

About this Book

Contents

Syllabus

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

Notes

- 1. Welcome to Programming to Data Science
- 2. Jupyter Notebook Tour & Python Review
- 3. Getting help, object inspection, loading data
- 4. Pandas DataFrames
- 5. More Loading Data, Indexing, and Iterables
- 6. Exploratory Data Analysis
- 7. Visualization
- 8. Exploratory Data Analysis
- 9. Reshaping Data
- 10. More Reshaping
- 11. Missing Data and Inconsistent coding
- 12. Building Datasets From multiple Sources
- 13. Reviewing Merges & Databases
- 14. Web Scraping
- 15. Intro to Machine learning
- 16. Interpetting and Evaluating Naive Bayes
- 17. Making Predictions in Generative Model
- 18. Midsemester feedback and Decision Trees
- 19. Decision Tree Setting and more Evaluation
- 20. Linear Regression
- 21. Interprettting Regression
- 22. Clustering
- 23. Clustering
- 24. Evaluating Clustering
- 25. ML Task Review and Cross Validation
- 26. SVM and Parameter Optimizing
- 27. Model Comparison
- 28. Model Selection
- 29. Learning Curves
- 30. Intro to NLP- representing text data
- 31. More NLP & Solving problems with ML
- 32. Neural Networks
- 33. Predicting with Neural Networks
- 34. Review, IDEA, & Preparing for Deep Learning
- 35. Neural Networks with Keras
- 36. Convolutional Neural Netwrks

Assignments

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

Portfolio

- Portfolio Dates and Key Facts
- Submission Introductions
- Formatting Tips
- Portfolio Check 1 Ideas
- Check 2 Ideas

- [Check 4 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](#)
- [Letters to Future students](#)

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

This class meets MWF 3-3:50pm in Chafee Social Sci Center 235.

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

[Course Calendar on BrightSpace](#)



Tip

[subscribe to that calendar](#) in your favorite calendar application

Navigating the Sections

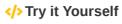
The Syllabus section has logistical operations for the course broken down into sections. You can also read straight through by starting in the first one and navigating to the next section using the arrow navigation at the end of the page.

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

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

Reading each page

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



Notes will have exercises marked like this



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



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.



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



Chances to earn ram tokens are highlighted this way.



Questions that are asked in class, but unanswered at that time will be answered in the notes and marked with a box like this. Short questions will be in the margin note

Basic Facts

About this course

Data science exists at the intersection of computer science, statistics, and machine learning. 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.

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 substantial 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 this syllabus

This syllabus is a *living* document and accessible from BrightSpace, as a pdf for download directly online at <https://rhodyprog4ds.github.io/BrownFall21/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 You can view the date of changes and exactly what changes were made on the Github [commit history](#) page.

Creating an [issue](#) is also a good way to ask questions about anything in the course it will prompt additions and expand the FAQ section.

About your instructor

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

Dr. Sarah M Brown is a second year 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. You can learn more about me at my [website](#) or my research on my [lab site](#).

You can call me Professor Brown or Dr. Brown, I use she/her pronouns.

The best way to contact me is e-mail or an issue on an assignment repo. For more details, see the [Communication Section](#)

Tools and Resources

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

All of the tools and resources below are either:

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

BrightSpace

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

This is also where your grades will appear and how I will post announcements.

For announcements, you can [customize](#) how you receive them.

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 Manual

The course manual will have content including the class policies, scheduling, class notes, assignment information, and additional resources. This will be linked from Brightspace and available publicly online at <https://rhodyprog4ds.github.io/BrownFall21/>. Links to the course reference text and code documentation will also be included here in the assignments and class notes.

GitHub Classroom

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

Programming Environment

This is 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](#)

Important

TL;DR [\[1\]](#)

- check Brightspace
- Log in to Prismia Chat
- Make a GitHub Account
- Install Python
- Install Git

Note

Seeing the BrightSpace site requires logging in with your URI SSO and being enrolled in the course

⚠ Warning

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

Recommendation:

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

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

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

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

5. Next install Anaconda's Python from the website provided by the instructor and use the top download link under the Linux options.
6. You will then see a .sh file in your downloads, move this into your Linux files.
7. Make sure you are in your home directory (something like `home/YOURUSERNAME`), do this by using the `pwd` command.
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 only, Fall 2021 is in person)

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

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

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

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

[1] Too long, didn't read.

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.

>Note

all Git instructions will be given as instructions for the command line interface and GitHub specific instructions via the web interface. You may choose to use GitHub desktop or built in IDE tools, but the instructional team may not be able to help.

💡 A tip from Dr. Brown

I use [atom](#), but I decided to use it by downloading both Atom and Sublime and trying different things in each for a week. I liked Atom better after that and I've stuck with it since. I used Atom to write all of the content in this syllabus. VSCode will also work, if needed

Schedule

The course will meet MWF 3-3:50pm in Chafee Social Sci Center 235. 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.

Programming assignments that will be due each week Tuesday by 11:59pm.

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, Naive Bayes, classification performance metrics	[classification, evaluate]
7	decision trees, cross validation	[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.

Note

On the [Course Calendar on BrightSpace](#) page you can get a feed link to add to the calendar of your choice by clicking on the subscribe (star) button on the top right of the page. Class is for 1 hour there because of Brightspace/zoom integration limitations, but that calendar includes the zoom link.

	skill	Level 1	Level 2	Level 3
keyword				
python	pythonic code writing	python code that mostly runs, occasional pep8 adherence	python code that reliably runs, frequent pep8 adherence	reliable, efficient, pythonic code that consistently adheres to pep8
process	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
access	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
construct	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
summarize	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
visualize	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
prepare	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
classification	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
regression	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
clustering	Clustering	describe what clustering is	apply basic clustering	apply multiple clustering techniques, and interpret results
evaluate	Evaluate model performance	Explain basic performance metrics for different data science tasks	Apply and interpret basic model evaluation metrics to a held out test set	Evaluate a model with multiple metrics and cross validation
optimize	Optimize model parameters	Identify when model parameters need to be optimized	Optimize basic model parameters such as model order	Select optimal parameters based of multiple quantitative criteria and automate parameter tuning
compare	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
representation	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 inappropriately 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
workflow	use industry standard data science tools and workflows to solve data science problems	Solve well structured fully specified problems with a single tool pipeline	Solv well-structured, open-ended problems, apply common structure to learn new features of standard tools	Independently scope and solve realistic data science problems OR independently learn related tools and describe strengths and weaknesses of common tools

Assignments and Skills

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

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	# Assignments
keyword														
python	1	1	0	1	1	0	0	0	0	0	0	0	0	4
process	1	1	0	0	0	0	1	1	1	1	1	0	0	7
access	0	1	1	1	1	0	0	0	0	0	0	0	0	4
construct	0	0	0	0	1	0	1	1	0	0	0	0	0	3
summarize	0	0	1	1	1	1	1	1	1	1	1	1	1	11
visualize	0	0	1	1	0	1	1	1	1	1	1	1	1	10
prepare	0	0	0	1	1	0	0	0	0	0	0	0	0	2
classification	0	0	0	0	0	1	1	0	0	1	0	0	0	3
regression	0	0	0	0	0	0	0	1	0	0	1	0	0	2
clustering	0	0	0	0	0	0	0	0	1	0	1	0	0	2
evaluate	0	0	0	0	0	0	1	1	0	1	1	0	0	4
optimize	0	0	0	0	0	0	0	0	0	1	1	0	0	2
compare	0	0	0	0	0	0	0	0	0	0	1	0	1	2
representation	0	0	0	0	0	0	0	0	0	0	0	1	1	2
workflow	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
python	reliable, efficient, pythonic code that consistently adheres to pep8	1	1	0	1
process	Compare different ways that data science can facilitate decision making	0	1	1	1
access	access data from both common and uncommon formats and identify best practices for formats in different contexts	1	1	0	1
construct	merge data that is not automatically aligned	1	1	0	1
summarize	Compute and interpret various summary statistics of subsets of data	1	1	0	1
visualize	generate complex plots with pandas and plotting libraries and customize with matplotlib or additional parameters	1	1	0	1
prepare	apply data reshaping, cleaning, and filtering manipulations reliably and correctly by assessing data as received	1	1	0	1
classification	fit and apply classification models and select appropriate classification models for different contexts	0	1	1	1
regression	fit and explain regularized or nonlinear regression	0	1	1	1
clustering	apply multiple clustering techniques, and interpret results	0	1	1	1
evaluate	Evaluate a model with multiple metrics and cross validation	0	1	1	1
optimize	Select optimal parameters based of multiple quantitative criteria and automate parameter tuning	0	0	1	1
compare	Evaluate tradeoffs between different model comparison types	0	0	1	1
representation	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
workflow	Independently scope and solve realistic data science problems OR independently learn related tools and describe strengths and weaknesses of common tools	0	0	1	1

Grading

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

Principles of Grading

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

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

- Earning a C in this class means you have a general understanding of Data Science and could participate in a basic conversation about all of the topics we cover. I expect everyone to reach this level.
- Earning a B means that you could solve simple data science problems on your own and complete parts of more complex problems as instructed by, for example, a supervisor in an internship or entry level job. This is a very accessible goal, it does not require you to get anything on the first try or to explore topics on your own. I expect most students to reach this level.

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

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

How it works

There are 15 skills that you will be graded on in this course. While learning these skills, you will work through a progression of learning. Your grade will be based on earning 45 achievements that are organized into 15 skill groups with 3 levels for each.

These map onto letter grades roughly as follows:

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

You will have at least three opportunities to earn every level 2 achievement. You will have at least two opportunities to earn every level 3 achievement. You will have three types of opportunities to demonstrate your current skill level: participation, assignments, and a portfolio.

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

- To earn level 1 achievements, you will need to demonstrate basic awareness of the required concepts and know approximately what to do, but you may need specific instructions of which things to do or to look up examples to modify every step of the way. You can earn level 1 achievements in class, assignments, or portfolio submissions.
- To earn level 2 achievements you will need to demonstrate understanding of the concepts and the ability to apply them with instruction after earning the level 1 achievement for that skill. You can earn level 2 achievements in assignments or portfolio submissions.
- To earn level 3 achievements you will be required to consistently execute each skill and demonstrate deep understanding of the course material, after achieving level 2 in that skill. You can earn level 3 achievements only through your portfolio submissions.

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

Participation

While attending synchronous class sessions, there will be understanding checks and in class exercises. Completing in class exercises and correctly answering questions in class can earn level 1 achievements. In class questions will be administered through the classroom chat platform Prismia.chat; these records will be used to update your skill progression. You can also earn level 1 achievements from adding annotation to a section of the class notes.

Assignments

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

Assignments will each evaluate certain skills. After your assignment is reviewed, you will get qualitative feedback on your work, and an assessment of your demonstration of the targeted skills.

Portfolio Checks

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

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

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

TLDR

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

Detailed mechanics

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

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

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

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

The letter grade can be computed as follows

⚠ Warning

If you will skip an assignment, please accept the GitHub assignment and then close the Feedback pull request with a comment. This way we can make sure that you have support you need.

💡 Note

In this example, you will have also achieved level 1 on all of the skills, because it is a prerequisite to level 2.

```

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

```

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

```

compute_grade(15,15,15)

'A'

compute_grade(14,14,14)

'C-'

```

Or use `assert` to test it formally

```

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

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

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

```

Late work

Late assignments will not be graded. Every skill will be assessed through more than one assignment, so missing assignments occasionally not necessarily hurt your grade. If you do not submit any assignments that cover a given skill, you may earn the level 2 achievement in that skill through a portfolio check, but you will not be able to earn the level 3 achievement in that skill. If you submit work that is not complete, however, it will be assessed and receive feedback. Submitting pseudocode or code with errors and comments about what you have tried could earn a level 1 achievement. Additionally, most assignments cover multiple skills, so partially completing the assignment may earn level 2 for one, but not all. Submitting *something* even if it is not perfect is important to keeping conversation open and getting feedback and help continuously.

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

In this issue, include:

1. A new deadline proposal
2. What additional work you plan to add
3. Why the extension is important to your learning
4. Why the extension will not hinder your ability to complete the next assignment on time.

This request should be no more than 7 sentences.

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

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 or assignment 13.

Getting an A Without Perfection

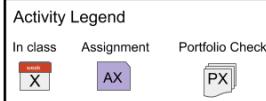
Note

You may visit office hours to discuss assignments that you did not complete on time to get feedback and check your own understanding, but they will not count toward skill demonstration.

Map to an A

How Achievements were earned

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



Other Activities

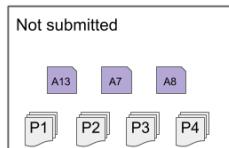
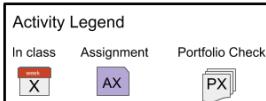
- 1 Attended, but did not understand
- 2 Submitted, but incorrect
- 3 Missed class
- 4 Not submitted
- 5 Submitted, but incorrect
- 6 Not submitted
- 7 Submitted, but incorrect
- 8 Not submitted
- 9 Submitted, but incorrect
- 10 Attended, but all level 1 complete
- 11 Attended, but all level 1 complete
- 12 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.

Getting a B with minimal work

Map to a B easily

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

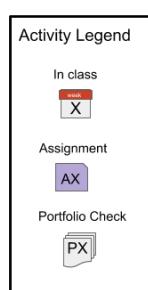


In this example, the student earned all level 1 achievements in class and all level 2 on assignments. This student was content with getting a B and chose to not submit a portfolio.

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.

Ram Tokens

Ram Tokens in this course will be used as a currency for extra effort. You can earn Ram Tokens by doing work that supports your learning or class activities, but do not directly demonstrate achievements. You can spend Ram Tokens to get extra grading. This will be mostly applicable to Portfolio Checks. In Checks 3 & 4, some achievements will not be eligible for grading as per the [table](#). However, you can exchange Ram Tokens to make more achievements eligible for assessment. This system rewards you for putting in consistent effort, even if it takes you many tries to understand a concept.

To accumulate Ram Tokens, you submit a 'Deposit' to the [Ram Token Bank: http://drsmb.co/amtoken](http://drsmb.co/amtoken) with a link to what you did to earn a token. To apply Ram tokens for extra grading, submit the same form, with a link to the assignment and add the Ramtoken label to the Feedback PR.

Grading Policies

Late Work

Late assignments will not be graded. Every skill will be assessed through more than one assignment, so missing assignments occasionally not necessarily hurt your grade. If you do not submit any assignments that cover a given skill, you may earn the level 2 achievement in that skill through a portfolio check, but you will not be able to earn the level 3 achievement in that skill. If you submit work that is not complete, however, it will be assessed and receive feedback. Submitting pseudocode or code with errors and comments about what you have tried could earn a level 1 achievement. Additionally, most assignments cover multiple skills, so partially completing the assignment may earn level 2 for one, but not all. Submitting *something* even if it is not perfect is important to keeping conversation open and getting feedback and help continuously.

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

In this issue, include:

1. A new deadline proposal
2. What additional work you plan to add
3. Why the extension is important to your learning
4. Why the extension will not hinder your ability to complete the next assignment on time.

This request should be no more than 7 sentences.

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

Regrading

Re-request a review on your Feedback Pull request.

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

For specific questions, reply to a specific comment.

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

Support

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. Appointments and locations will be visible in the TutorTrac system on September 14th, 2020. The TutorTrac application is available through [URI Microsoft 365 single sign-on](#) and by visiting [aec.uri.edu](#). More detailed information and instructions can be found on the AEC tutoring page.
- **Academic Skills Development** resources helps students plan work, manage time, and study more effectively. In Fall 2020, all Academic Skills and Strategies programming are offered both online and in-person. UCS160: Success in Higher Education is a one-credit course on developing a more effective approach to studying. Academic Consultations are 30-minute, 1 to 1 appointments that students can schedule on Starfish with Dr. David Hayes to address individual academic issues. Study Your Way to Success is a self-guided web portal connecting students to tips and strategies on studying and time management related topics. For more information on these programs, visit the Academic Skills Page or contact Dr. Hayes directly at davidhayes@uri.edu.
- 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](#).

General URI Policies

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](#). There you will also find people and resources to help.

Disability Services for Students Statement:

Note

You may visit office hours to discuss assignments that you did not complete on time to get feedback and check your own understanding, but they will not count toward skill demonstration.

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, or emailing: dss@etal.uri.edu. We are available to meet with students enrolled in Kingston as well as Providence courses.

Academic Honesty

Students are expected to be honest in all academic work. A student's name on any written work, quiz or exam shall be regarded as assurance that the work is the result of the student's own independent thought and study. Work should be stated in the student's own words, properly attributed to its source. Students have an obligation to know how to quote, paraphrase, summarize, cite and reference the work of others with integrity. The following are examples of academic dishonesty.

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

URI COVID-19 Statement

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

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

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

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

Course Communications

Help Hours

```
/tmp/ipykernel_3267/2146052215.py:1: FutureWarning: this method is deprecated in
favour of `Styler.hide(axis='index')`:
help_df.style.hide_index()
```

Day	Time	Location	Host
Monday	9:30:00 AM-10:30 AM	in person Tyler Hall 140	Chamudi
Monday	12:30:00 PM-2:00 PM	in person Tyler Hall 139	Chamudi
Tuesday	2:00 PM-3:00 PM	gather.town	Sarah
Wednesday	4:00:00 PM-5:00 PM	in person Tyler Hall 139	Chamudi
Wednesday	1:30:00 PM-3:00 PM	in person Tyler Hall 140	Chamudi
Wednesday	7:00:00 PM-8:30	gather.town	Sarah
By appointment	scheduling link on Brightspace	in person Tyler 134	Sarah

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

To reach out, By usage

```
/tmp/ipykernel_3267/3027175348.py:2: FutureWarning: this method is deprecated in
favour of `Styler.hide(axis='index')`:
display(HTML(df.style.hide_index()._repr_html_()))
```

usage	platform	area	note
in class	prismia	chat	outside of class time this is not monitored closely
any time	prismia	message board	for discussion with peers
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	github	pull request on website	remember to request ram tokens if applicable
matters that don't fit into another category	e-mail	brownsarahm@uri.edu to	remember to include '[CSC310]' or '[DSP310]' (note 'verbatim' no space)

Note

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

By Platform

Use e-mail for

```
/tmp/ipykernel_3267/2135006347.py:3: FutureWarning: this method is deprecated in  
favour of `Styler.hide(axis='index')`  
display(HTML(data.drop(columns='platform').style.hide_index().__repr_html__()))
```

usage	area	note
matters that don't fit into another category	to brownsarahm@uri.edu	remember to include '[CSC310]' or '[DSP310]' (note 'verbatim' no space)

Use github for

```
/tmp/ipykernel_3267/2135006347.py:3: FutureWarning: this method is deprecated in  
favour of `Styler.hide(axis='index')`  
display(HTML(data.drop(columns='platform').style.hide_index().__repr_html__()))
```

usage	area	note
private questions to your assignment	issue on assignment repo	eg bugs in your code"
for general questions that can help others	issue on course website	eg what the instructions of an assignment mean or questions about the syllabus
to share resources	pull request on website	remember to request ram tokens if applicable

Use prismia for

```
/tmp/ipykernel_3267/2135006347.py:3: FutureWarning: this method is deprecated in  
favour of `Styler.hide(axis='index')`  
display(HTML(data.drop(columns='platform').style.hide_index().__repr_html__()))
```

usage	area	note
in class	chat	outside of class time this is not monitored closely
any time	message board	for discussion with peers
any time	download transcript	use after class to get preliminary notes eg if you miss a class

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.

Note

Whether you use CSC or DSP does not matter.

1. Welcome to Programming to Data Science

Today's goals:

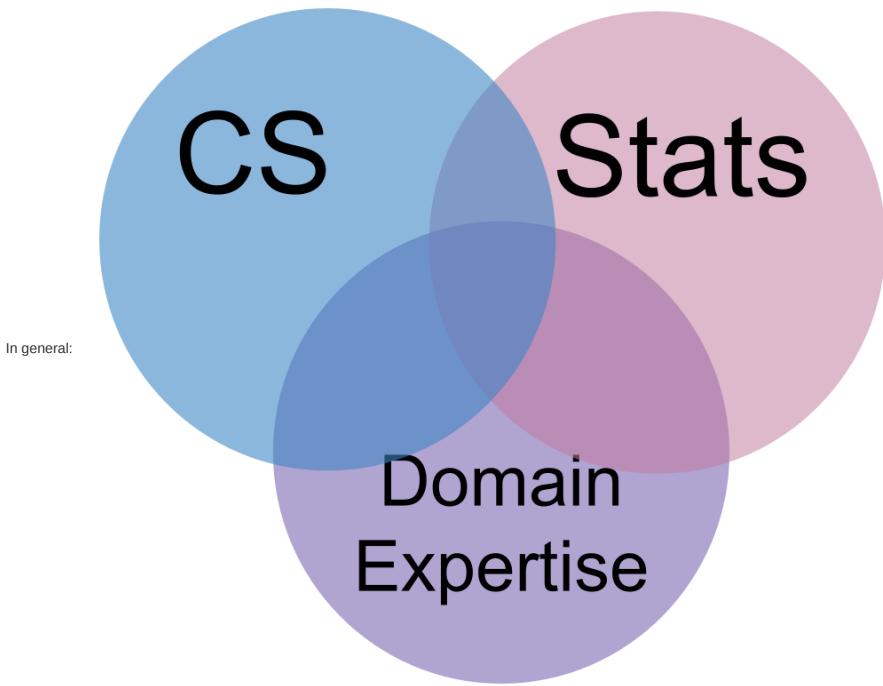
1. Operate tools for in-class participation
2. Understand what Data Science is, in broad terms
3. Understand the syllabus (grading, topics covered, schedule, etc)
4. Understand how to learn in this course

1.1. Prismia Chat

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

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

1.2. What is Data Science

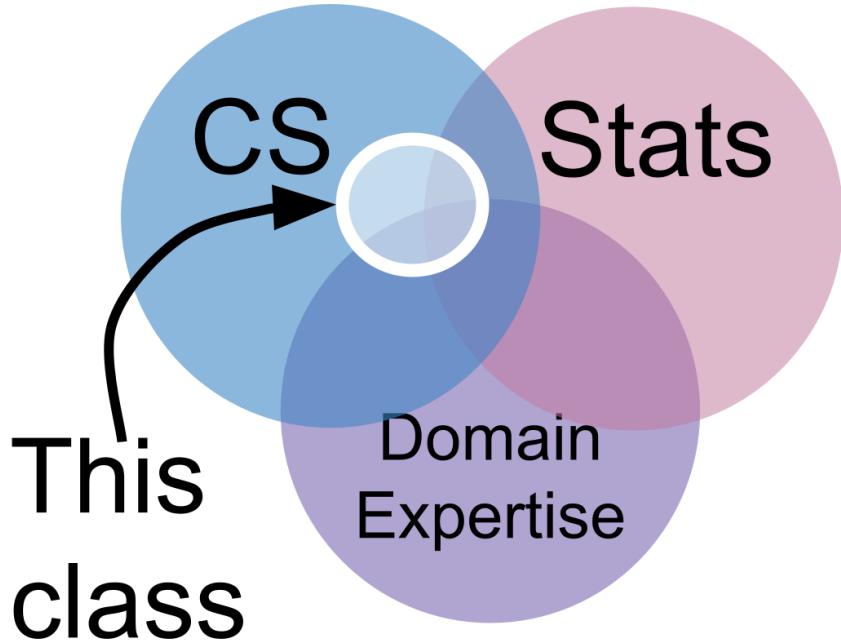


statistics is the type of math we use to make sense of data. Formally, a statistic is just a function of data.

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

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

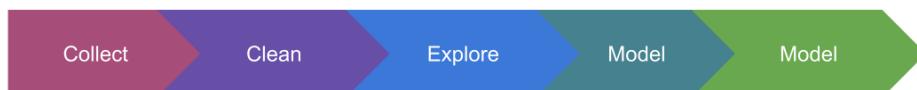
For this class



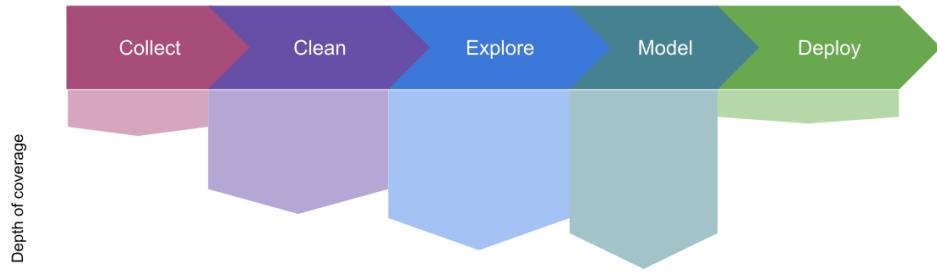
We'll focus on the programming as our main means of studying data science, but we will use bits of the other parts. In particular, you're encouraged to choose datasets that you have domain expertise about, or that you want to learn about.

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

1.2.1. How does data science happen?



1.2.2. how we'll cover it, 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

1.2.2.1. how we'll cover it in, time



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

1.3. How this class will work

- today is an exception
- in general we'll be live coding

Let's look at the [syllabus](#)

Read carefully to make sure you understand the grading; it's not typical points and an average.

Class is designed to avoid this:



1.4.

1.5. Learning Cycle

Julia Evans

@b0rk

Reply
Copy link

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

5:35 PM · Jun 11, 2021

4.2K

[Read 92 replies](#)

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

1.6. Check your understanding of the syllabus

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

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

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

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

1. When & what are you expected to read for this class?

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

2. Jupyter Notebook Tour & Python Review

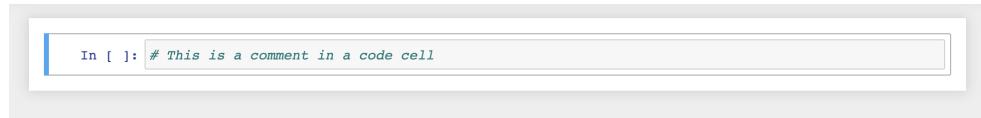
2.1. A jupyter notebook tour

Launch a [jupyter notebook](#):

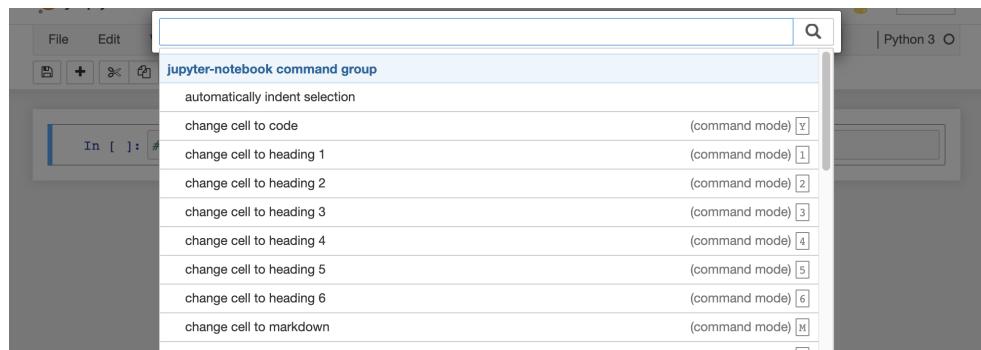
- on Windows, use anaconda terminal
- on Mac/Linux, use terminal

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

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



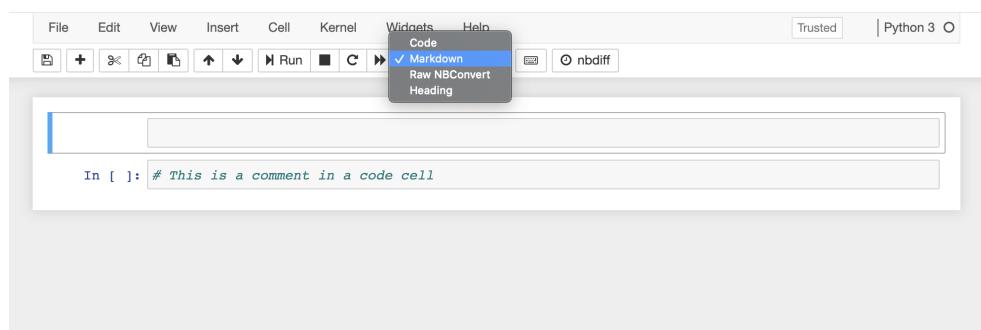
When you press a key in command mode it works like a shortcut. For example **p** shows the command search menu.



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. The border is green in edit mode



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



++

This is a markdown cell

- we can make

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

2.1.1. Notebook Reminders

Blue border is command mode, green border is edit mode

use Escape to get to command mode

Common command mode actions:

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

use enter/return to get to edit mode

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

A screenshot of a Jupyter Notebook cell. The input line contains the expression `4+6`. The output line shows the result `10`.

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

A screenshot of a Jupyter Notebook cell. The input lines contain `name = 'sarah'`, `4+5`, and `name * 3`. The output line shows the string `'sarahsarahsarah'`.

Note

For a little more python review, see my [2020 CSC310 notes](#) this is just enough for this assignment.

2.2. Just enough Git for Assignment 1

2.2.1. Assignment 1:

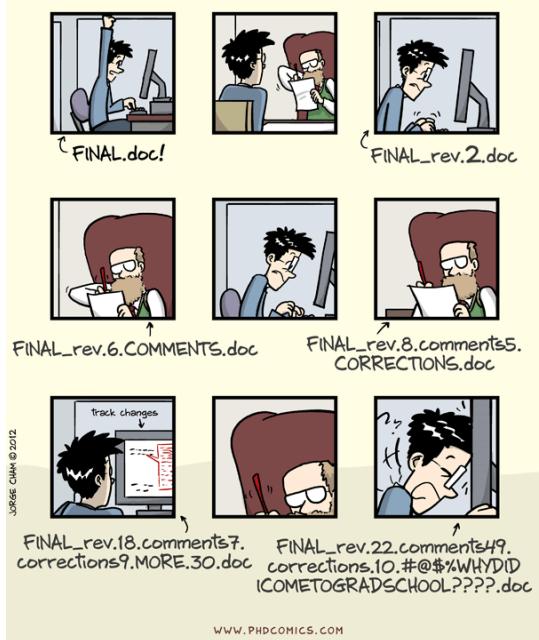
Goals for this assignment

- setup your portfolio
- check that you understand the grading
- review Python basics
- practice with git and GitHub

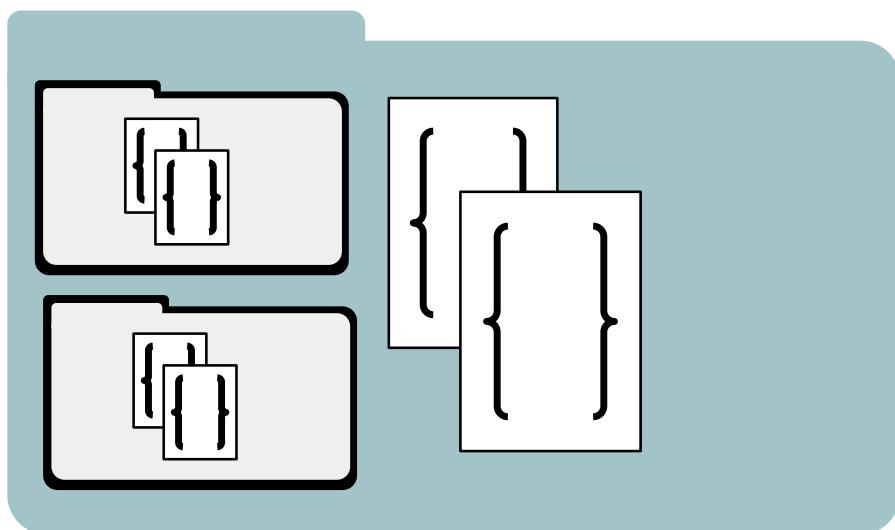
2.2.2. Why Version control

We often want to keep track of the different versions in case we want to go back, but this can be painful:

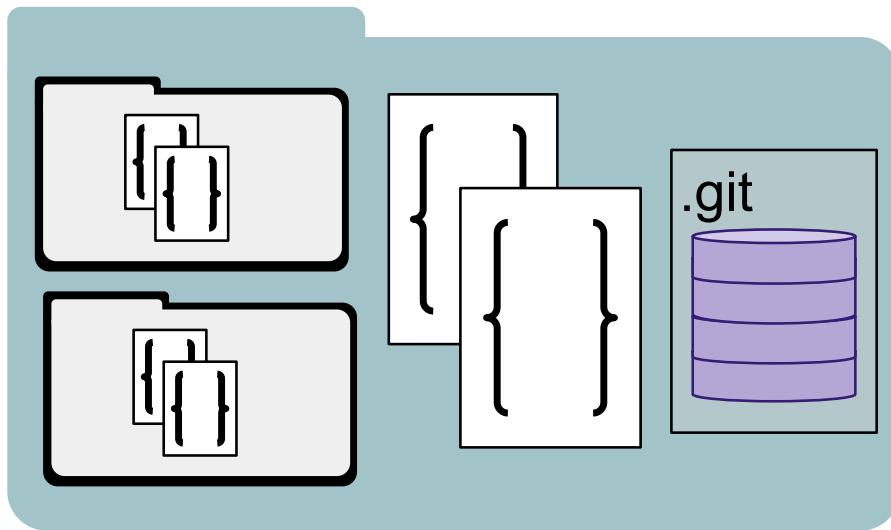
"FINAL".doc



We typically organize projects in folder

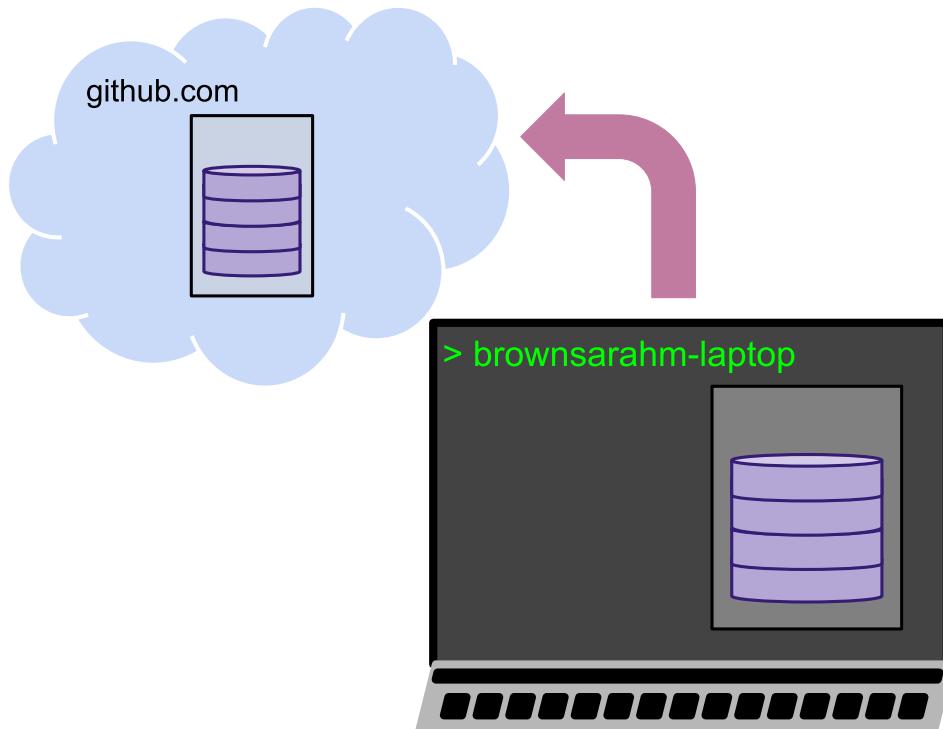


A [repository](#) is a folder with a hidden directory named `.git`.



The `git` application manages that hidden directory, we don't write to it directly, which is why we keep it hidden.

Git is a distributed system, you have a local version and a remote version.



Once a repository exists on GitHub, we get a local copy by cloning it after we get its address from the GitHub interface, by clicking on the green code button that is below the menu area to the right. It's at the top right corner of the list of files in the repository.

brownsarahm update toc to include notebook

- .github correct path for jupytext conversion
- about move notebook
- template_files convert notebooks to md
- .gitignore merge gh changes and ignore
- README.md Initial commit

Clone with HTTPS Use SSH
Use Git or checkout with SVN using the web URL.
<https://github.com/rhodyprog4ds/portfolio-brownsarahm>

Open with GitHub Desktop **Download ZIP**

For this part, use GitBash on windows or terminal otherwise: If you set up a Personal Access Token you can use the https version

After `cd/to/where/you/want/your/repo/locally:`

```
git clone https://github.com/rhodyprog4ds/portfolio-example
```

If you set up ssh keys you use that instead

```
git clone git@github.com:rhodyprog4ds/portfolio-example.git
```

Once it's cloned, then you can navigate into the new folder:

```
cd portfolio-example
```

Then you can change files, for example adding to the intro.

Some common actions in Git, you'll want.

Check on the status of your repository:

```
git status
```

Add files to the staging area:

```
git add filename
```

Add all changes to the staging area:

```
git add .
```

Commit your changes to the repository:

```
git commit -m 'a message that will help your future self know what this part is'
```

Push your changes to GitHub

```
git push
```

Pull changes from GitHub

```
git pull
```

You can also go through these same basic steps: add, commit, push

2.3. More on git

- [GitHub Hello World](#)
- [Software Carpentry Git Novice Lesson](#)

Also, in Spring 2022, I'm teaching a section of CSC392: Topics in Computing, Introduction to Computer Systems, that will cover tools of the trade (git, bash, etc) and how they all work in great detail.

2.4. More on Python

Read [Pep 8](#) to see what good style in Python is.

3. Getting help, object inspection, loading data

3.1. First, Don't Worry members

Class Response Summary:



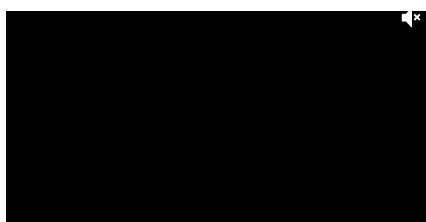
EVERYBODY CALM DOWN

I GOT THIS

[funny_CS memes](#)



Hey, Don't even worry about it.



USES CLASSIC MEME FORMAT



PEOPLE LIKE IT



3.2. Getting Help in Jupyter

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

Given this code excerpt, how could you print out "Sarah_Brown"?

```
first = 'Sarah'  
last = 'Brown'
```

We can use jupyter popup help with shift +tab or ?

```
print?
```

Or the base python `help` function

```
help(print)
```

```
Help on built-in function print in module builtins:  
  
print(...)  
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)  
  
    Prints the values to a stream, or to sys.stdout by default.  
    Optional keyword arguments:  
        file: a file-like object (stream); defaults to the current sys.stdout.  
        sep:  string inserted between values, default a space.  
        end:  string appended after the last value, default a newline.  
        flush: whether to forcibly flush the stream.
```

Notice that function can take multiple arguments and has a keyword argument (must be used like `argument=value`) described as `sep=' '`. This means that by default it adds a space

```
print(first,last)
```

```
Sarah Brown
```

But we can change the separator.

```
print(first,last, sep='-' )  
# shift + tab for help
```

```
Sarah_Brown
```

Note that it also defaults to end to use \n

```
print(first,last)  
print('hello')
```

```
Sarah Brown  
hello
```

Where does this help information come from?

```
def compute_grade(num_level1,num_level2,num_level3):  
    """  
    Computes a grade for CSC/DSP310 from numbers of achievements at each level  
  
    Parameters:  
    -----  
    num_level1 : int  
        number of level 1 achievements earned  
    num_level2 : int  
        number of level 2 achievements earned  
    num_level3 : int  
        number of level 3 achievements earned  
  
    Returns:  
    -----  
    letter_grade : string  
        letter grade with modifier (+/-)  
    ...  
    if num_level1 == 15:  
        if num_level2 == 15:  
            if num_level3 == 15:  
                grade = 'A'  
            elif num_level3 >= 10:  
                grade = 'A.'  
            elif num_level3 >=5:  
                grade = 'B+'  
            else:  
                grade = 'B'  
            elif num_level2 >=10:  
                grade = 'B-'  
            elif num_level2 >=5:  
                grade = 'C+'  
            else:  
                grade = 'C'  
            elif num_level1 >= 10:  
                grade = 'C.'  
            elif num_level1 >= 5:  
                grade = 'D+'  
            elif num_level1 >=3:  
                grade = 'D'  
            else:  
                grade = 'F'  
  
    return grade
```

ⓘ Note

You can copy code from the notes, try hovering over this

We can apply `help` on the function we wrote

```
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 (+/-)
```

It gets the docstring

3.3. Everything is an Object in Python

we can use the builtin function `type` to inspect them, and get attributes with .

```
type(compute_grade)
```

```
function
```

```
compute_grade.__name__
```

```
'compute_grade'
```

```
c = 4.5
```

```
type(c)
```

```
float
```

```
c= 'hello'
```

```
type(c)
```

```
str
```

When do we use single vs double quotes?

- You can use either, unless you need to put one inside the string then use the other.

```
my_sentence = "The professor's name is Dr. Brown"
```

```
my_sentence = 'The professor's name is Dr. Brown'
```

```
Input In [15]
my_sentence = 'The professor's name is Dr. Brown'
^
SyntaxError: invalid syntax
```

Yes we can escape special characters:

```
my_sentence = 'The professor\'s name is Dr. Brown'
```

but, it's less readable and not recommended.

3.4. Good Code is always relative

In programming for data science, we are often trying to tell a story.

💡 Try it yourself

How might this goal change your code for this class relative to other code you have written or could imagine writing?

Python is a fully [open source project](#) and as such is governed by [community standards](#) and [conventions](#).

💡 Try it yourself

Find PEP8 (note that following it is part of earning python achievements)

The [documentation](#) for the full language is online too.

Guido van Rossum was the first main developer and wrote [essays](#) about python too.

it's [pretty popular](#)

3.5. Coffee Data

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

How was this dataset collected?

- reviews added to DB
- then scraped

Where did it come from?

- coffee Quality Institute's trained reviewers.

what format is it provided in?

- csv (Comma Separated Values)

what other information is in this repository?

- the code to scrape

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

File	Raw	Blame	Copy
robusta_ratings_raw.csv	Raw	Blame	Copy
Jdbc add updated data and cleaning script	Raw	Blame	Copy
1 contributor	Raw	Blame	Copy
29 lines (29 sloc) 14.3 KB	Raw	Blame	Copy
Search this file...	Raw	Blame	Copy
1 quality_score view_certificate_1 view_certificate_2 Cupping Protocol and Descriptors View Green Analysis Details Request a Sample Species Owner Country of Origin	Raw	Blame	Copy
2 0 83.75	Raw	Blame	Copy
3 0 83.50	Raw	Blame	Copy
4 0 83.25	Raw	Blame	Copy

We'll save that url as a variable to work with it.

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

We will use a library called Pandas

```
import pandas as pd
# import library and give it an alias (nickname) pd
```

```
pd.read_csv(data_url)
```

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	...	Color	Category.Two.Defects
0	1 Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ankole coffee producers coop	1488	...	Green	2
1	2 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	3170	...	NaN	2
2	3 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	1000m	...	Green	0
3	4 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1212	...	Green	7
4	5 Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0	katuka development trust ltd	1200-1300	...	Green	3
5	6 Robusta	andrew hetzel	India	NaN	NaN	(self)	NaN	cafemakers, llc	3000'	...	Green	0
6	7 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers	750m	...	Green	0
7	8 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	7	sethuraman estate	14/1148/2017/18	kaapi royale	3140	...	Bluish-Green	0
8	9 Robusta	nishant gurjer	India	sethuraman estate	RKR	sethuraman estate	14/1148/2016/17	kaapi royale	1000	...	Green	0
9	10 Robusta	ugacof	Uganda	ishaka	NaN	nsubuga umar	0	ugacof ltd	900-1300	...	Green	6
10	11 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1095	...	Green	1
11	12 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	RC AB	sethuraman estate	14/1148/2016/12	kaapi royale	1000	...	Green	0
12	13 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers	750m	...	Green	1
13	14 Robusta	kasozi coffee farmers association	Uganda	kasozi coffee farmers	NaN	NaN	0	kasozi coffee farmers association	1367	...	Green	7
14	15 Robusta	ankole coffee producers coop	Uganda	kyangundu coop society	NaN	ankole coffee producers coop union ltd	0	ankole coffee producers coop	1488	...	Green	2
15	16 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	1000m	...	Green	0
16	17 Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc	750m	...	Blue-Green	0
17	18 Robusta	kawacom uganda ltd	Uganda	bushenyi	NaN	kawacom	0	kawacom uganda ltd	1600	...	Green	1
18	19 Robusta	nitubaasa ltd	Uganda	kigezi coffee farmers association	NaN	nitubaasa	0	nitubaasa ltd	1745	...	Green	2
19	20 Robusta	mannya coffee project	Uganda	mannya coffee project	NaN	mannya coffee project	0	mannya coffee project	1200	...	Green	1
20	21 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers	750m	...	Bluish-Green	1
21	22 Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc	750m	...	Green	0
22	23 Robusta	andrew hetzel	United States	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc	3000'	...	Green	0
23	24 Robusta	luis robles	Ecuador	robustasa	Lavado 1	our own lab	NaN	robustasa	NaN	...	Blue-Green	1
24	25 Robusta	luis robles	Ecuador	robustasa	Lavado 3	own laboratory	NaN	robustasa	40	...	Blue-Green	0
25	26 Robusta	james moore	United States	fazenda cazengo	NaN	cafe cazengo	NaN	global opportunity fund	795 meters	...	NaN	6
26	27 Robusta	cafe politico	India	NaN	NaN	NaN	14-1118-2014-0087	cafe politico	NaN	...	Green	1
27	28 Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaN	cafe politico	NaN	...	None	9

28 rows x 44 columns

💡 Try it yourself

Read the data in again, but with the index correct and save it to a variable.

```
coffee_df = pd.read_csv(data_url, index_col='Unnamed: 0')
```

Once we read it in, we can view the first 5 rows with the `head` method.

```
coffee_df.head()
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	Region	...	Color	Category.Two	Defects	E
1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ankole coffee producers coop	1488	sheema south western	...	Green	2	2	
2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	3170	chikmagalur karnataka india	...	NaN	2	2	
3	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	1000m	chikmagalur	...	Green	0	0	
4	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1212	central	...	Green	7	7	
5	Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0	katuka development trust ltd	1200-1300	luwero central region	...	Green	3	3	

5 rows × 43 columns

Important

Remember to comment & annotate your code

3.6. Follow Up questions

3.6.1. General Questions

How do you create code to scrape data from a website and compile it into a csv file?

Will we be using pandas a lot during the semester?

3.6.2. Clarifying

How do you auto finish your directories

How do you properly shut down Jupyter Notebook

Is pd some sort of variable we set or was it built in?

How should I be organized for this class? Keep it all in a single folder? Keep it on GitHub?

I'm still not sure how to keep everything together in a portfolio for the semester?

I am still wondering if I am using anaconda or just normal terminal

Can I push this code into my portfolio using the anaconda terminal

3.6.3. Grading Questions

How do we keep track of which achievements we've earned?

I don't really have many questions from today, but I was wondering if office hours were posted.

Will we always submit homework through the portfolio folder in github?

I'm just confused as how to view my feedback from the assignment

3.6.4. Questions we'll answer later this week

- does each column have a number assigned to it in data frames?
- Can other data types be imported into a notebook and edited the same way as .csv files?

3.7. More Practice

- How could you check if pd is built in or if we defined it?
- If we wanted to see more than 5 rows when printing the head of the dataset how would we do so?

Ram Token Opportunity

Contribute possible practice questions to the notes using the suggest an edit button behind the GitHub menu at the top of the page.

4. Pandas DataFrames

Today, we're going to explore [DataFrames](#) in greater detail. We'll continue using that same coffee dataset.

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

4.1. More about loading libraries

We can import pandas without the alias `pd` if we want, but then we have to use the full name everywhere

```
import pandas
```

```
pandas.read_csv(coffee_data_url)
```

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	...	Color	Category.Two.Defects
0	1 Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ankole coffee producers coop	1488	...	Green	2
1	2 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	3170	...	Nan	2
2	3 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	1000m	...	Green	0
3	4 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1212	...	Green	7
4	5 Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0	katuka development trust ltd	1200-1300	...	Green	3
5	6 Robusta	andrew hetzel	India	NaN	NaN	(self)	NaN	cafemakers, llc	3000'	...	Green	0
6	7 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers	750m	...	Green	0
7	8 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	7	sethuraman estate	14/1148/2017/18	kaapi royale	3140	...	Bluish-Green	0
8	9 Robusta	nishant gurjer	India	sethuraman estate	RKR	sethuraman estate	14/1148/2016/17	kaapi royale	1000	...	Green	0
9	10 Robusta	ugacof	Uganda	ishaka	NaN	nsubuga umar	0	ugacof ltd	900-1300	...	Green	6
10	11 Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1095	...	Green	1
11	12 Robusta	nishant gurjer	India	sethuraman estate kaapi royale	RC AB	sethuraman estate	14/1148/2016/12	kaapi royale	1000	...	Green	0
12	13 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers	750m	...	Green	1
13	14 Robusta	kasozi coffee farmers association	Uganda	kasozi coffee farmers	NaN	NaN	0	kasozi coffee farmers association	1367	...	Green	7
14	15 Robusta	ankole coffee producers coop	Uganda	kyangundu coop society	NaN	ankole coffee producers coop union ltd	0	ankole coffee producers coop	1488	...	Green	2
15	16 Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	1000m	...	Green	0
16	17 Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc	750m	...	Blue-Green	0
17	18 Robusta	kawacom uganda ltd	Uganda	bushenyi	NaN	kawacom	0	kawacom uganda ltd	1600	...	Green	1
18	19 Robusta	nitubaasa ltd	Uganda	kigezi coffee farmers association	NaN	nitubaasa	0	nitubaasa ltd	1745	...	Green	2
19	20 Robusta	mannya coffee project	Uganda	mannya coffee project	NaN	mannya coffee project	0	mannya coffee project	1200	...	Green	1
20	21 Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers	750m	...	Bluish-Green	1
21	22 Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc	750m	...	Green	0
22	23 Robusta	andrew hetzel	United States	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc	3000'	...	Green	0
23	24 Robusta	luis robles	Ecuador	robustasa	Lavado 1	our own lab	NaN	robustasa	NaN	...	Blue-Green	1
24	25 Robusta	luis robles	Ecuador	robustasa	Lavado 3	own laboratory	NaN	robustasa	40	...	Blue-Green	0
25	26 Robusta	james moore	United States	fazenda cazengo	NaN	cafe cazengo	NaN	global opportunity fund	795	...	Nan	6
26	27 Robusta	cafe politico	India	NaN	NaN	NaN	14-1118-2014-0087	cafe politico	NaN	...	Green	1
27	28 Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaN	cafe politico	NaN	...	None	9

28 rows × 44 columns

We'll use `pd` because that's the more common convention and so that we can type fewer characters throughout our code

```
import pandas as pd
```

4.2. Examining DataFrames

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

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

```
df.head()
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	Region	...	Color	Category.Two.Defects	E
1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ankole coffee producers coop	1488	sheema south western	...	Green	2	
2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	3170	chikmagalur karnataka india	...	NaN	2	:
3	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	1000m	chikmagalur	...	Green	0	
4	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1212	central	...	Green	7	
5	Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0	katuka development trust ltd	1200-1300	luwero central region	...	Green	3	

5 rows × 43 columns

Using help, we can see that that head takes one parameter and has a default value of 5, which is why we got 5 rows, but we can get 2 instead

```
df.head(2)
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	Region	...	Color	Category.Two.Defects	Expiration
1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ankole coffee producers coop	1488	sheema south western	...	Green	2	June 2
2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	3170	chikmagalur karnataka india	...	NaN	2	Oct 31st, 2

2 rows × 43 columns

We can look at the last rows with `tail`

```
df.tail(3)
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	Region	...	Color	Category.Two.Defects	Expiration	Certi
26	Robusta	james moore	United States	fazenda cazengo	NaN	cafe cazengo	NaN	global opportunity fund	795 meters	kwanza norte province, angola	...	NaN	6	December 23rd, 2015	Sf
27	Robusta	cafe politico	India	NaN	NaN	NaN	14-1118-2014-0087	cafe politico	NaN	NaN	...	Green	1	August 25th, 2015	Sf
28	Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaN	cafe politico	NaN	NaN	...	None	9	August 25th, 2015	Sf

3 rows × 43 columns

I told you this was a DataFrame, but we can check with type.

```
type(df)
```

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

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

```
df.columns
```

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

These are a special type called Index

```
type(df.columns)
```

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

It also has an index

```
df.index
```

```
Int64Index([ 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')
```

and values

```
df.values
```



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

it also knows its own shape

```
df.shape
```

```
(28, 43)
```

we can use builtin functions on our DataFrame too not just its own methods and attributes.

```
len(df)
```

```
28
```

Why does `len` turn green? it's a python reserve word

4.3. Building a Data Frame programmatically

One way to build a data frame is from a dictionary:

```
people = {'names': ['Sarah', 'Connor', 'Kenza'],
          'username': ['brownsarahm', 'sudoPsych', 'kbdlh']}
```

```
people
```

```
{'names': ['Sarah', 'Connor', 'Kenza'],
 'username': ['brownsarahm', 'sudoPsych', 'kbdlh']}
```

```
type(people)
```

```
dict
```

```
people_df = pd.DataFrame(people)
people_df
```

	names	username
0	Sarah	brownsarahm
1	Connor	sudoPsych
2	Kenza	kbdlh

```
type(people['names'])
```

```
list
```

```
type(people)
```

```
dict
```

```
type({4, 5, 5})
```

```
set
```

```
{4, 5}
```

```
people['names']
```

```
['Sarah', 'Connor', 'Kenza']
```

```
type(set(people['names']))
```

```
set
```

```
unique_people = set(people['names'])
type(unique_people)
```

```
set
```

```
[ 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')
```

```
[ for col in df.columns:
    print(col.split('.'))
```

```
['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']
```

```
[ for key,value in people.items():
    print(key,':',value)
```

```
names : ['Sarah', 'Connor', 'Kenza']
username : ['brownsarahm', 'sudoPsych', 'kdblh']
```

```
[ df['Owner']
```

```
1     ankole coffee producers coop
2             nishant gurjer
3             andrew hetzel
4             ugacof
5     katuka development trust ltd
6             andrew hetzel
7             andrew hetzel
8             nishant gurjer
9             nishant gurjer
10            ugacof
11            ugacof
12            nishant gurjer
13            andrew hetzel
14     kasozi coffee farmers association
15             ankole coffee producers coop
16             andrew hetzel
17             andrew hetzel
18             kawacom uganda ltd
19             nitubasa ltd
20             mannya coffee project
21             andrew hetzel
22             andrew hetzel
23             andrew hetzel
24             luis robles
25             luis robles
26             james moore
27             cafe politico
28             cafe politico
Name: Owner, dtype: object
```

```
[ df.Owner
```

```
1      ankole coffee producers coop
2          nishant gurjer
3          andrew hetzel
4          ugacof
5      katuka development trust ltd
6          andrew hetzel
7          andrew hetzel
8          nishant gurjer
9          nishant gurjer
10         ugacof
11         ugacof
12         nishant gurjer
13         andrew hetzel
14  kasozi coffee farmers association
15      ankole coffee producers coop
16          andrew hetzel
17          andrew hetzel
18      kawacom uganda ltd
19          nitubasa ltd
20      mannya coffee project
21          andrew hetzel
22          andrew hetzel
23          andrew hetzel
24          luis robles
25          luis robles
26          james moore
27      cafe politico
28      cafe politico
```

Name: Owner, dtype: object

{ df

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	Region	...	Color	Category.Two.Defects
1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ankole coffee producers coop	1488	sheema south western	...	Green	2
2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	3170	chikmagalur karnataka india	...	NaN	2
3	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	1000m	chikmagalur	...	Green	0
4	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1212	central	...	Green	7
5	Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0	katuka development trust ltd	1200-1300	luwero central region	...	Green	3
6	Robusta	andrew hetzel	India	NaN	NaN	(self)	NaN	cafemakers, llc	3000'	chikmagalur	...	Green	0
7	Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers	750m	chikmagalur	...	Green	0
8	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	7	sethuraman estate	14/1148/2017/18	kaapi royale	3140	chikmagalur karnataka india	...	Bluish-Green	0
9	Robusta	nishant gurjer	India	sethuraman estate	RKR	sethuraman estate	14/1148/2016/17	kaapi royale	1000	chikmagalur karnataka	...	Green	0
10	Robusta	ugacof	Uganda	ishaka	NaN	nsubuga umar	0	ugacof ltd	900-1300	western	...	Green	6
11	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1095	iganga namadropo eastern	...	Green	1
12	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	RC AB	sethuraman estate	14/1148/2016/12	kaapi royale	1000	chikmagalur karnataka	...	Green	0
13	Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers	750m	chikmagalur	...	Green	1
14	Robusta	kasozi coffee farmers association	Uganda	kasozi coffee farmers	NaN	NaN	0	kasozi coffee farmers association	1367	eastern	...	Green	7
15	Robusta	ankole coffee producers coop	Uganda	kyangundu coop society	NaN	ankole coffee producers coop union ltd	0	ankole coffee producers coop	1488	south western	...	Green	2
16	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	1000m	chikmagalur	...	Green	0
17	Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc	750m	chikmagalur	...	Blue-Green	0
18	Robusta	kawacom uganda ltd	Uganda	bushenyi	NaN	kawacom	0	kawacom uganda ltd	1600	western	...	Green	1
19	Robusta	nitubaasa ltd	Uganda	kigezi coffee farmers association	NaN	nitubaasa	0	nitubaasa ltd	1745	western	...	Green	2
20	Robusta	mannya coffee project	Uganda	mannya coffee project	NaN	mannya coffee project	0	mannya coffee project	1200	southern	...	Green	1
21	Robusta	andrew hetzel	India	sethuraman estates	NaN	NaN	NaN	cafemakers	750m	chikmagalur	...	Bluish-Green	1
22	Robusta	andrew hetzel	India	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc	750m	chikmagalur	...	Green	0
23	Robusta	andrew hetzel	United States	sethuraman estates	NaN	sethuraman estates	NaN	cafemakers, llc	3000'	chikmagalur	...	Green	0
24	Robusta	luis robles	Ecuador	robustasa	Lavado 1	our own lab	NaN	robustasa	NaN	san juan, playas	...	Blue-Green	1
25	Robusta	luis robles	Ecuador	robustasa	Lavado 3	own laboratory	NaN	robustasa	40	san juan, playas	...	Blue-Green	0
26	Robusta	james moore	United States	fazenda cazengo	NaN	cafe cazengo	NaN	global opportunity fund	795 meters	kwanza norte province, angola	...	NaN	6
27	Robusta	cafe politico	India	NaN	NaN	NaN	14-1118-2014-0087	cafe politico	NaN	NaN	...	Green	1
28	Robusta	cafe politico	Vietnam	NaN	NaN	NaN	NaN	cafe politico	NaN	NaN	...	None	9

28 rows x 43 columns

Key points:

write three things to remember from today's class

4.4. Questions After Classroom

many overlapping questions today

4.5. General

How to know which function to use in certain problems or situations



4.6. Clarifying

Is there a way to have a set show the duplicates that get discarded?

being able to access the code somewhere without asking to scroll would be nice

4.7. Course Admin

When will homeworks be posted/due typically?

4.8. Questions we'll answer later

can you use cast a pandas dataframe into a set?

4.9. Try it yourself

- Create variables of three different types with facts about yourself. Use descriptive variable names relative to the contents, not their types.

```
title = 'dr' #string
office_number = 134 # int
courses_taught = ['Programming for Data Science',
                  'Machine Learning for Science & Society']
```

- Create a list, again with a descriptive name, and print out the types

```
about_prof_brown_list = [title, office_number, courses_taught]

# regular for loop
for fact in about_prof_brown_list:
    print (type(fact))
```

```
<class 'str'>
<class 'int'>
<class 'list'>
```

- Write a function, `type_extractor` that takes a list and a type and returns the item of that type from the list
- Test your function on all three items from your dictionary.
- Use one type of jupyter help on your function, what does it display? If it doesn't display anything modify your function so that help will work.
- Make yourself notes in the most memorable way for you about what a DataFrame is.

Ram Token Opportunity

Contribute possible practice questions to the notes using the suggest an edit button behind the GitHub menu at the top of the page.

5. More Loading Data, Indexing, and Iterables

As always, we'll start with loading pandas.

```
import pandas as pd
```

5.1. Checking in on help hours

if you missed class, check over the office hours schedule and e-mail if you can or cannot attend at least one time

5.2. Portfolio Preparation and Maintainance

We'll spend a little time today getting your portfolio ready for the first check.

5.2.1. Access your portfolio

Go to your portfolio

- from the [course organization](#)
- from the list of your recent repositories on the left hand side of the [GitHub home page](#)

optionally, open it locally as well (we're going to update content and)

5.2.2. Start your Know, Want to Know, Learned Table

In each portfolio submission introduction, you'll reflect on what you've learned. To get ready for that, we'll first make note of what you already know and what you want to know.

- edit `submission_1_intro` in your portfolio locally or on GitHub:
- In the KWL section in the first two bullets after each skill with what you know and want to know. You can edit these in more detail later.

This will render as a table in your built portfolio, for reference on the syntax, [refer to the Tables section of the jupyterbook Myst Markdown cheatsheet](#).

⚠ Warning

If you work on this in the GitHub website, be sure to pull these changes locally before you start working offline next

5.2.3. Merge the setup work

Once you're done, Go to your pull request tab, and select the feedback Pull Requests. Commit any suggestions if you'd like and then merge the PR.

⚠ Warning

only do this after grading

Note

To view the feedback, after merging the PR, remove `is:open` from the search bar on the PR page

5.3. Indexing

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

What will `topics[-1]` return?

```
{ topics[-1]
```

```
'pandas'
```

Using negative indices starts from the right. The last element is -1. The first is 0.

5.4. Reading DataFrames from Websites

We'll first read from the course website.

```
{ course_comms_url =  
  'https://rhodyprog4ds.github.io/BrownFall21/syllabus/communication.html'
```

So far, we've read data in from a .csv file with `pd.read_csv` and created a DataFrame with the constructor `pd.DataFrame` using a dictionary. Pandas provides many interfaces for reading in data. They're described on the [Pandas IO page](#).

We can use the `read_html` method to read from this page. We know that it has multiple tables on the page, so let's see what it does:

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

```
[  
  Day Time Location \  
 0 Monday 9:30:00 AM-10:30 AM inperson Tyler Hall 140  
 1 Monday 12:30:00 PM-2:00 PM inperson Tyler Hall 139  
 2 Tuesday 2:00 PM-3:00 PM gather.town  
 3 Wednesday 4:00:00 PM-5:00 PM inperson Tyler Hall 139  
 4 Wednesday 1:30:00 PM-3:00 PM inperson Tyler Hall 140  
 5 Wednesday 7:00:00 PM-8:30 PM gather.town  
 6 By appointment scheduling link on Brightspace in person Tyler 134  
  
  Host  
 0 Chamudi  
 1 Chamudi  
 2 Sarah  
 3 Chamudi  
 4 Chamudi  
 5 Sarah  
 6 Sarah ,  
          usage platform \  
 0             in class prismia  
 1             any time prismia  
 2             any time prismia  
 3     private questions to your assignment github  
 4   for general questions that can help others github  
 5             to share resources github  
 6 matters that don't fit into another category e-mail  
  
          note  
 0   chat outside of class time this is not monitored cl...  
 1   message board for discussion with peers  
 2 download transcript use after class to get preliminary notes eg if...  
 3 issue on assignment repo eg bugs in your code"  
 4 issue on course website eg what the instructions of an assignment mean...  
 5 pull request on website remember to request ram tokens if applicable  
 6 to brownsarahm@uri.edu remember to include '[CSC310]' or '[DSP310]' (... ,  
          usage area \  
 0 matters that don't fit into another category to brownsarahm@uri.edu  
  
          note  
 0 remember to include '[CSC310]' or '[DSP310]' (... ,  
          usage area \  
 0   private questions to your assignment issue on assignment repo  
 1 for general questions that can help others issue on course website  
 2             to share resources pull request on website  
  
          note  
 0   eg what the instructions of an assignment mean...  
 1   remember to request ram tokens if applicable ,  
 2   usage area \  
 0 in class chat  
 1 any time message board  
 2 any time download transcript  
  
          note  
 0 outside of class time this is not monitored cl...  
 1 for discussion with peers  
 2 use after class to get preliminary notes eg if... ]
```

It appears to have read all of them, let's check the type:

```
{ type(pd.read_html(course_comms_url))
```

```
list
```

Since we know it's a list, we'll save it to a variable that indicates that.

```
{ comms_list = pd.read_html(course_comms_url)
```

If we get just the first element,

```
{ type(comms_list[0])
```

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

it's a DataFrame and prints accordingly.

Note

Using the documentation for a library (and the base language) is totally expected and normal part of programming. That's what you should use as your primary source for questions in this class. Other sources can become outdated pretty quickly as the language changes, but most of the libraries we'll use have processes in place to ensure that their own documentation gets updated at the same time the code does.

Warning

If you use other sources and get advised to solutions that are deprecated you may not earn achievements for that work.

```
{ comms_list[0]
```

	Day	Time	Location	Host
0	Monday	9:30:00 AM-10:30 AM	inperson Tyler Hall 140	Chamudi
1	Monday	12:30:00 PM-2:00 PM	inperson Tyler Hall 139	Chamudi
2	Tuesday	2:00 PM-3:00 PM	gather.town	Sarah
3	Wednesday	4:00:00 PM-5:00 PM	inperson Tyler Hall 139	Chamudi
4	Wednesday	1:30:00 PM-3:00 PM	inperson Tyler Hall 140	Chamudi
5	Wednesday	7:00:00 PM-8:30	gather.town	Sarah
6	By appointment	scheduling link on Brightspace	in person Tyler 134	Sarah

Since it's a list, we can use base python's `len` function to check how many tables there are

```
{ len(comms_list)
```

```
5
```

We've seen the first table and know it's the help hours, so we can save that to a separate variable and use it

```
{ help_df = comms_list[0]
```

We've inspected the dataframe some before, but we can also check the type of each column.

```
{ help_df.dtypes
```

```
Day      object  
Time     object  
Location  object  
Host      object  
dtype: object
```

Further Reading

You can read more about the [details of data types](#) in Pandas in the documentation

5.5. How are objects printed in jupyter?

Question from class

Q: Why does it have `dtype:object` after the type for each row? A: the last line is information about the object that is being printed out.

To understand this, let's save the thing we're curious to a variable so we can examine it multiple ways more easily.

```
{ help_df_types = help_df.dtypes
```

Next we'll check the type of this object and its shape

```
{ type(help_df_types)
```

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

Note

this section is added, it didn't happen this way in class. This section, describes **how** I figured out the answer to the question about why that extra line is displayed.

a `Series` is like a DataFrame, but just one row with headings, and then rotated.

```
{ help_df_types.shape
```

```
(4, )
```

This means that it's length is 4 and it's a 1 dimensional object; the column headers have converted to an index and are treated as metadata, but not a part of the actual data.

So, the line we're interested in is not a part of the object, because it's length 4 and the thing we're curious about is the fifth line.

We'll pick one variable from the DataFrame and check its type

```
{ type(help_df['Day'])
```

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

This is also a Series, so let's check its output

```
{ help_df['Day']
```

```
0      Monday  
1      Monday  
2      Tuesday  
3      Wednesday  
4      Wednesday  
5      Wednesday  
6  By appointment  
Name: Day, dtype: object
```

The last line of this one is information about the Series, its name, and its `dtype`.

Let's make another series, and see how it prints

```
{ pd.Series([5,4,5])
```

```
0    5  
1    4  
2    5  
dtype: int64
```

The last line is the `dtype` of the Series; so in our original object, that last line is because the list of `dtypes` is the of type `object`.

```
{ help_df_types
```

```
Day      object
Time     object
Location  object
Host      object
dtype: object
```

5.6. How do we know what to check?

we examined the DataFrame so far by (me) knowing what to look for.

In python objects you can programmatically find what to look for with the `__dict__` attribute or we can rely on the [online documentation](#) or use it via help.

In ipython (what we use in jupyter, by default) we can use the `?` for help

```
{ pd.DataFrame?
```

```
{ help(pd.DataFrame)
```

💡 Everything is Data

writing good documentation lets people who use your code get help for free Not only do help tools use the docs, but that website is generated programmatically using a tool called Sphinx from the documentation inside the code. You can access the docstring of a python using the `.__doc__` attribute

⚠️ Caution

[ipython help](#) read about how it works, if it doesn't work for you to try to figure out why

```

Help on class DataFrame in module pandas.core.frame:

class DataFrame(pandas.core.generic.NDFrame, pandas.core.arraylike.OpsMixin)
|_ DataFrame(data=None, index: 'Axes | None' = None, columns: 'Axes | None' =
None, dtype: 'Dtype | None' = None, copy: 'bool | None' = None)
    |
    | Two-dimensional, size-mutable, potentially heterogeneous tabular data.
    |
    | Data structure also contains labeled axes (rows and columns).
    | Arithmetic operations align on both row and column labels. Can be
    | thought of as a dict-like container for Series objects. The primary
    | pandas data structure.
    |
    | Parameters
    | -----
    | data : ndarray (structured or homogeneous), Iterable, dict, or DataFrame
    |     Dict can contain Series, arrays, constants, dataclass or list-like
    |     objects. If
    |     data is a dict, column order follows insertion-order. If a dict contains
    |     Series
    |         which have an index defined, it is aligned by its index.
    |     ..
    |     .. versionchanged:: 0.25.0
    |         If data is a list of dicts, column order follows insertion-order.
    |
    | index : Index or array-like
    |     Index to use for resulting frame. Will default to RangeIndex if
    |     no indexing information part of input data and no index provided.
    | columns : Index or array-like
    |     Column labels to use for resulting frame when data does not have them,
    |     defaulting to RangeIndex(0, 1, 2, ..., n). If data contains column labels,
    |     will perform column selection instead.
    | dtype : dtype, default None
    |     Data type to force. Only a single dtype is allowed. If None, infer.
    | copy : bool or None, default None
    |     Copy data from inputs.
    |     For dict data, the default of None behaves like ``copy=True``. For
    |     DataFrame
    |         or 2d ndarray input, the default of None behaves like ``copy=False``.
    |     ..
    |     .. versionchanged:: 1.3.0
    |
    | See Also
    | -----
    | DataFrame.from_records : Constructor from tuples, also record arrays.
    | DataFrame.from_dict : From dicts of Series, arrays, or dicts.
    | read_csv : Read a comma-separated values (csv) file into DataFrame.
    | read_table : Read general delimited file into DataFrame.
    | read_clipboard : Read text from clipboard into DataFrame.
    |
    | Examples
    | -----
    | Constructing DataFrame from a dictionary.
    |
    |>>> d = {'col1': [1, 2], 'col2': [3, 4]}
    |>>> df = pd.DataFrame(data=d)
    |>>> df
    |   col1  col2
    |0      1      3
    |1      2      4
    |
    |Notice that the inferred dtype is int64.
    |
    |>>> df.dtypes
    |col1    int64
    |col2    int64
    |dtype: object
    |
    |To enforce a single dtype:
    |
    |>>> df = pd.DataFrame(data=d, dtype=np.int8)
    |>>> df.dtypes
    |col1    int8
    |col2    int8
    |dtype: object
    |
    |Constructing DataFrame from a dictionary including Series:
    |
    |>>> d = {'col1': [0, 1, 2, 3], 'col2': pd.Series([2, 3], index=[2, 3])}
    |>>> pd.DataFrame(data=d, index=[0, 1, 2, 3])
    |   col1  col2
    |0      0    NaN
    |1      1    NaN
    |2      2    2.0
    |3      3    3.0
    |
    |Constructing DataFrame from numpy ndarray:
    |
    |>>> df2 = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
    |...                 columns=['a', 'b', 'c'])
    |>>> df2
    |   a  b  c
    |0  1  2  3
    |1  4  5  6
    |2  7  8  9
    |
    |Constructing DataFrame from a numpy ndarray that has labeled columns:
    |
    |>>> data = np.array([(1, 2, 3), (4, 5, 6), (7, 8, 9)],
    |...                 dtype=[("a", "i4"), ("b", "i4"), ("c", "i4")])
    |>>> df3 = pd.DataFrame(data, columns=['c', 'a'])
    |...

```

```

>>> df3
   c  a
0  3  1
1  6  4
2  9  7

Constructing DataFrame from dataclass:

>>> from dataclasses import make_dataclass
>>> Point = make_dataclass("Point", [("x", int), ("y", int)])
>>> pd.DataFrame([Point(0, 0), Point(0, 3), Point(2, 3)])
   x  y
0  0  0
1  0  3
2  2  3

Method resolution order:
    DataFrame
    pandas.core.generic.NDFrame
    pandas.core.base.PandasObject
    pandas.core.accessor.DirNamesMixin
    pandas.core.indexing.IndexingMixin
    pandas.core.arraylike.OpsMixin
    builtins.object

Methods defined here:
    __divmod__(self, other) -> 'tuple[DataFrame, DataFrame]'

    __getitem__(self, key)

    __init__(self, data=None, index: 'Axes | None' = None, columns: 'Axes | None' = None, dtype: 'Dtype | None' = None, copy: 'bool | None' = None)
        Initialize self. See help(type(self)) for accurate signature.

    __len__(self) -> 'int'
        Returns length of info axis, but here we use the index.

    __matmul__(self, other: 'AnyArrayLike | DataFrame | Series') -> 'DataFrame | Series'
        Matrix multiplication using binary '@' operator in Python>=3.5.

    __rdivmod__(self, other) -> 'tuple[DataFrame, DataFrame]'

    __repr__(self) -> 'str'
        Return a string representation for a particular DataFrame.

    __rmatmul__(self, other)
        Matrix multiplication using binary '@' operator in Python>=3.5.

    __setitem__(self, key, value)
        add(self, other, axis='columns', level=None, fill_value=None)
            Get Addition of dataframe and other, element-wise (binary operator `add`).

            Equivalent to ``dataframe + other```, but with support to substitute a
            fill_value
            for missing data in one of the inputs. With reverse version, `radd`.

            Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
            arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.

    Parameters
    -----
    other : scalar, sequence, Series, or DataFrame
        Any single or multiple element data structure, or list-like object.
    axis : {0 or 'index', 1 or 'columns'}
        Whether to compare by the index (0 or 'index') or columns
        (1 or 'columns'). For Series input, axis to match Series index on.
    level : int or label
        Broadcast across a level, matching Index values on the
        passed MultiIndex level.
    fill_value : float or None, default None
        Fill existing missing (NaN) values, and any new element needed for
        successful DataFrame alignment, with this value before computation.
        If data in both corresponding DataFrame locations is missing
        the result will be missing.

    Returns
    -----
    DataFrame
        Result of the arithmetic operation.

    See Also
    -----
    DataFrame.add : Add DataFrames.
    DataFrame.sub : Subtract DataFrames.
    DataFrame.mul : Multiply DataFrames.
    DataFrame.div : Divide DataFrames (float division).
    DataFrame.truediv : Divide DataFrames (float division).
    DataFrame.floordiv : Divide DataFrames (integer division).
    DataFrame.mod : Calculate modulo (remainder after division).
    DataFrame.pow : Calculate exponential power.

    Notes
    -----
    Mismatched indices will be unioned together.

    Examples
    -----
    >>> df = pd.DataFrame({'angles': [0, 3, 4],
    ...                      'degrees': [360, 180, 360]},
    ...                     index=['circle', 'triangle', 'rectangle'])
    >>> df
      angles  degrees
circle      0      360
triangle    3      180
rectangle   4      360

    Add a scalar with operator version which return the same
    results.

    >>> df + 1
      angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

    >>> df.add(1)
      angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

    Divide by constant with reverse version.

    >>> df.div(10)
      angles  degrees
circle      0.0     36.0
triangle    0.3     18.0
rectangle   0.4     36.0

```

```

>>> df.rdiv(10)
      angles   degrees
circle      inf  0.027778
triangle  3.333333  0.055556
rectangle  2.500000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
      angles   degrees
circle      -1     358
triangle     2     178
rectangle    3     358

>>> df.sub([1, 2], axis='columns')
      angles   degrees
circle      -1     358
triangle     2     178
rectangle    3     358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
      ...      axis='index')
      angles   degrees
circle      -1     359
triangle     2     179
rectangle    3     359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4]}, ...
...                           index=['circle', 'triangle', 'rectangle'])
>>> other
      angles
circle      0
triangle    3
rectangle   4

>>> df * other
      angles   degrees
circle      0      NaN
triangle    9      NaN
rectangle  16      NaN

>>> df.mul(other, fill_value=0)
      angles   degrees
circle      0      0.0
triangle    9      0.0
rectangle  16      0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
...                               index=[['A', 'A', 'A', 'B', 'B'],
...                                     ['circle', 'triangle', 'rectangle',
...                                      'square', 'pentagon', 'hexagon']])
>>> df_multindex
      angles   degrees
A circle      0     360
triangle     3     180
rectangle    4     360
B square      4     360
pentagon     5     540
hexagon      6     720

>>> df.div(df_multindex, level=1, fill_value=0)
      angles   degrees
A circle    NaN     1.0
triangle   1.0     1.0
rectangle  1.0     1.0
B square    0.0     0.0
pentagon   0.0     0.0
hexagon    0.0     0.0

agg = aggregate(self, func=None, axis: 'Axis' = 0, *args, **kwargs)
aggregate(self, func=None, axis: 'Axis' = 0, *args, **kwargs)
Aggregate using one or more operations over the specified axis.

Parameters
-----
func : function, str, list or dict
    Function to use for aggregating the data. If a function, must either
    work when passed a DataFrame or when passed to DataFrame.apply.

    Accepted combinations are:
        - function
        - string function name
        - list of functions and/or function names, e.g. ``[np.sum, 'mean']``
        - dict of axis labels -> functions, function names or list of such.
axis : {0 or 'index', 1 or 'columns'}, default 0
    If 0 or 'index': apply function to each column.
    If 1 or 'columns': apply function to each row.

*args
    Positional arguments to pass to `func`.

**kwargs
    Keyword arguments to pass to `func`.

Returns
-----
scalar, Series or DataFrame

The return can be:
* scalar : when Series.agg is called with single function
* Series : when DataFrame.agg is called with a single function
* Dataframe : when DataFrame.agg is called with several functions

Return scalar, Series or DataFrame.

The aggregation operations are always performed over an axis, either the
index (default) or the column axis. This behavior is different from
`numpy` aggregation functions ('mean', 'median', 'prod', 'sum', 'std',
'var'), where the default is to compute the aggregation of the flattened
array, e.g., ``numpy.mean(arr_2d)`` as opposed to
``numpy.mean(arr_2d, axis=0)``.

`agg` is an alias for `aggregate`. Use the alias.

See Also
-----
DataFrame.apply : Perform any type of operations.
DataFrame.transform : Perform transformation type operations.
core.groupby.GroupBy : Perform operations over groups.
core.resample.Resampler : Perform operations over resampled bins.
core.window.Rolling : Perform operations over rolling window.
core.window.Expanding : Perform operations over expanding window.

```

```

|     core.window.ExponentialMovingWindow : Perform operation over exponential
| weighted
|         window.
|
| Notes
| -----
| 'agg' is an alias for `aggregate`. Use the alias.
|
| Functions that mutate the passed object can produce unexpected
| behavior or errors and are not supported. See :ref:`gotchas.udf-mutation`
| for more details.
|
| A passed user-defined-function will be passed a Series for evaluation.
|
| Examples
| -----
|>>> df = pd.DataFrame([[1, 2, 3],
|...                      [4, 5, 6],
|...                      [7, 8, 9],
|...                      [np.nan, np.nan, np.nan]],
|...                      columns=['A', 'B', 'C'])
|
| Aggregate these functions over the rows.
|
|>>> df.agg(['sum', 'min'])
|      A   B   C
| sum  12.0 15.0 18.0
| min  1.0  2.0  3.0
|
| Different aggregations per column.
|
|>>> df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']})
|      A   B
| sum  12.0  NaN
| min  1.0  2.0
| max  NaN  8.0
|
| Aggregate different functions over the columns and rename the index of the
| resulting
| DataFrame.
|
|>>> df.agg(x=('A', 'max'), y=('B', 'min'), z=('C', np.mean))
|      A   B
| x  7.0  NaN  NaN
| y  NaN  2.0  NaN
| z  NaN  NaN  6.0
|
| Aggregate over the columns.
|
|>>> df.agg("mean", axis="columns")
| 0    2.0
| 1    5.0
| 2    8.0
| 3    NaN
| dtype: float64
|
| align(self, other, join: 'str' = 'outer', axis: 'Axis | None' = None, level:
| 'Level | None' = None, copy: 'bool' = True, fill_value=None, method: 'str | None'
| = None, limit=None, fill_axis: 'Axis' = 0, broadcast_axis: 'Axis | None' = None) ->
| > DataFrame
|     Align two objects on their axes with the specified join method.
|
| Join method is specified for each axis Index.
|
| Parameters
| -----
| other : DataFrame or Series
| join : {'outer', 'inner', 'left', 'right'}, default 'outer'
| axis : allowed axis of the other object, default None
|     Align on index (0), columns (1), or both (None).
| level : int or level name, default None
|     Broadcast across a level, matching Index values on the
|     passed MultiIndex level.
| copy : bool, default True
|     Always returns new objects. If copy=False and no reindexing is
|     required then original objects are returned.
| fill_value : scalar, default np.NAN
|     Value to use for missing values. Defaults to NaN, but can be any
|     "compatible" value.
| method : {'backfill', 'bfill', 'pad', 'ffill', None}, default None
|     Method to use for filling holes in reindexed Series:
|
|     - pad / ffill: propagate last valid observation forward to next valid.
|     - backfill / bfill: use NEXT valid observation to fill gap.
|
| limit : int, default None
|     If method is specified, this is the maximum number of consecutive
|     NAN values to forward/backward fill. In other words, if there is
|     a gap with more than this number of consecutive NaNs, it will only
|     be partially filled. If method is not specified, this is the
|     maximum number of entries along the entire axis where NaNs will be
|     filled. Must be greater than 0 if not None.
| fill_axis : {0 or 'index', 1 or 'columns'}, default 0
|     Filling axis, method and limit.
| broadcast_axis : {0 or 'index', 1 or 'columns'}, default None
|     Broadcast values along this axis, if aligning two objects of
|     different dimensions.
|
| Returns
| -----
| (left, right) : (DataFrame, type of other)
|     Aligned objects.
|
| Examples
| -----
|>>> df = pd.DataFrame(
|...      [[1, 2, 3, 4], [6, 7, 8, 9]], columns=["D", "B", "E", "A"], index=
|[1, 2]
|... )
|>>> other = pd.DataFrame(
|...      [[10, 20, 30, 40], [60, 70, 80, 90], [600, 700, 800, 900]],
|...      columns=["A", "B", "C", "D"],
|...      index=[2, 3, 4],
|... )
|>>> df
|      D   B   E   A
| 1  1   2   3   4
| 2  6   7   8   9
|>>> other
|      A   B   C   D
| 2  10  20  30  40
| 3  60  70  80  90
| 4  600 700 800 900
|
| Align on columns:
|
|>>> left, right = df.align(other, join="outer", axis=1)
|>>> left
|      A   B   C   D   E
| 1  4   2  NaN  1   3
| 2  9   7  NaN  6   8

```

```

>>> right
      A   B   C   D   E
2  10   20   30   40  NaN
3  60   70   80   90  NaN
4 600  700  800  900  NaN

We can also align on the index:

>>> left, right = df.align(other, join="outer", axis=0)
>>> left
      D   B   E   A
1  1.0  2.0  3.0  4.0
2  6.0  7.0  8.0  9.0
3  NaN  NaN  NaN  NaN
4  NaN  NaN  NaN  NaN
>>> right
      A   B   C   D
1  NaN  NaN  NaN  NaN
2  10.0 20.0 30.0 40.0
3  60.0 70.0 80.0 90.0
4 600.0 700.0 800.0 900.0

Finally, the default 'axis=None' will align on both index and columns:

>>> left, right = df.align(other, join="outer", axis=None)
>>> left
      A   B   C   D   E
1  4.0  2.0  NaN  1.0  3.0
2  9.0  7.0  NaN  6.0  8.0
3  NaN  NaN  NaN  NaN  NaN
4  NaN  NaN  NaN  NaN  NaN
>>> right
      A   B   C   D   E
1  NaN  NaN  NaN  NaN  NaN
2  10.0 20.0 30.0 40.0  NaN
3  60.0 70.0 80.0 90.0  NaN
4 600.0 700.0 800.0 900.0  NaN

all(self, axis=0, bool_only=None, skipna=True, level=None, **kwargs)
    Return whether all elements are True, potentially over an axis.

Returns True unless there at least one element within a series or
along a Dataframe axis that is False or equivalent (e.g. zero or
empty).

Parameters
-----
axis : {0 or 'index', 1 or 'columns', None}, default 0
    Indicate which axis or axes should be reduced.

    * 0 / 'index' : reduce the index, return a Series whose index is the
        original column labels.
    * 1 / 'columns' : reduce the columns, return a Series whose index is
        the
        original index.
    * None : reduce all axes, return a scalar.

bool_only : bool, default None
    Include only boolean columns. If None, will attempt to use everything,
    then use only boolean data. Not implemented for Series.

skipna : bool, default True
    Exclude NA/null values. If the entire row/column is NA and skipna is
    True, then the result will be True, as for an empty row/column.
    If skipna is False, then NA are treated as True, because these are not
    equal to zero.

level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.

**kwargs : any, default None
    Additional keywords have no effect but might be accepted for
    compatibility with NumPy.

Returns
-----
Series or DataFrame
    If level is specified, then, DataFrame is returned; otherwise, Series
    is returned.

See Also
-----
Series.all : Return True if all elements are True.
DataFrame.any : Return True if one (or more) elements are True.

Examples
-----
**Series**

>>> pd.Series([True, True]).all()
True
>>> pd.Series([True, False]).all()
False
>>> pd.Series([], dtype="float64").all()
True
>>> pd.Series([np.nan]).all()
True
>>> pd.Series([np.nan]).all(skipna=False)
True

**DataFrames**

Create a dataframe from a dictionary.

>>> df = pd.DataFrame({'col1': [True, True], 'col2': [True, False]})
>>> df
   col1   col2
0  True   True
1  True  False

Default behaviour checks if column-wise values all return True.

>>> df.all()
   col1   True
   col2  False
dtype: bool

Specify ``axis='columns'`` to check if row-wise values all return True.

>>> df.all(axis='columns')
0   True
1   False
dtype: bool

Or ``axis=None`` for whether every value is True.

>>> df.all(axis=None)
False

any(self, axis=0, bool_only=None, skipna=True, level=None, **kwargs)
    Return whether any element is True, potentially over an axis.

Returns False unless there is at least one element within a series or

```

along a Dataframe axis that is True or equivalent (e.g. non-zero or non-empty).

Parameters

axis : {0 or 'index', 1 or 'columns', None}, default 0
 Indicate which axis or axes should be reduced.

- * 0 / 'index' : reduce the index, return a Series whose index is the original column labels.
- * 1 / 'columns' : reduce the columns, return a Series whose index is the original index.
- * None : reduce all axes, return a scalar.

bool_only : bool, default None
 Include only boolean columns. If None, will attempt to use everything, then use only boolean data. Not implemented for Series.

skipna : bool, default True
 Exclude NA/null values. If the entire row/column is NA and skipna is True, then the result will be False, as for an empty row/column.
 If skipna is False, then NA are treated as True, because these are not equal to zero.

level : int or level name, default None
 If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series.

**kwargs : any, default None
 Additional keywords have no effect but might be accepted for compatibility with NumPy.

Returns

Series or DataFrame
 If level is specified, then DataFrame is returned; otherwise, Series is returned.

See Also

numpy.any : Numpy version of this method.
 Series.any : Return whether any element is True.
 Series.all : Return whether all elements are True.
 DataFrame.any : Return whether any element is True over requested axis.
 DataFrame.all : Return whether all elements are True over requested axis.

Examples

Series

For Series input, the output is a scalar indicating whether any element is True.

```
>>> pd.Series([False, False]).any()
False
>>> pd.Series([True, False]).any()
True
>>> pd.Series([], dtype="float64").any()
False
>>> pd.Series([np.nan]).any()
False
>>> pd.Series([np.nan]).any(skipna=False)
True
```

DataFrame

Whether each column contains at least one True element (the default).

```
>>> df = pd.DataFrame({"A": [1, 2], "B": [0, 2], "C": [0, 0]})
>>> df
   A  B  C
0  1  0  0
1  2  2  0

>>> df.any()
A    True
B    True
C   False
dtype: bool
```

Aggregating over the columns.

```
>>> df = pd.DataFrame({"A": [True, False], "B": [1, 2]})
>>> df
   A  B
0  True  1
1 False  2

>>> df.any(axis='columns')
0  True
1  True
dtype: bool

>>> df = pd.DataFrame({"A": [True, False], "B": [1, 0]})
>>> df
   A  B
0  True  1
1 False  0

>>> df.any(axis='columns')
0  True
1 False
dtype: bool
```

Aggregating over the entire DataFrame with ``axis=None``.

```
>>> df.any(axis=None)
True
```

'any' for an empty DataFrame is an empty Series.

```
>>> pd.DataFrame([]).any()
Series([], dtype: bool)
```

append(self, other, ignore_index: 'bool' = False, verify_integrity: 'bool' = False, sort: 'bool' = False) -> 'DataFrame'
 Append rows of 'other' to the end of caller, returning a new object.

.. deprecated:: 1.4.0
 Use :func:`concat` instead. For further details see :ref:`whatsnew_140.deprecations.frame_series_append`

Columns in 'other' that are not in the caller are added as new columns.

Parameters

other : DataFrame or Series/dict-like object, or list of these
 The data to append.

ignore_index : bool, default False
 If True, the resulting axis will be labeled 0, 1, ..., n - 1.

verify_integrity : bool, default False
 If True, raise ValueError on creating index with duplicates.

sort : bool, default False

```

Sort columns if the columns of `self` and `other` are not aligned.
.. versionchanged:: 1.0.0
    Changed to not sort by default.

Returns
-----
DataFrame
    A new DataFrame consisting of the rows of caller and the rows of
`other`.

See Also
-----
concat : General function to concatenate DataFrame or Series objects.

Notes
-
If a list of dict/series is passed and the keys are all contained in
the DataFrame's index, the order of the columns in the resulting
DataFrame will be unchanged.

Iteratively appending rows to a DataFrame can be more computationally
intensive than a single concatenate. A better solution is to append
those rows to a list and then concatenate the list with the original
DataFrame all at once.

Examples
-----
>>> df = pd.DataFrame([[1, 2], [3, 4]], columns=list('AB'), index=['x',
'y'])
>>> df
   A  B
x  1  2
y  3  4
>>> df2 = pd.DataFrame([[5, 6], [7, 8]], columns=list('AB'), index=['x',
'y'])
>>> df.append(df2)
   A  B
x  1  2
y  3  4
x  5  6
y  7  8

With `ignore_index` set to True:

>>> df.append(df2, ignore_index=True)
   A  B
0  1  2
1  3  4
2  5  6
3  7  8

The following, while not recommended methods for generating DataFrames,
show two ways to generate a DataFrame from multiple data sources.

Less efficient:

>>> df = pd.DataFrame(columns=['A'])
>>> for i in range(5):
...     df = df.append({'A': i}, ignore_index=True)
>>> df
   A
0  0
1  1
2  2
3  3
4  4

More efficient:

>>> pd.concat([pd.DataFrame([i], columns=['A']) for i in range(5)],
...             ignore_index=True)
   A
0  0
1  1
2  2
3  3
4  4

apply(self, func: 'AggFuncType', axis: 'Axis' = 0, raw: 'bool' = False,
result_type=None, args=(), **kwargs)
Apply a function along an axis of the DataFrame.

Objects passed to the function are Series objects whose index is
either the DataFrame's index (`axis=0`) or the DataFrame's columns
(`axis=1`). By default (`result_type=None`), the final return type
is inferred from the return type of the applied function. Otherwise,
it depends on the `result_type` argument.

Parameters
-----
func : function
    Function to apply to each column or row.
axis : {0 or 'index', 1 or 'columns'}, default 0
    Axis along which the function is applied:
        * 0 or 'index': apply function to each column.
        * 1 or 'columns': apply function to each row.

raw : bool, default False
    Determines if row or column is passed as a Series or ndarray object:
        * ``False`` : passes each row or column as a Series to the
            function.
        * ``True`` : the passed function will receive ndarray objects
            instead.
        If you are just applying a NumPy reduction function this will
            achieve much better performance.

result_type : {'expand', 'reduce', 'broadcast', None}, default None
    These only act when ``axis=1`` (columns):
        * 'expand' : list-like results will be turned into columns.
        * 'reduce' : returns a Series if possible rather than expanding
            list-like results. This is the opposite of 'expand'.
        * 'broadcast' : results will be broadcast to the original shape
            of the DataFrame, the original index and columns will be
            retained.

The default behaviour (None) depends on the return value of the
applied function: list-like results will be returned as a Series
of those. However if the apply function returns a Series these
are expanded to columns.

args : tuple
    Positional arguments to pass to `func` in addition to the
    array/series.

**kwargs
    Additional keyword arguments to pass as keywords arguments to
    `func`.

