

# About this Book

## Contents

### Syllabus

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

### Notes

- 1. Welcome & What is Data Science
- 2. Iterables and Pandas Data Frames
- 3. DataFrames from other sources
- 4. Exploratory Data Analysis
- 5. Visualization
- 6. Tidy Data and Structural Repairs
- 7. Reparing values
- 8. Merging Data
- 9. Web Scraping
- 10. Evaluating ML Algorithms
- 11. Intro to ML & Naive Bayes
- 12. Understanding Classification
- 13. Clustering
- 14. Clustering Metrics
- 15. Regression
- 16. Interpreting Regression

### Assignments

- 1. Assignment 1: Setup, Syllabus, and Review
- 2. Assignment 2: Practicing Python and Accessing Data
- 3. Assignment 3: Exploratory Data Analysis
- 4. Assignment 4: Cleaning Data
- 5. Assignment 5: Constructing Datasets and Using Databases

[Skip to main content](#)

- 6. Assignment 6: Auditing Algorithms
- 7. Assignment 7
- 8. Assignment 8: Clustering
- 9. Assignment 9: Linear Regression

## Portfolio

- Portfolio
- Formatting Tips
- Portfolio Check 1 Ideas
- Check 2 Ideas
- Check 3 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

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

This class meets TTh 5-6:15pm in Engineering 040.

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

# Land University of Rhode Island Land Acknowledgment

## Note

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

[Skip to main content](#)

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

## Navigating the Sections

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

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

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

## Reading each page

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

### Try it Yourself

Notes will have exercises marked like this

### Question from Class

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

### Further reading

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

### Hint

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

### Think Ahead

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

### Click here!



Special tips will be formatted like this

### Check your Comprehension

Questions to use to check your comprehension will looklike this

# About

## About the topic

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

## About the goals and preparation

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

Basic programming skills (CSC201 or CSC211) are a prerequisite to this course. This course is a prerequisite course to 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 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, and online at [rhodyprog4ds.github.io/BrownFall23/syllabus](https://rhodyprog4ds.github.io/BrownFall23/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.

# About your instructor

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

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

## Important

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

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

# Tools and Resources

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

All of the tools and resources below are either:

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

## BrightSpace

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

## Prismia chat

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

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

## Course website

The course manual will have content including the class policies, scheduling, class notes, assignment information, and additional

[Skip to main content](#)

## 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 use the [GitHub CLI](#) or create a Personal Access Token for each device you use. In order to use the command line with SSH, set up your public key.

## Programming Environment

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

### Requirements:

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

#### ⚠ Warning

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

### Recommendation:

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

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

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

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

[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](#).

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

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

week	topics	skills
1	[admin, python review]	process
2	Loading data, Python review	[access, prepare, summarize]
3	Exploratory Data Analysis	[summarize, visualize]
4	Data Cleaning	[prepare, summarize, visualize]
5	Databases, Merging DataFrames	[access, construct, summarize]
6	Modeling, classification performance metrics, cross validation	[evaluate]
7	Naive Bayes, decision trees	[classification, evaluate]
8	Regression	[regression, evaluate]
9	Clustering	[clustering, evaluate]
10	SVM, parameter tuning	[optimize, tools]
11	KNN, Model comparison	[compare, tools]
12	Text Analysis	[unstructured]
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 adherance	python code that reliably runs, frequent pep8 adherance	reliable, efficient, pythonic code that consistently adheres to pep8
<b>process</b>	describe data science as a process	Identify basic components of data science	Describe and define each stage of the data science process	Compare different ways that data science can facilitate decision making
<b>access</b>	access data in multiple formats	load data from at least one format; identify the most common data formats	Load data for processing from the most common formats; Compare and contrast most common formats	access data from both common and uncommon formats and identify best practices for formats in different contexts
<b>construct</b>	construct datasets from multiple sources	identify what should happen to merge datasets or when they can be merged	apply basic merges	merge data that is not automatically aligned
<b>summarize</b>	Summarize and describe data	Describe the shape and structure of a dataset in basic terms	compute summary standard statistics of a whole dataset and grouped data	Compute and interpret various summary statistics of subsets of data
<b>visualize</b>	Visualize data	identify plot types, generate basic plots from pandas	generate multiple plot types with complete labeling with pandas and seaborn	generate complex plots with pandas and plotting libraries and customize with matplotlib or additional parameters
<b>prepare</b>	prepare data for analysis	identify if data is or is not ready for analysis, potential problems with data	apply data reshaping, cleaning, and filtering as directed	apply data reshaping, cleaning, and filtering manipulations reliably and correctly by assessing data as received
<b>evaluate</b>	Evaluate model performance	Explain and compute basic performance metrics for different data science tasks	Apply and interpret basic model evaluation metrics to a held out test set	Evaluate a model with multiple metrics and cross validation
<b>classification</b>	Apply classification	identify and describe what classification is, apply pre-fit classification models	fit, apply, and interpret preselected classification model to a dataset	fit and apply classification models and select appropriate classification models for different contexts
<b>regression</b>	Apply Regression	identify what data that can be used for regression looks like	fit and interpret linear regression models	fit and explain regularized or nonlinear regression
<b>clustering</b>	Clustering	describe what clustering is	apply basic clustering	apply multiple clustering techniques, and interpret results
<b>optimize</b>	Optimize model parameters	Identify when model parameters need to be optimized	Optimize basic model parameters such as model order	Select optimal parameters based of mutiple quantiative 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.

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

- [ ] use of control structures

- [ ] correct calls to functions
- [ ] correct use of variables
- [ ] use of logical operators

## python-level2

*python code that reliably runs, frequent pep8 adherence*

- [ ] descriptive variable names
- [ ] pythonic loops
- [ ] effective 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
- [ ] identify 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 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*

## **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
- [ ] interpret 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 calculations 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 appropriate 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 and compute basic performance metrics for different data science tasks*

- [ ] apply at least one metric
- [ ] interpret model performance in context

## **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**

Evaluate a model with multiple metrics and cross validation

- [ ] 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

*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 quantiateve 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 quantitateve 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 achievable 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.

[Skip to main content](#)

This is a hard student who is below a C. Level 3s are any work in

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 earned all achievements by the 3rd portfolio check, you do not need to submit the fourth one.

**The easiest way to succeed at your portfolio is to extend your assignments**

## 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

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.

The letter grade can be computed as follows



In this achieve beca

## ! Important

this will be revealed after assignment 1

## Grading Examples

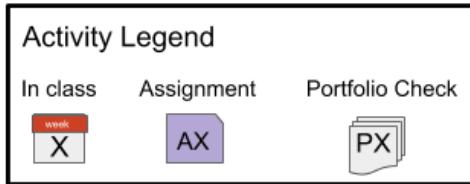
If you always attend and get everything correct, you will earn an A and you won't need to submit the 4th portfolio check.

### 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	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	11	A13	P3
unstructured	week 12	A13	P4
tools	week 11	A13	P3



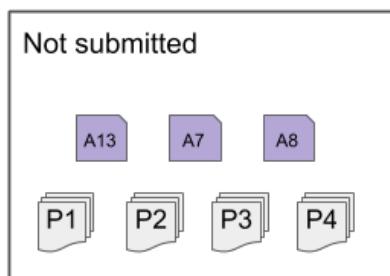
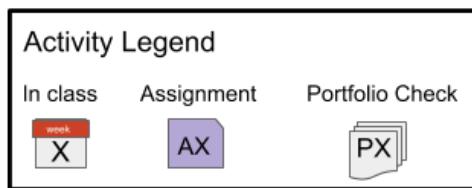
### Other Activities

- 1 Attended, but did not understand
- A4 Submitted, but incorrect
- 6 Missed class
- A6 Not submitted
- A7 Submitted, but incorrect
- A8 Not submitted
- A12 Not submitted
- 13 Attended, but all level 1 complete
- 14 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	week 1	A3	
process	week 1	A1	
access	2	A2	
construct	week 5	A5	
summarize	week 3	A3	
visualize	3	A3	
prepare	week 4	A4	
classification	10	A6	
regression	8	A11	
clustering	week 9	A9	
evaluate	week 7	A10	
optimize	week 10	A10	
compare	11	A11	
unstructured	week 12	A12	
tools	week 11	A12	

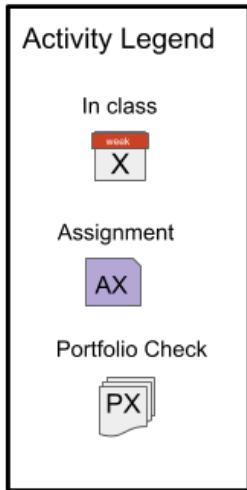


In this example, the student earned all level 1 achievements in class and all level 2 on assignments. This student was content with getting a B and chose to not submit a portfolio.

Getting a B while having trouble

# Map to a B, having trouble

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



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

## Grading Policies

### Attendance

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

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

Absences do not require notification.

### Assignment Deadlines and Late Work

**Late assignments will not be graded.** Extensions will not be granted for assignments. Every skill will be assessed through more than one assignment, so missing assignments occasionally will not necessarily impact your grade. If you do not submit **any** assignments that cover a given skill, you may earn the level 2 achievement in that skill through a portfolio check, but you will have fewer chances to earn level 3 in that skill.

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

### Important

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

## Portfolio Deadlines and Extensions

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

In this issue, include:

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

### Important

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

This request should be no more than 7 sentences.

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

## Academic Dishonesty

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

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

- in a portfolio, all achievements attempted in the dishonest component are permanently ineligible.
- in an assignment the level three achievements for the skills of focus in the assignment are ineligible, and the relevant level two for those skills requires meeting the standard for the level 3.

the requirements for prepare levels in a portfolio in order to earn prepare level 2.

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

## Regrading

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

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

## Course Style Guide

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

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

## Hard Requirements

### ⚠ Warning

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

1. All code must be submitted in a notebook file (.ipynb or myst)
2. Code must run or have explicit questions and comments about what was done about the errors
3. Python comments (`# comment text`) inside code cells should **only** be used to explain complex code that is not explained in the course notes. Using such code requires a citation for the source.
4. Each code cell should represent at most one conceptually complete step in terms of the analysis.
5. Every code cell must be motivated by text in markdown before it
6. Every code cell's output must be interpreted
7. the `print` function can only be used when it improves the readability over using jupyter's display, must be justified
8. No deprecated or dangerous code constructs without justification
9. All assignment questions must be answered in markdown cells
10. Notebook files may not have extraneous metadata in them

## Additional Style

[Skip to main content](#)

### Important

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

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

## Support

### Warning

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

## Academic Enhancement Center

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

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

## **⚠ Warning**

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

## **Anti-Bias Statement:**

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

## **Mental Health and Wellness**

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

## **Disability Services for Students Statement:**

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

## **Academic Honesty**

Students are expected to be honest in all academic work. A student's name on any written work, quiz or exam shall be 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 directly or indirectly through the use of generative AI
- Facilitating or aiding another's academic dishonesty
- Submitting the same paper for more than one course without prior approval from the instructors

# Communications & Office Hours

## ⚠ Warning

Due to Dept Seminar Office hours on 10/13 will be at 5pm instead of 4pm.

## Announcements

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

## Help Hours

Day	Time	Location	Host
Monday	12pm-2pm	Zoom	Mark
Monday	4-5pm	Zoom	Dr. Brown
Friday	4-5pm	134 Tyler	Dr. Brown

Zoom links are on the course organization page of [GitHub](#)

## To reach out, By usage

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

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

## ℹ Note

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

# Tips

## For assignment help

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

## Using issues

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

## For E-mail

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

i Note

What  
not m

# 1. Welcome & What is Data Science

## 1.1. Prismia Chat

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

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

## 1.2. 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

The audience is different, so the form is different.

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

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

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

So programming for data science is more like *writing* it has a narrative flow and is made to be seen more than some other programming that you may have done.

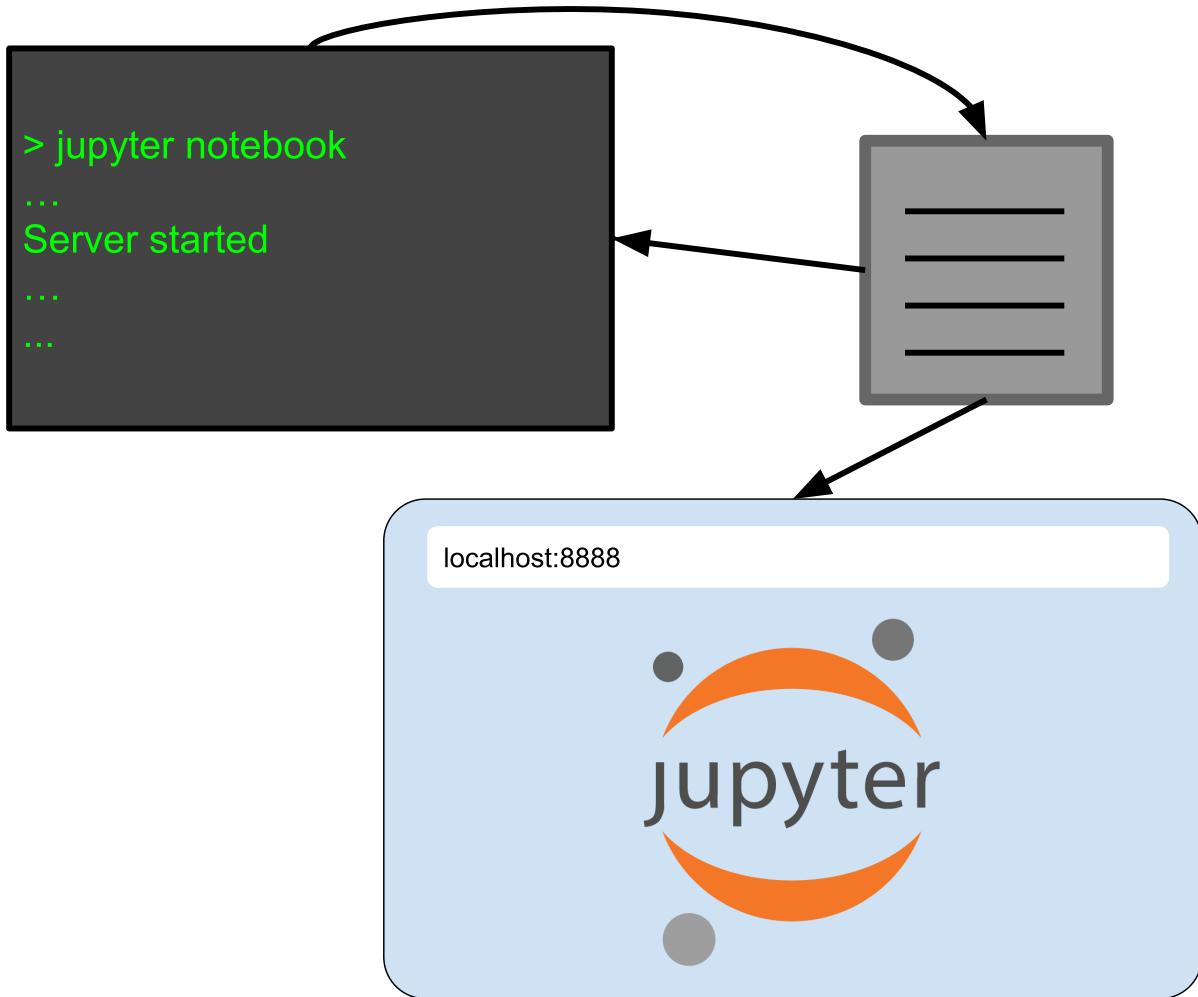
## 1.4. Jupyter Lab and Jupyter notebooks

Launch a `jupyter lab` server:

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

### 1.4.1. What just happened?

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



### 1.4.2. A jupyter notebook tour

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

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

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

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

This is a markdown cell

- we can make
- itemized lists of
- bullet points

1. and we can make numbered

2. lists and not have to worry

4. if we add a step in the middle later

```
# this is a comment in a code cell  
3+9
```

12

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

We can set variables

```
name = 'sarah'
```

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

we can put a variable there to see it

```
name
```

```
'sarah'
```

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

it only does that for the last line, so this one displays nothing

### ! 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*3
```

```
'sarahsarahsarah'
```

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

use enter/return to get to edit mode

## 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.

```
print(name, course)
```

```
sarah csc310
```

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

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

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

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

```
sarah_csc310
```

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

```
help(print)
```

Help on built-in function print in module builtins:

```
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.
```

This looks similar to the one above

```
print(name + '_' + course)
```

```
sarah_csc310
```

but if we want to use more parameters, using that parameter becomes more helpful.

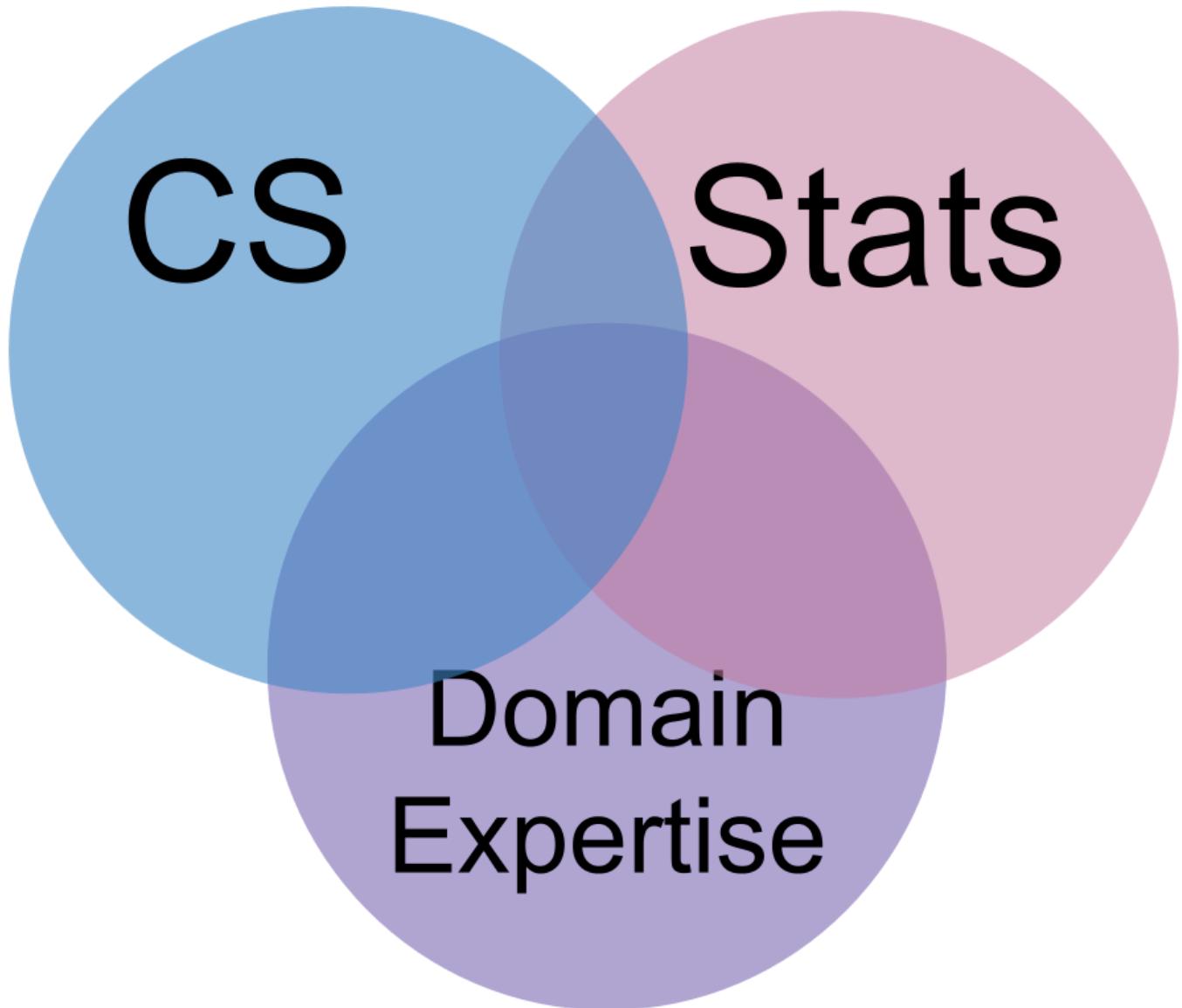
```
print(name, course, 'hello', "bye", sep='_')
```

```
sarah_csc310_hello_bye
```

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

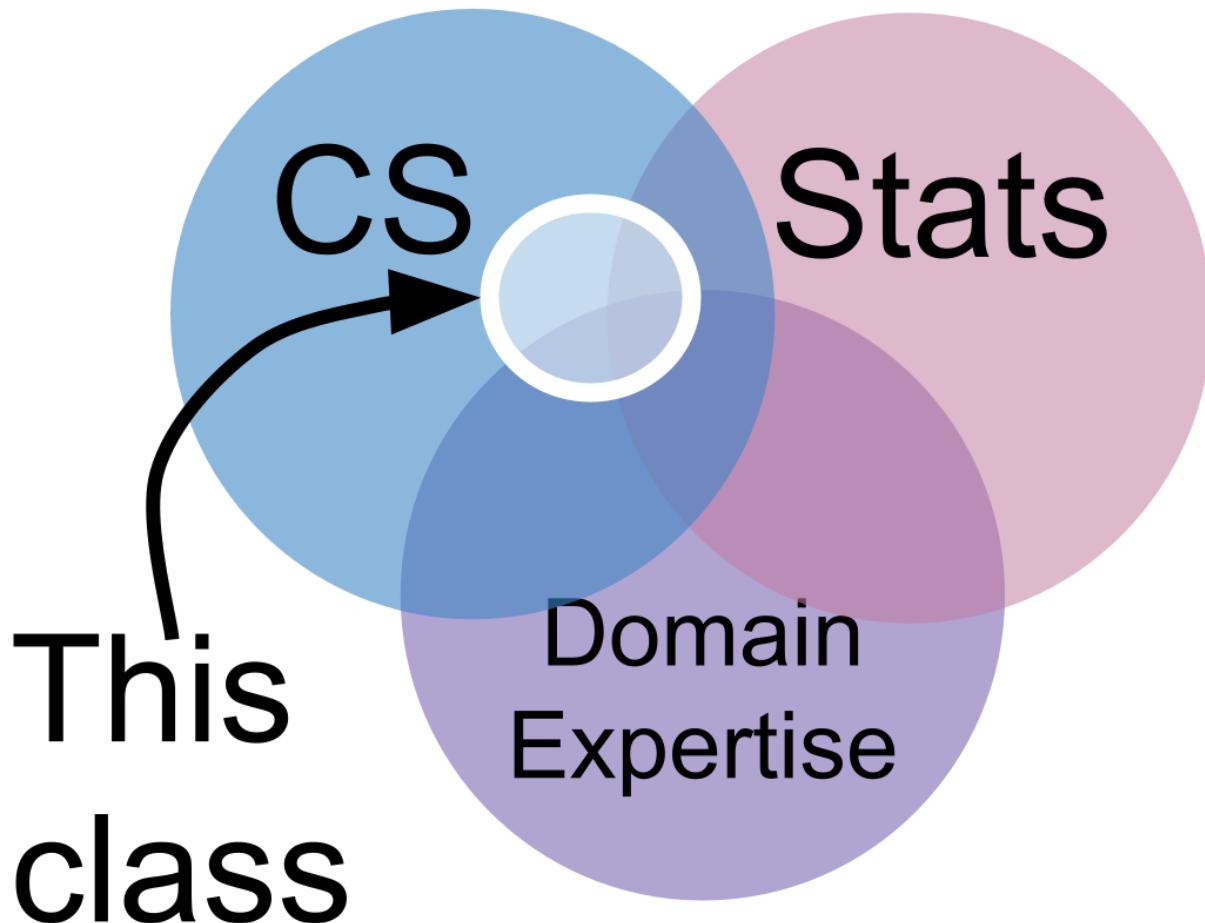


**statistics** is the type of math we use to make sense of data. Formally, a statistic is just a function of data.

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

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,

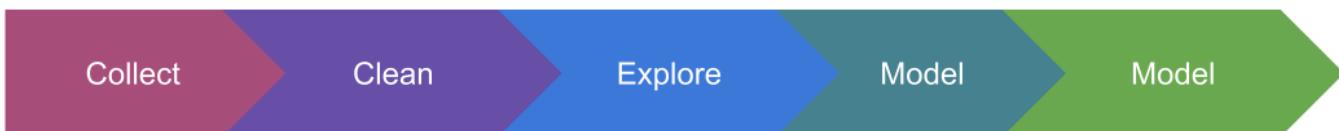


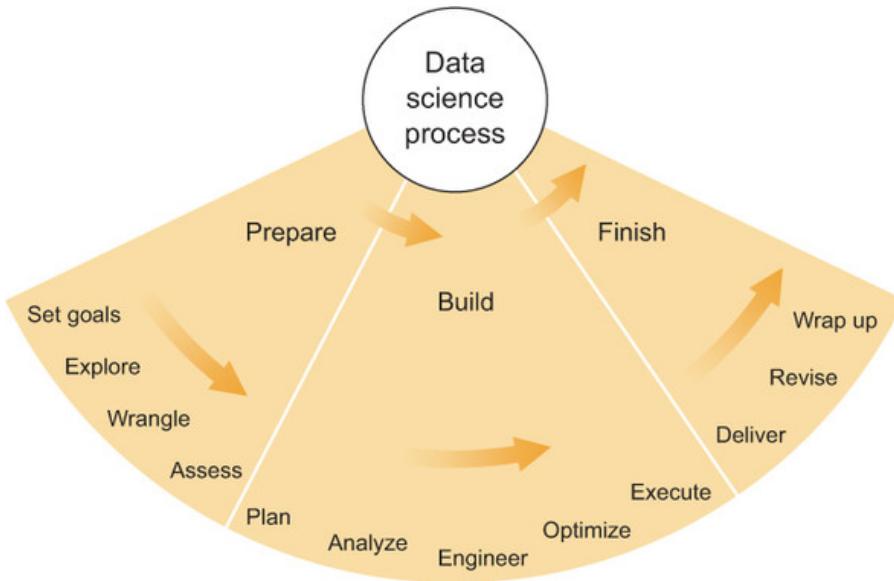
We'll focus on the programming as our main means of studying data science, but we will use bits of the other parts. In particular, you're encouraged to choose datasets that you have domain expertise about, or that you want to learn about.

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

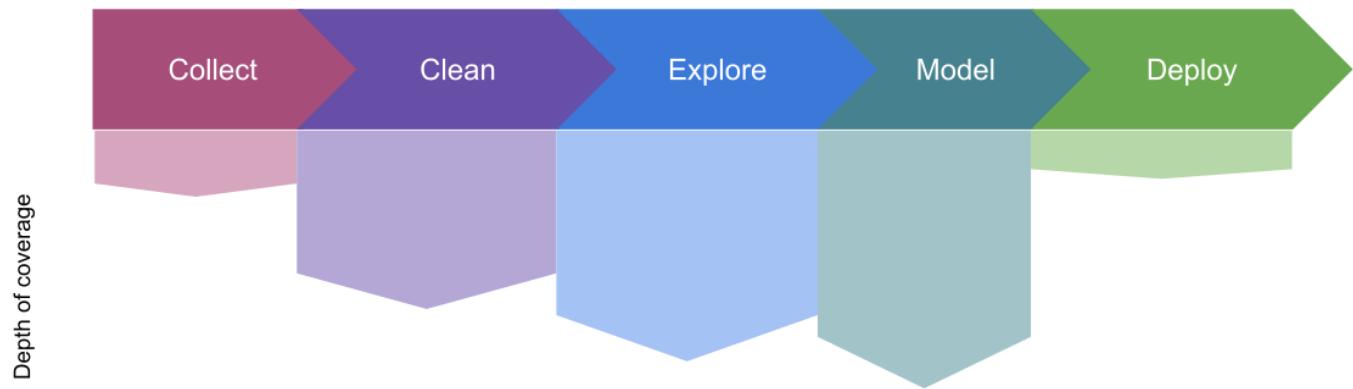
### 1.6.2. How does data science happen?

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.



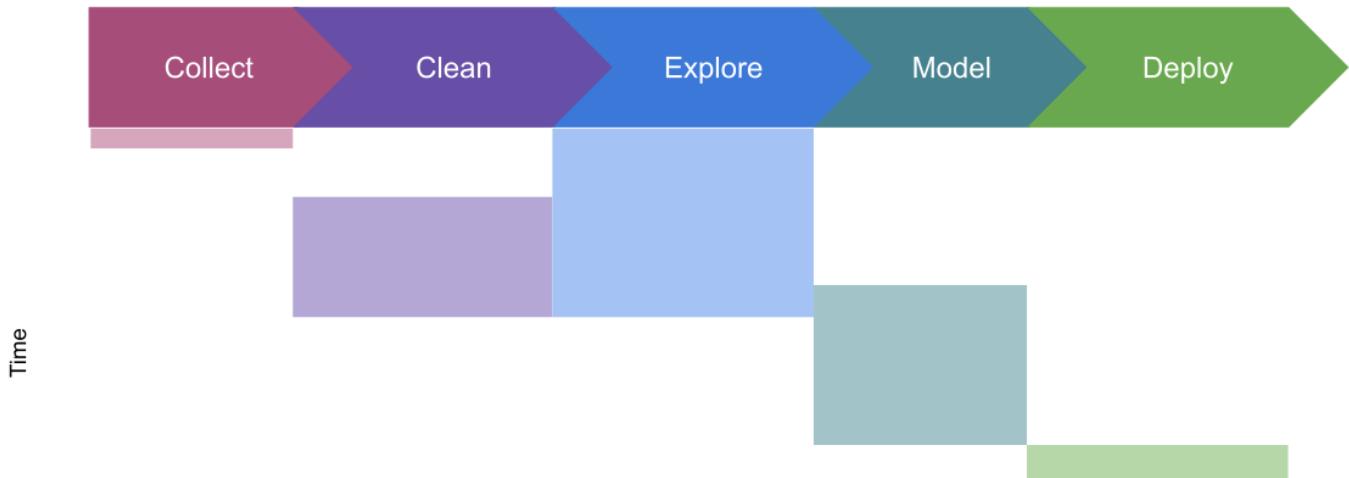


### 1.6.3. how we'll cover Data Science, in depth



- *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. 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.

## 1.8. Functions

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

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

A few things to note:

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

In Data Science, `numpydoc` style docstrings are popular.

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

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

```
greeting(name)
```

```
'hi sarah'
```

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

```
hi surbhi
```

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

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

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

## 1.9. Conditionals

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

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

key points in this function:

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

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

```
'hi sarah'
```

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

```
'hello  sarah'
```

## 1.10. Hints

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

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

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

## 2. Iterables and Pandas Data Frames

### 2.1. House Keeping

#### 2.1.1. Grading is not done,

you will get a notificaiton when yours is

#### 2.1.2. Closing Jupyter server.

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

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

or

do ctrl+C a second time.

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

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

#### Important

Remember to close your jupyter server

```

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

```

When we run the cell above that adds the function to memory.

Now that it is run, jupyter can show us `compute_grade` as an option when we tab complete after typing the first few letters.

When we restarted the kernel, we saw that before running the cell above, the tab complete did not work.

### **!** Important

this is important to understand what works when and why so that you know what to expect and can get unstuck

```
compute_grade(15,15,14)
```

```
'A- '
```

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

`assert` succeeds quietly

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

```
-----
AssertionError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 assert compute_grade(15,15,15) =='B'

AssertionError:
```

but fails with a specific error

The docstring is important, because it is the help.

```
help(compute_grade)
```

```
Help on function compute_grade in module __main__:

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 (+/-)
```

i Not

In cla  
help,  
notes  
functi

## 2.3. Everything is Data

Data we will see:

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

## 2.4. Why inspection in code?

[Skip to main content](#)

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

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

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

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

## 2.5. everything is an object

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

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

```
type(a)
```

```
int
```

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

strings are iterable type, meaning that they can be indexed into, or their elements iterated over. For a more technical definition, see the [official python glossary entry](#)

```
type(b)
```

```
str
```

we can select one element

```
b[0]
```

```
'm'
```

or multiple, this is called slicing.

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

negative numbers count from the right.

```
b[-1]
```

```
'y'
```

decimals default to float

```
type(c)
```

```
float
```

a variable can hold a whole function.

```
type(d)
```

```
builtin_function_or_method
```

functions are also objects like any other type in python

we can use the variable just like the function itself

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

```
hello
```

```
print('hello')
```

```
hello
```

## 2.6. Tabular Data

Structured data is easier to work with than other data.

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

## 2.7. Getting familiar with the dataset

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

- reviews added to DB
- then scraped

Where did it come from?

- coffee Quality Institute's trained reviewers.

what format is it provided in?

- csv (Comma Separated Values)

what other information is in this repository?

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

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

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

#### Further Reading

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

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

## 2.8. Loading data

Get raw url for the dataset click on the raw button on the csv page, then copy the url.

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

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

```
import pandas as pd
```

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

we will load the data with `pd.read_csv()`

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

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number
0	1 Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0
1	2 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21
2	3 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000
3	4 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0
4	5 Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0
5	6 Robusta	andrew hetzel	India	NaN	NaN	(self)	NaN
6	7 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN
7	8 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	7	sethuraman estate	14/1148/2017/18
8	9 Robusta	nishant gurjer	India	sethuraman estate	RKR	sethuraman estate	14/1148/2016/17
9	10 Robusta	ugacof	Uganda	ishaka	NaN	nsubuga umar	0
10	11 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0
11	12 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	RC AB	sethuraman estate	14/1148/2016/12
12	13 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN
13	14 Robusta	kasozi coffee farmers association	Uganda	kasozi coffee farmers	NaN	NaN	0
14	15 Robusta	ankole coffee producers coop	Uganda	kyangundu coop society	NaN	ankole coffee producers coop union ltd	0
15	16 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000
16	17 Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN

[Skip to main content](#)

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

28 rows × 44 columns

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

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

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

we can look at it again using the jupyter display

```
coffee_df
```

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number
0	1 Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0
1	2 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21
2	3 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000
3	4 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0
4	5 Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0
5	6 Robusta	andrew hetzel	India	NaN	NaN	(self)	NaN
6	7 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN
7	8 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	7	sethuraman estate	14/1148/2017/18
8	9 Robusta	nishant gurjer	India	sethuraman estate	RKR	sethuraman estate	14/1148/2016/17
9	10 Robusta	ugacof	Uganda	ishaka	NaN	nsubuga umar	0
10	11 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0
11	12 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	RC AB	sethuraman estate	14/1148/2016/12
12	13 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN
13	14 Robusta	kasozi coffee farmers association	Uganda	kasozi coffee farmers	NaN	NaN	0
14	15 Robusta	ankole coffee producers coop	Uganda	kyangundu coop society	NaN	ankole coffee producers coop union ltd	0
15	16 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000
16	17 Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN

[Skip to main content](#)

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

28 rows × 44 columns

Next we examine the type

```
type(coffee_df)
```

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

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

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

```
coffee_df.columns
```

```
Index(['Unnamed: 0', 'Species', 'Owner', 'Country.of-Origin', 'Farm.Name',  
       'Lot.Number', 'Mill', 'ICO.Number', 'Company', 'Altitude', 'Region',  
       'Producer', 'Number.of.Bags', 'Bag.Weight', 'In.Country.Partner',  
       'Harvest.Year', 'Grading.Date', 'Owner.1', 'Variety',  
       'Processing.Method', 'Fragrance...Aroma', 'Flavor', 'Aftertaste',  
       'Salt...Acid', 'Bitter...Sweet', 'Mouthfeel', 'Uniform.Cup',  
       'Clean.Cup', 'Balance', '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')
```

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

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

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

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

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

It also has an index (first column, visually) but it is special because this is how you can index the data.

```
coffee_df.index
```

```
RangeIndex(start=0, stop=28, step=1)
```

Right now this is an autogenerated index, but we can also use the `index_col` parameter to set that up front.

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

	<b>Species</b>	<b>Owner</b>	<b>Country.of.Origin</b>	<b>Farm.Name</b>	<b>Lot.Number</b>	<b>Mill</b>	<b>ICO.Number</b>	<b>Company</b>
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
6	Robusta	andrew hetzel	India	NaN	NaN	(self)	NaN	cafemakers, llc
7	Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers
8	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	7	sethuraman estate	14/1148/2017/18	kaapi royale
9	Robusta	nishant gurjer	India	sethuraman estate	RKR	sethuraman estate	14/1148/2016/17	kaapi royale
10	Robusta	ugacof	Uganda	ishaka	NaN	nsubuga umar	0	ugacof ltd
11	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd
12	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	RC AB	sethuraman estate	14/1148/2016/12	kaapi royale
13	Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers
14	Robusta	kasozi coffee farmers association	Uganda	kasozi coffee farmers	NaN	NaN	0	kasozi coffee farmers association
15	Robusta	ankole coffee producers coop	Uganda	kyangundu coop society	NaN	ankole coffee producers coop union ltd	0	ankole coffee producers coop
16	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate
17	Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc

[Skip to main content](#)

	<b>Species</b>	<b>Owner</b>	<b>Country.of.Origin</b>	<b>Farm.Name</b>	<b>Lot.Number</b>	<b>Mill</b>	<b>ICO.Number</b>	<b>Company</b>
<b>19</b>	Robusta	nitubaasa ltd	Uganda	kigezi coffee farmers association	NaN	nitubaasa	0	nitubaasa ltd
<b>20</b>	Robusta	mannya coffee project	Uganda	mannya coffee project	NaN	mannya coffee project	0	mannya coffee project
<b>21</b>	Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers
<b>22</b>	Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc
<b>23</b>	Robusta	andrew hetzel	United States	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc
<b>24</b>	Robusta	luis robles	Ecuador	robustasa	Lavado 1	our own lab	NaN	robustasa
<b>25</b>	Robusta	luis robles	Ecuador	robustasa	Lavado 3	own laboratory	NaN	robustasa
<b>26</b>	Robusta	james moore	United States	fazenda cazengo	NaN	cafe cazengo	NaN	global opportunity fund
<b>27</b>	Robusta	cafe politico	India	NaN	NaN	NaN	14-1118-2014-0087	cafe politico
<b>28</b>	Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaN	cafe politico

28 rows × 43 columns

```
coffee_df.index
```

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

Now we see that it uses the actual first column as the index that is bolded.

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

```
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

### Try it yourself

How can you look at the first 3 or last 2 rows?

and the last 5 with `tail`

```
coffee_df.tail()
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	I
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

### Important

We did not do this step in class

```
coffee_df.shape
```

```
(28, 43)
```

We can pick out columns by name.

```
coffee_df['Color']
```

```
1      Green
2      NaN
3      Green
4      Green
5      Green
6      Green
7      Green
8    Bluish-Green
9      Green
10     Green
11     Green
12     Green
13     Green
14     Green
15     Green
16     Green
17   Blue-Green
18     Green
19     Green
20     Green
21   Bluish-Green
22     Green
23     Green
24   Blue-Green
25   Blue-Green
26     NaN
27     Green
28     NaN
Name: Color, dtype: object
```

a single column is a new type, called Series

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

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

We can pick out rows using the `loc` accessor. It is a tricky concept because it is **indexing** so it uses square brackets `[]` but it uses a `.` like a method. This is a sort of atypical syntax, but we do not use it very often. We pick out single columns a lot, so that has a nice easy syntax like above, but this is rare, so it got the less elegant syntax.

```
coffee_df.loc[1]
```

```

Species                               Robusta
Owner                                ankole coffee producers coop
Country.of-Origin                      Uganda
Farm.Name                            kyangundu cooperative society
Lot.Number                           NaN
Mill                                 ankole coffee producers
ICO.Number                           0
Company                             ankole coffee producers coop
Altitude                            1488
Region                              sheema south western
Producer                            Ankole coffee producers coop
Number.of.Bags                      300
Bag.Weight                           60 kg
In.Country.Partner                  Uganda Coffee Development Authority
Harvest.Year                         2013
Grading.Date                         June 26th, 2014
Owner.1                             Ankole coffee producers coop
Variety                             NaN
Processing.Method                   NaN
Fragrance...Aroma                    7.83
Flavor                             8.08
Aftertaste                           7.75
Salt...Acid                          7.92
Bitter...Sweet                        8.0
Mouthfeel                           8.25
Uniform.Cup                         10.0
Clean.Cup                            10.0
Balance                             7.92
Cupper.Points                       8.0
Total.Cup.Points                     83.75
Moisture                            0.12
Category.One.Defects                0
Quakers                            0
Color                               Green
Category.Two.Defects                2
Expiration                           June 26th, 2015
Certification.Body                  Uganda Coffee Development Authority
Certification.Address               e36d0270932c3b657e96b7b0278dfd85dc0fe743
Certification.Contact              03077a1c6bac60e6f514691634a7f6eb5c85aae8
unit_of_measurement                 m
altitude_low_meters                1488.0
altitude_high_meters               1488.0
altitude_mean_meters                1488.0
Name: 1, dtype: object

```

We can also slice in dataframes, just like in strings.

```

subset_df = coffee_df.loc[5:8]
subset_df

```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company
5	Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0	katuka development trust ltd
6	Robusta	andrew hetzel	India	NaN	NaN	(self)	NaN	cafemakers, llc
7	Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers
8	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	7	sethuraman estate	14/1148/2017/18	kaapi royale

4 rows × 43 columns

Now `loc[1]` will give a key error because there is no `1` in the index.

```
subset_df.loc[1]
```

```

-----
KeyError                                         Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in 
3652     try:
-> 3653         return self._engine.get_loc(casted_key)
3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc
File /opt/hostedtoolcache/Python/3.8.18/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:2606, in pandas._libs.hashtable.Int64HashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:2630, in pandas._libs.hashtable.Int64HashTable.get_item()

KeyError: 1

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

KeyError                                         Traceback (most recent call last)
Cell In[34], line 1
----> 1 subset_df.loc[1]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexing.py:1103, in _Loc
1100     axis = self.axis or 0
1102     maybe_callable = com.apply_if_callable(key, self.obj)
-> 1103     return self._getitem_axis(maybe_callable, axis=axis)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexing.py:1343, in _Loc
1341 # fall thru to straight lookup
1342 self._validate_key(key, axis)
-> 1343     return self._get_label(key, axis=axis)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexing.py:1293, in _Loc
1291     def _get_label(self, label, axis: AxisInt):
1292         # GH#5567 this will fail if the label is not present in the axis.
-> 1293         return self.obj.xs(label, axis=axis)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/generic.py:4095, in NDFrame
4093         new_index = index[loc]
4094     else:
-> 4095         loc = index.get_loc(key)
4097         if isinstance(loc, np.ndarray):
4098             if loc.dtype == np.bool_:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655, in 
3653     return self._engine.get_loc(casted_key)
3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
3656 except TypeError:
3657     # If we have a listlike key, _check_indexing_error will raise
3658     # InvalidIndexError. Otherwise we fall through and re-raise
3659     # the TypeError.
3660     self._check_indexing_error(key)

KeyError: 1

```

the only values that will work in `loc` are the ones in the index:

```
subset_df.index
```

```
Index([5, 6, 7, 8], dtype='int64')
```

however, with `iloc` they are indexed by integer values starting with 0.

[Skip to main content](#)

```
subset_df.iloc[1]
```

```
Species                    Robusta
Owner                     andrew hetzel
Country.of-Origin          India
Farm.Name                  NaN
Lot.Number                 NaN
Mill                       (self)
ICO.Number                 NaN
Company                    cafemakers, llc
Altitude                   3000'
Region                     chikmagalur
Producer                   Sethuraman Estates
Number.of.Bags              200
Bag.Weight                 1 kg
In.Country.Partner         Specialty Coffee Association
Harvest.Year                2012
Grading.Date                February 29th, 2012
Owner.1                     Andrew Hetzel
Variety                     NaN
Processing.Method           NaN
Fragrance...Aroma            8.0
Flavor                      7.92
Aftertaste                  7.67
Salt...Acid                  8.0
Bitter...Sweet                7.75
Mouthfeel                   7.75
Uniform.Cup                  10.0
Clean.Cup                     10.0
Balance                      7.92
Copper.Points                7.75
Total.Cup.Points              82.75
Moisture                     0.0
Category.One.Defects          0
Quakers                      0
Color                        Green
Category.Two.Defects          0
Expiration                   February 28th, 2013
Certification.Body           Specialty Coffee Association
Certification.Address        ff7c18ad303d4b603ac3f8cff7e611ffc735e720
Certification.Contact        352d0cf7f3e9be14dad7df644ad65efc27605ae2
unit_of_measurement             m
altitude_low_meters           3000.0
altitude_high_meters           3000.0
altitude_mean_meters           3000.0
Name: 6, dtype: object
```

## 2.9. Questions After Class

2.9.1. I think this something I need to figure out how do the localhost or just utilizing a url in VS Code? I was late to the class, I never got how to do the jupyter lab thing.

For this class, you need to use jupyter notebooks without extraneous metadata. If you use jupyter inside of vs code, it adds extraneous metadata that makes it hard to grade and VS code, in my experience, does not provide the most helpful autocomplete for Data Science.

Please see office hours to get help with it.

## 2.9.2. Is the Python we use in Jupyter lab notebooks any different from traditional Python?

the Python is mostly all the same. There are different python interpreters that have some slightly different behaviors, but mostly only in the display. As a matter of technicality, jupyter uses the `ipython` python as the kernel.

## 2.9.3. Does index just list all the rows?

the index is the *name* of the rows the same way that the column headers are the *name* of the columns.

## 2.9.4. How you copied the file url from github.

Click the `raw button` and then copy the URL from your browser's url bar.

## 2.9.5. How did we change the index?

we changed the index from the inferred (figured out by pandas) `RangeIndex` to a column of the data by adding the `index_col=0` parameter to our `read_csv` call.

## 2.9.6. I would like to learn more about the panda commands

We will continue learning more pandas features for the next few weeks.

## 2.9.7. are we gonna have to use what we learned today in a bigger program in the future

Yes, these features we used today are the basis of all of the data analysis we will do all semester. However, we will not be writing "programs" the way you may have for other classes, we will be doing data analyses, which are more narrative.

## 2.9.8. How can I use Jupyter to clean data?

jupyter is a way to work with python code. We will learn what clean data looks like and more ways to manipulate dataframes to make it clean in two weeks.

## 2.9.9. Why is taking data from columns much more common than taking it from rows?

We set our data up so that each column is a variable. We often want to treat different variables differently, but do the same thing to all of the rows.

## 2.9.10. I was wondering more about the Index variable type and was also curious as to what that could be used for.

the `Index` type from `pandas` is a component of a `DataFrame`, we will use them implicitly whenever we work with a part of a dataframe and explicitly when we clean data.

## 2.9.11. I know by convention we use the typical alias for importing libraries, but is it okay to use our own alias for our own private programs?

using nonstandard aliases is a bad habit to develop and I cannot endorse it. Technically the code will run, but in class it will be a style violation.

## 2.9.12. does the panda's data start indexing at 1 because 0 is where the table headers are located?

Indexing using `loc` started at 1 because the dataset had 1 there, in the second example it started at 5.

using `iloc` starts at 0.

## 2.9.13. In the dataset we loaded, I noticed that there were some zeros, which are nulls, I'm guessing we have to clean those out, and I was wondering how?

zeros are a value, nulls are encoded in different ways. We will learn how to deal with missing values in two weeks.

## 2.9.14. How often do data sets need to be cleaned/manipulated before proper analysis can be done?

Real data, will almost always need to be fixed a little bit.

## 2.9.15. how does being able to view specific rows/columns help us make conclusions about data?

For example, maybe one column is the thing you are interested in, you may want to know on stats on the one column.

## 2.9.16. Are assignments always a certain level or can one assignment be done to be a level 1 or a level 2 assignment

Going forward, Assignments are always targeted at level 2. In class prisma questions will assess at level 1. An incomplete attempt at an assignment might be evaluated only at level 1, so that can be a way to make up for missed class and then you earn the level 2 in the next assignment that assess that skill.

## 2.9.17. will I be able to use data from an existing lab that I work in for certain assignment?

Yes, as long as the data is allowed to be shared. Please confirm with your PI.

## 2.9.18. When we submit to GitHub, do we need to do anything other than upload the file?

For your portfolio, no. For other assignments, there will be a step to do, and there will be instructions in the assignment.

## 2.9.19. Are we going to have to create our own datasets for any future assignments rather than downloading datasets from the Internet?

Assignment 2 you will build a dataset about datasets, but other than that you will mostly use datasets that you find online or that you have for another purpose.

## 2.9.20. How to better navigate github and not second guess my posts there

### ⚠ Warning

this will be added later.

# 3. DataFrames from other sources

Today we will:

- continue examining the dataframe object
- see more ways to load data
- make sure you are set up for assignment 2

## 3.1. Indexing review

```
topics = ['what is data science', 'jupyter', 'conditional','functions', 'lists', 'dictionaries','pandas' ]  
topics[-1]
```

```
'pandas'
```

negative numbers count from the right

## 3.2. Reserve words

these are words you do not want to use for variable names

Python reserve words turn green:

```
print
```

```
<function print>
```

```
def
```

```
Cell In[3], line 1
  def
    ^
SyntaxError: invalid syntax
```

### 3.3. Built in iterable types

These are four different iterable constructions:

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

We can see their types

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

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

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

```
b
```

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

```
a
```

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

Where we index lists with numbers

```
a[0]
```

```
'a'
```

we can access items, the values in a dictionary using square brackets and the keys

```
b['b']
```

```
1
```

## 3.4. Building iterables quick with Comprehensions

- list comprehensions are **super** handy

we can make a list using a loop all in one line. The constructions above for `a` and `b` are called list and dictionary **comprehensions**. It is equivalent to using a loop, but a more concise way to build a list with a loop.

