes=('_2018', '_2019'), )
df_1819_outer
```

	TEAM_ID_2018	PLAYER_ID	SEASON_2018	PLAYER_NAME	TEAM_ID_2019	SEASON_2019	
0	1.610613e+09	202695	2018.0	Kawhi Leonard	1.610613e+09	2019.0	
1	1.610613e+09	1627783	2018.0	Pascal Siakam	1.610613e+09	2019.0	
2	1.610613e+09	201188	2018.0	Marc Gasol	1.610613e+09	2019.0	
3	1.610613e+09	201188	2018.0	Marc Gasol	1.610613e+09	2019.0	
4	1.610613e+09	201980	2018.0	Danny Green	1.610613e+09	2019.0	
...	...	...	...	...	...	...	
922		NaN	1629097	NaN	Terry Larrier	1.610613e+09	2019.0
923		NaN	203461	NaN	Anthony Bennett	1.610613e+09	2019.0
924		NaN	203906	NaN	Devyn Marble	1.610613e+09	2019.0
925		NaN	1629755	NaN	Hassani Gravett	1.610613e+09	2019.0
926		NaN	1629721	NaN	JaKeenan Gant	1.610613e+09	2019.0

927 rows × 6 columns

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.

df\_1819\_inner.dtypes

```
TEAM_ID_2018      int64
PLAYER_ID        int64
SEASON_2018      int64
PLAYER_NAME      object
TEAM_ID_2019      int64
SEASON_2019      int64
dtype: object
```

df\_1819\_outer.dtypes

```
TEAM_ID_2018      float64
PLAYER_ID        int64
SEASON_2018      float64
PLAYER_NAME      object
TEAM_ID_2019      float64
SEASON_2019      float64
dtype: object
```

nan is a float

```
import numpy as np
type(np.nan)
```

float

```
df_1819left = pd.merge(df_18, df_19, how='left', on='PLAYER_ID', suffixes=('_2018', '_2019'), )
df_18only = df_1819left[df_1819left['SEASON_2019'].isna()]
df_18only
```

	TEAM_ID_2018	PLAYER_ID	SEASON_2018	PLAYER_NAME	TEAM_ID_2019	SEASON_2019
9	1610612761	202391	2018	NaN	NaN	NaN
11	1610612761	201975	2018	NaN	NaN	NaN
18	1610612744	101106	2018	NaN	NaN	NaN
23	1610612744	2733	2018	NaN	NaN	NaN
24	1610612744	201973	2018	NaN	NaN	NaN
...	...	...	...	...	...	...
749	1610612752	1629246	2018	NaN	NaN	NaN
750	1610612748	1629159	2018	NaN	NaN	NaN
751	1610612762	1629163	2018	NaN	NaN	NaN
752	1610612743	1629150	2018	NaN	NaN	NaN
753	1610612738	1629167	2018	NaN	NaN	NaN

216 rows × 6 columns

```
len(df_18only['PLAYER_ID'].unique())
```

178

```
df_18only['PLAYER_ID'].value_counts()
```

```
PLAYER_ID
1629150    4
202325     3
201160     3
1628393    3
202328     3
...
1628515    1
1627816    1
1628979    1
2736       1
1629167    1
Name: count, Length: 178, dtype: int64
```

## 9.3. Getting Data from Databases

### 9.3.1. What is a Database?

A common attitude in Data Science is:

[Skip to main content](#)

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 involving 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**

### 9.3.2. 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 0x7f33c6df8a60>)
```

```
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 0x7f33c6e130a0>
```

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()
```

```
[]
```

We can run another query with `execute` then `fetch` that result. This query gives us the column names.

The schema of a database is the description of its setup and layout. The `*` means to get all.

```
cursor.execute("SELECT * FROM INFORMATION_SCHEMA.COLUMNS ")
```

```
-----
OperationalError                                Traceback (most recent call last)
cell In[25], line 1
----> 1 cursor.execute("SELECT * FROM INFORMATION_SCHEMA.COLUMNS ")

OperationalError: no such table: INFORMATION_SCHEMA.COLUMNS
```

Then we use `fetchall` to get the results of the query.

```
cursor.fetchall()
```

## 9.4. 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

```
pd.read_sql('SELECT * FROM teams', conn).head(1)
```

index	LEAGUE_ID	TEAM_ID	MIN_YEAR	MAX_YEAR	ABBREVIATION	NICKNAME	YEARFOUNDED	CIT
0	0	1610612737	1949	2019	ATL	Hawks	1949	Atlanta

### 9.4.1. 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

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

#### 9.4.2. 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 information on 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 conference.

```
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.unique)
player_counts_1819_team.idxmax()
```

```
('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 compare the two directly:

```
player_counts_1819_team['new', 'new'], player_counts_1819_team['same', 'new']
```

```
(119 125)
```

[Skip to main content](#)

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.

## 9.5. Questions After Class

### 9.5.1. Is there a max DB size?

Generally, no. In specific instances, yes. For example, [MSFT SQL Server](#) has a max size of 524,272 terabytes.

### 9.5.2. when can pandas not use SQL databases?

The most important limit here is realy that the cmoputer you are working on will have limits on how much data you can pull from the database into local RAM.

### 9.5.3. how do you learn more about different quieries you can use for sql?

[Wizard zines](#) has a good reference, but it is not free. I have some of their other work though and it is all high quality. [this preview](#) is especially helpful for me If the cost is prohibitive for you, but the preview of this looks like something you would like, send me an e-mail.

This cheatsheet is also good.

### 9.5.4. What other SQL 'keywords' in the queries are there? ex: SELECT, FROM, WHERE

[quick reference](#)

## 10. Web Scraping

```
import requests  
from bs4 import BeautifulSoup
```

[Skip to main content](#)

## ⚠ Warning

If it says it cannot load one of the libraries, use pip inside your notebook to install, then restart your kernel (Kernel menu, choose restart)

```
pip install beautifulsoup4
```

## 10.1. 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

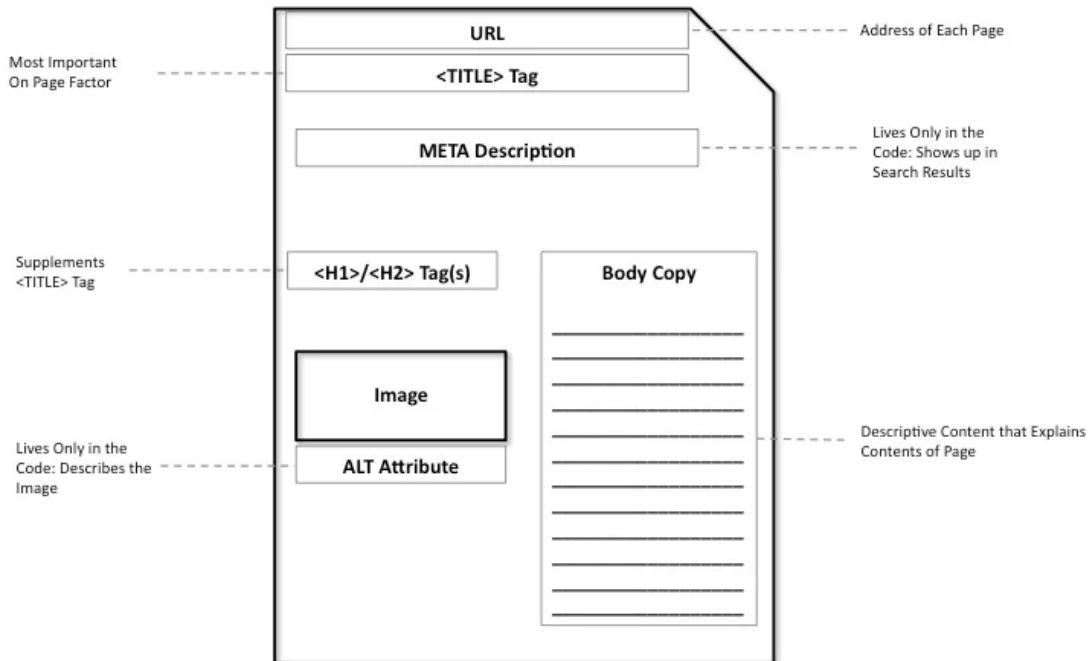
This gives us a list of DataFrames that come from the website. `pandas` gets tables by looking in the html for the site and finding the `<table>` tags.

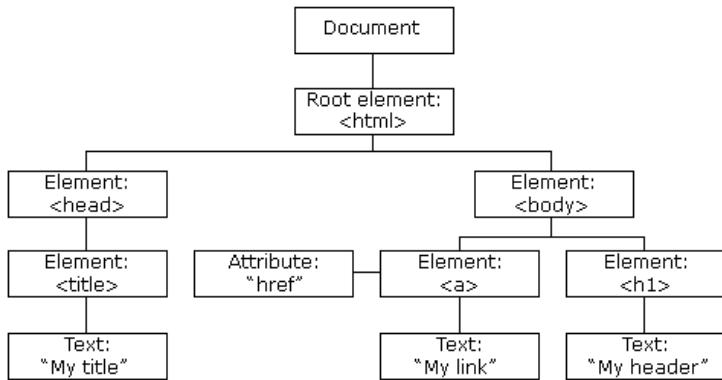
Not

This  
default

## 10.2. Everything is Data

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





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

## 10.3. 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.

### ⚠ 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

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_url = 'https://web.uri.edu/cs/people/'
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<meta charset="UTF-8"><script type="text/javascript">(
```

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

▶ Show code cell output

### 10.3.1. Looking at tags

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

```
cs_people.a
```

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

```
cs_people.div
```

```

<div class="site" id="page">
<a class="skip-link screen-reader-text" href="#content">Skip to content</a>
<div id="masthead">
<header class="site-header" id="brandbar" role="banner">
<div id="identity-print">
<input aria-label="Toggle visibility of the search box." id="gsform-toggle" role="presentation" type="checkbox"
<label for="gsform-toggle" id="gsform"><span>Search</span></label>
<form action="https://www.uri.edu/search" id="gs" method="get" name="global_general_search_form">
<input name="cx" type="hidden" value="016863979916529535900:17qai8akniu">
<input name="cof" type="hidden" value="FORID:11"/>
<label for="gs-query" id="gs-query-label">Searchbox</label>
<input id="gs-query" name="q" placeholder="Search" role="searchbox" type="text" value="" />
<input class="searchsubmit" id="gs-submit" name="searchsubmit" type="submit" value="Search"/>
</input></form>
</div>
<div id="globalbanner-wrapper">
<div id="globalbanner">
<a href="https://www.uri.edu/" title="University of Rhode Island"><div id="identity">University of Rhode Island</div>
<div id="gateways">
<input aria-label="Open the audience gateways menu when browsing on mobile" id="gateways-toggle" role="presentation"
<label for="gateways-toggle" id="gateways-label"><span>You</span></label>
<ul id="gateways-menu" role="menu">
<li><a href="https://www.uri.edu/gateway/future-students" role="menuitem">Future Students</a></li>
<li><a href="https://www.uri.edu/gateway/students" role="menuitem">Students</a></li>
<li><a href="https://www.uri.edu/gateway/faculty" role="menuitem">Faculty</a></li>
<li><a href="https://www.uri.edu/gateway/staff" role="menuitem">Staff</a></li>
<li><a href="https://www.uri.edu/gateway/families" role="menuitem">Parents and Families</a></li>
<li><a href="https://www.uri.edu/gateway/alumni" role="menuitem">Alumni</a></li>
<li><a href="https://www.uri.edu/gateway/community" role="menuitem">Community</a></li>
</ul>
</div>
</div>
</div>
</header><!-- #brandbar -->
<header id="siteheader">
<div class="light" id="sitebanner">
<div id="sb-backdrop">
<div id="sb-background-image" style="background-image:url(<https://web.uri.edu/wp-content/uploads/sites/1531/01/uri-new-light-blue-sky-1920x1080.jpg>)"></div>
<div id="sb-screen"></div>
</div>
<div id="sitebranding">
<div id="siteidentity">
<h1 class="site-title">
<a href="https://web.uri.edu/cs/" rel="home">
                    Department of Computer Science and Statistics </a>
</h1>
<h2 class="site-description">College of Arts and Sciences</h2>
</div>
<div id="sitesocial">
</div>
</div><!-- #sitebranding -->
<div id="sitebanner">
<div class="content-width" id="navigation">
<nav aria-label="Breadcrumb" id="breadcrumbs">
<ol><li><a href="https://www.uri.edu/">URI</a></li><li><a href="https://web.uri.edu/artsci">Arts and Sciences</a></li>
</ol>
<div id="localnav">
<section class="cl-wrapper cl-menu-wrapper"><div class="cl-menu" data-name="Site Menu" data-show-title="0" id="cl-menu">
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8255" id="menu-item-8255"><a href="https://web.uri.edu/cs/">Computer Science and Statistics</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8806" id="menu-item-8806"><a href="https://web.uri.edu/cs/courses">Courses</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page current-menu-item page_item page-item-629 menu-item-10871" id="menu-item-10871"><a href="https://web.uri.edu/cs/courses">Courses</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-9661" id="menu-item-9661"><a href="https://web.uri.edu/cs/courses">Courses</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-10871" id="menu-item-10871"><a href="https://web.uri.edu/cs/courses">Courses</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8267" id="menu-item-8267"><a href="https://web.uri.edu/cs/courses">Courses</a></li>
</ul></div></section></div>
</div>
</header>
</div>
<div class="site-content" id="content">
<main class="site-main" id="main" role="main">
<article class="post-629 page type-page status-publish hentry" id="post-629">

```

[Skip to main content](#)

```
<section class="cl-wrapper cl-menu-wrapper"><div class="cl-menu" data-name="people" data-show-title="0" id="">
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8746" id="menu-item-8746"><a href="https://web.uri.edu/cs/people/marco-alvarez/">Marco Alvarez</a>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-11844" id="menu-item-11844"><a href="https://web.uri.edu/cs/people/samantha-armenti/">Samantha Armenti</a>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-11845" id="menu-item-11845"><a href="https://web.uri.edu/cs/people/sarah-brown/">Sarah Brown</a>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-11846" id="menu-item-11846"><a href="https://web.uri.edu/cs/people/michael-conti/">Michael Conti</a>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-11847" id="menu-item-11847"><a href="https://web.uri.edu/cs/people/noah-daniels/">Noah Daniels</a>
</ul></div></section>
<h2>Full-time Faculty</h2>

<div class="uri-people-tool cl-tiles halves"><div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/marco-alvarez/"><img alt="" class="u-photo wp-post-image" height="120" />
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/marco-alvarez/">Marco Alvarez</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor | Director of Graduate Studies</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5009</span> - <a class="u-email" href="mailto:malvarez@uri.edu">malvarez@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card">
<header>
<div class="header">
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/samantha-armenti/">Samantha Armenti</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Teaching Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:sarmenti@uri.edu">sarmenti@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/sarah-brown/"><img alt="" class="u-photo wp-post-image" height="300" />
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/sarah-brown/">Sarah Brown</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:brownsarahm@uri.edu">brownsarahm@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/michael-conti/"><img alt="" class="u-photo wp-post-image" height="2475" />
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/michael-conti/">Michael Conti</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Teaching Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:michaelconti@uri.edu">michaelconti@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/noah-daniels/"><img alt="" class="u-photo wp-post-image" height="219" />
</figure>
```

```
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:noah_daniels@uri.edu">noah_daniels@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/lisa-dipippo/"><img alt="" class="u-photo wp-post-image" height="126" l</a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/lisa-dipippo/">Lisa DiPippo</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Professor | Chair</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:ldipippo@uri.edu">ldipippo@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/victor-fay-wolfe/"><img alt="" class="u-photo wp-post-image" height="126" l</a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/victor-fay-wolfe/">Victor Fay-Wolfe</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:vfaywolfe@uri.edu">vfaywolfe@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card">
<header>
<div class="header">
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/lutz-hamel/">Lutz Hamel</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor </p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:lutzhamel@uri.edu">lutzhamel@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/abdeljawab-hendawi/"><img alt="" class="u-photo wp-post-image" height="126" l</a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/abdeljawab-hendawi/">Abdeljawab Hendawi</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor</p>
<p class="people-department">Data Science | Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5738</span> - <a class="u-email" href="mailto:hendawi@uri.edu">hendawi@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
```

```
<a href="https://web.uri.edu/cs/meet/jean-yves-herve/"><img alt="" class="u-photo wp-post-image" height="299">
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/jean-yves-herve/">Jean-Yves Hervé</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:jyh@cs.uri.edu">jyh@cs.uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/natallia-katenka/"><img alt="" class="u-photo wp-post-image" height="200">
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/natallia-katenka/">Natallia Katenka</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor | Director of Undergraduate Studies</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><a class="u-email" href="mailto:nkatenka@uri.edu">nkatenka@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card">
<header>
<div class="header">
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/soheyb-kouider/">Soheyb Kouider</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Teaching Professor</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.2562</span> - <a class="u-email" href="mailto:soheyb@uri.edu">soheyb@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/edmund-lamagna/"><img alt="" class="u-photo wp-post-image" height="150">
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/edmund-lamagna/">Edmund Lamagna</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:eal@cs.uri.edu">eal@cs.uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/indrani-mandal/"><img alt="" class="u-photo wp-post-image" height="280">
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/indrani-mandal/">Indrani Mandal</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Teaching Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:indrani_mandal@uri.edu">indrani_mandal@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
```

[Skip to main content](#)

```
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/gavino-puggioni/"><img alt="" class="u-photo wp-post-image" height="160" width="160"/>
</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 | Statistics Section Head | Director of Graduate Studies</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.4388</span> - <a class="u-email" href="mailto:gppuggioni@uri.edu">gppuggioni@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card">
<header>
<div class="header">
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/jonathan-schrader/">Jonathan Schrader</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Teaching Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:jonathan.schrader@uri.edu">jonathan.schrader@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/krishna-venkatasubramanian/"><img alt="" class="u-photo wp-post-image" height="160" width="160"/>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/krishna-venkatasubramanian/">Krishna Venkatasubramanian</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:krish@uri.edu">krish@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/jing-wu/"><img alt="" class="u-photo wp-post-image" height="200" loading="lazy" style="width:100%; height:100%; object-fit: cover;"/>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/jing-wu/">Jing Wu</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.4504</span> - <a class="u-email" href="mailto:jing_wu@uri.edu">jing_wu@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/yichi-zhang/"><img alt="" class="u-photo wp-post-image" height="240" loading="lazy" style="width:100%; height:100%; object-fit: cover;"/>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/yichi-zhang/">Yichi Zhang</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor</p>
<p class="people-department">Statistics</p>
</div>
</div>
```

[Skip to main content](#)

```
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/guangyu-zhu/"><img alt="" class="u-photo wp-post-image" height="200" lo>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/guangyu-zhu/">Guangyu Zhu</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><a class="u-email" href="mailto:guangyuzhu@uri.edu">guangyuzhu@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="inside">
<h2>Adjunct Faculty and Limited Joint Appointments</h2>
</div>
<div class="uri-people-tool cl-tiles halves"><div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/ashley-buchanan/"><img alt="" class="u-photo wp-post-image" height="966" lo>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/ashley-buchanan/">Ashley Buchanan</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Limited Joint Appointment</p>
<p class="people-department">Biostatistics</p>
<p class="people-misc"><span class="p-tel">401.874.4739</span> - <a class="u-email" href="mailto:buchanan@uri.edu">buchanan@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/nina-kajiji/"><img alt="" class="u-photo wp-post-image" height="125" lo>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/nina-kajiji/">Nina Kajiji</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Adjunct Associate Professor</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><a class="u-email" href="mailto:nina@uri.edu">nina@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/rachel-schwartz/"><img alt="" class="u-photo wp-post-image" height="120" lo>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/rachel-schwartz/">Rachel Schwartz</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor - Limited Joint Appointment</p>
<p class="people-department">Biological Sciences</p>
<p class="people-misc"><span class="p-tel">401.874.5404</span> - <a class="u-email" href="mailto:rsschwartz@uri.edu">rsschwartz@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
```

```
<a href="https://web.uri.edu/cs/meet/ying-zhang/"><img alt="" class="u-photo wp-post-image" height="250" loading="lazy" /></a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/ying-zhang/">Ying Zhang</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor - Limited Joint Appointment</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.4915</span> - <a class="u-email" href="mailto:yingzhang@uri.edu">yingzhang@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
</div>
<hr/>
<div>
<p><!-- .entry-content --></p>
</div><!-- #post-## -->
</main><!-- #main -->
</div><!-- #content -->
<div id="actionbar-wrapper">
<div id="actionbar" role="menu">
<a href="https://www.uri.edu/connect/?utm_campaign=request-info&utm_source=action-bar&utm_medium=web">Connect</a>
</div><!-- #actionbar-wrapper -->
<footer id="globalfooter">
<div id="basement">
<div id="storagebins">
<div id="sb-university">
<input checked="" id="sb-university-toggle" name="storagebin" role="presentation" type="radio" value="university"/>
<label aria-label="Open the University footer menu when browsing on mobile." for="sb-university-toggle"><span>University</span></label>
<ul aria-label="The University footer menu." role="menu">
<li><a href="https://www.uri.edu/about/leadership/" role="menuitem">Leadership</a></li>
<li><a href="https://web.uri.edu/diversity/" role="menuitem">Diversity and Inclusion</a></li>
<li><a href="https://web.uri.edu/global/" role="menuitem">Global</a></li>
<li><a href="https://www.uri.edu/about/campuses/" role="menuitem">Campuses</a></li>
<li><a href="https://www.uri.edu/safety/" role="menuitem">Safety</a></li>
</ul>
</div>
<div id="sb-campus-life">
<input id="sb-campus-life-toggle" name="storagebin" role="presentation" type="radio" value="campus-life"/>
<label aria-label="Open the Campus Life footer menu when browsing on mobile." for="sb-campus-life-toggle"><span>Campus Life</span></label>
<ul aria-label="The Campus Life footer menu." role="menu">
<li><a href="https://web.uri.edu/housing/" role="menuitem">Housing</a></li>
<li><a href="https://web.uri.edu/dining/" role="menuitem">Dining</a></li>
<li><a href="https://www.uri.edu/athletics/" role="menuitem">Athletics and Recreation</a></li>
<li><a href="https://www.uri.edu/campus-life/health-and-wellness/" role="menuitem">Health and Wellness</a></li>
<li><a href="https://events.uri.edu" role="menuitem">Events</a></li>
</ul>
</div>
<div id="sb-academics">
<input id="sb-academics-toggle" name="storagebin" role="presentation" type="radio" value="academics"/>
<label aria-label="Open the Academics footer menu when browsing on mobile." for="sb-academics-toggle"><span>Academics</span></label>
<ul aria-label="The Academics footer menu." role="menu">
<li><a href="https://www.uri.edu/academics/" role="menuitem">Undergraduate</a></li>
<li><a href="https://web.uri.edu/graduate-school/" role="menuitem">Graduate</a></li>
<li><a href="https://web.uri.edu/advising/" role="menuitem">Advising</a></li>
<li><a href="https://web.uri.edu/library/" role="menuitem">Libraries</a></li>
<li><a href="https://web.uri.edu/career/students/" role="menuitem">Internships</a></li>
</ul>
</div>
</div>
<div id="gimmicks">
<!-- Tides Widget -->
<div class="uri-tides-widget darkmode" data-height="22"><span class="status"></span></div>
<hr/>
<!-- Social Media Component -->
<aside class="cl-wrapper cl-social-wrapper"><ul class="cl-social light"><li><a class="cl-social-facebook" href="https://www.facebook.com/uri">Facebook</a></li><li><a class="cl-social-twitter" href="https://twitter.com/URI">>Twitter</a></li><li><a class="cl-social-youtube" href="https://www.youtube.com/user/URI">YouTube</a></li><li><a class="cl-social-linkedin" href="https://www.linkedin.com/company/university-of-rhode-island/">LinkedIn</a></li><li><a class="cl-social-instagram" href="https://www.instagram.com/uri/">Instagram</a></li><li><a class="cl-social-twitch" href="https://www.twitch.tv/uri">Twitch</a></li><li><a class="cl-social-tumblr" href="https://uri.tumblr.com">Tumblr</a></li><li><a class="cl-social-pinterest" href="https://www.pinterest.com/uri/">Pinterest</a></li><li><a class="cl-social-flickr" href="https://www.flickr.com/photos/uri/">Flickr</a></li><li><a class="cl-social-yelp" href="https://www.yelp.com/biz/university-of-rhode-island-providence">Yelp</a></li><li><a class="cl-social-scholar" href="https://scholar.google.com/citations?hl=en&user=QWVlZGJyAAAAJ">Google Scholar</a></li><li><a class="cl-social-researchgate" href="https://www.researchgate.net/profile/University_of_Rhode_Island">ResearchGate</a></li><li><a class="cl-social-orcid" href="https://orcid.org/0000-0002-1655-111X">ORCID</a></li><li><a class="cl-social-googleplus" href="https://plus.google.com/u/0/113415350343413173333">Google+</a></li><li><a class="cl-social-mail" href="mailto:info@uri.edu">Email</a></li></ul>
</aside>
<div id="tagline"></div>
<div id="legal">
<p>Copyright © <a class="subtle" href="http://www.uri.edu/">University of Rhode Island</a> | University of Rhode Island is an equal opportunity employer committed to the principles of affirmative action. <a class="jobs" href="https://www.uri.edu/jobs">Jobs</a></p>
</div>
</div>
</div>
```

[Skip to main content](#)

```
cs_people.h3
```

```
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/marco-alvarez/">Marco Alvarez</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

### 10.3.2. 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.

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

▶ Show code cell output

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

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

24

#### ! Important

answer to questions about searching [from the docs](#)

```
type(people_items)
```

```
bs4.element.ResultSet
```

We can also look at only the first instance

```
people_items[0]
```

```
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/marco-alvarez/"><img alt="" class="u-photo wp-post-image" height="120" />
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/marco-alvarez/">Marco Alvarez</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor | Director of Graduate Studies</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5009</span> - <a class="u-email" href="mailto:malvarez@uri.edu"></a>
<div style="clear:both;"></div>
</div>
</div>
```

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').a.string
```

```
'Marco Alvarez'
```

Then we can use a list comprehension to build a list of them.

```
names = [name.a.string for name in cs_people.find_all('h3', 'p-name')]
```

```
['Marco Alvarez',
 'Samantha Armenti',
 'Sarah Brown',
 'Michael Conti',
 'Noah Daniels',
 'Lisa DiPippo',
 'Victor Fay-Wolfe',
 'Lutz Hamel',
 'Abdeltawab Hendawi',
 'Jean-Yves Hervé',
 'Natallia Katenka',
 'Soheyb Kouider',
 'Edmund Lamagna',
 'Indrani Mandal',
 'Gavino Puggioni',
 'Jonathan Schrader',
 'Krishna Venkatasubramanian',
 'Jing Wu',
 'Yichi Zhang',
 'Guangyu Zhu',
 'Ashley Buchanan',
 'Nina Kajiji',
 'Rachel Schwartz',
 'Ying Zhang']
```

```
people_items[0]
```

```
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/marco-alvarez/"><img alt="" class="u-photo wp-post-image" height="120" />
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/marco-alvarez/">Marco Alvarez</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor | Director of Graduate Studies</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5009</span> - <a class="u-email" href="mailto:malvarez@uri.edu">malvarez@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
```

```
people_items[0].find('p', 'people-title').string
```

```
'Associate Professor | Director of Graduate Studies'
```

How to pull out the titles for each person (eg Assistant Teaching Professor, Associate Professor)

```
titles = [t.string for t in cs_people.find_all("p", "people-title")]
```

```
titles
```

```
['Associate Professor | Director of Graduate Studies',
 'Assistant Teaching Professor',
 'Assistant Professor',
 'Assistant Teaching Professor',
 'Assistant Professor',
 'Professor | Chair',
 'Professor',
 'Associate Professor ',
 'Assistant Professor',
 'Associate Professor',
 'Associate Professor | Director of Undergraduate Studies',
 'Associate Teaching Professor',
 'Professor',
 'Assistant Teaching Professor',
 'Associate Professor | Statistics Section Head | Director of Graduate Studies',
 'Assistant Teaching Professor',
 'Assistant Professor',
 'Associate Professor',
 'Assistant Professor ',
 'Assistant Professor',
 'Limited Joint Appointment',
 'Adjunct Associate Professor',
 'Assistant Professor - Limited Joint Appointment',
 'Assistant Professor - Limited Joint Appointment']
```

on one item, the `p` tag seems to work, but that is because the tag gives only the first instance,

```
people_items[0].find('p')
```

but we see if we use this for all, it is way more information than we were looking for.

```
[t.string for t in cs_people.find_all("p")]
```

```
['Associate Professor | Director of Graduate Studies',
 'Computer Science',
 None,
 'Assistant Teaching Professor',
 'Computer Science',
 'sarmenti@uri.edu',
 'Assistant Professor',
 'Computer Science',
 'brownsarahm@uri.edu',
 'Assistant Teaching Professor',
 'Computer Science',
 'michaelconti@uri.edu',
 'Assistant Professor',
 'Computer Science',
 'noah_daniels@uri.edu',
 'Professor | Chair',
 'Computer Science',
 'ldipippo@uri.edu',
 'Professor',
 'Computer Science',
 'vfaywolfe@uri.edu',
 'Associate Professor',
 'Computer Science',
 'lutzhamel@uri.edu',
 'Assistant Professor',
 'Data Science | Computer Science',
 None,
 'Associate Professor',
 'Computer Science',
 'jyh@cs.uri.edu',
 'Associate Professor | Director of Undergraduate Studies',
 'Statistics',
 'nkatenka@uri.edu',
 'Associate Teaching Professor',
 'Statistics',
 None,
 'Professor',
 'Computer Science',
 'eal@cs.uri.edu',
 'Assistant Teaching Professor',
 'Computer Science',
 'indrani_mandal@uri.edu',
 'Associate Professor | Statistics Section Head | Director of Graduate Studies',
 'Statistics',
 None,
 'Assistant Teaching Professor',
 'Computer Science',
 'jonathan.schrader@uri.edu',
 'Assistant Professor',
 'Computer Science',
 'krish@uri.edu',
 'Associate Professor',
 'Statistics',
 None,
 'Assistant Professor',
 'Statistics',
 'yichizhang@uri.edu',
 'Assistant Professor',
 'Statistics',
 'guangyuzhu@uri.edu',
 None,
 'Limited Joint Appointment',
 'Biostatistics',
 None,
 'Adjunct Associate Professor',
 'Computer Science',
 'nina@uri.edu',
 'Assistant Professor - Limited Joint Appointment',
 'Biological Sciences',
 None,
 'Assistant Professor - Limited Joint Appointment'
```

```
'\n',
None,
None]
```

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')]
```

```
css_df = pd.DataFrame({'name':names, 'title':titles, 'e-mails':emails, 'discipline':disciplines})
css_df.head()
```

	name	title	e-mails	discipline
0	Marco Alvarez	Associate Professor   Director of Graduate Stu...	malvarez@uri.edu	Computer Science
1	Samantha Armenti	Assistant Teaching Professor	sarmenti@uri.edu	Computer Science
2	Sarah Brown	Assistant Professor	brownsarahm@uri.edu	Computer Science
3	Michael Conti	Assistant Teaching Professor	michaelconti@uri.edu	Computer Science
4	Noah Daniels	Assistant Professor	noah_daniels@uri.edu	Computer Science

## 10.4. 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/marco-alvarez/">Marco Alvarez</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/marco-alvarez/'}
```

It's a dictionary and the attribute we want is the key we want. Nowe, we do the same thing we did above, request, pull the content from the response and then use the parser.

```
alvarez_url = people_items[0].find('h3', 'p-name').a.attrs['href']
alvarez_html = requests.get(alvarez_url).content
alvarez_info = BeautifulSoup(alvarez_html, 'html.parser')
```

```
alvarez_info.find_all('li','people-location')
```

```
[<li class="people-location"><strong>Office Location:</strong> Tyler 255</li>]
```

it's an iterable, so we pull the item out

```
alvarez_info.find_all('li','people-location')[0]
```

```
<li class="people-location"><strong>Office Location:</strong> Tyler 255</li>
```

Then we get the content.

```
alvarez_info.find_all('li','people-location')[0].string
```

this time this doesn't work, so we can use the python `__dict__` to inspect the object and see where it stored what we want.

```
alvarez_info.find_all('li','people-location')[0].__dict__
```

```
{'parser_class': bs4.BeautifulSoup,
 'name': 'li',
 'namespace': None,
 '_namespaces': {},
 'prefix': None,
 'sourceline': 371,
 'sourcepos': 329,
 'known_xml': False,
 'attrs': {'class': ['people-location']},
 'contents': [<strong>Office Location:</strong>, ' Tyler 255'],
 'parent': <ul class="people-list">
<li class="people-title">Associate Professor | Director of Graduate Studies</li> <li class="people-department">
  <strong>Office Location:</strong> Tyler 255</li>
</ul>,
 'previous_element': ' ',
 'next_element': <strong>Office Location:</strong>,
 'next_sibling': ' ',
 '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)}
```

it's the second element of content

```
alvarez_info.find_all('li','people-location')[0].contents[1]
```

```
' Tyler 255'
```

Now that we know how to do it, we can put it in a loop.

```
offices = []
for name_link in cs_people.find_all('h3', 'p-name'):
    url = name_link.a.attrs['href']
    person_html = requests.get(url).content
    person_info = BeautifulSoup(person_html, 'html.parser')
    try:
        offices.append(person_info.find_all('li', 'people-location')[0].contents[1])
    except:
        offices.append(pd.NA)

css_df['office'] = offices
```

We got an error at first, so we added the `try` and `except` to handle when there is no office location.

```
css_df.head()
```

	name	title	e-mails	discipline	office
0	Marco Alvarez	Associate Professor   Director of Graduate Stu...	malvarez@uri.edu	Computer Science	Tyler 255
1	Samantha Armenti	Assistant Teaching Professor	sarmenti@uri.edu	Computer Science	Tyler 129
2	Sarah Brown	Assistant Professor	brownsarahm@uri.edu	Computer Science	Tyler 134
3	Michael Conti	Assistant Teaching Professor	michaelconti@uri.edu	Computer Science	Tyler 137
4	Noah Daniels	Assistant Professor	noah_daniels@uri.edu	Computer Science	Tyler 250

```
css_df.to_csv('css_faculty.csv')
```

## 10.5. Questions after class

### 10.5.1. what does .a do?

it gives the first instance of the `<a>` tag

### 10.5.2. 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.

### 10.5.3. In theory, you could parse images and potentially their metadata with this method?

This method could be a way to download images and the text that is around them, yes. This is how a lot of image datasets are built for machine learning.

### 10.5.4. Is API the website's way of specifying what information it will allow for you to have?

What we did today did not use any API. An API call would use the request library, and similar patterns to what we did, especially the end of class. However an API call would typically respond with json, not html.

### 10.5.5. In the web-scraping of the offices, there were two strings, 'CBLS 377', and 'CBLS Building 487' how would we use pandas to normalize things like this?"

We could use the `replace` method that we used last week.

## 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
- gives you a chance to learn *only* evaluation instead of evaluation + an ML task

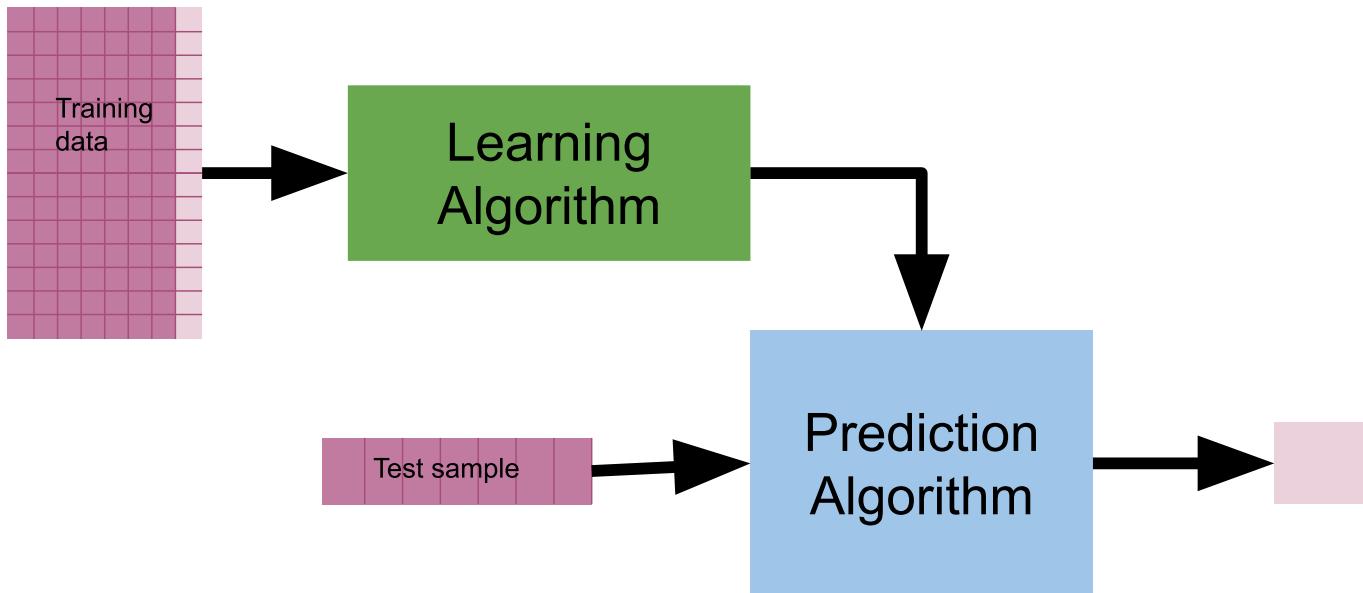
### 11.1. What is a Machine Learning Algorithm?

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.