```

```

    Returns
    -----
    Series or DataFrame
        Result of applying ``func`` along the given axis of the
        DataFrame.

    See Also
    -----
    DataFrame.applymap: For elementwise operations.
    DataFrame.aggregate: Only perform aggregating type operations.
    DataFrame.transform: Only perform transforming type operations.

    Notes
    -----
    Functions that mutate the passed object can produce unexpected
    behavior or errors and are not supported. See :ref:`gotchas.udf-mutation`
    for more details.

    Examples
    -----
    >>> df = pd.DataFrame([[4, 9]] * 3, columns=['A', 'B'])
    >>> df
       A   B
    0   4   9
    1   4   9
    2   4   9

    Using a numpy universal function (in this case the same as
    ``np.sqrt(df)``):

    >>> df.apply(np.sqrt)
       A   B
    0  2.0  3.0
    1  2.0  3.0
    2  2.0  3.0

    Using a reducing function on either axis

    >>> df.apply(np.sum, axis=0)
    A   12
    B   27
    dtype: int64

    >>> df.apply(np.sum, axis=1)
    0   13
    1   13
    2   13
    dtype: int64

    Returning a list-like will result in a Series

    >>> df.apply(lambda x: [1, 2], axis=1)
    0   [1, 2]
    1   [1, 2]
    2   [1, 2]
    dtype: object

    Passing ``result_type='expand'`` will expand list-like results
    to columns of a Dataframe

    >>> df.apply(lambda x: [1, 2], axis=1, result_type='expand')
    0   1
    0   1
    1   1
    1   2
    2   1
    2   2

    Returning a Series inside the function is similar to passing
    ``result_type='expand'``. The resulting column names
    will be the Series index.

    >>> df.apply(lambda x: pd.Series([1, 2], index=['foo', 'bar']), axis=1)
    foo  bar
    0   1   2
    1   1   2
    2   1   2

    Passing ``result_type='broadcast'`` will ensure the same shape
    result, whether list-like or scalar is returned by the function,
    and broadcast it along the axis. The resulting column names will
    be the originals.

    >>> df.apply(lambda x: [1, 2], axis=1, result_type='broadcast')
    A   B
    0   1   2
    1   1   2
    2   1   2

    applymap(self, func: 'PythonFuncType', na_action: 'str | None' = None,
    **kwargs) -> 'DataFrame'
        Apply a function to a Dataframe elementwise.

        This method applies a function that accepts and returns a scalar
        to every element of a DataFrame.

    Parameters
    -----
    func : callable
        Python function, returns a single value from a single value.
    na_action : {None, 'ignore'}, default None
        If 'ignore', propagate NaN values, without passing them to func.

        .. versionadded:: 1.2

    **kwargs
        Additional keyword arguments to pass as keywords arguments to
        'func'.

        .. versionadded:: 1.3.0

    Returns
    -----
    DataFrame
        Transformed DataFrame.

    See Also
    -----
    DataFrame.apply : Apply a function along input axis of DataFrame.

    Examples
    -----
    >>> df = pd.DataFrame([[1, 2.12], [3.356, 4.567]])
    >>> df
       0      1
    0  1.000  2.120
    1  3.356  4.567

    >>> df.applymap(lambda x: len(str(x)))
       0      1
    0  3      4
    1  5      5

```

```

Like Series.map, NA values can be ignored:

>>> df_copy = df.copy()
>>> df_copy.iloc[0, 0] = pd.NA
>>> df_copy.applymap(lambda x: len(str(x)), na_action='ignore')
   0   1
0 <NA>  4
1      5  5

Note that a vectorized version of `func` often exists, which will
be much faster. You could square each number elementwise.

>>> df.applymap(lambda x: x**2)
   0   1
0  1.000000  4.494400
1 11.262736 20.857489

But it's better to avoid applymap in that case.

>>> df ** 2
   0   1
0  1.000000  4.494400
1 11.262736 20.857489

| asfreq(self, freq: 'Frequency', method=None, how: 'str | None' = None,
| normalize: 'bool' = False, fill_value=None) -> 'DataFrame'
|     Convert time series to specified frequency.

| Returns the original data conformed to a new index with the specified
| frequency.

| If the index of this DataFrame is a :class:`pandas.PeriodIndex`, the new
| index
| is the result of transforming the original index with
| :meth:`PeriodIndex.asfreq <pandas.PeriodIndex.asfreq>` (so the original
| index
| will map one-to-one to the new index).

| Otherwise, the new index will be equivalent to ``pd.date_range(start, end,
| freq=freq)`` where ``start`` and ``end`` are, respectively, the first and
| last entries in the original index (see :func:`pandas.date_range`). The
| values corresponding to any timesteps in the new index which were not
| present
| in the original index will be null (`NaN`), unless a method for filling
| such unknowns is provided (see the ``method`` parameter below).

| The :meth:`resample` method is more appropriate if an operation on each
| group of
| timesteps (such as an aggregate) is necessary to represent the data at the
| new
| frequency.

Parameters
-----
freq : DateOffset or str
    Frequency DateOffset or string.
method : {'backfill'/'bfill', 'pad'/'ffill'}, default None
    Method to use for filling holes in reindexed Series (note this
    does not fill NaNs that already were present):
        * 'pad' / 'ffill': propagate last valid observation forward to next
        valid
        * 'backfill' / 'bfill': use NEXT valid observation to fill.
how : {'start', 'end'}, default end
    For PeriodIndex only (see PeriodIndex.asfreq).
normalize : bool, default False
    Whether to reset output index to midnight.
fill_value : scalar, optional
    Value to use for missing values, applied during upsampling (note
    this does not fill NaNs that already were present).

Returns
-----
DataFrame
    DataFrame object reindexed to the specified frequency.

See Also
-----
reindex : Conform DataFrame to new index with optional filling logic.

Notes
-----
To learn more about the frequency strings, please see `this link
<https://pandas.pydata.org/pandas-docs/stable/user\_guide/timeseries.html#offset-aliases>`__.

Examples
-----
Start by creating a series with 4 one minute timestamps.

>>> index = pd.date_range('1/1/2000', periods=4, freq='T')
>>> series = pd.Series([0.0, None, 2.0, 3.0], index=index)
>>> df = pd.DataFrame({'s': series})
>>> df
   s
2000-01-01 00:00:00  0.0
2000-01-01 00:01:00  NaN
2000-01-01 00:02:00  2.0
2000-01-01 00:03:00  3.0

Upsample the series into 30 second bins.

>>> df.asfreq(freq='30S')
   s
2000-01-01 00:00:00  0.0
2000-01-01 00:00:30  NaN
2000-01-01 00:01:00  NaN
2000-01-01 00:01:30  NaN
2000-01-01 00:02:00  2.0
2000-01-01 00:02:30  NaN
2000-01-01 00:03:00  3.0

Upsample again, providing a ``fill value``.

>>> df.asfreq(freq='30S', fill_value=9.0)
   s
2000-01-01 00:00:00  0.0
2000-01-01 00:00:30  9.0
2000-01-01 00:01:00  NaN
2000-01-01 00:01:30  9.0
2000-01-01 00:02:00  2.0
2000-01-01 00:02:30  9.0
2000-01-01 00:03:00  3.0

Upsample again, providing a ``method``.

>>> df.asfreq(freq='30S', method='bfill')
   s
2000-01-01 00:00:00  0.0
2000-01-01 00:00:30  NaN
2000-01-01 00:01:00  NaN

```

```

2000-01-01 00:01:30    2.0
2000-01-01 00:02:00    2.0
2000-01-01 00:02:30    3.0
2000-01-01 00:03:00    3.0

assign(self, **kwargs) -> 'DataFrame'
    Assign new columns to a DataFrame.

    Returns a new object with all original columns in addition to new ones.
    Existing columns that are re-assigned will be overwritten.

Parameters
-----
**kwargs : dict of {str: callable or Series}
    The column names are keywords. If the values are
    callable, they are computed on the DataFrame and
    assigned to the new columns. The callable must not
    change input DataFrame (though pandas doesn't check it).
    If the values are not callable, (e.g. a Series, scalar, or array),
    they are simply assigned.

Returns
-----
DataFrame
    A new DataFrame with the new columns in addition to
    all the existing columns.

Notes
-----
Assigning multiple columns within the same ``assign`` is possible.
Later items in ``**kwargs`` may refer to newly created or modified
columns in 'df'; items are computed and assigned into 'df' in order.

Examples
-----
>>> df = pd.DataFrame({'temp_c': [17.0, 25.0],
...                      index=['Portland', 'Berkeley'])
>>> df
   temp_c
Portland  17.0
Berkeley  25.0

Where the value is a callable, evaluated on `df`:

>>> df.assign(temp_f=lambda x: x.temp_c * 9 / 5 + 32)
   temp_c  temp_f
Portland  17.0  62.6
Berkeley  25.0  77.0

Alternatively, the same behavior can be achieved by directly
referencing an existing Series or sequence:

>>> df.assign(temp_f=df['temp_c'] * 9 / 5 + 32)
   temp_c  temp_f
Portland  17.0  62.6
Berkeley  25.0  77.0

You can create multiple columns within the same assign where one
of the columns depends on another one defined within the same assign:

>>> df.assign(temp_f=lambda x: x['temp_c'] * 9 / 5 + 32,
...             temp_k=lambda x: (x['temp_f'] + 459.67) * 5 / 9)
   temp_c  temp_f  temp_k
Portland  17.0  62.6  290.15
Berkeley  25.0  77.0  298.15

bfill(self: 'DataFrame', axis: 'None | Axis' = None, inplace: 'bool' = False,
limit: 'None | int' = None, downcast=None) -> 'DataFrame | None'
    Synonym for :meth:`DataFrame.fillna` with ``method='bfill'``.

Returns
-----
Series/DataFrame or None
    Object with missing values filled or None if ``inplace=True``.

boxplot = boxplot_frame(self, column=None, by=None, ax=None, fontsize=None,
rot=0, grid=True, figsize=None, layout=None, return_type=None, backend=None,
**kwargs)
    Make a box plot from DataFrame columns.

    Make a box-and-whisker plot from DataFrame columns, optionally grouped
    by some other columns. A box plot is a method for graphically depicting
    groups of numerical data through their quartiles.
    The box extends from the Q1 to Q3 quartile values of the data,
    with a line at the median (Q2). The whiskers extend from the edges
    of box to show the range of the data. By default, they extend no more than
    `1.5 * IQR (IQR = Q3 - Q1)` from the edges of the box, ending at the
    farthest
    data point within that interval. Outliers are plotted as separate dots.

    For further details see
    Wikipedia's entry for 'boxplot <https://en.wikipedia.org/wiki/Box\_plot>_.

Parameters
-----
column : str or list of str, optional
    Column name or list of names, or vector.
    Can be any valid input to :meth:`pandas.DataFrame.groupby`.
by : str or array-like, optional
    Column in the DataFrame to :meth:`pandas.DataFrame.groupby`.
    One box-plot will be done per value of columns in 'by'.
ax : object of class matplotlib.axes.Axes, optional
    The matplotlib axes to be used by boxplot.
fontsize : float or str
    Tick label font size in points or as a string (e.g., 'large').
rot : int or float, default 0
    The rotation angle of labels (in degrees)
    with respect to the screen coordinate system.
grid : bool, default True
    Setting this to True will show the grid.
figsize : A tuple (width, height) in inches
    The size of the figure to create in matplotlib.
layout : tuple (rows, columns), optional
    For example, (3, 5) will display the subplots
    using 3 columns and 5 rows, starting from the top-left.
return_type : {'axes', 'dict', 'both'} or None, default 'axes'
    The kind of object to return. The default is 'axes'.
        * 'axes' returns the matplotlib axes the boxplot is drawn on.
        * 'dict' returns a dictionary whose values are the matplotlib
        Lines of the boxplot.
        * 'both' returns a namedtuple with the axes and dict.
        * when grouping with 'by', a Series mapping columns to
        'return_type' is returned.

    If ``return_type`` is 'None', a NumPy array
    of axes with the same shape as ``layout`` is returned.

backend : str, default None
    Backend to use instead of the backend specified in the option
    ``plotting.backend``. For instance, 'matplotlib'. Alternatively, to
    specify the ``plotting.backend`` for the whole session, set

```

```

``pd.options.plotting.backend``.
.. versionadded:: 1.0.0

**kwargs
    All other plotting keyword arguments to be passed to
    :func:`matplotlib.pyplot.boxplot`.

Returns
-----
result
    See Notes.

See Also
-----
Series.plot.hist: Make a histogram.
matplotlib.pyplot.boxplot : Matplotlib equivalent plot.

Notes
-----
The return type depends on the `return_type` parameter:

* 'axes' : object of class matplotlib.axes.Axes
* 'dict' : dict of matplotlib.lines.Line2D objects
* 'both' : a namedtuple with structure (ax, lines)

For data grouped with ``by``, return a Series of the above or a numpy
array:

* :class:`~pandas.Series`
* :class:`~numpy.ndarray` (for ``return_type = None``)

Use ``return_type='dict'`` when you want to tweak the appearance
of the lines after plotting. In this case a dict containing the Lines
making up the boxes, caps, fliers, medians, and whiskers is returned.

Examples
-----
Boxplots can be created for every column in the dataframe
by ``df.boxplot()`` or indicating the columns to be used:

.. plot::
   :context: close-figs

   >>> np.random.seed(1234)
   >>> df = pd.DataFrame(np.random.randn(10, 4),
   ...                      columns=['Col1', 'Col2', 'Col3', 'Col4'])
   >>> boxplot = df.boxplot(column=['Col1', 'Col2', 'Col3']) # doctest:
+SKIP

Boxplots of variables distributions grouped by the values of a third
variable can be created using the option ``by``. For instance:

.. plot::
   :context: close-figs

   >>> df = pd.DataFrame(np.random.randn(10, 2),
   ...                      columns=['Col1', 'Col2'])
   >>> df['X'] = pd.Series(['A', 'A', 'A', 'A',
   ...                        'B', 'B', 'B', 'B'])
   >>> boxplot = df.boxplot(by='X')

A list of strings (i.e. ``['X', 'Y']``) can be passed to boxplot
in order to group the data by combination of the variables in the x-axis:

.. plot::
   :context: close-figs

   >>> df = pd.DataFrame(np.random.randn(10, 3),
   ...                      columns=['Col1', 'Col2', 'Col3'])
   >>> df['X'] = pd.Series(['A', 'A', 'A', 'A',
   ...                        'B', 'B', 'B', 'B'])
   >>> df['Y'] = pd.Series(['A', 'B', 'A', 'B',
   ...                        'B', 'A', 'B', 'A'])
   >>> boxplot = df.boxplot(column=['Col1', 'Col2'], by=['X', 'Y'])

The layout of boxplot can be adjusted giving a tuple to ``layout``:

.. plot::
   :context: close-figs

   >>> boxplot = df.boxplot(column=['Col1', 'Col2'], by='X',
   ...                         layout=(2, 1))

Additional formatting can be done to the boxplot, like suppressing the
grid (``grid=False``), rotating the labels in the x-axis (i.e. ``rot=45``)
or changing the fontsize (i.e. ``fontsize=15``):

.. plot::
   :context: close-figs

   >>> boxplot = df.boxplot(grid=False, rot=45, fontsize=15) # doctest:
+SKIP

The parameter ``return_type`` can be used to select the type of element
returned by ``boxplot``. When ``return_type='axes'`` is selected,
the matplotlib axes on which the boxplot is drawn are returned:

   >>> boxplot = df.boxplot(column=['Col1', 'Col2'], return_type='axes')
   >>> type(boxplot)
   <class 'matplotlib.axes._subplots.AxesSubplot'>

When grouping with ``by``, a Series mapping columns to ``return_type``
is returned:

   >>> boxplot = df.boxplot(column=['Col1', 'Col2'], by='X',
   ...                         return_type='axes')
   >>> type(boxplot)
   <class 'pandas.core.series.Series'>

If ``return_type`` is ``None``, a NumPy array of axes with the same shape
as ``layout`` is returned:

   >>> boxplot = df.boxplot(column=['Col1', 'Col2'], by='X',
   ...                         return_type=None)
   >>> type(boxplot)
   <class 'numpy.ndarray'>

clip(self: 'DataFrame', lower=None, upper=None, axis: 'Axis | None' = None,
inplace: 'bool' = False, *args, **kwargs) -> 'DataFrame | None'
    Trim values at input threshold(s).

Assigns values outside boundary to boundary values. Thresholds
can be singular values or array like, and in the latter case
the clipping is performed element-wise in the specified axis.

Parameters
-----
lower : float or array-like, default None

```

```

    Minimum threshold value. All values below this
    threshold will be set to it. A missing
    threshold (e.g `NA') will not clip the value.
upper : float or array-like, default None
    Maximum threshold value. All values above this
    threshold will be set to it. A missing
    threshold (e.g `NA') will not clip the value.
axis : int or str axis name, optional
    Align object with lower and upper along the given axis.
inplace : bool, default False
    Whether to perform the operation in place on the data.
*args, **kwargs
    Additional keywords have no effect but might be accepted
    for compatibility with numpy.

Returns
-----
Series or DataFrame or None
    Same type as calling object with the values outside the
    clip boundaries replaced or None if ``inplace=True``.

See Also
-----
Series.clip : Trim values at input threshold in series.
DataFrame.clip : Trim values at input threshold in dataframe.
numpy.clip : Clip (limit) the values in an array.

Examples
-----
>>> data = {'col_0': [9, -3, 0, -1, 5], 'col_1': [-2, -7, 6, 8, -5]}
>>> df = pd.DataFrame(data)
>>> df
   col_0  col_1
0      9     -2
1     -3     -7
2      0      6
3     -1      6
4      5     -5

Clips per column using lower and upper thresholds:

>>> df.clip(-4, 6)
   col_0  col_1
0      6     -2
1     -3     -4
2      0      6
3     -1      6
4      5     -4

Clips using specific lower and upper thresholds per column element:

>>> t = pd.Series([2, -4, -1, 6, 3])
>>> t
0    2
1   -4
2   -1
3    6
4    3
dtype: int64

>>> df.clip(t, t + 4, axis=0)
   col_0  col_1
0      6      2
1     -3     -4
2      0      3
3      6      8
4      5      3

Clips using specific lower threshold per column element, with missing
values:
>>> t = pd.Series([2, -4, np.NaN, 6, 3])
>>> t
0    2.0
1   -4.0
2    NaN
3    6.0
4    3.0
dtype: float64

>>> df.clip(t, axis=0)
   col_0  col_1
0      9      2
1     -3     -4
2      0      6
3      6      8
4      5      3

combine(self, other: 'DataFrame', func, fill_value=None, overwrite: 'bool' =
True) -> 'DataFrame'
    Perform column-wise combine with another DataFrame.

    Combines a DataFrame with `other` DataFrame using `func`
    to element-wise combine columns. The row and column indexes of the
    resulting DataFrame will be the union of the two.

Parameters
-----
other : DataFrame
    The DataFrame to merge column-wise.
func : function
    Function that takes two series as inputs and return a Series or a
    scalar. Used to merge the two dataframes column by columns.
fill_value : scalar value, default None
    The value to fill NaNs with prior to passing any column to the
    merge func.
overwrite : bool, default True
    If True, columns in 'self' that do not exist in 'other' will be
    overwritten with NaNs.

Returns
-----
DataFrame
    Combination of the provided DataFrames.

See Also
-----
DataFrame.combine_first : Combine two DataFrame objects and default to
    non-null values in frame calling the method.

Examples
-----
Combine using a simple function that chooses the smaller column.

>>> df1 = pd.DataFrame({'A': [0, 0], 'B': [4, 4]})
>>> df2 = pd.DataFrame({'A': [1, 1], 'B': [3, 3]})
>>> take_smaller = lambda s1, s2: s1 if s1.sum() < s2.sum() else s2
>>> df1.combine(df2, take_smaller)
   A  B
0  0  3
1  0  3

```

```

Example using a true element-wise combine function.

>>> df1 = pd.DataFrame({'A': [5, 0], 'B': [2, 4]})
>>> df2 = pd.DataFrame({'A': [1, 1], 'B': [3, 3]})
>>> df1.combine(df2, np.minimum)
   A   B
0  1  2
1  0  3

Using `fill_value` fills Nones prior to passing the column to the
merge function.

>>> df1 = pd.DataFrame({'A': [0, 0], 'B': [None, 4]})
>>> df2 = pd.DataFrame({'A': [1, 1], 'B': [3, 3]})
>>> df1.combine(df2, take_smaller, fill_value=-5)
   A   B
0  0 -5.0
1  0  4.0

However, if the same element in both dataframes is None, that None
is preserved

>>> df1 = pd.DataFrame({'A': [0, 0], 'B': [None, 4]})
>>> df2 = pd.DataFrame({'A': [1, 1], 'B': [None, 3]})
>>> df1.combine(df2, take_smaller, fill_value=-5)
   A   B
0  0 -5.0
1  0  3.0

Example that demonstrates the use of `overwrite` and behavior when
the axis differ between the dataframes.

>>> df1 = pd.DataFrame({'A': [0, 0], 'B': [4, 4]})
>>> df2 = pd.DataFrame({'B': [3, 3], 'C': [-10, 1]}, index=[1, 2])
>>> df1.combine(df2, take_smaller)
   A   B   C
0  NaN  NaN  NaN
1  NaN  3.0 -10.0
2  NaN  3.0  1.0

>>> df1.combine(df2, take_smaller, overwrite=False)
   A   B   C
0  0.0  NaN  NaN
1  0.0  3.0 -10.0
2  NaN  3.0  1.0

Demonstrating the preference of the passed in dataframe.

>>> df2 = pd.DataFrame({'B': [3, 3], 'C': [1, 1]}, index=[1, 2])
>>> df2.combine(df1, take_smaller)
   A   B   C
0  0.0  NaN  NaN
1  0.0  3.0  NaN
2  NaN  3.0  NaN

>>> df2.combine(df1, take_smaller, overwrite=False)
   A   B   C
0  0.0  NaN  NaN
1  0.0  3.0  1.0
2  NaN  3.0  1.0

combine_first(self, other: 'DataFrame') -> 'DataFrame'
Update null elements with value in the same location in 'other'.

Combine two DataFrame objects by filling null values in one DataFrame
with non-null values from other DataFrame. The row and column indexes
of the resulting DataFrame will be the union of the two.

Parameters
-----
other : DataFrame
    Provided DataFrame to use to fill null values.

Returns
-----
DataFrame
    The result of combining the provided DataFrame with the other object.

See Also
-----
DataFrame.combine : Perform series-wise operation on two DataFrames
    using a given function.

Examples
-----
>>> df1 = pd.DataFrame({'A': [None, 0], 'B': [None, 4]})
>>> df2 = pd.DataFrame({'A': [1, 1], 'B': [3, 3]})
>>> df1.combine_first(df2)
   A   B
0  1.0  3.0
1  0.0  4.0

Null values still persist if the location of that null value
does not exist in 'other'.

>>> df1 = pd.DataFrame({'A': [None, 0], 'B': [4, None]})
>>> df2 = pd.DataFrame({'B': [3, 3], 'C': [1, 1]}, index=[1, 2])
>>> df1.combine_first(df2)
   A   B   C
0  NaN  4.0  NaN
1  0.0  3.0  1.0
2  NaN  3.0  1.0

compare(self, other: 'DataFrame', align_axis: 'Axis' = 1, keep_shape: 'bool' =
False, keep_equal: 'bool' = False) -> 'DataFrame'
    Compare to another DataFrame and show the differences.

.. versionadded:: 1.1.0

Parameters
-----
other : DataFrame
    Object to compare with.

align_axis : {0 or 'index', 1 or 'columns'}, default 1
    Determine which axis to align the comparison on.

    * 0, or 'index' : Resulting differences are stacked vertically
        with rows drawn alternately from self and other.
    * 1, or 'columns' : Resulting differences are aligned horizontally
        with columns drawn alternately from self and other.

keep_shape : bool, default False
    If true, all rows and columns are kept.
    Otherwise, only the ones with different values are kept.

keep_equal : bool, default False
    If true, the result keeps values that are equal.
    Otherwise, equal values are shown as NaNs.

```

Returns

DataFrame
 DataFrame that shows the differences stacked side by side.
 The resulting index will be a MultiIndex with 'self' and 'other'
 stacked alternately at the inner level.

Raises

ValueError
 When the two DataFrames don't have identical labels or shape.

See Also

Series.compare : Compare with another Series and show differences.
DataFrame.equals : Test whether two objects contain the same elements.

Notes

Matching NaNs will not appear as a difference.

Can only compare identically-labeled
(i.e. same shape, identical row and column labels) DataFrames

Examples

```
>>> df = pd.DataFrame(
...     {
...         "col1": ["a", "a", "b", "b", "a"],
...         "col2": [1.0, 2.0, 3.0, np.nan, 5.0],
...         "col3": [1.0, 2.0, 3.0, 4.0, 5.0]
...     },
...     columns=["col1", "col2", "col3"],
... )
>>> df
   col1  col2  col3
0    a    1.0   1.0
1    a    2.0   2.0
2    b    3.0   3.0
3    b    NaN   4.0
4    a    5.0   5.0

>>> df2 = df.copy()
>>> df2.loc[0, 'col1'] = 'c'
>>> df2.loc[2, 'col3'] = 4.0
>>> df2
   col1  col2  col3
0    c    1.0   1.0
1    a    2.0   2.0
2    b    3.0   4.0
3    b    NaN   4.0
4    a    5.0   5.0

Align the differences on columns

>>> df.compare(df2)
      col1      col3
      self other self other
0    a      c  NaN  NaN
2  NaN  NaN  3.0  4.0

Stack the differences on rows

>>> df.compare(df2, align_axis=0)
      col1      col3
0  self      a  NaN
  other      c  NaN
2  self  NaN  3.0
  other  NaN  4.0

Keep the equal values

>>> df.compare(df2, keep_equal=True)
      col1      col3
      self other self other
0    a      c  1.0  1.0
2    b      b  3.0  4.0

Keep all original rows and columns

>>> df.compare(df2, keep_shape=True)
      col1      col2      col3
      self other self other self other
0    a      c  NaN  NaN  NaN  NaN
1  NaN  NaN  NaN  NaN  NaN  NaN
2  NaN  NaN  NaN  NaN  3.0  4.0
3  NaN  NaN  NaN  NaN  NaN  NaN
4  NaN  NaN  NaN  NaN  NaN  NaN

Keep all original rows and columns and also all original values

>>> df.compare(df2, keep_shape=True, keep_equal=True)
      col1      col2      col3
      self other self other self other
0    a      c  1.0  1.0  1.0
1    a      a  2.0  2.0  2.0
2    b      b  3.0  3.0  4.0
3    b      b  NaN  NaN  4.0
4    a      a  5.0  5.0  5.0
```

`corr(self, method: 'str | Callable[[np.ndarray, np.ndarray], float]' = 'pearson', min_periods: 'int' = 1) -> 'DataFrame'`
Compute pairwise correlation of columns, excluding NA/null values.

Parameters

method : {'pearson', 'kendall', 'spearman'} or callable
 Method of correlation:
 * `pearson` : standard correlation coefficient
 * `kendall` : Kendall Tau correlation coefficient
 * `spearman` : Spearman rank correlation
 * `callable`: callable with input two 1d ndarrays
 and returning a float. Note that the returned matrix from `corr`
 will have 1 along the diagonals and will be symmetric
 regardless of the callable's behavior.
min_periods : int, optional
 Minimum number of observations required per pair of columns
 to have a valid result. Currently only available for Pearson
 and Spearman correlation.

Returns

DataFrame
 Correlation matrix.

See Also

DataFrame.corrwith : Compute pairwise correlation with another
 DataFrame or Series.

```

Series.corr : Compute the correlation between two Series.

Examples
-----
>>> def histogram_intersection(a, b):
...     v = np.minimum(a, b).sum().round(decimals=1)
...     return v
>>> df = pd.DataFrame([(., .2), (.0, .3), (.6, .0), (.2, .1)],
...                      columns=['dogs', 'cats'])
>>> df.corr(method=histogram_intersection)
   dogs  cats
dogs  1.0  0.3
cats  0.3  1.0

corrwith(self, other, axis: 'Axis' = 0, drop=False, method='pearson') ->
'Series'
    Compute pairwise correlation.

Pairwise correlation is computed between rows or columns of DataFrame with rows or columns of Series or DataFrame. DataFrames are first aligned along both axes before computing the correlations.

Parameters
-----
other : DataFrame, Series
    Object with which to compute correlations.
axis : {0 or 'index', 1 or 'columns'}, default 0
    The axis to use. 0 or 'index' to compute column-wise, 1 or 'columns' for
    row-wise.
drop : bool, default False
    Drop missing indices from result.
method : {'pearson', 'kendall', 'spearman'} or callable
    Method of correlation:
        * pearson : standard correlation coefficient
        * kendall : Kendall Tau correlation coefficient
        * spearman : Spearman rank correlation
        * callable: callable with input two 1d ndarrays
            and returning a float.

Returns
-----
Series
    Pairwise correlations.

See Also
-----
DataFrame.corr : Compute pairwise correlation of columns.

count(self, axis: 'Axis' = 0, level: 'Level' | None = None, numeric_only:
'bool' = False)
    Count non-NA cells for each column or row.

    The values 'None', 'NaN', 'NaT', and optionally 'numpy.inf' (depending
    on `pandas.options.mode.use_inf_as_na`) are considered NA.

Parameters
-----
axis : {0 or 'index', 1 or 'columns'}, default 0
    If 0 or 'index' counts are generated for each column.
    If 1 or 'columns' counts are generated for each row.
level : int or str, optional
    If the axis is a 'MultiIndex' (hierarchical), count along a
    particular 'level', collapsing into a 'DataFrame'.
    A 'str' specifies the level name.
numeric_only : bool, default False
    Include only 'float', 'int' or 'boolean' data.

Returns
-----
Series or DataFrame
    For each column/row the number of non-NA/null entries.
    If 'level' is specified returns a 'DataFrame'.

See Also
-----
Series.count: Number of non-NA elements in a Series.
DataFrame.value_counts: Count unique combinations of columns.
DataFrame.shape: Number of DataFrame rows and columns (including NA
elements).
DataFrame.isna: Boolean same-sized DataFrame showing places of NA
elements.

Examples
-----
Constructing DataFrame from a dictionary:

>>> df = pd.DataFrame({'Person':
...                     ['John', 'Myla', 'Lewis', 'John', 'Myla'],
...                     'Age': [24., np.nan, 21., 33, 26],
...                     'Single': [False, True, True, True, False]})
>>> df
   Person  Age  Single
0   John  24.0   False
1   Myla   NaN    True
2   Lewis  21.0   True
3   John  33.0   True
4   Myla  26.0   False

Notice the uncounted NA values:

>>> df.count()
Person    5
Age      4
Single   5
dtype: int64

Counts for each **row**:

>>> df.count(axis='columns')
0    3
1    2
2    3
3    3
4    3
dtype: int64

cov(self, min_periods: 'int' | None = None, ddof: 'int' | None = 1) ->
'DataFrame'
    Compute pairwise covariance of columns, excluding NA/null values.

    Compute the pairwise covariance among the series of a DataFrame.
    The returned data frame is the 'covariance matrix
    <https://en.wikipedia.org/wiki/Covariance_matrix>__ of the columns
    of the DataFrame.

    Both NA and null values are automatically excluded from the
    calculation. (See the note below about bias from missing values.)
    A threshold can be set for the minimum number of

```

observations for each value created. Comparisons with observations below this threshold will be returned as ``NaN``.

This method is generally used for the analysis of time series data to understand the relationship between different measures across time.

Parameters

`min_periods : int, optional`
Minimum number of observations required per pair of columns to have a valid result.

`ddof : int, default 1`
Delta degrees of freedom. The divisor used in calculations is ``N - ddof``, where ``N`` represents the number of elements.

.. versionadded:: 1.1.0

Returns

`DataFrame`
The covariance matrix of the series of the DataFrame.

See Also

`Series.cov` : Compute covariance with another Series.
`core.window.ExponentialMovingWindow.cov`: Exponential weighted sample covariance.
`core.window.Expanding.cov` : Expanding sample covariance.
`core.window.Rolling.cov` : Rolling sample covariance.

Notes

Returns the covariance matrix of the DataFrame's time series.
The covariance is normalized by N-ddof.

For DataFrames that have Series that are missing data (assuming that data is `missing at random` <https://en.wikipedia.org/wiki/Missing_data#Missing_at_random>`) the returned covariance matrix will be an unbiased estimate of the variance and covariance between the member Series.

However, for many applications this estimate may not be acceptable because the estimate covariance matrix is not guaranteed to be positive semi-definite. This could lead to estimate correlations having absolute values which are greater than one, and/or a non-invertible covariance matrix. See [Estimation of covariance matrices](https://en.wikipedia.org/w/index.php?title=Estimation_of_covariance_matrices) for more details.

Examples

```
>>> df = pd.DataFrame([(1, 2), (0, 3), (2, 0), (1, 1)],
...                   columns=['dogs', 'cats'])
>>> df.cov()
      dogs      cats
dogs  0.666667 -1.000000
cats -1.000000  1.666667

>>> np.random.seed(42)
>>> df = pd.DataFrame(np.random.randn(1000, 5),
...                   columns=['a', 'b', 'c', 'd', 'e'])
>>> df.cov()
           a         b         c         d         e
a  0.998438 -0.020161  0.059277 -0.008943  0.014144
b -0.020161  1.059352 -0.008543 -0.024738  0.009826
c  0.059277 -0.008543  1.019670 -0.001486 -0.000271
d -0.008943 -0.024738 -0.001486  0.921297 -0.013692
e  0.014144  0.009826 -0.000271 -0.013692  0.977795

**Minimum number of periods**
```

This method also supports an optional ``min_periods`` keyword that specifies the required minimum number of non-NA observations for each column pair in order to have a valid result:

```
>>> np.random.seed(42)
>>> df = pd.DataFrame(np.random.randn(20, 3),
...                   columns=['a', 'b', 'c'])
>>> df.loc[df.index[:5], 'a'] = np.nan
>>> df.loc[df.index[5:10], 'b'] = np.nan
>>> df.cov(min_periods=12)
           a         b         c
a  0.316741      NaN -0.150812
b      NaN  1.248003  0.191417
c -0.150812  0.191417  0.895202

cummax(self, axis=None, skipna=True, *args, **kwargs)
Return cumulative maximum over a DataFrame or Series axis.
```

Returns a DataFrame or Series of the same size containing the cumulative maximum.

Parameters

`axis : {0 or 'index', 1 or 'columns'}, default 0`
The index or the name of the axis. 0 is equivalent to None or 'index'.

`skipna : bool, default True`
Exclude NA/null values. If an entire row/column is NA, the result will be NA.

`*args, **kwargs`
Additional keywords have no effect but might be accepted for compatibility with NumPy.

Returns

`Series or DataFrame`
Return cumulative maximum of Series or DataFrame.

See Also

`core.window.Expanding.max` : Similar functionality but ignores ``NaN`` values.
`DataFrame.max` : Return the maximum over DataFrame axis.
`DataFrame.cummax` : Return cumulative maximum over DataFrame axis.
`DataFrame.cummin` : Return cumulative minimum over DataFrame axis.
`DataFrame.cumsum` : Return cumulative sum over DataFrame axis.
`DataFrame.cumprod` : Return cumulative product over DataFrame axis.

Examples

Series

```
>>> s = pd.Series([2, np.nan, 5, -1, 0])
>>> s
0    2.0
1    NaN
2    5.0
3   -1.0
```

```

4    0.0
dtype: float64

By default, NA values are ignored.

>>> s.cummax()
0    2.0
1    NaN
2    5.0
3    5.0
4    5.0
dtype: float64

To include NA values in the operation, use ``skipna=False``

>>> s.cummax(skipna=False)
0    2.0
1    NaN
2    NaN
3    NaN
4    NaN
dtype: float64

**DataFrame**

>>> df = pd.DataFrame([[2.0, 1.0],
...                      [3.0, np.nan],
...                      [1.0, 0.0]],
...                     columns=list('AB'))
>>> df
   A    B
0  2.0  1.0
1  3.0  NaN
2  1.0  0.0

By default, iterates over rows and finds the maximum
in each column. This is equivalent to ``axis=None`` or ``axis='index'``.

>>> df.cummax()
   A    B
0  2.0  1.0
1  3.0  NaN
2  3.0  1.0

To iterate over columns and find the maximum in each row,
use ``axis=1``

>>> df.cummax(axis=1)
   A    B
0  2.0  2.0
1  3.0  NaN
2  1.0  1.0

cummin(self, axis=None, skipna=True, *args, **kwargs)
    Return cumulative minimum over a DataFrame or Series axis.

Returns a DataFrame or Series of the same size containing the cumulative
minimum.

Parameters
-----
axis : {0 or 'index', 1 or 'columns'}, default 0
    The index or the name of the axis. 0 is equivalent to None or 'index'.
skipna : bool, default True
    Exclude NA/null values. If an entire row/column is NA, the result
    will be NA.
*args, **kwargs
    Additional keywords have no effect but might be accepted for
    compatibility with NumPy.

Returns
-----
Series or DataFrame
    Return cumulative minimum of Series or DataFrame.

See Also
-----
core.window.Expanding.min : Similar functionality
    but ignores "NaN" values.
DataFrame.min : Return the minimum over
    DataFrame axis.
DataFrame.cummax : Return cumulative maximum over DataFrame axis.
DataFrame.cummin : Return cumulative minimum over DataFrame axis.
DataFrame.cumsum : Return cumulative sum over DataFrame axis.
DataFrame.cumprod : Return cumulative product over DataFrame axis.

Examples
-----
**Series**

>>> s = pd.Series([2, np.nan, 5, -1, 0])
>>> s
0    2.0
1    NaN
2    5.0
3   -1.0
4    0.0
dtype: float64

By default, NA values are ignored.

>>> s.cummin()
0    2.0
1    NaN
2    2.0
3   -1.0
4   -1.0
dtype: float64

To include NA values in the operation, use ``skipna=False``

>>> s.cummin(skipna=False)
0    2.0
1    NaN
2    NaN
3    NaN
4    NaN
dtype: float64

**DataFrame**

>>> df = pd.DataFrame([[2.0, 1.0],
...                      [3.0, np.nan],
...                      [1.0, 0.0]],
...                     columns=list('AB'))
>>> df
   A    B
0  2.0  1.0
1  3.0  NaN
2  1.0  0.0

```

By default, iterates over rows and finds the minimum in each column. This is equivalent to ``axis=None`` or ``axis='index'``.

```
>>> df.cummin()
   A    B
0  2.0  1.0
1  2.0  NaN
2  1.0  0.0
```

To iterate over columns and find the minimum in each row, use ``axis=1``

```
>>> df.cummin(axis=1)
   A    B
0  2.0  1.0
1  3.0  NaN
2  1.0  0.0
```

`cumprod(self, axis=None, skipna=True, *args, **kwargs)`
Return cumulative product over a DataFrame or Series axis.

Returns a DataFrame or Series of the same size containing the cumulative product.

Parameters

axis : {0 or 'index', 1 or 'columns'}, default 0
The index or the name of the axis. 0 is equivalent to None or 'index'.
skipna : bool, default True
Exclude NA/null values. If an entire row/column is NA, the result will be NA.
*args, **kwargs
Additional keywords have no effect but might be accepted for compatibility with NumPy.

Returns

Series or DataFrame
Return cumulative product of Series or DataFrame.

See Also

core.window.Expanding.prod : Similar functionality but ignores ``NaN`` values.
DataFrame.prod : Return the product over DataFrame axis.
DataFrame.cummax : Return cumulative maximum over DataFrame axis.
DataFrame.cummin : Return cumulative minimum over DataFrame axis.
DataFrame.cumsum : Return cumulative sum over DataFrame axis.
DataFrame.cumprod : Return cumulative product over DataFrame axis.

Examples

Series

```
>>> s = pd.Series([2, np.nan, 5, -1, 0])
>>> s
0    2.0
1    NaN
2    5.0
3   -1.0
4    0.0
dtype: float64
```

By default, NA values are ignored.

```
>>> s.cumprod()
0    2.0
1    NaN
2   10.0
3  -10.0
4   -0.0
dtype: float64
```

To include NA values in the operation, use ``skipna=False``

```
>>> s.cumprod(skipna=False)
0    2.0
1    NaN
2    NaN
3    NaN
4    NaN
dtype: float64
```

DataFrame

```
>>> df = pd.DataFrame([[2.0, 1.0],
...                     [3.0, np.nan],
...                     [1.0, 0.0]],
...                     columns=list('AB'))
>>> df
   A    B
0  2.0  1.0
1  3.0  NaN
2  1.0  0.0
```

By default, iterates over rows and finds the product in each column. This is equivalent to ``axis=None`` or ``axis='index'``.

```
>>> df.cumprod()
   A    B
0  2.0  1.0
1  6.0  NaN
2  6.0  0.0
```

To iterate over columns and find the product in each row, use ``axis=1``

```
>>> df.cumprod(axis=1)
   A    B
0  2.0  2.0
1  3.0  NaN
2  1.0  0.0
```

`cumsum(self, axis=None, skipna=True, *args, **kwargs)`
Return cumulative sum over a DataFrame or Series axis.

Returns a DataFrame or Series of the same size containing the cumulative sum.

Parameters

axis : {0 or 'index', 1 or 'columns'}, default 0
The index or the name of the axis. 0 is equivalent to None or 'index'.
skipna : bool, default True
Exclude NA/null values. If an entire row/column is NA, the result will be NA.
*args, **kwargs
Additional keywords have no effect but might be accepted for compatibility with NumPy.

Returns

Series or DataFrame
 Return cumulative sum of Series or DataFrame.

See Also

`core.window.Expanding.sum` : Similar functionality
 but ignores ``NaN`` values.
`DataFrame.sum` : Return the sum over
 Dataframe axis.
`DataFrame.cummax` : Return cumulative maximum over DataFrame axis.
`DataFrame.cummin` : Return cumulative minimum over DataFrame axis.
`DataFrame.cumsum` : Return cumulative sum over DataFrame axis.
`DataFrame.cumprod` : Return cumulative product over DataFrame axis.

Examples

****Series****

```
>>> s = pd.Series([2, np.nan, 5, -1, 0])
>>> s
0    2.0
1    NaN
2    5.0
3   -1.0
4    0.0
dtype: float64

By default, NA values are ignored.

>>> s.cumsum()
0    2.0
1    NaN
2    7.0
3    6.0
4    6.0
dtype: float64

To include NA values in the operation, use ``skipna=False``

>>> s.cumsum(skipna=False)
0    2.0
1    NaN
2    NaN
3    NaN
4    NaN
dtype: float64
```

****DataFrame****

```
>>> df = pd.DataFrame([[2.0, 1.0],
...                     [3.0, np.nan],
...                     [1.0, 0.0]],
...                     columns=list('AB'))
>>> df
      A      B
0  2.0  1.0
1  3.0  NaN
2  1.0  0.0

By default, iterates over rows and finds the sum
in each column. This is equivalent to ``axis=None`` or ``axis='index'``.

>>> df.cumsum()
      A      B
0  2.0  1.0
1  5.0  NaN
2  6.0  1.0

To iterate over columns and find the sum in each row,
use ``axis=1``

>>> df.cumsum(axis=1)
      A      B
0  2.0  3.0
1  3.0  NaN
2  1.0  1.0

diff(self, periods: int = 1, axis: 'Axis' = 0) -> 'DataFrame'
First discrete difference of element.

Calculates the difference of a Dataframe element compared with another
element in the Dataframe (default is element in previous row).



Parameters  
-----  
periods : int, default 1  
    Periods to shift for calculating difference, accepts negative  
    values.  
axis : {0 or 'index', 1 or 'columns'}, default 0  
    Take difference over rows (0) or columns (1).



Returns  
-----  
Dataframe  
    First differences of the Series.



See Also  
-----  
Dataframe.pct_change: Percent change over given number of periods.  
Dataframe.shift: Shift index by desired number of periods with an  
    optional time freq.  
Series.diff: First discrete difference of object.



Notes  
----  
For boolean dtypes, this uses :meth:`operator.xor` rather than  
:meth:`operator.sub`.  
The result is calculated according to current dtype in Dataframe,  
however dtype of the result is always float64.



Examples  
-----



Difference with previous row



```
>>> df = pd.DataFrame({'a': [1, 2, 3, 4, 5, 6],
... 'b': [1, 1, 2, 3, 5, 8],
... 'c': [1, 4, 9, 16, 25, 36]})
>>> df
 a b c
0 1 1 1
1 2 1 4
2 3 2 9
3 4 3 16
4 5 5 25
5 6 8 36

>>> df.diff()
 a b c
```


```

```

0   NaN   NaN   NaN
1   1.0   0.0   3.0
2   1.0   1.0   5.0
3   1.0   1.0   7.0
4   1.0   2.0   9.0
5   1.0   3.0  11.0

Difference with previous column

>>> df.diff(axis=1)
      a    b    c
0   NaN   0   0
1   NaN -1   3
2   NaN -1   7
3   NaN -1  13
4   NaN   0  20
5   NaN   2  28

Difference with 3rd previous row

>>> df.diff(periods=3)
      a    b    c
0   NaN   NaN   NaN
1   NaN   NaN   NaN
2   NaN   NaN   NaN
3   3.0   2.0  15.0
4   3.0   4.0  21.0
5   3.0   6.0  27.0

Difference with following row

>>> df.diff(periods=-1)
      a    b    c
0  -1.0   0.0  -3.0
1  -1.0  -1.0  -5.0
2  -1.0  -1.0  -7.0
3  -1.0  -2.0  -9.0
4  -1.0  -3.0 -11.0
5   NaN   NaN   NaN

Overflow in input dtype

>>> df = pd.DataFrame({'a': [1, 0]}, dtype=np.uint8)
>>> df.diff()
      a
0   NaN
1  255.0

div = truediv(self, other, axis='columns', level=None, fill_value=None)
divide = truediv(self, other, axis='columns', level=None, fill_value=None)

dot(self, other: 'AnyArrayLike | DataFrame') -> 'DataFrame | Series'
    Compute the matrix multiplication between the DataFrame and other.

    This method computes the matrix product between the DataFrame and the
    values of an other Series, DataFrame or a numpy array.

    It can also be called using ``self @ other`` in Python >= 3.5.

Parameters
-----
other : Series, DataFrame or array-like
    The other object to compute the matrix product with.

Returns
-----
Series or DataFrame
    If other is a Series, return the matrix product between self and
    other as a Series. If other is a DataFrame or a numpy.array, return
    the matrix product of self and other in a DataFrame of a np.array.

See Also
-----
Series.dot: Similar method for Series.

Notes
-----
The dimensions of DataFrame and other must be compatible in order to
compute the matrix multiplication. In addition, the column names of
DataFrame and the index of other must contain the same values, as they
will be aligned prior to the multiplication.

The dot method for Series computes the inner product, instead of the
matrix product here.

Examples
-----
Here we multiply a DataFrame with a Series.

>>> df = pd.DataFrame([[0, 1, -2, -1], [1, 1, 1, 1]])
>>> s = pd.Series([1, 2, 1])
>>> df.dot(s)
0   -4
1    5
dtype: int64

Here we multiply a DataFrame with another DataFrame.

>>> other = pd.DataFrame([[0, 1], [1, 2], [-1, -1], [2, 0]])
>>> df.dot(other)
0    1
0    1
1    4
1    2
2    2

Note that the dot method give the same result as @

>>> df @ other
0    1
0    1
1    4
1    2
2    2

The dot method works also if other is an np.array.

>>> arr = np.array([[0, 1], [1, 2], [-1, -1], [2, 0]])
>>> df.dot(arr)
0    1
0    1
1    4
1    2
2    2

Note how shuffling of the objects does not change the result.

>>> s2 = s.reindex([1, 0, 2, 3])
>>> df.dot(s2)
0    -4
1     5
2    12
dtype: int64

drop(self, labels=None, axis: 'Axis' = 0, index=None, columns=None, level:
'Level' | None' = None, inplace: 'bool' = False, errors: 'str' = 'raise')
|     Drop specified labels from rows or columns.

```

Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. When using a multi-index, labels on different levels can be removed by specifying the level. See the user guide <advanced.shown_levels> for more information about the now unused levels.

Parameters

labels : single label or list-like
 Index or column labels to drop. A tuple will be used as a single label and not treated as a list-like.

axis : {0 or 'index', 1 or 'columns'}, default 0
 Whether to drop labels from the index (0 or 'index') or columns (1 or 'columns').

index : single label or list-like
 Alternative to specifying axis ('`labels, axis=0`' is equivalent to ``index=labels``).

columns : single label or list-like
 Alternative to specifying axis ('`labels, axis=1`' is equivalent to ``columns=labels``).

level : int or level name, optional
 For MultiIndex, level from which the labels will be removed.

inplace : bool, default False
 If False, return a copy. Otherwise, do operation inplace and return None.

errors : {'ignore', 'raise'}, default 'raise'
 If 'ignore', suppress error and only existing labels are dropped.

Returns

DataFrame or None
 Dataframe without the removed index or column labels or None if ``inplace=True``.

Raises

KeyError
 If any of the labels is not found in the selected axis.

See Also

DataFrame.loc : Label-location based indexer for selection by label.
 DataFrame.dropna : Return DataFrame with labels on given axis omitted where (all or any) data are missing.
 DataFrame.drop_duplicates : Return DataFrame with duplicate rows removed, optionally only considering certain columns.
 Series.drop : Return Series with specified index labels removed.

Examples

```
>>> df = pd.DataFrame(np.arange(12).reshape(3, 4),
...                     columns=['A', 'B', 'C', 'D'])
>>> df
   A  B  C  D
0  0  1  2  3
1  4  5  6  7
2  8  9  10 11

Drop columns

>>> df.drop(['B', 'C'], axis=1)
   A  D
0  0  3
1  4  7
2  8  11

>>> df.drop(columns=['B', 'C'])
   A  D
0  0  3
1  4  7
2  8  11

Drop a row by index

>>> df.drop([0, 1])
   A  B  C  D
2  8  9  10 11

Drop columns and/or rows of MultiIndex DataFrame

>>> midx = pd.MultiIndex(levels=[['lama', 'cow', 'falcon'],
...                           ['speed', 'weight', 'length']],
...                        codes=[[0, 0, 0, 1, 1, 2, 2, 2],
...                               [0, 1, 2, 0, 1, 2, 0, 1, 2]])
>>> df = pd.DataFrame(index=midx, columns=['big', 'small'],
...                     data=[[45, 30], [200, 100], [1.5, 1], [30, 20],
...                           [250, 150], [1.5, 0.8], [320, 250],
...                           [1, 0.8], [0.3, 0.2]])
>>> df
           big      small
lama  speed  45.0    30.0
      weight 200.0   100.0
      length  1.5    1.0
cow   speed  30.0    20.0
      weight 250.0   150.0
      length  1.5    0.8
falcon speed 320.0   250.0
      weight  1.0    0.8
      length  0.3    0.2

Drop a specific index combination from the MultiIndex DataFrame, i.e., drop the combination ``'falcon'`` and ``'weight'``, which deletes only the corresponding row

>>> df.drop(index=('falcon', 'weight'))
           big      small
lama  speed  45.0    30.0
      weight 200.0   100.0
      length  1.5    1.0
cow   speed  30.0    20.0
      weight 250.0   150.0
      length  1.5    0.8
falcon speed 320.0   250.0
      length  0.3    0.2

>>> df.drop(index='cow', columns='small')
           big
lama  speed  45.0
      weight 200.0
      length  1.5
falcon speed 320.0
      weight  1.0
      length  0.3

>>> df.drop(index='length', level=1)
           big      small
lama  speed  45.0    30.0
      weight 200.0   100.0
cow   speed  30.0    20.0
```

```

        weight  250.0  150.0
falcon   speed  320.0  250.0
        weight  1.0    0.8

| drop_duplicates(self, subset: 'Hashable | Sequence[Hashable] | None' = None,
| keep: "Literal['first'] | Literal['last']" = 'first', inplace:
| 'bool' = False, ignore_index: 'bool' = False) -> 'DataFrame | None'
|     Return DataFrame with duplicate rows removed.

| Considering certain columns is optional. Indexes, including time indexes
| are ignored.

Parameters
-----
subset : column label or sequence of labels, optional
    Only consider certain columns for identifying duplicates, by
    default use all of the columns.
keep : ('first', 'last', False), default 'first'
    Determines which duplicates (if any) to keep.
    - 'first' : Drop duplicates except for the first occurrence.
    - 'last' : Drop duplicates except for the last occurrence.
    - False : Drop all duplicates.
inplace : bool, default False
    Whether to drop duplicates in place or to return a copy.
ignore_index : bool, default False
    If True, the resulting axis will be labeled 0, 1, ..., n - 1.
.. versionadded:: 1.0.0

Returns
-----
DataFrame or None
    DataFrame with duplicates removed or None if ``inplace=True``.

See Also
-----
DataFrame.value_counts: Count unique combinations of columns.

Examples
-----
Consider dataset containing ramen rating.

>>> df = pd.DataFrame({
...     'brand': ['Yum Yum', 'Yum Yum', 'Indomie', 'Indomie', 'Indomie'],
...     'style': ['cup', 'cup', 'cup', 'pack', 'pack'],
...     'rating': [4, 4, 3.5, 15, 5]
... })
>>> df
   brand style rating
0  Yum Yum   cup    4.0
1  Yum Yum   cup    4.0
2  Indomie   cup    3.5
3  Indomie  pack   15.0
4  Indomie  pack    5.0

By default, it removes duplicate rows based on all columns.

>>> df.drop_duplicates()
   brand style rating
0  Yum Yum   cup    4.0
1  Indomie   cup    3.5
2  Indomie  pack   15.0
3  Indomie  pack    5.0

To remove duplicates on specific column(s), use ``subset``.

>>> df.drop_duplicates(subset=['brand'])
   brand style rating
0  Yum Yum   cup    4.0
2  Indomie   cup    3.5

To remove duplicates and keep last occurrences, use ``keep``.

>>> df.drop_duplicates(subset=['brand', 'style'], keep='last')
   brand style rating
1  Yum Yum   cup    4.0
2  Indomie   cup    3.5
4  Indomie  pack    5.0

| dropna(self, axis: 'Axis' = 0, how: 'str' = 'any', thresh=None, subset:
| 'IndexLabel' = None, inplace: 'bool' = False)
|     Remove missing values.
|
| See the :ref:`User Guide <missing_data>` for more on which values are
| considered missing, and how to work with missing data.

Parameters
-----
axis : {0 or 'index', 1 or 'columns'}, default 0
    Determine if rows or columns which contain missing values are
    removed.

    * 0, or 'index' : Drop rows which contain missing values.
    * 1, or 'columns' : Drop columns which contain missing value.
.. versionchanged:: 1.0.0

    Pass tuple or list to drop on multiple axes.
    Only a single axis is allowed.

how : {'any', 'all'}, default 'any'
    Determine if row or column is removed from DataFrame, when we have
    at least one NA or all NA.

    * 'any' : If any NA values are present, drop that row or column.
    * 'all' : If all values are NA, drop that row or column.

thresh : int, optional
    Require that many non-NA values.
subset : column label or sequence of labels, optional
    Labels along other axis to consider, e.g. if you are dropping rows
    these would be a list of columns to include.
inplace : bool, default False
    If True, do operation inplace and return None.

Returns
-----
DataFrame or None
    DataFrame with NA entries dropped from it or None if ``inplace=True``.

See Also
-----
DataFrame.isna: Indicate missing values.
DataFrame.notna : Indicate existing (non-missing) values.
DataFrame.fillna : Replace missing values.
Series.dropna : Drop missing values.
Index.dropna : Drop missing indices.

Examples
-----
>>> df = pd.DataFrame({'name': ['Alfred', 'Batman', 'Catwoman'],

```

```

...
"toy": [np.nan, 'Batmobile', 'Bullwhip'],
"born": [pd.NaT, pd.Timestamp("1940-04-25"),
          pd.NaT])
>>> df
      name      toy      born
0   Alfred     NaN     NaT
1   Batman  Batmobile 1940-04-25
2  Catwoman  Bullwhip     NaT

Drop the rows where at least one element is missing.

>>> df.dropna()
      name      toy      born
1   Batman  Batmobile 1940-04-25

Drop the columns where at least one element is missing.

>>> df.dropna(axis='columns')
      name
0   Alfred
1   Batman
2  Catwoman

Drop the rows where all elements are missing.

>>> df.dropna(how='all')
      name      toy      born
0   Alfred     NaN     NaT
1   Batman  Batmobile 1940-04-25
2  Catwoman  Bullwhip     NaT

Keep only the rows with at least 2 non-NA values.

>>> df.dropna(thresh=2)
      name      toy      born
1   Batman  Batmobile 1940-04-25
2  Catwoman  Bullwhip     NaT

Define in which columns to look for missing values.

>>> df.dropna(subset=['name', 'toy'])
      name      toy      born
1   Batman  Batmobile 1940-04-25
2  Catwoman  Bullwhip     NaT

Keep the DataFrame with valid entries in the same variable.

>>> df.dropna(inplace=True)
>>> df
      name      toy      born
1   Batman  Batmobile 1940-04-25

duplicated(self, subset: 'Hashable | Sequence[Hashable] | None' = None, keep:
"literal['first'] | Literal['last'] | Literal[False]" = 'first') -> 'Series'
Return boolean Series denoting duplicate rows.

Considering certain columns is optional.

Parameters
-----
subset : column label or sequence of labels, optional
    Only consider certain columns for identifying duplicates, by
    default use all of the columns.
keep : {'first', 'last', False}, default 'first'
    Determines which duplicates (if any) to mark.

    - ``first`` : Mark duplicates as ``True`` except for the first
occurrence.
    - ``last`` : Mark duplicates as ``True`` except for the last
occurrence.
    - False : Mark all duplicates as ``True``.

Returns
-----
Series
    Boolean series for each duplicated rows.

See Also
-----
Index.duplicated : Equivalent method on index.
Series.duplicated : Equivalent method on Series.
Series.drop_duplicates : Remove duplicate values from Series.
DataFrame.drop_duplicates : Remove duplicate values from DataFrame.

Examples
-----
Consider dataset containing ramen rating.

>>> df = pd.DataFrame({
...     'brand': ['Yum Yum', 'Yum Yum', 'Indomie', 'Indomie', 'Indomie'],
...     'style': ['cup', 'cup', 'cup', 'pack', 'pack'],
...     'rating': [4, 4, 3.5, 15, 5]
... })
>>> df
   brand style  rating
0  Yum Yum    cup    4.0
1  Yum Yum    cup    4.0
2   Indomie    cup    3.5
3   Indomie   pack   15.0
4   Indomie   pack    5.0

By default, for each set of duplicated values, the first occurrence
is set on False and all others on True.

>>> df.duplicated()
0   False
1    True
2   False
3   False
4   False
dtype: bool

By using 'last', the last occurrence of each set of duplicated values
is set on False and all others on True.

>>> df.duplicated(keep='last')
0    True
1   False
2   False
3   False
4   False
dtype: bool

By setting ``keep`` on False, all duplicates are True.

>>> df.duplicated(keep=False)
0    True
1    True
2   False
3   False
4   False

```

```

dtype: bool
To find duplicates on specific column(s), use ``subset``.

>>> df.duplicated(subset=['brand'])
0    False
1     True
2    False
3     True
4    True
dtype: bool

eq(self, other, axis='columns', level=None)
Get Equal to of DataFrame and other, element-wise (binary operator `eq`).

Among flexible wrappers (`eq`, `ne`, `le`, `lt`, `ge`, `gt`) to comparison
operators.

Equivalent to `==`, `!=`, `<=`, `<`, `>=`, `>` with support to choose axis
(rows or columns) and level for comparison.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}, default 'columns'
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns').
level : int or label
    Broadcast across a level, matching Index values on the passed
    MultiIndex level.

Returns
-----
DataFrame of bool
    Result of the comparison.

See Also
-----
DataFrame.eq : Compare DataFrames for equality elementwise.
DataFrame.ne : Compare DataFrames for inequality elementwise.
DataFrame.le : Compare DataFrames for less than inequality
    or equality elementwise.
DataFrame.lt : Compare DataFrames for strictly less than
    inequality elementwise.
DataFrame.ge : Compare DataFrames for greater than inequality
    or equality elementwise.
DataFrame.gt : Compare DataFrames for strictly greater than
    inequality elementwise.

Notes
-----
Mismatched indices will be unioned together.
`NaN` values are considered different (i.e. `NaN` != `NaN`).

Examples
-----
>>> df = pd.DataFrame({'cost': [250, 150, 100],
...                      'revenue': [100, 250, 300]},
...                     index=['A', 'B', 'C'])
>>> df
   cost  revenue
A    250      100
B    150      250
C    100      300

Comparison with a scalar, using either the operator or method:

>>> df == 100
   cost  revenue
A  False    True
B  False   False
C  True   False

>>> df.eq(100)
   cost  revenue
A  False    True
B  False   False
C  True   False

When `other` is a :class:`Series`, the columns of a DataFrame are aligned
with the index of `other` and broadcast:

>>> df != pd.Series([100, 250], index=["cost", "revenue"])
   cost  revenue
A  True    True
B  True   False
C  False   True

Use the method to control the broadcast axis:

>>> df.ne(pd.Series([100, 300], index=["A", "D"]), axis='index')
   cost  revenue
A  True    False
B  True    True
C  True    True
D  True    True

When comparing to an arbitrary sequence, the number of columns must
match the number elements in `other`:

>>> df == [250, 100]
   cost  revenue
A  True    True
B  False   False
C  False   False

Use the method to control the axis:

>>> df.eq([250, 250, 100], axis='index')
   cost  revenue
A  True    False
B  False   True
C  True   False

Compare to a DataFrame of different shape.

>>> other = pd.DataFrame({'revenue': [300, 250, 100, 150]},
...                        index=['A', 'B', 'C', 'D'])
>>> other
   revenue
A      300
B      250
C      100
D      150

>>> df.gt(other)
   cost  revenue
A  False   False
B  False   False
C  False    True

```

```

    D False False
    Compare to a MultiIndex by level.

    >>> df_multindex = pd.DataFrame({'cost': [250, 150, 100, 150, 300, 220],
    ...                                'revenue': [100, 250, 300, 200, 175,
225]}, ...
    ...                                index=[['Q1', 'Q1', 'Q1', 'Q2', 'Q2',
'Q2'], ...
    ...                                ['A', 'B', 'C', 'A', 'B', 'C']])
    >>> df_multindex
      cost revenue
Q1 A   250     100
      B   150     250
      C   100     300
Q2 A   150     200
      B   300     175
      C   220     225

    >>> df.le(df_multindex, level=1)
      cost revenue
Q1 A   True  True
      B   True  True
      C   True  True
Q2 A  False  True
      B   True  False
      C   True  False

eval(self, expr: 'str', inplace: 'bool' = False, **kwargs)
Evaluate a string describing operations on DataFrame columns.

Operates on columns only, not specific rows or elements. This allows
`eval` to run arbitrary code, which can make you vulnerable to code
injection if you pass user input to this function.

Parameters
-----
expr : str
    The expression string to evaluate.
inplace : bool, default False
    If the expression contains an assignment, whether to perform the
    operation inplace and mutate the existing DataFrame. Otherwise,
    a new DataFrame is returned.
**kwargs
    See the documentation for :func:`eval` for complete details
    on the keyword arguments accepted by
    :meth:`~pandas.DataFrame.query`.

Returns
-----
ndarray, scalar, pandas object, or None
    The result of the evaluation or None if ``inplace=True``.

See Also
-----
DataFrame.query : Evaluates a boolean expression to query the columns
    of a frame.
DataFrame.assign : Can evaluate an expression or function to create new
    values for a column.
eval : Evaluate a Python expression as a string using various
    backends.

Notes
-----
For more details see the API documentation for :func:`~eval`.
For detailed examples see :ref:`enhancing performance with eval
<enhancingperf.eval>`.

Examples
-----
>>> df = pd.DataFrame({'A': range(1, 6), 'B': range(10, 0, -2)})
>>> df
   A   B
0  1  10
1  2   8
2  3   6
3  4   4
4  5   2
>>> df.eval('A + B')
0   11
1   10
2    9
3    8
4    7
dtype: int64

Assignment is allowed though by default the original DataFrame is not
modified.

>>> df.eval('C = A + B')
   A   B   C
0  1  10  11
1  2   8  10
2  3   6   9
3  4   4   8
4  5   2   7
>>> df
   A   B
0  1  10
1  2   8
2  3   6
3  4   4
4  5   2

Use ``inplace=True`` to modify the original DataFrame.

>>> df.eval('C = A + B', inplace=True)
>>> df
   A   B   C
0  1  10  11
1  2   8  10
2  3   6   9
3  4   4   8
4  5   2   7

Multiple columns can be assigned to using multi-line expressions:

>>> df.eval(
...     ...
...     ... C = A + B
...     ... D = A - B
...     ...
...     ...
...     )
   A   B   C   D
0  1  10  11 -9
1  2   8  10 -6
2  3   6   9 -3
3  4   4   8  0
4  5   2   7  3

| explode(self, column: 'IndexLabel', ignore_index: 'bool' = False) ->

```

```

'DataFrame'
    Transform each element of a list-like to a row, replicating index values.
    .. versionadded:: 0.25.0

Parameters
-----
column : IndexLabel
    Column(s) to explode.
    For multiple columns, specify a non-empty list with each element
    be str or tuple, and all specified columns their list-like data
    on same row of the frame must have matching length.

    .. versionadded:: 1.3.0
        Multi-column explode

ignore_index : bool, default False
    If True, the resulting index will be labeled 0, 1, ..., n - 1.

    .. versionadded:: 1.1.0

Returns
-----
DataFrame
    Exploded lists to rows of the subset columns;
    index will be duplicated for these rows.

Raises
-----
ValueError :
    * If columns of the frame are not unique.
    * If specified columns to explode is empty list.
    * If specified columns to explode have not matching count of
        elements rowwise in the frame.

See Also
-----
DataFrame.unstack : Pivot a level of the (necessarily hierarchical)
    index labels.
DataFrame.melt : Unpivot a DataFrame from wide format to long format.
Series.explode : Explode a DataFrame from list-like columns to long
format.

Notes
-----
This routine will explode list-likes including lists, tuples, sets,
Series, and np.ndarray. The result dtype of the subset rows will
be object. Scalars will be returned unchanged, and empty list-likes will
result in a np.nan for that row. In addition, the ordering of rows in the
output will be non-deterministic when exploding sets.

Reference :ref:`the user guide <reshaping.explode>` for more examples.

Examples
-----
>>> df = pd.DataFrame({'A': [[0, 1, 2], 'foo', [], [3, 4]],
...                      'B': [1, ...,
...                            'C': [['a', 'b', 'c'], np.nan, [], ['d', 'e']]})
>>> df
      A      B      C
0  [0, 1, 2]  1  [a, b, c]
1    foo     1       NaN
2      []     1      []
3  [3, 4]     1  [d, e]

Single-column explode.

>>> df.explode('A')
      A      B      C
0  0  1  [a, b, c]
0  1  1  [a, b, c]
0  2  1  [a, b, c]
1  foo  1       NaN
2  NaN  1      []
3  3  1  [d, e]
3  4  1  [d, e]

Multi-column explode.

>>> df.explode(list('AC'))
      A      B      C
0  0  1  a
0  1  1  b
0  2  1  c
1  foo  1  NaN
2  NaN  1  NaN
3  3  1  d
3  4  1  e

    ffill(self: 'DataFrame', axis: 'None | Axis' = None, inplace: 'bool' = False,
limit: 'None | int' = None, downcast=None) -> 'DataFrame | None'
    Synonym for :meth:`DataFrame.fillna` with ``method='ffill'``.

Returns
-----
Series/DataFrame or None
    Object with missing values filled or None if ``inplace=True``.

    fillna(self, value: 'object | ArrayLike | None' = None, method: 'FillnaOptions
| None' = None, axis: 'Axis | None' = None, inplace: 'bool' = False, limit=None,
downcast=None) -> 'DataFrame | None'
    Fill NA/NaN values using the specified method.

Parameters
-----
value : scalar, dict, Series, or DataFrame
    Value to use to fill holes (e.g. 0), alternately a
    dict/Series/DataFrame of values specifying which value to use for
    each index (for a Series) or column (for a DataFrame). Values not
    in the dict/Series/Dataframe will not be filled. This value cannot
    be a list.
method : {'backfill', 'bfill', 'pad', 'ffill', None}, default None
    Method to use for filling holes in reindexed Series
    pad / ffill: propagate last valid observation forward to next valid
    backfill / bfill: use next valid observation to fill gap.
axis : {0 or 'index', 1 or 'columns'}
    Axis along which to fill missing values.
inplace : bool, default False
    If True, fill in-place. Note: this will modify any
    other views on this object (e.g., a no-copy slice for a column in a
    DataFrame).
limit : int, default None
    If method is specified, this is the maximum number of consecutive
    NaN values to forward/backward fill. In other words, if there is
    a gap with more than this number of consecutive NaNs, it will only
    be partially filled. If method is not specified, this is the
    maximum number of entries along the entire axis where NaNs will be
    filled. Must be greater than 0 if not None.
downcast : dict, default is None
    A dict of item->dtype of what to downcast if possible,
    or the string 'infer' which will try to downcast to an appropriate

```

```

equal type (e.g. float64 to int64 if possible).

Returns
-----
DataFrame or None
    Object with missing values filled or None if ``inplace=True``.

See Also
-----
interpolate : Fill NaN values using interpolation.
reindex : Conform object to new index.
asfreq : Convert TimeSeries to specified frequency.

Examples
-----
>>> df = pd.DataFrame([[np.nan, 2, np.nan, 0],
...                      [3, 4, np.nan, 1],
...                      [np.nan, np.nan, np.nan, np.nan],
...                      [np.nan, 3, np.nan, 4]],
...                      columns=list("ABCD"))
>>> df
   A    B    C    D
0  NaN  2.0  NaN  0.0
1  3.0  4.0  NaN  1.0
2  NaN  NaN  NaN  NaN
3  NaN  3.0  NaN  4.0

Replace all NaN elements with 0s.

>>> df.fillna(0)
   A    B    C    D
0  0.0  2.0  0.0  0.0
1  3.0  4.0  0.0  1.0
2  0.0  0.0  0.0  0.0
3  0.0  3.0  0.0  4.0

We can also propagate non-null values forward or backward.

>>> df.fillna(method="ffill")
   A    B    C    D
0  NaN  2.0  NaN  0.0
1  3.0  4.0  NaN  1.0
2  3.0  4.0  NaN  1.0
3  3.0  3.0  NaN  4.0

Replace all NaN elements in column 'A', 'B', 'C', and 'D', with 0, 1,
2, and 3 respectively.

>>> values = {"A": 0, "B": 1, "C": 2, "D": 3}
>>> df.fillna(value=values)
   A    B    C    D
0  0.0  2.0  2.0  0.0
1  3.0  4.0  2.0  1.0
2  0.0  1.0  2.0  3.0
3  0.0  3.0  2.0  4.0

Only replace the first NaN element.

>>> df.fillna(value=values, limit=1)
   A    B    C    D
0  0.0  2.0  2.0  0.0
1  3.0  4.0  NaN  1.0
2  NaN  1.0  NaN  3.0
3  NaN  3.0  NaN  4.0

When filling using a DataFrame, replacement happens along
the same column names and same indices

>>> df2 = pd.DataFrame(np.zeros((4, 4)), columns=list("ABCDE"))
>>> df.fillna(df2)
   A    B    C    D
0  0.0  2.0  0.0  0.0
1  3.0  4.0  0.0  1.0
2  0.0  0.0  0.0  NaN
3  0.0  3.0  0.0  4.0

Note that column D is not affected since it is not present in df2.

floordiv(self, other, axis='columns', level=None, fill_value=None)
    Get Integer division of dataframe and other, element-wise (binary operator
'floordiv').

    Equivalent to ``dataframe // other``, but with support to substitute a
fill_value
    for missing data in one of the inputs. With reverse version, `rfloordiv`.

    Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful DataFrame alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                      index=['circle', 'triangle', 'rectangle'])
>>> df

```

```

        angles  degrees
circle      0      360
triangle    3      180
rectangle   4      360

Add a scalar with operator version which return the same
results.

>>> df + 1
        angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

>>> df.add(1)
        angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

Divide by constant with reverse version.

>>> df.div(10)
        angles  degrees
circle     0.0      36.0
triangle   0.3      18.0
rectangle  0.4      36.0

>>> df.rdiv(10)
        angles  degrees
circle     inf     0.027778
triangle  3.33333  0.055556
rectangle 2.50000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
        angles  degrees
circle     -1      358
triangle   2      178
rectangle  3      358

>>> df.sub([1, 2], axis='columns')
        angles  degrees
circle     -1      358
triangle   2      178
rectangle  3      358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
        axis='index')
        angles  degrees
circle     -1      359
triangle   2      179
rectangle  3      359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4]}, ...
...                           index=['circle', 'triangle', 'rectangle'])
>>> other
        angles
circle     0
triangle   3
rectangle  4

>>> df * other
        angles  degrees
circle     0      NaN
triangle   9      NaN
rectangle  16     NaN

>>> df.mul(other, fill_value=0)
        angles  degrees
circle     0      0.0
triangle   9      0.0
rectangle  16     0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
...                           index=[['A', 'A', 'A', 'B', 'B'],
...                                 ['circle', 'triangle', 'rectangle',
...                                  'square', 'pentagon', 'hexagon']])
>>> df_multindex
        angles  degrees
A circle     0      360
          triangle  3      180
          rectangle  4      360
B square     4      360
          pentagon   5      540
          hexagon    6      720

>>> df.div(df_multindex, level=1, fill_value=0)
        angles  degrees
A circle     NaN     1.0
          triangle  1.0     1.0
          rectangle  1.0     1.0
B square     0.0     0.0
          pentagon   0.0     0.0
          hexagon    0.0     0.0

ge(self, other, axis='columns', level=None)
Get Greater than or equal to of datafram and other, element-wise (binary
operator 'ge').

Among flexible wrappers ('eq', 'ne', 'le', 'lt', 'ge', 'gt') to comparison
operators.

Equivalent to '==' , '!=', '<=' , '<' , '>=' , '>' with support to choose axis
(rows or columns) and level for comparison.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}, default 'columns'
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns').
level : int or label
    Broadcast across a level, matching Index values on the passed
    MultiIndex level.

Returns
-----
Dataframe of bool
    Result of the comparison.

See Also
```

```

-----
| DataFrame.eq : Compare DataFrames for equality elementwise.
| DataFrame.ne : Compare DataFrames for inequality elementwise.
| DataFrame.le : Compare DataFrames for less than inequality
|   or equality elementwise.
| DataFrame.lt : Compare DataFrames for strictly less than
|   inequality elementwise.
| DataFrame.ge : Compare DataFrames for greater than inequality
|   or equality elementwise.
| DataFrame.gt : Compare DataFrames for strictly greater than
|   inequality elementwise.
|
| Notes
| -----
| Mismatched indices will be unioned together.
| 'NaN' values are considered different (i.e. `NaN` != `NaN`).
|
| Examples
| -----
|>>> df = pd.DataFrame({'cost': [250, 150, 100],
|...                      'revenue': [100, 250, 300]},
|...                     index=['A', 'B', 'C'])
|>>> df
|      cost    revenue
| A    250       100
| B    150       250
| C    100       300
|
| Comparison with a scalar, using either the operator or method:
|
|>>> df == 100
|      cost    revenue
| A    False     True
| B    False     False
| C    True      False
|
|>>> df.eq(100)
|      cost    revenue
| A    False     True
| B    False     False
| C    True      False
|
| When 'other' is a :class:`Series`, the columns of a DataFrame are aligned
| with the index of 'other' and broadcast:
|
|>>> df != pd.Series([100, 250], index=["cost", "revenue"])
|      cost    revenue
| A    True      True
| B    True      False
| C    False     True
|
| Use the method to control the broadcast axis:
|
|>>> df.ne(pd.Series([100, 300], index=["A", "D"]), axis='index')
|      cost    revenue
| A    True      False
| B    True      True
| C    True      True
| D    True      True
|
| When comparing to an arbitrary sequence, the number of columns must
| match the number elements in 'other':
|
|>>> df == [250, 100]
|      cost    revenue
| A    True      True
| B    False     False
| C    False     False
|
| Use the method to control the axis:
|
|>>> df.eq([250, 250, 100], axis='index')
|      cost    revenue
| A    True      False
| B    False     True
| C    True      False
|
| Compare to a DataFrame of different shape.
|
|>>> other = pd.DataFrame({'revenue': [300, 250, 100, 150]},
|...                      index=['A', 'B', 'C', 'D'])
|>>> other
|      revenue
| A      300
| B      250
| C      100
| D      150
|
|>>> df.gt(other)
|      cost    revenue
| A    False     False
| B    False     False
| C    True      True
| D    False     False
|
| Compare to a MultiIndex by level.
|
|>>> df_multindex = pd.DataFrame({'cost': [250, 150, 100, 150, 300, 220],
|...                                 'revenue': [100, 250, 300, 200, 175,
|225]}, ... index=[['Q1', 'Q1', 'Q1', 'Q2', 'Q2',
|'Q2'], ... [ 'A', 'B', 'C', 'A', 'B', 'C']])
|>>> df_multindex
|      cost    revenue
| Q1 A    250       100
|       B    150       250
|       C    100       300
| Q2 A    150       200
|       B    300       175
|       C    220       225
|
|>>> df.le(df_multindex, level=1)
|      cost    revenue
| Q1 A    True      True
|       B    True      True
|       C    True      True
| Q2 A    False     True
|       B    True      False
|       C    True      False
|
| groupby(self, by=None, axis: 'Axis' = 0, level: 'Level | None' = None,
| as_index: 'bool' = True, sort: 'bool' = True, group_keys: 'bool' = True, squeeze:
| 'bool' | lib.Nodefault = <no_default>, observed: 'bool' = False, dropna: 'bool' =
| True) -> 'DataFrameGroupBy'
|   Group DataFrame using a mapper or by a Series of columns.
|
| A groupby operation involves some combination of splitting the
| object, applying a function, and combining the results. This can be
| used to group large amounts of data and compute operations on these
| groups.

```

```

Parameters
-----
by : mapping, function, label, or list of labels
    Used to determine the groups for the groupby.
    If ``by`` is a function, it's called on each value of the object's
    index. If a dict or Series is passed, the Series or dict VALUES
    will be used to determine the groups (the Series' values are first
    aligned; see ``.align()`` method). If a list or ndarray of length
    equal to the selected axis is passed (see the `groupby user guide
    <https://pandas.pydata.org/pandas-docs/stable/user\_guide/groupby.html#splitting-an-object-into-groups>),
    the values are used as-is to determine the groups. A label or list
    of labels may be passed to group by the columns in ``self``.
    Notice that a tuple is interpreted as a (single) key.
axis : {0 or 'index', 1 or 'columns'}, default 0
    Split along rows (0) or columns (1).
level : int, level name, or sequence of such, default None
    If the axis is a MultiIndex (hierarchical), group by a particular
    level or levels.
as_index : bool, default True
    For aggregated output, return object with group labels as the
    index. Only relevant for DataFrame input. as_index=False is
    effectively "SQL-style" grouped output.
sort : bool, default True
    Sort group keys. Get better performance by turning this off.
    Note this does not influence the order of observations within each
    group. Groupby preserves the order of rows within each group.
group_keys : bool, default True
    When calling apply, add group keys to index to identify pieces.
squeeze : bool, default False
    Reduce the dimensionality of the return type if possible,
    otherwise return a consistent type.

.. deprecated:: 1.1.0

observed : bool, default False
    This only applies if any of the groupers are Categoricals.
    If True: only show observed values for categorical groupers.
    If False: show all values for categorical groupers.
dropna : bool, default True
    If True, and if group keys contain NA values, NA values together
    with row/column will be dropped.
    If False, NA values will also be treated as the key in groups.

.. versionadded:: 1.1.0

Returns
-----
DataFrameGroupBy
    Returns a groupby object that contains information about the groups.

See Also
-----
resample : Convenience method for frequency conversion and resampling
    of time series.

Notes
-----
See the `user guide
<https://pandas.pydata.org/pandas-docs/stable/groupby.html>`__ for more
detailed usage and examples, including splitting an object into groups,
iterating through groups, selecting a group, aggregation, and more.

Examples
-----
>>> df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
...                                'Parrot', 'Parrot'],
...                      'Max Speed': [380., 370., 24., 26.]})
>>> df
   Animal  Max Speed
0  Falcon      380.0
1  Falcon      370.0
2  Parrot       24.0
3  Parrot       26.0
>>> df.groupby(['Animal']).mean()
           Max Speed
Animal
Falcon      375.0
Parrot       25.0

**Hierarchical Indexes**

We can groupby different levels of a hierarchical index
using the `level` parameter:

>>> arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
...             ['Captive', 'Wild', 'Captive', 'Wild']]
>>> index = pd.MultiIndex.from_arrays(arrays, names=['Animal', 'Type'])
>>> df = pd.DataFrame({'Max Speed': [390., 350., 30., 20.]},
...                      index=index)
>>> df
           Max Speed
Animal Type
Falcon Captive      390.0
        Wild          350.0
Parrot Captive       30.0
        Wild          20.0
>>> df.groupby(level=0).mean()
           Max Speed
Animal
Falcon      370.0
Parrot       25.0
>>> df.groupby(level="Type").mean()
           Max Speed
Type
Captive      210.0
Wild         185.0

We can also choose to include NA in group keys or not by setting
`dropna` parameter, the default setting is `True`.

>>> l = [[1, 2, 3], [1, None, 4], [2, 1, 3], [1, 2, 2]]
>>> df = pd.DataFrame(l, columns=["a", "b", "c"])
>>> df.groupby(by=["b"]).sum()
      a   c
b
1.0  2   3
2.0  2   5
>>> df.groupby(by=["b"], dropna=False).sum()
      a   c
b
1.0  2   3
2.0  2   5
NaN  1   4
>>> l = [[{"a": 12, "b": 12}, {"a": 12.3, "b": 33}, {"a": 12.3, "b": 123}, {"a": 1, "b": 1}]

>>> df = pd.DataFrame(l, columns=["a", "b", "c"])

```

```

>>> df.groupby(by="a").sum()
      b      c
a  13.0  13.0
b  12.3  123.0

>>> df.groupby(by="a", dropna=False).sum()
      b      c
a  13.0  13.0
b  12.3  123.0
NaN 12.3  33.0

gt(self, other, axis='columns', level=None)
    Get Greater than of dataframe and other, element-wise (binary operator
'gt').

    Among flexible wrappers ('eq', 'ne', 'le', 'lt', 'ge', 'gt') to comparison
operators.

    Equivalent to '==', '!=', '<=' , '<' , '>=' , '>' with support to choose axis
(rows or columns) and level for comparison.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}, default 'columns'
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns').
level : int or label
    Broadcast across a level, matching Index values on the passed
    MultiIndex level.

Returns
-----
DataFrame of bool
    Result of the comparison.

See Also
-----
DataFrame.eq : Compare DataFrames for equality elementwise.
DataFrame.ne : Compare DataFrames for inequality elementwise.
DataFrame.le : Compare DataFrames for less than inequality
    or equality elementwise.
DataFrame.lt : Compare DataFrames for strictly less than
    inequality elementwise.
DataFrame.ge : Compare DataFrames for greater than inequality
    or equality elementwise.
DataFrame.gt : Compare DataFrames for strictly greater than
    inequality elementwise.

Notes
-----
Mismatched indices will be unioned together.
'NaN' values are considered different (i.e. `NaN` != `NaN`).

Examples
-----
>>> df = pd.DataFrame({'cost': [250, 150, 100],
...                      'revenue': [100, 250, 300]},
...                     index=['A', 'B', 'C'])
>>> df
   cost  revenue
A    250       100
B    150       250
C    100       300

Comparison with a scalar, using either the operator or method:

>>> df == 100
      cost  revenue
A  False    True
B  False   False
C  True   False

>>> df.eq(100)
      cost  revenue
A  False    True
B  False   False
C  True   False

When 'other' is a :class:`Series`, the columns of a DataFrame are aligned
with the index of 'other' and broadcast:

>>> df != pd.Series([100, 250], index=["cost", "revenue"])
      cost  revenue
A  True    True
B  True   False
C  False   True

Use the method to control the broadcast axis:

>>> df.ne(pd.Series([100, 300], index=["A", "D"]), axis='index')
      cost  revenue
A  True    False
B  True     True
C  True     True
D  True     True

When comparing to an arbitrary sequence, the number of columns must
match the number elements in 'other':

>>> df == [250, 100]
      cost  revenue
A  True    True
B  False   False
C  False   False

Use the method to control the axis:

>>> df.eq([250, 250, 100], axis='index')
      cost  revenue
A  True    False
B  False   True
C  True   False

Compare to a DataFrame of different shape.

>>> other = pd.DataFrame({'revenue': [300, 250, 100, 150]},
...                        index=['A', 'B', 'C', 'D'])
>>> other
      revenue
A        300
B        250
C        100
D        150

>>> df.gt(other)
      cost  revenue
A  False   False

```

```

B False False
C False True
D False False

Compare to a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'cost': [250, 150, 100, 150, 300, 220],
...                                'revenue': [100, 250, 300, 200, 175,
225]}, ...
...                                index=[['Q1', 'Q1', 'Q1', 'Q2', 'Q2',
'Q2'], ...
...                                ['A', 'B', 'C', 'A', 'B', 'C']])
>>> df_multindex
   cost revenue
Q1 A    250     100
      B    150     250
      C    100     300
Q2 A    150     200
      B    300     175
      C    220     225

>>> df.le(df_multindex, level=1)
   cost revenue
Q1 A    True  True
      B    True  True
      C    True  True
Q2 A   False  True
      B    True  False
      C    True  False

hist = hist_frame(data: 'DataFrame', column: 'IndexLabel' = None, by=None,
grid: 'bool' = True, xlabelsize: 'int | None' = None, xrot: 'float | None' = None,
ylabelsize: 'int | None' = None, yrot: 'float | None' = None, ax=None, sharex:
'bool' = False, sharey: 'bool' = False, figsize: 'tuple[int, int] | None' = None,
layout: 'tuple[int, int] | None' = None, bins: 'int | Sequence[int]' = 10,
backend: 'str | None' = None, legend: 'bool' = False, **kwargs)
| Make a histogram of the DataFrame's columns.

A `histogram`_ is a representation of the distribution of data.
This function calls :meth:`matplotlib.pyplot.hist`, on each series in
the Dataframe, resulting in one histogram per column.

.. _histogram: https://en.wikipedia.org/wiki/Histogram

Parameters
-----
data : DataFrame
    The pandas object holding the data.
column : str or sequence, optional
    If passed, will be used to limit data to a subset of columns.
by : object, optional
    If passed, then used to form histograms for separate groups.
grid : bool, default True
    Whether to show axis grid lines.
xlabelsize : int, default None
    If specified changes the x-axis label size.
xrot : float, default None
    Rotation of x axis labels. For example, a value of 90 displays the
    x labels rotated 90 degrees clockwise.
ylabelsize : int, default None
    If specified changes the y-axis label size.
yrot : float, default None
    Rotation of y axis labels. For example, a value of 90 displays the
    y labels rotated 90 degrees clockwise.
ax : Matplotlib axes object, default None
    The axes to plot the histogram on.
sharex : bool, default True if ax is None else False
    In case subplots=True, share x axis and set some x axis labels to
    invisible; defaults to True if ax is None otherwise False if an ax
    is passed in.
    Note that passing in both an ax and sharex=True will alter all x axis
    labels for all subplots in a figure.
sharey : bool, default False
    In case subplots=True, share y axis and set some y axis labels to
    invisible.
figsize : tuple, optional
    The size in inches of the figure to create. Uses the value in
    'matplotlib.rcParams' by default.
layout : tuple, optional
    Tuple of (rows, columns) for the layout of the histograms.
bins : int or sequence, default 10
    Number of histogram bins to be used. If an integer is given, bins + 1
    bin edges are calculated and returned. If bins is a sequence, gives
    bin edges, including left edge of first bin and right edge of last
    bin. In this case, bins is returned unmodified.
backend : str, default None
    Backend to use instead of the backend specified in the option
    ``plotting.backend``. For instance, 'matplotlib'. Alternatively, to
    specify the ``plotting.backend`` for the whole session, set
    ``pd.options.plotting.backend``.

.. versionadded:: 1.0.0

legend : bool, default False
    Whether to show the legend.

.. versionadded:: 1.1.0

**kwargs
    All other plotting keyword arguments to be passed to
    :meth:`matplotlib.pyplot.hist`.

Returns
-----
matplotlib.AxesSubplot or numpy.ndarray of them

See Also
-----
matplotlib.pyplot.hist : Plot a histogram using matplotlib.

Examples
-----
This example draws a histogram based on the length and width of
some animals, displayed in three bins

.. plot::
    :context: close-figs

    >>> df = pd.DataFrame({
...     'length': [1.5, 0.5, 1.2, 0.9, 3],
...     'width': [0.7, 0.2, 0.15, 0.2, 1.1]
... }, index=['pig', 'rabbit', 'duck', 'chicken', 'horse'])
    >>> hist = df.hist(bins=3)

idxmax(self, axis: 'Axis' = 0, skipna: 'bool' = True) -> 'Series'
Return index of first occurrence of maximum over requested axis.

NA/null values are excluded.

Parameters
-----
```

```

    -----
    axis : {0 or 'index', 1 or 'columns'}, default 0
        The axis to use. 0 or 'index' for row-wise, 1 or 'columns' for column-
wise.
    skipna : bool, default True
        Exclude NA/null values. If an entire row/column is NA, the result
        will be NA.

    Returns
    -----
    Series
        Indexes of maxima along the specified axis.

    Raises
    -----
    ValueError
        * If the row/column is empty

    See Also
    -----
    Series.idxmax : Return index of the maximum element.

    Notes
    -----
    This method is the DataFrame version of ``ndarray.argmax``.

    Examples
    -----
    Consider a dataset containing food consumption in Argentina.

    >>> df = pd.DataFrame({'consumption': [10.51, 103.11, 55.48],
...                      'co2_emissions': [37.2, 19.66, 1712]},
...                      index=['Pork', 'Wheat Products', 'Beef'])

    >>> df
           consumption  co2_emissions
    Pork            10.51          37.20
    Wheat Products   103.11         19.66
    Beef             55.48         1712.00

    By default, it returns the index for the maximum value in each column.

    >>> df.idxmax()
    consumption      Wheat Products
    co2_emissions      Beef
    dtype: object

    To return the index for the maximum value in each row, use
    'axis="columns"'.

    >>> df.idxmax(axis="columns")
    Pork            co2_emissions
    Wheat Products      consumption
    Beef             co2_emissions
    dtype: object

    idxmin(self, axis: 'Axis' = 0, skipna: 'bool' = True) -> 'Series'
    Return index of first occurrence of minimum over requested axis.

    NA/null values are excluded.

    Parameters
    -----
    axis : {0 or 'index', 1 or 'columns'}, default 0
        The axis to use. 0 or 'index' for row-wise, 1 or 'columns' for column-
wise.
    skipna : bool, default True
        Exclude NA/null values. If an entire row/column is NA, the result
        will be NA.

    Returns
    -----
    Series
        Indexes of minima along the specified axis.

    Raises
    -----
    ValueError
        * If the row/column is empty

    See Also
    -----
    Series.idxmin : Return index of the minimum element.

    Notes
    -----
    This method is the DataFrame version of ``ndarray.argmin``.

    Examples
    -----
    Consider a dataset containing food consumption in Argentina.

    >>> df = pd.DataFrame({'consumption': [10.51, 103.11, 55.48],
...                      'co2_emissions': [37.2, 19.66, 1712]},
...                      index=['Pork', 'Wheat Products', 'Beef'])

    >>> df
           consumption  co2_emissions
    Pork            10.51          37.20
    Wheat Products   103.11         19.66
    Beef             55.48         1712.00

    By default, it returns the index for the minimum value in each column.

    >>> df.idxmin()
    consumption      Pork
    co2_emissions     Wheat Products
    dtype: object

    To return the index for the minimum value in each row, use
    'axis="columns"'.

    >>> df.idxmin(axis="columns")
    Pork            consumption
    Wheat Products      co2_emissions
    Beef             consumption
    dtype: object

    info(self, verbose: 'bool | None' = None, buf: 'WriteBuffer[str] | None' =
None, max_cols: 'int | None' = None, memory_usage: 'bool | str | None' = None,
show_counts: 'bool | None' = None, null_counts: 'bool | None' = None) -> 'None'
    Print a concise summary of a DataFrame.

    This method prints information about a DataFrame including
    the index dtype and columns, non-null values and memory usage.

    Parameters
    -----
    data : DataFrame
        Dataframe to print information about.
    verbose : bool, optional

```

```

    Whether to print the full summary. By default, the setting in
    ``pandas.options.display.max_info_columns`` is followed.
buf : writable buffer, defaults to sys.stdout
    Where to send the output. By default, the output is printed to
    sys.stdout. Pass a writable buffer if you need to further process
    the output. max_cols : int, optional
    When to switch from the verbose to the truncated output. If the
    DataFrame has more than `max_cols` columns, the truncated output
    is used. By default, the setting in
    ``pandas.options.display.max_info_columns`` is used.
memory_usage : bool, str, optional
    Specifies whether total memory usage of the DataFrame
    elements (including the index) should be displayed. By default,
    this follows the ``pandas.options.display.memory_usage`` setting.

    True always show memory usage. False never shows memory usage.
    A value of 'deep' is equivalent to "True with deep introspection".
    Memory usage is shown in human-readable units (base-2
    representation). Without deep introspection a memory estimation is
    made based in column dtype and number of rows assuming values
    consume the same memory amount for corresponding dtypes. With deep
    memory introspection, a real memory usage calculation is performed
    at the cost of computational resources.

show_counts : bool, optional
    Whether to show the non-null counts. By default, this is shown
    only if the DataFrame is smaller than
    ``pandas.options.display.max_info_rows`` and
    ``pandas.options.display.max_info_columns``. A value of True always
    shows the counts, and False never shows the counts.

null_counts : bool, optional
    .. deprecated:: 1.2.0
        Use show_counts instead.

Returns
-----
None
    This method prints a summary of a DataFrame and returns None.

See Also
-----
DataFrame.describe: Generate descriptive statistics of DataFrame
    columns.
DataFrame.memory_usage: Memory usage of DataFrame columns.

Examples
-----
>>> int_values = [1, 2, 3, 4, 5]
>>> text_values = ['alpha', 'beta', 'gamma', 'delta', 'epsilon']
>>> float_values = [0.0, 0.25, 0.5, 0.75, 1.0]
>>> df = pd.DataFrame({"int_col": int_values, "text_col": text_values,
...                     "float_col": float_values})
>>> df
   int_col text_col  float_col
0         1    alpha     0.00
1         2    beta      0.25
2         3   gamma     0.50
3         4   delta     0.75
4         5  epsilon     1.00

Prints information of all columns:

>>> df.info(verbose=True)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype  
---  -- 
 0   int_col   5 non-null      int64  
 1   text_col  5 non-null      object  
 2   float_col 5 non-null      float64 
dtypes: float64(1), int64(1), object(1)
memory usage: 248.0+ bytes

Prints a summary of columns count and its dtypes but not per column
information:

>>> df.info(verbose=False)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Columns: 3 entries, int_col to float_col
dtypes: float64(1), int64(1), object(1)
memory usage: 248.0+ bytes

Pipe output of DataFrame.info to buffer instead of sys.stdout, get
buffer content and writes to a text file:

>>> import io
>>> buffer = io.StringIO()
>>> df.info(buf=buffer)
>>> s = buffer.getvalue()
>>> with open("df_info.txt", "w",
...             encoding="utf-8") as f: # doctest: +SKIP
...     f.write(s)
260

The 'memory_usage' parameter allows deep introspection mode, specially
useful for big DataFrames and fine-tune memory optimization:

>>> random_strings_array = np.random.choice(['a', 'b', 'c'], 10 ** 6)
>>> df = pd.DataFrame({
...     'column_1': np.random.choice(['a', 'b', 'c'], 10 ** 6),
...     'column_2': np.random.choice(['a', 'b', 'c'], 10 ** 6),
...     'column_3': np.random.choice(['a', 'b', 'c'], 10 ** 6)
... })
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype  
---  -- 
 0   column_1  1000000 non-null object  
 1   column_2  1000000 non-null object  
 2   column_3  1000000 non-null object  
dtypes: object(3)
memory usage: 22.9+ MB

>>> df.info(memory_usage='deep')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype  
---  -- 
 0   column_1  1000000 non-null object  
 1   column_2  1000000 non-null object  
 2   column_3  1000000 non-null object  
dtypes: object(3)
memory usage: 165.9 MB

insert(self, loc: 'int', column: 'Hashable', value: 'Scalar | AnyArrayLike',
allow_duplicates: 'bool' = False) -> 'None'
|     Insert column into DataFrame at specified location.

```

```

Raises a ValueError if `column` is already contained in the DataFrame,
unless `allow_duplicates` is set to True.

Parameters
-----
loc : int
    Insertion index. Must verify  $0 \leq loc \leq \text{len(columns)}$ .
column : str, number, or hashable object
    Label of the inserted column.
value : Scalar, Series, or array-like
allow_duplicates : bool, optional default False

See Also
-----
Index.insert : Insert new item by index.

Examples
-----
>>> df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
>>> df
   col1  col2
0      1      3
1      2      4
>>> df.insert(1, "newcol", [99, 99])
>>> df
   col1  newcol  col2
0      1      99      3
1      2      99      4
>>> df.insert(0, "col1", [100, 100], allow_duplicates=True)
>>> df
   col1  col1  newcol  col2
0  100      1      99      3
1  100      2      99      4

Notice that pandas uses index alignment in case of `value` from type
Series':
>>> df.insert(0, "col0", pd.Series([5, 6], index=[1, 2]))
>>> df
   col0  col1  col1  newcol  col2
0  5.0    100      1      99      3
1  5.0    100      2      99      4

[ interpolate(self: 'DataFrame', method: 'str' = 'linear', axis: 'Axis' = 0,
limit: 'int | None' = None, inplace: 'bool' = False, limit_direction: 'str | None'
= None, limit_area: 'str | None' = None, downcast: 'str | None' = None, **kwargs)
-> 'DataFrame | None'
    Fill NaN values using an interpolation method.

Please note that only ``method='linear'`` is supported for
DataFrame/Series with a MultiIndex.

Parameters
-----
method : str, default 'linear'
    Interpolation technique to use. One of:
        * 'linear': Ignore the index and treat the values as equally
            spaced. This is the only method supported on MultiIndexes.
        * 'time': Works on daily and higher resolution data to interpolate
            given length of interval.
        * 'index', 'values': use the actual numerical values of the index.
        * 'pad': Fill in NaNs using existing values.
        * 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'spline',
            'barcentric', 'polynomial'. Passed to
            `scipy.interpolate.interp1d`. These methods use the numerical
            values of the index. Both 'polynomial' and 'spline' require that
            you also specify an 'order' (int), e.g.
            `df.interpolate(method='polynomial', order=5)`.
        * 'krogh', 'piecewise_polynomial', 'spline', 'chip', 'akima',
            'cubicpline': Wrappers around the SciPy interpolation methods of
            similar names. See 'Notes'.
        * 'from_derivatives': Refers to
            `scipy.interpolate.BPoly.from_derivatives` which
            replaces 'piecewise_polynomial' interpolation method in
            scipy 0.18.
axis : {0 or 'index', 1 or 'columns', None}, default None
    Axis to interpolate along.
limit : int, optional
    Maximum number of consecutive NaNs to fill. Must be greater than
    0.
inplace : bool, default False
    Update the data in place if possible.
limit_direction : {'forward', 'backward', 'both'}, Optional
    Consecutive NaNs will be filled in this direction.

    If limit is specified:
        * If 'method' is 'pad' or 'ffill', 'limit_direction' must be
        'forward'.
        * If 'method' is 'backfill' or 'bfill', 'limit_direction' must be
        'backwards'.

    If 'limit' is not specified:
        * If 'method' is 'backfill' or 'bfill', the default is 'backward'
        * else the default is 'forward'

    .. versionchanged:: 1.1.0
        raises ValueError if 'limit_direction' is 'forward' or 'both' and
        method is 'backfill' or 'bfill'.
        raises ValueError if 'limit_direction' is 'backward' or 'both' and
        method is 'pad' or 'ffill'.

limit_area : {'None', 'inside', 'outside'}, default None
    If limit is specified, consecutive NaNs will be filled with this
    restriction.
        * 'None': No fill restriction.
        * 'inside': Only fill NaNs surrounded by valid values
            (interpolate).
        * 'outside': Only fill NaNs outside valid values (extrapolate).

downcast : optional, 'infer' or None, defaults to None
    Downcast dtypes if possible.
**kwargs : optional
    Keyword arguments to pass on to the interpolating function.

Returns
-----
Series or DataFrame or None
    Returns the same object type as the caller, interpolated at
    some or all ``NaN`` values or None if ``inplace=True``.

See Also
-----
fillna : Fill missing values using different methods.
scipy.interpolate.Akima1DInterpolator : Piecewise cubic polynomials
    (Akima interpolator).
scipy.interpolate.BPoly.from_derivatives : Piecewise polynomial in the
    Bernstein basis.

```

```

scipy.interpolate.interpid : Interpolate a 1-D function.
scipy.interpolate.KroghInterpolator : Interpolate polynomial (Krogh
    interpolator).
scipy.interpolate.PchipInterpolator : PCHIP 1-d monotonic cubic
    interpolation.
scipy.interpolate.CubicSpline : Cubic spline data interpolator.

Notes
-----
The 'krogh', 'piecewise_polynomial', 'spline', 'pchip' and 'akima'
methods are wrappers around the respective SciPy implementations of
similar names. These use the actual numerical values of the index.
For more information on their behavior, see the
`SciPy documentation
<https://docs.scipy.org/doc/scipy/reference/interpolate.html#univariate-interpolation>`_
and `SciPy tutorial
<https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html>`_.

Examples
-----
Filling in ``NaN`` in a :class:`~pandas.Series` via linear
interpolation.

>>> s = pd.Series([0, 1, np.nan, 3])
>>> s
0    0.0
1    1.0
2    NaN
3    3.0
dtype: float64
>>> s.interpolate()
0    0.0
1    1.0
2    2.0
3    3.0
dtype: float64

Filling in ``NaN`` in a Series by padding, but filling at most two
consecutive ``NaN`` at a time.

>>> s = pd.Series([np.nan, "single_one", np.nan,
...                 "fill_two_more", np.nan, np.nan, np.nan,
...                 4.71, np.nan])
>>> s
0      NaN
1  single_one
2      NaN
3  fill_two_more
4      NaN
5      NaN
6      NaN
7      4.71
8      NaN
dtype: object
>>> s.interpolate(method='pad', limit=2)
0      NaN
1  single_one
2  single_one
3  fill_two_more
4  fill_two_more
5  fill_two_more
6      NaN
7      4.71
8      4.71
dtype: object

Filling in ``NaN`` in a Series via polynomial interpolation or splines:
Both 'polynomial' and 'spline' methods require that you also specify
an ``order`` (int).

>>> s = pd.Series([0, 2, np.nan, 8])
>>> s.interpolate(method='polynomial', order=2)
0    0.000000
1    2.000000
2    4.666667
3    8.000000
dtype: float64

Fill the DataFrame forward (that is, going down) along each column
using linear interpolation.

Note how the last entry in column 'a' is interpolated differently,
because there is no entry after it to use for interpolation.
Note how the first entry in column 'b' remains ``NaN``, because there
is no entry before it to use for interpolation.

>>> df = pd.DataFrame([(0.0, np.nan, -1.0, 1.0),
...                      (np.nan, 2.0, np.nan, np.nan),
...                      (2.0, 3.0, np.nan, 9.0),
...                      (np.nan, 4.0, -4.0, 16.0)],
...                      columns=list('abcd'))
>>> df
   a    b    c    d
0 0.0  NaN -1.0  1.0
1 NaN  2.0  NaN  NaN
2 2.0  3.0  NaN  9.0
3 NaN  4.0 -4.0 16.0
>>> df.interpolate(method='linear', limit_direction='forward', axis=0)
   a    b    c    d
0 0.0  NaN -1.0  1.0
1 1.0  2.0 -2.0  5.0
2 2.0  3.0 -3.0  9.0
3 2.0  4.0 -4.0 16.0

Using polynomial interpolation.

>>> df['d'].interpolate(method='polynomial', order=2)
0    1.0
1    4.0
2    9.0
3   16.0
Name: d, dtype: float64

isin(self, values) -> 'DataFrame'
Whether each element in the DataFrame is contained in values.

Parameters
-----
values : iterable, Series, DataFrame or dict
    The result will only be true at a location if all the
    labels match. If 'values' is a Series, that's the index. If
    'values' is a dict, the keys must be the column names,
    which must match. If 'values' is a DataFrame,
    then both the index and column labels must match.

Returns
-----
DataFrame
    Dataframe of booleans showing whether each element in the DataFrame
    is contained in values.

```

See Also

DataFrame.eq: Equality test for DataFrame.
Series.isin: Equivalent method on Series.
Series.str.contains: Test if pattern or regex is contained within a string of a Series or Index.

Examples

`>>> df = pd.DataFrame({'num_legs': [2, 4], 'num_wings': [2, 0]},
... index=['falcon', 'dog'])
>>> df`

	num_legs	num_wings
falcon	2	2
dog	4	0

When ``values`` is a list check whether every value in the DataFrame is present in the list (which animals have 0 or 2 legs or wings)

`>>> df.isin([0, 2])`

	num_legs	num_wings
falcon	True	True
dog	False	True

To check if ``values`` is *not* in the DataFrame, use the ``~`` operator:

`>>> ~df.isin([0, 2])`

	num_legs	num_wings
falcon	False	False
dog	True	False

When ``values`` is a dict, we can pass values to check for each column separately:

`>>> df.isin({'num_wings': [0, 3]})`

	num_legs	num_wings
falcon	False	False
dog	False	True

When ``values`` is a Series or DataFrame the index and column must match. Note that 'falcon' does not match based on the number of legs in other.

`>>> other = pd.DataFrame({'num_legs': [8, 3], 'num_wings': [0, 2]},
... index=['spider', 'falcon'])
>>> df.isin(other)`

	num_legs	num_wings
falcon	False	True
dog	False	False

`isna(self) -> 'DataFrame'`
Detect missing values.

Return a boolean same-sized object indicating if the values are NA. NA values, such as None or :attr:`numpy.NaN`, gets mapped to True values. Everything else gets mapped to False values. Characters such as empty strings ``''`` or :attr:`numpy.inf` are not considered NA values (unless you set ``pandas.options.mode.use_inf_as_na = True``).

Returns

DataFrame
Mask of bool values for each element in DataFrame that indicates whether an element is an NA value.

See Also

DataFrame.isnull : Alias of isna.
DataFrame.notna : Boolean inverse of isna.
DataFrame.dropna : Omit axes labels with missing values.
isna : Top-level isna.

Examples

Show which entries in a DataFrame are NA.

`>>> df = pd.DataFrame(dict(age=[5, 6, np.NaN],
... born=[pd.NaT, pd.Timestamp('1939-05-27'),
... pd.Timestamp('1940-04-25')],
... name=['Alfred', 'Batman', ''],
... toy=[None, 'Batmobile', 'Joker']))
>>> df`

	age	born	name	toy
0	5.0	NaT	Alfred	None
1	6.0	1939-05-27	Batman	Batmobile
2	NaN	1940-04-25		Joker

`>>> df.isna()`

	age	born	name	toy
0	False	True	False	True
1	False	False	False	False
2	True	False	False	False

Show which entries in a Series are NA.

`>>> ser = pd.Series([5, 6, np.NaN])
>>> ser`

	0	1	2
0	5.0		
1	6.0		
2	NaN		

`dtype: float64`

`>>> ser.isna()`

	0	1	2
0	False		
1	False		
2	True		

`dtype: bool`

`isnull(self) -> 'DataFrame'`
DataFrame.isnull is an alias for DataFrame.isna.

Detect missing values.

Return a boolean same-sized object indicating if the values are NA. NA values, such as None or :attr:`numpy.NaN`, gets mapped to True values. Everything else gets mapped to False values. Characters such as empty strings ``''`` or :attr:`numpy.inf` are not considered NA values (unless you set ``pandas.options.mode.use_inf_as_na = True``).

Returns

DataFrame
Mask of bool values for each element in DataFrame that indicates whether an element is an NA value.