```
a_long = []
for char in 'abcde':
    a_long.append(char)
```

Notice that even in this for loop the `lopo` variable is a conceptually meaningful variable and we iterate over the items in an iterable type object. This is in contrast to creating a loop variable that is an integer. This loop style is considered good pythonic strategy.

For more detail, see the Python docs [section on looping strategies](#)

```
a_long
```

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



### Hint

**Programming is a practice** the goal is not to memorize everything, but be exposed to enough that you remember what you can look up later

`enumerate` is a built in function that allows you to get both a number and an item for your use in a loop or comprehension. You can read the help below or the [technical details in the official Python Docs](#)

```
help(enumerate)
```

```
Help on class enumerate in module builtins:

class enumerate(object)
| enumerate(iterable, start=0)
|
| Return an enumerate object.
|
| iterable
|     an object supporting iteration
|
| The enumerate object yields pairs containing a count (from start, which
| defaults to zero) and a value yielded by the iterable argument.
|
| enumerate is useful for obtaining an indexed list:
|     (0, seq[0]), (1, seq[1]), (2, seq[2]), ...
|
| Methods defined here:
|
| __getattribute__(self, name, /)
|     Return getattr(self, name).
|
| __iter__(self, /)
|     Implement iter(self).
|
| __next__(self, /)
|     Implement next(self).
|
| __reduce__(...)
|     Return state information for pickling.
|
| -----
| Static methods defined here:
|
| __new__(*args, **kwargs) from builtins.type
|     Create and return a new object. See help(type) for accurate signature.
```

## 3.5. Read DataFrames from HTML

let's use `read_html` on the course communications page and then inspect what we get to figure it out

```
course_comms_url = 'https://rhodyprog4ds.github.io/BrownFall23/syllabus/communication.html'
```

we will first need out library

```
import pandas as pd
```

then we will read it in without saving it and look at the output to see what it looks like.

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

```
[ Day      Time    Location     Host
0 Monday   12pm-2pm   Zoom     Mark
1 Monday    4-5pm     Zoom  Dr. Brown
2 Friday    4-5pm  134 Tyler  Dr. Brown,
               usage platform \
               in class prismia
               any time prismia
               private questions to your assignment github
               for general questions that can help others github
               to share resources or ask general questions in... github
               matters that don't fit into another category e-mail

               area \
               chat
               download transcript
               issue on assignment repo
               issue on course website
               discussion on community repo
               to brownsarahm@uri.edu

               note
0 outside of class time this is not monitored cl...
1 use after class to get preliminary notes eg if...
2                         eg bugs in your code"
3 eg what the instructions of an assignment mean...
4                         include links in your portfolio
5 remember to include `[CSC310]` or `[DSP310]` (... )
```

now we will save it to a variable for future use.

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

we can check the type, it is a list as we noted from looking at the output.

```
type(comm_df_list)
```

```
list
```

and each item in the list is a DataFrame

```
type(comm_df_list[0])
```

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

DataFrames also have a `shape` attribute, to tell us the number of rows and columns.

```
comm_df_list[0].shape
```

```
(3, 4)
```

```
achievements_url = 'https://rhodyprog4ds.github.io/BrownFall23/syllabus/achievements.html'
```

This is a good job for a list comprehension.

```
shape_list_comp = [df.shape for df in pd.read_html(achievements_url)]  
shape_list_comp
```

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

Again, we can write this out as a for loop with append, but the comprehension is more concise.

```
shape_list = []  
for df in pd.read_html(achievements_url):  
    shape_list.append(df.shape)
```

in the comprehension structure the `[]` are what make it a list, they make anything a list

```
type([1,2,3])
```

```
list
```

## 3.6. More DataFrame Indexing

we'll go back to our coffee data

```
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)
```

See again our shape

```
coffee_df.shape
```

```
(28, 43)
```

and the first few rows

```
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

[Skip to main content](#)

we can also see a random sample, not only the head and tail

```
coffee_df.sample(3)
```

	Species	Owner	Country.of-Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Alt
28	Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaN	cafe politico	
2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	
14	Robusta	kasozi coffee farmers association	Uganda	kasozi coffee farmers	NaN	NaN	0	kasozi coffee farmers association	

3 rows × 43 columns

### 💡 Hint

printing out the list of columns is a helpful way to get them to copy-paste for later selection to ensure no typos. In a polished notebook, you could then delete a cell like the one below, but it's really helpful while you are working

```
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')
```

We can subset columns by passing a list of multiple columns to use for indexing

```
columns_of_interest = ['Owner', 'Country.of-Origin']
coffee_df[columns_of_interest].head(1)
```

	Owner	Country.of-Origin
1	ankole coffee producers coop	Uganda

it has to be a list though, if we put them in one set of square brackets, it is a tuple and we get a `KeyError` because it looks for **one** column that has the name `'Owner', 'Country.of-Origin'`

```
coffee_df['Owner', 'Country.of-Origin']
```

[Skip to main content](#)

```

-----
KeyError Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in 
3652     try:
-> 3653         return self._engine.get_loc(casted_key)
3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc
File /opt/hostedtoolcache/Python/3.8.18/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: ('Owner', 'Country.of.Origin')

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

KeyError Traceback (most recent call last)
Cell In[31], line 1
----> 1 coffee_df['Owner', 'Country.of.Origin']

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame.__getitem__
3759     if self.columns.nlevels > 1:
3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
3762     if is_integer(indexer):
3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655, in 
3653     return self._engine.get_loc(casted_key)
3654     except KeyError as err:
-> 3655     raise KeyError(key) from err
3656     except TypeError:
3657         # If we have a listlike key, _check_indexing_error will raise
3658         # InvalidIndexError. Otherwise we fall through and re-raise
3659         # the TypeError.
3660         self._check_indexing_error(key)

KeyError: ('Owner', 'Country.of.Origin')

```

instead we can use 2 sets col square brackets if we do not want a separate variable

```
coffee_df[['Owner', 'Country.of.Origin']].head(1)
```

	Owner	Country.of.Origin
1	ankole coffee producers coop	Uganda

## 3.7. Subsetting by values

We can do boolean operators on a `pandas.Series` and it will do it automatically to every element

```
is_green = coffee_df['Color'] == 'Green'
is_green
```

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

then we can look at the shape and see that it is the same shape as the column we selected.

```
is_green.shape, coffee_df['Color'].shape
```

```
((28,), (28,))
```

now we can use that to subset the rows

```
green_coffee_df = coffee_df[is_green]
green_coffee_df.head()
```

	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	1
3	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	10
4	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1
5	Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0	katuka development trust ltd	1
6	Robusta	andrew hetzel	India	NaN	NaN	(self)	NaN	cafemakers, llc	3

5 rows × 43 columns

and look at the shape to see

```
green_coffee_df.shape
```

```
(20, 43)
```

```
is_green.sum()
```

```
20
```

```
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')
```

## 3.8. Python has no switch

we use dictionaries in those kind of cases

[Skip to main content](#)

```
score_text = {False:'low',
              True:'high'}
```

Here we can switch from a list of true/false to high low

this gives true/false values for if the flavor is above or below 7

```
coffee_df['Flavor']>=7
```

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

and this is high/low instead

```
[score_text[flavor_comp] for flavor_comp in coffee_df['Flavor']>=7]
```

```
['high',
 'high',
 'low',
 'low']
```

## 3.9. Keeping a clean notebook

we can put code in a python file and include it in our notebooks to use it

this can be useful if:

- you have a long hard to read thing that distracts from your other analysis
- you have a function you want to reuse a lot
- (unlikely in class) you need to make your own library!

I created a separate file called `example.py` and defined a variable in it like:

```
name = 'sarah'
```

now I can import that and use it.

```
from example import name
```

```
name
```

```
'sarah'
```

## 3.10. Additional hints

[Skip to main content](#)

- using any pandas method is okay, including some we have not seen if it is a single method, for example the [select\\_dtypes](#) method [docs](#)
- Your task is partially to learn other IO methods, so the [pandas docs](#) IO page is a good resource

## 3.11. Questions After Class

### ! Important

some questions are not answered below because they are explained in the notes above or they are too vague, you can come to office hours if you have a question that is not here or post a more detailed question on this repo or your assignment

### 3.11.1. what is pandas?

it is a Python library. Read more at the [user guide](#)

### 3.11.2. My question is how is the data frame being accessed from the url and how can I understand it more clearly?

[pd.read\\_](#) functions can do web requests and read data online and load it directly into memory. To understand in greater detail, I recommend the [docs](#) and then follow through the links through there to the level of depth that you want.

### 3.11.3. what does the shape of a dataframe do?

It is just information that we can do

### 3.11.4. Why do we need dictionaries to create new rows in the dataframes rather than operators?

We did not use the dictionary to create new rows, we used it to map values to other values. We will see this pattern throughout the course.

### 3.11.5. how to figure out which dataframes from html are useful

we have to look at them.

### 3.11.6. How to download datasets

For your assignment, you can load directly with a URL

### 3.11.7. Is the sum() method only counting true values, and if so, is it simply treating them as 1?

```
int(True), int(False)
```

```
(1, 0)
```

### 3.11.8. Why does the thing that happens right before a for in loop apply to all of the values? I think I know but just to be sure

in a list comprehension the part before the for is like the loop body, see above where I defined `a_long` and compare it to the definition of `a`

### 3.11.9. I would like to learn more about dictionaries

I recommend starting in the python language docs section on [dictionaries](#) they are a very powerful structure and the text there is technical, but there are plenty of links. It is really good practice to get good at parsing through technical docs like this.

## 4. Exploratory Data Analysis

Now we get to start actual data science!

### 4.1. First, a note on assignments

#### 4.1.1. the goal

- I am not looking for “an answer”
- You should not be either
- I am looking for evidence that you **understand** the material (thus far including prereqs)
- You should be trying to **understand** material

This means that in office hours, I am going to:

- ask you questions to help you think about the problem and the material
- help direct your attention to the right part of the error message to figure out what is wrong

#### 4.1.2. Getting help

Sending me a screenshot is almost guaranteed to *not* get you a help. Not because I do not want to, but because I literally do not have the information to get you an answer.

Typically when someone do not know how to fix something from the error message, it is because they are reading the wrong part of the error message or looking at the wrong part of the code trying to find the problem.

This means they end up screenshotting that wrong thing, so I literally **cannot** tell what is wrong from the screenshot.

I am not being stubborn, I just literally cannot tell what is wrong because I do not have enough information. Debugging code requires context, if you deprive me that, then I cannot help.

To get asynchronous help:

- upload your whole notebook, errors and all
- create an issue on that repo

## 4.2. This week: Exploratory Data Analysis

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

### 4.2.1. 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

```
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)
```

```
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

## 4.3. Describing a Dataset

[Skip to main content](#)

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`

```
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

```
coffee_df.describe()
```

[Skip to main content](#)

	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.

### 4.3.1. Understanding Quantiles

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

### 4.3.2. Individual variable

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

```
coffee_df['Balance'].describe()
```

```
-----  
AttributeError                                     Traceback (most recent call last)  
/tmp/ipykernel_2007/957071290.py in ?()  
----> 1 coffee_df['Balance'].descirbe()  
  
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/generic.py in ?(self, name)  
 5985         and name not in self._accessors  
 5986         and self._info_axis._can_hold_identifiers_and_holds_name(name)  
 5987     ):  
 5988         return self[name]  
-> 5989     return object.__getattribute__(self, name)  
  
AttributeError: 'Series' object has no attribute 'descirbe'
```

This is an `AttributeError` because there is no method or property of a `pd.Series` that is named `'descirbe'` that tells me it is a typo.

```
coffee_df['Balance'].describe()
```

```
count    28.000000  
mean     7.541786  
std      0.526076  
min      5.250000  
25%     7.500000  
50%     7.670000  
75%     7.830000  
max      8.000000  
Name: Balance, dtype: float64
```

## 4.4. Individual statistics

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

```
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
```

```
Number.of.Bags      1.00
Harvest.Year       2012.00
Fragrance...Aroma   6.75
Flavor             6.67
Aftertaste          6.50
Salt...Acid         6.83
Bitter...Sweet      6.67
Mouthfeel           5.08
Uniform.Cup         9.33
Clean.Cup           9.33
Balance             5.25
Copper.Points       6.92
Total.Cup.Points    73.75
Moisture            0.00
Category.One.Defects 0.00
Quakers             0.00
Category.Two.Defects 0.00
altitude_low_meters 40.00
altitude_high_meters 40.00
altitude_mean_meters 40.00
dtype: float64
```

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

```
7.83
```

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

```
7.559642857142856
```

## 4.5. 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')
```

So far the stats above are only for numerical features

[Skip to main content](#)

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`

What is the most common country of origin?

```
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` method 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`

[Skip to main content](#)

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

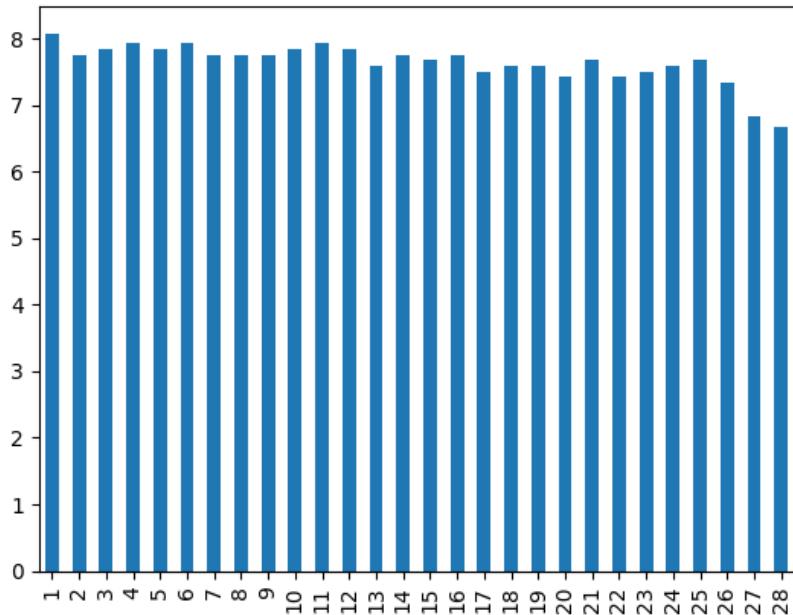
```
'India'
```

## 4.6. Which country scores highest on flavor?

Let's try to answer this with plots first

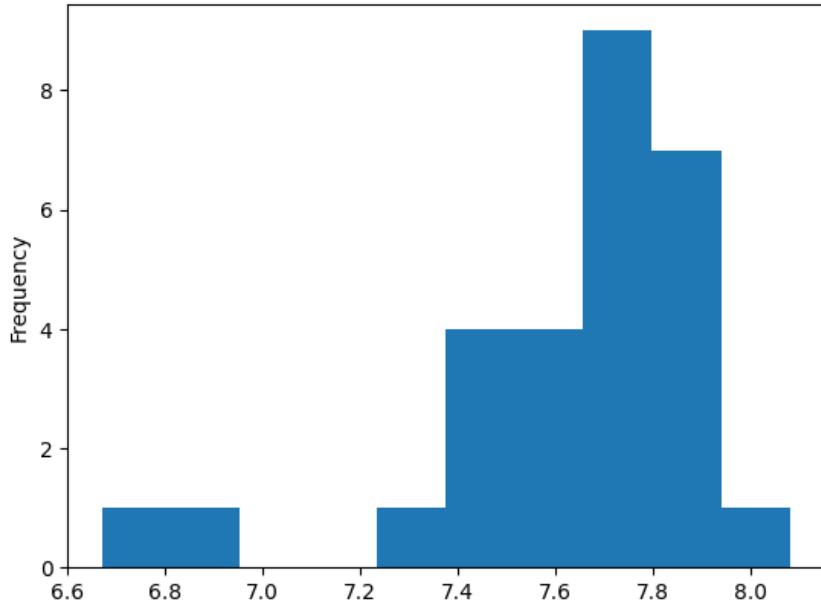
```
coffee_df['Flavor'].plot(kind='bar')
```

```
<Axes: >
```



```
coffee_df['Flavor'].plot(kind='hist')
```

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



Seaborn give us *opinionated defaults*

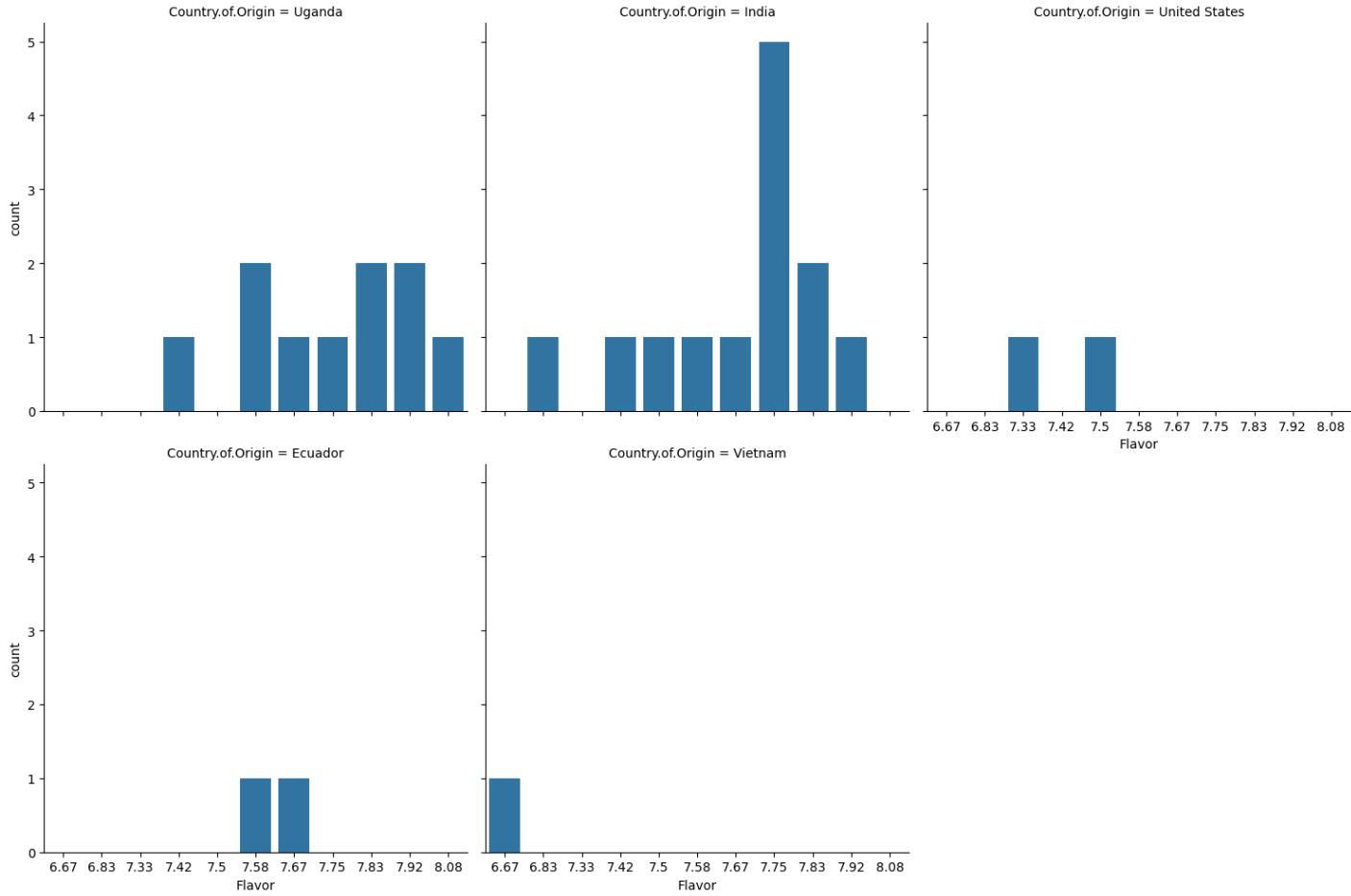
```
import seaborn as sns
```

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

So we can choose a type of plot and give it the whole dataframe and pass the variables to different parameters for them to be used in different ways.

```
sns.catplot(data =coffee_df, x='Flavor', kind ='count', col='Country.of.Origin',  
            col_wrap=3)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9acd0b4f10>
```



4.6.1. Are any coffees more than one standard deviation above the average in `Flavor`?

```
coffee_df['Flavor'].mean() + coffee_df['Flavor'].std(), coffee_df['Flavor'].max()
```

```
(7.93437014076549, 8.08)
```

Yes, one is more than one standard deviation from the mean.

## 4.7. Questions after class

4.7.1. can pycharm access urls?

Probably, it should not block them and they are language features, but I do not use it to know.

4.7.2. can we do what we did today in pycharm?

[Skip to main content](#)

For this class, you need to submit notebook files, so pycharm will not work.

Pycharm is not typically used in datascience.

#### 4.7.3. Are the bars on the Seaborn plot random colors, or do the colors have some meaning?

here, each different value is a different color. We wil control it more on Thursday

#### 4.7.4. what types of jobs use what we learned today

Any data science role would use these skills daily.

#### 4.7.5. Is there an equivalent for the index of the minimum value, such as idxmin()?

#### 4.7.6. What is the std

standard deviation [pandas docs on calculation](#). The [wikipedia](#) article on standard deviation is a good source on this.

#### 4.7.7. Why can we not use matplotlib?

You *can* and you may for some customizaiton, but seaborn usually makes things faster to code and easier to read. I will not teach much matplotlib, but if you know it, up to date, not outdated, you may use it in some cases.

#### 4.7.8. One question I have is what exactly the series object can be used for

A series is one column or one row of a dataframe.

#### 4.7.9. how these lines of code can be used more efficiently in larger data sets?

For extremely large datasets if it's too much to plot, you might, for exaple, sample the dataset to only plot a subset.

Otherwise this will still work.

#### 4.7.10. Can you change what the plot graphs rather than an index vs frequency?

yes, we can change the plot to whatever we want.

#### 4.7.11. actually I would like to know like with our country/flavor example can they be put in the same plot but with different colors for the different countries to each country side by side for a given score

yes

## 4.7.12. Can assignment 3 can be release so we can pre-read it over.

Yes, it is posted now

## 4.8. Questions we'll answer in class on thursday

### ⚠ Important

These are **great** questions, I'm just not going to answer them here and then again in class on Thursday

- What other types of graphs could be used?
- Is there a way to use both x and y in sns without creating multiple charts?
- Using matplotlib, we never got to use it to class, and I want to know the difference between that and seaborn
- How to change the colors of the plots.
- What do we usually tend to look for in terms of patterns when trying to formulate questions about the data, or several pieces of data?

## 5. Visualization

### ⚠ Warning

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.

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

## 5.1. 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, index_col=0)
robusta_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

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

[Skip to main content](#)

```
cols_to_compare = ['Mouthfeel', 'Aftertaste']
robusta_df[cols_to_compare].std()
```

```
Mouthfeel      0.725152
Aftertaste     0.342469
dtype: float64
```

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

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

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

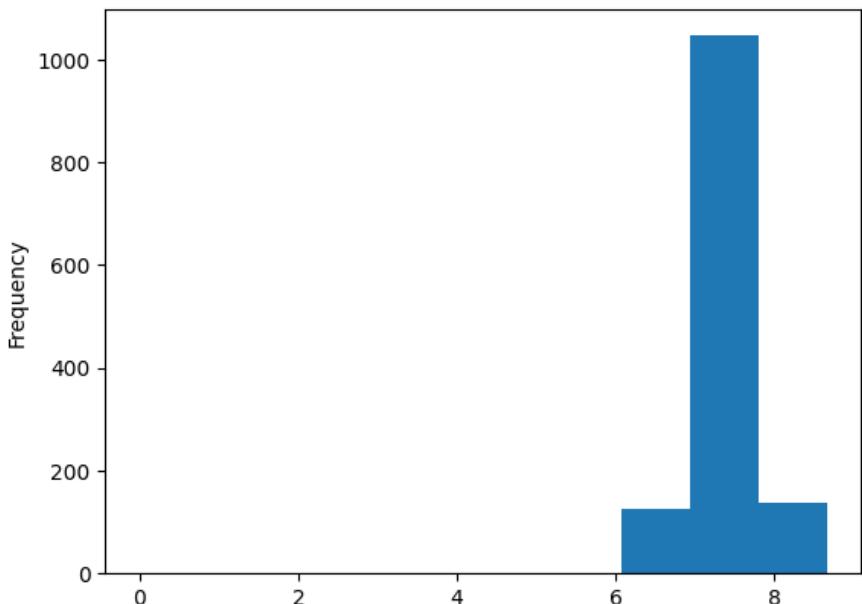
	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	Region
1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural development plc	1950- 2200	gu hambe

1 rows × 43 columns

Recall, we can use built in plots in pandas.

```
coffee_df['Aftertaste'].plot(kind='hist')
```

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



[Skip to main content](#)

## 5.2. 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.

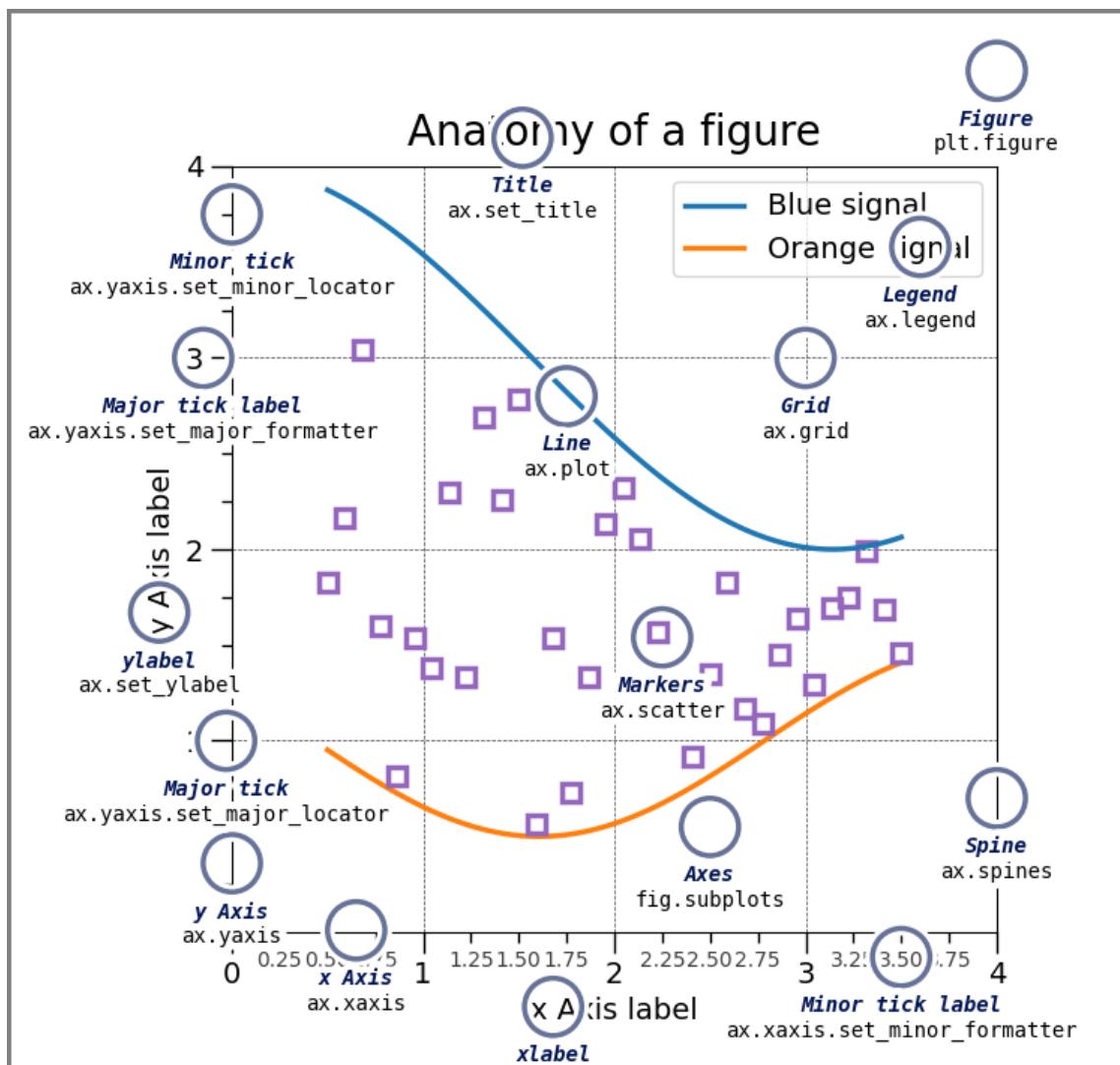
There are lots of type of plots, we saw the basic patterns of how to use them and we've used a few types, but we cannot (and do not need to) go through every single type. There are general patterns that you can use that will help you think about what type of plot you might want and help you understand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

### 5.2.1. 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.



Skip to main content

Thi

Learn

3 for

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

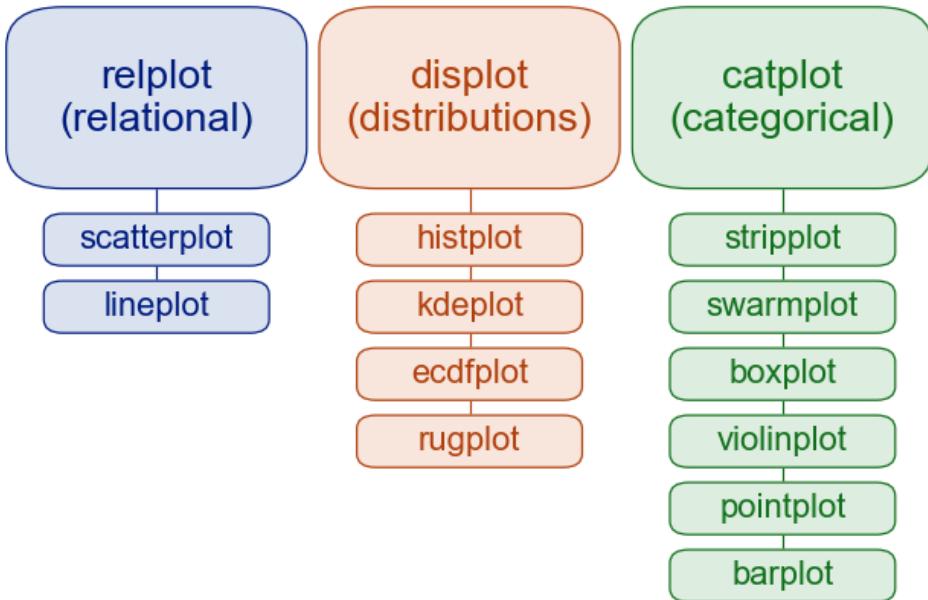
### 💡 Try it Yourself

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.

## 5.2.2. 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 thie 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.

## 5.2.3. More

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

## 5.2.4. Styling in Seaborn

### ❗ Important

This was not covered in class, but can be helpful

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

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

more on colors

```
robusta_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
```

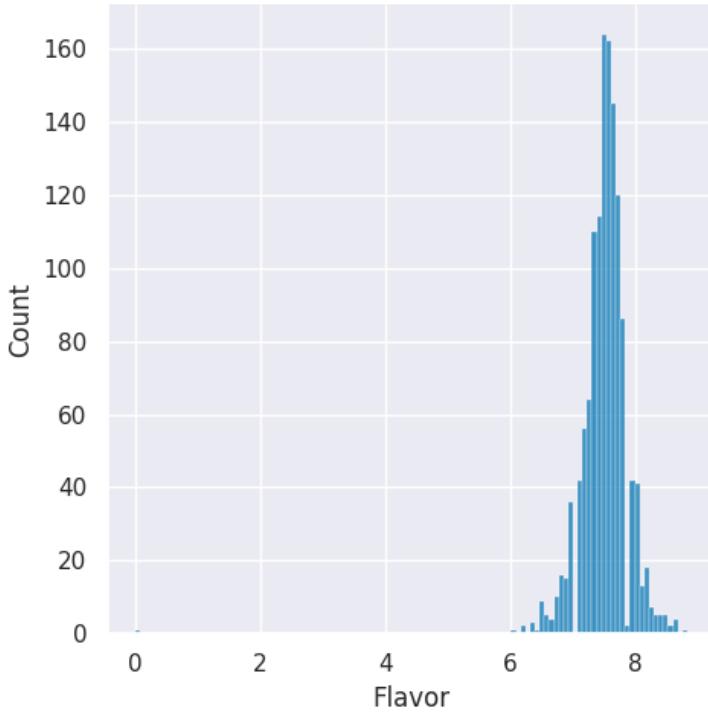
```
Index(['Species', 'Owner', 'Country.of-Origin', 'Farm.Name', 'Lot.Number',
       'Mill', 'ICO.Number', 'Company', 'Altitude', 'Region', 'Producer',
       'Number.of.Bags', 'Bag.Weight', 'In.Country.Partner', 'Harvest.Year',
       'Grading.Date', 'Owner.1', 'Variety', 'Processing.Method', 'Aroma',
       'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance', 'Uniformity',
       'Clean.Cup', 'Sweetness', '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')
```

## ! 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 paramters to modify plots visually. I **strongly recommend reading it often**. I recommend reading their tutorial too

```
sns.displot(data = coffee_df, x='Flavor')
```

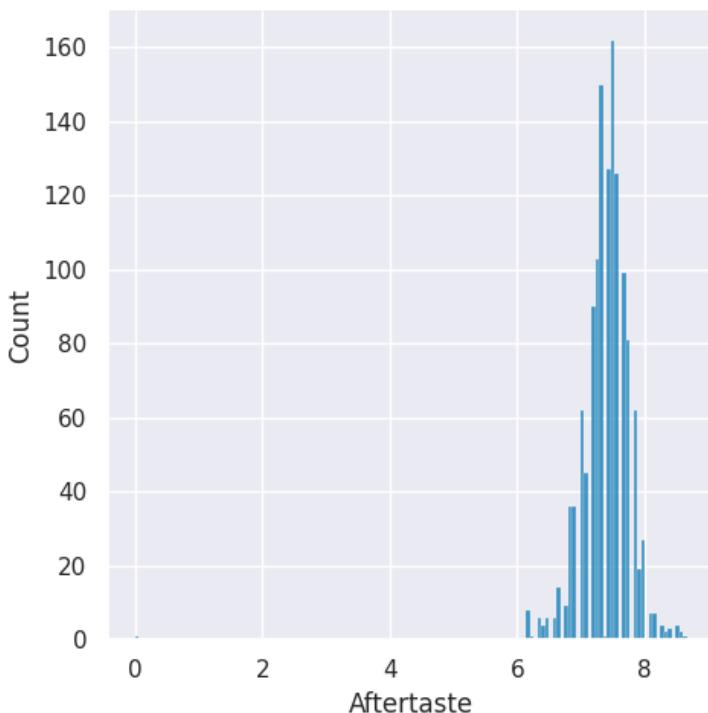
```
<seaborn.axisgrid.FacetGrid at 0x7f1598501a30>
```



Note explain the layout warning

```
sns.displot(data = coffee_df, x='Aftertaste',)
```

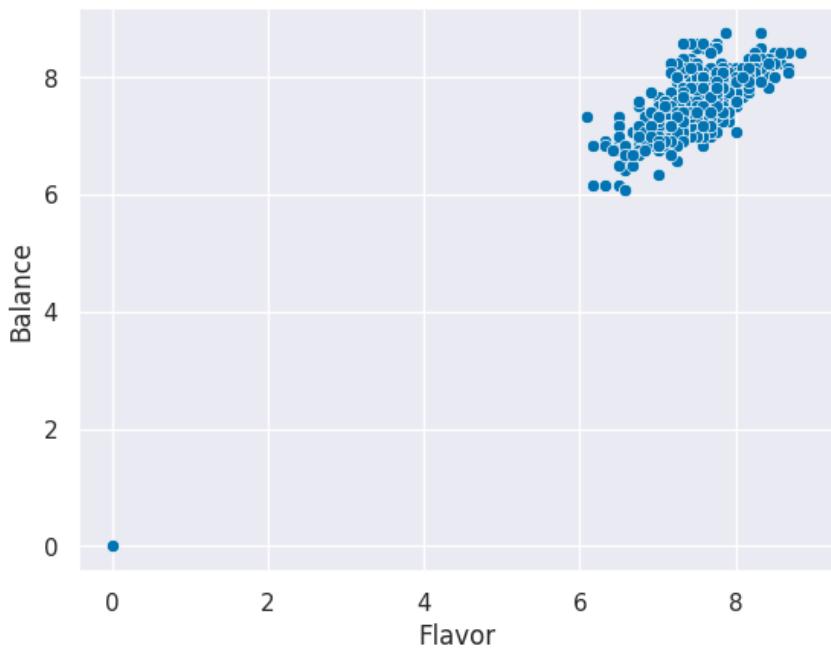
```
<seaborn.axisgrid.FacetGrid at 0x7f155a133c40>
```



[Skip to main content](#)

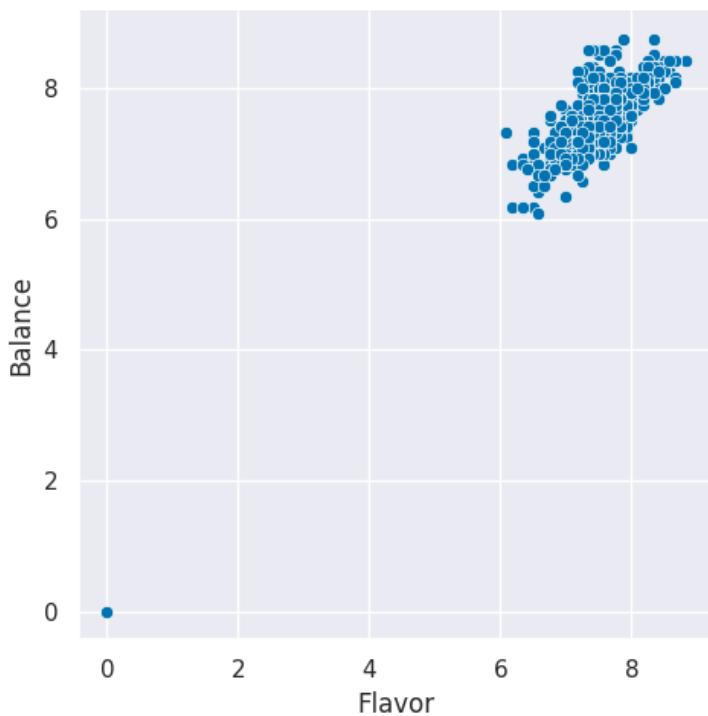
```
sns.scatterplot(data=coffee_df, x='Flavor', y='Balance')
```

```
<Axes: xlabel='Flavor', ylabel='Balance'>
```



```
sns.relplot(data=coffee_df, x='Flavor', y='Balance',)
```

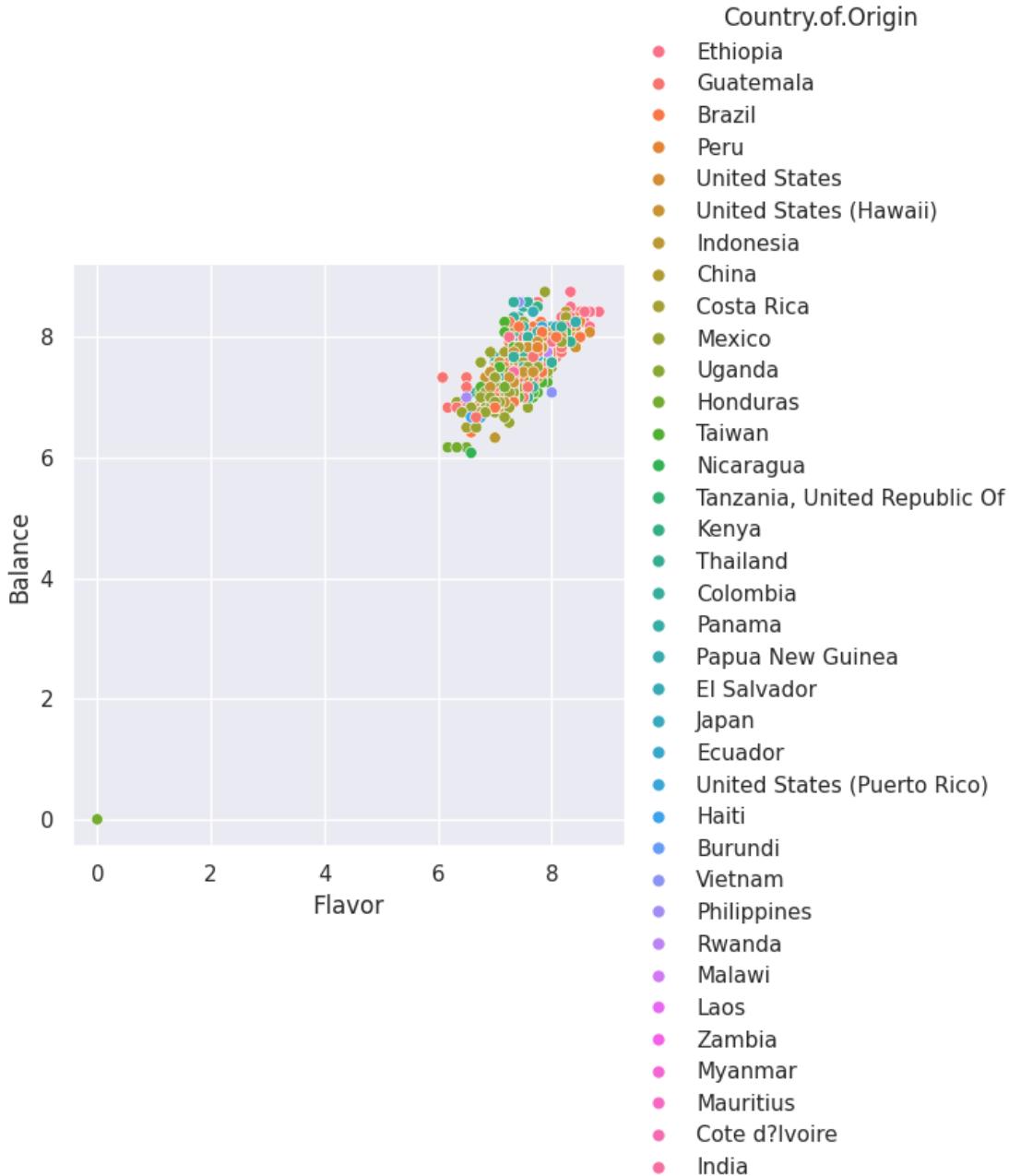
```
<seaborn.axisgrid.FacetGrid at 0x7f1559df2ee0>
```



Display functions return an object that can be passed to customize the plots further

[Skip to main content](#)

```
g = sns.relplot(data=coffee_df, x='Flavor', y='Balance',  
                 hue='Country.of.Origin',)
```



```
type(g)
```

```
seaborn.axisgrid.FacetGrid
```

```
g. # tab can show the options for attributes andmethods on this object
```

```
Cell In[16], line 1
    g. # tab can show the options for attributes andmethods on this object
    ^
SyntaxError: invalid syntax
```

## 5.3. Bags per country

How many bags of coffee are produced per country?