```
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.

- `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 recidivated
- `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>	<b>distance_to_last_commitment</b>
<b>0</b>	3	34		F African-American	25 - 45	Low	Male	0		-1.0
<b>1</b>	4	24		F African-American	Less than 25	Low	Male	4		-1.0
<b>2</b>	8	41		F Caucasian	25 - 45	Medium	Male	14		-1.0
<b>3</b>	10	39		M Caucasian	25 - 45	Low	Female	0		-1.0
<b>4</b>	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.

```
pd.get_dummies(compas_df['score_text'])
```

		<b>High</b>	<b>Low</b>	<b>Medium</b>
	<b>0</b>	False	True	False
	<b>1</b>	False	True	False
	<b>2</b>	False	False	True
	<b>3</b>	False	True	False
	<b>4</b>	False	True	False
	...	...	...	...
	<b>5273</b>	False	True	False
	<b>5274</b>	True	False	False
	<b>5275</b>	False	False	True
	<b>5276</b>	False	True	False
	<b>5277</b>	False	True	False

5278 rows × 3 columns

```
compas_onehot = pd.concat([compas_df,pd.get_dummies(compas_df['score_text'])],axis=1)
```

[Skip to main content](#)

We could have done the above line in one neater step, but in class I for this was an option.

```
compas_df_onehot = pd.get_dummies(compas_df, columns=['score_text'])
```

Next lets look at the thresholds that were used so that we know what the mean

```
compas_onehot.groupby('score_text')['decile_score'].agg(['min', 'max'])
```

	min	max
score_text		
High	8	10
Low	1	4
Medium	5	7

We will also audit with respect to second threshold.

```
compas_onehot['MedHigh'] = compas_onehot['High'] + compas_onehot['Medium']
```

## 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['High'])
```

```
0.6288366805608185
```

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*.

```
metrics.accuracy_score(compas_onehot['two_year_recid'], compas_onehot['MedHigh'])
```

```
0.6582038651004168
```

```
metrics.confusion_matrix(compas_onehot['two_year_recid'], compas_onehot['MedHigh'])
```

```
array([[1872,  923],  
       [ 881, 1602]])
```

### i 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
- true positives( $C_{1,1}$ ): got a high score and were re-arrested

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. 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_df['two_year_recid'], compas_df['score_text_High'])
```

```

-----
KeyError                                     Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3652, in 
3651     try:
-> 3652         return self._engine.get_loc(casted_key)
3653     except KeyError as err:
3654
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc()
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pandas._libs.index.IndexEngine.get_loc()
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: 'score_text_High'

The above exception was the direct cause of the following exception:

KeyError                                     Traceback (most recent call last)
Cell In[11], line 1
-> 1 metrics.recall_score(compas_df['two_year_recid'],compas_df['score_text_High'])

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/frame.py:3760, in DataFrame._get_item_multilevel
3758     if self.columns.nlevels > 1:
3759         return self._getitem_multilevel(key)
-> 3760     indexer = self.columns.get_loc(key)
3761     if is_integer(indexer):
3762         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3654, in 
3652     return self._engine.get_loc(casted_key)
3653 except KeyError as err:
-> 3654     raise KeyError(key) from err
3655 except TypeError:
3656     # If we have a listlike key, _check_indexing_error will raise
3657     # InvalidIndexError. Otherwise we fall through and re-raise
3658     # the TypeError.
3659     self._check_indexing_error(key)

KeyError: 'score_text_High'

```

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.recall_score(compas_onehot['two_year_recid'],compas_onehot['MedHigh'])
```

```
0.6451872734595248
```

```
metrics.precision_score(compas_onehot['two_year_recid'],compas_onehot['MedHigh'])
```

```
0.6344554455445545
```

## 11.6. Per Group Scores

```
acc_fx = lambda d: metrics.accuracy_score(d['two_year_recid'], d['MedHigh'])
compas_onehot.groupby('race').apply(acc_fx)
```

```
race
African-American    0.649134
Caucasian          0.671897
dtype: float64
```

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[15], 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
```

```
recall_fx = lambda d: metrics.recall_score(d['two_year_recid'], d['MedHigh'])
compas_onehot.groupby('race').apply(recall_fx)
```

```
race
African-American    0.715232
Caucasian          0.503650
dtype: float64
```

```
precision_fx = lambda d: metrics.precision_score(d['two_year_recid'], d['MedHigh'])
compas_onehot.groupby('race').apply(precision_fx)
```

```
race
African-American    0.649535
Caucasian          0.594828
dtype: float64
```

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

[Skip to main content](#)

The Fair Machine Learning book's classification Chapter has a section on relationships between criteria with the proofs.

### ! 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.

## 11.7. Prepare for Next Class

install aif360

## 11.8. Portfolio

Audience is not *me*, but a generally knowledgeable person. For example:

- a student deciding if they want to take this course or not. They know how to code, but not datascience.
- a person familiar with the domain your data is from (eg a sports fan if sports data)
- a future employer who wants to know about your skills

any of these people know big ideas, but not exactly what happened in class. You can specify which audience you're targetting in the introduction (which is the one piece that I'm the audience for)

Goal is to show what you understand and are able to do not only what you *can* do, because you can do a lot of simple things by finding answers online. we want you to understand enough that when you start seeing new, real problems, you're able to do these things on your own

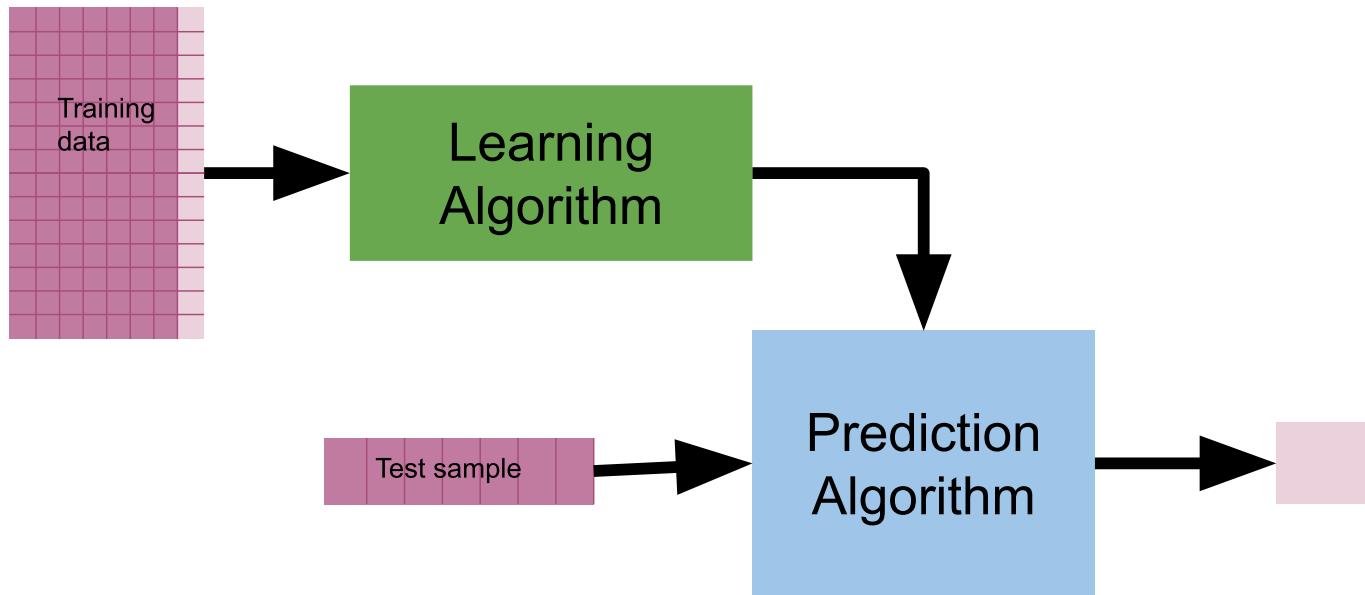
- level 1: you can follow a conversation
- level 2: you can do it if someone gives you a rough plan
- level 3: you can do it, given only an end goal Think of this more like a report with code as the figures than a coding assignment. To see what you've learned we should be able to read through on piece of text, not compare two files so it could be like:

## 12. Auditing with AIF360

### 12.1. What is ML?

i Note

If you  
that is  
out to



```
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 'tempeh': LawSchoolGPADataset will be unavailable. To install, run:  
pip install 'aif360[LawSchoolGPA]'
```

```
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, run:  
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 '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.

## 12.2. The COMPAS data

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

First, we will look at it

```
compas_df.head()
```

[Skip to main content](#)

<b>id</b>	<b>age</b>	<b>c_charge_degree</b>	<b>race</b>	<b>age_cat</b>	<b>sex</b>	<b>priors_count</b>	<b>days_b_screening_arrest</b>	<b>decile_score</b>	<b>is_low</b>
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.

We use the `two_year_recid` as the basis of our audit because it is the real outcome that the designers of COMPAS were hoping to predict. Since the COMPAS score is on a scale of 1-10, we transform to a binary variable by thresholding it (eg all above 1 are 1, below are 0). We use the `score_text` instead of `decile_score` in our thresholding so that we use a recommended threshold.

More 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)
```

Let's review computing the accuracy with sklearn:

```
skmetrics.accuracy_score(compas_df['two_year_recid'],
                         compas_df['score_text_High'])
```

0.6288366805608185

```
skmetrics.accuracy_score(compas_df['two_year_recid'],
                         compas_df['score_text_MedHigh'])
```

0.6582038651004168

## 12.3. What about breaking it down by race?

Recall, we used groupby to get the per race score by creating a `lambda` function that we could apply to the groupby object.

[Skip to main content](#)

```
compas_race = compas_df.groupby('race')
```

We can apply our method to each part of the groupby object with `apply`

```
acc_fx = lambda d: skmetrics.accuracy_score(d['two_year_recid'],
                                             d['score_text_MedHigh'])

compas_race.apply(acc_fx).reset_index().rename(columns={0:'accuracy'})
```

	race	accuracy
0	African-American	0.649134
1	Caucasian	0.671897

## 12.4. ML Notation

We use standard notation in machine learning, and in fair machine learning specifically.

This is important because we want to be able to communicate, like we call the horizontal and vertical axes of a plot the `x` and `y` axes.

The AIF 360 package we are about to use and sklearn both use this notation.

- *target* or *labels*, denoted by for one sample (row)  $i$   $\mathbf{y}_i$ .
- whole column of the target variable is  $Y$
- “hat” notation for predictions/ output of prediction algorithm  $\hat{y}_i$  and  $\hat{Y}$
- “protected attribute”  $a_i$  and  $A$

we use lowercase for one sample and uppercase for many.

```
help(skmetrics.accuracy_score)
```

Help on function accuracy\_score in module sklearn.metrics.\_classification:

```
accuracy_score(y_true, y_pred, *, normalize=True, sample_weight=None)
    Accuracy classification score.
```

In multilabel classification, this function computes subset accuracy:  
the set of labels predicted for a sample must \*exactly\* match the  
corresponding set of labels in y\_true.

Read more in the :ref:`User Guide <accuracy\_score>`.

#### Parameters

-----

```
y_true : 1d array-like, or label indicator array / sparse matrix
    Ground truth (correct) labels.
```

```
y_pred : 1d array-like, or label indicator array / sparse matrix
    Predicted labels, as returned by a classifier.
```

```
normalize : bool, default=True
    If ``False``, return the number of correctly classified samples.
    Otherwise, return the fraction of correctly classified samples.
```

```
sample_weight : array-like of shape (n_samples,), default=None
    Sample weights.
```

#### Returns

-----

```
score : float
    If ``normalize == True``, return the fraction of correctly
    classified samples (float), else returns the number of correctly
    classified samples (int).
```

The best performance is 1 with ``normalize == True`` and the number  
of samples with ``normalize == False``.

#### See Also

-----

```
balanced_accuracy_score : Compute the balanced accuracy to deal with
    imbalanced datasets.
```

```
jaccard_score : Compute the Jaccard similarity coefficient score.
```

```
hamming_loss : Compute the average Hamming loss or Hamming distance between
    two sets of samples.
```

```
zero_one_loss : Compute the Zero-one classification loss. By default, the
    function will return the percentage of imperfectly predicted subsets.
```

#### Notes

-----

In binary classification, this function is equal to the `jaccard\_score`  
function.

#### Examples

-----

```
>>> from sklearn.metrics import accuracy_score
>>> y_pred = [0, 2, 1, 3]
>>> y_true = [0, 1, 2, 3]
>>> accuracy_score(y_true, y_pred)
0.5
>>> accuracy_score(y_true, y_pred, normalize=False)
2
```

In the multilabel case with binary label indicators:

```
>>> import numpy as np
>>> accuracy_score(np.array([[0, 1], [1, 1]]), np.ones((2, 2)))
0.5
```

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

```
race_num_map = {r:i for i,r, in enumerate(compas_df['race'].value_counts().index)}
```

```
{'African-American': 0, 'Caucasian': 1}
```

```
compas_df['race'].replace(race_num_map)
```

```
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
```

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	race	two_year_recid	score_text_MedHigh
3	0	1	0
4	0	1	0

The scoring object requires that we have special data structures that wrap a DataFrame.

[Skip to main content](#)

Next we will make two versions, one with race & the ground truth and the other 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'])
```

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`.

```
# here we want actual favorable outcome
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 also the label names are 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
```

i Note  
remember  
object

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.

### 12.5.1. 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. It is like dividing things equally.

```
compas_fair_scorer.disparate_impact()
```

```
0.6336457196581771
```

US court doctrine says that this quantity has to be above .8 for employment decisions. Does COMPAS pass this criterion?

### 12.6. 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_scoring.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.

Black people were given a low score and then re-arrested only a little more than half as often as white people. (White people were give an low score and rearrested almost twice as often)

To make a single metric, we might take a ratio. This is where the journalists found bias.

```
compas_fair_scoring.false_positive_rate_ratio()
```

```
0.5737241916634204
```

This metric would be fair with a value of 1.

got a high score and did not re-arrested as a percentage of those who got a high score

We can look at the other type of error

```
compas_fair_scoring.false_negative_rate(True)
```

```
0.22014051522248243
```

```
compas_fair_scoring.false_negative_rate(False)
```

```
0.4233817701453104
```

```
compas_fair_scoring.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

[Skip to main content](#)

## 12.6.1. 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
```

**note** if time, discuss:

- What should this look like if it is fair?
- what could this metric hide?

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.

## 12.7. Sufficiency and Calibration

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()
```

```
1.2118532033913119
```

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.

To put it all together, we can make a plot. First we'll make a DataFrame

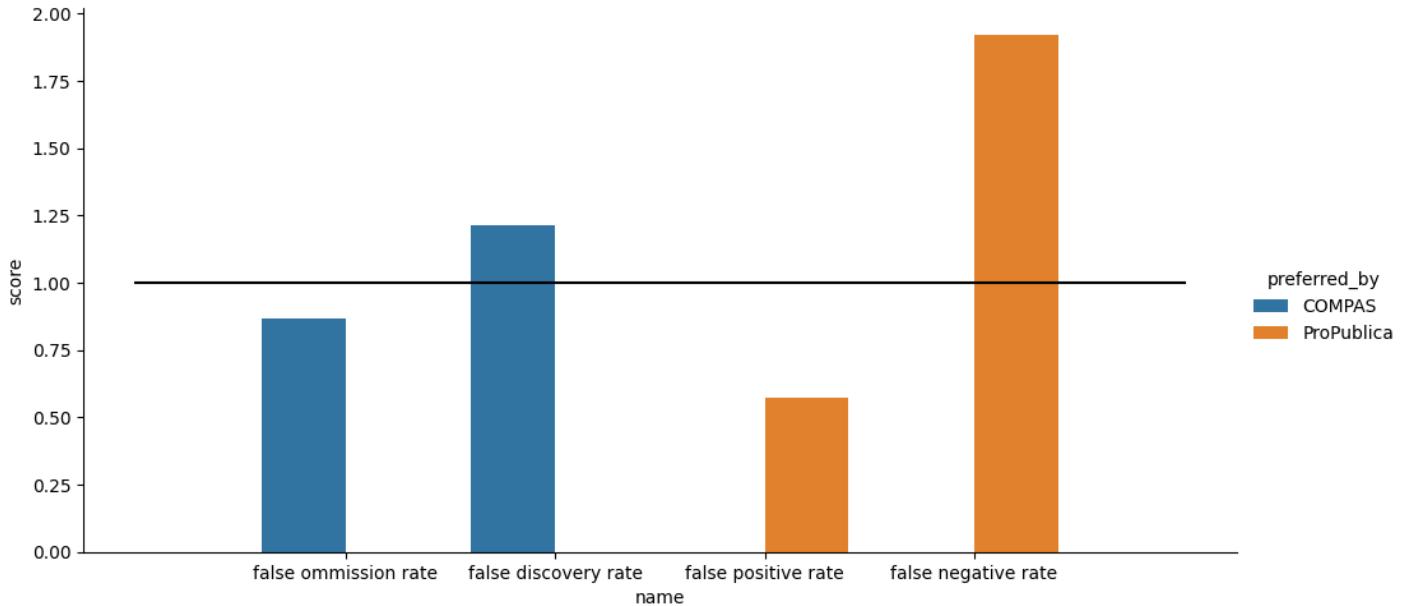
```
ratios = [{  
    'score':compas_fair_scorer.false_omission_rate_ratio(),  
    'name': 'false omission rate',  
    'group':'sufficiency',  
    'preferred_by':'COMPAS'},  
    {'score':compas_fair_scorer.false_discovery_rate_ratio(),  
    'name': 'false discovery rate',  
    'group':'sufficiency',  
    'preferred_by':'COMPAS'},  
    {'score':compas_fair_scorer.false_positive_rate_ratio(),  
    'name': 'false positive rate',  
    'group':'separation',  
    'preferred_by':'ProPublica'},  
    {'score':compas_fair_scorer.false_negative_rate_ratio(),  
    'name': 'false negative rate',  
    'group':'separation',  
    'preferred_by':'ProPublica'}]  
ratio_df = pd.DataFrame(ratios)  
ratio_df
```

	score	name	group	preferred_by
0	0.864977	false omission rate	sufficiency	COMPAS
1	1.211853	false discovery rate	sufficiency	COMPAS
2	0.573724	false positive rate	separation	ProPublica
3	1.923234	false negative rate	separation	ProPublica

```
%matplotlib inline
```

```
sns.catplot(data=ratio_df,y='score',x='name',hue='preferred_by',  
            kind='bar',aspect=2)  
sns.lineplot(x = [-1,4],y=[1,1],color='black',legend=False)
```

```
<Axes: xlabel='name', ylabel='score'>
```



These are all ratios, so 1 is fair. COMPAS does okay on the measures it was designed around and poorly on the ones the journalists preferred.

```
compas_fair_scorer.false_omission_rate_difference()
```

```
-0.05470767896532869
```

```
compas_fair_scorer.false_discovery_rate_ratio()
```

```
1.2118532033913119
```

## 13. Intro to ML & Naive Bayes

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.

## 13.1. What is a Model?

A model is a simplified representation of some part of the world. A famous quote about models is:

13.2. All models are wrong, but some are useful –George Box[^wiki]

13.3. 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](#)

## 13.4. Models in Machine Learning

---

13.5. 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

$$y_i = f(\mathbf{x}_i)$$

---

## 13.6. $i$ would be the index of a DataFrame

---

## 13.7. 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
- 

## 13.8. 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.

$$y_i = f(\mathbf{x}_i)$$

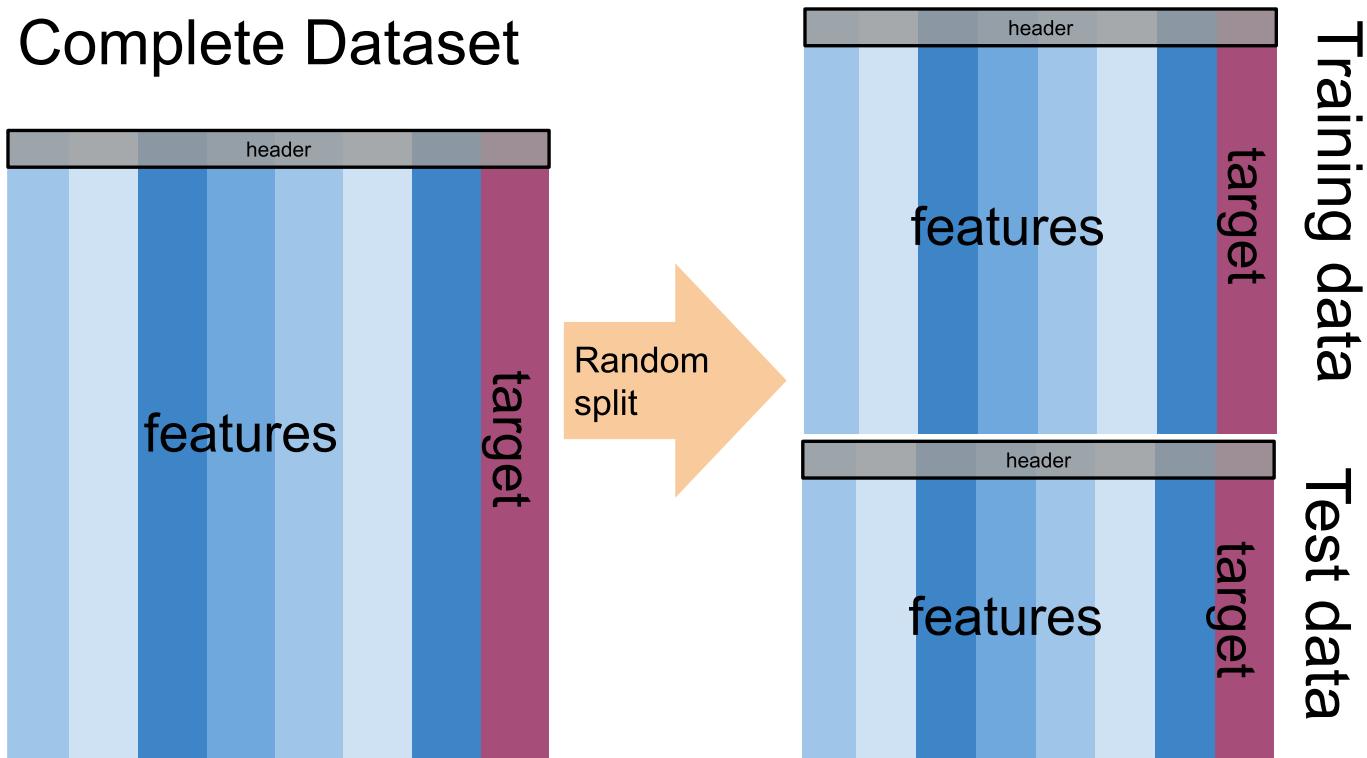
- if  $y_i$  are discrete (eg flower species) we are doing **classification**
- if  $y_i$  are continuous (eg height) we are doing **regression**

## 13.9. 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.



```

flowchart LR
    A[whole dataset] -->|random split sample-wise| B[training data]
    A -->|random split sample-wise| C[test data]
    B -->|lalgo| D[palgo]
    C -->|lalgo| E[palgo]
    D -->|palgo| F[pred]
    E -->|palgo| G[pred]
    F -->|pred| H[metrics]
    G -->|pred| I[metrics]
    H -->|metrics| J[metrics]
    I -->|metrics| J
    J -->|score| K[scores/ performance]
  
```

## 13.10. Let's Practice:

First machine learning model: Naive bayes

```
%matplotlib inline
```

```
import pandas as pd
import seaborn as sns
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
iris_df = sns.load_dataset('iris')
```

To start we will look at the data

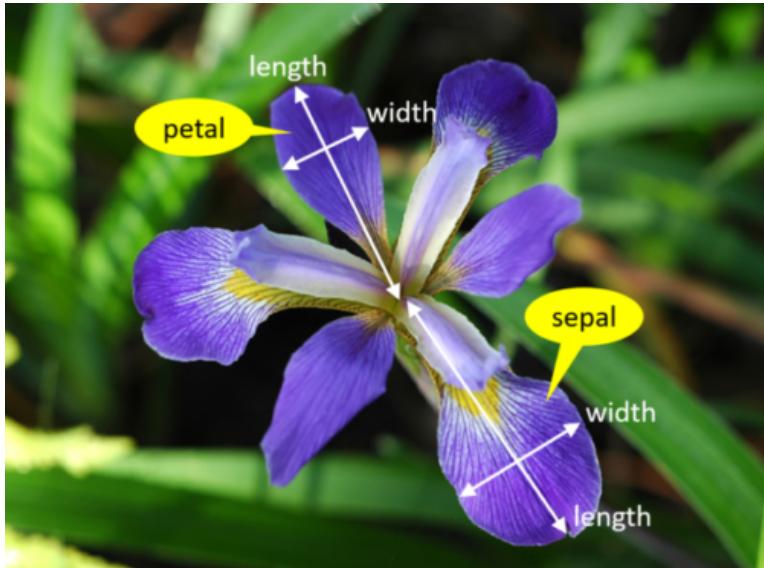
```
iris_df.head(1)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa

And notice that there are equal number of samples for each value of the species.

```
iris_df['species'].value_counts()
```

```
species
setosa      50
versicolor  50
virginica   50
Name: count, dtype: int64
```



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

[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'
```

```
X_train, X_test, y_train, y_test = train_test_split(iris_df[feature_vars], iris_df[target_var], random_state=1)
```

```
X_train.shape
```

```
(112, 4)
```

```
X_test.shape
```

```
(38, 4)
```

```
iris_df.shape
```

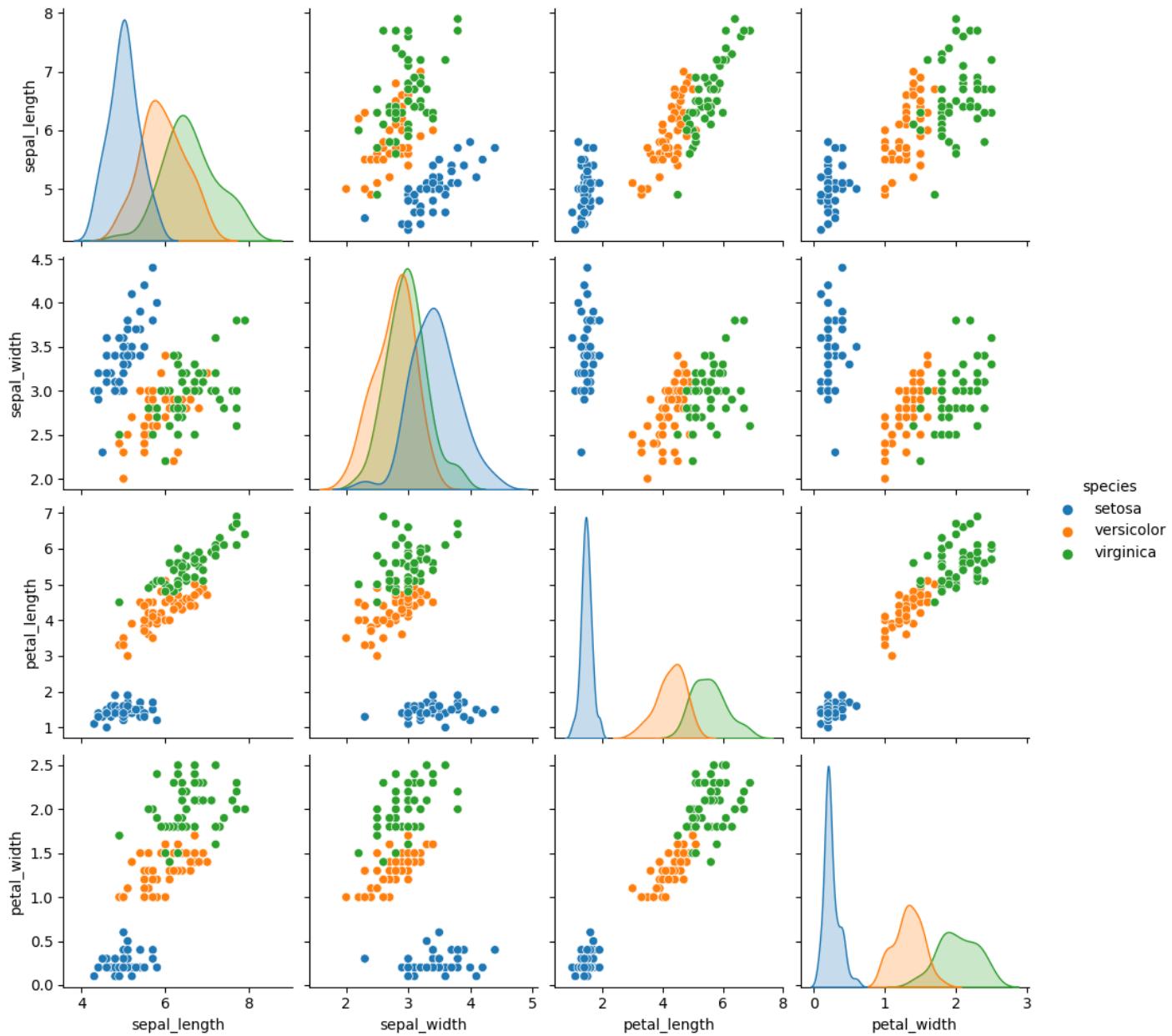
```
(150, 5)
```

```
112/150
```

```
0.7466666666666667
```

```
sns.pairplot(iris_df, hue='species')
```

```
<seaborn.axisgrid.PairGrid at 0x7fd6b07c6370>
```



## 13.11. What does Naive Bayes do?

docs

Naive = independent features Bayes = most probable

Bayes Estimator

```
gnb = GaussianNB()
```

```
type(gnb)
```

[Skip to main content](#)

```
sklearn.naive_bayes.GaussianNB
```

```
gnb.__dict__
```

```
{'priors': None, 'var_smoothing': 1e-09}
```

```
gnb.fit(X_train,y_train)
```

```
▼ GaussianNB
```

```
GaussianNB()
```

```
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_': 3.1405070153061226e-09,
 'theta_': array([[4.94594595, 3.35675676, 1.46756757, 0.23513514],
    [5.96470588, 2.74411765, 4.23529412, 1.30882353],
    [6.51707317, 2.95853659, 5.52926829, 2.00243902]]),
 'var_': array([[0.10842951, 0.13434624, 0.02651571, 0.01146823],
    [0.28875433, 0.10658305, 0.22346021, 0.04139274],
    [0.36775729, 0.09754908, 0.31280191, 0.07877454]]),
 'class_count_': array([37., 34., 41.]),
 'class_prior_': array([0.33035714, 0.30357143, 0.36607143])}
```

```
gnb.score(X_test, y_test)
```

```
0.9736842105263158
```

```
y_pred = gnb.predict(X_test)
```

```
y_pred[:3]
```

```
array(['setosa', 'versicolor', 'versicolor'], dtype='<U10')
```

```
y_test[:3]
```

```
14      setosa
98      versicolor
75      versicolor
Name: species, dtype: object
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[13,  0,  0],  
       [ 0, 15,  1],  
       [ 0,  0,  9]])
```

```
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	13
versicolor	1.00	0.94	0.97	16
virginica	0.90	1.00	0.95	9
accuracy			0.97	38
macro avg	0.97	0.98	0.97	38
weighted avg	0.98	0.97	0.97	38

```
sum(y_pred ==y_test)
```

```
37
```

```
37/len(y_test)
```

```
0.9736842105263158
```

```
y_train_pred = gnb.predict(X_train)
```

```
sum(y_train_pred == y_train)
```

```
106
```

```
len(y_train)
```

```
112
```

```
106/112
```

```
0.9464285714285714
```

```

import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
iris_df = sns.load_dataset('iris')

```

## 14.1. Classifying Irises

```
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

Our goal is to predict the *species* from the *measurements*. In machine learning, we call the species the *target variable*. The three species of irises, setosa, virginica and versicolor are called the *classes*. Since the target variable is categorical, this prediction task is a classification problem.

```

# dataset vars:
# 'petal_width', 'sepal_length', 'species', 'sepal_width', 'petal_length',
feature_vars = ['petal_width', 'sepal_length', 'sepal_width', 'petal_length',]
target_var = 'species'
X_train, X_test, y_train, y_test = train_test_split(iris_df[feature_vars], iris_df[target_var], random_state=0)

```

Using the `random_state` makes it so that we get the same “random” set each type we run the code. [see docs](#)

Notice that in the training data the indices are randomly ordered unlike in the original DataFrame.

```
X_train.head()
```

	petal_width	sepal_length	sepal_width	petal_length
137	1.8	6.4	3.1	5.5
84	1.5	5.4	3.0	4.5
27	0.2	5.2	3.5	1.5
127	1.8	6.1	3.0	4.9
132	2.2	6.4	2.8	5.6

Next we can instantiate our estimator object which in this case is the `GaussianNB`

[Skip to main content](#)

```
gnb = GaussianNB()  
gnb.fit(X_train,y_train)
```

```
▼ GaussianNB  
GaussianNB()
```

We use the fit method to “learn” or find the values of the parameters.

```
gnb.score(X_test,y_test)
```

```
0.9666666666666667
```

Then we can check how well it works

```
y_pred = gnb.predict(X_test)
```

And see the actual predictions

```
y_pred == y_test
```

```
114    True  
62     True  
33     True  
107    True  
7      True  
100    True  
40     True  
86     True  
76     True  
71     True  
134    False  
51     True  
73     True  
54     True  
63     True  
37     True  
78     True  
90     True  
45     True  
16     True  
121    True  
66     True  
24     True  
8      True  
126    True  
22     True  
44     True  
97     True  
93     True  
26     True  
Name: species, dtype: bool
```

The s  
includ  
imple  
includ  
that c  
algori

## 14.2. How does GNB make predictions?

```
gnb.__dict__
```

```
{'priors': None,
 'var_smoothing': 1e-09,
 'classes_': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'feature_names_in_': array(['petal_width', 'sepal_length', 'sepal_width', 'petal_length'],
    dtype=object),
 'n_features_in_': 4,
 'epsilon_': 3.159332638888889e-09,
 'theta_': array([[0.24102564, 5.02051282, 3.4025641 , 1.46153846],
   [1.32432432, 5.88648649, 2.76216216, 4.21621622],
   [2.03181818, 6.63863636, 2.98863636, 5.56590909]]),
 'var_': array([[0.01113741, 0.12932282, 0.1417883 , 0.02031559],
   [0.04075968, 0.26387144, 0.10397371, 0.23000731],
   [0.06444215, 0.38918905, 0.10782542, 0.29451963]]),
 'class_count_': array([39., 37., 44.]),
 'class_prior_': array([0.325      , 0.30833333, 0.36666667])}
```

The attributes of the `estimator` object (`gnb`) describe the data (eg the class list) and the model's parameters. The `theta_` ( $\theta$ ) represents the mean and the `var_` ( $\sigma$ ) represents the variance of the distributions.

```
gnb.theta_
```

```
array([[0.24102564, 5.02051282, 3.4025641 , 1.46153846],
   [1.32432432, 5.88648649, 2.76216216, 4.21621622],
   [2.03181818, 6.63863636, 2.98863636, 5.56590909]])
```

### 💡 Try it Yourself

Could you use what we learned about EDA to find the mean and variance of each feature for each species of flower?

Because the GaussianNB classifier calculates parameters that describe the assumed distribution of the data it is called a generative classifier. From a generative classifier, we can generate synthetic data that is from the distribution the classifier learned. If this data looks like our real data, then the model assumptions fit well.