See Also

DataFrame.isnull : Alias of isna.

```

DataFrame.notna : Boolean inverse of isna.
DataFrame.dropna : Omit axes labels with missing values.
isna : Top-level isna.

Examples
-----
Show which entries in a DataFrame are NA.

>>> df = pd.DataFrame(dict(age=[5, 6, np.NaN],
...                           born=[pd.NaT, pd.Timestamp('1939-05-27'),
...                                 pd.Timestamp('1940-04-25')],
...                           name=['Alfred', 'Batman', ''],
...                           toy=[None, 'Batmobile', 'Joker']))
>>> df
   age      born     name     toy
0  5.0      NaT  Alfred    None
1  6.0  1939-05-27  Batman  Batmobile
2  NaN  1940-04-25      Joker

>>> df.isna()
   age      born     name     toy
0  False   True  False  True
1  False  False  False  False
2  True  False  False  False

Show which entries in a Series are NA.

>>> ser = pd.Series([5, 6, np.NaN])
>>> ser
0    5.0
1    6.0
2    NaN
dtype: float64

>>> ser.isna()
0  False
1  False
2  True
dtype: bool

items(self) -> 'Iterable[tuple[Hashable, Series]]'
Iterate over (column name, Series) pairs.

Iterates over the DataFrame columns, returning a tuple with
the column name and the content as a Series.

Yields
-----
label : object
    The column names for the DataFrame being iterated over.
content : Series
    The column entries belonging to each label, as a Series.

See Also
-----
DataFrame.iterrows : Iterate over DataFrame rows as
    (index, Series) pairs.
DataFrame.itertuples : Iterate over DataFrame rows as namedtuples
    of the values.

Examples
-----
>>> df = pd.DataFrame({'species': ['bear', 'bear', 'marsupial'],
...                      'population': [1864, 22000, 80000]},
...                      index=['panda', 'polar', 'koala'])
>>> df
   species  population
panda     bear        1864
polar     bear       22000
koala  marsupial     80000
>>> for label, content in df.items():
...     print(f'label: {label}')
...     print(f'content: {content}', sep='\n')
...
label: species
content:
panda      bear
polar      bear
koala  marsupial
Name: species, dtype: object
label: population
content:
panda      1864
polar      22000
koala     80000
Name: population, dtype: int64

iteritems(self) -> 'Iterable[tuple[Hashable, Series]]'
Iterate over (column name, Series) pairs.

Iterates over the DataFrame columns, returning a tuple with
the column name and the content as a Series.

Yields
-----
label : object
    The column names for the DataFrame being iterated over.
content : Series
    The column entries belonging to each label, as a Series.

See Also
-----
DataFrame.iterrows : Iterate over DataFrame rows as
    (index, Series) pairs.
DataFrame.itertuples : Iterate over DataFrame rows as namedtuples
    of the values.

Examples
-----
>>> df = pd.DataFrame({'species': ['bear', 'bear', 'marsupial'],
...                      'population': [1864, 22000, 80000]},
...                      index=['panda', 'polar', 'koala'])
>>> df
   species  population
panda     bear        1864
polar     bear       22000
koala  marsupial     80000
>>> for label, content in df.items():
...     print(f'label: {label}')
...     print(f'content: {content}', sep='\n')
...
label: species
content:
panda      bear
polar      bear
koala  marsupial
Name: species, dtype: object
label: population
content:
panda      1864
polar      22000

```

```

koala    80000
Name: population, dtype: int64
iterrows(self) -> 'Iterable[tuple[Hashable, Series]]'
    Iterate over DataFrame rows as (index, Series) pairs.

Yields
-----
index : label or tuple of label
    The index of the row. A tuple for a `MultiIndex`.
data : Series
    The data of the row as a Series.

See Also
-----
DataFrame.itertuples : Iterate over DataFrame rows as namedtuples of the
values.
DataFrame.items : Iterate over (column name, Series) pairs.

Notes
-----
1. Because ``iterrows`` returns a Series for each row,
   it does **not** preserve dtypes across the rows (dtypes are
   preserved across columns for DataFrames). For example,
>>> df = pd.DataFrame([[1, 1.5], columns=['int', 'float']])
>>> row = next(df.iterrows())[1]
>>> row
int    1.0
float  1.5
Name: 0, dtype: float64
>>> print(row['int'].dtype)
float64
>>> print(df['int'].dtype)
int64

To preserve dtypes while iterating over the rows, it is better
to use :meth:`itertuples` which returns namedtuples of the values
and which is generally faster than ``iterrows``.

2. You should **never modify** something you are iterating over.
   This is not guaranteed to work in all cases. Depending on the
   data types, the iterator returns a copy and not a view, and writing
   to it will have no effect.

itertuples(self, index: 'bool' = True, name: 'str | None' = 'Pandas') ->
'Iterable[tuple[Any, ...]]'
    Iterate over DataFrame rows as namedtuples.

Parameters
-----
index : bool, default True
    If True, return the index as the first element of the tuple.
name : str or None, default "Pandas"
    The name of the returned namedtuples or None to return regular
    tuples.

Returns
-----
iterator
    An object to iterate over namedtuples for each row in the
    DataFrame with the first field possibly being the index and
    following fields being the column values.

See Also
-----
DataFrame.iterrows : Iterate over DataFrame rows as (index, Series)
    pairs.
DataFrame.items : Iterate over (column name, Series) pairs.

Notes
-----
The column names will be renamed to positional names if they are
invalid Python identifiers, repeated, or start with an underscore.
On python versions < 3.7 regular tuples are returned for DataFrames
with a large number of columns (>254).

Examples
-----
>>> df = pd.DataFrame({'num_legs': [4, 2], 'num_wings': [0, 2],
...                      index=['dog', 'hawk'])
>>> df
   num_legs  num_wings
dog         4          0
hawk        2          2
>>> for row in df.itertuples():
...     print(row)
...
Pandas(Index='dog', num_legs=4, num_wings=0)
Pandas(Index='hawk', num_legs=2, num_wings=2)

By setting the `index` parameter to False we can remove the index
as the first element of the tuple:
>>> for row in df.itertuples(index=False):
...     print(row)
...
Pandas(num_legs=4, num_wings=0)
Pandas(num_legs=2, num_wings=2)

With the `name` parameter set we set a custom name for the yielded
namedtuples:
>>> for row in df.itertuples(name='Animal'):
...     print(row)
...
Animal(Index='dog', num_legs=4, num_wings=0)
Animal(Index='hawk', num_legs=2, num_wings=2)

join(self, other: 'DataFrame | Series', on: 'IndexLabel | None' = None, how:
'str' = 'left', lsuffix: 'str' = '', rsuffix: 'str' = '', sort: 'bool' = False) ->
'DataFrame'
    Join columns of another DataFrame.

Join columns with 'other' DataFrame either on index or on a key
column. Efficiently join multiple DataFrame objects by index at once by
passing a list.

Parameters
-----
other : DataFrame, Series, or list of DataFrame
    Index should be similar to one of the columns in this one. If a
    Series is passed, its name attribute must be set, and that will be
    used as the column name in the resulting joined DataFrame.
on : str, list of str, or array-like, optional
    Column or index level name(s) in the caller to join on the index
    in 'other', otherwise joins index-on-index. If multiple
    values given, the 'other' DataFrame must have a MultiIndex. Can
    pass an array as the join key if it is not already contained in
    the calling DataFrame. Like an Excel VLOOKUP operation.
how : {'left', 'right', 'outer', 'inner'}, default 'left'

```

```

How to handle the operation of the two objects.

* left: use calling frame's index (or column if on is specified)
* right: use `other`'s index.
* outer: form union of calling frame's index (or column if on is
specified) with `other`'s index, and sort it.
lexicographically.
* inner: form intersection of calling frame's index (or column if
on is specified) with `other`'s index, preserving the order
of the calling's one.
* cross: creates the cartesian product from both frames, preserves the
order
of the left keys.

.. versionadded:: 1.2.0

lsuffix : str, default ''
    Suffix to use from left frame's overlapping columns.
rsuffix : str, default ''
    Suffix to use from right frame's overlapping columns.
sort : bool, default False
    Order result DataFrame lexicographically by the join key. If False,
    the order of the join key depends on the join type (how keyword).

Returns
-----
DataFrame
    A dataframe containing columns from both the caller and `other`.

See Also
-----
DataFrame.merge : For column(s)-on-column(s) operations.

Notes
-----
Parameters `on`, `lsuffix`, and `rsuffix` are not supported when
passing a list of `DataFrame` objects.

Support for specifying index levels as the `on` parameter was added
in version 0.23.0.

Examples
-----
>>> df = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'],
...                      'A': ['A0', 'A1', 'A2', 'A3', 'A4', 'A5']})

>>> df
   key    A
0   K0   A0
1   K1   A1
2   K2   A2
3   K3   A3
4   K4   A4
5   K5   A5

>>> other = pd.DataFrame({'key': ['K0', 'K1', 'K2'],
...                         'B': ['B0', 'B1', 'B2']})

>>> other
   key    B
0   K0   B0
1   K1   B1
2   K2   B2

Join DataFrames using their indexes.

>>> df.join(other, lsuffix='_caller', rsuffix='_other')
   key_caller  A  key_other  B
0           K0  A0       K0  B0
1           K1  A1       K1  B1
2           K2  A2       K2  B2
3           K3  A3      NaN  B2
4           K4  A4      NaN  NaN
5           K5  A5      NaN  NaN

If we want to join using the key columns, we need to set key to be
the index in both `df` and `other`. The joined DataFrame will have
key as its index.

>>> df.set_index('key').join(other.set_index('key'))
   A  B
key
K0  A0  B0
K1  A1  B1
K2  A2  B2
K3  A3  NaN
K4  A4  NaN
K5  A5  NaN

Another option to join using the key columns is to use the `on`
parameter. DataFrame.join always uses `other`'s index but we can use
any column in `df`. This method preserves the original DataFrame's
index in the result.

>>> df.join(other.set_index('key'), on='key')
   key    A  B
0   K0   A0  B0
1   K1   A1  B1
2   K2   A2  B2
3   K3   A3  NaN
4   K4   A4  NaN
5   K5   A5  NaN

Using non-unique key values shows how they are matched.

>>> df = pd.DataFrame({'key': ['K0', 'K1', 'K1', 'K3', 'K0', 'K1'],
...                      'A': ['A0', 'A1', 'A2', 'A3', 'A4', 'A5']})

>>> df
   key    A
0   K0   A0
1   K1   A1
2   K1   A2
3   K3   A3
4   K0   A4
5   K1   A5

>>> df.join(other.set_index('key'), on='key')
   key    A  B
0   K0   A0  B0
1   K1   A1  B1
2   K1   A2  B1
3   K3   A3  NaN
4   K0   A4  B0
5   K1   A5  B1

kurt(self, axis: 'Axis | None | lib.NoDefault' = <no_default>, skipna=True,
level=None, numeric_only=None, **kwargs)
    Return unbiased kurtosis over requested axis.

Kurtosis obtained using Fisher's definition of

```

```

kurtosis (kurtosis of normal == 0.0). Normalized by N-1.

Parameters
-----
axis : {index (0), columns (1)}
    Axis for the function to be applied on.
skipna : bool, default True
    Exclude NA/null values when computing the result.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.
**kwargs
    Additional keyword arguments to be passed to the function.

Returns
-----
Series or DataFrame (if level specified)

kurtosis = kurt(self, axis: 'Axis | None | lib.NoDefault' = <no_default>,
skipna=True, level=None, numeric_only=None, **kwargs)

le(self, other, axis='columns', level=None)
    Get Less than or equal to of dataframe and other, element-wise (binary
operator 'le').

    Among flexible wrappers ('eq', 'ne', 'le', 'lt', 'ge', 'gt') to comparison
operators.

    Equivalent to '==', '!=', '<=' , '<', '>=' , '>' with support to choose axis
(rows or columns) and level for comparison.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}, default 'columns'
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns').
level : int or label
    Broadcast across a level, matching Index values on the passed
    MultiIndex level.

Returns
-----
DataFrame of bool
    Result of the comparison.

See Also
-----
DataFrame.eq : Compare DataFrames for equality elementwise.
DataFrame.ne : Compare DataFrames for inequality elementwise.
DataFrame.le : Compare DataFrames for less than inequality
    or equality elementwise.
DataFrame.lt : Compare DataFrames for strictly less than
    inequality elementwise.
DataFrame.ge : Compare DataFrames for greater than inequality
    or equality elementwise.
DataFrame.gt : Compare DataFrames for strictly greater than
    inequality elementwise.

Notes
-----
Mismatched indices will be unioned together.
`NaN` values are considered different (i.e. `NaN` != `NaN`).

Examples
-----
>>> df = pd.DataFrame({'cost': [250, 150, 100],
...                      'revenue': [100, 250, 300]},
...                     index=['A', 'B', 'C'])
>>> df
   cost  revenue
A    250      100
B    150      250
C    100      300

Comparison with a scalar, using either the operator or method:

>>> df == 100
   cost  revenue
A  False    True
B  False   False
C  True   False

>>> df.eq(100)
   cost  revenue
A  False    True
B  False   False
C  True   False

When 'other' is a :class:`Series`, the columns of a DataFrame are aligned
with the index of 'other' and broadcast:

>>> df != pd.Series([100, 250], index=["cost", "revenue"])
   cost  revenue
A  True    True
B  True   False
C  False   True

Use the method to control the broadcast axis:

>>> df.ne(pd.Series([100, 300], index=["A", "D"]), axis='index')
   cost  revenue
A  True    True
B  True    True
C  True    True
D  True    True

When comparing to an arbitrary sequence, the number of columns must
match the number elements in 'other':

>>> df == [250, 100]
   cost  revenue
A  True    True
B  False   False
C  False   False

Use the method to control the axis:

>>> df.eq([250, 250, 100], axis='index')
   cost  revenue
A  True    False
B  False   True
C  True   False

Compare to a DataFrame of different shape.

>>> other = pd.DataFrame({'revenue': [300, 250, 100, 150]},
```

```

    ...                               index=['A', 'B', 'C', 'D'])

>>> other
      revenue
A     300
B     250
C     100
D     150

>>> df.gt(other)
      cost   revenue
A  False   False
B  False   False
C  False    True
D  False   False

Compare to a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'cost': [250, 150, 100, 150, 300, 220],
...                                'revenue': [100, 250, 300, 200, 175,
225]}, ...
| ...                               index=[['Q1', 'Q1', 'Q1', 'Q2', 'Q2',
'Q2'], ...
| ...                               ['A', 'B', 'C', 'A', 'B', 'C']])
>>> df_multindex
      cost   revenue
Q1 A   250    100
      B   150    250
      C   100    300
Q2 A   150    200
      B   300    175
      C   220    225

>>> df.le(df_multindex, level=1)
      cost   revenue
Q1 A   True   True
      B   True   True
      C   True   True
Q2 A  False   True
      B   True  False
      C   True  False

| lookup(self, row_labels: 'Sequence[IndexLabel]', col_labels:
| 'Sequence[IndexLabel]') -> 'np.ndarray'
| Label-based "fancy indexing" function for DataFrame.
| Given equal-length arrays of row and column labels, return an
| array of the values corresponding to each (row, col) pair.

.. deprecated:: 1.2.0
    Dataframe.lookup is deprecated,
    use DataFrame.melt and DataFrame.loc instead.
    For further details see
    :ref:`Looking up values by index/column labels <indexing.lookup>`.

Parameters
-----
row_labels : sequence
    The row labels to use for lookup.
col_labels : sequence
    The column labels to use for lookup.

Returns
-----
numpy.ndarray
    The found values.

lt(self, other, axis='columns', level=None)
    Get Less than of dataframe and other, element-wise (binary operator `lt`).

    Among flexible wrappers (`eq`, `ne`, `le`, `lt`, `ge`, `gt`) to comparison
    operators.

    Equivalent to `==`, `!=`, `<=`, `<`, `>=`, `>` with support to choose axis
    (rows or columns) and level for comparison.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}, default 'columns'
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns').
level : int or label
    Broadcast across a level, matching Index values on the passed
    MultiIndex level.

Returns
-----
DataFrame of bool
    Result of the comparison.

See Also
-----
DataFrame.eq : Compare DataFrames for equality elementwise.
DataFrame.ne : Compare DataFrames for inequality elementwise.
DataFrame.le : Compare DataFrames for less than inequality
    or equality elementwise.
DataFrame.lt : Compare DataFrames for strictly less than
    inequality elementwise.
DataFrame.ge : Compare DataFrames for greater than inequality
    or equality elementwise.
DataFrame.gt : Compare DataFrames for strictly greater than
    inequality elementwise.

Notes
-----
Mismatched indices will be unioned together.
`NaN` values are considered different (i.e. `NaN` != `NaN`).

Examples
-----
>>> df = pd.DataFrame({'cost': [250, 150, 100],
...                      'revenue': [100, 250, 300]},
...                      index=['A', 'B', 'C'])
>>> df
      cost   revenue
A   250    100
B   150    250
C   100    300

Comparison with a scalar, using either the operator or method:

>>> df == 100
      cost   revenue
A  False   True
B  False  False
C  True  False

>>> df.eq(100)
      cost   revenue
A  False   True
B  False  False
C  True  False

```

```

C  True  False
When 'other' is a :class:`Series`, the columns of a DataFrame are aligned
with the index of 'other' and broadcast:
>>> df != pd.Series([100, 250], index=["cost", "revenue"])
   cost  revenue
A  True    True
B  True    False
C  False   True
Use the method to control the broadcast axis:
>>> df.ne(pd.Series([100, 300], index=["A", "D"]), axis='index')
   cost  revenue
A  True    True
B  True    True
C  True    True
D  True    True
When comparing to an arbitrary sequence, the number of columns must
match the number elements in 'other':
>>> df == [250, 100]
   cost  revenue
A  True    True
B  False   False
C  False   False
Use the method to control the axis:
>>> df.eq([250, 250, 100], axis='index')
   cost  revenue
A  True    False
B  False   True
C  True    False
Compare to a DataFrame of different shape.
>>> other = pd.DataFrame({'revenue': [300, 250, 100, 150]},
...                           index=['A', 'B', 'C', 'D'])
>>> other
   revenue
A      300
B      250
C      100
D      150
>>> df.gt(other)
   cost  revenue
A  False   False
B  False   False
C  False   True
D  False   False
Compare to a MultiIndex by level.
>>> df_multindex = pd.DataFrame({'cost': [250, 150, 100, 150, 300, 220],
...                                'revenue': [100, 250, 300, 200, 175,
225]}, ...
...                           index=[['Q1', 'Q1', 'Q1', 'Q2', 'Q2',
'Q2'], ...
...                                ['A', 'B', 'C', 'A', 'B', 'C']])
>>> df_multindex
   cost  revenue
Q1 A    250     100
      B    150     250
      C    100     300
Q2 A    150     200
      B    300     175
      C    220     225
>>> df.le(df_multindex, level=1)
   cost  revenue
Q1 A  True    True
      B  True    True
      C  True    True
Q2 A  False   True
      B  True   False
      C  True   False
mad(self, axis=None, skipna=True, level=None)
Return the mean absolute deviation of the values over the requested axis.

Parameters
-----
axis : {index (0), columns (1)}
    Axis for the function to be applied on.
skipna : bool, default True
    Exclude NA/null values when computing the result.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.

Returns
-----
Series or DataFrame (if level specified)

mask(self, cond, other=nan, inplace=False, axis=None, level=None,
errors='raise', try_cast=<no_default>)
Replace values where the condition is True.

Parameters
-----
cond : bool Series/DataFrame, array-like, or callable
    Where 'cond' is False, keep the original value. Where
    True, replace with corresponding value from 'other'.
    If 'cond' is callable, it is computed on the Series/DataFrame and
    should return boolean Series/DataFrame or array. The callable must
    not change input Series/DataFrame (though pandas doesn't check it).
other : scalar, Series/DataFrame, or callable
    Entries where 'cond' is True are replaced with
    corresponding value from 'other'.
    If other is callable, it is computed on the Series/DataFrame and
    should return scalar or Series/DataFrame. The callable must not
    change input Series/DataFrame (though pandas doesn't check it).
inplace : bool, default False
    Whether to perform the operation in place on the data.
axis : int, default None
    Alignment axis if needed.
level : int, default None
    Alignment level if needed.
errors : str, {'raise', 'ignore'}, default 'raise'
    Note that currently this parameter won't affect
    the results and will always coerce to a suitable dtype.
    - 'raise' : allow exceptions to be raised.
    - 'ignore' : suppress exceptions. On error return original object.

try_cast : bool, default None
    Try to cast the result back to the input type (if possible).

```

```

    .. deprecated:: 1.3.0
        Manually cast back if necessary.

>Returns
-----
Same type as caller or None if ``inplace=True``.

>See Also
-----
:func:`DataFrame.where` : Return an object of same shape as
    self.

>Notes
-----
The mask method is an application of the if-then idiom. For each
element in the calling DataFrame, if ``cond`` is ``False`` the
element is used; otherwise the corresponding element from the DataFrame
``other`` is used.

The signature for :func:`DataFrame.where` differs from
:func:`numpy.where`. Roughly ``df1.where(m, df2)`` is equivalent to
``np.where(m, df1, df2)``.

For further details and examples see the ``mask`` documentation in
:ref:`indexing <indexing.where_mask>`.

>Examples
-----
>>> s = pd.Series(range(5))
>>> s.where(s > 0)
0      NaN
1      1.0
2      2.0
3      3.0
4      4.0
dtype: float64
>>> s.mask(s > 0)
0      0.0
1      NaN
2      NaN
3      NaN
4      NaN
dtype: float64

>>> s.where(s > 1, 10)
0      10
1      10
2      2
3      3
4      4
dtype: int64
>>> s.mask(s > 1, 10)
0      0
1      1
2      10
3      10
4      10
dtype: int64

>>> df = pd.DataFrame(np.arange(10).reshape(-1, 2), columns=['A', 'B'])
>>> df
   A   B
0   0   1
1   2   3
2   4   5
3   6   7
4   8   9
>>> m = df % 3 == 0
>>> df.where(m, -df)
   A   B
0   0  -1
1  -2   3
2  -4  -5
3  -6  -7
4  -8   9
>>> df.where(m, -df) == np.where(m, df, -df)
   A   B
0  True  True
1  True  True
2  True  True
3  True  True
4  True  True
>>> df.where(m, -df) == df.mask(~m, -df)
   A   B
0  True  True
1  True  True
2  True  True
3  True  True
4  True  True

| max(self, axis: 'int | None | lib.NoDefault' = <no_default>, skipna=True,
| level=None, numeric_only=None, **kwargs)
|     Return the maximum of the values over the requested axis.

| If you want the *index* of the maximum, use ``idxmax``. This is the
| equivalent of the ``numpy.ndarray`` method ``argmax``.

>Parameters
-----
axis : {index (0), columns (1)}
    Axis for the function to be applied on.
skipna : bool, default True
    Exclude NA/null values when computing the result.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.
**kwargs
    Additional keyword arguments to be passed to the function.

>Returns
-----
Series or DataFrame (if level specified)

>See Also
-----
Series.sum : Return the sum.
Series.min : Return the minimum.
Series.max : Return the maximum.
Series.idxmin : Return the index of the minimum.
Series.idxmax : Return the index of the maximum.
DataFrame.sum : Return the sum over the requested axis.
DataFrame.min : Return the minimum over the requested axis.
DataFrame.max : Return the maximum over the requested axis.
DataFrame.idxmin : Return the index of the minimum over the requested
axis.
| DataFrame.idxmax : Return the index of the maximum over the requested
axis.

```

```

Examples
-----
>>> idx = pd.MultiIndex.from_arrays([
...     ['warm', 'warm', 'cold', 'cold'],
...     ['dog', 'falcon', 'fish', 'spider'],
...     names=['blooded', 'animal']])
>>> s = pd.Series([4, 2, 0, 8], name='legs', index=idx)
>>> s
blooded animal
warm    dog    4
        falcon  2
cold    fish    0
        spider  8
Name: legs, dtype: int64

>>> s.max()
8

mean(self, axis: 'int | None | lib.NoDefault' = <no_default>, skipna=True,
level=None, numeric_only=None, **kwargs)
    Return the mean of the values over the requested axis.

Parameters
-----
axis : {index (0), columns (1)}
    Axis for the function to be applied on.
skipna : bool, default True
    Exclude NA/null values when computing the result.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.
**kwargs
    Additional keyword arguments to be passed to the function.

Returns
-----
Series or DataFrame (if level specified)

median(self, axis: 'int | None | lib.NoDefault' = <no_default>, skipna=True,
level=None, numeric_only=None, **kwargs)
    Return the median of the values over the requested axis.

Parameters
-----
axis : {index (0), columns (1)}
    Axis for the function to be applied on.
skipna : bool, default True
    Exclude NA/null values when computing the result.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.
**kwargs
    Additional keyword arguments to be passed to the function.

Returns
-----
Series or DataFrame (if level specified)

melt(self, id_vars=None, value_vars=None, var_name=None, value_name='value',
col_level: 'Level | None' = None, ignore_index: 'bool' = True) -> 'DataFrame'
    Unpivot a DataFrame from wide to long format, optionally leaving
    identifiers set.

    This function is useful to massage a DataFrame into a format where one
    or more columns are identifier variables ('id_vars'), while all other
    columns, considered measured variables ('value_vars'), are "unpivoted" to
    the row axis, leaving just two non-identifier columns, 'variable' and
    'value'.

Parameters
-----
id_vars : tuple, list, or ndarray, optional
    Column(s) to use as identifier variables.
value_vars : tuple, list, or ndarray, optional
    Column(s) to unpivot. If not specified, uses all columns that
    are not set as 'id_vars'.
var_name : scalar
    Name to use for the 'variable' column. If None it uses
    ``frame.columns.name`` or 'variable'.
value_name : scalar, default 'value'
    Name to use for the 'value' column.
col_level : int or str, optional
    If columns are a MultiIndex then use this level to melt.
ignore_index : bool, default True
    If True, original index is ignored. If False, the original index is
    retained.
    Index labels will be repeated as necessary.

    .. versionadded:: 1.1.0

Returns
-----
DataFrame
    Unpivoted DataFrame.

See Also
-----
melt : Identical method.
pivot_table : Create a spreadsheet-style pivot table as a DataFrame.
DataFrame.pivot : Return reshaped DataFrame organized
    by given index / column values.
DataFrame.explode : Explode a DataFrame from list-like
    columns to long format.

Notes
-----
Reference :ref:`the user guide <reshaping.melt>` for more examples.

Examples
-----
>>> df = pd.DataFrame({'A': {0: 'a', 1: 'b', 2: 'c'},
...                      'B': {0: 1, 1: 3, 2: 5},
...                      'C': {0: 2, 1: 4, 2: 6}})
>>> df
   A  B  C
0  a  1  2
1  b  3  4
2  c  5  6

>>> df.melt(id_vars=['A'], value_vars=['B'])
   A variable  value
0  a          B      1
1  b          B      3
2  c          B      5

```

```

>>> df.melt(id_vars=['A'], value_vars=['B', 'C'])
   A variable  value
0  a           B    1
1  b           B    3
2  c           B    5
3  a           C    2
4  b           C    4
5  c           C    6

The names of 'variable' and 'value' columns can be customized:

>>> df.melt(id_vars=['A'], value_vars=['B'],
...           var_name='myVarname', value_name='myValname')
   A myVarname  myValname
0  a           B        1
1  b           B        3
2  c           B        5

Original index values can be kept around:

>>> df.melt(id_vars=['A'], value_vars=['B', 'C'], ignore_index=False)
   A variable  value
0  a           B    1
1  b           B    3
2  c           B    5
0  a           C    2
1  b           C    4
2  c           C    6

If you have multi-index columns:

>>> df.columns = [list('ABC'), list('DEF')]
>>> df
   A  B  C
   D  E  F
0  a  1  2
1  b  3  4
2  c  5  6

>>> df.melt(col_level=0, id_vars=['A'], value_vars=['B'])
   A variable  value
0  a           B    1
1  b           B    3
2  c           B    5

>>> df.melt(id_vars=[('A', 'D')], value_vars=[('B', 'E')])
   (A, D) variable_0 variable_1  value
0  a           B           E    1
1  b           B           E    3
2  c           B           E    5

memory_usage(self, index: 'bool' = True, deep: 'bool' = False) -> 'Series'
Return the memory usage of each column in bytes.

The memory usage can optionally include the contribution of
the index and elements of 'object' dtype.

This value is displayed in `DataFrame.info` by default. This can be
suppressed by setting ``pandas.options.display.memory_usage`` to False.

Parameters
-----
index : bool, default True
    Specifies whether to include the memory usage of the DataFrame's
    index in returned Series. If ``index=True``, the memory usage of
    the index is the first item in the output.
deep : bool, default False
    If True, introspect the data deeply by interrogating
    'object' dtypes for system-level memory consumption, and include
    it in the returned values.

Returns
-----
Series
    A Series whose index is the original column names and whose values
    is the memory usage of each column in bytes.

See Also
-----
numpy.ndarray.nbytes : Total bytes consumed by the elements of an
    ndarray.
Series.memory_usage : Bytes consumed by a Series.
Categorical : Memory-efficient array for string values with
    many repeated values.
DataFrame.info : Concise summary of a DataFrame.

Examples
-----
>>> dtypes = ['int64', 'float64', 'complex128', 'object', 'bool']
>>> data = dict([(t, np.ones(shape=5000, dtype=int).astype(t))
...               for t in dtypes])
>>> df = pd.DataFrame(data)
>>> df.head()
   int64  float64      complex128  object  bool
0      1       1.0      1.0+0.0j     1  True
1      1       1.0      1.0+0.0j     1  True
2      1       1.0      1.0+0.0j     1  True
3      1       1.0      1.0+0.0j     1  True
4      1       1.0      1.0+0.0j     1  True

>>> df.memory_usage()
Index          128
int64        40000
float64      40000
complex128   80000
object       40000
bool         5000
dtype: int64

>>> df.memory_usage(index=False)
int64        40000
float64      40000
complex128   80000
object       40000
bool         5000
dtype: int64

The memory footprint of 'object' dtype columns is ignored by default:

>>> df.memory_usage(deep=True)
Index          128
int64        40000
float64      40000
complex128   80000
object      180000
bool         5000
dtype: int64

Use a Categorical for efficient storage of an object-dtype column with
many repeated values.

```

```

>>> df['object'].astype('category').memory_usage(deep=True)
5244

| merge(self, right: 'DataFrame | Series', how: 'str' = 'inner', on: 'IndexLabel'
| None' = None, left_on: 'IndexLabel | None' = None, right_on: 'IndexLabel | None'
| = None, left_index: 'bool' = False, right_index: 'bool' = False, sort: 'bool' =
| False, suffixes: 'Suffixes' = ('_x', '_y'), copy: 'bool' = True, indicator: 'bool'
| = False, validate: 'str | None' = None) -> 'DataFrame'
|     Merge DataFrame or named Series objects with a database-style join.
|
|     A named Series object is treated as a DataFrame with a single named
column.
|
|     The join is done on columns or indexes. If joining columns on
columns, the DataFrame indexes *will be ignored*. Otherwise if joining
indexes
|     on indexes or indexes on a column or columns, the index will be passed on.
When performing a cross merge, no column specifications to merge on are
allowed.
|
|     .. warning::
|
|         If both key columns contain rows where the key is a null value, those
rows will be matched against each other. This is different from usual
SQL
|         join behaviour and can lead to unexpected results.

Parameters
-----
right : DataFrame or named Series
    Object to merge with.
how : {'left', 'right', 'outer', 'inner', 'cross'}, default 'inner'
    Type of merge to be performed.

join;
|     * left: use only keys from left frame, similar to a SQL left outer
join;
|     preserve key order.
|     * right: use only keys from right frame, similar to a SQL right outer
join;
|     preserve key order.
|     * outer: use union of keys from both frames, similar to a SQL full
outer
|     join; sort keys lexicographically.
|     * inner: use intersection of keys from both frames, similar to a SQL
inner
|     join; preserve the order of the left keys.
|     * cross: creates the cartesian product from both frames, preserves the
order
|     of the left keys.
|
|     .. versionadded:: 1.2.0

on : label or list
    Column or index level names to join on. These must be found in both
DataFrames. If 'on' is None and not merging on indexes then this
defaults
    to the intersection of the columns in both DataFrames.
left_on : label or list, or array-like
    Column or index level names to join on in the left DataFrame. Can also
be an array or list of arrays of the length of the left DataFrame.
These arrays are treated as if they are columns.
right_on : label or list, or array-like
    Column or index level names to join on in the right DataFrame. Can
also
    be an array or list of arrays of the length of the right DataFrame.
These arrays are treated as if they are columns.
left_index : bool, default False
    Use the index from the left DataFrame as the join key(s). If it is a
MultiIndex, the number of keys in the other DataFrame (either the
index
    or a number of columns) must match the number of levels.
right_index : bool, default False
    Use the index from the right DataFrame as the join key. Same caveats
as
    left_index.
sort : bool, default False
    Sort the join keys lexicographically in the result DataFrame. If
False,
        the order of the join keys depends on the join type (how keyword).
suffixes : list-like, default is ("_x", "_y")
    A length-2 sequence where each element is optionally a string
indicating the suffix to add to overlapping column names in
'left' and 'right' respectively. Pass a value of 'None' instead
of a string to indicate that the column name from 'left' or
'right' should be left as-is, with no suffix. At least one of the
values must not be None.
copy : bool, default True
    If False, avoid copy if possible.
indicator : bool or str, default False
    If True, adds a column to the output DataFrame called '_merge' with
information on the source of each row. The column can be given a
different
|     name by providing a string argument. The column will have a
Categorical
|     type with the value of "left_only" for observations whose merge key
only
|     appears in the left DataFrame, "right_only" for observations
whose merge key only appears in the right DataFrame, and "both"
if the observation's merge key is found in both DataFrames.

validate : str, optional
    If specified, checks if merge is of specified type.
    * "one_to_one" or "1:1": check if merge keys are unique in both
    left and right datasets.
    * "one_to_many" or "1:m": check if merge keys are unique in left
    dataset.
    * "many_to_one" or "m:1": check if merge keys are unique in right
    dataset.
    * "many_to_many" or "m:m": allowed, but does not result in checks.

Returns
-----
DataFrame
    A DataFrame of the two merged objects.

See Also
-----
merge_ordered : Merge with optional filling/interpolation.
merge_asof : Merge on nearest keys.
DataFrame.join : Similar method using indices.

Notes
-----
Support for specifying index levels as the 'on', 'left_on', and
'right_on' parameters was added in version 0.23.0
Support for merging named Series objects was added in version 0.24.0

Examples
-----

```

```

>>> df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
...                      'value': [1, 2, 3, 5]})
>>> df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
...                      'value': [5, 6, 7, 8]})

>>> df1
   lkey value
0  foo     1
1  bar     2
2  baz     3
3  foo     5

>>> df2
   rkey value
0  foo     5
1  bar     6
2  baz     7
3  foo     8

Merge df1 and df2 on the lkey and rkey columns. The value columns have
the default suffixes, _x and _y, appended.

>>> df1.merge(df2, left_on='lkey', right_on='rkey')
   lkey  value_x  rkey  value_y
0  foo        1  foo        5
1  foo        1  foo        8
2  foo        5  foo        5
3  foo        5  foo        8
4  bar        2  bar        6
5  baz        3  baz        7

Merge DataFrames df1 and df2 with specified left and right suffixes
appended to any overlapping columns.

>>> df1.merge(df2, left_on='lkey', right_on='rkey',
...             suffixes=('_left', '_right'))
   lkey  value_left  rkey  value_right
0  foo        1  foo        5
1  foo        1  foo        8
2  foo        5  foo        5
3  foo        5  foo        8
4  bar        2  bar        6
5  baz        3  baz        7

Merge DataFrames df1 and df2, but raise an exception if the DataFrames
have
any overlapping columns.

>>> df1.merge(df2, left_on='lkey', right_on='rkey', suffixes=(False,
False))
Traceback (most recent call last):
...
ValueError: columns overlap but no suffix specified:
Index(['value'], dtype='object')

>>> df1 = pd.DataFrame({'a': ['foo', 'bar'], 'b': [1, 2]})
>>> df2 = pd.DataFrame({'a': ['foo', 'baz'], 'c': [3, 4]})

>>> df1
   a  b
0  foo  1
1  bar  2

>>> df2
   a  c
0  foo  3
1  baz  4

>>> df1.merge(df2, how='inner', on='a')
   a  b  c
0  foo  1  3
1  bar  2  NaN

>>> df1.merge(df2, how='left', on='a')
   a  b  c
0  foo  1  3.0
1  bar  2  NaN

>>> df1 = pd.DataFrame({'left': ['foo', 'bar']})
>>> df2 = pd.DataFrame({'right': [7, 8]})

>>> df1
   left
0  foo
1  bar

>>> df2
   right
0    7
1    8

>>> df1.merge(df2, how='cross')
   left  right
0  foo      7
1  foo      8
2  bar      7
3  bar      8

min(self, axis: 'int | None | lib.NoDefault' = <no_default>, skipna=True,
level=None, numeric_only=None, **kwargs)
Return the minimum of the values over the requested axis.

If you want the *index* of the minimum, use ``idxmin``. This is the
equivalent of the ``numpy.ndarray`` method ``argmin``.

Parameters
-----
axis : {index (0), columns (1)}
    Axis for the function to be applied on.
skipna : bool, default True
    Exclude NA/null values when computing the result.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.
**kwargs
    Additional keyword arguments to be passed to the function.

Returns
-----
Series or DataFrame (if level specified)

See Also
-----
Series.sum : Return the sum.
Series.min : Return the minimum.
Series.max : Return the maximum.
Series.idxmin : Return the index of the minimum.
Series.idxmax : Return the index of the maximum.
DataFrame.sum : Return the sum over the requested axis.
DataFrame.min : Return the minimum over the requested axis.
DataFrame.max : Return the maximum over the requested axis.
DataFrame.idxmin : Return the index of the minimum over the requested
axis.
DataFrame.idxmax : Return the index of the maximum over the requested
axis.

```

```

Examples
-----
>>> idx = pd.MultiIndex.from_arrays([
...     ['warm', 'warm', 'cold', 'cold'],
...     ['dog', 'falcon', 'fish', 'spider'],
...     names=['blooded', 'animal']])
>>> s = pd.Series([4, 2, 0, 8], name='legs', index=idx)
>>> s
blooded animal
warm    dog    4
        falcon  2
cold    fish    0
        spider  8
Name: legs, dtype: int64

>>> s.min()
0

mod(self, other, axis='columns', level=None, fill_value=None)
    Get Modulo of dataframe and other, element-wise (binary operator `mod`).

    Equivalent to ``dataframe % other`` , but with support to substitute a
fill_value
    for missing data in one of the inputs. With reverse version, `rmod`.

    Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful DataFrame alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                     index=['circle', 'triangle', 'rectangle'])
>>> df
   angles  degrees
circle      0      360
triangle    3      180
rectangle   4      360

Add a scalar with operator version which return the same
results.

>>> df + 1
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

>>> df.add(1)
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

Divide by constant with reverse version.

>>> df.div(10)
   angles  degrees
circle      0.0     36.0
triangle    0.3     18.0
rectangle   0.4     36.0

>>> df.rdiv(10)
   angles  degrees
circle      inf     0.027778
triangle    3.33333  0.055556
rectangle   2.50000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
   angles  degrees
circle      -1      358
triangle    2       178
rectangle   3       358

>>> df.sub([1, 2], axis='columns')
   angles  degrees
circle      -1      358
triangle    2       178
rectangle   3       358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
... 'rectangle']), ...
           axis='index')
   angles  degrees
circle      -1      359
triangle    2       179
rectangle   3       359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4]},
...                       index=['circle', 'triangle', 'rectangle'])

```

```

>>> other
      angles
circle     0
triangle   3
rectangle  4

>>> df * other
      angles  degrees
circle      0      NaN
triangle    9      NaN
rectangle  16      NaN

>>> df.mul(other, fill_value=0)
      angles  degrees
circle      0      0.0
triangle    9      0.0
rectangle  16      0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
...                                index=[['A', 'A', 'A', 'B', 'B'],
...                                       ['circle', 'triangle', 'rectangle',
...                                        'square', 'pentagon', 'hexagon']])
>>> df_multindex
      angles  degrees
A circle     0     360
triangle     3     180
rectangle    4     360
B square     4     360
pentagon     5     540
hexagon      6     720

>>> df.div(df_multindex, level=1, fill_value=0)
      angles  degrees
A circle     NaN     1.0
triangle    1.0     1.0
rectangle   1.0     1.0
B square     0.0     0.0
pentagon    0.0     0.0
hexagon     0.0     0.0

mode(self, axis: 'Axis' = 0, numeric_only: 'bool' = False, dropna: 'bool' =
True) -> 'DataFrame'
Get the mode(s) of each element along the selected axis.

The mode of a set of values is the value that appears most often.
It can be multiple values.

Parameters
-----
axis : {0 or 'index', 1 or 'columns'}, default 0
    The axis to iterate over while searching for the mode:
    * 0 or 'index' : get mode of each column
    * 1 or 'columns' : get mode of each row.

numeric_only : bool, default False
    If True, only apply to numeric columns.
dropna : bool, default True
    Don't consider counts of NaN/NaT.

Returns
-----
DataFrame
    The modes of each column or row.

See Also
-----
Series.mode : Return the highest frequency value in a Series.
Series.value_counts : Return the counts of values in a Series.

Examples
-----
>>> df = pd.DataFrame([('bird', 2, 2),
...                      ('mammal', 4, np.nan),
...                      ('arthropod', 8, 0),
...                      ('bird', 2, np.nan)],
...                      index=['falcon', 'horse', 'spider', 'ostrich'],
...                      columns=['species', 'legs', 'wings'])
>>> df
   species  legs  wings
falcon     bird    2   2.0
horse     mammal   4   NaN
spider   arthropod   8   0.0
ostrich    bird    2   NaN

By default, missing values are not considered, and the mode of wings
are both 0 and 2. Because the resulting DataFrame has two rows,
the second row of ``species`` and ``legs`` contains ``NaN``.

>>> df.mode()
   species  legs  wings
0   bird     2.0    0.0
1   NaN     NaN    2.0

Setting ``dropna=False`` ``NaN`` values are considered and they can be
the mode (like for wings).

>>> df.mode(dropna=False)
   species  legs  wings
0   bird     2     NaN
1   NaN     NaN    2.0

Setting ``numeric_only=True``, only the mode of numeric columns is
computed, and columns of other types are ignored.

>>> df.mode(numeric_only=True)
   legs  wings
0   2.0    0.0
1   NaN    2.0

To compute the mode over columns and not rows, use the axis parameter:

>>> df.mode(axis='columns', numeric_only=True)
   0   1
falcon  2.0  NaN
horse   4.0  NaN
spider   0.0  8.0
ostrich  2.0  NaN

mul(self, other, axis='columns', level=None, fill_value=None)
    Get Multiplication of dataframe and other, element-wise (binary operator
'mul').

    Equivalent to ``dataframe * other``, but with support to substitute a
fill_value
    for missing data in one of the inputs. With reverse version, `rmul`.

    Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to

```

```

arithmetic operators: `+`, `-`, `*`, `/`, `//`, `%`, `**`.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful Dataframe alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                     index=['circle', 'triangle', 'rectangle'])
>>> df
   angles  degrees
circle      0      360
triangle    3      180
rectangle   4      360

Add a scalar with operator version which return the same
results.

>>> df + 1
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

>>> df.add(1)
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

Divide by constant with reverse version.

>>> df.div(10)
   angles  degrees
circle     0.0      36.0
triangle   0.3      18.0
rectangle  0.4      36.0

>>> df.rdiv(10)
   angles  degrees
circle     inf      0.027778
triangle  3.333333  0.055556
rectangle 2.500000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
   angles  degrees
circle     -1      358
triangle    2      178
rectangle   3      358

>>> df.sub([1, 2], axis='columns')
   angles  degrees
circle     -1      358
triangle    2      178
rectangle   3      358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']),
...          axis='index')
   angles  degrees
circle     -1      359
triangle    2      179
rectangle   3      359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4],
...                         index=['circle', 'triangle', 'rectangle'])
>>> other
   angles
circle     0
triangle   3
rectangle  4

>>> df * other
   angles  degrees
circle     0      NaN
triangle   9      NaN
rectangle  16     NaN

>>> df.mul(other, fill_value=0)
   angles  degrees
circle     0      0.0
triangle   9      0.0
rectangle  16     0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]},
...                               index=[['A', 'A', 'A', 'B', 'B'],
...                                     ['circle', 'triangle', 'rectangle',
'square', 'pentagon', 'hexagon']])
>>> df_multindex

```

```

    angles  degrees
A circle      0     360
triangle      3     180
rectangle     4     360
B square      4     360
pentagon      5     540
hexagon       6     720

>>> df.div(df_multindex, level=1, fill_value=0)
          angles  degrees
A circle      NaN     1.0
triangle     1.0     1.0
rectangle    1.0     1.0
B square      0.0     0.0
pentagon     0.0     0.0
hexagon      0.0     0.0

multiply = mul(self, other, axis='columns', level=None, fill_value=None)

ne(self, other, axis='columns', level=None)
    Get Not equal to of dataframe and other, element-wise (binary operator
'ne').

    Among flexible wrappers ('eq', 'ne', 'le', 'lt', 'ge', 'gt') to comparison
operators.

    Equivalent to '==' , '!=', '<=' , '<', '>=' , '>' with support to choose axis
(rows or columns) and level for comparison.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}, default 'columns'
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns').
level : int or label
    Broadcast across a level, matching Index values on the passed
    MultiIndex level.

Returns
-----
DataFrame of bool
    Result of the comparison.

See Also
-----
DataFrame.eq : Compare DataFrames for equality elementwise.
DataFrame.ne : Compare DataFrames for inequality elementwise.
DataFrame.le : Compare DataFrames for less than inequality
    or equality elementwise.
DataFrame.lt : Compare DataFrames for strictly less than
    inequality elementwise.
DataFrame.ge : Compare DataFrames for greater than inequality
    or equality elementwise.
DataFrame.gt : Compare DataFrames for strictly greater than
    inequality elementwise.

Notes
-----
Mismatched indices will be unioned together.
'NaN' values are considered different (i.e. 'NaN' != 'NaN').

Examples
-----
>>> df = pd.DataFrame({'cost': [250, 150, 100],
...                      'revenue': [100, 250, 300]},
...                     index=['A', 'B', 'C'])
>>> df
   cost  revenue
A    250      100
B    150      250
C    100      300

Comparison with a scalar, using either the operator or method:

>>> df == 100
   cost  revenue
A  False   True
B  False  False
C  True  False

>>> df.eq(100)
   cost  revenue
A  False   True
B  False  False
C  True  False

When 'other' is a :class:`Series`, the columns of a DataFrame are aligned
with the index of 'other' and broadcast:

>>> df != pd.Series([100, 250], index=["cost", "revenue"])
   cost  revenue
A  True   True
B  True  False
C  False  True

Use the method to control the broadcast axis:

>>> df.ne(pd.Series([100, 300], index=["A", "D"]), axis='index')
   cost  revenue
A  True  False
B  True   True
C  True   True
D  True   True

When comparing to an arbitrary sequence, the number of columns must
match the number elements in 'other':

>>> df == [250, 100]
   cost  revenue
A  True   True
B  False  False
C  False  False

Use the method to control the axis:

>>> df.eq([250, 250, 100], axis='index')
   cost  revenue
A  True  False
B  False   True
C  True  False

Compare to a DataFrame of different shape.

>>> other = pd.DataFrame({'revenue': [300, 250, 100, 150],
...                         index=['A', 'B', 'C', 'D'])
>>> other
   revenue
A      300
B      250
C      100
D      150

```

```

C      100
D      150

>>> df.gt(other)
   cost  revenue
A  False    False
B  False    False
C  False     True
D  False    False

Compare to a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'cost': [250, 150, 100, 150, 300, 220],
...                                'revenue': [100, 250, 300, 200, 175,
225]}, ...
...                                index=[['Q1', 'Q1', 'Q1', 'Q2', 'Q2',
'Q2'], ...
...                                ['A', 'B', 'C', 'A', 'B', 'C']])
>>> df_multindex
   cost  revenue
Q1 A    250     100
      B    150     250
      C    100     300
Q2 A    150     200
      B    300     175
      C    220     225

>>> df.le(df_multindex, level=1)
   cost  revenue
Q1 A  True    True
      B  True    True
      C  True    True
Q2 A False    True
      B  True   False
      C  True   False

nlargest(self, n: 'int', columns: 'IndexLabel', keep: 'str' = 'first') ->
'dataframe'
Return the first `n` rows ordered by `columns` in descending order.

Return the first `n` rows with the largest values in `columns`, in
descending order. The columns that are not specified are returned as
well, but not used for ordering.

This method is equivalent to
``df.sort_values(columns, ascending=False).head(n)``, but more
performant.

Parameters
-----
n : int
    Number of rows to return.
columns : label or list of labels
    Column label(s) to order by.
keep : {'first', 'last', 'all'}, default 'first'
    Where there are duplicate values:
    - ``'first'`` : prioritize the first occurrence(s)
    - ``'last'`` : prioritize the last occurrence(s)
    - ``'all'`` : do not drop any duplicates, even it means
        selecting more than `n` items.

>Returns
-----
DataFrame
    The first `n` rows ordered by the given columns in descending
    order.

See Also
-----
DataFrame.nsmallest : Return the first `n` rows ordered by `columns` in
    ascending order.
DataFrame.sort_values : Sort DataFrame by the values.
DataFrame.head : Return the first `n` rows without re-ordering.

Notes
-----
This function cannot be used with all column types. For example, when
specifying columns with ``object`` or ``category`` dtypes, ``TypeError`` is
raised.

Examples
-----
>>> df = pd.DataFrame({'population': [59000000, 65000000, 434000,
...                                     434000, 434000, 337000, 11300,
...                                     11300, 11300],
...                                'GDP': [1937894, 2583560, 12011, 4520, 12128,
...                                     17036, 182, 38, 311],
...                                'alpha-2': ['IT', 'FR', 'MT', 'MV', 'BN',
...                                     'IS', 'NR', 'TV', 'AI'],
...                                index=['Italy', 'France', 'Malta',
...                                     'Maldives', 'Brunei', 'Iceland',
...                                     'Nauru', 'Tuvalu', 'Anguilla'])
```

	population	GDP	alpha-2
Italy	59000000	1937894	IT
France	65000000	2583560	FR
Malta	434000	12011	MT
Maldives	434000	4520	MV
Brunei	434000	12128	BN
Iceland	337000	17036	IS
Nauru	11300	182	NR
Tuvalu	11300	38	TV
Anguilla	11300	311	AI

In the following example, we will use ``nlargest`` to select the three rows having the largest values in column "population".

```

>>> df.nlargest(3, 'population')
   population  GDP alpha-2
France    65000000  2583560   FR
Italy     59000000  1937894   IT
Malta     434000    12011   MT

When using ``keep='last````', ties are resolved in reverse order:
```

	population	GDP	alpha-2
France	65000000	2583560	FR
Italy	59000000	1937894	IT
Malta	434000	12011	MT

When using ``keep='all'````', all duplicate items are maintained:

```

>>> df.nlargest(3, 'population', keep='all')
   population  GDP alpha-2
France    65000000  2583560   FR
Italy     59000000  1937894   IT
Malta     434000    12011   MT
Maldives   434000    4520    MV
Brunei    434000    12128   BN
```

To order by the largest values in column "population" and then "GDP", we can specify multiple columns like in the next example.

```
>>> df.nlargest(3, ['population', 'GDP'])
   population      GDP
France    65000000  2583560
Italy     59000000  1937894
Brunei     434000   12128
```

`notna(self) -> 'DataFrame'`
 Detect existing (non-missing) values.

Return a boolean same-sized object indicating if the values are not NA. Non-missing values get mapped to True. Characters such as empty strings `''` or :attr:`numpy.inf` are not considered NA values (unless you set ``pandas.options.mode.use_inf_as_na = True``). NA values, such as None or :attr:`numpy.NaN`, get mapped to False values.

Returns

`DataFrame`
 Mask of bool values for each element in DataFrame that indicates whether an element is not an NA value.

See Also

`DataFrame.notnull` : Alias of `notna`.
`DataFrame.isna` : Boolean inverse of `notna`.
`DataFrame.dropna` : Omit axes labels with missing values.
`notna` : Top-level `notna`.

Examples

Show which entries in a DataFrame are not NA.

```
>>> df = pd.DataFrame(dict(age=[5, 6, np.NaN],
...                           born=[pd.NaT, pd.Timestamp('1939-05-27'),
...                                 pd.Timestamp('1940-04-25')],
...                           name=['Alfred', 'Batman', ''],
...                           toy=[None, 'Batmobile', 'Joker']))
>>> df
   age      born      name      toy
0  5.0      NaT    Alfred      None
1  6.0  1939-05-27    Batman  Batmobile
2  NaN  1940-04-25        ''       Joker
```

```
>>> df.notna()
   age      born      name      toy
0  True    False    True    False
1  True    True     True     True
2  False   True     True     True
```

Show which entries in a Series are not NA.

```
>>> ser = pd.Series([5, 6, np.NaN])
>>> ser
0    5.0
1    6.0
2    NaN
dtype: float64
```

```
>>> ser.notna()
0    True
1    True
2    False
dtype: bool
```

`notnull(self) -> 'DataFrame'`
`DataFrame.notnull` is an alias for `DataFrame.notna`.

Detect existing (non-missing) values.

Return a boolean same-sized object indicating if the values are not NA. Non-missing values get mapped to True. Characters such as empty strings `''` or :attr:`numpy.inf` are not considered NA values (unless you set ``pandas.options.mode.use_inf_as_na = True``). NA values, such as None or :attr:`numpy.NaN`, get mapped to False values.

Returns

`DataFrame`
 Mask of bool values for each element in DataFrame that indicates whether an element is not an NA value.

See Also

`DataFrame.notnull` : Alias of `notna`.
`DataFrame.isna` : Boolean inverse of `notna`.
`DataFrame.dropna` : Omit axes labels with missing values.
`notna` : Top-level `notna`.

Examples

Show which entries in a DataFrame are not NA.

```
>>> df = pd.DataFrame(dict(age=[5, 6, np.NaN],
...                           born=[pd.NaT, pd.Timestamp('1939-05-27'),
...                                 pd.Timestamp('1940-04-25')],
...                           name=['Alfred', 'Batman', ''],
...                           toy=[None, 'Batmobile', 'Joker']))
>>> df
   age      born      name      toy
0  5.0      NaT    Alfred      None
1  6.0  1939-05-27    Batman  Batmobile
2  NaN  1940-04-25        ''       Joker
```

```
>>> df.notna()
   age      born      name      toy
0  True    False    True    False
1  True    True     True     True
2  False   True     True     True
```

Show which entries in a Series are not NA.

```
>>> ser = pd.Series([5, 6, np.NaN])
>>> ser
0    5.0
1    6.0
2    NaN
dtype: float64
```

```
>>> ser.notna()
0    True
1    True
2    False
dtype: bool
```

`nsmallest(self, n: 'int', columns: 'IndexLabel', keep: 'str' = 'first') ->`

```

'DataFrame'
|   Return the first `n` rows ordered by `columns` in ascending order.
|   Return the first `n` rows with the smallest values in `columns`, in
|   ascending order. The columns that are not specified are returned as
|   well, but not used for ordering.
|
|   This method is equivalent to
|   ``df.sort_values(columns, ascending=True).head(n)``, but more
|   performant.
|
| Parameters
| -----
| n : int
|     Number of items to retrieve.
| columns : list or str
|     Column name or names to order by.
| keep : {'first', 'last', 'all'}, default 'first'
|     Where there are duplicate values:
|
|     - ``'first'`` : take the first occurrence.
|     - ``'last'`` : take the last occurrence.
|     - ``'all'`` : do not drop any duplicates, even if it means
|       selecting more than `n` items.
|
| Returns
| -----
| DataFrame
|
| See Also
| -----
| DataFrame.nlargest : Return the first `n` rows ordered by `columns` in
|   descending order.
| DataFrame.sort_values : Sort DataFrame by the values.
| DataFrame.head : Return the first `n` rows without re-ordering.
|
| Examples
| -----
|>>> df = pd.DataFrame({'population': [59000000, 65000000, 434000,
...                                         434000, 434000, 337000, 337000,
...                                         11300, 11300],
...                         'GDP': [1937894, 2583560, 12011, 4520, 12128,
...                                         17036, 182, 38, 311],
...                         'alpha-2': ['IT', 'FR', 'MT', 'MV', 'BN',
...                                         'IS', 'NR', 'TV', 'AI'],
...                         'index':['Italy', 'France', 'Malta',
...                                         'Maldives', 'Brunei', 'Iceland',
...                                         'Nauru', 'Tuvalu', 'Anguilla']})
|>>> df
      population    GDP alpha-2
Italy      59000000  1937894    IT
France     65000000  2583560    FR
Malta      434000    12011     MT
Maldives    434000    4520      MV
Brunei     434000    12128     BN
Iceland    337000    17036     IS
Nauru      337000    182       NR
Tuvalu     11300     38        TV
Anguilla    11300     311       AI
Iceland    337000    17036     IS

In the following example, we will use ``nsmallest`` to select the
three rows having the smallest values in column "population".
|
|>>> df.nsmallest(3, 'population')
      population    GDP alpha-2
Tuvalu     11300     38        TV
Anguilla    11300     311       AI
Iceland    337000    17036     IS

When using ``keep='last''``, ties are resolved in reverse order:
|
|>>> df.nsmallest(3, 'population', keep='last')
      population    GDP alpha-2
Anguilla    11300     311       AI
Tuvalu     11300     38        TV
Nauru      337000    182       NR

When using ``keep='all'``, all duplicate items are maintained:
|
|>>> df.nsmallest(3, 'population', keep='all')
      population    GDP alpha-2
Tuvalu     11300     38        TV
Anguilla    11300     311       AI
Iceland    337000    17036     IS
Nauru      337000    182       NR

To order by the smallest values in column "population" and then "GDP", we
can
specify multiple columns like in the next example.
|
|>>> df.nsmallest(3, ['population', 'GDP'])
      population    GDP alpha-2
Tuvalu     11300     38        TV
Anguilla    11300     311       AI
Nauru      337000    182       NR

nunique(self, axis: 'Axis' = 0, dropna: 'bool' = True) -> 'Series'
Count number of distinct elements in specified axis.

Return Series with number of distinct elements. Can ignore Na
values.

Parameters
-----
axis : {0 or 'index', 1 or 'columns'}, default 0
    The axis to use. 0 or 'index' for row-wise, 1 or 'columns' for
    column-wise.
dropna : bool, default True
    Don't include NaN in the counts.
|
| Returns
| -----
| Series
|
| See Also
| -----
| Series.nunique: Method nunique for Series.
| DataFrame.count: Count non-NA cells for each column or row.
|
| Examples
| -----
|>>> df = pd.DataFrame({'A': [4, 5, 6], 'B': [4, 1, 1]})
|>>> df.nunique()
A    3
B    2
dtype: int64

|>>> df.nunique(axis=1)
0    1
1    2
2    2

```

```

    dtype: int64

pivot(self, index=None, columns=None, values=None) -> 'DataFrame'
    Return reshaped DataFrame organized by given index / column values.

    Reshape data (produce a "pivot" table) based on column values. Uses
    unique values from specified 'index' / 'columns' to form axes of the
    resulting DataFrame. This function does not support data
    aggregation, multiple values will result in a MultiIndex in the
    columns. See the :ref:`User Guide <reshaping>` for more on reshaping.

    Parameters
    -----
    index : str or object or a list of str, optional
        Column to use to make new frame's index. If None, uses
        existing index.

        .. versionchanged:: 1.1.0
            Also accept list of index names.

    columns : str or object or a list of str
        Column to use to make new frame's columns.

        .. versionchanged:: 1.1.0
            Also accept list of columns names.

    values : str, object or a list of the previous, optional
        Column(s) to use for populating new frame's values. If not
        specified, all remaining columns will be used and the result will
        have hierarchically indexed columns.

    Returns
    -----
    DataFrame
        Returns reshaped DataFrame.

    Raises
    -----
    ValueError:
        When there are any `index`, `columns` combinations with multiple
        values. `DataFrame.pivot_table` when you need to aggregate.

    See Also
    -----
    DataFrame.pivot_table : Generalization of pivot that can handle
        duplicate values for one index/column pair.
    DataFrame.unstack : Pivot based on the index values instead of a
        column.
    wide_to_long : Wide panel to long format. Less flexible but more
        user-friendly than melt.

    Notes
    -----
    For finer-tuned control, see hierarchical indexing documentation along
    with the related stack/unstack methods.

    Reference :ref:`the user guide <reshaping.pivot>` for more examples.

    Examples
    -----
    >>> df = pd.DataFrame({'foo': ['one', 'one', 'one', 'two', 'two',
    ...                           'two'],
    ...                      'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
    ...                      'baz': [1, 2, 3, 4, 5, 6],
    ...                      'zoo': ['x', 'y', 'z', 'q', 'w', 't']})

    >>> df
      foo   bar  baz  zoo
    0  one     A   1    x
    1  one     B   2    y
    2  one     C   3    z
    3  two     A   4    q
    4  two     B   5    w
    5  two     C   6    t

    >>> df.pivot(index='foo', columns='bar', values='baz')
    bar  A   B   C
    foo
    one  1   2   3
    two  4   5   6

    >>> df.pivot(index='foo', columns='bar')['baz']
    bar  A   B   C
    foo
    one  1   2   3
    two  4   5   6

    >>> df.pivot(index='foo', columns='bar', values=['baz', 'zoo'])
    baz  zoo
    bar  A   B   C   A   B   C
    foo
    one  1   2   3   x   y   z
    two  4   5   6   q   w   t

    You could also assign a list of column names or a list of index names.

    >>> df = pd.DataFrame({
    ...     "lev1": [1, 1, 1, 2, 2, 2],
    ...     "lev2": [1, 1, 2, 1, 1, 2],
    ...     "lev3": [1, 2, 1, 2, 1, 2],
    ...     "lev4": [1, 2, 3, 4, 5, 6],
    ...     "values": [0, 1, 2, 3, 4, 5]}
    ...)

    >>> df
      lev1  lev2  lev3  lev4  values
    0  1     1     1     1     0
    1  1     1     2     2     1
    2  1     2     1     3     2
    3  2     1     2     4     3
    4  2     1     1     5     4
    5  2     2     2     6     5

    >>> df.pivot(index="lev1", columns=["lev2", "lev3"], values="values")
    lev2  1     2
    lev3  1     2     1     2
    lev1
    1    0.0  1.0  2.0  NaN
    2    4.0  3.0  NaN  5.0

    >>> df.pivot(index=["lev1", "lev2"], columns=["lev3"], values="values")
    lev1  lev2
    1    1  0.0  1.0
          2  2.0  NaN
    2    1  4.0  3.0
          2  NaN  5.0

    A ValueError is raised if there are any duplicates.

    >>> df = pd.DataFrame({"foo": ['one', 'one', 'two', 'two'],
    ...                      "bar": ['A', 'A', 'B', 'C'],
    ...                      "baz": [1, 2, 3, 4]})

    >>> df

```



```

...
...                 aggfunc={'D': np.mean,
...                               'E': np.mean})
>>> table
      D         E
A   C
bar large  5.500000  7.500000
     small  5.500000  8.500000
foo large  2.000000  4.500000
     small  2.333333  4.333333

We can also calculate multiple types of aggregations for any given
value column.

>>> table = pd.pivot_table(df, values=['D', 'E'], index=['A', 'C'],
...                         aggfunc={'D': np.mean,
...                                   'E': [min, max, np.mean]})

>>> table
      D         E
          mean max      mean min
A   C
bar large  5.500000  9  7.500000  6
     small  5.500000  9  8.500000  8
foo large  2.000000  5  4.500000  4
     small  2.333333  6  4.333333  2

pop(self, item: 'Hashable') -> 'Series'
    Return item and drop from frame. Raise KeyError if not found.

Parameters
-----
item : label
    Label of column to be popped.

Returns
-----
Series

Examples
-----
>>> df = pd.DataFrame([('falcon', 'bird', 389.0),
...                      ('parrot', 'bird', 24.0),
...                      ('lion', 'mammal', 80.5),
...                      ('monkey', 'mammal', np.nan)],
...                     columns=('name', 'class', 'max_speed'))
>>> df
   name   class  max_speed
0  falcon   bird        389.0
1  parrot   bird         24.0
2    lion  mammal        80.5
3  monkey  mammal        NaN

>>> df.pop('class')
0    bird
1    bird
2  mammal
3  mammal
Name: class, dtype: object

>>> df
   name  max_speed
0  falcon       389.0
1  parrot        24.0
2    lion        80.5
3  monkey        NaN

pow(self, other, axis='columns', level=None, fill_value=None)
    Get Exponential power of dataframe and other, element-wise (binary
operator 'pow').

    Equivalent to ``dataframe ** other`` , but with support to substitute a
fill_value
    for missing data in one of the inputs. With reverse version, `rpow`.

    Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful DataFrame alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                     index=['circle', 'triangle', 'rectangle'])

>>> df
      angles  degrees
circle      0      360
triangle    3      180
rectangle   4      360

Add a scalar with operator version which return the same
results.

>>> df + 1
      angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

```

```

>>> df.add(1)
      angles  degrees
circle      1     361
triangle    4     181
rectangle   5     361

Divide by constant with reverse version.

>>> df.div(10)
      angles  degrees
circle      0.0     36.0
triangle    0.3     18.0
rectangle   0.4     36.0

>>> df.rdiv(10)
      angles  degrees
circle      inf  0.027778
triangle  3.333333  0.055556
rectangle 2.500000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
      angles  degrees
circle      -1     358
triangle    2     178
rectangle   3     358

>>> df.sub([1, 2], axis='columns')
      angles  degrees
circle      -1     358
triangle    2     178
rectangle   3     358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
           axis='index')
      angles  degrees
circle      -1     359
triangle    2     179
rectangle   3     359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4],
...                         index=['circle', 'triangle', 'rectangle'])
>>> other
      angles
circle      0
triangle    3
rectangle   4

>>> df * other
      angles  degrees
circle      0     NaN
triangle    9     NaN
rectangle  16     NaN

>>> df.mul(other, fill_value=0)
      angles  degrees
circle      0     0.0
triangle    9     0.0
rectangle  16     0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                 'degrees': [360, 180, 360, 360, 540,
720]}, ...
...                               index=[['A', 'A', 'A', 'B', 'B'],
...                                     ['circle', 'triangle', 'rectangle',
...                                      'square', 'pentagon', 'hexagon']])
>>> df_multindex
      angles  degrees
A circle      0     360
triangle      3     180
rectangle     4     360
B square      4     360
pentagon      5     540
hexagon       6     720

>>> df.div(df_multindex, level=1, fill_value=0)
      angles  degrees
A circle      NaN     1.0
triangle    1.0     1.0
rectangle   1.0     1.0
B square      0.0     0.0
pentagon     0.0     0.0
hexagon      0.0     0.0

prod(self, axis=None, skipna=True, level=None, numeric_only=None, min_count=0,
**kwargs)
Return the product of the values over the requested axis.

Parameters
-----
axis : {index (0), columns (1)}
    Axis for the function to be applied on.
skipna : bool, default True
    Exclude NA/null values when computing the result.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.
min_count : int, default 0
    The required number of valid values to perform the operation. If fewer
than
    ``min_count`` non-NA values are present the result will be NA.
**kwargs
    Additional keyword arguments to be passed to the function.

Returns
-----
Series or DataFrame (if level specified)

See Also
-----
Series.sum : Return the sum.
Series.min : Return the minimum.
Series.max : Return the maximum.
Series.idxmin : Return the index of the minimum.
Series.idxmax : Return the index of the maximum.
DataFrame.sum : Return the sum over the requested axis.
DataFrame.min : Return the minimum over the requested axis.
DataFrame.max : Return the maximum over the requested axis.
DataFrame.idxmin : Return the index of the minimum over the requested
axis.
| DataFrame.idxmax : Return the index of the maximum over the requested

```

```

axis.

| Examples
| -----
| By default, the product of an empty or all-NA Series is ``1``
| >>> pd.Series([], dtype="float64").prod()
| 1.0
|
| This can be controlled with the ``min_count`` parameter
| >>> pd.Series([], dtype="float64").prod(min_count=1)
| nan
|
| Thanks to the ``skipna`` parameter, ``min_count`` handles all-NA and
| empty series identically.
| >>> pd.Series([np.nan]).prod()
| 1.0
|
| >>> pd.Series([np.nan]).prod(min_count=1)
| nan
|
| product = prod(self, axis=None, skipna=True, level=None, numeric_only=None,
| min_count=0, **kwargs)
| |
| | quantile(self, q=0.5, axis='Axis' = 0, numeric_only: 'bool' = True,
| | interpolation: 'str' = 'linear')
| |     Return values at the given quantile over requested axis.
|
| Parameters
| -----
| q : float or array-like, default 0.5 (50% quantile)
|     Value between 0 <= q <= 1, the quantile(s) to compute.
| axis : {0, 1, 'index', 'columns'}, default 0
|     Equals 0 or 'index' for row-wise, 1 or 'columns' for column-wise.
| numeric_only : bool, default True
|     If False, the quantile of datetime and timedelta data will be
|     computed as well.
| interpolation : {'linear', 'lower', 'higher', 'midpoint', 'nearest'}
|     This optional parameter specifies the interpolation method to use,
|     when the desired quantile lies between two data points `i` and `j`:
|     * linear: `i + (j - i) * fraction`, where `fraction` is the
|       fractional part of the index surrounded by `i` and `j`.
|     * lower: `i`.
|     * higher: `j`.
|     * nearest: `i` or `j` whichever is nearest.
|     * midpoint: `(i + j) / 2.
|
| Returns
| -----
| Series or DataFrame
|
| If ``q`` is an array, a DataFrame will be returned where the
| index is ``q``, the columns are the columns of self, and the
| values are the quantiles.
| If ``q`` is a float, a Series will be returned where the
| index is the columns of self and the values are the quantiles.
|
| See Also
| -----
| core.window.Rolling.quantile: Rolling quantile.
| numpy.percentile: Numpy function to compute the percentile.
|
| Examples
| -----
| >>> df = pd.DataFrame(np.array([[1, 1], [2, 10], [3, 100], [4, 100]]),
| ...           columns=['a', 'b'])
| >>> df.quantile(.1)
| a    1.3
| b    3.7
| Name: 0.1, dtype: float64
| >>> df.quantile(.5)
|      a    b
| 0.1  1.3  3.7
| 0.5  2.5  55.0
|
| Specifying 'numeric_only=False' will also compute the quantile of
| datetime and timedelta data.
|
| >>> df = pd.DataFrame({'A': [1, 2],
| ...                      'B': [pd.Timestamp('2010'),
| ...                            pd.Timestamp('2011')],
| ...                      'C': [pd.Timedelta('1 days'),
| ...                            pd.Timedelta('2 days')]}
| ...                     )
| >>> df.quantile(0.5, numeric_only=False)
| A    1.5
| B    2010-07-02 12:00:00
| C    1 days 12:00:00
| Name: 0.5, dtype: object
|
| query(self, expr: 'str', inplace: 'bool' = False, **kwargs)
| Query the columns of a DataFrame with a boolean expression.
|
| Parameters
| -----
| expr : str
|     The query string to evaluate.
|
| You can refer to variables
| in the environment by prefixing them with an '@' character like
| ``@a + b``.
|
| You can refer to column names that are not valid Python variable names
| by surrounding them in backticks. Thus, column names containing spaces
| or punctuations (besides underscores) or starting with digits must be
| surrounded by backticks. (For example, a column named "Area (cm^2)"
| would
| keywords
|     be referenced as ``'Area (cm^2)``). Column names which are Python
|     (like "list", "for", "import", etc) cannot be used.
|
| For example, if one of your columns is called ``a a`` and you want
| to sum it with ``b``, your query should be ``a a + b``.
|
| .. versionadded:: 0.25.0
|     Backtick quoting introduced.
|
| .. versionadded:: 1.0.0
|     Expanding functionality of backtick quoting for more than only
|     spaces.
|
| inplace : bool
|     Whether the query should modify the data in place or return
|     a modified copy.
| **kwargs
|     See the documentation for :func:`eval` for complete details
|     on the keyword arguments accepted by :meth:`DataFrame.query`.
|
| Returns
| -----

```

```

-----
DataFrame or None
    Dataframe resulting from the provided query expression or
    None if ``inplace=True``.

See Also
-----
eval : Evaluate a string describing operations on
    Dataframe columns.
DataFrame.eval : Evaluate a string describing operations on
    Dataframe columns.

Notes
-----
The result of the evaluation of this expression is first passed to
:attr:`DataFrame.loc` and if that fails because of a
multidimensional key (e.g., a DataFrame) then the result will be passed
to :meth:`DataFrame.__getitem__`.

This method uses the top-level :func:`eval` function to
evaluate the passed query.

The :meth:`~pandas.DataFrame.query` method uses a slightly
modified Python syntax by default. For example, the ``&`` and ``|``
(bitwise) operators have the precedence of their boolean cousins,
:keyword:`and` and :keyword:`or`. This *is* syntactically valid Python,
however the semantics are different.

You can change the semantics of the expression by passing the keyword
argument ``parser='python'``. This enforces the same semantics as
evaluation in Python space. Likewise, you can pass ``engine='python'``
to evaluate an expression using Python itself as a backend. This is not
recommended as it is inefficient compared to using ``numexpr`` as the
engine.

The :attr:`DataFrame.index` and
:attr:`DataFrame.columns` attributes of the
:class:`pandas.DataFrame` instance are placed in the query namespace
by default, which allows you to treat both the index and columns of the
frame as a column in the frame.

The identifier ``index`` is used for the frame index; you can also
use the name of the index to identify it in a query. Please note that
Python keywords may not be used as identifiers.

For further details and examples see the ``query`` documentation in
:ref:`indexing <indexing.query>`.

*Backtick quoted variables*

Backtick quoted variables are parsed as literal Python code and
are converted internally to a Python valid identifier.
This can lead to the following problems.

During parsing a number of disallowed characters inside the backtick
quoted string are replaced by strings that are allowed as a Python
identifier.
These characters include all operators in Python, the space character, the
question mark, the exclamation mark, the dollar sign, and the euro sign.
For other characters that fall outside the ASCII range (U+0001..U+007F)
and those that are not further specified in PEP 3131,
the query parser will raise an error.
This excludes whitespace different than the space character,
but also the hashtag (as it is used for comments) and the backtick
itself (backtick can also not be escaped).

In a special case, quotes that make a pair around a backtick can
confuse the parser.
For example, ```it's` > `that's`` will raise an error,
as it forms a quoted string (``'s > 'that'``) with a backtick inside.

See also the Python documentation about lexical analysis
(https://docs.python.org/3/reference/lexical_analysis.html)
in combination with the source code in
:mod:`pandas.core.computation.parsing` .

Examples
-----
>>> df = pd.DataFrame({'A': range(1, 6),
...                      'B': range(10, 0, -2),
...                      'C C': range(10, 5, -1)})
>>> df
   A   B   C C
0  1  10   10
1  2   8    9
2  3   6    8
3  4   4    7
4  5   2    6
>>> df.query('A > B')
   A   B   C C
4  5   2    6

The previous expression is equivalent to

>>> df[df.A > df.B]
   A   B   C C
4  5   2    6

For columns with spaces in their name, you can use backtick quoting.

>>> df.query('B == `C C`')
   A   B   C C
0  1  10   10

The previous expression is equivalent to

>>> df[df.B == df['C C']]
   A   B   C C
0  1  10   10

radd(self, other, axis='columns', level=None, fill_value=None)
    Get Addition of dataframe and other, element-wise (binary operator
    'radd').

    Equivalent to ``other + dataframe``, but with support to substitute a
fill_value
    for missing data in one of the inputs. With reverse version, `add`.

Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None

```

Fill existing missing (NaN) values, and any new element needed for successful DataFrame alignment, with this value before computation. If data in both corresponding DataFrame locations is missing the result will be missing.

Returns

Dataframe
Result of the arithmetic operation.

See Also

Dataframe.add : Add DataFrames.
Dataframe.sub : Subtract DataFrames.
Dataframe.mul : Multiply DataFrames.
Dataframe.div : Divide DataFrames (float division).
Dataframe.truediv : Divide DataFrames (float division).
Dataframe.floordiv : Divide DataFrames (integer division).
Dataframe.mod : Calculate modulo (remainder after division).
Dataframe.pow : Calculate exponential power.

Notes

Mismatched indices will be unioned together.

Examples

```
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                     index=['circle', 'triangle', 'rectangle'])

>>> df
   angles  degrees
circle      0      360
triangle    3      180
rectangle   4      360

Add a scalar with operator version which return the same results.

>>> df + 1
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

>>> df.add(1)
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

Divide by constant with reverse version.

>>> df.div(10)
   angles  degrees
circle     0.0      36.0
triangle   0.3      18.0
rectangle  0.4      36.0

>>> df.rdiv(10)
   angles  degrees
circle     inf     0.027778
triangle  3.33333  0.055556
rectangle 2.50000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
   angles  degrees
circle     -1      358
triangle   2      178
rectangle  3      358

>>> df.sub([1, 2], axis='columns')
   angles  degrees
circle     -1      358
triangle   2      178
rectangle  3      358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
          axis='index')
   angles  degrees
circle     -1      359
triangle   2      179
rectangle  3      359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4],
...                         'index': ['circle', 'triangle', 'rectangle']})

>>> other
   angles
circle      0
triangle    3
rectangle   4

>>> df * other
   angles  degrees
circle      0      NaN
triangle    9      NaN
rectangle  16      NaN

>>> df.mul(other, fill_value=0)
   angles  degrees
circle      0      0.0
triangle    9      0.0
rectangle  16      0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]},
...                               ...
...                               index=[['A', 'A', 'A', 'B', 'B'],
...                                     ['circle', 'triangle', 'rectangle',
'square', 'pentagon', 'hexagon']])
>>> df_multindex
   angles  degrees
A circle      0      360
triangle      3      180
rectangle     4      360
B square      4      360
pentagon      5      540
hexagon       6      720

>>> df.div(df_multindex, level=1, fill_value=0)
   angles  degrees
A circle    NaN      1.0
triangle   1.0      1.0
rectangle  1.0      1.0
```

```

B square      0.0      0.0
pentagon     0.0      0.0
hexagon      0.0      0.0

rdiv = rtruediv(self, other, axis='columns', level=None, fill_value=None)

reindex(self, labels=None, index=None, columns=None, axis=None, method=None,
copy=True, level=None, fill_value=nan, limit=None, tolerance=None)
    Conform Series/Dataframe to new index with optional filling logic.

    Places NA/NaN in locations having no value in the previous index. A new
object
is produced unless the new index is equivalent to the current one and
``copy=False``.

Parameters
-----
keywords for axes : array-like, optional
    New labels / index to conform to, should be specified using
    keywords. Preferably an Index object to avoid duplicating data.

method : {None, 'backfill'/'bfill', 'pad'/'ffill', 'nearest'}
    Method to use for filling holes in reindexed Dataframe.
    Please note: this is only applicable to Dataframes/Series with a
monotonically increasing/decreasing index.

* None (default): don't fill gaps
* pad / ffill: Propagate last valid observation forward to next
valid.
* backfill / bfill: Use next valid observation to fill gap.
* nearest: Use nearest valid observations to fill gap.

copy : bool, default True
    Return a new object, even if the passed indexes are the same.
level : int or name
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : scalar, default np.NaN
    Value to use for missing values. Defaults to NaN, but can be any
    "compatible" value.
limit : int, default None
    Maximum number of consecutive elements to forward or backward fill.
tolerance : optional
    Maximum distance between original and new labels for inexact
matches. The values of the index at the matching locations must
satisfy the equation ``abs(index[indexer] - target) <= tolerance``.

Tolerance may be a scalar value, which applies the same tolerance
to all values, or list-like, which applies variable tolerance per
element. List-like includes list, tuple, array, Series, and must be
the same size as the index and its dtype must exactly match the
index's type.

Returns
-----
Series/DataFrame with changed index.

See Also
-----
DataFrame.set_index : Set row labels.
DataFrame.reset_index : Remove row labels or move them to new columns.
DataFrame.reindex_like : Change to same indices as other DataFrame.

Examples
-----
`'DataFrame.reindex'` supports two calling conventions

* ``{(index=index_labels, columns=column_labels, ...)}``
* ``{(labels, axis={'index', 'columns'}, ...)}``

We *highly* recommend using keyword arguments to clarify your
intent.

Create a dataframe with some fictional data.

>>> index = ['Firefox', 'Chrome', 'Safari', 'IE10', 'Konqueror']
>>> df = pd.DataFrame({'http_status': [200, 200, 404, 404, 301],
...                      'response_time': [0.04, 0.02, 0.07, 0.08, 1.0]},
...                      index=index)
>>> df
   http_status  response_time
Firefox        200           0.04
Chrome         200           0.02
Safari          404           0.07
IE10            404           0.08
Konqueror       301           1.00

Create a new index and reindex the dataframe. By default
values in the new index that do not have corresponding
records in the dataframre are assigned ``NaN``.

>>> new_index = ['Safari', 'Iceweasel', 'Comodo Dragon', 'IE10',
...               'Chrome']
>>> df.reindex(new_index)
   http_status  response_time
Safari          404.0           0.07
Iceweasel        NaN             NaN
Comodo Dragon    NaN             NaN
IE10            404.0           0.08
Chrome           200.0           0.02

We can fill in the missing values by passing a value to
the keyword ``fill_value``. Because the index is not monotonically
increasing or decreasing, we cannot use arguments to the keyword
``method`` to fill the ``NaN`` values.

>>> df.reindex(new_index, fill_value=0)
   http_status  response_time
Safari          404           0.07
Iceweasel        0           0.00
Comodo Dragon    0           0.00
IE10            404           0.08
Chrome           200           0.02

>>> df.reindex(new_index, fill_value='missing')
   http_status  response_time
Safari          404           0.07
Iceweasel        missing        missing
Comodo Dragon    missing        missing
IE10            404           0.08
Chrome           200           0.02

We can also reindex the columns.

>>> df.reindex(columns=['http_status', 'user_agent'])
   http_status  user_agent
Firefox        200           NaN
Chrome         200           NaN
Safari          404           NaN
IE10            404           NaN

```

```

Konqueror      301      NaN
Or we can use "axis-style" keyword arguments
>>> df.reindex(['http_status', 'user_agent'], axis="columns")
      http_status user_agent
Firefox        200      NaN
Chrome         200      NaN
Safari          404      NaN
IE10           404      NaN
Konqueror      301      NaN

To further illustrate the filling functionality in
``reindex``, we will create a dataframe with a
monotonically increasing index (for example, a sequence
of dates).

>>> date_index = pd.date_range('1/1/2010', periods=6, freq='D')
>>> df2 = pd.DataFrame({'prices': [100, 101, np.nan, 100, 89, 88]},
...                      index=date_index)
>>> df2
   prices
2010-01-01  100.0
2010-01-02  101.0
2010-01-03    NaN
2010-01-04  100.0
2010-01-05   89.0
2010-01-06   88.0

Suppose we decide to expand the dataframe to cover a wider
date range.

>>> date_index2 = pd.date_range('12/29/2009', periods=10, freq='D')
>>> df2.reindex(date_index2)
   prices
2009-12-29    NaN
2009-12-30    NaN
2009-12-31    NaN
2010-01-01  100.0
2010-01-02  101.0
2010-01-03    NaN
2010-01-04  100.0
2010-01-05   89.0
2010-01-06   88.0
2010-01-07    NaN

The index entries that did not have a value in the original data frame
(for example, '2009-12-29') are by default filled with ``NaN``.
If desired, we can fill in the missing values using one of several
options.

For example, to back-propagate the last valid value to fill the ``NaN``
values, pass ``bfill`` as an argument to the ``method`` keyword.

>>> df2.reindex(date_index2, method='bfill')
   prices
2009-12-29  100.0
2009-12-30  100.0
2009-12-31  100.0
2010-01-01  100.0
2010-01-02  101.0
2010-01-03    NaN
2010-01-04  100.0
2010-01-05   89.0
2010-01-06   88.0
2010-01-07    NaN

Please note that the ``NaN`` value present in the original dataframe
(at index value 2010-01-03) will not be filled by any of the
value propagation schemes. This is because filling while reindexing
does not look at dataframe values, but only compares the original and
desired indexes. If you do want to fill in the ``NaN`` values present
in the original dataframe, use the ``fillna()`` method.

See the :ref:`user guide <basics.reindexing>` for more.

rename(self, mapper: 'Renamer | None' = None, *, index: 'Renamer | None' =
None, columns: 'Renamer | None' = None, axis: 'Axis | None' = None, copy: 'bool' =
True, inplace: 'bool' = False, level: 'Level | None' = None, errors: 'str' =
'ignore') -> 'DataFrame | None'
    Alter axes labels.

Function / dict values must be unique (1-to-1). Labels not contained in
a dict / Series will be left as-is. Extra labels listed don't throw an
error.

See the :ref:`user guide <basics.rename>` for more.

Parameters
-----
mapper : dict-like or function
    Dict-like or function transformations to apply to
    that axis' values. Use either ``mapper`` and ``axis`` to
    specify the axis to target with ``mapper``; or ``index`` and
    ``columns``.
index : dict-like or function
    Alternative to specifying axis ('`mapper, axis=0``'
    is equivalent to ``index=mapper``).
columns : dict-like or function
    Alternative to specifying axis ('`mapper, axis=1``'
    is equivalent to ``columns=mapper``).
axis : {0 or 'index', 1 or 'columns'}, default 0
    Axis to target with ``mapper``. Can be either the axis name
    ('`index`', '`columns`') or number (0, 1). The default is 'index'.
copy : bool, default True
    Also copy underlying data.
inplace : bool, default False
    Whether to return a new DataFrame. If True then value of copy is
    ignored.
level : int or level name, default None
    In case of a MultiIndex, only rename labels in the specified
    level.
errors : {'ignore', 'raise'}, default 'ignore'
    If 'raise', raise a 'KeyError' when a dict-like `mapper`, `index`,
    or `columns` contains labels that are not present in the Index
    being transformed.
    If 'ignore', existing keys will be renamed and extra keys will be
    ignored.

Returns
-----
DataFrame or None
    DataFrame with the renamed axis labels or None if ``inplace=True``.

Raises
-----
KeyError
    If any of the labels is not found in the selected axis and
    "errors='raise'".

See Also

```

```

-----
DataFrame.rename_axis : Set the name of the axis.

Examples
-----
`DataFrame.rename` supports two calling conventions

* ``(index=index_mapper, columns=columns_mapper, ...)``
* ``(mapper, axis={'index', 'columns'}, ...)``
We *highly* recommend using keyword arguments to clarify your intent.

Rename columns using a mapping:

>>> df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
>>> df.rename(columns={"A": "a", "B": "c"})
   a   c
0  1  4
1  2  5
2  3  6

Rename index using a mapping:

>>> df.rename(index={0: "x", 1: "y", 2: "z"})
   A   B
x  1  4
y  2  5
z  3  6

Cast index labels to a different type:

>>> df.index
RangeIndex(start=0, stop=3, step=1)
>>> df.rename(index=str).index
Index(['0', '1', '2'], dtype='object')

>>> df.rename(columns={"A": "a", "B": "b", "C": "c"}, errors="raise")
Traceback (most recent call last):
KeyError: ['C'] not found in axis

Using axis-style parameters:

>>> df.rename(str.lower, axis='columns')
   a   b
0  1  4
1  2  5
2  3  6

>>> df.rename({1: 2, 2: 4}, axis='index')
   A   B
0  1  4
2  2  5
4  3  6

reorder_levels(self, order: 'Sequence[Axis]', axis: 'Axis' = 0) -> 'DataFrame'
    Rearrange index levels using input order. May not drop or duplicate
levels.

Parameters
-----
order : list of int or list of str
    List representing new level order. Reference level by number
    (position) or by key (label).
axis : {0 or 'index', 1 or 'columns'}, default 0
    Where to reorder levels.

Returns
-----
DataFrame

Examples
-----
>>> data = [
...     "class": ["Mammals", "Mammals", "Reptiles"],
...     "diet": ["Omnivore", "Carnivore", "Carnivore"],
...     "species": ["Humans", "Dogs", "Snakes"],
... ]
>>> df = pd.DataFrame(data, columns=["class", "diet", "species"])
>>> df.set_index(["class", "diet"])
>>> df
      species
class   diet
Mammals Omnivore      Humans
          Carnivore      Dogs
          Carnivore      Snakes

Let's reorder the levels of the index:

>>> df.reorder_levels(["diet", "class"])
      species
      class
      diet
      Omnivore Mammals      Humans
      Carnivore Mammals      Dogs
      Carnivore Reptiles      Snakes

replace(self, to_replace=None, value=<no_default>, inplace: 'bool' = False,
limit=None, regex: 'bool' = False, method: 'str | lib.Nodefault' = <no_default>
Replace values given in `to_replace` with `value`.

Values of the DataFrame are replaced with other values dynamically.

This differs from updating with ``.loc`` or ``.iloc``, which require
you to specify a location to update with some value.

Parameters
-----
to_replace : str, regex, list, dict, Series, int, float, or None
    How to find the values that will be replaced.

    * numeric, str or regex:
        - numeric: numeric values equal to `to_replace` will be
          replaced with `value`.
        - str: string exactly matching `to_replace` will be replaced
          with `value`.
        - regex: regexes matching `to_replace` will be replaced with
          `value`.

    * list of str, regex, or numeric:
        - First, if `to_replace` and `value` are both lists, they
          **must** be the same length.
        - Second, if ``regex=True`` then all of the strings in **both**
          lists will be interpreted as regexes otherwise they will match
          directly. This doesn't matter much for `value` since there
          are only a few possible substitution regexes you can use.
        - str, regex and numeric rules apply as above.

    * dict:

```

- Dicts can be used to specify different replacement values for different existing values. For example, ``{'a': 'b', 'y': 'z}'` replaces the value 'a' with 'b' and 'y' with 'z'. To use a dict in this way the 'value' parameter should be 'None'.

- For a DataFrame a dict can specify that different values should be replaced in different columns. For example, ``{'a': 1, 'b': 'z}'` looks for the value 1 in column 'a' and the value 'z' in column 'b' and replaces these values with whatever is specified in 'value'. The 'value' parameter should not be 'None' in this case. You can treat this as a special case of passing two lists except that you are specifying the column to search in.

- For a DataFrame nested dictionaries, e.g., ``{'a': {'b': np.nan}}`` , are read as follows: look in column 'a' for the value 'b' and replace it with NaN. The 'value' parameter should be 'None' to use a nested dict in this way. You can nest regular expressions as well. Note that column names (the top-level dictionary keys in a nested dictionary) **cannot** be regular expressions.

* None:

- This means that the 'regex' argument must be a string, compiled regular expression, or list, dict, ndarray or Series of such elements. If 'value' is also 'None' then this **must** be a nested dictionary or Series.

See the examples section for examples of each of these.

value : scalar, dict, list, str, regex, default None
 Value to replace any values matching 'to_replace' with.
 For a DataFrame a dict of values can be used to specify which value to use for each column (columns not in the dict will not be filled). Regular expressions, strings and lists or dicts of such objects are also allowed.

inplace : bool, default False
 If True, performs operation inplace and returns None.
limit : int, default None
 Maximum size gap to forward or backward fill.
regex : bool or same types as 'to_replace', default False
 Whether to interpret 'to_replace' and/or 'value' as regular expressions. If this is 'True' then 'to_replace' *must* be a string. Alternatively, this could be a regular expression or a list, dict, or array of regular expressions in which case 'to_replace' must be 'None'.
method : {'pad', 'ffill', 'bfill', 'None'}
 The method to use when for replacement, when 'to_replace' is a scalar, list or tuple and 'value' is 'None'.
 .. versionchanged:: 0.23.0
 Added to DataFrame.

Returns

 DataFrame
 Object after replacement.

Raises

AssertionError
 * If 'regex' is not a ``bool`` and `to_replace` is not 'None'.
TypeError
 * If 'to_replace' is not a scalar, array-like, ``dict``, or ``None``.
 * If 'to_replace' is a ``dict`` and 'value' is not a ``list``.
 * If 'to_replace' is ``None`` and 'regex' is not compilable into a regular expression or is a list, dict, ndarray, or Series.
 * When replacing multiple ``bool`` or ``datetime64`` objects and the arguments to 'to_replace' does not match the type of the value being replaced
ValueError
 * If a ``list`` or an ``ndarray`` is passed to 'to_replace' and 'value' but they are not the same length.

See Also

 DataFrame.fillna : Fill NA values.
 DataFrame.where : Replace values based on boolean condition.
 Series.str.replace : Simple string replacement.

Notes

 * Regex substitution is performed under the hood with ``re.sub``. The rules for substitution for ``re.sub`` are the same.
 * Regular expressions will only substitute on strings, meaning you cannot provide, for example, a regular expression matching floating point numbers and expect the columns in your frame that have a numeric dtype to be matched. However, if those floating point numbers *are* strings, then you can do this.
 * This method has a *lot* of options. You are encouraged to experiment and play with this method to gain intuition about how it works.
 * When dict is used as the 'to_replace' value, it is like key(s) in the dict are the to_replace part and value(s) in the dict are the value parameter.

Examples

Scalar 'to_replace' and 'value'

```
>>> s = pd.Series([1, 2, 3, 4, 5])
>>> s.replace(1, 5)
0    5
1    2
2    3
3    4
4    5
dtype: int64
```

```
>>> df = pd.DataFrame({'A': [0, 1, 2, 3, 4],
...                      'B': [5, 6, 7, 8, 9],
...                      'C': ['a', 'b', 'c', 'd', 'e']})
>>> df.replace(0, 5)
   A  B  C
0  5  5  a
1  1  6  b
2  2  7  c
3  3  8  d
4  4  9  e
```

List-like 'to_replace'

```
>>> df.replace([0, 1, 2, 3], 4)
   A  B  C
0  4  5  a
1  4  6  b
```

```

2 4 7 c
3 4 8 d
4 4 9 e

>>> df.replace([0, 1, 2, 3], [4, 3, 2, 1])
   A   B   C
0  4  5  a
1  3  6  b
2  2  7  c
3  1  8  d
4  4  9  e

>>> s.replace([1, 2], method='bfill')
0 3
1 3
2 3
3 4
4 5
dtype: int64

**dict-like `to_replace`**

>>> df.replace({0: 10, 1: 100})
   A   B   C
0 10 5 a
1 100 6 b
2 2 7 c
3 3 8 d
4 4 9 e

>>> df.replace({'A': 0, 'B': 5}, 100)
   A   B   C
0 100 100 a
1 1 6 b
2 2 7 c
3 3 8 d
4 4 9 e

>>> df.replace({'A': {0: 100, 4: 400}})
   A   B   C
0 100 5 a
1 1 6 b
2 2 7 c
3 3 8 d
4 400 9 e

**Regular expression `to_replace`**

>>> df = pd.DataFrame({'A': ['bat', 'foo', 'bait'],
...                      'B': ['abc', 'bar', 'xyz']})
>>> df.replace(to_replace=r'^ba.', value='new', regex=True)
   A   B
0  new abc
1  foo new
2  bait xyz

>>> df.replace({'A': r'^ba.'}, {'A': 'new'}, regex=True)
   A   B
0  new abc
1  foo bar
2  bait xyz

>>> df.replace(regex=r'^ba.', value='new')
   A   B
0  new abc
1  foo new
2  bait xyz

>>> df.replace(regex={r'^ba.': 'new', 'foo': 'xyz'})
   A   B
0  new abc
1  xyz new
2  bait xyz

>>> df.replace(regex=[r'^ba.', 'foo'], value='new')
   A   B
0  new abc
1  new new
2  bait xyz

Compare the behavior of ``s.replace({'a': None})`` and
``s.replace('a', None)`` to understand the peculiarities
of the `to_replace` parameter:

>>> s = pd.Series([10, 'a', 'a', 'b', 'a'])

When one uses a dict as the `to_replace` value, it is like the
value(s) in the dict are equal to the `value` parameter.
``s.replace({'a': None})`` is equivalent to
``s.replace(to_replace={'a': None}, value=None, method=None)``:

>>> s.replace({'a': None})
0    10
1    None
2    None
3     b
4     b
dtype: object

When ``value`` is not explicitly passed and `to_replace` is a scalar, list
or tuple, `replace` uses the method parameter (default 'pad') to do the
replacement. So this is why the 'a' values are being replaced by 10
in rows 1 and 2 and 'b' in row 4 in this case.

>>> s.replace('a')
0    10
1    10
2    10
3     b
4     b
dtype: object

On the other hand, if ``None`` is explicitly passed for ``value``, it will
be respected:

>>> s.replace('a', None)
0    10
1    None
2    None
3     b
4     b
dtype: object

.. versionchanged:: 1.4.0
   Previously the explicit ``None`` was silently ignored.

| resample(self, rule, axis=0, closed: 'str | None' = None, label: 'str | None'
| = None, convention: 'str' = 'start', kind: 'str | None' = None, loffset=None,
| base: 'int | None' = None, on=None, level=None, origin: 'str |
| TimestampConvertibleTypes' = 'start_day', offset: 'TimedeltaConvertibleTypes |
| None' = None) -> 'Resampler'

```

```

Resample time-series data.

Convenience method for frequency conversion and resampling of time series.
The object must have a datetime-like index ('DatetimeIndex',
'PeriodIndex', or 'TimedeltaIndex'), or the caller must pass the label of a datetime-like
series/index to the ``on`` keyword parameter.

Parameters
-----
rule : DateOffset, Timedelta or str
    The offset string or object representing target conversion.
axis : {0 or 'index', 1 or 'columns'}, default 0
    Which axis to use for up_ or down-sampling. For 'Series' this
    will default to 0, i.e. along the rows. Must be
    'DatetimeIndex', 'TimedeltaIndex' or 'PeriodIndex'.
closed : {'right', 'left'}, default None
    Which side of bin interval is closed. The default is 'left'
    for all frequency offsets except for 'M', 'A', 'Q', 'BM',
    'BA', 'BQ', and 'W' which all have a default of 'right'.
label : {'right', 'left'}, default None
    Which bin edge label to label bucket with. The default is 'left'
    for all frequency offsets except for 'M', 'A', 'Q', 'BM',
    'BA', 'BQ', and 'W' which all have a default of 'right'.
convention : {'start', 'end', 's', 'e'}, default 'start'
    For 'PeriodIndex' only, controls whether to use the start or
    end of 'rule'.
kind : {'timestamp', 'period'}, optional, default None
    Pass 'timestamp' to convert the resulting index to a
    'DateTimeIndex' or 'period' to convert it to a 'PeriodIndex'.
    By default the input representation is retained.
loffset : Timedelta, default None
    Adjust the resampled time labels.

.. deprecated:: 1.1.0
    You should add the offset to the `df.index` after the resample.
    See below.

base : int, default 0
    For frequencies that evenly subdivide 1 day, the "origin" of the
    aggregated intervals. For example, for '5min' frequency, base could
    range from 0 through 4. Defaults to 0.

.. deprecated:: 1.1.0
    The new arguments that you should use are 'offset' or 'origin'.

on : str, optional
    For a DataFrame, column to use instead of index for resampling.
    Column must be datetime-like.
level : str or int, optional
    For a MultiIndex, level (name or number) to use for
    resampling. 'level' must be datetime-like.
origin : Timestamp or str, default 'start_day'
    The timestamp on which to adjust the grouping. The timezone of origin
    must match the timezone of the index.
    If string, must be one of the following:
        - 'epoch': 'origin' is 1970-01-01
        - 'start': 'origin' is the first value of the timeseries
        - 'start_day': 'origin' is the first day at midnight of the timeseries

.. versionadded:: 1.1.0
        - 'end': 'origin' is the last value of the timeseries
        - 'end_day': 'origin' is the ceiling midnight of the last day

.. versionadded:: 1.3.0

offset : Timedelta or str, default is None
    An offset Timedelta added to the origin.

.. versionadded:: 1.1.0

Returns
-----
pandas.core.Resampler
    :class:`~pandas.core.Resampler` object.

See Also
-----
Series.resample : Resample a Series.
DataFrame.resample : Resample a DataFrame.
groupby : Group DataFrame by mapping, function, label, or list of labels.
asfreq : Reindex a DataFrame with the given frequency without grouping.

Notes
-----
See the `user guide
<https://pandas.pydata.org/pandas-docs/stable/user\_guide/timeseries.html#resampling>`__ for more.

To learn more about the offset strings, please see `this link
<https://pandas.pydata.org/pandas-docs/stable/user\_guide/timeseries.html#dateoffset-objects>`__.

Examples
-----
Start by creating a series with 9 one minute timestamps.

>>> index = pd.date_range('1/1/2000', periods=9, freq='T')
>>> series = pd.Series(range(9), index=index)
>>> series
2000-01-01 00:00:00    0
2000-01-01 00:01:00    1
2000-01-01 00:02:00    2
2000-01-01 00:03:00    3
2000-01-01 00:04:00    4
2000-01-01 00:05:00    5
2000-01-01 00:06:00    6
2000-01-01 00:07:00    7
2000-01-01 00:08:00    8
Freq: T, dtype: int64

Downsample the series into 3 minute bins and sum the values
of the timestamps falling into a bin.

>>> series.resample('3T').sum()
2000-01-01 00:00:00    3
2000-01-01 00:03:00   12
2000-01-01 00:06:00   21
Freq: 3T, dtype: int64

Downsample the series into 3 minute bins as above, but label each
bin using the right edge instead of the left. Please note that the
value in the bucket used as the label is not included in the bucket,
which it labels. For example, in the original series the
bucket ``2000-01-01 00:03:00`` contains the value 3, but the summed
value in the resampled bucket with the label ``2000-01-01 00:03:00``.
does not include 3 (if it did, the summed value would be 6, not 3).
To include this value close the right side of the bin interval as

```

```

illustrated in the example below this one.

>>> series.resample('3T', label='right').sum()
2000-01-01 00:03:00    3
2000-01-01 00:06:00   12
2000-01-01 00:09:00   21
Freq: 3T, dtype: int64

Downsample the series into 3 minute bins as above, but close the right
side of the bin interval.

>>> series.resample('3T', label='right', closed='right').sum()
2000-01-01 00:00:00    0
2000-01-01 00:03:00    6
2000-01-01 00:06:00   15
2000-01-01 00:09:00   15
Freq: 3T, dtype: int64

Upsample the series into 30 second bins.

>>> series.resample('30S').asfreq()[0:5]  # Select first 5 rows
2000-01-01 00:00:00    0.0
2000-01-01 00:00:30    NaN
2000-01-01 00:01:00    1.0
2000-01-01 00:01:30    NaN
2000-01-01 00:02:00    2.0
Freq: 30S, dtype: float64

Upsample the series into 30 second bins and fill the ``NaN`` values
using the ``pad`` method.

>>> series.resample('30S').pad()[0:5]
2000-01-01 00:00:00    0
2000-01-01 00:00:30    0
2000-01-01 00:01:00    1
2000-01-01 00:01:30    1
2000-01-01 00:02:00    2
Freq: 30S, dtype: int64

Upsample the series into 30 second bins and fill the
``NaN`` values using the ``bfill`` method.

>>> series.resample('30S').bfill()[0:5]
2000-01-01 00:00:00    0
2000-01-01 00:00:30    1
2000-01-01 00:01:00    1
2000-01-01 00:01:30    2
2000-01-01 00:02:00    2
Freq: 30S, dtype: int64

Pass a custom function via ``apply``

>>> def custom_resampler(arraylike):
...     return np.sum(arraylike) + 5
...
>>> series.resample('3T').apply(custom_resampler)
2000-01-01 00:00:00    8
2000-01-01 00:03:00   17
2000-01-01 00:06:00   26
Freq: 3T, dtype: int64

For a Series with a PeriodIndex, the keyword `convention` can be
used to control whether to use the start or end of `rule`.

Resample a year by quarter using 'start' `convention`. Values are
assigned to the first quarter of the period.

>>> s = pd.Series([1, 2], index=pd.period_range('2012-01-01',
...                                              freq='A',
...                                              periods=2))
>>> s
2012    1
2013    2
Freq: A-DEC, dtype: int64
>>> s.resample('Q', convention='start').asfreq()
2012Q1    1.0
2012Q2    NaN
2012Q3    NaN
2012Q4    NaN
2013Q1    2.0
2013Q2    NaN
2013Q3    NaN
2013Q4    NaN
Freq: Q-DEC, dtype: float64

Resample quarters by month using 'end' `convention`. Values are
assigned to the last month of the period.

>>> q = pd.Series([1, 2, 3, 4], index=pd.period_range('2018-01-01',
...                                              freq='Q',
...                                              periods=4))
>>> q
2018Q1    1
2018Q2    2
2018Q3    3
2018Q4    4
Freq: Q-DEC, dtype: int64
>>> q.resample('M', convention='end').asfreq()
2018-03    1.0
2018-04    NaN
2018-05    NaN
2018-06    2.0
2018-07    NaN
2018-08    NaN
2018-09    3.0
2018-10    NaN
2018-11    NaN
2018-12    4.0
Freq: M, dtype: float64

For DataFrame objects, the keyword `on` can be used to specify the
column instead of the index for resampling.

>>> d = {'price': [10, 11, 9, 13, 14, 18, 17, 19],
...        'volume': [50, 60, 40, 100, 50, 100, 40, 50]}
>>> df = pd.DataFrame(d)
>>> df['week_starting'] = pd.date_range('01/01/2018',
...                                         periods=8,
...                                         freq='W')
>>> df
   price  volume week_starting
0      10      50  2018-01-07
1      11      60  2018-01-14
2       9      40  2018-01-21
3     13     100  2018-01-28
4     14      50  2018-02-04
5     18     100  2018-02-11
6     17      40  2018-02-18
7     19      50  2018-02-25
>>> df.resample('M', on='week_starting').mean()
           price    volume
week_starting
2018-01-07    10.0     50.0
2018-01-14    11.0     60.0
2018-01-21     9.0     40.0
2018-01-28    13.0    100.0
2018-02-04    14.0     50.0
2018-02-11    18.0    100.0
2018-02-18    17.0     40.0
2018-02-25    19.0     50.0

```

```

week_starting
2018-01-31    10.75   62.5
2018-02-28    17.00   60.0

For a DataFrame with MultiIndex, the keyword `level` can be used to
specify on which level the resampling needs to take place.

>>> days = pd.date_range('1/1/2000', periods=4, freq='D')
>>> d2 = {'price': [10, 11, 9, 13, 14, 18, 17, 19],
...         'volume': [50, 60, 40, 100, 50, 100, 40, 50]}
>>> df2 = pd.DataFrame(
...     d2,
...     index=pd.MultiIndex.from_product(
...         [days, ['morning', 'afternoon']],
...         )
... )
>>> df2
      price  volume
2000-01-01 morning    10     50
                  afternoon   11     60
2000-01-02 morning    13     40
                  afternoon   14    100
2000-01-03 morning    14     50
                  afternoon   18     100
2000-01-04 morning    17     40
                  afternoon   19     50
>>> df2.resample('D', level=0).sum()
      price  volume
2000-01-01    21    110
2000-01-02    22    140
2000-01-03    32    150
2000-01-04    36    90

If you want to adjust the start of the bins based on a fixed timestamp:

>>> start, end = '2000-10-01 23:30:00', '2000-10-02 00:30:00'
>>> rng = pd.date_range(start, end, freq='7min')
>>> ts = pd.Series(np.arange(len(rng)) * 3, index=rng)
>>> ts
2000-10-01 23:30:00    0
2000-10-01 23:37:00    3
2000-10-01 23:44:00    6
2000-10-01 23:51:00    9
2000-10-01 23:58:00   12
2000-10-02 00:05:00   15
2000-10-02 00:12:00   18
2000-10-02 00:19:00   21
2000-10-02 00:26:00   24
Freq: T, dtype: int64

>>> ts.resample('17min').sum()
2000-10-01 23:14:00    0
2000-10-01 23:31:00    9
2000-10-01 23:48:00   21
2000-10-02 00:05:00   54
2000-10-02 00:22:00   24
Freq: 17T, dtype: int64

>>> ts.resample('17min', origin='epoch').sum()
2000-10-01 23:18:00    0
2000-10-01 23:35:00   18
2000-10-01 23:52:00   27
2000-10-02 00:09:00   39
2000-10-02 00:26:00   24
Freq: 17T, dtype: int64

>>> ts.resample('17min', origin='2000-01-01').sum()
2000-10-01 23:24:00    3
2000-10-01 23:41:00   15
2000-10-01 23:58:00   45
2000-10-02 00:15:00   45
Freq: 17T, dtype: int64

If you want to adjust the start of the bins with an `offset` Timedelta,
the two
following lines are equivalent:

>>> ts.resample('17min', origin='start').sum()
2000-10-01 23:30:00    9
2000-10-01 23:47:00   21
2000-10-02 00:04:00   54
2000-10-02 00:21:00   24
Freq: 17T, dtype: int64

>>> ts.resample('17min', offset='23h30min').sum()
2000-10-01 23:30:00    9
2000-10-01 23:47:00   21
2000-10-02 00:04:00   54
2000-10-02 00:21:00   24
Freq: 17T, dtype: int64

If you want to take the largest Timestamp as the end of the bins:

>>> ts.resample('17min', origin='end').sum()
2000-10-01 23:35:00    0
2000-10-01 23:52:00   18
2000-10-02 00:09:00   27
2000-10-02 00:26:00   63
Freq: 17T, dtype: int64

In contrast with the `start_day`, you can use `end_day` to take the
ceiling
midnight of the largest Timestamp as the end of the bins and drop the bins
not containing data:

>>> ts.resample('17min', origin='end_day').sum()
2000-10-01 23:38:00    3
2000-10-01 23:55:00   15
2000-10-02 00:12:00   45
2000-10-02 00:29:00   45
Freq: 17T, dtype: int64

To replace the use of the deprecated `base` argument, you can now use
`offset`,
in this example it is equivalent to have `base=2`:

>>> ts.resample('17min', offset='2min').sum()
2000-10-01 23:16:00    0
2000-10-01 23:33:00    9
2000-10-01 23:50:00   36
2000-10-02 00:07:00   39
2000-10-02 00:24:00   24
Freq: 17T, dtype: int64

To replace the use of the deprecated `loffset` argument:

>>> from pandas.tseries.frequencies import to_offset
>>> loffset = '19min'
>>> ts_out = ts.resample('17min').sum()
>>> ts_out.index = ts_out.index + to_offset(loffset)
>>> ts_out

```

```

2000-10-01 23:33:00      0
2000-10-01 23:50:00      9
2000-10-02 00:07:00     21
2000-10-02 00:24:00     54
2000-10-02 00:41:00     24
Freq: 17T, dtype: int64

| reset_index(self, level: 'Hashable | Sequence[Hashable] | None' = None, drop:
|   'bool' = False, inplace: 'bool' = False, col_level: 'Hashable' = 0, col_fill:
|   'Hashable' = '') -> 'DataFrame | None'
|   Reset the index, or a level of it.
|
|   Reset the index of the DataFrame, and use the default one instead.
|   If the DataFrame has a MultiIndex, this method can remove one or more
|   levels.
|
| Parameters
| -----
|   level : int, str, tuple, or list, default None
|       Only remove the given levels from the index. Removes all levels by
|       default.
|   drop : bool, default False
|       Do not try to insert index into dataframe columns. This resets
|       the index to the default integer index.
|   inplace : bool, default False
|       Modify the DataFrame in place (do not create a new object).
|   col_level : int or str, default 0
|       If the columns have multiple levels, determines which level the
|       labels are inserted into. By default it is inserted into the first
|       level.
|   col_fill : object, default ''
|       If the columns have multiple levels, determines how the other
|       levels are named. If None then the index name is repeated.
|
| Returns
| -----
|   DataFrame or None
|       DataFrame with the new index or None if ``inplace=True``.
|
| See Also
| -----
|   DataFrame.set_index : Opposite of reset_index.
|   DataFrame.reindex : Change to new indices or expand indices.
|   DataFrame.reindex_like : Change to same indices as other DataFrame.
|
| Examples
| -----
>>> df = pd.DataFrame([('bird', 389.0),
...                      ('bird', 24.0),
...                      ('mammal', 80.5),
...                      ('mammal', np.nan)],
...                     index=['falcon', 'parrot', 'lion', 'monkey'],
...                     columns=('class', 'max_speed'))
>>> df
   class  max_speed
falcon    bird    389.0
parrot    bird     24.0
lion     mammal    80.5
monkey    mammal     NaN

When we reset the index, the old index is added as a column, and a
new sequential index is used:

>>> df.reset_index()
   index  class  max_speed
0  falcon    bird    389.0
1  parrot    bird     24.0
2    lion  mammal    80.5
3  monkey  mammal     NaN

We can use the `drop` parameter to avoid the old index being added as
a column:

>>> df.reset_index(drop=True)
   class  max_speed
0    bird    389.0
1    bird     24.0
2  mammal    80.5
3  mammal     NaN

You can also use `reset_index` with `MultiIndex`.

>>> index = pd.MultiIndex.from_tuples([('bird', 'falcon'),
...                                      ('bird', 'parrot'),
...                                      ('mammal', 'lion'),
...                                      ('mammal', 'monkey')],
...                                     names=['class', 'name'])
>>> columns = pd.MultiIndex.from_tuples([('speed', 'max'),
...                                         ('species', 'type')])
>>> df = pd.DataFrame([(389.0, 'fly'),
...                      (24.0, 'fly'),
...                      (80.5, 'run'),
...                      (np.nan, 'jump')],
...                     index=index,
...                     columns=columns)
>>> df
   speed species
      max   type
class name
bird  falcon  389.0   fly
        parrot   24.0   fly
mammal  lion   80.5   run
        monkey    NaN  jump

If the index has multiple levels, we can reset a subset of them:

>>> df.reset_index(level='class')
   class  speed species
      max   type
name
falcon    bird  389.0   fly
parrot    bird   24.0   fly
lion     mammal   80.5   run
monkey    mammal    NaN  jump

If we are not dropping the index, by default, it is placed in the top
level. We can place it in another level:

>>> df.reset_index(level='class', col_level=1)
   speed species
   class  max   type
name
falcon    bird  389.0   fly
parrot    bird   24.0   fly
lion     mammal   80.5   run
monkey    mammal    NaN  jump

When the index is inserted under another level, we can specify under
which one with the parameter `col_fill`:

>>> df.reset_index(level='class', col_level=1, col_fill='species')

```

```

          species  speed species
              class    max   type
name
falcon      bird  389.0    fly
parrot      bird   24.0    fly
lion        mammal  80.5   run
monkey      mammal    NaN  jump

If we specify a nonexistent level for 'col_fill', it is created:
>>> df.reset_index(level='class', col_level=1, col_fill='genus')
          genus  speed species
              class    max   type
name
falcon      bird  389.0    fly
parrot      bird   24.0    fly
lion        mammal  80.5   run
monkey      mammal    NaN  jump

rfloordiv(self, other, axis='columns', level=None, fill_value=None)
| Get Integer division of dataframe and other, element-wise (binary operator
| rfloordiv').
|
| Equivalent to ``other // dataframe``, but with support to substitute a
fill_value
| for missing data in one of the inputs. With reverse version, `floordiv`.
|
| Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
| arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.
|
Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful DataFrame alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                     index=['circle', 'triangle', 'rectangle'])
>>> df
   angles  degrees
circle      0      360
triangle    3      180
rectangle   4      360

Add a scalar with operator version which return the same
results.

>>> df + 1
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

>>> df.add(1)
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

Divide by constant with reverse version.

>>> df.div(10)
   angles  degrees
circle     0.0      36.0
triangle   0.3      18.0
rectangle  0.4      36.0

>>> df.rdiv(10)
   angles  degrees
circle     inf     0.027778
triangle  3.33333  0.055556
rectangle 2.500000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
   angles  degrees
circle     -1      358
triangle    2      178
rectangle   3      358

>>> df.sub([1, 2], axis='columns')
   angles  degrees
circle     -1      358
triangle    2      178
rectangle   3      358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
           axis='index')
   angles  degrees
circle     -1      359
triangle    2      179
rectangle   3      359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4]},
```

```

...                                         index=['circle', 'triangle', 'rectangle'])

>>> other
      angles
circle     0
triangle   3
rectangle  4

>>> df * other
      angles  degrees
circle      0      NaN
triangle    9      NaN
rectangle  16      NaN

>>> df.mul(other, fill_value=0)
      angles  degrees
circle      0      0.0
triangle    9      0.0
rectangle  16      0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
...                                index=[['A', 'A', 'B', 'B', 'B'],
...                                       ['circle', 'triangle', 'rectangle',
...                                        'square', 'pentagon', 'hexagon']])
...
>>> df_multindex
      angles  degrees
A circle     0     360
triangle    3     180
rectangle   4     360
B square     4     360
pentagon    5     540
hexagon     6     720

>>> df.div(df_multindex, level=1, fill_value=0)
      angles  degrees
A circle     NaN     1.0
triangle   1.0     1.0
rectangle  1.0     1.0
B square     0.0     0.0
pentagon    0.0     0.0
hexagon     0.0     0.0

rmod(self, other, axis='columns', level=None, fill_value=None)
Get Modulo of dataframe and other, element-wise (binary operator `rmod`).

Equivalent to ``other % dataframe``', but with support to substitute a
fill_value
for missing data in one of the inputs. With reverse version, `mod`.

Among flexible wrappers (`add`, `sub`, `mul`, `div`, `mod`, `pow`) to
arithmetic operators: `+`, `-`, `*`, `/`, `//`, `%`, `**`.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful DataFrame alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                     index=['circle', 'triangle', 'rectangle'])
...
>>> df
      angles  degrees
circle     0     360
triangle   3     180
rectangle  4     360

Add a scalar with operator version which return the same
results.

>>> df + 1
      angles  degrees
circle     1     361
triangle   4     181
rectangle  5     361

>>> df.add(1)
      angles  degrees
circle     1     361
triangle   4     181
rectangle  5     361

Divide by constant with reverse version.

>>> df.div(10)
      angles  degrees
circle     0.0    36.0
triangle   0.3    18.0
rectangle  0.4    36.0

>>> df.rdiv(10)
      angles  degrees
circle     inf    0.027778
triangle  3.33333  0.055556
rectangle 2.50000  0.027778

```

```

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
          angles  degrees
circle      -1     358
triangle     2     178
rectangle    3     358

>>> df.sub([1, 2], axis='columns')
          angles  degrees
circle      -1     358
triangle     2     178
rectangle    3     358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
          axis='index')
          angles  degrees
circle      -1     359
triangle     2     179
rectangle    3     359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4]}, ...
           index=['circle', 'triangle', 'rectangle'])

>>> other
          angles
circle      0
triangle    3
rectangle   4

>>> df * other
          angles  degrees
circle      0      NaN
triangle    9      NaN
rectangle   16     NaN

>>> df.mul(other, fill_value=0)
          angles  degrees
circle      0      0.0
triangle    9      0.0
rectangle   16     0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
           ...                               index=[['A', 'A', 'A', 'B', 'B'],
...                                         ['circle', 'triangle', 'rectangle',
...                                          'square', 'pentagon', 'hexagon']])
>>> df_multindex
          angles  degrees
A circle      0     360
triangle      3     180
rectangle     4     360
B square      4     360
pentagon      5     540
hexagon       6     720

>>> df.div(df_multindex, level=1, fill_value=0)
          angles  degrees
A circle      NaN     1.0
triangle     1.0     1.0
rectangle    1.0     1.0
B square      0.0     0.0
pentagon     0.0     0.0
hexagon      0.0     0.0

rmul(self, other, axis='columns', level=None, fill_value=None)
    Get Multiplication of dataframe and other, element-wise (binary operator
`rmul`).

    Equivalent to ``other * dataframe`` , but with support to substitute a
fill_value
        for missing data in one of the inputs. With reverse version, `mul`.

    Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful DataFrame alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]}, ...
...                     index=['circle', 'triangle', 'rectangle'])
>>> df
          angles  degrees
circle      0     360
triangle    3     180
rectangle   4     360

Add a scalar with operator version which return the same
results.