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

```
-----
ValueError                                Traceback (most recent call last)
cell In[17], line 1
----> 1 sns.catplot(data=coffee_df, x='Country.of.Origin',y='Number.of.Bags',
      2         kind='count');

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/seaborn/categorical.py:2764, in catplot
2762     y = 1
2763     elif x is not None and y is not None:
-> 2764         raise ValueError("Cannot pass values for both `x` and `y` .")
2766 p = Plotter(
2767     data=data,
2768     variables=dict(x=x, y=y, hue=hue, row=row, col=col, units=units),
(...),
2774     legend=legend,
2775 )
2777 for var in ["row", "col"]:
2778     # Handle faceting variables that lack name information

ValueError: Cannot pass values for both `x` and `y`.
```

```
coffee_df.shape
```

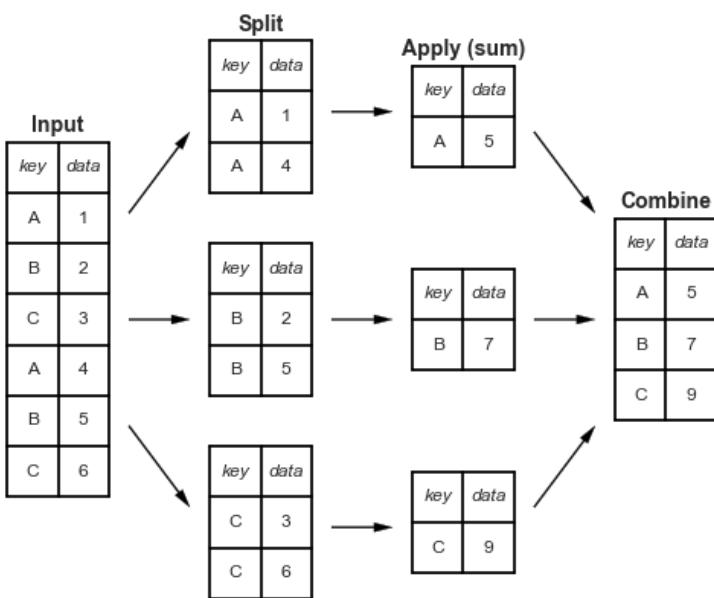
```
(1311, 43)
```

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

```

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

```



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

[Skip to main content](#)

```
country_grouped
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f1559a16a30>
```

```
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
<b>Brazil</b>	30534
<b>Burundi</b>	520
<b>China</b>	55
<b>Colombia</b>	41204
<b>Costa Rica</b>	10354
<b>Cote d'Ivoire</b>	2
<b>Ecuador</b>	1
<b>El Salvador</b>	4449
<b>Ethiopia</b>	11761
<b>Guatemala</b>	36868
<b>Haiti</b>	390
<b>Honduras</b>	13167
<b>India</b>	20
<b>Indonesia</b>	1658
<b>Japan</b>	20
<b>Kenya</b>	3971
<b>Laos</b>	81
<b>Malawi</b>	557
<b>Mauritius</b>	1
<b>Mexico</b>	24140
<b>Myanmar</b>	10
<b>Nicaragua</b>	6406
<b>Panama</b>	537
<b>Papua New Guinea</b>	7
<b>Peru</b>	2336
<b>Philippines</b>	259
<b>Rwanda</b>	150
<b>Taiwan</b>	1914
<b>Tanzania, United Republic Of</b>	3760
<b>Thailand</b>	1310
<b>Uganda</b>	3868
<b>United States</b>	361
<b>United States (Hawaii)</b>	833
<b>United States (Puerto Rico)</b>	71

```
'a b'.split(' ')
```

```
['a', 'b']
```

```
a,b = 'a b'.split(' ')
```

```
a
```

```
'a'
```

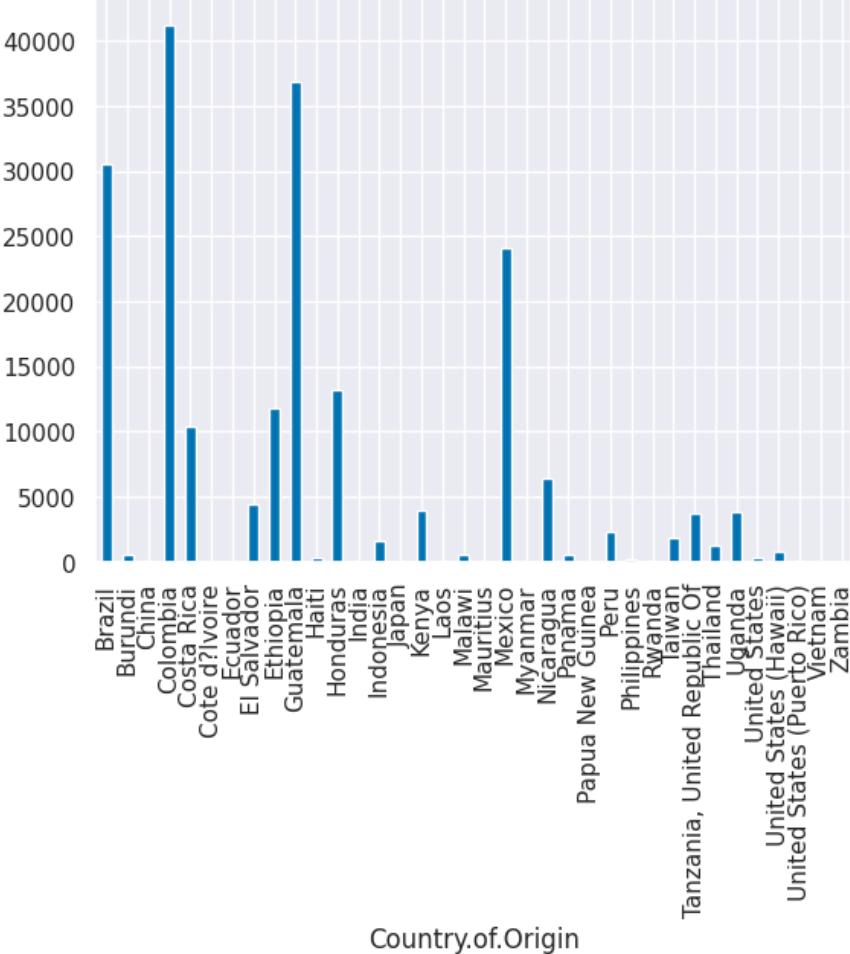
```
b
```

```
'b'
```

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

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

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



```
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', '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')
```

```
flavor_by_color = coffee_df.groupby('Color')['Flavor']
flavor_by_color.mean()
```

```
Color
Blue-Green      7.577317
Bluish-Green    7.581518
Green           7.491482
Name: Flavor, dtype: float64
```

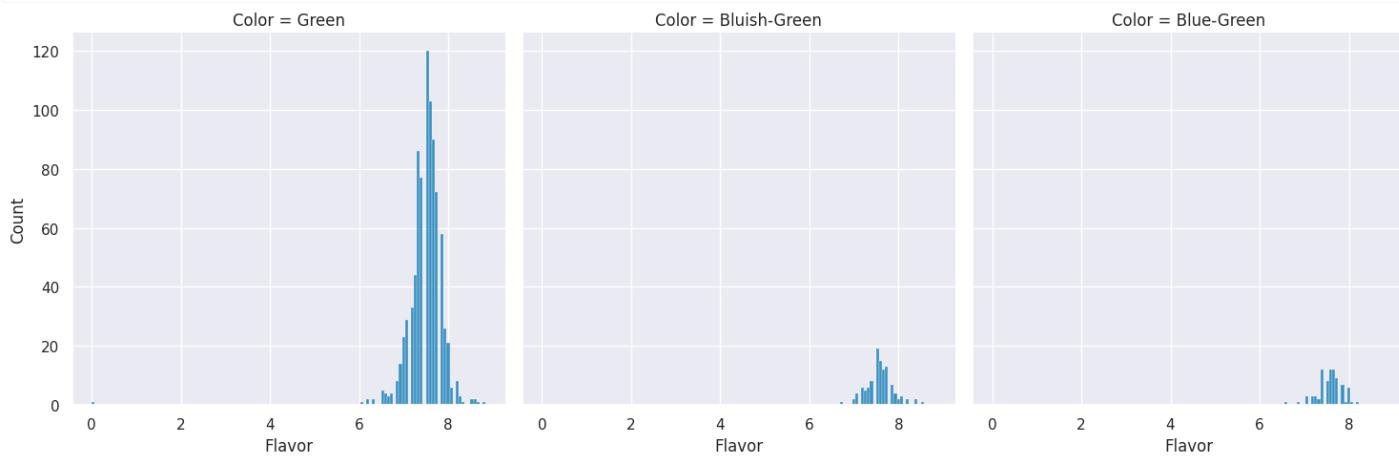
[Skip to main content](#)

```
flavor_by_color.std()
```

```
Color
Blue-Green      0.276513
Bluish-Green    0.301241
Green           0.413324
Name: Flavor, dtype: float64
```

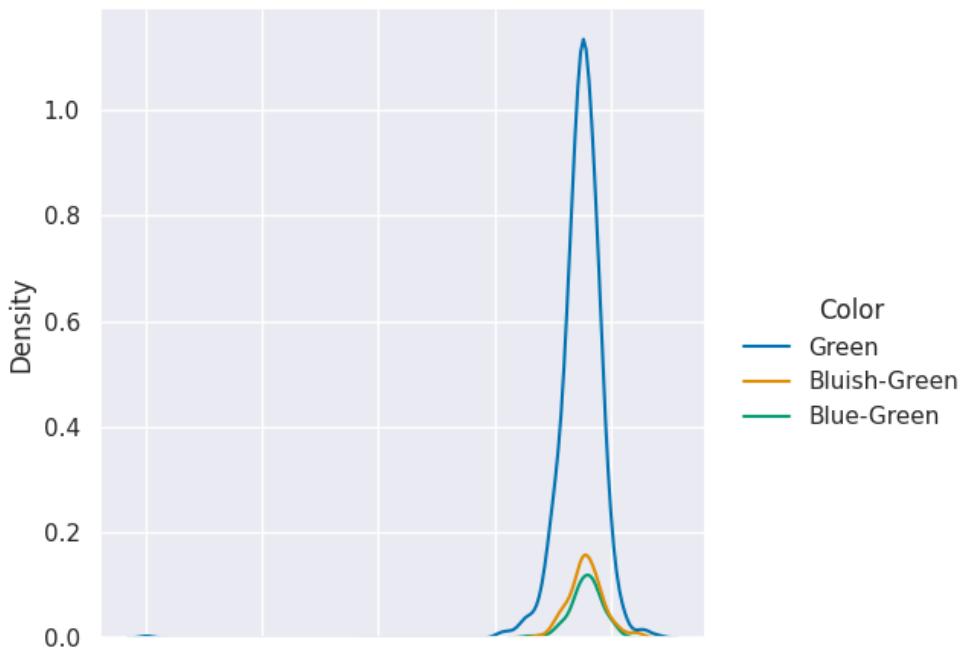
```
sns.displot(data=coffee_df, x='Flavor', col='Color' )
```

```
<seaborn.axisgrid.FacetGrid at 0x7f1555d02760>
```



```
sns.displot(data=coffee_df, x='Flavor', hue='Color' , kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f1551fb0610>
```



[Skip to main content](#)

## 5.4. Filtering a DataFrame

Now, we'll take just the country names out

How does color vary by country of origin for the top 10 countries with the most ratings?

```
df_country = coffee_df['Country.of.Origin'].value_counts()  
top_countries = df_country[:10].index  
top_countries
```

```
Index(['Mexico', 'Colombia', 'Guatemala', 'Brazil', 'Taiwan',  
       'United States (Hawaii)', 'Honduras', 'Costa Rica', 'Ethiopia',  
       'Tanzania, United Republic Of'],  
      dtype='object', name='Country.of.Origin')
```

### 5.4.1. Filtering with a lambda and the `apply` method?

```
check_top_country = lambda r: r['Country.of.Origin'] in top_countries
```

```
'Mexico' in top_countries
```

```
True
```

```
'USA' in top_countries
```

```
False
```

```
top_country_df = coffee_df[coffee_df.apply(check_top_country, axis=1)]
```

### 5.4.2. Filtering with `isin`

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)
```

```
1      True
2      True
3      True
4      True
5      True
...
1307   True
1308   False
1309   False
1310   True
1312   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,  
 False,  
 True,  
 True,  
 True,  
 True,  
 False,  
 False,  
 True,  
 True,  
 False,  
 False,  
 True,  
 False,  
 True,  
 True,  
 False,  
 True,  
 True,  
 True,  
 True,  
 False,  
 True,  
 True,  
 False,  
 True,  
 True,  
 True,  
 True,  
 True,  
 True,  
 False,  
 False,  
 False,  
 False,  
 True,  
 False,  
 False,  
 True,  
 False,  
 True,  
 False,  
 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.

```
top_coffee_df = coffee_df[coffee_df['Country.of.Origin'].isin(top_countries)]
top_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	agricultural developmet plc	1950-2200	gu hambe

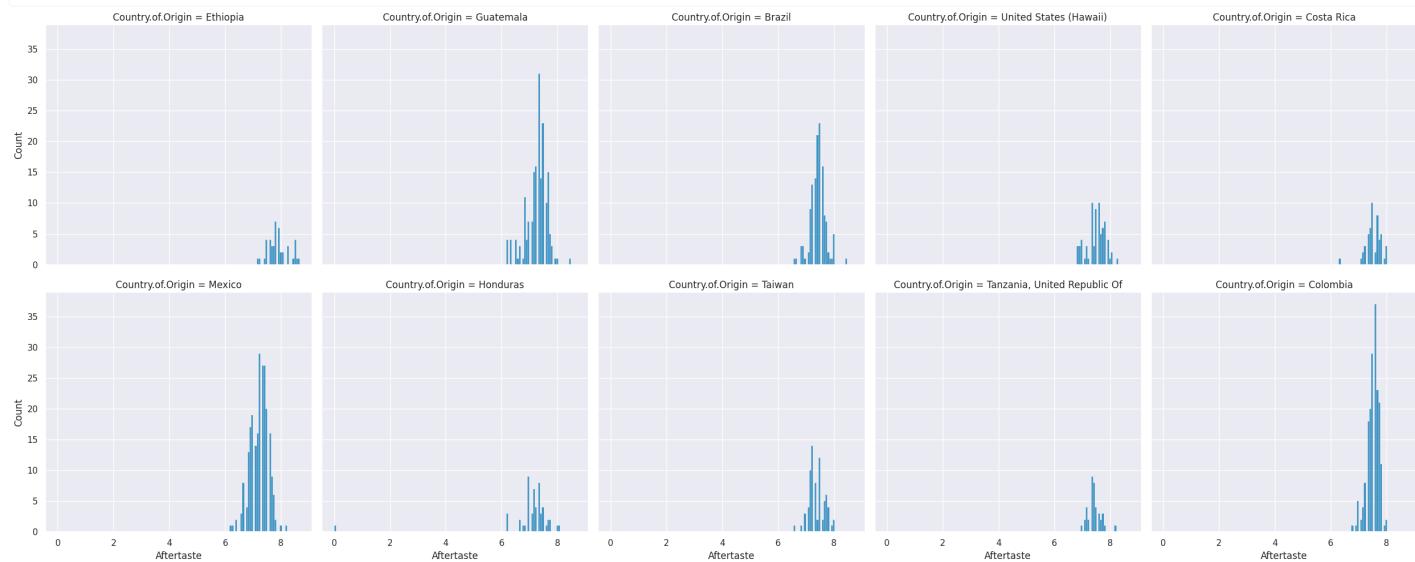
1 rows × 43 columns

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

```
((1068, 43), (1311, 43))
```

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

```
<seaborn.axisgrid.FacetGrid at 0x7f155874d3a0>
```



## 5.5. Variable types and data types

Related but not the same.

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

For example, `int16, int32, int64`

We can also have mathematical types of numbers

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

[Skip to main content](#)

- **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.

## 5.6. Questions After Class

### 5.6.1. how can I use data to make different graphs?

You can use the help function on any of the plot functions we have seen. You can also use the documentation.

The [seaborn gallery](#) is a good place to get ideas

### 5.6.2. Will we learn about different types of graphs?

We will use a few more types in class, but also you will probably use the documentation to learn more types. The [seaborn gallery](#) is a good place to start.

### 5.6.3. What are the advantages with work with the lambda function?

The `lambda` function is convenient because it is defined on one line.

### 5.6.4. Is a lambda function similar to an arrow function in Javascript?

It does look similar, but I am not an expert in javascript.

### 5.6.5. How do you decide on the best graphs to use for each piece of data?

Different types of plots make different conclusions easy to see.

### 5.6.6. why do you use axis = 1 for the check\_top\_countries

### 5.6.7. could you do group by instead of value counts for the last question?

yes

### 5.6.8. more about different ways to plot data and how to get specific rows and compare them

5.6.9. is there a document that shows us all the diffrent commands that allow us to do things to files

## 6. Tidy Data and Structural Repairs

### 6.1. Intro

This week, we'll be cleaning data.

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

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

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

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

#### ⚠ Warning

this is incomplete, but will get filled in this week.

```
import pandas as pd
import seaborn as sns

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

### 6.2. What is Tidy Data

Read in the three csv files described below and store them in a list of dataFrames

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

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

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

```
study_df_list[0]
```

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

```
study_df_list[1]
```

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

```
study_df_list[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

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

- find the average effect per person across treatments
- find the average effect per treatment across people

This works differently for these three versions.

```
df_a = study_df_list[0]
df_a.mean(numeric_only=True, )
```

```
treatmenta    9.500000
treatmentb    4.666667
dtype: float64
```

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

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

```
df_a.mean(numeric_only=True, axis=1)
```

```
0      2.0
1     13.5
2      2.0
dtype: float64
```

```
df_b = study_df_list[1]
df_b
```

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

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

```
study_df_list[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

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

```
study_df_list[2].groupby('treatment').mean(numeric_only=True)
```

	result
treatment	
<b>a</b>	9.500000
<b>b</b>	4.666667

```
study_df_list[2].groupby('person').mean(numeric_only=True)
```

	result
person	
<b>Jane Doe</b>	13.5
<b>John Smith</b>	2.0
<b>Mary Johnson</b>	2.0

## 6.3. Tidying Data

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

```
df_a.melt(id_vars='name', value_vars=['treatmenta', 'treatmentb'])
```

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

When we melt a dataset:

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

```
df_a.melt(id_vars='name', value_vars=['treatmenta', 'treatmentb'],
           var_name= 'treatment_type',
           value_name='result',)
```

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

## 6.4. Transforming the Coffee Data

Let's do it for our coffee data:

```
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, index_col=0)
```

	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	gu hambe
2	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural development plc	1950-2200	gu hambe

2 rows × 43 columns

`coffee_df.columns`

```
Index(['Species', 'Owner', 'Country.of-Origin', 'Farm.Name', 'Lot.Number',
       'Mill', 'ICO.Number', 'Company', 'Altitude', 'Region', 'Producer',
       'Number.of.Bags', 'Bag.Weight', 'In.Country.Partner', 'Harvest.Year',
       'Grading.Date', 'Owner.1', 'Variety', 'Processing.Method', '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')
```

```
scores_of_interest = ['Aroma',
                      'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance', 'Uniformity',
                      'Clean.Cup', 'Sweetness', ]
```

`coffee_df[scores_of_interest].head(2)`

	Aroma	Flavor	Aftertaste	Acidity	Body	Balance	Uniformity	Clean.Cup	Sweetness
1	8.67	8.83	8.67	8.75	8.50	8.42	10.0	10.0	10.0
2	8.75	8.67	8.50	8.58	8.42	8.42	10.0	10.0	10.0

We can make this tall using melt to transform with that set of questions and then rename the `value` and `variable` columns to be descriptive.

```
coffee_df_tall = coffee_df.melt(id_vars='Country.of-Origin', value_vars=scores_of_interest,
                                var_name='rating_type',
                                value_name='score')
coffee_df_tall.head(1)
```

	Country.of-Origin	rating_type	score
0	Ethiopia	Aroma	8.67

Notice that the actual column names inside the `scores_of_interest` variable become the values in the variable column.

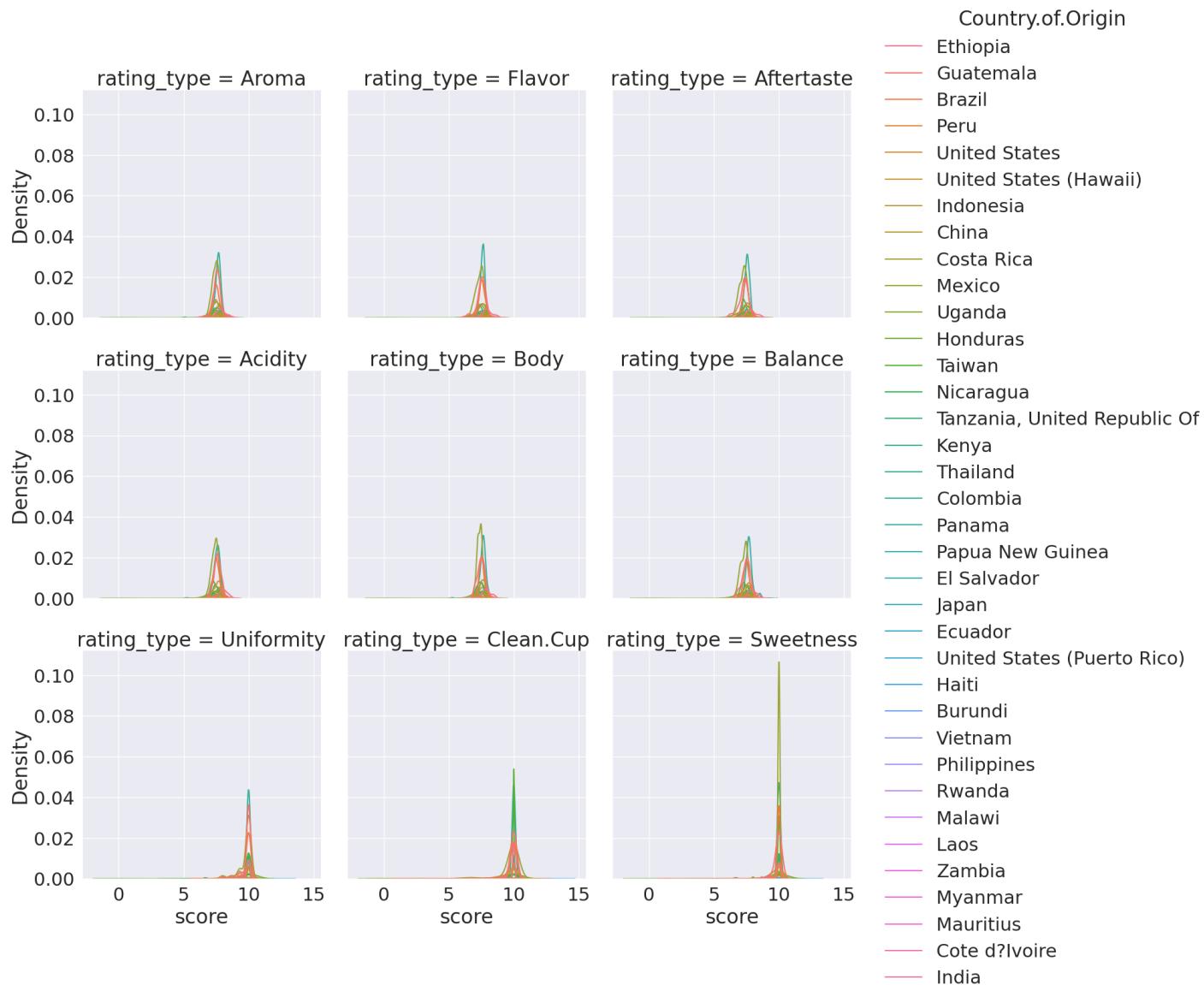
This one we can plot in more ways:

[Skip to main content](#)

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

```
/tmp/ipykernel_2061/2608901824.py:1: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `wa
sns.displot(data = coffee_df_tall, kind = 'kde', x='score',
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9dbe92fee0>
```



find why warning

## 6.5. Filtering Data

### ⚠ Warning

this was not in class this week, but is added here for completeness

[Skip to main content](#)

```
high_prod = coffee_df[coffee_df['Number.of.Bags'] > 250]  
high_prod.shape
```

(368, 43)

```
coffee_df.shape
```

(1311, 43)

We see that filters and reduces. We can use any boolean expression in the square brackets.

```
top_balance = coffee_df[coffee_df['Balance'] > coffee_df['Balance'].quantile(.75)]  
top_balance.shape
```

(252, 43)

We can confirm that we got only the top 25% of balance scores:

```
top_balance.describe()
```

	Number.of.Bags	Aroma	Flavor	Aftertaste	Acidity	Body	Balance	Uniformity	C
count	252.000000	252.000000	252.000000	252.000000	252.000000	252.000000	252.000000	252.000000	252.000000
mean	153.337302	7.808889	7.837659	7.734167	7.824881	7.780159	8.003095	9.885278	1
std	126.498576	0.355319	0.318172	0.302481	0.320253	0.317712	0.213056	0.363303	1
min	0.000000	5.080000	7.170000	6.920000	7.080000	5.250000	7.830000	6.670000	1
25%	12.750000	7.647500	7.670000	7.500000	7.670000	7.670000	7.830000	10.000000	1
50%	165.500000	7.780000	7.830000	7.750000	7.830000	7.750000	7.920000	10.000000	1
75%	275.000000	8.000000	8.000000	7.920000	8.000000	7.920000	8.080000	10.000000	1
max	360.000000	8.750000	8.830000	8.670000	8.750000	8.580000	8.750000	10.000000	1

We can also use the `isin` method to filter by comparing to an iterable type

```
total_per_country = coffee_df.groupby('Country.of.Origin')['Number.of.Bags'].sum()  
top_countries = total_per_country.sort_values(ascending=False)[:10].index  
top_coffee_df = coffee_df[coffee_df['Country.of.Origin'].isin(top_countries)]
```

## 6.6. More manipulations

### ⚠ Warning

this was not in place this week, but is added here for completeness

[Skip to main content](#)

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
0	730000000	392000000	580200000
1	315040009	580000000	967290000

This dataset 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
0	a	730000000
1	a	315040009
2	b	392000000
3	b	580000000
4	c	580200000
5	c	967290000

However, I want an additional variable, so I will 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

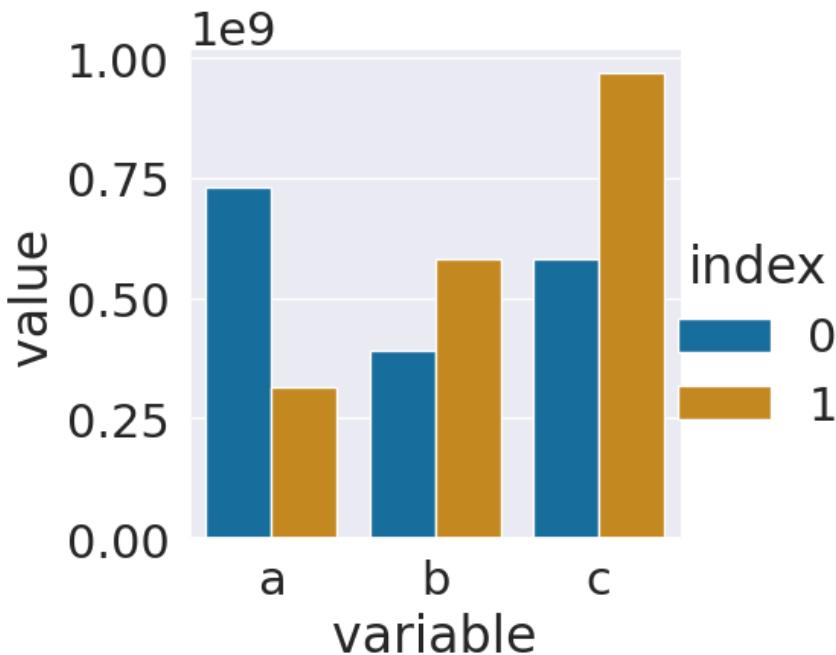
```
ls_tall_df = large_num_df.reset_index().melt(id_vars='index')  
ls_tall_df
```

	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 0x7f9dbbb406a0>
```



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

[Skip to main content](#)

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.

## 6.7. Questions after Class

6.7.1. In the data we had about treatment a and treatment b. Say there was another column that provided information about the age of the people. Could we create another value variable to or would we put it in the `value_vars` list along with treatment a and treatmentb?

We can have multiple variables as the `id_vars` so that the dataset would have 4 columns instead of 3.

6.7.2. Are there any prebuilt methods to identify and remove extreme outliers?

There may be, that is a good thing to look in the documentation for. However, typically the definition of an outlier is best made within context, so the filtering strategy that we just used would be the way to remove them, after doing some EDA.

6.7.3. Is there a specific metric to which a dataset must meet to consider it ‘cleaned’?

It's “clean” when it will work for the analysis you want to do. This means clean enough for one analysis may not be enough for another goal. Domain expertise is always important.

6.7.4. how does melt work underneath the hood?

The [melt documentation](#) is the place to read about the details. It also includes a link to the [source code](#) if reading how it is implemented in code is more helpful to you than reading English.

6.7.5. Do people ever write their melted datasets to new files for redistribution, maybe to make things easier for future researchers?

Yes! If you look in the R Tidy Tuesday datasets most distribute the “raw” and cleaned data.

6.7.6. the ending graphs are hard to read. Is there a way to label specific countries kde lines?

Yes

6.7.7. how do you get the index of something in a row

I do not understand this question, please reach out with more info so that I can help you

6.7.8. i was trying to do something earlier where I got the max of a column but wanted to

[Skip to main content](#)

most pandas methods allow you to use `axis` to apply them across rows instead of columns

### 6.7.9. cleaning out different data types

Once the data is loaded into pandas, we can use all of the same techniques

### 6.7.10. Is there a scenario where a dataset cannot be cleaned using pandas methods?

yes, there are cases where cleaning the already loaded dataframe is not the best way to make the necessary fixes. One tool that can help is called [Openrefine](#) it is an open source program that provides a GUI based interface to clean data, but also outputs a script of what actions were taking, to make the cleaning replicable.

This is something you could learn about and try for the level 3 prepare achievement in your portfolio. One resource for learning it is the [Data Carpentry lesson on Data Cleaning for Ecologists](#)

### 6.7.11. Is there another standard format for sharing cleaned datasets?

.csv is the widely accepted preferred format for well strucutured tabular data that is of reasonable size. If it is too large, a database might (next week!) can be useful.

### 6.7.12. With extremely large datasets, is there a way to find what pieces of the dataset are missing values other than manually checking

Using strategies like we did to day, by testing dropping and the EDA strategies that we used last week can help. If it is a very large dataset, you may need to use more powerful system, but the basics work out the same.

Also, remember in most contexts, you will have some relevant domain knowledge that can help you.

## 7. 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.

### 7.1. Cleaning Data review

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](#).

### 7.1.1. A Cleaning Data Recipe

not everything possible, but good enough for this course

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

```
import pandas as pd
import seaborn as sns
import numpy as np

sns.set_theme(palette= "colorblind")
# toy data set
na_toy_df = pd.DataFrame(data = [[1,3,4,5],[2 ,6, pd.NA,3]])

# coffee data
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)

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

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

## 7.2. What is clean enough?

This is a great question, without an easy answer.

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

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

in 2020 there was a whole workshop on missing

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

There are also many classic approaches both when training and when [applying models](#).

[example application in breast cancer detection](#)

Even in pandas, dealing with [missing values](#) is under [experimentation](#) as to how to represent it symbolically

Missing values even causes the [datatypes](#) to change

- dropna
- fillna

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()
```

```
--> NameError                                     Traceback (most recent call last)
Cell In[2], line 1
----> 1 coffee_df_fixedcols.info()

NameError: name 'coffee_df_fixedcols' is not defined
```

## 7.2.1. 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
```

```
--> NameError                                     Traceback (most recent call last)
Cell In[3], line 1
----> 1 coffee_df_fixedcols['lot_number'].dtype

NameError: name 'coffee_df_fixedcols' is not defined
```

And we can look at the value counts.

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

```
--> NameError                                     Traceback (most recent call last)
Cell In[4], line 1
----> 1 coffee_df_fixedcols['lot_number'].value_counts()

NameError: name 'coffee_df_fixedcols' is not defined
```

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'

[Skip to main content](#)

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

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

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 reassigned the pointer. It's under discussion to `deprecate`
- write to a new DataFrame
- add a column

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

```
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 development plc	1950-2200	gu hambe

1 rows × 44 columns

## 7.3. Dropping

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

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

We can look at dropping in this toy data set.

```
na_toy_df
```

	0	1	2	3
0	1	3	4	5
1	2	6	<NA>	3

```
na_toy_df.dtypes
```

[Skip to main content](#)

```
0    int64  
1    int64  
2    object  
3    int64  
dtype: object
```

```
na_toy_df.dropna()
```

	0	1	2	3
0	1	3	4	5

```
na_toy_df.dropna(axis=1)
```

	0	1	3
0	1	3	5
1	2	6	3

```
na_toy_df.mean()
```

```
0    1.5  
1    4.5  
2    4.0  
3    4.0  
dtype: object
```

why is this object?

### 7.3.1. Dropping missing values

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

```
coffee_df.shape
```

```
(1311, 44)
```

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

```
(130, 44)
```

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

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

[Skip to main content](#)

```
(1084, 44)
```

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

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

```
(1311, 43)
```

```
n_rows, _ = coffee_df.shape
```

## 7.4. Inconsistent values

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

Let's look at the `in_country_partner` column

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

In.Country.Partner	
Specialty Coffee Association	295
AMECAFE	205
Almacafé	178
Asociacion Nacional Del Café	155
Brazil Specialty Coffee Association	67
Instituto Hondureño del Café	60
Blossom Valley International	58
Africa Fine Coffee Association	49
Specialty Coffee Association of Costa Rica	42
NUCOFFEE	36
Uganda Coffee Development Authority	22
Kenya Coffee Traders Association	22
Ethiopia Commodity Exchange	18
Specialty Coffee Institute of Asia	16
METAD Agricultural Development plc	15
Yunnan Coffee Exchange	12
Salvadoran Coffee Council	11
Specialty Coffee Association of Indonesia	10
Centro Agroecológico del Café A.C.	8
Asociación de Cafés Especiales de Nicaragua	8
Coffee Quality Institute	7
Asociación Mexicana De Cafés y Cafeterías De Especialidad A.C.	6
Tanzanian Coffee Board	6
Torch Coffee Lab Yunnan	2
Specialty Coffee Ass	1
Central De Organizaciones Productoras De Café y Cacao Del Perú - Central Café & Cacao	1
Blossom Valley International\n	1

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

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

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

```
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', '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',
 'lot_number_clean', 'in_country_partner_clean'],
 dtype='object')
```

```
coffee_df_clean = coffee_df.rename(lambda s: s.lower().replace('.','_'),axis=1)
coffee_df_clean.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 × 45 columns

## 7.5. JSONs

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.

```
course_gh_df.head(2)
```

	id	type	actor	repo	payload	public	created_at	
0	33072164078	PushEvent	{'id': 10656079, 'login': 'brownsarahm', 'disp...}	{'id': 688125102, 'name': 'rhodyprog4ds/BrownF...}	{'repository_id': 688125102, 'push_id': 156821...}	True	2023-11-03 12:58:40+00:00	{'id': 695 'rhodypr
1	33056386530	PushEvent	{'id': 41898282, 'login': 'github-actions[bot]...}	{'id': 688125102, 'name': 'rhodyprog4ds/BrownF...}	{'repository_id': 688125102, 'push_id': 156740...}	True	2023-11-02 21:18:25+00:00	{'id': 695 'rhodypr

### 7.5.1. Casting Review

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

```
a = '5'
```

we can check type

```
type(a)
```

```
str
```

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

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

```
int
```

## 7.5.2. Handling Data Within a Data Frame

We can see each row is a Series type.

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

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

The individual values in the actor column is then a dictionary

```
type(course_gh_df.loc[0]['actor'])
```

```
dict
```

We can use the series constructor to transform it.

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

```
id                               10656079
login                           brownsarahm
display_login                   brownsarahm
gravatar_id
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 )

```
course_gh_df['actor'].apply(pd.Series).head(1)
```

	<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?

compared to the original:

```
course_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>	33072164078	PushEvent	{'id': 10656079, 'login': 'brownsarahm', 'display_login': 'brownsarahm'}	{'repository_id': 688125102, 'name': 'rhodyprog4ds/BrownF...'} {'id': 695, 'push_id': 156821...}	True	2023-11-03 12:58:40+00:00	'rhodypr...

## 7.5.3. Unpacking at scale

here we see how the list comprehensions we looked at in isolation before start to come in handy.

[Skip to main content](#)

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

`pd.concat` takes a list of dataframes and puts the together in one DataFrame. see its docs for more detail

So, we use a list comprehension to iterate over all of the columns that we want to transform, transform them, store the fixed `DataFrame`s in a list and concat them together into a single new `DataFrame`

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

	<code>id</code>	<code>login</code>	<code>display_login</code>	<code>gravatar_id</code>	<code>url</code>
0	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>

1 rows × 30 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([course_gh_df[col].apply(pd.Series, ).rename(  
    columns= lambda i_col: col + '_' + i_col )  
    for col in js_col], axis=1).head()
```

	<code>actor_id</code>	<code>actor_login</code>	<code>actor_display_login</code>	<code>actor_gravatar_id</code>	<code>actor_url</code>
0	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>
1	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>
2	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>
3	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>
4	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>

5 rows × 30 columns

The `rename` method can take a `lambda` function to rename columns in a pattern. we want to combine the original column name with the new column name. `col + '_' + i_col` does this where `i_col` is the column name after the `.apply(pd.Series)` and the `col` is the column name of the original column before unpacking.

To finish off, we can first get the columns that are not in the unpacked, put them in a list, then add the two lists together before concatenating them all together.

```
cols_not_unpacked_list = [course_gh_df[[col for col in  
    course_gh_df.columns if not(col in js_col)]]]  
unpacked_cols_list = [course_gh_df[col].apply(pd.Series, ).rename(  
    columns= lambda i_col: col + '_' + i_col )  
    for col in is_col]
```

[Skip to main content](#)

	<a href="#">id</a>	<a href="#">type</a>	<a href="#">public</a>	<a href="#">created_at</a>	<a href="#">actor_id</a>	<a href="#">actor_login</a>	<a href="#">actor_display_login</a>	<a href="#">actor</a>
0	33072164078	PushEvent	True	2023-11-03 12:58:40+00:00	10656079	brownsarahm	brownsarahm	
1	33056386530	PushEvent	True	2023-11-02 21:18:25+00:00	41898282	github-actions[bot]	github-actions	
2	33056300355	PushEvent	True	2023-11-02 21:13:53+00:00	10656079	brownsarahm	brownsarahm	
3	32999780538	PushEvent	True	2023-11-01 02:02:55+00:00	41898282	github-actions[bot]	github-actions	
4	32999707705	ReleaseEvent	True	2023-11-01 01:58:44+00:00	10656079	brownsarahm	brownsarahm	
5	32999693640	CreateEvent	True	2023-11-01 01:57:54+00:00	10656079	brownsarahm	brownsarahm	
6	32999678673	PushEvent	True	2023-11-01 01:57:01+00:00	10656079	brownsarahm	brownsarahm	
7	32885738683	PushEvent	True	2023-10-27 01:32:41+00:00	41898282	github-actions[bot]	github-actions	
8	32885656799	ReleaseEvent	True	2023-10-27 01:26:18+00:00	10656079	brownsarahm	brownsarahm	
9	32885646237	CreateEvent	True	2023-10-27 01:25:29+00:00	10656079	brownsarahm	brownsarahm	
10	32885641572	PushEvent	True	2023-10-27 01:25:06+00:00	10656079	brownsarahm	brownsarahm	
11	32820758925	PushEvent	True	2023-10-25 01:51:38+00:00	41898282	github-actions[bot]	github-actions	
12	32820714691	ReleaseEvent	True	2023-10-25 01:48:20+00:00	10656079	brownsarahm	brownsarahm	
13	32820693647	CreateEvent	True	2023-10-25 01:46:50+00:00	10656079	brownsarahm	brownsarahm	
14	32820689144	PushEvent	True	2023-10-25 01:46:29+00:00	10656079	brownsarahm	brownsarahm	
15	32785677507	PushEvent	True	2023-10-23 21:36:13+00:00	41898282	github-actions[bot]	github-actions	
16	32785576615	PushEvent	True	2023-10-23 21:31:07+00:00	10656079	brownsarahm	brownsarahm	
17	32707776243	PushEvent	True	2023-10-20 00:25:40+00:00	41898282	github-actions[bot]	github-actions	
18	32707755904	ReleaseEvent	True	2023-10-20 00:24:10+00:00	10656079	brownsarahm	brownsarahm	
19	32707730227	CreateEvent	True	2023-10-20 00:22:06+00:00	10656079	brownsarahm	brownsarahm	
20	32707719407	PushEvent	True	2023-10-20 00:21:14+00:00	10656079	brownsarahm	brownsarahm	
21	32641556076	PushEvent	True	2023-10-18 01:04:35+00:00	41898282	github-actions[bot]	github-actions	
22	32641475242	PushEvent	True	2023-10-18	10656079	brownsarahm	brownsarahm	

[Skip to main content](#)

	<b>id</b>	<b>type</b>	<b>public</b>	<b>created_at</b>	<b>actor_id</b>	<b>actor_login</b>	<b>actor_display_login</b>	<b>actor_</b>
23	32604871288	PushEvent	True	2023-10-16 21:07:09+00:00	41898282	github-actions[bot]	github-actions	
24	32604765467	PushEvent	True	2023-10-16 21:02:15+00:00	10656079	brownsarahm	brownsarahm	
25	32566128012	PushEvent	True	2023-10-15 01:03:33+00:00	41898282	github-actions[bot]	github-actions	
26	32566122910	PushEvent	True	2023-10-15 01:02:38+00:00	10656079	brownsarahm	brownsarahm	
27	32553726043	IssueCommentEvent	True	2023-10-13 21:04:12+00:00	10656079	brownsarahm	brownsarahm	
28	32553152626	PushEvent	True	2023-10-13 20:29:21+00:00	41898282	github-actions[bot]	github-actions	
29	32553120988	ReleaseEvent	True	2023-10-13 20:27:28+00:00	10656079	brownsarahm	brownsarahm	

30 rows × 34 columns

## 7.6. Questions after class

### 7.6.1. After you do analysis with a specific column and cleaned it for that, should you restore the original dataframe and reclean it to do a different analysis?

You might, if the analyses are completely different and unrelated. More often, however, we would clean the whole dataset, save the cleaning script/notebook (can have more context), and save the cleaned dataset to a csv. Building more breadth of understanding of these practices, is what you will do with the last part of A4. Your task there is to look at a few examples of cleaning that I have gathered for you and answer questions that start to build your intuition with this.

Ultimately though, cleaning data is something that you do not know everything there is to know about it in one shot, over time you see more and more examples.

### 7.6.2. I don't fully understand the lambda function

If you want a technical specific understanding of it, I recommend the Python language documentation on [lambda functions](#) and the [wikipedia article on anonymous functions](#) for more breadth and other related concepts across languages.

At a practical level it is a shorthand syntax for defining a small function. For example the following two functions do the same thing.

```
repeat_lambda = lambda content, reps: content*reps

def repeat_func(content, reps):
    return content*reps
```

First, we can examine them

```
type(repeat_lambda), type( repeat_func)
```

[Skip to main content](#)

```
(function, function)
```

they are both callable, but slightly different types.

Now we can call our functions:

```
repeat_lambda('a', 3) == repeat_func('a', 3)
```

```
True
```

and this is not a specific case, but always works. We can do a small random experiment to see

We'll use the string library to get a string of the alphabet

```
import string
string.ascii_uppercase
```

```
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

We can pick even a random length, then random characters and a random number of repetitions

```
rand_length = np.random.randint(10)
random_content = np.random.choice(list(string.ascii_uppercase), size=rand_length)
rand_reps = np.random.randint(10)

random_content, rand_reps
```

```
(array(['R', 'H'], dtype='<U1'), 2)
```

We can still apply this and see that it is the same.

```
repeat_lambda(random_content, rand_reps) == repeat_func(random_content, rand_reps)
```

```
-----  
UFuncTypeError                                     Traceback (most recent call last)  
Cell In[40], line 1  
----> 1 repeat_lambda(random_content, rand_reps) == repeat_func(random_content, rand_reps)  
  
Cell In[35], line 1, in <lambda>(content, reps)  
----> 1 repeat_lambda = lambda content, reps: content*reps  
  3 def repeat_func(content, reps):  
  4     return content*reps  
  
UFuncTypeError: ufunc 'multiply' did not contain a loop with signature matching types (dtype('<U1'), dtype('<U1'))
```

### 7.6.3. Json use cases vs csv use cases

[Skip to main content](#)

## 7.6.4. Why are so many datasets so messy in the first place?

## 7.6.5. Are there more resources to see when its appropriate to fill in missing data with certain values?

I have not found a lot of good resources on this, unfortunately. Data Science is a complex discipline and **very new** especially at the undergraduate level. The first data science degrees were only at the graduate level.

The complexity lies in integrating information from computer science, statistics, and *domain knowledge*. Domain knowledge is going to be different in every dataset.

It is okay to now know for sure the best thing to do. The most important thing is document what you did and why so that you can justify the choices and consider their impact later in your analysis.

## 7.6.6. Can we get a more in-depth explanation of what is going on in the last piece of code you provided?

above

## 7.6.7. What is the normal percent of NAs that need to be filled for most people to get rid of that line?

Again, unfortunately there are not fixed rules.

Missing 10% of only 50 samples might be detrimental, where missing 30% of 10000 could be okay.

It depends what you are going to do with the data after cleaning, what the threshold is.

# 8. Merging Data

## 8.1. 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(1)
```

	TEAM_ID	PLAYER_ID	SEASON
0	1610612761	202695	2018

```
df_19.head(1)
```

	PLAYER_NAME	TEAM_ID	PLAYER_ID	SEASON
0	Royce O'Neale	1610612762	1626220	2019

### ! Important

Remember `shape` is a property, not a method, so it does not need `()`

Let's make note of the shape of each

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

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

### 8.1.1. What if we want to analyze them together?

We can stack them, but this does not make it easy to see , for example, who changed teams.

```
pd.concat([df_18, df_19])
```

	TEAM_ID	PLAYER_ID	SEASON	PLAYER_NAME
0	1610612761	202695	2018	NaN
1	1610612761	1627783	2018	NaN
2	1610612761	201188	2018	NaN
3	1610612761	201980	2018	NaN
4	1610612761	200768	2018	NaN
...	...	...	...	...
621	1610612745	203461	2019	Anthony Bennett
622	1610612737	1629034	2019	Ray Spalding
623	1610612744	203906	2019	Devyn Marble
624	1610612753	1629755	2019	Hassani Gravett
625	1610612754	1629721	2019	JaKeenan Gant

1374 rows × 4 columns

we can see that this is the total number of rows:

748+626

1374

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

### 8.1.2. How can we find which players changed teams?

To do this we want to have one player column and a column with each year's team.

We can use a merge to do that.

```
pd.merge(df_18, df_19).head(2)
```

TEAM_ID	PLAYER_ID	SEASON	PLAYER_NAME
---------	-----------	--------	-------------

if we merge them without any parameters, it tries to merge on all shared columns. We want to merge them using the `PLAYER_ID` column though, we would say that we are “merging on player ID” and we use the `on` parameter to do it. In this case, 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').head(2)
```

TEAM_ID_x	PLAYER_ID	SEASON_x	PLAYER_NAME	TEAM_ID_y	SEASON_y
-----------	-----------	----------	-------------	-----------	----------

0	1610612761	202695	2018	Kawhi Leonard	1610612746
---	------------	--------	------	---------------	------------

[Skip to main content](#)

Since there are other columns that appear in both DataFrames, they get a suffix, which by default is `x` or `y`, we can specify them though.