### ⚠️ Warning

the details of this math are not required understanding, but this describes the following block of code

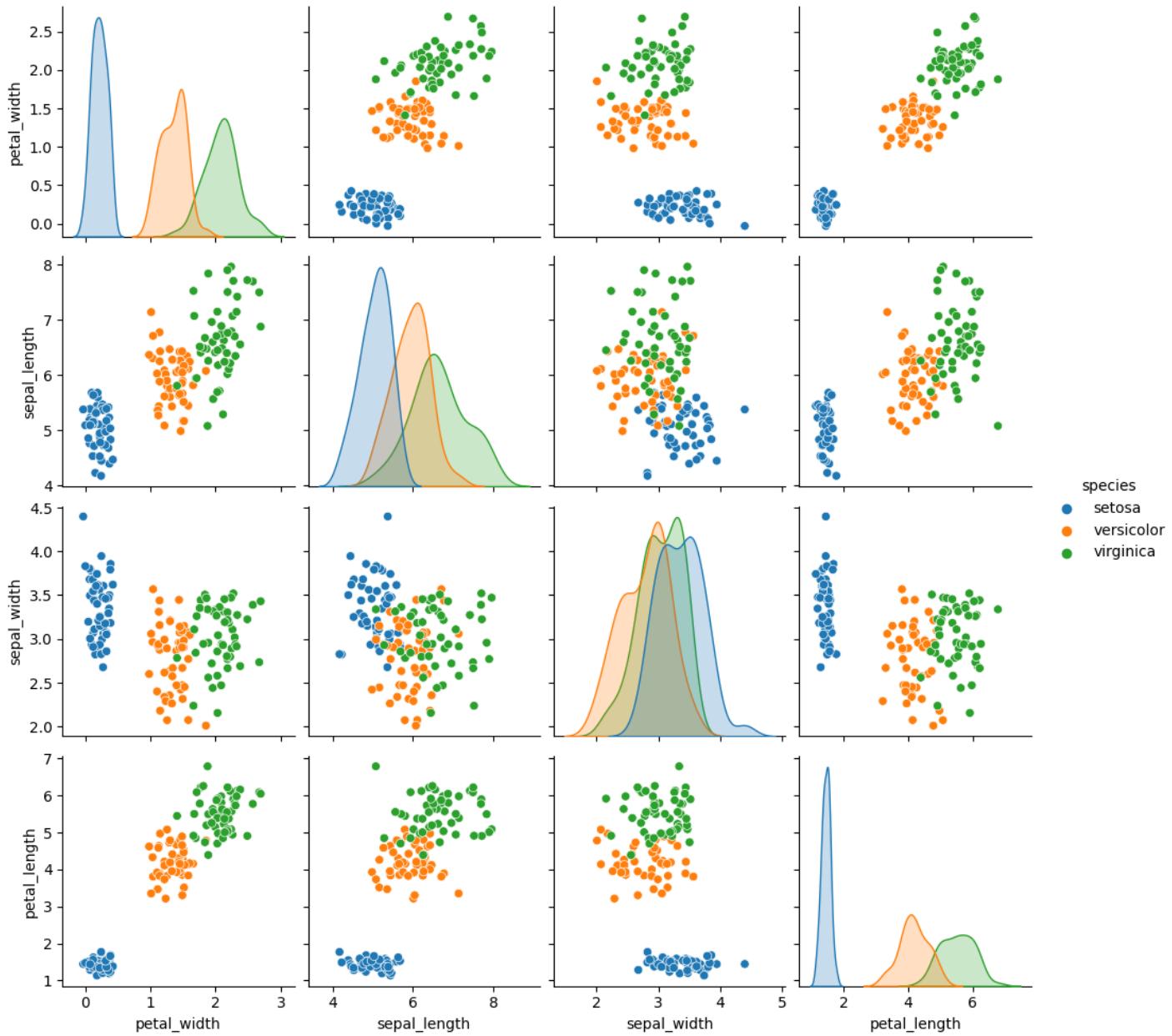
To do this, we extract the mean and variance parameters from the model (`gnb.theta_, gnb.sigma_`) and `zip` them together to create an iterable object that in each iteration returns one value from each list (`for th, sig in zip(gnb.theta_, gnb.sigma_)`). We do this inside of a list comprehension and 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.

Hints  
to try  
code,  
running  
it up  
wrapping

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.

```
N = 50
gnb_df = pd.DataFrame(np.concatenate([np.random.multivariate_normal(theta, sig*np.eye(4), N)
                                       for theta, sig 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')
```

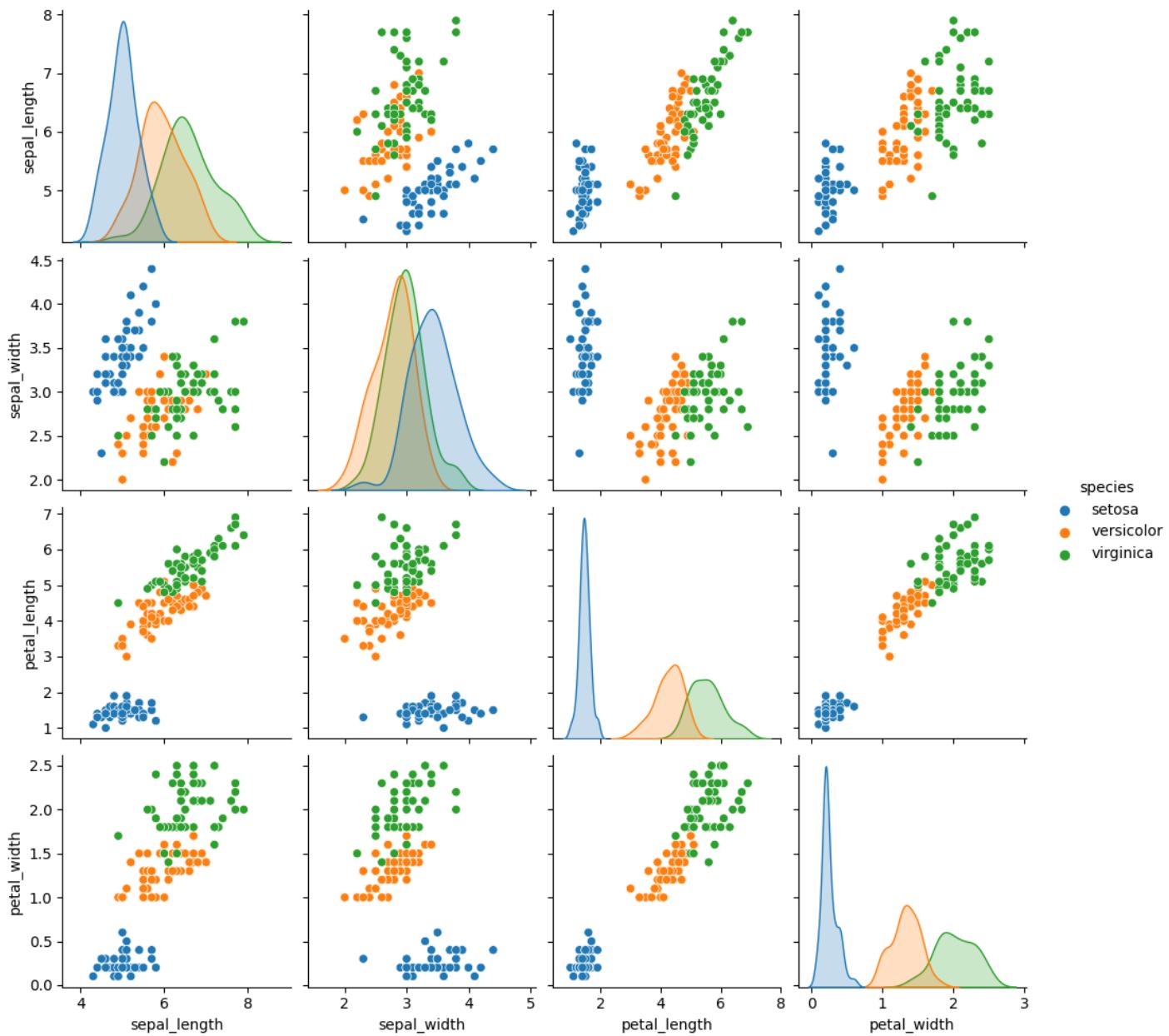
```
<seaborn.axisgrid.PairGrid at 0x7f248ca7ddc0>
```



```
sns.pairplot(data=iris_df, hue='species')
```

[Skip to main content](#)

```
<seaborn.axisgrid.PairGrid at 0x7f24785a0160>
```



This one looks pretty close to the actual data. The biggest difference is that these data are all in uniformly circular-ish blobs and the ones above are not.

That means that the naive assumption doesn't hold perfectly on this data.

When we use the predict method, it uses those parameters to calculate the likelihood of the sample according to a Gaussian distribution (normal) for each class and then calculates the probability of the sample belonging to each class and returns the one with the highest probability.

## 14.3. Interpreting probabilities

We can look directly at the probabilities

[Skip to main content](#)

```
array([[1.63380783e-232, 2.18878438e-006, 9.99997811e-001],
       [1.82640391e-082, 9.99998304e-001, 1.69618390e-006],
       [1.00000000e+000, 7.10250510e-019, 3.65449801e-028],
       [1.58508262e-305, 1.04649020e-006, 9.99998954e-001],
       [1.00000000e+000, 8.59168655e-017, 4.22159374e-027],
       [6.39815011e-321, 1.56450314e-010, 1.00000000e+000],
       [1.00000000e+000, 1.09797313e-016, 5.30276557e-027],
       [1.25122812e-146, 7.74052109e-001, 2.25947891e-001],
       [5.34357526e-150, 9.07564955e-001, 9.24350453e-002],
       [5.67261712e-093, 9.99882109e-001, 1.17891111e-004],
       [2.3865144e-210, 5.29609631e-001, 4.70390369e-001],
       [8.12047631e-132, 9.43762575e-001, 5.62374248e-002],
       [5.25177109e-132, 9.98864361e-001, 1.13563851e-003],
       [1.24498038e-139, 9.49838641e-001, 5.01613586e-002],
       [4.08232760e-140, 9.88043864e-001, 1.19561365e-002],
       [1.00000000e+000, 7.12837229e-019, 4.10162749e-029],
       [4.19553996e-131, 9.87944980e-001, 1.20550201e-002],
       [4.13286716e-111, 9.99942383e-001, 5.76167389e-005],
       [1.00000000e+000, 2.24933112e-015, 3.63624519e-026],
       [1.00000000e+000, 9.86750131e-016, 2.42355087e-025],
       [1.85930865e-186, 1.66966805e-002, 9.83303319e-001],
       [8.83060167e-130, 9.92757232e-001, 7.24276827e-003],
       [1.00000000e+000, 4.26380344e-013, 4.34222344e-023],
       [1.00000000e+000, 1.28045851e-016, 1.26708019e-027],
       [2.43739221e-168, 1.83516225e-001, 8.16483775e-001],
       [1.00000000e+000, 2.62431469e-018, 6.72573168e-029],
       [1.00000000e+000, 3.20605389e-011, 1.52433420e-020],
       [2.20964201e-110, 9.99291229e-001, 7.08771072e-004],
       [1.39297338e-046, 9.99999972e-001, 2.81392389e-008],
       [1.00000000e+000, 1.85943966e-013, 1.58833385e-023]])
```

We can see that for most of these one value is close to one and the others are close to zero. One way to emphasize this is to round them.

```
y_prob = gnb.predict_proba(X_test)
np.round(y_prob,2)
```

```
array([[0. , 0. , 1. ],
       [0. , 1. , 0. ],
       [1. , 0. , 0. ],
       [0. , 0. , 1. ],
       [1. , 0. , 0. ],
       [0. , 0. , 1. ],
       [1. , 0. , 0. ],
       [0. , 0.77, 0.23],
       [0. , 0.91, 0.09],
       [0. , 1. , 0. ],
       [0. , 0.53, 0.47],
       [0. , 0.94, 0.06],
       [0. , 1. , 0. ],
       [0. , 0.95, 0.05],
       [0. , 0.99, 0.01],
       [1. , 0. , 0. ],
       [0. , 0.99, 0.01],
       [0. , 1. , 0. ],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [0. , 0.02, 0.98],
       [0. , 0.99, 0.01],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [0. , 0.18, 0.82],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [0. , 1. , 0. ],
       [0. , 1. , 0. ]])
```

## ! Important

The following did not happen in class, but I added more explanation so that it is more clear.

These are hard to interpret as is, one option is to plot them

```
# 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','correct'],
                               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	virginica	virginica	0,virginica	0,virginica	True	setosa	1.633808e-232
1	1	versicolor	versicolor	1,versicolor	1,versicolor	True	setosa	1.826404e-82
2	2	setosa	setosa	2,setosa	2,setosa	True	setosa	1.000000e+00
3	3	virginica	virginica	3,virginica	3,virginica	True	setosa	1.585083e-305
4	4	setosa	setosa	4,setosa	4,setosa	True	setosa	1.000000e+00

Now we have a data frame where each row is one the probability of one sample belonging to one class. So there's a total of

number\_of\_samples\*number\_of\_classes rows

```
prob_df_melted.shape
```

(90, 8)

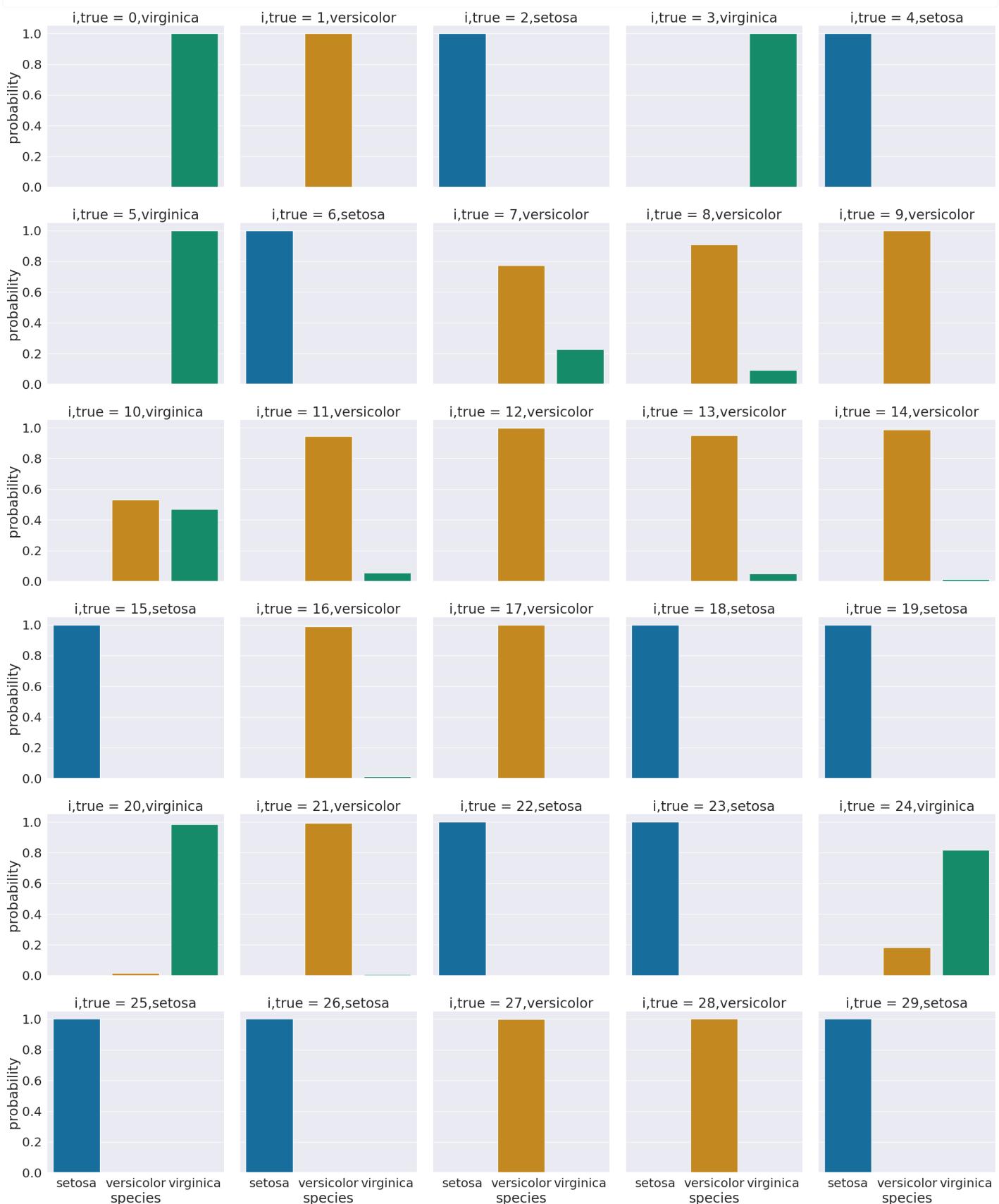
```
len(y_pred)*len(gnb.classes_)
```

90

One way to look at these is to, for each sample in the test set, make a bar chart of the probability it belongs to each class. We added to the data frame information so that we can plot this with the true class in the title using col = 'i,true'

```
sns.set_theme(font_scale=2, palette= "colorblind")
# 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')
```

&lt;seaborn.axisgrid.FacetGrid at 0x7f247864e910&gt;



We see that most samples have nearly all of their probability mass (all probabilities in a distribution sum (or integrate if

[Skip to main content](#)

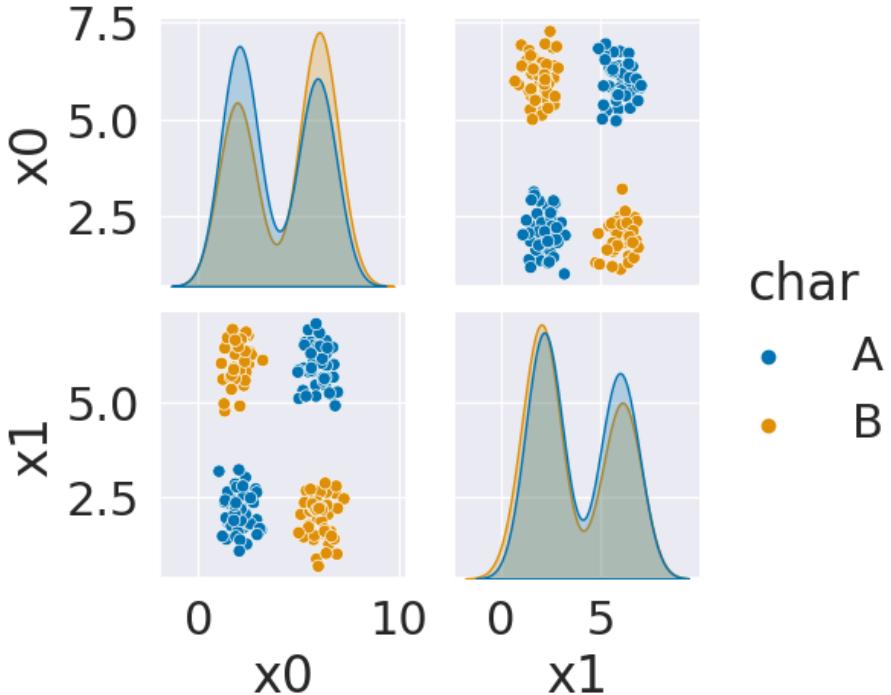
I used the fo Seab choos palette distin forms

## 14.4. What about a harder dataset?

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/f425ba121cc0c4dd8bcaa7ebb2ff0b4
df6= pd.read_csv(corner_data,usecols=[1, 2, 3])
gnb_corners = GaussianNB()
sns.pairplot(data=df6, hue='char',hue_order=['A', 'B'])
```

```
<seaborn.axisgrid.PairGrid at 0x7f247868d880>
```



As we can see in this dataset, these classes are quite separated. We might expect a pretty good performance.

```
df6.head()
```

	x0	x1	char
0	6.14	2.10	B
1	2.22	2.39	A
2	2.27	5.44	B
3	1.03	3.19	A
4	2.25	1.71	A

```
Xcorners_train, Xcorners_test, ycorners_train, ycorners_test = train_test_split(
    df6.drop(columns='char'), df6['char'])
gnb_corners.fit(Xcorners_train, ycorners_train)
```

[Skip to main content](#)

▼ GaussianNB

GaussianNB()

```
gnb_corners.score(Xcorners_test,ycorners_test)
```

0.7

But we do not get a very good classification score.

To see why, we can look at what it learned.

```
N = 100
gnb_corners_df = pd.DataFrame(np.concatenate([np.random.multivariate_normal(th, sig*np.eye(2),N)
                                              for th, sig in zip(gnb_corners.theta_,gnb.sigma_)]),
                                columns = ['x0','x1'])
gnb_corners_df['char'] = [ci for cl in [[c]*N for c in gnb_corners.classes_] for ci in cl]

sns.pairplot(data =gnb_corners_df, hue='char',hue_order=['A','B'])
```

```
-----
AttributeError                                     Traceback (most recent call last)
Cell In[23], line 3
      1 N = 100
      2 gnb_corners_df = pd.DataFrame(np.concatenate([np.random.multivariate_normal(th, sig*np.eye(2),N)
----> 3          for th, sig in zip(gnb_corners.theta_,gnb.sigma_)]),
      4          columns = ['x0','x1'])
      5 gnb_corners_df['char'] = [ci for cl in [[c]*N for c in gnb_corners.classes_] for ci in cl]
      6 sns.pairplot(data =gnb_corners_df, hue='char',hue_order=['A','B'])

AttributeError: 'GaussianNB' object has no attribute 'sigma_'
```

## 14.5. Decision Trees

The Gaussian Naive Bayes model we have seen so far, worked on this really well separated Gaussian distributed data. Not all data is that easy to classify. Sometimes coming up with a description of the data for a generative model is hard and it might not even be important. A scientist will find that valuable, but it is not always needed. Another way to think about classification is to focus on writing a rule (or set of rules) for determining the label from the features. This type of classifier is called discriminative, because it focuses on discriminating (in the literal, differentiate sense, not the socially loaded differentiate on the basis of an attribute of the person in unfair ways sense) between the classes.

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 axis at a time.

It is a **discriminative** model, because it describes how to discriminate (in the sense of differentiate) between the classes.

```
from sklearn import tree
```

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.

[Skip to main content](#)

```
dt = tree.DecisionTreeClassifier()
dt.fit(Xcorners_train,ycorners_train)
```

```
▼ DecisionTreeClassifier
DecisionTreeClassifier()
```

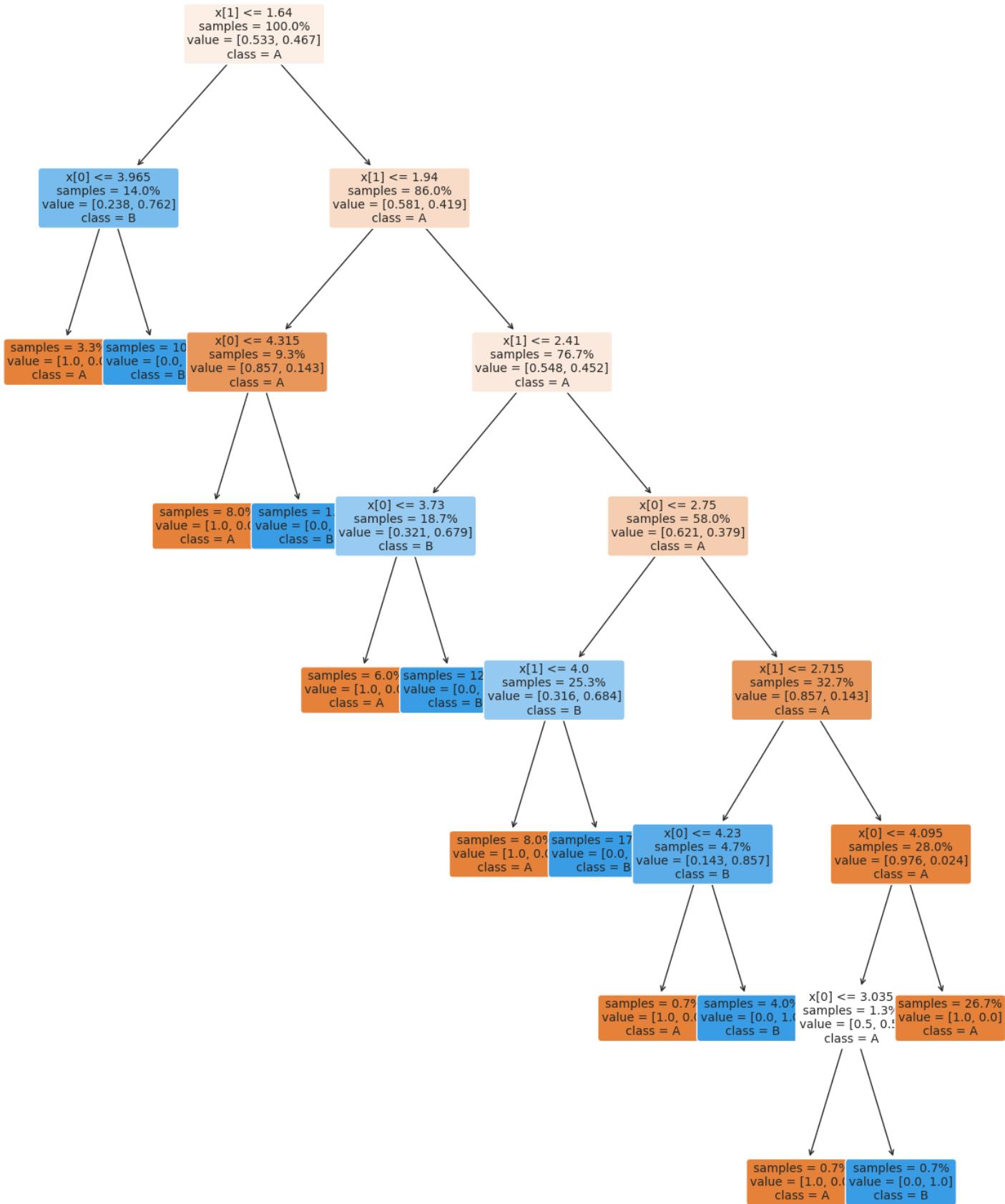
```
dt.score(Xcorners_test,ycorners_test)
```

```
0.94
```

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)
```

```
[Text(0.246875, 0.9375, 'x[1] <= 1.64\nsamples = 100.0%\nvalue = [0.533, 0.467]\nnclass = A'),  
Text(0.1, 0.8125, 'x[0] <= 3.965\nsamples = 14.0%\nvalue = [0.238, 0.762]\nnclass = B'),  
Text(0.05, 0.6875, 'samples = 3.3%\nvalue = [1.0, 0.0]\nnclass = A'),  
Text(0.15, 0.6875, 'samples = 10.7%\nvalue = [0.0, 1.0]\nnclass = B'),  
Text(0.39375, 0.8125, 'x[1] <= 1.94\nsamples = 86.0%\nvalue = [0.581, 0.419]\nnclass = A'),  
Text(0.25, 0.6875, 'x[0] <= 4.315\nsamples = 9.3%\nvalue = [0.857, 0.143]\nnclass = A'),  
Text(0.2, 0.5625, 'samples = 8.0%\nvalue = [1.0, 0.0]\nnclass = A'),  
Text(0.3, 0.5625, 'samples = 1.3%\nvalue = [0.0, 1.0]\nnclass = B'),  
Text(0.5375, 0.6875, 'x[1] <= 2.41\nsamples = 76.7%\nvalue = [0.548, 0.452]\nnclass = A'),  
Text(0.4, 0.5625, 'x[0] <= 3.73\nsamples = 18.7%\nvalue = [0.321, 0.679]\nnclass = B'),  
Text(0.35, 0.4375, 'samples = 6.0%\nvalue = [1.0, 0.0]\nnclass = A'),  
Text(0.45, 0.4375, 'samples = 12.7%\nvalue = [0.0, 1.0]\nnclass = B'),  
Text(0.675, 0.5625, 'x[0] <= 2.75\nsamples = 58.0%\nvalue = [0.621, 0.379]\nnclass = A'),  
Text(0.55, 0.4375, 'x[1] <= 4.0\nsamples = 25.3%\nvalue = [0.316, 0.684]\nnclass = B'),  
Text(0.5, 0.3125, 'samples = 8.0%\nvalue = [1.0, 0.0]\nnclass = A'),  
Text(0.6, 0.3125, 'samples = 17.3%\nvalue = [0.0, 1.0]\nnclass = B'),  
Text(0.8, 0.4375, 'x[1] <= 2.715\nsamples = 32.7%\nvalue = [0.857, 0.143]\nnclass = A'),  
Text(0.7, 0.3125, 'x[0] <= 4.23\nsamples = 4.7%\nvalue = [0.143, 0.857]\nnclass = B'),  
Text(0.65, 0.1875, 'samples = 0.7%\nvalue = [1.0, 0.0]\nnclass = A'),  
Text(0.75, 0.1875, 'samples = 4.0%\nvalue = [0.0, 1.0]\nnclass = B'),  
Text(0.9, 0.3125, 'x[0] <= 4.095\nsamples = 28.0%\nvalue = [0.976, 0.024]\nnclass = A'),  
Text(0.85, 0.1875, 'x[0] <= 3.035\nsamples = 1.3%\nvalue = [0.5, 0.5]\nnclass = A'),  
Text(0.8, 0.0625, 'samples = 0.7%\nvalue = [1.0, 0.0]\nnclass = A'),  
Text(0.9, 0.0625, 'samples = 0.7%\nvalue = [0.0, 1.0]\nnclass = B'),  
Text(0.95, 0.1875, 'samples = 26.7%\nvalue = [1.0, 0.0]\nnclass = A')]
```



## 14.6. 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(Xcorners_train,ycorners_train)
dt2.score(Xcorners_test,ycorners_test)
```

0.46

We might need to play with different parameters to get it just how we want it. A simpler model can be better because it is easier to understand and sometimes will generalize, or apply to new data, better.

## 14.7. Example Loading UCI data



### Hint

This is a hint for a7

Remember in classification, we predict a categorical variable from related continuous feature variables.

The data is a `.data` file, but if you look at it, it will actually be comma, tab, or space delimited, so `read_csv` will work.

```
pd.read_csv('glass.data').head()
```

```

-----
FileNotFoundError                                 Traceback (most recent call last)
Cell In[29], line 1
----> 1 pd.read_csv('glass.data').head()

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:912, in
899 kwds_defaults = _refine_defaults_read(
900     dialect,
901     delimiter,
902     (
903         ...
904         dtype_backend=dtype_backend,
905     )
906     kwds.update(kwds_defaults)
--> 912 return _read(filepath_or_buffer, kwds)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:577, in
574 _validate_names(kwds.get("names", None))
576 # Create the parser.
--> 577 parser = TextFileReader(filepath_or_buffer, **kwds)
579 if chunksize or iterator:
580     return parser

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:1407, in
1404     self.options["has_index_names"] = kwds["has_index_names"]
1406 self.handles: IOHandles | None = None
-> 1407 self._engine = self._make_engine(f, self.engine)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:1661, in
1659     if "b" not in mode:
1660         mode += "b"
--> 1661 self.handles = get_handle(
1662     f,
1663     mode,
1664     encoding=self.options.get("encoding", None),
1665     compression=self.options.get("compression", None),
1666     memory_map=self.options.get("memory_map", False),
1667     is_text=is_text,
1668     errors=self.options.get("encoding_errors", "strict"),
1669     storage_options=self.options.get("storage_options", None),
1670 )
1671 assert self.handles is not None
1672 f = self.handles.handle

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/common.py:859, in get_handle
854 elif isinstance(handle, str):
855     # Check whether the filename is to be opened in binary mode.
856     # Binary mode does not support 'encoding' and 'newline'.
857     if ioargs.encoding and "b" not in ioargs.mode:
858         # Encoding
--> 859         handle = open(
860             handle,
861             ioargs.mode,
862             encoding=ioargs.encoding,
863             errors=errors,
864             newline="",
865         )
866     else:
867         # Binary mode
868         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'glass.data'

```

I would notice that the first row is data, not headers so I would go to the `glass.names` file to find the column names

in the `glass.names` file there is a lot of other information, but the important part is a list of the columns:

1. Id number: 1 to 214
2. RI: refractive index

[Skip to main content](#)

```
are attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron
11. Type of glass:
```

If I were going to work on this, I would copy that section into a string and then delete that () section after sodium.

```
names= ''' 1. Id number: 1 to 214
2. RI: refractive index
3. Na: Sodium
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron
11. Type of glass:'''
```

Then instead of cutting each one to make a list manually, I might do the following:

```
col_names = [cn.split('.')[1].split(':')[0].strip().replace(' ', '_')
for cn in names.split('\n')]
```

```
['Id_number',
'RI',
'Na',
'Mg',
'Al',
'Si',
'K',
'Ca',
'Ba',
'Fe',
'Type_of_glass']
```

```
pd.read_csv('glass.data', names=col_names).head()
```

```

-----
FileNotFoundError                                     Traceback (most recent call last)
Cell In[32], line 1
----> 1 pd.read_csv('glass.data', names=col_names).head()

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:912, in
899 kwds_defaults = _refine_defaults_read(
900     dialect,
901     delimiter,
902     (
903         ...
904         dtype_backend=dtype_backend,
905     )
906     kwds.update(kwds_defaults)
--> 912 return _read(filepath_or_buffer, kwds)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:577, in
574 _validate_names(kwds.get("names", None))
576 # Create the parser.
--> 577 parser = TextFileReader(filepath_or_buffer, **kwds)
579 if chunksize or iterator:
580     return parser

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:1407, in
1404     self.options["has_index_names"] = kwds["has_index_names"]
1406 self.handles: IOHandles | None = None
-> 1407 self._engine = self._make_engine(f, self.engine)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:1661, in
1659     if "b" not in mode:
1660         mode += "b"
--> 1661 self.handles = get_handle(
1662     f,
1663     mode,
1664     encoding=self.options.get("encoding", None),
1665     compression=self.options.get("compression", None),
1666     memory_map=self.options.get("memory_map", False),
1667     is_text=is_text,
1668     errors=self.options.get("encoding_errors", "strict"),
1669     storage_options=self.options.get("storage_options", None),
1670 )
1671 assert self.handles is not None
1672 f = self.handles.handle

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/common.py:859, in get_handle
854 elif isinstance(handle, str):
855     # Check whether the filename is to be opened in binary mode.
856     # Binary mode does not support 'encoding' and 'newline'.
857     if ioargs.encoding and "b" not in ioargs.mode:
858         # Encoding
--> 859         handle = open(
860             handle,
861             ioargs.mode,
862             encoding=ioargs.encoding,
863             errors=errors,
864             newline="",
865         )
866     else:
867         # Binary mode
868         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'glass.data'

```

This looks good, so I would save it now. I might even save it as csv

```
glass_df = pd.read_csv('glass.data', names=col_names)
```

```
-----  
FileNotFoundError                                     Traceback (most recent call last)  
Cell In[33], line 1  
----> 1 glass_df = pd.read_csv('glass.data', names=col_names)  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:912, in  
899 kwds_defaults = _refine_defaults_read(  
900     dialect,  
901     delimiter,  
(...)  
908     dtype_backend=dtype_backend,  
909 )  
910 kwds.update(kwds_defaults)  
--> 912 return _read(filepath_or_buffer, kwds)  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:577, in  
574 _validate_names(kwds.get("names", None))  
576 # Create the parser.  
--> 577 parser = TextFileReader(filepath_or_buffer, **kwds)  
579 if chunksize or iterator:  
580     return parser  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:1407, in  
1404     self.options["has_index_names"] = kwds["has_index_names"]  
1406 self.handles: IOHandles | None = None  
-> 1407 self._engine = self._make_engine(f, self.engine)  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/parsers/readers.py:1661, in  
1659     if "b" not in mode:  
1660         mode += "b"  
-> 1661 self.handles = get_handle(  
1662     f,  
1663     mode,  
1664     encoding=self.options.get("encoding", None),  
1665     compression=self.options.get("compression", None),  
1666     memory_map=self.options.get("memory_map", False),  
1667     is_text=is_text,  
1668     errors=self.options.get("encoding_errors", "strict"),  
1669     storage_options=self.options.get("storage_options", None),  
1670 )  
1671 assert self.handles is not None  
1672 f = self.handles.handle  
  
File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/pandas/io/common.py:859, in get_handle  
854 elif isinstance(handle, str):  
855     # Check whether the filename is to be opened in binary mode.  
856     # Binary mode does not support 'encoding' and 'newline'.  
857     if ioargs.encoding and "b" not in ioargs.mode:  
858         # Encoding  
-> 859         handle = open(  
860             handle,  
861             ioargs.mode,  
862             encoding=ioargs.encoding,  
863             errors=errors,  
864             newline="",  
865         )  
866     else:  
867         # Binary mode  
868         handle = open(handle, ioargs.mode)  
  
FileNotFoundError: [Errno 2] No such file or directory: 'glass.data'
```

```
glass_df['Type_of_glass'].value_counts()
```

```
-----  
NameError                                 Traceback (most recent call last)  
Cell In[34], line 1  
----> 1 glass_df['Type_of_glass'].value_counts()  
  
NameError: name 'glass_df' is not defined
```

This shows that there are 6 classes, and they are different sizes.