```

```

>>> df + 1
      angles  degrees
circle      1     361
triangle    4     181
rectangle   5     361

>>> df.add(1)
      angles  degrees
circle      1     361
triangle    4     181
rectangle   5     361

Divide by constant with reverse version.

>>> df.div(10)
      angles  degrees
circle      0.0    36.0
triangle    0.3    18.0
rectangle   0.4    36.0

>>> df.rdiv(10)
      angles  degrees
circle      inf    0.027778
triangle   3.333333 0.055556
rectangle  2.500000 0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
      angles  degrees
circle      -1    358
triangle    2     178
rectangle   3     358

>>> df.sub([1, 2], axis='columns')
      angles  degrees
circle      -1    358
triangle    2     178
rectangle   3     358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
        ...      axis='index')
      angles  degrees
circle      -1    359
triangle    2     179
rectangle   3     359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4]}, ...
        ...           index=['circle', 'triangle', 'rectangle'])
>>> other
      angles
circle      0
triangle    3
rectangle   4

>>> df * other
      angles  degrees
circle      0      NaN
triangle    9      NaN
rectangle  16      NaN

>>> df.mul(other, fill_value=0)
      angles  degrees
circle      0      0.0
triangle    9      0.0
rectangle  16      0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
        ...           index=[['A', 'A', 'A', 'B', 'B'],
...                           ['circle', 'triangle', 'rectangle',
...                            'square', 'pentagon', 'hexagon']])
>>> df_multindex
      angles  degrees
A circle      0    360
triangle      3    180
rectangle     4    360
B square      4    360
pentagon      5    540
hexagon       6    720

>>> df.div(df_multindex, level=1, fill_value=0)
      angles  degrees
A circle    NaN    1.0
triangle   1.0    1.0
rectangle  1.0    1.0
B square    0.0    0.0
pentagon   0.0    0.0
hexagon    0.0    0.0

round(self, decimals: int | dict[InputLabel, int] | Series = 0, *args,
**kwargs) -> 'DataFrame'
Round a DataFrame to a variable number of decimal places.

Parameters
-----
decimals : int, dict, Series
    Number of decimal places to round each column to. If an int is
    given, round each column to the same number of places.
    Otherwise dict and Series round to variable numbers of places.
    Column names should be in the keys if 'decimals' is a
    dict-like, or in the index if 'decimals' is a Series. Any
    columns not included in 'decimals' will be left as is. Elements
    of 'decimals' which are not columns of the input will be
    ignored.
*args
    Additional keywords have no effect but might be accepted for
    compatibility with numpy.
**kwargs
    Additional keywords have no effect but might be accepted for
    compatibility with numpy.

Returns
-----
DataFrame
    A DataFrame with the affected columns rounded to the specified
    number of decimal places.

See Also
-----
numpy.around : Round a numpy array to the given number of decimals.
Series.round : Round a Series to the given number of decimals.

Examples
-----

```

```

>>> df = pd.DataFrame([(0.21, .32), (.01, .67), (.66, .03), (.21, .18)], columns=['dogs', 'cats'])
...
>>> df
   dogs  cats
0  0.21  0.32
1  0.01  0.67
2  0.66  0.03
3  0.21  0.18

By providing an integer each column is rounded to the same number
of decimal places

>>> df.round(1)
   dogs  cats
0  0.2   0.3
1  0.0   0.7
2  0.7   0.0
3  0.2   0.2

With a dict, the number of places for specific columns can be
specified with the column names as key and the number of decimal
places as value

>>> df.round({'dogs': 1, 'cats': 0})
   dogs  cats
0  0.2   0.0
1  0.0   1.0
2  0.7   0.0
3  0.2   0.0

Using a Series, the number of places for specific columns can be
specified with the column names as index and the number of
decimal places as value

>>> decimals = pd.Series([0, 1], index=['cats', 'dogs'])
>>> df.round(decimals)
   dogs  cats
0  0.2   0.0
1  0.0   1.0
2  0.7   0.0
3  0.2   0.0

rpow(self, other, axis='columns', level=None, fill_value=None)
Get Exponential power of dataframe and other, element-wise (binary
operator 'rpow').
Equivalent to ``other ** dataframe``, but with support to substitute a
fill_value
for missing data in one of the inputs. With reverse version, 'pow'.
Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful DataFrame alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]}, index=['circle', 'triangle', 'rectangle'])
>>> df
   angles  degrees
circle      0     360
triangle    3     180
rectangle   4     360

Add a scalar with operator version which return the same
results.

>>> df + 1
   angles  degrees
circle      1     361
triangle    4     181
rectangle   5     361

>>> df.add(1)
   angles  degrees
circle      1     361
triangle    4     181
rectangle   5     361

Divide by constant with reverse version.

>>> df.div(10)
   angles  degrees
circle      0.0    36.0
triangle    0.3    18.0
rectangle   0.4    36.0

>>> df.rdiv(10)
   angles  degrees
circle      inf   0.027778
triangle   3.333333  0.055556
rectangle  2.500000  0.027778

Subtract a list and Series with axis with operator version.

```

```

>>> df = [1, 2]
      angles  degrees
circle      -1     358
triangle     2     178
rectangle    3     358

>>> df.sub([1, 2], axis='columns')
      angles  degrees
circle      -1     358
triangle     2     178
rectangle    3     358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
      ...      axis='index')
      angles  degrees
circle      -1     359
triangle     2     179
rectangle    3     359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4],
...                         index=['circle', 'triangle', 'rectangle'])

>>> other
      angles
circle      0
triangle    3
rectangle   4

>>> df * other
      angles  degrees
circle      0      NaN
triangle    9      NaN
rectangle  16      NaN

>>> df.mul(other, fill_value=0)
      angles  degrees
circle      0      0.0
triangle    9      0.0
rectangle  16      0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
      ...      index=[['A', 'A', 'A', 'B', 'B', 'B'],
...                  ['circle', 'triangle', 'rectangle',
'square', 'pentagon', 'hexagon']])
>>> df_multindex
      angles  degrees
A circle      0     360
triangle     3     180
rectangle    4     360
B square      4     360
pentagon     5     540
hexagon      6     720

>>> df.div(df_multindex, level=1, fill_value=0)
      angles  degrees
A circle      NaN     1.0
triangle    1.0     1.0
rectangle   1.0     1.0
B square      0.0     0.0
pentagon    0.0     0.0
hexagon     0.0     0.0

rsub(self, other, axis='columns', level=None, fill_value=None)
      Get Subtraction of datafram and other, element-wise (binary operator
'rsub').

      Equivalent to ``other - datafram``, but with support to substitute a
fill_value
      for missing data in one of the inputs. With reverse version, `sub`.

      Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
      Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
      Whether to compare by the index (0 or 'index') or columns
      (1 or 'columns'). For Series input, axis to match Series index on.
level : int or Label
      Broadcast across a level, matching Index values on the
      passed MultiIndex level.
fill_value : float or None, default None
      Fill existing missing (NaN) values, and any new element needed for
      successful Dataframe alignment, with this value before computation.
      If data in both corresponding Dataframe locations is missing
      the result will be missing.

Returns
-----
DataFrame
      Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                      index=['circle', 'triangle', 'rectangle'])
>>> df
      angles  degrees
circle      0     360
triangle    3     180
rectangle   4     360

Add a scalar with operator version which return the same
results.

>>> df + 1
      angles  degrees

```

```

circle      1      361
triangle   4      181
rectangle  5      361

>>> df.add(1)
          angles  degrees
circle      1      361
triangle   4      181
rectangle  5      361

Divide by constant with reverse version.

>>> df.div(10)
          angles  degrees
circle      0.0     36.0
triangle   0.3     18.0
rectangle  0.4     36.0

>>> df.rdiv(10)
          angles  degrees
circle      inf    0.02778
triangle  3.33333  0.05556
rectangle 2.50000  0.02778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
          angles  degrees
circle      -1     358
triangle    2      178
rectangle   3      358

>>> df.sub([1, 2], axis='columns')
          angles  degrees
circle      -1     358
triangle    2      178
rectangle   3      358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
          axis='index')
          angles  degrees
circle      -1     359
triangle    2      179
rectangle   3      359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4]}, ...
...                         index=['circle', 'triangle', 'rectangle'])
>>> other
          angles
circle      0
triangle   3
rectangle  4

>>> df * other
          angles  degrees
circle      0      NaN
triangle   9      NaN
rectangle  16     0.0

>>> df.mul(other, fill_value=0)
          angles  degrees
circle      0      0.0
triangle   9      0.0
rectangle  16     0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
...                               index=[['A', 'A', 'A', 'B', 'B'],
...                                     ['circle', 'triangle', 'rectangle',
...                                      'square', 'pentagon', 'hexagon']])
>>> df_multindex
          angles  degrees
A circle      0     360
triangle     3     180
rectangle    4     360
B square      4     360
pentagon     5     540
hexagon      6     720

>>> df.div(df_multindex, level=1, fill_value=0)
          angles  degrees
A circle    NaN     1.0
triangle   1.0     1.0
rectangle  1.0     1.0
B square    0.0     0.0
pentagon   0.0     0.0
hexagon    0.0     0.0

rtruediv(self, other, axis='columns', level=None, fill_value=None)
    Get Floating division of dataframe and other, element-wise (binary
operator `rtruediv`).

    Equivalent to ``other / dataframe``, but with support to substitute a
fill_value
    for missing data in one of the inputs. With reverse version, `truediv`.

Among flexible wrappers (`add`, `sub`, `mul`, `div`, `mod`, `pow`) to
arithmetic operators: `+`, `-`, `*`, `/`, `//`, `%`, `**`.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful Dataframe alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
Dataframe.add : Add DataFrames.
Dataframe.sub : Subtract DataFrames.
Dataframe.mul : Multiply DataFrames.

```

```

DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                     index=['circle', 'triangle', 'rectangle'])
>>> df
   angles  degrees
circle      0     360
triangle    3     180
rectangle   4     360

Add a scalar with operator version which return the same
results.

>>> df + 1
   angles  degrees
circle      1     361
triangle    4     181
rectangle   5     361

>>> df.add(1)
   angles  degrees
circle      1     361
triangle    4     181
rectangle   5     361

Divide by constant with reverse version.

>>> df.div(10)
   angles  degrees
circle      0.0    36.0
triangle    0.3    18.0
rectangle   0.4    36.0

>>> df.rdiv(10)
   angles  degrees
circle      inf   0.027778
triangle   3.333333  0.055556
rectangle  2.500000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
   angles  degrees
circle      -1    358
triangle    2     178
rectangle   3     358

>>> df.sub([1, 2], axis='columns')
   angles  degrees
circle      -1    358
triangle    2     178
rectangle   3     358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
          axis='index')
   angles  degrees
circle      -1    359
triangle    2     179
rectangle   3     359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4],
...                         'index':['circle', 'triangle', 'rectangle'],
...                         'other_angles': [0, 3, 4],
...                         'other_degrees': [360, 180, 360],
...                         'other_index': ['circle', 'triangle', 'rectangle']})
>>> other
   angles
circle      0
triangle    3
rectangle   4

>>> df * other
   angles  degrees
circle      0     NaN
triangle    9     NaN
rectangle  16     NaN

>>> df.mul(other, fill_value=0)
   angles  degrees
circle      0     0.0
triangle    9     0.0
rectangle  16     0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                 'degrees': [360, 180, 360, 360, 540,
720]}, ...
...                                 index=[['A', 'A', 'A', 'B', 'B'],
...                                        ['circle', 'triangle', 'rectangle',
...                                         'square', 'pentagon', 'hexagon']])
>>> df_multindex
   angles  degrees
A circle      0     360
triangle      3     180
rectangle     4     360
B square      4     360
pentagon      5     540
hexagon       6     720

>>> df.div(df_multindex, level=1, fill_value=0)
   angles  degrees
A circle      NaN     1.0
triangle    1.0     1.0
rectangle   1.0     1.0
B square      0.0     0.0
pentagon     0.0     0.0
hexagon      0.0     0.0

select_dtypes(self, include=None, exclude=None) -> 'DataFrame'
Return a subset of the DataFrame's columns based on the column dtypes.

Parameters
-----
include, exclude : scalar or list-like
    A selection of dtypes or strings to be included/excluded. At least
    one of these parameters must be supplied.

Returns
-----

```

```

DataFrame
The subset of the frame including the dtypes in ``include`` and
excluding the dtypes in ``exclude``.

Raises
-----
ValueError
    * If both of ``include`` and ``exclude`` are empty
    * If ``include`` and ``exclude`` have overlapping elements
    * If any kind of string dtype is passed in.

See Also
-----
DataFrame.dtypes: Return Series with the data type of each column.

Notes
-----
* To select all *numeric* types, use ``np.number`` or ``'number'``
* To select strings you must use the ``object`` dtype, but note that
this will return *all* object dtype columns
* See the `numpy dtype hierarchy`_
<https://numpy.org/doc/stable/reference/arrays.scalars.html>_
* To select datetimes, use ``np.datetime64``, ``'datetime'`` or
``'datetime64'``
* To select timedeltas, use ``np.timedelta64``, ``'timedelta'`` or
``'timedelta64'``
* To select Pandas categorical dtypes, use ``'category'``
* To select Pandas datetimetz dtypes, use ``'datetimetz'`` (new in
0.20.0) or ``'datetime64[ns, tz]'``

Examples
-----
>>> df = pd.DataFrame({'a': [1, 2] * 3,
...                      'b': [True, False] * 3,
...                      'c': [1.0, 2.0] * 3})
>>> df
   a    b    c
0  1  True  1.0
1  2 False  2.0
2  1  True  1.0
3  2 False  2.0
4  1  True  1.0
5  2 False  2.0

>>> df.select_dtypes(include='bool')
   b
0  True
1 False
2  True
3 False
4  True
5 False

>>> df.select_dtypes(include=['float64'])
   c
0  1.0
1  2.0
2  1.0
3  2.0
4  1.0
5  2.0

>>> df.select_dtypes(exclude=['int64'])
   b    c
0  True  1.0
1 False  2.0
2  True  1.0
3 False  2.0
4  True  1.0
5 False  2.0

sem(self, axis=None, skipna=True, level=None, ddof=1, numeric_only=None,
**kwargs)
Return unbiased standard error of the mean over requested axis.

Normalized by N-1 by default. This can be changed using the ddof argument

Parameters
-----
axis : {index (0), columns (1)}
skipna : bool, default True
    Exclude NA/null values. If an entire row/column is NA, the result
    will be NA.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
ddof : int, default 1
    Delta Degrees of Freedom. The divisor used in calculations is N -
ddof,
    where N represents the number of elements.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.

Returns
-----
Series or DataFrame (if level specified)

set_axis(self, labels, axis: 'Axis' = 0, inplace: 'bool' = False)
Assign desired index to given axis.

Indexes for column or row labels can be changed by assigning
a list-like or Index.

Parameters
-----
labels : list-like, Index
    The values for the new index.

axis : {0 or 'index', 1 or 'columns'}, default 0
    The axis to update. The value 0 identifies the rows, and 1 identifies
the columns.

inplace : bool, default False
    Whether to return a new DataFrame instance.

Returns
-----
renamed : DataFrame or None
    An object of type DataFrame or None if ``inplace=True``.

See Also
-----
DataFrame.rename_axis : Alter the name of the index or columns.

Examples
-----
>>> df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})

Change the row labels.

```

```

>>> df.set_axis(['a', 'b', 'c'], axis='index')
   A   B
   a  1  4
   b  2  5
   c  3  6

   Change the column labels.

>>> df.set_axis(['I', 'II'], axis='columns')
   I   II
   0  1  4
   1  2  5
   2  3  6

   Now, update the labels inplace.

>>> df.set_axis(['i', 'ii'], axis='columns', inplace=True)
   i   ii
   0  1  4
   1  2  5
   2  3  6

set_index(self, keys, drop: 'bool' = True, append: 'bool' = False, inplace:
'bool' = False, verify_integrity: 'bool' = False)
   Set the DataFrame index using existing columns.

Set the DataFrame index (row labels) using one or more existing
columns or arrays (of the correct length). The index can replace the
existing index or expand on it.

Parameters
-----
keys : label or array-like or list of labels/arrays
   This parameter can be either a single column key, a single array of
   the same length as the calling DataFrame, or a list containing an
   arbitrary combination of column keys and arrays. Here, "array"
   encompasses :class:`Series`, :class:`Index`, ``np.ndarray``, and
   instances of :class:`~collections.abc.Iterator`.
drop : bool, default True
   Delete columns to be used as the new index.
append : bool, default False
   Whether to append columns to existing index.
inplace : bool, default False
   If True, modifies the DataFrame in place (do not create a new object).
verify_integrity : bool, default False
   Check the new index for duplicates. Otherwise defer the check until
   necessary. Setting to False will improve the performance of this
   method.

Returns
-----
DataFrame or None
   Changed row labels or None if ``inplace=True``.

See Also
-----
DataFrame.reset_index : Opposite of set_index.
DataFrame.reindex : Change to new indices or expand indices.
DataFrame.reindex_like : Change to same indices as other DataFrame.

Examples
-----
>>> df = pd.DataFrame({'month': [1, 4, 7, 10],
...                      'year': [2012, 2014, 2013, 2014],
...                      'sale': [55, 40, 84, 31]})

   month  year  sale
0      1  2012    55
1      4  2014    40
2      7  2013    84
3     10  2014    31

   Set the index to become the 'month' column:

>>> df.set_index('month')
   year  sale
month
1    2012    55
4    2014    40
7    2013    84
10   2014    31

   Create a MultiIndex using columns 'year' and 'month':

>>> df.set_index(['year', 'month'])
   sale
year month
2012 1    55
2014 4    40
2013 7    84
2014 10   31

   Create a MultiIndex using an Index and a column:

>>> df.set_index([pd.Index([1, 2, 3, 4]), 'year'])
   month  sale
year
1  2012  1    55
2  2014  4    40
3  2013  7    84
4  2014  10   31

   Create a MultiIndex using two Series:

>>> s = pd.Series([1, 2, 3, 4])
>>> df.set_index([s, s**2])
   month  year  sale
1 1      1  2012    55
2 4      4  2014    40
3 9      7  2013    84
4 16     10 2014    31

shift(self, periods=1, freq: 'Frequency | None' = None, axis: 'Axis' = 0,
fill_value=<no_default>) -> 'DataFrame'
   Shift index by desired number of periods with an optional time 'freq'.

When 'freq' is not passed, shift the index without realigning the data.
If 'freq' is passed (in this case, the index must be date or datetime,
or it will raise a 'NotImplementedError'), the index will be
increased using the periods and the 'freq'. 'freq' can be inferred
when specified as "infer" as long as either freq or inferred_freq
attribute is set in the index.

Parameters
-----
periods : int
   Number of periods to shift. Can be positive or negative.
freq : DateOffset, tseries.offsets, timedelta, or str, optional
   Offset to use from the tseries module or time rule (e.g. 'EOM').
   If 'freq' is specified then the index values are shifted but the

```

```

    data is not realigned. That is, use `freq` if you would like to
    extend the index when shifting and preserve the original data.
    If `freq` is specified as "infer" then it will be inferred from
    the freq or inferred_freq attributes of the index. If neither of
    those attributes exist, a ValueError is thrown.
    axis : {0 or 'index', 1 or 'columns', None}, default None
        Shift direction.
    fill_value : object, optional
        The scalar value to use for newly introduced missing values.
        the default depends on the dtype of `self`.
        For numeric data, ``np.nan`` is used.
        For datetime, timedelta, or period data, etc. :attr:`NaT` is used.
        For extension dtypes, ``self.dtype.na_value`` is used.
    .. versionchanged:: 1.1.0

    Returns
    ------
    DataFrame
        Copy of input object, shifted.

    See Also
    -----
    Index.shift : Shift values of Index.
    DatetimeIndex.shift : Shift values of DatetimeIndex.
    PeriodIndex.shift : Shift values of PeriodIndex.
    tshift : Shift the time index, using the index's frequency if
        available.

    Examples
    -----
    >>> df = pd.DataFrame({"Col1": [10, 20, 15, 30, 45],
   ...                      "Col2": [13, 23, 18, 33, 48],
   ...                      "Col3": [17, 27, 22, 37, 52]},
   ...                      index=pd.date_range("2020-01-01", "2020-01-05"))
    >>> df
              Col1  Col2  Col3
2020-01-01    10    13    17
2020-01-02    20    23    27
2020-01-03    15    18    22
2020-01-04    30    33    37
2020-01-05    45    48    52

    >>> df.shift(periods=3)
              Col1  Col2  Col3
2020-01-01    NaN    NaN    NaN
2020-01-02    NaN    NaN    NaN
2020-01-03    NaN    NaN    NaN
2020-01-04  10.0  13.0  17.0
2020-01-05  20.0  23.0  27.0

    >>> df.shift(periods=1, axis="columns")
              Col1  Col2  Col3
2020-01-01    NaN    10    13
2020-01-02    NaN    20    23
2020-01-03    NaN    15    18
2020-01-04    NaN    30    33
2020-01-05    NaN    45    48

    >>> df.shift(periods=3, fill_value=0)
              Col1  Col2  Col3
2020-01-01     0     0     0
2020-01-02     0     0     0
2020-01-03     0     0     0
2020-01-04    10     13    17
2020-01-05    20     23    27

    >>> df.shift(periods=3, freq="D")
              Col1  Col2  Col3
2020-01-04    10    13    17
2020-01-05    20    23    27
2020-01-06    15    18    22
2020-01-07    30    33    37
2020-01-08    45    48    52

    >>> df.shift(periods=3, freq="infer")
              Col1  Col2  Col3
2020-01-04    10    13    17
2020-01-05    20    23    27
2020-01-06    15    18    22
2020-01-07    30    33    37
2020-01-08    45    48    52

    skew(self, axis: 'int | None | lib.NoDefault' = <no_default>, skipna=True,
level=None, numeric_only=None, **kwargs)
    Return unbiased skew over requested axis.

    Normalized by N-1.

    Parameters
    -----
    axis : {index (0), columns (1)}
        Axis for the function to be applied on.
    skipna : bool, default True
        Exclude NA/null values when computing the result.
    level : int or level name, default None
        If the axis is a MultiIndex (hierarchical), count along a
        particular level, collapsing into a Series.
    numeric_only : bool, default None
        Include only float, int, boolean columns. If None, will attempt to use
        everything, then use only numeric data. Not implemented for Series.
    **kwargs
        Additional keyword arguments to be passed to the function.

    Returns
    ------
    Series or DataFrame (if level specified)

    sort_index(self, axis: 'Axis' = 0, level: 'Level | None' = None, ascending:
    'bool | int | Sequence[bool | int]' = True, inplace: 'bool' = False, kind: 'str' =
    'quicksort', na_position: 'str' = 'last', sort_remaining: 'bool' = True,
    ignore_index: 'bool' = False, key: 'IndexKeyFunc' = None)
    Sort object by labels (along an axis).

    Returns a new DataFrame sorted by label if `inplace` argument is
    ``False``, otherwise updates the original DataFrame and returns None.

    Parameters
    -----
    axis : {0 or 'index', 1 or 'columns'}, default 0
        The axis along which to sort. The value 0 identifies the rows,
        and 1 identifies the columns.
    level : int or level name or list of ints or list of level names
        If not None, sort on values in specified index level(s).
    ascending : bool or list-like of bools, default True
        Sort ascending vs. descending. When the index is a MultiIndex the
        sort direction can be controlled for each level individually.
    inplace : bool, default False
        If True, perform operation in-place.
    kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, default
    'quicksort'

```

```

|     Choice of sorting algorithm. See also :func:`numpy.sort` for more
|     information. `mergesort` and `stable` are the only stable algorithms.
For
|     DataFrames, this option is only applied when sorting on a single
|     column or label.
na_position : {'first', 'last'}, default 'last'
|     Puts NaNs at the beginning if 'first'; 'last' puts NaNs at the end.
|     Not implemented for MultiIndex.
sort_remaining : bool, default True
|     If True and sorting by level and index is multilevel, sort by other
|     levels too (in order) after sorting by specified level.
ignore_index : bool, default False
|     If True, the resulting axis will be labeled 0, 1, ..., n - 1.
|     .. versionadded:: 1.0.0
key : callable, optional
|     If not None, apply the key function to the index values
|     before sorting. This is similar to the `key` argument in the
|     builtin :meth:`sorted` function, with the notable difference that
|     this `key` function should be *vectorized*. It should expect an
|     ``Index`` and return an ``Index`` of the same shape. For MultiIndex
|     inputs, the key is applied *per level*.
|     .. versionadded:: 1.1.0
Returns
|     -----
|     DataFrame or None
|     The original DataFrame sorted by the labels or None if
`inplace=True`.
See Also
|     -----
Series.sort_index : Sort Series by the index.
DataFrame.sort_values : Sort DataFrame by the value.
Series.sort_values : Sort Series by the value.

Examples
|     -----
>>> df = pd.DataFrame([1, 2, 3, 4, 5], index=[100, 29, 234, 1, 150],
...                      columns=['A'])
>>> df.sort_index()
A
1    4
29   2
100  1
150  5
234  3

By default, it sorts in ascending order, to sort in descending order,
use ``ascending=False``.

>>> df.sort_index(ascending=False)
A
234  3
150  5
100  1
29   2
1    4

A key function can be specified which is applied to the index before
sorting. For a ``MultiIndex`` this is applied to each level separately.

>>> df = pd.DataFrame({'a': [1, 2, 3, 4]}, index=['A', 'B', 'C', 'D'])
>>> df.sort_index(key=lambda x: x.str.lower())
a
A  1
B  2
C  3
D  4

sort_values(self, by, axis: 'Axis' = 0, ascending=True, inplace: 'bool' =
False, kind: 'str' = 'quicksort', na_position: 'str' = 'last', ignore_index:
'bool' = False, key: 'ValueKeyFunc' = None)
|     Sort by the values along either axis.

Parameters
|     -----
by : str or list of str
|     Name or list of names to sort by.
|     - if `axis` is 0 or ``'index'`` then `by` may contain index
|       levels and/or column labels.
|     - if `axis` is 1 or ``'columns'`` then `by` may contain column
|       levels and/or index labels.
axis : {0 or 'index', 1 or 'columns'}, default 0
|     Axis to be sorted.
ascending : bool or list of bool, default True
|     Sort ascending vs. descending. Specify list for multiple sort
|     orders. If this is a list of booleans, must match the length of
|     the by.
inplace : bool, default False
|     If True, perform operation in-place.
kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, default
'quicksort'
|     Choice of sorting algorithm. See also :func:`numpy.sort` for more
|     information. `mergesort` and `stable` are the only stable algorithms.
For
|     DataFrames, this option is only applied when sorting on a single
|     column or label.
na_position : {'first', 'last'}, default 'last'
|     Puts NaNs at the beginning if 'first'; 'last' puts NaNs at the
|     end.
ignore_index : bool, default False
|     If True, the resulting axis will be labeled 0, 1, ..., n - 1.
|     .. versionadded:: 1.0.0
key : callable, optional
|     Apply the key function to the values
|     before sorting. This is similar to the `key` argument in the
|     builtin :meth:`sorted` function, with the notable difference that
|     this `key` function should be *vectorized*. It should expect a
|     ``Series`` and return a Series with the same shape as the input.
|     It will be applied to each column in `by` independently.
|     .. versionadded:: 1.1.0
Returns
|     -----
|     DataFrame or None
|     DataFrame with sorted values or None if ``inplace=True``.
See Also
|     -----
DataFrame.sort_index : Sort a DataFrame by the index.
Series.sort_values : Similar method for a Series.

Examples
|     -----

```

```

>>> df = pd.DataFrame({
...     'col1': ['A', 'A', 'B', np.nan, 'D', 'C'],
...     'col2': [2, 1, 9, 8, 7, 4],
...     'col3': [0, 1, 9, 4, 2, 3],
...     'col4': ['a', 'B', 'c', 'd', 'e', 'F']
... })
>>> df
   col1  col2  col3  col4
0    A      2      0    a
1    A      1      1    B
2    B      9      9    c
3  NaN      8      4    D
4    D      7      2    e
5    C      4      3    F

Sort by col1

>>> df.sort_values(by=['col1'])
   col1  col2  col3  col4
0    A      2      0    a
1    A      1      1    B
2    B      9      9    c
5    C      4      3    F
4    D      7      2    e
3  NaN      8      4    D

Sort by multiple columns

>>> df.sort_values(by=['col1', 'col2'])
   col1  col2  col3  col4
1    A      1      1    B
0    A      2      0    a
2    B      9      9    c
5    C      4      3    F
4    D      7      2    e
3  NaN      8      4    D

Sort Descending

>>> df.sort_values(by='col1', ascending=False)
   col1  col2  col3  col4
4    D      7      2    e
5    C      4      3    F
2    B      9      9    c
0    A      2      0    a
1    A      1      1    B
3  NaN      8      4    D

Putting NAs first

>>> df.sort_values(by='col1', ascending=False, na_position='first')
   col1  col2  col3  col4
3  NaN      8      4    D
4    D      7      2    e
5    C      4      3    F
2    B      9      9    c
0    A      2      0    a
1    A      1      1    B

Sorting with a key function

>>> df.sort_values(by='col4', key=lambda col: col.str.lower())
   col1  col2  col3  col4
0    A      2      0    a
1    A      1      1    B
2    B      9      9    c
3  NaN      8      4    D
4    D      7      2    e
5    C      4      3    F

Natural sort with the key argument,
using the `natsort <https://github.com/SethMMorton/natsort>` package.

>>> df = pd.DataFrame({
...     "time": ['0hr', '128hr', '72hr', '48hr', '96hr'],
...     "value": [10, 20, 30, 40, 50]
... })
>>> df
   time  value
0   0hr     10
1  128hr    20
2   72hr     30
3   48hr     40
4   96hr     50

>>> from natsort import index_natsorted
>>> df.sort_values(
...     by="time",
...     key=lambda x: np.argsort(index_natsorted(df["time"])))
   time  value
0   0hr     10
3   48hr    40
2   72hr    30
4   96hr    50
1  128hr   20

stack(self, level: 'Level' = -1, dropna: 'bool' = True)
Stack the prescribed level(s) from columns to index.

Return a reshaped DataFrame or Series having a multi-level
index with one or more new inner-most levels compared to the current
DataFrame. The new inner-most levels are created by pivoting the
columns of the current dataframe:

- if the columns have a single level, the output is a Series;
- if the columns have multiple levels, the new index
  level(s) is (are) taken from the prescribed level(s) and
  the output is a DataFrame.

Parameters
-----
level : int, str, list, default -1
    Level(s) to stack from the column axis onto the index
    axis, defined as one index or label, or a list of indices
    or labels.
dropna : bool, default True
    Whether to drop rows in the resulting Frame/Series with
    missing values. Stacking a column level onto the index
    axis can create combinations of index and column values
    that are missing from the original dataframe. See Examples
    section.

Returns
-----
DataFrame or Series
    Stacked dataframe or series.

See Also
-----
DataFrame.unstack : Unstack prescribed level(s) from index axis
    onto column axis.

```

```

DataFrame.pivot : Reshape dataframe from long format to wide
    format.
DataFrame.pivot_table : Create a spreadsheet-style pivot table
    as a DataFrame.

Notes
-----
The function is named by analogy with a collection of books
being reorganized from being side by side on a horizontal
position (the columns of the dataframe) to being stacked
vertically on top of each other (in the index of the
dataframe).

Reference :ref:`the user guide <reshaping.stackting>` for more examples.

Examples
-----
**Single level columns**

>>> df_single_level_cols = pd.DataFrame([[0, 1], [2, 3]],
...                                         index=['cat', 'dog'],
...                                         columns=['weight', 'height'])

Stacking a dataframe with a single level column axis returns a Series:

>>> df_single_level_cols
   weight  height
cat      0       1
dog      2       3
>>> df_single_level_cols.stack()
   weight
cat      0
   height
dog      2
   height
            3
dtype: int64

**Multi level columns: simple case**

>>> multicol1 = pd.MultiIndex.from_tuples([('weight', 'kg'),
...                                         ('weight', 'pounds')])
>>> df_multi_level_cols1 = pd.DataFrame([[1, 2], [2, 4]],
...                                         index=['cat', 'dog'],
...                                         columns=multicol1)

Stacking a dataframe with a multi-level column axis:

>>> df_multi_level_cols1
   weight
   kg  pounds
cat     1       2
dog     2       4
>>> df_multi_level_cols1.stack()
   weight
cat kg      1
   pounds    2
dog kg      2
   pounds    4

**Missing values**

>>> multicol2 = pd.MultiIndex.from_tuples([('weight', 'kg'),
...                                         ('height', 'm')])
>>> df_multi_level_cols2 = pd.DataFrame([[1.0, 2.0], [3.0, 4.0]],
...                                         index=['cat', 'dog'],
...                                         columns=multicol2)

It is common to have missing values when stacking a dataframe
with multi-level columns, as the stacked dataframe typically
has more values than the original dataframe. Missing values
are filled with NaNs:

>>> df_multi_level_cols2
   weight  height
   kg      m
cat     1.0    2.0
dog     3.0    4.0
>>> df_multi_level_cols2.stack()
   height  weight
cat kg    NaN    1.0
   m      2.0    NaN
dog kg    NaN    3.0
   m      4.0    NaN

**Prescribing the level(s) to be stacked**

The first parameter controls which level or levels are stacked:

>>> df_multi_level_cols2.stack(0)
   kg      m
cat height  NaN  2.0
   weight  1.0  NaN
dog height  NaN  4.0
   weight  3.0  NaN
>>> df_multi_level_cols2.stack([0, 1])
cat height  m    2.0
   weight  kg   1.0
dog height  m    4.0
   weight  kg   3.0
dtype: float64

**Dropping missing values**

>>> df_multi_level_cols3 = pd.DataFrame([[None, 1.0], [2.0, 3.0]],
...                                         index=['cat', 'dog'],
...                                         columns=multicol2)

Note that rows where all values are missing are dropped by
default but this behaviour can be controlled via the dropna
keyword parameter:

>>> df_multi_level_cols3
   weight  height
   kg      m
cat  NaN    1.0
dog  2.0    3.0
>>> df_multi_level_cols3.stack(dropna=False)
   height  weight
cat kg    NaN    NaN
   m      1.0    NaN
dog kg    NaN    2.0
   m      3.0    NaN
>>> df_multi_level_cols3.stack(dropna=True)
   height  weight
cat m     1.0    NaN
dog kg    NaN    2.0
   m      3.0    NaN

std(self, axis=None, skipna=True, level=None, ddof=1, numeric_only=None,
**kwargs)
    Return sample standard deviation over requested axis.

```

```

    Normalized by N-1 by default. This can be changed using the ddof argument.

Parameters
-----
axis : {index (0), columns (1)}
skipna : bool, default True
    Exclude NA/null values. If an entire row/column is NA, the result
    will be NA.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
ddof : int, default 1
    Delta Degrees of Freedom. The divisor used in calculations is N -
ddof,
    where N represents the number of elements.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.

Returns
-----
Series or DataFrame (if level specified)

Notes
-----
To have the same behaviour as `numpy.std`, use `ddof=0` (instead of the
default `ddof=1`)

Examples
-----
>>> df = pd.DataFrame({'person_id': [0, 1, 2, 3],
...                      'age': [21, 25, 62, 43],
...                      'height': [1.61, 1.87, 1.49, 2.01]}
...                     ).set_index('person_id')
>>> df
   age  height
person_id
0      21    1.61
1      25    1.87
2      62    1.49
3      43    2.01

The standard deviation of the columns can be found as follows:

>>> df.std()
age      18.786076
height   0.237417

Alternatively, `ddof=0` can be set to normalize by N instead of N-1:

>>> df.std(ddof=0)
age      16.269219
height   0.205609

sub(self, other, axis='columns', level=None, fill_value=None)
    Get Subtraction of dataframe and other, element-wise (binary operator
`sub`).
    Equivalent to ``dataframe - other`` , but with support to substitute a
fill_value
    for missing data in one of the inputs. With reverse version, `rsub`.

    Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
    arithmetic operators: '+', '-', '*', '/', '//', '%', '**'.

Parameters
-----
other : scalar, sequence, Series, or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful Dataframe alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-----
DataFrame
    Result of the arithmetic operation.

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                     index=['circle', 'triangle', 'rectangle'])
>>> df
   angles  degrees
circle      0     360
triangle    3     180
rectangle   4     360

Add a scalar with operator version which return the same
results.

>>> df + 1
   angles  degrees
circle      1     361
triangle    4     181
rectangle   5     361

>>> df.add(1)
   angles  degrees
circle      1     361
triangle    4     181
rectangle   5     361

Divide by constant with reverse version.

>>> df.div(10)
   angles  degrees

```

```

circle      0.0     36.0
triangle   0.3     18.0
rectangle  0.4     36.0

>>> df.rdiv(10)
          angles    degrees
circle      inf  0.027778
triangle  3.333333  0.055556
rectangle  2.500000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
          angles    degrees
circle      -1     358
triangle    2     178
rectangle   3     358

>>> df.sub([1, 2], axis='columns')
          angles    degrees
circle      -1     358
triangle    2     178
rectangle   3     358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
          axis='index')
          angles    degrees
circle      -1     359
triangle    2     179
rectangle   3     359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4],
...                         index=['circle', 'triangle', 'rectangle'])
>>> other
          angles
circle      0
triangle   3
rectangle  4

>>> df * other
          angles    degrees
circle      0     0.0
triangle   9     0.0
rectangle  16    0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
...                                index=[['A', 'A', 'A', 'B', 'B', 'B'],
...                                       ['circle', 'triangle', 'rectangle',
'square', 'pentagon', 'hexagon']])
>>> df_multindex
          angles    degrees
A circle      0     360
triangle     3     180
rectangle    4     360
B square      4     360
pentagon     5     540
hexagon      6     720

>>> df.div(df_multindex, level=1, fill_value=0)
          angles    degrees
A circle      NaN     1.0
triangle    1.0     1.0
rectangle   1.0     1.0
B square      0.0     0.0
pentagon     0.0     0.0
hexagon      0.0     0.0

subtract = sub(self, other, axis='columns', level=None, fill_value=None)

sum(self, axis=None, skipna=True, level=None, numeric_only=None, min_count=0,
**kwargs)
    Return the sum of the values over the requested axis.

    This is equivalent to the method ``numpy.sum``.

Parameters
-----
axis : {index (0), columns (1)}
    Axis for the function to be applied on.
skipna : bool, default True
    Exclude NA/null values when computing the result.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.
min_count : int, default 0
    The required number of valid values to perform the operation. If fewer
than
    ``min_count`` non-NA values are present the result will be NA.
**kwargs
    Additional keyword arguments to be passed to the function.

Returns
-----
Series or DataFrame (if level specified)

See Also
-----
Series.sum : Return the sum.
Series.min : Return the minimum.
Series.max : Return the maximum.
Series.idxmin : Return the index of the minimum.
Series.idxmax : Return the index of the maximum.
DataFrame.sum : Return the sum over the requested axis.
DataFrame.min : Return the minimum over the requested axis.
DataFrame.max : Return the maximum over the requested axis.
DataFrame.idxmin : Return the index of the minimum over the requested
axis.
DataFrame.idxmax : Return the index of the maximum over the requested
axis.

Examples
-----
>>> idx = pd.MultiIndex.from_arrays([
...     ['warm', 'warm', 'cold', 'cold'],
...     ['dog', 'falcon', 'fish', 'spider']],

```

```

...     names=['blooded', 'animal'])
>>> s = pd.Series([4, 2, 0, 8], name='legs', index=idx)
>>> s
blooded    animal
warm        dog      4
cold        falcon   2
Name: legs, dtype: int64

>>> s.sum()
14

By default, the sum of an empty or all-NA Series is ``0``.

>>> pd.Series([], dtype="float64").sum() # min_count=0 is the default
0.0

This can be controlled with the ``min_count`` parameter. For example, if
you'd like the sum of an empty series to be NaN, pass ``min_count=1``.

>>> pd.Series([], dtype="float64").sum(min_count=1)
nan

Thanks to the ``skipna`` parameter, ``min_count`` handles all-NA and
empty series identically.

>>> pd.Series([np.nan]).sum()
0.0

>>> pd.Series([np.nan]).sum(min_count=1)
nan

swaplevel(self, i: 'Axis' = -2, j: 'Axis' = -1, axis: 'Axis' = 0) ->
'DataFrame'
Swap levels i and j in a :class:`MultiIndex`.

Default is to swap the two innermost levels of the index.

Parameters
-----
i, j : int or str
    Levels of the indices to be swapped. Can pass level name as string.
axis : {0 or 'index', 1 or 'columns'}, default 0
    The axis to swap levels on. 0 or 'index' for row-wise, 1 or
    'columns' for column-wise.

Returns
-----
DataFrame
    DataFrame with levels swapped in MultiIndex.

Examples
-----
>>> df = pd.DataFrame(
...     {"Grade": ["A", "B", "A", "C"]},
...     index=[
...         ["Final exam", "Final exam", "Coursework", "Coursework"],
...         ["History", "Geography", "History", "Geography"],
...         ["January", "February", "March", "April"],
...     ],
... )
>>> df
          Grade
Final exam History January    A
                  Geography February   B
Coursework History March     A
                  Geography April      C

In the following example, we will swap the levels of the indices.
Here, we will swap the levels column-wise, but levels can be swapped row-
wise
in a similar manner. Note that column-wise is the default behaviour.
By not supplying any arguments for i and j, we swap the last and second to
last indices.

>>> df.swaplevel()
          Grade
Final exam January History    A
                  February Geography   B
Coursework March History    A
                  April Geography      C

By supplying one argument, we can choose which index to swap the last
index with. We can for example swap the first index with the last one as
follows.

>>> df.swaplevel(0)
          Grade
January History Final exam    A
February Geography Final exam   B
March History Coursework    A
April Geography Coursework      C

We can also define explicitly which indices we want to swap by supplying
values
for both i and j. Here, we for example swap the first and second indices.

>>> df.swaplevel(0, 1)
          Grade
History Final exam January    A
Geography Final exam February   B
History Coursework March     A
Geography Coursework April      C

to_dict(self, orient: 'str' = 'dict', into=<class 'dict'>)
Convert the DataFrame to a dictionary.

The type of the key-value pairs can be customized with the parameters
(see below).

Parameters
-----
orient : str {'dict', 'list', 'series', 'split', 'records', 'index'}
    Determines the type of the values of the dictionary.

    - 'dict' (default) : dict like {column -> {index -> value}}
    - 'list' : dict like {column -> [values]}
    - 'series' : dict like {column -> Series(values)}
    - 'split' : dict like
        {'index' -> [index], 'columns' -> [columns], 'data' -> [values]}
    - 'tight' : dict like
        {'index' -> [index], 'columns' -> [columns], 'data' -> [values],
        'index_names' -> [index.names], 'column_names' -> [column.names]}
    - 'records' : list like
        [{column -> value}, ... , {column -> value}]
    - 'index' : dict like {index -> {column -> value}}

Abbreviations are allowed. 's` indicates 'series' and 'sp'
indicates 'split'.

```

```

    .. versionadded:: 1.4.0
        ``tight`` as an allowed value for the ``orient`` argument

into : class, default dict
    The collections.abc.Mapping subclass used for all Mappings
    in the return value. Can be the actual class or an empty
    instance of the mapping type you want. If you want a
    collections.defaultdict, you must pass it initialized.

Returns
-----
dict, list or collections.abc.Mapping
    Return a collections.abc.Mapping object representing the DataFrame.
    The resulting transformation depends on the 'orient' parameter.

See Also
-----
DataFrame.from_dict: Create a DataFrame from a dictionary.
DataFrame.to_json: Convert a DataFrame to JSON format.

Examples
-----
>>> df = pd.DataFrame({'col1': [1, 2],
...                      'col2': [0.5, 0.75]},
...                      index=['row1', 'row2'])
>>> df
   col1  col2
row1    1  0.50
row2    2  0.75
>>> df.to_dict()
{'col1': {'row1': 1, 'row2': 2}, 'col2': {'row1': 0.5, 'row2': 0.75}}

You can specify the return orientation.

>>> df.to_dict('series')
{'col1': row1    1
             row2    2
Name: col1, dtype: int64,
'col2': row1  0.50
             row2  0.75
Name: col2, dtype: float64}

>>> df.to_dict('split')
{'index': ['row1', 'row2'], 'columns': ['col1', 'col2'],
 'data': [[1, 0.5], [2, 0.75]]}

>>> df.to_dict('records')
[{'col1': 1, 'col2': 0.5}, {'col1': 2, 'col2': 0.75}]

>>> df.to_dict('index')
{'row1': {'col1': 1, 'col2': 0.5}, 'row2': {'col1': 2, 'col2': 0.75}}

>>> df.to_dict('tight')
{'index': ['row1', 'row2'], 'columns': ['col1', 'col2'],
 'data': [[1, 0.5], [2, 0.75]], 'index_names': [None], 'column_names':
[None]}

You can also specify the mapping type.

>>> from collections import OrderedDict, defaultdict
>>> df.to_dict(into=OrderedDict)
OrderedDict([('col1', OrderedDict([('row1', 1), ('row2', 2)])),
             ('col2', OrderedDict([('row1', 0.5), ('row2', 0.75))))])

If you want a `defaultdict`, you need to initialize it:

>>> dd = defaultdict(list)
>>> df.to_dict('records', into=dd)
[defaultdict(<class 'list'>, {'col1': 1, 'col2': 0.5}),
 defaultdict(<class 'list'>, {'col1': 2, 'col2': 0.75})]

to_feather(self, path: 'FilePath | WriteBuffer[bytes]', **kwargs) -> 'None'
    Write a DataFrame to the binary Feather format.

Parameters
-----
path : str, path object, file-like object
    String, path object (implementing ``os.PathLike[str]``), or file-like
    object implementing a binary ``write()`` function. If a string or a
path,
    it will be used as Root Directory path when writing a partitioned
dataset.
    **kwargs :
        Additional keywords passed to :func:`pyarrow.feather.write_feather`.
        Starting with pyarrow 0.17, this includes the ``compression``,
        ``compression_level``, ``chunksize`` and ``version`` keywords.

    .. versionadded:: 1.1.0

Notes
-----
This function writes the dataframe as a `feather file
<https://arrow.apache.org/docs/python/feather.html>`. Requires a default
index. For saving the DataFrame with your custom index use a method that
supports custom indices e.g. `to_parquet`.

| to_gbq(self, destination_table: 'str', project_id: 'str' | None = None,
| chunksize: 'int' | None = None, reauth: 'bool' = False, if_exists: 'str' = 'fail',
| auth_local_webserver: 'bool' = False, table_schema: 'list[dict[str, str]]' | None =
| None, location: 'str' | None = None, progress_bar: 'bool' = True,
| credentials=None) -> 'None'
|     Write a DataFrame to a Google BigQuery table.

| This function requires the `pandas-gbq package
| <https://pandas-gbq.readthedocs.io>`__.

See the `How to authenticate with Google BigQuery
<https://pandas-gbq.readthedocs.io/en/latest/howto/authentication.html>`__
guide for authentication instructions.

Parameters
-----
destination_table : str
    Name of table to be written, in the form ``dataset.tablename``.
project_id : str, optional
    Google BigQuery Account project ID. Optional when available from
    the environment.
chunksize : int, optional
    Number of rows to be inserted in each chunk from the dataframe.
    Set to ``None`` to load the whole dataframe at once.
reauth : bool, default False
    Force Google BigQuery to re-authenticate the user. This is useful
    if multiple accounts are used.
if_exists : str, default 'fail'
    Behavior when the destination table exists. Value can be one of:
        ``'fail'``
            If table exists raise pandas_gbq.TableCreationError.
        ``'replace'``
            If table exists, drop it, recreate it, and insert data.
        ``'append'``
            Append data to an existing table.

```

```

    If table exists, insert data. Create if does not exist.
auth_local_webserver : bool, default False
    Use the `local webserver flow` instead of the `console flow`_
when getting user credentials.

    ... _local webserver flow:
    https://google-auth-oauthlib.readthedocs.io/en/latest/reference/google_auth_oauthlib.flow.html#google_
auth_oauthlib.flow.InstalledAppFlow.run_local_server
    ... _console flow:
    https://google-auth-oauthlib.readthedocs.io/en/latest/reference/google_auth_oauthlib.flow.html#google_
auth_oauthlib.flow.InstalledAppFlow.run_console

    *New in version 0.2.0 of pandas-gbq*.
table_schema : list of dicts, optional
List of BigQuery table fields to which according DataFrame
columns conform to, e.g. ``[{'name': 'col1', 'type':
'STRING'}, ...]``. If schema is not provided, it will be
generated according to dtypes of DataFrame columns. See
BigQuery API documentation on available names of a field.

    *New in version 0.3.1 of pandas-gbq*.
location : str, optional
Location where the load job should run. See the `BigQuery locations
documentation
<https://cloud.google.com/bigquery/docs/dataset-locations>`_ for a
list of available locations. The location must match that of the
target dataset.

    *New in version 0.5.0 of pandas-gbq*.
progress_bar : bool, default True
    Use the library `tqdm` to show the progress bar for the upload,
chunk by chunk.

    *New in version 0.5.0 of pandas-gbq*.
credentials : google.auth.credentials.Credentials, optional
    Credentials for accessing Google APIs. Use this parameter to
override default credentials, such as to use Compute Engine
:Class:`google.auth.compute_engine.Credentials` or Service
Account :Class:`google.oauth2.service_account.Credentials`
directly.

    *New in version 0.8.0 of pandas-gbq*.

See Also
-----
pandas_gbq.to_gbq : This function in the pandas-gbq library.
read_gbq : Read DataFrame from Google BigQuery.

| to_html(self, buf: 'FilePath | WriteBuffer[str] | None' = None, columns:
| 'Sequence[str] | None' = None, col_space: 'ColspaceArgType | None' = None, header:
| 'bool | Sequence[str]' = True, index: 'bool' = True, na_rep: 'str' = 'NaN',
| formatters: 'FormattersType | None' = None, float_format: 'FloatFormatType | None'
| = None, sparsify: 'bool | None' = None, index_names: 'bool' = True, justify: 'str
| None' = None, max_rows: 'int | None' = None, max_cols: 'int | None' = None,
| show_dimensions: 'bool | str' = False, decimal: 'str' = ',', bold_rows: 'bool' =
| True, classes: 'str | list | tuple | None' = None, escape: 'bool' = True,
| notebook: 'bool' = False, border: 'int | None' = None, table_id: 'str | None' =
| None, render_links: 'bool' = False, encoding: 'str | None' = None)
|     Render a DataFrame as an HTML table.

Parameters
-----
buf : str, Path or StringIO-like, optional, default None
    Buffer to write to. If None, the output is returned as a string.
columns : sequence, optional, default None
    The subset of columns to write. Writes all columns by default.
col_space : str or int, list or dict of int or str, optional
    The minimum width of each column in CSS length units. An int is
assumed to be px units.

    .. versionadded:: 0.25.0
        Ability to use str.
header : bool, optional
    Whether to print column labels, default True.
index : bool, optional, default True
    Whether to print index (row) labels.
na_rep : str, optional, default 'NaN'
    String representation of 'NaN' to use.
formatters : list, tuple or dict of one-param. functions, optional
    Formatter functions to apply to columns' elements by position or
name.
    The result of each function must be a unicode string.
    List/tuple must be of length equal to the number of columns.
float_format : one-parameter function, optional, default None
    Formatter function to apply to columns' elements if they are
floats. This function must return a unicode string and will be
applied only to the non-'NaN' elements, with 'NaN' being
handled by 'na_rep'.

    .. versionchanged:: 1.2.0

sparsify : bool, optional, default True
    Set to False for a DataFrame with a hierarchical index to print
every multiindex key at each row.
index_names : bool, optional, default True
    Prints the names of the indexes.
justify : str, default None
    How to justify the column labels. If None uses the option from
the print configuration (controlled by set_option), 'right' out
of the box. Valid values are
    * left
    * right
    * center
    * justify
    * justify-all
    * start
    * end
    * inherit
    * match-parent
    * initial
    * unset.
max_rows : int, optional
    Maximum number of rows to display in the console.
max_cols : int, optional
    Maximum number of columns to display in the console.
show_dimensions : bool, default False
    Display DataFrame dimensions (number of rows by number of columns).
decimal : str, default ','
    Character recognized as decimal separator, e.g. ',' in Europe.

bold_rows : bool, default True
    Make the row labels bold in the output.
classes : str or list or tuple, default None
    CSS class(es) to apply to the resulting html table.
escape : bool, default True
    Convert the characters <, >, and & to HTML-safe sequences.
notebook : {True, False}, default False
    Whether the generated HTML is for IPython Notebook.

```

```

border : int
    A ``border=border`` attribute is included in the opening
    ``<table>`` tag. Default ``pd.options.display.html.border``.
table_id : str, optional
    A css id is included in the opening ``<table>`` tag if specified.
render_links : bool, default False
    Convert URLs to HTML links.
encoding : str, default "utf-8"
    Set character encoding.

    .. versionadded:: 1.0.0

Returns
-----
str or None
    If buf is None, returns the result as a string. Otherwise returns
    None.

See Also
-----
to_string : Convert DataFrame to a string.

to_markdown(self, buf: 'IO[str] | str | None' = None, mode: 'str' = 'wt',
index: 'bool' = True, storage_options: 'StorageOptions' = None, **kwargs) -> 'str'
| None'
    Print DataFrame in Markdown-friendly format.

    .. versionadded:: 1.0.0

Parameters
-----
buf : str, Path or StringIO-like, optional, default None
    Buffer to write to. If None, the output is returned as a string.
mode : str, optional
    Mode in which file is opened, "wt" by default.
index : bool, optional, default True
    Add index (row) labels.

    .. versionadded:: 1.1.0
storage_options : dict, optional
    Extra options that make sense for a particular storage connection,
e.g.
| host, port, username, password, etc. For HTTP(S) URLs the key-value
pairs
| are forwarded to ``urllib`` as header options. For other URLs (e.g.
| starting with "s3://", and "gcs://") the key-value pairs are forwarded
to
| ``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.

    .. versionadded:: 1.2.0

**kwargs
    These parameters will be passed to `tabulate
<https://pypi.org/project/tabulate>`_.

Returns
-----
str
    DataFrame in Markdown-friendly format.

Notes
-----
Requires the `tabulate <https://pypi.org/project/tabulate>`_ package.

Examples
-----
>>> df = pd.DataFrame(
...     data={"animal_1": ["elk", "pig"], "animal_2": ["dog",
"quetzal"]})
...
>>> print(df.to_markdown())
+---+---+
| animal_1 | animal_2 |
+---+---+
| 0 | elk   | dog   |
| 1 | pig   | quetzal |
+---+---+
Output markdown with a tabulate option.

>>> print(df.to_markdown(tablefmt="grid"))
+---+---+
| animal_1 | animal_2 |
+---+---+
| 0 | elk   | dog   |
| 1 | pig   | quetzal |
+---+---+

to_numpy(self, dtype: 'np.dtypeLike | None' = None, copy: 'bool' = False,
na_value=<no_default>) -> 'np.ndarray'
    Convert the DataFrame to a NumPy array.

By default, the dtype of the returned array will be the common NumPy
dtype of all types in the DataFrame. For example, if the dtypes are
``float16`` and ``float32``, the results dtype will be ``float32``.
This may require copying data and coercing values, which may be
expensive.

Parameters
-----
dtype : str or numpy.dtype, optional
    The dtype to pass to :meth:`numpy.asarray`.
copy : bool, default False
    Whether to ensure that the returned value is not a view on
    another array. Note that ``copy=False`` does not *ensure* that
    ``to_numpy()`` is no-copy. Rather, ``copy=True`` ensure that
    a copy is made, even if not strictly necessary.
na_value : Any, optional
    The value to use for missing values. The default value depends
    on ``dtype`` and the dtypes of the DataFrame columns.

    .. versionadded:: 1.1.0

Returns
-----
numpy.ndarray

See Also
-----
Series.to_numpy : Similar method for Series.

Examples
-----
>>> pd.DataFrame({"A": [1, 2], "B": [3, 4]}).to_numpy()
array([[1, 3],
       [2, 4]])

With heterogeneous data, the lowest common type will have to
be used.

>>> df = pd.DataFrame({"A": [1, 2], "B": [3.0, 4.5]})
>>> df.to_numpy()

```

```

array([[1., 3.],
       [2., 4.5]])

For a mix of numeric and non-numeric types, the output array will
have object dtype.

>>> df['C'] = pd.date_range('2000', periods=2)
>>> df.to_numpy()
array([[1, 3.0, Timestamp('2000-01-01 00:00:00')],
       [2, 4.5, Timestamp('2000-01-02 00:00:00')]], dtype=object)

to_parquet(self, path: 'FilePath | WriteBuffer[bytes] | None' = None, engine:
'str' = 'auto', compression: 'str | None' = 'snappy', index: 'bool | None' = None,
partition_cols: 'list[str] | None' = None, storage_options: 'StorageOptions' =
None, **kwargs) -> 'bytes | None'
    Write a DataFrame to the binary parquet format.

This function writes the DataFrame as a `parquet` file
<https://parquet.apache.org/>_. You can choose different parquet
backends, and have the option of compression. See
:ref:`the user guide <io.parquet>` for more details.

Parameters
-----
path : str, path object, file-like object, or None, default None
    String, path object (implementing ``os.PathLike[str]``), or file-like
    object implementing a binary ``write()`` function. If None, the result
    is
    returned as bytes. If a string or path, it will be used as Root
Directory
    path when writing a partitioned dataset.

.. versionchanged:: 1.2.0

Previously this was "fname"

engine : {'auto', 'pyarrow', 'fastparquet'}, default 'auto'
    Parquet library to use. If 'auto', then the option
    ``'io.parquet.engine`` is used. The default ``'io.parquet.engine``'
    behavior is to try 'pyarrow', falling back to 'fastparquet' if
    'pyarrow' is unavailable.
compression : {'snappy', 'gzip', 'brotli', None}, default 'snappy'
    Name of the compression to use. Use ``None`` for no compression.
index : bool, default None
    If ``True``, include the DataFrame's index(es) in the file output.
    If ``False``, they will not be written to the file.
    If ``None``, similar to ``True`` the DataFrame's index(es)
    will be saved. However, instead of being saved as values,
    the RangeIndex will be stored as a range in the metadata so it
    doesn't require much space and is faster. Other indexes will
    be included as columns in the file output.
partition_cols : list, optional, default None
    Column names by which to partition the dataset.
    Columns are partitioned in the order they are given.
    Must be None if path is not a string.
storage_options : dict, optional
    Extra options that make sense for a particular storage connection,
e.g.
    host, port, username, password, etc. For HTTP(S) URLs the key-value
pairs
    are forwarded to ``urllib`` as header options. For other URLs (e.g.
    starting with "s3://", and "gcs://") the key-value pairs are forwarded
to
    ``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.

.. versionadded:: 1.2.0

**kwargs
    Additional arguments passed to the parquet library. See
    :ref:`pandas io <io.parquet>` for more details.

Returns
-----
bytes if no path argument is provided else None

See Also
-----
read_parquet : Read a parquet file.
DataFrame.to_csv : Write a csv file.
DataFrame.to_sql : Write to a sql table.
DataFrame.to_hdf : Write to hdf.

Notes
-----
This function requires either the `fastparquet`  

<https://pypi.org/project/fastparquet/>_ or `pyarrow`  

<https://arrow.apache.org/docs/python/>_ library.

Examples
-----
>>> df = pd.DataFrame(data={'col1': [1, 2], 'col2': [3, 4]})
>>> df.to_parquet('df.parquet.gzip',
...                 compression='gzip') # doctest: +SKIP
>>> pd.read_parquet('df.parquet.gzip') # doctest: +SKIP
   col1  col2
0     1     3
1     2     4

If you want to get a buffer to the parquet content you can use a
io.BytesIO
object, as long as you don't use partition_cols, which creates multiple
files.
|
>>> import io
>>> f = io.BytesIO()
>>> df.to_parquet(f)
>>> f.seek(0)
|
>>> content = f.read()

to_period(self, freq: 'Frequency | None' = None, axis: 'Axis' = 0, copy:
'bool' = True) -> 'DataFrame'
    Convert DataFrame from DatetimeIndex to PeriodIndex.

Convert DataFrame from DatetimeIndex to PeriodIndex with desired
frequency (inferred from index if not passed).

Parameters
-----
freq : str, default
    Frequency of the PeriodIndex.
axis : {0 or 'index', 1 or 'columns'}, default 0
    The axis to convert (the index by default).
copy : bool, default True
    If False then underlying input data is not copied.

Returns
-----
DataFrame with PeriodIndex

Examples
|

```

```

-----
>>> idx = pd.to_datetime(
...     [
...         "2001-03-31 00:00:00",
...         "2002-05-31 00:00:00",
...         "2003-08-31 00:00:00",
...     ]
... )

>>> idx
DatetimeIndex(['2001-03-31', '2002-05-31', '2003-08-31'],
dtype='datetime64[ns]', freq=None)

>>> idx.to_period("M")
PeriodIndex(['2001-03', '2002-05', '2003-08'], dtype='period[M']')

For yearly frequency

>>> idx.to_period("Y")
PeriodIndex(['2001', '2002', '2003'], dtype='period[A-DEC']')

to_records(self, index=True, column_dtypes=None, index_dtypes=None) ->
np.recarray'
Convert DataFrame to a NumPy record array.

Index will be included as the first field of the record array if
requested.

Parameters
-----
index : bool, default True
    Include index in resulting record array, stored in 'index'
    field or using the index label, if set.
column_dtypes : str, type, dict, default None
    If a string or type, the data type to store all columns. If
    a dictionary, a mapping of column names and indices (zero-indexed)
    to specific data types.
index_dtypes : str, type, dict, default None
    If a string or type, the data type to store all index levels. If
    a dictionary, a mapping of index level names and indices
    (zero-indexed) to specific data types.

This mapping is applied only if `index=True`.


Returns
-----
numpy.recarray
    NumPy ndarray with the DataFrame labels as fields and each row
    of the DataFrame as entries.

See Also
-----
DataFrame.from_records: Convert structured or record ndarray
    to DataFrame.
numpy.recarray: An ndarray that allows field access using
    attributes, analogous to typed columns in a
    spreadsheet.

Examples
-----
>>> df = pd.DataFrame({'A': [1, 2], 'B': [0.5, 0.75]},
...                     index=['a', 'b'])
>>> df
   A      B
a  1  0.50
b  2  0.75
>>> df.to_records()
rec.array([('a', 1, 0.5), ('b', 2, 0.75)],
          dtype=[('index', 'O'), ('A', '<i8'), ('B', '<f8')])

If the DataFrame index has no label then the recarray field name
is set to 'index'. If the index has a label then this is used as the
field name:

>>> df.index = df.index.rename("I")
>>> df.to_records()
rec.array([('a', 1, 0.5), ('b', 2, 0.75)],
          dtype=[('I', 'O'), ('A', '<i8'), ('B', '<f8')])

The index can be excluded from the record array:

>>> df.to_records(index=False)
rec.array([(1, 0.5), (2, 0.75)],
          dtype=[('A', '<i8'), ('B', '<f8')])

Data types can be specified for the columns:

>>> df.to_records(column_dtypes={"A": "int32"})
rec.array([('a', 1, 0.5), ('b', 2, 0.75)],
          dtype=[('I', 'O'), ('A', '<i4'), ('B', '<f8')])

As well as for the index:

>>> df.to_records(index_dtypes="S2")
rec.array([('a', 1, 0.5), ('b', 2, 0.75)],
          dtype=[('I', 'S2'), ('A', '<i8'), ('B', '<f8')])

>>> index_dtypes = f"<{df.index.str.len().max()}"
>>> df.to_records(index_dtypes=index_dtypes)
rec.array([('a', 1, 0.5), ('b', 2, 0.75)],
          dtype=[('I', 'S1'), ('A', '<i8'), ('B', '<f8')])

to_stata(self, path: 'FilePath | WriteBuffer[bytes]', convert_dates:
'dict[Hashable, str] | None' = None, write_index: 'bool' = True, byteorder: 'str |
None' = None, time_stamp: 'datetime.datetime' | None' = None, data_label: 'str |
None' = None, variable_labels: 'dict[Hashable, str] | None' = None, version: 'int |
None' = 114, convert_strl: 'Sequence[Hashable]' | None' = None, compression:
'CompressionOptions' = 'infer', storage_options: 'StorageOptions' = None, *,
value_labels: 'dict[Hashable, dict[float | int, str]] | None' = None) -> 'None'
| Export DataFrame object to Stata data format.

Writes the DataFrame to a Stata dataset file.
"dta" files contain a Stata dataset.

Parameters
-----
path : str, path object, or buffer
    String, path object (implementing ``os.PathLike[str]``), or file-like
    object implementing a binary ``write()`` function.

    .. versionchanged:: 1.0.0

    Previously this was "fname"

convert_dates : dict
    Dictionary mapping columns containing datetime types to stata
    internal format to use when writing the dates. Options are 'tc',
    'td', 'tm', 'tw', 'th', 'tq', 'ty'. Column can be either an integer
    or a name. Datetime columns that do not have a conversion type
    specified will be converted to 'tc'. Raises NotImplementedError if
    a datetime column has timezone information.

```

```

    write_index : bool
        Write the index to Stata dataset.
    byteorder : str
        Can be ">", "<", "little", or "big". default is `sys.byteorder`.
    time_stamp : datetime
        A datetime to use as file creation date. Default is the current
        time.
    data_label : str, optional
        A label for the data set. Must be 80 characters or smaller.
    variable_labels : dict
        Dictionary containing columns as keys and variable labels as
        values. Each label must be 80 characters or smaller.
    version : {114, 117, 118, 119, None}, default 114
        Version to use in the output dtf file. Set to None to let pandas
        decide between 118 or 119 formats depending on the number of
        columns in the frame. Version 114 can be read by Stata 10 and
        later. Version 117 can be read by Stata 13 or later. Version 118
        is supported in Stata 14 and later. Version 119 is supported in
        Stata 15 and later. Version 114 limits string variables to 244
        characters or fewer while versions 117 and later allow strings
        with lengths up to 2,000,000 characters. Versions 118 and 119
        support Unicode characters, and version 119 supports more than
        32,767 variables.

    Version 119 should usually only be used when the number of
    variables exceeds the capacity of dtf format 118. Exporting
    smaller datasets in format 119 may have unintended consequences,
    and, as of November 2020, Stata SE cannot read version 119 files.

    .. versionchanged:: 1.0.0

        Added support for formats 118 and 119.

    convert_strl : list, optional
        List of column names to convert to string columns to Stata StrL
        format. Only available if version is 117. Storing strings in the
        StrL format can produce smaller dtf files if strings have more than
        8 characters and values are repeated.
    compression : str or dict, default 'infer'
        For on-the-fly compression of the output data. If 'infer' and 'path'
        path-like, then detect compression from the following extensions:
    'gz',
    |     '.bz2', '.zip', '.xz', or '.zst' (otherwise no compression). Set to
    |     ``None`` for no compression. Can also be a dict with key ``'method'``
    set
    |     to one of {``'zip'``, ``'gzip'``, ``'bz2'``, ``'zstd'``} and other
    |     key-value pairs are forwarded to ``'zipfile.Zipfile'``,
    |     ``gzip.GzipFile``,
    |     ``bz2.BZ2File`` , or ``zstandard.ZstdDecompressor`` , respectively. As
    an
    |     example, the following could be passed for faster compression and to
    create
    |     a reproducible gzip archive:
    |     ``compression={'method': 'gzip', 'compresslevel': 1, 'mtime': 1}``.
    |
    .. versionadded:: 1.1.0
    |
    .. versionchanged:: 1.4.0 Zstandard support.

    storage_options : dict, optional
        Extra options that make sense for a particular storage connection,
e.g.
    |     host, port, username, password, etc. For HTTP(S) URLs the key-value
pairs
    |     are forwarded to ``urllib`` as header options. For other URLs (e.g.
to
    |     starting with ``s3://``, and ``gcs://``) the key-value pairs are forwarded
    |     ``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.
    |
    .. versionadded:: 1.2.0

    value_labels : dict of dicts
        Dictionary containing columns as keys and dictionaries of column value
        to labels as values. Labels for a single variable must be 32,000
        characters or smaller.

    .. versionadded:: 1.4.0

Raises
-----
NotImplementedError
    * If datetimes contain timezone information
    * Column dtype is not representable in Stata
ValueError
    * Columns listed in convert_dates are neither datetime64[ns]
        or datetime.datetime
    * Column listed in convert_dates is not in DataFrame
    * Categorical label contains more than 32,000 characters

See Also
-----
read_stata : Import Stata data files.
io.stata.StataWriter : Low-level writer for Stata data files.
io.stata.StataWriter117 : Low-level writer for version 117 files.

Examples
-----
>>> df = pd.DataFrame({'animal': ['falcon', 'parrot', 'falcon',
...                               'parrot'],
...                      'speed': [350, 18, 361, 15]})
>>> df.to_stata('animals.dta') # doctest: +SKIP

    to_string(self, buf: 'FilePath | WriteBuffer[str] | None' = None, columns:
    'Sequence[str] | None' = None, col_space: 'int | list[int] | dict[Hashable, int]' |
    None = None, header: 'bool | Sequence[str]' = True, index: 'bool' = True, na_rep:
    'str' = 'NaN', formatters: 'fmt.FormattersType' | None = None, float_format:
    'fmt.FloatFormatType' | None = None, sparsify: 'bool | None' = None, index_names:
    'bool' = True, justify: 'str | None' = None, max_rows: 'int | None' = None,
    max_cols: 'int | None' = None, show_dimensions: 'bool' = False, decimal: 'str' =
    '.', line_width: 'int | None' = None, min_rows: 'int | None' = None, max_colwidth:
    'int | None' = None, encoding: 'str | None' = None) -> 'str | None'
    Render a DataFrame to a console-friendly tabular output.

Parameters
-----
buf : str, Path or StringIO-like, optional, default None
    Buffer to write to. If None, the output is returned as a string.
columns : sequence, optional, default None
    The subset of columns to write. Writes all columns by default.
col_space : int, list or dict of int, optional
    The minimum width of each column. If a list of ints is given every
    integers corresponds with one column. If a dict is given, the key references the
    column, while the value defines the space to use.
header : bool or sequence of str, optional
    Write out the column names. If a list of strings is given, it is
    assumed to be aliases for the column names.
index : bool, optional, default True
    Whether to print index (row) labels.
na_rep : str, optional, default 'NaN'
    String representation of 'NaN' to use.
formatters : list, tuple or dict of one-param. functions, optional

```

```

    Formatter functions to apply to columns' elements by position or
    name.
    The result of each function must be a unicode string.
    List/tuple must be of length equal to the number of columns.
    float_format : one-parameter function, optional, default None
        Formatter function to apply to columns' elements if they are
        floats. This function must return a unicode string and will be
        applied only to the non-``NaN`` elements, with ``NaN`` being
        handled by ``na_rep``.
    ...
    .. versionchanged:: 1.2.0

    sparsify : bool, optional, default True
        Set to False for a DataFrame with a hierarchical index to print
        every multiindex key at each row.
    index_names : bool, optional, default True
        Prints the names of the indexes.
    justify : str, default None
        How to justify the column labels. If None uses the option from
        the print configuration (controlled by set_option), 'right' out
        of the box. Valid values are
        * left
        * right
        * center
        * justify
        * justify-all
        * start
        * end
        * inherit
        * match-parent
        * initial
        * unset.
    max_rows : int, optional
        Maximum number of rows to display in the console.
    max_cols : int, optional
        Maximum number of columns to display in the console.
    show_dimensions : bool, default False
        Display DataFrame dimensions (number of rows by number of columns).
    decimal : str, default ','
        Character recognized as decimal separator, e.g. ',' in Europe.
    line_width : int, optional
        Width to wrap a line in characters.
    min_rows : int, optional
        The number of rows to display in the console in a truncated repr
        (when number of rows is above `max_rows`).
    max_colwidth : int, optional
        Max width to truncate each column in characters. By default, no limit.

    ...
    .. versionadded:: 1.0.0
    encoding : str, default "utf-8"
        Set character encoding.

    ...
    .. versionadded:: 1.0

    Returns
    ------
    str or None
        If buf is None, returns the result as a string. Otherwise returns
        None.

    See Also
    -----
    to_html : Convert DataFrame to HTML.

    Examples
    -----
    >>> d = {'col1': [1, 2, 3], 'col2': [4, 5, 6]}
    >>> df = pd.DataFrame(d)
    >>> print(df.to_string())
      col1  col2
    0     1     4
    1     2     5
    2     3     6

    to_timestamp(self, freq: 'Frequency | None' = None, how: 'str' = 'start',
    axis: 'Axis' = 0, copy: 'bool' = True) -> 'DatetimeIndex'
    Cast to DatetimeIndex of timestamps, at *beginning* of period.

    Parameters
    -----
    freq : str, default frequency of PeriodIndex
        Desired frequency.
    how : {'s', 'e', 'start', 'end'}
        Convention for converting period to timestamp; start of period
        vs. end.
    axis : {0 or 'index', 1 or 'columns'}, default 0
        The axis to convert (the index by default).
    copy : bool, default True
        If False then underlying input data is not copied.

    Returns
    ------
    DataFrame with DatetimeIndex

    to_xml(self, path_or_buffer: 'FilePath | WriteBuffer[bytes] | WriteBuffer[str]
    | None' = None, index: 'bool' = True, root_name: 'str | None' = 'data', row_name:
    'str | None' = 'row', na_rep: 'str | None' = None, attr_cols: 'list[str] | None' =
    None, elem_cols: 'list[str] | None' = None, namespaces: 'dict[str | None, str] |
    None' = None, prefix: 'str | None' = None, encoding: 'str' = 'utf-8',
    xml_declaration: 'bool | None' = True, pretty_print: 'bool | None' = True, parser:
    'str | None' = 'xml', stylesheet: 'FilePath | ReadBuffer[str] | ReadBuffer[bytes]
    | None' = None, compression: 'CompressionOptions' = 'infer', storage_options:
    'StorageOptions' = None) -> 'str | None'
    Render a DataFrame to an XML document.

    ...
    .. versionadded:: 1.3.0

    Parameters
    -----
    path_or_buffer : str, path object, file-like object, or None, default None
        String, path object (implementing ``os.PathLike[str]``), or file-like
        object implementing a ``write()`` function. If None, the result is
    returned
        as a string.
    index : bool, default True
        Whether to include index in XML document.
    root_name : str, default 'data'
        The name of root element in XML document.
    row_name : str, default 'row'
        The name of row element in XML document.
    na_rep : str, optional
        Missing data representation.
    attr_cols : list-like, optional
        List of columns to write as attributes in row element.
        Hierarchical columns will be flattened with underscore
        delimiting the different levels.
    elem_cols : list-like, optional
        List of columns to write as children in row element. By default,
        all columns output as children of row element. Hierarchical

```

```

columns will be flattened with underscore delimiting the
different levels.
namespaces : dict, optional
    All namespaces to be defined in root element. Keys of dict
    should be prefix names and values of dict corresponding URIs.
    Default namespaces should be given empty string key. For
    example, ::

        namespaces = {"": "https://example.com"}

prefix : str, optional
    Namespace prefix to be used for every element and/or attribute
    in document. This should be one of the keys in ``namespaces`` dict.
encoding : str, default 'utf-8'
    Encoding of the resulting document.
xml_declaration : bool, default True
    Whether to include the XML declaration at start of document.
pretty_print : bool, default True
    Whether output should be pretty printed with indentation and
    line breaks.
parser : {'lxml', 'etree'}, default 'lxml'
    Parser module to use for building of tree. Only 'lxml' and
    'etree' are supported. With 'lxml', the ability to use XSLT
    stylesheet is supported.
stylesheet : str, path object or file-like object, optional
    A URL, file-like object, or a raw string containing an XSLT
    script used to transform the raw XML output. Script should use
    layout of elements and attributes from original output. This
    argument requires ``lxml`` to be installed. Only XSLT 1.0
    scripts and not later versions is currently supported.
compression : str or dict, default 'infer'
    For on-the-fly compression of the output data. If 'infer' and
    'path_or_buffer'
        path-like, then detect compression from the following extensions:
        ``.gz``,
        ``.bz2``,
        ``.zip``,
        ``.xz``,
        ``.zst`` (otherwise no compression). Set to
        ``None`` for no compression. Can also be a dict with key ``method`` -
set
        to one of {``zip``},
        ``gzip``},
        ``bz2``},
        ``zstd``} and other
        key-value pairs are forwarded to ``zipfile.ZipFile``,
``gzip.GzipFile``,
        ``bz2.BZ2File`` or ``zstandard.ZstdDecompressor``, respectively. As
an
        example, the following could be passed for faster compression and to
create
        a reproducible gzip archive:
        ``compression={method': 'gzip', 'compresslevel': 1, 'mtime': 1}``.
        .. versionchanged:: 1.4.0 Zstandard support.

storage_options : dict, optional
e.g.
    Extra options that make sense for a particular storage connection,
pairs
    host, port, username, password, etc. For HTTP(S) URLs the key-value
to
    pairs are forwarded to ``urllib`` as header options. For other URLs (e.g.
    starting with "s3://", and "gcs://") the key-value pairs are forwarded
    to ``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.

Returns
-----
None or str
    If ``io`` is None, returns the resulting XML format as a
    string. Otherwise returns None.

See Also
-----
to_json : Convert the pandas object to a JSON string.
to_html : Convert DataFrame to a html.

Examples
-----
>>> df = pd.DataFrame({'shape': ['square', 'circle', 'triangle'],
...                      'degrees': [360, 360, 180],
...                      'sides': [4, np.nan, 3]})

>>> df.to_xml() # doctest: +SKIP
<?xml version='1.0' encoding='utf-8'?>
<data>
  <row>
    <index>0</index>
    <shape>square</shape>
    <degrees>360</degrees>
    <sides>4.0</sides>
  </row>
  <row>
    <index>1</index>
    <shape>circle</shape>
    <degrees>360</degrees>
    <sides/>
  </row>
  <row>
    <index>2</index>
    <shape>triangle</shape>
    <degrees>180</degrees>
    <sides>3.0</sides>
  </row>
</data>

>>> df.to_xml(attr_cols=[..., 'index', 'shape', 'degrees', 'sides'])
...     ]) # doctest: +SKIP
<?xml version='1.0' encoding='utf-8'?>
<data>
  <row index="0" shape="square" degrees="360" sides="4.0"/>
  <row index="1" shape="circle" degrees="360"/>
  <row index="2" shape="triangle" degrees="180" sides="3.0"/>
</data>

>>> df.to_xml(namespaces={"doc": "https://example.com"}, ...
...             prefix="doc") # doctest: +SKIP
<?xml version='1.0' encoding='utf-8'?>
<doc xmlns:doc="https://example.com">
  <doc:row>
    <doc:index>0</doc:index>
    <doc:shape>square</doc:shape>
    <doc:degrees>360</doc:degrees>
    <doc:sides>4.0</doc:sides>
  </doc:row>
  <doc:row>
    <doc:index>1</doc:index>
    <doc:shape>circle</doc:shape>
    <doc:degrees>360</doc:degrees>
    <doc:sides/>
  </doc:row>
  <doc:row>
    <doc:index>2</doc:index>
    <doc:shape>triangle</doc:shape>
    <doc:degrees>180</doc:degrees>
  </doc:row>
</doc>
```

```

        <doc:sides>3.0</doc:sides>
    </doc:row>
</doc:data>

    transform(self, func: 'AggFuncType', axis: 'Axis' = 0, *args, **kwargs) ->
'DataFrame'
    Call ``func`` on self producing a DataFrame with the same axis shape as
self.

    Parameters
    -----
    func : function, str, list-like or dict-like
        Function to use for transforming the data. If a function, must either
        work when passed a DataFrame or when passed to DataFrame.apply. If
        func
            is both list-like and dict-like, dict-like behavior takes precedence.

        Accepted combinations are:
        -
        - function
        - string function name
        - list-like of functions and/or function names, e.g. ``[np.exp,
'sqrt']``
            - dict-like of axis labels -> functions, function names or list-like
of such.
        axis : {0 or 'index', 1 or 'columns'}, default 0
            If 0 or 'index': apply function to each column.
            If 1 or 'columns': apply function to each row.

    *args
        Positional arguments to pass to `func`.

    **kwargs
        Keyword arguments to pass to `func`.

    Returns
    -----
    DataFrame
        A DataFrame that must have the same length as self.

    Raises
    -----
    ValueError : If the returned DataFrame has a different length than self.

    See Also
    -----
    DataFrame.agg : Only perform aggregating type operations.
    DataFrame.apply : Invoke function on a DataFrame.

    Notes
    -----
    Functions that mutate the passed object can produce unexpected
behavior or errors and are not supported. See :ref:`gotchas.udf-mutation`
for more details.

    Examples
    -----
    >>> df = pd.DataFrame({'A': range(3), 'B': range(1, 4)})
    >>> df
      A   B
    0   0   1
    1   1   2
    2   2   3
    >>> df.transform(lambda x: x + 1)
      A   B
    0   1   2
    1   2   3
    2   3   4

    Even though the resulting DataFrame must have the same length as the
input DataFrame, it is possible to provide several input functions:

    >>> s = pd.Series(range(3))
    >>> s
    0    0
    1    1
    2    2
    dtype: int64
    >>> s.transform([np.sqrt, np.exp])
           sqrt      exp
    0  0.000000  1.000000
    1  1.000000  2.718282
    2  1.414214  7.389056

    You can call transform on a GroupBy object:

    >>> df = pd.DataFrame({
...     "Date": [
...         "2015-05-08", "2015-05-07", "2015-05-06", "2015-05-05",
...         "2015-05-08", "2015-05-07", "2015-05-06", "2015-05-05"],
...     "Data": [5, 8, 6, 1, 50, 100, 60, 120],
... })
    >>> df
      Date  Data
    0  2015-05-08    5
    1  2015-05-07    8
    2  2015-05-06    6
    3  2015-05-05    1
    4  2015-05-08   50
    5  2015-05-07  100
    6  2015-05-06   60
    7  2015-05-05  120
    >>> df.groupby('Date')['Data'].transform('sum')
    0    55
    1   108
    2    66
    3   121
    4    55
    5   108
    6    66
    7   121
    Name: Data, dtype: int64

    >>> df = pd.DataFrame({
...     "c": [1, 1, 1, 2, 2, 2, 2],
...     "type": ["m", "n", "o", "m", "n", "n", "n"]
... })
    >>> df
      c  type
    0  1    m
    1  1    n
    2  1    o
    3  2    m
    4  2    m
    5  2    n
    6  2    n
    >>> df['size'] = df.groupby('c')['type'].transform(len)
    >>> df
      c  type  size
    0  1    m    3
    1  1    n    3
    2  1    o    3
    3  2    m    4

```

```

    4 2     m     4
    5 2     n     4
    6 2     n     4

    transpose(self, *args, copy: 'bool' = False) -> 'DataFrame'
        Transpose index and columns.

        Reflect the DataFrame over its main diagonal by writing rows as columns
        and vice-versa. The property :attr:`.T` is an accessor to the method
        :meth:`transpose`.

    Parameters
    -----
    *args : tuple, optional
        Accepted for compatibility with NumPy.
    copy : bool, default False
        Whether to copy the data after transposing, even for DataFrames
        with a single dtype.

        Note that a copy is always required for mixed dtype DataFrames,
        or for DataFrames with any extension types.

    Returns
    -----
    DataFrame
        The transposed DataFrame.

    See Also
    -----
    numpy.transpose : Permute the dimensions of a given array.

    Notes
    -----
    Transposing a DataFrame with mixed dtypes will result in a homogeneous
    DataFrame with the 'object' dtype. In such a case, a copy of the data
    is always made.

    Examples
    -----
    **Square DataFrame with homogeneous dtype**

    >>> d1 = {'col1': [1, 2], 'col2': [3, 4]}
    >>> df1 = pd.DataFrame(data=d1)
    >>> df1
       col1  col2
    0     1     3
    1     2     4

    >>> df1_transposed = df1.T # or df1.transpose()
    >>> df1_transposed
       0
    col1  1
    col2  2
    col2  3  4

    When the dtype is homogeneous in the original DataFrame, we get a
    transposed DataFrame with the same dtype:

    >>> df1.dtypes
    col1    int64
    col2    int64
    dtype: object
    >>> df1_transposed.dtypes
    0    int64
    1    int64
    dtype: object

    **Non-square DataFrame with mixed dtypes**

    >>> d2 = {'name': ['Alice', 'Bob'],
    ...         'score': [9.5, 8],
    ...         'employed': [False, True],
    ...         'kids': [0, 0]}
    >>> df2 = pd.DataFrame(data=d2)
    >>> df2
      name  score  employed  kids
    0  Alice    9.5     False    0
    1    Bob     8.0      True    0

    >>> df2_transposed = df2.T # or df2.transpose()
    >>> df2_transposed
          0    1
    name    Alice   Bob
    score    9.5   8.0
    employed  False  True
    kids      0    0

    When the DataFrame has mixed dtypes, we get a transposed DataFrame with
    the 'object' dtype:

    >>> df2.dtypes
    name      object
    score    float64
    employed    bool
    kids      int64
    dtype: object
    >>> df2_transposed.dtypes
    0      object
    1      object
    dtype: object

    truediv(self, other, axis='columns', level=None, fill_value=None)
    | Get Floating division of dataframe and other, element-wise (binary
    operator 'truediv').

    | Equivalent to ``dataframe / other``', but with support to substitute a
    fill_value
    | for missing data in one of the inputs. With reverse version, `rtruediv`.

    | Among flexible wrappers ('add', 'sub', 'mul', 'div', 'mod', 'pow') to
    arithmetic operators: '+', '-', '**', '/', '//', '%', '**'.

    Parameters
    -----
    other : scalar, sequence, Series, or DataFrame
        Any single or multiple element data structure, or list-like object.
    axis : {0 or 'index', 1 or 'columns'}
        Whether to compare by the index (0 or 'index') or columns
        (1 or 'columns'). For Series input, axis to match Series index on.
    level : int or label
        Broadcast across a level, matching Index values on the
        passed MultiIndex level.
    fill_value : float or None, default None
        Fill existing missing (NaN) values, and any new element needed for
        successful DataFrame alignment, with this value before computation.
        If data in both corresponding DataFrame locations is missing
        the result will be missing.

    Returns
    -----
    DataFrame
        Result of the arithmetic operation.

```

```

See Also
-----
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
-----
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                      'degrees': [360, 180, 360]},
...                     index=['circle', 'triangle', 'rectangle'])
>>> df
   angles  degrees
circle      0      360
triangle    3      180
rectangle   4      360

Add a scalar with operator version which return the same
results.

>>> df + 1
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

>>> df.add(1)
   angles  degrees
circle      1      361
triangle    4      181
rectangle   5      361

Divide by constant with reverse version.

>>> df.div(10)
   angles  degrees
circle     0.0      36.0
triangle   0.3      18.0
rectangle  0.4      36.0

>>> df.rdiv(10)
   angles  degrees
circle     inf      0.027778
triangle  3.333333  0.055556
rectangle 2.500000  0.027778

Subtract a list and Series with axis with operator version.

>>> df - [1, 2]
   angles  degrees
circle     -1      358
triangle   2      178
rectangle  3      358

>>> df.sub([1, 2], axis='columns')
   angles  degrees
circle     -1      358
triangle   2      178
rectangle  3      358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle',
'rectangle']), ...
          axis='index')
   angles  degrees
circle     -1      359
triangle   2      179
rectangle  3      359

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4],
...                         'index': ['circle', 'triangle', 'rectangle']})
>>> other
   angles
circle     0
triangle   3
rectangle  4

>>> df * other
   angles  degrees
circle     0      NaN
triangle   9      NaN
rectangle  16     NaN

>>> df.mul(other, fill_value=0)
   angles  degrees
circle     0      0.0
triangle   9      0.0
rectangle  16     0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                                'degrees': [360, 180, 360, 360, 540,
720]}, ...
...                               index=[['A', 'A', 'A', 'B', 'B'],
...                                     ['circle', 'triangle', 'rectangle',
...                                      'square', 'pentagon', 'hexagon']])
>>> df_multindex
   angles  degrees
A circle     0      360
triangle    3      180
rectangle   4      360
B square     4      360
pentagon    5      540
hexagon     6      720

>>> df.div(df_multindex, level=1, fill_value=0)
   angles  degrees
A circle    NaN      1.0
triangle   1.0      1.0
rectangle  1.0      1.0
B square    0.0      0.0
pentagon   0.0      0.0
hexagon    0.0      0.0

unstack(self, level: 'Level' = -1, fill_value=None)
Pivot a level of the (necessarily hierarchical) index labels.

Returns a DataFrame having a new level of column labels whose inner-most
level

```

consists of the pivoted index labels.

If the index is not a MultiIndex, the output will be a Series (the analogue of stack when the columns are not a MultiIndex).

Parameters

level : int, str, or list of these, default -1 (last level)
Level(s) of index to unstack, can pass level name.
fill_value : int, str or dict
Replace NaN with this value if the unstack produces missing values.

Returns

Series or DataFrame

See Also

DataFrame.pivot : Pivot a table based on column values.
DataFrame.stack : Pivot a level of the column labels (inverse operation from 'unstack').

Notes

Reference :ref:`the user guide <reshaping.stackting>` for more examples.

Examples

```
>>> index = pd.MultiIndex.from_tuples([('one', 'a'), ('one', 'b'),
...                                         ('two', 'a'), ('two', 'b')])
>>> s = pd.Series(np.arange(1.0, 5.0), index=index)
>>> s
one   a    1.0
      b    2.0
two   a    3.0
      b    4.0
dtype: float64

>>> s.unstack(level=-1)
      a   b
one  1.0  2.0
two  3.0  4.0

>>> s.unstack(level=0)
      one  two
      a    1.0
      b    2.0
      two  a    3.0
      b    4.0
dtype: float64
```

update(self, other, join: 'str' = 'left', overwrite: 'bool' = True,
filter_func=None, errors: 'str' = 'ignore') -> 'None'
Modify in place using non-NA values from another DataFrame.

Aligns on indices. There is no return value.

Parameters

other : DataFrame, or object coercible into a DataFrame
Should have at least one matching index/column label
with the original DataFrame. If a Series is passed,
its name attribute must be set, and that will be
used as the column name to align with the original DataFrame.
join : {'left'}, default 'left'
Only left join is implemented, keeping the index and columns of the
original object.
overwrite : bool, default True
How to handle non-NA values for overlapping keys:

- * True: overwrite original DataFrame's values
with values from other' .
- * False: only update values that are NA in
the original Dataframe.

filter_func : callable(1d-array) -> bool 1d-array, optional
Can choose to replace values other than NA. Return True for values
that should be updated.
errors : {'raise', 'ignore'}, default 'ignore'
If 'raise', will raise a ValueError if the DataFrame and 'other'
both contain non-NA data in the same place.

Returns

None : method directly changes calling object

Raises

ValueError

- * When 'errors='raise'' and there's overlapping non-NA data.
- * When 'errors' is not either ''ignore'' or ''raise''

NotImplementedError

- * If 'join != 'left''

See Also

dict.update : Similar method for dictionaries.
DataFrame.merge : For column(s)-on-column(s) operations.