```
pd.merge(df_18, df_19, on='PLAYER_ID', suffixes=('_18', '_19')).head(2)
```

	TEAM_ID_18	PLAYER_ID	SEASON_18	PLAYER_NAME	TEAM_ID_19	SEASON_19
0	1610612761	202695	2018	Kawhi Leonard	1610612746	2019
1	1610612761	1627783	2018	Pascal Siakam	1610612761	2019

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.

## 8.2. Merge type examples

```
left = pd.DataFrame(  
    {  
        "key": ["K0", "K1", "K2", "K3"],  
        "A": ["A0", "A1", "A2", "A3"],  
        "B": ["B0", "B1", "B2", "B3"],  
    }  
)  
  
right = pd.DataFrame(  
    {  
        "key": ["K0", "K1", "K2", "K3"],  
        "C": ["C0", "C1", "C2", "C3"],  
        "D": ["D0", "D1", "D2", "D3"],  
    }  
)  
left
```

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

```
right
```

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2

[Skip to main content](#)

```
pd.merge(left,right)
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

```
left = pd.DataFrame(  
    {  
        "key1": ["K0", "K0", "K1", "K2"],  
        "key2": ["K0", "K1", "K0", "K1"],  
        "A": ["A0", "A1", "A2", "A3"],  
        "B": ["B0", "B1", "B2", "B3"],  
    }  
)  
  
right = pd.DataFrame(  
    {  
        "key1": ["K0", "K1", "K1", "K2"],  
        "key2": ["K0", "K0", "K0", "K0"],  
        "C": ["C0", "C1", "C2", "C3"],  
        "D": ["D0", "D1", "D2", "D3"],  
    }  
)  
  
result = pd.merge(left, right, on=["key1", "key2"])
```

```
result
```

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

```
pd.merge(left, right, on=["key1", "key2"], how='outer')
```

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN
5	K2	K0	NaN	NaN	C3	D3

[Skip to main content](#)

### 3.2.1. Which players played in 2018, but not 2019:

We have different types of merges, inner is both, out is either. Left and right keep all the rows of one DataFrame. We can use left with `df_18` as the left DataFrame to see which players played only in 18.

```
df_18_only = pd.merge(df_18, df_19, on='PLAYER_ID', suffixes=('_18', '_19'), how='left')
df_18_only.head(2)
```

	TEAM_ID_18	PLAYER_ID	SEASON_18	PLAYER_NAME	TEAM_ID_19	SEASON_19
0	1610612761	202695	2018	Kawhi Leonard	1.610613e+09	2019.0
1	1610612761	1627783	2018	Pascal Siakam	1.610613e+09	2019.0

```
df_18_only.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 754 entries, 0 to 753
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   TEAM_ID_18    754 non-null    int64  
 1   PLAYER_ID     754 non-null    int64  
 2   SEASON_18     754 non-null    int64  
 3   PLAYER_NAME   538 non-null    object  
 4   TEAM_ID_19     538 non-null    float64 
 5   SEASON_19     538 non-null    float64 
dtypes: float64(2), int64(3), object(1)
memory usage: 35.5+ KB
```

```
len(df_18_only[df_18_only['TEAM_ID_19'].isna()]['PLAYER_ID'].unique())
```

178

```
df_1819_outer = pd.merge(df_18, df_19, on='PLAYER_ID', suffixes=('_18', '_19'), how='outer')
```

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_outer.dtypes
```

```
TEAM_ID_18      float64
PLAYER_ID       int64
SEASON_18       float64
PLAYER_NAME     object
TEAM_ID_19       float64
SEASON_19       float64
dtype: object
```

nan is a float

```
import numpy as np
```

[Skip to main content](#)

```
float
```

```
df_1819_outer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 927 entries, 0 to 926
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   TEAM_ID_18    754 non-null    float64
 1   PLAYER_ID     927 non-null    int64  
 2   SEASON_18     754 non-null    float64
 3   PLAYER_NAME   711 non-null    object  
 4   TEAM_ID_19    711 non-null    float64
 5   SEASON_19     711 non-null    float64
dtypes: float64(4), int64(1), object(1)
memory usage: 43.6+ KB
```

Back the the question, we can also use a left merge. To pick out those rows:

```
df_1819_outer['TEAM_ID_19'].isna()
```

```
0    False
1    False
2    False
3    False
4    False
...
922  False
923  False
924  False
925  False
926  False
Name: TEAM_ID_19, Length: 927, dtype: bool
```

this gives us a boolean list of `False` where there is a value and `True` where there is `nan`. Since we applied this to the `TEAM_ID_19` column, it gives us a `True` for each row that represents a player playing in 2018, but not 19.

However this still has repetitions for the players that played for two teams in 2018. If we tke the unique values fro the `PLAYER_ID` column we get the IDs for the players who played in 18, but not 19. Then we can use `len` (a built in python function) to get the number of players that played in 2018, but not 2019.

```
len(df_1819_outer[df_1819_outer['TEAM_ID_19'].isna()]['PLAYER_ID'].unique())
```

178

```
df_18.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 748 entries, 0 to 747
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   TEAM_ID     748 non-null    int64  
 1   PLAYER_ID   748 non-null    int64  
 2   SEASON      748 non-null    int64  
dtypes: int64(3)
memory usage: 17.7 KB
```

## 8.3. Getting Data from Databases

### 8.3.1. What is a Database?

A common attitude in Data Science is:

If your data fits in memory there is no advantage to putting it in a database: it will only be slower and more frustrating. —  
Hadley Wickham

Businesses and research organizations nearly always have too much data to feasibly work without a database. Instead, they use different tools which are designed to scale to very large amounts of data. These tools are largely databases like Snowflake or Google's BigQuery and distributed computing frameworks like Apache Spark.

#### ⚠ Warning

We are going to focus on the case of getting data out of a Database so that you can use it and making sure you know what a Database is.

You could spend a whole semester on databases:

- CSC436 covers how to implement them in detail (recommended, but requires CSC212)
- BAI456 only how to use them (counts for DS majors, but if you want to understand them deeper, the CSC one is recommended)

For the purpose of this class the key attributes of a database are:

- it is a collection of tables
- the data is accessed live from disk (not RAM)
- you send a query to the database to get the data (or your answer)

Databases can be designed in many different ways. For examples two popular ones.

- [SQLite](#) is optimized for transactional workloads, which means a high volume of requests that 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

### 8.3.2. 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 0x7f7d3f21f3d0>)
```

Next, we set up a connection, that links the the notebook to the database. To use it, we add a cursor.

```
conn = sqlite3.connect('nba1819.db')  
cursor = conn.cursor()
```

We can use `execute` to pass SQL queries through the cursor to the database.

```
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
```

```
<sqlite3.Cursor at 0x7f7d3f2222d0>
```

Then we use `fetchall` to get the 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 use `pd.read_sql` to send queries, get the result sand transform them into a DataFrame all at once

We can pass the exact same queries if we want.

```
pd.read_sql("SELECT name FROM sqlite_master WHERE type='table';", conn)
```

	name
0	teams
1	conferences
2	playerGameStats2018
3	playerGameStats2019
4	teamGameStats2018
5	teamGameStats2019
6	playerTeams2018
7	playerTeams2019
8	teamDailyRankings2018
9	teamDailyRankings2019
10	playerNames

or we can get all of one of the tables:

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

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

#### 8.4.1. 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	0	1610612761

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

#### 8.4.2. 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)
```

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.

[Skip to main content](#)

```
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']
```

Again 135 is more than 119.

We can also make this a little neater to print it as a DataFrame. If we use `reset_index` it will make a DataFrame, but the count column will still be named `PLAYER_ID` so we can rename it.

```
player_counts_1819_team.reset_index().rename(columns={'PLAYER_ID': 'num_players'})
```

	conference_1819	team_1819	num_players
0	new	new	119
1	same	new	135
2	same	same	263

All in all, this gives us a good answer that we can get with data and display answers and this is one way that using multiple data sources can help answer richer questions.

```
conn.close()
```

## 8.5. Questions After Class

### 8.5.1. How to merge multiple data frames at a time/ filter, I dont know if that is possible

You can filter and merge in more complex ways in a database in at least some cases, but in pandas merge is strictly two at a time.

### 8.5.2. What do you recommend I know about SQL from someone who has not been exposed to it much before this class?

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.

### 8.5.3. What other SQL ‘keywords’ in the queries are there? ex: SELECT, FROM, WHERE

[quick reference](#)

### 8.5.4. 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.

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

[Skip to main content](#)

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

### 8.5.6. When is sql more advantageous to use?

When you use a database. It's a query language

### 8.5.7. how are databases and sql queries better than dataframes other than large datasets

It allows you to have a single file instead of separate ones, but that's it. The real motivation for databases is their advantages for large datasets.

On slower computers with less memory this was more important.

**learning** databases is also a good way to learn about schemas and structure, if you learn in depth, like a full course.

### 8.5.8. Is it possible to do something analogous to a merge by making a SQL query through pandas?

yes! you can send any sql query through pandas

### 8.5.9. so is the fetchall() take the names(titles) for the datas from database?

It takes the output of the query.

### 8.5.10. When you run database is there a clear way to see what structure the data is in?

You have to pull queries to get the data.

### 8.5.11. how can I practice on my own?

Download

### 8.5.12. when is the first portfolio check again?

Just posted now to the portfolio page

### 8.5.13. Is there a guideline for asking queries that we will learn about?

the queries should match your questions

.....

It could be that the whole database is, but you run a query for a subset that can fit in memory and then you want to plot it using seaborn.

For today, we also did that so that you can get a chance to see some SQL queries, without having to install a separate program on your computers.

## 9. Web Scraping

### Note

Dates:

- A5 due 10/10
- P1 due 10/16 (ideally you have already started this)
- A6 due 10/18 (it's short, I promise)
- then back to assignments due Monday

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

### Warning

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

```
pip install beautifulsoup4
```

then restart your kernel (Kernel menu, choose restart)

### 9.1. Getting Data From Websites

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

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

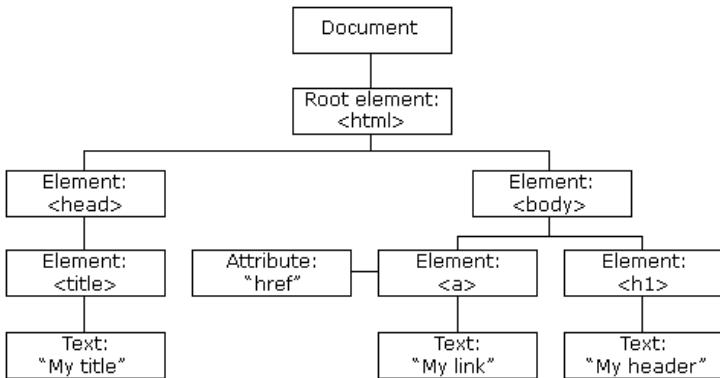
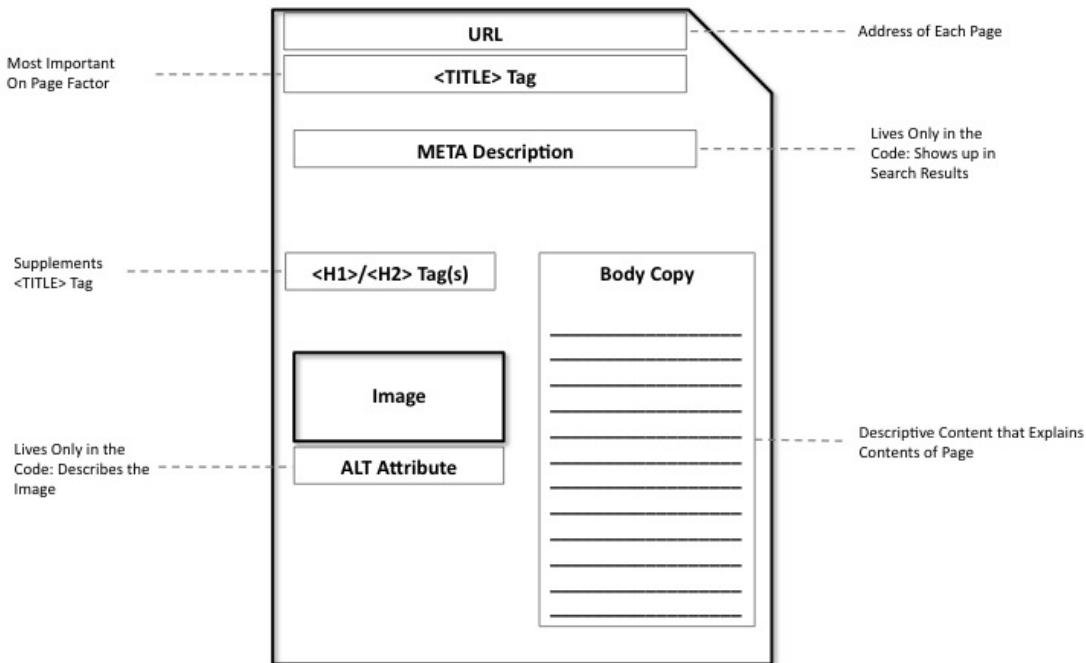
► Show code cell output

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

### 9.2. Everything is Data

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

[Skip to main content](#)



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

## 9.3. Scraping a URI website

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

from the people page of the department website.

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

i Note

the in  
for di

## ⚠ Warning

With great power comes great responsibility.

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

We'll save the URL for easy use

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

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

```
cs_people_html = requests.get(cs_people_url).content
```

This is raw:

```
cs_people_html[:200]
```

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

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

```
cs_people = BeautifulSoup(cs_people_html, 'html.parser')
```

```
type(cs_people)
```

```
bs4.BeautifulSoup
```

### 9.3.1. Looking at tags

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

```
cs_people.a
```

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

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

```
cs_people.h3
```

[Skip to main content](#)

Not

here  
by loc  
chara

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

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

### 9.3.2. Searching the source

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

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

```
bs4.element.ResultSet
```

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

```
[<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/gavino-puggioni/"><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 | Chair</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.4388</span> <br/> <a class="u-email" href="mailto:gppuggioni@uri.edu">gppuggioni@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" width="126"/>
</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 | Director of Undergraduate Studies</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/natallia-katenka/"><img alt="" class="u-photo wp-post-image" height="200" width="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 Data Science</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 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="200" width="200"/>
</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 | Director of Graduate Studies</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" width="200"/>
</figure>
```

[Skip to main content](#)

```
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor | Director of Graduate Studies</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.4504</span> <br/> <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/marco-alvarez/"><img alt="" class="u-photo wp-post-image" height="120" width="120" alt="Marco Alvarez profile picture" /></a>
</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</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5009</span> <br/> <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">Associate 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" width="300" alt="Sarah Brown profile picture" /></a>
</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" width="2475" alt="Michael Conti profile picture" /></a>
</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">Associate 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">
```

```
</div>
<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: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/victor-fay-wolfe/"><img alt="" class="u-photo wp-post-image" height="120" width="120" alt="Portrait of Victor Fay-Wolfe" data-bbox="113 113 188 128"/></img></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/abdeltawab-hendawi/"><img alt="" class="u-photo wp-post-image" height="120" width="120" alt="Portrait of Abdeltawab Hendawi" data-bbox="113 113 188 128"/></img></a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/abdeltawab-hendawi/">Abdeltawab 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> <br/> <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>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/jean-yves-herve/"><img alt="" class="u-photo wp-post-image" height="290" width="290" alt="Portrait of Jean-Yves Hervé" data-bbox="113 113 188 128"/></img></a>
</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>,
```

```
<ul>
<li 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> <br/> <a class="u-email" href="mailto:soheyb.kouider@uri.edu">soheyb.kouider@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" width="150" alt="Portrait of Edmund Lamagna" /></a>
</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" width="280" alt="Portrait of Indrani Mandal" /></a>
</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">Associate 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>
</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/shaun-wallace/"><img alt="Shaun Wallace" class="u-photo wp-post-image" height="150" width="150" alt="Portrait of Shaun Wallace" /></a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/shaun-wallace/">Shaun Wallace</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:shaun.wallace@uri.edu">shaun.wallace@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>,
```

[Skip to main content](#)

```
<ulv class="header">
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/yunshu-jasmine-wang/">Yunshu (Jasmine) Wang</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Teaching Professor</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><a class="u-email" href="mailto:yunshu_wang@uri.edu">yunshu_wang@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/haihan-mark-yu/"><img alt="Haihan Yu" class="u-photo wp-post-image" he:</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/haihan-mark-yu/">Haihan (Mark) Yu</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:haihan.yu@uri.edu">haihan.yu@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" lo:</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>
<p class="people-misc"><a class="u-email" href="mailto:yichizhang@uri.edu">yichizhang@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/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="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="96</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> <br/> <a class="u-email" href="mailto:buchana
```

[Skip to main content](#)

```

</ul>,
<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> <br/> <a class="u-email" href="mailto:rsschwa
<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/ying-zhang/"><img alt="" class="u-photo wp-post-image" height="250" lo
</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> <br/> <a class="u-email" href="mailto:yingzha
<div style="clear:both;"></div>
</div>
</div>]

```

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)

```

27

### ! Important

answer to questions about searching from the docs

We can also look at only the first instance

[Skip to main content](#)

```
<div class="peopleitem h-card has-thumbnail">
<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 | Chair</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.4388</span> <br/> <a class="u-email" href="mailto:gppuggioni@uri.edu">gppuggioni@uri.edu</a></p>
<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')
```

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

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

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

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

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

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

```
'Gavino Puggioni'
```

Finally, now that we know how to get one out, we can put it all in a list comprehension

```
names = [person.find('h3', 'p-name').a.string for person in people_items]
```

## 9.4. Pulling more information

First, we look at the whole person entry again.

```
people_items[0]
```

Skip to main content

! Imp  
In the  
similar  
replic  
the a  
you d  
intern  
audie  
concl  
you a

```

<div class="peopleitem h-card has-thumbnail">
<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 | Chair</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.4388</span> <br/> <a class="u-email" href="mailto:gppuggioni@uri.edu">gppuggioni@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>

```

How to pull out the titles for each person (eg Assistant Teaching Professor, Associate Professor)

on one item, the `p` tag with the `people-title` class gets us what we want and then we can

```
[person.find('p', 'people-title').a.string for person in people_items]
```

```

-----
AttributeError                                 Traceback (most recent call last)
Cell In[19], line 1
----> 1 [person.find('p', 'people-title').a.string for person in people_items]

Cell In[19], line 1, in <listcomp>(.0)
----> 1 [person.find('p', 'people-title').a.string for person in people_items]

AttributeError: 'NoneType' object has no attribute 'string'

```

This gives an error because there is no `<a>` tag inside the `<p>` tag

Python's null concept (outside of pandas and numpy that have nan float values) is `None` we can assign it to a variable if we want.

```
a = None
```

Its type is `NoneType`

```
type(a)
```

`NoneType`

So the error message says that the thing we are applying `.string` to is `None`, since that is the `.a` which means there is no `<a>` inside a `<p class = 'people-title'>`

If we take out the `a` it works and then we can iterate and store like above.

```
titles = [person.find('p', 'people-title').string for person in people_items]
```

[Skip to main content](#)

```
disciplines = [d.string for d in cs_people.find_all("p", 'people-department')]
emails = [e.string for e in cs_people.find_all("a", 'u-email')]
```

We can finally use the DataFrame constructor to make it a table. I chose to use a dictionary in class

```
css_df = pd.DataFrame({'name':names, 'title':titles, 'email':emails, 'discipline':disciplines})
css_df
```

	name	title	email	discipline
0	Gavino Puggioni	Associate Professor   Chair	gpuggioni@uri.edu	Statistics
1	Lisa DiPippo	Professor   Director of Undergraduate Studies	ldipippo@uri.edu	Computer Science
2	Natallia Katenka	Associate Professor   Director of Data Science	nkatenka@uri.edu	Statistics
3	Krishna Venkatasubramanian	Assistant Professor   Director of Graduate Stu...	krish@uri.edu	Computer Science
4	Jing Wu	Associate Professor   Director of Graduate Stu...	jing_wu@uri.edu	Statistics
5	Marco Alvarez	Associate Professor	malvarez@uri.edu	Computer Science
6	Samantha Armenti	Associate Teaching Professor	sarmenti@uri.edu	Computer Science
7	Sarah Brown	Assistant Professor	brownsarahm@uri.edu	Computer Science
8	Michael Conti	Associate Teaching Professor	michaelconti@uri.edu	Computer Science
9	Noah Daniels	Associate Professor	noah_daniels@uri.edu	Computer Science
10	Victor Fay-Wolfe	Professor	vfaywolfe@uri.edu	Computer Science
11	Lutz Hamel	Associate Professor	lutzhamel@uri.edu	Computer Science
12	Abdeltawab Hendawi	Assistant Professor	hendawi@uri.edu	Data Science   Computer Science
13	Jean-Yves Hervé	Associate Professor	jyh@cs.uri.edu	Computer Science
14	Soheyb Kouider	Associate Teaching Professor	soheyb@uri.edu	Statistics
15	Edmund Lamagna	Professor	eal@cs.uri.edu	Computer Science
16	Indrani Mandal	Associate Teaching Professor	indrani_mandal@uri.edu	Computer Science
17	Jonathan Schrader	Assistant Teaching Professor	jonathan.schrader@uri.edu	Computer Science
18	Shaun Wallace	Assistant Professor	shaun.wallace@uri.edu	Computer Science
19	Yunshu (Jasmine) Wang	Assistant Teaching Professor	yunshu_wang@uri.edu	Statistics
20	Haihan (Mark) Yu	Assistant Professor	haihan.yu@uri.edu	Statistics
21	Yichi Zhang	Assistant Professor	yichizhang@uri.edu	Statistics
22	Guangyu Zhu	Assistant Professor	guangyuzhu@uri.edu	Statistics
23	Ashley Buchanan	Limited Joint Appointment	buchanan@uri.edu	Biostatistics
24	Nina Kajiji	Adjunct Associate Professor	nina@uri.edu	Computer Science
25	Rachel Schwartz	Assistant Professor – Limited Joint Appointment	rsschwartz@uri.edu	Biological Sciences
26	Ying Zhang	Assistant Professor – Limited Joint Appointment	yingzhang@uri.edu	Computer Science

We could also use a list of list and separate list of column names.

```
css_df_b = pd.DataFrame(data=[names,titles,emails,disciplines],
                        columns=['names','titles','emails','disciplines'])
```

[Skip to main content](#)

```

-----
AssertionError                                         Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/internals/construction.py:933
933     try:
--> 934         columns = _validate_or_indexify_columns(contents, columns)
935     except AssertionError as err:
936         # GH#26429 do not raise user-facing AssertionError

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/internals/construction.py:979
979     if not is_mi_list and len(columns) != len(content): # pragma: no cover
980         # caller's responsibility to check for this...
--> 981     raise AssertionError(
982         f"{len(columns)} columns passed, passed data had "
983         f"{len(content)} columns"
984     )
985 if is_mi_list:
986     # check if nested list column, length of each sub-list should be equal

AssertionError: 4 columns passed, passed data had 27 columns

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

ValueError                                         Traceback (most recent call last)
Cell In[25], line 1
----> 1 css_df_b = pd.DataFrame(data=[names,titles,emails,disciplines],
2                               columns=['names','titles','emails','disciplines'])
3 css_df_b

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:782, in DataFrame.__init__(self, data, index, columns, dtype, copy, **kwargs)
780     if columns is not None:
781         columns = ensure_index(columns)
--> 782     arrays, columns, index = nested_data_to_arrays(
783         # error: Argument 3 to "nested_data_to_arrays" has incompatible
784         # type "Optional[Collection[Any]]"; expected "Optional[Index]"
785         data,
786         columns,
787         index, # type: ignore[arg-type]
788         dtype,
789     )
790     mgr = arrays_to_mgr(
791         arrays,
792         columns,
(...):
795         typ=manager,
796     )
797 else:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/internals/construction.py:495
495     if is_named_tuple(data[0]) and columns is None:
496         columns = ensure_index(data[0]._fields)
--> 498     arrays, columns = to_arrays(data, columns, dtype=dtype)
499     columns = ensure_index(columns)
501 if index is None:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/internals/construction.py:837
837     data = [tuple(x) for x in data]
838     arr = _list_to_arrays(data)
--> 840     content, columns = _finalize_columns_and_data(arr, columns, dtype)
841 return content, columns

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/internals/construction.py:934
934     columns = _validate_or_indexify_columns(contents, columns)
935     except AssertionError as err:
936         # GH#26429 do not raise user-facing AssertionError
--> 937         raise ValueError(err) from err
939 if len(contents) and contents[0].dtype == np.object_:
940     contents = convert_object_array(contents, dtype=dtype)

ValueError: 4 columns passed, passed data had 27 columns

```

## 9.5. Crawling and scraping

Remember we pulled the names out of links, when in the browser, we click on the links, we see that they are to a profile page. On these pages, they have the office number. Let's add those to our dataframe.

First, we will do it for one person, then make a loop.

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

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

We see that the information that we want is in the `href` attribute, to read that, we check the [documentation](#). This tells us there is a `.attrs` attribute of the python object we are working with.

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

```
{'href': 'https://web.uri.edu/cs/meet/gavino-puggioni/'}
```

It's a dictionary and the attribute we want is the key we want.

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

Now, we do the same thing we did above, request, pull the content from the response and then use the parser.

```
puggioni_html = requests.get(puggioni_url).content
puggioni_info = BeautifulSoup(puggioni_html, 'html.parser')
```

then we find the tag and class we need from inspecting and pull that.

```
puggioni_info.find_all('li', 'people-location')
```

```
[<li class="people-location"><strong>Office Location:</strong> Tyler Hall 254</li>]
```

it's an interable, so we pull the item out

```
puggioni_info.find_all('li', 'people-location')[0]
```

```
<li class="people-location"><strong>Office Location:</strong> Tyler Hall 254</li>
```

Then we try to pull the string out an that is empty

```
puggioni_info.find_all('li', 'people-location')[0].string
```

Here, we could go to the documentation and look up what the object contains, but instead we can use object serialization.

We can use the python `__dict__` to inspect the object and see where it stored what we want.

```
puggioni_info.find_all('li','people-location')[0].__dict__
```

```
{'parser_class': bs4.BeautifulSoup,
'name': 'li',
'namespace': None,
'_namespaces': {},
'prefix': None,
'sourceline': 372,
'sourcepos': 303,
'known_xml': False,
'attrs': {'class': ['people-location']},
'contents': [<strong>Office Location:</strong>, ' Tyler Hall 254'],
'parent': <ul class="people-list">
<li class="people-title">Associate Professor | Chair</li> <li class="people-department">Statistics</li> <li class="people-department">Mathematics</li>
</ul>,
'previous_element': ' ',
'next_element': <strong>Office Location:</strong>,
'next_sibling': '\n',
'previous_sibling': ' ',
'hidden': False,
'can_be_empty_element': False,
'cdata_list_attributes': {'*': ['class', 'accesskey', 'dropzone'],
'a': ['rel', 'rev'],
'link': ['rel', 'rev'],
'td': ['headers'],
'th': ['headers'],
'form': ['accept-charset'],
'object': ['archive'],
'area': ['rel'],
'icon': ['sizes'],
'iframe': ['sandbox'],
'output': ['for']},
'preserve_whitespace_tags': {'pre', 'textarea'},
'interesting_string_types': (bs4.element.NavigableString, bs4.element.CData)}
```

we see its the second element in a list in the `'content'` value

```
puggioni_info.find_all('li','people-location')[0].contents[1]
```

```
' Tyler Hall 254'
```

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 added the `try` and `except` to handle when there is no office location. This is something in practice you would often think to do due an error.

Here we check the `info` and we can see how it is.

```
css_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27 entries, 0 to 26
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   name        27 non-null    object  
 1   title       27 non-null    object  
 2   email       27 non-null    object  
 3   discipline  27 non-null    object  
 4   office      25 non-null    object  
dtypes: object(5)
memory usage: 1.2+ KB
```

We can also, finally save out our ready dataset:

```
css_df.to_csv('css_faculty.csv')
```

## 9.6. Questions after class

### 9.6.1. what does .a do?

it gives the first instance of the `<a>` tag

### 9.6.2. is it worth it to try and web scrape a page that is poorly written?

If it is important information. In these cases, you might have to do more manual parsing or even some manual fixes.

For this class, no.

### 9.6.3. Is there a way to check robots.txt through BeautifulSoup or must that be done manually in a browser?

it could maybe be read programmaticlaly, but it doesn't necessarily save time to do it that way.

### 9.6.4. What else can I do with inspect?

It lets you view the code. It's most often used to debug websites.

### 9.6.5. when web scraping if the html is not set up well is it possible to change the html

[Skip to main content](#)

Technically you could manually edit a copy of it.

## 9.6.6. Are there instances where you can get data from websites that are not in tabular form?

Web scraping is for when the website is not in tabular form. It should be structured, but the structure does not need to come from a single page. It could be that there are many pages structured similarly and you build most of the columns from the other pages, not the starting page.

For example from the [teams page of the nba](#) you can get to a page with info about each team that includes all time records and the current rosters. On these individual pages, most info is an actual table, so you can use `pd.read_html` for those, but the crawling part from the first page would count.

## 9.6.7. A source table would be the people's page on the URI website, but when you click on their individual names does that count as another source table?

Not as we did above because we combined the data by adding another column. If you built a whole table on each of the sub-pages it would count.

## 9.7. Portfolio Question

### 9.7.1. I guess how to further edit the submission\_1.intro. I know about the chapters you gotta add, but what else? Is submission\_1.intro the only file you gotta edit?

Yes you edit that file and the `_toc.yml`. There are instructions on the [portfolio](#) page

There are also [formatting tips](#) and ideas

### 9.7.2. for portfolio one, can we submit whatever we want? like as little OR as much?

Yes, exactly!

However, I do want to **really** encourage you to submit whatever you are thinking of, even if it is not as complete as you want. If you submit, you will get feedback even if you do not earn all of the achievements you try. That will prepare you for the next one.

## 10. 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

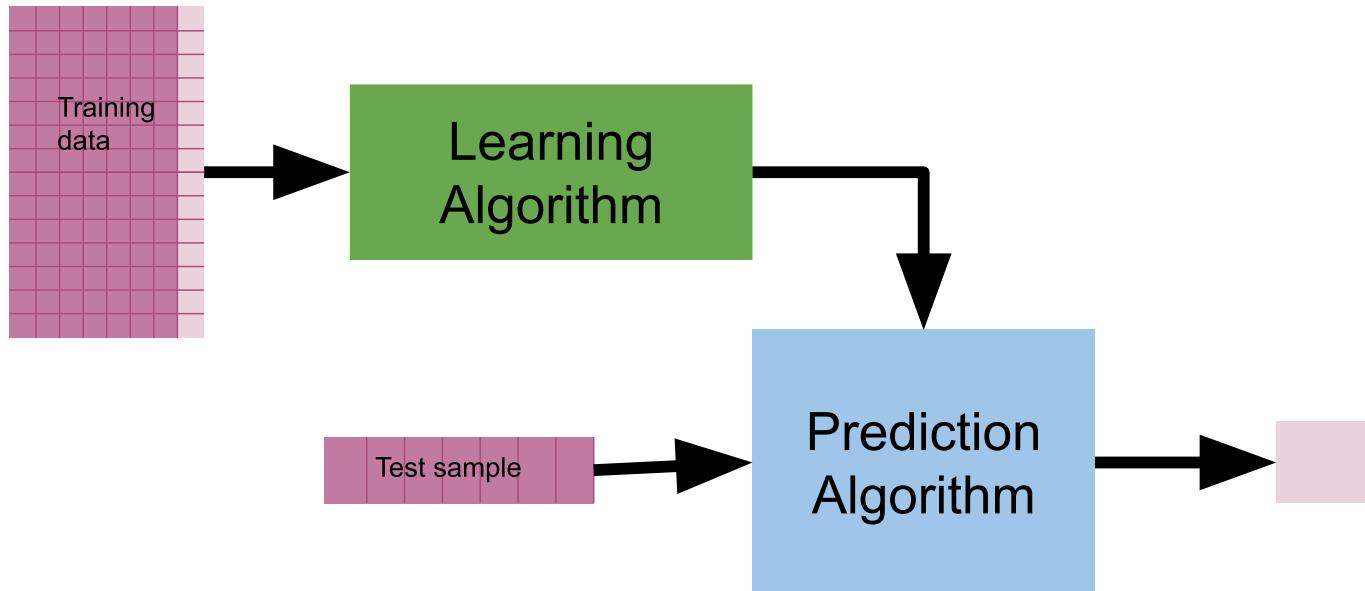
## 10.1. What is a Machine Learning Algorithm?

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

Each of these terms does have slightly different meanings and usage, but sometimes they're used close to exchangeably.

## 10.2. How can we tell if ML is working?

We measure the performance of the prediction algorithm, to determine if the learning algorithm worked.

## 10.3. Replicating the COMPAS Audit

We are going to replicate the audit from ProPublica Machine Bias

### 10.3.1. 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.

### 10.3.2. 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).

- `priors_count`: number of prior charges
- `days_b_screening_arrest`: number of days between charge date and arrest where defendant was screened for compas score
- `decile_score`: COMPAS score from 1 to 10 (low risk to high risk)
- `is_recid`: if the defendant recidivized
- `two_year_recid`: if the defendant within two years
- `c_jail_in`: date defendant was imprisoned
- `c_jail_out`: date defendant was released from jail
- `length_of_stay`: length of jail stay

```
compas_clean_url = 'https://raw.githubusercontent.com/ml4sts/outreach-compas/main/data/compas_c.csv'
compas_df = pd.read_csv(compas_clean_url)
compas_df.head()
```

	<code>id</code>	<code>age</code>	<code>c_charge_degree</code>	<code>race</code>	<code>age_cat</code>	<code>score_text</code>	<code>sex</code>	<code>priors_count</code>	<code>days_b_screening_arrest</code>	<code>decile_score</code>
0	3	34		F African-American	25 - 45	Low	Male	0		-1.0
1	4	24		F African-American	Less than 25	Low	Male	4		-1.0
2	8	41		F Caucasian	25 - 45	Medium	Male	14		-1.0
3	10	39		M Caucasian	25 - 45	Low	Female	0		-1.0
4	14	27		F Caucasian	25 - 45	Low	Male	0		-1.0

## 10.4. One-hot Encoding

We will audit first to see how good the algorithm is by treating the predictions as either high or not high. One way we can get to that point is to transform the `score_text` column from one column with three values, to 3 binary columns.

First lets understand the `score_text`

```
compas_df.groupby('score_text')[['decile_score']].agg(['min', 'max'])
```

		min	max
	score_text		
	High	8	10
	Low	1	4
	Medium	5	7

[Skip to main content](#)

```
pd.get_dummies(compas_df['score_text'])
```

	High	Low	Medium
0	False	True	False
1	False	True	False
2	False	False	True
3	False	True	False
4	False	True	False
...	...	...	...
5273	False	True	False
5274	True	False	False
5275	False	False	True
5276	False	True	False
5277	False	True	False

5278 rows × 3 columns

Since we actually want all of the columns with that one expanded, we will apply it a different way to save it to a variable.

```
compas_df_onehot = pd.get_dummies(compas_df, columns=['score_text'])
```

We will also audit with respect to a second threshold.

```
compas_df_onehot['score_text_medhigh'] = (compas_df_onehot['score_text_High'] +  
                                         compas_df_oneho['score_text_Medium'])
```

```
-----  
NameError                                                 Traceback (most recent call last)  
Cell In[6], line 2  
      1 compas_df_onehot['score_text_medhigh'] = (compas_df_onehot['score_text_High'] +  
----> 2                                         compas_df_oneho['score_text_Medium'])  
  
NameError: name 'compas_df_oneho' is not defined
```

```
compas_df_onehot.head(10)
```

	<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>0</b>	3	34		F African-American	25 - 45	Male	0	-1.0	3
<b>1</b>	4	24		F African-American	Less than 25	Male	4	-1.0	4
<b>2</b>	8	41		F Caucasian	25 - 45	Male	14	-1.0	6
<b>3</b>	10	39		M Caucasian	25 - 45	Female	0	-1.0	1
<b>4</b>	14	27		F Caucasian	25 - 45	Male	0	-1.0	4
<b>5</b>	15	23		M African-American	Less than 25	Male	3	0.0	6
<b>6</b>	16	37		M Caucasian	25 - 45	Female	0	0.0	1
<b>7</b>	18	41		F African-American	25 - 45	Male	0	-1.0	4
<b>8</b>	19	47		F Caucasian	Greater than 45	Female	1	-20.0	1
<b>9</b>	20	31		F African-American	25 - 45	Male	7	22.0	3

## 10.5. Sklearn Performance metrics

The first thing we usually check is the accuracy: the percentage of all samples that are correct.

```
metrics.accuracy_score(compas_df_onehot['two_year_recid'], compas_df_onehot['score_text_High'])
```

```
0.6288366805608185
```

```
metrics.accuracy_score(compas_df_onehot['two_year_recid'], compas_df_onehot['score_text_medhigh'])
```

```

-----
KeyError Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in
 3652     try:
-> 3653         return self._engine.get_loc(casted_key)
 3654     except KeyError as err:
 3655
 3656     File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc
 3657
 3658     File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pandas._libs.index.IndexEngine.get_loc
 3659     File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
 3660     File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()
 3661
 3662 KeyError: 'score_text_medhigh'

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

KeyError Traceback (most recent call last)
Cell In[9], line 1
----> 1 metrics.accuracy_score(compas_df_onehot['two_year_recid'], compas_df_onehot['score_text_medhigh'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame.get_loc(self, key)
 3759     if self.columns.nlevels > 1:
 3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
 3762     if is_integer(indexer):
 3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655, in
 3653     return self._engine.get_loc(casted_key)
 3654     except KeyError as err:
-> 3655         raise KeyError(key) from err
 3656     except TypeError:
 3657         # If we have a listlike key, _check_indexing_error will raise
 3658         # InvalidIndexError. Otherwise we fall through and re-raise
 3659         # the TypeError.
 3660         self._check_indexing_error(key)

KeyError: 'score_text_medhigh'

```

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 *negative prediction*.

[docs](#)

```
metrics.confusion_matrix(compas_df_onehot['two_year_recid'], compas_df_onehot['score_text_medhigh'])
```

Not

the w  
matrix

```

-----
KeyError Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in 
3652     try:
-> 3653         return self._engine.get_loc(casted_key)
3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc
File /opt/hostedtoolcache/Python/3.8.18/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_medhigh'

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

KeyError Traceback (most recent call last)
Cell In[10], line 1
----> 1 metrics.confusion_matrix(compas_df_onehot['two_year_recid'], compas_df_onehot['score_text_medhigh'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame._get_item_multilevel
3759     if self.columns.nlevels > 1:
3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
3762     if is_integer(indexer):
3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655, in 
3653     return self._engine.get_loc(casted_key)
3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
3656 except TypeError:
3657     # If we have a listlike key, _check_indexing_error will raise
3658     # InvalidIndexError. Otherwise we fall through and re-raise
3659     # the TypeError.
3660     self._check_indexing_error(key)

KeyError: 'score_text_medhigh'

```

(1872+1602)/len(compas\_df\_onehot)

0.6582038651004168

### **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](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

...and you can see that our accuracy score and model is unchanged.

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.

## 10.6. Precision and Recall

Two common ones in CS are recall and precision.

Recall is:

$$R = \frac{C_{1,1}}{C_{1,0} + C_{1,1}}$$

```
metrics.recall_score(compas_df_onehot['two_year_recid'], compas_df_onehot['score_text_medhigh'])
```

```
-----
KeyError                                                 Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in 
3652     try:
-> 3653         return self._engine.get_loc(casted_key)
 3654     except KeyError as err:
 3655         raise KeyError(key) from err
 3656     except TypeError:
 3657         # If we have a listlike key, _check_indexing_error will raise
 3658         # InvalidIndexError. Otherwise we fall through and re-raise
 3659         # the TypeError.
 3660         self._check_indexing_error(key)

KeyError: 'score_text_medhigh'
```

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

```
KeyError                                                 Traceback (most recent call last)
Cell In[12], line 1
----> 1 metrics.recall_score(compas_df_onehot['two_year_recid'], compas_df_onehot['score_text_medhigh'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame.get_loc(self, key)
 3759     if self.columns.nlevels > 1:
 3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
 3762     if is_integer(indexer):
 3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655, in 
 3653     return self._engine.get_loc(casted_key)
 3654     except KeyError as err:
-> 3655         raise KeyError(key) from err
 3656     except TypeError:
 3657         # If we have a listlike key, _check_indexing_error will raise
 3658         # InvalidIndexError. Otherwise we fall through and re-raise
 3659         # the TypeError.
 3660         self._check_indexing_error(key)

KeyError: 'score text medhigh'
```

[Skip to main content](#)

That is, among the truly positive class how many were correctly predicted? In COMPAS, it's the percentage of the re-arrested people who got a high score.

$$\text{Precision is } P = \frac{C_{1,1}}{C_{0,1} + C_{1,1}}$$

```
metrics.precision_score(compas_df_onehot['two_year_recid'], compas_df_onehot['score_text_medhigh'])
```

```
-----
KeyError                                 Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in
3652     try:
-> 3653         return self._engine.get_loc(casted_key)
3654     except KeyError as err:
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas.
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:176, in pandas.
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_medhigh'
```

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

```
KeyError                                 Traceback (most recent call last)
Cell In[13], line 1
-> 1 metrics.precision_score(compas_df_onehot['two_year_recid'], compas_df_onehot['score_text_medhigh'])

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame.
3759     if self.columns.nlevels > 1:
3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
3762     if is_integer(indexer):
3763         indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3655, in
3653     return self._engine.get_loc(casted_key)
3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
3656 except TypeError:
3657     # If we have a listlike key, _check_indexing_error will raise
3658     # InvalidIndexError. Otherwise we fall through and re-raise
3659     # the TypeError.
3660     self._check_indexing_error(key)

KeyError: 'score_text_medhigh'
```

```
compas_df_onehot.columns
```

```
Index(['id', 'age', 'c_charge_degree', 'race', 'age_cat', 'sex',
       'priors_count', 'days_b_screening_arrest', 'decile_score', 'is_recid',
       'two_year_recid', 'c_jail_in', 'c_jail_out', 'length_of_stay',
       'score_text_High', 'score_text_Low', 'score_text_Medium'],
      dtype='object')
```

## 10.7. Per Group Scores

[Skip to main content](#)

```
acc_fx = lambda d: metrics.accuracy_score(d['two_year_recid'],d['score_text_medhigh'])
compas_df_onehot.groupby('race').apply(acc_fx)
```

```
-----  
KeyError Traceback (most recent call last)  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in  
3652     try:  
-> 3653         return self._engine.get_loc(casted_key)  
3654     except KeyError as err:  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc  
  
File /opt/hostedtoolcache/Python/3.8.18/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_medhigh'  
  
The above exception was the direct cause of the following exception:  
  
KeyError Traceback (most recent call last)  
Cell In[15], line 2  
  1 acc_fx = lambda d: metrics.accuracy_score(d['two_year_recid'],d['score_text_medhigh'])  
-> 2 compas_df_onehot.groupby('race').apply(acc_fx)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1353,  
1351     with option_context("mode.chained_assignment", None):  
1352         try:  
-> 1353             result = self._python_apply_general(f, self._selected_obj)  
1354         except TypeError:  
1355             # gh-20949  
1356             # try again, with .apply acting as a filtering  
(...)  
1360             # fails on *some* columns, e.g. a numeric operation  
1361             # on a string grouper column  
1363         return self._python_apply_general(f, self._obj_with_exclusions)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1402,  
1367     @final  
1368     def _python_apply_general(  
1369         self,  
(...)  
1374         is_agg: bool = False,  
1375     ) -> NDFrameT:  
1376         """  
1377             Apply function f in python space  
1378         (...)  
1400             data after applying f  
1401         """  
-> 1402             values, mutated = self.grouper.apply(f, data, self.axis)  
1403             if not _indexed_same is None:  
1404                 not_indexed_same = mutated  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:767, in BaseGrouper._python_apply_general  
765     # group might be modified  
766     group_axes = group.axes  
-> 767     res = f(group)  
768     if not mutated and not _is_indexed_like(res, group_axes, axis):  
769         mutated = True  
  
Cell In[15], line 1, in <lambda>(d)  
-> 1 acc_fx = lambda d: metrics.accuracy_score(d['two_year_recid'],d['score_text_medhigh'])  
  2 compas_df_onehot.groupby('race').apply(acc_fx)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame.get_loc  
3759     if self.columns.nlevels > 1:  
3760         return self._getitem_multilevel(key)  
-> 3761     indexer = self.columns.get_loc(key)  
3762     if is_integer(indexer):  
3763         indexer = [indexer]
```

```
3653     return self._engine.get_loc(casted_key)
3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
3656 except TypeError:
3657     # If we have a listlike key, _check_indexing_error will raise
3658     # InvalidIndexError. Otherwise we fall through and re-raise
3659     # the TypeError.
3660     self._check_indexing_error(key)
```

```
KeyError: 'score_text_medhigh'
```

That lambda + apply is equivalent to:

```
race_acc = []
for race, rdf in compas_race:
    acc = skmetrics.accuracy_score(rdf['two_year_recid'],
                                    rdf['score_text_MedHigh'])
    race_acc.append([race, acc])
pd.DataFrame(race_acc, columns=['race', 'accuracy'])
```

```
-----
NameError                                 Traceback (most recent call last)
Cell In[16], line 2
      1 race_acc = []
----> 2 for race, rdf in compas_race:
      3     acc = skmetrics.accuracy_score(rdf['two_year_recid'],
      4                                     rdf['score_text_MedHigh'])
      5     race_acc.append([race, acc])

NameError: name 'compas_race' is not defined
```

then we can do the same thing for recall and precision.

```
recall_fx = lambda d: metrics.recall_score(d['two_year_recid'], d['score_text_medhigh'])
compas_df_onehot.groupby('race').apply(recall_fx)
```

```

-----
KeyError                                         Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in 
3652     try:
-> 3653         return self._engine.get_loc(casted_key)
3654     except KeyError as err:

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc

File /opt/hostedtoolcache/Python/3.8.18/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_medhigh'

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

KeyError                                         Traceback (most recent call last)
Cell In[17], line 2
    1 recall_fx = lambda d: metrics.recall_score(d['two_year_recid'],d['score_text_medhigh'])
-> 2 compas_df_onehot.groupby('race').apply(recall_fx)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1353,
1351     with option_context("mode.chained_assignment", None):
1352         try:
-> 1353             result = self._python_apply_general(f, self._selected_obj)
1354         except TypeError:
1355             # gh-20949
1356             # try again, with .apply acting as a filtering
1357             ...
1358             # fails on *some* columns, e.g. a numeric operation
1359             # on a string grouper column
1360             return self._python_apply_general(f, self._obj_with_exclusions)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1402,
1367 @final
1368 def _python_apply_general(
1369     self,
1370     ...
1371     is_agg: bool = False,
1372     ) -> NDFrameT:
1373     """
1374         Apply function f in python space
1375     ...
1376     data after applying f
1377     """
1378     values, mutated = self.grouper.apply(f, data, self.axis)
1379     if not _indexed_same is None:
1380         not_indexed_same = mutated

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:767, in BaseGrouper._python_apply_general
765     # group might be modified
766     group_axes = group.axes
-> 767     res = f(group)
768     if not mutated and not _is_indexed_like(res, group_axes, axis):
769         mutated = True

Cell In[17], line 1, in <lambda>(d)
-> 1 recall_fx = lambda d: metrics.recall_score(d['two_year_recid'],d['score_text_medhigh'])
2 compas_df_onehot.groupby('race').apply(recall_fx)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame.getitem_multilevel
3759     if self.columns.nlevels > 1:
3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
3762     if is_integer(indexer):
3763         indexer = [indexer]

```

[Skip to main content](#)

```
3653     return self._engine.get_loc(castede_key)
3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
3656 except TypeError:
3657     # If we have a listlike key, _check_indexing_error will raise
3658     # InvalidIndexError. Otherwise we fall through and re-raise
3659     # the TypeError.
3660     self._check_indexing_error(key)
```

KeyError: 'score\_text\_medhigh'

```
precision_fx = lambda d: metrics.precision_score(d['two_year_recid'],d['score_text_medhigh'])
compas_df_onehot.groupby('race').apply(precision_fx)
```

```
-----  
KeyError Traceback (most recent call last)  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in  
3652     try:  
-> 3653         return self._engine.get_loc(casted_key)  
3654     except KeyError as err:  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc  
  
File /opt/hostedtoolcache/Python/3.8.18/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_medhigh'  
  
The above exception was the direct cause of the following exception:  
  
KeyError Traceback (most recent call last)  
Cell In[18], line 2  
  1 precision_fx = lambda d: metrics.precision_score(d['two_year_recid'],d['score_text_medhigh'])  
-> 2 compas_df_onehot.groupby('race').apply(precision_fx)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1353,  
1351     with option_context("mode.chained_assignment", None):  
1352         try:  
-> 1353             result = self._python_apply_general(f, self._selected_obj)  
1354         except TypeError:  
1355             # gh-20949  
1356             # try again, with .apply acting as a filtering  
(...)  
1360             # fails on *some* columns, e.g. a numeric operation  
1361             # on a string grouper column  
1363         return self._python_apply_general(f, self._obj_with_exclusions)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1402,  
1367     @final  
1368     def _python_apply_general(  
1369         self,  
(...)  
1374         is_agg: bool = False,  
1375     ) -> NDFrameT:  
1376         """  
1377             Apply function f in python space  
1378         (...)  
1400             data after applying f  
1401         """  
-> 1402             values, mutated = self.grouper.apply(f, data, self.axis)  
1403             if not _indexed_same is None:  
1404                 not_indexed_same = mutated  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:767, in BaseGrouper._grouper_groupby.  
765     # group might be modified  
766     group_axes = group.axes  
-> 767     res = f(group)  
768     if not mutated and not _is_indexed_like(res, group_axes, axis):  
769         mutated = True  
  
Cell In[18], line 1, in <lambda>(d)  
-> 1     precision_fx = lambda d: metrics.precision_score(d['two_year_recid'],d['score_text_medhigh'])  
  2     compas_df_onehot.groupby('race').apply(precision_fx)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame.get_loc  
3759     if self.columns.nlevels > 1:  
3760         return self._getitem_multilevel(key)  
-> 3761     indexer = self.columns.get_loc(key)  
3762     if is_integer(indexer):  
3763         indexer = [indexer]
```

[Skip to main content](#)

```
3653     return self._engine.get_loc(casted_key)
3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
3656 except TypeError:
3657     # If we have a listlike key, _check_indexing_error will raise
3658     # InvalidIndexError. Otherwise we fall through and re-raise
3659     # the TypeError.
3660     self._check_indexing_error(key)

KeyError: 'score_text_medhigh'
```

The recall tells us that the model has very different impact on people. On the other hand the precision tells us the scores mean about the same thing for Black and White people.