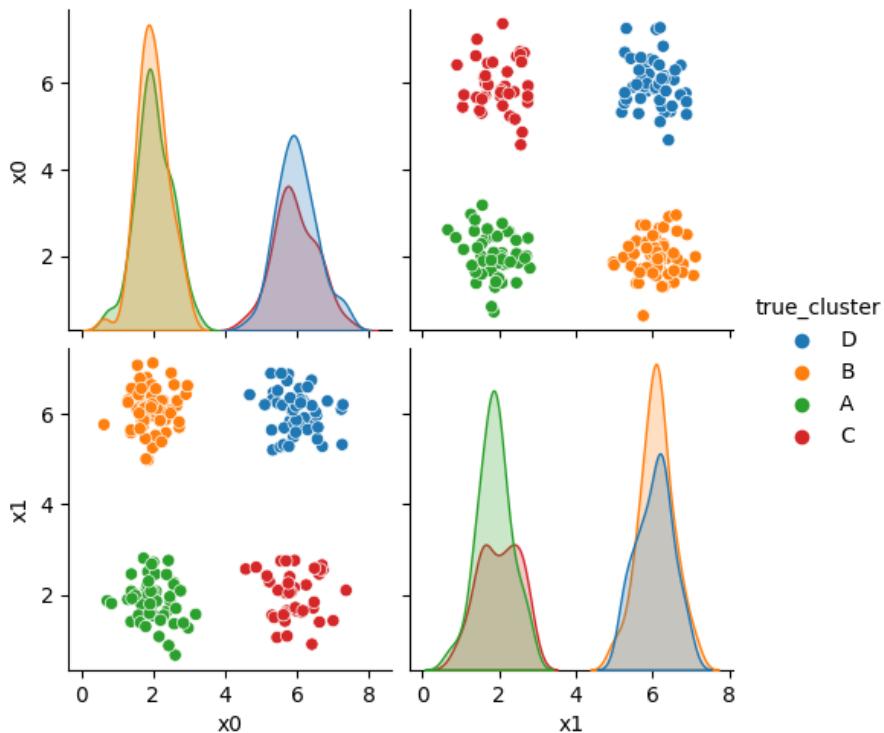
## 15. Clustering

```
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
```

```
%matplotlib inline
```

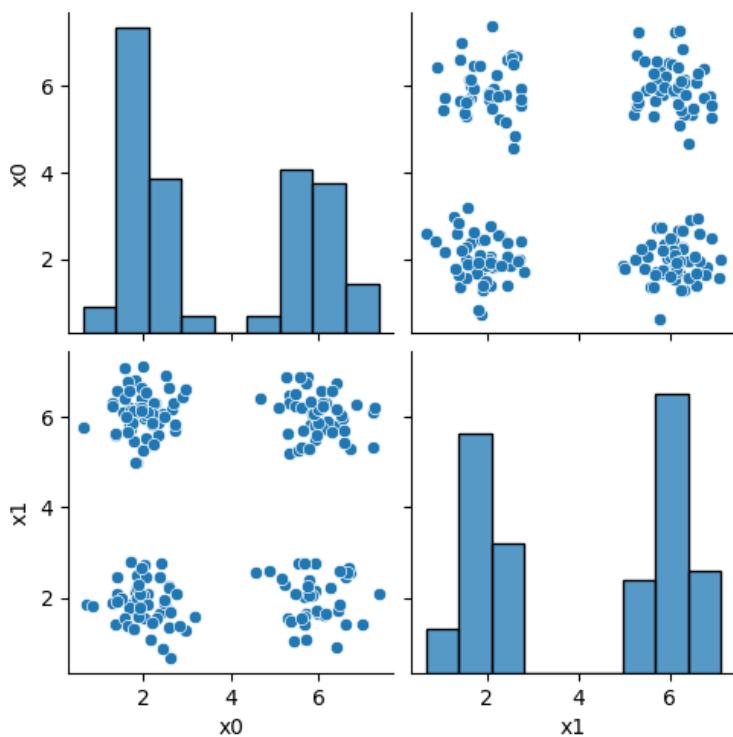
```
C = 4  
N = 200  
offset = 2  
spacing = 2  
  
# choose the first C uppercase letters using the builtin string class  
classes = list(string.ascii_uppercase[:C])  
# get the number of grid locations needed  
G = int(np.ceil(np.sqrt(C)))  
# get the locations for each axis  
grid_locs = a = np.linspace(offset, 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)}  
# random variances  
sigma = {c: i*.5 for c, i in zip(classes,np.random.random(4))}  
  
#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], .25*np.eye(2)) for c in clusters_true]  
# rounding to make display neater later  
df = pd.DataFrame(data = data,columns = ['x' + str(i) for i in range(2)]).round(2)  
  
# store in dataFrame  
df['true_cluster'] = clusters_true  
  
sns.pairplot(data = df, hue='true_cluster')
```

```
<seaborn.axisgrid.PairGrid at 0x7f9f64ed3f40>
```



```
sns.pairplot(data = df)
```

```
<seaborn.axisgrid.PairGrid at 0x7f9f273a0af0>
```



```
data_color = df['true_cluster']
```

[Skip to main content](#)

```
K = 4
mu = df[data_cols].sample(n=K).values
mu
```

```
array([[1.8 , 5. ],
       [2.46, 1.96],
       [2.46, 1.88],
       [5.9 , 5.49]])
```

```
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]
```

```
sns.color_palette('tab20',8)
```



```
df[data_cols].head(1)
```

	x0	x1
0	5.27	6.89

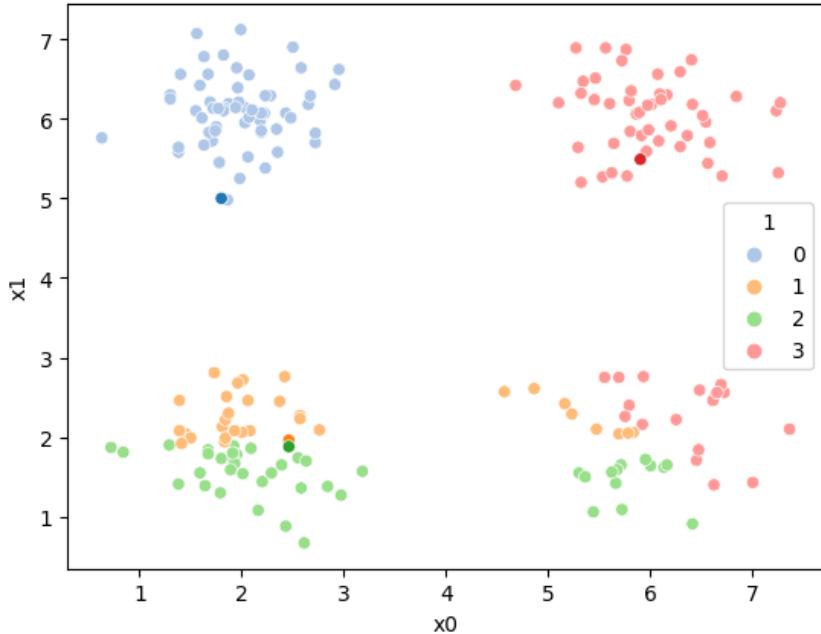
```
mu[0]
```

```
array([1.8, 5. ])
```

```
df['1']= pd.concat([(df[data_cols]-mu_i)**2].sum(axis=1) for mu_i in mu],axis=1).idxmin(axis=1)
```

```
sfig = sns.scatterplot(data=df, x='x0',y='x1', palette=cmap_pt, hue='1')
mu_df = mu_to_df(mu,1)
sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
```

```
<Axes: xlabel='x0', ylabel='x1'>
```

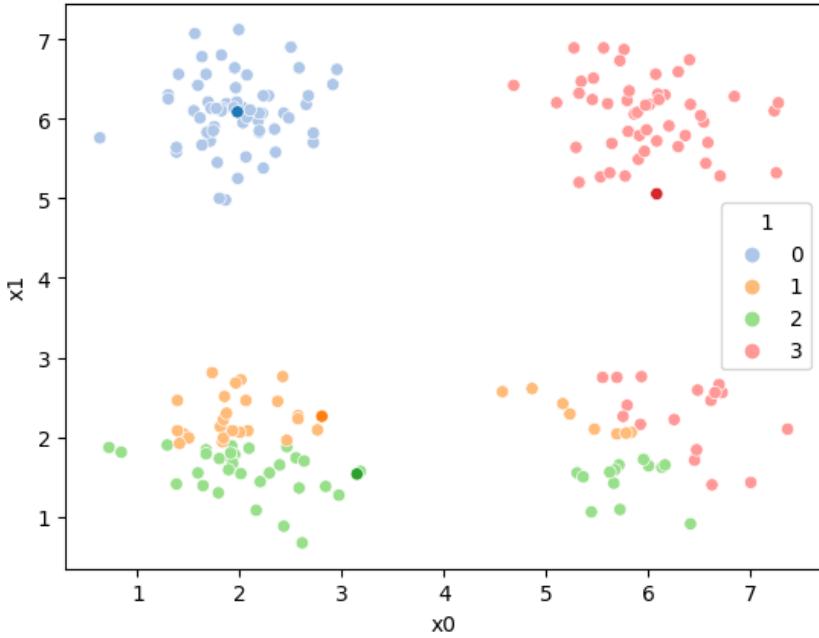


```
df.head()
```

	x0	x1	true_cluster	1
0	5.27	6.89		D 3
1	1.69	6.21		B 0
2	1.96	6.39		B 0
3	6.36	5.79		D 3
4	1.92	1.89		A 2

```
mu = df.groupby('1')[data_cols].mean().values
sfig = sns.scatterplot(data=df, x='x0',y='x1', palette=cmap_pt, hue='1')
mu_df = mu_to_df(mu,2)
sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
```

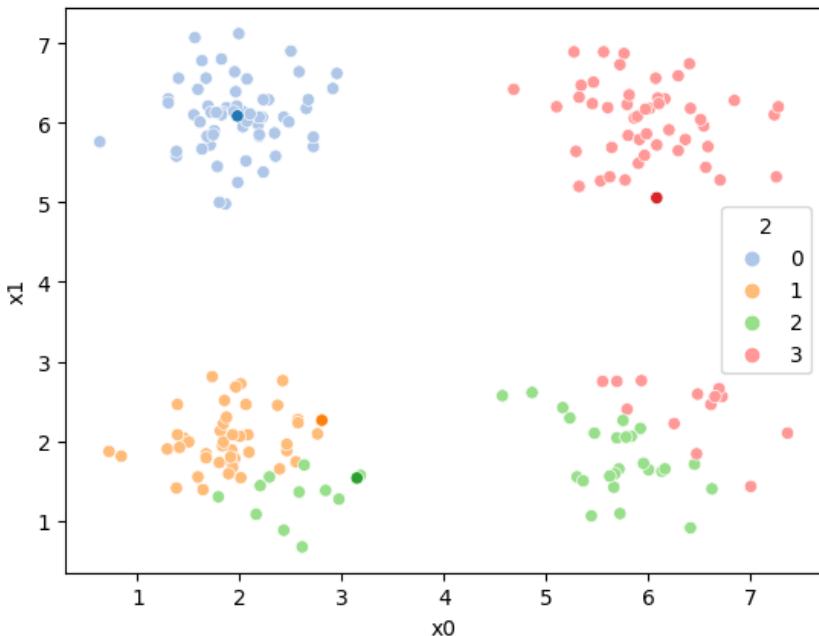
```
<Axes: xlabel='x0', ylabel='x1'>
```



Now we can update the assignments again

```
df['2'] = pd.concat([(df[data_cols]-mu_i)**2].sum(axis=1) for mu_i in mu],axis=1).idxmin(axis=1)
sfig = sns.scatterplot(data=df, x='x0',y='x1', palette=cmap_pt, hue='2')
mu_df = mu_to_df(mu,2)
sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
```

```
<Axes: xlabel='x0', ylabel='x1'>
```



```
df.head()
```

[Skip to main content](#)

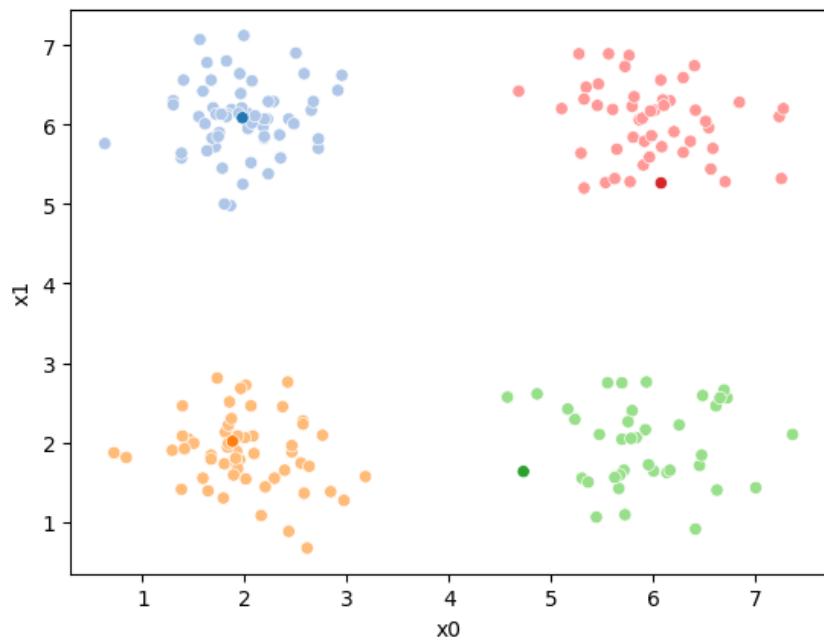
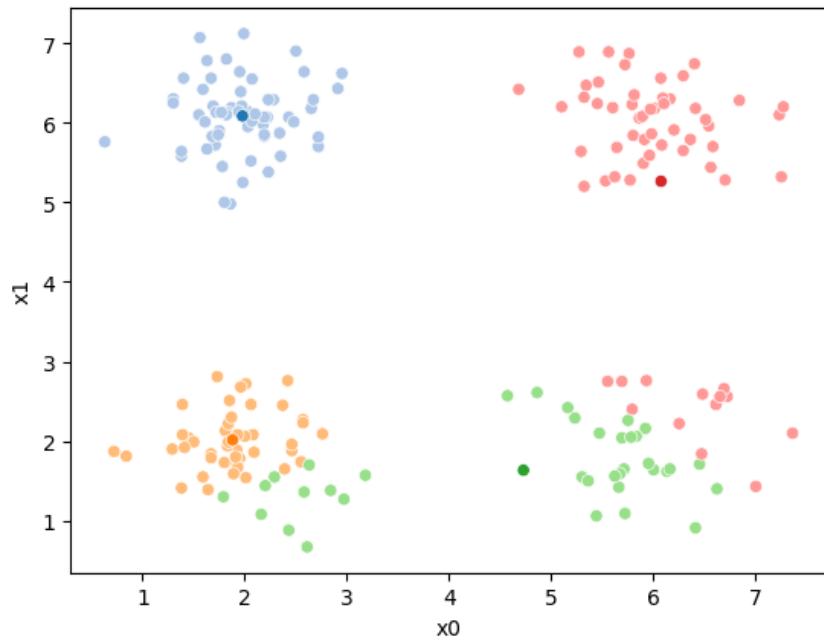
	x0	x1	true_cluster	1	2
<b>0</b>	5.27	6.89		D	3 3
<b>1</b>	1.69	6.21		B	0 0
<b>2</b>	1.96	6.39		B	0 0
<b>3</b>	6.36	5.79		D	3 3
<b>4</b>	1.92	1.89		A	2 1

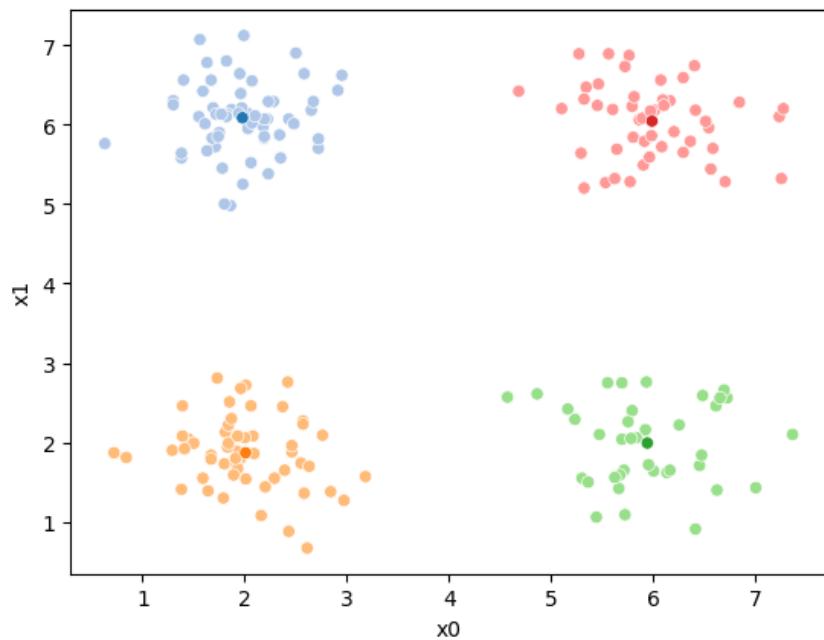
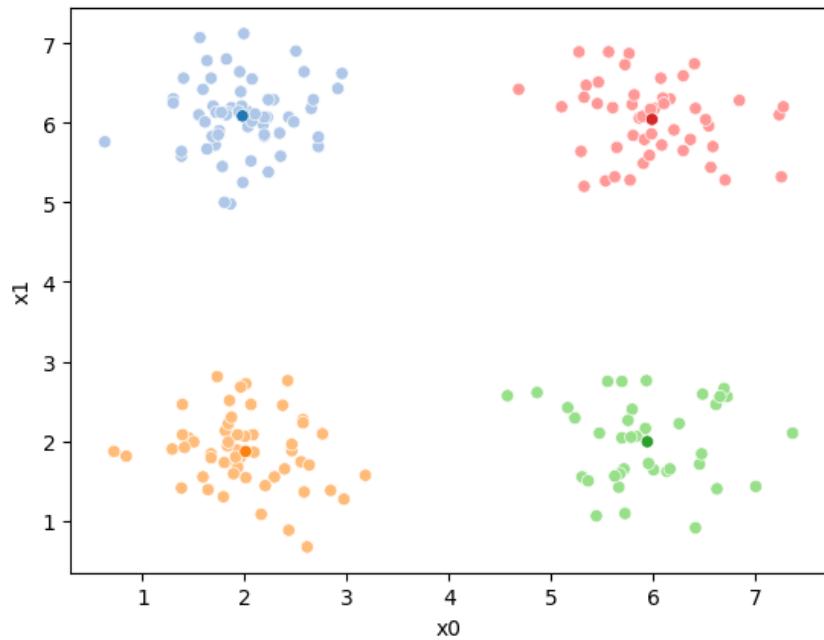
```
i = 2
mu_list = [mu_to_df(mu,1),mu_to_df(mu,i)]
cur_old = str(i-1)
cur_new = str(i)
while sum(df[cur_old] !=df[cur_new]) >0:
    cur_old = cur_new
    i +=1
    cur_new = str(i)
    #      update the means and plot with current generating assignments
    mu = df.groupby(cur_old)[data_cols].mean().values
    mu_df = mu_to_df(mu,i)
    mu_list.append(mu_df)
    fig = plt.figure()
    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)
    file_num = str(i*2 -1).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)
#    plt.plot(mu[:,0],mu[:,1],marker='s',linewidth=0)

    file_num = str(i*2).zfill(2)
    sfig.get_figure().savefig('kmeans' +file_num + '.png')

n_iter = i
```





df

	x0	x1	true_cluster	1	2	3	4
0	5.27	6.89		D	3	3	3
1	1.69	6.21		B	0	0	0
2	1.96	6.39		B	0	0	0
3	6.36	5.79		D	3	3	3
4	1.92	1.89		A	2	1	1
...	...	...		...	...	...	...
195	1.94	6.14		B	0	0	0
196	1.61	6.01		B	0	0	0
197	2.07	6.55		B	0	0	0
198	1.91	1.80		A	2	1	1
199	5.77	5.28		D	3	3	3

200 rows × 7 columns

## 16. Evaluating Clustering Solutions

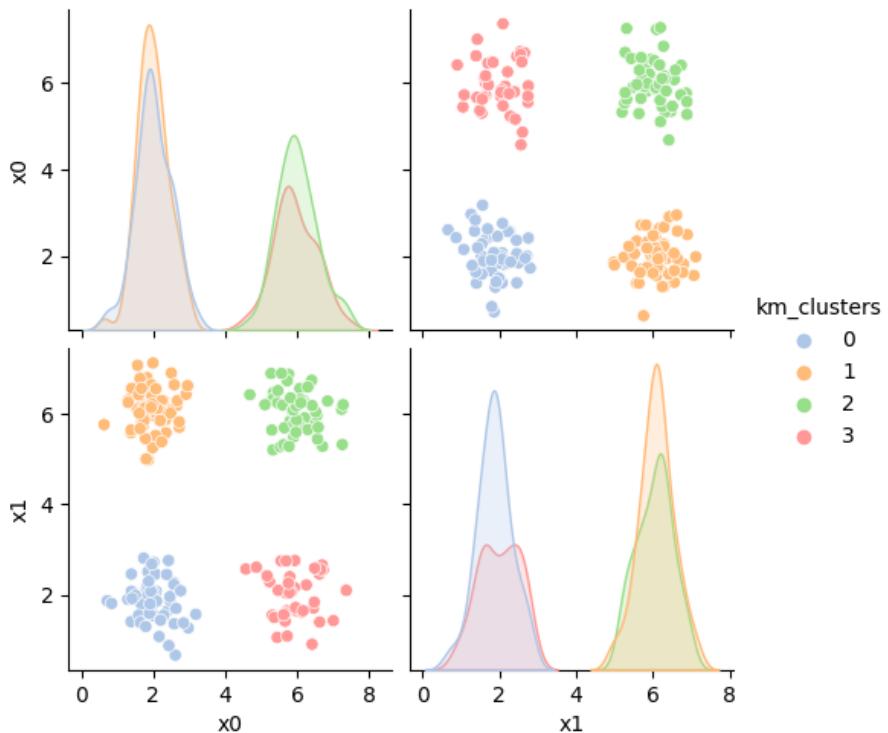
```
km4 = KMeans(n_clusters=4)
```

```
df['km_clusters'] = km4.fit_predict(df[data_cols])
```

```
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
```

```
sns.pairplot(data=df, vars=data_cols, hue='km_clusters', palette=cmap_pt)
```

```
<seaborn.axisgrid.PairGrid at 0x7f9f27491c10>
```



```
metrics.silhouette_score(df[data_cols], df['km_clusters'])
```

```
0.7670634424815012
```

```
metrics.silhouette_score(df[data_cols], df['true_cluster'])
```

```
0.7670634424815012
```

```
metrics.silhouette_score(df[data_cols], df['1'])
```

```
0.4114875650301688
```

```
metrics.silhouette_score(df[data_cols], df['2'])
```

```
0.5687778670373187
```

```
df.head()
```

[Skip to main content](#)

	x0	x1	true_cluster	1	2	3	4	km_clusters
0	5.27	6.89		D	3	3	3	3
1	1.69	6.21		B	0	0	0	0
2	1.96	6.39		B	0	0	0	0
3	6.36	5.79		D	3	3	3	3
4	1.92	1.89		A	2	1	1	1

```
metrics.adjusted_mutual_info_score(df['true_cluster'], df['km_clusters'])
```

1.0

```
metrics.adjusted_mutual_info_score(df['true_cluster'], df['1'])
```

0.7017044643558646

```
metrics.adjusted_mutual_info_score(df['true_cluster'], df['2'])
```

0.8036297838658948

## 17. Comparing Classification and Clustering Data

Datasets for classification must have a target variable observed, but we can drop it to use it for clustering

### 17.1. KMeans review

- 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

### 17.2. 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
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)
```

```
%matplotlib inline
```

[Skip to main content](#)

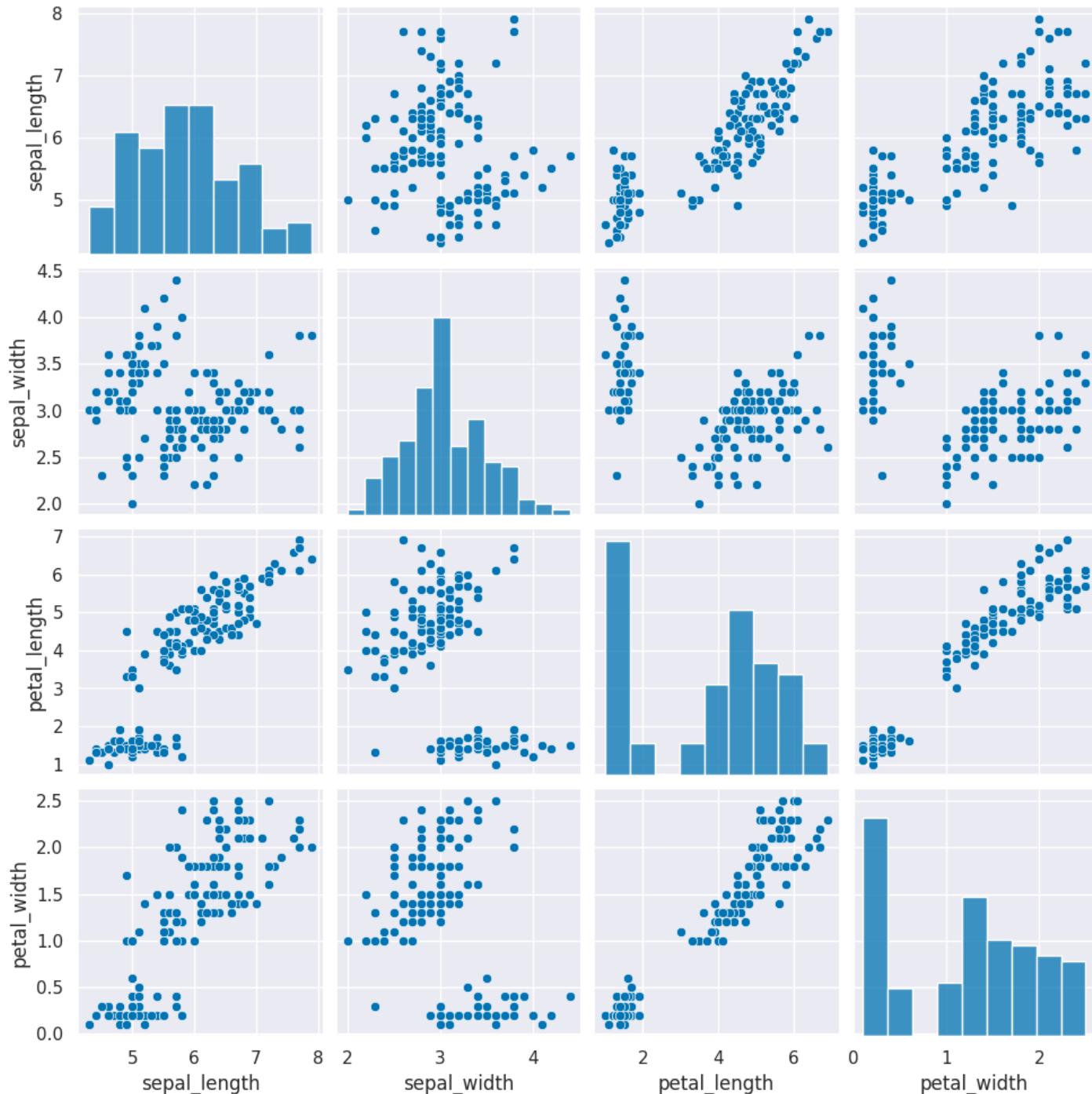
Load the iris data from seaborn

```
iris_df = sns.load_dataset('iris')
```

What plotting command will Create a grid of scatterplots of the data, without coloring the points differently

```
sns.pairplot(iris_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7f7df728c7c0>
```



[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'll do this by picking out the measurement columns, but we could also drop the species for now.

```
measurement_cols = ['sepal_length', 'petal_length', 'sepal_width', 'petal_width']
iris_X = iris_df[measurement_cols]
```

Alternate, equivalent

```
# iris_X =iris_df.drop(columns=['species']) # equivalent to above
```

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.

 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.

Use the `fit predict` method and look at what it outputs.

```
km.fit predict(iris X)
```

/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/\_kmeans.py:870: FutureWarning:  
warnings.warn(

[Skip to main content](#)

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.

Now that we know what these are, we can save them to a variable.

```
cluster_assignments = km.fit_predict(iris_X)
```

```
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:  
warnings.warn(  
"
```

Use the `get_params` method to look at the parameters. Read the documentation to see what they mean.

```
km.get_params(deep=True)
```

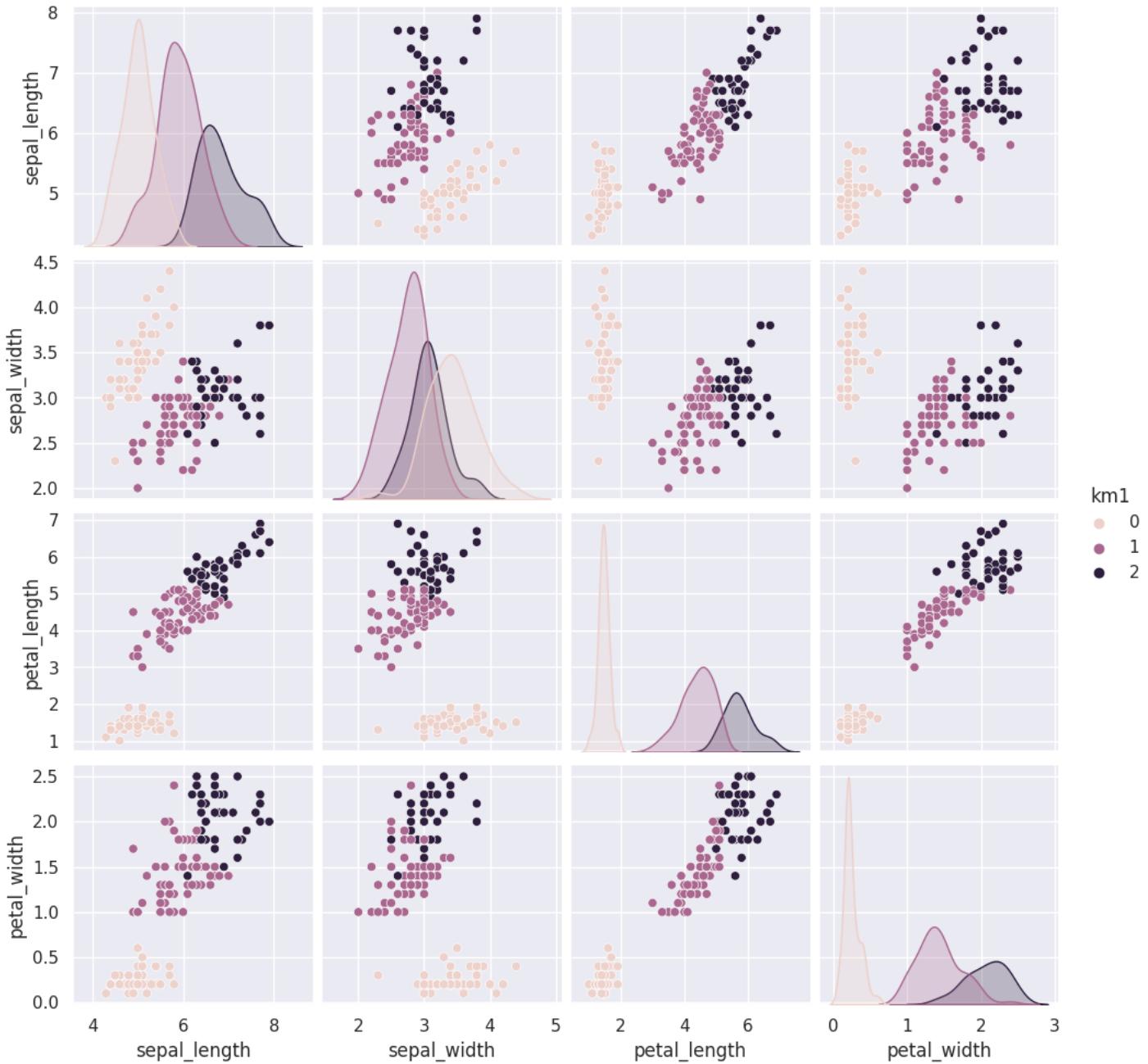
```
{'algorithm': 'lloyd',  
'copy_x': True,  
'init': 'k-means++',  
'max_iter': 300,  
'n_clusters': 3,  
'n_init': 'warn',  
'random_state': None,  
'tol': 0.0001,  
'verbose': 0}
```

## 17.3. Visualizing the outputs

Add the predictions as a new column to the original `iris_df` and make a `pairplot` with the points colored by what the clustering learned.

```
iris_df['km1'] = cluster_assignments  
sns.pairplot(data=iris_df, hue='km1')
```

```
<seaborn.axisgrid.PairGrid at 0x7f7df728c610>
```

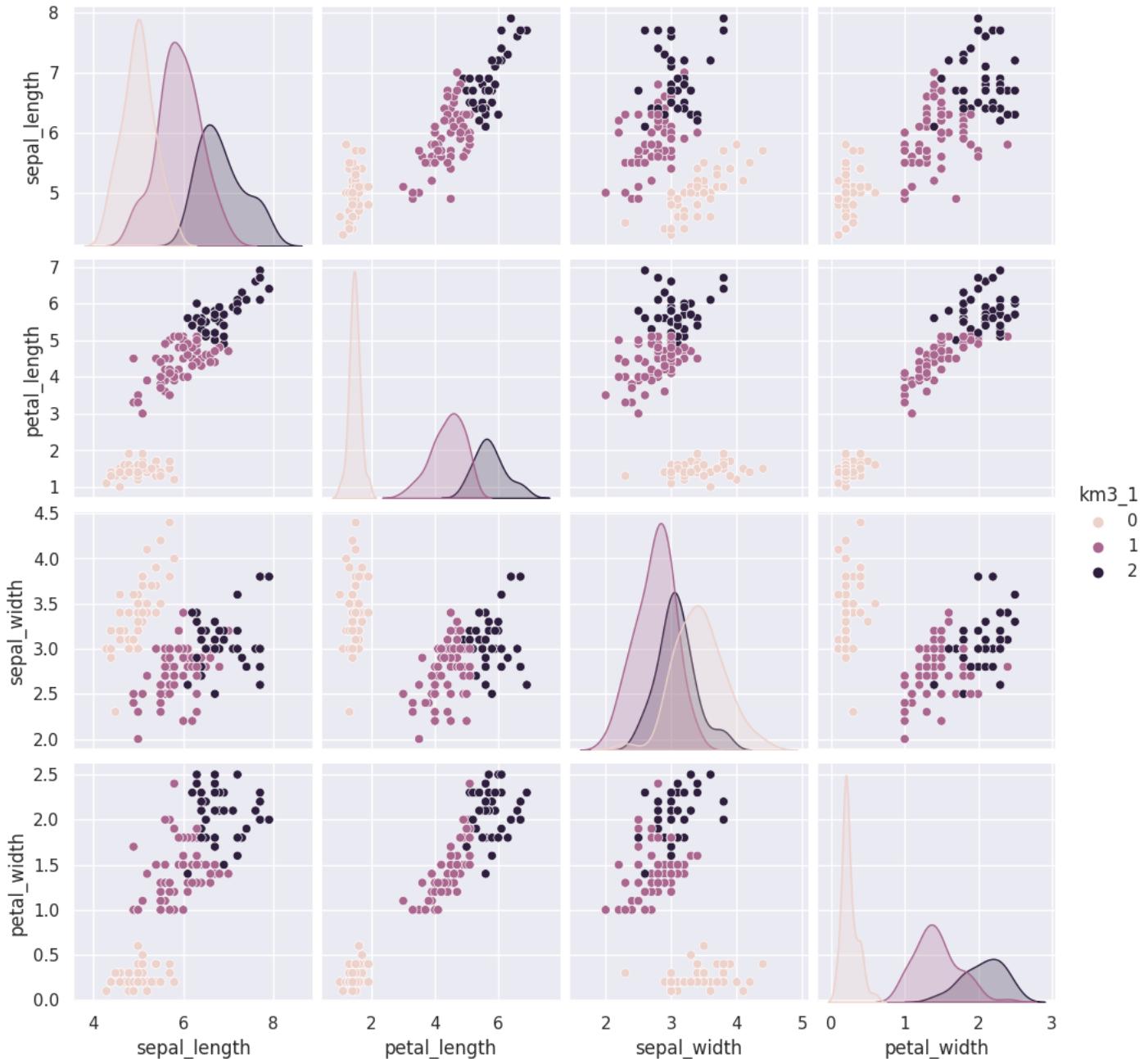


We can use the `vars` parameter to plot only the measurement columns and not the cluster labels. We didn't have to do this before, because `species` is strings, so seaborn knows to not plot it, but the cluster predictions are also numerical, so by default seaborn plots them.