Examples

```
>>> df = pd.DataFrame({'A': [1, 2, 3],
...                      'B': [400, 500, 600]})
>>> new_df = pd.DataFrame({'B': [4, 5, 6],
...                         'C': [7, 8, 9]})
>>> df.update(new_df)
>>> df
   A   B
0   1   4
1   2   5
2   3   6
```

The DataFrame's length does not increase as a result of the update,
only values at matching index/column labels are updated.

```
>>> df = pd.DataFrame({'A': ['a', 'b', 'c'],
...                      'B': ['x', 'y', 'z']})
>>> new_df = pd.DataFrame({'B': ['d', 'e', 'f', 'g', 'h', 'i']})
>>> df.update(new_df)
>>> df
   A   B
0   a   d
1   b   e
2   c   f
```

For Series, its name attribute must be set.

```

>>> df = pd.DataFrame({'A': ['a', 'b', 'c'],
...                   'B': ['x', 'y', 'z']})
>>> new_column = pd.Series(['d', 'e'], name='B', index=[0, 2])
>>> df.update(new_column)
>>> df
   A    B
0  a    d
1  b    y
2  c    e
>>> df = pd.DataFrame({'A': ['a', 'b', 'c'],
...                   'B': ['x', 'y', 'z']})
>>> new_df = pd.DataFrame({'B': ['d', 'e']}, index=[1, 2])
>>> df.update(new_df)
>>> df
   A    B
0  a    x
1  b    d
2  c    e

If 'other' contains NaNs the corresponding values are not updated
in the original DataFrame.

>>> df = pd.DataFrame({'A': [1, 2, 3],
...                   'B': [400, 500, 600]})
>>> new_df = pd.DataFrame({'B': [4, np.nan, 6]}) 
>>> df.update(new_df)
>>> df
   A      B
0  1    4.0
1  2  500.0
2  3    6.0

value_counts(self, subset: 'Sequence[Hashable] | None' = None, normalize:
'bool' = False, sort: 'bool' = True, ascending: 'bool' = False, dropna: 'bool' =
True)
    Return a Series containing counts of unique rows in the DataFrame.

.. versionadded:: 1.1.0

Parameters
-----
subset : list-like, optional
    Columns to use when counting unique combinations.
normalize : bool, default False
    Return proportions rather than frequencies.
sort : bool, default True
    Sort by frequencies.
ascending : bool, default False
    Sort in ascending order.
dropna : bool, default True
    Don't include counts of rows that contain NA values.

.. versionadded:: 1.3.0

Returns
-----
Series

See Also
-----
Series.value_counts: Equivalent method on Series.

Notes
-----
The returned Series will have a MultiIndex with one level per input
column. By default, rows that contain any NA values are omitted from
the result. By default, the resulting Series will be in descending
order so that the first element is the most frequently-occurring row.

Examples
-----
>>> df = pd.DataFrame({'num_legs': [2, 4, 4, 6],
...                     'num_wings': [2, 0, 0, 0]},
...                     index=['falcon', 'dog', 'cat', 'ant'])
>>> df
   num_legs  num_wings
falcon        2          2
dog           4          0
cat           4          0
ant           6          0

>>> df.value_counts()
num_legs  num_wings
4            0          2
2            2          1
6            0          1
dtype: int64

>>> df.value_counts(sort=False)
num_legs  num_wings
2            2          1
4            0          2
6            0          1
dtype: int64

>>> df.value_counts(ascending=True)
num_legs  num_wings
2            2          1
6            0          1
4            0          2
dtype: int64

>>> df.value_counts(normalize=True)
num_legs  num_wings
4            0         0.50
2            2         0.25
6            0         0.25
dtype: float64

With 'dropna' set to 'False' we can also count rows with NA values.

>>> df = pd.DataFrame({'first_name': ['John', 'Anne', 'John', 'Beth'],
...                     'middle_name': ['Smith', pd.NA, pd.NA, 'Louise']})
>>> df
   first_name  middle_name
0      John        Smith
1      Anne        <NA>
2      John        <NA>
3      Beth       Louise

>>> df.value_counts()
first_name  middle_name
Beth        Louise          1
John        Smith          1
Anne        NaN             1
dtype: int64

>>> df.value_counts(dropna=False)
first_name  middle_name
Anne        NaN             1
Beth        Louise          1

```

```

John      Smith      1
NaN      NaN      1
dtype: int64

var(self, axis=None, skipna=True, level=None, ddof=1, numeric_only=None,
**kwargs)
    Return unbiased variance over requested axis.

    Normalized by N-1 by default. This can be changed using the ddof argument.

Parameters
-----
axis : {index (0), columns (1)}
skipna : bool, default True
    Exclude NA/null values. If an entire row/column is NA, the result
    will be NA.
level : int or level name, default None
    If the axis is a MultiIndex (hierarchical), count along a
    particular level, collapsing into a Series.
ddof : int, default 1
    Delta Degrees of Freedom. The divisor used in calculations is N -
ddof,
    where N represents the number of elements.
numeric_only : bool, default None
    Include only float, int, boolean columns. If None, will attempt to use
    everything, then use only numeric data. Not implemented for Series.

Returns
-----
Series or DataFrame (if level specified)

Examples
-----
>>> df = pd.DataFrame({'person_id': [0, 1, 2, 3],
...                      'age': [21, 25, 62, 43],
...                      'height': [1.61, 1.87, 1.49, 2.01]})
...                      ).set_index('person_id')
>>> df
   age  height
person_id
0     21    1.61
1     25    1.87
2     62    1.49
3     43    2.01

>>> df.var()
age      352.916667
height    0.056367

Alternatively, ``ddof=0`` can be set to normalize by N instead of N-1:

>>> df.var(ddof=0)
age      264.687500
height    0.042275

where(self, cond, other=<no_default>, inplace=False, axis=None, level=None,
errors='raise', try_cast=<no_default>)
    Replace values where the condition is False.

Parameters
-----
cond : bool Series/DataFrame, array-like, or callable
    Where 'cond' is True, keep the original value. Where
    False, replace with corresponding value from 'other'.
    If 'cond' is callable, it is computed on the Series/DataFrame and
    should return boolean Series/DataFrame or array. The callable must
    not change input Series/DataFrame (though pandas doesn't check it).
other : scalar, Series/DataFrame, or callable
    Entries where 'cond' is False are replaced with
    corresponding value from 'other'.
    If other is callable, it is computed on the Series/DataFrame and
    should return scalar or Series/DataFrame. The callable must not
    change input Series/DataFrame (though pandas doesn't check it).
inplace : bool, default False
    Whether to perform the operation in place on the data.
axis : int, default None
    Alignment axis if needed.
level : int, default None
    Alignment level if needed.
errors : str, {'raise', 'ignore'}, default 'raise'
    Note that currently this parameter won't affect
    the results and will always coerce to a suitable dtype.
    - 'raise' : allow exceptions to be raised.
    - 'ignore' : suppress exceptions. On error return original object.

try_cast : bool, default None
    Try to cast the result back to the input type (if possible).
    .. deprecated:: 1.3.0
        Manually cast back if necessary.

Returns
-----
Same type as caller or None if ``inplace=True``.

See Also
-----
:func:`DataFrame.mask` : Return an object of same shape as
self.

Notes
-----
The where method is an application of the if-then idiom. For each
element in the calling DataFrame, if ``cond`` is ``True`` the
element is used; otherwise the corresponding element from the DataFrame
``other`` is used.

The signature for :func:`DataFrame.where` differs from
:func:`numpy.where`. Roughly ``df1.where(m, df2)`` is equivalent to
``np.where(m, df1, df2)``.

For further details and examples see the ``where`` documentation in
:ref:`indexing.where_masks`.

Examples
-----
>>> s = pd.Series(range(5))
>>> s.where(s > 0)
0      NaN
1      1.0
2      2.0
3      3.0
4      4.0
dtype: float64
>>> s.mask(s > 0)
0      0.0
1      NaN
2      NaN
3      NaN
4      NaN

```

```

dtype: float64
>>> s.where(s > 1, 10)
0    10
1     1
2     2
3     3
4     4
dtype: int64
>>> s.mask(s > 1, 10)
0     0
1     1
2    10
3    10
4    10
dtype: int64

>>> df = pd.DataFrame(np.arange(10).reshape(-1, 2), columns=['A', 'B'])
>>> df
   A   B
0   0   1
1   2   3
2   4   5
3   6   7
4   8   9
>>> m = df % 3 == 0
>>> df.where(m, -df)
   A   B
0  0  -1
1 -2   3
2 -4  -5
3  6   7
4 -8   9
>>> df.where(m, -df) == np.where(m, df, -df)
   A   B
0  True  True
1  True  True
2  True  True
3  True  True
4  True  True
>>> df.where(m, -df) == df.mask(~m, -df)
   A   B
0  True  True
1  True  True
2  True  True
3  True  True
4  True  True
-----
Class methods defined here:

from_dict(data, orient: 'str' = 'columns', dtype: 'Dtype | None' = None,
          columns=None) -> 'DataFrame' from builtins.type
    Construct DataFrame from dict of array-like or dicts.

Creates DataFrame object from dictionary by columns or by index
allowing dtype specification.

Parameters
-----
data : dict
    Of the form {field : array-like} or {field : dict}.
orient : {'columns', 'index', 'tight'}, default 'columns'
    The "orientation" of the data. If the keys of the passed dict
    should be the columns of the resulting DataFrame, pass 'columns'.
    (default). Otherwise if the keys should be rows, pass 'index'.
    If 'tight', assume a dict with keys ['index', 'columns', 'data',
    'index_names', 'column_names'].

    .. versionadded:: 1.4.0
        'tight' as an allowed value for the ``orient`` argument

dtype : dtype, default None
    Data type to force, otherwise infer.
columns : list, default None
    Column labels to use when ``orient='index'``. Raises a ValueError
    if used with ``orient='columns'`` or ``orient='tight'``.

Returns
-----
DataFrame

See Also
-----
DataFrame.from_records : DataFrame from structured ndarray, sequence
    of tuples or dicts, or DataFrame.
DataFrame : DataFrame object creation using constructor.
DataFrame.to_dict : Convert the DataFrame to a dictionary.

Examples
-----
By default the keys of the dict become the DataFrame columns:

>>> data = {'col_1': [3, 2, 1, 0], 'col_2': ['a', 'b', 'c', 'd']}
>>> pd.DataFrame.from_dict(data)
   col_1  col_2
0      3    a
1      2    b
2      1    c
3      0    d

Specify ``orient='index'`` to create the DataFrame using dictionary
keys as rows:

>>> data = {'row_1': [3, 2, 1, 0], 'row_2': ['a', 'b', 'c', 'd']}
>>> pd.DataFrame.from_dict(data, orient='index')
   0  1  2  3
row_1  3  2  1  0
row_2  a  b  c  d

When using the 'index' orientation, the column names can be
specified manually:

>>> pd.DataFrame.from_dict(data, orient='index',
...                           columns=['A', 'B', 'C', 'D'])
   A  B  C  D
row_1  3  2  1  0
row_2  a  b  c  d

Specify ``orient='tight'`` to create the DataFrame using a 'tight'
format:

>>> data = {'index': [('a', 'b'), ('a', 'c')],
...           'columns': [('x', 1), ('y', 2)],
...           'data': [[1, 3], [2, 4]],
...           'index_names': ['n1', 'n2'],
...           'column_names': ['z1', 'z2']}
>>> pd.DataFrame.from_dict(data, orient='tight')
z1  x  y
z2  1  2
n1 n2

```

```

    a  b  1  3
    c  2  4

from_records(data, index=None, exclude=None, columns=None, coerce_float:
'bool' = False, nrows: 'int | None' = None) -> 'DataFrame' from builtins.type
    Convert structured or record ndarray to DataFrame.

    Creates a DataFrame object from a structured ndarray, sequence of
    tuples or dicts, or DataFrame.

Parameters
-----
data : structured ndarray, sequence of tuples or dicts, or DataFrame
    Structured input data.
index : str, list of fields, array-like
    Field of array to use as the index, alternately a specific set of
    input labels to use.
exclude : sequence, default None
    Columns or fields to exclude.
columns : sequence, default None
    Column names to use. If the passed data do not have names
    associated with them, this argument provides names for the
    columns. Otherwise this argument indicates the order of the columns
    in the result (any names not found in the data will become all-NA
    columns).
coerce_float : bool, default False
    Attempt to convert values of non-string, non-numeric objects (like
    decimal.Decimal) to floating point, useful for SQL result sets.
nrows : int, default None
    Number of rows to read if data is an iterator.

Returns
-----
DataFrame

See Also
-----
DataFrame.from_dict : DataFrame from dict of array-like or dicts.
DataFrame : DataFrame object creation using constructor.

Examples
-----
Data can be provided as a structured ndarray:

>>> data = np.array([(3, 'a'), (2, 'b'), (1, 'c'), (0, 'd')],
...                  dtype=[('col_1', 'i4'), ('col_2', 'U1')])
>>> pd.DataFrame.from_records(data)
   col_1 col_2
0      3    a
1      2    b
2      1    c
3      0    d

Data can be provided as a list of dicts:

>>> data = [{'col_1': 3, 'col_2': 'a'},
...            {'col_1': 2, 'col_2': 'b'},
...            {'col_1': 1, 'col_2': 'c'},
...            {'col_1': 0, 'col_2': 'd'}]
>>> pd.DataFrame.from_records(data)
   col_1 col_2
0      3    a
1      2    b
2      1    c
3      0    d

Data can be provided as a list of tuples with corresponding columns:

>>> data = [(3, 'a'), (2, 'b'), (1, 'c'), (0, 'd')]
>>> pd.DataFrame.from_records(data, columns=['col_1', 'col_2'])
   col_1 col_2
0      3    a
1      2    b
2      1    c
3      0    d

-----
Readonly properties defined here:

T

axes
    Return a list representing the axes of the DataFrame.

    It has the row axis labels and column axis labels as the only members.
    They are returned in that order.

Examples
-----
>>> df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
>>> df.axes
[RangeIndex(start=0, stop=2, step=1), Index(['col1', 'col2'],
dtype='object')]

shape
    Return a tuple representing the dimensionality of the DataFrame.

See Also
-----
ndarray.shape : Tuple of array dimensions.

Examples
-----
>>> df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
>>> df.shape
(2, 2)

>>> df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4],
...                      'col3': [5, 6]})
>>> df.shape
(2, 3)

style
    Returns a Styler object.

    Contains methods for building a styled HTML representation of the
    DataFrame.

See Also
-----
io.formats.style.Styler : Helps style a DataFrame or Series according to
the
    data with HTML and CSS.

values
    Return a Numpy representation of the DataFrame.

... warning::
    We recommend using :meth:`DataFrame.to_numpy` instead.

```

Only the values in the DataFrame will be returned, the axes labels will be removed.

Returns

numpy.ndarray
 The values of the DataFrame.

See Also

DataFrame.to_numpy : Recommended alternative to this method.
DataFrame.index : Retrieve the index labels.
DataFrame.columns : Retrieving the column names.

Notes

The dtype will be a lower-common-denominator dtype (implicit upcasting); that is to say if the dtypes (even of numeric types) are mixed, the one that accommodates all will be chosen. Use this with care if you are not dealing with the blocks.

e.g. If the dtypes are float16 and float32, dtype will be upcast to float32. If dtypes are int32 and uint8, dtype will be upcast to int32. By :func:`numpy.find_common_type` convention, mixing int64 and uint64 will result in a float64 dtype.

Examples

A DataFrame where all columns are the same type (e.g., int64) results in an array of the same type.

```
>>> df = pd.DataFrame({'age': [ 3, 29],
...                     'height': [94, 170],
...                     'weight': [31, 115]})

>>> df
   age  height  weight
0    3      94      31
1   29     170     115

>>> df.dtypes
age    int64
height    int64
weight    int64
dtype: object
>>> df.values
array([[ 3, 94, 31],
       [29, 170, 115]])

A DataFrame with mixed type columns(e.g., str/object, int64, float32)
results in an ndarray of the broadest type that accommodates these
mixed types (e.g., object).
```

```
>>> df2 = pd.DataFrame([('parrot', 24.0, 'second'),
...                      ('lion', 80.5, 1),
...                      ('monkey', np.nan, None)],
...                     columns=['name', 'max_speed', 'rank'])

>>> df2.dtypes
name    object
max_speed    float64
rank    object
dtype: object
>>> df2.values
array([('parrot', 24.0, 'second'),
       ('lion', 80.5, 1),
       ('monkey', nan, None)], dtype=object)
```

Data descriptors defined here:

columns
 The column labels of the DataFrame.

index
 The index (row labels) of the DataFrame.

Data and other attributes defined here:

annotations = {'_AXIS_TO_AXIS_NUMBER': 'dict[Axis, int]', '_access...'

plot = <class 'pandas.plotting._core.PlotAccessor'>
 Make plots of Series or DataFrame.

 Uses the backend specified by the
 option ``plotting.backend``. By default, matplotlib is used.

Parameters

data : Series or DataFrame
 The object for which the method is called.
x : label or position, default None
 Only used if data is a DataFrame.
y : label, position or list of label, positions, default None
 Allows plotting of one column versus another. Only used if data is a
 DataFrame.
kind : str
 The kind of plot to produce:
 - 'line' : line plot (default)
 - 'bar' : vertical bar plot
 - 'barh' : horizontal bar plot
 - 'hist' : histogram
 - 'box' : boxplot
 - 'kde' : Kernel Density Estimation plot
 - 'density' : same as 'kde'
 - 'area' : area plot
 - 'pie' : pie plot
 - 'scatter' : scatter plot (DataFrame only)
 - 'hexbin' : hexbin plot (DataFrame only)
ax : matplotlib axes object, default None
 An axes of the current figure.
subplots : bool, default False
 Make separate subplots for each column.
sharex : bool, default True if ax is None else False
 In case ``subplots=True``, share x axis and set some x axis labels
 to invisible; defaults to True if ax is None otherwise False if
 an ax is passed in; Be aware, that passing in both an ax and
 ``sharex=True`` will alter all x axis labels for all axis in a figure.
sharey : bool, default False
 In case ``subplots=True``, share y axis and set some y axis labels to
 invisible.
 layout : tuple, optional
 (rows, columns) for the layout of subplots.
 figsize : a tuple (width, height) in inches
 Size of a figure object.
use_index : bool, default True
 Use index as ticks for x axis.
title : str or list
 Title to use for the plot. If a string is passed, print the string
 at the top of the figure. If a list is passed and 'subplots' is
 True, print each item in the list above the corresponding subplot.
grid : bool, default None (matlab style default)

```

    Axis grid lines.
legend : bool or {'reverse'}
    Place legend on axis subplots.
style : list or dict
    The matplotlib line style per column.
logx : bool or 'sym', default False
    Use log scaling or symlog scaling on x axis.
    .. versionchanged:: 0.25.0

logy : bool or 'sym' default False
    Use log scaling or symlog scaling on y axis.
    .. versionchanged:: 0.25.0

loglog : bool or 'sym', default False
    Use log scaling or symlog scaling on both x and y axes.
    .. versionchanged:: 0.25.0

xticks : sequence
    Values to use for the xticks.
yticks : sequence
    Values to use for the yticks.
xlim : 2-tuple/list
    Set the x limits of the current axes.
ylim : 2-tuple/list
    Set the y limits of the current axes.
xlabel : label, optional
    Name to use for the xlabel on x-axis. Default uses index name as
xlabel, or the
    x-column name for planar plots.
    .. versionadded:: 1.1.0
    .. versionchanged:: 1.2.0

        Now applicable to planar plots ('scatter', 'hexbin').

ylabel : label, optional
    Name to use for the ylabel on y-axis. Default will show no ylabel, or
the
    y-column name for planar plots.
    .. versionadded:: 1.1.0
    .. versionchanged:: 1.2.0

        Now applicable to planar plots ('scatter', 'hexbin').

rot : int, default None
    Rotation for ticks (xticks for vertical, yticks for horizontal
    plots).
fontsize : int, default None
    Font size for xticks and yticks.
colormap : str or matplotlib colormap object, default None
    Colormap to select colors from. If string, load colormap with that
    name from matplotlib.
colorbar : bool, optional
    If True, plot colorbar (only relevant for 'scatter' and 'hexbin'
    plots).
position : float
    Specify relative alignments for bar plot layout.
    From 0 (left/bottom-end) to 1 (right/top-end). Default is 0.5
    (center).
table : bool, Series or DataFrame, default False
    If True, draw a table using the data in the DataFrame and the data
    will be transposed to meet matplotlib's default layout.
    If a Series or DataFrame is passed, use passed data to draw a
    table.
yerr : DataFrame, Series, array-like, dict and str
    See :ref:`Plotting with Error Bars <visualization.errorbars>` for
    detail.
xerr : DataFrame, Series, array-like, dict and str
    Equivalent to yerr.
stacked : bool, default False in line and bar plots, and True in area plot
    If True, create stacked plot.
sort_columns : bool, default False
    Sort column names to determine plot ordering.
secondary_y : bool or sequence, default False
    Whether to plot on the secondary y-axis if a list/tuple, which
    columns to plot on secondary y-axis.
mark_right : bool, default True
    When using a secondary_y axis, automatically mark the column
    labels with "(right)" in the legend.
include_bool : bool, default is False
    If True, boolean values can be plotted.
backend : str, default None
    Backend to use instead of the backend specified in the option
    ``plotting.backend``. For instance, 'matplotlib'. Alternatively, to
    specify the ``plotting.backend`` for the whole session, set
    ``pd.options.plotting.backend``.

    .. versionadded:: 1.0.0

**kwargs
    Options to pass to matplotlib plotting method.

Returns
-----
:class:`matplotlib.axes.Axes` or numpy.ndarray of them
    If the backend is not the default matplotlib one, the return value
    will be the object returned by the backend.

Notes
-----
- See matplotlib documentation online for more on this subject
- If 'kind' = 'bar' or 'barr', you can specify relative alignments
    for bar plot layout by 'position' keyword.
    From 0 (left/bottom-end) to 1 (right/top-end). Default is 0.5
    (center)

sparse = <class 'pandas.core.arrays.accessor.SparseFrameAccesso...
    DataFrame accessor for sparse data.

    .. versionadded:: 0.25.0

Methods inherited from pandas.core.generic.NDFrame:
_____
__abs__(self: 'NDFrameT') -> 'NDFrameT'

__array__(self, dtype: 'npt.DTypeLike | None' = None) -> 'np.ndarray'

| __array_ufunc__(self, ufunc: 'np.ufunc', method: 'str', *inputs: 'Any',
**kwargs: 'Any')

|     __array_wrap__(self, result: 'np.ndarray', context: 'tuple[Callable,
tuple[Any, ...], int] | None' = None)
|         Gets called after a ufunc and other functions.

Parameters
_____
result: np.ndarray

```

```

    The result of the ufunc or other function called on the NumPy array
    returned by __array__.
context: tuple of (func, tuple, int)
    This parameter is returned by ufuncs as a 3-element tuple: (name of
the
    ufunc, arguments of the ufunc, domain of the ufunc), but is not set by
    other numpy functions.q

Notes
-----
Series implements __array_ufunc__ so this not called for ufunc on Series.

__bool__ = __nonzero__(self)

__contains__(self, key) -> 'bool_t'
    True if the key is in the info axis

__copy__(self: 'NDFrameT', deep: 'bool_t' = True) -> 'NDFrameT'

__deepcopy__(self: 'NDFrameT', memo=None) -> 'NDFrameT'
    Parameters
    -----
    memo, default None
        Standard signature. Unused

__delitem__(self, key) -> 'None'
    Delete item

__finalize__(self: 'NDFrameT', other, method: 'str | None' = None, **kwargs) -
> 'NDFrameT'
    Propagate metadata from other to self.

    Parameters
    -----
    other : the object from which to get the attributes that we are going
        to propagate
    method : str, optional
        A passed method name providing context on where ``__finalize__``
        was called.

    ... warning::
        The value passed as `method` are not currently considered
        stable across pandas releases.

__getattr__(self, name: 'str')
    After regular attribute access, try looking up the name
    This allows simpler access to columns for interactive use.

__getstate__(self) -> 'dict[str, Any]'

__iadd__(self, other)

__iand__(self, other)

__ifloordiv__(self, other)

__imod__(self, other)

__imul__(self, other)

__invert__(self)

__ior__(self, other)

__ipow__(self, other)

__isub__(self, other)

__iter__(self)
    Iterate over info axis.

    Returns
    -----
    iterator
        Info axis as iterator.

__itruediv__(self, other)

__ixor__(self, other)

__neg__(self)

__nonzero__(self)

__pos__(self)

__round__(self: 'NDFrameT', decimals: 'int' = 0) -> 'NDFrameT'

__setattr__(self, name: 'str', value) -> 'None'
    After regular attribute access, try setting the name
    This allows simpler access to columns for interactive use.

__setstate__(self, state)

abs(self: 'NDFrameT') -> 'NDFrameT'
    Return a Series/DataFrame with absolute numeric value of each element.

    This function only applies to elements that are all numeric.

    Returns
    -----
    abs
        Series/DataFrame containing the absolute value of each element.

See Also
--------
numpy.absolute : Calculate the absolute value element-wise.

Notes
-----
For ``complex`` inputs, ``1.2 + 1j``, the absolute value is
:math:`\sqrt{a^2 + b^2}`.

Examples
-----
Absolute numeric values in a Series.

>>> s = pd.Series([-1.10, 2, -3.33, 4])
>>> s.abs()
0    1.10
1    2.00
2    3.33
3    4.00
dtype: float64

Absolute numeric values in a Series with complex numbers.

>>> s = pd.Series([1.2 + 1j])
>>> s.abs()
0    1.56205

```

```

dtype: float64
Absolute numeric values in a Series with a Timedelta element.

>>> s = pd.Series([pd.Timedelta('1 days')])
>>> s.abs()
0   1 days
dtype: timedelta64[ns]

Select rows with data closest to certain value using argsort (from
`StackOverflow <https://stackoverflow.com/a/17758115>`__).

>>> df = pd.DataFrame({
...     'a': [4, 5, 6, 7],
...     'b': [10, 20, 30, 40],
...     'c': [100, 50, -30, -50]
... })
>>> df
   a    b    c
0  4   10  100
1  5   20   50
2  6   30  -30
3  7   40  -50
>>> df.loc[(df.c - 43).abs().argsort()]
   a    b    c
1  5   20   50
0  4   10  100
2  6   30  -30
3  7   40  -50

add_prefix(self: 'NDFrameT', prefix: 'str') -> 'NDFrameT'
Prefix labels with string `prefix`.

For Series, the row labels are prefixed.
For DataFrame, the column labels are prefixed.

Parameters
-----
prefix : str
    The string to add before each label.

Returns
-----
Series or DataFrame
    New Series or DataFrame with updated labels.

See Also
-----
Series.add_suffix: Suffix row labels with string `suffix`.
DataFrame.add_suffix: Suffix column labels with string `suffix`.

Examples
-----
>>> s = pd.Series([1, 2, 3, 4])
>>> s
0    1
1    2
2    3
3    4
dtype: int64

>>> s.add_prefix('item_')
item_0    1
item_1    2
item_2    3
item_3    4
dtype: int64

>>> df = pd.DataFrame({'A': [1, 2, 3, 4], 'B': [3, 4, 5, 6]})
>>> df
   A   B
0  1   3
1  2   4
2  3   5
3  4   6

>>> df.add_prefix('col_')
   col_A   col_B
0        1        3
1        2        4
2        3        5
3        4        6

add_suffix(self: 'NDFrameT', suffix: 'str') -> 'NDFrameT'
Suffix labels with string `suffix`.

For Series, the row labels are suffixed.
For DataFrame, the column labels are suffixed.

Parameters
-----
suffix : str
    The string to add after each label.

Returns
-----
Series or DataFrame
    New Series or DataFrame with updated labels.

See Also
-----
Series.add_prefix: Prefix row labels with string `prefix`.
DataFrame.add_prefix: Prefix column labels with string `prefix`.

Examples
-----
>>> s = pd.Series([1, 2, 3, 4])
>>> s
0    1
1    2
2    3
3    4
dtype: int64

>>> s.add_suffix('_item')
0_item    1
1_item    2
2_item    3
3_item    4
dtype: int64

>>> df = pd.DataFrame({'A': [1, 2, 3, 4], 'B': [3, 4, 5, 6]})
>>> df
   A   B
0  1   3
1  2   4
2  3   5
3  4   6

>>> df.add_suffix('_col')
   A_col   B_col

```

```

0      1      3
1      2      4
2      3      5
3      4      6

asof(self, where, subset=None)
    Return the last row(s) without any NaNs before `where`.

    The last row (for each element in `where`, if list) without any
    NaN is taken.
    In case of a :class:`~pandas.DataFrame`, the last row without NaN
    considering only the subset of columns (if not `None`)

    If there is no good value, NaN is returned for a Series or
    a Series of NaN values for a DataFrame

Parameters
-----
where : date or array-like of dates
    Date(s) before which the last row(s) are returned.
subset : str or array-like of str, default 'None'
    For DataFrame, if not 'None', only use these columns to
    check for NaNs.

Returns
-----
scalar, Series, or DataFrame

The return can be:
* scalar : when 'self' is a Series and 'where' is a scalar
* Series: when 'self' is a Series and 'where' is an array-like,
  or when 'self' is a DataFrame and 'where' is a scalar
* Dataframe : when 'self' is a DataFrame and 'where' is an
  array-like

Return scalar, Series, or DataFrame.

See Also
-----
merge_asof : Perform an asof merge. Similar to left join.

Notes
-----
Dates are assumed to be sorted. Raises if this is not the case.

Examples
-----
A Series and a scalar 'where'.

>>> s = pd.Series([1, 2, np.nan, 4], index=[10, 20, 30, 40])
>>> s
10    1.0
20    2.0
30    NaN
40    4.0
dtype: float64

>>> s.asof(20)
2.0

For a sequence 'where', a Series is returned. The first value is
NaN, because the first element of 'where' is before the first
index value.

>>> s.asof([5, 20])
5    NaN
20    2.0
dtype: float64

Missing values are not considered. The following is ``2.0'', not
NaN, even though NaN is at the index location for ``30''.

>>> s.asof(30)
2.0

Take all columns into consideration

>>> df = pd.DataFrame({'a': [10, 20, 30, 40, 50],
...                     'b': [None, None, None, None, 500]},
...                     index=pd.DatetimeIndex(['2018-02-27 09:01:00',
...                                         '2018-02-27 09:02:00',
...                                         '2018-02-27 09:03:00',
...                                         '2018-02-27 09:04:00',
...                                         '2018-02-27 09:05:00']))
>>> df.asof(pd.DatetimeIndex(['2018-02-27 09:03:30',
...                             '2018-02-27 09:04:30']))
   a    b
2018-02-27 09:03:30  NaN  NaN
2018-02-27 09:04:30  NaN  NaN

Take a single column into consideration

>>> df.asof(pd.DatetimeIndex(['2018-02-27 09:03:30',
...                             '2018-02-27 09:04:30']),
...             subset=['a'])
   a    b
2018-02-27 09:03:30  30.0  NaN
2018-02-27 09:04:30  40.0  NaN

astype(self: 'NDFrameT', dtype, copy: 'bool_t' = True, errors: 'str' =
'raise') -> 'NDFrameT'
    cast a pandas object to a specified dtype ``dtype``.

Parameters
-----
dtype : data type, or dict of column name -> data type
    Use a numpy.dtype or Python type to cast entire pandas object to
    the same type. Alternatively, use {col: dtype, ...}, where col is a
    column label and dtype is a numpy.dtype or Python type to cast one
    or more of the DataFrame's columns to column-specific types.

copy : bool, default True
    Return a copy when ``copy=True`` (be very careful setting
    ``copy=False`` as changes to values then may propagate to other
    pandas objects).

errors : {'raise', 'ignore'}, default 'raise'
    Control raising of exceptions on invalid data for provided dtype.

    - ``'raise'`` : allow exceptions to be raised
    - ``'ignore'`` : suppress exceptions. On error return original object.

Returns
-----
casted : same type as caller

See Also
-----
to_datetime : Convert argument to datetime.
to_timedelta : Convert argument to timedelta.
to_numeric : Convert argument to a numeric type.
numpy.ndarray.astype : Cast a numpy array to a specified type.

```

```

Notes
-----
... deprecated:: 1.3.0

    Using ``astype`` to convert from timezone-naive dtype to
    timezone-aware dtype is deprecated and will raise in a
    future version. Use :meth:`Series.dt.tz_localize` instead.

Examples
-----
Create a DataFrame:

>>> d = {'col1': [1, 2], 'col2': [3, 4]}
>>> df = pd.DataFrame(data=d)
>>> df.dtypes
col1    int64
col2    int64
dtype: object

Cast all columns to int32:

>>> df.astype('int32').dtypes
col1    int32
col2    int32
dtype: object

Cast col1 to int32 using a dictionary:

>>> df.astype({'col1': 'int32'}).dtypes
col1    int32
col2    int64
dtype: object

Create a series:

>>> ser = pd.Series([1, 2], dtype='int32')
>>> ser
0    1
1    2
dtype: int32
>>> ser.astype('int64')
0    1
1    2
dtype: int64

Convert to categorical type:

>>> ser.astype('category')
0    1
1    2
dtype: category
Categories (2, int64): [1, 2]

Convert to ordered categorical type with custom ordering:

>>> from pandas.api.types import CategoricalDtype
>>> cat_dtype = CategoricalDtype(
...     categories=[2, 1], ordered=True)
>>> ser.astype(cat_dtype)
0    1
1    2
dtype: category
Categories (2, int64): [2 < 1]

Note that using ``copy=False`` and changing data on a new
pandas object may propagate changes:

>>> s1 = pd.Series([1, 2])
>>> s2 = s1.astype('int64', copy=False)
>>> s2[0] = 10
>>> s1 # note that s1[0] has changed too
0    10
1    2
dtype: int64

Create a series of dates:

>>> ser_date = pd.Series(pd.date_range('20200101', periods=3))
>>> ser_date
0   2020-01-01
1   2020-01-02
2   2020-01-03
dtype: datetime64[ns]

at_time(self: 'NDFrame', time, asof: 'bool_t' = False, axis=None) ->
'NDFrameT'
| Select values at particular time of day (e.g., 9:30AM).

Parameters
-----
time : datetime.time or str
axis : {0 or 'index', 1 or 'columns'}, default 0

Returns
-----
Series or DataFrame

Raises
-----
TypeError
    If the index is not a :class:`DatetimeIndex`.

See Also
-----
between_time : Select values between particular times of the day.
first : Select initial periods of time series based on a date offset.
last : Select final periods of time series based on a date offset.
DatetimeIndex.indexer_at_time : Get just the index locations for
    values at particular time of the day.

Examples
-----
>>> i = pd.date_range('2018-04-09', periods=4, freq='12H')
>>> ts = pd.DataFrame({'A': [1, 2, 3, 4]}, index=i)
>>> ts
              A
2018-04-09 00:00:00  1
2018-04-09 12:00:00  2
2018-04-10 00:00:00  3
2018-04-10 12:00:00  4

>>> ts.at_time('12:00')
              A
2018-04-09 12:00:00  2
2018-04-10 12:00:00  4

backfill = bfill(self: 'NDFrameT', axis: 'None | Axis' = None, inplace:
'bool_t' = False, limit: 'None | int' = None, downcast=None) -> 'NDFrameT | None'
Synonym for :meth:`DataFrame.fillna` with ``method='bfill'``.

```

```

    Returns
    ----
    Series/DataFrame or None
        Object with missing values filled or None if ``inplace=True``.

    between_time(self: 'NDFrameT', start_time, end_time, include_start: 'bool_t' |
lib.NoDefault' <no_default>, include_end: 'bool_t' | lib.NoDefault' =
<no_default>, inclusive: 'str' | None' = None, axis=None) -> 'NDFrameT'
        Select values between particular times of the day (e.g., 9:00-9:30 AM).

        By setting ``start_time`` to be later than ``end_time``,
        you can get the times that are *not* between the two times.

    Parameters
    ----
    start_time : datetime.time or str
        Initial time as a time filter limit.
    end_time : datetime.time or str
        End time as a time filter limit.
    include_start : bool, default True
        Whether the start time needs to be included in the result.

        .. deprecated:: 1.4.0
            Arguments `include_start` and `include_end` have been deprecated
            to standardize boundary inputs. Use `inclusive` instead, to set
            each bound as closed or open.

    include_end : bool, default True
        Whether the end time needs to be included in the result.

        .. deprecated:: 1.4.0
            Arguments `include_start` and `include_end` have been deprecated
            to standardize boundary inputs. Use `inclusive` instead, to set
            each bound as closed or open.

    inclusive : {'both', 'neither', 'left', 'right'}, default "both"
        Include boundaries; whether to set each bound as closed or open.
    axis : {0 or 'index', 1 or 'columns'}, default 0
        Determine range time on index or columns value.

    Returns
    ----
    Series or DataFrame
        Data from the original object filtered to the specified dates range.

    Raises
    ----
    TypeError
        If the index is not a :class:`DatetimeIndex`.

    See Also
    ----
    at_time : Select values at a particular time of the day.
    first : Select initial periods of time series based on a date offset.
    last : Select final periods of time series based on a date offset.
    DatetimeIndex.indexer_between_time : Get just the index locations for
        values between particular times of the day.

    Examples
    ----
    >>> i = pd.date_range('2018-04-09', periods=4, freq='1D20min')
    >>> ts = pd.DataFrame({'A': [1, 2, 3, 4]}, index=i)
    >>> ts
              A
    2018-04-09 00:00:00  1
    2018-04-10 00:20:00  2
    2018-04-11 00:40:00  3
    2018-04-12 01:00:00  4

    >>> ts.between_time('0:15', '0:45')
              A
    2018-04-10 00:20:00  2
    2018-04-11 00:40:00  3

    You get the times that are *not* between two times by setting
    ``start_time`` later than ``end_time``:

    >>> ts.between_time('0:45', '0:15')
              A
    2018-04-09 00:00:00  1
    2018-04-12 01:00:00  4

    bool(self)
    Return the bool of a single element Series or DataFrame.

    This must be a boolean scalar value, either True or False. It will raise a
    ValueError if the Series or DataFrame does not have exactly 1 element, or
    that
    |   element is not boolean (integer values 0 and 1 will also raise an
    exception).
    |
    Returns
    ----
    bool
    The value in the Series or DataFrame.

    See Also
    ----
    Series.astype : Change the data type of a Series, including to boolean.
    DataFrame.astype : Change the data type of a DataFrame, including to
    boolean.
    numpy.bool_ : NumPy boolean data type, used by pandas for boolean values.

    Examples
    ----
    The method will only work for single element objects with a boolean value:

    >>> pd.Series([True]).bool()
    True
    >>> pd.Series([False]).bool()
    False

    >>> pd.DataFrame({'col': [True]}).bool()
    True
    >>> pd.DataFrame({'col': [False]}).bool()
    False

    convert_dtypes(self: 'NDFrameT', infer_objects: 'bool_t' = True,
convert_string: 'bool_t' = True, convert_integer: 'bool_t' = True,
convert_boolean: 'bool_t' = True, convert_floating: 'bool_t' = True) -> 'NDFrameT'
        Convert columns to best possible dtypes using dtypes supporting ``pd.NA``.

        .. versionadded:: 1.0.0

    Parameters
    ----
    infer_objects : bool, default True
        Whether object dtypes should be converted to the best possible types.
    convert_string : bool, default True
        Whether object dtypes should be converted to ``StringDtype()``.
    convert_integer : bool, default True
        Whether, if possible, conversion can be done to integer extension
        types.

```

```

convert_boolean : bool, defaults True
    Whether object dtypes should be converted to ``BooleanDtypes()``.
convert_floating : bool, defaults True
    Whether, if possible, conversion can be done to floating extension
types.
    If `convert_integer` is also True, preference will be give to integer
dtypes if the floats can be faithfully casted to integers.

    .. versionadded:: 1.2.0

Returns
-----
Series or DataFrame
    Copy of input object with new dtype.

See Also
-----
infer_objects : Infer dtypes of objects.
to_datetime : Convert argument to datetime.
to_timedelta : Convert argument to timedelta.
to_numeric : Convert argument to a numeric type.

Notes
-----
By default, ``convert_dtypes`` will attempt to convert a Series (or each
Series in a DataFrame) to dtypes that support ``pd.NA``. By using the
options ``convert_string``, ``convert_integer``, ``convert_boolean`` and
``convert_floating``, it is possible to turn off individual conversions
to ``StringDtype`` the integer extension types, ``BooleanDtype``
or floating extension types, respectively.

For object-dtyped columns, if ``infer_objects`` is ``True``, use the
inference rules as during normal Series/DataFrame construction. Then, if possible,
convert to ``StringDtype``, ``BooleanDtype`` or an appropriate integer
or floating extension type, otherwise leave as ``object``.

If the dtype is integer, convert to an appropriate integer extension type.

If the dtype is numeric, and consists of all integers, convert to an
appropriate integer extension type. Otherwise, convert to an
appropriate floating extension type.

    .. versionchanged:: 1.2
        Starting with pandas 1.2, this method also converts float columns
        to the nullable floating extension type.

In the future, as new dtypes are added that support ``pd.NA``, the results
of this method will change to support those new dtypes.

Examples
-----
>>> df = pd.DataFrame(
...     {
...         "a": pd.Series([1, 2, 3], dtype=np.dtype("int32")),
...         "b": pd.Series(["x", "y", "z"], dtype=np.dtype("O")),
...         "c": pd.Series([True, False, np.nan], dtype=np.dtype("O")),
...         "d": pd.Series(["h", "i", np.nan], dtype=np.dtype("O")),
...         "e": pd.Series([10, np.nan, 20], dtype=np.dtype("float")),
...         "f": pd.Series([np.nan, 100.5, 200], dtype=np.dtype("float")),
...     }
... )
Start with a DataFrame with default dtypes.

>>> df
   a   b      c   d   e      f
0  1   x    True   h  10.0   NaN
1  2   y   False   i   NaN  100.5
2  3   z     NaN  NaN  20.0  200.0

>>> df.dtypes
a    int32
b    object
c    object
d    object
e   float64
f   float64
dtype: object

Convert the DataFrame to use best possible dtypes.

>>> dfn = df.convert_dtypes()
>>> dfn
   a   b      c   d      e      f
0  1   x    True   h    10    <NA>
1  2   y   False   i    <NA>  100.5
2  3   z     NaN  <NA>    20  200.0

>>> dfn.dtypes
a    Int32
b    string
c    boolean
d    string
e    Int64
f    Float64
dtype: object

Start with a Series of strings and missing data represented by ``np.nan``.

>>> s = pd.Series(["a", "b", np.nan])
>>> s
0    a
1    b
2    NaN
dtype: object

Obtain a Series with dtype ``StringDtype``.

>>> s.convert_dtypes()
0    a
1    b
2    <NA>
dtype: string

copy(self: 'NDFrameT', deep: 'bool_` = True) -> 'NDFrameT'
    Make a copy of this object's indices and data.

    When ``deep=True`` (default), a new object will be created with a
    copy of the calling object's data and indices. Modifications to
    the data or indices of the copy will not be reflected in the
    original object (see notes below).

    When ``deep=False`` , a new object will be created without copying
    the calling object's data or index (only references to the data
    and index are copied). Any changes to the data of the original
    will be reflected in the shallow copy (and vice versa).

Parameters
-----

```

```

    deep : bool, default True
        Make a deep copy, including a copy of the data and the indices.
        With ``deep=False`` neither the indices nor the data are copied.

    Returns
    ------
    copy : Series or DataFrame
        Object type matches caller.

    Notes
    -----
    When ``deep=True``, data is copied but actual Python objects
    will not be copied recursively, only the reference to the object.
    This is in contrast to `copy.deepcopy` in the Standard Library,
    which recursively copies object data (see examples below).

    While ``Index`` objects are copied when ``deep=True``, the underlying
    numpy array is not copied for performance reasons. Since ``Index`` is
    immutable, the underlying data can be safely shared and a copy
    is not needed.

    Examples
    -----
    >>> s = pd.Series([1, 2], index=["a", "b"])
    >>> s
    a    1
    b    2
    dtype: int64

    >>> s_copy = s.copy()
    >>> s_copy
    a    1
    b    2
    dtype: int64

    **Shallow copy versus default (deep) copy:**

    >>> s = pd.Series([1, 2], index=["a", "b"])
    >>> deep = s.copy()
    >>> shallow = s.copy(deep=False)

    Shallow copy shares data and index with original.

    >>> s is shallow
    False
    >>> s.values is shallow.values and s.index is shallow.index
    True

    Deep copy has own copy of data and index.

    >>> s is deep
    False
    >>> s.values is deep.values or s.index is deep.index
    False

    Updates to the data shared by shallow copy and original is reflected
    in both; deep copy remains unchanged.

    >>> s[0] = 3
    >>> shallow[1] = 4
    >>> s
    a    3
    b    4
    dtype: int64
    >>> shallow
    a    3
    b    4
    dtype: int64
    >>> deep
    a    1
    b    2
    dtype: int64

    Note that when copying an object containing Python objects, a deep copy
    will copy the data, but will not do so recursively. Updating a nested
    data object will be reflected in the deep copy.

    >>> s = pd.Series([[1, 2], [3, 4]])
    >>> deep = s.copy()
    >>> s[0][0] = 10
    >>> s
    0    [10, 2]
    1    [3, 4]
    dtype: object
    >>> deep
    0    [10, 2]
    1    [3, 4]
    dtype: object

    describe(self: 'NDFrameT', percentiles=None, include=None, exclude=None,
    datetimelike_is_numeric=False) -> 'NDFrameT'
        Generate descriptive statistics.

        Descriptive statistics include those that summarize the central
        tendency, dispersion and shape of a
        dataset's distribution, excluding ``NaN`` values.

        Analyzes both numeric and object series, as well
        as ``DataFrame`` column sets of mixed data types. The output
        will vary depending on what is provided. Refer to the notes
        below for more detail.

    Parameters
    -----
    percentiles : list-like of numbers, optional
        The percentiles to include in the output. All should
        fall between 0 and 1. The default is
        ``[.25, .5, .75]``, which returns the 25th, 50th, and
        75th percentiles.
    include : 'all', list-like of dtypes or None (default), optional
        A white list of data types to include in the result. Ignored
        for ``Series``. Here are the options:
        - 'all' : All columns of the input will be included in the output.
        - A list-like of dtypes : Limits the results to the
          provided data types.
        To limit the result to numeric types submit
        ``'numpy.number'``. To limit it instead to object columns submit
        the ``'numpy.object'`` data type. Strings
        can also be used in the style of
        ``'select_dtypes'`` (e.g. ``df.describe(include=['O'])``). To
        select pandas categorical columns, use ``'category'``.
    exclude : list-like of dtypes or None (default), optional,
        A black list of data types to omit from the result. Ignored
        for ``Series``. Here are the options:
        - A list-like of dtypes : Excludes the provided data types
          from the result. To exclude numeric types submit
          ``'numpy.number'``. To exclude object columns submit the data
          type ``'numpy.object'``. Strings can also be used in the style of

```

```

    ``select_dtypes`` (e.g. ``df.describe(exclude=['O'])``). To
    exclude pandas categorical columns, use ``'category'``.
- None (default) : The result will exclude nothing.
datetime_is_numeric : bool, default False
    Whether to treat datetime dtypes as numeric. This affects statistics
    calculated for the column. For DataFrame input, this also
    controls whether datetime columns are included by default.

.. versionadded:: 1.1.0

>Returns
-----
Series or DataFrame
    Summary statistics of the Series or Dataframe provided.

See Also
-----
DataFrame.count: Count number of non-NA/null observations.
DataFrame.max: Maximum of the values in the object.
DataFrame.min: Minimum of the values in the object.
DataFrame.mean: Mean of the values.
DataFrame.std: Standard deviation of the observations.
DataFrame.select_dtypes: Subset of a DataFrame including/excluding
    columns based on their dtype.

Notes
-----
For numeric data, the result's index will include ``count``,
``mean``, ``std``, ``min``, ``max`` as well as lower, ``50`` and
upper percentiles. By default the lower percentile is ``25`` and the
upper percentile is ``75``. The ``50`` percentile is the
same as the median.

For object data (e.g. strings or timestamps), the result's index
will include ``count``, ``unique``, ``top``, and ``freq``. The ``top``
is the most common value. The ``freq`` is the most common value's
frequency. Timestamps also include the ``first`` and ``last`` items.

If multiple object values have the highest count, then the
``count`` and ``top`` results will be arbitrarily chosen from
among those with the highest count.

For mixed data types provided via a ``DataFrame``, the default is to
return only an analysis of numeric columns. If the dataframe consists
only of object and categorical data without any numeric columns, the
default is to return an analysis of both the object and categorical
columns. If ``include='all'`` is provided as an option, the result
will include a union of attributes of each type.

The ``include`` and ``exclude`` parameters can be used to limit
which columns in a ``DataFrame`` are analyzed for the output.
The parameters are ignored when analyzing a ``Series``.

Examples
-----
Describing a numeric ``Series``.

>>> s = pd.Series([1, 2, 3])
>>> s.describe()
count    3.0
mean     2.0
std      1.0
min     1.0
25%    1.5
50%    2.0
75%    2.5
max     3.0
dtype: float64

Describing a categorical ``Series``.

>>> s = pd.Series(['a', 'a', 'b', 'c'])
>>> s.describe()
count    4
unique   3
top     a
freq    2
dtype: object

Describing a timestamp ``Series``.

>>> s = pd.Series([
...     np.datetime64("2000-01-01"),
...     np.datetime64("2010-01-01"),
...     np.datetime64("2010-01-01")
... ])
>>> s.describe(datetime_is_numeric=True)
count    3
mean    2006-09-01 08:00:00
min    2006-01-01 00:00:00
25%    2004-12-31 12:00:00
50%    2010-01-01 00:00:00
75%    2010-01-01 00:00:00
max    2010-01-01 00:00:00
dtype: object

Describing a ``DataFrame``. By default only numeric fields
are returned.

>>> df = pd.DataFrame({'categorical': pd.Categorical(['d', 'e', 'f']),
...                      'numeric': [1, 2, 3],
...                      'object': ['a', 'b', 'c']}
... )
>>> df.describe()
        numeric
count    3.0
mean     2.0
std      1.0
min     1.0
25%    1.5
50%    2.0
75%    2.5
max     3.0

Describing all columns of a ``DataFrame`` regardless of data type.

>>> df.describe(include='all') # doctest: +SKIP
            categorical  numeric  object
count    3    3.0    3
unique   3    NaN    3
top      f    NaN    a
freq     1    NaN    1
mean    NaN    2.0   NaN
std     NaN    1.0   NaN
min    NaN    1.0   NaN
25%    NaN    1.5   NaN
50%    NaN    2.0   NaN
75%    NaN    2.5   NaN
max    NaN    3.0   NaN

Describing a column from a ``DataFrame`` by accessing it as

```

```

an attribute.

>>> df.numeric.describe()
count    3.0
mean     2.0
std      1.0
min     1.0
25%    1.5
50%    2.0
75%    2.5
max     3.0
Name: numeric, dtype: float64

Including only numeric columns in a ``DataFrame`` description.

>>> df.describe(include=[np.number])
   numeric
count    3.0
mean     2.0
std      1.0
min     1.0
25%    1.5
50%    2.0
75%    2.5
max     3.0

Including only string columns in a ``DataFrame`` description.

>>> df.describe(include=[object]) # doctest: +SKIP
   object
count     3
unique     3
top       a
freq      1

Including only categorical columns from a ``DataFrame`` description.

>>> df.describe(include=['category'])
   categorical
count     3
unique     3
top       d
freq      1

Excluding numeric columns from a ``DataFrame`` description.

>>> df.describe(exclude=[np.number]) # doctest: +SKIP
   categorical object
count     3     3
unique     3     3
top       f     a
freq      1     1

Excluding object columns from a ``DataFrame`` description.

>>> df.describe(exclude=[object]) # doctest: +SKIP
   categorical numeric
count     3     3.0
unique     3     NaN
top       f     NaN
freq      1     NaN
mean      NaN    2.0
std       NaN    1.0
min      NaN    1.0
25%      NaN    1.5
50%      NaN    2.0
75%      NaN    2.5
max      NaN    3.0

droplevel(self: 'NDFrameT', level, axis=0) -> 'NDFrameT'
Return Series/DataFrame with requested index / column level(s) removed.

Parameters
-----
level : int, str, or list-like
    If a string is given, must be the name of a level
    If list-like, elements must be names or positional indexes
    of levels.

axis : {0 or 'index', 1 or 'columns'}, default 0
    Axis along which the level(s) is removed:
        * 0 or 'index': remove level(s) in column.
        * 1 or 'columns': remove level(s) in row.

Returns
-----
Series/DataFrame
    Series/DataFrame with requested index / column level(s) removed.

Examples
-----
>>> df = pd.DataFrame([
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12]
... ]).set_index([0, 1]).rename_axis(['a', 'b'])

>>> df.columns = pd.MultiIndex.from_tuples([
...     ('c', 'e'), ('d', 'f')
... ], names=['level_1', 'level_2'])

>>> df
level_1  c  d
level_2  e  f
a b
1 2    3  4
5 6    7  8
9 10   11 12

>>> df.droplevel('a')
level_1  c  d
level_2  e  f
b
2      3  4
6      7  8
10     11 12

>>> df.droplevel('level_2', axis=1)
level_1  c  d
a b
1 2    3  4
5 6    7  8
9 10   11 12

equals(self, other: 'object') -> 'bool_t'
Test whether two objects contain the same elements.

This function allows two Series or DataFrames to be compared against
each other to see if they have the same shape and elements. NaNs in
the same location are considered equal.

```

The row/column index do not need to have the same type, as long as the values are considered equal. Corresponding columns must be of the same dtype.

Parameters

other : Series or DataFrame
The other Series or DataFrame to be compared with the first.

Returns

bool
True if all elements are the same in both objects, False otherwise.

See Also

Series.eq : Compare two Series objects of the same length and return a Series where each element is True if the element in each Series is equal, False otherwise.

DataFrame.eq : Compare two DataFrame objects of the same shape and return a DataFrame where each element is True if the respective element in each DataFrame is equal, False otherwise.

testing.assert_series_equal : Raises an AssertionError if left and right are not equal. Provides an easy interface to ignore inequality in dtypes, indexes and precision among others.

testing.assert_frame_equal : Like assert_series_equal, but targets Dataframes.

numpy.array_equal : Return True if two arrays have the same shape and elements, False otherwise.

Examples

```
>>> df = pd.DataFrame({1: [10], 2: [20]})  
>>> df  
   1  2  
0  10  20  
  
DataFrames df and exactly_equal have the same types and values for their elements and column labels, which will return True.
```

```
>>> exactly_equal = pd.DataFrame({1: [10], 2: [20]})  
>>> exactly_equal  
   1  2  
0  10  20  
>>> df.equals(exactly_equal)  
True
```

DataFrames df and different_column_type have the same element types and values, but have different types for the column labels, which will still return True.

```
>>> different_column_type = pd.DataFrame({1.0: [10], 2.0: [20]})  
>>> different_column_type  
   1.0  2.0  
0    10    20  
>>> df.equals(different_column_type)  
True
```

DataFrames df and different_data_type have different types for the same values for their elements, and will return False even though their column labels are the same values and types.

```
>>> different_data_type = pd.DataFrame({1: [10.0], 2: [20.0]})  
>>> different_data_type  
   1  2  
0  10.0  20.0  
>>> df.equals(different_data_type)  
False
```

```
| ewm(self, com: 'float | None' = None, span: 'float | None' = None, halflife:  
'float | Timedelta | convertible types | None' = None, alpha: 'float | None' = None,  
min_periods: 'int | None' = 0, adjust: 'bool_t' = True, ignore_na: 'bool_t' =  
False, axis: 'axis' = 0, times: 'str | np.ndarray | DataFrame | Series | None' =  
None, method: 'str' = 'single') -> 'ExponentialMovingWindow'  
|     Provide exponentially weighted (EW) calculations.  
|  
be  
| Exactly one parameter: ``com``, ``span``, ``halflife``, or ``alpha`` must  
| provided.  
|  
Parameters  
-----  
com : float, optional  
    Specify decay in terms of center of mass  
    :math:`\alpha = 1 / (1 + com)`, for :math:`com \geq 0`.  
span : float, optional  
    Specify decay in terms of span  
    :math:`\alpha = 2 / (span + 1)`, for :math:`span \geq 1`.  
halflife : float, str, timedelta, optional  
    Specify decay in terms of half-life  
    :math:`\alpha = 1 - \exp(-\ln(2) / halflife)`, for  
    :math:`halflife > 0`.  
If ``times`` is specified, the time unit (str or timedelta) over which  
an observation decays to half its value. Only applicable to ``mean()``,  
and halflife value will not apply to the other functions.  
.. versionadded:: 1.1.0  
alpha : float, optional  
    Specify smoothing factor :math:`\alpha` directly  
    :math:`0 < \alpha \leq 1`.  
min_periods : int, default 0  
    Minimum number of observations in window required to have a value;  
    otherwise, result is ``np.nan``.  
adjust : bool, default True  
    Divide by decaying adjustment factor in beginning periods to account  
    for imbalance in relative weightings (viewing EWMA as a moving  
average).  
weights  
|     - When ``adjust=True`` (default), the EW function is calculated using  
|       :math:w_i = (1 - \alpha)^i. For example, the EW moving average of  
the series  
|       [:math:x_0, x_1, ..., x_t] would be:  
|  
|       .. math::  
|           y_t = \frac{x_t + (1 - \alpha)x_{t-1} + (1 - \alpha)^2 x_{t-2} +  
... + (1 - \alpha)^t x_0}{1 + (1 - \alpha) + (1 - \alpha)^2 + ... + (1 - \alpha)^t}
```

```

\alpha)^t}
| |
| - When ``adjust=False``, the exponentially weighted function is
calculated recursively:
|
| .. math::
|   \begin{aligned}
|     y_0 &= x_0 \\
|     y_t &= (1 - \alpha) y_{t-1} + \alpha x_t,
|   \end{aligned}
| \end{aligned}
| ignore_na : bool, default False
| Ignore missing values when calculating weights.
|
| - When ``ignore_na=False`` (default), weights are based on absolute
positions.
| For example, the weights of :math:`x_0` and :math:`x_2` used in
calculating the final weighted average of [:math:`x_0`], None, [:math:`x_2`]] are
:math:`(1-\alpha)^2` and :math:`1` if ``adjust=True``; and
:math:`(1-\alpha)^2` and :math:`\alpha` if ``adjust=False``.
|
| - When ``ignore_na=True`` , weights are based
on relative positions. For example, the weights of :math:`x_0` and
:math:`x_2` used in calculating the final weighted average of
[:math:`x_0`], None, [:math:`x_2`]] are :math:`1-\alpha` and :math:`1`
if
|   ``adjust=True`` , and :math:`1-\alpha` and :math:`\alpha` if
``adjust=False``.
|
| axis : {0, 1}, default 0
| If ``0`` or ``index``, calculate across the rows.
|
| If ``1`` or ``columns``, calculate across the columns.
|
times : str, np.ndarray, Series, default None
|
| .. versionadded:: 1.1.0
|
| Only applicable to ``mean()``.
|
| Times corresponding to the observations. Must be monotonically
increasing and
| ``datetime64[ns]`` dtype.
|
| If 1-D array like, a sequence with the same shape as the observations.
|
| .. deprecated:: 1.4.0
|   If str, the name of the column in the DataFrame representing the
times.
|
method : str {'single', 'table'}, default 'single'
| .. versionadded:: 1.4.0
|
| Execute the rolling operation per single column or row (``'single'``)
| or over the entire object (``'table'``).
|
| This argument is only implemented when specifying ``engine='numba'``
in the method call.
|
| Only applicable to ``mean()``.
|
Returns
-----
`ExponentialMovingWindow` subclass
|
See Also
-----
rolling : Provides rolling window calculations.
expanding : Provides expanding transformations.
|
Notes
-----
See :ref:`Windowing Operations <window.exponentially_weighted>` for further usage details and examples.
|
Examples
-----
>>> df = pd.DataFrame({'B': [0, 1, 2, np.nan, 4]})
>>> df
   B
0  0.0
1  1.0
2  2.0
3  NaN
4  4.0

>>> df.ewm(com=0.5).mean()
   B
0  0.000000
1  0.750000
2  1.615385
3  1.615385
4  3.670213

>>> df.ewm(alpha=2 / 3).mean()
   B
0  0.000000
1  0.750000
2  1.615385
3  1.615385
4  3.670213

**adjust**

>>> df.ewm(com=0.5, adjust=True).mean()
   B
0  0.000000
1  0.750000
2  1.615385
3  1.615385
4  3.670213

>>> df.ewm(com=0.5, adjust=False).mean()
   B
0  0.000000
1  0.666667
2  1.555556
3  1.555556
4  3.650794

**ignore_na**

>>> df.ewm(com=0.5, ignore_na=True).mean()
   B
0  0.000000
1  0.750000
2  1.615385
3  1.615385
4  3.225000

>>> df.ewm(com=0.5, ignore_na=False).mean()
   B

```

```

0 0.000000
1 0.750000
2 1.615385
3 1.615385
4 3.670213

**times**

Exponentially weighted mean with weights calculated with a timedelta
`halflife` relative to ``times``.

>>> times = ['2020-01-01', '2020-01-03', '2020-01-10', '2020-01-15',
'2020-01-17']
>>> df.ewm(halflife='4 days', times=pd.DatetimeIndex(times)).mean()
B
0 0.000000
1 0.585786
2 1.523889
3 1.523889
4 3.233686

expanding(self, min_periods: 'int' = 1, center: 'bool_t | None' = None, axis:
'axis' = 0, method: 'str' = 'single') -> 'Expanding'
Provide expanding window calculations.

Parameters
-----
min_periods : int, default 1
    Minimum number of observations in window required to have a value;
    otherwise, result is ``np.nan``.

center : bool, default False
    If False, set the window labels as the right edge of the window index.
    If True, set the window labels as the center of the window index.
    .. deprecated:: 1.1.0

axis : int or str, default 0
    If ``0`` or ``'index'``, roll across the rows.
    If ``1`` or ``'columns'``, roll across the columns.

method : str {'single', 'table'}, default 'single'
    Execute the rolling operation per single column or row (``'single'``)
    or over the entire object (``'table'``).

    This argument is only implemented when specifying ``engine='numba'``
    in the method call.

    .. versionadded:: 1.3.0

Returns
-----
`'Expanding'` subclass

See Also
-----
rolling : Provides rolling window calculations.
ewm : Provides exponential weighted functions.

Notes
-----
See :ref:`Windowing Operations <window.expanding>` for further usage
details and examples.

Examples
-----
>>> df = pd.DataFrame({'B': [0, 1, 2, np.nan, 4]})
>>> df
B
0 0.0
1 1.0
2 2.0
3 NaN
4 4.0

**min_periods**

Expanding sum with 1 vs 3 observations needed to calculate a value.

>>> df.expanding(1).sum()
B
0 0.0
1 1.0
2 3.0
3 3.0
4 7.0
>>> df.expanding(3).sum()
B
0 NaN
1 NaN
2 3.0
3 3.0
4 7.0

filter(self: 'NDFrameT', items=None, like: 'str | None' = None, regex: 'str | None' = None, axis=None) -> 'NDframeT'
| Subset the dataframe rows or columns according to the specified index
labels.
|
| Note that this routine does not filter a dataframe on its
contents. The filter is applied to the labels of the index.

Parameters
-----
items : list-like
    Keep labels from axis which are in items.
like : str
    Keep labels from axis for which "like in label == True".
regex : str (regular expression)
    Keep labels from axis for which re.search(regex, label) == True.
axis : {0 or 'index', 1 or 'columns', None}, default None
    The axis to filter on, expressed either as an index (int)
    or axis name (str). By default this is the info axis,
    'index' for Series, 'columns' for DataFrame.

Returns
-----
same type as input object

See Also
-----
DataFrame.loc : Access a group of rows and columns
    by label(s) or a boolean array.

Notes
-----
The ``'items'``, ``'like'``, and ``'regex'`` parameters are

```

enforced to be mutually exclusive.

`axis`` defaults to the info axis that is used when indexing with `[]``.

Examples

```
>>> df = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6]]),
...                   index=['mouse', 'rabbit'],
...                   columns=['one', 'two', 'three'])
>>> df
   one  two  three
mouse    1    2    3
rabbit   4    5    6

>>> # select columns by name
>>> df.filter(items=['one', 'three'])
   one  three
mouse    1    3
rabbit   4    6

>>> # select columns by regular expression
>>> df.filter(regex='e$', axis=1)
   one  three
mouse    1    3
rabbit   4    6

>>> # select rows containing 'bbi'
>>> df.filter(like='bbi', axis=0)
   one  two  three
rabbit   4    5    6
```

`first(self, offset) -> 'NDFrameT'`
Select initial periods of time series data based on a date offset.

When having a DataFrame with dates as index, this function can select the first few rows based on a date offset.

Parameters

```
offset : str, DateOffset or datetime.timedelta
    The offset length of the data that will be selected. For instance,
    '1M' will display all the rows having their index within the first
month.
```

Returns

```
Series or DataFrame
    A subset of the caller.
```

Raises

```
TypeError
    If the index is not a :class:`DatetimeIndex`
```

See Also

```
last : Select final periods of time series based on a date offset.
at_time : Select values at a particular time of the day.
between_time : Select values between particular times of the day.
```

Examples

```
>>> i = pd.date_range('2018-04-09', periods=4, freq='2D')
>>> ts = pd.DataFrame({'A': [1, 2, 3, 4]}, index=i)
>>> ts
   A
2018-04-09  1
2018-04-11  2
2018-04-13  3
2018-04-15  4

Get the rows for the first 3 days:
>>> ts.first('3D')
   A
2018-04-09  1
2018-04-11  2

Notice the data for 3 first calendar days were returned, not the first
3 days observed in the dataset, and therefore data for 2018-04-13 was
not returned.
```

`first_valid_index(self) -> 'Hashable | None'`
Return index for first non-NA value or None, if no non-NA value is found.

Returns

```
scalar : type of index
```

Notes

```
If all elements are non-NA/null, returns None.
Also returns None for empty Series/DataFrame.
```

`get(self, key, default=None)`
Get item from object for given key (ex: DataFrame column).

Returns default value if not found.

Parameters

```
key : object
```

Returns

```
value : same type as items contained in object
```

Examples

```
>>> df = pd.DataFrame(
...     [
...         [24.3, 75.7, "high"],
...         [31, 87.8, "high"],
...         [22, 71.6, "medium"],
...         [35, 95, "medium"],
...     ],
...     columns=["temp_celsius", "temp_fahrenheit", "windspeed"],
...     index=pd.date_range(start="2014-02-12", end="2014-02-15",
freq="D"),
... )
>>> df
   temp_celsius  temp_fahrenheit windspeed
2014-02-12      24.3          75.7    high
2014-02-13      31.0          87.8    high
2014-02-14      22.0          71.6  medium
2014-02-15      35.0          95.0  medium

>>> df.get(["temp_celsius", "windspeed"])
   temp_celsius windspeed
   temp_celsius windspeed
```

```

2014-02-12      24.3    high
2014-02-13      31.0    high
2014-02-14      22.0  medium
2014-02-15      35.0  medium

If the key isn't found, the default value will be used.

>>> df.get(["temp_celsius", "temp_kelvin"], default="default_value")
'default_value'

head(self: 'NDFrameT', n: 'int' = 5) -> 'NDFrameT'
Return the first `n` rows.

This function returns the first `n` rows for the object based
on position. It is useful for quickly testing if your object
has the right type of data in it.

For negative values of `n`, this function returns all rows except
the last `n` rows, equivalent to ``df[:-n]``.

Parameters
-----
n : int, default 5
    Number of rows to select.

Returns
-----
same type as caller
    The first `n` rows of the caller object.

See Also
-----
DataFrame.tail: Returns the last `n` rows.

Examples
-----
>>> df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',
...                                'monkey', 'parrot', 'shark', 'whale', 'zebra']}
>>> df
   animal
0 alligator
1 bee
2 falcon
3 lion
4 monkey
5 parrot
6 shark
7 whale
8 zebra

Viewing the first 5 lines

>>> df.head()
   animal
0 alligator
1 bee
2 falcon
3 lion
4 monkey

Viewing the first `n` lines (three in this case)

>>> df.head(3)
   animal
0 alligator
1 bee
2 falcon

For negative values of `n`

>>> df.head(-3)
   animal
0 alligator
1 bee
2 falcon
3 lion
4 monkey
5 parrot

infer_objects(self: 'NDFrameT') -> 'NDFrameT'
Attempt to infer better dtypes for object columns.

Attempts soft conversion of object-dtyped
columns, leaving non-object and unconvertible
columns unchanged. The inference rules are the
same as during normal Series/DataFrame construction.

Returns
-----
converted : same type as input object

See Also
-----
to_datetime : Convert argument to datetime.
to_timedelta : Convert argument to timedelta.
to_numeric : Convert argument to numeric type.
convert_dtypes : Convert argument to best possible dtype.

Examples
-----
>>> df = pd.DataFrame({"A": [1, 2, 3]})
>>> df = df.iloc[1:]
>>> df
   A
1  1
2  2
3  3

>>> df.dtypes
A    object
dtype: object

>>> df.infer_objects().dtypes
A    int64
dtype: object

keys(self)
Get the 'info axis' (see Indexing for more).

This is index for Series, columns for DataFrame.

Returns
-----
Index
    Info axis.

last(self: 'NDFrameT', offset) -> 'NDFrameT'
Select final periods of time series data based on a date offset.

For a DataFrame with a sorted DatetimeIndex, this function
selects the last few rows based on a date offset.

```

```

Parameters
-----
offset : str, DateOffset, dateutil.relativedelta
    The offset length of the data that will be selected. For instance,
    '3D' will display all the rows having their index within the last 3
days.

Returns
-----
Series or DataFrame
    A subset of the caller.

Raises
-----
TypeError
    If the index is not a :class:`DatetimeIndex`.

See Also
-----
first : Select initial periods of time series based on a date offset.
at_time : Select values at a particular time of the day.
between_time : Select values between particular times of the day.

Examples
-----
>>> i = pd.date_range('2018-04-09', periods=4, freq='2D')
>>> ts = pd.DataFrame({'A': [1, 2, 3, 4]}, index=i)
>>> ts
   A
2018-04-09  1
2018-04-11  2
2018-04-13  3
2018-04-15  4

Get the rows for the last 3 days:

>>> ts.last('3D')
   A
2018-04-13  3
2018-04-15  4

Notice the data for 3 last calendar days were returned, not the last
3 observed days in the dataset, and therefore data for 2018-04-11 was
not returned.

last_valid_index(self) -> 'Hashable | None'
Return index for last non-NA value or None, if no non-NA value is found.

Returns
-----
scalar : type of index

Notes
-----
If all elements are non-NA/null, returns None.
Also returns None for empty Series/DataFrame.

pad = ffill(self: 'NDFrameT', axis: 'None | Axis' = None, inplace: 'bool_t' =
False, limit: 'None | int' = None, downcast=None) -> 'NDFrameT | None'
Synonym for :meth:`DataFrame.fillna` with ``method='ffill'``.

Returns
-----
Series/DataFrame or None
    Object with missing values filled or None if ``inplace=True``.

pct_change(self: 'NDFrameT', periods=1, fill_method='pad', limit=None,
freq=None, **kwargs) -> 'NDFrameT'
    Percentage change between the current and a prior element.

    Computes the percentage change from the immediately previous row by
default. This is useful in comparing the percentage of change in a time
series of elements.

Parameters
-----
periods : int, default 1
    Periods to shift for forming percent change.
fill_method : str, default 'pad'
    How to handle NAs before computing percent changes.
limit : int, default None
    The number of consecutive NAs to fill before stopping.
freq : DateOffset, timedelta, or str, optional
    Increment to use from time series API (e.g. 'M' or BDay()).
**kwargs
    Additional keyword arguments are passed into
    `DataFrame.shift` or `Series.shift`.

Returns
-----
chg : Series or DataFrame
    The same type as the calling object.

See Also
-----
Series.diff : Compute the difference of two elements in a Series.
DataFrame.diff : Compute the difference of two elements in a DataFrame.
Series.shift : Shift the index by some number of periods.
DataFrame.shift : Shift the index by some number of periods.

Examples
-----
**Series**

>>> s = pd.Series([90, 91, 85])
>>> s
0    90
1    91
2    85
dtype: int64

>>> s.pct_change()
0      NaN
1    0.01111
2   -0.065934
dtype: float64

>>> s.pct_change(periods=2)
0      NaN
1      NaN
2   -0.055556
dtype: float64

See the percentage change in a Series where filling NAs with last
valid observation forward to next valid.

>>> s = pd.Series([90, 91, None, 85])
>>> s
0    90.0
1    91.0

```

```

2      NaN
3     85.0
dtype: float64

>>> s.pct_change(fill_method='ffill')
0      NaN
1   0.011111
2   0.000000
3  -0.065934
dtype: float64

**DataFrame**

Percentage change in French franc, Deutsche Mark, and Italian lira from
1980-01-01 to 1980-03-01.

>>> df = pd.DataFrame({
...     'FR': [4.0495, 4.0963, 4.3149],
...     'GR': [1.7246, 1.7482, 1.8519],
...     'IT': [804.74, 810.61, 860.13]],
...     index=['1980-01-01', '1980-02-01', '1980-03-01'])
>>> df
    FR      GR      IT
1980-01-01  4.0495  1.7246  804.74
1980-02-01  4.0963  1.7482  810.01
1980-03-01  4.3149  1.8519  860.13

>>> df.pct_change()
    FR      GR      IT
1980-01-01    NaN    NaN    NaN
1980-02-01  0.013810  0.013684  0.006549
1980-03-01  0.053365  0.059318  0.061876

Percentage of change in GOOG and APPL stock volume. Shows computing
the percentage change between columns.

>>> df = pd.DataFrame({
...     '2016': [1769950, 30586265],
...     '2015': [1500923, 40912316],
...     '2014': [1371819, 41403351],
...     index=['GOOG', 'APPL'])
>>> df
    2016      2015      2014
GOOG  1769950  1500923  1371819
APPL  30586265  40912316  41403351

>>> df.pct_change(axis='columns', periods=-1)
    2016      2015      2014
GOOG  0.179241  0.094112  NaN
APPL -0.252395 -0.011860  NaN

| pipe(self, func: 'Callable[..., T] | tuple[Callable[..., T], str]', *args,
**kwargs) -> 'T'
| Apply chainable functions that expect Series or DataFrames.

Parameters
-----
func : function
    Function to apply to the Series/DataFrame.
    ```args``', and ```kwargs``' are passed into ```func``'.
 Alternatively a ```(callable, data_keyword)`' tuple where
    ```data_keyword``' is a string indicating the keyword of
    ```callable`' that expects the Series/DataFrame.
args : iterable, optional
 Positional arguments passed into ```func``'.
kwargs : mapping, optional
 A dictionary of keyword arguments passed into ```func``'.

Returns

object : the return type of ```func``'.

See Also

DataFrame.apply : Apply a function along input axis of DataFrame.
DataFrame.applymap : Apply a function elementwise on a whole DataFrame.
Series.map : Apply a mapping correspondence on a
 :class:`pandas.Series`.

Notes

Use ```.pipe``` when chaining together functions that expect
Series, DataFrames or GroupBy objects. Instead of writing

>>> func(g(h(df)), arg1=a), arg2=b, arg3=c) # doctest: +SKIP

You can write

>>> (df.pipe(h)
... .pipe(g, arg1=a)
... .pipe(func, arg2=b, arg3=c)
...) # doctest: +SKIP

If you have a function that takes the data as (say) the second
argument, pass a tuple indicating which keyword expects the
data. For example, suppose ```f``' takes its data as ```arg2```:

>>> (df.pipe(h)
... .pipe(g, arg1=a)
... .pipe((func, 'arg2'), arg1=a, arg3=c)
...) # doctest: +SKIP

rank(self: 'NDFrameT', axis=0, method: 'str' = 'average', numeric_only:
'bool_t | None | lib.NoDefault' = <no_default>, na_option: 'str' = 'keep',
ascending: 'bool_t' = True, pct: 'bool_t' = False) -> 'NDFrameT'
 Compute numerical data ranks (1 through n) along axis.

By default, equal values are assigned a rank that is the average of the
ranks of those values.

Parameters

axis : {0 or 'index', 1 or 'columns'}, default 0
 Index to direct ranking.
method : {'average', 'min', 'max', 'first', 'dense'}, default 'average'
 How to rank the group of records that have the same value (i.e. ties):
 * average: average rank of the group
 * min: lowest rank in the group
 * max: highest rank in the group
 * first: ranks assigned in order they appear in the array
 * dense: like 'min', but rank always increases by 1 between groups.

numeric_only : bool, optional
 For DataFrame objects, rank only numeric columns if set to True.
na_option : {'keep', 'top', 'bottom'}, default 'keep'
 How to rank NaN values:
 * keep: assign NaN rank to NaN values
 * top: assign lowest rank to NaN values
 * bottom: assign highest rank to NaN values

```

```

 ascending : bool, default True
 Whether or not the elements should be ranked in ascending order.
 pct : bool, default False
 Whether or not to display the returned rankings in percentile
 form.

Returns

same type as caller
 Return a Series or DataFrame with data ranks as values.

See Also

core.groupby.GroupBy.rank : Rank of values within each group.

Examples

>>> df = pd.DataFrame(data={'Animal': ['cat', 'penguin', 'dog',
... 'spider', 'snake'],
... 'Number_legs': [4, 2, 4, 8, np.nan]})

>>> df
 Animal Number_legs
0 cat 4.0
1 penguin 2.0
2 dog 4.0
3 spider 8.0
4 snake NaN

The following example shows how the method behaves with the above
parameters:

* default_rank: this is the default behaviour obtained without using
any parameter.
* max_rank: setting ``method = 'max'`` the records that have the
same values are ranked using the highest rank (e.g.: since 'cat'
and 'dog' are both in the 2nd and 3rd position, rank 3 is assigned)
* NA_bottom: choosing ``na_option = 'bottom'``, if there are records
with NaN values they are placed at the bottom of the ranking.
* pct_rank: when setting ``pct = True``, the ranking is expressed as
percentile rank.

>>> df['default_rank'] = df['Number_legs'].rank()
>>> df['max_rank'] = df['Number_legs'].rank(method='max')
>>> df['NA_bottom'] = df['Number_legs'].rank(na_option='bottom')
>>> df['pct_rank'] = df['Number_legs'].rank(pct=True)
>>> df
 Animal Number_legs default_rank max_rank NA_bottom pct_rank
0 cat 4.0 2.5 3.0 2.5 0.625
1 penguin 2.0 1.0 1.0 1.0 0.250
2 dog 4.0 2.5 3.0 2.5 0.625
3 spider 8.0 4.0 4.0 4.0 1.000
4 snake NaN NaN NaN 5.0 NaN

| reindex_like(self: 'NDFrame', other, method: 'str | None' = None, copy:
| 'bool_t' = True, limit=None, tolerance=None) -> 'NDFrameT'
| Return an object with matching indices as other object.

Conform the object to the same index on all axes. Optional
filling logic, placing NaN in locations having no value
in the previous index. A new object is produced unless the
new index is equivalent to the current one and copy=False.

Parameters

other : Object of the same data type
 Its row and column indices are used to define the new indices
 of this object.
method : {None, 'backfill'/'bfill', 'pad'/'ffill', 'nearest'}
 Method to use for filling holes in reindexed DataFrame.
 Please note: this is only applicable to DataFrames/Series with a
 monotonically increasing/decreasing index.

 * None (default): don't fill gaps
 * pad / ffill: propagate last valid observation forward to next
 valid
 * backfill / bfill: use next valid observation to fill gap
 * nearest: use nearest valid observations to fill gap.

copy : bool, default True
 Return a new object, even if the passed indexes are the same.
limit : int, default None
 Maximum number of consecutive labels to fill for inexact matches.
tolerance : optional
 Maximum distance between original and new labels for inexact
 matches. The values of the index at the matching locations must
 satisfy the equation ``abs(index[indexer] - target) <= tolerance``.

 Tolerance may be a scalar value, which applies the same tolerance
 to all values, or list-like, which applies variable tolerance per
 element. List-like includes list, tuple, array, Series, and must be
 the same size as the index and its dtype must exactly match the
 index's type.

Returns

Series or DataFrame
 Same type as caller, but with changed indices on each axis.

See Also

DataFrame.set_index : Set row labels.
DataFrame.reset_index : Remove row labels or move them to new columns.
DataFrame.reindex : Change to new indices or expand indices.

Notes

Same as calling
```.reindex(index=other.index, columns=other.columns, ...)```.

Examples
-----
>>> df1 = pd.DataFrame([[24.3, 75.7, 'high'],
...                      [31, 87.8, 'high'],
...                      [22, 71.6, 'medium'],
...                      [35, 95, 'medium']],
...                     columns=['temp_celsius', 'temp_fahrenheit',
...                               'windspeed'],
...                     index=pd.date_range(start='2014-02-12',
...                                         end='2014-02-15', freq='D')))

>>> df1
   temp_celsius  temp_fahrenheit  windspeed
2014-02-12      24.3           75.7     high
2014-02-13      31.0           87.8     high
2014-02-14      22.0           71.6   medium
2014-02-15      35.0           95.0   medium

>>> df2 = pd.DataFrame([[28, 'low'],
...                      [30, 'low'],
...                      [35.1, 'medium']],
...

```

```

...
    columns=['temp_celsius', 'windspeed'],
    index=pd.DatetimeIndex(['2014-02-12', '2014-02-13',
                           '2014-02-15']))
...
>>> df2
      temp_celsius windspeed
2014-02-12      28.0     low
2014-02-13      30.0     low
2014-02-15      35.1   medium

>>> df2.reindex_like(df1)
      temp_celsius temp_fahrenheit windspeed
2014-02-12      28.0        NaN     low
2014-02-13      30.0        NaN     low
2014-02-14        NaN        NaN    NaN
2014-02-15      35.1        NaN   medium

rename_axis(self, mapper=None, index=None, columns=None, axis=None, copy=True,
inplace=False)
    Set the name of the axis for the index or columns.

Parameters
-----
mapper : scalar, list-like, optional
    Value to set the axis name attribute.
index, columns : scalar, list-like, dict-like or function, optional
    A scalar, list-like, dict-like or functions transformations to
    apply to that axis' values.
    Note that the ``columns`` parameter is not allowed if the
    object is a Series. This parameter only apply for DataFrame
    type objects.

    Use either ``mapper`` and ``axis`` to
    specify the axis to target with ``mapper``, or ``index``
    and/or ``columns``.
axis : {0 or 'index', 1 or 'columns'}, default 0
    The axis to rename.
copy : bool, default True
    Also copy underlying data.
inplace : bool, default False
    Modifies the object directly, instead of creating a new Series
    or DataFrame.

Returns
-----
Series, DataFrame, or None
    The same type as the caller or None if ``inplace=True``.

See Also
-----
Series.rename : Alter Series index labels or name.
DataFrame.rename : Alter DataFrame index labels or name.
Index.rename : Set new names on index.

Notes
-----
``DataFrame.rename_axis`` supports two calling conventions

* ``{(index=index_mapper, columns=columns_mapper, ...)}``
* ``{(mapper, axis={'index', 'columns'}, ...)}``

The first calling convention will only modify the names of
the index and/or the names of the Index object that is the columns.
In this case, the parameter ``copy`` is ignored.

The second calling convention will modify the names of the
corresponding index if mapper is a list or a scalar.
However, if mapper is dict-like or a function, it will use the
deprecated behavior of modifying the axis *labels*.

We *highly* recommend using keyword arguments to clarify your
intent.

Examples
-----
**Series**

>>> s = pd.Series(["dog", "cat", "monkey"])
>>> s
0      dog
1      cat
2  monkey
dtype: object
>>> s.rename_axis("animal")
animal
0      dog
1      cat
2  monkey
dtype: object

**DataFrame**

>>> df = pd.DataFrame({'num_legs': [4, 4, 2],
...                      'num_arms': [0, 0, 2]},
...                     ["dog", "cat", "monkey"])
>>> df
   num_legs  num_arms
dog         4          0
cat         4          0
monkey      2          2
>>> df = df.rename_axis("animal")
>>> df
   num_legs  num_arms
animal
dog         4          0
cat         4          0
monkey      2          2
>>> df = df.rename_axis("limbs", axis="columns")
>>> df
   limbs  num_legs  num_arms
animal
dog         4          0
cat         4          0
monkey      2          2

**MultiIndex**

>>> df.index = pd.MultiIndex.from_product([['mammal'],
...                                         ['dog', 'cat', 'monkey']],
...                                         names=['type', 'name'])
>>> df
   limbs  num_legs  num_arms
type  name
mammal dog         4          0
       cat         4          0
       monkey      2          2

>>> df.rename_axis(index={'type': 'class'})
   limbs  num_legs  num_arms
class  name
mammal dog         4          0

```

```

cat          4      0
monkey       2      2

>>> df.rename_axis(columns=str.upper)
LIMBS      num_legs  num_arms
type        name
mammal    dog        4      0
            cat        4      0
            monkey     2      2

rolling(self, window: 'int | timedelta | BaseOffset | BaseIndexer',
min_periods: 'int | None' = None, center: 'bool_t' = False, win_type: 'str | None'
= None, on: 'str | None' = None, axis: 'Axis' = 0, closed: 'str | None' = None,
method: 'str' = 'single')
| Provide rolling window calculations.

Parameters
-----
window : int, offset, or BaseIndexer subclass
Size of the moving window.

If an integer, the fixed number of observations used for
each window.

If an offset, the time period of each window. Each
window will be a variable sized based on the observations included in
the time-period. This is only valid for datetimelike indexes.
To learn more about the offsets & frequency strings, please see `this
link
<https://pandas.pydata.org/pandas-docs/stable/user\_guide/timeseries.html#offset-aliases>`__.

If a BaseIndexer subclass, the window boundaries
based on the defined `get_window_bounds` method. Additional rolling
keyword arguments, namely ``min_periods``, ``center``, and
``closed`` will be passed to ``get_window_bounds``.

min_periods : int, default None
Minimum number of observations in window required to have a value;
otherwise, result is ``np.nan``.

For a window that is specified by an offset, ``min_periods`` will
default to 1.

For a window that is specified by an integer, ``min_periods`` will
default
to the size of the window.

center : bool, default False
If False, set the window labels as the right edge of the window index.
If True, set the window labels as the center of the window index.

win_type : str, default None
If ``None``, all points are evenly weighted.

If a string, it must be a valid `scipy.signal` window function
<https://docs.scipy.org/doc/scipy/reference/signal.windows.html#module-scipy.signal.windows>`__.

Certain Scipy window types require additional parameters to be passed
in the aggregation function. The additional parameters must match
the keywords specified in the Scipy window type method signature.

on : str, optional
For a DataFrame, a column label or Index level on which
to calculate the rolling window, rather than the DataFrame's index.

Provided integer column is ignored and excluded from result since
an integer index is not used to calculate the rolling window.

axis : int or str, default 0
If ``0`` or ``'index'``, roll across the rows.
If ``1`` or ``'columns'``, roll across the columns.

closed : str, default None
If ``'right'``, the first point in the window is excluded from
calculations.
If ``'left'``, the last point in the window is excluded from
calculations.
If ``'both'``, the no points in the window are excluded from
calculations.
If ``'neither'``, the first and last points in the window are excluded
from calculations.

Default ``None`` (``'right'``).

.. versionchanged:: 1.2.0
The closed parameter with fixed windows is now supported.

method : str {'single', 'table'}, default 'single'
.. versionadded:: 1.3.0
Execute the rolling operation per single column or row (``'single'``)
or over the entire object (``'table'``).

This argument is only implemented when specifying ``engine='numba'``
in the method call.

Returns
-----
``Window`` subclass if a ``win_type`` is passed
``Rolling`` subclass if ``win_type`` is not passed

See Also
-----
expanding : Provides expanding transformations.
ewm : Provides exponential weighted functions.

Notes
-----
See :ref:`Windowing Operations <window.generic>` for further usage details
and examples.

Examples
-----
>>> df = pd.DataFrame({'B': [0, 1, 2, np.nan, 4]})
>>> df
   B
0  0.0
1  1.0
2  2.0
3  NaN

```

```

4 4.0
**window**
Rolling sum with a window length of 2 observations.

>>> df.rolling(2).sum()
   B
0  NaN
1  1.0
2  3.0
3  NaN
4  NaN

Rolling sum with a window span of 2 seconds.

>>> df_time = pd.DataFrame({'B': [0, 1, 2, np.nan, 4]}, index=[pd.Timestamp('20130101 09:00:00'), pd.Timestamp('20130101 09:00:02'), pd.Timestamp('20130101 09:00:03'), pd.Timestamp('20130101 09:00:05'), pd.Timestamp('20130101 09:00:06')])
>>> df_time
   B
2013-01-01 09:00:00  0.0
2013-01-01 09:00:02  1.0
2013-01-01 09:00:03  2.0
2013-01-01 09:00:05  NaN
2013-01-01 09:00:06  4.0

>>> df_time.rolling('2s').sum()
   B
2013-01-01 09:00:00  0.0
2013-01-01 09:00:02  1.0
2013-01-01 09:00:03  3.0
2013-01-01 09:00:05  NaN
2013-01-01 09:00:06  4.0

Rolling sum with forward looking windows with 2 observations.

>>> indexer = pd.api.indexers.FixedForwardWindowIndexer(window_size=2)
>>> df.rolling(window=indexer, min_periods=1).sum()
   B
0  1.0
1  3.0
2  2.0
3  4.0
4  4.0

**min_periods**

Rolling sum with a window length of 2 observations, but only needs a
minimum of 1
observation to calculate a value.

>>> df.rolling(2, min_periods=1).sum()
   B
0  0.0
1  1.0
2  3.0
3  2.0
4  4.0

**center**

Rolling sum with the result assigned to the center of the window index.

>>> df.rolling(3, min_periods=1, center=True).sum()
   B
0  1.0
1  3.0
2  3.0
3  6.0
4  4.0

>>> df.rolling(3, min_periods=1, center=False).sum()
   B
0  0.0
1  1.0
2  3.0
3  3.0
4  6.0

**win_type**

Rolling sum with a window length of 2, using the Scipy ``gaussian`` window type. ``std`` is required in the aggregation function.

>>> df.rolling(2, win_type='gaussian').sum(std=3)
   B
0  NaN
1  0.986207
2  2.958621
3  NaN
4  NaN

sample(self: 'NDFrameT', n: 'int | None' = None, frac: 'float | None' = None,
replace: 'bool_t' = False, weights=None, random_state: 'RandomState | None' = None,
axis: 'Axis | None' = None, ignore_index: 'bool_t' = False) -> 'NDFrameT'
| Return a random sample of items from an axis of object.

You can use `random_state` for reproducibility.

Parameters
-----
n : int, optional
    Number of items from axis to return. Cannot be used with `frac`.
    Default = 1 if `frac` = None.
frac : float, optional
    Fraction of axis items to return. Cannot be used with `n`.
replace : bool, default False
    Allow or disallow sampling of the same row more than once.
weights : str or ndarray-like, optional
    Default 'None' results in equal probability weighting.
    If passed a Series, will align with target object on index. Index
    values in weights not found in sampled object will be ignored and
    index values in sampled object not in weights will be assigned
    weights of zero.
    If called on a DataFrame, will accept the name of a column
    when axis = 0.
    Unless weights are a Series, weights must be same length as axis
    being sampled.
    If weights do not sum to 1, they will be normalized to sum to 1.
    Missing values in the weights column will be treated as zero.
    Infinite values not allowed.
random_state : int, array-like, BitGenerator, np.random.RandomState,
np.random.Generator, optional
    If int, array-like, or BitGenerator, seed for random number generator.
    If np.random.RandomState or np.random.Generator, use as given.

```

```

.. versionchanged:: 1.1.0
    array-like and BitGenerator object now passed to
np.random.RandomState()
        as seed

.. versionchanged:: 1.4.0
    np.random.Generator objects now accepted

axis : {0 or 'index', 1 or 'columns', None}, default None
    Axis to sample. Accepts axis number or name. Default is stat axis
    for given data type (0 for Series and DataFrames).
ignore_index : bool, default False
    If True, the resulting index will be labeled 0, 1, ..., n - 1.

.. versionadded:: 1.3.0

>Returns
-----
Series or DataFrame
    A new object of same type as caller containing `n` items randomly
    sampled from the caller object.

See Also
-----
DataFrameGroupBy.sample: Generates random samples from each group of a
    DataFrame object.
SeriesGroupBy.sample: Generates random samples from each group of a
    Series object.
numpy.random.choice: Generates a random sample from a given 1-D numpy
    array.

Notes
-----
If `frac` > 1, `replacement` should be set to `True`.

Examples
-----
>>> df = pd.DataFrame({'num_legs': [2, 4, 8, 0],
...                     'num_wings': [2, 0, 0, 0],
...                     'num_specimen_seen': [10, 2, 1, 8]},
...                     index=['falcon', 'dog', 'spider', 'fish'])
>>> df
   num_legs  num_wings  num_specimen_seen
falcon      2          2                  10
dog         4          0                  2
spider      8          0                  1
fish         0          0                  8

Extract 3 random elements from the ``Series`` ``df['num_legs']``:
Note that we use `random_state` to ensure the reproducibility of
the examples.

>>> df['num_legs'].sample(n=3, random_state=1)
fish      0
spider    8
falcon    2
Name: num_legs, dtype: int64

A random 50% sample of the ``DataFrame`` with replacement:

>>> df.sample(frac=0.5, replace=True, random_state=1)
   num_legs  num_wings  num_specimen_seen
dog         4          0                  2
fish         0          0                  8

An upsample sample of the ``DataFrame`` with replacement:
Note that `replace` parameter has to be `True` for `frac` parameter > 1.

>>> df.sample(frac=2, replace=True, random_state=1)
   num_legs  num_wings  num_specimen_seen
dog         4          0                  2
fish         0          0                  8
falcon      2          2                  10
falcon      2          2                  10
fish         0          0                  8
dog         4          0                  2
fish         0          0                  8
dog         4          0                  2

Using a DataFrame column as weights. Rows with larger value in the
`num_specimen_seen` column are more likely to be sampled.

>>> df.sample(n=2, weights='num_specimen_seen', random_state=1)
   num_legs  num_wings  num_specimen_seen
falcon      2          2                  10
fish         0          0                  8

set_flags(self: 'NDFrameT', *, copy: 'bool_t' = False,
allows_duplicate_labels: 'bool_t' | None = None) -> 'NDFrameT'
    Return a new object with updated flags.

>Returns
-----
Series or DataFrame
    The same type as the caller.

See Also
-----
DataFrame.attrs : Global metadata applying to this dataset.
DataFrame.flags : Global flags applying to this object.

Notes
-----
This method returns a new object that's a view on the same data
as the input. Mutating the input or the output values will be reflected
in the other.

This method is intended to be used in method chains.

"Flags" differ from "metadata". Flags reflect properties of the
pandas object (the Series or DataFrame). Metadata refer to properties
of the dataset, and should be stored in :attr:`DataFrame.attrs`.

Examples
-----
>>> df = pd.DataFrame({"A": [1, 2]})
>>> df.flags.allow_duplicates
True
>>> df2 = df.set_flags(allow_duplicates=False)
>>> df2.flags.allow_duplicates
False

slice_shift(self: 'NDFrameT', periods: 'int' = 1, axis=0) -> 'NDFrameT'
    Equivalent to `shift` without copying data.

```

The shifted data will not include the dropped periods and the shifted axis will be smaller than the original.

```
.. deprecated:: 1.2.0
    slice_shift is deprecated,
    use DataFrame/Series.shift instead.
```

Parameters

periods : int
Number of periods to move, can be positive or negative.

Returns

shifted : same type as caller

Notes
While the `slice_shift` is faster than `shift`, you may pay for it later during alignment.

squeeze(self, axis=None)
Squeeze 1 dimensional axis objects into scalars.

Series or DataFrames with a single element are squeezed to a scalar. DataFrames with a single column or a single row are squeezed to a Series. Otherwise the object is unchanged.

This method is most useful when you don't know if your object is a Series or DataFrame, but you do know it has just a single column. In that case you can safely call `squeeze` to ensure you have a Series.

Parameters

axis : {0 or 'index', 1 or 'columns', None}, default None
A specific axis to squeeze. By default, all length-1 axes are squeezed.

Returns

Dataframe, Series, or scalar
The projection after squeezing 'axis' or all the axes.

See Also

Series.iloc : Integer-location based indexing for selecting scalars.
DataFrame.iloc : Integer-location based indexing for selecting Series.
Series.to_frame : Inverse of DataFrame.squeeze for a single-column DataFrame.

Examples

```
>>> primes = pd.Series([2, 3, 5, 7])

Slicing might produce a Series with a single value:

>>> even_primes = primes[primes % 2 == 0]
>>> even_primes
0    2
dtype: int64

>>> even_primes.squeeze()
2

Squeezing objects with more than one value in every axis does nothing:

>>> odd_primes = primes[primes % 2 == 1]
>>> odd_primes
1    3
2    5
3    7
dtype: int64

>>> odd_primes.squeeze()
1    3
2    5
3    7
dtype: int64

Squeezing is even more effective when used with DataFrames.

>>> df = pd.DataFrame([[1, 2], [3, 4]], columns=['a', 'b'])
>>> df
   a   b
0  1  2
1  3  4

Slicing a single column will produce a DataFrame with the columns having only one value:

>>> df_a = df[['a']]
>>> df_a
   a
0  1
1  3

So the columns can be squeezed down, resulting in a Series:

>>> df_a.squeeze('columns')
0    1
1    3
Name: a, dtype: int64

Slicing a single row from a single column will produce a single scalar DataFrame:

>>> df_0a = df.loc[df.index < 1, ['a']]
>>> df_0a
   a
0  1

Squeezing the rows produces a single scalar Series:

>>> df_0a.squeeze('rows')
a    1
Name: 0, dtype: int64

Squeezing all axes will project directly into a scalar:

>>> df_0a.squeeze()
1

swapaxes(self: 'NDFrameT', axis1, axis2, copy=True) -> 'NDFrameT'  
Interchange axes and swap values axes appropriately.
```

Returns

y : same as input

```
| tail(self: 'NDFrameT', n: 'int' = 5) -> 'NDFrameT'
```

```

Return the last `n` rows.

This function returns last `n` rows from the object based on
position. It is useful for quickly verifying data, for example,
after sorting or appending rows.

For negative values of `n`, this function returns all rows except
the first `n` rows, equivalent to ``df[n:]``.

Parameters
-----
n : int, default 5
    Number of rows to select.

Returns
-----
type of caller
    The last `n` rows of the caller object.

See Also
-----
DataFrame.head : The first `n` rows of the caller object.

Examples
-----
>>> df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',
...                                'monkey', 'parrot', 'shark', 'whale', 'zebra']})
>>> df
   animal
0 alligator
1 bee
2 falcon
3 lion
4 monkey
5 parrot
6 shark
7 whale
8 zebra

Viewing the last 5 lines

>>> df.tail()
   animal
4 monkey
5 parrot
6 shark
7 whale
8 zebra

Viewing the last `n` lines (three in this case)

>>> df.tail(3)
   animal
6 shark
7 whale
8 zebra

For negative values of `n`

>>> df.tail(-3)
   animal
3 lion
4 monkey
5 parrot
6 shark
7 whale
8 zebra

take(self: 'NDFrameT', indices, axis=0, is_copy: 'bool_t | None' = None,
**kwargs) -> 'NDframeT'
    Return the elements in the given *positional* indices along an axis.

This means that we are not indexing according to actual values in
the index attribute of the object. We are indexing according to the
actual position of the element in the object.

Parameters
-----
indices : array-like
    An array of ints indicating which positions to take.
axis : {0 or 'index', 1 or 'columns', None}, default 0
    The axis on which to select elements. ``0`` means that we are
    selecting rows, ``1`` means that we are selecting columns.
is_copy : bool
    Before pandas 1.0, ``is_copy=False`` can be specified to ensure
    that the return value is an actual copy. Starting with pandas 1.0,
    ``take`` always returns a copy, and the keyword is therefore
    deprecated.
    .. deprecated:: 1.0.0
**kwargs
    For compatibility with :meth:`numpy.take`. Has no effect on the
    output.

Returns
-----
taken : same type as caller
    An array-like containing the elements taken from the object.

See Also
-----
DataFrame.loc : Select a subset of a DataFrame by labels.
DataFrame.iloc : Select a subset of a DataFrame by positions.
numpy.take : Take elements from an array along an axis.

Examples
-----
>>> df = pd.DataFrame([('falcon', 'bird', 389.0),
...                      ('parrot', 'bird', 24.0),
...                      ('lion', 'mammal', 80.5),
...                      ('monkey', 'mammal', np.nan)],
...                     columns=['name', 'class', 'max_speed'],
...                     index=[0, 2, 3, 1])
>>> df
      name  class  max_speed
0  falcon    bird     389.0
2  parrot    bird      24.0
3    lion  mammal      80.5
1  monkey  mammal      NaN

Take elements at positions 0 and 3 along the axis 0 (default).

Note how the actual indices selected (0 and 1) do not correspond to
our selected indices 0 and 3. That's because we are selecting the 0th
and 3rd rows, not rows whose indices equal 0 and 3.

>>> df.take([0, 3])
      name  class  max_speed
0  falcon    bird     389.0
1  monkey  mammal      NaN

```

```

Take elements at indices 1 and 2 along the axis 1 (column selection).

>>> df.take([1, 2], axis=1)
   class  max_speed
0  bird      389.0
2  bird       24.0
3 mammal      80.5
1 mammal      NaN

We may take elements using negative integers for positive indices,
starting from the end of the object, just like with Python lists.

>>> df.take([-1, -2])
   name  class  max_speed
1 monkey  mammal      NaN
3 lion    mammal      80.5

to_clipboard(self, excel: 'bool_t' = True, sep: 'str | None' = None, **kwargs)
-> 'None'
    Copy object to the system clipboard.

    Write a text representation of object to the system clipboard.
    This can be pasted into Excel, for example.

Parameters
-----
excel : bool, default True
    Produce output in a csv format for easy pasting into excel.

    - True, use the provided separator for csv pasting.
    - False, write a string representation of the object to the clipboard.

sep : str, default '\\t'
    Field delimiter.

**kwargs
    These parameters will be passed to DataFrame.to_csv.

See Also
-----
DataFrame.to_csv : Write a DataFrame to a comma-separated values
                  (csv) file.
read_clipboard : Read text from clipboard and pass to read_csv.

Notes
-----
Requirements for your platform.

- Linux : `xclip`, or `xsel` (with `PyQt4` modules)
- Windows : none
- macOS : none

Examples
-----
Copy the contents of a DataFrame to the clipboard.

>>> df = pd.DataFrame([[1, 2, 3], [4, 5, 6]], columns=['A', 'B', 'C'])

>>> df.to_clipboard(sep=',') # doctest: +SKIP
... # Wrote the following to the system clipboard:
... # ,A,B,C
... # 0,1,2,3
... # 1,4,5,6

We can omit the index by passing the keyword 'index' and setting
it to false.

>>> df.to_clipboard(sep=',', index=False) # doctest: +SKIP
... # Wrote the following to the system clipboard:
... # A,B,C
... # 1,2,3
... # 4,5,6

| to_csv(self, path_or_buf: 'FilePath | WriteBuffer[bytes] | WriteBuffer[str] | None' = None, sep: 'str' = ',', na_rep: 'str' = '', float_format: 'str | None' = None, columns: 'Sequence[Hashable] | None' = None, header: 'bool_t | list[str]' = True, index: 'bool_t' = True, index_label: 'IndexLabel | None' = None, mode: 'str' = 'w', encoding: 'str | None' = None, compression: 'CompressionOptions' = 'infer', quoting: 'int | None' = None, quotechar: 'str' = '"', line_terminator: 'str | None' = None, chunksize: 'int | None' = None, date_format: 'str | None' = None, doublequote: 'bool_t' = True, escapechar: 'str | None' = None, decimal: 'str' = '.', errors: 'str' = 'strict', storage_options: 'StorageOptions' = None) -> 'str | None'
|     Write object to a comma-separated values (csv) file.

Parameters
-----
path_or_buf : str, path object, file-like object, or None, default None
    String, path object (implementing os.PathLike[str]), or file-like
    object implementing a write() function. If None, the result is
    returned as a string. If a non-binary file object is passed, it should
    be opened with 'newline=''', disabling universal newlines. If a binary
    file object is passed, 'mode' might need to contain a 'b''.

.. versionchanged:: 1.2.0

    Support for binary file objects was introduced.

sep : str, default ','
    String of length 1. Field delimiter for the output file.
na_rep : str, default ''
    Missing data representation.
float_format : str, default None
    Format string for floating point numbers.
columns : sequence, optional
    Columns to write.
header : bool or list of str, default True
    Write out the column names. If a list of strings is given it is
    assumed to be aliases for the column names.
index : bool, default True
    Write row names (Index).
index_label : str or sequence, or False, default None
    Column label for index column(s) if desired. If None is given, and
    'header' and 'index' are True, then the index names are used. A
    sequence should be given if the object uses MultiIndex. If
    False do not print fields for index names. Use index_label=False
    for easier importing in R.
mode : str
    Python write mode, default 'w'.
encoding : str, optional
    A string representing the encoding to use in the output file,
    defaults to 'utf-8'. 'encoding' is not supported if 'path_or_buf'
    is a non-binary file object.
compression : str or dict, default 'infer'
    For on-the-fly compression of the output data. If 'infer' and 'xs'
    path-like, then detect compression from the following extensions:
'gz',
'.bz2', '.zip', '.xz', or '.zst' (otherwise no compression). Set to
'None' for no compression. Can also be a dict with key ``method``.
set
    to one of {'zip', 'gzip', 'bz2', 'zstd'} and other
    key-value pairs are forwarded to ``Zipfile.Zipfile``.

```

```
``gzip.GzipFile``
|   ``bz2.BZ2File``, or ``zstandard.ZstdDecompressor``, respectively. As
an
|   example, the following could be passed for faster compression and to
create
|   a reproducible gzip archive:
|   ``compression={'method': 'gzip', 'compresslevel': 1, 'mtime': 1}``.
|
|   .. versionchanged:: 1.0.0
|
|       May now be a dict with key 'method' as compression mode
|       and other entries as additional compression options if
|       compression mode is 'zip'.
|
|   .. versionchanged:: 1.1.0
|
|       Passing compression options as keys in dict is
|       supported for compression modes 'gzip', 'bz2', 'zstd', and 'zip'.
|
|   .. versionchanged:: 1.2.0
|
|       Compression is supported for binary file objects.
|
|   .. versionchanged:: 1.2.0
|
|       Previous versions forwarded dict entries for 'gzip' to
|       'gzip.open' instead of 'gzip.GzipFile' which prevented
|       setting 'mtime'.
|
|   quoting : optional constant from csv module
|       Defaults to csv.QUOTE_MINIMAL. If you have set a `float_format`
|       then floats are converted to strings and thus csv.QUOTE_NONNUMERIC
|       will treat them as non-numeric.
|   quotechar : str, default `"`
|       String of length 1. Character used to quote fields.
|   line_terminator : str, optional
|       The newline character or character sequence to use in the output
|       file. Defaults to `os.linesep`, which depends on the OS in which
|       this method is called (`'\n'` for linux, `'\r\n'` for Windows, i.e.).
|   chunksize : int or None
|       Rows to write at a time.
|   date_format : str, default None
|       Format string for datetime objects.
|   doublequote : bool, default True
|       Control quoting of `quotechar` inside a field.
|   escapechar : str, default None
|       String of length 1. Character used to escape `sep` and `quotechar`
|       when appropriate.
|   decimal : str, default `.`,
|       Character recognized as decimal separator. E.g. use `,'` for
|       European data.
|   errors : str, default 'strict'
|       Specifies how encoding and decoding errors are to be handled.
|       See the errors argument for :func:`open` for a full list
|       of options.
|
|   .. versionadded:: 1.1.0
|
|   storage_options : dict, optional
|       Extra options that make sense for a particular storage connection,
e.g.
|       host, port, username, password, etc. For HTTP(S) URLs the key-value
pairs
|       are forwarded to ``urllib`` as header options. For other URLs (e.g.
|       starting with ``s3://``, and ``gcs://``) the key-value pairs are forwarded
to
|       ``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.
|
|   .. versionadded:: 1.2.0
|
| Returns
| -----
| None or str
|     If path_or_buf is None, returns the resulting csv format as a
|     string. Otherwise returns None.
|
| See Also
| -----
| read_csv : Load a CSV file into a DataFrame.
| to_excel : Write DataFrame to an Excel file.
|
| Examples
| -----
| >>> df = pd.DataFrame({'name': ['Raphael', 'Donatello'],
| ...                 'mask': ['red', 'purple'],
| ...                 'weapon': ['sai', 'bo staff']})
| >>> df.to_csv(index=False)
| name,mask,weapon\nRaphael,red,sai\nDonatello,purple,bo staff\n
|
| Create 'out.zip' containing 'out.csv'
|
| >>> compression_opts = dict(method='zip',
| ...                             archive_name='out.csv') # doctest: +SKIP
| >>> df.to_csv('out.zip', index=False,
| ...             compression=compression_opts) # doctest: +SKIP
|
| To write a csv file to a new folder or nested folder you will first
need to create it using either Pathlib or os:
|
| >>> from pathlib import Path # doctest: +SKIP
| >>> filepath = Path('folder/subfolder/out.csv') # doctest: +SKIP
| >>> filepath.parent.mkdir(parents=True, exist_ok=True) # doctest: +SKIP
| >>> df.to_csv(filepath) # doctest: +SKIP
|
| >>> import os # doctest: +SKIP
| >>> os.makedirs('folder/subFolder', exist_ok=True) # doctest: +SKIP
| >>> df.to_csv('folder/subFolder/out.csv') # doctest: +SKIP
|
|     to_excel(self, sheet_name='str' = 'Sheet1', na_rep='str' = '',
|     float_format='str | None' = None, columns=None, header=True, index=True,
|     index_label=None, startrow=0, startcol=0, engine=None, merge_cells=True,
|     encoding=None, inf_rep='inf', verbose=True, freeze_panes=None, storage_options:
|     'StorageOptions' = None) -> 'None'
|     |
|     Write object to an Excel sheet.
|
|     To write a single object to an Excel .xlsx file it is only necessary to
|     specify a target file name. To write to multiple sheets it is necessary to
|     create an 'ExcelWriter' object with a target file name, and specify a
sheet
|     in the file to write to.
|
|     Multiple sheets may be written to by specifying unique 'sheet_name'.
|     With all data written to the file it is necessary to save the changes.
|     Note that creating an 'ExcelWriter' object with a file name that already
|     exists will result in the contents of the existing file being erased.
|
| Parameters
| -----
| excel_writer : path-like, file-like, or ExcelWriter object
|     File path or existing ExcelWriter.
|     sheet_name : str, default 'Sheet1'
```

```

    Name of sheet which will contain DataFrame.
na_rep : str, default ''
    Missing data representation.
float_format : str, optional
    Format string for floating point numbers. For example
    ``float_format=".2f"`` will format 0.1234 to 0.12.
columns : sequence or list of str, optional
    Columns to write.
header : bool or list of str, default True
    Write out the column names. If a list of string is given it is
    assumed to be aliases for the column names.
index : bool, default True
    Write row names (index).
index_label : str or sequence, optional
    Column label for index column(s) if desired. If not specified, and
    'header' and 'index' are True, then the index names are used. A
    sequence should be given if the DataFrame uses MultiIndex.
startrow : int, default 0
    Upper left cell row to dump data frame.
startcol : int, default 0
    Upper left cell column to dump data frame.
engine : str, optional
    Write engine to use, 'openpyxl' or 'xlsxwriter'. You can also set this
    via the options ``io.excel.xlsx.writer``, ``io.excel.xls.writer``, and
    ``io.excel.xlsm.writer``.