Researchers established that these are mutually exclusive, provably. We cannot have both, so it is very important to think about what the performance metrics mean and how your algorithm will be used in order to choose how to prepare a model. We will train models starting next week, but knowing these goals in advance is essential.

Importantly, this is not a statistical, computational choice that data can answer for us. This is about *human* values (and to some extent the law; certain domains have legal protections that require a specific condition).

The Fair Machine Learning book's classification Chapter has a section on relationships between criteria with the proofs.

Looking at this gives a larger seeming difference to make it more clear, the error rate is almost twice as high.

```
1- compas_df_onehot.groupby('race').apply(recall_fx)
```

```

-----
KeyError                                         Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in 
3652     try:
-> 3653         return self._engine.get_loc(casted_key)
3654     except KeyError as err:
3655         raise KeyError(f"Label '{key}' not found in {self._info_name}").with_traceback(err.__traceback__)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc
3656     if key not in self._data:
3657         if self._na_index is None:
3658             if key == self._na_value:
3659                 return self._na_index
3660             else:
3661                 raise KeyError(f"Label '{key}' not found in {self._info_name}")
3662         else:
3663             if key == self._na_value:
3664                 return self._na_index
3665             else:
3666                 return self._na_index.get_loc(key)
3667
3668     if self._na_index is not None:
3669         if key == self._na_value:
3670             return self._na_index
3671         else:
3672             return self._data.get_loc(key)
3673
3674     return self._data.get_loc(key)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()
3675     return _PyDict_GetItemImpl(self, key, &found)
3676     Py_ssize_t idx = _PyDict_HashKey(key);
3677     PyObject **slot = &self->table[idx];
3678     if (slot != NULL) {
3679         PyObject **oldslot = slot;
3680         if (_PyDict_NeedRehash(self, idx)) {
3681             _PyDict_RehashEntries(self);
3682             slot = &self->table[idx];
3683         }
3684         if (*slot == slot) {
3685             if (_PyDict_HasPair(slot, key, &found)) {
3686                 return slot;
3687             }
3688         }
3689     }
3690     if (!found) {
3691         PyErr_SetKeyError(key);
3692         return NULL;
3693     }
3694     return slot;

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()
3695     return _PyDict_GetItemImpl(self, key, &found)
3696     Py_ssize_t idx = _PyDict_HashKey(key);
3697     PyObject **slot = &self->table[idx];
3698     if (slot != NULL) {
3699         PyObject **oldslot = slot;
3700         if (_PyDict_NeedRehash(self, idx)) {
3701             _PyDict_RehashEntries(self);
3702             slot = &self->table[idx];
3703         }
3704         if (*slot == slot) {
3705             if (_PyDict_HasPair(slot, key, &found)) {
3706                 return slot;
3707             }
3708         }
3709     }
3710     if (!found) {
3711         PyErr_SetKeyError(key);
3712         return NULL;
3713     }
3714     return slot;

KeyError: 'score_text_medhigh'

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

KeyError                                         Traceback (most recent call last)
Cell In[19], line 1
----> 1 1- compas_df_onehot.groupby('race').apply(recall_fx)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1353,
1351     with option_context("mode.chained_assignment", None):
1352         try:
-> 1353             result = self._python_apply_general(f, self._selected_obj)
1354         except TypeError:
1355             # gh-20949
1356             # try again, with .apply acting as a filtering
1357             (...)

1358             # fails on *some* columns, e.g. a numeric operation
1359             # on a string grouper column
1360             return self._python_apply_general(f, self._obj_with_exclusions)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1402,
1367 @final
1368 def _python_apply_general(
1369     self,
1370     (...),
1371     is_agg: bool = False,
1372     ) -> NDFrameT:
1373     """
1374     Apply function f in python space
1375     (...),
1376     data after applying f
1377     """
1378     values, mutated = self.grouper.apply(f, data, self.axis)
1379     if not_indexed_same is None:
1380         not_indexed_same = mutated

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:767, in BaseGrouper._python_apply_general
765     # group might be modified
766     group_axes = group.axes
-> 767     res = f(group)
768     if not mutated and not _is_indexed_like(res, group_axes, axis):
769         mutated = True

Cell In[17], line 1, in <lambda>(d)
----> 1 recall_fx = lambda d: metrics.recall_score(d['two_year_recid'],d['score_text_medhigh'])
2 compas_df_onehot.groupby('race').apply(recall_fx)

File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame.get_loc
3759     if self.columns.nlevels > 1:
3760         return self._getitem_multilevel(key)
-> 3761     indexer = self.columns.get_loc(key)
3762     if is_integer(indexer):
3763         indexer = [indexer]

```

```
3654 except KeyError as err:  
-> 3655     raise KeyError(key) from err  
3656 except TypeError:  
3657     # If we have a listlike key, _check_indexing_error will raise  
3658     # InvalidIndexError. Otherwise we fall through and re-raise  
3659     # the TypeError.  
3660     self._check_indexing_error(key)
```

KeyError: 'score\_text\_medhigh'

```
1- compas_df_onehot.groupby('race').apply(precision_fx)
```

```
-----  
KeyError Traceback (most recent call last)  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/indexes/base.py:3653, in  
3652     try:  
-> 3653         return self._engine.get_loc(casted_key)  
3654     except KeyError as err:  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/_libs/index.pyx:147, in pandas._libs.index.IndexEngine.get_loc  
  
File /opt/hostedtoolcache/Python/3.8.18/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_medhigh'  
  
The above exception was the direct cause of the following exception:  
  
KeyError Traceback (most recent call last)  
Cell In[20], line 1  
-> 1 1- compas_df_onehot.groupby('race').apply(precision_fx)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1353,  
1351     with option_context("mode.chained_assignment", None):  
1352         try:  
-> 1353             result = self._python_apply_general(f, self._selected_obj)  
1354         except TypeError:  
1355             # gh-20949  
1356             # try again, with .apply acting as a filtering  
(...)  
1360             # fails on *some* columns, e.g. a numeric operation  
1361             # on a string grouper column  
1363             return self._python_apply_general(f, self._obj_with_exclusions)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/groupby.py:1402,  
1367     @final  
1368     def _python_apply_general(  
1369         self,  
(...)  
1374         is_agg: bool = False,  
1375     ) -> NDFrameT:  
1376         """  
1377             Apply function f in python space  
1378         (...)  
1400             data after applying f  
1401         """  
-> 1402         values, mutated = self.grouper.apply(f, data, self.axis)  
1403         if not_indexed_same is None:  
1404             not_indexed_same = mutated  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/groupby/ops.py:767, in BaseGrouper.apply  
765     # group might be modified  
766     group_axes = group.axes  
-> 767     res = f(group)  
768     if not mutated and not _is_indexed_like(res, group_axes, axis):  
769         mutated = True  
  
Cell In[18], line 1, in <lambda>(d)  
-> 1 precision_fx = lambda d: metrics.precision_score(d['two_year_recid'],d['score_text_medhigh'])  
2 compas_df_onehot.groupby('race').apply(precision_fx)  
  
File /opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pandas/core/frame.py:3761, in DataFrame.get_loc  
3759     if self.columns.nlevels > 1:  
3760         return self._getitem_multilevel(key)  
-> 3761     indexer = self.columns.get_loc(key)  
3762     if is_integer(indexer):  
3763         indexer = [indexer]
```

```
5054 except KeyError as err:  
-> 3655     raise KeyError(key) from err  
3656 except TypeError:  
3657     # If we have a listlike key, _check_indexing_error will raise  
3658     # InvalidIndexError. Otherwise we fall through and re-raise  
3659     # the TypeError.  
3660     self._check_indexing_error(key)  
  
KeyError: 'score_text_medhigh'
```

### ! Important

I mixed up precision with the probability of false alarm/sensitivity that would also show a high gap in class, that is why I was expecting there to be a gap. The precision scores are close. This value can be calculated from the confusion matrix, but is not implemented in sklearn.

### ! 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.

## 10.8. Questions After Class

### 10.8.1. Are there other methods that calculate the statistics we got through subtraction (1 - recall for example)?

If you look at the reference table, for each metric, 1-metric, is another metric. For example 1- accuracy is error rate. 1- recall is the false negative rate.

### 10.8.2. how to combat biased data

This is still a really open area of research and it is context dependent. The COMPAS audit showed that there are different ways that we can write equations to try to represent fairness and researchers proved that these cannot all be true at the same time.

In this case, I would actually not say that the "data is biased" because in statistics, biased data refers to a type of not correct measurement. This data reflects the way the actual **world** is biased.

### 10.8.3. I guess when should and shouldn't you use one-hot?

We use one hot encoding when we want to numerically represent categorical data and if we want to do something that is like binary for one of the values for a variable with more than 2 values.

### 10.8.4. What are some good intro resources you suggest for looking more into some of the math that make up these machine learning models that we will use in this class?

! Note

If you  
that is  
out to

For the purposes of this course, the `sklearn` documentation is a good starting point. It has the specific equations that are implemented in the package, which is relevant for some models that have multiple ways they can be implemented.

### 10.8.5. How does the confusion matrix work?

We will come back to this, but honestly the wikipedia articl and the sklearn docs are my go- to sources on this topic.

### 10.8.6. Should the actual number rating be taken into to account for things like confidence intervals to get a better analysis?

We will look at confidence intervals later, but in a strict sense, calcuating a confidence interval on a classifier accuracy is really tricky. We will do it in the way that many people do, but getting a proper set of error bounds on a classifier performance is not well defined.

## 11. 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.

### 11.1. What is a Model?

A model is a simplified representation of some part of the world. A famous quote about models is:

### 11.2. All models are wrong, but some are useful –George Box[^wiki]

### 11.3. 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](#)

[Skip to main content](#)

## 11.4. Models in Machine Learning

11.5. 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

$$y_i = f(\mathbf{x}_i)$$

## 11.6. $i$ would be the index of a DataFrame

## 11.7. 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

 flowchart for above definitions

### Further Reading

sklearn provides a popular flowchart for choosing a specific model

### 11.7.1. 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**

## 11.8. Machine Learning Pipeline

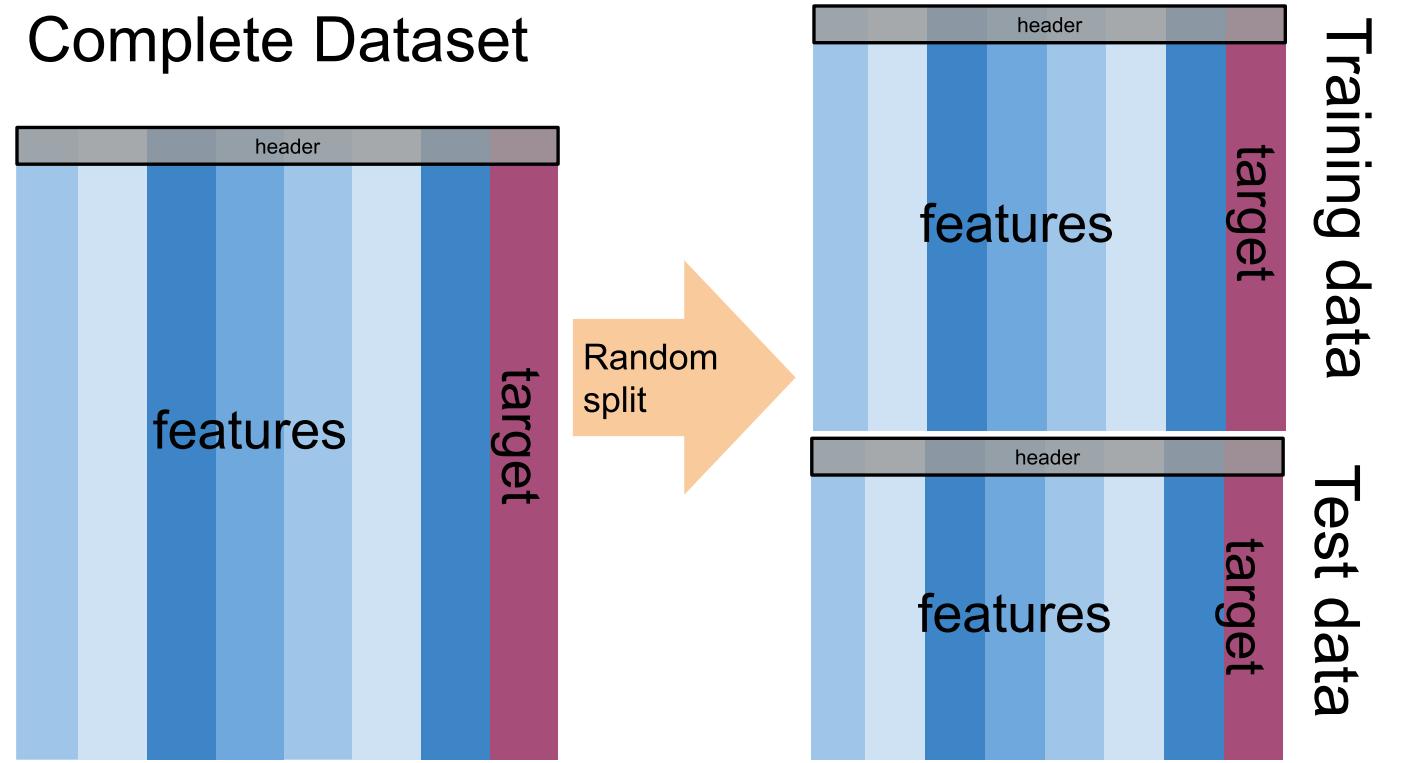
To do machine learning we start with **training data** which we put as input to the **learning algorithm**. A learning algorithm might be

[Skip to main content](#)

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 --> D[lalgo(learning algorithm)]; B --> E[palgo(prediction algorithm)]; C --> F[palgo]; F --> G[pred[[predictions]]]; G --> H[metrics{scoring function}]; H --> I[metrics]; I --> J[score[[scores/ performance metrics]]]
```

## 11.9. Let's Practice:

First machine learning model: Naive bayes

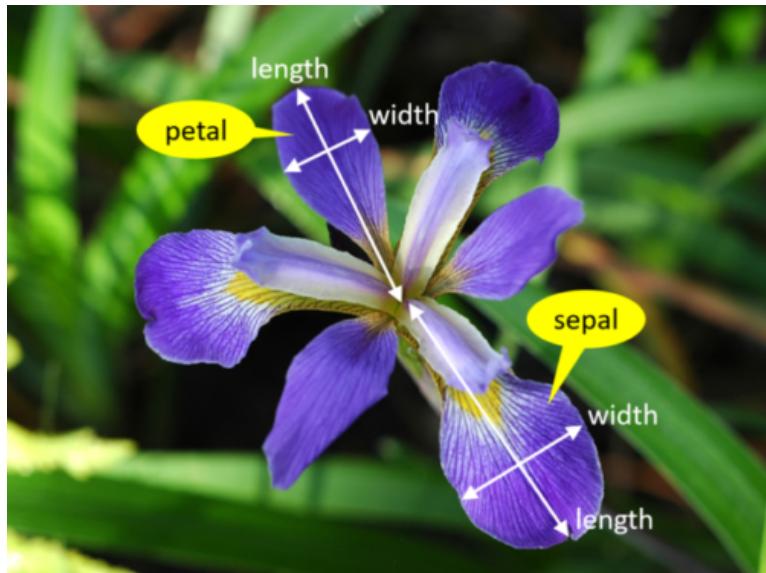
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, accuracy_score
iris_df = sns.load_dataset('iris')
```

To start we will look at the data

[Skip to main content](#)

```
iris_df.sample(5)
```

	sepal_length	sepal_width	petal_length	petal_width	species
110	6.5	3.2	5.1	2.0	virginica
54	6.5	2.8	4.6	1.5	versicolor
125	7.2	3.2	6.0	1.8	virginica
91	6.1	3.0	4.6	1.4	versicolor
90	5.5	2.6	4.4	1.2	versicolor



We're trying to build an automatic flower classifier that, for measurements of a new flower returns the predicted species. To do this, we have a DataFrame with columns for species, petal width, petal length, sepal length, and sepal width. The species is what type of flower it is the petal and sepal are parts of the flower.

#### **Note**

This cell is hidden because it is not necessary for the narrative structure of our analysis, but it was useful for creating the next cell

► Show code cell 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'
```

## 11.10. What does Naive Bayes do?

[Skip to main content](#)

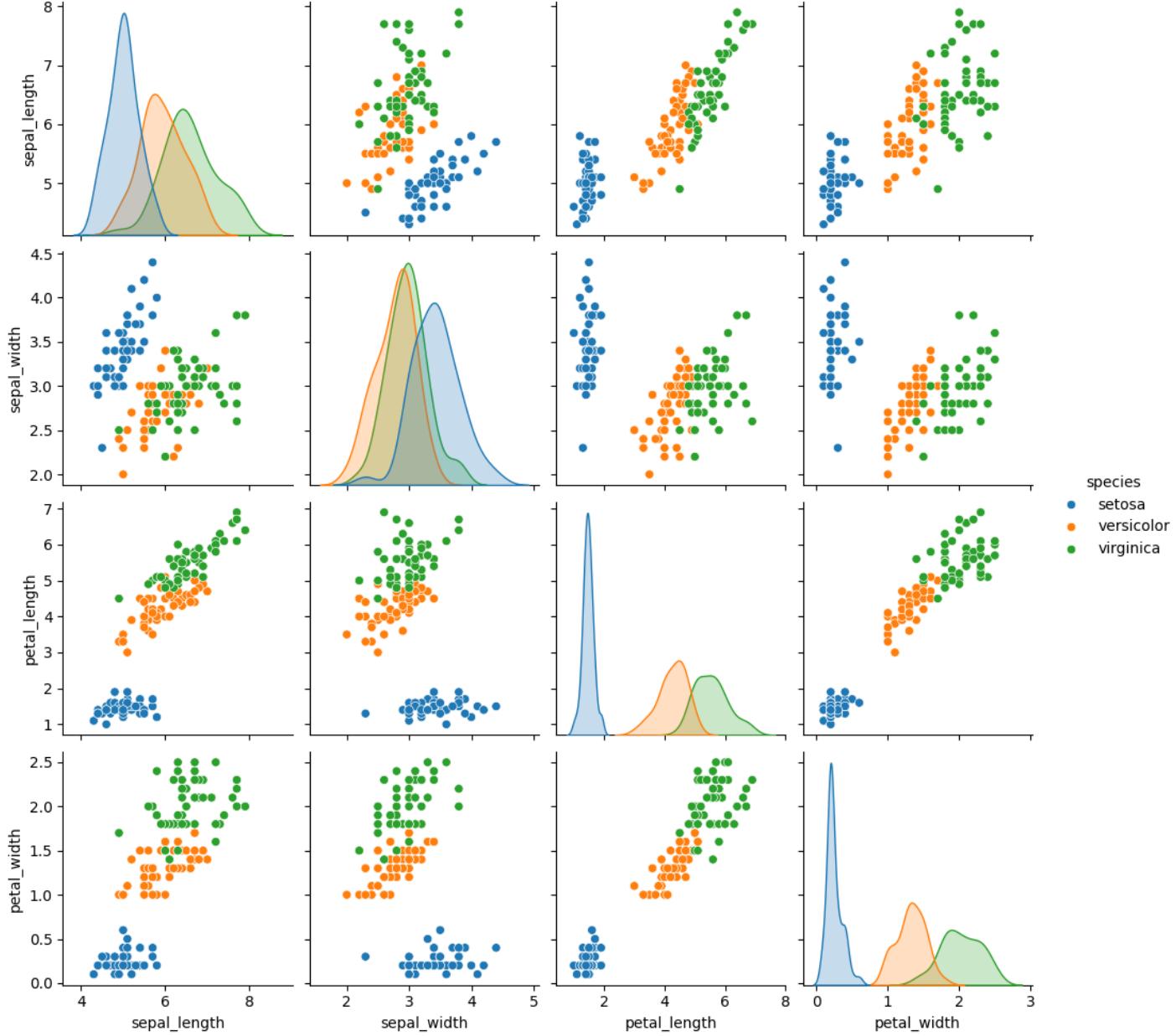
Naive = independent features Bayes = most probable

## Bayes Estimator

We can look at this data using a pair plot. It plots each pair of numerical variables in a grid of scatterplots and on the diagonal (where it would be a variable with itself) shows the distribution of that variable.

```
sns.pairplot(data=iris_df, hue=target_var)
```

```
<seaborn.axisgrid.PairGrid at 0x7fa0f0287790>
```



This data is reasonably **separable** because the different species (indicated with colors in the plot) do not overlap much. We see that the features are distributed sort of like a normal, or Gaussian, distribution. In 2D a Gaussian distribution is like a hill, so we expect to see more points near the center and fewer on the edge of circle-ish blobs. These blobs are slightly like ovals, but not too

[Skip to main content](#)

This means that the assumptions of the Gaussian Naive Bayes model are met well enough we can expect the classifier to do well.

## 11.11. Separating Training and Test Data

To do machine learning, we split the data both sample wise (rows if tidy) and variable-wise (columns if tidy). First, we'll designate the columns to use as features and as the target.

The features are the input that we wish to use to predict the target.

Next, we'll use a sklearn function to split the data randomly into test and train portions.

```
X_train, X_test, y_train, y_test = train_test_split(iris_df[feature_vars], iris_df[target_var], random_state=42)
```

This function returns multiple values, the docs say that it returns `twice as many` as it is passed. We passed two separate things, the features and the labels separated, so we get train and test each for both.

### Note

If you get different numbers for the index than I do here or run the train test split multiple times and see things change, you have a different random seed above.

```
X_train.head()
```

	sepal_length	sepal_width	petal_length	petal_width
61	5.9	3.0	4.2	1.5
92	5.8	2.6	4.0	1.2
112	6.8	3.0	5.5	2.1
2	4.7	3.2	1.3	0.2
141	6.9	3.1	5.1	2.3

We can see by default how many samples it puts the training set:

```
len(X_train)/len(iris_df)
```

```
0.7466666666666667
```

So by default we get a 75-25 split. But we can change it.

## 11.12. Instantiating our Model Object

Next we will instantiate the object for our `model`. In `sklearn` they call these objects `estimator`. All estimators have a similar usage. First we instantiate the object and set any `hyperparameters`.

Instantiating the object says we are assuming a particular type of model. In this case Gaussian Naive Bayes. This sets several assumptions in one form:

- we assume data are Gaussian (normally) distributed
- the features are uncorrelated/independent (Naive)
- the best way to predict is to find the highest probability (Bayes)

this is one example of a Bayes Estimator

```
gnb = GaussianNB()
```

At this point the object is not very interesting

```
gnb.__dict__
```

```
{'priors': None, 'var_smoothing': 1e-09}
```

The fit method uses the data to learn the model's parameters. In this case, a Gaussian distribution is characterized by a mean and variance; so the GNB classifier is characterized by one mean and one variance for each class (in 4d, like our data)

```
gnb.fit(X_train,y_train)
```

▼ GaussianNB  
GaussianNB()

The attributes of the estimator object (`gnb`) describe the data (eg the class list) and the model's parameters. The `theta_` ( $\theta$ ) represents the mean and the `sigma_` ( $\sigma$ ) represents the variance of the distributions.

```
gnb.__dict__
```

```
{'priors': None,
 'var_smoothing': 1e-09,
 'classes_': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'feature_names_in_': array(['sepal_length', 'sepal_width', 'petal_length', 'petal_width'],
   dtype=object),
 'n_features_in_': 4,
 'epsilon_': 3.2135586734693885e-09,
 'theta_': array([[4.9972973 , 3.38918919, 1.45405405, 0.24054054],
   [5.91764706, 2.75882353, 4.19117647, 1.30882353],
   [6.66341463, 2.9902439 , 5.58292683, 2.03902439]]),
 'var_': array([[0.12242513, 0.14474799, 0.01978087, 0.01159971],
   [0.2649827 , 0.11124568, 0.22139274, 0.0408045 ],
   [0.4071981 , 0.11453897, 0.30483046, 0.06579417]]),
 'class_count_': array([37., 34., 41.]),
 'class_prior_': array([0.33035714, 0.30357143, 0.36607143])}
```

Once we fit, we can predict

```
y_pred = gnb.predict(X_test)
```

[Skip to main content](#)

```
len(y_pred), len(X_test)
```

```
(38, 38)
```

We can evaluate this using the score we learned last week

```
accuracy_score(y_test,y_pred)
```

```
1.0
```

or by checking it manually:

```
sum(y_pred ==y_test)/len(y_test)
```

```
1.0
```

Estimator objects also have a score method. If the estimator is a classifier, that score is accuracy. We will see that for other types of estimators it is different types.

```
gnb.score(X_test,y_test)
```

```
1.0
```

## 11.13. Questions After Class

### Note

I used questions from previous semesters because few questions were asked.

### 11.13.1. Can you use machine learning for any type of data?

Yes the features for example could be an image instead of four numbers. It could also be text. The basic ideas are the same for more complex data, so we are going to spend a lot of time building your understanding of what ML *is* on simple data. Past students have successfully applied ML in more complex data after this course because once you have a good understanding of the core ideas, applying it to other forms of data is easier to learn on your own.

### 11.13.2. Can we check how well the model did using the y\_test df?

we could compare them directly or using `score` that does.

```
114    True
62    True
33    True
107   True
7     True
100   True
40    True
86    True
76    True
71    True
134   True
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
137   True
84    True
27    True
127   True
132   True
59    True
18    True
83    True
Name: species, dtype: bool
```

```
sum(y_pred == y_test)/len(y_test)
```

```
1.0
```

```
gnb.score(X_test,y_test)
```

```
1.0
```

We can also use any of the other metrics we saw, we'll practice more on Wednesday

### 11.13.3. I want to know more about the the test\_train\_split() function

the docs are a good place to start.

### 11.13.4. Could we use the comparison stuff we've been learning to test the ML algorithms

[Skip to main content](#)

Yes!!

### 11.13.5. How should we set the random set we had for x\_test to a specific group?

the `random_state` parameter will fix it.

### 11.13.6. Are there any good introductions to ScikitLearn that you are aware of?

Scikit Learn User Guide is the best one and they have a large example gallery.

## 12. Understanding Classification

Today we are going to work on understandign what happens when a classifier makes

### Important

You can provide feedback on the course so far

```
import pandas as pd
import seaborn as sns
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
sns.set_theme(palette='colorblind') # this improves contrast

from sklearn.metrics import confusion_matrix, classification_report
```

Our goal is to predict the species from the measurements. Set up the problem by putting the variables in the correct roles (features and target) so that train test split will work.

```
iris_df = sns.load_dataset('iris')
```

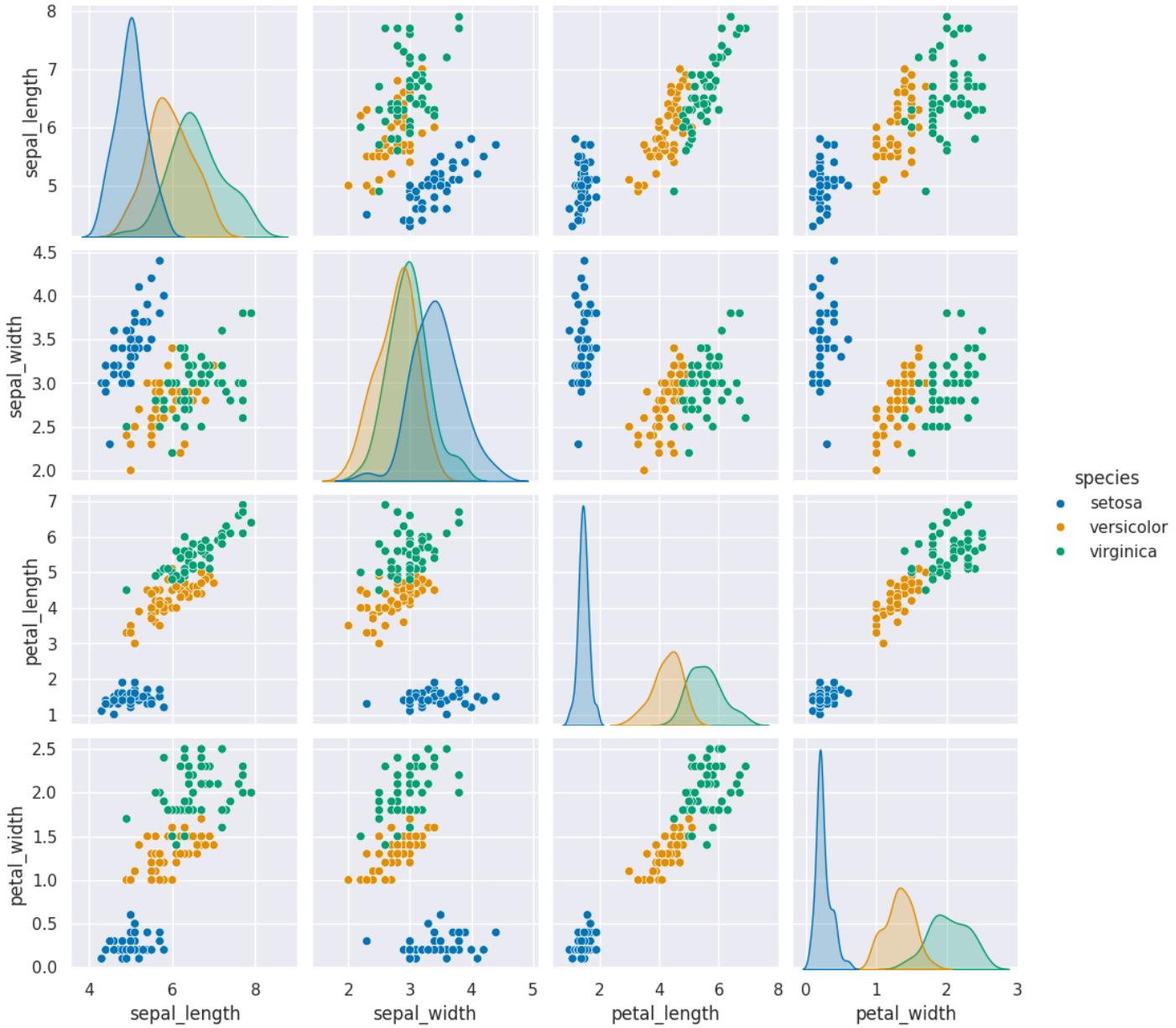
Today we will use a different random state, recall that the `random_state` is like a name for the sequence of psuedo random numbers that it will use.

```
# 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=3)
```

Again, we will plot the data with the target as the colors

```
sns.pairplot(data = iris_df, hue='species')
```

```
<seaborn.axisgrid.PairGrid at 0x7f8a0349c970>
```



We notice that this data meets the assumptions of our model:

- it is separable (for all classification)
- the classes are each one blob that is roughly a circle or oval (gaussian)
- the ovals are not too
- 

```
gnb = GaussianNB()
```

```
gnb.fit(X_train,y_train)
```

```
[... GaussianNB]
```

[Skip to main content](#)

```
y_pred = gnb.predict(X_test)
```

```
gnb.score(X_test,y_test)
```

```
0.9736842105263158
```

We can also get a report with a few metrics.

- Recall is the percent of each species that were predicted correctly.
- Precision is the percent of the ones predicted to be in a species that are truly that species.
- the F1 score is combination of the two

```
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	15
versicolor	1.00	0.92	0.96	12
virginica	0.92	1.00	0.96	11
accuracy			0.97	38
macro avg	0.97	0.97	0.97	38
weighted avg	0.98	0.97	0.97	38

```
y_test.value_counts()
```

```
species
setosa      15
versicolor   12
virginica    11
Name: count, dtype: int64
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[15,  0,  0],
       [ 0, 11,  1],
       [ 0,  0, 11]])
```

```
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_': 2.9996867028061227e-09,
'theta_': array([[0.24      , 5.04285714, 3.46285714, 1.46571429],
   [1.32631579, 5.89210526, 2.77894737, 4.23947368],
   [2.00769231, 6.54358974, 2.98461538, 5.53589744]]),
'var_': array([[0.01268572, 0.0767347 , 0.09604898, 0.02911021],
   [0.03878117, 0.21862189, 0.10376732, 0.21186288],
   [0.06635109, 0.39425378, 0.10540434, 0.29204471]]),
'class_count_': array([35., 38., 39.]),
'class_prior_': array([0.3125    , 0.33928571, 0.34821429])}
```

```
import numpy as np
```

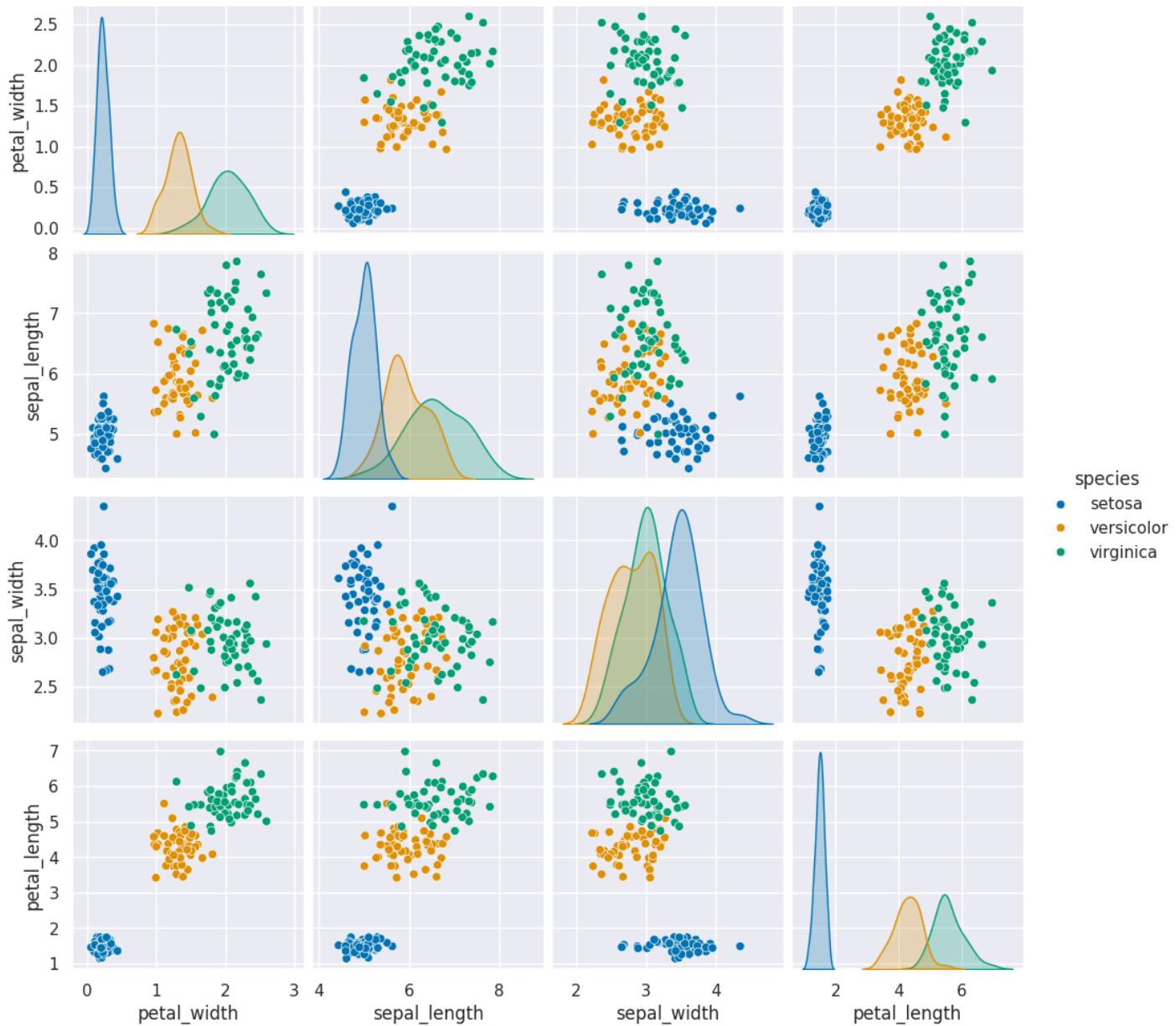
## 12.1. What does a generative model mean?

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.var_)`). We do this inside of a list comprehension and for each `th, sig` where `th` is from `gnb.theta_` and `sig` is from `gnb.var_` 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.

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(th, sig*np.eye(4), N)
                                      for th, sig in zip(gnb.theta_, gnb.var_)])
                      .T,
                      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 0x7f89fd82aa00>
```



## 12.2. How does it make the predictions?

```
gnb.predict_proba(X_test)
```

[Skip to main content](#)

```

array([[1.00000000e+000, 3.10850430e-018, 9.83936388e-027],
[1.00000000e+000, 2.17607686e-017, 6.11974996e-026],
[1.00000000e+000, 2.45846863e-014, 9.60562575e-023],
[1.00000000e+000, 4.29382017e-016, 8.65569962e-025],
[1.00000000e+000, 5.49915897e-016, 2.38180484e-023],
[1.99870101e-311, 1.36174299e-013, 1.00000000e+000],
[2.69395984e-069, 9.99960647e-001, 3.93530354e-005],
[1.00000000e+000, 1.40826499e-017, 5.89539964e-026],
[1.08156958e-173, 1.40618746e-003, 9.98593813e-001],
[2.90745994e-094, 9.75192625e-001, 2.48073750e-002],
[5.39591648e-072, 9.99825700e-001, 1.74300433e-004],
[1.00000000e+000, 5.38453222e-018, 1.35082184e-026],
[1.30426983e-066, 9.99973771e-001, 2.62288452e-005],
[6.11553492e-070, 9.99973058e-001, 2.69419127e-005],
[2.01494877e-214, 4.34577051e-009, 9.99999996e-001],
[1.00000000e+000, 5.75883870e-018, 2.19570953e-026],
[1.15708888e-120, 9.13375722e-001, 8.66242784e-002],
[1.35987974e-251, 2.26297317e-011, 1.00000000e+000],
[5.67885061e-148, 1.14141144e-002, 9.88585886e-001],
[1.00000000e+000, 1.36229423e-018, 5.09772462e-027],
[2.60919798e-128, 1.70228595e-001, 8.29771405e-001],
[2.54851174e-189, 2.95677415e-006, 9.99997043e-001],
[3.04997634e-180, 3.40898084e-007, 9.99999659e-001],
[2.00705701e-114, 8.76744267e-001, 1.23255733e-001],
[1.00000000e+000, 2.97690385e-014, 4.03753917e-022],
[2.46428706e-258, 9.61343301e-013, 1.00000000e+000],
[1.52504682e-153, 4.39107652e-001, 5.60892348e-001],
[8.50956394e-064, 9.99989920e-001, 1.00798872e-005],
[1.15904764e-034, 9.99999826e-001, 1.73844955e-007],
[5.34461569e-112, 7.42611536e-001, 2.57388464e-001],
[1.00000000e+000, 1.76204469e-018, 3.45582363e-026],
[1.00000000e+000, 1.05535449e-014, 7.37686742e-023],
[3.87070789e-184, 3.08105361e-006, 9.99996919e-001],
[2.82918552e-099, 9.88848742e-001, 1.11512583e-002],
[1.00000000e+000, 2.84646765e-017, 6.28162840e-026],
[1.00000000e+000, 7.38253736e-018, 3.74617670e-026],
[6.06665872e-137, 4.97058492e-002, 9.50294151e-001],
[1.00000000e+000, 1.30813624e-014, 6.57792025e-024]])

```

```

# 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()

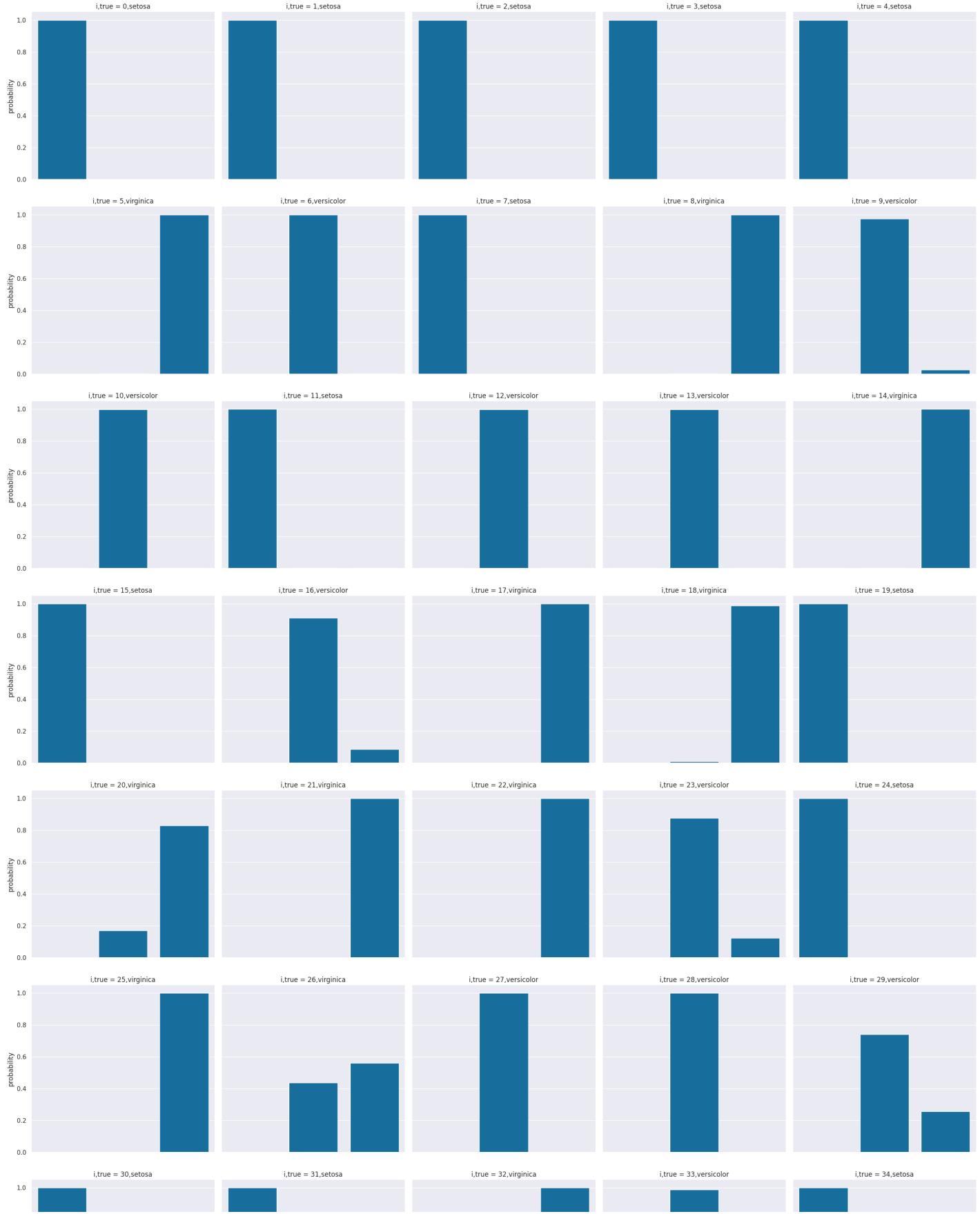
```

	index	predicted_species	true_species	i,pred	i,true	correct	species	probability
0	0	setosa	setosa	0,setosa	0, setosa	True	setosa	1.0
1	1	setosa	setosa	1, setosa	1, setosa	True	setosa	1.0
2	2	setosa	setosa	2, setosa	2, setosa	True	setosa	1.0
3	3	setosa	setosa	3, setosa	3, setosa	True	setosa	1.0
4	4	versicolor	versicolor	0,versicolor	0, versicolor	True	versicolor	1.0

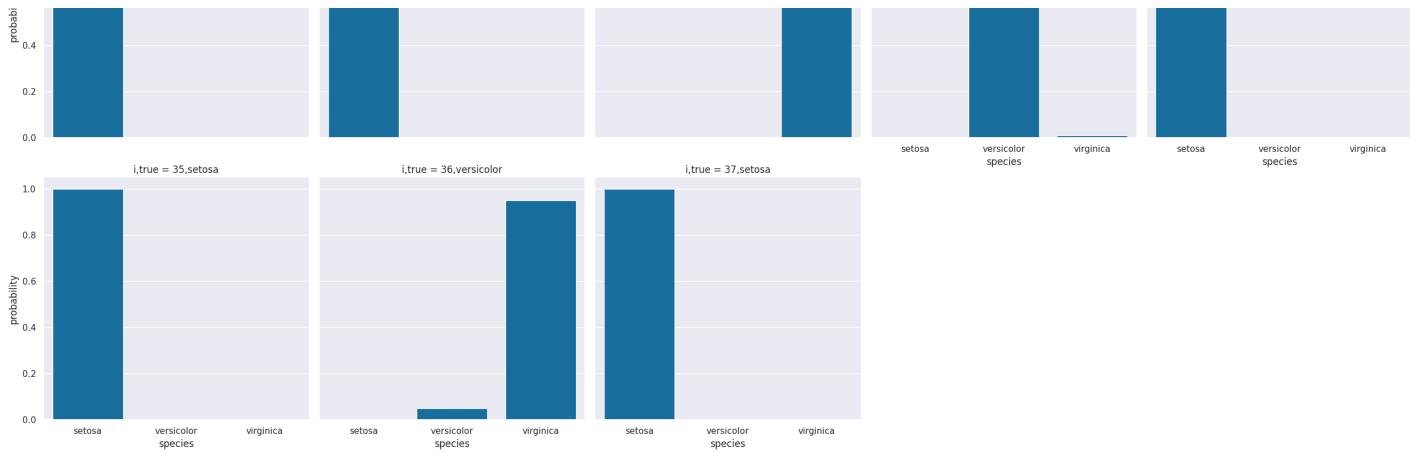
[Skip to main content](#)

```
# 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')
```

<seaborn.axisgrid.FacetGrid at 0x7f89fd8edfd0>



[Skip to main content](#)



## 12.3. What if the assumptions are not met?

Using a toy dataset here shows an easy to see challenge for the classifier that we have seen so far. Real datasets will be hard in different ways, and since they're higher dimensional, it's harder to visualize the cause.

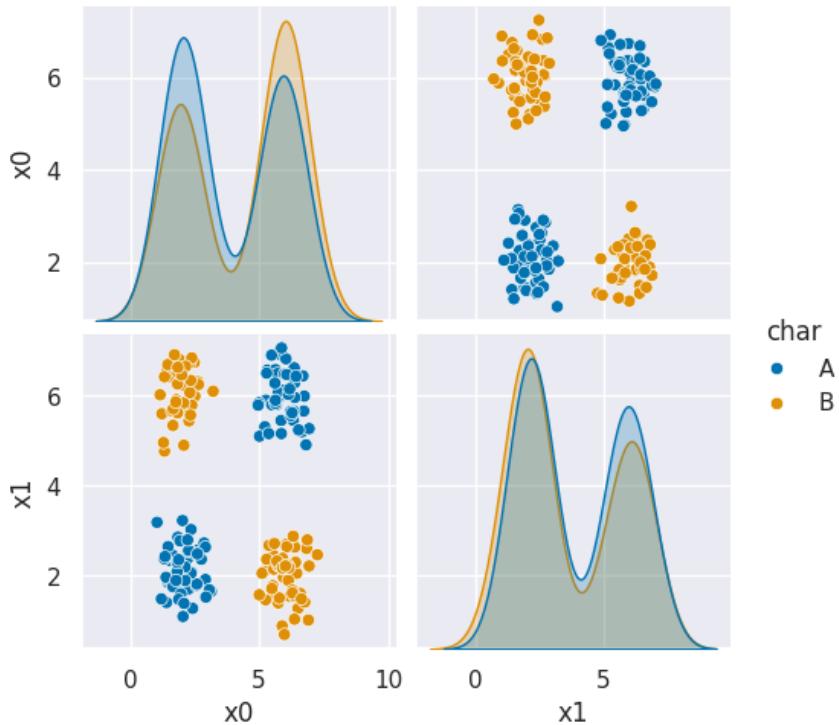
```
corner_data = 'https://raw.githubusercontent.com/rhodyprog4ds/06-naive-bayes/f425ba121cc0c4dd8bcaa7ebb2ff0b4
df6 = pd.read_csv(corner_data,usecols=[1,2,3])
```

```
df6.head(1)
```

	x0	x1	char
0	6.14	2.1	B

```
sns.pairplot(data=df6, hue='char', hue_order=['A', 'B'])
```

```
<seaborn.axisgrid.PairGrid at 0x7f89f327ed60>
```



As we can see in this dataset, these classes are quite separated.