```
iris_df['km3_1'] = cluster_assignments  
sns.pairplot(data=iris_df, hue='km3_1', vars=measurement_cols)
```

[Skip to main content](#)

```
<seaborn.axisgrid.PairGrid at 0x7f7df0393ee0>
```



## 17.4. Clustering Persistence

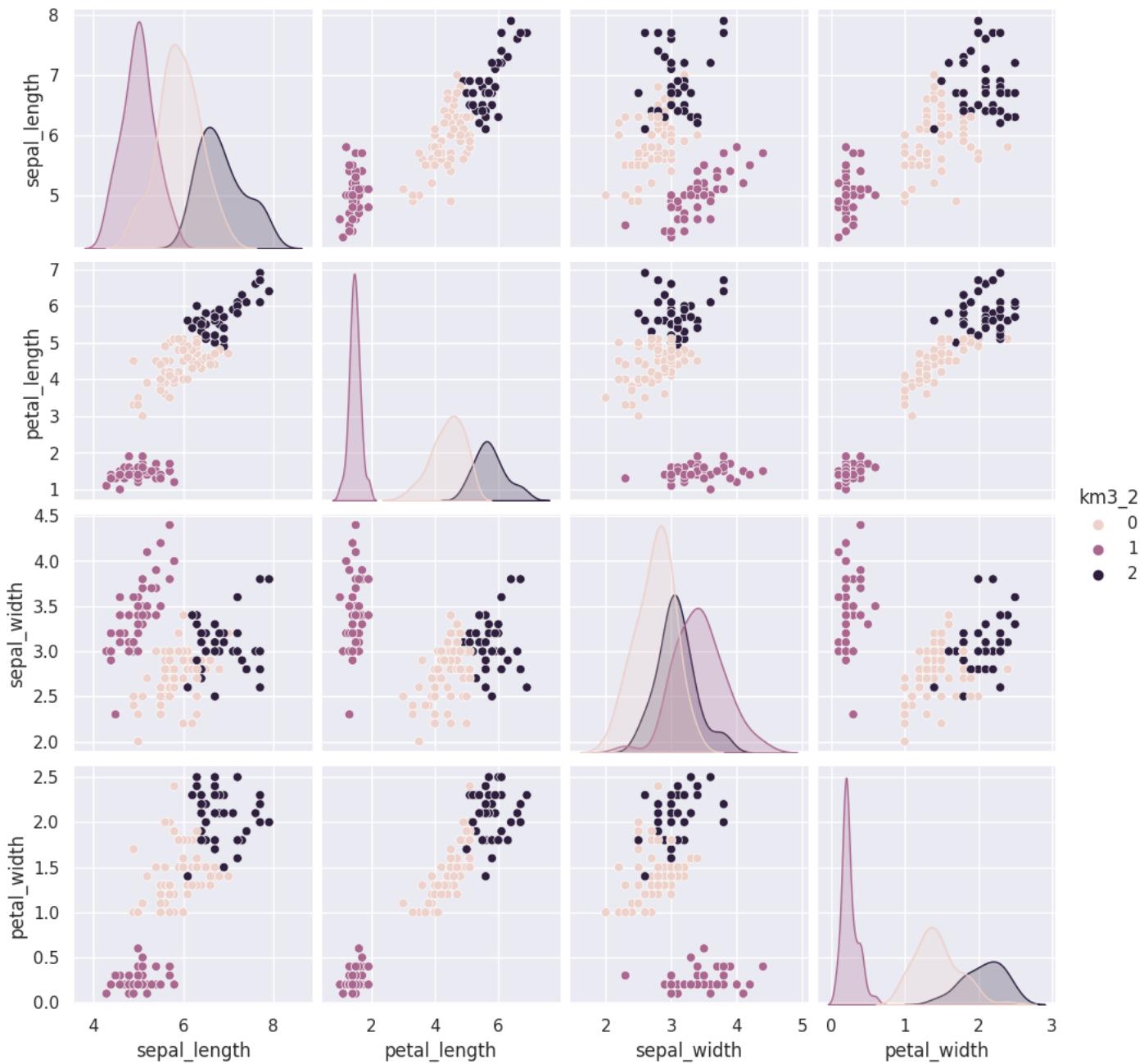
We can run kmeans a few more times and plot each time and/or compare with a neighbor/ another group.

```
iris_df['km3_2'] = km.fit_predict(iris_X)
sns.pairplot(data=iris_df, hue='km3_2', vars=measurement_cols)
```

[Skip to main content](#)

```
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:  
    warnings.warn(
```

```
<seaborn.axisgrid.PairGrid at 0x7f7de8f37b80>
```

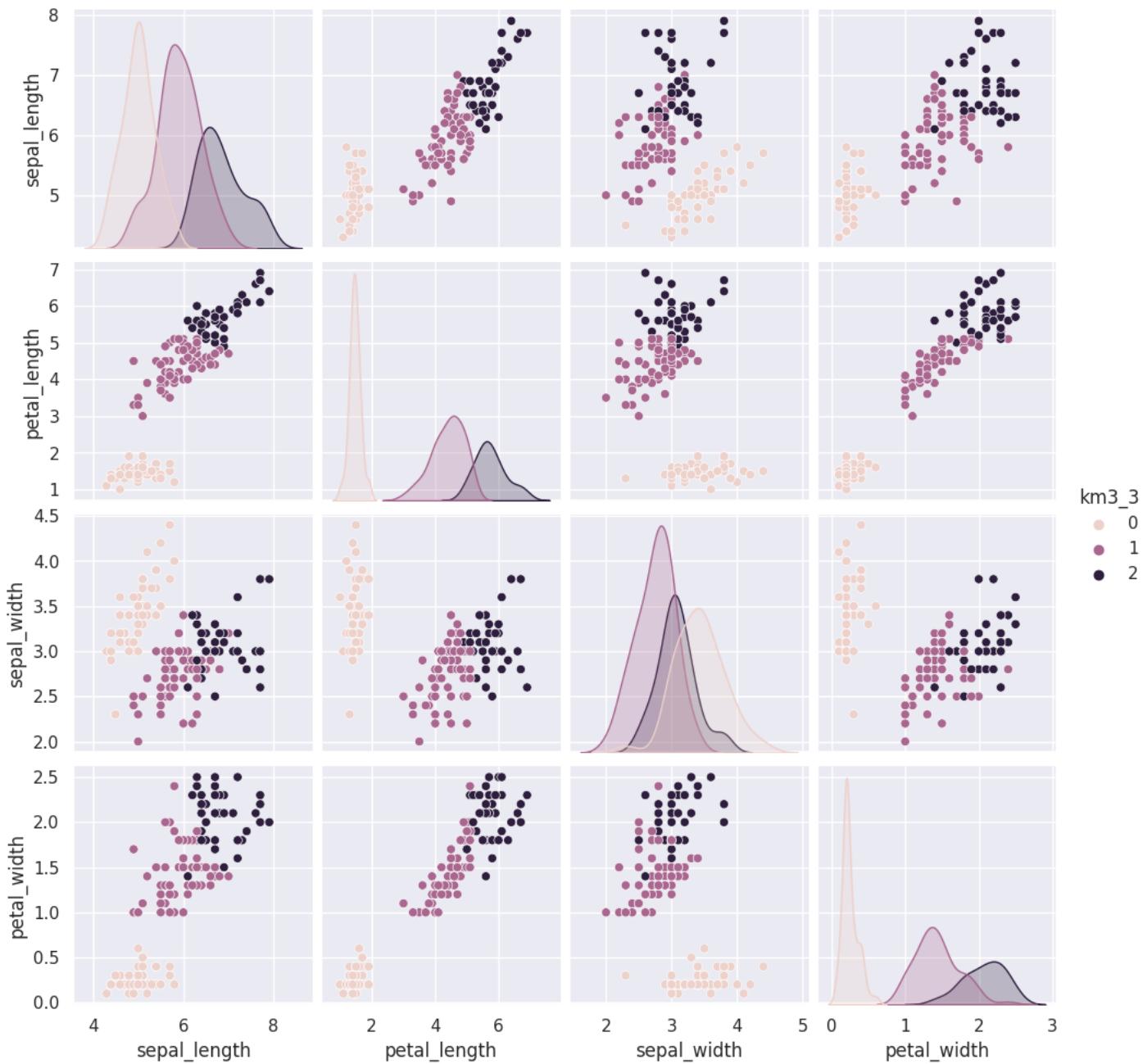


```
iris_df['km3_3'] = km.fit_predict(iris_X)  
sns.pairplot(data=iris_df, hue='km3_3', vars=measurement_cols)
```

[Skip to main content](#)

```
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:  
    warnings.warn(
```

```
<seaborn.axisgrid.PairGrid at 0x7f7de82d94c0>
```

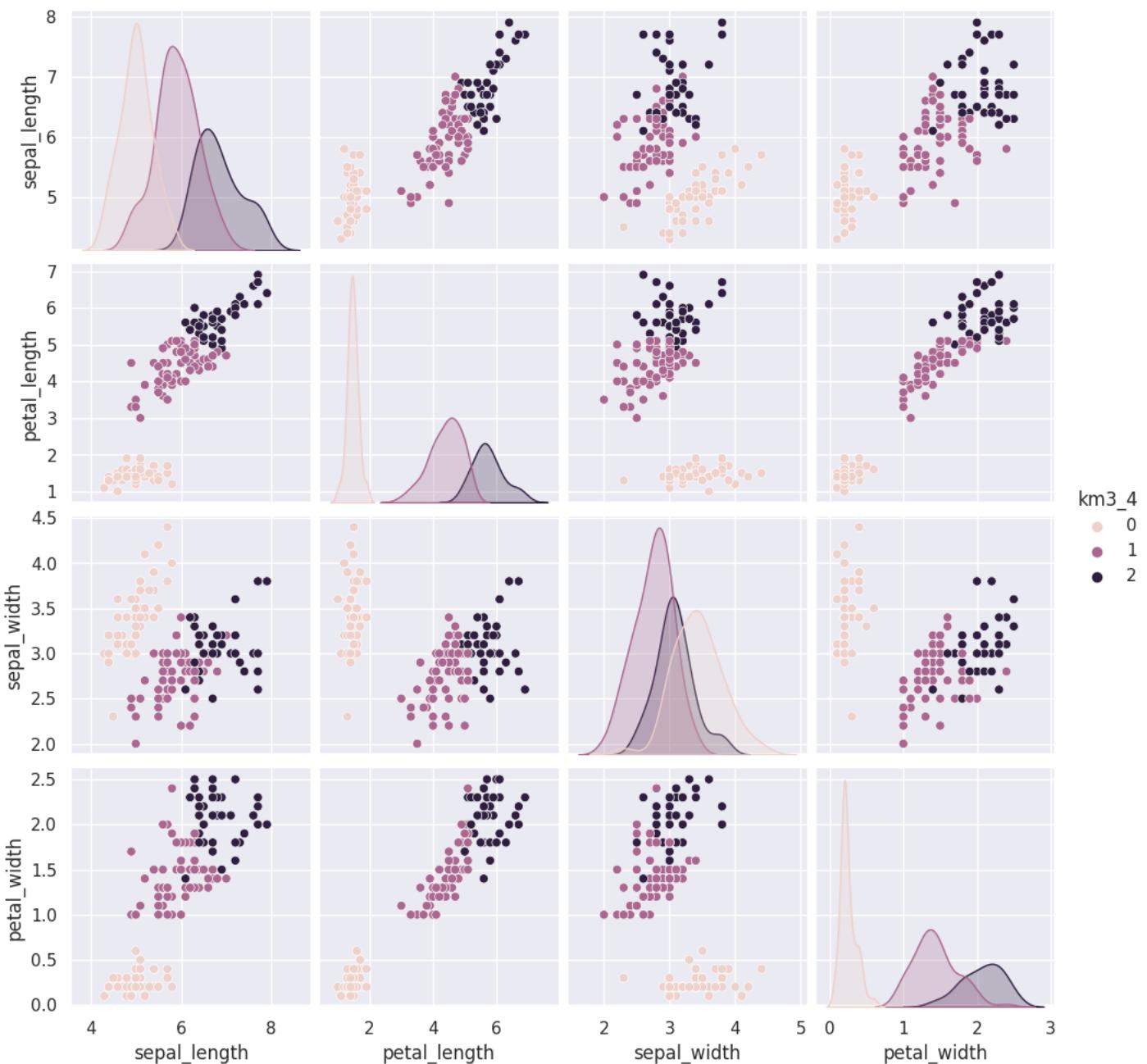


```
iris_df['km3_4'] = km.fit_predict(iris_X)  
sns.pairplot(data=iris_df, hue='km3_4', vars=measurement_cols)
```

[Skip to main content](#)

```
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:  
    warnings.warn(
```

```
<seaborn.axisgrid.PairGrid at 0x7f7de7a50940>
```



Note: (do not send this)

give them time to compare the solutions and ask here.

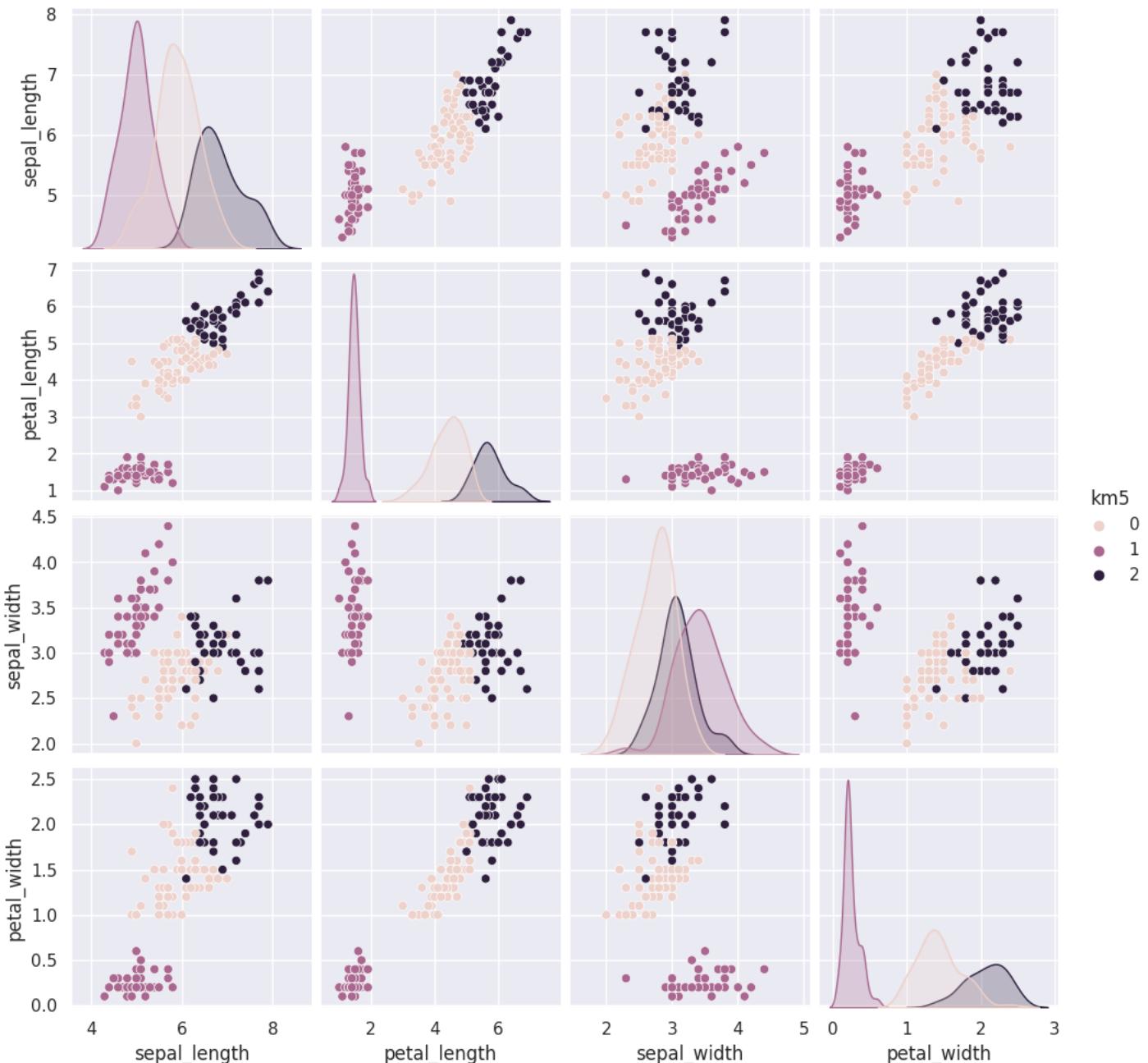
We could also use a loop (or list comprehension) to repeat kmeans multiple times.

```
for i in [5,6,7]:  
    iris_df['km' + str(i)] = km.fit_predict(iris_X)
```

[Skip to main content](#)

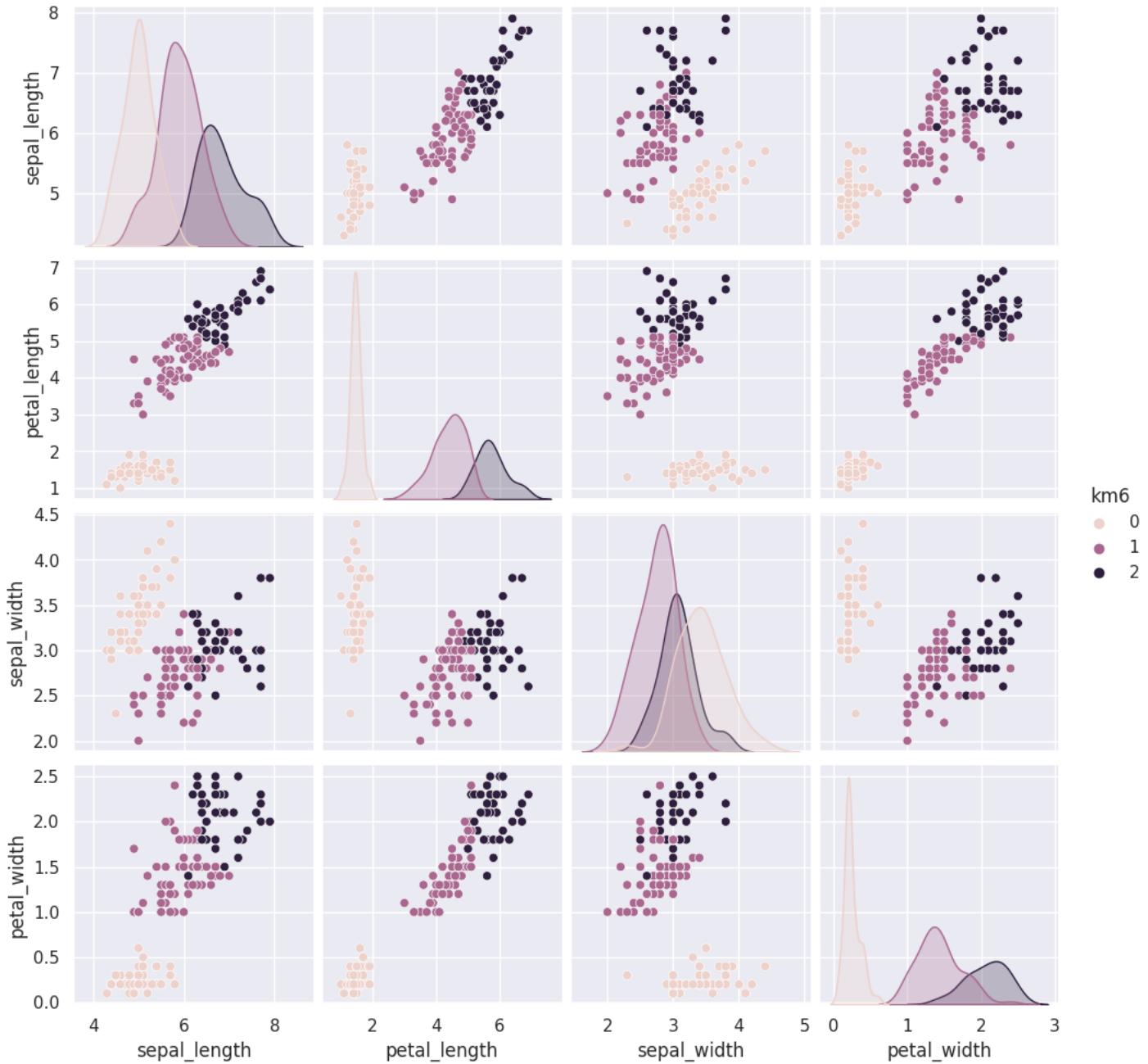
```
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:  
    warnings.warn(  
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:  
    warnings.warn(  
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:  
    warnings.warn(
```

```
<seaborn.axisgrid.PairGrid at 0x7f7df03f8eb0>
```



```
sns.pairplot(data=iris_df, hue='km6', vars=measurement_cols)
```

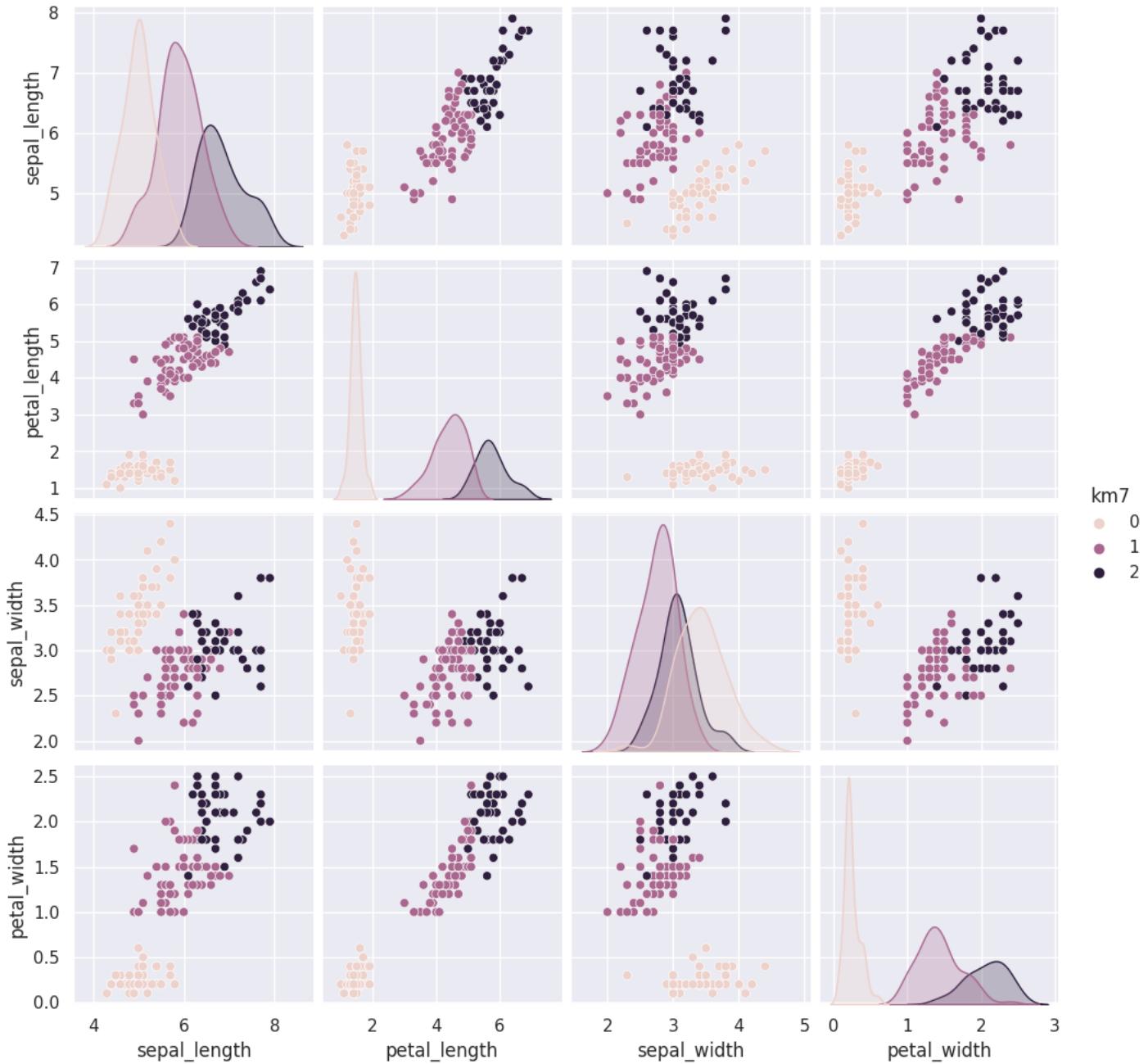
```
<seaborn.axisgrid.PairGrid at 0x7f7de6ceb370>
```



```
sns.pairplot(data=iris_df, hue='km7', vars=measurement_cols)
```

[Skip to main content](#)

```
<seaborn.axisgrid.PairGrid at 0x7f7de62d63d0>
```



The *grouping* of the points stay the same across different runs, but which color each group gets assigned to changes. Look at the 5th time compared to the ones before and 6 compared to that. Which blob is which color changes.

Today, we saw that the clustering solution was pretty similar each time in terms of which points were grouped together, but the labeling of the groups (which one was each number) was different each time. We also saw that clustering can only number the clusters, it can't match them with certainty to the species. This makes evaluating clustering somewhat different, so we need new metrics.

## 17.5. Clustering Evaluation

[Skip to main content](#)

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

Use the prismia drawing tool (the scribble next to the send button) to draw data (use two colors) that would have a silhouette score of near zero.

(solution is overlapping completely)

Draw clustering solution that would have a silhouette score above

Compute the Silhouette score for a couple iterations and show that they agree.

```
iris_df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	km1	km3_1	km3_2	km3_3	km3_4	km5	krr
0	5.1	3.5	1.4	0.2	setosa	0	0	1	0	0	1	
1	4.9	3.0	1.4	0.2	setosa	0	0	1	0	0	1	
2	4.7	3.2	1.3	0.2	setosa	0	0	1	0	0	1	
3	4.6	3.1	1.5	0.2	setosa	0	0	1	0	0	1	
4	5.0	3.6	1.4	0.2	setosa	0	0	1	0	0	1	

```
metrics.silhouette_score(iris_df[measurement_cols],iris_df['km3_1'])
```

```
0.55281901235641
```

```
metrics.silhouette_score(iris_df[measurement_cols],iris_df['km3_4'])
```

```
0.55281901235641
```

these are very consistent, but what if we try other numbers of clusters

```
km2 = KMeans(n_clusters=2)
iris_df['km2'] = km2.fit_predict(iris_X)
```

```
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
warnings.warn(
```

```
metrics.silhouette_score(iris_df[measurement_cols],iris_df['km2'])
```

```
km4 = KMeans(n_clusters=4)
iris_df['km4'] = km4.fit_predict(iris_X)
```

```
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
warnings.warn(
```

```
metrics.silhouette_score(iris_df[measurement_cols], iris_df['km4'])
```

```
0.4980505049972882
```

We see now that 2 clusters actually describes this data better, even though we were able to classify the three species better. This is a common thing to observe.

While we sometimes describe things as discrete, in nature a lot of things vary fairly continuously. Clustering works best for things that are truly discrete, but can be useful even when it is not a perfect fit.

## 17.6. Mutual Information

When we know the truth, we can see if the learned clusters are related to the true groups, we can't compare them like accuracy but we can use a metric that is intuitively like a correlation for categorical variables, the mutual information.

The `adjusted_mutual_info_score` method in the `metrics` module computes a version of mutual information that is normalized to have good properties. Apply that to the two different clustering solutions and to a solution for K=4.

```
metrics.adjusted_mutual_info_score(iris_df['species'], iris_df['km3_1'])
```

```
0.7551191675800486
```

```
metrics.adjusted_mutual_info_score(iris_df['species'], iris_df['km2'])
```

```
0.653838071376278
```

```
metrics.adjusted_mutual_info_score(iris_df['species'], iris_df['km4'])
```

```
0.7172081944051023
```

Notice here, that the true number is best but adding more clusters makes the MI very similar. If the additional cluster is one of the true ones split, it can still find the original boundaries and so it will still get a high score.

Other types of clustering: sklearn overview

classifier comparison

## 17.7. Questions

17.7.1. Is there a way to see all of the seaborn palettes?

17.7.2. Can you clarify how `fit_predict` works?

It is equivalent to calling both the `fit` and the `predict` on the same samples.

17.7.3. I'm very interested in neural networks. I would love to have more lectures on it

We will!

17.7.4. What types of data is clustering not very useful?

Groups that are not separate

17.7.5. Was this the formula for  $s = \frac{b-a}{\max(a,b)}$  how the silhouette score is computed?

yes

## 18. Regression Preview

### 18.1. Preview Regression

- the beginning of sklearn's regression tutorial carefully, and skim the rest of it.
- the linear regression example. cut this code into multiple cells and run it, looking at interim points.
- the example page from this stat course

### 18.2. Be ready for Thursday

- what are the assumptions of regression generally?
- what are the specific assumptions of linear regression?
- what does data for regression need to have in terms of variable types?
- what are at least three example applications of regression that are not included in the resources above?

## 19. Regression

```
%matplotlib inline
```

```
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')
```

```
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

Split the data to use 80% of the data to train a model to predict the tip from the total bill.

```
tips_X = tips_df['total_bill']
tips_y = tips_df['tip']
```

```
type(tips_X.values)
```

```
numpy.ndarray
```

```
tips_X.values.shape
```

```
(244, )
```

```
tips_X.values[:,np.newaxis].shape
```

```
(244, 1)
```

```
tips_X = tips_X.values[:,np.newaxis]
tips_X_train,tips_X_test, tips_y_train, tips_y_test = train_test_split(tips_X, tips_y,
train_size=.8)
```

```
regr = linear_model.LinearRegression()
```

[Skip to main content](#)

```
type(tips_X)
```

```
numpy.ndarray
```

```
regr.fit(tips_X_train,tips_y_train)
```

```
▼ LinearRegression  
LinearRegression()
```

```
regr.get_params()
```

```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'positive': False}
```

```
regr.coef_
```

```
array([0.1098025])
```

```
regr.intercept_
```

```
0.8588487994455964
```

```
tips_y_pred = regr.predict(tips_X_test)
```

```
tips_y_pred[0]
```

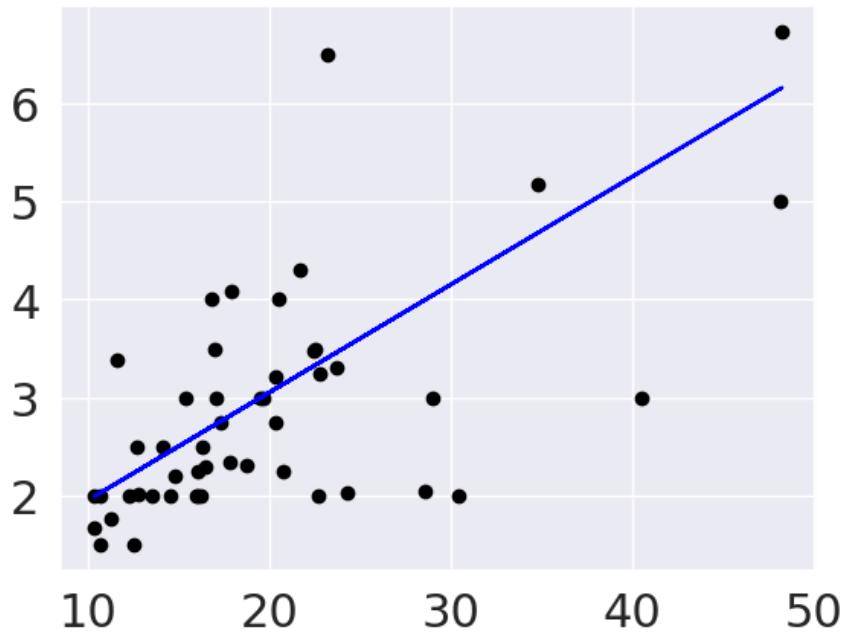
```
3.9937102246507266
```

```
tips_X_test[0]*regr.coef_ + regr.intercept_
```

```
array([3.99371022])
```

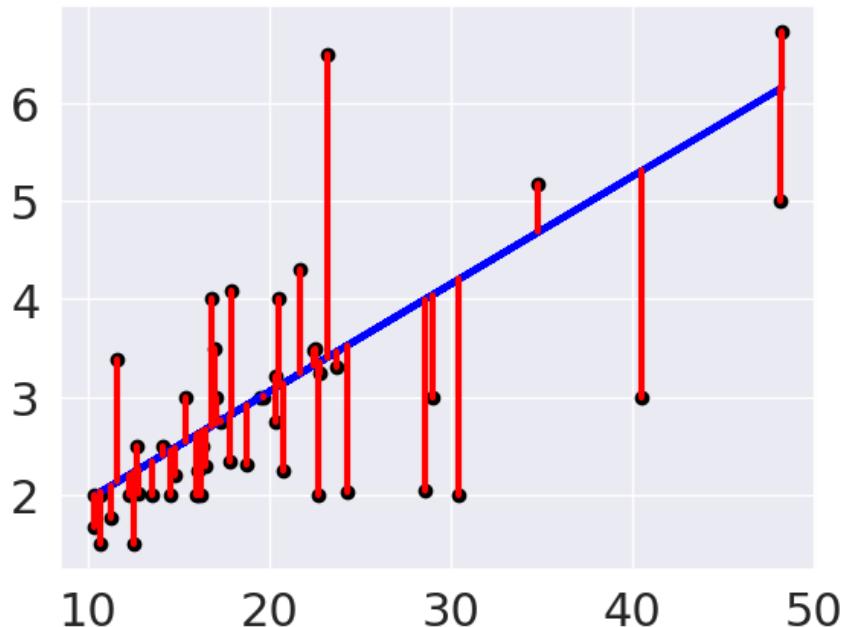
```
plt.scatter(tips_X_test,tips_y_test, color='black')  
plt.plot(tips_X_test,tips_y_pred, color='blue')
```

```
[<matplotlib.lines.Line2D at 0x7f5126f19eb0>]
```



```
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,yp], color='red', linewidth=3)  
     for x, yp, yt in zip(tips_X_test, tips_y_pred,tips_y_test)];  
plt.scatter(tips_X_test, tips_y_test, color='black')
```

```
<matplotlib.collections.PathCollection at 0x7f5124d63ee0>
```



```
mean_squared_error(tips_y_test,tips_y_pred)
```

```
0.8745320879785059
```

```
tips_y_test.mean()
```

```
2.8685714285714288
```

```
mean_squared_error(tips_y_test,tips_y_pred)/tips_y_test.mean()
```

```
0.3048667637375269
```

```
r2_score(tips_y_test,tips_y_pred)
```

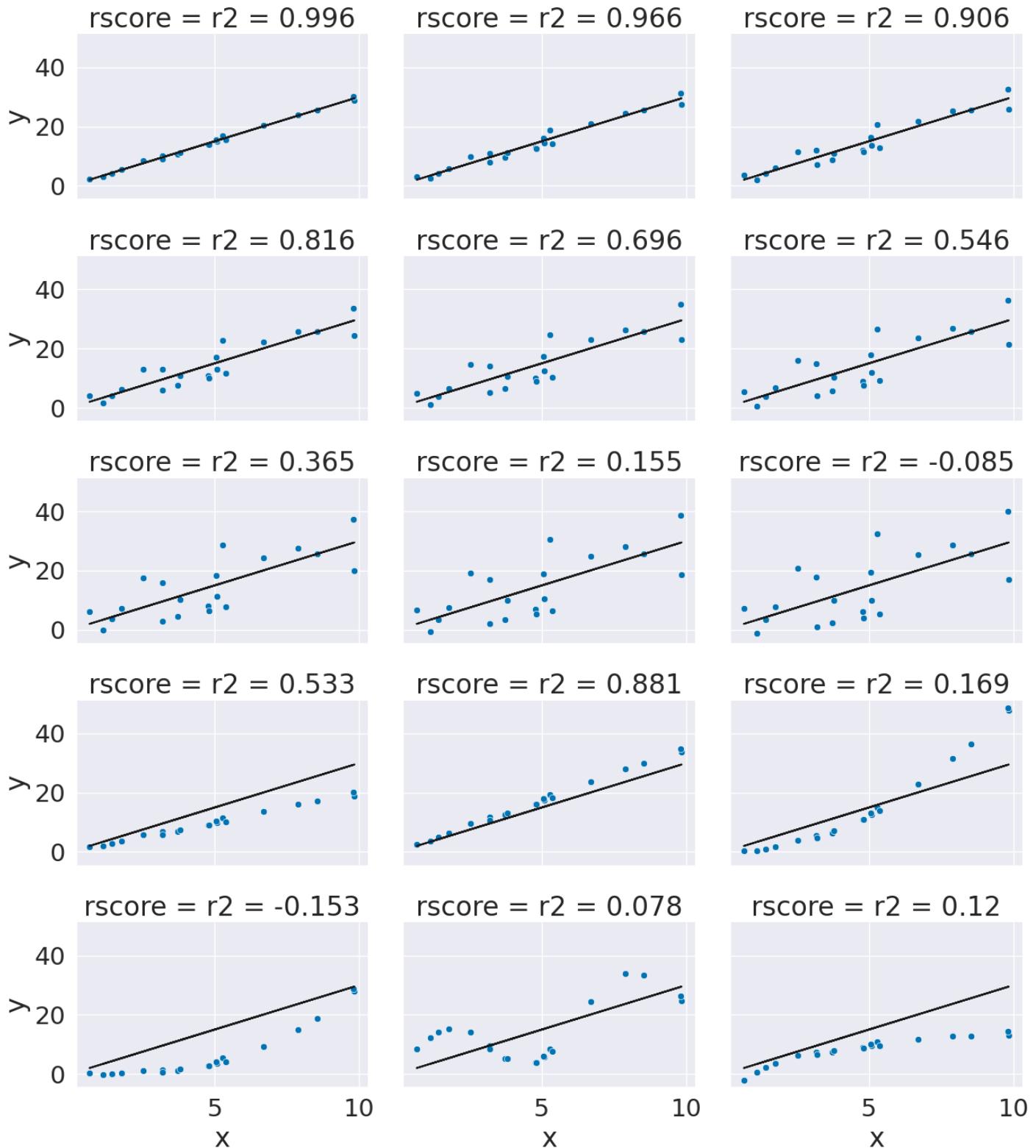
```
0.3374142265679132
```