.. deprecated:: 1.2.0

longer
    As the `xlwt <https://pypi.org/project/xlwt/>`__ package is no
| maintained, the ``xlwt`` engine will be removed in a future
version
    of pandas.

merge_cells : bool, default True
    Write MultiIndex and Hierarchical Rows as merged cells.
encoding : str, optional
    Encoding of the resulting excel file. Only necessary for xlwt,
    other writers support unicode natively.
inf_rep : str, default 'inf'
    Representation for infinity (there is no native representation for
    infinity in Excel).
verbose : bool, default True
    Display more information in the error logs.
freeze_panes : tuple of int (length 2), optional
    Specifies the one-based bottommost row and rightmost column that
    is to be frozen.
storage_options : dict, optional
    Extra options that make sense for a particular storage connection,
e.g.
    host, port, username, password, etc. For HTTP(S) URLs the key-value
pairs
    are forwarded to ``urllib`` as header options. For other URLs (e.g.
to
    starting with "s3://", and "gcs://") the key-value pairs are forwarded
    to ``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.
    .. versionadded:: 1.2.0

See Also
-----
to_csv : Write DataFrame to a comma-separated values (csv) file.
ExcelWriter : Class for writing DataFrame objects into excel sheets.
read_excel : Read an Excel file into a pandas DataFrame.
read_csv : Read a comma-separated values (csv) file into DataFrame.

Notes
-----
For compatibility with :meth:`DataFrame.to_csv`,  

to_excel serializes lists and dicts to strings before writing.

Once a workbook has been saved it is not possible to write further  

data without rewriting the whole workbook.

Examples
-----
Create, write to and save a workbook:

>>> df1 = pd.DataFrame([['a', 'b'], ['c', 'd']],
...                      index=['row 1', 'row 2'],
...                      columns=['col 1', 'col 2'])
>>> df1.to_excel("output.xlsx") # doctest: +SKIP

To specify the sheet name:

>>> df1.to_excel("output.xlsx",
...                 sheet_name='Sheet_name_1') # doctest: +SKIP

If you wish to write to more than one sheet in the workbook, it is  

necessary to specify an ExcelWriter object:

>>> df2 = df1.copy()
>>> with pd.ExcelWriter('output.xlsx') as writer: # doctest: +SKIP
...     df1.to_excel(writer, sheet_name='Sheet_name_1')
...     df2.to_excel(writer, sheet_name='Sheet_name_2')

ExcelWriter can also be used to append to an existing Excel file:

>>> with pd.ExcelWriter('output.xlsx',
...                         mode='a') as writer: # doctest: +SKIP
...     df.to_excel(writer, sheet_name='Sheet_name_3')

To set the library that is used to write the Excel file,  

you can pass the 'engine' keyword (the default engine is  

automatically chosen depending on the file extension):

>>> df1.to_excel('output1.xlsx', engine='xlsxwriter') # doctest: +SKIP

| to_hdf(self, path_or_buf, key: 'str', mode: 'str' = 'a', complevel: 'int' |
| None' = None, complib: 'str' | None' = None, append: 'bool_t' = False, format: 'str' | |
| None' = None, index: 'bool_t' = True, min_itemsize: 'int' | dict[str, int] | |
| None' = None, nan_rep=None, dropna: 'bool_t' | None' = None, data_columns: |
| 'Literal[True] | list[str] | None' = None, errors: 'str' = 'strict', encoding: |
| 'str' = 'UTF-8') -> 'None'
| Write the contained data to an HDF5 file using HDFStore.

Hierarchical Data Format (HDF) is self-describing, allowing an  

application to interpret the structure and contents of a file with no  

outside information. One HDF file can hold a mix of related objects  

which can be accessed as a group or as individual objects.

In order to add another DataFrame or Series to an existing HDF file  

please use append mode and a different a key.

.. warning::

    One can store a subclass of ``DataFrame`` or ``Series`` to HDF5,  

    but the type of the subclass is lost upon storing.

For more information see the :ref:`user guide <io.hdf5>`.
```

```

Parameters
-----
path_or_buf : str or pandas.HDFStore
    File path or HDFStore object.
key : str
    Identifier for the group in the store.
mode : {'a', 'w', 'r+'}, default 'a'
    Mode to open file:
        - 'w': write, a new file is created (an existing file with
            the same name would be deleted).
        - 'a': append, an existing file is opened for reading and
            writing, and if the file does not exist it is created.
        - 'r+'. similar to 'a', but the file must already exist.
        complevel : {0-9}, default None
            Specifies a compression level for data.
            A value of 0 or None disables compression.
complib : {'zlib', 'lzo', 'bzip2', 'blosc'}, default 'zlib'
            Specifies the compression library to be used.
            As of v0.20.2 these additional compressors for Blosc are supported
            (default if no compressor specified: 'blosc:blosc1z'):
            {'blosc:blosclz', 'blosc:lz4', 'blosc:lz4hc', 'blosc:snappy',
            'blosc:zlib', 'blosc:zstd'}.
            Specifying a compression library which is not available issues
            a ValueError.
append : bool, default False
    For Table formats, append the input data to the existing.
format : {'fixed', 'table', None}, default 'fixed'
    Possible values:
        - 'fixed': Fixed format. Fast writing/reading. Not-appendable,
            nor searchable.
        - 'table': Table format. Write as a PyTables Table structure
            which may perform worse but allow more flexible operations
            like searching / selecting subsets of the data.
        - If None, pd.get_option('io.hdf.default_format') is checked,
            followed by fallback to "fixed".
errors : str, default 'strict'
    Specifies how encoding and decoding errors are to be handled.
    See the errors argument for :func:`open` for a full list
    of options.
encoding : str, default "UTF-8"
min_itemsize : dict or int, optional
    Map column names to minimum string sizes for columns.
nan_rep : Any, optional
    How to represent null values as str.
    Not allowed with append=True.
data_columns : list of columns or True, optional
    List of columns to create as indexed data columns for on-disk
    queries, or True to use all columns. By default only the axes
    of the object are indexed. See :ref:`io.hdf5-query-data-columns`.
    Applicable only to format='table'.

See Also
-----
read_hdf : Read from HDF file.
DataFrame.to_parquet : Write a DataFrame to the binary parquet format.
DataFrame.to_sql : Write to a SQL table.
DataFrame.to_feather : Write out feather-format for DataFrames.
DataFrame.to_csv : Write out to a csv file.

Examples
-----
>>> df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]},
...     index=['a', 'b', 'c']) # doctest: +SKIP
>>> df.to_hdf('data.h5', key='df', mode='w') # doctest: +SKIP

We can add another object to the same file:

>>> s = pd.Series([1, 2, 3, 4]) # doctest: +SKIP
>>> s.to_hdf('data.h5', key='s') # doctest: +SKIP

Reading from HDF file:

>>> pd.read_hdf('data.h5', 'df') # doctest: +SKIP
A   B
a   1
b   2
c   3
>>> pd.read_hdf('data.h5', 's') # doctest: +SKIP
0   1
1   2
2   3
3   4
dtype: int64

    to_json(self, path_or_buf: 'FilePath | WriteBuffer[bytes] | WriteBuffer[str] | None' = None, orient: 'str | None' = None, date_format: 'str | None' = None, double_precision: 'int' = 10, force_ascii: 'bool_t' = True, date_unit: 'str' = 'ms', default_handler: 'Callable[[Any], JSONSerializable] | None' = None, lines: 'bool_t' = False, compression: 'CompressionOptions' = 'infer', index: 'bool_t' = True, indent: 'int | None' = None, storage_options: 'StorageOptions' = None) -> 'str | None'
        Convert the object to a JSON string.

    Note NaN's and None will be converted to null and datetime objects
    will be converted to UNIX timestamps.

Parameters
-----
path_or_buf : str, path object, file-like object, or None, default None
    String, path object (implementing os.PathLike[str]), or file-like
    object implementing a write() function. If None, the result is
    returned as a string.
orient : str
    Indication of expected JSON string format.

    * Series:
        - default is 'index'
        - allowed values are: {'split', 'records', 'index', 'table'}.

    * DataFrame:
        - default is 'columns'
        - allowed values are: {'split', 'records', 'index', 'columns',
            'values', 'table'}.

    * The format of the JSON string:
        - 'split' : dict like {'index' -> [index], 'columns' -> [columns],
            'data' -> [values]}
        - 'records' : list like [[column -> value], ... , {column ->
            value}]
        - 'index' : dict like {index -> {column -> value}}
        - 'columns' : dict like {column -> {index -> value}}
        - 'values' : just the values array
        - 'table' : dict like {'schema': {schema}, 'data': {data}},

    Describing the data, where data component is like

```

```

``orient='records'``.

    date_format : {None, 'epoch', 'iso'}
        Type of date conversion. 'epoch' = epoch milliseconds,
        'iso' = ISO8601. The default depends on the 'orient'. For
        'orient='table'```, the default is 'iso'. For all other orients,
        the default is 'epoch'.
    double_precision : int, default 10
        The number of decimal places to use when encoding
        floating point values.
    force_ascii : bool, default True
        Force encoded string to be ASCII.
    date_unit : str, default 'ms' (milliseconds)
        The time unit to encode to, governs timestamp and ISO8601
        precision. One of 's', 'ms', 'us', 'ns' for second, millisecond,
        microsecond, and nanosecond respectively.
    default_handler : callable, default None
        Handler to call if object cannot otherwise be converted to a
        suitable format for JSON. Should receive a single argument which is
        the object to convert and return a serialisable object.
    lines : bool, default False
        If 'orient' is 'records' write out line-delimited json format. Will
        throw ValueError if incorrect 'orient' since others are not
        list-like.
    compression : str or dict, default 'infer'
        For on-the-fly compression of the output data. If 'infer' and
    'path_or_buf'
    | path-like, then detect compression from the following extensions:
    '| 'gz',
    '|   '.bz2', '.zip', '.xz', or '.zst' (otherwise no compression). Set to
    '|   ``None`` for no compression. Can also be a dict with key ``'method'``
    set
    |   to one of {``'zip'``, ``'gzip'``, ``'bz2'``, ``'zstd'``} and other
    |   key-value pairs are forwarded to `` zipfile.ZipFile``,
    ``gzip.GzipFile``,
    |   `` bz2.BZ2File`` , or `` zstandard.ZstdDecompressor`` , respectively. As
    an
    |   example, the following could be passed for faster compression and to
    create
    |   a reproducible gzip archive:
    |   ``compression={'method': 'gzip', 'compresslevel': 1, 'mtime': 1}``.
    |
    .. versionchanged:: 1.4.0 Zstandard support.

    index : bool, default True
        Whether to include the index values in the JSON string. Not
        including the index ('`index=False`') is only supported when
        orient is 'split' or 'table'.
    indent : int, optional
        Length of whitespace used to indent each record.
    |
    .. versionadded:: 1.0.0

    storage_options : dict, optional
        Extra options that make sense for a particular storage connection,
e.g.
    | host, port, username, password, etc. For HTTP(S) URLs the key-value
pairs
    | pairs are forwarded to ``urllib`` as header options. For other URLs (e.g.
to
    | starting with "s3://", and "gcs://"") the key-value pairs are forwarded
    | ``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.
    |
    .. versionadded:: 1.2.0

Returns
-----
None or str
    If path_or_buf is None, returns the resulting json format as a
    string. Otherwise returns None.

See Also
-----
read_json : Convert a JSON string to pandas object.

Notes
-----
The behavior of ``indent=0`` varies from the stdlib, which does not
indent the output but does insert newlines. Currently, ``indent=0`` and the default ````indent=None```` are equivalent in pandas, though this
may change in a future release.

``orient='table'`` contains a 'pandas_version' field under 'schema'.
This stores the version of 'pandas' used in the latest revision of the
schema.

Examples
-----
>>> import json
>>> df = pd.DataFrame(
...     [[{"a": "b"}, {"c": "d"}],
...     index=["row 1", "row 2"],
...     columns=["col 1", "col 2"],
... )
>>> result = df.to_json(orient="split")
>>> parsed = json.loads(result)
>>> json.dumps(parsed, indent=4) # doctest: +SKIP
{
    "columns": [
        "col 1",
        "col 2"
    ],
    "index": [
        "row 1",
        "row 2"
    ],
    "data": [
        [
            "a",
            "b"
        ],
        [
            "c",
            "d"
        ]
    ]
}

Encoding/decoding a Dataframe using ``records`` formatted JSON.
Note that index labels are not preserved with this encoding.

>>> result = df.to_json(orient="records")
>>> parsed = json.loads(result)
>>> json.dumps(parsed, indent=4) # doctest: +SKIP
[
    {
        "col 1": "a",
        "col 2": "b"
    },
    {

```

```

        "col 1": "c",
        "col 2": "d"
    }
]

Encoding/decoding a Dataframe using ```index``` formatted JSON:

>>> result = df.to_json(orient="index")
>>> parsed = json.loads(result)
>>> json.dumps(parsed, indent=4) # doctest: +SKIP
{
    "row 1": {
        "col 1": "a",
        "col 2": "b"
    },
    "row 2": {
        "col 1": "c",
        "col 2": "d"
    }
}

Encoding/decoding a Dataframe using ```columns``` formatted JSON:

>>> result = df.to_json(orient="columns")
>>> parsed = json.loads(result)
>>> json.dumps(parsed, indent=4) # doctest: +SKIP
{
    "col 1": {
        "row 1": "a",
        "row 2": "c"
    },
    "col 2": {
        "row 1": "b",
        "row 2": "d"
    }
}

Encoding/decoding a Dataframe using ```values``` formatted JSON:

>>> result = df.to_json(orient="values")
>>> parsed = json.loads(result)
>>> json.dumps(parsed, indent=4) # doctest: +SKIP
[
    [
        "a",
        "b"
    ],
    [
        "c",
        "d"
    ]
]

Encoding with Table Schema:

>>> result = df.to_json(orient="table")
>>> parsed = json.loads(result)
>>> json.dumps(parsed, indent=4) # doctest: +SKIP
{
    "schema": {
        "fields": [
            {
                "name": "index",
                "type": "string"
            },
            {
                "name": "col 1",
                "type": "string"
            },
            {
                "name": "col 2",
                "type": "string"
            }
        ],
        "primaryKey": [
            "index"
        ],
        "pandas_version": "1.4.0"
    },
    "data": [
        {
            "index": "row 1",
            "col 1": "a",
            "col 2": "b"
        },
        {
            "index": "row 2",
            "col 1": "c",
            "col 2": "d"
        }
    ]
}

| to_latex(self, buf=None, columns=None, col_space=None, header=True,
| index=True, na_rep='NaN', formatters=None, float_format=None, sparsify=None,
| index_names=True, bold_rows=False, column_format=None, longtable=None,
| escape=None, encoding=None, decimal=',', multicolumn=None,
| multicolumn_format=None, multirow=None, caption=None, label=None, position=None)
| Render object to a LaTeX tabular, longtable, or nested table.
|
| Requires ``\usepackage{booktabs}``. The output can be copy/pasted
| into a main LaTeX document or read from an external file
| with ``\input{table.tex}``.

.. versionchanged:: 1.0.0
   Added caption and label arguments.

.. versionchanged:: 1.2.0
   Added position argument, changed meaning of caption argument.

Parameters
-----
buf : str, Path or StringIO-like, optional, default None
    Buffer to write to. If None, the output is returned as a string.
columns : list of label, optional
    The subset of columns to write. Writes all columns by default.
col_space : int, optional
    The minimum width of each column.
header : bool or list of str, default True
    Write out the column names. If a list of strings is given,
    it is assumed to be aliases for the column names.
index : bool, default True
    Write row names (index).
na_rep : str, default 'NaN'
    Missing data representation.
formatters : list of functions or dict of {str: function}, optional
    Formatter functions to apply to columns' elements by position or
    name. The result of each function must be a unicode string.
    List must be of length equal to the number of columns.
float_format : one-parameter function or str, optional, default None
    Formatter for floating point numbers. For example

```

```

    ``float_format`` and ``float_format`` will
    both result in 0.1234 being formatted as 0.12.
sparsify : bool, optional
    Set to False for a DataFrame with a hierarchical index to print
    every multiindex key at each row. By default, the value will be
    read from the config module.
index_names : bool, default True
    Prints the names of the indexes.
bold_rows : bool, default False
    Make the row labels bold in the output.
column_format : str, optional
    The columns format as specified in 'LaTeX table format
    <https://en.wikibooks.org/wiki/LaTeX/Tables>' e.g. 'rcl' for 3
    columns. By default, 'l' will be used for all columns except
    columns of numbers, which default to 'r'.
longtable : bool, optional
    By default, the value will be read from the pandas config
    module. Use a longtable environment instead of tabular. Requires
    adding a \usepackage{longtable} to your LaTeX preamble.
escape : bool, optional
    By default, the value will be read from the pandas config
    module. When set to False prevents from escaping latex special
    characters in column names.
encoding : str, optional
    A string representing the encoding to use in the output file,
    defaults to 'utf-8'.
decimal : str, default ','
    Character recognized as decimal separator, e.g. ',' in Europe.
multicolumn : bool, default True
    Use \multicolumn to enhance MultiIndex columns.
    The default will be read from the config module.
multicolumn_format : str, default 'l'
    The alignment for multicolumns, similar to 'column_format'.
    The default will be read from the config module.
multirow : bool, default False
    Use \multirow to enhance MultiIndex rows. Requires adding a
    \usepackage{multirow} to your LaTeX preamble. Will print
    centered labels (instead of top-aligned) across the contained
    rows, separating groups via clines. The default will be read
    from the pandas config module.
caption : str or tuple, optional
    Tuple (full_caption, short_caption),
    which results in ``\caption[short_caption]{full_caption}``;
    if a single string is passed, no short caption will be set.

    .. versionadded:: 1.0.0

    .. versionchanged:: 1.2.0
        Optionally allow caption to be a tuple ``(full_caption,
        short_caption)``.

label : str, optional
    The LaTeX label to be placed inside ``\label{}`` in the output.
    This is used with ``\ref{}`` in the main ``.tex`` file.

    .. versionadded:: 1.0.0

position : str, optional
    The LaTeX positional argument for tables, to be placed after
    ``\begin{}`` in the output.

    .. versionadded:: 1.2.0

    Returns
    -----
    str or None
        If buf is None, returns the result as a string. Otherwise
returns
None.

See Also
-----
Styler.to_latex : Render a DataFrame to LaTeX with conditional formatting.
DataFrame.to_string : Render a DataFrame to a console-friendly
tabular output.
DataFrame.to_html : Render a DataFrame as an HTML table.

Examples
-----
>>> df = pd.DataFrame(dict(name=['Raphael', 'Donatello'],
...                         mask=['red', 'purple'],
...                         weapon=['sai', 'bo staff']))
>>> print(df.to_latex(index=False)) # doctest: +SKIP
\begin{tabular}{lll}
\toprule
name & mask & weapon \\
\midrule
Raphael & red & sai \\
Donatello & purple & bo staff \\
\bottomrule
\end{tabular}

    to_pickle(self, path, compression='CompressionOptions' = 'infer', protocol:
'int' = 5, storage_options: 'StorageOptions' = None) -> 'None'
Pickle (serialize) object to file.

Parameters
-----
path : str
    File path where the pickled object will be stored.
compression : str or dict, default 'infer'
    For on-the-fly compression of the output data. If 'infer' and 'path'
    path-like, then detect compression from the following extensions:
'.gz',
    '.bz2', '.zip', '.xz', or '.zst' (otherwise no compression). Set to
    ''None'' for no compression. Can also be a dict with key ''method''.
set
    to one of {``'zip'``, ``'gzip'``, ``'bz2'``, ``'zstd'``} and other
    key-value pairs are forwarded to ``zipfile.Zipfile``,
``gzip.GzipFile``,
    ``bz2.BZ2File``, or ``zstandard.ZstdDecompressor``, respectively. As
an
    example, the following could be passed for faster compression and to
create
    a reproducible gzip archive:
    ``compression={'method': 'gzip', 'compresslevel': 1, 'mtime': 1}``.
protocol : int
    Int which indicates which protocol should be used by the pickler,
    default HIGHEST_PROTOCOL (see [1]_ paragraph 12.1.2). The possible
    values are 0, 1, 2, 3, 4, 5. A negative value for the protocol
    parameter is equivalent to setting its value to HIGHEST_PROTOCOL.

    .. [1] https://docs.python.org/3/library/pickle.html.

storage_options : dict, optional
    Extra options that make sense for a particular storage connection,
e.g.
    host, port, username, password, etc. For HTTP(S) URLs the key-value
pairs
    are forwarded to ``urllib`` as header options. For other URLs (e.g.
    starting with "s3://", and "gcs://") the key-value pairs are forwarded
to

```

```

``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.
.. versionadded:: 1.2.0

See Also
-----
read_pickle : Load pickled pandas object (or any object) from file.
DataFrame.to_hdf : Write DataFrame to an HDF5 file.
DataFrame.to_sql : Write DataFrame to a SQL database.
DataFrame.to_parquet : Write a DataFrame to the binary parquet format.

Examples
-----
>>> original_df = pd.DataFrame({"foo": range(5), "bar": range(5, 10)}) # doctest: +SKIP
| >>> original_df # doctest: +SKIP
|    foo   bar
|    0     5
|    1     6
|    2     7
|    3     8
|    4     9
|>>> original_df.to_pickle("./dummy.pkl") # doctest: +SKIP

| unpickled_df = pd.read_pickle("./dummy.pkl") # doctest: +SKIP
| >>> unpickled_df # doctest: +SKIP
|    foo   bar
|    0     5
|    1     6
|    2     7
|    3     8
|    4     9

| to_sql(self, name: 'str', con, schema=None, if_exists: 'str' = 'fail', index:
| 'bool_t' = True, index_label=None, chunksize=None, dtype: 'DtypeArg | None' =
| None, method=None) -> 'int | None'
|     Write records stored in a DataFrame to a SQL database.

| Databases supported by SQLAlchemy [1]_ are supported. Tables can be
| newly created, appended to, or overwritten.

Parameters
-----
name : str
    Name of SQL table.
con : sqlalchemy.engine.(Engine or Connection) or sqlite3.Connection
    Using SQLAlchemy makes it possible to use any DB supported by that
    library. Legacy support is provided for sqlite3.Connection objects.
The user
|     is responsible for engine disposal and connection closure for the
SQLAlchemy
|     connectable See `here
<https://docs.sqlalchemy.org/en/13/core/connections.html>`_.

schema : str, optional
    Specify the schema (if database flavor supports this). If None, use
    default schema.
if_exists : {'fail', 'replace', 'append'}, default 'fail'
    How to behave if the table already exists.

    * fail: Raise a ValueError.
    * replace: Drop the table before inserting new values.
    * append: Insert new values to the existing table.

index : bool, default True
    Write DataFrame index as a column. Uses `index_label` as the column
    name in the table.
index_label : str or sequence, default None
    Column label for index column(s). If None is given (default) and
    'index' is True, then the index names are used.
    A sequence should be given if the DataFrame uses MultiIndex.
chunksize : int, optional
    Specify the number of rows in each batch to be written at a time.
    By default, all rows will be written at once.
dtype : dict or scalar, optional
    Specifying the datatype for columns. If a dictionary is used, the
    keys should be the column names and the values should be the
    SQLAlchemy types or strings for the sqlite3 legacy mode. If a
    scalar is provided, it will be applied to all columns.
method : {None, 'multi', callable}, optional
    Controls the SQL insertion clause used:

    * None : Uses standard SQL ``INSERT`` clause (one per row).
    * 'multi': Pass multiple values in a single ``INSERT`` clause.
    * callable with signature ``((pd_table, conn, keys, data_iter))``.

    Details and a sample callable implementation can be found in the
    section :ref:`insert method <io.sql.method>`.

Returns
-----
None or int
    Number of rows affected by to_sql. None is returned if the callable
    passed into ``method`` does not return the number of rows.

    The number of returned rows affected is the sum of the ``rowcount``
    attribute of ``sqlite3.Cursor`` or SQLAlchemy connectable which may
not
    reflect the exact number of written rows as stipulated in the
    `sqlite3
<https://docs.python.org/3/library/sqlite3.html#sqlite3.Cursor.rowcount>`__ or
| SQLAlchemy
<https://docs.sqlalchemy.org/en/14/core/connections.html#sqlalchemy.engine.BaseCur
| sorResult.rowcount>`__.

    .. versionadded:: 1.4.0

Raises
-----
ValueError
    When the table already exists and `if_exists` is 'fail' (the
    default).

See Also
-----
read_sql : Read a DataFrame from a table.

Notes
-----
Timezone aware datetime columns will be written as
``Timestamp with timezone`` type with SQLAlchemy if supported by the
database. Otherwise, the datetimes will be stored as timezone unaware
timestamps local to the original timezone.

References
-----
.. [1] https://docs.sqlalchemy.org
.. [2] https://www.python.org/dev/peps/pep-0249/

Examples
-----

```

```

Create an in-memory SQLite database.

>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://', echo=False)

Create a table from scratch with 3 rows.

>>> df = pd.DataFrame({'name' : ['User 1', 'User 2', 'User 3']})
>>> df
   name
0 User 1
1 User 2
2 User 3

>>> df.to_sql('users', con=engine)
3
>>> engine.execute("SELECT * FROM users").fetchall()
[(0, 'User 1'), (1, 'User 2'), (2, 'User 3')]

An `sqlalchemy.engine.Connection` can also be passed to `con`:

>>> with engine.begin() as connection:
...     df1 = pd.DataFrame({'name' : ['User 4', 'User 5']})
...     df1.to_sql('users', con=connection, if_exists='append')
2

This is allowed to support operations that require that the same
DBAPI connection is used for the entire operation.

>>> df2 = pd.DataFrame({'name' : ['User 6', 'User 7']})
>>> df2.to_sql('users', con=engine, if_exists='append')
2
>>> engine.execute("SELECT * FROM users").fetchall()
[(0, 'User 1'), (1, 'User 2'), (2, 'User 3'),
(0, 'User 4'), (1, 'User 5'), (0, 'User 6'),
(1, 'User 7')]

Overwrite the table with just ``df2``.

>>> df2.to_sql('users', con=engine, if_exists='replace',
...             index_label='id')
2
>>> engine.execute("SELECT * FROM users").fetchall()
[(0, 'User 6'), (1, 'User 7')]

Specify the dtype (especially useful for integers with missing values).
Notice that while pandas is forced to store the data as floating point,
the database supports nullable integers. When fetching the data with
Python, we get back integer scalars.

>>> df = pd.DataFrame({"A": [1, None, 2]})
>>> df
   A
0 1.0
1 NaN
2 2.0

>>> from sqlalchemy.types import Integer
>>> df.to_sql('integers', con=engine, index=False,
...             dtype={"A": Integer()})
3

>>> engine.execute("SELECT * FROM integers").fetchall()
[(1,), (None,), (2,)]
```

to_xarray(self)
Return an xarray object from the pandas object.

Returns

xarray.DataArray or xarray.Dataset
 Data in the pandas structure converted to Dataset if the object is
 a DataFrame, or a DataArray if the object is a Series.

See Also

`DataFrame.to_hdf` : Write DataFrame to an HDFS file.
`DataFrame.to_parquet` : Write a DataFrame to the binary parquet format.

Notes

See the `xarray docs <<https://xarray.pydata.org/en/stable/>>`__

Examples

```

>>> df = pd.DataFrame([('falcon', 'bird', 389.0, 2),
...                     ('parrot', 'bird', 24.0, 2),
...                     ('lion', 'mammal', 80.5, 4),
...                     ('monkey', 'mammal', np.nan, 4)],
...                     columns=['name', 'class', 'max_speed',
...                               'num_legs'])
>>> df
   name    class  max_speed  num_legs
0 falcon    bird      389.0        2
1 parrot    bird       24.0        2
2 lion     mammal      80.5        4
3 monkey   mammal        NaN        4

>>> df.to_xarray()
<xarray.Dataset>
Dimensions:  (index: 4)
Coordinates:
 * index      (index) int64 0 1 2 3
Data variables:
    name      (index) object 'falcon' 'parrot' 'lion' 'monkey'
    class     (index) object 'bird' 'bird' 'mammal' 'mammal'
    max_speed (index) float64 389.0 24.0 80.5 nan
    num_legs  (index) int64 2 2 4 4

>>> df['max_speed'].to_xarray()
<xarray.DataArray 'max_speed' (index: 4)>
array([389.,  24.,  80.5,  nan])
Coordinates:
 * index      (index) int64 0 1 2 3
Data variables:
    max_speed (index) float64 389.0 24.0 80.5 nan
```

Coordinates:

- * index (index) int64 0 1 2 3

```

>>> dates = pd.to_datetime(['2018-01-01', '2018-01-01',
...                         '2018-01-02', '2018-01-02'])
>>> df_multiindex = pd.DataFrame({'date': dates,
...                                 'animal': ['falcon', 'parrot',
...                                            'falcon', 'parrot'],
...                                 'speed': [350, 18, 361, 15]})
>>> df_multiindex = df_multiindex.set_index(['date', 'animal'])

>>> df_multiindex
              speed
date    animal
2018-01-01 falcon    350
          parrot     18
2018-01-02 falcon    361
          parrot     15
```

```

>>> df_multiindex.to_xarray()
<xarray.Dataset>
Dimensions: (animal: 2, date: 2)
Coordinates:
  * date      (date) datetime64[ns] 2018-01-01 2018-01-02
  * animal    (animal) object 'falcon' 'parrot'
Data variables:
  speed     (date, animal) int64 350 18 361 15

| truncate(self: 'NDFrameT', before=None, after=None, axis=None, copy: 'bool_t'
= True) -> 'NDFrameT'
  Truncate a Series or DataFrame before and after some index value.

  This is a useful shorthand for boolean indexing based on index
  values above or below certain thresholds.

Parameters
-----
before : date, str, int
  Truncate all rows before this index value.
after : date, str, int
  Truncate all rows after this index value.
axis : {0 or 'index', 1 or 'columns'}, optional
  Axis to truncate. Truncates the index (rows) by default.
copy : bool, default is True,
  Return a copy of the truncated section.

Returns
-----
type of caller
  The truncated Series or DataFrame.

See Also
-----
DataFrame.loc : Select a subset of a DataFrame by label.
DataFrame.iloc : Select a subset of a DataFrame by position.

Notes
-----
If the index being truncated contains only datetime values,
`'before'` and `'after'` may be specified as strings instead of
Timestamps.

Examples
-----
>>> df = pd.DataFrame({'A': ['a', 'b', 'c', 'd', 'e'],
...                      'B': ['f', 'g', 'h', 'i', 'j'],
...                      'C': ['k', 'l', 'm', 'n', 'o']},
...                      index=[1, 2, 3, 4, 5])
>>> df
   A  B  C
1  a  f  k
2  b  g  l
3  c  h  m
4  d  i  n
5  e  j  o

>>> df.truncate(before=2, after=4)
   A  B  C
2  b  g  l
3  c  h  m
4  d  i  n

The columns of a DataFrame can be truncated.

>>> df.truncate(before="A", after="B", axis="columns")
   A  B
1  a  f
2  b  g
3  c  h
4  d  i
5  e  j

For Series, only rows can be truncated.

>>> df['A'].truncate(before=2, after=4)
2
3
4
Name: A, dtype: object

The index values in ``truncate`` can be datetimes or string
dates.

>>> dates = pd.date_range('2016-01-01', '2016-02-01', freq='s')
>>> df = pd.DataFrame(index=dates, data={'A': 1})
>>> df.tail()
          A
2016-01-31 23:59:56 1
2016-01-31 23:59:57 1
2016-01-31 23:59:58 1
2016-01-31 23:59:59 1
2016-02-01 00:00:00 1

>>> df.truncate(before=pd.Timestamp('2016-01-05'),
...                 after=pd.Timestamp('2016-01-10')).tail()
          A
2016-01-09 23:59:56 1
2016-01-09 23:59:57 1
2016-01-09 23:59:58 1
2016-01-09 23:59:59 1
2016-01-10 00:00:00 1

Because the index is a DatetimeIndex containing only dates, we can
specify `'before'` and `'after'` as strings. They will be coerced to
Timestamps before truncation.

>>> df.truncate('2016-01-05', '2016-01-10').tail()
          A
2016-01-09 23:59:56 1
2016-01-09 23:59:57 1
2016-01-09 23:59:58 1
2016-01-09 23:59:59 1
2016-01-10 00:00:00 1

Note that ``truncate`` assumes a 0 value for any unspecified time
component (midnight). This differs from partial string slicing, which
returns any partially matching dates.

>>> df.loc['2016-01-05':'2016-01-10', :].tail()
          A
2016-01-10 23:59:55 1
2016-01-10 23:59:56 1
2016-01-10 23:59:57 1
2016-01-10 23:59:58 1
2016-01-10 23:59:59 1

tshift(self: 'NDFrameT', periods: 'int' = 1, freq=None, axis: 'Axis' = 0) ->
'NDFrameT'
  Shift the time index, using the index's frequency if available.

```

```

... deprecated:: 1.1.0
    Use `shift` instead.

Parameters
-----
periods : int
    Number of periods to move, can be positive or negative.
freq : DateOffset, timedelta, or str, default None
    Increment to use from the tseries module
    or time rule expressed as a string (e.g. 'EOM').
axis : {0 or 'index', 1 or 'columns', None}, default 0
    Corresponds to the axis that contains the Index.

Returns
-----
shifted : Series/DataFrame

Notes
-----
If freq is not specified then tries to use the freq or inferred_freq
attributes of the index. If neither of those attributes exist, a
ValueError is thrown

tz_convert(self: 'NDFrameT', tz, axis=0, level=None, copy: 'bool_t' = True) ->
'NDFrameT'
    Convert tz-aware axis to target time zone.

Parameters
-----
tz : str or tzinfo object
axis : the axis to convert
level : int, str, default None
    If axis is a MultiIndex, convert a specific level. Otherwise
    must be None.
copy : bool, default True
    Also make a copy of the underlying data.

Returns
-----
{klass}
    Object with time zone converted axis.

Raises
-----
TypeError
    If the axis is tz-naive.

tz_localize(self: 'NDFrameT', tz, axis=0, level=None, copy: 'bool_t' = True,
ambiguous='raise', nonexistent='str' = 'raise') -> 'NDFrameT'
    Localize tz-naive index of a Series or DataFrame to target time zone.

This operation localizes the Index. To localize the values in a
timezone-naive Series, use :meth:`Series.dt.tz_localize`.

Parameters
-----
tz : str or tzinfo
axis : the axis to localize
level : int, str, default None
    If axis is a MultiIndex, localize a specific level. Otherwise
    must be None.
copy : bool, default True
    Also make a copy of the underlying data.
ambiguous : 'infer', bool ndarray, 'NaT', default 'raise'
    When clocks moved backward due to DST, ambiguous times may arise.
    For example in Central European Time (UTC+01), when going from
    03:00 DST to 02:00 non-DST, 02:30:00 local time occurs both at
    00:30:00 UTC and at 01:30:00 UTC. In such a situation, the
    'ambiguous' parameter dictates how ambiguous times should be
    handled.
        - 'infer' will attempt to infer fall dst-transition hours based on
        order
        - bool ndarray where True signifies a DST time, False designates
        a non-DST time (note that this flag is only applicable for
        ambiguous times)
        - 'NaT' will return NaT where there are ambiguous times
        - 'raise' will raise an AmbiguousTimeError if there are ambiguous
        times.
nonexistent : str, default 'raise'
    A nonexistent time does not exist in a particular timezone
    where clocks moved forward due to DST. Valid values are:
        - 'shift_forward' will shift the nonexistent time forward to the
        closest existing time
        - 'shift_backward' will shift the nonexistent time backward to the
        closest existing time
        - 'NaT' will return NaT where there are nonexistent times
        - timedelta objects will shift nonexistent times by the timedelta
        - 'raise' will raise an NonExistentTimeError if there are
        nonexistent times.

Returns
-----
Series or DataFrame
    Same type as the input.

Raises
-----
TypeError
    If the TimeSeries is tz-aware and tz is not None.

Examples
-----
Localize local times:

>>> s = pd.Series([1,
...                 index=pd.DatetimeIndex(['2018-09-15 01:30:00'])))
>>> s.tz_localize('CET')
2018-09-15 01:30:00+02:00    1
dtype: int64

Be careful with DST changes. When there is sequential data, pandas
can infer the DST time:

>>> s = pd.Series(range(7),
...                 index=pd.DatetimeIndex(['2018-10-28 01:30:00',
...                                         '2018-10-28 02:00:00',
...                                         '2018-10-28 02:30:00',
...                                         '2018-10-28 02:00:00',
...                                         '2018-10-28 02:30:00',
...                                         '2018-10-28 03:00:00',
...                                         '2018-10-28 03:30:00']))
>>> s.tz_localize('CET', ambiguous='infer')
2018-10-28 01:30:00+02:00    0
2018-10-28 02:00:00+02:00    1
2018-10-28 02:30:00+02:00    2
2018-10-28 02:00:00+01:00    3
2018-10-28 02:30:00+01:00    4
2018-10-28 03:00:00+01:00    5
2018-10-28 03:30:00+01:00    6

```

```

dtype: int64

In some cases, inferring the DST is impossible. In such cases, you can
pass an ndarray to the ambiguous parameter to set the DST explicitly

>>> s = pd.Series(range(3),
...                 index=pd.DatetimeIndex(['2018-10-28 01:20:00',
...                                         '2018-10-28 02:36:00',
...                                         '2018-10-28 03:46:00']))
>>> s.tz_localize('CET', ambiguous=np.array([True, True, False]))
2018-10-28 01:20:00+02:00    0
2018-10-28 02:36:00+02:00    1
2018-10-28 03:46:00+01:00    2
dtype: int64

If the DST transition causes nonexistent times, you can shift these
dates forward or backward with a timedelta object or ``shift_forward``
or ``shift_backward``.

>>> s = pd.Series(range(2),
...                 index=pd.DatetimeIndex(['2015-03-29 02:30:00',
...                                         '2015-03-29 03:30:00']))
>>> s.tz_localize('Europe/Warsaw', nonexistent='shift_forward')
2015-03-29 03:00:00+02:00    0
2015-03-29 03:30:00+02:00    1
dtype: int64
>>> s.tz_localize('Europe/Warsaw', nonexistent='shift_backward')
2015-03-29 01:59:59.999999999+01:00    0
2015-03-29 03:30:00+02:00    1
dtype: int64
>>> s.tz_localize('Europe/Warsaw', nonexistent=pd.Timedelta('1H'))
2015-03-29 03:30:00+02:00    0
2015-03-29 03:30:00+02:00    1
dtype: int64

xs(self, key, axis=0, level=None, drop_level: 'bool_t' = True)
Return cross-section from the Series/DataFrame.

This method takes a 'key' argument to select data at a particular
level of a MultiIndex.

Parameters
-----
key : label or tuple of label
    Label contained in the index, or partially in a MultiIndex.
axis : {0 or 'index', 1 or 'columns'}, default 0
    Axis to retrieve cross-section on.
level : object, defaults to first n levels (n=1 or len(key))
    In case of a key partially contained in a MultiIndex, indicate
    which levels are used. Levels can be referred by label or position.
drop_level : bool, default True
    If False, returns object with same levels as self.

Returns
-----
Series or DataFrame
    Cross-section from the original Series or DataFrame
    corresponding to the selected index levels.

See Also
-----
DataFrame.loc : Access a group of rows and columns
    by label(s) or a boolean array.
DataFrame.iloc : Purely integer-location based indexing
    for selection by position.

Notes
-----
`xs` can not be used to set values.

MultiIndex Slicers is a generic way to get/set values on
any level or levels.
It is a superset of `xs` functionality, see
:ref:`MultiIndex Slicers <advanced.mi_slicers>`.

Examples
-----
>>> d = {'num_legs': [4, 4, 2, 2],
...         'num_wings': [0, 0, 2, 2],
...         'class': ['mammal', 'mammal', 'mammal', 'bird'],
...         'animal': ['cat', 'dog', 'bat', 'penguin'],
...         'locomotion': ['walks', 'walks', 'flies', 'walks']}
>>> df = pd.DataFrame(data=d)
>>> df.set_index(['class', 'animal', 'locomotion'])
>>> df
      num_legs  num_wings
class animal  locomotion
mammal cat    walks        4      0
          dog    walks        4      0
          bat    flies        2      2
bird   penguin walks        2      2

Get values at specified index

>>> df.xs('mammal')
      num_legs  num_wings
animal locomotion
cat    walks        4      0
dog    walks        4      0
bat    flies        2      2

Get values at several indexes

>>> df.xs(['mammal', 'dog'])
      num_legs  num_wings
locomotion
walks        4      0

Get values at specified index and level

>>> df.xs('cat', level=1)
      num_legs  num_wings
class locomotion
mammal walks        4      0

Get values at several indexes and levels

>>> df.xs(['bird', 'walks'],
...         level=[0, 'locomotion'])
      num_legs  num_wings
animal
penguin      2        2

Get values at specified column and axis

>>> df.xs('num_wings', axis=1)
      num_legs  num_wings
class animal  locomotion
mammal cat    walks        0
          dog    walks        0
          bat    flies        2
bird   penguin walks        2

```

```
Name: num_wings, dtype: int64
-----
Readonly properties inherited from pandas.core.generic.NDFrame:
dtypes
    Return the dtypes in the DataFrame.

    This returns a Series with the data type of each column.
    The result's index is the original DataFrame's columns. Columns
    with mixed types are stored with the ``object`` dtype. See
    :ref:`the User Guide <basics.dtypes>` for more.

    Returns
    ------
    pandas.Series
        The data type of each column.

Examples
-----
>>> df = pd.DataFrame({'float': [1.0],
...                      'int': [1],
...                      'datetime': [pd.Timestamp('20180310')],
...                      'string': ['foo']})
>>> df.dtypes
float            float64
int             int64
datetime       datetime64[ns]
string          object
dtype: object

empty
    Indicator whether Series/DataFrame is empty.

    True if Series/DataFrame is entirely empty (no items), meaning any of the
    axes are of length 0.

    Returns
    ------
    bool
        If Series/DataFrame is empty, return True, if not return False.

See Also
-----
Series.dropna : Return series without null values.
DataFrame.dropna : Return DataFrame with labels on given axis omitted
    where (all or any) data are missing.

Notes
-----
If Series/DataFrame contains only NaNs, it is still not considered empty.

See
the example below.

Examples
-----
An example of an actual empty DataFrame. Notice the index is empty:

>>> df_empty = pd.DataFrame({'A' : []})
>>> df_empty
Empty DataFrame
Columns: [A]
Index: []
>>> df_empty.empty
True

If we only have NaNs in our DataFrame, it is not considered empty! We
will need to drop the NaNs to make the DataFrame empty:

>>> df = pd.DataFrame({'A' : [np.nan]})
>>> df
   A
0  NaN
>>> df.empty
False
>>> df.dropna().empty
True

>>> ser_empty = pd.Series({'A' : []})
>>> ser_empty
A   []
dtype: object
>>> ser_empty.empty
False
>>> ser_empty = pd.Series()
>>> ser_empty.empty
True

flags
    Get the properties associated with this pandas object.

    The available flags are

    * :attr:`Flags.allows_duplicate_labels`

See Also
-----
Flags : Flags that apply to pandas objects.
DataFrame.attrs : Global metadata applying to this dataset.

Notes
-----
"Flags" differ from "metadata". Flags reflect properties of the
pandas object (the Series or DataFrame). Metadata refer to properties
of the dataset, and should be stored in :attr:`DataFrame.attrs`.

Examples
-----
>>> df = pd.DataFrame({"A": [1, 2]})
>>> df.flags
<Flags(allow_duplicates=True)>

    Flags can be get or set using ``````

    >>> df.flags.allows_duplicate_labels
True
    >>> df.flags.allows_duplicate_labels = False

    Or by slicing with a key

    >>> df.flags["allows_duplicate_labels"]
False
    >>> df.flags["allows_duplicate_labels"] = True

ndim
    Return an int representing the number of axes / array dimensions.

    Return 1 if Series. Otherwise return 2 if DataFrame.

See Also
-----
```

```
ndarray.ndim : Number of array dimensions.

Examples
-----
>>> s = pd.Series({'a': 1, 'b': 2, 'c': 3})
>>> s.ndim
1

>>> df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
>>> df.ndim
2

size
Return an int representing the number of elements in this object.

Return the number of rows if Series. Otherwise return the number of
rows times number of columns if DataFrame.

See Also
-----
ndarray.size : Number of elements in the array.

Examples
-----
>>> s = pd.Series({'a': 1, 'b': 2, 'c': 3})
>>> s.size
3

>>> df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
>>> df.size
4

Data descriptors inherited from pandas.core.generic.NDFrame:

attrs
Dictionary of global attributes of this dataset.

.. warning::
    attrs is experimental and may change without warning.

See Also
-----
DataFrame.flags : Global flags applying to this object.

Data and other attributes inherited from pandas.core.generic.NDFrame:

__array_priority__ = 1000

Methods inherited from pandas.core.base.PandasObject:

__sizeof__(self) -> 'int'
    Generates the total memory usage for an object that returns
    either a value or Series of values

Methods inherited from pandas.core.accessor.DirNamesMixin:

__dir__(self) -> 'list[str]'
    Provide method name lookup and completion.

Notes
-----
Only provide 'public' methods.

Data descriptors inherited from pandas.core.accessor.DirNamesMixin:

__dict__
    dictionary for instance variables (if defined)

__weakref__
    list of weak references to the object (if defined)

Readonly properties inherited from pandas.core.indexing.IndexingMixin:

at
Access a single value for a row/column label pair.

Similar to ``loc``, in that both provide label-based lookups. Use
``at`` if you only need to get or set a single value in a DataFrame
or Series.

Raises
-----
KeyError
    If 'label' does not exist in DataFrame.

See Also
-----
DataFrame.iat : Access a single value for a row/column pair by integer
    position.
DataFrame.loc : Access a group of rows and columns by label(s).
Series.at : Access a single value using a label.

Examples
-----
>>> df = pd.DataFrame([[0, 2, 3], [0, 4, 1], [10, 20, 30]],
...                      index=[4, 5, 6], columns=['A', 'B', 'C'])
>>> df
   A   B   C
4   0   2   3
5   0   4   1
6  10  20  30

Get value at specified row/column pair

>>> df.at[4, 'B']
2

Set value at specified row/column pair

>>> df.at[4, 'B'] = 10
>>> df.at[4, 'B']
10

Get value within a Series

>>> df.loc[5].at['B']
4

iat
Access a single value for a row/column pair by integer position.

Similar to ``iloc``, in that both provide integer-based lookups. Use
``iat`` if you only need to get or set a single value in a DataFrame
or Series.
```

```

Raises
-----
IndexError
    When integer position is out of bounds.

See Also
-----
DataFrame.at : Access a single value for a row/column label pair.
DataFrame.loc : Access a group of rows and columns by label(s).
DataFrame.iloc : Access a group of rows and columns by integer
position(s).

Examples
-----
>>> df = pd.DataFrame([[0, 2, 3], [0, 4, 1], [10, 20, 30]],
...                      columns=['A', 'B', 'C'])
>>> df
   A   B   C
0  0   2   3
1  0   4   1
2 10  20  30

Get value at specified row/column pair

>>> df.iat[1, 2]
1

Set value at specified row/column pair

>>> df.iat[1, 2] = 10
>>> df.iat[1, 2]
10

Get value within a series

>>> df.loc[0].iat[1]
2

iloc
Purely integer-location based indexing for selection by position.

```.iloc[]``` is primarily integer position based (from ``0`` to
``length-1`` of the axis), but may also be used with a boolean
array.

Allowed inputs are:
- An integer, e.g. ``5``.
- A list or array of integers, e.g. ``[4, 3, 0]``.
- A slice object with ints, e.g. ``1:7``.
- A boolean array.
- A ``callable`` function with one argument (the calling Series or
DataFrame) and that returns valid output for indexing (one of the
above).

This is useful in method chains, when you don't have a reference to the
calling object, but would like to base your selection on some value.

```.iloc[]`` will raise ``IndexError`` if a requested indexer is
out-of-bounds, except *slice* indexers which allow out-of-bounds
indexing (this conforms with python/numpy *slice* semantics).

See more at :ref:`Selection by Position <indexing.integer>`.

See Also
-----
DataFrame.iat : Fast integer location scalar accessor.
DataFrame.loc : Purely label-location based indexer for selection by
label.
Series.iloc : Purely integer-location based indexing for
selection by position.

Examples
-----
>>> mydict = [{"a": 1, "b": 2, "c": 3, "d": 4},
...             {"a": 100, "b": 200, "c": 300, "d": 400},
...             {"a": 1000, "b": 2000, "c": 3000, "d": 4000}]
>>> df = pd.DataFrame(mydict)
>>> df
   a   b   c   d
0   1   2   3   4
1  100  200  300  400
2 1000 2000 3000 4000

**Indexing just the rows**

With a scalar integer.

>>> type(df.iloc[0])
<class 'pandas.core.series.Series'>
>>> df.iloc[0]
a    1
b    2
c    3
d    4
Name: 0, dtype: int64

With a list of integers.

>>> df.iloc[[0]]
   a   b   c   d
0   1   2   3   4
>>> type(df.iloc[[0]])
<class 'pandas.core.frame.DataFrame'>

>>> df.iloc[[0, 1]]
   a   b   c   d
0   1   2   3   4
1  100  200  300  400

With a `slice` object.

>>> df.iloc[:3]
   a   b   c   d
0   1   2   3   4
1  100  200  300  400
2 1000 2000 3000 4000

With a boolean mask the same length as the index.

>>> df.iloc[[True, False, True]]
   a   b   c   d
0   1   2   3   4
2 1000 2000 3000 4000

With a callable, useful in method chains. The 'x' passed
to the ``lambda`` is the DataFrame being sliced. This selects
the rows whose index label even.

>>> df.iloc[lambda x: x.index % 2 == 0]
   a   b   c   d
0   1   2   3   4
2 1000 2000 3000 4000

```

```

    0   1   2   3   4
2 1000 2000 3000 4000

**Indexing both axes**

You can mix the indexer types for the index and columns. Use `:` to
select the entire axis.

With scalar integers.

>>> df.iloc[0, 1]
2

With lists of integers.

>>> df.iloc[[0, 2], [1, 3]]
   b   d
0   2   4
2 2000 4000

With 'slice' objects.

>>> df.iloc[1:3, 0:3]
   a   b   c
1 100 200 300
2 1000 2000 3000

With a boolean array whose length matches the columns.

>>> df.iloc[:, [True, False, True, False]]
   a   c
0   1   3
1 100 300
2 1000 3000

With a callable function that expects the Series or DataFrame.

>>> df.iloc[:, lambda df: [0, 2]]
   a   c
0   1   3
1 100 300
2 1000 3000

loc
Access a group of rows and columns by label(s) or a boolean array.

`loc[]` is primarily label based, but may also be used with a
boolean array.

Allowed inputs are:

- A single label, e.g. ``5`` or ``'a'``, (note that ``5`` is
interpreted as a *label* of the index, and **never** as an
integer position along the index).
- A list or array of labels, e.g. ``['a', 'b', 'c']``.
- A slice object with labels, e.g. ``'a':'f'``.

.. warning:: Note that contrary to usual python slices, **both** the
start and the stop are included

- A boolean array of the same length as the axis being sliced,
e.g. ``[True, False, True]``.
- An alignable boolean Series. The index of the key will be aligned before
masking.
- An alignable Index. The Index of the returned selection will be the
input.
- A ``callable`` function with one argument (the calling Series or
DataFrame) and that returns valid output for indexing (one of the above)

See more at :ref:`Selection by Label <indexing.labels>`.

Raises
-----
KeyError
    If any items are not found.
IndexingError
    If an indexed key is passed and its index is unalignable to the frame
index.

See Also
-----
DataFrame.at : Access a single value for a row/column label pair.
DataFrame.iloc : Access group of rows and columns by integer position(s).
DataFrame_xs : Returns a cross-section (row(s) or column(s)) from the
Series/DataFrame.
Series.loc : Access group of values using labels.

Examples
-----
**Getting values**

>>> df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
...     index=['cobra', 'viper', 'sidewinder'],
...     columns=['max_speed', 'shield'])
>>> df
      max_speed  shield
cobra         1        2
viper         4        5
sidewinder    7        8

Single label. Note this returns the row as a Series.

>>> df.loc['viper']
max_speed    4
shield       5
Name: viper, dtype: int64

List of labels. Note using ``[]`` returns a DataFrame.

>>> df.loc[['viper', 'sidewinder']]
      max_speed  shield
viper         4        5
sidewinder    7        8

Single label for row and column

>>> df.loc['cobra', 'shield']
2

Slice with labels for row and single label for column. As mentioned
above, note that both the start and stop of the slice are included.

>>> df.loc['cobra':'viper', 'max_speed']
cobra    1
viper    4
Name: max_speed, dtype: int64

Boolean list with the same length as the row axis

>>> df.loc[[False, False, True]]
      max_speed  shield

```

```

sidewinder      7      8
Alignable boolean Series:
>>> df.loc[pd.Series([False, True, False],
...                 index=['viper', 'sidewinder', 'cobra'])]
          max_speed shield
sidewinder      7      8
Index (same behavior as ``df.reindex``)
>>> df.loc[pd.Index(["cobra", "viper"], name="foo")]
          max_speed shield
foo
cobra           1      2
viper           4      5
Conditional that returns a boolean Series
>>> df.loc[df['shield'] > 6]
          max_speed shield
sidewinder      7      8
Conditional that returns a boolean Series with column labels specified
>>> df.loc[df['shield'] > 6, ['max_speed']]
          max_speed
sidewinder      7
Callable that returns a boolean Series
>>> df.loc[lambda df: df['shield'] == 8]
          max_speed shield
sidewinder      7      8
**Setting values**
Set value for all items matching the list of labels
>>> df.loc[['viper', 'sidewinder'], ['shield']] = 50
>>> df
          max_speed shield
cobra           1      2
viper           4     50
sidewinder      7     50
Set value for an entire row
>>> df.loc['cobra'] = 10
>>> df
          max_speed shield
cobra          10     10
viper           4     50
sidewinder      7     50
Set value for an entire column
>>> df.loc[:, 'max_speed'] = 30
>>> df
          max_speed shield
cobra          30     10
viper          30     50
sidewinder      30     50
Set value for rows matching callable condition
>>> df.loc[df['shield'] > 35] = 0
>>> df
          max_speed shield
cobra          30     10
viper           0      0
sidewinder      0      0
**Getting values on a DataFrame with an index that has integer labels**
Another example using integers for the index
>>> df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
...                   index=[7, 8, 9], columns=['max_speed', 'shield'])
>>> df
          max_speed shield
7             1      2
8             4      5
9             7      8
Slice with integer labels for rows. As mentioned above, note that both
the start and stop of the slice are included.
>>> df.loc[7:9]
          max_speed shield
7             1      2
8             4      5
9             7      8
**Getting values with a MultiIndex**
A number of examples using a DataFrame with a MultiIndex
>>> tuples = [
...     ('cobra', 'mark i'), ('cobra', 'mark ii'),
...     ('sidewinder', 'mark i'), ('sidewinder', 'mark ii'),
...     ('viper', 'mark ii'), ('viper', 'mark iii')
... ]
>>> index = pd.MultiIndex.from_tuples(tuples)
>>> values = [[12, 2], [0, 4], [10, 20],
...             [1, 4], [7, 1], [16, 36]]
>>> df = pd.DataFrame(values, columns=['max_speed', 'shield'],
...                     index=index)
>>> df
          max_speed shield
cobra  mark i       12      2
              mark ii      0      4
sidewinder  mark i       10     20
              mark ii      1      4
viper    mark ii       7      1
              mark iii     16     36
Single label. Note this returns a DataFrame with a single index.
>>> df.loc['cobra']
          max_speed shield
mark i       12      2
mark ii      0      4
Single index tuple. Note this returns a Series.
>>> df.loc[('cobra', 'mark ii')]
max_speed      0
shield         4
Name: (cobra, mark ii), dtype: int64

```

```

Single label for row and column. Similar to passing in a tuple, this
returns a Series.

>>> df.loc['cobra', 'mark i']
max_speed    12
shield       2
Name: (cobra, mark i), dtype: int64

Single tuple. Note using ``[[]]`` returns a DataFrame.

>>> df.loc[['cobra', 'mark ii']]
           max_speed  shield
cobra mark ii          0      4

Single tuple for the index with a single label for the column

>>> df.loc[('cobra', 'mark i'), 'shield']
2

Slice from index tuple to single label

>>> df.loc[('cobra', 'mark i'):'viper']
           max_speed  shield
cobra   mark i       12      2
                  mark ii      0      4
sidewinder mark i       10     20
                  mark ii      1      4
viper    mark ii       7      1
                  mark iii     16     36

Slice from index tuple to index tuple

>>> df.loc[('cobra', 'mark i'):(('viper', 'mark ii'))]
           max_speed  shield
cobra   mark i       12      2
                  mark ii      0      4
sidewinder mark i       10     20
                  mark ii      1      4
viper    mark ii       7      1

-----
Methods inherited from pandas.core.arraylike.OpsMixin:

__add__(self, other)
__and__(self, other)
__eq__(self, other)
    Return self==value.
__floordiv__(self, other)
__ge__(self, other)
    Return self>=value.
__gt__(self, other)
    Return self>value.
__le__(self, other)
    Return self<=value.
__lt__(self, other)
    Return self<value.
__mod__(self, other)
__mul__(self, other)
__ne__(self, other)
    Return self!=value.
__or__(self, other)
__pow__(self, other)
__radd__(self, other)
__rand__(self, other)
__rfloordiv__(self, other)
__rmod__(self, other)
__rmul__(self, other)
__ror__(self, other)
__rpow__(self, other)
__rsub__(self, other)
__rtruediv__(self, other)
__rxor__(self, other)
__sub__(self, other)
__truediv__(self, other)
__xor__(self, other)

-----
Data and other attributes inherited from pandas.core.arraylike.OpsMixin:

__hash__ = None

```

👉 Try it yourself!

Make a list of the shape of all of the tables on the syllabus Achievements page.

```

achievements_url =
'https://rhodyprog4ds.github.io/BrownFall21/syllabus/achievements.html'

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

```

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

This solution uses a list comprehension which allows us to compress a loop. It's equivalent to the following with a for loop

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

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

5.7. Lambdas and Dictionaries for switching

What if we want to print out the first column for the DataFrame if it has more than 3 columns and the whole thing if it has 3 or less columns?

Two ways of writing a function

```
# with the def key
def first_col_f(d):
    return d[d.columns[0]]

# lambda (anonymous function)
first_col_l = lambda d: d[d.columns[0]]

first_col_f(help_df) == first_col_l(help_df)
```

```
0    True
1    True
2    True
3    True
4    True
5    True
6    True
Name: Day, dtype: bool
```

Question from class

Python does have [ternary operators](#) but the dictionary is a more common way to achieve this and goal and more common patterns are better for readability

Further Reading

You can refer to the [official documentation](#) on [lambda](#) functions for a brief syntax description or this [tutorial on Real Python](#) for more context, history, and examples.

Important

We'll see lambdas again and again. Starting with summarizing data and again when cleaning. Understanding how to use these will help.

Try it yourself

read the code excerpt above carefully and try to match up the parts of a function: its name, the parameter list, and the body. Try writing your own lambda function.

Lambdas are an example of an [anonymous function](#)

We can put functions in dictionaries, or even define a lambda right in the dictionary.

```
df_display = {True: lambda d: d[d.columns[0]],
              False: lambda d: d}

for df in pd.read_html(achievements_url):
    _, n_cols = df.shape
    print(df_display[n_cols>3](df))
```

```

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

    skills
    Unnamed: 2_level_1
0      process
1  [access, prepare, summarize]
2  [summarize, visualize]
3  [prepare, summarize, visualize]
4  [access, construct, summarize]
5  [classification, evaluate]
6  [classification, evaluate]
7  [regression, evaluate]
8  [clustering, evaluate]
9  [optimize, tools]
10  [compare, tools]
11  [unstructured]
12  [unstructured, tools]
13  [tools, compare]
0   python
1   process
2   access
3   construct
4   summarize
5   visualize
6   prepare
7   classification
8   regression
9   clustering
10  evaluate
11  optimize
12  compare
13  representation
14  workflow
Name: (Unnamed: 0_level_0, keyword), dtype: object
0      python
1      process
2      access
3      construct
4      summarize
5      visualize
6      prepare
7      classification
8      regression
9      clustering
10     evaluate
11     optimize
12     compare
13     representation
14     workflow
Name: (Unnamed: 0_level_0, keyword), dtype: object
0      python
1      process
2      access
3      construct
4      summarize
5      visualize
6      prepare
7      classification
8      regression
9      clustering
10     evaluate
11     optimize
12     compare
13     representation
14     workflow
Name: (Unnamed: 0_level_0, keyword), dtype: object

```

In that excerpt `df_display` has a key that is defined to be true or false. The value for each item in the dictionary (separated by commas ,) is a lambda function, both of which take a parameter `d`, and one of which returns the first column `d[d.columns[0]]` and the other for which returns the whole data frame `d`.

In the loop, we set index into the dictionary with the key `n_cols > 3` with `df_display[n_cols>3]` sometimes it will be true and other times it will be false, which matches the two keys defined in the dictionary. Then the parameter it takes is `d`, which we pass the whole data frame `df` with the `(df)` at the end of the line.

👉 Try it Yourself!

What does the `_` do?

```

Try using that dictionary outside of the loop. What is the value for a given key
if you print it out like 'df_display[a_key_value]'? What if you use 'True' or
'False' directly? What if you try a number? What is the type of that? What if you
pass something that's not a DataFrame as the parameter? Make notes about these outputs

```

5.8. Questions after class

Ram Token Opportunity

add a question with a pull request; earn 1-2 ram tokens for submitting a question with the answer (with sources)

5.9. More Practice

- What `type` is the shape of a `pandas.DataFrame`?
- use a list comprehension to create a list that you could use as column names for data that consists of `N` measurements. Set `N=5` for now, but you suspect that the number might change.

```

N = 4
meas_cols = ['meas' + str(i) for i in range(N)]

```

- create a list of items with different types, then Create a dictionary with the types as keys using a dictionary comprehension. Dictionary comprehensions are similar to list comprehensions, in their form.

```
about_prof_brown_list = ['Dr. ', 134, ['CSC310', 'CSC592', 'CSC392']]
about_prof_brown_dict = {type(fact):fact for fact in about_prof_brown_list}
```

- Create a `lambda` function to print return the first 2 rows of a data frame

Ram Token Opportunity

Contribute possible practice questions to the notes using the suggest an edit button behind the GitHub menu at the top of the page.

6. Exploratory Data Analysis

```
import pandas as pd
```

6.1. Staying Organized

- See the new [File structure](#) section.
- Be sure to accept assignments and close the feedback PR if you will not work on them

6.2. Data Frame from lists of Lists

On Friday, we collectively made a list of strings

```
# %load http://drsmrb.co/310read
# share a sentence or a few words about how class is going or
# one takeaway you have from class so far.
# and attribute with a name after a hyphen(-)
# You can remain anonymous (this page & the notes will be fully public)
# by attributing it to a celebrity or pseudonym, but include *some* sort of
# attribution
sentence_list = [
    'Programming is a Practice. - Dr. Sarah Brown',
    "So far it's going pretty good. - Matt",
    'Pretty good pace, Githu is still a little confusing is all - A',
    "Class is going well, I'm really excited for the new skills that I will attain
through this course. - Diondra",
    'Good straight forward - Adam',
    'Good pace and engaging - Jacob',
    'I like cheese - Anon',
    'Going well, I enjoy python - Aiden',
    "Class is going very well, I'm excited to learn more about manipulating data sets
- Greg",
    'Really enjoying the class so far, Clear instructions and outcomes - Michael',
    'So far this class is going well, very engaging lectures. - Anon',
    'Great pace and notes have been really useful - Brandon',
    'Good pace and easy to follow - Muhammad',
    'Class is going well and engaging, but also a little difficult getting into the
swing of things - Isaiah',
    'Well paced, informative, and helpful - Vinnie',
    'Spectacular - Michael Jackson',
    'Very interesting! I am enjoying it a lot so far. Getting to experience pandas as
well as using github/jupyter has been cool. One thing I would change though is
slowing down the pace a bit. - Max'
]
```

We can check that by using type, first on the whole thing

```
type(sentence_list)
```

```
list
```

And then we can index into the list. Lists can be indexed by integers:

```
sentence_list[4]
```

```
'Good straight forward - Adam'
```

is one item from the list and then check its type:

```
type(sentence_list[4])
```

```
str
```

First, we'll convert our list of strings, to a list of lists with a list comprehension. This will [iterate](#) over each string in that list and apply the string method `split`. Since we passed the parameter `'-'`, it will split at that character.

```
[sent_attr.split('-') for sent_attr in sentence_list]
```

Question From Class

The `split` method will split at every occurrence of the character passed.

Try it Yourself

Try splitting based on a different character (eg `' '`). What happens?

```

[['Programming is a Practice. ', ' Dr. Sarah Brown'],
['So far it's going pretty good. ', ' Matt'],
['Pretty good pace, Github is still a little confusing is all ', ' A'],
['Class is going well, I'm really excited for the new skills that I will attain through this course. ',
'Diondra'],
['Good straight forward ', ' Adam'],
['Good pace and engaging ', ' Jacob'],
['I like cheese ', ' Anon'],
['Going well, I enjoy python ', ' Aiden'],
['Class is going very well, I'm excited to learn more about manipulating data sets ',
' Greg'],
['Really enjoying the class so far, Clear instructions and outcomes ',
'Michael'],
['So far this class is going well, very engaging lectures. ', ' Anon'],
['Great pace and notes have been really useful ', ' Brandon'],
['Good pace and easy to follow ', ' Muhammad'],
['Class is going well and engaging, but also a little difficult getting into the swing of things ',
' Isaiah'],
['Well paced, informative, and helpful ', ' Vinnie'],
['Spectacular ', ' Michael Jackson'],
['Very interesting! I am enjoying it a lot so far. Getting to experience pandas as well as using github/jupyter has been cool. One thing I would change though is slowing down the pace a bit. ',
' Max']]
```

If we save it to a variable, we can analyze it better, for example indexing it

```

list_of_lists = [sent_attr.split('-') for sent_attr in sentence_list]
list_of_lists[0]
```

```
'Programming is a Practice. ', ' Dr. Sarah Brown'
```

This is a list, which we can check with:

```

type(list_of_lists[0])
```

```
list
```

If we take one item from that, it's a string

```

list_of_lists[0][1]
```

```
' Dr. Sarah Brown'
```

The list of lists is the same length as our original sentence_list, because it was made from that.

```

len(sentence_list)
```

```
17
```

```

len(list_of_lists)
```

```
17
```

Then we can pass our list of lists to the DataFrame constructor and set the column headings with the `columns` parameter, which accepts a list.

```

pd.DataFrame([sent_attr.split('-') for sent_attr in sentence_list],
columns=['sentence', 'attribution'])
```

	sentence	attribution
0	Programming is a Practice. Dr. Sarah Brown	
1	So far it's going pretty good.	Matt
2	Pretty good pace, Github is still a little con...	A
3	Class is going well, I'm really excited for th...	Diondra
4	Good straight forward	Adam
5	Good pace and engaging	Jacob
6	I like cheese	Anon
7	Going well, I enjoy python	Aiden
8	Class is going very well, I'm excited to learn...	Greg
9	Really enjoying the class so far, Clear instru...	Michael
10	So far this class is going well, very engaging...	Anon
11	Great pace and notes have been really useful	Brandon
12	Good pace and easy to follow	Muhammad
13	Class is going well and engaging, but also a l...	Isaiah
14	Well paced, informative, and helpful	Vinnie
15	Spectacular Michael Jackson	
16	Very interesting! I am enjoying it a lot so fa...	Max

💡 Hint

We built the list of lists here with a list comprehension because it's only a few items, but for a list of longer lists, you might use a for loop and `append`

💡 Question From Class

While the list comprehension `iterate` over each string in that list, because it's a list, in order to `index` into it, we have to use an integer.

Iterating is taking each member in an object in turn, indexing is picking out a specified item. Iterating typically is done with the `for` keyword (either in a loop or a comprehension), indexing is done with `[]`

6.3. Summarizing Data

```

coffee_data_url = 'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/robusta_data_cleaned.csv'
coffee_df = pd.read_csv(coffee_data_url)

```

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.

```
coffee_df.head()
```

	Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	...	Color	Category.Two.Defects	E
0	1	Robusta	ankole coffee producers coop	Uganda	kyangundu cooperative society	NaN	ankole coffee producers	0	ankole coffee producers coop	1488	...	Green	2	Ji
1	2	Robusta	nishant gurjer	India	sethuraman estate kaapi royale	25	sethuraman estate	14/1148/2017/21	kaapi royale	3170	...	NaN	2	3:
2	3	Robusta	andrew hetzel	India	sethuraman estate	NaN	NaN	0000	sethuraman estate	1000m	...	Green	0	A
3	4	Robusta	ugacof	Uganda	ugacof project area	NaN	ugacof	0	ugacof ltd	1212	...	Green	7	:
4	5	Robusta	katuka development trust ltd	Uganda	katikamu capca farmers association	NaN	katuka development trust	0	katuka development trust ltd	1200-1300	...	Green	3	Ji

5 rows × 44 columns

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

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

```
coffee_df.describe()
```

	Unnamed: 0	Number.of.Bags	Harvest.Year	Fragrance...Aroma	Flavor	Aftertaste	Salt...Acid	Bitter...Sweet	Mouthfeel	Uniform.Cup	...	Balance	Cupper.Poin
count	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	...	28.000000	28.00000
mean	14.500000	168.000000	2013.964286	7.702500	7.630714	7.559643	7.657143	7.675714	7.506786	9.904286	...	7.541786	7.7614:
std	8.225975	143.226317	1.346660	0.296156	0.303656	0.342469	0.261773	0.317063	0.725152	0.238753	...	0.526076	0.3305l
min	1.000000	1.000000	2012.000000	6.750000	6.670000	6.500000	6.830000	6.670000	5.080000	9.330000	...	5.250000	6.9200l
25%	7.750000	1.000000	2013.000000	7.580000	7.560000	7.397500	7.560000	7.580000	7.500000	10.000000	...	7.500000	7.5800l
50%	14.500000	170.000000	2014.000000	7.670000	7.710000	7.670000	7.710000	7.750000	7.670000	10.000000	...	7.670000	7.8300l
75%	21.250000	320.000000	2015.000000	7.920000	7.830000	7.770000	7.830000	7.830000	7.830000	10.000000	...	7.830000	7.9200l
max	28.000000	320.000000	2017.000000	8.330000	8.080000	7.920000	8.000000	8.420000	8.250000	10.000000	...	8.000000	8.5800l

8 rows × 21 columns

We can also select one variable at a time

```
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

To dig in on what the quantiles really mean, we can compute one manually.

First, we sort the data, then for the 25%, we select the point in index 6 because because there are 28 values.

```

balance_sorted = coffee_df['Balance'].sort_values().values
balance_sorted[6]

```

7.5

We can also extract each of the statistics that the `describe` method calculates individually, by name. The quantiles are tricky, we can't use `.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(.25)
```

7.560000000000005

We can also pass other values

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

7.83

Calculate the mean of the `Aftertaste` column:

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

7.559642857142856

💡 further reading

On the [documentation page for describe](#) the "See Also" shows the links to the documentation of most of the individual functions. This is a good way to learn about other things, or find something when you are not quite sure what it would be named. Go to a function that's similar to what you want and then look at the related functions.

6.4. What about the nonnumerical variables?

For example the color

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

0 Green
1 NaN
2 Green
3 Green
4 Green
Name: Color, dtype: object

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

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

Green 20
Blue-Green 3
Bluish-Green 2
None 1
Name: Color, 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?

What country is most prevalent in this dataset?

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

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

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

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

'India'

❓ Question From Class

Q: Can we calculate the mode to find the most prevalent? A: Yes. We can also use the mode function, which works on both numerical or nonnumerical values.

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

0 India
Name: Country.of.Origin, dtype: object

6.5. Questions After Class

6.5.1. General Questions

6.5.1.1. How to know what functions are compatible with other functions?

This is something that builds up over time, but one thing to look for is to check the types. That's why we've been using `type()` so often in class. Having a goal for your analysis will also help.

6.5.1.2. Best place to find all the individual functions based on the `.describe()` function

Each one goes by the same name, mostly, but that also the [Pandas documentation](#) is the best way to find more information on every pandas method. On any function, see the See also section too.

6.5.1.3. Are there panda functions to read files other than CSV?

Yes! See the hint on [assignment_2](#) for more information on this

6.5.2. Clarifying

6.5.2.1. Is `sent_attr` in the DataFrame loop its own variable that we assign, or is it a base Python function?

That's a variable that we created, it's equivalent to a loop variable.

6.5.2.2. Difference between `%load` and how we've been importing links to datasets in previous classes?

the pandas functions read data in and put it specifically into a DataFrame.

The load magic reads a file in and basically pastes it into the jupyter cell. This one doesn't actually put the data into memory in python.

6.5.2.3. When I ran `[sent_attr.split('-') for sent_attr in sentence_list]` in Jupyter, I got a nameerror

I also did at first, because I hadn't yet run the cell above so `sentence_list` was not defined.

Why do we use the value quantile instead of using quartile since we can use mean to find mean?

It's just the name of the method.

6.5.3. Course Admin and Assignment Questions

6.5.3.1. When are the achievements we earn in class going to be inputted in brightspace?

Assignment achievements will be posted after grading is complete. Level 1s from class will be posted most weeks on Friday or over the weekend. Right now we're a little behind as we get everything set up.

6.5.3.2. Is there anything we have to do for the assignment after committing the changes to the files?

Yes, after committing, push. Once you can see your files on the GitHub website interface, we can see them for grading

6.5.3.3. When we push the homework are we pushing it to the portfolio?

No, when you accept the assignment it will create a new repository for that assignment. That's why it's important to accept each assignment.

6.5.3.4. Still a little unsure about what the num_numerical specifically is, is it just the number of numerical columns?

Yes the number of numerical columns.

6.6. More Practice

1. Produce a table with the mean for each score.
2. Which variables have the most missing data?
3. What's the total number of bags of coffee produced?
4. Which ratings have similar ranges? (max, min)
5. Which rating are most consistent across coffees?
6. What score cutoff could you apply in order to select the top 10 best for Balance?

7. Visualization

```
import pandas as pd
import seaborn as sns
sns.set_theme(palette= "colorblind")
```

7.1. Jupyter FAQ

Question from class

Why doesn't my jupyter print things out that are on the last line?

When we create a variable and then put that on the last line of a cell, jupyter displays it.

```
name = 'sarah'
name
```



```
'sarah'
```

How it displays it depends on the type

```
type(name)
```



```
str
```

For a string, it uses `print`

```
print(name)
```



```
sarah
```

so this and the one above look the same. For objects that have a `_repr_html_` method, jupyter uses that, and uses html to render the object in a more visually appealing way.

7.2. Review of describe

we're going to work with the arabica data today, because it's a little bigger and more interesting for plotting

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

We can describe it again, to see it has mostly the same variables we saw before, but some different as well.

```
coffee_df.describe()
```

	Unnamed: 0	Number.of.Bags	Aroma	Flavor	Aftertaste	Acidity	Body	Balance	Uniformity	Clean.Cup	Sweetness	Cupper.Poir
count	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000	1311.000000
mean	656.000763	153.887872	7.563806	7.518070	7.397696	7.533112	7.517727	7.517506	9.833394	9.83312	9.903272	7.4978
std	378.598733	129.733734	0.378666	0.399979	0.405119	0.381599	0.359213	0.406316	0.559343	0.77135	0.530832	0.4746
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	328.500000	14.500000	7.420000	7.330000	7.250000	7.330000	7.330000	7.330000	10.000000	10.000000	10.000000	7.2500
50%	656.000000	175.000000	7.580000	7.580000	7.420000	7.500000	7.500000	7.500000	10.000000	10.000000	10.000000	7.5000
75%	983.500000	275.000000	7.750000	7.750000	7.580000	7.750000	7.670000	7.750000	10.000000	10.000000	10.000000	7.7500
max	1312.000000	1062.000000	8.750000	8.830000	8.670000	8.750000	8.580000	8.750000	10.000000	10.000000	10.000000	10.000000

Question from class

Why do we need the `()` on the describe but not on just the data

As is often the case, again this comes back to the type.

```
[1]: type(coffee_df)
```

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

is a data frame which has the `_repr_html_` method

```
[2]: coffee_df
```

	Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	one ha	Color..	Category	Two Defects	Ex
0	1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural deve...	1950-	Green				
1	2	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultur...	1950-2200	Green				
2	3	Arabica	grounds for health admin	Guatemala	san marcos barrancas "san cristobal cuch	NaN	NaN	NaN	NaN	1600 - 1800 m	...	NaN		0	M
3	4	Arabica	yidnekachew dabessa	Ethiopia	yidnekachew dabessa coffee plantation	NaN	wolensu	NaN	yidnekachew debessa coffee plantation	1800-2200	...	Green		2	25
4	5	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural development plc	1950-2200	...	Green		2	A
...	
1306	1307	Arabica	juan carlos garcia lopez	Mexico	el centenario	NaN	esperanza, municipio juchique de ferrer, ve...	1104328663	terra mia	900	...	None		20	Se 17
1307	1308	Arabica	myriam kaplan-pasternak	Haiti	200 farms	NaN	coeb koperativ ekselsyo basen (350 members)	NaN	haiti coffee	~350m	...	Blue-Green		16	M
1308	1309	Arabica	exportadora atlantic, s.a.	Nicaragua	finca las marias	017-053-0211/017-053-0212	beneficio atlantic condega	017-053-0211/017-053-0212	exportadora atlantic s.a	1100	...	Green		5	J
1309	1310	Arabica	juan luis alvarado romero	Guatemala	finca el limon	NaN	beneficio serben	11/853/165	unicafe	4650	...	Green		4	M
1310	1312	Arabica	bismarck castro	Honduras	los hicaques	103	cigrah s.a de c.v.	13-111-053	cigrah s.a de c.v.	1400	...	Green		2	A

1311 rows x 44 columns

so it prints nicely as t did the `coffee_df.describe()`

If we leave the `()` off we don't get nice formatting

```
[3]: coffee_df.describe
```

Hint

when objects do not have the `_repr_html_` method their output includes the type

```

<bound method NDFrame.describe of      Unnamed: 0  Species
Owner Country.of.Origin \
0      1 Arabica           metad plc      Ethiopia
1      2 Arabica           metad plc      Ethiopia
2      3 Arabica grounds for health admin  Guatemala
3      4 Arabica yidnekachew dabessa    Ethiopia
4      5 Arabica           metad plc      Ethiopia
...
1306   1307 Arabica juan carlos garcia lopez  Mexico
1307   1308 Arabica myriam kaplan-pasternak  Haiti
1308   1309 Arabica exportadora atlantic, s.a. Nicaragua
1309   1310 Arabica juan luis alvarado romero  Guatemala
1310   1312 Arabica bismarck castro        Honduras

                                         Farm.Name          Lot.Number \
0                               metad plc      NaN
1                               metad plc      NaN
2  san marcos barrancas "san cristobal cuch  NaN
3  yidnekachew dabessa coffee plantation    NaN
4                               metad plc      NaN
...
1306   1307 el centenario          NaN
1307   1308 200 farms            NaN
1308   1309 finca las marias  017-053-0211/ 017-053-0212
1309   1310 finca el limon        NaN
1310   1310 los hicaques        103

                                         Mill \
0                               metad plc
1                               metad plc
2                               NaN
3                               wolensu
4                               metad plc
...
1306 la esperanza, municipio juchique de ferrer, ve...
1307 coeb koperativ ekselsyo basen (350 members)
1308 beneficio atlantic condega
1309 beneficio serben
1310 cigrah s.a de c.v

                                         ICO.Number          Company \
0  2014/2015      metad agricultural developmet plc
1  2014/2015      metad agricultural developmet plc
2  NaN             NaN
3  NaN yidnekachew debessa coffee plantation
4  2014/2015      metad agricultural developmet plc
...
1306  1104328662          terra mia
1307  NaN              haiti coffee
1308  017-053-0211/ 017-053-0212  exportadora atlantic s.a
1309  11/853/165          unicafe
1310  13-111-053         cigrah s.a de c.v

Altitude ... Color Category.Two.Defects \
0  1950-2200 ... Green  0
1  1950-2200 ... Green  1
2  1600 - 1800 m ... NaN  0
3  1800-2200 ... Green  2
4  1950-2200 ... Green  2
...
1306  900 ... None  20
1307  ~350m ... Blue-Green  16
1308  1100 ... Green  5
1309  4650 ... Green  4
1310  1400 ... Green  2

Expiration Certification.Body \
0 April 3rd, 2016 METAD Agricultural Development plc
1 April 3rd, 2016 METAD Agricultural Development plc
2 May 31st, 2011 Specialty Coffee Association
3 March 25th, 2016 METAD Agricultural Development plc
4 April 3rd, 2016 METAD Agricultural Development plc
...
1306 September 17th, 2013 AMECAFE
1307 May 24th, 2013 Specialty Coffee Association
1308 June 6th, 2018 Instituto Hondureño del Café
1309 May 24th, 2013 Asociacion Nacional Del Café
1310 April 28th, 2018 Instituto Hondureño del Café

Certification.Address \
0 309fcf77415a3661ae83e027ffef5dad786e44
1 309fcf77415a3661ae83e027ffef5dad786e44
2 36d0d00a3724338ba7937c52a378d085f2172daa
3 309fcf77415a3661ae83e027ffef5dad786e44
4 309fcf77415a3661ae83e027ffef5dad786e44
...
1306 59e396ad6e22a1c22b248f958e1da2bd8af85272
1307 36d0d00a3724338ba7937c52a378d085f2172daa
1308 b4660a57e9f8cc613ae5b8f02bfce8634c763ab4
1309 b1f20fe3a819fd8b2ee0eb8f0dc3da256604f1e53
1310 b4660a57e9f8cc613ae5b8f02bfce8634c763ab4

Certification.Contact unit_of_measurement \
0 19fef5a731de2db57d16da10287413f5f99bc2dd  m
1 19fef5a731de2db57d16da10287413f5f99bc2dd  m
2 0878a7d49d35ddb0fe2ce69a2062cce45a660  m
3 19fef5a731de2db57d16da10287413f5f99bc2dd  m
4 19fef5a731de2db57d16da10287413f5f99bc2dd  m
...
1306 0eb4ee5b3f47b20b49548a2fd1e7d4a2b70d0a7  m
1307 0878a7d49d35ddb0fe2ce69a2062cce45a660  m
1308 7f521ca403540f81ec99daec7da19c2788393880  m
1309 724f04ad10ed31db9d260f0fdf21ba48be8a95  ft
1310 7f521ca403540f81ec99daec7da19c2788393880  m

altitude_low_meters altitude_high_meters altitude_mean_meters
0      1950.00       2200.00       2075.00
1      1950.00       2200.00       2075.00
2      1600.00       1800.00       1700.00
3      1800.00       2200.00       2000.00
4      1950.00       2200.00       2075.00
...
1306  900.00        900.00       900.00
1307  350.00        350.00       350.00
1308  1100.00       1100.00      1100.00
1309  1417.32       1417.32      1417.32
1310  1400.00       1400.00      1400.00

```

[1311 rows x 44 columns]

so lets check the type of that.

`type(coffee_df.describe)`

`method`

it's a `bound method` or a function that *will* be applied to the DataFrame, but we didn't actually run the method. To see that it hasn't run, we can use an ipython[1] `%timeit`

```
%timeit  
coffee_df.describe
```

70.9 ns ± 0.0306 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)

```
%%timeit  
coffee_df.describe()
```

26.6 ms ± 285 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

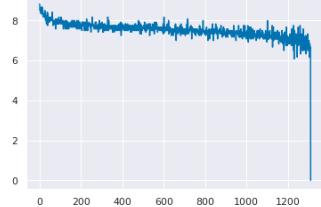
Note that without the `()` it runs much much faster, signaling that it did less finding the method, is less calculation than computing statistics on the data

7.3. Basic plots in pandas

Pandas gives us basic plots.

```
coffee_df['Flavor'].plot()
```

<AxesSubplot:>



Since we chose a series, it plotted that data as line vs the index.

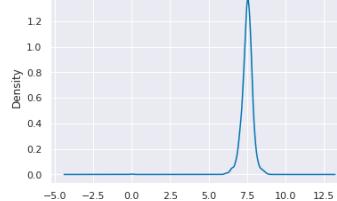
```
coffee_df.index
```

```
RangeIndex(start=0, stop=1311, step=1)
```

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

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

<AxesSubplot:ylabel='Density'>

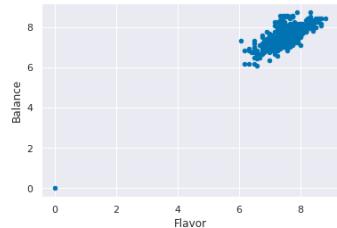


We can also plot two variables as a scatter plot, by specifying the `x`, `y` and `kind`

```
coffee_df.plot(x='Flavor',y='Balance', kind='scatter')
```

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.

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



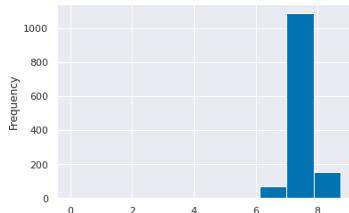
Let's Make a histogram plot of the Balance variable

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

Further reading

The [magic functions](#) can be defined for a cell or a line. `Timeit` is actually a standard python library that you can call on the command line or in a python script as well, but jupyter, by way of ipython gives us easy access to it with nice defaults.

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



Question from class

Can we plot two histograms with `coffee_df[['Balance', 'Flavor']].plot(kind='hist')`

```
:tags: ["raises-exception"]
coffee_df[['Balance', 'Flavor']].plot(kind='hist')
```

```
Input In [18]
:tags: ["raises-exception"]
 ^
SyntaxError: invalid syntax
```

Let's break down why that errors. When we append things to the left, python interprets them by passing the output of one step to the input of the next one. So `coffee_df[['Balance']].plot(kind='hist')` first made a series, then plotted it. In the above, we again got the series, which works

```
coffee_df[['Balance']].head(2)
```

```
0    8.42
1    8.42
Name: Balance, dtype: float64
```

But then, we tried to index it with 'Flavor', but we don't have that any more

```
coffee_df[['Balance', 'Flavor']]
```

```
KeyError Traceback (most recent call last)
Input In [20], in <cell line: 1>()
----> 1 coffee_df[['Balance', 'Flavor']]

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/series.py:958, in Series.__getitem__(self, key)
 955     return self._values[key]
 957 elif key_is_scalar:
--> 958     return self._get_value(key)
 960 if is_hashable(key):
 961     # Otherwise index.get_value will raise InvalidIndexError
 962     try:
 963         # For labels that don't resolve as scalars like tuples and
frozensets

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/series.py:1069, in Series._get_value(self, label, takeable)
 1066     return self._values[label]
 1068 # Similar to Index.get_value, but we do not fall back to positional
-> 1069 loc = self.index.get_loc(label)
 1070 return self.index._get_values_for_loc(self, loc, label)

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/indexes/range.py:389, in RangeIndex.get_loc(self, key,
method, tolerance)
 387     raise KeyError(key) from err
 388     self._check_indexing_error(key)
--> 389     raise KeyError(key)
 390 return super().get_loc(key, method=method, tolerance=tolerance)

KeyError: 'Flavor'
```

So we get a key error and we know this is the part of the line we have to change.

We need to index into the DataFrame and pick two columns at once. When we index, we can use the name of a variable as a string or a list. We can build this list on the fly and python executes from the inside out.

The outer `[]` index and the inner `[]` make a list

```
coffee_df[['Balance', 'Flavor']].head(2)
```

	Balance	Flavor
0	8.42	8.83
1	8.42	8.67

we could also build the list first, then index for readability

```
hist_vars = ['Balance', 'Flavor'].head(2)
coffee_df[hist_vars]
```

```
AttributeError Traceback (most recent call last)
Input In [22], in <cell line: 1>()
----> 1 hist_vars = ['Balance', 'Flavor'].head(2)
      2 coffee_df[hist_vars]

AttributeError: 'list' object has no attribute 'head'
```

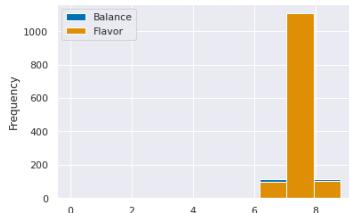
This gives us a data frame, which we can plot.

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

Note

Since I know these will be DataFrames, I'm appending the `head()` method to reduce the output for presentation.

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



We'll see ways to improve this on Friday.

7.4. Plotting in Python

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

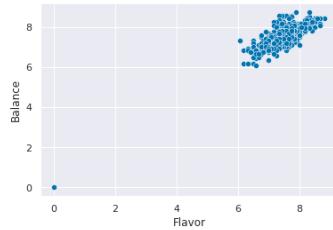
Pandas and seaborn use matplotlib under the hood.

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.

We can get that basic plot back.

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

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



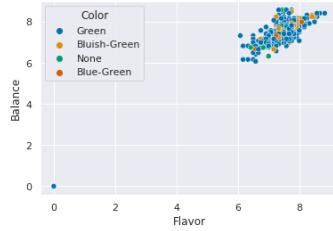
Think Ahead

Learning ggplot is a way to earn level 3 for visualize

But now we have more power to investigate more relationships in the data.

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

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

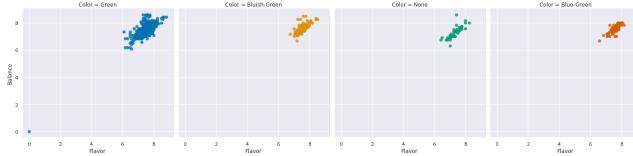


From this we can see that the color doesn't appear to be related to the flavor or balance scores, but that the flavor and balance are related.

We can also break this apart. `lmplot` is a higher level plotting function so it allows us to create grids of plots and by default also includes a regression line. We'll turn that off for now, with `,fit_reg=False`.

```
sns.lmplot(data=coffee_df,x='Flavor',y='Balance',hue='Color',  
           col='Color',fit_reg=False)
```

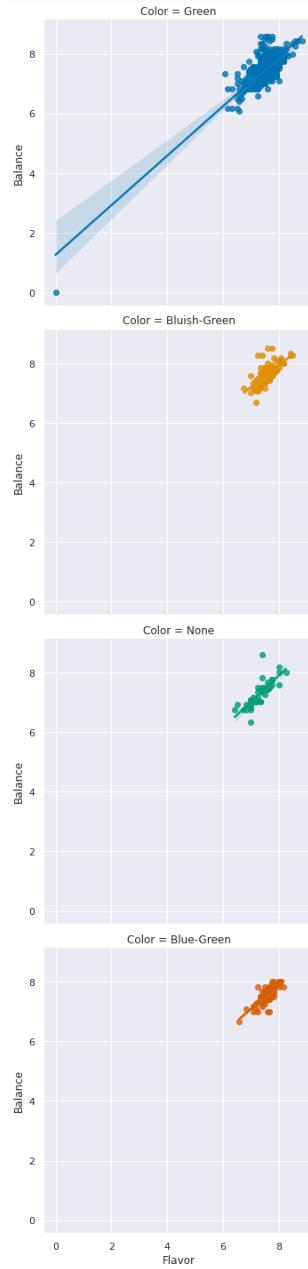
```
<seaborn.axisgrid.FacetGrid at 0x7fe27e5dc340>
```



`col` stands for column. We can also use `row`

```
sns.lmplot(data=coffee_df,x='Flavor',y='Balance',hue='Color',  
           row='Color')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe271595d30>
```



We can also use both together:

```
sns.lmplot(data=coffee_df,x='Flavor',y='Balance',hue='Color',
            row='Color',col='Variety')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe2711c7610>
```



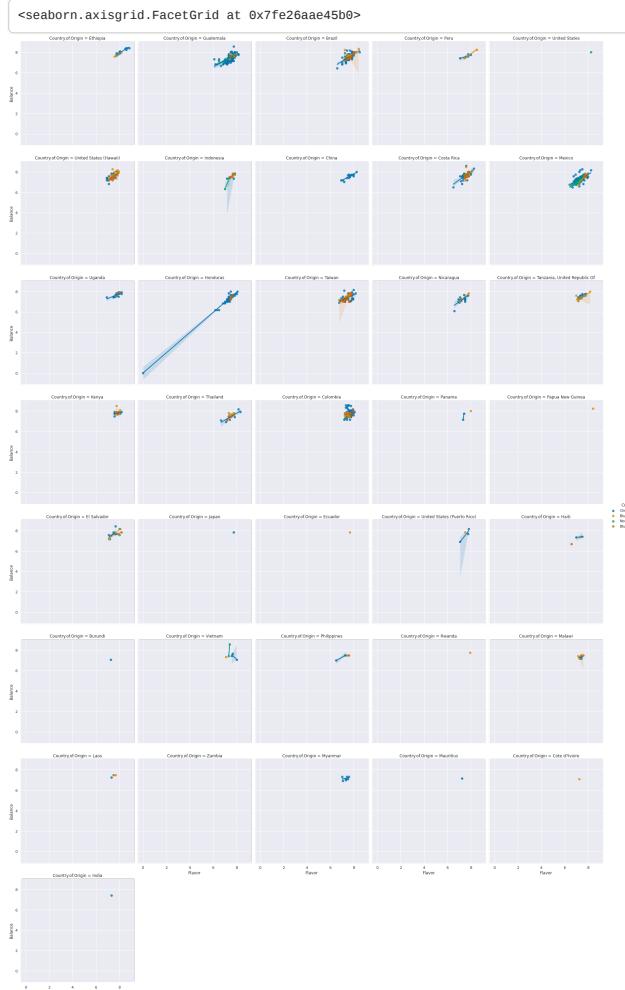
How could we choose which countries to select to make this not show the ones with very few points?

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

Mexico	236
Colombia	183
Guatemala	181
Brazil	132
Taiwan	75
United States (Hawaii)	73
Honduras	53
Costa Rica	51
Ethiopia	44
Tanzania, United Republic Of	40
Thailand	32
Uganda	26
Nicaragua	26
Kenya	25
El Salvador	21
Indonesia	20
China	16
Malawi	11
Peru	10
United States	8
Myanmar	8
Vietnam	7
Haiti	6
Philippines	5
Panama	4
United States (Puerto Rico)	4
Laos	3
Burundi	2
Ecuador	1
Rwanda	1
Japan	1
Zambia	1
Papua New Guinea	1
Mauritius	1
Cote d'Ivoire	1
India	1
Name: Country.of.Origin, dtype: int64	

Or we can focus on the countries, but wrap them.

```
sns.lmplot(data=coffee_df,x='Flavor',y='Balance',hue='Color',
           col='Country.of.Origin',col_wrap=5)
```



7.5. Questions after class

Ram Token Opportunity

add a question with a pull request; earn 1-2 ram tokens for submitting a question with the answer (with sources)

7.6. More practice

1. Plot the kde for the `Aftertaste`
2. How does `Total.Cup.Points` vary by `Certification.Body`
3. Are moisture and sweetness related? Does that relationship vary by Color?

[1] the kernel of python we're using

8. Exploratory Data Analysis

```
import pandas as pd
import seaborn as sns
sns.set_theme(palette= "colorblind")

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

Which of the following scores is distributed most similarly to Sweetness?

```
scores_of_interest = ['Flavor','Balance','Aroma','Body',
'Uniformity','Aftertaste','Sweetness']
```

First step is to subset the data:

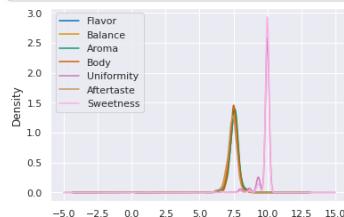
```
coffee_df[scores_of_interest].head(2)
```

	Flavor	Balance	Aroma	Body	Uniformity	Aftertaste	Sweetness
0	8.83	8.42	8.67	8.50	10.0	8.67	10.0
1	8.67	8.42	8.75	8.42	10.0	8.50	10.0

Then we produce a kde plot

```
coffee_df[scores_of_interest].plot(kind='kde')
```

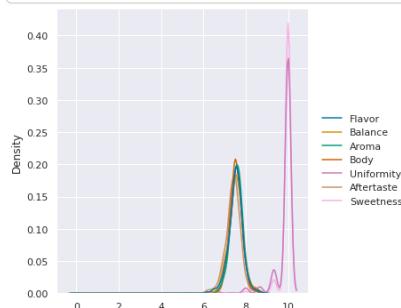
```
<AxesSubplot:ylabel='Density'>
```



We could also do it with seaborn

```
sns.displot(data=coffee_df[scores_of_interest],kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe8d6509d90>
```



If we forget the parameter `kind`, we get its default value, which is histogram

```
print('\n'.join(sns.displot.__doc__.split('\n')[5:10]))
```

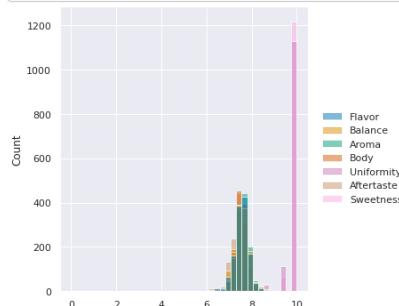
```
``kind`` parameter selects the approach to use:
- :func:`histplot` (with ``kind="hist"``; the default)
- :func:`kdeplot` (with ``kind="kde"``)
- :func:`ecdfplot` (with ``kind="ecdf"``; univariate-only)
```

```
sns.displot(data=coffee_df[scores_of_interest])
```

Note

If you show this excerpt, you'll see how I was able to select only a subset of the docstring to display in the notebook, programmatically. You're not required to know how to do it, but if you're curious, you can see.

```
<seaborn.axisgrid.FacetGrid at 0x7fe8e0b966d0>
```



8.1. Summarizing with two variables

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

On Monday, we saw how to see how many reviews there were per country, using `value_counts()` on the `Country.of.Origin` column.

```
coffee_df['Country.of.Origin'].value_counts().head(2)
```

```
Mexico    236
Colombia   183
Name: Country.of.Origin, dtype: int64
```

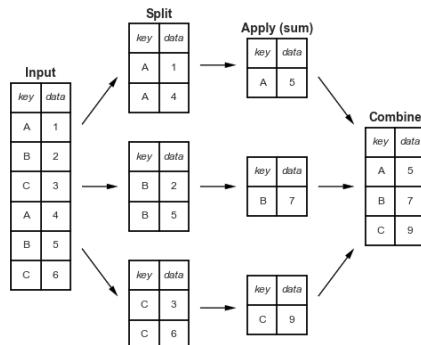
The data also has `Number.of.Bags` however. How can we check which has the most bags?

We can do this with groupby.

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

```
Country.of.Origin
Brazil           30534
Burundi          520
China             55
Colombia         41264
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
```

What just happened?



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

8.2. How does Groupby Work?

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

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

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fe90c546700>
```

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

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

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

| Note

I used this feature to build the separate view of the communication channels on this website. You can view that source using the github icon on that page.

| Note

I tried putting this dictionary into the dataframe for display purposes using the regular constructor and got an error, so I googled about making one from a dictionary to get the docs, which is how I learned about the `from_dict` method and its `orient` parameter which solved my problems.

Number.of.Bags.Sum	
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

is the same as what we did before

💡 Question from class

How can we sort it?

First, we'll make it a variable to keep the code legible, then we'll use `sort_values()`

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

```
Country.of.Origin
Mauritius           1
Ecuador             1
Cote d'Ivoire       2
Papua New Guinea    7
Vietnam              10
Myanmar              10
Zambia               13
India                20
Japan                20
China                55
United States (Puerto Rico) 71
Laos                 81
Rwanda               150
Philippines          259
United States         361
Haiti                390
Burundi              520
Panama               537
Malawi               557
United States (Hawaii) 833
Thailand             1310
Indonesia            1658
Taiwan               1914
Peru                 2336
Tanzania, United Republic Of 3760
Uganda               3868
Kenya                3971
El Salvador           4449
Nicaragua             6406
Costa Rica            10354
Ethiopia              11761
Honduras              13167
Mexico                24140
Brazil                30534
Guatemala             36868
Colombia              41204
Name: Number.of.Bags, dtype: int64
```

Which, by default uses ascending order, the method has an `ascending` parameter and its default value is `True`, so we can switch it to `False` to get descending

```
bag_total_df.sort_values(ascending=False,)
```

Country.of.Origin	Number.of.Bags
Colombia	41204
Guatemala	36868
Brazil	30534
Mexico	24140
Honduras	13167
Ethiopia	11761
Costa Rica	10354
Nicaragua	6496
El Salvador	4449
Kenya	3971
Uganda	3868
Tanzania, United Republic Of	3760
Peru	2336
Taiwan	1914
Indonesia	1658
Thailand	1310
United States (Hawaii)	833
Malawi	557
Panama	537
Burundi	529
Haiti	390
United States	361
Philippines	259
Rwanda	150
Laos	81
United States (Puerto Rico)	71
China	55
India	20
Japan	20
Zambia	13
Myanmar	10
Vietnam	10
Papua New Guinea	7
Cote d'Ivoire	2
Ecuador	1
Mauritius	1

Name: Number.of.Bags, dtype: int64

8.3. Customizing Data Summaries

We've looked at an overall summary with `describe` on all the variables, describe on one variable, individual statistics on all variables, and individual statistics on one variable. We can also build summaries of multiple variables with a custom subset of summary statistics, with `aggregate` or using its alias `agg`

```
country_grouped.agg({'Number.of.Bags':'sum',
                     'Balance':['mean', 'count'],})
```

Learning Tip

When a person reads a line of code, they have to use their working memory to hold the whole thing in order to make sense of it. Human working memory only holds 5-9 things at a time, that's why a phone number is 7 digits without the area code, (when people used to have to actually dial the digits, they also didn't need the area codes for short-distance calls, which were most of the calls).

How many concepts does a person have to hold in working memory at once to parse the second version?

If you are writing the code, and building it up, you hold the previous pieces in your memory differently than a person coming to it for the first time.

Country.of.Origin	Number.of.Bags		Balance
	sum	mean	count
Brazil	30534	7.531515	132
Burundi	520	7.415000	2
China	55	7.548125	16
Colombia	41204	7.708415	183
Costa Rica	10354	7.637255	51
Cote d'Ivoire	2	7.080000	1
Ecuador	1	7.830000	1
El Salvador	4449	7.711429	21
Ethiopia	11761	7.972273	44
Guatemala	36868	7.469890	181
Haiti	390	7.056667	6
Honduras	13167	7.163962	53
India	20	7.420000	1
Indonesia	1658	7.520000	20
Japan	20	7.830000	1
Kenya	3971	7.800400	25
Laos	81	7.416667	3
Malawi	557	7.371818	11
Mauritius	1	7.170000	1
Mexico	24140	7.328686	236
Myanmar	10	7.133750	8
Nicaragua	6406	7.278462	26
Panama	537	7.875000	4
Papua New Guinea	7	8.250000	1
Peru	2336	7.666000	10
Philippines	259	7.400000	5
Rwanda	150	7.750000	1
Taiwan	1914	7.426000	75
Tanzania, United Republic Of	3760	7.469750	40
Thailand	1310	7.524063	32
Uganda	3868	7.660769	26
United States	361	7.947500	8
United States (Hawaii)	833	7.644110	73
United States (Puerto Rico)	71	7.647500	4
Vietnam	10	7.547143	7
Zambia	13	7.420000	1

We could also string this together, but splitting into interim variables makes code more readable. Shorter lines are easier to read (and sometimes auto-enforced on projects). Also, by giving a good variable name to the interim states we get more description, which can help a human better read it.