```
x_train, x_test, y_train, y_test = train_test_split(df6[['x0', 'x1']], df6['char'], random_state=4)
```

```
gnb_corners = GaussianNB()
gnb_corners.fit(X_train,y_train)
gnb_corners.score(X_test, y_test)
```

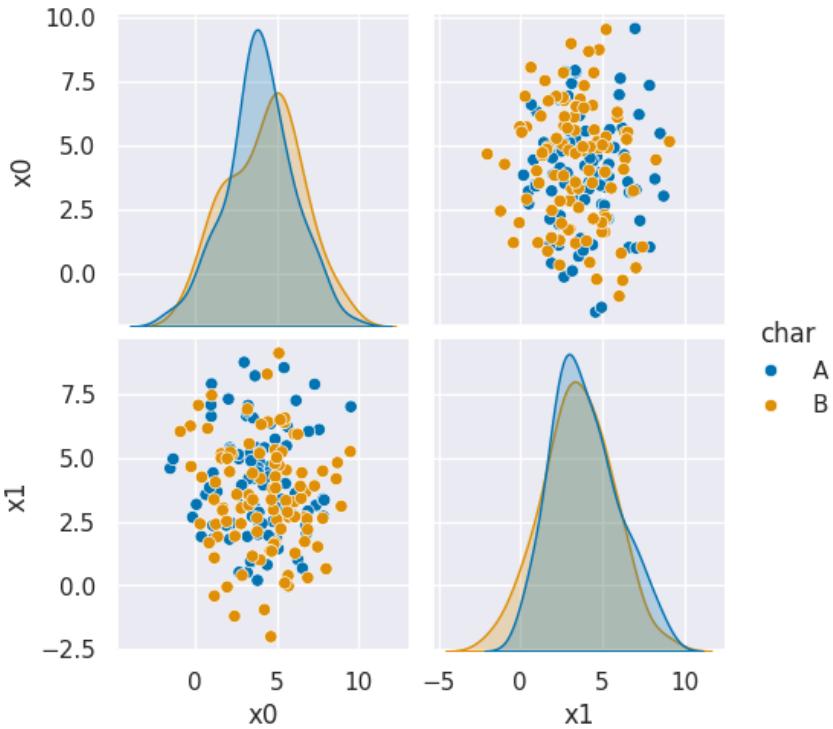
```
0.72
```

But we do not get a very good classification score.

To see why, we can look at what it learned.

```
N = 100
gnb_df = pd.DataFrame(np.concatenate([np.random.multivariate_normal(th, sig*np.eye(2),N)
    for th, sig in zip(gnb_corners.theta_,gnb_corners.var_)]),
    columns = ['x0','x1'])
gnb_df['char'] = [ci for cl in [[c]*N for c in gnb_corners.classes_] for ci in cl]
sns.pairplot(data =gnb_df, hue='char',hue_order=['A', 'B'])
```

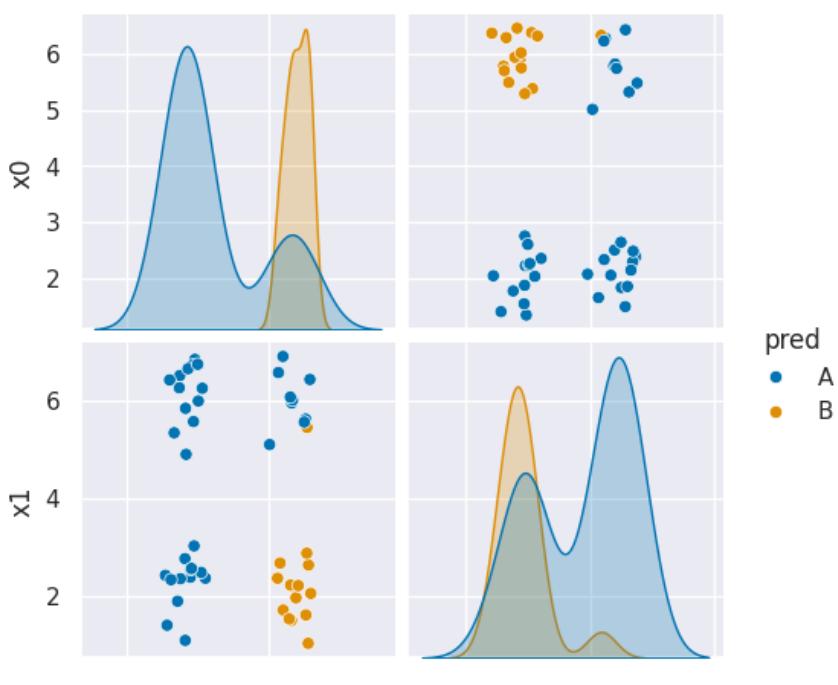
```
<seaborn.axisgrid.PairGrid at 0x7f89f3291460>
```



```
df6_pred = X_test.copy()  
df6_pred['pred'] = gnb_corners.predict(X_test)
```

```
sns.pairplot(data =df6_pred, hue ='pred', hue_order =['A','B'])
```

```
<seaborn.axisgrid.PairGrid at 0x7f89f0a87ca0>
```



[Skip to main content](#)

This does not look much like the data and it's hard to tell which is higher at any given point in the 2D space. We know though, that it has missed the mark. We can also look at the actual predictions.

If you try this again, split, fit, plot, it will learn different decisions, but always at least about 25% of the data will have to be classified incorrectly.

### 12.3.1. Decision Trees

This data does not fit the assumptions of the Naive Bayes model, but a decision tree has a different rule. It can be more complex, but for the scikit learn one relies on splitting the data at a series of points along one axis at a time.

It is a **discriminative** model, because it describes how to discriminate (in the sense of differentiate) between the classes. This is in contrast to the **generative** model that describes how the data is distributed.

```
dt = tree.DecisionTreeClassifier()
```

```
dt.fit(X_train,y_train)
```

```
▼ DecisionTreeClassifier  
DecisionTreeClassifier()
```

```
dt.score(X_test,y_test)
```

```
1.0
```

The sklearn estimator objects (that correspond to different models) all have the same API, so the `fit`, `predict`, and `score` methods are the same as above. We will see this also in regression and clustering. What each method does in terms of the specific calculations will vary depending on the model, but they're always there.

the `tree` module also allows you to plot the tree to examine it.

```
plt.figure(figsize=(15,20))  
tree.plot_tree(dt, rounded =True, class_names = ['A', 'B'],  
    proportion=True, filled =True, impurity=False, fontsize=10);
```



On the iris dataset, the sklearn docs include a diagram showing the decision boundary. You should be able to modify this for another classifier.

## 12.4. Setting Classifier Parameters

The decision tree we had above has a lot more layers than we would expect. This is really simple data so we still got perfect classification. However, the more complex the model, the more risk that it will learn something noisy about the training data that doesn't hold up in the test set.

Fortunately, we can control the parameters to make it find a simpler decision boundary.

```
dt2 = tree.DecisionTreeClassifier(max_depth=2)
dt2.fit(X_train,y_train)
dt2.score(X_test,y_test)
```

```
plt.figure(figsize=(15,20))
tree.plot_tree(dt2, rounded =True, class_names = ['A', 'B'],
proportion=True, filled =True, impurity=False, fontsize=10);
```

We can compute other metrics from the confusion matrix:

- [sklearn](#)
- [wiki](#)

## 12.5. Questions After Class

### 12.5.1. When should you start focusing on portfolio check 2?

When you get P1 feedback

### 12.5.2. when will p1 check be graded?

As soon as possible.

### 12.5.3. why not use a decision tree initially for the iris dataset?

To teach the Gaussian Naive Bayes classifier, because learning different types of classifiers helps you understand the concept better.

### 12.5.4. Are there any popular machine learning models that use decision trees?

Yes, a lot of medical applications do, because since they are easy to understand, it is easier for healthcare providers to trust them.

### 12.5.5. Do predictive algorithms have pros and cons? Or is there a standard?

[Skip to main content](#)

Each algorithm has different properties and strengths and weaknesses. Some are more popular than others, but they all do have weaknesses.

## 12.5.6. Different kinds of trees/charts, analyze what they mean and how to translate those

## 12.5.7. is it possible after training the model to add more data to it ?

The models we will see in class, no. However, there is a thing called “online learning” that involves getting more data on a regular basis to improve its performance.

# 13. Clustering

Clustering is unsupervised learning. That means we do not have the labels to learn from. We aim to learn both the labels for each point and some way of characterizing the classes at the same time.

Computationally, this is a harder problem. Mathematically, we can typically solve problems when we have a number of equations equal to or greater than the number of unknowns. For  $N$  data points in  $d$  dimensions and  $K$  clusters, we have  $N$  equations and  $N + K * d$  unknowns. This means we have a harder problem to solve.

For today, we'll see K-means clustering which is defined by  $K$  a number of clusters and a mean (center) for each one. There are other K-centers algorithms for other types of centers.

Clustering is a stochastic (random) algorithm, so it can be a little harder to debug the models and measure performance. For this reason, we are going to look a little more closely at what it actually does than we did with classification.

```
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
```

## 13.1. How does Kmeans work?

We will start with some synthetic data and then see how the clustering works.

```

# ----- Set parameters, change these to adjust the sample
# number of classes (groups)
C = 4
# number of dimensions of the features
D=2
# number of samples
N = 200
# minimum of the means
offset = 2
# distance between means
spacing = 2
# set the variance of the blobs
var = .25

# ----- Get the class labels
# choose the first C uppercase letters using the builtin string class
classes = list(string.ascii_uppercase[:C])

# ----- Pick some means
# get the number of grid locations needed
G = int(np.ceil(np.sqrt(C)))

# get the locations for each axis
grid_locs = 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 np.meshgrid(grid_locs,grid_locs)][:C]

# store in dictionary with class labels
mu = {c: i for c, i in zip(classes,means)}

# ----- Sample the data
# randomly choose a class for each point, with equal probability
clusters_true = np.random.choice(classes,N)
# draw a random point according to the means from above for each point
data = [np.random.multivariate_normal(mu[c], var*np.eye(D)) for c in clusters_true]

# ----- Store in a dataframe
# rounding to make display neater later
df = pd.DataFrame(data = data,columns = ['x' + str(i) for i in range(D)]).round(2)

# add true cluster
df['true_cluster'] = clusters_true

```

This gives us a small dataframe with 2 features (`D=2`) and four clusters (`C=4`).

```
df.head()
```

	x0	x1	true_cluster
0	2.85	2.11	A
1	5.30	1.74	C
2	2.22	6.74	B
3	3.00	1.54	A
4	1.43	4.92	B

We can see the data with the labels

```
sns.pairplot(data=df, hue='true_cluster')
```

[Skip to main content](#)

Not

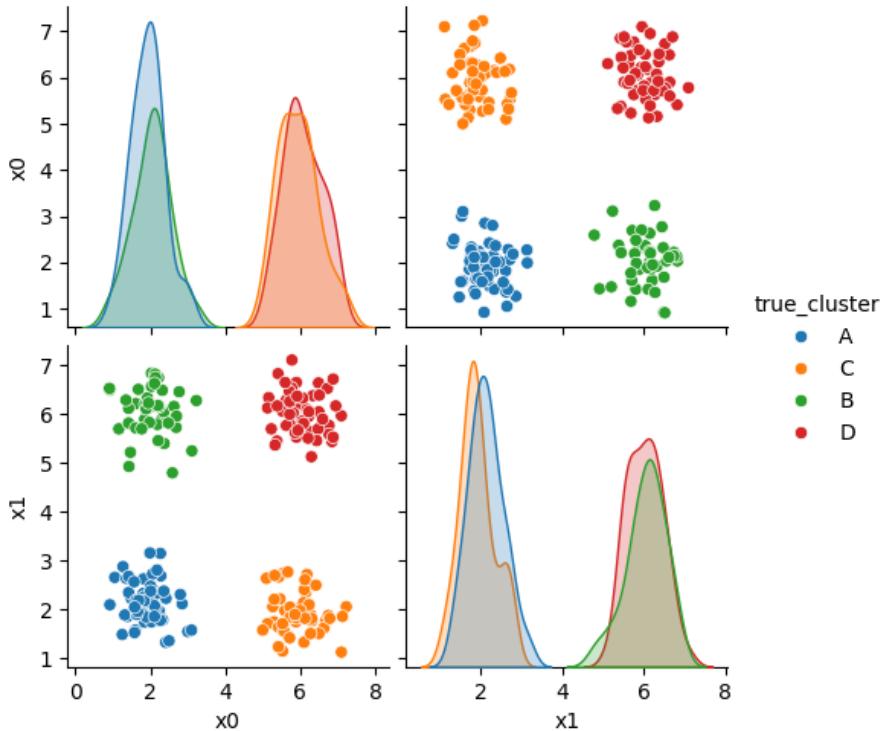
This  
but is  
part is  
for it  
together

And  
related

Hint

One v  
code  
paran  
how t  
and  
impac

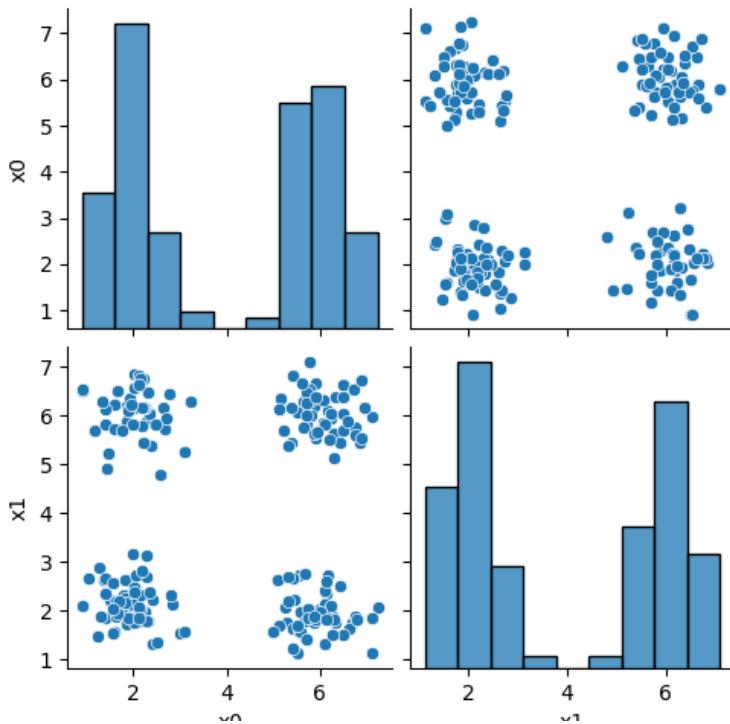
```
<seaborn.axisgrid.PairGrid at 0x7fcacf4175b0>
```



this is what the clustering algorithm will see.

```
sns.pairplot(data=df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fcacf7c607c0>
```



[Skip to main content](#)

## 13.2. Kmeans

First we will make a variable that we can use to pick out the feature columns

```
data_cols = ['x0', 'x1']
```

Next, we'll pick 4 random points to be the starting points as the means.

```
K = 4  
mu_0 = df[data_cols].sample(n=K).values  
mu_0
```

```
array([[1.36, 2.63],  
       [5.73, 2.04],  
       [1.89, 6.1 ],  
       [1.9 , 6.11]])
```

### Note

I changed this to `mu_0` while I was fixing the loop below to be more explicit instead of overwriting it.

```
i = 0
```

Here, we'll set up some helper code.

```
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]
```



For me  
seaborn

You can see that this whole color palette is paired colors, and what the `cmap_pt` and `cmap_mu` do is take odd and even subsets of the palette into two separate

```
sns.color_palette('tab20',8)
```



Now, we will compute, for each sample which of those four points it is closest to first by taking the difference, squaring it, then summing along each row.

```
[((df[data_cols]-mu_i)**2).sum(axis=1) for mu_i in mu_0]
```

```
[0    2.4905
1    16.3157
2    17.6317
3    3.8777
4    5.2490
...
195   30.1573
196   1.0676
197   43.7090
198   0.3965
199   1.1709
Length: 200, dtype: float64,
0    8.2993
1    0.2749
2    34.4101
3    7.7029
4    26.7844
...
195   13.1485
196   11.4658
197   18.4628
198   17.2225
199   13.1405
Length: 200, dtype: float64,
0    16.8417
1    30.6377
2    0.5185
3    22.0257
4    1.6040
...
195   16.5961
196   14.1338
197   25.6072
198   16.5797
199   18.1109
Length: 200, dtype: float64,
0    16.9025
1    30.6569
2    0.4993
3    22.0949
4    1.6370
...
195   16.5241
196   14.1992
197   25.5050
198   16.6673
199   18.1917
Length: 200, dtype: float64]
```

This gives us a list of 4 data DataFrames, one for each mean ( $\mu$ ), with one row for each point in the dataset with the distance from that point to the corresponding mean. We can stack these into one DataFrame.

```
pd.concat([(df[data_cols]-mu_i)**2].sum(axis=1) for mu_i in mu_0],axis=1).head()
```

	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>
<b>0</b>	2.4905	8.2993	16.8417	16.9025
<b>1</b>	16.3157	0.2749	30.6377	30.6569
<b>2</b>	17.6317	34.4101	0.5185	0.4993
<b>3</b>	3.8777	7.7029	22.0257	22.0949
<b>4</b>	5.2490	26.7844	1.6040	1.6370

[Skip to main content](#)

Now we have one row per sample and one column per mean, with the distance from that point to the mean. What we want is to calculate the assignment, which mean is closest, for each point. Using `idxmin` with `axis=1` we take the minimum across each row and returns the index (location) of that minimum.

```
pd.concat([(df[data_cols]-mu_i)**2).sum(axis=1) for mu_i in mu_0],axis=1).idxmin(axis=1)
```

```
0      0
1      1
2      3
3      0
4      2
..
195    1
196    0
197    1
198    0
199    0
Length: 200, dtype: int64
```

We'll save all of this in a column named `'0'`. Since it is our 0th iteration.

This is called the **assignment** step.

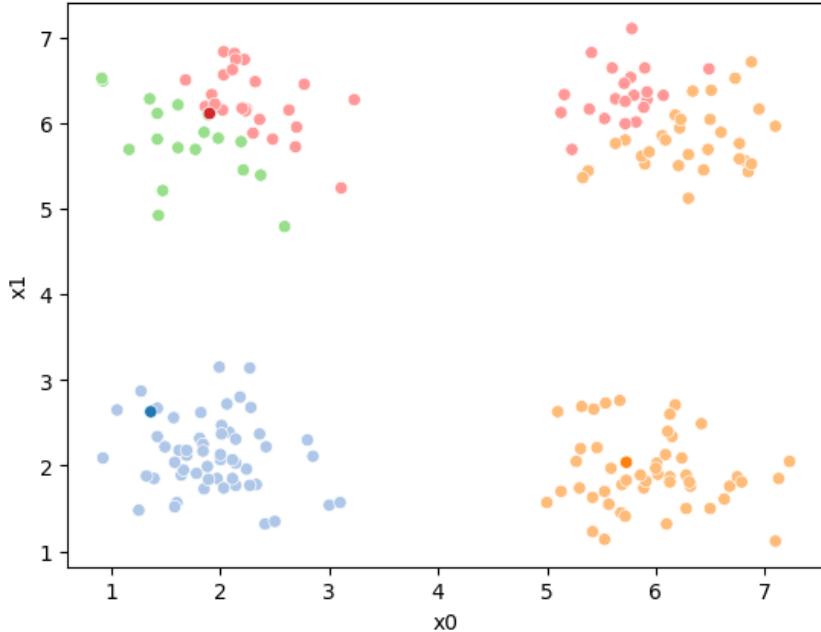
```
df[str(i)] = pd.concat([(df[data_cols]-mu_i)**2).sum(axis=1) for mu_i in mu_0],axis=1).idxmin(axis=1)
df.head()
```

	x0	x1	true_cluster	0
0	2.85	2.11	A	0
1	5.30	1.74	C	1
2	2.22	6.74	B	3
3	3.00	1.54	A	0
4	1.43	4.92	B	2

Now we can plot the data, save the axis, and plot the means on top of that. Seaborn plotting functions return an axis, by saving that to a variable, we can pass it to the `ax` parameter of another plotting function so that both plotting functions go on the same figure.

```
sfig = sns.scatterplot(data=df,x='x0',y='x1',hue='0',palette=cmap_pt,legend=False)
# plt.plot(mu[:,0],mu[:,1],marker='s',linewidth=0)
mu_df = mu_to_df(mu_0,i)
sns.scatterplot(data=mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
# sfig.get_figure().savefig('kmeans01.png')
```

```
<Axes: xlabel='x0', ylabel='x1'>
```



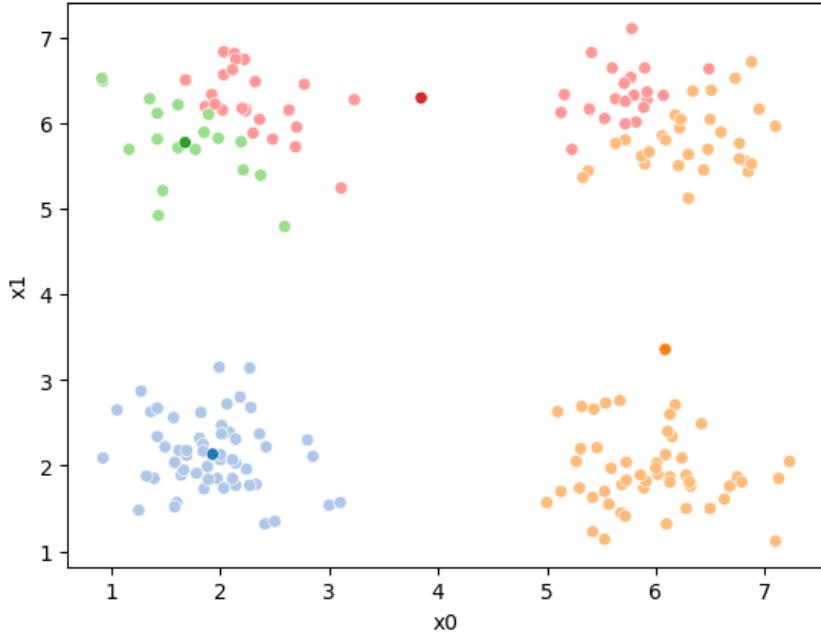
We see that each point is assigned to the lighter shade of its matching mean. These points are the one that is closest to each point, but they're not the centers of the point clouds. Now, we can compute new means of the points assigned to each cluster, using `groupby`.

```
mu_1 = df.groupby('0')[data_cols].mean().values
```

We can plot these again, the same data, but with the new means.

```
fig = plt.figure()
mu_df = mu_to_df(mu_1,1)
sfig = sns.scatterplot(data =df,x='x0',y='x1',hue='0',palette=cmap_pt,legend=False)
sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
```

```
<Axes: xlabel='x0', ylabel='x1'>
```



We see that now the means are in the center of each cluster, but that there are now points in one color that are assigned to other clusters.

So, again we can update the assignments.

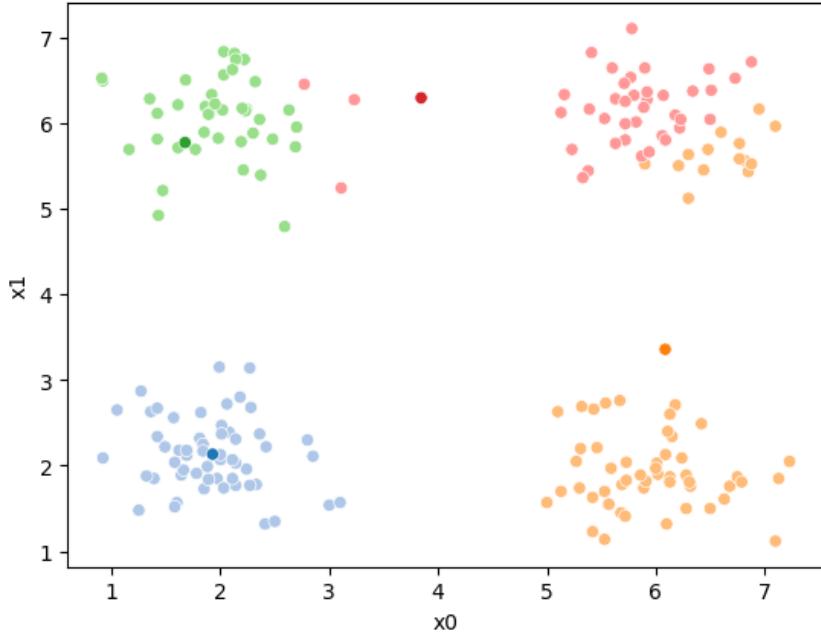
```
i=1 #increment
df[str(i)] = pd.concat([(df[data_cols]-mu_i)**2).sum(axis=1) for mu_i in mu_1],axis=1).idxmin(axis=1)
df.head()
```

	x0	x1	true_cluster	0	1
0	2.85	2.11		A	0 0
1	5.30	1.74		C	1 1
2	2.22	6.74		B	3 2
3	3.00	1.54		A	0 0
4	1.43	4.92		B	2 2

And plot again.

```
sfig = sns.scatterplot(data=df,x='x0',y='x1',hue=str(i),palette=cmap_pt,legend=False)
# plt.plot(mu[:,0],mu[:,1],marker='s',linewidth=0)
mu_df = mu_to_df(mu_1,i)
sns.scatterplot(data=mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
```

```
<Axes: xlabel='x0', ylabel='x1'>
```



If we keep going back and forth like this, eventually, the assignment step will not change any assignments. We call this condition convergence. We can implement the algorithm with a while loop.

### ⚠ Correction

In the following I swapped the order of the mean update and assignment steps.

My previous version had a different *initialization* (the above part) so it was okay for the steps to be in the other order.

```

mu_list = [mu_to_df(mu_0,0),mu_to_df(mu_1,1)]
cur_old = str(i-1)
cur_new = str(i)
mu = mu_1
while sum(df[cur_old] !=df[cur_new]) >0:
    cur_old = cur_new
    i +=1
    cur_new = str(i)
    print(cur_new)
    #      update the means and plot with current generating assignments
    mu = df.groupby(cur_old)[data_cols].mean().values
    mu_df = mu_to_df(mu,i)
    mu_list.append(mu_df)

    fig = plt.figure()
    # plot with old assignments
    sfig = sns.scatterplot(data =df,x='x0',y='x1',hue=cur_old,palette=cmap_pt,legend=False)
    sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
    file_num = str(i*2 -1).zfill(2)
    sfig.get_figure().savefig('kmeans' +file_num + '.png')

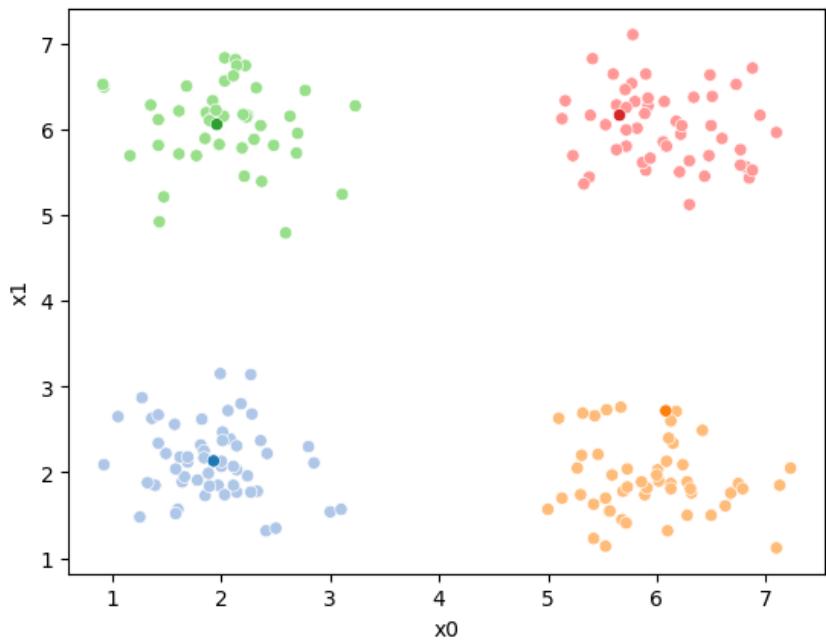
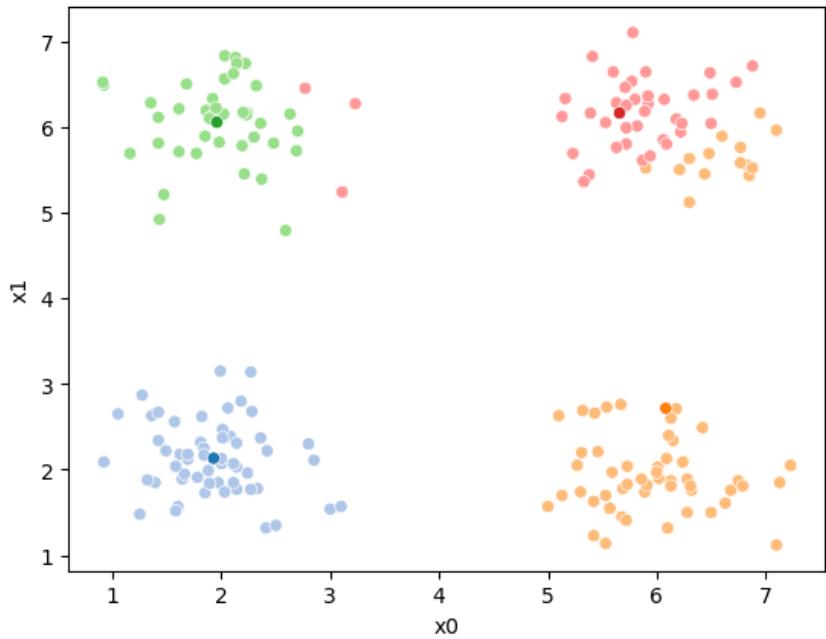
    #      update the assigments and plot with the associated means
    df[cur_new] = pd.concat([( (df[data_cols]-mu_i)**2).sum(axis=1) for mu_i in mu],axis=1).idxmin(axis=1)

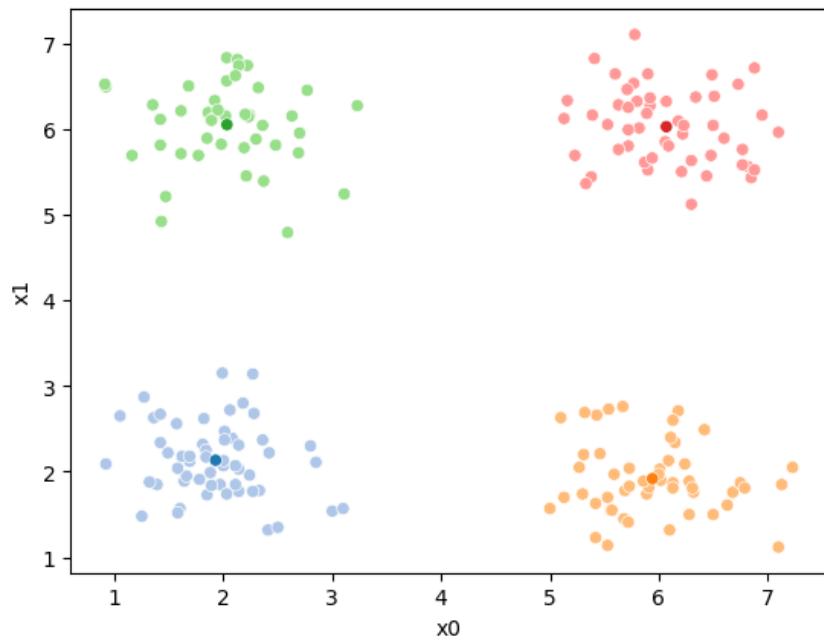
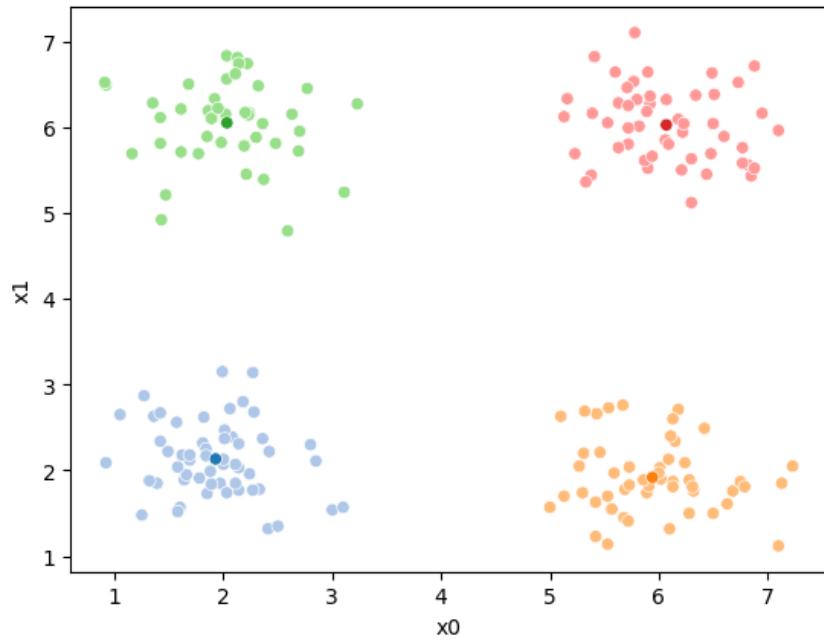
    fig = plt.figure()
    sfig = sns.scatterplot(data =df,x='x0',y='x1',hue=cur_new,palette=cmap_pt,legend=False)
    sns.scatterplot(data =mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)
    #      plt.plot(mu[:,0],mu[:,1],marker='s',linewidth=0)
    # we are making 2 images per iteration
    file_num = str(i*2).zfill(2)
    sfig.get_figure().savefig('kmeans' +file_num + '.png')

```

2

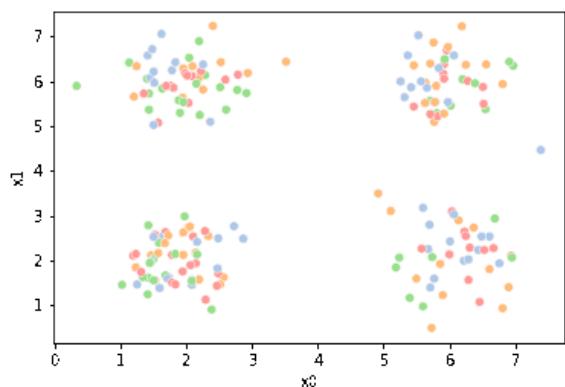
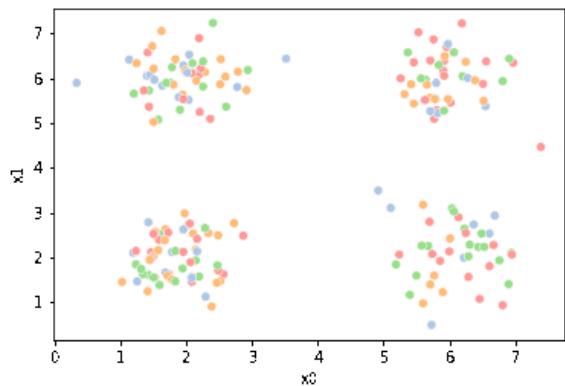
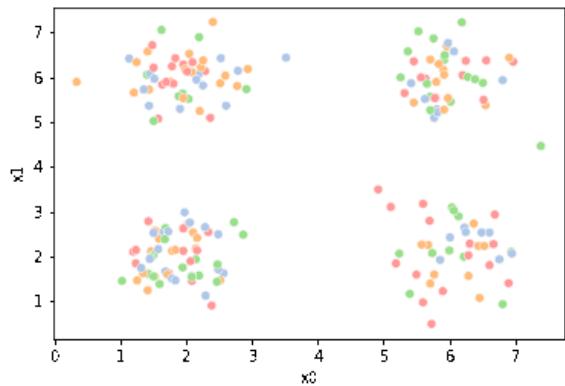
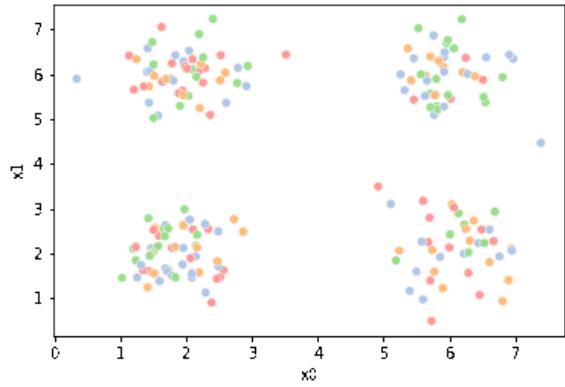
3



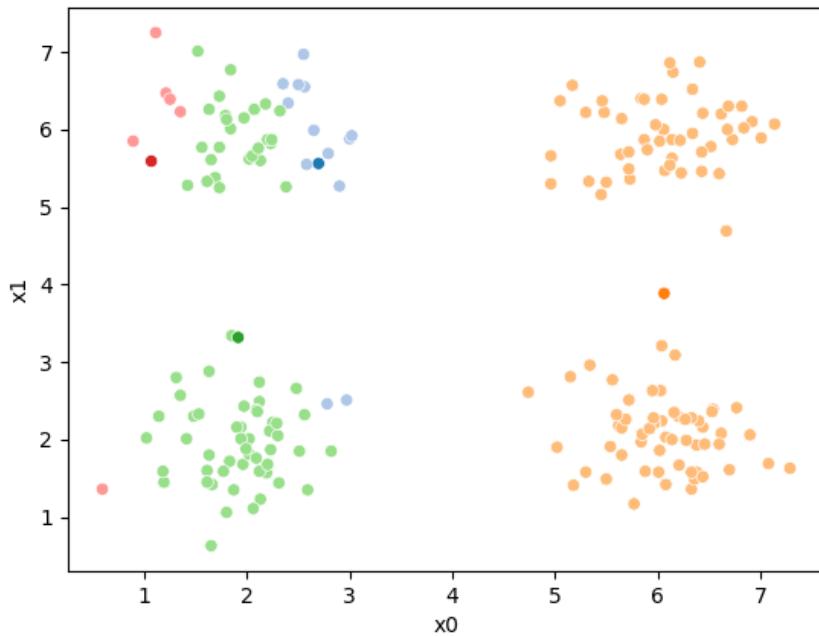


This algorithm is random. So each time we run it, it looks a little different.

The `savefig` step allows me to save the separate images and then I converted them into gifs. I have saved some from today as well as some past ones.



[Skip to main content](#)



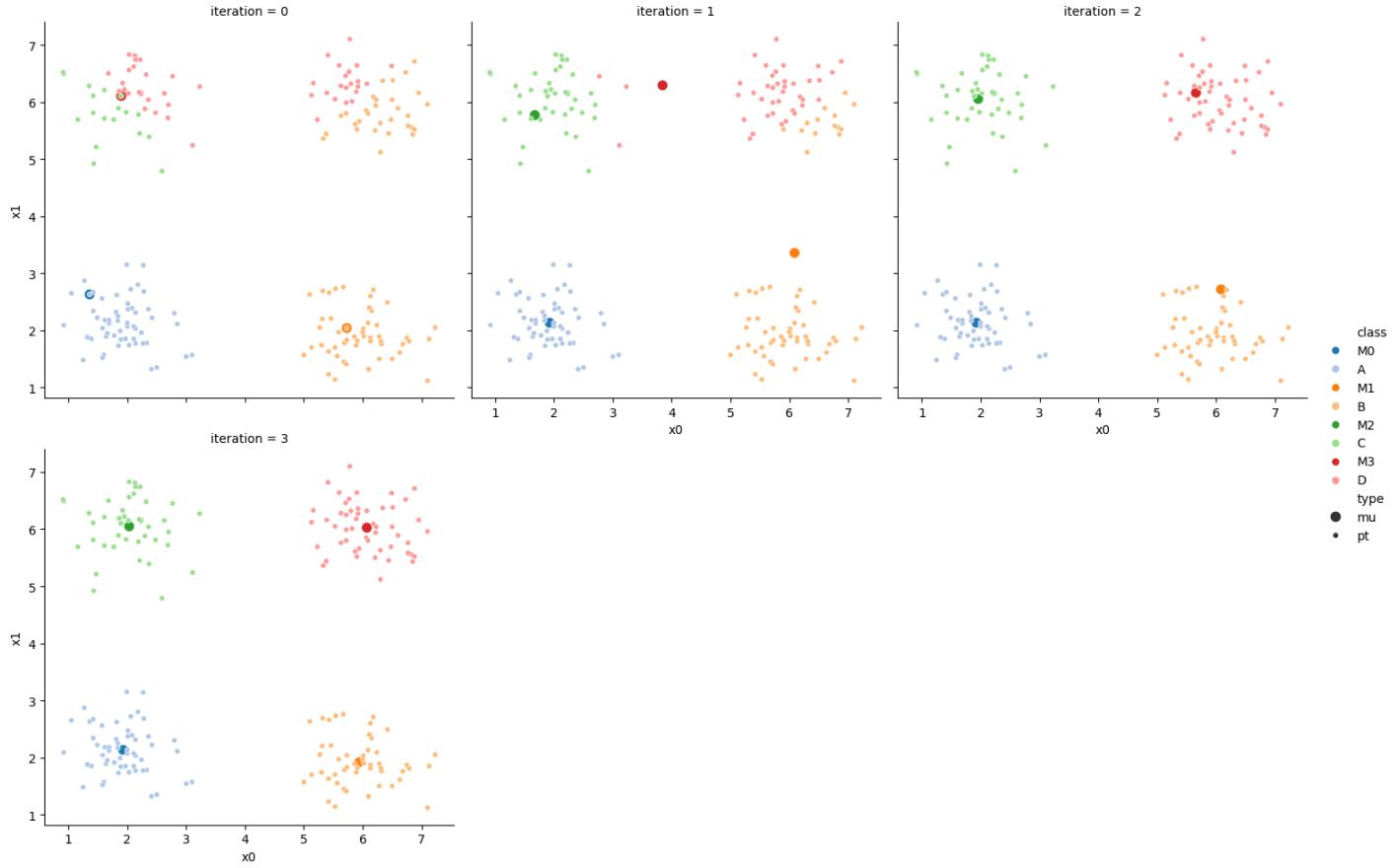
```

df_vis = df.melt(id_vars = ['x0','x1'], var_name ='iteration',value_name='class')
df_vis.replace({'class':{i:c for i,c in enumerate(string.ascii_uppercase[:C])}},inplace=True)

df_vis['type'] = 'pt'
df_mu_vis = pd.concat([pd.concat(mu_list),df_vis])
cmap = sns.color_palette('tab20',8)
n_iter = i

sfig = sns.relplot(data=df_mu_vis,x='x0',y='x1',hue='class',col='iteration',
    col_wrap=3,hue_order = ['M0','A','M1','B','M2','C','M3','D'],
    palette = cmap,size='type',col_order=[str(i) for i in range(n_iter+1)])

```



and another run I saved that has a lot of iterations:



### 13.3. Clustering with KMeans

We can fit it with the `fit` method. We have to instantiate it with the number of clusters.

Splitting to test and train is not always necessary, because we are not testing predictions.

```
kmeans = KMeans(n_clusters=4, random_state=0).fit(df[data_cols])
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning
super().__check_params_vs_input(X, default_n_init=10)
```

Once it is fit the learned labels for each sample are saved.

```
kmeans.labels_
```

```
array([0, 2, 3, 0, 3, 1, 0, 0, 1, 1, 0, 0, 3, 1, 1, 1, 1, 3, 2, 0, 1, 3, 1,
       3, 1, 3, 1, 3, 2, 3, 0, 3, 3, 0, 1, 1, 2, 0, 2, 1, 3, 2, 1, 2, 0,
       1, 2, 2, 0, 0, 2, 3, 0, 2, 1, 3, 1, 3, 2, 2, 1, 0, 3, 3, 2, 3, 2,
       0, 3, 1, 1, 1, 3, 2, 0, 0, 2, 2, 1, 1, 0, 2, 2, 3, 1, 0, 1, 1, 3,
       2, 1, 0, 3, 2, 0, 0, 2, 1, 0, 2, 0, 2, 1, 0, 2, 2, 3, 1, 0, 0, 0,
       0, 3, 1, 0, 0, 1, 1, 3, 2, 2, 1, 1, 2, 0, 2, 2, 3, 0, 0, 2, 3, 0,
       0, 1, 1, 3, 0, 3, 2, 1, 0, 0, 2, 0, 2, 0, 3, 0, 0, 1, 0, 3, 0, 3,
       1, 3, 2, 1, 3, 2, 2, 3, 2, 1, 2, 2, 3, 0, 2, 2, 2, 3, 0, 3, 0, 3,
       1, 1, 0, 1, 1, 1, 3, 2, 2, 2, 2, 1, 0, 0, 2, 3, 3, 1, 0, 1,
       0, 0], dtype=int32)
```

As are the cluster centers.