```
x = 10*np.random.random(20)
y_pred = 3*x
ex_df = pd.DataFrame(data = x,columns = ['x'])
ex_df['y_pred'] = y_pred
n_levels = range(1,18,2)
noise = (np.random.random(20)-.5)*2
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

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)]
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

xy_df = ex_df.melt(id_vars=['x','y_pred'],var_name='rscore',value_name='y')
# sns.lmplot(x='x',y='y', data = xy_df,col='rscore',col_wrap=3,)
g = sns.FacetGrid(data = xy_df,col='rscore',col_wrap=3,aspect=1.5,height=3)
g.map(plt.plot, 'x','y_pred',color='k')
g.map(sns.scatterplot, "x", "y",)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f5124d20610>
```



```
regr.score(tips_X_test,tips_y_test)
```

```
0.3374142265679132
```

[Skip to main content](#)

```
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

```
tips_X2 = tips_df[['total_bill', 'size']]  
tips_X2_train, tips_X2_test, tips_y2_train, tips_y2_test = train_test_split(tips_X2, tips_y,  
                           train_size=.8)
```

```
regr2 = linear_model.LinearRegression()  
regr2.fit(tips_X2_train, tips_y2_train)
```

```
▼ LinearRegression  
LinearRegression()
```

```
regr2.coef_
```

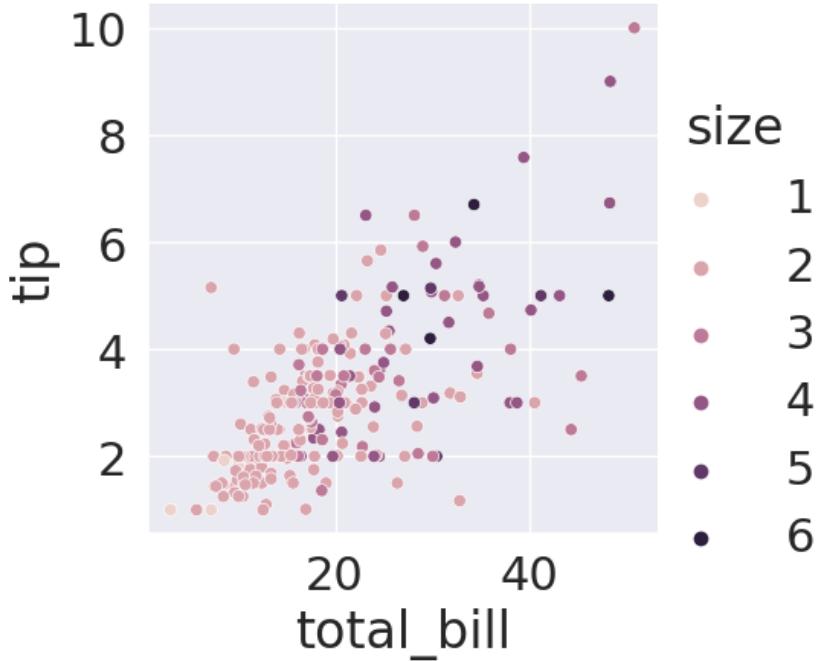
```
array([0.09773974,  0.14412751])
```

```
regr2.score(tips_X2_test, tips_y2_test)
```

```
0.4505793675784008
```

```
sns.relplot(data=tips_df, x='total_bill', y='tip',  
            hue='size')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f51241753d0>
```



```
from sklearn import datasets
```

```
# 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)
```

```
regr_db = linear_model.LinearRegression()
regr_db.fit(X_train, y_train)
```

```
▼ LinearRegression
LinearRegression()
```

```
y_pred = regr_db.predict(X_test)
r2_score(y_test,y_pred)
```

```
0.5195333332288746
```

```
diabetes_X.shape
```

```
(442, 10)
```

```
lasso = linear_model.Lasso()
lasso.fit(X_train, y_train)
lasso.score(X_test,y_test)
```

[Skip to main content](#)

```
0.42249260665177746
```

```
lasso = linear_model.Lasso(.5)
lasso.fit(X_train, y_train)
lasso.score(X_test,y_test)
```

```
0.527228369284263
```

```
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.          ])
```

link to polynomial regression

## 20. ML Task Review and Cross Validation

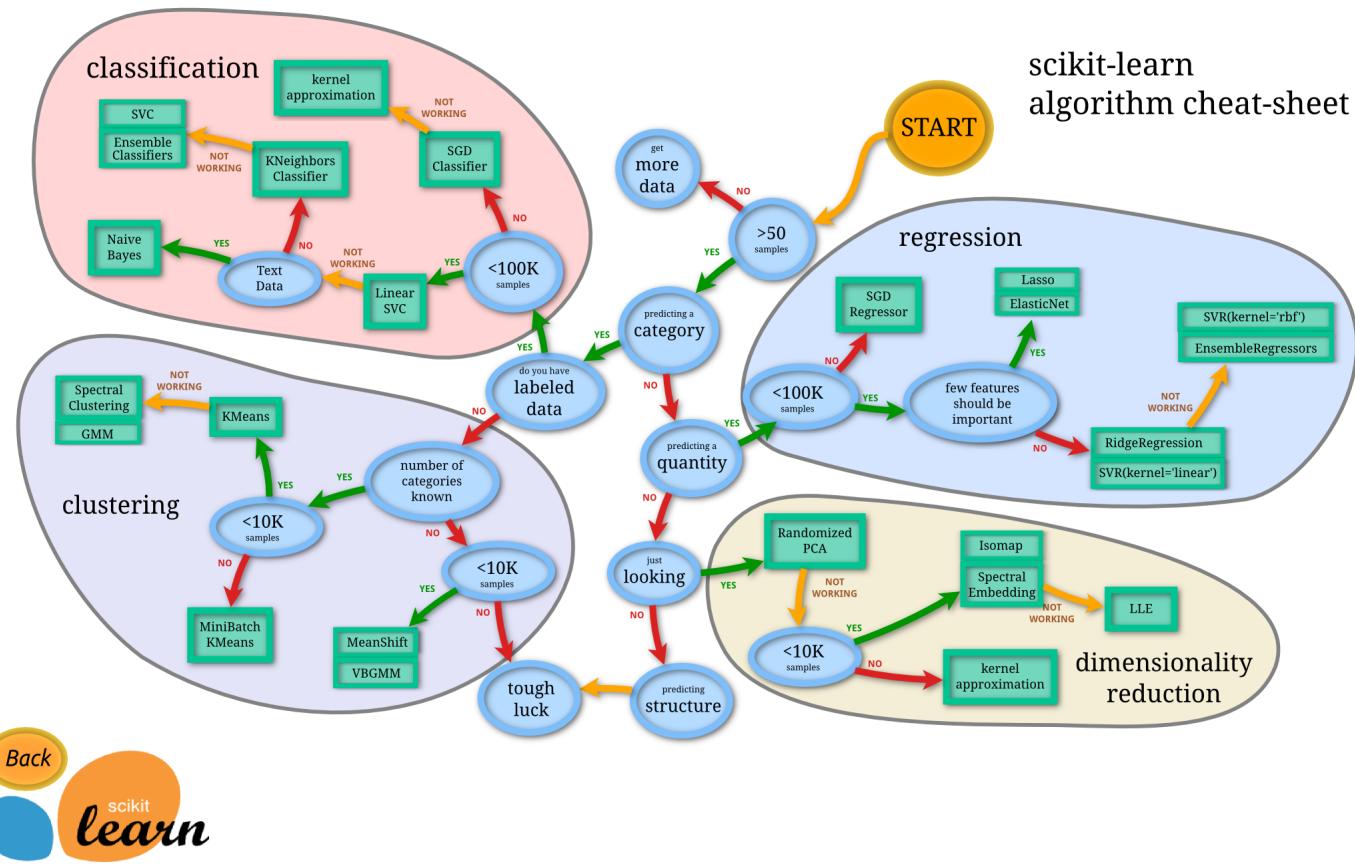
### 20.1. Relationship between Tasks

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.

Sklearn provides a nice flow chart for thinking through this.



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 and the last portfolios, 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.

### Further Reading

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.

## 20.2. 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.

We have seen that the test train splits, which are random, influence the performance.

```
# 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 import metrics
```

We'll use the Iris data with a decision tree.

```
iris_df = sns.load_dataset('iris')

iris_X = iris_df.drop(columns=['species'])
iris_y = iris_df['species']
```

```
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.

```
iris_X_train, iris_X_test, iris_y_train, iris_y_test = train_test_split(iris_X,iris_y)
dt.fit(iris_X_train,iris_y_train)
dt.score(iris_X_test,iris_y_test)
```

```
0.9736842105263158
```

Since it is random, if we repeat this, we will generally get a different value

```
iris_X_train, iris_X_test, iris_y_train, iris_y_test = train_test_split(iris_X,iris_y)
dt.fit(iris_X_train,iris_y_train)
dt.score(iris_X_test,iris_y_test)
```

```
0.8947368421052632
```

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_train,iris_y_train)
```

```
array([1.        , 0.91304348, 0.90909091, 1.        , 0.95454545])
```

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.

We get back a score for each section or "fold" of the data. We can average those to get a single estimate.

```
cross_val_score(dt, iris_X_train,iris_y_train).mean()
```

```
0.9553359683794467
```

We can change it to 10-fold.

```
cross_val_score(dt, iris_X_train,iris_y_train,cv=10)
```

```
array([1.        , 1.        , 0.90909091, 0.90909091, 0.90909091,
       1.        , 1.        , 1.        , 1.        , 0.90909091])
```

```
cross_val_score(dt, iris_X_train,iris_y_train,cv=10).mean()
```

```
0.9727272727272727
```

## 20.3. What Does Cross validation really do?

### ! Important

This is extra detail that was not presented in class.

It uses StratifiedKFold for classification, but since we're using regression it will use `KFold`. `test_train_split` uses `ShuffleSplit` by default, let's load that too to see what it does.

## ⚠ Warning

The key in the following is to get the *concepts* not all of the details in how I evaluate and visualize. I could have made figures separately to explain the concept, but I like to show that Python is self contained.

```
from sklearn.model_selection import KFold, ShuffleSplit
```

```
kf = KFold(n_splits = 10)
```

When we use the `split` method it gives us a generator.

```
kf.split(diabetes_X, diabetes_y)
```

```
-----  
NameError                                                 Traceback (most recent call last)  
Cell In[12], line 1  
----> 1 kf.split(diabetes_X, diabetes_y)  
  
NameError: name 'diabetes_X' is not defined
```

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.

```
N_samples = len(diabetes_y)  
kf_tt_df = pd.DataFrame(index=list(range(N_samples)))  
i = 1  
for train_idx, test_idx in kf.split(diabetes_X, diabetes_y):  
    kf_tt_df['split ' + str(i)] = ['unused']*N_samples  
    kf_tt_df['split ' + str(i)][train_idx] = 'Train'  
    kf_tt_df['split ' + str(i)][test_idx] = 'Test'  
    i +=1
```

```
-----  
NameError                                                 Traceback (most recent call last)  
Cell In[13], line 1  
----> 1 N_samples = len(diabetes_y)  
      2 kf_tt_df = pd.DataFrame(index=list(range(N_samples)))  
      3 i = 1  
  
NameError: name 'diabetes_y' is not defined
```

We can count how many times 'Test' and 'Train' appear

```
count_test = lambda part: len([v for v in part if v=='Test'])  
count_train = lambda part: len([v for v in part if v=='Train'])
```

When we apply this along `axis=1` we to check that each sample is used exactly 1 time

```
sum(kf_tt_df.apply(count_test, axis = 1) ==1)
```

How would  
actual test

```
NameError Traceback (most recent call last)
Cell In[15], line 1
----> 1 sum(kf_tt_df.apply(count_test, axis = 1) ==1)

NameError: name 'kf_tt_df' is not defined
```

and exactly 9 training sets

```
sum(kf_tt_df.apply(count_test, axis = 1) ==9)
```

```
NameError Traceback (most recent call last)
Cell In[16], line 1
----> 1 sum(kf_tt_df.apply(count_test, axis = 1) ==9)

NameError: name 'kf_tt_df' is not defined
```

the describe helps ensure that all fo the values are exa

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=1, linecolor='gray')
colorbar = g.collections[0].colorbar
colorbar.set_ticklabels(['Train','Test'])
```

```
NameError Traceback (most recent call last)
Cell In[17], line 2
  1 cmap = sns.color_palette("tab10",10)
----> 2 g = sns.heatmap(kf_tt_df.replace({'Test':1,'Train':0}),cmap=cmap[7:9],cbar_kws={'ticks':[.25,.75]}, linewidths=1, linecolor='gray')
  3 colorbar = g.collections[0].colorbar
  4 colorbar.set_ticklabels(['Train','Test'])

NameError: name 'kf_tt_df' is not defined
```

Note that unlike `test_train_split` this does not always randomize and shuffle the data before splitting.

If we apply those `lambda` functions along `axis=0`, we can see the size of each test set

```
kf_tt_df.apply(count_test, axis = 0)
```

```
NameError Traceback (most recent call last)
Cell In[18], line 1
----> 1 kf_tt_df.apply(count_test, axis = 0)

NameError: name 'kf_tt_df' is not defined
```

and training set:

[Skip to main content](#)

Tip  
sns.  
string p

```
kf_tt_df.apply(count_train, axis = 0)
```

```
-----  
NameError                                                 Traceback (most recent call last)  
Cell In[19], line 1  
----> 1 kf_tt_df.apply(count_train, axis = 0)  
  
NameError: name 'kf_tt_df' is not defined
```

We can verify that these splits are the same size as what `test_train_split` does using the right settings. 10-fold splits the data into 10 parts and tests on 1, so that makes a test size of  $1/10=.$ .1, so we can use the `train_test_split` and check the length.

```
X_train2,X_test2, y_train2,y_test2 = train_test_split(diabetes_X, diabetes_y ,  
                                                    test_size=.1,random_state=0)  
  
[len(split) for split in [X_train2,X_test2,]]
```

Under the hood `train_test_split` uses `ShuffleSplit`. We can do a similar experiment as above to see what `ShuffleSplit` does.

```
skf = ShuffleSplit(10)  
N_samples = len(diabetes_y)  
ss_tt_df = pd.DataFrame(index=list(range(N_samples)))  
i = 1  
for train_idx, test_idx in skf.split(diabetes_X, diabetes_y):  
    ss_tt_df['split ' + str(i)] = ['unused']*N_samples  
    ss_tt_df['split ' + str(i)][train_idx] = 'Train'  
    ss_tt_df['split ' + str(i)][test_idx] = 'Test'  
    i +=1  
  
ss_tt_df
```

```
-----  
NameError                                                 Traceback (most recent call last)  
Cell In[20], line 2  
----> 1 skf = ShuffleSplit(10)  
      2 N_samples = len(diabetes_y)  
      3 ss_tt_df = pd.DataFrame(index=list(range(N_samples)))  
      4 i = 1  
  
NameError: name 'diabetes_y' is not defined
```

And plot

```
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]}, 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]}, l:  
      3     linecolor='gray')  
      4 colorbar = g.collections[0].colorbar  
      5 colorbar.set_ticklabels(['Train', 'Test'])  
  
NameError: name 'ss_tt_df' is not defined
```

## 20.4. Cross validation with clustering

We can use *any* estimator object here.

```
km = cluster.KMeans(n_clusters=3)
```

```
cross_val_score(km, iris_X_train, )
```

```
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:  
    warnings.warn(  
/opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:  
    warnings.warn(
```

```
array([-16.69612641, -10.94944732, -14.44301708, -12.71019484,  
       -12.32547752])
```

```
km.score()
```

```
-----  
TypeError Traceback (most recent call last)  
Cell In[24], line 1  
----> 1 km.score()  
  
TypeError: score() missing 1 required positional argument: 'x'
```

## 20.5. Grid Search Optimization

We can optimize, however to determining the different parameter settings.

A simple way to do this is to fit the model for different parameters and score for each and compare.

```
param_grid = {'n_clusters':[2,3,4,5,6]}  
km_opt = GridSearchCV(km, param_grid, metrics.silhouette_score)
```

[Skip to main content](#)

```
--> TypeError
Cell In[25], line 2
    1 param_grid = {'n_clusters':[2,3,4,5,6]}
-> 2 km_opt = GridSearchCV(km, param_grid,metrics.silhouette_score)
TypeError: __init__() takes 3 positional arguments but 4 were given
```

The `GridSearchCV` object is constructed first and requires an estimator object and a dictionary that describes the parameter grid to search over. The dictionary has the parameter names as the keys and the values are the values for that parameter to test.

The `fit` method on the Grid Search object fits all of the separate models.

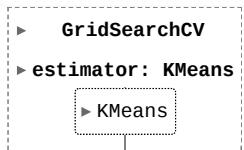
In this case we optimize of a one dimensional “grid” just a set of values for one parameter, the number of clusters.

```
param_grid = {'n_clusters':[2,3,4,5,6]}
km_opt = GridSearchCV(km, param_grid)
```

```
iris_X_train.shape
```

```
(112, 4)
```

```
km_opt.fit(iris_X_train)
```



## Important

I still need to explore this question. A volunteer who wants to do this for a portfolio section can do that as well

```
km_opt.best_params_
```

```
{'n_clusters': 6}
```

```
type(km_opt.best_estimator_)
```

```
sklearn.cluster._kmeans.KMeans
```

We note that this is a dictionary, so to make it more readable, we can make it a DataFrame.

```
pd.DataFrame(km_opt.cv_results_)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_clusters	params	split0_test_score
0	0.005939	0.000320	0.001145	0.000038	2	{'n_clusters': 2}	-34.91315
1	0.008089	0.000566	0.001132	0.000011	3	{'n_clusters': 3}	-16.69612
2	0.009690	0.000513	0.001118	0.000022	4	{'n_clusters': 4}	-11.70108
3	0.010558	0.000165	0.001106	0.000007	5	{'n_clusters': 5}	-10.19888
4	0.012072	0.000482	0.001101	0.000009	6	{'n_clusters': 6}	-8.75269

## 20.6. Optimizing a Decision Tree

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 we have seen some before. Max depth is the height of the tree and min samples per leaf makes it keeps the leaf sizes small.

```
dt = tree.DecisionTreeClassifier()
params_dt = {'criterion':['gini', 'entropy'], 'max_depth':[2,3,4],
             'min_samples_leaf':list(range(2,20,2))}
```

what parameters give the highest accuracy? and is the most accurate one also the fastest one?

```
dt_opt = GridSearchCV(dt,params_dt)
dt_opt.fit(iris_X_train,iris_y_train)
```

```
► GridSearchCV
  ► estimator: DecisionTreeClassifier
    ► DecisionTreeClassifier
```

We will fit it with default CV settings. And we can see the best parameters

```
dt_opt.best_params_
```

```
{'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 2}
```

and we can use `ti` to get predictions

```
y_pred = dt_opt.predict(iris_X_test)
```

```
dt_df = pd.DataFrame(dt_opt.cv_results_)
dt_df.shape
```

```
(54, 16)
```

```
dt_df.columns
```

```
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_criterion', 'param_max_depth', 'param_min_samples_leaf',
       'params', 'split0_test_score', 'split1_test_score', 'split2_test_score',
       'split3_test_score', 'split4_test_score', 'mean_test_score',
       'std_test_score', 'rank_test_score'],
      dtype='object')
```

```
dt_df['mean_score_time'].idxmin() == dt_df['mean_test_score'].idxmax()
```

```
False
```

```
dt_df['mean_test_score'].idxmax(), dt_df['mean_score_time'].idxmin()
```

```
(0, 19)
```

### ! 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
```

If we have 500 samples and use `train_size=.8` in `train_test_split` and the default values for `GridSearchCV`, how many samples are in each validation set?

```
N = 500
train_size = .8
cv = 5
```

```
train_size*N/cv
```

```
80.0
```

```
iris_X, iris_y = datasets.load_iris(return_X_y=True)
```

```
iris_X_train, iris_X_test, iris_y_train, iris_y_test = train_test_split(iris_X,iris_y)
```

## 21.1. Fitting an SVM

Now let's compare with a different model, we'll use the parameter optimized version for that model.

the sklearn docs have a good description

```
svm_clf = svm.SVC()
```

```
param_grid = {'kernel':['linear','rbf'], 'C':[.5, 1, 10]}
```

```
svm_opt = GridSearchCV(svm_clf,param_grid)
svm_opt.fit(iris_X_train,iris_y_train)
```

```
► GridSearchCV
► estimator: SVC
  ► SVC
```

```
svm_df = pd.DataFrame(svm_opt.cv_results_)
```

[Skip to main content](#)

```
svm_df
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_kernel	params	split0_test_size
0	0.000839	0.000202	0.000501	0.000067	0.5	linear	{'C': 0.5, 'kernel': 'linear'}	1
1	0.000801	0.000010	0.000516	0.000021	0.5	rbf	{'C': 0.5, 'kernel': 'rbf'}	1
2	0.000711	0.000012	0.000463	0.000011	1	linear	{'C': 1, 'kernel': 'linear'}	1
3	0.000770	0.000018	0.000491	0.000009	1	rbf	{'C': 1, 'kernel': 'rbf'}	1
4	0.000718	0.000019	0.000455	0.000011	10	linear	{'C': 10, 'kernel': 'linear'}	1
5	0.000729	0.000010	0.000469	0.000012	10	rbf	{'C': 10, 'kernel': 'rbf'}	1

```
svm_opt.best_params_
```

```
{'C': 0.5, 'kernel': 'rbf'}
```

## 21.2. Other ways to compare models

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).

We can also compare models on their interpretability: a decision tree is easy to explain how it makes a decision.

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.

## 22. Model Comparison

```

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')

```

```

# 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)

# 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)

# optmize 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_)

```

So we have redone some familiar code. We have found the optimal parameters for best accuracy for two different classifiers, SVM and Decision tree on our data.

This is extra detail we did not do in class for time reasons

We can use EDA to understand how the score varied across all of the parameter settings we tried.

```

```{code-cell} ipython3
sv_df['mean_test_score'].describe()

```

```

dt_df['mean_test_score'].describe()

```

```

count    90.000000
mean     0.942778
std      0.004181
min      0.933333
25%     0.941667
50%     0.941667
75%     0.941667
max      0.950000
Name: mean_test_score, dtype: float64

```

From this we see that in both cases the standard deviation (std) is really low. This tells us that the parameter changes didn't impact the performance much. Combined with the overall high accuracy this tells us that the data is probably really easy to classify. If the performance had been uniformly bad, it might have instead told us that we did not try a wide enough range of parameters.

To confirm how many parameter settings we have used we can check a couple different ways. First, above in the count of the describe.

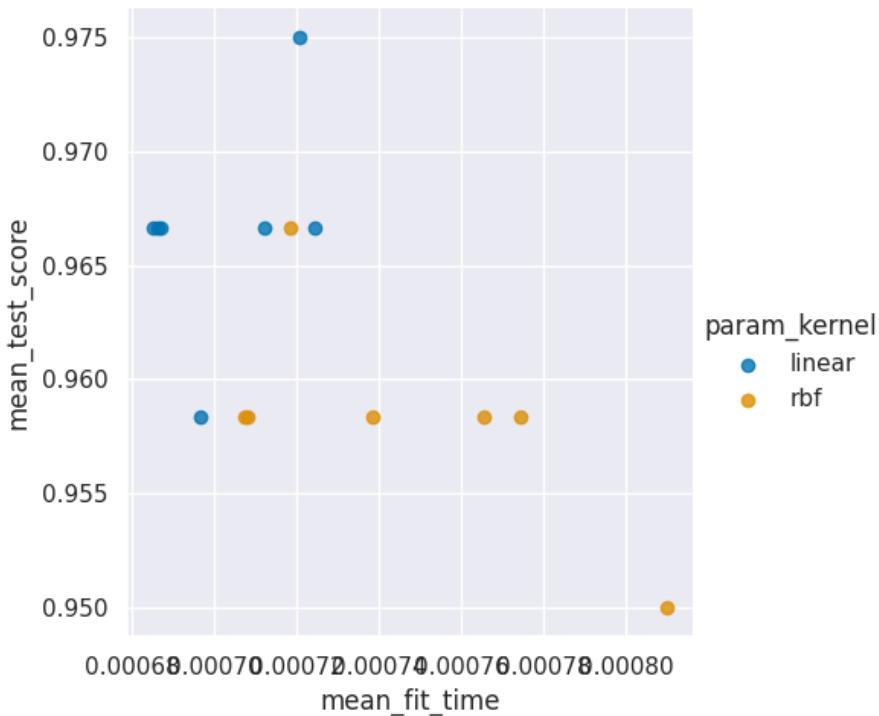
We can also calculate directly from the parameter grids before we even do the fit.

```

description_vars = ['param_C', 'param_kernel', 'params',]
vars_to_plot = ['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time']
svm_time = sv_df.melt(id_vars= description_vars,
                      value_vars=vars_to_plot)
sns.lmplot(data=sv_df, x='mean_fit_time',y='mean_test_score',
            hue='param_kernel',fit_reg=False)

```

<seaborn.axisgrid.FacetGrid at 0x7fc51519d640>



## 22.1. How does the timing vary?

Let's dig in and see which one is better.

[Skip to main content](#)

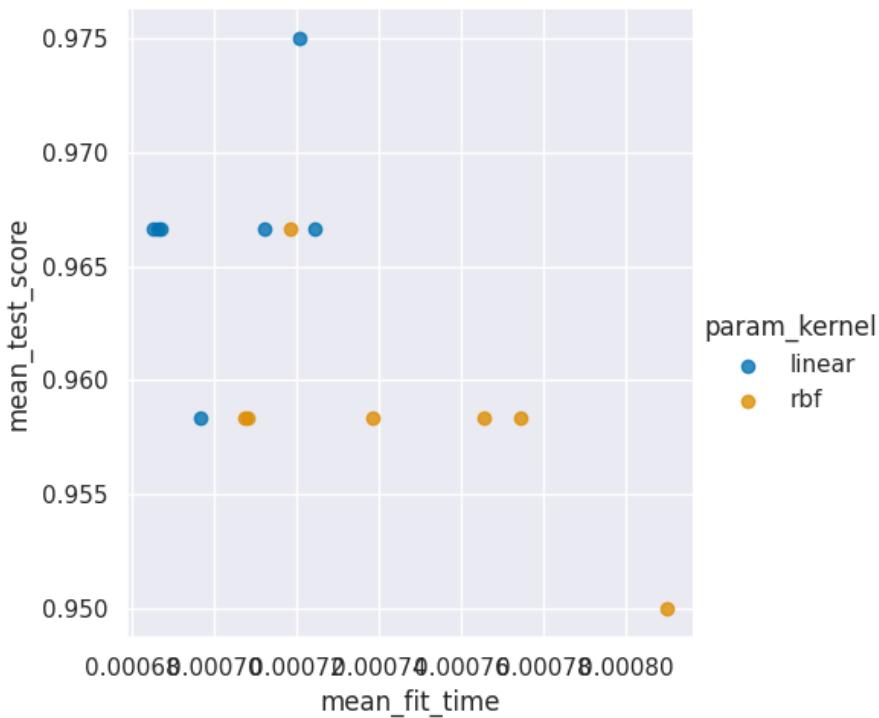
```
sv_df.head(1)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_kernel	params	split0_test_score
0	0.000725	0.000047	0.000452	0.000016	0.5	linear	{'C': 0.5, 'kernel': 'linear'}	0.916

```
%matplotlib inline
```

```
svm_time = sv_df.melt(id_vars=['param_C', 'param_kernel', 'params'],
                      value_vars=['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time'])
sns.lmplot(data=sv_df, x='mean_fit_time', y='mean_test_score',
            hue='param_kernel', fit_reg=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fc51519ddf0>
```



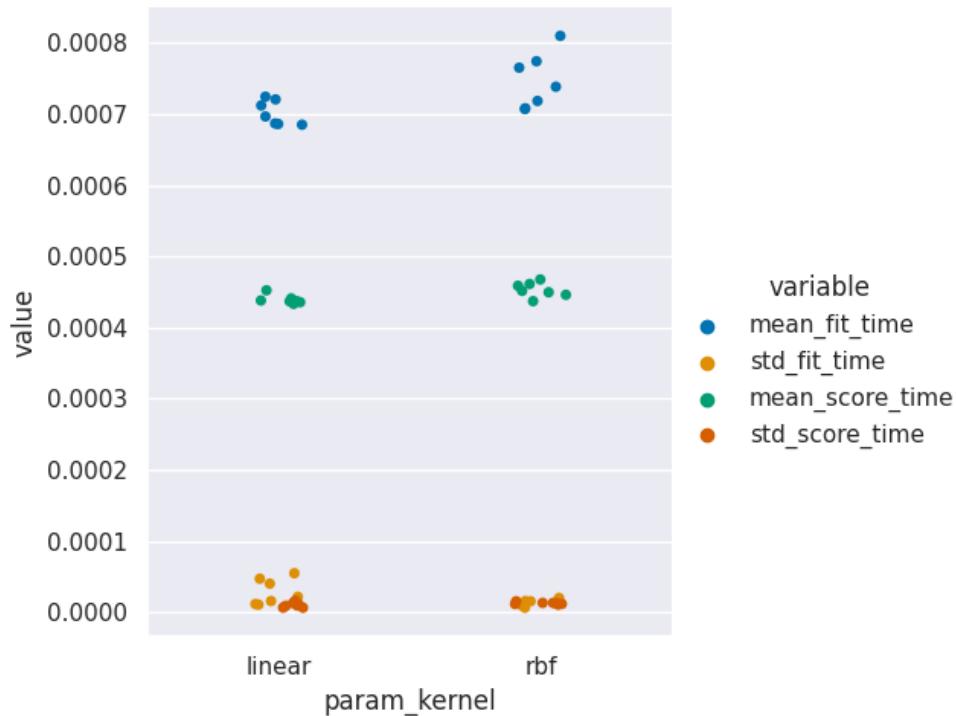
```
svm_time.head()
```

	param_C	param_kernel	params	variable	value
0	0.5	linear	{'C': 0.5, 'kernel': 'linear'}	mean_fit_time	0.000725
1	0.5	rbf	{'C': 0.5, 'kernel': 'rbf'}	mean_fit_time	0.000810
2	0.75	linear	{'C': 0.75, 'kernel': 'linear'}	mean_fit_time	0.000687
3	0.75	rbf	{'C': 0.75, 'kernel': 'rbf'}	mean_fit_time	0.000774
4	1	linear	{'C': 1, 'kernel': 'linear'}	mean_fit_time	0.000685

[Skip to main content](#)

```
sns.catplot(data= svm_time, x='param_kernel',y='value',hue='variable')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fc512d16d60>
```



## 22.2. How does training impact our model?

Now we can create the learning curve.

```
train_sizes, train_scores, test_scores, fit_times, score_times = model_selection.learning_curve(svm_opt.best_
train_sizes= [.4,.5,.6,.8],return_times=True)
```

It returns the list of the counts for each training size (we input percentages and it returns counts)

```
[train_sizes, train_scores, test_scores, fit_times, score_times]
```

```
[array([38, 48, 57, 76]),
 array([[1.          , 0.97368421, 0.97368421, 0.97368421, 0.97368421],
       [1.          , 0.97916667, 0.97916667, 0.97916667, 0.97916667],
       [1.          , 0.96491228, 0.98245614, 0.98245614, 0.98245614],
       [0.98684211, 0.98684211, 0.98684211, 0.98684211, 0.98684211]]),
 array([[0.91666667, 1.          , 0.875      , 1.          , 0.83333333],
       [0.875      , 1.          , 0.95833333, 1.          , 0.91666667],
       [0.95833333, 1.          , 0.95833333, 1.          , 0.91666667],
       [0.91666667, 1.          , 0.95833333, 1.          , 0.875      ]]),
 array([[0.00081515, 0.00063896, 0.00068641, 0.00065112, 0.00070548],
       [0.00070596, 0.00072908, 0.00062966, 0.00065351, 0.00063157],
       [0.00063896, 0.00064754, 0.00066471, 0.00064516, 0.00063753],
       [0.00070333, 0.0007205 , 0.00072002, 0.00071907, 0.00073934]]),
 array([[0.00053382, 0.00042462, 0.00044012, 0.00044084, 0.00044131],
       [0.00043774, 0.00044489, 0.00047207, 0.00045323, 0.00044942],
       [0.00045466, 0.00042129, 0.00042415, 0.00042319, 0.00044537],
       [0.00045156, 0.00042915, 0.00043058, 0.00043201, 0.00045252]])]
```

We can use our skills in transforming data to make it easier to examine just a subset of the scores.

```
train_scores.mean(axis=1)
```

```
array([0.97894737, 0.98333333, 0.98245614, 0.98684211])
```

```
[train_sizes, train_scores.mean(axis=1), test_scores.mean(axis=1),
 fit_times.mean(axis=1), score_times.mean(axis=1)]
```

```
[array([38, 48, 57, 76]),
 array([0.97894737, 0.98333333, 0.98245614, 0.98684211]),
 array([0.925      , 0.95      , 0.96666667, 0.95      ]),
 array([0.00069942, 0.00066996, 0.00064678, 0.00072045]),
 array([0.00045614, 0.00045147, 0.00043373, 0.00043917]])
```

```
np.asarray([train_sizes, train_scores.mean(axis=1), test_scores.mean(axis=1),
           fit_times.mean(axis=1), score_times.mean(axis=1)]).T
```

```
array([[3.8000000e+01, 9.78947368e-01, 9.25000000e-01, 6.99424744e-04,
        4.56142426e-04],
       [4.8000000e+01, 9.8333333e-01, 9.50000000e-01, 6.69956207e-04,
        4.51469421e-04],
       [5.7000000e+01, 9.82456140e-01, 9.66666667e-01, 6.46781921e-04,
        4.33731079e-04],
       [7.6000000e+01, 9.86842105e-01, 9.50000000e-01, 7.20453262e-04,
        4.39167023e-04]])
```

```
curve_df = pd.DataFrame(data = np.asarray([train_sizes, train_scores.mean(axis=1), test_scores.mean(axis=1),
   fit_times.mean(axis=1), score_times.mean(axis=1)]).T,
                           columns = ['train_sizes', 'train_scores', 'test_scores', 'fit_times', 'score_times'])
```

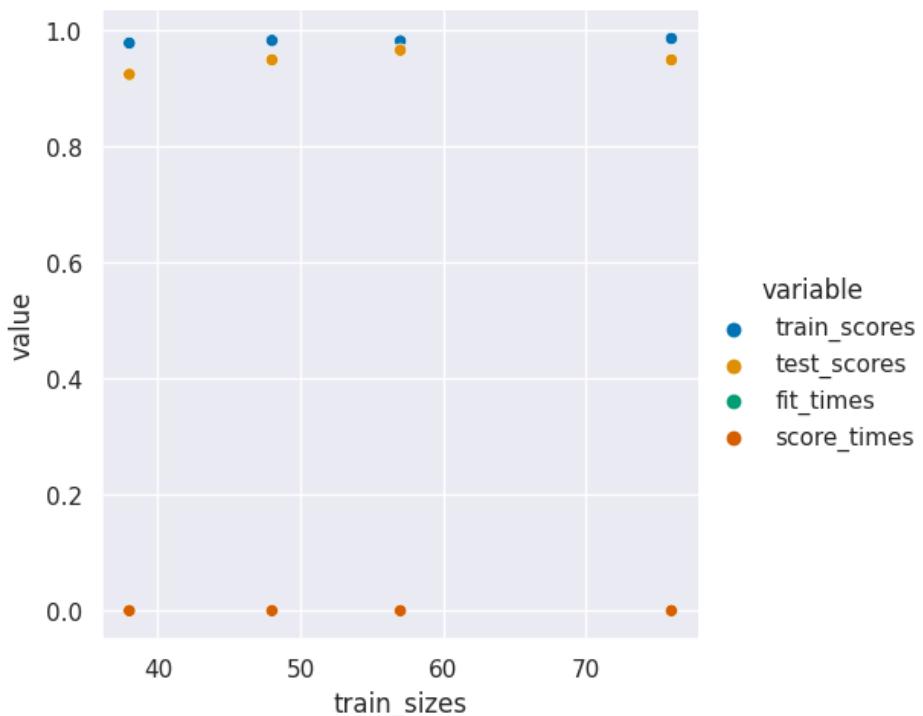
```
curve_df.head()
```

Hint  
This is  
there  
assign  
result

	train_sizes	train_scores	test_scores	fit_times	score_times
0	38.0	0.978947	0.925000	0.000699	0.000456
1	48.0	0.983333	0.950000	0.000670	0.000451
2	57.0	0.982456	0.966667	0.000647	0.000434
3	76.0	0.986842	0.950000	0.000720	0.000439

```
curve_df_tall = curve_df.melt(id_vars='train_sizes',
sns.relplot(data =curve_df_tall,x='train_sizes',y ='value',hue ='variable' )
```

<seaborn.axisgrid.FacetGrid at 0x7fc512d26a60>



We can see here that the training score and test score are basically the same. This means we're doing about as well as we can at learning the a generalizable model and we probably hav enough data for this task.