```
{ coffee_df.groupby('Country.of.Origin').agg({'Number.of.Bags':'sum', 'Balance': ['mean', 'count']}) }
```

Country.of.Origin	Number.of.Bags	Balance	count
	sum	mean	
Brazil	30534	7.531515	132
Burundi	520	7.415000	2
China	55	7.548125	16
Colombia	41204	7.708415	183
Costa Rica	10354	7.637255	51
Cote d'Ivoire	2	7.080000	1
Ecuador	1	7.830000	1
El Salvador	4449	7.711429	21
Ethiopia	11761	7.972273	44
Guatemala	36868	7.469890	181
Haiti	390	7.056667	6
Honduras	13167	7.163962	53
India	20	7.420000	1
Indonesia	1658	7.520000	20
Japan	20	7.830000	1
Kenya	3971	7.800400	25
Laos	81	7.416667	3
Malawi	557	7.371818	11
Mauritius	1	7.170000	1
Mexico	24140	7.328686	236
Myanmar	10	7.133750	8
Nicaragua	6406	7.278462	26
Panama	537	7.875000	4
Papua New Guinea	7	8.250000	1
Peru	2336	7.666000	10
Philippines	259	7.400000	5
Rwanda	150	7.750000	1
Taiwan	1914	7.426000	75
Tanzania, United Republic Of	3760	7.469750	40
Thailand	1310	7.524063	32
Uganda	3868	7.660769	26
United States	361	7.947500	8
United States (Hawaii)	833	7.644110	73
United States (Puerto Rico)	71	7.647500	4
Vietnam	10	7.547143	7
Zambia	13	7.420000	1

Question from Class

What does the `count` do? or how can we figure it out?

In this case, let's compare `count` to `shape` we know that `shape` returns the number of rows and columns. Also `shape` is an attribute, not a method (so no `()` for `shape`). However, there's a more important difference.

```
coffee_df['Balance', 'Farm.Name', 'Lot.Number'].shape
```

```

-----
KeyError Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/indexes/base.py:3621, in Index.get_loc(self, key, method,
tolerance)
3620 try:
-> 3621     return self._engine.get_loc(casted_key)
3622 except KeyError as err:
3623     raise KeyError(key) from err

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/_libs/index.pxi:136, in pandas._libs.index.IndexEngine.get_loc()

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/_libs/index.pxi:163, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:5198, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:5206, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: ('Balance', 'Farm.Name', 'Lot.Number')

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

KeyError Traceback (most recent call last)
Input In [18], in <cell line: 1>()
----> 1 coffee_df['Balance','Farm.Name','Lot.Number'].shape

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/frame.py:3505, in DataFrame.__getitem__(self, key)
3503 if self.columns.nlevels > 1:
3504     return self._getitem_multilevel(key)
-> 3505 indexer = self.columns.get_loc(key)
3506 if is_integer(indexer):
3507     indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/indexes/base.py:3623, in Index.get_loc(self, key, method,
tolerance)
3621     return self._engine.get_loc(casted_key)
3622 except KeyError as err:
-> 3623     raise KeyError(key) from err
3624 except TypeError:
3625     # If we have a listlike key, _check_indexing_error will raise
3626     # InvalidIndexError. Otherwise we fall through and re-raise
3627     # the TypeError.
3628     self._check_indexing_error(key)

KeyError: ('Balance', 'Farm.Name', 'Lot.Number')

```

```
[ coffee_df['Balance','Farm.Name','Lot.Number'].count() ]
```

```

-----
KeyError Traceback (most recent call last)
File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/indexes/base.py:3621, in Index.get_loc(self, key, method,
tolerance)
3620 try:
-> 3621     return self._engine.get_loc(casted_key)
3622 except KeyError as err:
3623     raise KeyError(key) from err

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/_libs/index.pxi:136, in pandas._libs.index.IndexEngine.get_loc()

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/_libs/index.pxi:163, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:5198, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:5206, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: ('Balance', 'Farm.Name', 'Lot.Number')

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

KeyError Traceback (most recent call last)
Input In [19], in <cell line: 1>()
----> 1 coffee_df['Balance','Farm.Name','Lot.Number'].count()

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/frame.py:3505, in DataFrame.__getitem__(self, key)
3503 if self.columns.nlevels > 1:
3504     return self._getitem_multilevel(key)
-> 3505 indexer = self.columns.get_loc(key)
3506 if is_integer(indexer):
3507     indexer = [indexer]

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/indexes/base.py:3623, in Index.get_loc(self, key, method,
tolerance)
3621     return self._engine.get_loc(casted_key)
3622 except KeyError as err:
-> 3623     raise KeyError(key) from err
3624 except TypeError:
3625     # If we have a listlike key, _check_indexing_error will raise
3626     # InvalidIndexError. Otherwise we fall through and re-raise
3627     # the TypeError.
3628     self._check_indexing_error(key)

KeyError: ('Balance', 'Farm.Name', 'Lot.Number')

```

Here we get different number for `Balance` and `Farm Name`. The shape tells us how many rows there are, while count tells us how many are not null.

We can verify visually that some are null with:

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

Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	...	Color	Category.Two.Defects	Expiration
0	1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural development plc	1950-2200	...	Green	0 April 3rd, 201
1	2	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural development plc	1950-2200	...	Green	1 April 3rd, 201
2	3	Arabica	grounds for health admin	Guatemala	san marcos barrancas "san cristobal cuch	NaN	NaN	NaN	NaN	1600 - 1800 m	...	NaN	0 May 31st, 201
3	4	Arabica	yidnekachew dabessa	Ethiopia	yidnekachew dabessa coffee plantation	NaN	wolensu	NaN	yidnekachew debessa coffee plantation	1800-2200	...	Green	2 Marc 25th, 201
4	5	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural development plc	1950-2200	...	Green	2 April 3rd, 201

5 rows × 44 columns

Correction

This response is slightly corrected from what I said in class, because in the balance column, it does match, but in general it doesn't. `count` on grouped will only be the same as `value_count` when `count` is applied to a column that does not have any missing values where the grouping variable has a value.

On the balance column alone in class, we checked that the grouped count matched the country value counts.

```
country_grouped[['Balance', 'Farm.Name', 'Lot.Number']].count().sort_values(by='Balance')
    ascending=False)
```

```
Input In [21]
ascending=False)
^
SyntaxError: invalid syntax
```

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

Mexico	236
Colombia	183
Guatemala	181
Brazil	132
Taiwan	75
United States (Hawaii)	73
Honduras	53
Costa Rica	51
Ethiopia	44
Tanzania, United Republic Of	40
Thailand	32
Uganda	26
Nicaragua	26
Kenya	25
El Salvador	21
Indonesia	20
China	16
Malawi	11
Peru	10
United States	8
Myanmar	8
Vietnam	7
Haiti	6
Philippines	5
Panama	4
United States (Puerto Rico)	4
Laos	3
Burundi	2
Ecuador	1
Rwanda	1
Japan	1
Zambia	1
Papua New Guinea	1
Mauritius	1
Cote d'Ivoire	1
India	1
Name: Country.of.Origin, dtype: int64	

In class, they were the same, because `Balance` doesn't have missing values. `Farm.Name` and `Lot.Number` have a lot of missing values, so they're different numbers.

Another function we could use when we first examine a dataset is `info` this tells us about the NaN values up front.

```
coffee_df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1311 entries, 0 to 1310
Data columns (total 44 columns):
 #   Column           Non-Null Count Dtype  
--- 
 0   Unnamed: 0        1311 non-null   int64  
 1   Species          1311 non-null   object  
 2   Owner            1304 non-null   object  
 3   Country.of.Origin 1310 non-null   object  
 4   Farm.Name        955 non-null    object  
 5   Lot.Number       270 non-null    object  
 6   Mill             1001 non-null   object  
 7   ICO.Number       1165 non-null   object  
 8   Company          1102 non-null   object  
 9   Altitude         1088 non-null   object  
 10  Region           1254 non-null   object  
 11  Producer         1081 non-null   object  
 12  Number.of.Bags  1311 non-null   int64  
 13  Bag.Weight      1311 non-null   object  
 14  In.Country.Partner 1311 non-null   object  
 15  Harvest.Year    1264 non-null   object  
 16  Grading.Date    1311 non-null   object  
 17  Owner.1          1304 non-null   object  
 18  Variety          1110 non-null   object  
 19  Processing.Method 1159 non-null   object  
 20  Aroma            1311 non-null   float64 
 21  Flavor           1311 non-null   float64 
 22  Aftertaste       1311 non-null   float64 
 23  Acidity          1311 non-null   float64 
 24  Body              1311 non-null   float64 
 25  Balance           1311 non-null   float64 
 26  Uniformity       1311 non-null   float64 
 27  Clean.Cup        1311 non-null   float64 
 28  Sweetness         1311 non-null   float64 
 29  Cupper.Points   1311 non-null   float64 
 30  Total.Cup.Points 1311 non-null   float64 
 31  Moisture          1311 non-null   float64 
 32  Category.One.Defects 1311 non-null   int64  
 33  Quakers          1310 non-null   float64 
 34  Color             1095 non-null   object  
 35  Category.Two.Defects 1311 non-null   int64  
 36  Expiration        1311 non-null   object  
 37  Certification.Body 1311 non-null   object  
 38  Certification.Address 1311 non-null   object  
 39  Certification.Contact 1311 non-null   object  
 40  unit_of_measurement 1311 non-null   object  
 41  altitude_low_meters 1084 non-null   float64 
 42  altitude_high_meters 1084 non-null   float64 
 43  altitude_mean_meters 1084 non-null   float64 
dtypes: float64(16), int64(4), object(24)
memory usage: 450.8+ KB

```

8.4. Using summaries for visualization

Important

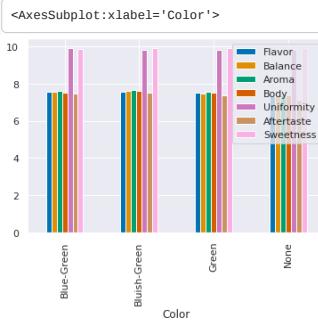
This section is an extension, that we didn't get to in class, but might help in your assignment

For example, we can group by color and take the mean of each of the scores from the beginning of class and then use a bar chart.

```

color_grouped = coffee_df.groupby('Color')[scores_of_interest].mean()
color_grouped.plot(kind='bar')

```



8.5. Questions after class

Ram Token Opportunity

add a question with a pull request; earn 1-2 ram tokens for submitting a question with the answer (with sources)

8.6. More Practice

- Make a table that totals number of bags and mean and count of scored for each of the variables in the `scores_of_interest` list.
- Make a bar chart of the mean score for each variable `scores_of_interest` grouped by country.

9. Reshaping Data

Today, we'll begin reshaping data. We'll cover:

- filtering
- applying a function to all rows
- what is tidy data
- reshaping data into tidy data

First some setup:

```

import pandas as pd
import seaborn as sns

sns.set_theme(font_scale=2, palette='colorblind')
arabica_data_url = 'https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/arabica_data_cleaned.csv'

```

9.1. Cleaning Data

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
- changing the representation of information

9.2. 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']

```

```

df_list = [pd.read_csv(url_base + file, na_values= '') for file in datasets]

```

```

df_list[0]

```

	name	treatmentsa	treatmentsb
0	John Smith	-	2
1	Jane Doe	16	11
2	Mary Johnson	3	1

```

df_list[1]

```

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

```

df_list[2]

```

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

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

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

This works differently for these three versions.

```

df_list[0].mean()

```

```

/tmp/ipykernel_2392/2274987639.py:1: FutureWarning: Dropping of nuisance columns
in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future
version this will raise TypeError. Select only valid columns before calling the
reduction.
df_list[0].mean()

```

```

treatmentsa    -54.333333
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

```

df_list[0].mean(axis=1)

```

```

/tmp/ipykernel_2392/1371725361.py:1: FutureWarning: Dropping of nuisance columns
in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future
version this will raise TypeError. Select only valid columns before calling the
reduction.
df_list[0].mean(axis=1)

```

```

0      2.0
1     11.0
2      1.0
dtype: float64

```

Tip

I used `set_theme` to change both the font size and the color palette. Seaborn has a [detailed guide](#) for choosing colors. The `colorblind` palette uses colors that are distinguishable under most common forms of color blindness.

and this is not well labeled.

Let's try the next one.

```
{ df_list[1].mean() }
```

```
/tmp/ipykernel_2392/1105758803.py:1: FutureWarning: Dropping of nuisance columns
in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future
version this will raise TypeError. Select only valid columns before calling the
reduction.
df_list[1].mean()
```

```
John Smith      -1.0
Jane Doe       13.5
Mary Johnson    2.0
dtype: float64
```

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

```
{ df_list[1].mean(axis=1) }
```

```
/tmp/ipykernel_2392/1112167831.py:1: FutureWarning: Dropping of nuisance columns
in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future
version this will raise TypeError. Select only valid columns before calling the
reduction.
df_list[1].mean(axis=1)
```

```
0    9.5
1    6.0
dtype: float64
```

For the third one, however, we can use groupby

```
{ df_list[2].groupby('person').mean() }
```

```
/tmp/ipykernel_2392/906930014.py:1: FutureWarning: Dropping invalid columns in
DataFrameGroupBy.mean is deprecated. In a future version, a TypeError will be
raised. Before calling .mean, select only columns which should be valid for the
function.
df_list[2].groupby('person').mean()
```

result
person
Jane Doe 805.5
John Smith -1.0
Mary Johnson 15.5

```
{ df_list[2].groupby('treatment').mean() }
```

```
/tmp/ipykernel_2392/2480310521.py:1: FutureWarning: Dropping invalid columns in
DataFrameGroupBy.mean is deprecated. In a future version, a TypeError will be
raised. Before calling .mean, select only columns which should be valid for the
function.
df_list[2].groupby('treatment').mean()
```

result
treatment
a -54.333333
b 703.666667

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

9.3. 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.

```
{ treat_df = df_list[0] }
```

Let's look at it again, so we can see

```
{ treat_df.head() }
```

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

Correction

I fixed the three data files so the spaces can be removed. You will need to

```
{ treat_df.melt(value_vars = ['treatmenta','treatmentb'],
                 id_vars = ['name'],
                 value_name = 'result', var_name = 'treatment' ) }
```

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

```

tidy_treat_df = treat_df.melt(value_vars = ['treatmenta','treatmentb'],
                             id_vars = ['name'],
                             value_name = 'result', var_name = 'treatment' )

tidy_treat_df.groupby('name').mean()

/tmp/ipykernel_2392/3450628511.py:1: FutureWarning: Dropping invalid columns in
DataFrameGroupBy.mean is deprecated. In a future version, a TypeError will be
raised. Before calling .mean, select only columns which should be valid for the
function.
    tidy_treat_df.groupby('name').mean()

```

name

Jane Doe

John Smith

Mary Johnson

9.4. Filtering Data by a column

Let's go back to the coffee dataset

```

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

```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	Region	...	Color	Category.Two.Defects	Expiration
1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	guji-hambela	...	Green	0	April 3rd, 2016
2	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	guji-hambela	...	Green	1	April 3rd, 2016
3	Arabica	grounds for health admin	Guatemala	san marcos barrancas "san cristobal cuch	NaN	NaN	NaN	NaN	1600 - 1800 m	NaN	...	NaN	0	May 31st, 2011
4	Arabica	yidnekachew dabessa	Ethiopia	yidnekachew dabessa coffee plantation	NaN	wolensu	NaN	yidnekachew debessa coffee plantation	1800-2200	oromia	...	Green	2	March 25th, 2016
5	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	guji-hambela	...	Green	2	April 3rd, 2016

5 rows × 43 columns

Recall on Friday we computed the total number of bags per country.

```

# compute total bags per country
bag_total_df = coffee_df.groupby('Country.of.Origin')['Number.of.Bags'].sum()

```

We can subset this to get only the countries with over 15000 using a boolean mask:

```

bag_total_df[bag_total_df>15000]

```

Country.of.Origin	
Brazil	30534
Colombia	41294
Guatemala	36868
Mexico	24140
Name: Number.of.Bags	dtype: int64

what we put in the [] has to be the same length and each element has to be boolean

```

len(bag_total_df>15000)

```

36

```

mask = bag_total_df>15000
type(mask[0])

```

numpy.bool_

9.5. Augmenting a dataset

We want the names of the countries as a list, so we extract the index of that series and then cast it to a list.

```

high_prod_countries = list(bag_total_df[bag_total_df>15000].index)

```

Next we want to be able to check if a country is in this list, so we'll make a lambda that can do that

```
high_prod = lambda c: c in high_prod_countries
```

Recall, the `lambda` keyword makes a function

```
type(high_prod)
```

```
function
```

We can test it

```
high_prod('Mexico'), high_prod('Ethiopia')
```

```
(True, False)
```

Now, we can apply that lambda function to each country in our whole coffee data frame, and save that to a new DataFrame.

```
coffee_df['high_production'] = coffee_df['Country.of.Origin'].apply(high_prod)
```

```
coffee_df.head()
```

	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	Region	...	Category.Two.Defects	Expiration	Certific
1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	guji-hambela	...	0	April 3rd, 2016	Dev
2	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	guji-hambela	...	1	April 3rd, 2016	Dev
3	Arabica	grounds for health admin	Guatemala	san marcos barrancas "san cristobal cuch	NaN	NaN	NaN	NaN	1600 - 1800 m	NaN	...	0	May 31st, 2011	Spe
4	Arabica	yidnekachew dabessa	Ethiopia	yidnekachew dabessa coffee plantation	NaN	wolensu	NaN	yidnekachew debessa coffee plantation	1800-2200	oromia	...	2	March 25th, 2016	Dev
5	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	guji-hambela	...	2	April 3rd, 2016	Dev

5 rows × 44 columns

Finally, we can filter the whole data frame using that new column.

```
high_prod_coffee_df = coffee_df[coffee_df['high_production']]
```

❓ Question from class

How can we get the ones not on that list?

```
low_prod_coffee_df = coffee_df[coffee_df['high_production']==False]
```

👉 Try it Yourself

Replace the FIXMEs in the excerpt below to reshape the data to have a value column with the value of the score and a Score column that indicates which score is in that . Keep the color and country as values

```
scores_of_interest = ['Balance', 'Aroma', 'Body', 'Aftertaste']
attrs_of_interest = ['Country.of.Origin', 'Color']
high_prod_coffee_df_melted = high_prod_coffee_df.melt(
    id_vars = FIXME,
    value_vars = FIXME,
    value_name = 'Value',
    var_name = 'Score')
```

so that it looks like the following

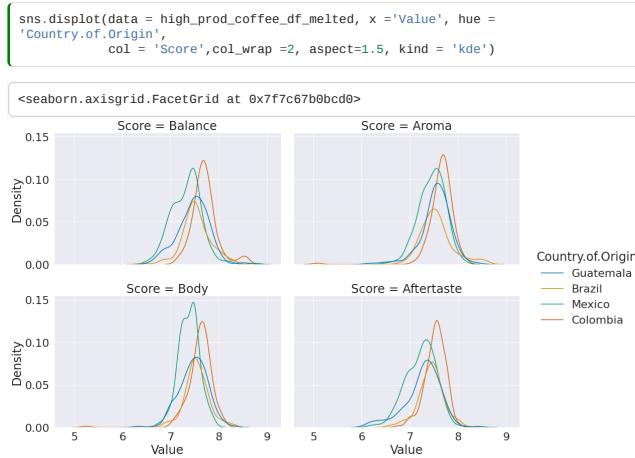
```
high_prod_coffee_df_melted = high_prod_coffee_df.melt(
    id_vars = ["Country.of.Origin", "Color"],
    value_vars = ['Balance', 'Aroma', 'Body', 'Aftertaste'],
    value_name = 'Value',
    var_name = 'Score')

high_prod_coffee_df_melted.head()
```

	Country.of.Origin	Color	Score	Value
0	Guatemala	NaN	Balance	8.42
1	Brazil	Bluish-Green	Balance	8.33
2	Mexico	Green	Balance	8.17
3	Brazil	Green	Balance	8.00
4	Brazil	Green	Balance	8.00

👉 Try it Yourself

Plot the distribution of each score on a separate subplot and use a different color for each country. Use a kde for the distributions.



10. More Reshaping

Continuing from Friday.

```

import pandas as pd
import seaborn as sns

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

coffee_df = pd.read_csv(arabica_data_url)

# compute ____ per ____
bag_total_df = coffee_df.groupby('Country.of.Origin')['Number.of.Bags'].sum()

# subset the summary Series for countries with over 15000 total and store as a list
high_prod_countries = list(bag_total_df[bag_total_df>15000].index)

# a lambda function that checks if a string c is one of the # countries in high_prod_countries
high_prod = lambda c: c in high_prod_countries

# add a column that indicates that the country is a high producer
coffee_df['high_production'] = coffee_df['Country.of.Origin'].apply(high_prod)

# filter based on production level threshold
high_prod_coffee_df = coffee_df[coffee_df['high_production']]

```

What happened when we filtered the data?

```

coffee_df.shape, high_prod_coffee_df.shape

```

```
((1311, 45), (732, 45))
```

We have many fewer rows.

Now that we've filtered the data. Let's practice reshaping data to by Tidy again.

```

# replace the FIXMES
scores_of_interest = ['Balance','Aroma','Body','Aftertaste']
attrs_of_interest = ['Country.of.Origin','Color']
high_prod_coffee_df_melted = high_prod_coffee_df.melt(
    id_vars = attrs_of_interest,
    value_vars = scores_of_interest,
    var_name = 'Score')

```

What happened?

```

high_prod_coffee_df_melted.shape

```

```
(2928, 4)
```

Now the shape is 4 times as long (because the length of the list we passed to value_vars is 4). And it has 4 columns: the length of the list we passed to id_vars + 2 (variable, value)

```

len(scores_of_interest)

```

```
4
```

```

len(scores_of_interest)*len(high_prod_coffee_df)

```

```
2928
```

We can see the column names and what they have in them here:

```

high_prod_coffee_df_melted.head()

```

	Country.of.Origin	Color	Score	value
0	Guatemala	NaN	Balance	8.42
1	Brazil	Bluish-Green	Balance	8.33
2	Mexico	Green	Balance	8.17
3	Brazil	Green	Balance	8.00
4	Brazil	Green	Balance	8.00

Note that we passed a value to `var_name` to make that column named "Score". We could also not pass that

```
high_prod_coffee_df.melt(
    id_vars = attrs_of_interest,
    value_vars = scores_of_interest)
```

	Country.of.Origin	Color	variable	value
0	Guatemala	NaN	Balance	8.42
1	Brazil	Bluish-Green	Balance	8.33
2	Mexico	Green	Balance	8.17
3	Brazil	Green	Balance	8.00
4	Brazil	Green	Balance	8.00
...
2923	Mexico	Green	Aftertaste	6.42
2924	Mexico	Green	Aftertaste	6.83
2925	Brazil	Green	Aftertaste	6.83
2926	Mexico	None	Aftertaste	6.25
2927	Guatemala	Green	Aftertaste	6.67

2928 rows × 4 columns

then we have `variable` and `value` as column names.

Try it yourself

How could you rename the `value` column?

The head has only 'Balance' in the 'Score' column, we could use `sample` to pick a random subset of the rows instead to see different values.

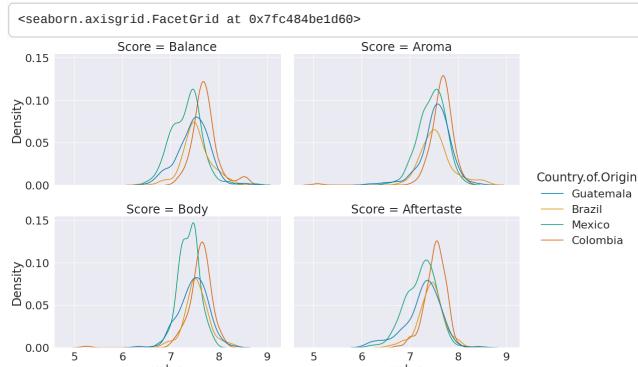
```
high_prod_coffee_df_melted.sample(5)
```

	Country.of.Origin	Color	Score	value
1977	Brazil	Green	Body	7.67
903	Colombia	Green	Aroma	8.00
1651	Colombia	Green	Body	7.75
1856	Brazil	Green	Body	7.50
621	Guatemala	Green	Balance	7.33

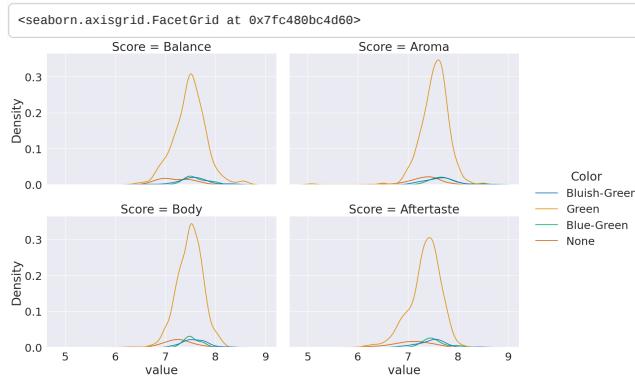
What does this let us do?

One thing is it makes plots easier, because seaborn is organized around tidy data.

```
sns.displot(data= high_prod_coffee_df_melted,
            x='value',hue='Country.of.Origin',
            col = 'Score', col_wrap=2, kind='kde',aspect =1.5)
```

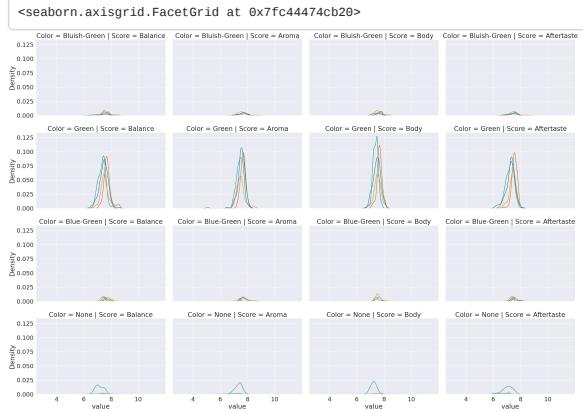


```
sns.displot(data= high_prod_coffee_df_melted,
            x='value',hue='Color',
            col = 'Score', col_wrap=2, kind='kde',aspect =1.5)
```



```
sns.displot(data= high_prod_coffee_df_melted,
            x='value',hue='Country.of-Origin',
            col = 'Score', row='Color', kind='kde',aspect =1.5)
```

```
/opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/seaborn/distributions.py:316: UserWarning: Dataset has 0 variance;
skipping density estimate. Pass 'warn_singular=False' to disable this warning.
warnings.warn(msg, UserWarning)
/opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/seaborn/distributions.py:316: UserWarning: Dataset has 0 variance;
skipping density estimate. Pass 'warn_singular=False' to disable this warning.
warnings.warn(msg, UserWarning)
/opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/seaborn/distributions.py:316: UserWarning: Dataset has 0 variance;
skipping density estimate. Pass 'warn_singular=False' to disable this warning.
warnings.warn(msg, UserWarning)
/opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/seaborn/distributions.py:316: UserWarning: Dataset has 0 variance;
skipping density estimate. Pass 'warn_singular=False' to disable this warning.
warnings.warn(msg, UserWarning)
```



10.1. Unpacking Jsons

```
rhodyprog4ds_gh_events_url = 'https://api.github.com/orgs/rhodyprog4ds/events'
```

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

		id	type	actor	repo	payload	public	created_at	org
0	23017297788	10656079	PushEvent	{'id': 'brownsarahm', 'login': 'brownsarahm', 'dis...	{'id': '400283911', 'name': 'rhodyprog4ds/BrownF...', 'type': 'Repository'}	{'push_id': '10511442158', 'size': 1, 'distinct_ids': 1}	True	2022-07-21 22:52:08+00:00	{'id': '69595187', 'login': 'rhodyprog4ds', 'name': 'rhodyprog4ds'}
1	23017256816	41898282	PushEvent	{'id': 'github-actions[bot]', 'login': 'github-actions[bot]'}	{'id': '400283911', 'name': 'rhodyprog4ds/BrownF...', 'type': 'Repository'}	{'push_id': '10511421441', 'size': 1, 'distinct_ids': 1}	True	2022-07-21 22:48:25+00:00	{'id': '69595187', 'login': 'rhodyprog4ds', 'name': 'rhodyprog4ds'}
2	23017147996	10656079	PushEvent	{'id': 'brownsarahm', 'login': 'brownsarahm', 'dis...	{'id': '400283911', 'name': 'rhodyprog4ds/BrownF...', 'type': 'Repository'}	{'push_id': '10511370784', 'size': 1, 'distinct_ids': 1}	True	2022-07-21 22:40:28+00:00	{'id': '69595187', 'login': 'rhodyprog4ds', 'name': 'rhodyprog4ds'}
3	23017000728	10656079	PushEvent	{'id': 'brownsarahm', 'login': 'brownsarahm', 'dis...	{'id': '400283911', 'name': 'rhodyprog4ds/BrownF...', 'type': 'Repository'}	{'push_id': '10511300617', 'size': 1, 'distinct_ids': 1}	True	2022-07-21 22:28:48+00:00	{'id': '69595187', 'login': 'rhodyprog4ds', 'name': 'rhodyprog4ds'}
4	23016409837	10656079	PushEvent	{'id': 'brownsarahm', 'login': 'brownsarahm', 'dis...	{'id': '400283911', 'name': 'rhodyprog4ds/BrownF...', 'type': 'Repository'}	{'push_id': '10511004204', 'size': 1, 'distinct_ids': 1}	True	2022-07-21 21:41:28+00:00	{'id': '69595187', 'login': 'rhodyprog4ds', 'name': 'rhodyprog4ds'}

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

```
type(course_gh_df['actor'])
```

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

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

```
{ 5  
5  
type(5)  
int  
str(5)  
'5'
```

To unpack one column we can cast each element of the column to a series and then stack them back together.

First, let's look at one row of one column

```
{ course_gh_df['actor'][0]  
  
{'id': 10656079,  
 'login': 'brownsarahm',  
 'display_login': 'brownsarahm',  
 'gravatar_id': '',  
 'url': 'https://api.github.com/users/brownsarahm',  
 'avatar_url': 'https://avatars.githubusercontent.com/u/10656079?'}
```

Now let's cast it to a Series

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

What we want is to do this over and over and stack them.

The `apply` method does this for us, in one compact step.

```
{ course_gh_df['actor'].apply(pd.Series)  
  
   id      login display_login gravatar_id          url      avatar_url  
0 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?  
1 41898282    github- actions[bot]  github-actions https://api.github.com/users/github- actions[bot] https://avatars.githubusercontent.com/u/41898282?  
2 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?  
3 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?  
4 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?  
5 41898282    github- actions[bot]  github-actions https://api.github.com/users/github- actions[bot] https://avatars.githubusercontent.com/u/41898282?  
6 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?  
7 41898282    github- actions[bot]  github-actions https://api.github.com/users/github- actions[bot] https://avatars.githubusercontent.com/u/41898282?  
8 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?  
9 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?  
10 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?  
11 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?  
12 10656079 brownsarahm brownsarahm https://api.github.com/users/brownsarahm https://avatars.githubusercontent.com/u/10656079?
```

How can we do this for all of the columns and put them back together after?

First, let's make a list of the columns we need to convert.

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

When we use `.apply(pd.Series)` we get a a DataFrame.

```
{ type(course_gh_df['actor'].apply(pd.Series))  
  
pandas.core.frame.DataFrame
```

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

to illustrate, it's nice to make small DataFrames.

```
{ df1 = pd.DataFrame([[1,2,3],[3,4,7]], columns = ['A', 'B', 't'])  
df2 = pd.DataFrame([[10,20,30],[30,40,70]], columns = ['AA', 'BB', 't'])  
df1
```

```
A B t
0 1 2 3
1 3 4 7
```

```
[df2]
```

```
AA BB t
0 10 20 30
1 30 40 70
```

If we use concat with the default settings, it stacks them vertically and aligns any columns that have the same name.

```
[pd.concat([df1,df2])]
```

	A	B	t	AA	BB
0	1.0	2.0	3	NaN	NaN
1	3.0	4.0	7	NaN	NaN
0	NaN	NaN	30	10.0	20.0
1	NaN	NaN	70	30.0	40.0

So, since the original DataFrames were both 2 rows with 3 columns each, with one column name appearing in both, we end up with a new DataFrame with shape (4,5) and it fills with `NaN` in the top right and the bottom left.

```
[pd.concat([df1,df2]).shape]
```

```
(4, 5)
```

We can use the `axis` parameter to tell it how to combine them. The default is `axis=0`, but `axis=1` will combine along rows.

```
[pd.concat([df1,df2], axis =1)]
```

	A	B	t	AA	BB	t
0	1	2	3	10	20	30
1	3	4	7	30	40	70

So now we get no `NaN` values, because both DataFrames have the same number of rows and the same index.

```
[df1.index == df2.index]
```

```
array([ True,  True])
```

and we have a total of 6 columns and 2 rows.

```
[pd.concat([df1,df2], axis =1).shape]
```

```
(2, 6)
```

Back to our gh data, we want to make a list of DataFrames where each DataFrame corresponds to one of the columns in the original DataFrame, but unpacked and then stack them horizontally (`axis=1`) because each DataFrame in the list is based on the same original DataFrame, they again have the same index.

```
[pd.concat([course_gh_df[cur_col].apply(pd.Series) for cur_col in js_cols], axis=1)]
```

	id	login	display_login	gravatar_id	url	avatar_url	id
0	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
1	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot]	https://avatars.githubusercontent.com/u/41898282?	400283911
2	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
3	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
4	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
5	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot]	https://avatars.githubusercontent.com/u/41898282?	337917892
6	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	337917892
7	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot]	https://avatars.githubusercontent.com/u/41898282?	501809605
8	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	501809605
9	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	501809605
10	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	404401191
11	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	404401191
12	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	404401191

13 rows × 25 columns

Try it Yourself

examine the list of DataFrames to see what structure they share and do not share

In this case we get the same 30 rows, because that's what the API gave us and turned our 4 columns from `js_cols` into 26 columns.

```
pd.concat([course_gh_df[cur_col].apply(pd.Series) for cur_col in js_cols],  
          axis=1).shape
```

(13, 25)

If we had used the default, we'd end up with 120 rows (30*4) and we have only 19 columns, because there are subfield names that are shared across the original columns. (eg most have an `id`)

```
pd.concat([course_gh_df[cur_col].apply(pd.Series) for cur_col in js_cols],  
          axis=0).shape
```

(52, 18)

Try it yourself

How could you anticipate how many are shared?

we might want to rename the new columns so that they have the original column name prepended to the new name. This will help us distinguish between the different `id` columns

pandas has a `rename` method for this.

and this is another job for lambdas.

```
pd.concat([course_gh_df[cur_col].apply(pd.Series).rename(columns = lambda c:  
cur_col + ' ' + c)  
          for cur_col in js_cols],  
          axis=1)
```

	actor_id	actor_login	actor_display_login	actor_gravatar_id	actor_url	actor_avatar_url	repo_id
0	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
1	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot]	https://avatars.githubusercontent.com/u/41898282?	400283911
2	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
3	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
4	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
5	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot]	https://avatars.githubusercontent.com/u/41898282?	337917892 rho
6	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	337917892 rho
7	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot]	https://avatars.githubusercontent.com/u/41898282?	501809605
8	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	501809605
9	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	501809605
10	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	404401191
11	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	404401191
12	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	404401191

13 rows × 25 columns

the `rename` method's `column` parameter can take a lambda defined inline, which is helpful, because we want that function to take one parameter (the current column name) and do the same thing to all of the columns within a single DataFrame, but to prepend a different thing for each DataFrame

```
pd.concat([course_gh_df[cur_col].apply(pd.Series).rename(columns = lambda c:  
cur_col + ' ' + c)  
          for cur_col in js_cols],  
          axis=1)
```

	actor_id	actor_login	actor_display_login	actor_gravatar_id	actor_url	actor_avatar_url	repo_id
0	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
1	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot]	https://avatars.githubusercontent.com/u/41898282?	400283911
2	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
3	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
4	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	400283911
5	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot]	https://avatars.githubusercontent.com/u/41898282?	337917892 rhod
6	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	337917892 rhod
7	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot]	https://avatars.githubusercontent.com/u/41898282?	501809605
8	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	501809605
9	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	501809605
10	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	404401191
11	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	404401191
12	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm	https://avatars.githubusercontent.com/u/10656079?	404401191

13 rows × 25 columns

So now, we have the unpacked columns with good column names, but we lost the columns that were originally good.

How can we append the new columns to the old ones? First we can make a DataFrame that's just the columns not on the list that we're going to expand.

```
course_gf_df_good = course_gh_df[[col for col in
                                    course_gh_df.columns if not(col in js_cols)]]
course_gf_df_good
```

	id	type	public	created_at
0	23017297788	PushEvent	True	2022-07-21 22:52:08+00:00
1	23017256816	PushEvent	True	2022-07-21 22:48:25+00:00
2	23017147996	PushEvent	True	2022-07-21 22:40:28+00:00
3	23017000728	PushEvent	True	2022-07-21 22:28:48+00:00
4	23016409837	PushEvent	True	2022-07-21 21:41:28+00:00
5	22594628501	PushEvent	True	2022-06-29 00:36:34+00:00
6	22594620942	PushEvent	True	2022-06-29 00:35:39+00:00
7	22258790417	CreateEvent	True	2022-06-09 21:15:41+00:00
8	22258731281	CreateEvent	True	2022-06-09 21:11:34+00:00
9	22258730912	CreateEvent	True	2022-06-09 21:11:33+00:00
10	21774885552	PushEvent	True	2022-05-13 12:54:30+00:00
11	21774865161	PushEvent	True	2022-05-13 12:53:22+00:00
12	21774810879	PushEvent	True	2022-05-13 12:50:21+00:00

Then we can prepend that to the list that we pass to `concat`. We have to put it in a list first, then use `+ to do that.`

```
pd.concat([course_gf_df_good]+[course_gh_df[col].apply(pd.Series,).rename(
    columns= lambda i_col: col + '-' + i_col )
    for col in js_cols],axis=1)
```

	<code>id</code>	<code>type</code>	<code>public</code>	<code>created_at</code>	<code>actor_id</code>	<code>actor_login</code>	<code>actor_display_login</code>	<code>actor_gravatar_id</code>	<code>actor_url</code>
0	23017297788	PushEvent	True	2022-07-21 22:52:08+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars
1	23017256816	PushEvent	True	2022-07-21 22:48:25+00:00	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot] https://avatars
2	23017147996	PushEvent	True	2022-07-21 22:40:28+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars
3	23017000728	PushEvent	True	2022-07-21 22:28:48+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars
4	23016409837	PushEvent	True	2022-07-21 21:41:28+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars
5	22594628501	PushEvent	True	2022-06-29 00:36:34+00:00	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot] https://avatars
6	22594620942	PushEvent	True	2022-06-29 00:35:39+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars
7	22258790417	CreateEvent	True	2022-06-09 21:15:41+00:00	41898282	github-actions[bot]	github-actions		https://api.github.com/users/github-actions[bot] https://avatars
8	22258731281	CreateEvent	True	2022-06-09 21:11:34+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars
9	22258730912	CreateEvent	True	2022-06-09 21:11:33+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars
10	21774885552	PushEvent	True	2022-05-13 12:54:30+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars
11	21774865161	PushEvent	True	2022-05-13 12:53:22+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars
12	21774810879	PushEvent	True	2022-05-13 12:50:21+00:00	10656079	brownsarahm	brownsarahm		https://api.github.com/users/brownsarahm https://avatars

13 rows x 29 columns

To see how the list math works

```
{'a'] + ['b', 'c', 'd']
```

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

results in one list

but without the `[]` we get a type error

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

```
-----  
TypeError          Traceback (most recent call last)  
Input In [39], in <cell line: 1>()  
----> 1 'a' + ['b', 'c', 'd']  
TypeError: can only concatenate str (not "list") to str
```

List operations return `None` and mutate the list in place so

```
orig_list = ['a']  
new_items = ['b', 'c', 'd']  
orig_list.extend(new_items)
```

outputs nothing because `None` was returned and it changes the original variable.

```
orig_list
```

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

```
type(orig_list.extend(new_items))
```

```
NoneType
```

is none.

10.2. Questions After Class

All clarifying questions today

10.2.1. How does Axis work?

the notes above are expanded a lot, which should help. You can see more examples in the [Tidy Data Explanation](#) and on the [Cheat Sheet](#).

The `axis` parameter is a parameter in a lot of pandas functions, you can see it used in most of the statistics we used last week as well, because though column operations are the default, we can do row-wise as well.

For more on concatenation, see the [Pandas user guide](#) or [API docs](#) sections on it.

10.2.2. How does melt work?

the notes are expanded a lot. Also see [Tidying data](#).

For the concept, you can also see the original [Tidy Data paper](#).

For the pandas method, see its [docs](#).

10.2.3. What about the NaNs that are still left?

those are Nan in the data because the events are different types and different types of events have different information available about them.

If we `groupby` event type and then look at, for example, the payload columns. We see that the `Nans` are explained by that. (remember, count tells how many are not `NaN`)

11. Missing Data and Inconsistent coding

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

sns.set_theme(palette= "colorblind")
na_toy_df = pd.DataFrame(data = [[1,3,4,5],[2 ,6, np.nan]])

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

coffee_df = pd.read_csv(arabica_data_url)

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

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. To see the types of things:

[Stanford Policy Lab Open Policing Project data readme Propublica Machine Bias](#) the "How we acquired data" section

11.1. Missing Values

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

[in 2020 there was a workshop on it](#)

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

[example application in breast cancer detection](#)

In pandas, even representing [missing values](#) is under [experimentation](#). Currently, it uses `numpy.Nan`, but the experiment is with `pd.NA`.

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

Pandas gives a few basic tools:

- drop with (`dropna`)
- fill with `fillna`

```
coffee_df.head()
```

	Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	...	Color	Category.Two.Defects	Expiration
0	1	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	...	Green	0	April 3rd 201
1	2	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	...	Green	1	April 3rd 201
2	3	Arabica	grounds for health admin	Guatemala	san marcos barrancas "san cristobal cuch	NaN	NaN	NaN	NaN	1600 - 1800 m	...	NaN	0	May 31st 201
3	4	Arabica	yidnekachew dabessa	Ethiopia	yidnekachew dabessa coffee plantation	NaN	wolensu	NaN	yidnekachew debessa coffee plantation	1800-2200	...	Green	2	Marc 25th, 201
4	5	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	...	Green	2	April 3rd 201

5 rows × 44 columns

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

We can look at the type:

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

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

And we can look at the value counts.

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

```

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

```

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

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

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

```

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

```

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

```

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

```

💡 Tip

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. Its under discussion to [deprecate](#)
- write to a new DataFrame
- add a column

We'll use adding a column:

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

💡 Question in Class

When I use value counts it treats the filled ones as different. Why?

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

```

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

```

```
{ coffee_df['lot_number_clean'].value_counts()
```

```

1                      1041
1                      18
020/17                  6
019/17                  5
102                     3
...
3-59-2318                  1
8885                     1
5055                     1
MCCFWXA15/16                  1
017-053-0211/ 017-053-0212      1
Name: lot_number_clean, Length: 222, dtype: int64

```

If we switch to 1 as a string, then we'd see all of the one values as the same thing.

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

```

1          1059
020/17      6
019/17      5
102         3
103         3
...
3-59-2318    1
8885        1
5655        1
MCCFWXA15/16 1
017-053-0211 1
Name: lot_number_clean, Length: 221, dtype: int64

```

This was our goal, so in this case, it's the right thing to do to overwrite the value.

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.

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

```

coffee_df.shape
(1311, 45)

```

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

```

coffee_df.dropna().shape
(130, 45)

```

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

```

coffee_df.dropna(subset=['altitude_low_meters']).shape
(1084, 45)

```

whatever you do, document it

Note

subset operates along columns by default, because axis is set to 0, by default.

Try it Yourself

use the `na_toy_df` DataFrame that's defined in the first cell, to experiment with subset and axis parameters to understand them better.

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, 44)

```

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

```

coffee_df.dropna(thresh=.85*n_rows, axis=1).shape
(1311, 34)

```

Important

Everthing after this is new material that we did not have time for in class, but is important and helpful in your assignment (and for your portfolio).

11.2. Inconsistent values

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

Let's look at the `In.Country.Partner` column

```

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

```

```

AMECAFE
265
Africa Fine Coffee Association
49
Almacafé
178
Asociacion Nacional Del Café
155
Asociación Mexicana De Cafés y Cafeterías De Especialidad A.C.
6
Asociación de Cafés Especiales de Nicaragua
8
Blossom Valley International
58
Blossom Valley International\n
1
Brazil Specialty Coffee Association
67
Central De Organizaciones Productoras De Café y Cacao Del Perú - Central Café &
Cacao 1
Centro Agroecológico del Café A.C.
8
Coffee Quality Institute
7
Ethiopia Commodity Exchange
18
Instituto Hondureño del Café
60
Kenya Coffee Traders Association
22
METAD Agricultural Development plc
15
NUCOFFEE
36
Salvadoran Coffee Council
11
Specialty Coffee Ass
1
Specialty Coffee Association
295
Specialty Coffee Association of Costa Rica
42
Specialty Coffee Association of Indonesia
10
Specialty Coffee Institute of Asia
16
Tanzanian Coffee Board
6
Torch Coffee Lab Yunnan
2
Uganda Coffee Development Authority
22
Yunnan Coffee Exchange
12
Name: In.Country.Partner, dtype: int64

```

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

This is another job for dictionaries, we make one with the value to replace as the key and the value to insert as the value.

```

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()

```

```

AMECAFE
265
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: in_country_partner_clean, dtype: int64

```

and now we see the corrected values. We can also pass lambdas or put lambdas in the dictionary if there are systemic patterns.

11.3. Fixing data at load time

Explore some of the different parameters in [read_csv](#)

How can we read in data that looks like this:

Ethnicity	Asian				Black				Hispanic			
	Female		Male		Female		Male		Female		Male	
Gender	count	mean	count	mean	count	mean	count	mean	count	mean	count	n
Age Cohort												
0 - 5	6	1544.33333							18	1431.33333	26	1366
51 +	-	-	-	-	-	-	-	-	-	-	-	-

```
pd.read_csv('fancy_formatting.xlsx', header = list(range(4)))
```

Many problems can be repaired with parameters in [read_csv](#).

11.4. 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?

Things to keep in mind:

- always save new copies of data when you mutate it
- add new columns rather than overwriting columns
- long variable names are better than ambiguous naming

11.5. Your observations from Monday:

I promised we'd come back to your observations on what problems could occur in data. Here they are, organized by rough categories of when/how to fix them.

We can fix while reading in data:

- decimal was indicated with ',' instead of '.' so pandas saw value as a string rather than a float
- missing header
- reading the index as a column
- large datasets might be too slow or not fit in memory
- missing data represented with a value or special character

We can fix by reshaping data:

- Data can get read into tables in bizarre ways depending on how the data was entered originally.
- every value in one column, instead of separated

We can repair by changing values or filtering:

- information represented inconsistently eg "Value" and " Value " or twenty-two instead of 22
- blank rows or blank columns or data that is N/A
- date/time information can be represented lots of different ways
- representing categorical with numbers that are ambiguous
- spaces or other symbols in column names
- some numbers as strings, others as ints within a column
- symbols being mis interpreted

Real problems, but beyond our scope:

- corrupt data files

11.6. More Practice

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](#).

12. Building Datasets From multiple Sources

```
1. remember, that  
1. with markdown  
1. you can make a numbered list  
1. using all `1.`
```

renders as:

1. remember, that
2. with markdown
3. you can make a numbered list
4. using all 1.

```

import pandas as pd

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

```

Today we're going to look at some data on NBA (National Basketball Association) players.

12.1. Combining Multiple Tables

```

p18 = pd.read_csv(course_data_url + '2018-players.csv')
p19 = pd.read_csv(course_data_url + '2019-players.csv')

p18.head()

```

	TEAM_ID	PLAYER_ID	SEASON
0	1610612761	202695	2018
1	1610612761	1627783	2018
2	1610612761	201188	2018
3	1610612761	201980	2018
4	1610612761	200768	2018

12.1.1. Stacking with Concat

We've seen one way of putting dataframes together with `concat` we used it when [Unpacking Jsons](#). In that case, we stacked them side by side, now we want to stack them vertically, which is the default of concat.

```

players_df = pd.concat([p18,p19])
players_df.head()

```

	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

This has the same columns, and we can see what happened more by looking at the shape of each.

```

players_df.shape

(1374, 4)

p18.shape, p19.shape

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

```

We can verify that the length of the new data frame is the sum of the original two DataFrames.

```

assert len(p18) + len(p19) == len(players_df)

```

12.1.2. Combining Data with Merge

What is we want to see which players changed teams from 2018 to 2019? We can do this with `merge`. For `merge` we have to tell it a left DataFrame and a right DataFrame and on what column to match them up.

The left and right DataFrames will be used different ways, but any DataFrame can be put in either position.

For this case, we will use 2018 data as left ,and 2019 as right and then merge `on='PLAYER_ID'`.

```

pd.merge(p18,p19,on='PLAYER_ID').head()

  TEAM_ID_X  PLAYER_ID  SEASON_X  PLAYER_NAME  TEAM_ID_Y  SEASON_Y
0  1610612761    202695    2018   Kawhi Leonard  1610612746    2019
1  1610612761    1627783    2018   Pascal Siakam  1610612761    2019
2  1610612761    201188    2018   Marc Gasol    1610612761    2019
3  1610612763    201188    2018   Marc Gasol    1610612761    2019
4  1610612761    201980    2018   Danny Green   1610612747    2019

```

Now, we get what we expect, but the column names have `_x` and `_y` on the end (as a [suffix](#), appended to the original). We'll add 2018 and 2019 respectively, separated with a `_`.

```

year_over_year = pd.merge(p18,p19,on='PLAYER_ID',suffixes=('_2018','_2019'))
year_over_year.head()

```

	TEAM_ID_2018	PLAYER_ID	SEASON_2018	PLAYER_NAME	TEAM_ID_2019	SEASON_2019
0	1610612761	202695	2018	Kawhi Leonard	1610612746	2019
1	1610612761	1627783	2018	Pascal Siakam	1610612761	2019
2	1610612761	201188	2018	Marc Gasol	1610612761	2019
3	1610612763	201188	2018	Marc Gasol	1610612761	2019
4	1610612761	201980	2018	Danny Green	1610612747	2019

Now that it's a little bit cleaner, we will examine how it works by looking at the shape.

```
[year_over_year.shape, p18.shape, p19.shape]
```

```
((538, 6), (748, 3), (626, 4))
```

This kept only the players that played both years, with repetitions for each team they played on for each year.

We can check the calculation with set math (python `set` type has operations for math sets like intersect and set difference)

```
# player IDs for each year, no repeats
p19_u = set(p19['PLAYER_ID'])
p18_u = set(p18['PLAYER_ID'])

# player IDs that played both years
p1819 = p18_u.intersection(p19_u)

# teams per player per year
teams_per_p18 = p18['PLAYER_ID'].value_counts()
teams_per_p19 = p19['PLAYER_ID'].value_counts()

# total number of team-player combinations
# multiply number of teams each player played for in 18 by number of teams in 19
# then sum. (most of these are 1's)
sum(teams_per_p19[p1819]* teams_per_p18[p1819])
```

```
/tmp/ipykernel_2471/3008611582.py:15: FutureWarning: Passing a set as an indexer is deprecated and will raise in a future version. Use a list instead.
    sum(teams_per_p19[p1819]* teams_per_p18[p1819])
/tmp/ipykernel_2471/3008611582.py:15: FutureWarning: Passing a set as an indexer is deprecated and will raise in a future version. Use a list instead.
    sum(teams_per_p19[p1819]* teams_per_p18[p1819])
```

```
538
```

We can also merge so that we keep all players on either team using `how='outer'` the default value for `how` is inner, which takes the intersection (but with duplicates, does some extra things as we saw). With `outer` it takes the union, but with extra handling for the duplicates.

```
[pd.merge(p18,p19,on='PLAYER_ID',suffixes=('_2018','_2019'),how='outer').shape]
```

```
(927, 6)
```

It's the total of the rows we had before, plus the total number of player-teams for players that only played in one of the two years.

```
#players tha tonly played in one year
o18 = p18_u.difference(p19_u)
o19 = p19_u.difference(p18_u)
# teams those players played for + the above 538
teams_per_p19[o19].sum() + teams_per_p18[o18].sum() + sum(teams_per_p19[p1819]* teams_per_p18[p1819])
```

```
/tmp/ipykernel_2471/955177081.py:5: FutureWarning: Passing a set as an indexer is deprecated and will raise in a future version. Use a list instead.
    teams_per_p19[o19].sum() + teams_per_p18[o18].sum() + sum(teams_per_p19[p1819]* teams_per_p18[p1819])
/tmp/ipykernel_2471/955177081.py:5: FutureWarning: Passing a set as an indexer is deprecated and will raise in a future version. Use a list instead.
    teams_per_p19[o19].sum() + teams_per_p18[o18].sum() + sum(teams_per_p19[p1819]* teams_per_p18[p1819])
/tmp/ipykernel_2471/955177081.py:5: FutureWarning: Passing a set as an indexer is deprecated and will raise in a future version. Use a list instead.
    teams_per_p19[o19].sum() + teams_per_p18[o18].sum() + sum(teams_per_p19[p1819]* teams_per_p18[p1819])
/tmp/ipykernel_2471/955177081.py:5: FutureWarning: Passing a set as an indexer is deprecated and will raise in a future version. Use a list instead.
    teams_per_p19[o19].sum() + teams_per_p18[o18].sum() + sum(teams_per_p19[p1819]* teams_per_p18[p1819])
```

```
927
```

We can save this to a variable

```
[year_over_year_outer = pd.merge(p18,p19,on='PLAYER_ID',suffixes=('_2018','_2019'),how='outer')]
```

then look at a few rows to see that it has indeed filled in with NaN in the places where there wasn't a value.

```
[year_over_year_outer.sample(10)]
```

	TEAM_ID_2018	PLAYER_ID	SEASON_2018	PLAYER_NAME	TEAM_ID_2019	SEASON_2019
603	1.610613e+09	101162	2018.0		NaN	NaN
106	1.610613e+09	101108	2018.0	Chris Paul	1.610613e+09	2019.0
675	1.610613e+09	1629156	2018.0		NaN	NaN
72	1.610613e+09	1627750	2018.0	Jamal Murray	1.610613e+09	2019.0
54	1.610613e+09	203086	2018.0	Meyers Leonard	1.610613e+09	2019.0
648	1.610613e+09	1629152	2018.0	DeVaughn Akoon-Purcell	1.610613e+09	2019.0
517	1.610613e+09	201585	2018.0		NaN	NaN
229	1.610613e+09	1628379	2018.0	Luke Kennard	1.610613e+09	2019.0
399	1.610613e+09	1628396	2018.0	Tony Bradley	1.610613e+09	2019.0
677	1.610613e+09	203130	2018.0		NaN	NaN

Note

We can also tell that there are NaN because it cast the year to float from int.

12.2. Merge types, in detail

We can examine how these things work more visually with smaller DataFrames:

```
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", "K2"],  
        "key2": ["K0", "K0", "K0"],  
        "C": ["C0", "C1", "C2", "C3"],  
        "D": ["D0", "D1", "D2", "D3"],  
    }  
)
```

```
left
```

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

```
right
```

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

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

	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

Note

inner is default, but we can state it to be more explicit

12.3. Duplicate Keys

```
left = pd.DataFrame({"A": [1, 2], "B": [2, 2]})  
right = pd.DataFrame({"A": [4, 5, 6], "B": [2, 2, 2]})  
result = pd.merge(left, right, on="B", how="outer")
```

```
left
```

	A	B
0	1	2
1	2	2

```
right
```

	A	B
0	4	2
1	5	2
2	6	2

```
result
```

A_x	B	A_y
0	1 2	4
1	1 2	5
2	1 2	6
3	2 2	4
4	2 2	5
5	2 2	6

If we ask pandas to validate the merge with `validate` and a specific type of merge, it will throw an error if that type of merge is not possible.

```
pd.merge(left, right, on='B', validate='one_to_one')

MergeError                                     Traceback (most recent call last)
Input In [26], in <cell line: 1>()
----> 1 pd.merge(left, right, on='B', validate='one_to_one')

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/reshape/merge.py:107, in merge(left, right, how, on, left_on,
right_on, left_index, right_index, sort, suffixes, copy, indicator, validate)
    90 @Substitution("\nleft : DataFrame or named Series")
    91 @Appender(_merge_doc, indents=0)
    92 def merge(
(...):
    95     validate: str | None = None,
    96 ) -> DataFrame:
--> 97     op = _MergeOperation(
    98         left,
    99         right,
   100         how=how,
   101         on=on,
   102         left_on=left_on,
   103         right_on=right_on,
   104         left_index=left_index,
   105         right_index=right_index,
   106         sort=sort,
   107         suffixes=suffixes,
   108         copy=copy,
   109         indicator=indicator,
   110         validate=validate,
   111     )
   112     return op.get_result()

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/reshape/merge.py:710, in _MergeOperation.__init__(self, left,
right, how, on, left_on, right_on, axis, left_index, right_index, sort, suffixes,
copy, indicator, validate)
    706 # If argument passed to validate,
    707 # check if columns specified as unique
    708 # are in fact unique.
    709 if validate is not None:
--> 710     self._validate(validate)

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/core/reshape/merge.py:1427, in _MergeOperation._validate(self,
validate)
    1425 if validate in ["one_to_one", "1:1"]:
    1426     if not left_unique and not right_unique:
--> 1427         raise MergeError(
    1428             "Merge keys are not unique in either left "
    1429             "or right dataset; not a one-to-one merge"
    1430         )
    1431     elif not left_unique:
    1432         raise MergeError(
    1433             "Merge keys are not unique in left dataset; not a one-to-one
merge"
    1434     )

MergeError: Merge keys are not unique in either left or right dataset; not a one-
to-one merge
```

Further Reading

The [pandas documentation](#) is a good place to read through their examples on how validate works. It's important to note the types of possible joins from the [beginning of the section](#)

12.4. Questions at the End of Class

12.4.1. How to remove certain columns in merges (i.e. the year columns in the basketball dataset since they are redundant)

They can be dropped after (or before) using `drop()`

12.4.2. Can we merge as many DataFrames as we would want?

Merge is specifically defined for two DataFrames, but the result returns a DataFrame, so it could be passed to another merge.

12.4.3. Is there no way to merge the left and right for the A and B data?

All merges are possible, they just each give a different result.

12.4.4. Can you use merging to check correlation between two different sets?

You can use merging to check *overlap* between two DataFrames, for actual sets, it's probably best to represent them as `set` type objects and use set operations.

However, these type of operations are not actually correlations, which we'll learn a bit about in a few weeks when we talk about regression

12.4.5. Is there any way to compare the team IDs in both datasets so that it outputs the players that changed teams between those times?

From the `year_over_year` DataFrame we made above, we can check which players changed teams. Since players also can change teams *within* a season, this is a tricky question, but we could, for example look only for players that have no overlapping teams from 2018 to 2019.

```

change_check = lambda r: not(r['TEAM_ID_2018']==r['TEAM_ID_2019'])
year_over_year_clean['CHANGE_TEAM'] =
year_over_year_clean.apply(change_check, axis=1)
year_over_year_clean.head()

```

```

NameError                                 Traceback (most recent call last)
Input In [27], in <cell line: 2>()
      1 change_check = lambda r: not(r['TEAM_ID_2018']==r['TEAM_ID_2019'])
----> 2 year_over_year_clean['CHANGE_TEAM'] =
year_over_year_clean.apply(change_check, axis=1)
      3 year_over_year_clean.head()

NameError: name 'year_over_year_clean' is not defined

```

Then we can filter the data by the new column, then take out only the player information and drop any duplicates for players that played in multiple teams in one year or the other.

```

y_o_y_changes = year_over_year_clean[year_over_year_clean['CHANGE_TEAM']]
y_o_y_change_players =
y_o_y_changes[['PLAYER_ID', 'PLAYER_NAME']].drop_duplicates()
y_o_y_change_players.head()

```

```

NameError                                 Traceback (most recent call last)
Input In [28], in <cell line: 1>()
----> 1 y_o_y_changes = year_over_year_clean[year_over_year_clean['CHANGE_TEAM']]
      2 y_o_y_change_players =
y_o_y_changes[['PLAYER_ID', 'PLAYER_NAME']].drop_duplicates()
      3 y_o_y_change_players.head()

NameError: name 'year_over_year_clean' is not defined

```

and we can check how many players that is

```
y_o_y_change_players.shape
```

```

NameError                                 Traceback (most recent call last)
Input In [29], in <cell line: 1>()
----> 1 y_o_y_change_players.shape

NameError: name 'y_o_y_change_players' is not defined

```

12.4.6. in year_over_year, where suffixes=('_2018','_2019'), why is there an underscore?

this is optional, but is best practice for pythonic variable naming. (and remember columns are like variables)

12.4.7. If we merge two dataframes which each have a column (or columns) that are the same, but each have different data types in those columns, what happens?

If they're just named the same, but not the "on" column, you'll get the two columns with different suffixes default ('_x', 'y'). If it's the on column, the merge will be empty

12.4.8. how does suffix to know to change the date?

Suffix appends what we tell it to to the column names that occur in both datasets

12.5. More Practice

1. Use a merge to figure out how many players did not return for the 2019 season
2. Also load the `conferences.csv` data. Use this to figure out how many players moved conferences from 2018 to 2019.

13. Reviewing Merges & Databases

```
import pandas as pd
```

```
pd.merge?
```

a review problem

If `weather_df` has a row for every date in the past 100 years, a column named `date` and columns for the weather that day and `birthdays_df` has a row for each member of the CS department with a column `birthdate` and a second column name, how would you merge these two datasets to find out the weather on the day each person was born?

```
bdy_weather = pd.merge(left= weather_df, right = birthdays_df,
left_on = 'date', right_on ='birthdate', how= 'inner')
```

13.1. Working with Databases

Off the shelf, pandas cannot read databases by default. We'll use the `sqlite3` library.

```
import sqlite3
```

First, we set up a connection, that links the the notebook to the database.

```
conn = sqlite3.connect('data/nba1819.db')
```

```
conn
```

```
<sqlite3.Connection at 0x7f7a62a545d0>
```

Note

[SQLite](#) is a type of database.

To use it, we add a cursor.

```
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 0x7f7a62b0e1f0>
```

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

```
cursor.fetchall()
```

```
[('teams',),
 ('conferences',),
 ('playerGameStats2018',),
 ('playerGameStats2019',),
 ('teamGameStats2018',),
 ('teamGameStats2019',),
 ('playerTeams2018',),
 ('playerTeams2019',),
 ('teamDailyRankings2018',),
 ('teamDailyRankings2019',),
 ('playerNames',)]
```

We can also pass queries to the database through pandas and then get it returned as a DataFrame.

```
pd.read_sql('SELECT PLAYER_ID, PLAYER_NAME FROM playerNames', conn).head(1)
```

	PLAYER_ID	PLAYER_NAME
0	202695	Kawhi Leonard

We can use `*` to get all of the columns and `LIMIT` to reduce the number of rows.

```
pd.read_sql('SELECT * FROM playerNames LIMIT 5', conn)
```

index	PLAYER_NAME	PLAYER_ID
0	Kawhi Leonard	202695
1	Pascal Siakam	1627783
2	Marc Gasol	201188
3	Danny Green	201980
4	Kyle Lowry	200768

How can get all of the data from the teams table?

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

index	LEAGUE_ID	TEAM_ID	MIN_YEAR	MAX_YEAR	ABBREVIATION	NICKNAME	YEARFOUNDED	CITY	ARENA	ARENACAPACITY	OWNER	GENERALM.
0	0	1610612737	1949	2019	ATL	Hawks	1949	Atlanta	State Farm Arena	18729.0	Tony Ressler	Travis Reed
1	1	1610612738	1946	2019	BOS	Celtics	1946	Boston	TD Garden	18624.0	Wyc Grousbeck	Darren Rovell
2	2	1610612740	2002	2019	NOP	Pelicans	2002	New Orleans	Smoothie King Center	NaN	Tom Benson	Trajan
3	3	1610612741	1966	2019	CHI	Bulls	1966	Chicago	United Center	21711.0	Jerry Reinsdorf	Gregg Doyel
4	4	1610612742	1980	2019	DAL	Mavericks	1980	Dallas	American Airlines Center	19200.0	Mark Cuban	Donna

13.2. More practice

For each of the following, think about what to query from the database and how to merge the tables after. Also, consider if these questions are all specific enough or require more decisions to be made.

1. How many players changed from the `East` conference to the `West` conference during the 2018 season?
2. How many from the `East` conference to the `West` conference from the 2018 season to the 2019 season?
3. Did teams that were founded earlier have better records in the 2018 season?

Think ahead

For a portfolio, you could ask questions like do events in a city impact if the home team wins later that week?

14. Web Scraping

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

Tip

To update a package while running jupyter in a notebook:

```
!pip install packagename update
```

then restart the kernel

14.1. Figuring out what to scrape

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

from the [people page](#) of the department website.

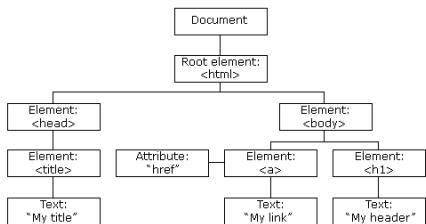
With great power comes great responsibility.

- always check [robots.txt](#)
- do not do things that the owner says not to do
- government websites are typically safe, because of open data rules

We can inspect the page to check that it's well structured by right clicking in a browser tab and looking at the same code that our browser sees in order to render the page.

We can basically think of web scraping as loading data that's not tabular but instead is formatted as html code. HTML code can be well structured and hierarchical within a single page, or you could collect information about a broad topic by using a little bit from many many pages. We're going to work from one page here.

HTML code consists of tags and the text of the page. The tags label the content and define different structure of the page.



Once we've decided that it will work, we can begin working.

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

14.2. Loading with requests

First, we `get` the data using the python [requests](#) library.

```
{ requests.get(cs_people_url) }
```

```
<Response [200]>
```

this returns an object, but we want the content from it, so we'll save that to a variable

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

Think Ahead

You can use this same basic logic, that anything can be data to consider other questions like, for example:

- How do the sizes of datasets for Tidy Tuesday vary? (you don't need to download and load all of the data, only traverse the readmes)
- On average, how many code cells are there in each class? How much does the amount of code in the posted notes vary from your notes you take in class?

This is literally just the html as a browser would see, but we pulled it into python.

14.3. Parsing with BeautifulSoup

Next, we'll use BeautifulSoup to parse the text. Parsing means to make sense of it. In this case, it transforms from a string or characters, to a datastructure that we can work with.

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

First we note that it now formats the new lines.

cs_people

```

<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="utf-8"/>
<meta content="width=device-width, initial-scale=1" name="viewport"/>
<link href="https://gmpg.org/xfn/11" rel="profile"/>
<title>People - Department of Computer Science and Statistics</title>
<meta content="max-image-preview:large" name="robots">
<link href="https://s.w.org" rel="dns-prefetch">
<link href="https://web.uri.edu/cs/feed/" rel="alternate" title="Department of Computer Science and Statistics » Feed" type="application/rss+xml"/>
<link href="https://web.uri.edu/cs/comments/feed/" rel="alternate" title="Department of Computer Science and Statistics » Comments Feed" type="application/rss+xml"/>
<script type="text/javascript">
window._wpemojiSettings =
{
   baseUrl:"https://s.w.org/images/core/emoji/13.0.1/72x72/",
   ext:".png",
   svgUrl:"https://s.w.org/images/core/emoji/13.0.1/svg/",
   svgExt:".svg",
   source:[{"concatemoji":"https://web.uri.edu/cs/wp-includes/js/wp-emoji-release.min.js?ver=5.7.1"}];
}
!function(e,a,t){var
n,r,o,i=a.createElement("canvas"),p=i.getContext("2d");function
s(e,t){var
a=String.fromCharCode;p.clearRect(0,0,i.width,i.height),p.fillText(a.apply(this,e),
[0,0);e=i.toDataURL();return
p.clearRect(0,0,i.width,i.height),p.fillText(a.apply(this,t),[0,0),e==i.toDataURL(
))}createElement("script");t.src=e,t.defer=t.type="text/javascript",a.getElements
ByTagName("[!head]")[0].append(t);for(o=Array("flag","emoji1"),t.supports=
{everything:!0,everythingExceptFlag:!0},r=0;r<o.length;r++)t.supports[o[r]]=functi
on(e)(if(p||p.fillText) return n;switch(p.textBaseline){top,p.font="600 32px
Arial",e)(case"flag":return s([127987,65039,8205,9895,65039])?!1:is([55356,56826,55356,56819,
[127987,65039,8203,9895,65039])?!1:is([55356,56826,55128,56423,56128,56418,56128,564
21,56128,56430,56128,56423,56128,56447],
[55356,56826,8203,55356,56819])&&is([55356,57332,56128,56423,56128,56430,8
203,56128,56423,8203,56128,56447]);case"emoji":return !s([55357,56424,8205,55356,57
212],[55357,56424,8203,55356,57212])}return n
}(o[r]),t.supports.everything=t.supports.everything&&t.supports[o[r]],"flag"!=o[r]
&&
(t.supports.everythingExceptFlag=t.supports.everythingExceptFlag&&t.supports[o[r]])
;t.supports.everythingExceptFlag=t.supports.everythingExceptFlag&&t.supports.flia
g,t.DOMReady=!1,t.readyCallback=function(){t.DOMReady=!0},t.supports.everything||
(n=funciton(){t.readyCallback()}).a.addEventListener?
(a.addEventListener("DOMContentLoaded",n,!1),e.addEventListener("load",n,!1)):
(e.attachEvent("onload",n),a.attachEvent("onreadystatechange",function()
{"complete"==a.readyState&&t.readyCallback()),(n=t.source||{}).concatemoji?
c(n.concatemoji):n.wpemoji&&n.twemoji&&(c(n.twemoji),c(n.wpemoji)))
(window,document>window._wpemojiSettings);
</script>
<style type="text/css">
img.wp-smiley,
img.emoji {
    display: inline !important;
    border: none !important;
    box-shadow: none !important;
    height: 1em !important;
    width: 1em !important;
    margin: 0 .07em !important;
    vertical-align: -0.1em !important;
    background: none !important;
    padding: 0 !important;
}
</style>
<link href="https://web.uri.edu/cs/wp-includes/css/dist/block-
library/style.min.css?ver=5.7.1" id="wp-block-library-css" media="all"

```

```

rel="stylesheet" type="text/css"/>
<link href="https://web.uri.edu/cs/wp-content/plugins/uri-component-library/css/cl_built.css?ver=5.0.2" id="uricl-css-css" media="all"
rel="stylesheet" type="text/css"/>
<link href="https://web.uri.edu/cs/wp-content/plugins/uri-courses/assets/courses.css?ver=5.7.1" id="uri-courses-styles-css" media="all"
rel="stylesheet" type="text/css"/>
<link href="https://web.uri.edu/cs/wp-content/plugins/uri-people-tool/assets/people.css?ver=5.7.1" id="uri-people-styles-css" media="all"
rel="stylesheet" type="text/css"/>
<link href="https://web.uri.edu/cs/wp-content/plugins/wp-lightbox-2/styles/lightbox.min.css?ver=1.3.4" id="wp-lightbox-2.min.css-css" media="all"
rel="stylesheet" type="text/css"/>
<link href="https://web.uri.edu/cs/wp-content/themes/uri-modern/style.css?ver=2.4.0" id="uri-modern-style-css" media="all" rel="stylesheet"
type="text/css"/>
<link href="https://web.uri.edu/cs/wp-content/plugins/tablepress/css/default.min.css?ver=1.13" id="tablepress-default-css" media="all" rel="stylesheet" type="text/css"/>
<script id="jquery-core-js" src="https://web.uri.edu/cs/wp-includes/js/jquery/jquery.min.js?ver=3.5.1" type="text/javascript">></script>
<script id="jquery-migrate-js" src="https://web.uri.edu/cs/wp-includes/js/jquery/jquery-migrate.min.js?ver=3.3.2" type="text/javascript"></script>
<link href="https://web.uri.edu/cs/wp-json/" rel="https://api.w.org/"><link href="https://web.uri.edu/cs/wp-json/v2/pages/629" rel="alternate" type="application/json"/><link href="https://web.uri.edu/cs/xmlrpc.php?rsd" rel="EditURI" title="RSID" type="application/rsd+xml"/>
<link href="https://web.uri.edu/cs/wp-includes/wlwmanifest.xml" rel="wlwmanifest" type="application/wlwmanifest+xml"/>
<meta content="WordPress 5.7.1" name="generator"/>
<link href="https://web.uri.edu/cs/people/" rel="canonical"/>
<link href="https://web.uri.edu/cs/p-629" rel="shortlink"/>
<link href="https://web.uri.edu/cs/wp-json/oembed/1.0/embed?url=https%3A%2F%2Fweb.uri.edu%2Fcs%2Fpeople%2F" rel="alternate" type="application/json+oembed"/>
<link href="https://web.uri.edu/cs/wp-json/oembed/1.0/embed?url=https%3A%2F%2Fweb.uri.edu%2Fcs%2Fpeople%2F&format=xml" rel="alternate" type="text/xml+oembed"/>
<meta content="People Full-time Faculty Adjunct Faculty and Limited Join Appointments" name="description"/>
<meta content="summary_large_image" name="twitter:card"/>
<meta content="@universityofri" name="twitter:site"/>
<meta content="https://web.uri.edu/cs/people/" name="twitter:creator"/>
<meta content="People" property="og:title"/>
<meta content="People Full-time Faculty Adjunct Faculty and Limited Join Appointments" property="og:description"/>
<meta content="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/logo-wordmark.png" property="og:image"/>
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm_start':new Date().getTime(),event:'gtm.js'});var f=d.createElementByTagName(s)[0],j=d.createElement(s),di=l!='dataLayer'?'&l='+l+'j':async=true;j.src='https://www.googletagmanager.com/gtm.js?id='+i+d;l=f.parentNode.insertBefore(j,f);})(window,document,'script','dataLayer','GTM-K5GL9W');

```

```

margin-top:5px;
margin-bottom:5px;
line-height:1.75;
background:#e4f2f2;
text-decoration:none;
float:left;
position:relative;
width:calc(100% * 4.68)
}
#calendar .event.four-day{
width:calc(76% * 4.68)
}
#calendar .event.mobile {
display:none
}
#calendar .event.scratch {
background:#E2D58B;
}
#calendar .event.lego {
background:#E1EDE8;
}
#calendar .event.datacience {
background:#FCB97D;
}
#calendar .event.giristech {
background:#C2CFB2;
}
#calendar .event.games {
background:#F79365;
}
#calendar .event.programming {
background:#BCDCE0;
}
#calendar .event.webdesign {
background:#E09FA2;
}
#calendar .event-desc {
color:#666;
margin:3px 0 7px;
text-decoration:none;
display:inline-block
}
#calendar .event-time {
margin:3px 0 7px 5px;
text-decoration:none;
display:inline-block
}
#calendar .other-month {
background:#f5f5f5;
color:#666
}
#desc {
display:none;
background:#eee;
box-shadow: 0 0 4px #999;
padding:10px;
color: #000;
}
#calendar .event-desc:hover+#desc {
display:block;
position:absolute;
z-index:2
}
.days+header h1 {
padding-top:2rem;
clear: left
}
@media(max-width:768px) {
#calendar .event img,#calendar .other-month,#calendar .weekdays {
display:none
}
#calendar .event {
width:calc(100% - 2em);
padding-left:20px;
padding-right:10px
}
#calendar .event.mobile,#calendar ul.days.mobile {
display:block
}
#calendar li {
height:auto!important;
border:1px solid #eddede;
width:100%;
padding:10px;
margin-bottom:-1px
}
#calendar .date {
float:none
}
}
.peopleitem.has-thumbnail img {
width: 200px;
}
</style>
<!-- Favicons -->
<link color="#005eff" href="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/safari-pinned-tab.svg" rel="mask-icon"/>
<link href="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/favicon.png" rel="icon" type="image/png"/>
<link href="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/apple-touch-icon.png" rel="apple-touch-icon"/>
<link href="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/apple-touch-icon-180x180.png" rel="apple-touch-icon" sizes="180x180"/>
</meta></link></meta></head>
<body class="page-template-default page page-id-629 page-parent group-blog inline-people">
<script><iframe height="0" src="https://www.googletagmanager.com/ns.html?id=GTM-K5GL9W" style="display:none;visibility:hidden" width="0"></iframe></noscript>
<div class="site" id="page">
<a class="skip-link screen-reader-text" href="#content">Skip to content</a>
<div id="masthead">
<header class="site-header" id="brandbar" role="banner">
<div id="identity-print"></div>
<div id="globalsearch" role="search">
<input aria-label="Toggle visibility of the search box." id="gsform-toggle" role="presentation" type="checkbox"/>
<label for="gsform-toggle" id="gsform"><span>Search</span></label>
<form action="https://www.uri.edu/search" id="gs" method="get" name="global_general_search_form">
<input name="cx" type="hidden" value="016863979916529535900:17qaibakniu">
<input name="cof" type="hidden" value="FORID:11"/>
<label for="gs-query" id="gs-query-label">Searchbox</label>
<input id="gs-q" name="q" placeholder="Search" role="searchbox" type="text" value="">
<input class="searchsubmit" id="gs-submit" name="searchsubmit" type="submit" value="Search"/>
</input></form>

```

```
</div>
<div id="globalbanner-wrapper">
<div id="globalbanner">
<a href="https://www.uri.edu/" title="University of Rhode Island"><div id="identity">University of Rhode Island</div></a>
<div id="gateways">
<input aria-label="Open the audience gateways menu when browsing on mobile" id="gateways-toggle" role="presentation" type="checkbox"/>
<label for="gateways-toggle" id="gateways-label"><span>You</span></label>
<ul id="gateways-menu" role="menu">
<li><a href="https://www.uri.edu/gateway/future-students" role="menuitem">Future Students</a></li>
<li><a href="https://www.uri.edu/gateway/students" role="menuitem">Students</a></li>
<li><a href="https://www.uri.edu/gateway/faculty" role="menuitem">Faculty</a></li>
<li><a href="https://www.uri.edu/gateway/staff" role="menuitem">Staff</a></li>
<li><a href="https://www.uri.edu/gateway/families" role="menuitem">Parents and Families</a></li>
<li><a href="https://www.uri.edu/gateway/alumni" role="menuitem">Alumni</a></li>
<li><a href="https://www.uri.edu/gateway/community" role="menuitem">Community</a></li>
</ul>
</div>
</div>
</div>
</header><!-- #brandbar -->
<header id="siteheader">
<div class="light" id="sitebanner">
<div id="sb-backdrop">
<div id="sb-background-image" style="background-image:url(<https://web.uri.edu/cs/files/cropped-Big-Data.jpg>)"/></div>
<div id="sb-screen"></div>
</div>
<div id="sitebranding">
<div id="siteidentity">
<h1 class="site-title">
<a href="https://web.uri.edu/cs/" rel="home">
          Department of Computer Science and Statistics
        </a>
      </h1>
<h2 class="site-description">College of Arts and Sciences</h2>
</div>
<div id="sitesocial">
</div>
</div><!-- #sitebranding -->
</div><!-- #sitebanner -->
<div class="content-width" id="navigation">
<nav aria-label="Breadcrumb" id="breadcrumbs">
<ol><li><a href="https://www.uri.edu/">URI</a></li><li><a href="https://web.uri.edu/artscli">Arts and Sciences</a></li><li><a href="https://web.uri.edu/cs">Department of Computer Science and Statistics</a></li><li aria-current="page">People</li></ol></nav>
<div id="localnav">
<section class="cl-wrapper cl-menu-wrapper"><div class="cl-menu" data-name="Site Menu" data-show-title="0" id="cl-localnav"><ul class="cl-menu-list cl-menu-list-no-js" id="menu-navigation"><li class="menu-item menu-item-type-custom menu-item-object-custom menu-item-home menu-item-8243" id="menu-item-8243"><a href="https://web.uri.edu/cs" title="Home">Home</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8255" id="menu-item-8255"><a href="https://web.uri.edu/cs/about/" title="About">About</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8806" id="menu-item-8806"><a href="https://web.uri.edu/cs/academics/">Academics</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page current-menu-item page_item page-item-629 current_page_item menu-item-8257" id="menu-item-8257"><a href="https://web.uri.edu/cs/people/" title="People">People</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-9661" id="menu-item-9661"><a href="https://web.uri.edu/cs/research/">Research</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-10871" id="menu-item-10871"><a href="https://web.uri.edu/cs/news-and-events/">News and Events</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8267" id="menu-item-8267"><a href="https://web.uri.edu/cs/contact/" title="Contact">Contact</a></li>
</ul></div></section></div>
</div>
</header>
</div>
<div class="site-content" id="content">
<main class="site-main" id="main" role="main">
<article class="post-629 page type-page status-publish hentry" id="post-629">
<div class="entry-content">
<h1>People</h1>
<section class="cl-wrapper cl-menu-wrapper"><div class="cl-menu" data-name="People" data-show-title="0" id=""><ul class="cl-menu-list cl-menu-list-no-js" id="menu-people"><li class="menu-item menu-item-type-post_type menu-item-object-page current-menu-item page_item page-item-629 current_page_item menu-item-8737" id="menu-item-8737"><a href="https://web.uri.edu/cs/people/" aria-current="page" href="https://web.uri.edu/people/">Faculty</a>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8746" id="menu-item-8746"><a href="https://web.uri.edu/cs/people/staff/">Staff</a></li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-11844" id="menu-item-11844"><a href="https://web.uri.edu/cs/people/faculty-emeriti/">Faculty Emeriti</a></li>
</ul></div></section>
<h2>Full-time Faculty</h2>

<div class="uri-people-tool cl-tiles halves"><div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/marco-alvarez/"></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">Assistant Professor | Director of Graduate Studies</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5009</span> - <a class="u-email" href="mailto:malvarez@uri.edu">malvarez@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card">
<header>
<div class="header">
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/samantha-armentini/">Samantha Armentini</a></h3>
</div>
</header>
```

```
<div class="inside">
  <p class="people-title p-job-title">Lecturer</p>
  <p class="people-department">Computer Science</p>
  <p class="people-misc"><a class="u-email" href="mailto:sarmenti@uri.edu">asarmenti@uri.edu</a></p>
  <div style="clear:both;"></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" loading="lazy" sizes="(max-width: 300px) 100vw, 300px">
          src="https://web.uri.edu/cs/files/Sarah-Brown.png"
          srcset="https://web.uri.edu/cs/files/Sarah-Brown.png 300w,
          https://web.uri.edu/cs/files/Sarah-Brown-150x150.png 150w" width="300"/></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 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" loading="lazy" sizes="(max-width: 2560px) 100vw, 2560px">
              src="https://web.uri.edu/cs/files/mike-conti-scaled.jpg"
              srcset="https://web.uri.edu/cs/files/mike-conti-scaled.jpg 2560w,
              https://web.uri.edu/cs/files/mike-conti-300x290.jpg 300w,
              https://web.uri.edu/cs/files/mike-conti-1024x990.jpg 1024w,
              https://web.uri.edu/cs/files/mike-conti-768x743.jpg 768w,
              https://web.uri.edu/cs/files/mike-conti-1536x1485.jpg 1536w,
              https://web.uri.edu/cs/files/mike-conti-2048x1980.jpg 2048w,
              https://web.uri.edu/cs/files/mike-conti-364x352.jpg 364w,
              https://web.uri.edu/cs/files/mike-conti-500x483.jpg 500w,
              https://web.uri.edu/cs/files/mike-conti-1000x967.jpg 1000w,
              https://web.uri.edu/cs/files/mike-conti-1280x1238.jpg 1280w,
              https://web.uri.edu/cs/files/mike-conti-2000x1934.jpg 2000w" width="2560"/></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">Lecturer</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 class="peopleitem h-card has-thumbnail">
      <header>
        <div class="header">
          <figure>
            <a href="https://web.uri.edu/cs/meet/noah-daniels/"><img alt="" class="u-photo wp-post-image" height="219" loading="lazy" sizes="(max-width: 219px) 100vw, 219px">
                src="https://web.uri.edu/cs/files/noah-daniels-web.png"
                srcset="https://web.uri.edu/cs/files/noah-daniels-web.png 219w,
                https://web.uri.edu/cs/files/noah-daniels-web-150x150.png 150w" width="219"/></a>
          </figure>
          <h3 class="p-name"><a href="https://web.uri.edu/cs/meet/noah-daniels/">Noah Daniels</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:noah_daniels@uri.edu">noah_daniels@uri.edu</a></p>
        <div style="clear:both;"></div>
      </div>
      <div class="peopleitem h-card has-thumbnail">
        <header>
          <div class="header">
            <figure>
              <a href="https://web.uri.edu/cs/meet/lisa-dipippo/"></a>
            </figure>
            <h3 class="p-name"><a href="https://web.uri.edu/cs/meet/lisa-dipippo/">Lisa DiPippo</a></h3>
          </div>
        </header>
        <div class="inside">
          <p class="people-title p-job-title">Professor | Chair</p>
          <p class="people-department">Computer Science</p>
          <p class="people-misc"><a class="u-email" href="mailto:ldipippo@uri.edu">ldipippo@uri.edu</a></p>
          <div style="clear:both;"></div>
        </div>
        <div class="peopleitem h-card has-thumbnail">
          <header>
            <div class="header">
              <figure>
                <a href="https://web.uri.edu/cs/meet/victor-fay-wolfe/"></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:wolfe@cs.uri.edu">wolfe@cs.uri.edu</a></p>
            <div style="clear:both;"></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">lutzhamel@uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Abdeljawab Hendawi</h3>

</div>

</header>

<div class="inside">

<p class="people-title p-job-title">Assistant Professor</p>

<p class="people-department">Data Science | Computer Sciences</p>

<p class="people-misc">401.874.5738 - hendawi@uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Jean-Yves Hervé</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">jyh@cs.uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Natalia Katenka</h3>

</div>

</header>

<div class="inside">

<p class="people-title p-job-title">Associate Professor | Director of Undergraduate Studies</p>

<p class="people-department">Statistics</p>

<p class="people-misc">nkatenka@uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card">

<header>

<div class="header">

<h3 class="p-name">Soheyb Kouider</h3>

</div>

</header>

<div class="inside">

<p class="people-title p-job-title">Lecturer</p>

<p class="people-department">Statistics</p>

<p class="people-misc">401.874.2562 - soheyb@uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Edmund Lamagna</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">eal@cs.uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Indrani Mandal</h3>

</div>

</header>

<div class="inside">

<p class="people-title p-job-title">Lecturer</p>

<p class="people-department">Computer Science</p>

<p class="people-misc">indrani_mandal@uri.edu </p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Gavino Puggioni</h3>

</div>

</header>

<div class="inside">

<p class="people-title p-job-title">Associate Professor | Statistics Section Head | Director of Graduate Studies</p>

<p class="people-department">Statistics</p>

<p class="people-misc">401.874.4388 - guggioni@uri.edu</p>

<div style="clear:both;"></div>

</div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Krishna Venkatasubramanian</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">kris@uri.edu</p>

<div style="clear:both;"></div>

</div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Jing Wu</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">401.874.4504 - jing_wu@uri.edu</p>

<div style="clear:both;"></div>

</div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Yichi Zhang</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">yichizhang@uri.edu</p>

<div style="clear:both;"></div>

</div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Guangyu Zhu</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">guangyuzhu@uri.edu</p>

<div style="clear:both;"></div>

</div>

```
</div>
<p>
<h2>Adjunct Faculty and Limited Joint Appointments</h2>
</p>
<div class="uri-people-tool cl-tiles halves"><div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/ashley-buchanan/"></a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/ashley-buchanan/">Ashley Buchanan</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Limited Joint Appointment</p>
<p class="people-department">Biostatistics</p>
<p class="people-misc"><span class="p-tel">401.874.4739</span> - <a class="u-email" href="mailto:buchanan@uri.edu">buchanan@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/nina-kajiji/"></a>
</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 class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/rachel-schwartz/"></a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/rachel-schwartz/">Rachel Schwartz</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor - Limited Joint Appointment</p>
<p class="people-department">Biological Sciences</p>
<p class="people-misc"><span class="p-tel">401.874.5404</span> - <a class="u-email" href="mailto:rsschwartz@uri.edu">rsschwartz@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/ying-zhang/"></a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/ying-zhang/">Ying Zhang</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor - Limited Joint Appointment</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.4915</span> - <a class="u-email" href="mailto:yingzhang@uri.edu">yingzhang@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<p>
</p></div><!-- .entry-content -->
</article><!-- #post-## -->
</main><!-- #main -->
</div><!-- #content -->
<div id="actionbar-wrapper">
<div id="actionbar" role="menu">
<a href="https://www.uri.edu/connect" id="action-connect" role="menuitem"><span role="presentation" title="Learn more about URI: Get in touch"></span>Connect</a>
<a href="https://www.uri.edu/apply" id="action-apply" role="menuitem"><span role="presentation"></span>Apply</a><a href="https://www.uri.edu/tour" id="action-tour" role="menuitem"><span role="presentation"></span>Tour</a><a href="https://www.uri.edu/give" id="action-give" role="menuitem"><span role="presentation"></span>Give</a> </div>
</div><!-- #actionbar-wrapper -->
<footer id="globalfooter">
<div id="baselement">
<div id="storagebins">
<div id="sb-university">
<input checked="" id="sb-university-toggle" name="storagebin" role="presentation" type="radio" value="university"/>
<label aria-label="Open the University footer menu when browsing on mobile." for="sb-university-toggle"><span>University</span></label>
<ul aria-label="The University footer menu." role="menu">
<li><a href="https://www.uri.edu/about/leadership/" role="menuitem">Leadership</a></li>
<li><a href="https://web.uri.edu/diversity/" role="menuitem">Diversity and Inclusion</a></li>
<li><a href="https://web.uri.edu/global/" role="menuitem">Global</a></li>
<li><a href="https://web.uri.edu/about/campuses/" role="menuitem">Campuses</a></li>
<li><a href="https://www.uri.edu/safety/" role="menuitem">Safety</a></li>
</ul>
</div>
<div id="sb-campus-life">
```

```
<input id="sb-campus-life-toggle" name="storagebin" role="presentation" type="radio" value="campus-life"/>
<label aria-label="Open the Campus Life footer menu when browsing on mobile." for="sb-campus-life-toggle"><span>Campus Life</span></label>
<ul aria-label="The Campus Life footer menu." role="menu">
<li<a href="https://web.uri.edu/housing/" role="menuitem">Housing</a></li>
<li<a href="https://web.uri.edu/dining/" role="menuitem">Dining</a></li>
<li<a href="https://web.uri.edu/athletics/" role="menuitem">Athletics and Recreation</a></li>
<li<a href="https://www.uri.edu/campus-life/health-and-wellness/" role="menuitem">Health and Wellness</a></li>
<li<a href="https://events.uri.edu" role="menuitem">Events</a></li>
</ul>
</div>
<div id="sb-academics">
<input id="sb-academics-toggle" name="storagebin" role="presentation" type="radio" value="academics"/>
<label aria-label="Open the Academics footer menu when browsing on mobile." for="sb-academics-toggle"><span>Academics</span></label>
<ul aria-label="The Academics Footer menu." role="menu">
<li<a href="https://www.uri.edu/academics/" role="menuitem">Undergraduate</a></li>
<li<a href="https://web.uri.edu/graduate-school/" role="menuitem">Graduate</a></li>
<li<a href="https://web.uri.edu/advising/" role="menuitem">Advising</a></li>
<li<a href="https://web.uri.edu/library/" role="menuitem">Libraries</a></li>
<li<a href="https://web.uri.edu/career/students/" role="menuitem">Internships</a></li>
</ul>
</div>
</div>
<div id="gimmicks">
<!-- Tides Widget -->
<div="" class="uri-tides-widget darkmode" data-height="22" data-station="8454049"><span class="status"></span></div>
</div>
<!-- Social Media Component -->
<aside class="cl-wrapper cl-social-wrapper"><ul class="cl-social light"><li><a class="cl-social-facebook" href="https://www.facebook.com/universityofri" title="Facebook">Facebook</a></li><li><a class="cl-social-instagram" href="https://www.instagram.com/universityofri" title="Instagram">Instagram</a></li><li><a class="cl-social-twitter" href="https://twitter.com/universityofri" title="Twitter">Twitter</a></li><li><a class="cl-social-youtube" href="https://www.youtube.com/user/UniversityOfRI" title="YouTube">YouTube</a></li></ul></aside>
</div>
<div id="tagline"></div>
<div id="legal">
<p>Copyright © <a class="subtle" href="http://www.uri.edu/">University of Rhode Island</a> | University of Rhode Island, Kingston, RI 02881, USA | 1.401.874.1000</p>
<p>URI is an equal opportunity employer committed to the principles of affirmative action. <a class="jobs" href="https://jobs.uri.edu/">Work at URI</a></p>
</div>
</footer><!-- #globalfooter -->
</div><!-- #page -->
<link href="https://web.uri.edu/cs/wp-content/plugins/uri-tides-1.2/css/tides.css?ver=5.7.1" id="uri-tides-css-css" media="all" rel="stylesheet" type="text/css">
<script id="uricl-js-js" src="https://web.uri.edu/cs/wp-content/plugins/uri-component-library/js/cl.built.js?ver=20220721" type="text/javascript"></script>
<script id="wp-jquery-lightbox-js-extra" type="text/javascript">
/* <![CDATA[ */
var JQLBSettings = {
"fitToSpeed": 0, "resizeSpeed": "400", "displayDownloadLink": "0", "navbarOnTop": "0"
,"loopImages": "", "resizeCenter": "", "marginSize": "", "linkTarget": "", "help": "", "prevLinkTitle": "previous image", "nextLinkTitle": "next image", "prevLinkText": "\u2190 ab Previous", "nextLinkText": "\u2192 ab", "closeTitle": "Close image gallery", "image": "Image", "off": "of"
,"download": "Download", "jqlb_overlay_opacity": "80", "jqlb_overlay_color": "#000000"
,"jqlb_overlay_close": "1", "jqlb_border_width": "10", "jqlb_border_color": "#ffffff", "jqlb_border_radius": "0"
,"jqlb_image_info_bgcolor": "#000000", "jqlb_image_info_text_color": "#000000", "jqlb_image_info_text_fontsize": "10"
,"jqlb_show_text_for_image": "1", "jqlb_next_image_title": "next image", "jqlb_previous_image_title": "previous image"
,"jqlb_next_button_image": "https://\u2225/web.uri.edu/cs\wp-content/plugins\wp-lightbox-2\styles\images\next.gif", "jqlb_previous_button_image": "https://\u2225/web.uri.edu/cs\wp-content\wp-plugins\wp-lightbox-2\styles\images\prev.gif", "jqlb_maximum_width": "", "jqlb_maximum_height": "", "jqlb_show_close_button": "1", "jqlb_close_image_title": "Close image gallery", "jqlb_close_image_max_height": "22", "jqlb_image_for_close_lightbox": "https://\u2225/web.uri.edu/cs\wp-content\wp-plugins\wp-lightbox-2\styles\images\closelabel.gif", "jqlb_keyboard_navigation": "1", "jqlb_popup_size_fix": "0"
};
/* ] ] */
</script>
<script id="wp-jquery-lightbox-j-s" src="https://web.uri.edu/cs/wp-content/plugins/wp-lightbox-2/js/dist/wp-lightbox-2.min.js?ver=1.3.4.1" type="text/javascript"></script>
<script id="uri-modern-navigation-j-s" src="https://web.uri.edu/cs/wp-content/themes/uri-modern/js/navigation.js?ver=2.4.0" type="text/javascript"></script>
<script id="uri-modern-smoothscroll-j-s" src="https://web.uri.edu/cs/wp-content/themes/uri-modern/js/smoothscroll.min.js?ver=2.4.0" type="text/javascript"></script>
<script id="uri-modern-skip-link-focus-fix-j-s" src="https://web.uri.edu/cs/wp-content/themes/uri-modern/js/skip-link-focus-fix.js?ver=2.4.0" type="text/javascript"></script>
<script id="uri-modern-scripts-j-s-extra" type="text/javascript">
/* <![CDATA[ */
var URIMODERN = {"base": "https://\u2225/web.uri.edu/cs/", "path": {"page": "https://\u2225/web.uri.edu/cs\people/", "theme": "https://\u2225/web.uri.edu/cs\wp-content\themes\uri-modern", "themes": "https://\u2225/web.uri.edu/cs\wp-content\themes\uri-modern\js\skip-link-focus-fix.js?ver=2.4.0"
,"plugins": "https://\u2225/web.uri.edu/cs\wp-content\plugins"}, "theme": {"name": "URI Modern", "version": "2.4.0", "textDomain": "uri"}, "is": {""404": false, "childTheme": false, "admin": false}, "features": []};
/* ] ] */
</script>
<script id="uri-modern-scripts-j-s" src="https://web.uri.edu/cs/wp-content/themes/uri-modern/js/script.min.js?ver=2.4.0" type="text/javascript"></script>
<script id="page-links-to-j-s" src="https://\u2225/web.uri.edu/cs\wp-content\plugins\page-links-to\dist\new-tab.js?ver=3.3.5" type="text/javascript"></script>
<script id="wp-embed-j-s" src="https://\u2225/web.uri.edu/cs\wp-includes\js\wp-embed.min.js?ver=5.7.1" type="text/javascript"></script>
<script id="uri-tides-j-s-extra" type="text/javascript">
/* <![CDATA[ */
var tides = {"temperature": {"metadata": {"id": "8454049", "name": "Quonset Point", "lat": "41.5868", "lon": "-71.4109"}, "data": [{"t": "2022-07-21 22:42", "v": "76.5", "f": "0,0,0"}]}, "tide": {"predictions": [{"t": "2022-07-20 05:52", "v": "3.810", "type": "H"}, {"t": "2022-07-20 11:02", "v": "0.537", "type": "L"}, {"t": "2022-07-21 01:25", "v": "4.151", "type": "H"}, {"t": "2022-07-21 06:44", "v": "3.499", "type": "H"}, {"t": "2022-07-21 11:52", "v": "0.699", "type": "L"}, {"t": "2022-07-21 19:17", "v": "3.999", "type": "H"}, {"t": "2022-07-22 02:14", "v": "0.984", "type": "L"}, {"t": "2022-07-22 07:37", "v": "3.260", "type": "H"}, {"t": "2022-07-22 20:13", "v": "3.865", "type": "H"}, {"t": "2022-07-23 03:03", "v": "0.973", "type": "L"}, {"t": "2022-07-23 08:34", "v": "3.129", "type": "H"}, {"t": "2022-07-23 13:41", "v": "0.826", "type": "L"}, {"t": "2022-07-23
```

We can use `prettify` method to format it more nicely. This would add tabs even if there were none in the source code.