```
kmeans.cluster_centers_
```

```
array([[1.92781818, 2.13363636],
       [6.07        , 6.02529412],
       [5.94254902, 1.91882353],
       [2.03255814, 6.04930233]])
```

In this toy dataset we can see that they match the true ones:

```
df.groupby('true_cluster')[data_cols].mean()
```

true_cluster	x0	x1
A	1.927818	2.133636
B	2.032558	6.049302
C	5.942549	1.918824
D	6.070000	6.025294

However, note that they can be in different orders.

## 13.4. Questions

### 13.4.1. what happens if we do not know how many clusters there are?

There are some clustering algorithms that can also learn the number of clusters, for example the [Dirichlet Process Gaussian Mixture Model](#).

Alternatively, you can try the algorithm with different numbers of clusters and evaluate to determine which one is the best fit.

### 13.4.2. what's the difference between synthetic data and data from the gaussian machine learning model.

They're both Gaussian data. This data is actually a little bit simpler than the GNB data, because all of the variances are the same.

[Skip to main content](#)

### 13.4.3. How helpful would clustering be over the previous classifiers if there are no discernible groups within 2D plots of the data?

Clustering is for a case of a different goal. It is for when there are no labels, but you have reason to believe that there are separate groups.

If the data has even just 3 features it is possible that the clusters are not visible in 2D plots but exist and are findable with KMeans.

### 13.4.4. How do clustering algorithms relate the data to the desired output. Is it purely for humans to be able to understand what the data is or is there some deeper systems going on.

Clustering can help you discover groups in the data even if they are not available in the real data. It is a way to discover something else that is going on that was not known in advance.

### 13.4.5. What are some more applications for clustering?

Some example applications:

- one of my lab mates in grad school, used unsupervised learning in collaboration with medical doctors to discover subtypes of COPD (Chronic Obstructive Pulmonary Disorder).
- Etsy Data Scientists used clustering to discover “styles” of products like geometric, farmhouse, goth, etc.
- My Dissertation research was a different type of unsupervised learning but similar, to use brain imaging data and discover what images different people in a study viewed which images as similar.
- a data scientist at ESPN might think there are “types” of players in a sport, they could use data on the players to discover it
- a data scientist at a retail company like Walmart or Target might think there are “types” of customers and what to discover them from the data on what people have purchased.

### 13.4.6. How can clustering benefit the analysis of data?

It allows you to find groups in the data that is not an available column.

## 14. Clustering Metrics

### 14.1. Comparing Classification and Clustering Data

Datasets for classification must have a target variable observed, but we can drop it to use it for clustering

### 14.2. 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

## 14.3. Clustering with Sci-kit Learn

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
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)
```

Today we will load the iris data from seaborn:

```
iris_df = sns.load_dataset('iris')
```

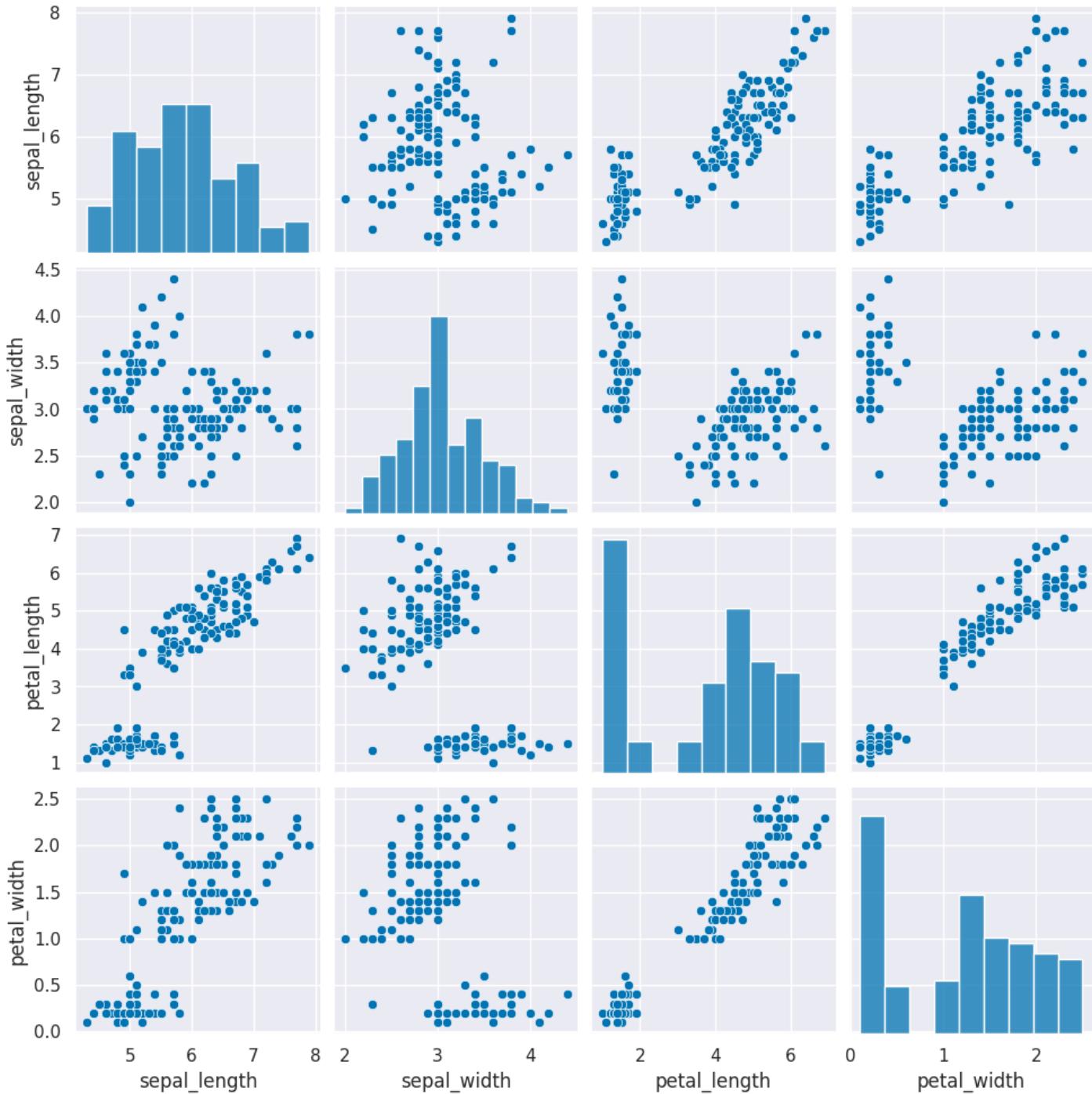
```
iris_df.head()
```

	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

Remember this is how the clustering algorithm sees the data, with no labels:

```
sns.pairplot(iris_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fd388fd55b0>
```



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.

```
iris_X = iris_df.drop(columns=['species'])
```

[Skip to main content](#)

```
iris_X.head()
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Next, we create a Kmeans estimator object with 3 clusters, since we know that the iris data has 3 species of flowers. We refer to these three groups as classes in classification (the goal is to label the classes...) and in clustering we sometimes borrow that word. Sometimes, clustering literature will be more abstract and refer to partitions, this is especially common in more mathematical/statistical work as opposed to algorithmic work on clustering.

```
km = KMeans(n_clusters=3)
```

We dropped the *column* that tells us which of the three classes that each sample(row) belongs to. We still have data from three species of flows.

### 💡 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.

```
km.fit_predict(iris_X)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning:  
super().__check_params_vs_input(X, default_n_init=10)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2,  
2, 2, 2, 1, 1, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2,  
2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1], dtype=int32)
```

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

[Skip to main content](#)

Now that we know what these are, we can save them to our DataFrame

```
iris_df['km3_1'] = km.fit_predict(iris_X)
```

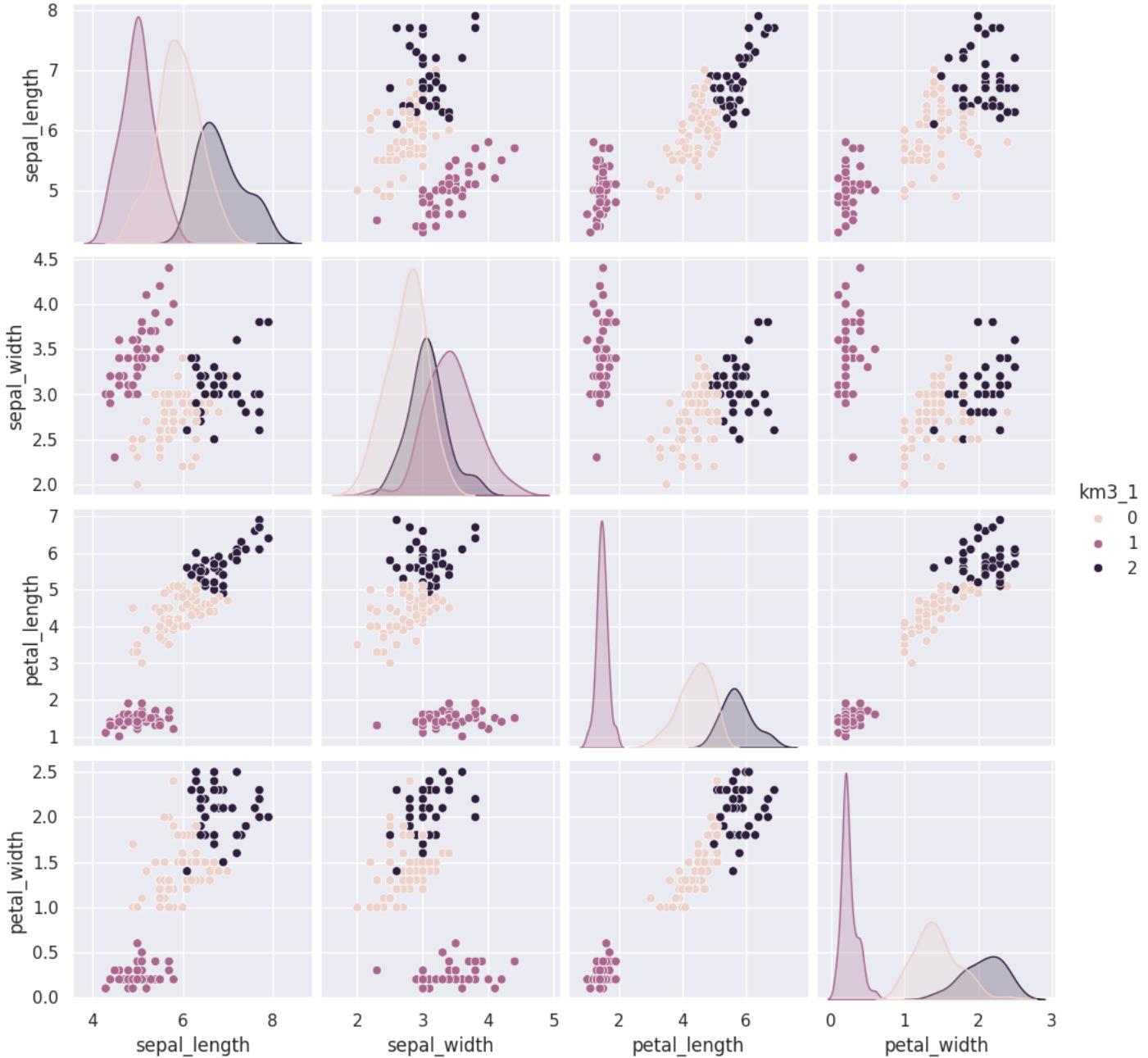
```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning:  
super().__check_params_vs_input(X, default_n_init=10)
```

## 14.4. 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.

```
sns.pairplot(data=iris_df, hue='km3_1')
```

```
<seaborn.axisgrid.PairGrid at 0x7fd3c48e4a60>
```



## 14.5. Clustering Persistence

We can run kmeans a few more times and plot each time and/or compare with a neighbor/ another group.

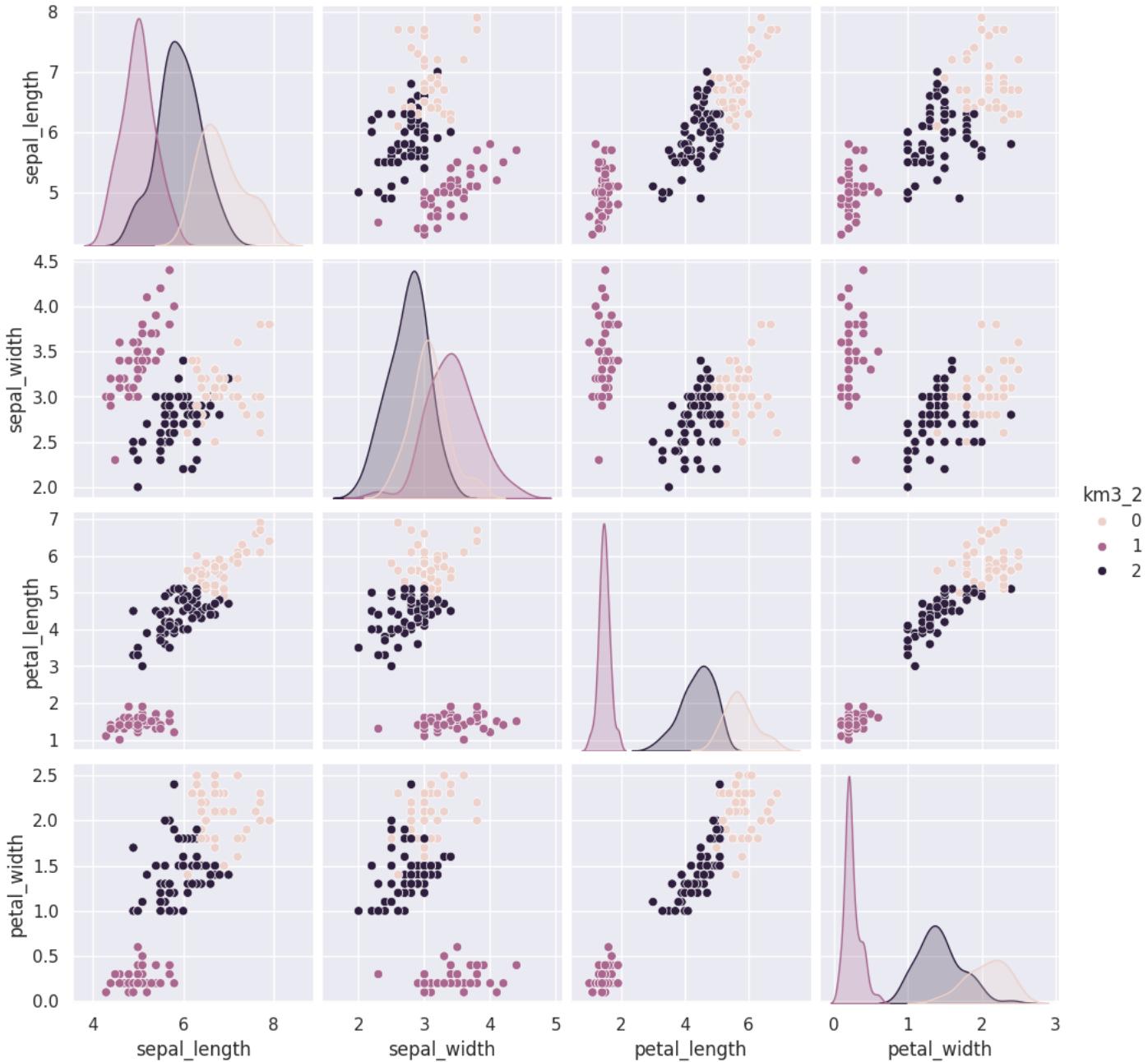
```
iris_df['km3_2'] = km.fit_predict(iris_X)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning:  
super().__check_params_vs_input(X, default_n_init=10)
```

seaborn plots them.

```
measurement_cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
sns.pairplot(data=iris_df,hue = 'km3_2', vars=measurement_cols)
```

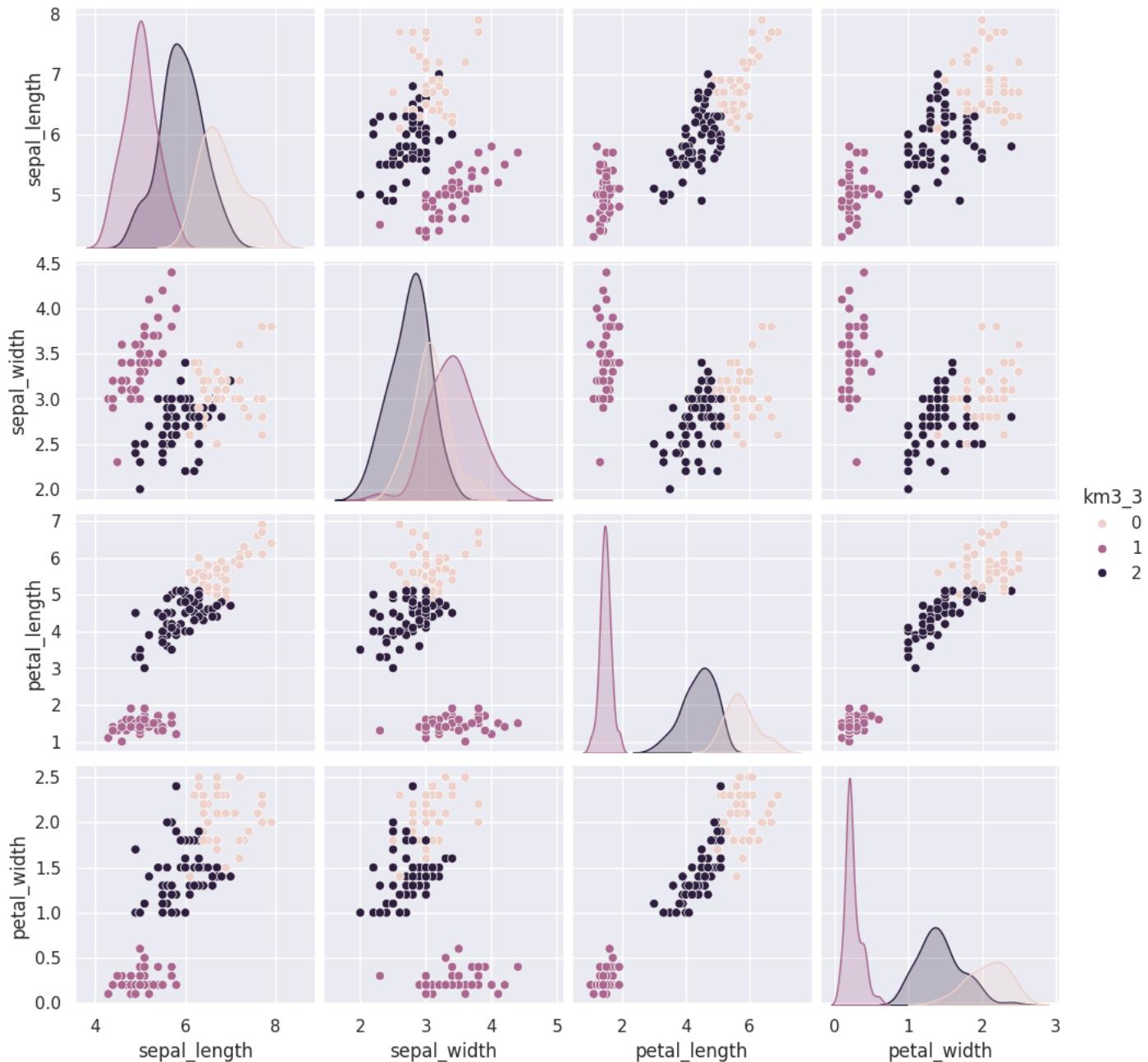
<seaborn.axisgrid.PairGrid at 0x7fd384070640>



```
iris_df['km3_3'] = km.fit_predict(iris_X)
measurement_cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
sns.pairplot(data=iris_df,hue = 'km3_3', vars=measurement_cols)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning:  
super().__check_params_vs_input(X, default_n_init=10)
```

```
<seaborn.axisgrid.PairGrid at 0x7fd3842989a0>
```



We will use a loop to add a few more.

```
for i in range(4,15):  
    iris_df['km3_' + str(i)] = km.fit_predict(iris_X)
```

Tip  
using  
make  
repeat

[Skip to main content](#)

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning
  super().__check_params_vs_input(X, default_n_init=10)
```

If we look at the assignments, we can see how similar they are.

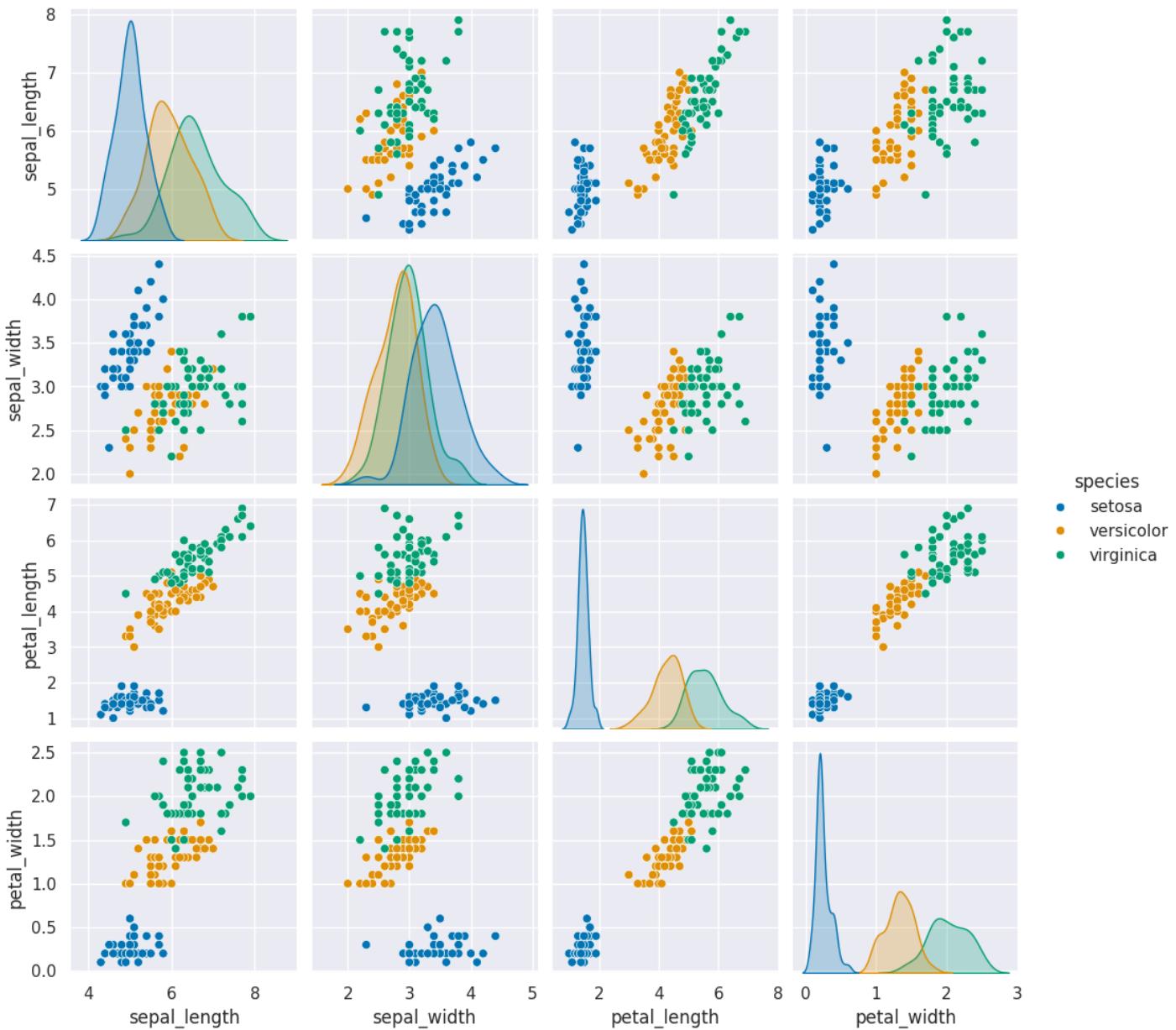
```
iris_df[measurement_cols].sample(10)
```

	sepal_length	sepal_width	petal_length	petal_width
90	5.5	2.6	4.4	1.2
21	5.1	3.7	1.5	0.4
80	5.5	2.4	3.8	1.1
62	6.0	2.2	4.0	1.0
114	5.8	2.8	5.1	2.4
19	5.1	3.8	1.5	0.3
13	4.3	3.0	1.1	0.1
41	4.5	2.3	1.3	0.3
69	5.6	2.5	3.9	1.1
95	5.7	3.0	4.2	1.2

Notice that while the numbers vary across the columns, the same rows have the same or different values across the columns. This means that each sample sometimes gets assigned to a different label, but the same ones are grouped together.

```
sns.pairplot(iris_df, hue='species', vars=measurement_cols)
```

```
<seaborn.axisgrid.PairGrid at 0x7fd379d42e20>
```



```
iris_df.columns
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species',
       'km3_1', 'km3_2', 'km3_3', 'km3_4', 'km3_5', 'km3_6', 'km3_7', 'km3_8',
       'km3_9', 'km3_10', 'km3_11', 'km3_12', 'km3_13', 'km3_14'],
      dtype='object')
```

The *grouping* of the points stay the same across different runs, but which color each group gets assigned to changes. Which blob is which color changes, but the partitions are basically the same.

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

[Skip to main content](#)

## 14.6. Clustering Evaluation

$$s = \frac{b - a}{\max(a, b)}$$

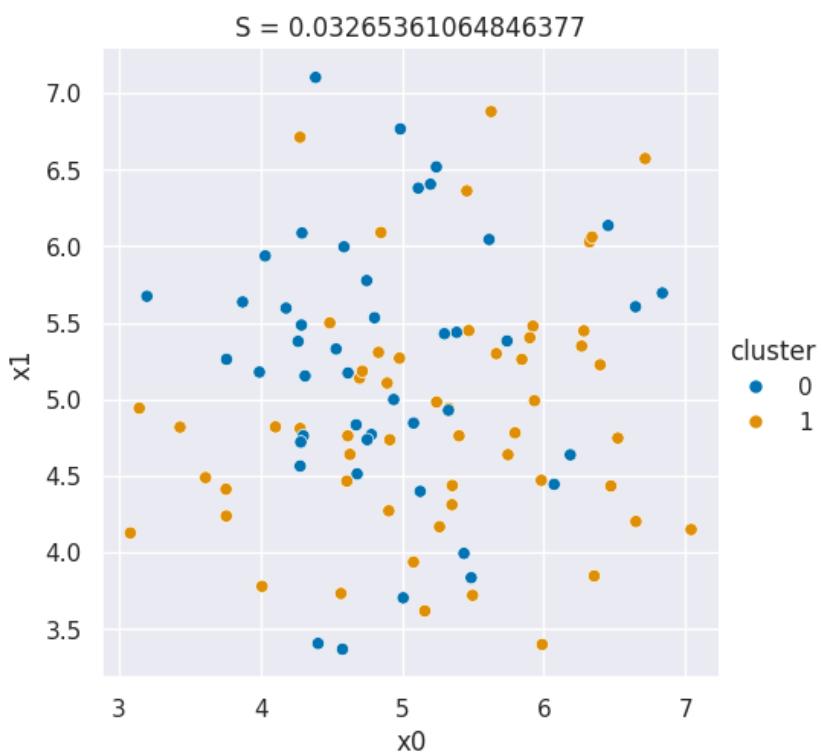
a: The mean distance between a sample and all other points in the same class.

b: The mean distance between a sample and all other points in the next nearest cluster.

This score computes a ratio of how close points are to points in the same cluster vs other clusters

We drew pictures in prismia, but we can also generate samples to make a plot that shows a bad silhouette score:

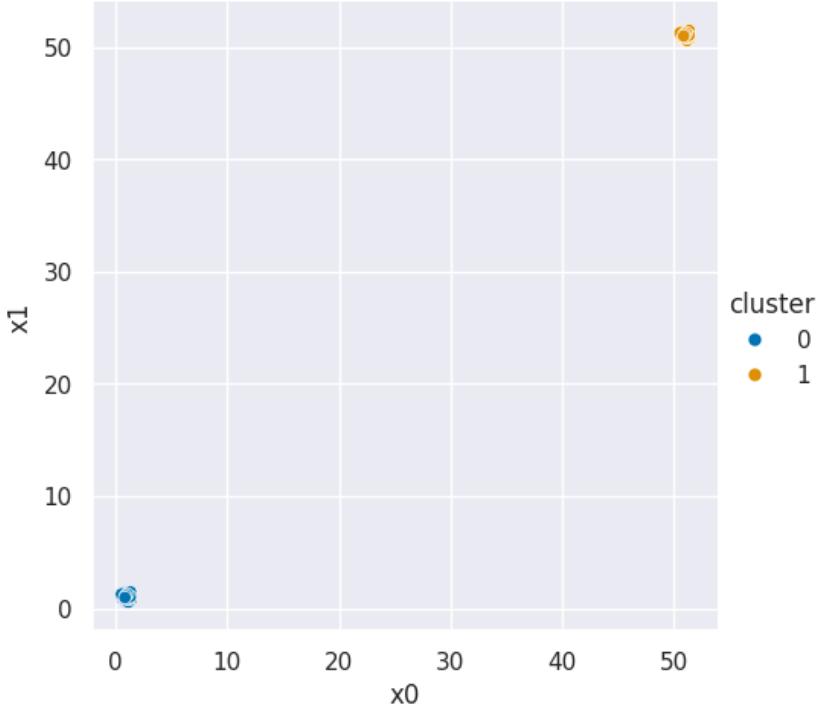
► Show code cell source



And we can make a plot for a good silhouette score

► Show code cell source

$$S = 0.9944347700098711$$



Then we can use the silhouette score on our existing clustering solutions.

```
metrics.silhouette_score(iris_X, iris_df['km3_2'])
```

```
0.5528190123564102
```

We can also apply it using a lambda:

```
sil = lambda col: metrics.silhouette_score(iris_X, col)
iris_df[['km3_1', 'km3_2', 'km3_3', 'km3_4', 'km3_5', 'km3_6', 'km3_7', 'km3_8',
         'km3_9', 'km3_10', 'km3_11', 'km3_12', 'km3_13', 'km3_14']].apply(sil)
```

```
km3_1      0.552819
km3_2      0.552819
km3_3      0.552819
km3_4      0.552819
km3_5      0.552819
km3_6      0.552819
km3_7      0.552819
km3_8      0.552819
km3_9      0.552819
km3_10     0.552819
km3_11     0.552819
km3_12     0.552819
km3_13     0.552819
km3_14     0.552819
dtype: float64
```

All of these solutions were the same

When we do not know the right number, we can try different numbers and compare the silhouette score.

```
km2 = KMeans(n_clusters=2)
iris_df['km2'] = km2.fit_predict(iris_X)

km4 = KMeans(n_clusters=4)
iris_df['km4'] = km4.fit_predict(iris_X)
metrics.silhouette_score(iris_df[measurement_cols],iris_df['km2']), metrics.silhouette_score(iris_df[measurement_cols],iris_df['km4'])
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning:
super().__check_params_vs_input(X, default_n_init=10)
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning:
super().__check_params_vs_input(X, default_n_init=10)
```

```
(0.6810461692117465, 0.49805050499728815)
```

For this dataset, 2 clusters describes the data best, even though the data came from 3 species of flowers. 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.

## 14.8. 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.

```
metrics.adjusted_mutual_info_score(iris_df['species'],iris_df['km3_1'])
```

```
0.7551191675800485
```

```
metrics.adjusted_mutual_info_score(iris_df['species'],iris_df['km3_2'])
```

```
0.7551191675800484
```

## 14.9. Questions After Class

### 14.9.1. How do you know the right number of clusters to use before knowing anything about the data?

You do not. You have to try it out. Either like we did above by trying different numbers and checking the score, or using a clustering

[Skip to main content](#)

## 14.9.2. How to determine which cluster is what?

That you have to assign based on looking at the samples. Remember in a *real* clustering situation, you do not know what the groups are.

Typically, once we find the clusters, we look at a few samples in each cluster to try to assign a name for it. For example in the Etsy case of clustering product images to find "styles" after they have the groups of images, they look at a few that are near the "center" of the cluster for each cluster to come up with a name for each one.

In the COPD example, after they found the groups, the doctors continued studying what was going on in each group. They will do genetic analysis and continue studying to try to decide a name for each.

## 14.9.3. Difference between k-means, elbow, and silhouette

K-means is the clustering algorithm. The elbow technique refers to plotting the score for different solutions and then picking the one at that looks like an elbow or right before it to be your final thing. Silhouette is a score to evaluate clustering solutions.

# 15. Regression

## 15.1. Feedback

See prismia notes for feedback qualitative comments

## 15.2. Setting up for regression

We're going to predict **tip** from **total bill**. This is a regression problem because the target, *tip* is:

1. available in the data (makes it supervised) and
2. a continuous value. The problems we've seen so far were all classification, species of iris and the character in that corners data were both categorical.

Using linear regression is also a good choice because it makes sense that the tip would be approximately linearly related to the total bill, most people pick some percentage of the total bill. If we our prior knowledge was that people typically tipped with some more complicated function, this would not be a good model.

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import pandas as pd
sns.set_theme(font_scale=2, palette='colorblind')
```

We will load this data from `seaborn`

```
tips_df = sns.load_dataset('tips')
tips_df.head()
```

[Skip to main content](#)

	total_bill	tip	sex	smoker	day	time	size	
0	16.99	1.01	Female		No	Sun	Dinner	2
1	10.34	1.66	Male		No	Sun	Dinner	3
2	21.01	3.50	Male		No	Sun	Dinner	3
3	23.68	3.31	Male		No	Sun	Dinner	2
4	24.59	3.61	Female		No	Sun	Dinner	4

Since we will use only one variable, we have to do some extra work. When we use sklearn with a DataFrame, it picks out the values, like this:

```
tips_df['total_bill'].values
```

```
array([16.99, 10.34, 21.01, 23.68, 24.59, 25.29, 8.77, 26.88, 15.04,
       14.78, 10.27, 35.26, 15.42, 18.43, 14.83, 21.58, 10.33, 16.29,
       16.97, 20.65, 17.92, 20.29, 15.77, 39.42, 19.82, 17.81, 13.37,
       12.69, 21.7 , 19.65, 9.55, 18.35, 15.06, 20.69, 17.78, 24.06,
       16.31, 16.93, 18.69, 31.27, 16.04, 17.46, 13.94, 9.68, 30.4 ,
       18.29, 22.23, 32.4 , 28.55, 18.04, 12.54, 10.29, 34.81, 9.94,
       25.56, 19.49, 38.01, 26.41, 11.24, 48.27, 20.29, 13.81, 11.02,
       18.29, 17.59, 20.08, 16.45, 3.07, 20.23, 15.01, 12.02, 17.07,
       26.86, 25.28, 14.73, 10.51, 17.92, 27.2 , 22.76, 17.29, 19.44,
       16.66, 10.07, 32.68, 15.98, 34.83, 13.03, 18.28, 24.71, 21.16,
       28.97, 22.49, 5.75, 16.32, 22.75, 40.17, 27.28, 12.03, 21.01,
       12.46, 11.35, 15.38, 44.3 , 22.42, 20.92, 15.36, 20.49, 25.21,
       18.24, 14.31, 14. , 7.25, 38.07, 23.95, 25.71, 17.31, 29.93,
       10.65, 12.43, 24.08, 11.69, 13.42, 14.26, 15.95, 12.48, 29.8 ,
       8.52, 14.52, 11.38, 22.82, 19.08, 20.27, 11.17, 12.26, 18.26,
       8.51, 10.33, 14.15, 16. , 13.16, 17.47, 34.3 , 41.19, 27.05,
       16.43, 8.35, 18.64, 11.87, 9.78, 7.51, 14.07, 13.13, 17.26,
       24.55, 19.77, 29.85, 48.17, 25. , 13.39, 16.49, 21.5 , 12.66,
       16.21, 13.81, 17.51, 24.52, 20.76, 31.71, 10.59, 10.63, 50.81,
       15.81, 7.25, 31.85, 16.82, 32.9 , 17.89, 14.48, 9.6 , 34.63,
       34.65, 23.33, 45.35, 23.17, 40.55, 20.69, 20.9 , 30.46, 18.15,
       23.1 , 15.69, 19.81, 28.44, 15.48, 16.58, 7.56, 10.34, 43.11,
       13. , 13.51, 18.71, 12.74, 13. , 16.4 , 20.53, 16.47, 26.59,
       38.73, 24.27, 12.76, 30.06, 25.89, 48.33, 13.27, 28.17, 12.9 ,
       28.15, 11.59, 7.74, 30.14, 12.16, 13.42, 8.58, 15.98, 13.42,
       16.27, 10.09, 20.45, 13.28, 22.12, 24.01, 15.69, 11.61, 10.77,
       15.53, 10.07, 12.6 , 32.83, 35.83, 29.03, 27.18, 22.67, 17.82,
       18.78])
```

This is because originally sklearn actually only worked on `numpy` arrays and so what they do now is pull it out of the DataFrame, it gives us a numpy array:

```
type(tips_df['total_bill'].values)
```

```
numpy.ndarray
```

All sklearn estimator objects are designed to work on multiple features, which means they expect that the features have a 2D shape, but when we pick out a single column's (pandas Series) values, we get a 1D shape:

```
tips_df['total_bill'].values.shape
```

[Skip to main content](#)

```
(244, )
```

We can change this by adding an axis. It doesn't change values, but changes the way it is represented enough that it has the properties we need.

```
tips_X = tips_df['total_bill'].values[:, np.newaxis]
```

This has the shape we want:

```
tips_X.shape
```

```
(244, 1)
```

and is still the same type

```
type(tips_X)
```

```
numpy.ndarray
```

The target is expected to be a single feature, so we do not need to do anything special.

```
tips_y = tips_df['tip']
```

Next, we split the data

```
tips_X_train, tips_X_test, tips_y_train, tips_y_test = train_test_split(tips_X, tips_y)
```

## 15.3. Fitting a regression model

We instantiate the object as we do with all other sklearn estimator objects.

```
regr = linear_model.LinearRegression()
```

and `fit` like the others too.

```
regr.fit(tips_X_train, tips_y_train)
```

```
▼ LinearRegression  
LinearRegression()
```

we also can predict as before

[Skip to main content](#)

## 15.4. Regression Predictions

Linear regression is making the assumption that the target is a linear function of the features so that we can use the equation for a line(for scalar variables):

$$y = mx + b$$

becomes equivalent to the following code assuming they were all vectors/matrices

```
i = 1 # could be any number from 0 ... len(target)
target[i] = regr.coef_*features[i]+ regr.intercept_
```

You can match these up one for one.

- `target` is  $y$  (this is why we usually put the `y` in the variable name)
- `regr.coef_` is the slope,  $m$
- `features` are the  $x$  (like for  $y$ , we label that way)
- and `regr.intercept_` is the  $b$  or the  $y$  intercept.

### ! Important

I have tried to expand the detail here substantively. Use the link at the top of the page to make an issue if you have specific questions

We will look at each of them and store them to variables here

```
slope = regr.coef_
slope
```

```
array([0.11221888])
```

```
intercept = regr.intercept_
intercept
```

```
0.7921598186362337
```

► Show code cell content

### ! Important

This is what our model *predicts* the tip will be based on the past data. It is important to note that this is not what the tip *should* be by any sort of virtues. For example, a typical normative rule for tipping is to tip 15% or 20%. the model we learned, from this data, however is `array([11.22188752])% + $0.7921598186362337`.

now we can pull out our first  $x$  and calculate a predicted value of  $y$  manually

[Skip to main content](#)

```
slope*tips_X_test[0] + intercept
```

```
array([2.6796813])
```

and see that this matches exactly the prediction from the `predict` method.

```
y_pred[0]
```

```
2.679681299195818
```

### ! Important

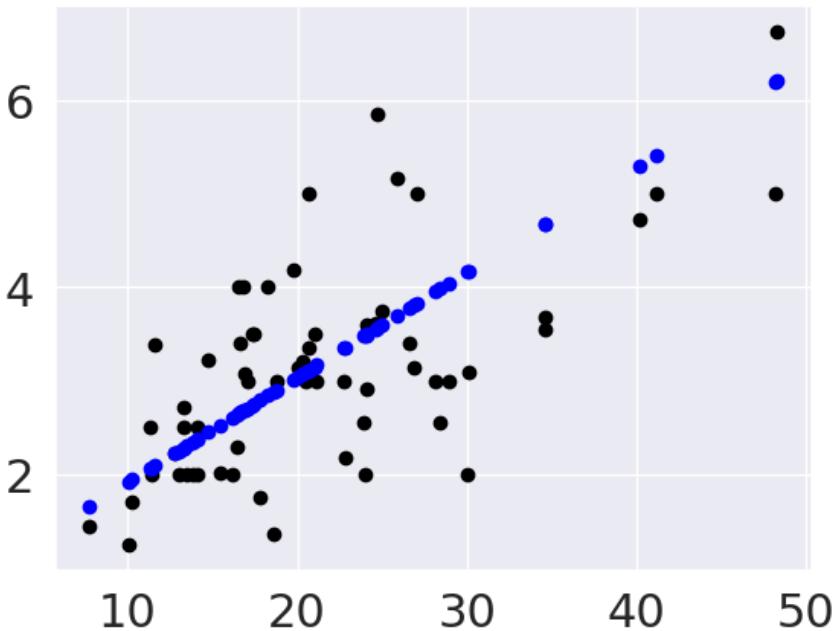
Note that my random seed is not fixed, so the values there are going to change each time the website is updated, but that these values will *always* be true

## 15.5. Visualizing Regression

Since we only have one feature, we can visualize what was learned here. We will use plain `matplotlib` plots because we are plotting from numpy arrays not data frames.

```
plt.scatter(tips_X_test,tips_y_test, color='black')
plt.scatter(tips_X_test,y_pred, color='blue')
```

```
<matplotlib.collections.PathCollection at 0x7f1856ff7670>
```

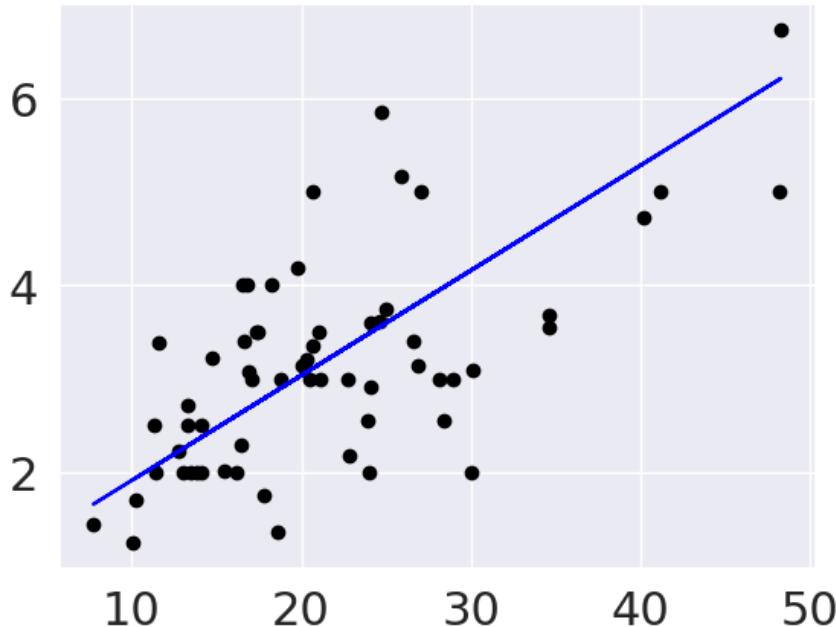


This plots the predictions in blue vs the real data in black. The above uses them both as scatter plots to make it more clear below.

[Skip to main content](#)

```
plt.scatter(tips_X_test, tips_y_test, color='black')
plt.plot(tips_X_test, y_pred, color='blue')
```

```
[<matplotlib.lines.Line2D at 0x7f1854ece250>]
```



Since the predictions are perfectly in a line, if I use a line plot instead of scatter to connect them all, it makes a line.

## 15.6. Evaluating Regression - Mean Squared Error

From the plot, we can see that there is some error for each point, so accuracy that we've been using, won't work. One idea is to look at how much error there is in each prediction, we can look at that visually first.

These red lines are the residuals, or errors in each prediction.

In this code block, I use `zip` a [builtin function in Python](#) to iterate over all of the test samples and predictions (they're all the same length) and plot each red line in a list comprehension. This could have been a for loop, but the comprehension is slightly more compact visually. The zip allows us to have a Pythonic, easy to read loop that iterates over multiple variables.

To make a vertical line, I make a line plot with just two values. My plotted data is: `[x, x], [yp, yt]` so the first point is `(x, yp)` and the second is `(x, yt)`.

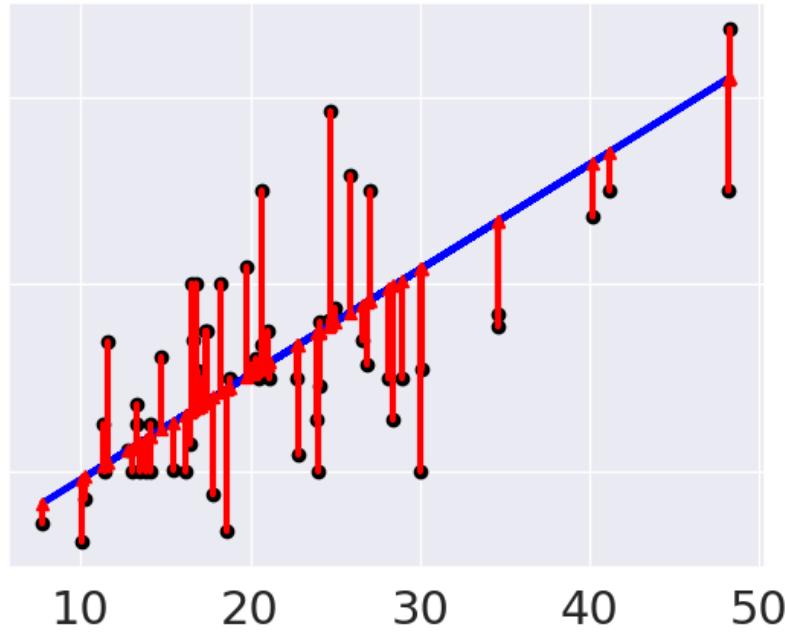
The `;` at the end of each line suppressed the text output.

```
# plot line prediction
plt.plot(tips_X_test, y_pred, color='blue', linewidth=3);

# draw vertical lines from each data point to its predict value
[plt.plot([x,x],[yp,yt], color='red', linewidth=3, markevery=[0], marker ='^')
 for x, yp, yt in zip(tips_X_test, y_pred,tips_y_test)];

# plot these last so they are visually on top
plt.scatter(tips_X_test, tips_y_test, color='black');
```

[Skip to main content](#)



From this we can see that the errors are relatively small, which matches what we saw with the mean squared error.

To see more about how the code above works:

▶ Show code cell content

We can use the average length of these red lines to capture the error. To get the length, we can take the difference between the prediction and the data for each point. Some would be positive and others negative, so we will square each one then take the average.

```
mean_squared_error(tips_y_test,y_pred)
```

```
0.7947789263150189
```

Which is equivalent to:

```
np.mean((tips_y_test-y_pred)**2)
```

```
0.7947789263150189
```

To interpret this, we can take the square root to get it back into dollars. This becomes equivalent to taking the mean absolute value of the error.