## 22.3. When do differences matter?

We can check calculate a confidence interval to determine more precisely when the performance of two models is meaningfully different.

This function calculates the 95% confidence interval. The range within which we are 95% confident the quantity we have estimated is truly within in. When we have more samples in the test set used to calculate the score, we are more confident in the estimate, so the interval is narrower.

```
def classification_confint(acc, n):
    """
    Compute the 95% confidence interval for a classification problem.
    acc -- classification accuracy
    n   -- number of observations used to compute the accuracy
    Returns a tuple (lb,ub)
    """
    interval = 1.96*np.sqrt(acc*(1-acc)/n)
    lb = max(0, acc - interval)
    ub = min(1.0, acc + interval)
    return (lb,ub)
```

```
svm_opt.best_score_, dt_opt.best_score_
```

```
(0.975, 0.95)
```

We can calculate the number of observations used to compute the accuracy using the size of the training data and the fact that we set it to 10-fold cross validation. That means that 10% (100/10) of the data was used for each fold and each validation set.

- 150 samples
- 80% training size (20% test size)
- 10 fold cross validation

```
.1*.8*150
```

```
12.000000000000002
```

And thn we can use this to find the range for each model

```
classification_confint(svm_opt.best_score_, 12)
```

```
(0.8866639937511323, 1.0)
```

```
classification_confint(dt_opt.best_score_, 12)
```

```
(0.8266860375572443, 1.0)
```

When the ranges overlap, th models are not signicanlty different.

```
N_diff = 500
classification_confint(svm_opt.best_score_, N_diff), classification_confint(dt_opt.best_score_, N_diff)
```

```
((0.9613150447571064, 0.9886849552428936),
(0.9308962830841744, 0.9691037169158255))
```

```
svm_opt.best_estimator_.score(iris_X_test,iris_y_test), dt_opt.best_estimator_.score(iris_X_test,iris_y_test)
```

```
(0.9666666666666667, 0.9333333333333333)
```

```
classification_confint(svm_opt.best_estimator_.score(iris_X_test,iris_y_test),12)
```

```
(0.865101872533033, 1.0)
```

```
classification_confint(dt_opt.best_estimator_.score(iris_X_test,iris_y_test),12)
```

```
(0.7921972025680066, 1.0)
```

## 22.4. Digits Dataset

Today, we'll load a new dataset and use the default sklearn data structure for datasets. We get back the default data stucture when we use a `load_` function without any parameters at all.

```
digits = datasets.load_digits()
```

This shows us that the type is defined by sklearn and they called it `bunch`:

```
type(digits)
```

```
sklearn.utils._bunch.Bunch
```

We can print it out to begin exploring it.

```
digits
```

```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
   [ 0.,  0.,  0., ..., 10.,  0.,  0.],
   [ 0.,  0.,  0., ..., 16.,  9.,  0.],
   ...,
   [ 0.,  0.,  1., ...,  6.,  0.,  0.],
   [ 0.,  0.,  2., ..., 12.,  0.,  0.],
   [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
'target': array([0, 1, 2, ..., 8, 9, 8]),
'frame': None,
'feature_names': ['pixel_0_0',
 'pixel_0_1',
 'pixel_0_2',
 'pixel_0_3',
 'pixel_0_4',
 'pixel_0_5',
 'pixel_0_6',
 'pixel_0_7',
 'pixel_1_0',
 'pixel_1_1',
 'pixel_1_2',
 'pixel_1_3',
 'pixel_1_4',
 'pixel_1_5',
 'pixel_1_6',
 'pixel_1_7',
 'pixel_2_0',
 'pixel_2_1',
 'pixel_2_2',
 'pixel_2_3',
 'pixel_2_4',
 'pixel_2_5',
 'pixel_2_6',
 'pixel_2_7',
 'pixel_3_0',
 'pixel_3_1',
 'pixel_3_2',
 'pixel_3_3',
 'pixel_3_4',
 'pixel_3_5',
 'pixel_3_6',
 'pixel_3_7',
 'pixel_4_0',
 'pixel_4_1',
 'pixel_4_2',
 'pixel_4_3',
 'pixel_4_4',
 'pixel_4_5',
 'pixel_4_6',
 'pixel_4_7',
 'pixel_5_0',
 'pixel_5_1',
 'pixel_5_2',
 'pixel_5_3',
 'pixel_5_4',
 'pixel_5_5',
 'pixel_5_6',
 'pixel_5_7',
 'pixel_6_0',
 'pixel_6_1',
 'pixel_6_2',
 'pixel_6_3',
 'pixel_6_4',
 'pixel_6_5',
 'pixel_6_6',
 'pixel_6_7',
 'pixel_7_0',
 'pixel_7_1',
 'pixel_7_2',
 'pixel_7_3',
 'pixel_7_4',
 'pixel_7_5']}
```

```

'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
   [ 0.,  0., 13., ..., 15.,  5.,  0.],
   [ 0.,  3., 15., ..., 11.,  8.,  0.],
   ...,
   [ 0.,  4., 11., ..., 12.,  7.,  0.],
   [ 0.,  2., 14., ..., 12.,  0.,  0.],
   [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

[[ 0.,  0.,  0., ...,  5.,  0.,  0.],
   [ 0.,  0.,  0., ...,  9.,  0.,  0.],
   [ 0.,  0.,  3., ...,  6.,  0.,  0.],
   ...,
   [ 0.,  0.,  1., ...,  6.,  0.,  0.],
   [ 0.,  0.,  1., ...,  6.,  0.,  0.],
   [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

[[ 0.,  0.,  0., ..., 12.,  0.,  0.],
   [ 0.,  0.,  3., ..., 14.,  0.,  0.],
   [ 0.,  0.,  8., ..., 16.,  0.,  0.],
   ...,
   [ 0.,  9., 16., ...,  0.,  0.,  0.],
   [ 0.,  3., 13., ..., 11.,  5.,  0.],
   [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

...,

[[ 0.,  0.,  1., ...,  1.,  0.,  0.],
   [ 0.,  0., 13., ...,  2.,  1.,  0.],
   [ 0.,  0., 16., ..., 16.,  5.,  0.],
   ...,
   [ 0.,  0., 16., ..., 15.,  0.,  0.],
   [ 0.,  0., 15., ..., 16.,  0.,  0.],
   [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

[[ 0.,  0.,  2., ...,  0.,  0.,  0.],
   [ 0.,  0., 14., ..., 15.,  1.,  0.],
   [ 0.,  4., 16., ..., 16.,  7.,  0.],
   ...,
   [ 0.,  0.,  0., ..., 16.,  2.,  0.],
   [ 0.,  0.,  4., ..., 16.,  2.,  0.],
   [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

[[ 0.,  0., 10., ...,  1.,  0.,  0.],
   [ 0.,  2., 16., ...,  1.,  0.,  0.],
   [ 0.,  0., 15., ..., 15.,  0.,  0.],
   ...,
   [ 0.,  4., 16., ..., 16.,  6.,  0.],
   [ 0.,  8., 16., ..., 16.,  8.,  0.],
   [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),

'DESCR': "... _digits_dataset:\n\nOptical recognition of handwritten digits dataset\n-----"

```

We note that it has key value pairs, and that the last one is called `DESCR` and is text that describes the data. If we send that to the `print` function it will be formatted more readably.

```
print(digits['DESCR'])
```

```
.. _digits_dataset:  
  
Optical recognition of handwritten digits dataset  
-----  
  
**Data Set Characteristics:**  
  
:Number of Instances: 1797  
:Number of Attributes: 64  
:Attribute Information: 8x8 image of integer pixels in the range 0..16.  
:Missing Attribute Values: None  
:Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)  
:Date: July; 1998
```

This is a copy of the test set of the UCI ML hand-written digits datasets  
<https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits>

The data set contains images of hand-written digits: 10 classes where each class refers to a digit.

Preprocessing programs made available by NIST were used to extract normalized bitmaps of handwritten digits from a preprinted form. From a total of 43 people, 30 contributed to the training set and different 13 to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of 4x4 and the number of on pixels are counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0..16. This reduces dimensionality and gives invariance to small distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469, 1994.

.. topic:: References

- C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their Applications to Handwritten Digit Recognition, MSc Thesis, Institute of Graduate Studies in Science and Engineering, Bogazici University.
- E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
- Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin. Linear dimensionality reduction using relevance weighted LDA. School of Electrical and Electronic Engineering Nanyang Technological University. 2005.
- Claudio Gentile. A New Approximate Maximal Margin Classification Algorithm. NIPS. 2000.

This tells us that we are going to be predicting what digit (0,1,2,3,4,5,6,7,8, or 9) is in the image.

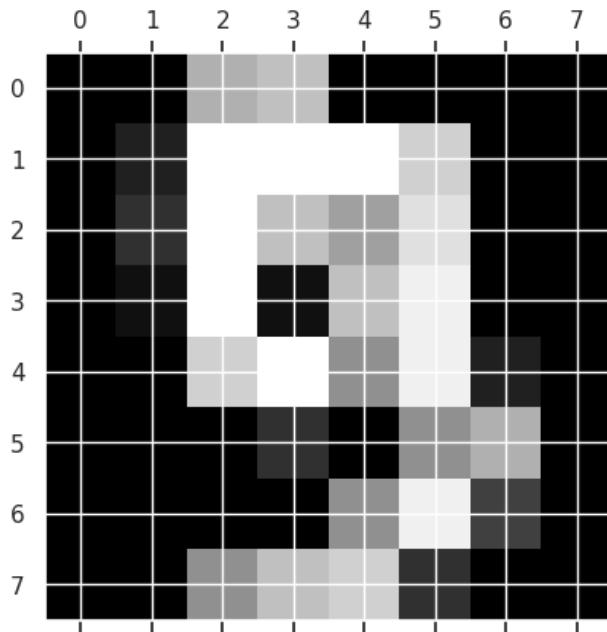
To get an idea of what the images look like, we can use `matshow` which is short for matrix show. It takes a 2D matrix and plots it as a grayscale image. To get the actual color bar, we use the matplotlib `plt.gray()`.

```
plt.gray()  
plt.matshow(digits.images[9])
```

Try  
plt.  
What  
other  
how p

```
<matplotlib.image.AxesImage at 0x7fc512b708e0>
```

```
<Figure size 640x480 with 0 Axes>
```



For easier ML, we will reload it differently:

```
digits_X, digits_y = datasets.load_digits(return_X_y=True)
```

This has one row for each sample and has reshaped the 8x8 image into a 64 length vector. So we have one 'feature' for each pixel in the images.

```
digits_X.shape
```

```
(1797, 64)
```

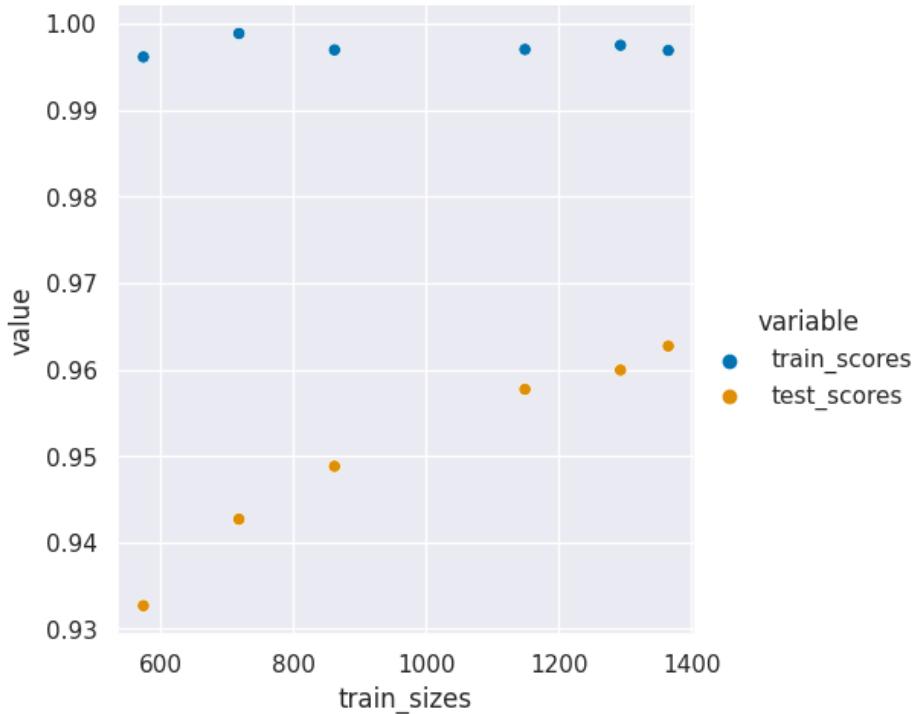
Now we can train a model and generate a learning curve.

```
svm_clf_digits = svm.SVC()
train_sizes, train_scores, test_scores, fit_times, score_times = model_selection.learning_curve(
    svm_clf_digits, digits_X, digits_y,
    train_sizes=[.4,.5,.6,.8,.9, .95], return_times=True)
```

```
digits_curve_df = pd.DataFrame(data = np.asarray([train_sizes, train_scores.mean(axis=1), test_scores.mean(axis=1),
   fit_times.mean(axis=1), score_times.mean(axis=1)]).T,
                                 columns = ['train_sizes', 'train_scores', 'test_scores', 'fit_times', 'score_times'])
```

```
digits_curve_df_tall = digits_curve_df.melt(id_vars = 'train_sizes', value_vars=[ 'train_scores', 'test_scores',
   'fit_times', 'score_times'], var_name='variable')
sns.relplot(data = digits_curve_df_tall, x = 'train_sizes', y='value', hue='variable',)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fc512ad3550>
```



This larger gap shows that a different model or maybe more data could help us learn better. The good news is that we are probably not overfitting, overfitting would occur when the training accuracy still improves but the test accuracy goes down.

## 23. Intro to NLP- representing text data

```
from sklearn.feature_extraction import text
from sklearn.metrics.pairwise import euclidean_distances
from sklearn import datasets
import pandas as pd
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

ng_X, ng_y = datasets.fetch_20newsgroups(categories =['comp.graphics', 'sci.crypt'],
   return_X_y = True)
```

### 23.1. Text as Data

Let's make a small dataset of sentences

```
sentences = [
'The semester is almost over. - Professor Brown',
'The weather is getting much warmer outside. - Ben',
'Dr. Pepper is the best soda ever - Ebrahima',
'Only a few more weeks before graduation. -Gabe',
'Hello friends. -Zach',
'Mr. Owl ate my metal worm. -Sath',
'I wonder what is for dinner? - Vinai',
'Hello everyone- jay ',]
```

[Skip to main content](#)

How can we analyze these? All of the machine learning models we have seen only use numerical features organized into a table with one row per sample and one column per feature.

That's actually generally true. All ML models require numerical features, at some point. The process of taking data that is not numerical and tabular, which is called unstructured, into structured (tabular) format we require is called feature extraction. There are many, many ways to do that. We'll see a few over the course of the rest of the semester. Some more advanced models hide the feature extraction, by putting it in the same function, but it's always there.

```
sentences[0]
```

```
'The semester is almost over. - Professor Brown'
```

```
s1 = sentences[0]
```

## 23.2. Terms

- document: unit of text we're analyzing (one sample)
- token: sequence of characters in some particular document that are grouped together as a useful semantic unit for processing (basically a word)
- stop words: no meaning, we don't need them (like a, the, an.). Note that this is context dependent
- dictionary: all of the possible words that a given system knows how to process

## 23.3. Bag of Words Representation

We're going to learn a representation called the bag of words. It ignores the order of the words within a document. To do this, we'll first extract all of the tokens (tokenize) the documents and then count how many times each word appears. This will be our numerical representation of the data.

Then we initialize our transformer, and use the fit transform method to fit the vectorizer model and apply it to this sentence.

```
counts = text.CountVectorizer()
```

```
counts.fit_transform([s1])
```

```
<1x7 sparse matrix of type '<class 'numpy.int64'>'  
with 7 stored elements in Compressed Sparse Row format>
```

We see it returns a sparse matrix. A sparse matrix means that it has a lot of 0s or missing values in it and so we only represent the data. If we were to represent all of the 0s or NAs we would have to use the same amount of memory for a full matrix and a sparse one.

Consider a matrix that's 10x10, that means 100 total values to store if we store it dense. If we have only a small number of values that are not 0, say 5%, storing all of that seems like a lot. As a sparse matrix, we can instead store the each value and its location.

Fur  
Trans  
of skl  
Estim  
focus  
for no

So 5% is 5 values, but we store 3 values for it (value, column, row) for 15. 15 is still a lot less than 100. So as long as less than 1/3 of the values are used sparse format is an advantage.

```
mfull = np.asarray([[1,0,0,0,0],[0,0,1,0,0],[0,0,0,1,0]])
```

```
-----  
NameError                                 Traceback (most recent call last)  
Cell In[7], line 1  
----> 1 mfull = np.asarray([[1,0,0,0,0],[0,0,1,0,0],[0,0,0,1,0]])  
  
NameError: name 'np' is not defined
```

but as a sparse matrix, we could store fewer values.

```
[[0,0,1],[1,2,1],[2,3,1]]# the above
```

```
[[0, 0, 1], [1, 2, 1], [2, 3, 1]]
```

Text data will often be sparse.

So any matrix where the number of total values is low enough, we can store it more efficiently by tracking the locations and values instead of all of the zeros.

To actually see it though we have to cast out of that into a regular array.

```
counts.fit_transform([s1]).toarray()
```

```
array([[1, 1, 1, 1, 1, 1]])
```

We can also examine attributes of the object.

```
counts.vocabulary_
```

```
{'the': 6,  
'semester': 5,  
'is': 2,  
'almost': 0,  
'over': 3,  
'professor': 4,  
'brown': 1}
```

We see that what it does is creates an ordered (the values are the order) list of words as the parameters of this model (ending in `_` is an attribute of the object or parameter of the model).

it puts the words in the `vocabulary_` attribute (aka the dictionary) in alphabetical order.

Tip

Notice tools things own, things available documents

Also, your than strategies with speed

## 23.4. Reformatting data

[Skip to main content](#)

```
sentences_split = [sent_attr.split('-') for sent_attr in sentences]  
sentences_split
```

```
[['The semester is almost over. ', ' Professor Brown'],
 ['The weather is getting much warmer outside. ', ' Ben'],
 ['Dr. Pepper is the best soda ever ', ' Ebrahima'],
 ['Only a few more weeks before graduation. ', ' Gabe'],
 ['Hello friends. ', ' Zach'],
 ['Mr. Owl ate my metal worm. ', ' Sath'],
 ['I wonder what is for dinner? ', ' Vinai'],
 ['Hello everyone', ' jay ']]
```

```
text_dict = {attr.strip():sentence.strip() for sentence,attr in sentences_split}
text dict
```

```
{'Professor Brown': 'The semester is almost over.',  
 'Ben': 'The weather is getting much warmer outside.',  
 'Ebrahima': 'Dr. Pepper is the best soda ever',  
 'Gabe': 'Only a few more weeks before graduation.',  
 'Zach': 'Hello friends.',  
 'Sath': 'Mr. Owl ate my metal worm.',  
 'Vinai': 'I wonder what is for dinner?',  
 'jay': 'Hello everyone'}
```

and now transform

```
mat = counts.fit_transform(text_dict.values()).toarray()
mat
```

From this we can see that the representation is the count of how many times each word appears.

Now we can apply it to all of the sentences, or our whole `corpus`. We can get the dictionary out in order using the `get_feature_names_out` method. This method has a generic name, not specific to text, because it's a property of transformers in general.

```
counts.get_feature_names_out()
```

```
array(['almost', 'ate', 'before', 'best', 'dinner', 'dr', 'ever',
       'everyone', 'few', 'for', 'friends', 'getting', 'graduation',
       'hello', 'is', 'metal', 'more', 'mr', 'much', 'my', 'only',
       'outside', 'over', 'owl', 'pepper', 'semester', 'soda', 'the',
       'warmer', 'weather', 'weeks', 'what', 'wonder', 'worm'],
      dtype=object)
```

We can use a dataframe again to see this more easily. We can put labels on both the index and the column headings.

```
sentence_df = pd.DataFrame(data = mat, columns = counts.get_feature_names_out(),
                           index = text_dict.keys())
sentence_df
```

	<b>almost</b>	<b>ate</b>	<b>before</b>	<b>best</b>	<b>dinner</b>	<b>dr</b>	<b>ever</b>	<b>everyone</b>	<b>few</b>	<b>for</b>	<b>...</b>	<b>pepper</b>	<b>semester</b>	<b>soda</b>	<b>the</b>	<b>w</b>
<b>Professor Brown</b>	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	
<b>Ben</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
<b>Ebrahima</b>	0	0	0	1	0	1	1	0	0	0	0	1	0	1	1	
<b>Gabe</b>	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
<b>Zach</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Sath</b>	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Vinai</b>	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
<b>jay</b>	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

8 rows × 34 columns

## 23.5. Basic text analysis

We can find the most common word

One guess

```
sentence_df.max()
```

```
almost      1
ate         1
before      1
best        1
dinner      1
dr          1
ever        1
everyone    1
few         1
for         1
friends     1
getting     1
graduation  1
hello       1
is          1
metal       1
more        1
mr          1
much        1
my          1
only        1
outside     1
over        1
owl         1
pepper      1
semester    1
soda        1
the         1
warmer      1
weather     1
weeks       1
what        1
wonder      1
worm        1
dtype: int64
```

This is the maximum number of times each word appears in single “document”, but it’s also not sorted, it’s alphabetical.

This shows the word that appears the most times.

To get what we want we need to sum, which by default is along the columns, or per word.

```
sentence_df.sum(axis=0).sort_values(ascending=False)
```

```
is          4
the         3
hello       2
almost      1
pepper      1
only        1
outside     1
over        1
owl         1
semester   1
much        1
soda        1
warmer     1
weather    1
weeks      1
what        1
wonder     1
my          1
mr          1
ate         1
more        1
metal       1
graduation 1
getting     1
friends    1
for         1
few         1
everyone   1
ever        1
dr          1
dinner     1
best        1
before      1
worm       1
dtype: int64
```

Then we get the location of the max with idx max.

```
sentence_df.sum().idxmax()
```

```
'is'
```

What is the total number of unique words across all sentences?

```
_, n_unique = sentence_df.shape
n_unique
```

```
34
```

this is also the *size of the dictionary*

We can get the total number of words in all of the sentences by summing twice.

```
sentence_df.sum().sum()
```

```
40
```

Who's sentence had the most words in it?

```
sentence_df.sum(axis=1)
```

```
Professor Brown      5
Ben                  7
Ebrahima             7
Gabe                 6
Zach                 2
Sath                 6
Vinai                5
jay                  2
dtype: int64
```

summing across rows shows how many words per sentence.

## 23.6. Distances in text

We can now use a distance function to calculate how far apart the different sentences are.

```
dists = euclidean_distances(sentence_df)
dists
```

```
array([[0.          , 2.82842712, 2.82842712, 3.31662479, 2.64575131,
       3.31662479, 2.82842712, 2.64575131],
      [2.82842712, 0.          , 3.16227766, 3.60555128, 3.          ,
       3.60555128, 3.16227766, 3.          ],
      [2.82842712, 3.16227766, 0.          , 3.60555128, 3.          ,
       3.60555128, 3.16227766, 3.          ],
      [3.31662479, 3.60555128, 3.60555128, 0.          , 2.82842712,
       3.46410162, 3.31662479, 2.82842712],
      [2.64575131, 3.          , 3.          , 2.82842712, 0.          ,
       2.82842712, 2.64575131, 1.41421356],
      [3.31662479, 3.60555128, 3.60555128, 3.46410162, 2.82842712,
       0.          , 3.31662479, 2.82842712],
      [2.82842712, 3.16227766, 3.16227766, 3.31662479, 2.64575131,
       3.31662479, 0.          , 2.64575131],
      [2.64575131, 3.          , 3.          , 2.82842712, 1.41421356,
       2.82842712, 2.64575131, 0.          ]])
```

This distance is only in terms of actual reused words. It does not contain anything about the meaning of the words

We can make this easier to read by making it a Data Frame.

```
dist_df = pd.DataFrame(data=dists, index= text_dict.keys(),
                       columns=text_dict.keys())
dist_df
```

	<b>Professor Brown</b>	<b>Ben</b>	<b>Ebrahima</b>	<b>Gabe</b>	<b>Zach</b>	<b>Sath</b>	<b>Vinai</b>	<b>jay</b>
<b>Professor Brown</b>	0.000000	2.828427	2.828427	3.316625	2.645751	3.316625	2.828427	2.645751
<b>Ben</b>	2.828427	0.000000	3.162278	3.605551	3.000000	3.605551	3.162278	3.000000
<b>Ebrahima</b>	2.828427	3.162278	0.000000	3.605551	3.000000	3.605551	3.162278	3.000000
<b>Gabe</b>	3.316625	3.605551	3.605551	0.000000	2.828427	3.464102	3.316625	2.828427
<b>Zach</b>	2.645751	3.000000	3.000000	2.828427	0.000000	2.828427	2.645751	1.414214
<b>Sath</b>	3.316625	3.605551	3.605551	3.464102	2.828427	0.000000	3.316625	2.828427
<b>Vinai</b>	2.828427	3.162278	3.162278	3.316625	2.645751	3.316625	0.000000	2.645751
<b>jay</b>	2.645751	3.000000	3.000000	2.828427	1.414214	2.828427	2.645751	0.000000

Who wrote the most similar to me? which two were most similar to one another?

text\_dict

```
{'Professor Brown': 'The semester is almost over.',
 'Ben': 'The weather is getting much warmer outside.',
 'Ebrahima': 'Dr. Pepper is the best soda ever',
 'Gabe': 'Only a few more weeks before graduation.',
 'Zach': 'Hello friends.',
 'Sath': 'Mr. Owl ate my metal worm.',
 'Vinai': 'I wonder what is for dinner?',
 'jay': 'Hello everyone'}
```

## 1. Assignment 1: Portfolio Setup, Data Science, and Python

Due:

Eligible skills: (links to checklists)

- ★<sup>[1]</sup> python level 1 and level 2
- ★process<sup>[2]</sup> level 1

### 1.1. Related notes

- Welcome and Introduction
- Syllabus and Python Review

### 1.2. To Do

#### ! Important

If you have trouble, check the GitHub FAQ on the left before e-mailing

1. Install required software from the Tools & Resource page
2. Create your portfolio, by
3. Learn about your portfolio from the README file on your repository.
4. edit `_config.yml` to set your name as author and change the logo if you wish
5. Fill in `about/index.md` with information about yourself(not evaluated, but useful) and your own definition of data science (graded for **level 1 process**)
6. Add a Jupyter notebook called `grading.ipynb` to the `about` folder and write a function that computes a grade for this course, with the docstring below.
7. Add the line `- file: about/grading` in your `_toc.yml` file.

#### ! Important

Do not merge your "Feedback" Pull Request

### 1.2.1. Docstring

```
'''  
    Computes a grade for CSC/DSP310 from numbers of achievements at each level  
  
    Parameters:  
    -----  
    num_level1 : int  
        number of level 1 achievements earned  
    num_level2 : int  
        number of level 2 achievements earned  
    num_level3 : int  
        number of level 3 achievements earned  
  
    Returns:  
    -----  
    letter_grade : string  
        letter grade with possible modifier (+/-)  
'''
```

### 1.2.2. Sample tests

Here are some sample tests you could run to confirm that your function works correctly:

```
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+'  
assert compute_grade(15,15,1) == 'F'
```

### 1.2.3. Notebook Checklist

[Skip to main content](#)

- 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.

#### 1.2.4. Grading Notes:

- a basic function that uses conditionals in python will earn **level 1 python**
- to earn **level 2 python** use pythonic code to write a loop that tests your function's correctness, by iterating over a list or dictionary. Remember you will have many chances to earn level 2 achievement in python, so you do not need to do this step for this assignment if you are not sure how.

---

[1] skills will be marked like this on the first time they are eligible. There will also be a ✎ on skills for the last assignment they are eligible

[2] process is a special skill. You'll earn level 1 in this assignment or a soon one and two in either portfolio 1 or assignments 6-10, then level 3 in portfolio 2,3, or 4.

## 2. Assignment 2: Practicing Python and Accessing Data

due :

### 2.1. Objective & Evaluation

Eligible skills: (links to checklists)

- **first chance** access 1 and 2
- python 1 and 2
- summarize 1 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`.

In this assignment, you'll practice/ review python skills by manipulating datasets and extracting basic information about them.

### 2.2. Related notes

- Grading review, Pandas, and Iterables
- Pandas and Indexing

### 2.3. Setting

Next week, we are going to learn about summarizing data. In this assignment, you are going to build a small dataset about datasets. In class next week, we will combine all of your datasets about datasets together in order to be able to answer questions like:

- how much total data did you all load

- how many total rows of data did each student load?

## 2.4. Tasks

First, . It contains a notebook with some template structure (and will set you up for grading).

### 2.4.1. Find Datasets

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.

In your notebook, create a markdown cell for each dataset that includes:

- heading of 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
- 1-2 questions you would like to answer with that dataset.

### 2.4.2. Store them for loading

Create a list of dictionaries in `datasets.py`, so that there is one dictionary for each dataset. Each dictionary should have the following keys:

<code>url</code>	with the url
<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> .

### 2.4.3. 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. 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.
2. record attributes about the dataset as in the table below in a list of lists or dictionary
3. Use that to create a DataFrame with columns that match the rows of the following table.

Hin

See t  
exam

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

Meta Data Description of the DataFrame to build

#### 2.4.4. Explore Your Datasets

In 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 last 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 first three rows
- display the 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
- display the first 3 multiples of 3 rows (eg 3,6,9) of the data for two columns of your choice

#### 2.4.5. Exploring data files

Continue in your `exploration.ipynb`. 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.

### 2.5. Submission

[Skip to main content](#)

## 2.6. Thinking ahead

### ! Important

This section is not required, but is intended to help you get started thinking about ideas for your portfolio. 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 this after submitting the assignment, since you do not have to get a grade for it. 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: \_\_\_\_\_

### ! Important

You have the option to work with a partner. You must plan this in advance so that you have access to collaborate. If you did not find a partner in class and you would like one, try to find one [on the class discussion](#). @brownsarahm if you do not get a reply.

### 3.1. Objective & Evaluation

This week your goal is to do a small exploratory data analysis for two datasets of your choice.

Eligible skills: (links to checklists)

- process 1
- access 1 and 2
- **first chance** summarize 1 and 2
- **first chance** visualize 1 and 2

### 3.2. Related notes

- [Exploratory Data Analysis \(EDA\)](#)
- [Visualization](#)

### 3.3. 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 last week, if they meet the criteria below~~.

[Skip to main content](#)

### 3.3.1. 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

### 3.3.2. Dataset 2 (d2)

must include at least:

- two categorical variables **and**
- one continuous valued variable

## 3.4. EDA

Use a separate notebook for each dataset, name them `dataset_01.ipynb` and `dataset_02.ipynb`.

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. Explore the dataset in a notebook enough to describe its structure use the heading `## Description`
  - shape
  - columns
  - variable types
  - overall summary statistics
3. Write a short description of what the data contains and what it could be used for
4. Include overall summary for the data and interpret what that means. This should include code that generates the statistical summary and sentences in English in a markdown cell with your conclusions and explanation of the statistical summary. Are there limitations in how to safely interpret the data that the summary helps you see? are the variables what you expect?
5. Ask and answer 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.
6. 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.

### 3.4.1. Question checklist

be sure that every question (all six, 3 per dataset) has:

- a heading

- interpretation that answers the question

### 3.4.2. 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.

### 3.4.3. 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

#### 💡 Tip

Be sure to start early and use help hours to make sure you have a plan for all of these.

## 3.5. Peer Review

#### ℹ Note

This is optional, but if you do a review, you only need to do one analysis each.

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.

In your review:

- Use inline comments to denote places that are confusing or if you see solutions to problems your classmate could not solve
- Use the template below for your summary review

### 3.5.1. Review Questions

1. How was the analysis overall to read? easy? hard? cohesive? jumpy?
2. Did the data summaries tell you enough about the data to understand the analysis and anticipate what kinds of questions could be answered? If not, what questions do you still have about the data?

4. Are the statistics and plots appropriate for the questions?
5. Are the interpretations complete, clear, and consistent with the statistics and plots?
6. What could be done to make the explanations more clear and complete?
7. What additional analysis might make the analysis more compelling and clear?

```
<!-- delete sections that are not needed -->
## Overall

This analysis was ...

## Data Summaries

- [ ] complete

To understand this analysis I still need to know ...

## Checklist

- [ ] questions fit the data
- [ ] questions are in natural language
- [ ] chosen statistics and plots match questions
- [ ] all statistics and plots have an interpretation in English

## Areas of improvement
```

### 3.5.2. Response

Respond to your review either inline comments, replies, or by updating your analysis accordingly.

#### Think Ahead

1. How could you make more customized summary tables?
2. Could you use any of the variables in this dataset to add more variables that would make interesting ways to apply split-apply-combine? (eg thresholding a continuous value to make a categorical value)

## 4. Assignment 4:

Due:

Eligible skills: (links to checklists)

- **first chance** prepare [1](#) and [2](#)
- access [1](#) and [2](#)
- python [1](#) and [2](#)
- summarize [1](#) and [2](#)
- visualize [1](#) and [2](#)

### 4.1. Related notes

- [Tidy Data and Reshaping Datasets](#)

[Skip to main content](#)



This section is intended for thinking about the assignment. If you have feedback, get to the momento part.

- Reparing values

## 4.2. 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.

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
- missing values
- a column that was continuous, but more interesting as a categorical
- too many header rows

### Think Ahead

*this box is not required, but ideas for portfolio* cleaning a dataset to make it able to answer questions that were not possible could satisfy the level 3 prepare requirements.

## 4.3. Clean example datasets

There are notebooks in the template that have instructions for how to work with each dataset, including how to load it and what high level cleaning should be done. Your job is to execute.

To earn prepare level 2, clean any dataset and do just enough exploratory data analysis to show that the data is usable (eg 1 stat and/or plot).

To also earn python level 2: clean the CS degrees dataset (use a function or lambda AND loop or list/dictionary comprehension)

To also earn access level 2: clean the airline data (to get data in a second file type).

To also earn summarize and/or visualize level 2: add extra exploratory data analyses of your cleaned dataset meeting the criteria from the checklist (eg follow a3 checklists).

This means that if you want to earn prepare, python, and access, you will need to clean two datasets.

### Hint

renaming things is often done well with a dictionary comprehension or lambda.

## 4.4. Study Cleaned Datasets

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 or additional notebook in your repository. (some example datasets are on the datasets page and one is in the notes are added to the course website)

[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 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.