```
mean_abs_error = np.sqrt(mean_squared_error(tips_y_test,y_pred))
mean_abs_error
```

```
0.8915037444200775
```

```
np.mean(np.abs(tips_y_test-y_pred))
```

```
0.7061743084626356
```

Still, to know if this is a big or small error, we have to compare it to the values we were predicting

```
tips_y_test.describe()
```

```
count    61.000000
mean     3.135738
std      1.124791
min      1.250000
25%     2.230000
50%     3.000000
75%     3.600000
max      6.730000
Name: tip, dtype: float64
```

```
avg_tip = tips_y_test.mean()
tip_var = tips_y_test.var()
avg_tip, tip_var
```

```
(3.1357377049180326, 1.2651548633879781)
```

► Show code cell content

We can see that the error is smaller than most of the tips.

## 15.7. Evaluating Regression - R<sup>2</sup>

We can also use the  $R^2$  score, the coefficient of determination.

If we have the following:

- $n = \text{len}(y_{\text{test}})$
- $y = y_{\text{test}}$
- $y_i = y_{\text{test}[i]}$
- $\hat{y} = y_{\text{pred}}$
- $\bar{y} = \frac{1}{n} \sum_{i=0}^n y_i = \text{sum}(y_{\text{test}}) / \text{len}(y_{\text{test}})$

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^n (y_i - \hat{y}_i)^2}{\sum_{i=0}^n (y_i - \bar{y})^2}$$

```
r2_score(tips_y_test, y_pred)
```

[Skip to main content](#)

This is a bit harder to interpret, but we can use some additional plots to visualize. This code simulates data by randomly picking 20 points, spreading them out and makes the “predicted” y values by picking a slope of 3. Then I simulated various levels of noise, by sampling noise and multiplying the same noise vector by different scales and adding all of those to a data frame with the column name the r score for if that column of target values was the truth.

Then I added some columns of y values that were with different slopes and different functions of x. These all have the small amount of noise.

```
x = 10*np.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

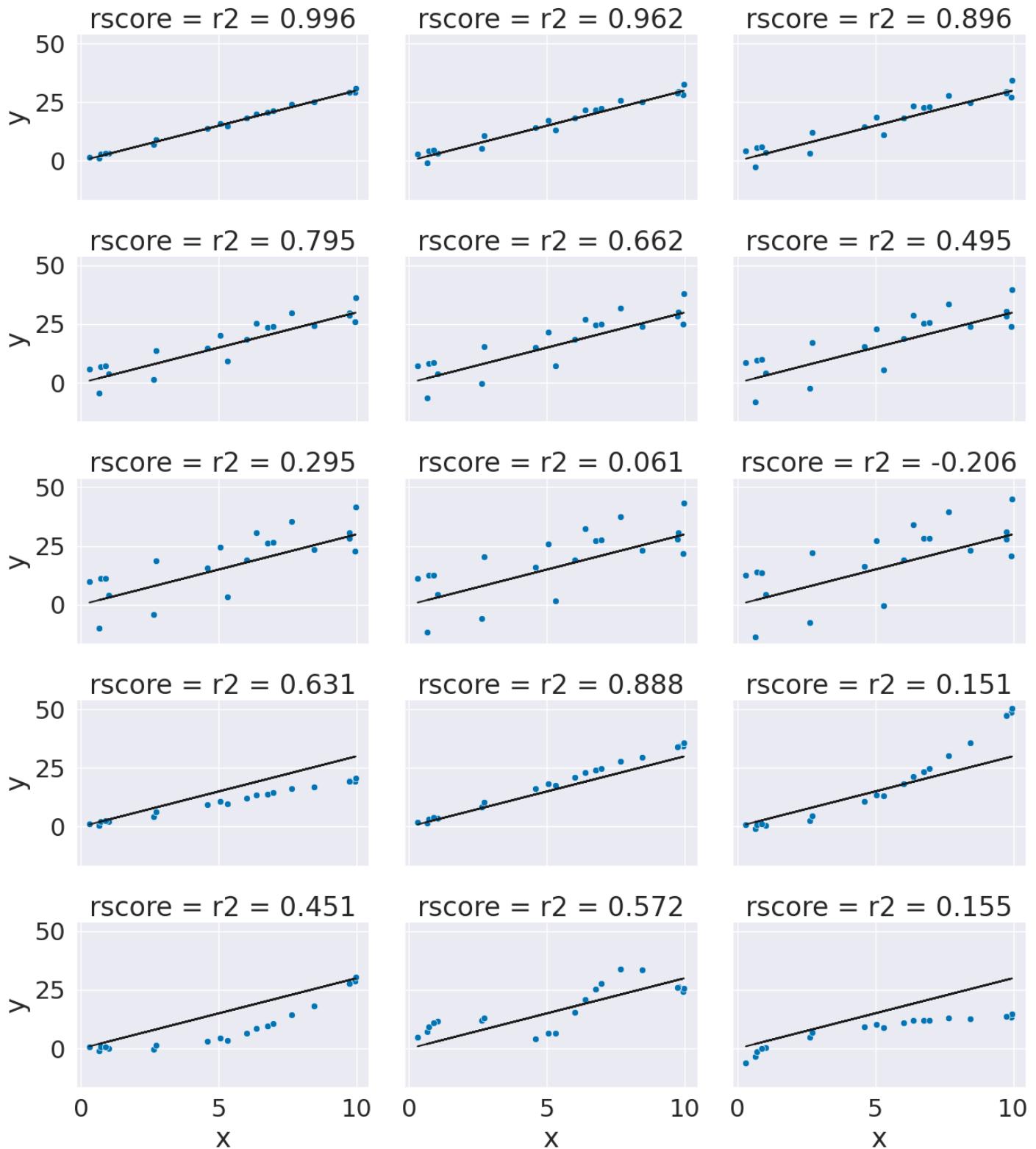
f_x_list = [2*x,3.5*x,.5*x**2, .03*x**3, 10*np.sin(x)+x*3,3*np.log(x**2)]
for fx in f_x_list:
    y_true = fx + noise
    ex_df['r2 = '+ str(np.round(r2_score(y_pred,y_true),3))] = y_true

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",)
```

Tip

Face  
than t  
we ha  
those  
partic

```
<seaborn.axisgrid.FacetGrid at 0x7f1853d7f310>
```



```
y_pred = regr.predict(tips_X_test)
```

```
mean_squared_error(tips_y_test,y_pred)
```

[Skip to main content](#)

```
0.7947789263150189
```

```
r2_score(tips_y_test,y_pred)
```

```
0.36132305606463144
```

```
regr.score(tips_X_test,tips_y_test)
```

```
0.36132305606463144
```

## 15.8. Questions after class

### 15.8.1. How do you know if a regression model is good or works for a dataset?

Let's break this down. Regression is supervised learning, so there needs to be variables that you can use as features and one to use as the target. To test if supervised learning makes sense, try filling column names in place of `features` and `target` in the following sentence, that describes a prediction task: we want to predict `target` from `features`.

Regression, specifically means that the target variable needs to be continuous, not discrete. For example, in today's class, we predicted the tip in dollars, which is continuous in this sense. This is not strictly continuous in the mathematical sense, but anything that is most useful to consider as continuous.

*logistic regression is actually a classification model despite the name*

### 15.8.2. So is regression is just used with numeric values in terms of predicting?

Yes linear regression requires numeric values for the features as well as the target.

Other types of regression can use other types of features.

All regression is for a continuous target.

### 15.8.3. What are the strengths of a regression model as opposed to the previous models we've seen?

Using regression, classification, or clustering is based on what type of problem you are solving.

Our simple flowchart for what task you are doing is:

```
flowchart TD
    A[labels{Do you have labels  
in the dataset?}] --> B[|yes|]
    B --> C[Supervised Learning]
    C --> D[labels{Are the labels?  
continuous?}]
    D --> E[Regression]
    E --> F[|no|]
    F --> G[Unsupervised Learning]
    G --> H[labels{Are the labels?  
discrete?}]
    H --> I[Classification]
    I --> J[Clustering]
```

## 15.8.4. Are linear models exclusively used for regression problems?

No, there are also linear classification models.

## 15.8.5. What does a negative R2 score mean?

It's a bad fit.

## 15.8.6. What is the difference between a good and bad rscore?

1 is perfect, low is bad. What is "good enough" is context dependent.

## 15.8.7. why were your values coming out negative?

Mine was not a very good fit. We'll see why on Thursday. Yours may have been better

## 15.8.8. What would be a common example of a real-world linear regression?

Lots of real models are linear regression, but with many inputs. It is often not a perfect fit, but good enough.

## 15.8.9. what are the coefficient and intercept ?

The coefficient is the slope of the line and the intercept is the value of the label the model predicts for features =0.

## 15.8.10. how do you add string data thats not binary e.g 'True' 'False' to make it fit in data?

You have to transform it to be numerical. If it is True and False, you can cast it to integers.

I recommend for A9 filtering on the UCI website to pick out data with numerical features.

In general, and you can for A9 if you want, use the `sklearn LabelEncoder` to transform the data.

# 16. Interpreting Regression

```
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')
```

## 16.1. Review - Univariate rearession

[Skip to main content](#)

```
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

```
tips_X = tips_df['total_bill'].values[:, np.newaxis]
tips_y = tips_df['tip']
tips_X_train, tips_X_test, tips_y_train, tips_y_test = train_test_split(tips_X, tips_y, random_state=5, train_s
```

```
regr = linear_model.LinearRegression()
regr.fit(tips_X_train, tips_y_train)
```

▼ LinearRegression  
LinearRegression()

and get a score

```
regr.score(tips_X_test, tips_y_test)
```

```
-0.03240944770933374
```

```
regr.coef_
```

```
array([0.11469301])
```

```
regr.intercept_
```

```
0.7426466344875204
```

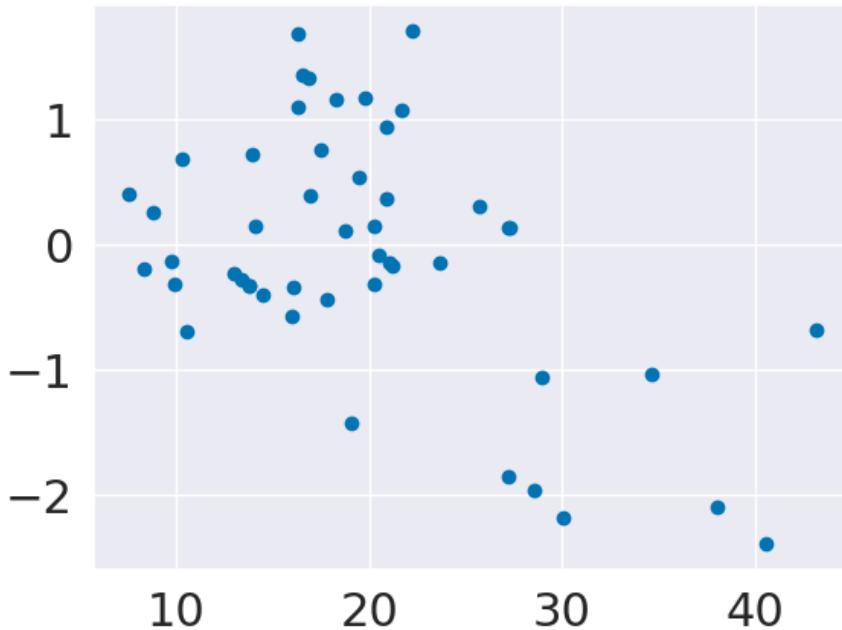
```
tips_y_pred = regr.predict(tips_X_test)
```

## 16.2. Examining residuals more closely

We can plot the residuals, or errors directly versus the input.

[Skip to main content](#)

```
<matplotlib.collections.PathCollection at 0x7f752c9ea520>
```



If our model was predicting uniformly well, these would be evenly distributed left to right.

## 16.3. Multivariate Regression

We can look at the estimator again and see what it learned. It describes the model like a line:

$$\hat{y} = mx + b$$

except in this case it's multivariate, so we can write it like:

$$\hat{y} = \beta^T x + \beta_0$$

where  $\beta$  is the `regr_db.coef_` and  $\beta_0$  is `regr_db.intercept_` and that's a vector multiplication and  $\hat{y}$  is `y_pred` and  $y$  is `y_test`.

In scalar form it can be written like

$$\hat{y} = \sum_{k=0}^d (x_k * \beta_k) + \beta_0$$

where there are  $d$  features, that is  $d = \text{len}(x\_test[k])$  and  $k$  indexed into it. For example in the below  $k = 0$

```
tips_X = tips_df[['total_bill', 'size']]
tips_y = tips_df['tip']
tips_X_train, tips_X_test, tips_y_train, tips_y_test = train_test_split(tips_X, tips_y, random_state=5, train_s
```

We do not need the newaxis once we have more than one feature

[Skip to main content](#)

```
tips_df[['total_bill', 'size']].values.shape
```

```
(244, 2)
```

```
tips_df[['total_bill', 'size']].values[:, np.newaxis].shape
```

```
(244, 1, 2)
```

```
regr = linear_model.LinearRegression()  
regr.fit(tips_X_train, tips_y_train)
```

▼ LinearRegression  
LinearRegression()

```
tips_y_pred_2 = regr.predict(tips_X_test)
```

```
regr.score(tips_X_test, tips_y_test)
```

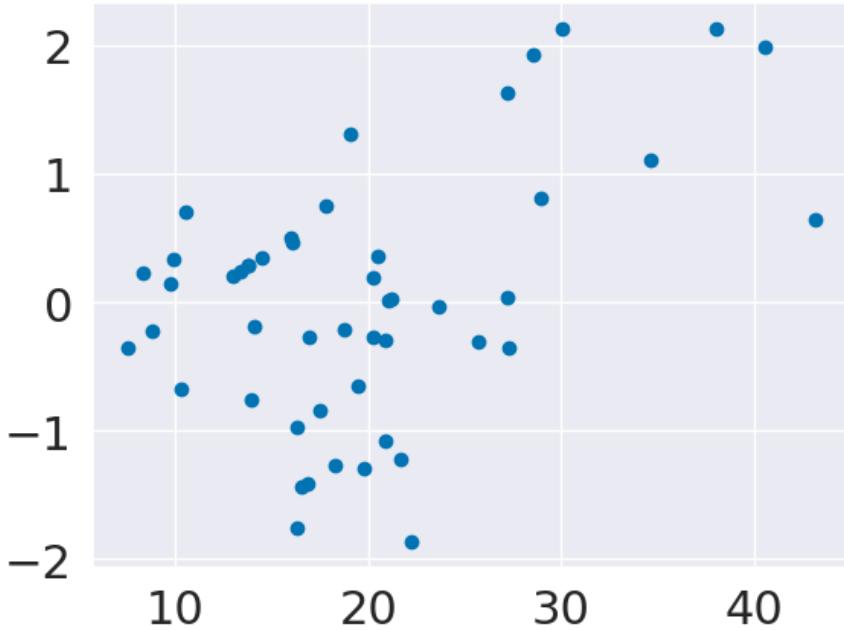
```
-0.017340127412395878
```

```
tips_y_pred_2.shape, tips_y_test.shape, tips_X_test.shape
```

```
((49,), (49,), (49, 2))
```

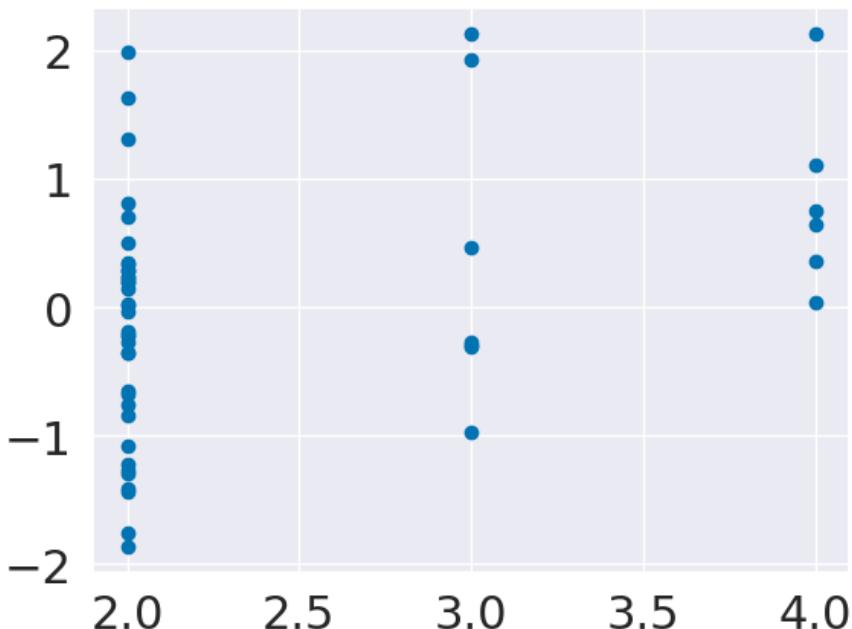
```
plt.scatter(tips_X_test['total_bill'], tips_y_pred_2 - tips_y_test)
```

```
<matplotlib.collections.PathCollection at 0x7f752c8cc2e0>
```



```
plt.scatter(tips_X_test['size'], tips_y_pred_2-tips_y_test)
```

```
<matplotlib.collections.PathCollection at 0x7f752c8b18e0>
```



regr.coef

```
array([0.10108061, 0.20096791])
```

```
0.48625594110725423
```

```
tips_X_test.head()
```

	total_bill	size
55	19.49	2
191	19.81	2
210	30.06	3
96	27.28	2
163	13.81	2

```
np.sum(tips_X_test.values[0]*regr.coef_) +regr.intercept_
```

```
2.85825279437057
```

```
tips_X_test.values[0][0]*regr.coef_[0] + tips_X_test.values[0][1]*regr.coef_[1] +regr.intercept_
```

```
2.85825279437057
```

```
tips_y_pred_2[0]
```

```
2.85825279437057
```

```
from sklearn.preprocessing import PolynomialFeatures
```

```
poly = PolynomialFeatures()
```

```
tips_X2_train = poly.fit_transform(tips_X_train)
```

```
regr2 = linear_model.LinearRegression()
regr2.fit(tips_X2_train,tips_y_train)
```

```
▼ LinearRegression
LinearRegression()
```

```
tips_X2_test = poly.fit_transform(tips_X_test)
tips_y_pred_sq = regr2.predict(tips_X2_test)
```

[Skip to main content](#)

- 0.041787136585906826

## 16.4. Questions After Class

### Note

most were something like re-explain what just happened, so I did not answer them separately below, but made sure to add in detail above

### 16.4.1. where can I find good datasets for practice?

The UCI repository is the best place to start. Use the `filters` to select for regression, of moderate size, with numerical features.

You could also use a dataset with both numerical and categorical features and only use the numerical ones. Remember how in A2, I had you subset data to select for numerical features? There was a goal to that.

### 16.4.2. Does making the data quadratic help to find quadratic patterns, or does it simply give the model more data?

It finds quadratic patterns. That model has the same number of samples and, in a sense, the same amount of "data" but it has sort of a different view on the same data. Computing functions of our data is not considered to be generating *more* data, it's thought of as a change of representation only.

When we added a second variable to the model, that was giving it more information about each sample and actually increasing the size of the data.

### 16.4.3. How specifically does the `poly.fit_transform` work?

For the details of the code, you can see the [documentation](#). From there you can see the source for the `PolynomialFeatures` class. That inherits from the `TransformerMixin` which defines the `fit_transform` method which calls, in sequence the `fit` and `transform` methods as they are defined in the Polynomial features class. In this case the fit calculates the number of total output features and transform transforms the input features and outputs the new features.

Mathematically, we provide it a matrix,  $X \in \mathcal{R}^{n \times d}$  where the columns are vectors  $\mathbf{x}_i \in \mathcal{R}^n$  for  $i \in [0, \dots, d]$  which means that there are  $d$  columns representing the original features and  $n$  samples. We get back a new matrix with columns:

$$[1, x_0, x_0^2, x_0 * x_1, x_0 * x_2, \dots, x_0 * x_d, x_1^2, x_1 * x_2, \dots, x_1 * x_d, \dots, x_d^2]$$

This means that we can use the same algorithm to find the weights as we do for linear regression, using the transformed features.

A linear regression solver can solve for the coefficients no matter what they are multiplied by.

### 16.4.4. Can you further explain the $Y = B_1x_1 + B_2x_1^2 + B_0$ for both single variables and multiple variables?

16.4.5. I think I would just like another run down of the equation or maybe even just written in explanations

## 1. Assignment 1: Setup, Syllabus, and Review

Due: 2023-09-11

### 1.1. Evaluation

Eligible skills: (links to checklists)

- ★<sup>[1]</sup> python level 1 and level 2
- ★process<sup>[2]</sup> level 1

### 1.2. Related notes

#### ⚠ Warning

the links below will not work until the relevant notes are posted, after class

- Welcome & What is Data Science

### 1.3. Instructions

#### ❗ Important

If you have trouble, check the GitHub FAQ on the left first

Your task is to:

1. Install required software from the Tools & Resource page (should have been done before the first class)
2. Create your portfolio, by accepting the assignment
3. Learn about your portfolio from the README file on your repository.
4. Follow instructions in the README to make your portfolio your own with information about yourself(not evaluated, but useful) and your own definition of data science (graded for **level 1 process**)
5. complete the `success.md`` file as per the instructions in the comments
6. Create a Jupyter notebook called `grading.ipynb` and write a function that computes a grade for this course, with the docstring below.
7. Upload the notebook to your repo directly on the main branch.
8. Add the line `- file: grading` in your `_toc.yml` file.

ⓘ Note  
After  
creati  
on yo  
you're  
@rha

To do  
green  
type a

### ! Important

the syntax of the line added to your `_toc.yml` has to be exact

### ⚠ Warning

Do not merge your “Feedback” Pull Request

## 1.3.1. Docstring

```
'''  
    Computes a grade for CSC/DSP310 from numbers of achievements at each level  
  
    Parameters:  
    -----  
    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 (+/-)  
'''
```

## 1.3.2. 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+'
```

## 1.3.3. Notebook Checklist

- a Markdown cell with a heading
- your function called `compute_grade`
- three calls to your function that verify it returns the correct value for different number of badges that produce at three different letter grades.

- 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 level 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

### Quick Facts

- due : 2023-09-18
- accept assignment

### 2.1. Objective & Evaluation

This assignment is an opportunity to earn level 1 and 2 achievements in **python** and **access** and begin working toward level 1 for **summarize**. You can also earn level 1 for **process**.

Eligible skills: (links to checklists)

- **first chance** access 1 and 2
- python 1 and 2
- summarize 1
- process 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

- Iterables and Pandas Data Frames
- DataFrames from other sources

### 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 students picked the same dataset?
- how many total rows of data did each student load?

## 2.4. 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.

## 2.5. Store info about data 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:

*Table 2.1* Meta data of the dictionaries

<code>url</code>	the full url of the dataset
<code>short_name</code>	a short name
<code>load_function</code>	(the actual function handle) what function should be used to load the data into a <code>pandas.DataFrame</code> .

### 💡 Hint

See below for how you will use the dictionary as help for how you should construct it

## 2.6. Make a dataset about your datasets

In a notebook called `dataset_of_datasets.ipynb`, import the list of dictionaries from the `datasets` module you created in the step above. Then iterate over the list of dictionaries, and:

1. load each dataset like `dataset_dict['load_function'](dataset_dict['url'])`
2. save it to a local csv using the short name you provided for the dataset as the file name, without writing the index column to the file.
3. record attributes about the dataset as in the table below in a list or dictionary of lists
4. Use that to create a DataFrame with columns that match the rows of the following table.

[Skip to main content](#)

💡 Hint  
See t  
exam

Table 2.2 Meta Data Description of the DataFrame to build

name	a short name for the dataset
source	a url to where you found the data
num_rows	number of rows in the dataset
num_columns	number of columns in the dataset
num_numerical	number of numerical variables in the dataset

## 2.7. 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 first 6 rows
- make a numpy array of only the numerical data and save it to a new variable (select these programmatically)
- was the format that the data was provided in a good format? why or why not?

For any other dataset:

- read it in from your local csv using a relative path
- display the heading with the last seven 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
- save every fifth row (5,10,15,...) of the data for two columns of your choice into a new DataFrame and display that

## 2.8. Exploring data files

There are two files in the data folder, both can be read in with `read_csv` but need some options or fixing.

- try to read in the `german.data` file, what happens with the default settings? What option do you need to use to make it look right?
- try to read in the `.csv` file that's included in the template repository, use the error messages you get to try to fix the file manually (any text editor, including jupyter can edit a `.csv`), making notes about what changes you made in a markdown cell.

### 💡 Hint

For the csv file in the template's data folder in Jupyter Lab, it will not let you edit a csv file, but you can change the file

[Skip to main content](#)

## 2.9. Submission

This time you have to separately submit from posting your code to make grading easier.

1. Go to the actions tab
2. Click the action called “Prepare & Submit” in the left hand sidebar
3. click the run workflow button on the right hand side.
4. Cilck run workflow

### 💡 Hint

see the [github docs](#) for screenshots of how to do these steps.

## 2.10. 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 these after submitting the assignment. If you want, you could discuss these ideas in office hours.

1. When might you prefer one datatype over another?
2. How does PEP 8 standard code help you be collaborative?
3. Learn about [Datasheets for Datasets](#) and find some examples, (eg this [google scholar result](#)) How could something like this impact your work as a data scientist?

## 3. Assignment 3: Exploratory Data Analysis

- \_\_Due:2023-09-25\_\_

### ❗ 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.

- If working alone make a team for yourself when you [accept the assigment](#)
- If you are working with a partner, coordinate so that the first person makes the team when accepting and then the second joins the same team when they [accept the assigment](#)

## 3.1. Objective & Evaluation

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
- 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.

### 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

[Skip to main content](#)

- variable types
  - overall summary statistics
3. Write a short description of what the data contains and what it could be used for
  4. Include an 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. (if you have one really complex question that can cover the checklists below, fewer than 3 questions is okay)
  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
- at least 1 statistic or plot
- interpretation that answers the question

#### ! Important

The code and question versions below are supposed to convey the same information, in different ways so that you can use the version that makes the most sense to you.

### 3.4.2. Dataset 1 Checklist

make sure that your `dataset_01.ipynb` has:

#### 3.4.2.1. Question version

#### ⚠ Warning

these 3 should be equivalent to what is in the code version below, see those to make this more concrete if it does not make sense

- One overview of the relationship of a categorical variable to many numerical variables
- One question about a categorical variable and one numerical variable
- One question about the relationship between 3 variables

#### 3.4.2.2. Code Version

- 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:

#### 3.4.3.1. Code version

- overall summary statistics
- two individual summary statistics for one variable
- one summary statistic grouped by two categorical variables
- a figure with a grid of subplots that correspond to two categorical variables

#### 3.4.3.2. Question version

##### Warning

these 3 should be equivalent to what is in the code version see those to make this more concrete if it does not make sense

- One question that is about overall trends across multiple variables. (the interpretation here is most important)
- One question that is about one variable's range or shape so that it requires to statistics to answer it.
- One question that is about how two categorical variables influence one numerical variable

##### 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

##### 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 list of questions below for your summary review

### 3.5.1. Review Questions

1. Describe overall how it was to read the analysis overall to read. Was it easy? hard? cohesive? jumpy?
2. How did the data summaries help prepare you to read the rest of the analysis? What do you think might be missing?
3. Do the questions make sense based on the data? Are they interesting questions? What could improve the questions
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?

### 3.5.2. Response

Respond to your review either inline comments, replies, or by updating your analysis accordingly.

## 3.6. Think Ahead

### 💡 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: Cleaning Data

Due: 2023-10-04

[accept assignment](#)

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 Structural Repairs](#)
- [Reparing values](#)

### 4.2. Check the Datasets you have worked with already

[Skip to main content](#)

⚠️ War...  
This s...  
intend...  
thinki...  
If you...  
feedb...  
get to...  
want...  
mom...  
part.

In the datasets you have used or come across but decided you could not work with in your past assignments identify at least one thing you could not do because the data was not in an appropriate format.

In a notebook file called `dataset_fix.ipynb` apply one fix and show one summary statistic or plot that was not possible before to show that it works.

Some examples:

- a column that was a list or dictionary
- missing values
- a column that was continuous, but more interesting as a categorical
- too many header rows
- a data set that was wide, but tall would be better for plotting

#### 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, named

`cleaning_notes.md`. (some example datasets are on the datasets page and one is in the notes are added to the course website)

1. What are 3 common problems to look for in a dataset? Describe them with examples.

[Skip to main content](#)

- Using one of the examples you found of cleaned data, give an example of a question or context that would require making different choices for cleaning than were made. Include a bit about the data, what was done, the question, what would need to be done instead and justification.
- 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](#)

date : 2023-10-10

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 Data
- Web Scraping

### 5.2. Constructing Datasets

Your goal is to programmatically construct a ready to analyze dataset from multiple sources.

- Your dataset must combine at least 2 source tables.
- At least one source table must come from a database or from web scraping (not `pd.read\_html`).
- You should use at least two different joins (this means either use 3 data sources or combine two datasets in two different ways)

[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. (this can be one simple statistic or plot as long as it is something that requires the merge you used)

For construct only, this can be very minimal EDA.

## 5.3. Additional achievements

To earn additional achievements, you must do more cleaning and/or exploratory data analysis.

### 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

this can be in your cleanup or your EDA

#### Thinking Ahead

Compare the level 2 skill definitions to level 3, how could you extend and adapt what you've done to meet level 3?

#### Thinking Ahead

You could also demonstrate understanding of how merges work by converting a dataset that is provided as a single table with redundant information into a number of smaller tables in a database.

[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: 2023-10-18

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

### 6.2. About the data

We have provided a [reconstructed version](#) of the [Adult Dataset](#), which is a popular benchmark dataset for training machine learning models that comes from a recent [paper](#) about the risks of that dataset. The classic [Adult](#) dataset tries to predict if a person makes more or less than 50k.

Researchers reconstructed the Adult dataset with the actual value of the income. We trained models to predict `income>=$10k`, `income>=$20k`, etc. We used three different learning algorithms, nicknamed 'LR', 'GPR', and 'RPR' for each target (`>10k`, `>20k`, ..., `>90k`).

- `adult_models_only.csv` has the model's predictions
- `adult_reconstruction_bin.csv` has the data.

Both 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 at least three different performance metrics. Compare and contrast performance in those metrics across racial or gender groups.

Include easy to read tables with your performance metrics and interpretations of the model's overall performance and any disparities that could be understood by a general audience.

## 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**

Eligible skills: (links to checklists)

- **first chance** classification [1](#) and [2](#)
- evaluate [1](#) and [2](#)
- summarize [1](#) and [2](#)
- visualize [1](#) and [2](#)
- python [1](#) and [2](#)

- Intro to ML & Naive Bayes
- Understanding Classification

## 7.2. Dataset and EDA

Choose a dataset that is well suited for classification and that has *all numerical features*. If you want to use a dataset with nonnumerical features you will have to convert the categorical features to one hot encoding.

### 💡 Hint

Use the UCI ML repository, it will let you filter data by the attributes of it you need.

1. Include a basic description of the data(what the features are)
2. Describe the classification task in your own words
3. Use EDA to determine if you expect the classification to get a high accuracy or not. What types of mistakes do you think will happen most?
4. Hypothesize which will classifier we have seen 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?
  - (if DT) Are there any leaves that are very small?
  - (if DT) Is this an interpretable number of levels?
  - (if GNB) do the parameters fit the data well?
  - (if 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?

## 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.

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*

### Thinking Ahead

*ideas for level 3, not required for A7*

Repeat the problem setup experiment with multiple test/train splits at each size 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**

## 8. Assignment 8: Clustering

accept the assignment **Due: 2023-11-01**

### 8.1. Evaluation

Eligible skills: (links to checklists)

- **first chance** clustering 1 and 2
- evaluate 1 and 2
- python 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



The s  
visua  
usefu  
perfo  
fitting  
to cre  
experi



The r  
max c  
acros  
error,

## 8.3. Instructions

Use the same dataset you used for assignment 7, unless there was a problem. If you skipped assignment 7, choose a dataset well suited for classification. See A7 for tips.

1. Describe what question you would be asking in applying clustering to this dataset. What does it mean if clustering does not work well?
2. How does this task compare to what the classification task on this dataset?
3. Apply Kmeans using the known, correct number of clusters,  $K$ .
4. Evaluate how well clustering worked on the data:
  - using a true clustering metric and
  - using visualization and
  - using a clustering metric that uses the ground truth labels
5. Include a discussion of your results that addresses the following:
  - 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)
6. Repeat your analysis using a 2 different numbers (1 higher, one lower) of clusters:
  - can you interpret the new clusters?
  - how do they relate to the original clusters? are they completely different, did one split?
  - is there a reasonable explanation for more clusters than there are classes in this dataset?

## 8.4. For classification

### Note

Do this only if you did not already earn classification level 2

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?
  - (if DT) Are there any leaves that are very small?
  - (if DT) Is this an interpretable number of levels?
  - (if GNB) do the parameters fit the data well?
  - (if 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?

# 9. Assignment 9: Linear Regression

## 9.1. Quick Facts

- accept the assignment
- Due: 2023-11-06

## 9.2. Assessment

Eligible skills: (links to checklists)

- **first chance** regression 1 and 2
- process 1 and 2
- evaluate 1 and 2
- summarize 1 and 2
- visualize 1 and 2

## 9.3. Related notes

•

## 9.4. Instructions

Find a dataset suitable for regression. We recommend a dataset from the UCI repository. Remember that you can use the filters to choose a dataset that is well-suited for regression.

### 9.4.1. 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, units, size of dataset, etc)
2. Write your own description of what the prediction task it, why regression is appropriate.
3. Fit a linear model on all numerical features with 75% training data.
4. Test it on 25% held out test data and measure the fit with two metrics and one plot
5. Inspect the model to answer:
  - What to the coefficients tell you?
  - What to the residuals tell you?
6. Repeat the split, train, and test steps 5 times.
  - Is the performance consistent enough you trust it?
7. Interpret the model and its performance in terms of the application. Some questions you might want to answer in order to do this include:
  - do you think this model is good enough to use for real?
  - is this a model you would trust?

[Skip to main content](#)

- 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.

## 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](#)

## Upcoming Checks

- Portfolio Check 1 is due October 16

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>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

## 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:
```

[Skip to main content](#)

```

# 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:
        # The arguments for the `git add` command (see the paragraph below for more info)
        # 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

```

The summary of for the `part` or whole submission, should match the skills to the chapters. Which prompt you're addressing is not important, the prompts are a *starting point* not the end goal of your portfolio.

## Data Files

Also note that for your portfolio to build, you will have to:

- include the data files in the repository and use a relative path OR
- load via url

using a full local path(eg that starts with `///file:`) **will not work** and will render your portfolio unreadable.

## Structure of plain markdown

Use a heading like this:

```
# Heading of page
## Heading 2
### Heading 3
```

in the file and it will appear in the sidebar.

You can also make text *italic* or **bold** with either `*asterics*` or `__underscores__` with `_one for italic_` or `**two for bold**` in either case

## File Naming

It is best practice to name files without spaces. 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

```
Here is a note!  
```
```

```
```{warning}  
Here is a warning!  
```
```

```
```{tip}  
Here is a tip!  
```
```

```
```{margin}  
Here is a margin note!  
```
```

For a complete list of options, see the [sphinx-book-theme](#) documentation.

## Links

Markdown syntax for links

```
[text to show](path/or/url)
```

## Configurations

Things like the menus and links at the top are controlled as `settings`, in [\\_config.yml](#). The following are some things that you might change in your configuration file.

### Show errors and continue

To show errors and continue running the rest, add the following to your configuration file:

```
# Execution settings  
execute:  
    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.

If your goal is, for example, a B+ (you need 5 level 3s) you could only do 1-2 skills per portfolio check (there are 4 checks).

## Earning Level 3s

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

### 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.

### 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.

## 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:  $7+5 = 12$ ,  $3+1 = 4$ )

- 43 (carried one into wrong column,  $7 + 5 = 12$ ,  $1+2 = 3$ ,  $3+1 = 3$ )

## Long single analysis

Collect data from multiple sources, prepare each for analysis, and merge them together then do some exploratory data analysis. Describe each step, interpret all outputs, and put the analysis in context of the Data Science Process.

This would be one long notebook that covers 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.

## 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.

Tip

If you achieve with t  
repre next p

## Process level 3

Process level 3 is a little different than most of the others. You may be able to work it into an analysis notebook, but likely, you'll need to do one of the following.

### Data Science Pipeline Comparisons

Find two different sources that describe the data science pipeline or lifecycle. Write a blog style post that discusses their differences and hypothesizes about why they may be different? Are they for different audiences? Is one domain specific? How do they emphasize different modeling tasks? Include a Recommendation for when you think each one is better

### Write a short story

Write a short story that explains the concepts of data science to demonstrate your understanding of process.

### Media Review

Watch/listen/read to an episode of a high quality<sup>[1]</sup> podcast or other type of media and write a blog style summary and review. Highlight what you learned and how it relates to topics covered in class.

Approved Media:

- Pod of Asclepius, Fall Series: The Philosophy of Data Science
- Chapter 1 & 2 of Think like a Data Scientist in particular, if you think these would be helpful to assign as reading or teach from at the beginning of the semester next year.
- Algorithms of Oppression (book)
- Weapons of Math Destruction (book)
- Coded Bias (film, available on netflix & PBS)

---

<sup>[1]</sup> 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.

## Check 3 Ideas

For Check 4, all of the prompts from check 1 & 2 apply, plus the following additional prompts.

If you have other ideas, you can also ask and those are likely possible.

## Organize your knowledge

Develop some sort of visual aid that demonstrates how you understand some aspect(s) of data science working. Think of this as something that future students could use to help them learning, so assume prior knowledge topics covered earlier than the one you are demonstrating.

## Extend any assignment

Assignments 7-12 are most relevant because they leave room to extend and ask new questions.

If you both reflect on what you had trouble with and extend you could earn level 2 and 3.

## Try alternative libraries/ tools

One option for workflow level 3 is to use other data science skills and reflect on how what we have learned so far helped you learn a new set of tool as an alternative way to do things.

## Try feature engineering or representation learning

Try different transformations and see how they impact how well a model performs. This could be using `sklearn.feature_extraction` tools or trying different types of neural network layers at the beginning.

## FAQ

This section will grow as questions are asked and new content is introduced to the site. You can submit questions:

- via e-mail to Dr. Brown ([brownsarahm](#)) or TA
- via Prismia.chat during class
- by creating an [issue](#)

## Syllabus and Grading FAQ

### How much does assignment x, class participation, or a portfolio check weigh in my grade?

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 ...?

multiple assignments, contact Dr. Brown.

## 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.

GitHub has [strong rules about authentication](#) You need to use SSH with a public/private key; HTTPS with a Personal Access Token or use the [GitHub CLI auth](#)

## 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

⌚ feedback had recent pushes 1 minute ago

[Compare & pull request](#)[main](#) [branches](#) [tags](#)[Go to file](#)[Add file](#)[Code](#)

brownsarahm update toc to include notebook

|                                |                                      |
|--------------------------------|--------------------------------------|
| <a href="#">.github</a>        | correct path for jupytext conversion |
| <a href="#">about</a>          | mvoe notebook                        |
| <a href="#">template_files</a> | convert notebooks to md              |
| <a href="#">.gitignore</a>     | merge gh changes and ignore          |
| <a href="#">README.md</a>      | 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](#)

**19 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](#)

| Branch          | Commit Message                       | Time Ago       |
|-----------------|--------------------------------------|----------------|
| main            | ✓ a6f7f45 15 minutes ago             | 14 commits     |
| feedback        | 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     |
| gh-pages        |                                      |                |
| someOtherBranch | 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](#)

**19 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](#)

 **brownsarahm** Merge pull request #1 from rhodyprog4ds/main ... [f301d90](#) 16 minutes ago [15 commits](#)

| File           | Commit Message                       | Time Ago       |
|----------------|--------------------------------------|----------------|
| .github        | correct path for jupytext conversion | 17 hours ago   |
| about          | mvoe notebook                        | 20 minutes ago |
| template_files | convert notebooks to md              | 17 hours ago   |
| ...            | merge gh changes and ignore          | 3 days ago     |

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.

[Skip to main content](#)

|                                      |                      |         |
|--------------------------------------|----------------------|---------|
| 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
```

[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").

The screenshot shows a GitHub repository page for 'rhodyprog4ds / portfolio-brownsarahm'. The 'Code' tab is selected. At the top, it says 'This branch is 11 commits behind main.' Below the code editor, there's a list of commits:

| Author      | Commit Message                       | Date       | Commits   |
|-------------|--------------------------------------|------------|-----------|
| brownsarahm | Setting up GitHub Classroom Feedback | 3 days ago | 3 commits |
|             | .github                              | 3 days ago |           |
|             | about                                | 3 days ago |           |
|             | template_files                       | 3 days ago |           |
|             | .gitignore                           | 3 days ago |           |
|             | README.md                            | 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

822cf5 3 days ago 3 commits

.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

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.

Then change the compare [feedback](#) on the right to [main](#). Once you do that the page will change to the "Open a Pull[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 indicates that the branches are 'Able to merge'. Below this, a user profile picture is shown next to the title 'Feedback'. There are 'Write' and 'Preview' tabs, along with a rich text editor toolbar with icons for bold, italic, underline, etc. A text area for leaving a comment is present, with placeholder text 'Leave a comment'. At the bottom, there is a note to 'Attach files by dragging & dropping, selecting or pasting them.' and a 'M+' button.

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

[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; val 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 item1 and item2                                  |
| (param)                | function calls                                                                 |
| (item1, item2)         | defining a tuple of item1 and item2                                            |
| {item1, item2}         | defining a set of item1 and item2                                              |
| {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 summing across 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
```

[Skip to main content](#)

```
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

[Skip to main content](#)

- 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

# 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

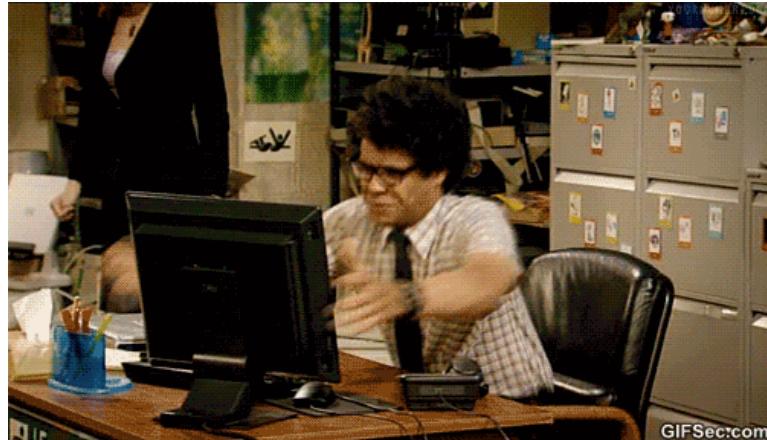
This section is for materials that are not specific to this course, but are likely useful. They are not generally required readings or installs, but are options or advice I provide frequently.

### 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.

### Where are your help tools?

In Python and Jupyter notebooks, what help tools do you have?

[Skip to main content](#)

A new book  
programming  
Brain As of  
by clicking  
contents se

# 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

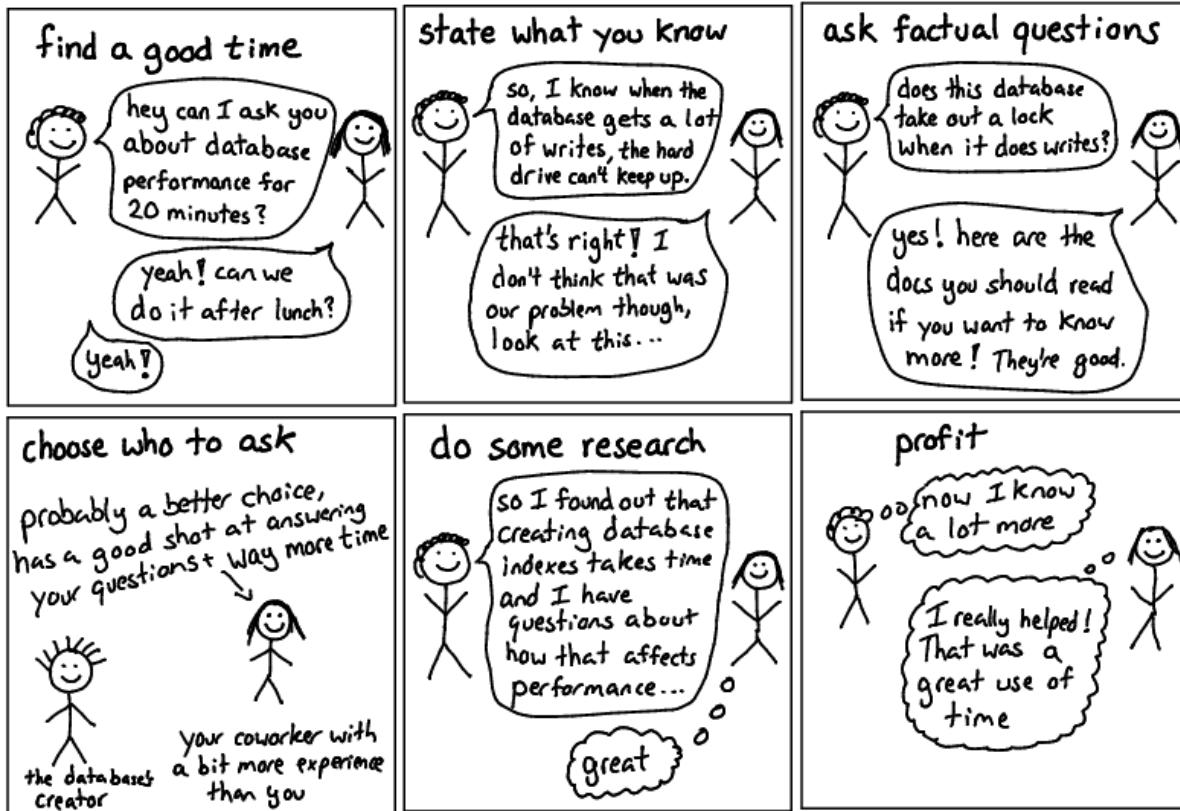
In portfolios, your goal is to extend and apply the concepts taught in class and practiced in assignments to solve more realistic problems. You may also reflect on your learning in order to demonstrate deep understanding. These will require significant reading beyond what we cover in class.

# Getting Help with Programming

## Asking Questions

JULIA EVANS  
@b0rk

### asking good questions



One of my favorite resources that describes how to ask good questions is [this blog post](#) by Julia Evans, a developer who writes comics about the things she learns in the course of her work and publisher of wizard zines.

## Describing what you have so far

Stackoverflow is a common place for programmers to post and answer questions.

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.

## 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](#)

Not

A fun  
debuc

To understand what the traceback is, how to read one, and common examples, see this post on Real Python.

One thing to try, is [friendly traceback](#) a python package that is designed to make that error message text more clear and help you figure out what to do next.

### Ram Token Opportunity

If you try out friendly traceback and find it helpful, add a testimonial here. using

```
```{epigraph}
```

## Terminals and Environments

### Why all this work?

Managing environments is **one of the hardest parts of programming** so, as instructors, we often design our courses around not having to do it. In this class, however, I'm choosing to take the risk and help you all through beginning to manage your own environments.

These issues will be the most painful in the course, I promise.

I think it's worth this type of pain though, because all 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 k  
teach  
so in  
new c  
that v  
under  
this s  
follow  
and th

If, for exan  
have neve  
and you're  
will be hur  
at that poi

# 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.