#### Important

Remember to run the “Submit” Workflow from the actions tab of your repository. see how on the How tos page

## 5. Assignment 5: Constructing Datasets and Using Databases

[accept the assignment](#)

**Due: 2023-03-01**

Eligible skills: (links to checklists)

- **first chance** construct [1](#) and [2](#)
-  [\[1\]](#) access [1](#) and [2](#)
-  [\[1\]](#) python [1](#) and [2](#)
-  [\[1\]](#) prepare [1](#) and [2](#)
- summarize [1](#) and [2](#)
- visualize [1](#) and [2](#)

### 5.1. Related notes

- [Merging and Databases](#)
- [Web Scraping](#)

### 5.2. Constructing Datasets

Your goal is to programmatically construct two ready to analyze datasets from multiple sources.

- Each dataset must combine at least 2 source tables(4 total source tables).
- At least one source table must come from a database or from web scraping.
- You should use at least three different joins(types of merges, or concat).

[Skip to main content](#)

The notebook you submit should include:

- a motivating question for why you're combining the datasets in an introduction section
- code and description of how you built and prepared each 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.
- For one pair of tables, show how a different merge could answer a different question.

For construct, this can be very minimal EDA.

The notebook you submit should include:

- a motivating question for why you're combining the datasets in an introduction section
- code and description of how you built and prepared each 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.

For construct, this can be very minimal EDA.

## 5.3. Additional achievements

To earn additional achievements, you must do more cleaning and/or exploratory data analysis.

### 5.3.1. Prepare level 2

To earn level 2 for prepare, you must manipulate either a component table or the final dataset. See your Achievement checklist for which aspects of prepare you still need, but 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

### 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 focus on the aspects of these achievements you have not successfully demonstrated.

### 5.3.3. Python Level 2

Use pythonic naming conventions throughout, AND:

- Use pythonic loops and a list or dictionary OR
- use a list or dictionary comprehension

### 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.

[1](1,2,3) skills will be marked like this on the last assignment they are eligible

## 6. Assignment 6: Auditing Algorithms

accept the assignment Due: 2023-03-09

Eligible skills: (links to checklists)

- **first chance** evaluate 1 and 2
- construct 1 and 2
- summarize 1 and 2
- visualize 1 and 2
- python 1 and 2

! Imp

the d  
be up  
soon,  
corre

### 6.1. Related notes

- Evaluating ML Algorithms
- Auditing with AIF360

### 6.2. About the data

We have provided a 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.

`adult_models_only.csv` has the model's predictions and `adult_reconstruction_bin.csv` has the data. Both have a unique identifier column included.

### Think Ahead

Why might the dataset have more samples in it than the model predictions one?

## 6.3. Complete an audit

Thoroughly audit any one model. In your audit, use three different performance metrics. Compare and contrast performance in those metrics across both 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.

If the model you chose was used for some real world decision what might the risks be?

## 6.4. Extend your Audit

### 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

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

[accept the assignment](#)

**Due: 2023-03-23**

[Skip to main content](#)

- **first chance** classification 1 and 2
- evaluate 1 and 2
- summarize 1 and 2
- visualize 1 and 2
- python 1 and 2

## 7.1. Related notes

- Intro to ML & Naive Bayes
- Predictions with naive Bayes and Decision Trees

## 7.2. Dataset and EDA

Choose a datasets 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 one hot encoding.

### Hint

Use the UCI ML repository, the “new beta version” 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?
4. Hypothesize which will classifier 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?

## 7.3. Basic Classification

1. Fit your chosen classifier with the default parameters on 50% of the data
2. Test it on 50% held out data and generate a classification report
3. Inspect the model to answer the questions appropriate to your model.
  - Does this model make sense?
  - (DT) Are there any leaves that are very small?
  - (DT) Is this an interpretable number of levels?
  - (GNB) do the parameters fit the data well?
  - (GNB) do the paramters generate similar synthetic data
4. Interpret the model and its performance in terms of the application. 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?

[Skip to main content](#)

## 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. Train a model (if decision tree set the max depth 2 less than the depth it found above) on 10%, 30%, ..., 90% of the data. Compute the [training accuracy](#) and test accuracy for each size training data in a DataFrame with columns ['train\_pct','n\_train\_samples','n\_test\_samples','train\_acc','test\_acc']
2. Plot the accuracies vs training percentage in a line graph.
3. Interpret these results. How does training vs test size impact the model's performance? Does it impact training and test accuracy the same way?

*use a loop for this part, possibly also a function*

### Tip

The s  
visual  
usefu  
perfo  
fitting  
to cre  
experi

### Thinking Ahead

*ideas for level 3, not required for A7*

Repeat the problem setup experiment with cross validation and plot with error bars.

- What is the tradeoff to be made in choosing a test/train size?
- What is the best test/train size for this dataset?

or with variations:

- allowing it to figure out the model depth for each training size, and recording the depth in the loop as well.
- repeating each size 10 items, then using summary statistics on that data

Use the extensions above to experiment further with other model parameters.

**some of this we'll learn how to automate in a few weeks, but getting the ideas by doing it yourself can help build understanding and intuition**

### Hints

The r  
max c  
across  
error,

## 8. Assignment 8: Clustering

accept the assignment **Due: 2023-03-29**

### 8.1. Evaluation

Eligible skills: (links to checklists)

- **first chance** clustering 1 and 2
- evaluate 1 and 2

- summarize 1 and 2
- visualize 1 and 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
- 

## 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.

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. Apply Kmeans using the known, correct number of clusters,  $K$ .
3. 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
4. Include a discussion of your results that addresses the following:
  - describes what the clustering means
  - 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)
5. 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 clusters than there are classes in this dataset?

## 9. Assignment 9: Linear Regression

### 9.1. Quick Facts

- accept the assignment
- Due: 2023-04-04

### 9.2. Assessment

Eligible skills: (links to checklists)

- process 1 and 2
- evaluate 1 and 2
- summarize 1 and 2
- visualize 1 and 2

## 9.3. Related notes

- Regression Preview
- Regression

## 9.4. Instructions

Find a dataset suitable for regression. We recommend a dataset from the UCI repository.

### 9.4.1. Linear Regression Basics

TLDR: Fit a linear regression model, measure the fit with two metrics, and make a plot that helps visualize the result.

1. Include a basic description of the data(what the features are)
2. Write your own description of what the prediction task it, why regression is appropriate.
3. Fit a linear model on all relevant 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:
  - Does this model make sense?
  - 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.

### 9.4.2. Part 2: Exploring Evaluation

Do an experiment to compare test set size vs performance:

1. Train a regression model on 10%, 30%, ... , 90% of the data. Save the results of both test and train performance for each size training data in a DataFrame with columns ['train\_pct', 'n\_train\_samples', 'n\_test\_samples', 'train\_r2', 'test\_r2']

[Skip to main content](#)

3. Interpret these results. How does training vs test size impact the model?

Tip

### Thinking Ahead

Try these experiments with a different type of regression.

The s  
visua  
usefu  
perfo  
fitting  
to cre  
experi

## 10. Assignment 10: Tuning Model Parameters

### 10.1. Quick Facts

- accept the assignment
- Due: 2023-04-14

### 10.2. Related notes

- ML Task Review and Cross Validation
- SVM and Model Optimization

### 10.3. Assessment

Eligible skills: (links to checklists)

- **first chance** optimization 1 and 2
- clustering 1 and 2
- regression 1 and 2
- classification 1 and 2
- evaluate (must use extra metrics to earn this here) 1 and 2
- summarize 1 and 2
- visualize 1 and 2

### 10.4. Instructions

**summary** Extend the work you did in assignment 7,8, or 9, by optimizing the model parameters.

1. Choose your dataset, task, and a model. It can be any model we have used in class so far. Include a brief description of the task(this can be copied from 7,8, or 9 if applicable but please include it for completeness).
2. Fit the model with default parameters and check the score. Interpret the score in context. (again can be reused)
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? Score the best fit model on a held out test set.
6. Examine and interpret the cross validation results. How do they vary in terms of time? Is the performance meaningfully

[Skip to main content](#)

7. Try varying the cross validation parameters (eg the number of folds and/or type of cross validation). Does this change your conclusions?

#### 💡 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

Assignment 11 will be to optimize two models and then compare two models on the same task

#### 🔔 Thinking Ahead

What other tradeoffs might you want to make in choosing a model? How could you present these results using your EDA skills?

## 11. Assignment 11: Model Comparison

### 11.1. Quick Facts

- accept the assignmnt
- Due: 2023-04-21

### 11.2. Related notes

- SVM and Model Optimization (partial)
- Model Comparison

### 11.3. Assessment

Eligible skills: (links to checklists)

- **first chance** compare 1 and 2
- optimization 1 and 2
- clustering 1 and 2
- regression 1 and 2
- classification 1 and 2
- evaluate (must use extra metrics to earn this here) 1 and 2
- summarize 1 and 2
- visualize 1 and 2



### 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

You can use the same dataset you used in one of the last few assignments

Choose a dataset, it can be appropriate for classification, regression, or clustering. Fit 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.

You can reuse a dataset you've used on for one of the previous assignments or choose another.



### Think Ahead

How would this decision making compare for a more complex model or in more realistic setting.

## 12. Assignment 12: Fake News

### 12.1. Quick Facts

- accept the assignment due date:2023-04-26\_

### 12.2. Related notes

- Intro to NLP- representing text data
- 

### 12.3. Assessment

Eligible skills: (links to checklists)

- first chance** representation 1 and 2 remember this is optional
- first chance** workflow 1 and 2 psuedo code and explanation is enough for this
- compare 1 and 2
- optimization 1 and 2
- clustering 1 and 2
- classification 1 and 2
- evaluate (must use extra metrics to earn this here) 1 and 2
- summarize 1 and 2
- visualize 1 and 2

## 12.4. Instructions

Use the dataset in the assignment template repo to answer the following questions.

1. Is the text or the title of an article more predictive of whether it is real or fake?
2. Are titles of real or fake news more similar to one another?

The data includes variables:

- 'text': contents of an article
- 'label': whether it is real or fake news
- 'title': title of the article

Include narrative around the code required to answer these and interpret the results to give an actual answer.

- Provide context on your answer and consider how strong it is based on what differences you can have in how you represent the data and how that might impact your model performance.
- Consider if the analysis you have done is enough evidence answer the question from the analysis you have completed or could something else change the answer.
- Use summary statistics and visualizations appropriately in order to explain your results.

### Hint

The data set contains a large number of articles (takes a long time to train), you can downsample this to something like a 1,000 articles or so in order to speed up training and evaluation (hint: use shuffle).

## Portfolio

This section of the site has a set of portfolio prompts and this page has instructions for portfolio submissions.

Starting in week 3 it is recommended that you spend some time each week working on items for your portfolio, that way when it's time to submit you only have a little bit to add before submission.

The portfolio is your only chance to earn Level 3 achievements, however, if you have not earned a level 2 for any of the skills in a given check, you could earn level 2 then instead. The prompts provide a starting point, but remember that to earn achievements, you'll be evaluated by the rubric. You can see the full rubric for all portfolios in the [syllabus](#). Your portfolio is also an opportunity to be creative, explore things, and answer your own questions that we haven't answered in class to dig deeper on the topics we're covering. Use the feedback you get on assignments to inspire your portfolio.

Each submission should include an introduction and a number of 'chapters'. The grade will be based on both that you demonstrate skills through your chapters that are inspired by the prompts and that your summary demonstrates that you *know* you learned the skills. See the [formatting tips](#) for advice on how to structure files.

On each chapter(for a file) of your portfolio, you should identify which skills by their keyword, you are applying.

You can view a (fake) example [in this repository](#) as a [pdf](#) or as a [rendered website](#)

- Portfolio Check 1 is due March 6
- Portfolio Check 2 is due April 7
- Portfolio check 3 is due April 24
- Portfolio check 4 is due May 9

Portfolio check 2 will assess the following *new* achievements in addition to an a chance to make up any that you have missed:

### Level 3

keyword	
<b>python</b>	reliable, efficient, pythonic code that consistently adheres to pep8
<b>process</b>	Compare different ways that data science can facilitate decision making
<b>access</b>	access data from both common and uncommon formats and identify best practices for formats in different contexts
<b>construct</b>	merge data that is not automatically aligned
<b>summarize</b>	Compute and interpret various summary statistics of subsets of data
<b>visualize</b>	generate complex plots with pandas and plotting libraries and customize with matplotlib or additional parameters
<b>prepare</b>	apply data reshaping, cleaning, and filtering manipulations reliably and correctly by assessing data as received
<b>evaluate</b>	Evaluate a model with multiple metrics and cross validation
<b>classification</b>	fit and apply classification models and select appropriate classification models for different contexts
<b>regression</b>	fit and explain regularized or nonlinear regression
<b>clustering</b>	apply multiple clustering techniques, and interpret results

## 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](#)

Your portfolio is a [jupyter book](#). This means a few things:

- it uses [myst markdown](#)
- it will run and compile Jupyter notebooks

This page will cover a few basic tips.

## Managing Files and version

You can either convert your ipynb files to earier to read locally or on GitHub.

The GitHub version means installing less locally, but means that after you push changes, you'll need to pull the changes that GitHub makes.

## To manage with a precommit hook jupytext conversion

change your `.pre-commit-config.yaml` file to match the following:

```
repos:
- repo: https://github.com/mwouts/jupytext
  rev: v1.10.0 # CURRENT_TAG/COMMIT_HASH
  hooks:
    - id: jupytext
      args: [--from, ipynb, --to, myst]
```

Run Precommit over all the files to actually apply that script to your repo.

```
pre-commit install
pre-commit run --all-files
```

If you do `git status` now, you should have a `.md` file for each `ipynb` file that was in your repository, now add and commit those.

Now, each time you commit, it will run jupytext first.

## To manage with a gh action jupytext conversion

create a file at `.github/workflows/jupytext.yml` and paste the following:

```
name: jupytext

# Only run this when the master branch changes
on:
  push:
    branches:
      - main
    # If your git repository has the Jupyter Book within some-subfolder next to
    # unrelated files, you can make this run only if a file within that specific
    # folder has been modified.
    #
    # paths:
    #   - some-subfolder/**

# This job installs dependencies, build the book, and pushes it to `gh-pages`
jobs:
  jupytext:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

    # Install dependencies
    - name: Set up Python 3.7
      uses: actions/setup-python@v1
      with:
        python-version: 3.7

      - name: Install dependencies
        run: |
          pip install jupytext
      - name: convert
        run: |
          jupytext */*.ipynb --to myst
          jupytext *.ipynb --to myst
      - uses: EndBug/add-and-commit@v4 # You can change this to use a specific version
        with:
```

[Skip to main content](#)

```

# Default: '.'
add: '.'

# The name of the user that will be displayed as the author of the commit
# Default: author of the commit that triggered the run
author_name: Your Name

# The email of the user that will be displayed as the author of the commit
# Default: author of the commit that triggered the run
author_email: you@uri.edu

# The local path to the directory where your repository is located. You should use actions/checkout :
# Default: '.'
cwd: '.'

# Whether to use the --force option on `git add`, in order to bypass eventual gitignores
# Default: false
force: true

# Whether to use the --signoff option on `git commit`
# Default: false
signoff: true

# The message for the commit
# Default: 'Commit from GitHub Actions'
message: 'convert notebooks to md'

# Name of the branch to use, if different from the one that triggered the workflow
# Default: the branch that triggered the workflow (from GITHUB_REF)
ref: 'main'

# Name of the tag to add to the new commit (see the paragraph below for more info)
# Default: ''
tag: "v1.0.0"

env:
  # This is necessary in order to push a commit to the repo
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }} # Leave this line unchanged

```

## 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
```

[Skip to main content](#)

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. Each `chapter` or file should have a descriptive file name (`with_no_spaces`) and descriptive title for it.

## Syncing markdown and ipynb files

If you have the precommit hook working, git will call a script and convert your notebook files from the ipynb format (which is json like) to Myst Markdown, which is more plain text with some header information. The markdown format works better with version control, largely because it doesn't contain the outputs.

If you don't get the precommit hook working, but you do get jupytext installed, you can set each file to sync.

## 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

[Skip to main content](#)

[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:
  allow_errors : true
```

## 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.

## Portfolio Check 1 Ideas

Remember you'll be graded against the [rubric] and the [achievement checklists], but these are ideas for the structure.

You can mix and match different formats to cover the skills collectively.

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).

### 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 all of the skills.

Be sure to check the checklists for how level 3s are more complex than level 2s. I recommend using office hours to help get ideas if you are not sure how to extend your analysis.

### Several shorter reflections/analyses

You could also submit a few shorter pieces that in total cover all of the skills. Some example formats:

### 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

Write a blog post styled Notebook that compares or analyzes something, for example:

- how do different ways of loading data compare
- describe best practices you've learned and show why they're good with examples

## Correction & Reflection

If you had trouble with an assignment so far, 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. Then go a little deeper in exploring the topic in that context to also earn level 3.

## 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.

## 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,  $7 + 5 = 12$ ,  $1+2 = 3$ ,  $3+1 = 3$ )

## Check 2 Ideas

For Check 2, all of the prompts from check 1 apply, plus the following additional prompts, since there are new skills.

If you have other ideas, you can also ask and those are likely possible.

# Level 1 Achievement Catchup

To make up level 1 achievements, include a detailed introduction file to your portfolio and one of the following (per skill):

- minor extensions to what we did in class
- answers to problems from the notes
- additional glossary terms
- psuedocode for one of the other prompts

## 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.

## Build a data set for Prediction

Build a dataset that works for prediction (classification, regression, or clustering) from other sources.

## Learn a new model

Repeat what you did in 7, 8, or 9, with a different model.

## 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.

## 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

[Skip to main content](#)

Tip

If you achieve with t  
repre  
next p

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] I 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 Beibhinn (beibhinn)
- 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?

There is no specific weight for any activities, because your grade is based on earning achievements for the skills listed in the [skills rubric](#).

However, if you do not submit (or earn no achievements from) assignments or portfolios, the maximum grade you can earn is a C. If you do not submit (or earn no achievements from) your portfolio, the maximum grade you can earn is a B.

### What time are assignments due?

End of day. I could start grading at any time the next morning. If your work is not there when I start grading it will not be graded, but if it is, I won't check the time stamp.

### Can I submit this assignment late if ...?

Late assignments are not accepted, however, your grade is based on the skills, not the assignments. All skills are assessed in at least two [assignments](#), so missing any one will not hurt your grade. If you need an accommodation because you cannot submit

## I don't understand my grade on this assignment

If you have questions about your grade, the best place to get feedback is to reply on the Feedback PR. Either reply directly to one of the inline comments, or the summary.

Be specific about what you think you should have earned and why.

## Git and GitHub

### I can't push to my repository, I get an error that updates were rejected

```
! [rejected] main -> main (fetch first)
error: failed to push some refs to <repository name>
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Your local version and github version are out of sync, you need to pull the changes from github to your local computer before you can push new changes there.

After you run

```
git pull
```

You'll probably have to resolve a merge conflict

### The content I added to my portfolio isn't in the pdf

There was an error in the original `_toc.yml` file, change yours to match the following:

```
format: jb-book
root: intro
parts:
  - caption: About
    chapters:
      - file: about/index
      - file: about/grading
  # - caption: Check 1
  #   chapters:
  #     - file: submission_1_intro
```

uncomment the later lines and add any new files you add.

### My command line says I cannot use a password

GitHub has [strong rules](#) about authentication. You need to use SSH with a public/private key; HTTPS with a [Personal Access Token](#)

[Skip to main content](#)

## My .ipynb file isn't showing in the staging area or didn't push

.ipynb files are json that include all of the output, including tables as html and plots as svg, so, unlike plain code files, they don't play well with version control.

Your portfolio has `/*/*.ipynb` in the `.gitignore` file, so that these files do not end up in your repository. Instead, you'll convert your notebooks to [Myst Markdown](#) with [jupytext](#) via a [precommit hook](#).

Your portfolio has the code to do this already, what you should do is make sure that `pre-commit` is installed and then run

`pre-commit install`

(see your portfolio's [README.md](#) file for more detail)

If this doesn't work, you can follow the alterntive in the porfolio readme.

If that doesn't work, and you have time before the deadline, create an issue to get help.

As a last resort, use the jupyter interface to download (File > Download as > ...)your notebook as `.md` if avialable or `.py` if not and then move that file from your Downloads folder to your repository. We'll set up another workflow for future work

## My portfolio won't compile

If there's an error your notebook it can't complete running. You can allow it to run if the error is on purpose by changing settings as mentioned on the formatting page.

## Help! I accidentally merged the Feedback Pull Request before my assignment was graded

That's ok. You can fix it.

You'll have to work offline and use GitHub in your browser together for this fix. The following instuctions will work in terminal on Mac or Linux or in GitBash for Windows. (see Programming Environment section on the tools page).

First get the url to clone your repository (unless you already have it cloned then skip ahead): on the main page for your repository, click the green "Code" button, then copy the url that's show

[Code](#)[Issues](#)[Pull requests](#)[Actions](#)[Projects](#)[Wiki](#)[Security](#)[Insights](#)[Settings](#)

**feedback** had recent pushes 1 minute ago

[Compare & pull request](#)[main](#)[5 branches](#)[1 tag](#)[Go to file](#)[Add file](#)[Code](#)**brownsarahm** update toc to include notebook

|                |                                      |
|----------------|--------------------------------------|
| .github        | correct path for jupytext conversion |
| about          | mvoe notebook                        |
| template_files | convert notebooks to md              |
| .gitignore     | merge gh changes and ignore          |
| README.md      | Initial commit                       |

[Clone with HTTPS](#)[Use SSH](#)

Use Git or checkout with SVN using the web URL.

<https://github.com/rhodyprog4ds/por>[Open with GitHub Desktop](#)[Download ZIP](#)

Next open a terminal or GitBash and type the following.

```
git clone
```

then past your url that you copied. It will look something like this, but the last part will be the current assignment repo and your username.

```
git clone https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
```

When you merged the Feedback pull request you advanced the **feedback** branch, so we need to hard reset it back to before you did any work. To do this, first check it out, by navigating into the folder for your repository (created when you cloned above) and then checking it out, and making sure it's up to date with the **remote** (the copy on GitHub)

```
cd portfolio-brownsarahm  
git checkout feedback  
git pull
```

Now, you have to figure out what commit to revert to, so go back to GitHub in your browser, and switch to the feedback branch there. Click on where it says **main** on the top right next to the branch icon and choose feedback from the list.

[rhodyprog4ds / portfolio-brownsarahm](#) Private

generated from rhodyprog4ds/portfolio

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

**feedback** had recent pushes 1 minute ago

Compare & pull request

main ▾ 5 branches 1 tag Go to file Add file ▾ Code ▾

Switch branches/tags X

Find or create a branch...

Branches Tags

✓ main default

feedback

gh-pages

someOtherBranch

| commit                               | time           | commits    |
|--------------------------------------|----------------|------------|
| a6f7f45 15 minutes ago               | 15 minutes ago | 14 commits |
| correct path for jupytext conversion | 17 hours ago   |            |
| mvoe notebook                        | 17 minutes ago |            |
| convert notebooks to md              | 17 hours ago   |            |
| merge gh changes and ignore          | 3 days ago     |            |
| Initial commit                       | 3 days ago     |            |

Now view the list of all of the commits to this branch, by clicking on the clock icon with a number of commits

[rhodyprog4ds / portfolio-brownsarahm](#) Private

generated from rhodyprog4ds/portfolio

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

**feedback** had recent pushes 15 minutes ago

Compare & pull request

feedback ▾ 5 branches 1 tag Go to file Add file ▾ Code ▾

This branch is 1 commit ahead of main.

Pull request Compare

| commit                                                       | time           | commits    |
|--------------------------------------------------------------|----------------|------------|
| f301d90 16 minutes ago                                       | 16 minutes ago | 15 commits |
| brownsarahm Merge pull request #1 from rhodyprog4ds/main ... |                |            |
| .github                                                      | 17 hours ago   |            |
| correct path for jupytext conversion                         |                |            |
| about                                                        | 20 minutes ago |            |
| mvoe notebook                                                |                |            |
| template_files                                               | 17 hours ago   |            |
| convert notebooks to md                                      |                |            |

On the commits page scroll down and find the commit titled "Setting up GitHub Classroom Feedback" and copy its hash, by clicking on the clipboard icon next to the short version.

|                                      |                      |         |
|--------------------------------------|----------------------|---------|
| more examples                        | <a href="#">View</a> | 9427c13 |
| brownsarahm committed 3 days ago     |                      |         |
| convert notebooks to md              | <a href="#">View</a> | e2f5b79 |
| brownsarahm committed 3 days ago     |                      |         |
| Update jupytext_ipynb_md.yml         | <a href="#">View</a> | 7bd76c6 |
| brownsarahm committed 3 days ago ✓   | Verified             |         |
| solution                             | <a href="#">View</a> | fbe6613 |
| brownsarahm committed 3 days ago ✓   |                      |         |
| Setting up GitHub Classroom Feedback | <a href="#">View</a> | 822cf5  |
| brownsarahm committed 3 days ago ✘   |                      |         |
| GitHub Classroom Feedback            | <a href="#">View</a> | f3e0297 |
| brownsarahm committed 3 days ago ✘   |                      |         |
| Initial commit                       | <a href="#">View</a> | 66c21c3 |
| brownsarahm committed 3 days ago ✓   |                      |         |

Newer    Older

Now, back on your terminal, type the following

```
git reset --hard
```

then paste the commit hash you copied, it will look something like the following, but your hash will be different.

```
git reset --hard 822cf51a70d356d448bcaede5b15282838a5028
```

If it works, your terminal will say something like

```
HEAD is now at 822cf5 Setting up GitHub Classroom Feedback
```

but the number on yours will be different.

Now your local copy of the `feedback` branch is reverted back as if you had not merged the pull request and what's left to do is to push those changes to GitHub. By default, GitHub won't let you push changes unless you have all of the changes that have been made on their side, so we have to tell Git to force GitHub to do this.

Since we're about to do something with forcing, we should first check that we're doing the right thing.

```
git status
```

and it should show something like

```
on branch feedback
Your branch is behind 'origin/feedback' by 12 commits.  and can be fast-forwarded.
```

[Skip to main content](#)

Your number of commits will probably be different but the important things to see here is that it says `On branch feedback` so that you know you're not deleting the `main` copy of your work and `Your branch is behind origin/feedback` to know that reverting worked.

Now to make GitHub match your reverted local copy.

```
git push origin -f
```

and you'll get something like this to know that it worked

```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
 + f301d90...822cfe5 feedback -> feedback (forced update)
```

Again, the numbers will be different and it will be your url, not mine.

Now back on GitHub, in your browser, click on the code tab. It should look something like this now. Notice that it says, "This branch is 11 commits behind main" your number will be different but it should be 1 less than the number you had when you checked `git status`. This is because we reverted the changes you made to main (11 for me) and the 1 commit for merging main into feedback. Also the last commit (at the top, should say "Setting up GitHub Classroom Feedback").

This branch is 11 commits behind main.

brownsarahm Setting up GitHub Classroom Feedback 3 days ago 3 commits

| File           | Commit Message            | Date       |
|----------------|---------------------------|------------|
| .github        | GitHub Classroom Feedback | 3 days ago |
| about          | Initial commit            | 3 days ago |
| template_files | Initial commit            | 3 days ago |
| .gitignore     | Initial commit            | 3 days ago |
| README.md      | Initial commit            | 3 days ago |

Now, you need to recreate your Pull Request, click where it says pull request.

generated from [rhodyprog4ds/portfolio](#)

[Code](#)

[Issues](#)

[Pull requests](#)

[Actions](#)

[Projects](#)

[Wiki](#)

[Security](#)

[Insights](#)

[Settings](#)

[feedback](#) ▼

[5 branches](#)

[1 tag](#)

[Go to file](#)

[Add file](#) ▼

[Code](#) ▼

This branch is 11 commits behind main.

[Pull request](#) [Compare](#)



brownsarahm Setting up GitHub Classroom Feedback

[X](#) 822cfe5 3 days ago [3 commits](#)

|                                |                           |            |
|--------------------------------|---------------------------|------------|
| <a href="#">.github</a>        | GitHub Classroom Feedback | 3 days ago |
| <a href="#">about</a>          | Initial commit            | 3 days ago |
| <a href="#">template_files</a> | Initial commit            | 3 days ago |
| <a href="#">.gitignore</a>     | Initial commit            | 3 days ago |
| <a href="#">README.md</a>      | Initial commit            | 3 days ago |

It will say there isn't anything to compare, but this is because it's trying to use [feedback](#) to update [main](#). We want to use [main](#) to update [feedback](#) for this PR. So we have to swap them. Change base from [main](#) to [feedback](#) by clicking on it and choosing [Feedback](#) from the list.

generated from [rhodyprog4ds/portfolio](#)

[Code](#)

[Issues](#)

[Pull requests](#)

[Actions](#)

[Projects](#)

[Wiki](#)

[Security](#)

[Insights](#)

[Settings](#)

## Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

[▼](#)

[base: main](#) ▼

[◀](#)

[compare: feedback](#) ▼

Choose a base ref

[Find a branch](#)

[Branches](#)

[Tags](#)

[✓ main](#)

[default](#)

[feedback](#)

[gh-pages](#)

[someOtherBranch](#)



**There isn't anything to compare.**

up to date with all commits from [feedback](#). Try [switching the base](#) for your comparison.

eletions.

Then change the compare [feedback](#) on the right to [main](#). Once you do that the page will change to the "Open a Pull Request" interface.

[Skip to main content](#)

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

The screenshot shows the GitHub pull request interface. At the top, there are dropdown menus for 'base: feedback' and 'compare: main'. A green checkmark icon indicates that the branches are 'Able to merge'. Below this, a user profile picture is visible next to the title 'Feedback'. There are two tabs: 'Write' (selected) and 'Preview'. A toolbar above the body contains icons for bold (B), italic (I), horizontal line (H), copy (C), and other GitHub-specific functions. The main body area has a placeholder 'Leave a comment' and a note at the bottom: 'Attach files by dragging & dropping, selecting or pasting them.' A small 'M+>' icon is in the bottom right corner of the body area.

Make the title “Feedback” put a note in the body and then click the green “Create Pull Request” button.

Now you’re done!

If you have trouble, create an issue and tag `@rhodyprog4ds/fall22instructors` for help.

## Code Errors

### Key Error

If you get a key error for a pandas operation, it means that the column name as you typed it is not in the DataFrame. Check the spelling, leading or trailing whitespace can be especially troubling.

### <bound method

You’re probably missing `( )` on a method, so Python returned the method itself as an object instead of calling it and returning the output.

## Glossary

### Ram Token Opportunity

Contribute glossary items and links for further reading using the suggest an edit button behind the GitHub menu at the top of the page

[Skip to main content](#)

## **aggregate**

to combine data in some way, a function that can produce a customized summary table

## **anonymous function**

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.

## **dictionary**

(data type) a mapping array that matches keys to values. (in NLP) all of the possible tokens a model knows

## **document**

unit of text for analysis (one sample). Could be one sentence, one paragraph, or an article, depending on the goal

## **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

## **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 involves 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

## 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>`

## 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

## suffix

additional part of the name that gets added to end of a name in a merge operation

## 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

## Split Apply Combine

a paradigm for splitting data into groups using a column, applying some function(aggregation, transformation, or filtration) to each piece and combining 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

## 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 call tracing back through the stack

## training accuracy

## 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
```

|   | 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 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.
- Google Dataset Search
- 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

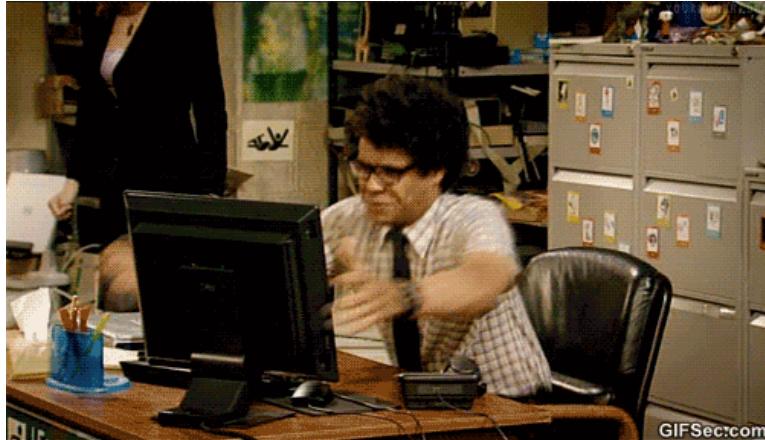
This section is for materials that are not specific to this course, but are likely useful. They are not generally required readings or installs, but are options or advice I provide frequently.

### on email

- [how to e-mail professors](#)

## 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.

A new book  
programming  
Brain As of  
by clicking  
contents so

### Where are your help tools?

In Python and Jupyter notebooks, what help tools do you have?

[Skip to main content](#)

# 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 hand, diving deeper into options and more complex applications of what you have already seen in class, while challenging, is 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.

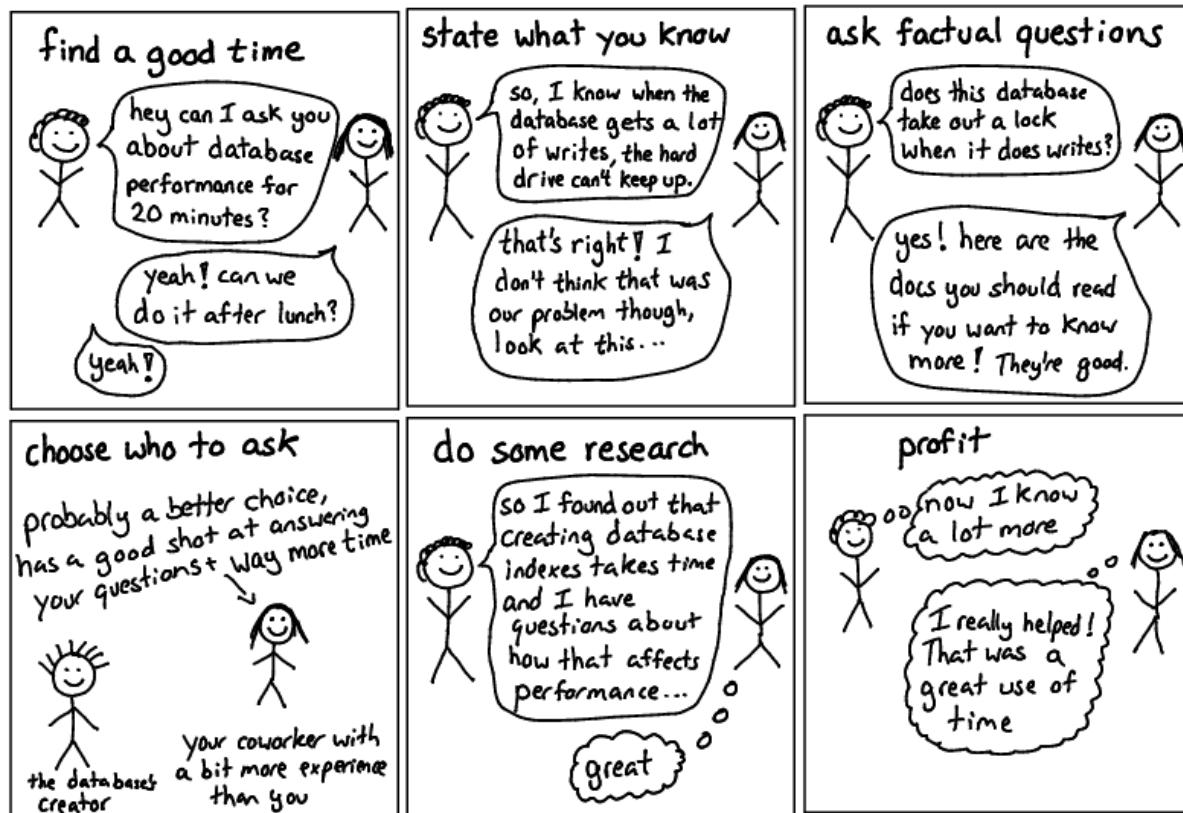
## Portfolios

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

## Asking Questions

JULIA EVANS  
@b0rk



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.

As such, they have written a good guide on [creating a minimal, reproducible example](#).

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.

i Note

A fun  
debug

## 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.

[Skip to main content](#)

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 of the code you ever run must run in *some* sort of environment. By giving you control, I'm hoping to increase your independence 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 realistic 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 manner.

## Windows

Windows has a sort of multiverse of terminal environments.

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

## 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](#).

 Note

We know teaching so in new that we understand this follows and th

If, for example, have never and you're will be hurt at that point

# 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 will 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.

## Letters to Future students

This section is a place for students enrolled in Fall 2020 to write letters to future students taking this class with Professor Brown. The websites for future sections will link back here for them to read.

## Contributed Notes

### Reviewing notes

Especially when it comes to the difficult topics make sure you go back through the notes and try and add to them yourself. Actively using the new topics will help you learn a lot better than just copying the notes.

## Attend Office Hours

Attending office hours will help you better understand course material, complete homework assignments, and learn new things that might not normally come up in class.

- David

## To contribute

Via GitHub directly:

1. Use the edit button above to add a note to this file following the example that's commented out
2. create a pull request