```

<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="utf-8"/>
<meta content="width=device-width, initial-scale=1" name="viewport"/>
<link href="http://gmpg.org/xfn/11" rel="profile"/>
<title>
    People - Department of Computer Science and Statistics
</title>
<meta content="max-image-preview:large" name="robots">
<link href="https://s.w.org" rel="dns-prefetch">
<link href="https://web.uri.edu/cs/feed/" rel="alternate" title="Department of Computer Science and Statistics » Feed" type="application/rss+xml"/>
<link href="https://web.uri.edu/cs/comments/feed/" rel="alternate" title="Department of Computer Science and Statistics » Comments Feed" type="application/rss+xml"/>
<script type="text/javascript">
    window._wpemojiSettings =
        {"baseUrl": "https://s.w.org/images/core/emoji/13.0.1/72x72/", "ext": ".png",
         "svgUrl": "https://s.w.org/images/core/emoji/13.0.1/svg/", "svgExt": ".svg",
         "source": {"concatemoji": "https://web.uri.edu/cs/wp-includes/js/wp-emoji-release.min.js?v=5.7.1"}};

        if(function(e,a,t){var
n,r,o,i=a.createElement("canvas"),p=i.getContext("2d");function
s(e,t){var
a=String.fromCharCode;p.clearRect(0,0,i.width,i.height),p.fillText(a.apply(this,e),
0,0);e=i.toDataURL();return
p.clearRect(0,0,i.width,i.height),p.fillText(a.apply(this,t),0,0),e==i.toDataURL()
}function c(e){var
t=a.createElement("script");t.src=e,t.defer=t.type="text/javascript",a.getElements
ByTagName("head")[0].appendChild(t)}for(o=Array("flag","emoji1"),t.supports=
{everything:!0,everythingExceptFlag:!0},r=0;r<o.length;r++)t.supports[o[r]]=functi
on(e){if(p||p.fillText)t.switch(p.textBaseline="top",p.font="600 32px
Arial",e)(case"flag":return s([127987,65039,8203,9895,65039])?1:[s([55356,56826,55356,56819],
[55356,56826,8203,55356,56819])&&s([55356,57332,56128,56423,56128,56418,56128,56418,
56128,56430,56128,56423,56128,56423,8203,56128,56418,8203,56128,56421,8203,56128,56430,8
203,56128,56423,8203,56128,56447]);case"emoji1":return!s([55357,56424,8205,55356,57212])?2:[s([55357,56424,8203,55356,57212])&return1]
(o[r]),t.supports.everything=t.supports.everything&&t.supports[o[r]],"flag"!=o[r]
&&
(t.supports.everythingExceptFlag=t.supports.everythingExceptFlag&&t.supports[o[r]])
);t.supports.everythingExceptFlag=t.supports.everythingExceptFlag&&t.supports.fl
a,g,t.DOMReady=!1,t.readyState=function(){t.DOMReady=!0},t.supports.everything||
(n=function(){t.readyStateCallback()},a.addEventListener("load",n,!1));
(a.addEventListener("DOMContentLoaded",n,!1),e.addEventListener("load",n,!1));
(e.attachEvent("onload",n),a.attachEvent("onreadystatechange",function()
("complete"==a.readyState&&t.readyStateCallback()),(n=t.source||{}).concatemoji?
(c.n.concatemoji):n._wpemoji&&n.twemoji&&(c.n.twemoji),c(n._wpemoji)))}
(window,document>window._wpemojiSettings);
    </script>
    <style type="text/css">
        img.wp-smiley,
        img.emoji {
            display: inline !important;
            border: none !important;
            box-shadow: none !important;
            height: 1em !important;
            width: 1em !important;
            margin: 0 .07em !important;
            vertical-align: -0.1em !important;
            background: none !important;
            padding: 0 !important;
        }
    </style>
    <link href="https://web.uri.edu/cs/wp-includes/css/dist/block-
library/style.min.css?ver=5.7.1" id="wp-block-library-css" media="all"
rel="stylesheet" type="text/css"/>
    <link href="https://web.uri.edu/cs/wp-content/plugins/uri-component-
library/css/cl-built.css?ver=5.0.2" id="uricl-css-css" media="all"
rel="stylesheet" type="text/css"/>
    <link href="https://web.uri.edu/cs/wp-content/plugins/uri-
courses/assets/courses.css?ver=5.7.1" id="uri-courses-styles-css" media="all"
rel="stylesheet" type="text/css"/>
    <link href="https://web.uri.edu/cs/wp-content/plugins/uri-people-
tool/assets/people.css?ver=5.7.1" id="uri-people-styles-css" media="all"
rel="stylesheet" type="text/css"/>
    <link href="https://web.uri.edu/cs/wp-content/plugins/wp-lightbox-
2/styles/lightbox.min.css?ver=1.3.4" id="wp-lightbox-2.min.css-css" media="all"
rel="stylesheet" type="text/css"/>

```

```
<link href="https://web.uri.edu/cs/wp-content/themes/uri-modern/style.css?ver=2.4.0" id="uri-modern-style-css" media="all" rel="stylesheet" type="text/css"/>
<link href="https://web.uri.edu/cs/wp-content/plugins/tablepress/css/default.min.css?ver=1.13" id="tablepress-default-css" media="all" rel="stylesheet" type="text/css"/>
<script id="jquery-core-js" src="https://web.uri.edu/cs/wp-includes/js/jquery/jquery.min.js?ver=3.5.1" type="text/javascript">
</script>
<script id="jquery-migrate-js" src="https://web.uri.edu/cs/wp-includes/js/jquery/jquery-migrate.min.js?ver=3.3.2" type="text/javascript">
</script>
<link href="https://web.uri.edu/cs/wp-json/" rel="https://api.w.org/"/>
<link href="https://web.uri.edu/cs/wp-json/wp/v2/pages/629" rel="alternate" type="application/json"/>
<link href="https://web.uri.edu/cs/xmlrpc.php?sd" rel="EditURI" title="RSD" type="application/rsd+xml"/>
<link href="https://web.uri.edu/cs/wp-includes/wlmanifest.xml" rel="wlmanifest" type="application/wlmanifest+xml"/>
<meta content="WordPress 5.7.1" name="generator">
<link href="https://web.uri.edu/cs/people/" rel="canonical"/>
<link href="https://web.uri.edu/cs/?p=629" rel="shortlink"/>
<link href="https://web.uri.edu/cs/wp-json/oembed/1.0/embed?url=https%3A%2F%2Fweb.uri.edu%2Fcs%2Fpeople%2F" rel="alternate" type="application/json+oembed"/>
<link href="https://web.uri.edu/cs/wp-json/oembed/1.0/embed?url=https%3A%2F%2Fweb.uri.edu%2Fcs%2Fpeople%2F&format=xml" rel="alternate" type="text/xml+oembed"/>
<meta content="People Full-time Faculty Adjunct Faculty and Limited Join Appointments" name="description"/>
<meta content="summary_large_image" name="twitter:card"/>
<meta content="@universityofrri" name="twitter:site"/>
<meta content="@universityofrri" name="twitter:creator"/>
<meta content="https://web.uri.edu/cs/people/" property="og:url"/>
<meta content="People" property="og:title"/>
<meta content="People Full-time Faculty Adjunct Faculty and Limited Join Appointments" property="og:description"/>
<meta content="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/logo-wordmark.png" property="og:image"/>
<script>
(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
new Date().getTime(),event:'gtm.js'});var f=d.createElementByTagName(s)[0],
j=d.createElement(s),dl=l?'dataLayer':'?&l='+l+'';j.async=true;j.src=
'https://www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.insertBefore(j,f);
})(window,document,'script','dataLayer','GTM-K5GL9W');
</script>
<style id="wp-custom-css" type="text/css">
.button-list .cl-button {
    margin-bottom: 1rem;
}
#calendar .date,#calendar-wrap header {
    text-align:center
}
#calendar {
    width:100%
}
#calendar a {
    color:#005eff;
    text-decoration:none
}
#calendar ul {
    list-style:none;
    padding:0;
    margin:0;
    width:100%
}
#calendar li {
    display:block;
    float:left;
    width:14.342%;
    padding:5px;
    box-sizing:border-box;
    border:1px solid #ccc;
    margin-right:-1px;
    margin-bottom:1px
}
#calendar ul.weekdays {
    height:40px;
    background:#002147
}
#calendar ul.weekdays li {
    text-align:center;
    text-transform:uppercase;
    line-height:1.2;
    border:none!important;
    padding:.75rem .5rem;
    color:#fff;
    font-size:.9rem
}
#calendar ul.days.mobile {
    display:none
}
#calendar .days li {
    height:15rem
}
#calendar .days.events-zero li {
    height:10rem
}
#calendar .days.events-one li {
    height:10.75rem
}
#calendar .days.events-two li {
    height:11.5rem
}
#calendar .days.events-three li {
    height:15rem
}
#calendar .date {
    margin-bottom:5px;
    padding:4px;
    color:#000;
    padding-right: 1rem;
    float:right;
}
#calendar .event {
    clear:both;
    display:block;
    font-size: 1rem;
    font-family: hind,arial,sans-serif;
    border-radius:30px;
    padding:.4rem .25rem .25rem 5rem;
    margin-top:5px;
    margin-bottom:5px;
    line-height:1.75;
    background:#e4f2f2;
    text-decoration:none;
    float:left;
    position:relative;
    width:calc(100% * 4.68)
}

```

```

.calendar .event.four-day{
    width:calc(76% * 4.68)
}
.calendar .event.mobile {
    display:none
}
.calendar .event.scratch {
    background:#E2D5B8;
}
.calendar .event.lego {
    background:#E1EDF8;
}
.calendar .event.datascience {
    background:#FCB97D;
}
.calendar .event.girlytech {
    background:#C2CFB2;
}
.calendar .event.games {
    background:#F79365;
}
.calendar .event.programming {
    background:#BCDCE0;
}
.calendar .event.webdesign {
    background:#E09FA2;
}
.calendar .event-desc {
    color:#666;
    margin:3px 0 7px;
    text-decoration:none;
    display:inline-block
}
.calendar .event-time {
    margin:3px 0 7px 5px;
    text-decoration:none;
    display:inline-block
}
.calendar .other-month {
    background:#f5f5f5;
    color:#666
}
.desc {
    display:none;
    background:#eee;
    box-shadow: 0 0 4px #999;
    padding:10px;
    color: #000;
}
.calendar .event-desc:hover+desc {
    display:block;
    position:absolute;
    z-index:2
}
.days+header h1 {
    padding-top:2rem;
    clear: left
}
@media(max-width:768px) {
    .calendar .event img,#calendar .other-month,#calendar .weekdays {
        display:none
    }
    .calendar .event {
        width:calc(100% - 2em);
        padding-left:20px;
        padding-right:10px
    }
    .calendar .event.mobile,#calendar ul.days.mobile {
        display:block
    }
    .calendar li {
        height:auto!important;
        border:1px solid #eddede;
        width:100%;
        padding:10px;
        margin-bottom:-1px
    }
    .calendar .date {
        float:none
    }
}
.peopleitem.has-thumbnail img {
    width: 200px;
}
</style>
<!-- Favicons -->
<link color="#005eff" href="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/safari-pinned-tab.svg" rel="mask-icon"/>
<link href="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/favicon.png" rel="icon" type="image/png"/>
<link href="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/apple-touch-icon.png" rel="apple-touch-icon"/>
<link href="https://web.uri.edu/cs/wp-content/themes/uri-modern/images/apple-touch-icon-180x180.png" rel="apple-touch-icon" sizes="180x180"/>
</meta>
</link>
</meta>
</head>
<body class="page-template-default page page-id-629 page-parent group-blog ln-people">
<noscript>
<iframe height="0" src="https://www.googletagmanager.com/ns.html?id=GTM-K5GL9W" style="display:none;visibility:hidden" width="0">
</iframe>
</noscript>
<div class="site" id="page">
<a class="skip-link screen-reader-text" href="#content">
    Skip to content
</a>
<div id="masthead">
<header class="site-header" id="brandbar" role="banner">
<div id="identity-print">
    
</div>
<div id="globalsearch" role="search">
    <input aria-label="Toggle visibility of the search box." id="gsform-toggle" type="checkbox" value="presentation" />
    <label for="gsform-toggle" id="gsform">
        <span>
            Search
        </span>
    </label>
    <form action="https://www.uri.edu/search" id="gs" method="get" name="global_general_search_form">
        <input name="cx" type="hidden" value="016863979916529535900:17qai8akniiu">
        <input name="cof" type="hidden" value="FORID:11"/>
        <label for="gs-query" id="gs-query-label">
            Searchbox
        </label>
    </form>
</div>
</header>
</div>

```

```

        </label>
        <input id="gs-query" name="q" placeholder="Search" role="searchbox"
type="text" value="" />
        <input class="searchsubmit" id="gs-submit" name="searchsubmit"
type="submit" value="Search" />
    </input>
</form>
</div>
<div id="globalbanner-wrapper">
<div id="globalbanner">
<a href="https://www.uri.edu/" title="University of Rhode Island">
    University of Rhode Island
</a>
</div>
<div id="gateways">
    <input aria-label="Open the audience gateways menu when browsing on
mobile" id="gateways-toggle" role="presentation" type="checkbox"/>
    <label for="gateways-toggle" id="gateways-label">
        <span>
            You
        </span>
    </label>
    <ul id="gateways-menu" role="menu">
        <li>
            <a href="https://www.uri.edu/gateway/future-students" role="menuitem">
                Future Students
            </a>
        </li>
        <li>
            <a href="https://www.uri.edu/gateway/students" role="menuitem">
                Students
            </a>
        </li>
        <li>
            <a href="https://www.uri.edu/gateway/faculty" role="menuitem">
                Faculty
            </a>
        </li>
        <li>
            <a href="https://www.uri.edu/gateway/staff" role="menuitem">
                Staff
            </a>
        </li>
        <li>
            <a href="https://www.uri.edu/gateway/families" role="menuitem">
                Parents and Families
            </a>
        </li>
        <li>
            <a href="https://www.uri.edu/gateway/alumni" role="menuitem">
                Alumni
            </a>
        </li>
        <li>
            <a href="https://www.uri.edu/gateway/community" role="menuitem">
                Community
            </a>
        </li>
    </ul>
</div>
</div>
</div>
</header>
<!-- #brandbar -->
<header id="siteheader">
<div class="light" id="sitebanner">
<div id="sb-backdrop">
<div id="sb-background-image" style="background-
image:url(<https://web.uri.edu/cs/files/cropped-Big-Data.jpg>)">
</div>
<div id="sb-screen">
</div>
</div>
<div id="sitebranding">
<div id="siteidentity">
<h1 class="site-title">
<a href="https://web.uri.edu/cs/" rel="home">
    Department of Computer Science and Statistics
</a>
</h1>
<h2 class="site-description">
    College of Arts and Sciences
</h2>
</div>
<div id="sitesocial">
</div>
</div>
<!-- #sitebranding -->
</div>
<!-- #sitebanner -->
<div class="content-width" id="navigation">
<nav aria-label="Breadcrumb" id="breadcrumbs">
<ol>
    <li>
        <a href="https://www.uri.edu/">
            URI
        </a>
    </li>
    <li>
        <a href="https://web.uri.edu/artsci">
            Arts and Sciences
        </a>
    </li>
    <li>
        <a href="https://web.uri.edu/cs">
            Department of Computer Science and Statistics
        </a>
    </li>
    <li aria-current="page">
        People
    </li>
</ol>
</nav>
<div id="localnav">
    <section class="cl-wrapper cl-menu-wrapper">
        <div class="cl-menu" data-name="Site Menu" data-show-title="0" id="cl-
localnav">
            <ul class="cl-menu-list cl-menu-list-no-js" id="menu-navigation">
                <li class="menu-item menu-item-type-custom menu-item-object-custom menu-
item-home menu-item-8243" id="menu-item-8243">
                    <a href="https://web.uri.edu/cs" title="
">
                        Home
                    </a>
                </li>
                <li class="menu-item menu-item-type-post_type menu-item-object-page menu-
item-8255" id="menu-item-8255">
                    <a href="https://web.uri.edu/cs/about/" title="
">
                        About
                    </a>
                </li>
            </ul>
        </div>
    </section>
</div>

```

```
</a>
</li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8806" id="menu-item-8806">
    <a href="https://web.uri.edu/cs/academics/">
        Academics
    </a>
</li>
<li class="menu-item menu-item-type-post_type menu-item-object-page current-menu-item page_item page-item-629 current_page_item menu-item-8257" id="menu-item-8257">
    <a aria-current="page" href="https://web.uri.edu/cs/people/" title="People"
    >
        People
    </a>
</li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-9661" id="menu-item-9661">
    <a href="https://web.uri.edu/cs/research/">
        Research
    </a>
</li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-10871" id="menu-item-10871">
    <a href="https://web.uri.edu/cs/news-and-events/">
        News and Events
    </a>
</li>
<li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8267" id="menu-item-8267">
    <a href="https://web.uri.edu/cs/contact/" title="Contact"
    >
        Contact
    </a>
</li>
</ul>
</div>
</section>
</div>
</div>
</header>
</div>
<div class="site-content" id="content">
<main class="site-main" id="main" role="main">
<article class="post-629 page type-page status-publish hentry" id="post-629">
<div class="entry-content">
    <h1>
        People
    </h1>
    <section class="cl-wrapper cl-menu-wrapper">
        <div class="cl-menu" data-name="people" data-show-title="0" id="">
            <ul class="cl-menu-list cl-menu-list-no-js" id="menu-people">
                <li class="menu-item menu-item-type-post_type menu-item-object-page current-menu-item page_item page-item-629 current_page_item menu-item-8737" id="menu-item-8737">
                    <a aria-current="page" href="https://web.uri.edu/cs/people/">
                        Faculty
                    </a>
                </li>
                <li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-8746" id="menu-item-8746">
                    <a href="https://web.uri.edu/cs/people/staff/">
                        Staff
                    </a>
                </li>
                <li class="menu-item menu-item-type-post_type menu-item-object-page menu-item-11844" id="menu-item-11844">
                    <a href="https://web.uri.edu/cs/people/faculty-emeriti/">
                        Faculty Emeriti
                    </a>
                </li>
            </ul>
        </div>
    </section>
    <h2>
        Full-time Faculty
    </h2>
    <div class="uri-people-tool cl-tiles halves">
        <div class="peopleitem h-card has-thumbnail">
            <header>
                <div class="header">
                    <figure>
                        <a href="https://web.uri.edu/cs/meet/marco-alvarez/">
                            
                        </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">
                    Assistant Professor | Director of Graduate Studies
                </p>
                <p class="people-department">
                    Computer Science
                </p>
                <p class="people-misc">
                    <span class="p-tei">
                        401.874.5009
                    </span>
                    -
                    <a class="u-email" href="mailto:malvarez@uri.edu">
                        malvarez@uri.edu
                    </a>
                </p>
                <div style="clear:both;">
                </div>
            </div>
        </div>
        <div class="peopleitem h-card">
            <header>
                <div class="header">
                    <h3 class="p-name">
                        <a href="https://web.uri.edu/cs/meet/samantha-armenti/">
                            Samantha Armenti
                        </a>
                    </h3>
                </div>
            </header>
            <div class="inside">
                <p class="people-title p-job-title">
                    Lecturer
                </p>
                <p class="people-department">
                    Computer Science
                </p>
            </div>
        </div>
    </div>

```

```
<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 class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/sarah-brown/">
    
</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/">
    
</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">
    Lecturer
</p>
<p class="people-department">
    Computer Science
</p>
<p class="people-misc">
<a class="u-email" href="mailto:michaelconti@uri.edu ">
    michaelconti@uri.edu
</a>
</p>
<div style="clear:both;">
</div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/noah-daniels/">
    
</a>
</figure>
<h3 class="p-name">
<a href="https://web.uri.edu/cs/meet/noah-daniels/">
    Noah Daniels
</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:noah_daniels@uri.edu">
    noah_daniels@uri.edu
</a>
</p>
<div style="clear:both;">
</div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/lisa-dipippo/">
    
</a>
</figure>
<h3 class="p-name">
```

```
<a href="https://web.uri.edu/cs/meet/lisa-dipippo/">
  Lisa DiPippo
</a>
</h3>
</div>
</header>
<div class="inside">
  <p class="people-title p-job-title">
    Professor | Chair
  </p>
  <p class="people-department">
    Computer Science
  </p>
  <p class="people-misc">
    <a class="u-email" href="mailto:ldipippo@uri.edu">
      ldipippo@uri.edu
    </a>
  </p>
  <div style="clear:both;">
  </div>
</div>
<div class="peopleitem h-card has-thumbnail">
  <header>
    <div class="header">
      <figure>
        <a href="https://web.uri.edu/cs/meet/victor-fay-wolfe/">
          
        </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:wolfe@cs.uri.edu">
        wolfe@cs.uri.edu
      </a>
    </p>
    <div style="clear:both;">
    </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 class="peopleitem h-card has-thumbnail">
      <header>
        <div class="header">
          <figure>
            <a href="https://web.uri.edu/cs/meet/abdelawab-hendawi/">
              
            </a>
          </figure>
          <h3 class="p-name">
            <a href="https://web.uri.edu/cs/meet/abdelawab-hendawi/">
              Abdelawab Hendawi
            </a>
          </h3>
        </div>
      </header>
      <div class="inside">
        <p class="people-title p-job-title">
          Assistant Professor
        </p>
        <p class="people-department">
          Data Science | Computer Science
        </p>
        <p class="people-misc">
          <span class="p-tel">
            401.874.5738
          </span>
          -
          <a class="u-email" href="mailto:hendawi@uri.edu">
            hendawi@uri.edu
          </a>
        </p>
        <div style="clear:both;">
        </div>
      </div>
      <div class="peopleitem h-card has-thumbnail">
        <header>
          <div class="header">
            <figure>
              <a href="https://web.uri.edu/cs/meet/jean-yves-herve/">
                
            </a>
          </figure>
        </div>
      </header>
      <div class="inside">
        <p class="people-title p-job-title">
          -
        </p>
        <p class="people-department">
          -
        </p>
        <p class="people-misc">
          -
        </p>
      </div>
    </div>
  </div>

```

```
</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>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/natallia-katenka/">

401.874.2562

</p>
<-
<a class="u-email" href="mailto:soheyb@uri.edu">
soheyb@uri.edu
</a>
</p>
<div style="clear:both;">
</div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/edmund-lamagna/">

```

```

        </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">
        Lecturer
    </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 class="peopleitem h-card has-thumbnail">
    <header>
        <div class="header">
            <figure>
                <a href="https://web.uri.edu/cs/meet/gavino-puggioni/">
                    
                </a>
            </figure>
            <h3 class="p-name">
                <a href="https://web.uri.edu/cs/meet/gavino-puggioni/">
                    Gavino Puggioni
                </a>
            </h3>
        </div>
    </header>
    <div class="inside">
        <p class="people-title p-job-title">
            Associate Professor | Statistics Section Head | Director of Graduate
        </p>
        <p class="people-department">
            Statistics
        </p>
        <p class="people-misc">
            <span class="p-tel">
                401.874.4388
            </span>
            -
            <a class="u-email" href="mailto:gppuggioni@uri.edu">
                gppuggioni@uri.edu
            </a>
        </p>
        <div style="clear:both;">
        </div>
    </div>
</div>
<div class="peopleitem h-card has-thumbnail">
    <header>
        <div class="header">
            <figure>
                <a href="https://web.uri.edu/cs/meet/krishna-venkatasubramanian/">
                    
                </a>
            </figure>
            <h3 class="p-name">
                <a href="https://web.uri.edu/cs/meet/krishna-venkatasubramanian/">
                    Krishna Venkatasubramanian
                </a>
            </h3>
        </div>
    </header>
    <div class="inside">
        <p class="people-title p-job-title">
            Assistant Professor
        </p>
        <p class="people-department">
            Computer Science
        </p>
        <p class="people-misc">
            <a class="u-email" href="mailto:krish@uri.edu">
                krish@uri.edu
            </a>
        </p>
        <div style="clear:both;">
        </div>
    </div>
</div>
<div class="peopleitem h-card has-thumbnail">
    <header>
        <div class="header">
            <figure>
                <a href="https://web.uri.edu/cs/meet/jing-wu/">
                    
                </a>
            </figure>
            <h3 class="p-name">
                <a href="https://web.uri.edu/cs/meet/jing-wu/">
                    Jing Wu
                </a>
            </h3>
        </div>
    </header>

```

```
</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">
<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/yichi-zhang/">
        
    </a>
</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">
<span class="p-tel">
    401.874.4504
</span>
<br/>
<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/">
        
    </a>
</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">
<span class="p-tel">
    401.874.4504
</span>
<br/>
<a class="u-email" href="mailto:guangyuzhu@uri.edu">
    guangyuzhu@uri.edu
</a>
</p>
<div style="clear:both;">
</div>
</div>
</div>
</div>
<p>
<h2>
    Adjunct Faculty and Limited Joint Appointments
</h2>
</p>
<div class="uri-people-tool cl-tiles halves">
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
    <a href="https://web.uri.edu/cs/meet/ashley-buchanan/">
        
    </a>
</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:ashley.buchanan@uri.edu">
    ashley.buchanan@uri.edu
</a>
</p>
<div style="clear:both;">
</div>
</div>
</div>
</div>
```

```
buchanan@uri.edu
</a>
</p>
<div style="clear:both;">
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<figure>
<a href="https://web.uri.edu/cs/meet/nina-kajiji/">

401.874.5404
</span>
<br/>
<a class="u-email" href="mailto:rsschwartz@uri.edu">
rsschwartz@uri.edu
</a>
</p>
<div style="clear:both;">
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<figure>
<a href="https://web.uri.edu/cs/meet/ying-zhang/">

401.874.4915
</span>
<br/>
<a class="u-email" href="mailto:yingzhang@uri.edu">
yingzhang@uri.edu
</a>
</p>
<div style="clear:both;">
</div>
</div>
</div>
<p>
</p>
</div>
<!-- .entry-content -->
</article>
<!-- #post-## -->
</main>
<!-- #main -->
</div>
<!-- #content -->
<div id="actionbar-wrapper">
<div id="actionbar" role="menu">
<a href="https://www.uri.edu/connect" id="action-connect" role="menuitem">

```

```
    Connect
  </a>
<a href="https://www.uri.edu/apply" id="action-apply" role="menuitem">
  <span role="presentation">
    </span>
  Apply
</a>
<a href="https://www.uri.edu/tour" id="action-tour" role="menuitem">
  <span role="presentation">
    </span>
  Tour
</a>
<a href="https://www.uri.edu/give" id="action-give" role="menuitem">
  <span role="presentation">
    </span>
  Give
</a>
</div>
</div>
<!-- #actionbar-wrapper -->
<footer id="globalfooter">
<div id="basement">
  <div id="storagebins">
    <div id="sb-university">
      <input checked="" id="sb-university-toggle" name="storagebin" type="radio" value="university"/>
      <label aria-label="Open the University footer menu when browsing on mobile." for="sb-university-toggle">
        <span>
          University
        </span>
      </label>
      <ul aria-label="The University footer menu." role="menu">
        <li>
          <a href="https://www.uri.edu/about/leadership/" role="menuitem">
            Leadership
          </a>
        </li>
        <li>
          <a href="https://web.uri.edu/diversity/" role="menuitem">
            Diversity and Inclusion
          </a>
        </li>
        <li>
          <a href="https://web.uri.edu/global/" role="menuitem">
            Global
          </a>
        </li>
        <li>
          <a href="https://web.uri.edu/about/campuses/" role="menuitem">
            Campuses
          </a>
        </li>
        <li>
          <a href="https://www.uri.edu/safety/" role="menuitem">
            Safety
          </a>
        </li>
      </ul>
    </div>
    <div id="sb-campus-life">
      <input id="sb-campus-life-toggle" name="storagebin" role="presentation" type="radio" value="campus-life"/>
      <label aria-label="Open the Campus Life footer menu when browsing on mobile." for="sb-campus-life-toggle">
        <span>
          Campus Life
        </span>
      </label>
      <ul aria-label="The Campus Life footer menu." role="menu">
        <li>
          <a href="https://web.uri.edu/housing/" role="menuitem">
            Housing
          </a>
        </li>
        <li>
          <a href="https://web.uri.edu/dining/" role="menuitem">
            Dining
          </a>
        </li>
        <li>
          <a href="https://www.uri.edu/athletics/" role="menuitem">
            Athletics and Recreation
          </a>
        </li>
        <li>
          <a href="https://www.uri.edu/campus-life/health-and-wellness/" role="menuitem">
            Health and Wellness
          </a>
        </li>
        <li>
          <a href="https://events.uri.edu" role="menuitem">
            Events
          </a>
        </li>
      </ul>
    </div>
    <div id="sb-academics">
      <input id="sb-academics-toggle" name="storagebin" role="presentation" type="radio" value="academics"/>
      <label aria-label="Open the Academics footer menu when browsing on mobile." for="sb-academics-toggle">
        <span>
          Academics
        </span>
      </label>
      <ul aria-label="The Academics footer menu." role="menu">
        <li>
          <a href="https://www.uri.edu/academics/" role="menuitem">
            Undergraduate
          </a>
        </li>
        <li>
          <a href="https://web.uri.edu/graduate-school/" role="menuitem">
            Graduate
          </a>
        </li>
        <li>
          <a href="https://web.uri.edu/advising/" role="menuitem">
            Advising
          </a>
        </li>
        <li>
          <a href="https://web.uri.edu/library/" role="menuitem">
            Libraries
          </a>
        </li>
        <li>
          <a href="https://web.uri.edu/career/students/" role="menuitem">
            Internships
          </a>
        </li>
      </ul>
    </div>
  </div>
</div>
```

```
</a>
</li>
</ul>
</div>
<div id="gimmicks">
<!-- Tides Widget -->
<div="" class="uri-tides-widget darkmode" data-height="22" data-
station="8454049">
<span class="status">
</span>
</div>
<hr/>
<!-- Social Media Component -->
<aside class="cl-wrapper cl-social-wrapper">
<ul class="cl-social light">
<li>
<a class="cl-social-facebook"
href="https://www.facebook.com/universityofri" title="Facebook">
    Facebook
</a>
</li>
<li>
<a class="cl-social-instagram"
href="https://www.instagram.com/universityofri/" title="Instagram">
    Instagram
</a>
</li>
<li>
<a class="cl-social-twitter" href="https://twitter.com/universityofri"
title="Twitter">
    Twitter
</a>
</li>
<li>
<a class="cl-social-youtube"
href="https://www.youtube.com/user/UniversityOfRI" title="YouTube">
    YouTube
</a>
</li>
</ul>
</aside>
</div>
<div id="tagline">
</div>
<div id="legal">
<p>
    Copyright ©
    <a class="subtle" href="http://www.uri.edu/">
        University of Rhode Island
    </a>
    | University of Rhode Island, Kingston, RI 02881, USA | 1.401.874.1000
</p>
<p>
    URI is an equal opportunity employer committed to the principles of
    affirmative action.
    <a class="jobs" href="https://jobs.uri.edu/">
        Work at URI
    </a>
</p>
</div>
</footer>
<!-- #globalfooter -->
</div>
<!-- #page -->
<link href="https://web.uri.edu/cs/wp-content/plugins/uri-tides-
1.2/css/tides.css?ver=5.7.1" id="uri-tides-css" media="all" rel="stylesheet"
type="text/css">
<script id="uri-cl-js-js" src="https://web.uri.edu/cs/wp-content/plugins/uri-
component-library/js/cl.built.js?ver=20220721" type="text/javascript">
</script>
<script id="wp-jquery-lightbox-js-extra" type="text/javascript">
/* <![CDATA[ */
var JQLBSettings =
{"fitToScreen": "0", "resizeSpeed": "400", "displayDownloadLink": "0", "navbarOnTop": "0"
, "loopImages": "0", "resizeCenter": "0", "marginSize": "0", "linkTarget": "0", "help": "0", "prev
LinkTitle": "previous image", "nextLinkTitle": "next image", "prevLinkText": "\u2022 Previous"
, "nextLinkText": "Next \u2022", "closeTitle": "close image
gallery", "image": "Image ", "of": " of
", "download": "Download", "jqlb_overlay_opacity": "80", "jqlb_overlay_color": "#000000"
, "jqlb_overlay_close": "1", "jqlb_border_width": "10", "jqlb_border_color": "#fffff
f", "jqlb_border_radius": "0", "jqlb_image_info_background_transparency": "100", "jqlb_imag
e_info_bg_color": "#ffffff", "jqlb_image_info_text_color": "#000000", "jqlb_image_info_
text_fontsize": "10", "jqlb_show_text_for_image": "1", "jqlb_next_image_title": "next
image", "jqlb_previous_image_title": "previous
image", "jqlb_next_button_image": "https://\u2022/web.uri.edu/cs\wp-
content\plugins\wp-lightbox-
2\styles\images\next.gif", "jqlb_previous_button_image": "https://\u2022/web.uri.edu\cs\wp-
content\wp-content\plugins\wp-lightbox-
2\styles\images\prev.gif", "jqlb_maximum_width": "", "jqlb_maximum_height": "", "jql
b_show_close_button": "1", "jqlb_close_image_title": "close image
gallery", "jqlb_close_image_max_height": "22", "jqlb_image_for_close_lightbox": "https:
//\u2022/web.uri.edu\cs\wp-content\plugins\wp-lightbox-
2\styles\images\closetlabel.gif", "jqlb_keyboard_navigation": "1", "jqlb_popup_size
_fix": "0"};
/* ]]> */
</script>
<script id="wp-jquery-lightbox-js" src="https://web.uri.edu/cs/wp-
content/plugins/wp-lightbox-2/js/dist/wp-lightbox-2.min.js?ver=1.3.4.1"
type="text/javascript">
</script>
<script id="uri-modern-navigation-js" src="https://web.uri.edu/cs/wp-
content/themes/uri-modern/js/navigation.js?ver=2.4.0" type="text/javascript">
</script>
<script id="uri-modern-smoothscroll-js" src="https://web.uri.edu/cs/wp-
content/themes/uri-modern/js/smoothscroll.min.js?ver=2.4.0"
type="text/javascript">
</script>
<script id="uri-modern-skip-link-focus-fix-js" src="https://web.uri.edu/cs/wp-
content/themes/uri-modern/js/skip-link-focus-fix.js?ver=2.4.0"
type="text/javascript">
</script>
<script id="uri-modern-scripts-js-extra" type="text/javascript">
/* <![CDATA[ */
var URIMODERN = {"base": "https://\u2022/web.uri.edu\cs", "path":
{/page/, "https://\u2022/web.uri.edu\cs\people/", "theme": "https://\u2022/web.uri.edu\cs\wp-
wp-content\themes\uri-modern", "themes": "https://\u2022/web.uri.edu\cs\wp-
content\themes", "plugins": "https://\u2022/web.uri.edu\cs\wp-
content\plugins"}, "theme": {"name": "URI
Modern", "version": "2.4.0", "textDomain": "uri"}, "is":
{"404": false, "childTheme": false, "admin": false}, "features": []};
/* ]]> */
</script>
<script id="uri-modern-scripts-js" src="https://web.uri.edu/cs/wp-
content/themes/uri-modern/js/script.min.js?ver=2.4.0" type="text/javascript">
</script>
<script id="page-links-to-js" src="https://web.uri.edu/cs/wp-
content/plugins/page-links-to/dist/new-tab.js?ver=3.3.5" type="text/javascript">
</script>
<script id="wp-embed-js" src="https://web.uri.edu/cs/wp-includes/js/wp-
```

It also makes the tags (structure in HTML) attributes.

```
cs_people.title
```

```
<title>People - Department of Computer Science and Statistics</title>
```

For tags that have multiple instances like the `<a>` tag that defines a link, it returns the first instance.

cs_people.a

14.4. Finding all instances of a tag

We can use `find_all` to make a list of all occurrences of a tag. For example, we could get all of the links from a page:

```
cs_people.find_all('a')
```

```
[<a class="skip-link screen-reader-text" href="#content">Skip to content</a>,
<a href="https://www.uri.edu/" title="University of Rhode Island"><div id="identity">University of Rhode Island</div</a>,
<a href="https://www.uri.edu/gateway/future-students" role="menuitem">Future Students</a>,
<a href="https://www.uri.edu/gateway/students" role="menuitem">Students</a>,
<a href="https://www.uri.edu/gateway/faculty" role="menuitem">Faculty</a>,
<a href="https://www.uri.edu/gateway/staff" role="menuitem">Staff</a>,
<a href="https://www.uri.edu/gateway/families" role="menuitem">Parents and Families</a>,
<a href="https://www.uri.edu/gateway/alumni" role="menuitem">Alumni</a>,
<a href="https://www.uri.edu/gateway/community" role="menuitem">Community</a>,
<a href="https://web.uri.edu/cs/" rel="home">
    Department of Computer Science and Statistics
</a>,
<a href="https://www.uri.edu/">URI</a>,
<a href="https://web.uri.edu/artscli">Arts and Sciences</a>,
<a href="https://web.uri.edu/cs">Department of Computer Science and Statistics</a>,
<a href="https://web.uri.edu/cs" title="Home">Home</a>,
<a href="https://web.uri.edu/cs/about/" title="About">About</a>,
<a href="https://web.uri.edu/cs/academics/">Academics</a>,
<a aria-current="page" href="https://web.uri.edu/cs/people/" title=">People">People</a>,
<a href="https://web.uri.edu/cs/research/">Research</a>,
<a href="https://web.uri.edu/cs/news-and-events/">News and Events</a>,
<a href="https://web.uri.edu/cs/contact/" title=">Contact">Contact</a>,
<a aria-current="page" href="https://web.uri.edu/cs/people/">Faculty</a>,
<a href="https://web.uri.edu/cs/people/staff/">Staff</a>,
<a href="https://web.uri.edu/cs/meet/marco-alvarez/">Faculty Emeriti</a>,
<a href="https://web.uri.edu/cs/files/marco-alvarez.png" alt="" class="u-photo wp-post-image" height="120" loading="lazy"
src="https://web.uri.edu/cs/files/marco-alvarez.png" width="120"/></a>,
<a href="https://web.uri.edu/cs/meet/marco-alvarez/">Marco Alvarez</a>,
<a class="u-email" href="mailto:malvarez@uri.edu">malvarez@uri.edu</a>,
<a href="https://web.uri.edu/cs/meet/samantha-armenti/">Samantha Armenti</a>,
<a class="u-email" href="mailto:sarmenti@uri.edu ">sarmenti@uri.edu </a>,
```

, Sarah Brown, bowsarah@uri.edu, , Michael Conti, michaelconti@uri.edu , , Noah Daniels, noah_daniels@uri.edu, , Lisa DiPippo, ldipippo@uri.edu, , Victor Fay Wolfe, wolfe@cs.uri.edu, Lutz Hamele, lutzhamel@uri.edu, , Abdeltawab Hendawi, hendawi@uri.edu, , Jean-Yves Hervé, jyh@cs.uri.edu, , Natalia Katenka, nakatenka@uri.edu, Soheyb Kouider, soheyb@uri.edu, , Edmund Lamagna, ea1@cs.uri.edu, , Indrani Mandal, indrani_mandal@uri.edu, , Gavino Puggioni, puggioni@uri.edu, , Krishna Venkatasubramanian, krish@uri.edu, , Jing Wu, jing_wu@uri.edu, , Yichi Zhang, yichizhang@uri.edu,

```

srcset="https://web.uri.edu/cs/files/Guangyu-Zhu-web.jpg 200w,
https://web.uri.edu/cs/files/Guangyu-Zhu-web-150x150.jpg 150w" width="200"/></a>,
<a href="https://web.uri.edu/cs/meet/guangyu-zhu/">Guangyu Zhu</a>,
<a class="u-email" href="mailto:guangyuzhu@uri.edu">guangyuzhu@uri.edu</a>,
<a href="https://web.uri.edu/cs/meet/ashley-buchanan/">

```

We noted before that each person's information is contained in a `div` tag with the `class = peopleitem, find_all` can also take values for attributes of a tag.

Attributes are the modifiers of a tag, in this case, a class is a label that defines formatting, and in this case, acts as metadata about what is in the div.

Scraping relies on the HTML code being well organized.

```
{ 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/marco-alvarez/"></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">Assistant Professor | Director of Graduate Studies</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5009</span> - <a class="u-email" href="mailto:malvarez@uri.edu">malvarez@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card">
<header>
<div class="header">
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/samantha-armenti/">Samantha Armenti</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Lecturer</p>
<p class="people-department">Computer Sciences</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/"></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">brownsarahm@uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Michael Conti</h3>

</div>

</header>

<div class="inside">

<p class="people-title p-job-title">Lecturer</p>

<p class="people-department">Computer Science</p>

<p class="people-misc">michaelconti@uri.edu </p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Noah Daniels</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">noah_daniels@uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Lisa DiPippo</h3>

</div>

</header>

<div class="inside">

<p class="people-title p-job-title">Professor | Chair</p>

<p class="people-department">Computer Science</p>

<p class="people-misc">ldipippo@uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Victor Fay-Wolfe</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">wolfe@cs.uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card">

<header>

<div class="header">

<h3 class="p-name">Lutz Hamel</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">lutzhamel@uri.edu</p>

<div style="clear:both;"></div>

</div>

<div class="peopleitem h-card has-thumbnail">

<header>

<div class="header">

<figure>

</figure>

<h3 class="p-name">Abdelatawb Hendawi</h3>

</div>

</header>

```
<div class="inside">
<p class="people-title p-job-title">Assistant Professor</p>
<p class="people-department">Data Science | Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5738</span> - <a class="u-email" href="mailto:hendawi@uri.edu">hendawi@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/jean-yves-herve/">
</a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/jean-yves-herve/">Jean-Yves Herve</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 class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/natalia-katenka/">
</a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/natalia-katenka/">Natalia Katenka</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Associate Professor | Director of Undergraduate Studies</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><a class="u-email" href="mailto:nkatenka@uri.edu">nkatenka@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
<div class="peopleitem h-card">
<header>
<div class="header">
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/soheyb-kouider/">Soheyb Kouider</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Lecturer</p>
<p class="people-department">Statistics</p>
<p class="people-misc"><span class="p-tel">401.874.2562</span> - <a class="u-email" href="mailto:soheyb@uri.edu">soheyb@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/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:ea@cs.uri.edu">ea@cs.uri.edu</a></p>
<div style="clear:both;"></div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/indrani-mandal/"></a>
</figure>
<h2 class="p-name"><a href="https://web.uri.edu/cs/meet/indrani-mandal/">Indrani Mandal</a></h2>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Lecturer</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 class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/gavino-puggioni/"></a>
</figure>
```

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

```

<div style="clear:both;"></div>
</div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/nina-kajiji/"></a>
</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/"></a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/rachel-schwartz/">Rachel Schwartz</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor - Limited Joint Appointment</p>
<p class="people-department">Biological Sciences</p>
<p class="people-misc"><span class="p-tel">401.874.5404</span> - <a class="u-email" href="mailto:rsschwartz@uri.edu">rsschwartz@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/ying-zhang/"></a>
</figure>
<h3 class="p-name"><a href="https://web.uri.edu/cs/meet/ying-zhang/">Ying Zhang</a></h3>
</div>
</header>
<div class="inside">
<p class="people-title p-job-title">Assistant Professor - Limited Joint Appointment</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.4915</span> - <a class="u-email" href="mailto:yingzhang@uri.edu">yingzhang@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>

```

We could use this to see how many people there are

```

len(cs_people.find_all("div", "peopleitem"))

```

23

or use multiple to see how many people have thumbnail

```

len(cs_people.find_all("div", {"has-thumbnail"}))

```

20

14.5. Finding data we can make tabular

We can look at the first one in detail to determine what to extract for each of the column.

```

cs_people.find_all("div", "peopleitem")[0]

```

```

<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/marco-alvarez/"></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">Assistant Professor | Director of Graduate Studies</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5009</span> - <a class="u-email" href="mailto:malvarez@uri.edu">malvarez@uri.edu</a></p>
<div style="clear:both;"></div>
</div>

```

We can see that the name is an `<h3>` tag with `class = "p-name"`

```

first_name = cs_people.find_all("h3", "p-name")[0]

```

We can examine this using our typical tools:

```

type(first_name)

```

```
bs4.element.Tag
```

It has attributes, since it's a tag object:

```
first_name.contents
```

```
[<a href="https://web.uri.edu/cs/meet/marco-alvarez/">Marco Alvarez</a>]
```

What we want is the string:

```
first_name.string
```

```
'Marco Alvarez'
```

Question in class

How can we extract the link?

That's a child, because the `<a>` tag is inside of the the `<h3>` tag, so let's explore the children.

```
[type(c) for c in first_name.children]
```

```
[bs4.element.Tag]
```

Alternatively, we can pick the `a` tag out by name.

```
first_name.a
```

```
<a href="https://web.uri.edu/cs/meet/marco-alvarez/">Marco Alvarez</a>
```

the url or `href` is an attribute of the `a` tag.

```
first_name.a.attrs
```

```
{'href': 'https://web.uri.edu/cs/meet/marco-alvarez/'}
```

```
first_name.a.attrs.href
```

```
-----  
AttributeError                                     Traceback (most recent call last)  
Input In [22], in <cell line: 1>()  
----> 1 first_name.a.attrs.href  
  
AttributeError: 'dict' object has no attribute 'href'
```

we can get it out using the `[]` to index into the `attrs` dictionary

```
first_name.a.attrs['href']
```

```
'https://web.uri.edu/cs/meet/marco-alvarez/'
```

14.6. Building a DataFrame

Now that we know what to look for, we can start building. First, we'll find all of the names and extract the string from each using a list comprehension.

```
names = [name.string for name in cs_people.find_all("h3", "p-name")]  
names
```

```
['Marco Alvarez',  
'Samantha Armenti',  
'Sarah Brown',  
'Michael Conti',  
'Noah Daniels',  
'Lisa DiPippo',  
'Victor Fay-Wolfe',  
'Lutz Hamel',  
'Abdeltawab Hendawi',  
'Jean-Yves Hervé',  
'Natalia Katzenka',  
'Soheyb Kouider',  
'Edmund Lamagna',  
'Indrani Mandal',  
'Gavino Puggioni',  
'Krishna Venkatasubramanian',  
'Jing Wu',  
'Yichi Zhang',  
'Guangyu Zhu',  
'Ashley Buchanan',  
'Nina Kajiji',  
'Rachel Schwartz',  
'Ying Zhang']
```

We can use the same process for each other attribute we want. First, we'll look at the whole peopleitem again, and then decide what we want.

```
cs_people.find_all("div", "peopleitem")[0]
```

```

<div class="peopleitem h-card has-thumbnail">
<header>
<div class="header">
<figure>
<a href="https://web.uri.edu/cs/meet/marco-alvarez/"></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">Assistant Professor | Director of Graduate
Studies</p>
<p class="people-department">Computer Science</p>
<p class="people-misc"><span class="p-tel">401.874.5009</span> - <a class="u-
email" href="mailto:malvarez@uri.edu">malvarez@uri.edu</a></p>
<div style="clear:both;"></div>
</div>
</div>

```

We'll extract the department, title, and e-mail.

```

disciplines = [d.string for d in cs_people.find_all("p",
                                                    'people-department')]
titles = [t.string for t in cs_people.find_all("p", "people-title")]
emails = [e.string for e in cs_people.find_all("a", 'u-email')]
pd.DataFrame({'name':names, 'title':titles,
               'e-mails':emails, 'discipline':disciplines})

```

		name	title	e-mails	discipline
0		Marco Alvarez	Assistant Professor Director of Graduate Stu...	malvarez@uri.edu	Computer Science
1		Samantha Armenti	Lecturer	sarmenti@uri.edu	Computer Science
2		Sarah Brown	Assistant Professor	brownsarahm@uri.edu	Computer Science
3		Michael Conti	Lecturer	michaelconti@uri.edu	Computer Science
4		Noah Daniels	Assistant Professor	noah_daniels@uri.edu	Computer Science
5		Lisa DiPippo	Professor Chair	ldipippo@uri.edu	Computer Science
6		Victor Fay-Wolfe	Professor	wolfe@cs.uri.edu	Computer Science
7		Lutz Hamel	Associate Professor	lutzhamel@uri.edu	Computer Science
8		Abdeltawab Hendawi	Assistant Professor	hendawi@uri.edu	Data Science Computer Science
9		Jean-Yves Hervé	Associate Professor	jyh@cs.uri.edu	Computer Science
10		Natalia Katenka	Associate Professor Director of Undergraduat...	nkatenka@uri.edu	Statistics
11		Soheyb Kouider	Lecturer	soheyb@uri.edu	Statistics
12		Edmund Lamagna	Professor	eal@cs.uri.edu	Computer Science
13		Indrani Mandal	Lecturer	indrani_mandal@uri.edu	Computer Science
14		Gavino Puggioni	Associate Professor Statistics Section Head...	gpuggioni@uri.edu	Statistics
15		Krishna Venkatasubramanian	Assistant Professor	krish@uri.edu	Computer Science
16		Jing Wu	Assistant Professor	jing_wu@uri.edu	Statistics
17		Yichi Zhang	Assistant Professor	yichizhang@uri.edu	Statistics
18		Guangyu Zhu	Assistant Professor	guangyuzhu@uri.edu	Statistics
19		Ashley Buchanan	Limited Joint Appointment	buchanan@uri.edu	Biostatistics
20		Nina Kajiji	Adjunct Associate Professor	nina@uri.edu	Computer Science
21		Rachel Schwartz	Assistant Professor – Limited Joint Appointment	rsschwarz@uri.edu	Biological Sciences
22		Ying Zhang	Assistant Professor – Limited Joint Appointment	yingzhang@uri.edu	Computer Science

```

sp22_csc_sta_url =
'https://raw.githubusercontent.com/rhodyprog4ds/rhodyds/main/data/reg_CSCSTA_courses.csv'

```

```

courses_df = pd.read_csv(sp22_csc_sta_url)
courses_df.head()

```

	Subject	Cat#	Component	Section	GenEd	Title	Max Size	Campus	Acad Org	Class Stat	Course Topic	Instr Name	Instr Name 2	Instr Name 3
0	CSC	101	LEC	1	GE	Computing Concepts	35	URI	COMP SCIEN	A	NaN	Fay-Wolfe,Victor	NaN	NaN
1	CSC	101	LEC	2	GE	Computing Concepts	35	ONLIN	COMP SCIEN	A	NaN	Fay-Wolfe,Victor	NaN	NaN
2	CSC	101	LEC	3	GE	Computing Concepts	35	ONLIN	COMP SCIEN	A	NaN	Staff	NaN	NaN
3	CSC	101	LEC	L01	GE	Computing Concepts	35	ONLIN	COMP SCIEN	A	NaN	Fay-Wolfe,Victor	NaN	NaN
4	CSC	104	LEC	1	GE	Puzzles+Games=Analytical Think	40	URI	COMP SCIEN	A	NaN	Mandal,Indrani	NaN	NaN

We saw the `attrs` for a link above, where there was only one attribute on the tag, but for example, the images on each person's card have many attributes.

```
{ cs_people.find_all("div", "peopleitem")[0].img.attrs }
```

```
{"width": '120',
'height': '120',
'src': 'https://web.uri.edu/cs/files/marco-alvarez.png',
'class': ['u-photo', 'wp-post-image'],
'alt': '',
'loading': 'lazy'}
```

```
{ cs_people.find_all("p") }
```

```

[<p class="people-title p-job-title">Assistant Professor | Director of Graduate  

Studies</p>,
<p class="people-department">Computer Sciences</p>,
<p class="people-misc"><span class="p-tel">+401.874.5009</span> - <a class="u-  

email" href="mailto:malvarez@uri.edu">malvarez@uri.edu</a></p>,
<p class="people-title p-job-title">Lecturer</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>,
<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:brownssarahm@uri.edu">brownssarahm@uri.edu</a></p>,
<p class="people-title p-job-title">Lecturer</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>,
<p class="people-title p-job-title">Assistant Professor</p>,
<p class="people-department">Computer Science</p>,
<p class="people-misc"><a class="u-email" href="mailto:noah_daniels@uri.edu">noah_daniels@uri.edu</a></p>,
<p class="people-title p-job-title">Professor | Chair</p>,
<p class="people-department">Computer Science</p>,
<p class="people-misc"><a class="u-email" href="mailto:idipippo@uri.edu">idipippo@uri.edu</a></p>,
<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:wolfe@cs.uri.edu">wolfe@cs.uri.edu</a></p>,
<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>,
<p class="people-title p-job-title">Assistant Professor</p>,
<p class="people-department">Data Science | Computer Science</p>,
<p class="people-misc"><span class="p-tel">+401.874.5738</span> - <a class="u-  

email" href="mailto:hendawi@uri.edu">hendawi@uri.edu</a></p>,
<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>,
<p class="people-title p-job-title">Associate Professor | Director of  

Undergraduate Studies</p>,
<p class="people-department">Statistics</p>,
<p class="people-misc"><a class="u-email" href="mailto:nkatena@uri.edu">nkatena@uri.edu</a></p>,
<p class="people-title p-job-title">Lecturer</p>,
<p class="people-department">Statistics</p>,
<p class="people-misc"><span class="p-tel">+401.874.2562</span> - <a class="u-  

email" href="mailto:soheyb@uri.edu">soheyb@uri.edu</a></p>,
<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>,
<p class="people-title p-job-title">Lecturer</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>,
<p class="people-title p-job-title">Associate Professor | Statistics Section  

Head | Director of Graduate Studies</p>,
<p class="people-department">Statistics</p>,
<p class="people-misc"><span class="p-tel">+401.874.4388</span> - <a class="u-  

email" href="mailto:gpuggioni@uri.edu">gpuggioni@uri.edu</a></p>,
<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:krisjh@uri.edu">krisjh@uri.edu</a></p>,
<p class="people-title p-job-title">Assistant Professor</p>,
<p class="people-department">Statistics</p>,
<p class="people-misc"><span class="p-tel">+401.874.4504</span> - <a class="u-  

email" href="mailto:jing_wu@uri.edu">jing_wu@uri.edu</a></p>,
<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>,
<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>,
<p>
<h2>Adjunct Faculty and Limited Joint Appointments</h2>
</p>,
<p class="people-title p-job-title">Limited Joint Appointment</p>,
<p class="people-department">Biostatistics</p>,
<p class="people-misc"><span class="p-tel">+401.874.4739</span> - <a class="u-  

email" href="mailto:buchanan@uri.edu">buchanan@uri.edu</a></p>,
<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>,
<p class="people-title p-job-title">Assistant Professor - Limited Joint  

Appointment</p>,
<p class="people-department">Biological Sciences</p>,
<p class="people-misc"><span class="p-tel">+401.874.5404</span> - <a class="u-  

email" href="mailto:sschwartz@uri.edu">sschwartz@uri.edu</a></p>,
<p class="people-title p-job-title">Assistant Professor - Limited Joint  

Appointment</p>,
<p class="people-department">Computer Science</p>,
<p class="people-misc"><span class="p-tel">+401.874.4915</span> - <a class="u-  

email" href="mailto:yingzhang@uri.edu">yingzhang@uri.edu</a></p>,
<p>
</p>,
<p>Copyright © <a class="subtle" href="http://www.uri.edu/">University of Rhode  

Island</a> | University of Rhode Island, Kingston, RI 02881, USA |  

1.401.874.1000</p>,
<p>URI is an equal opportunity employer committed to the principles of  

affirmative action. <a class="jobs" href="https://jobs.uri.edu/">Work at URI</a>
</p>]

```

URI websites are probably formatted consistently, so we could build information about more departments.

- [csc/sta emeriti](#)
- [a&s dean's office](#)
- [math](#)
- [philosophy](#)
- [business](#)

14.7. Thinking Ahead

The spreadsheet of spring classes in the department is posted:

```
sp22_csc_stu_url = 'https://raw.githubusercontent.com/rhodyprog4ds/rhodyds/main/data/reg\_CSCSTA\_courses.csv'
```

this is a minimal copy where I removed enrollments and locations in case those change. this is derived from the last version the Dean asked us to make corrections to, so things will definitely be different before registration opens (eg I'm teaching the CSC392 and it's listed as "Staff")

sections have definitely been added/removed and teaching assignments changed, so don't use this for making plans.

How could you merge this with the DataFrame we just scraped?

14.8. More Practice

1. Add a phone number column
2. On the page linked from each person's name, their office number; add a column for office number.
3. Make the code we wrote in class into a function so that you can pass a page and get back a DataFrame.
4. Parse the [course descriptions](#) page and make a DataFrame with columns for subject code, course number, course title, and course description.
5. Merge the descriptions with the table about enrollments and instructors.

15. Intro to Machine learning

When we *do* machine learning, this can also be called:

- data mining
- pattern recognition
- modeling

because we are looking for patterns in the data and typically then planning to use those patterns to make predictions or automate a task.

Each of these terms does have slightly different meanings and usage, but sometimes they're used close to exchangeably.

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

Further Reading

this [paper](#) by Christopher M. Bishop, a leading 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.

15.1. What is a Model?

A model is a simplified representation of some part of the world. A famous quote about models is:

All models are wrong, but some are useful –[George Box](#)[^wiki]

Hint

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.

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](#)

Further Reading

read more in the [Model Based Machine Learning Book](#)

15.2. Models in Machine Learning

Starting from a dataset, we first make an additional designation about how we will use the different variables (columns). We will call most of them the *features*, which we denote mathematically by \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

$$y_i = f(\mathbf{x}_i)$$

15.3. 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](#)

15.4. 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**

15.5. Machine Learning Pipeline

To do machine learning we start with **training data** which we put as input to the **learning algorithm**. A learning algorithm might be a generic optimization procedure or a specialized procedure for a specific model. The learning algorithm outputs a trained **model** or the parameters of the model. When we deploy a model we pair the **fit model** with a **prediction algorithm** or **decision** algorithm to evaluate a new sample in the world.

In experimenting and design, we need **testing data** to evaluate how well our learning algorithm understood the world. We need to use previously unseen data, because if we don't we can't tell if the prediction algorithm is using a rule that the learning algorithm produced or just looking up from a lookup table the result. This can be thought of like the difference between memorization and understanding.

When the model does well on the training data, but not on test data, we say that it does not generalize well.

15.6. Machine Learning with Scikit Learn

```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
sns.set_theme(palette= "colorblind")
```

```
type(GaussianNB)
```

```
abc.ABCMeta
```

```
type(train_test_split)
```

```
function
```

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.

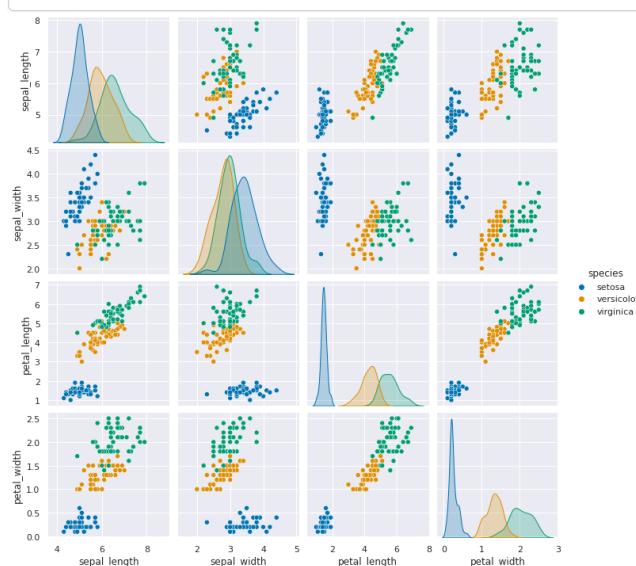
We'll load the data from Seaborn

```
iris_df = sns.load_dataset('iris')
```

First, we'll look at the data.

```
sns.pairplot(data=iris_df, hue='species')
```

```
<seaborn.axisgrid.PairGrid at 0x7f149d217d60>
```



We can see that this data is well suited for classification not only because in the world it makes sense, that we can use the measurements of flower petals to differentiate species, but also because the different species do not overlap too much. We call this **separable** data.

We'll pick out the target and features from this data frame

```
target = iris_df['species'].values
```

```
type(target)
```

```
numpy.ndarray
```

```
features = iris_df.values[:, :4]
```

```
features.shape
```

```
(150, 4)
```

Next, we use the sklearn function to create train and test data.

```
X_train, X_test, y_train, y_test = train_test_split(features,
```

```
target, random_state=0)
```

Try it Yourself

Try using a different random seed (value for `random_state`) or a different split size. Also try splitting by taking the first 80% of the data vs the last 20%. What happens?

We'll check what the split does by looking at the shape.

```
X_train.shape
```

```
(112, 4)
```

```
X_test.shape
```

Now we instantiate our classifier with the constructor method

```
gnb = GaussianNB()
gnb
```

GaussianNB
GaussianNB()

Try it Yourself

1. what other constructors have we used?
2. What happens if you skip this step?
3. Can you cascade this step?

This created an empty object, we can look at its contents:

```
gnb.__dict__
```

{'priors': None, 'var_smoothing': 1e-09}

We can fit our model, or learn the model parameters, with the `fit` method. The `fit` method implements the actual learning algorithm.

```
gnb.fit(X_train, y_train)
```

GaussianNB
GaussianNB()

Once we fit we can see it knows about our dataset now

```
gnb.__dict__
```

{'priors': None,
'var_smoothing': 1e-09,
'classes_': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
'n_features_in_': 4,
'epsilon_': 3.2135586734693877e-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 , 0.11242568, 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])}

We can apply our classifier with the `predict` method, which generates a prediction based on what it learned from the training data for each sample in the test data.

```
y_pred = gnb.predict(X_test)
```

y_pred

array(['virginica', 'versicolor', 'setosa', 'virginica', 'setosa',
'virginica', 'setosa', 'versicolor', 'versicolor', 'versicolor',
'versicolor', 'versicolor', 'versicolor', 'versicolor', 'setosa',
'setosa', 'virginica', 'versicolor', 'setosa', 'setosa',
'virginica', 'setosa', 'setosa', 'versicolor', 'versicolor',
'setosa', 'virginica', 'versicolor', 'setosa', 'virginica',
'virginica', 'versicolor', 'setosa', 'versicolor'], dtype='<U10')

We'll see better ways to evaluate on Friday, but as a first check, we can check if it matches

```
sum(y_pred == y_test)
```

38

and compare to how many.

```
len(y_test)
```

38

15.7. Questions at the End of Class

15.7.1. Clarifying Questions

15.7.1.1. Does this method work if the dataframe has more than one categorical variable

We can use categorical variables as features, Gaussian Naive Bayes assumes continuous valued features (in the model) but there are other classifiers that do not, including other types of Naive Bayes.

We could also, with a dataset with multiple categorical variables, build different classifiers, one to predict each categorical variable. For example, if our iris data also had a column with 'color' we could try to predict the color from the measurements. Since we know that the measurements predict the species, predicting color might only work if the different species come in different colors.

15.7.1.2. Were the train and test variables randomly generated from the library based off of the dataset we used?

The test and train variables were created by randomly subsetting the data that we had. For example, we could think of this as randomly permuting the samples and then choosing the first 75% as training and the rest as test, or we could think of it like creating randomly choosing 38 (.25*150, since there are 150 samples) numbers between 0 and 149 and masking and filtering.

Ram Token Opportunity

Submit code that uses numpy to illustrate one or both of these ways to create test train splits.

15.7.1.3. Is this method we learned today using linear regression?

No, we'll talk more about the model it does use on Friday and we'll cover linear regression in a couple of weeks.

15.7.2. Foreshadowing Questions

15.7.2.1. If we were to use a larger sample size for the test, would this improve results?

If we use more for the test set, we'd have less for training, so, in general, that would decrease performance. In this particular dataset, we can probably still do well with a small training set because the classes are separable.

Using more data for the test set, does mean we get a better estimate of the true test performance, however. We'll explore these type of tradeoffs over the rest of the course.

15.7.2.2. How robust/used is this specific machine learning algorithm? Is there more powerful open source ones we can play with

This specific model is not very robust, but it's a good starter. We will see more complex models which are also more robust in the future. There are many open source models that you can use. Most of the learning algorithms are, what's secret is typically the training data and some of the specific parameters provided to the learning algorithm. Scikit learn ([sklearn](#)) provides many. Other popular ones, for deep learning are tensorflow and keras.

15.7.3. Logistical Questions

15.7.3.1. Will we be scraping data from sites and try to implement machine learning?

Not in class, for the sake of time, but this would be a great portfolio exercise.

If you don't earn construct level 2 in assignment 5, you can also earn it on assignment 7 by doing this.

15.7.3.2. Will sklearn be used in assignments

Yes, we'll be using sklearn for the rest of the semester.

15.7.4. Context Questions

15.7.4.1. When did Google change to machine learning for their search queries?

The original page rank algorithm, focused mostly on ranking the pages by treating the internet like a graph. The major updates in around 2002-2003 changed how things worked a lot and started valuing other things. Valuing other things relies on the search history. For sure by 2010 when they added the text completion recommendations they were relying on previous queries at least equally, if not more than the graph structure of the Internet.

15.8. More Practice

See the try it yourself boxes above

16. Interpreting and Evaluating Naive Bayes

We'll pick up where we left off on Wednesday and we'll cover what the Naive Bayes model assumes and some more evaluation. Next week, we'll see a bit more about its predictions and see a new classifier.

```
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
iris_df = sns.load_dataset('iris')
sns.set_theme(palette= "colorblind")
```

Again we'll load the data and split it to test and train and indicate which variables to use as feature and the target.

```
feature_vars = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
target_var = 'species'
X_train, X_test, y_train, y_test = train_test_split(iris_df[feature_vars],
                                                    iris_df[target_var],
                                                    test_size=.5, random_state=0)
```

This time we also made the test size larger (50% instead of default 25%)

16.1. What does Naive Bayes do?

[docs](#)

Gaussian = features distributed according to the Gaussian Distribution (normal curve) Naive = independent features Bayes = most probable

[Bayes Estimator](#)

To see, first we'll fit the model, then examine its structure.

```
gnb= GaussianNB()
gnb.fit(X_train,y_train)
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_': 3.639903999999994e-09,
 'theta_': array([14.97586207, 3.35862069, 1.44827586, 0.23448276],
                [5.935 , 2.71  , 4.185 , 1.3  ],
                [6.77692308, 3.09230769, 5.73461538, 2.10769231]),
 'var_': array([[0.10321047, 0.13208066, 0.01629013, 0.00846612],
               [0.256275 , 0.0829 , 0.255275 , 0.046  ],
               [0.38869823, 0.10147929, 0.31303255, 0.04763314]]),
 'class_count_': array([29., 20., 26.]),
 'class_prior_': array([0.38666667, 0.26666667, 0.34666667])}

```

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.

💡 Try it Yourself

Could you use what we learned about EDA to find the mean and variance of each feature for each species of flower?

When we look at the data with a pair plot we see the distribution. This data is not perfectly Gaussian, but it's pretty close to unimodal (one peak) for each feature and they're relatively [normal curve](#) like (as opposed to a very sharp spike like a [laplacian](#))



Because the GaussianNB classifier calculates parameters that describe the assumed distribution of the data is is called a generative classifier. From a generative classifier, we can generate synthetic data that is from the distribution the classifier learned. If this data looks like our real data, then the model assumptions fit well.

⚠️ Warning

the details of this math are not required understanding, but this describes the following block of code

To do this, we extract the mean and variance parameters from the model (`gnb.theta_, gnb.sigma_`) and `zip` them together to create an iterable object that in each iteration returns one value from each list (`for th, sig in zip(gnb.theta_, gnb.sigma_)`). We do this inside of a list comprehension and for each `th, sig` where `th` is from `gnb.theta_` and `sig` is from `gnb.sigma_` we use `np.random.multivariate_normal` to get 20 samples. In a general [multivariate normal distribution](#) the second parameter is actually a covariance matrix. This describes both the variance of each individual feature and the correlation of the features. Since Naive Bayes is Naive it assumes the features are independent or have 0 correlation. So, to create the matrix from the vector of variances we multiply by `np.eye(4)` which is the identity matrix or a matrix with 1 on the diagonal and 0 elsewhere. Finally we stack the groups for each species together with `np.concatenate` (like `pd.concat` but works on numpy objects and `np.random.multivariate_normal` returns numpy arrays not data frames) and put all of that in a DataFrame using the feature names as the columns.

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 = 20
gnb_df = pd.DataFrame(np.concatenate([np.random.multivariate_normal(th,
sig*np.eye(4),N)
for th, sig in zip(gnb.theta_,gnb.sigma_)])
columns = gnb.feature_names_in_)
gnb_df['species'] = [c1 for c1 in [[c]*N for c in gnb.classes_] for c1 in c1]
sns.pairplot(data =gnb_df, hue='species')

```

💡 Further Reading

All of this is beyond the scope of this course, but may be of interest. The Scikit Learn [User Guide](#) is actually a really good place to learn the details of machine learning. It is high quality documentation from both a statistical and computer science perspective of every element of the library.

The `sklearn API` describes how the library is structured and organized. Because the library is so popular (and it's pretty well architected from a software perspective as well) if you are developing new machine learning techniques it's good to make them `sklearn` compatible.

For example, IBM's [AI360](#) is a package for doing fair machine learning which has a `sklearn` compatible interface (<https://ai360.readthedocs.io/en/latest/modules/sklearn.html>). Scikit Learn documentation also includes a [related projects](#) page.

💡 Tip

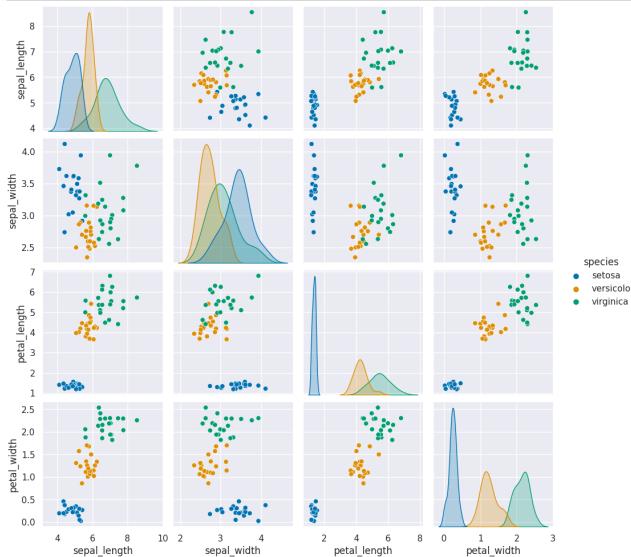
If you're interested in fair machine learning, let me know, this is what my research is

💡 Hint

to try understanding this block of code, try extracting pieces of it and running each piece individually. I built it up piece by piece and then wrapped them all together.

```
/opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/utils/deprecation.py:103: FutureWarning: Attribute `sigma_` was
deprecated in 1.0 and will be removed in 1.2. Use `var_` instead.
  warnings.warn(msg, category=FutureWarning)
```

```
<seaborn.axisgrid.PairGrid at 0x7f474832d700>
```



This one looks pretty close to the actual data. The biggest difference is that these data are all in uniformly circular-ish blobs and the ones above are not. That means that the naive assumption doesn't hold perfectly on this data.

16.2. Evaluating

On Wednesday, we used predict and checked the results

```
y_pred = gnb.predict(X_test)
y_pred
```

```
array(['virginica', 'versicolor', 'setosa', 'virginica', 'setosa',
       'virginica', 'setosa', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor', 'setosa',
       'versicolor', 'setosa', 'versicolor', 'versicolor', 'setosa',
       'setosa', 'virginica', 'versicolor', 'setosa', 'setosa',
       'virginica', 'setosa', 'setosa', 'versicolor', 'versicolor',
       'setosa', 'virginica', 'versicolor', 'setosa', 'virginica',
       'virginica', 'versicolor', 'setosa', 'versicolor', 'versicolor',
       'versicolor', 'virginica', 'setosa', 'virginica', 'setosa',
       'setosa', 'versicolor', 'virginica', 'virginica', 'versicolor',
       'virginica', 'versicolor', 'virginica', 'versicolor', 'versicolor',
       'virginica', 'versicolor', 'setosa', 'virginica', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'virginica', 'setosa',
       'setosa', 'virginica', 'versicolor', 'setosa', 'setosa',
       'versicolor'], dtype='<U10')
```

Then compared to see how many matched the ground truth.

```
sum(y_pred == y_test)
```

```
71
```

out of the total number

```
len(y_test)
```

```
75
```

Scikit Learn also provides a `score` method for all of the estimator object (different models). For a classifier, it provides the accuracy (percent correct).

```
gnb.score(X_test,y_test)
```

```
0.9466666666666666
```

which we could calculate with:

```
sum(y_pred == y_test)/len(y_test)
```

```
0.9466666666666666
```

The [confusion matrix](#) counts the number for each class (in this case the species) that truly have that value and that were predicted to have that value. The wikipedia article on [confusion matrices](#) also summarized all of the different metrics. Its written in terms of two, outcomes, but we have 3.

```
confusion_matrix(y_test, y_pred,labels = gnb.classes_)
```

```
array([[21,  0,  0],
       [ 0, 30,  0],
       [ 0,  4, 20]])
```

Further Reading

There are other classifiers that use roughly the same model but don't make the naive assumption. The more flexible is called Quadratic Discriminant Analysis.

- [mathematical formulation](#) for both
- [QDA code](#)
- [LDA code](#)

Tip

Note, we used the same random seed, but with a different test set size and got a different result because we have a different number of samples.

Try it yourself

Could you create a Data Frame with columns for predicted and true class, then get the count of how many samples were in each combination? Does it match the table.

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	21
versicolor	0.88	1.00	0.94	30
virginica	1.00	0.83	0.91	24
accuracy			0.95	75
macro avg	0.96	0.94	0.95	75
weighted avg	0.95	0.95	0.95	75

16.3. Questions

Ram token Opportunity

Contribute a question as an issue or contribute a solution to one of the try it yourself above.

17. Making Predictions in Generative Model

```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
iris_df = sns.load_dataset('iris')
```

We'll load the data again

```
iris_df.head(1)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa

Next we indicate the feature variables and target and split into test and train sets. We'll use 80% of the data for training.

```
feature_vars = ['sepal_length', 'sepal_width','petal_length', 'petal_width',]
target_var = 'species'

X_train, X_test, y_train, y_test = train_test_split(iris_df[feature_vars],
iris_df[target_var], train_size=.8, random_state=0)
```

We can confirm the shape is as expected

```
X_train.shape
```

```
(120, 4)
```

```
iris_df.shape
```

```
(150, 5)
```

```
.8*150
```

```
120.0
```

Next we again initialize and fit the classifier

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
GaussianNB()
GaussianNB()
```

We can compute predictions

```
y_pred = gnb.predict(X_test)
y_pred
```

```
array(['virginica', 'versicolor', 'setosa', 'virginica', 'setosa',
'virginica', 'setosa', 'versicolor', 'versicolor', 'versicolor',
'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor',
'versicolor', 'setosa', 'versicolor', 'setosa', 'setosa',
'verginica', 'setosa', 'setosa', 'versicolor', 'versicolor',
'setosa'], dtype='<U10')
```

And score the results

```
gnb.score(X_test,y_test)
```

```
0.9666666666666667
```

We saw last week that when we fit the Gaussian Naive Bayes, it computes a mean (θ) and variance (σ^2) and adds them to model parameters in attributes `gnb.theta_`, `gnb.sigma_`.

When we use the `predict` method, it uses those parameters to calculate the likelihood of the sample according to a Gaussian distribution (normal) for each class and then calculates the probability of the sample belonging to each class.

```
gnb.predict_proba(X_test)
```

```
array([[1.63380783e-232, 2.18878438e-006, 9.99997811e-001],
       [1.826404391e-082, 9.99998304e-001, 1.69618390e-006],
       [1.00000000e+000, 7.10250510e-019, 3.65449801e-028],
       [1.58508262e-305, 1.04649020e-006, 9.99998954e-001],
       [1.00000000e+000, 8.59168655e-017, 4.22159374e-027],
       [6.39815011e-321, 1.56450314e-010, 1.00000000e+000],
       [1.00000000e+000, 1.09797313e-016, 5.30276557e-027],
       [1.25122812e-146, 7.74052109e-001, 2.25947891e-001],
       [5.34357526e-159, 9.07564955e-001, 9.24359453e-002],
       [5.67261712e-093, 9.99882199e-001, 1.17891111e-004],
       [2.38651144e-210, 5.29669631e-001, 4.70399369e-001],
       [8.12047631e-132, 9.43762575e-001, 5.62374248e-002],
       [5.25177109e-132, 9.98864361e-001, 1.13563851e-003],
       [1.24498038e-139, 9.49838641e-001, 5.01613586e-002],
       [4.08232760e-140, 9.88043864e-001, 1.19561365e-002],
       [1.00000000e+000, 7.12837229e-019, 4.10162749e-029],
       [4.19553996e-131, 9.87944980e-001, 1.20550201e-002],
       [4.13286716e-111, 9.99942383e-001, 5.76167389e-005],
       [1.00000000e+000, 2.24933112e-015, 3.63624519e-026],
       [1.00000000e+000, 9.86750131e-016, 2.42355087e-025],
       [1.85930865e-186, 1.66966805e-002, 9.83303319e-001],
       [8.83060167e-130, 9.92757232e-001, 7.24276827e-003],
       [1.00000000e+000, 4.26389344e-013, 4.34222344e-023],
       [1.00000000e+000, 1.28045851e-016, 1.26708019e-027],
       [2.43739221e-168, 1.83516225e-001, 8.16483775e-001],
       [1.00000000e+000, 2.62431469e-018, 6.72573168e-029],
       [1.00000000e+000, 3.20605389e-011, 1.52433420e-020],
       [2.20964201e-110, 9.92921229e-001, 7.08771072e-004],
       [1.39297338e-046, 9.9999972e-001, 2.81392389e-008],
       [1.00000000e+000, 1.85943966e-013, 1.58833385e-023]])
```

These are hard to interpret as is, one option is to plot them

```
# make the probabilities into a dataframe labeled with classes & make the index a
separate column
prob_df = pd.DataFrame(data = gnb.predict_proba(X_test), columns = gnb.classes_
).reset_index()
# add the predictions
prob_df['predicted_species'] = y_pred
prob_df['true_species'] = y_test.values
# for plotting, make a column that combines the index & prediction
pred_text = lambda r: str( r['index'] ) + ',' + r['predicted_species']
prob_df['i,pred'] = prob_df.apply(pred_text, axis=1)
# same for ground truth
true_text = lambda r: str( r['index'] ) + ',' + r['true_species']
prob_df['correct'] = prob_df['predicted_species'] == prob_df['true_species']
# a dt 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'],value_vars =
gnb.classes_,
var_name = target_var, value_name = 'probability')
prob_df_melted.head()
```

index	predicted_species	true_species	i,pred	i,true	correct	species	probability
0	0	virginica	virginica	0,virginica	0,virginica	True	setosa 1.633808e-232
1	1	versicolor	versicolor	1,versicolor	1,versicolor	True	setosa 1.826404e-82
2	2	setosa	setosa	2,setosa	2,setosa	True	setosa 1.000000e+00
3	3	virginica	virginica	3,virginica	3,virginica	True	setosa 1.585083e-305
4	4	setosa	setosa	4,setosa	4,setosa	True	setosa 1.000000e+00

Now we have a data frame where each row is one the probability of one sample belonging to one class. So there's a total of `number_of_samples*number_of_classes` rows

```
prob_df_melted.shape
(90, 8)
len(y_pred)*len(gnb.classes_)
```

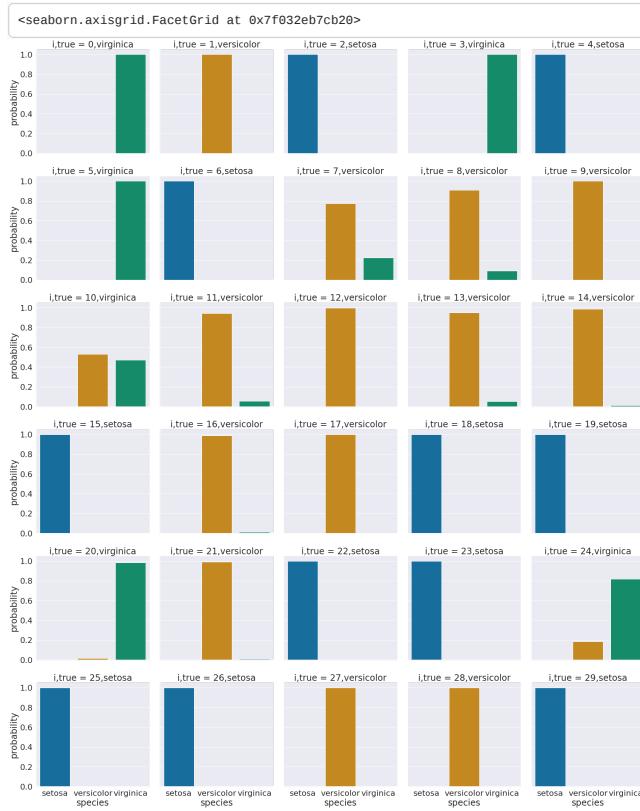
```
90
```

One way to look at these is to, for each sample in the test set, make a bar chart of the probability it belongs to each class. We added to the data frame information so that we can plot this with the true class in the title using `col = 'i,true'`

```
sns.set_theme(font_scale=2, palette="colorblind")
# plot a bar graph for each point labeled with the prediction
sns.catplot(data =prob_df_melted, x = 'species', y='probability' ,col ='i,true',
            col_wrap=5,kind='bar')
```

Tip

I used `set_theme` to change both the font size and the color palette. Seaborn has a [detailed guide](#) for choosing colors. The `colorblind` palette uses colors that are distinguishable under most common forms of color blindness.



We see that most samples have nearly all of their probability mass (all probabilities in a distribution sum to 1, or integrate if continuous) to 1, but a few samples are not.

👉 Try it yourself

Try adding a column that could change the headings to include an indicator of which are correct or not.

For now, we'll group and look at on average, what the distributions are for correct vs incorrect based on predictions.



We see that the errors were all for versicolor, and on average the distribution is very uncertain for those samples. Those samples are probably hard to distinguish. We could check by creating a data frame with the data and the information about predictions and correct values.

```

prob_data_df = pd.concat([prob_df,X_test.reset_index()],axis=1).drop(columns=['index'])
prob_data_df.head(2)

```

	setosa	versicolor	virginica	predicted_species	true_species	i,pred	correct	i,true	sepal_length	sepal_width	petal_length	petal_width	
0	1.633808e-232	0.000002	0.999998		virginica	virginica	0,virginica	True	0,virginica	5.8	2.8	5.1	2.4
1	1.826404e-82	0.999998	0.000002		versicolor	versicolor	1,versicolor	True	1,versicolor	6.0	2.2	4.0	1.0

feature_vars

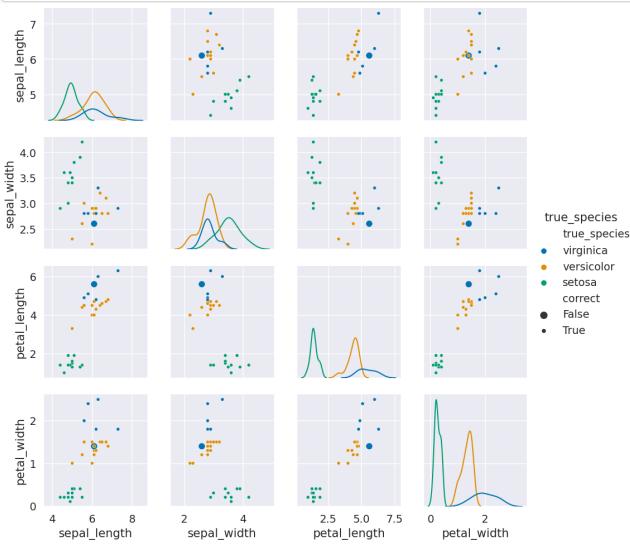
```
['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
```

```

g = sns.PairGrid(prob_data_df,x_vars=feature_vars,y_vars=
feature_vars,hue='true_species')
g.map_diag(sns.kdeplot)
g.map_offdiag(sns.scatterplot, size=prob_data_df["correct"])
g.add_legend()

```

<seaborn.axisgrid.PairGrid at 0x7f032df6ee0>



Here we see that the large dots (the incorrect ones) are all nearby to points of a different color. They were in fact samples that are similar to the other species. So again, this result makes sense and helps us see when classifiers that are a good fit for the data will still make mistakes.

We can also look at the probabilities of the predicted sample using max

```

p_predicted = np.max(gnb.predict_proba(X_test),axis=1)
p_predicted

```

```

array([0.99999781, 0.9999983 , 1.        , 0.99999895, 1.,
       1.        , 1.        , 0.77405211, 0.90756495, 0.99988211,
       0.52960963, 0.94376258, 0.99886436, 0.94983864, 0.98804386,
       1.        , 0.98794498, 0.99994238, 1.        , 1.        ,
       0.98330332, 0.99275723, 1.        , 1.        , 0.81648378,
       1.        , 1.        , 0.99929123, 0.99999997, 1.        ])

```

We see here that most of the predictions are pretty confident. We can also use the probabilities to then compute predictions and compare these to what the `predict` method gave, to confirm that this is how the predict method works.

```

pd.DataFrame(data = gnb.predict_proba(X_test), columns = gnb.classes_).
idxmax(axis=1) ==y_pred

```

```

0    True
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   True
28   True
29   True
dtype: bool

```

18. Midsemester feedback and Decision Trees

```

import pandas as pd
import seaborn as sns
import numpy as np
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
sns.set(palette='colorblind')

```

18.1. Feedback

Note

Analysis of the structured questions will be added

18.1.1. Key takeaways

- You're learning and happy with how much you're learning
- You've noticed and appreciate that the assignments build on what we cover in class
- You're reading the notes & like them a lot
- Assignment instructions are sometimes hard to understand

				Rank the following as what you feel the grading (when you do/not earn achievements) so far actually reflects about your performance in the class [How well I follow instructions]	Rank the following as what you feel the grading (when you do/not earn achievements) so far actually reflects about your performance in the class [What I understand about the material]	Rank the following as what you feel the grading (when you do/not earn achievements) so far actually reflects about your performance in the class [How much effort I put into assignments]	How fair do you think the amount each of the following is reflected in the grading [How well I follow instructions]	How fair do you think the amount each of the following is reflected in the grading [What I understand about the material]	How fair do you think the amount each of the following is reflected in the grading [How much effort I put into assignments]	Which of the following have you done to support your learning outside of class time?
How much do you think you've learned so far this semester?	How much of the material that's been taught do you feel you understand?	How do you think the achievements you've earned so far align with your understanding?								
0	4	4	I think the achievements underestimate what I ...	Reflected strongly in the grading	Reflected moderately in the grading	Reflected perfectly in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	read the notes online, reading the documentati...
1	5	4	I think they reflect my understanding well	Reflected strongly in the grading	Reflected perfectly in the grading	Reflected perfectly in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	read the notes online, experimenting with the ...
2	4	4	I think they reflect my understanding well	Reflected strongly in the grading	Reflected strongly in the grading	Reflected strongly in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	read the notes online, solving extra questions...
3	3	3	I think they reflect my understanding well	Reflected moderately in the grading	Reflected moderately in the grading	Reflected moderately in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	read the notes online, reading blogs or tutor...
4	3	3	I think the achievements overestimate what I k...	Reflected moderately in the grading	Reflected moderately in the grading	Reflected moderately in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	read the notes online, experimenting with the ...

First, we'll make the dataframe column names easier to work with and save a key of them as a dictionary that we then use to rename. First we look at them, then copy & paste them to make the dictionary.

```
feedback_df_raw.columns
Index(['How much do you think you\'ve learned so far this semester?',
       'How much of the material that\'s been taught do you feel you understand?',
       'How do you think the achievements you\'ve earned so far align with your understanding?',
       'Rank the following as what you feel the grading (when you do/not earn achievements) so far actually reflects about your performance in the class [How well I follow instructions]',
       'Rank the following as what you feel the grading (when you do/not earn achievements) so far actually reflects about your performance in the class [What I understand about the material]',
       'Rank the following as what you feel the grading (when you do/not earn achievements) so far actually reflects about your performance in the class [How much effort I put into assignments]',
       'How fair do you think the amount each of the following is reflected in the grading [How well I follow instructions]',
       'How fair do you think the amount each of the following is reflected in the grading [What I understand about the material]',
       'How fair do you think the amount each of the following is reflected in the grading [How much effort I put into assignments]',
       'Which of the following have you done to support your learning outside of class time?'],
      dtype='object')

short_names = {"How much do you think you've learned so far this semester?":'learned',
       "How much of the material that's been taught do you feel you understand?":'understand',
       "How do you think the achievements you've earned so far align with your understanding?":'achievements',
       "Rank the following as what you feel the grading (when you do/not earn achievements) so far actually reflects about your performance in the class [How well I follow instructions]":'grading_instructions',
       "Rank the following as what you feel the grading (when you do/not earn achievements) so far actually reflects about your performance in the class [What I understand about the material]":'grading_understanding',
       "Rank the following as what you feel the grading (when you do/not earn achievements) so far actually reflects about your performance in the class [How much effort I put into assignments]":'grading_effort',
       "How fair do you think the amount each of the following is reflected in the grading [How well I follow instructions]":'fairness_instructions',
       "How fair do you think the amount each of the following is reflected in the grading [What I understand about the material]":'fairness_understanding',
       "How fair do you think the amount each of the following is reflected in the grading [How much effort I put into assignments]":'fairness_effort',
       "Which of the following have you done to support your learning outside of class time?":'learning_activities'}
```

```
feedback_df_cols = feedback_df_raw.rename(columns = short_names)
feedback_df_cols.head()
```

learned	understand	achievements	grading_instructions	grading_understanding	grading_effort	fairness_instructions	fairness_understanding	fairness_effort	learning
0	4	4	I think the achievements underestimate what I ...	Reflected strongly in the grading	Reflected moderately in the grading	Reflected perfectly in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading
1	5	4	I think they reflect my understanding well	Reflected strongly in the grading	Reflected perfectly in the grading	Reflected perfectly in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading
2	4	4	I think they reflect my understanding well	Reflected strongly in the grading	Reflected strongly in the grading	Reflected strongly in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading
3	3	3	I think they reflect my understanding well	Reflected moderately in the grading	Reflected moderately in the grading	Reflected moderately in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading
4	3	3	I think the achievements overestimate what I ...	Reflected moderately in the grading	Reflected moderately in the grading	Reflected moderately in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading	Is fairly reflected in the grading

```
learning_lists = feedback_df_cols['learning_activities'].str.split(',')
learning_stacked = learning_lists.apply(pd.Series).stack()
learning_df = pd.get_dummies(learning_stacked).sum(level=0)
learning_df.head()
```

```
/tmp/ipykernel_2656/3116911913.py:3: FutureWarning: Using the level keyword in
DataFrame and Series aggregations is deprecated and will be removed in a future
version. Use groupby instead. df.sum(level=1) should use
df.groupby(level=1).sum().
learning_df = pd.get_dummies(learning_stacked).sum(level=0)
```

	attended Chamudi's office hours	attended Dr. Brown's office hours	attended Dr. Brown's office hours	download and run the notes	experimenting with the code from class	reading blogs or tutorials I find on my own	reading the documentation or course text	solving extra questions in the class notes	tinkering with code to answer other aspects of the material that I'm curious about	update the notes I took during class time	watching videos that I find on my own	read the notes online	reading the documentation or course text
0	0	0	0	0	0	1	1	0	0	0	0	1	0
1	0	0	0	0	1	0	1	0	0	0	1	1	0
2	1	0	0	0	0	0	0	1	0	0	0	1	0
3	0	0	0	0	0	1	0	0	0	0	0	1	0
4	0	0	0	0	1	0	1	0	0	0	1	1	0

```
feedback_df = pd.concat([feedback_df_cols,learning_df])
feedback_df.head(2)
```

learned understand achievements grading_instructions grading_understanding grading_effort fairness_instructions fairness_understanding fairness_effort learning

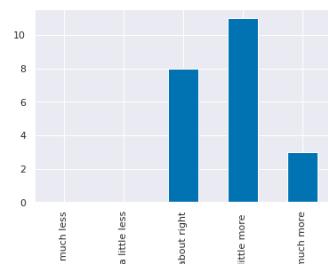
0	4.0	4.0	I think the achievements underestimate what I ...	Reflected strongly in the grading	Reflected moderately in the grading	Reflected perfectly in the grading	Is fairly reflected in the grading					
1	5.0	4.0	I think they reflect my understanding well	Reflected strongly in the grading	Reflected perfectly in the grading	Reflected perfectly in the grading	Is fairly reflected in the grading					

2 rows × 23 columns

```
el_meaning = {1: 'much less',
 2: 'a little less',
 3: 'about right',
 4: 'a little more ',
 5: 'much more'}

question_text = list(short_names.keys())
[list(short_names.values()).index('Learned')]
el_counts,_ = np.histogram(feedback_df['Learned'],bins = [i+.5 for i in range(6)])
el_df = pd.DataFrame(data = el_counts,index = el_meaning.values(),columns=[question_text,])

# el_df.rename_axis(index='amount relative to expectations',inplace=True)
el_df.plot.bar(legend=False);
# sns.displot(feedback_df['Learned'].replace(el_meaning))
```



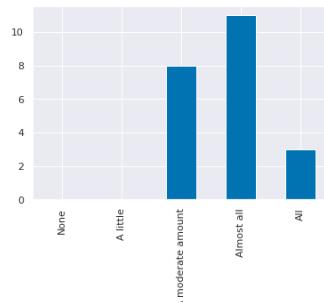
```

u_meaning = {0:'None', 1:'A little',3:'A moderate amount', 4:'Almost all',5:'All'}

question_text = list(short_names.keys())
[list(short_names.values()).index('understand')]
u_counts,_ = np.histogram(feedback_df['understand'],bins = [i+.5 for i in range(6)])
u_df = pd.DataFrame(data = el_counts,index = u_meaning.values(),columns=[question_text],)

u_df.plot.bar(legend=False);

```



18.1.2. Reminders based on requests

- all deadlines are on the Brightspace calendar, you can sync that with your own calendar/scheduling tool

18.1.3. Changes going forward

Assignments:

- clarified instructions on assignments 7+ & a note to myself to fix assignments 1-5 for next year
- a reminder when I post assignments to read before class Friday; time on Friday for clarifying questions
- Chamudi & I will meet, re: grading consistency
- [regrade request policy](#) posted on website
- more advice on choosing a dataset and some possible good ones
- will add “related notes” section
- Some skill changes are coming, will be posted soon, to make more chances on a few skills

In class:

- will continue working on remembering to send all of the code on prismia; feel free to post there to ask for it or raise your hand
- will denote key things that will relate to assignments as much as possible
- to make sure there's space for questions at the end of class, I'll use a google form exit ticket instead of prismia. I'll still answer all of those questions in the notes, but it makes it easier to ask both pace & follow up. It will always be at: <http://drsmb.co/310exit>

Office hours:

- I can change *when* but not how many hours per week, in particular no one has attended Tuesday afternoon, so I may move that.

18.1.4. Requests I will not fulfill

- regular in person office hours; by appointment only to ensure 1 person at a time
- more total office hours
- link directly where in the notes for assignments, part of the goal is for you to filter out the relevant parts from what we learn in a week in order to apply it. You can always ask questions in office hours though.

18.1.5. Notes

- this was anonymous, comments/updates/questions about grading specifically I cannot reply to; e-mail me if applicable
- use [this form](#) to list issues with assignment text/ portfolio requirements to help me improve them in exchange for Ram Tokens

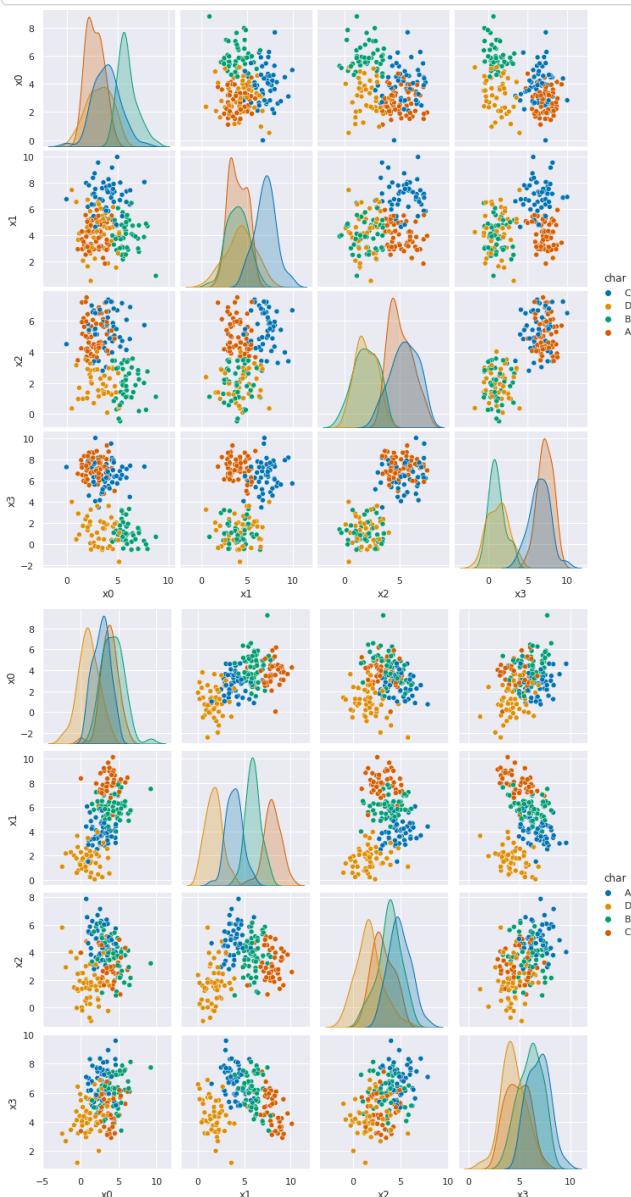
18.2. A6 data review

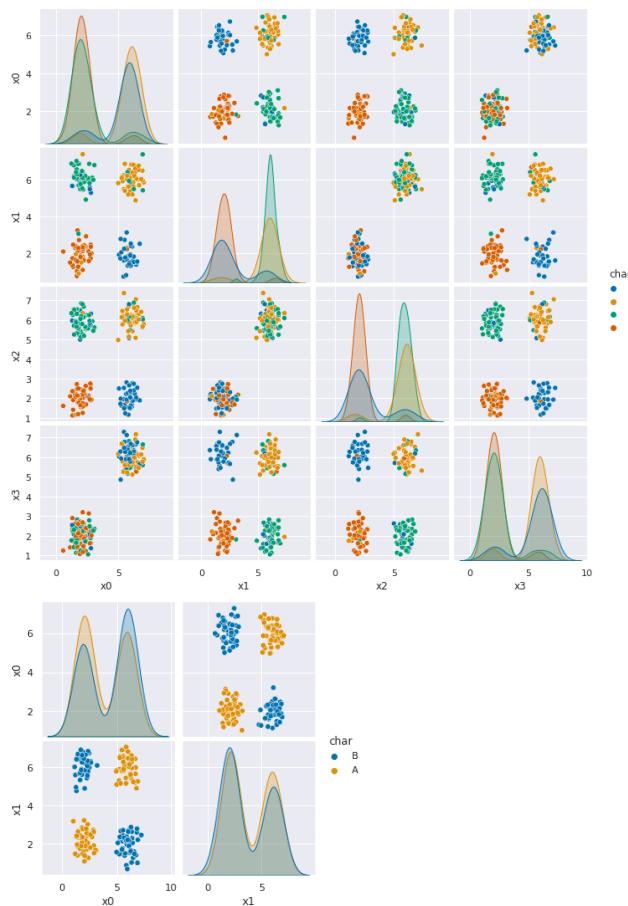
```

a6_data = 'https://raw.githubusercontent.com/rhodyprog4ds/06-naive-
bayes/f425ba121cc04dd8bcaa7ebb2ff0b40bb0b03bff/data/dataset'
req_datasets = [1,2,5,6]
data_urls = [a6_data + str(i) +'.csv' for i in req_datasets]
[sns.pairplot(data =pd.read_csv(url,index_col=0), hue='char') for url in
data_urls]

```

```
[<seaborn.axisgrid.PairGrid at 0x7f25ae8c6310>,
<seaborn.axisgrid.PairGrid at 0x7f25ae8c6310>,
<seaborn.axisgrid.PairGrid at 0x7f25ac3b59a0>,
<seaborn.axisgrid.PairGrid at 0x7f25ab92b130>]
```





Dataset 1 & 2 it should perform reasonably well. They're both Gaussian data, with some skew, but not too much and some overlap (more in 2).

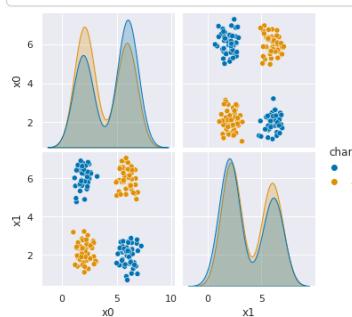
Dataset 5, doesn't do so well, but it's performance is clearly because there's some points in each group of points that are labeled differently than the others. This is to simulate the model for *label bias* which is one way that our traditional performance metrics can fool us. Our GNB classifier finds the four groups very reliably, but is "incorrect" on the points that are labeled differently. In real data this can occur in systematic ways, people from disadvantaged groups who otherwise may for example, qualify for a loan, have been denied historically. So, in training data they would be like the points in each group that are labeled differently.

Dataset 6, seems separable, but Gaussian Naive Bayes gets about 50% accuracy.

18.3. Let's explore that last one...

```
df6= pd.read_csv(data_urls[-1],usecols=[1,2,3])
sns.pairplot(data=df6, hue='char')
```

```
<seaborn.axisgrid.PairGrid at 0x7f25b0e42a90>
```



This data is *separable* even though the classifier we saw last week doesn't work well for it, because not all of the points for each class are in a single blob.

We could imagine a rule that would succeed: if $x0 < 4$ and $x1 < 4$, predict **A** otherwise predict **B**.

```
g = sns.JointGrid(data=df6, x='x0', y ='x1', hue='char')
g.plot_joint(sns.scatterplot)
g.plot_marginals(sns.kdeplot)
g.refine(x=4, y=4)
```

Further reading

For more on label bias [a paper I co-authored with an undergrad from Brown](#) after we worked together for 2 years. It also has citations to prior work on label bias in it.

Tip

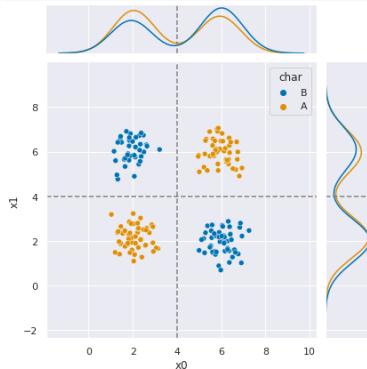
You can earn CSC499/491 by doing research with a faculty member. If fairness is of interest to you, send me an e-mail and we can talk! I am almost always accepting new students and sometimes have funding. We can also arrange to use workstudy or other funds if you qualify. I will have 1 paid position in the spring semester and 2 in summer + you could apply for the Arts & Sciences Fellow funding.

Further Reading

I customized this plot to show additional ideas on top of the data you can read about the [JointGrid](#) in the docs.

Using it will require the most up to date version of seaborn.

```
<seaborn.axisgrid.JointGrid at 0x7f25aaeae910>
```



The dashed line here shows that those boundaries would separate the classes. I stated that rule in a single if, but we could also imagine it like a tree, a set of binary decisions.

18.4. Learning a Decision Tree

We can learn a rule like this one from the data using a `DecisionTreeClassifier`. We'll instantiate one from the `tree` module we imported from `sklearn`.

```
dt = tree.DecisionTreeClassifier()
```

As usual, we split the data into test and train and features and labels:

```
x_train, X_test, y_train, y_test = train_test_split(df6[['x0','x1']],
                                                    df6['char'],
                                                    random_state = 3094)
```

We fit it just like we fit the Gaussian Naive Bayes, using the `fit` method.

```
dt.fit(X_train,y_train)
```

▼ `DecisionTreeClassifier`
`DecisionTreeClassifier()`

💡 Ram Token Opportunity

Contribute a diagram for this part of the notes

💡 Further Reading

- [Decision Tree Docs](#)
- [Decision Tree on Iris](#)

18.5. Examining a Decision Tree

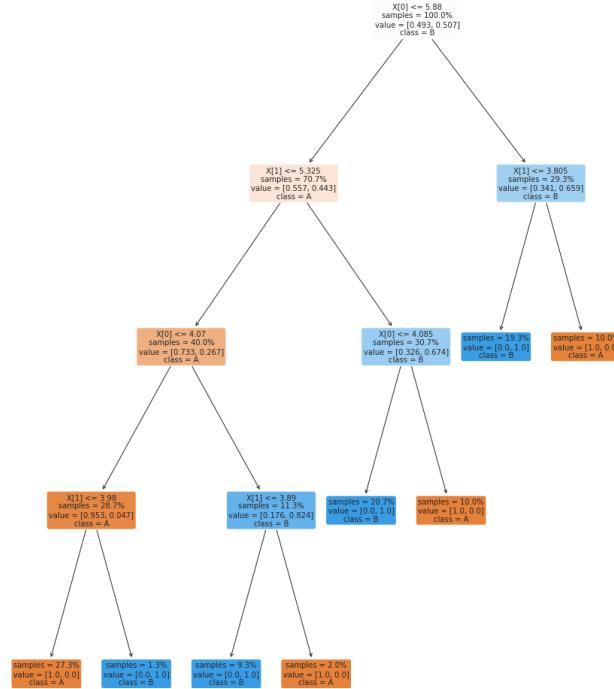
Since decision trees have common functions whether they're for regression or classification, the `tree` module provides some functions that we can use.

```
plt.figure(figsize=(15,20))
tree.plot_tree(dt, rounded =True, class_names = ['A','B'],
               proportion=True, filled =True, impurity=False, fontsize=10)
```

💡 Correction

I was able to make it slightly easier to read, but this was something that I had to look up on [stackoverflow](#)

```
[Text(0.6607142857142857, 0.9, 'X[0] <= 5.88\nsamples = 100.0%\nvalue = [0.493, 0.507]\nclass = B'),  
Text(0.4642857142857143, 0.7, 'X[1] <= 5.325\nsamples = 70.7%\nvalue = [0.557, 0.443]\nclass = A'),  
Text(0.2857142857142857, 0.5, 'X[0] <= 4.07\nsamples = 40.0%\nvalue = [0.733, 0.267]\nclass = A'),  
Text(0.14285714285714285, 0.3, 'X[1] <= 3.98\nsamples = 28.7%\nvalue = [0.953, 0.047]\nclass = A'),  
Text(0.07142857142857142, 0.1, 'samples = 27.3%\nvalue = [1.0, 0.0]\nclass = A'),  
Text(0.21428571428571427, 0.1, 'samples = 1.3%\nvalue = [0.0, 1.0]\nclass = B'),  
Text(0.42857142857142855, 0.3, 'X[1] <= 3.89\nsamples = 11.3%\nvalue = [0.176, 0.824]\nclass = B'),  
Text(0.35714285714285715, 0.1, 'samples = 9.3%\nvalue = [0.0, 1.0]\nclass = B'),  
Text(0.5, 0.1, 'samples = 2.0%\nvalue = [1.0, 0.0]\nclass = A'),  
Text(0.6428571428571429, 0.5, 'X[0] <= 4.085\nsamples = 30.7%\nvalue = [0.326, 0.674]\nclass = B'),  
Text(0.5714285714285714, 0.3, 'samples = 20.7%\nvalue = [0.0, 1.0]\nclass = B'),  
Text(0.7142857142857143, 0.3, 'samples = 10.0%\nvalue = [1.0, 0.0]\nclass = A'),  
Text(0.8571428571428571, 0.7, 'X[1] <= 3.805\nsamples = 29.3%\nvalue = [0.341, 0.659]\nclass = B'),  
Text(0.7857142857142857, 0.5, 'samples = 19.3%\nvalue = [0.0, 1.0]\nclass = B'),  
Text(0.9285714285714286, 0.5, 'samples = 10.0%\nvalue = [1.0, 0.0]\nclass = A')]
```



We can also get it out as text; since it returns a plain string, we'll pass it through the print function.

```
print(tree.export_text(dt))
```

```
--- feature_0 <= 5.88  
|   --- feature_1 <= 5.33  
|   |   --- feature_0 <= 4.07  
|   |   |   --- feature_1 <= 3.98  
|   |   |   |   class: A  
|   |   |   |   --- feature_1 > 3.98  
|   |   |   |   |   class: B  
|   |   |   --- feature_0 > 4.07  
|   |   |   |   --- feature_1 <= 3.89  
|   |   |   |   |   class: B  
|   |   |   |   --- feature_1 > 3.89  
|   |   |   |   |   |   class: A  
|   |   --- feature_1 > 5.33  
|   |   |   --- feature_0 <= 4.09  
|   |   |   |   class: B  
|   |   |   --- feature_0 > 4.09  
|   |   |   |   class: A  
--- feature_0 > 5.88  
|   --- feature_1 <= 3.80  
|   |   class: B  
|   --- feature_1 > 3.80  
|   |   class: A
```

this is the same tree as above.

👉 Try it yourself

Take the time to match this representation to the one above. Read through the parts of the above and see how much it tells you

This tree is many more levels (it's deeper) than the one we figured out by looking at the plotted data above. Before we experiment with changing that, let's see how well it works.

```
dt.score(X_test,y_test)
```

```
1.0
```

It does well, but let's see if we can change the depth and still do well?

18.6. Training parameters

For the Gaussian Naive Bayes classifier, we didn't change how it trained at all. It's a simple classifier, so it's not as impactful. Since Decision Trees are more complex, the model has hyper parameters and the training algorithm (`fit` method) has parameters.

With `sklearn` these parameters are set when the object is instantiated. We'll see more on Friday, but for now, we'll set the `max_depth`.

```
dt2 = tree.DecisionTreeClassifier(max_depth=2)
dt2.fit(X_train,y_train)
dt2.score(X_test,y_test)
```

0.74

```
print(tree.export_text(dt2))
```

```
|--- feature_0 <= 5.88
|   |--- feature_1 <= 5.33
|   |   |--- class: A
|   |   |--- feature_1 >  5.33
|   |   |--- class: B
|--- feature_0 >  5.88
|   |--- feature_1 <= 3.80
|   |   |--- class: B
|   |   |--- feature_1 >  3.80
|   |   |--- class: A
```

18.7. Questions After Class

18.7.1. What is it best used on?

Decision trees are good for lots of tabular data (not images).

18.7.2. Is there a maximum amount of branches for a tree

A tree will always have binary splits, so the maximum number of leaf nodes (ends) is the size of the data. That wouldn't be a very good classifier though, we'll see Friday we can make the size needed to split the data (create a new branch) larger as another way to control the tree.

18.7.3. Why do the decision trees work for some data but not others.

Varialbe performance is due to the data and

18.7.4. What could we really use these trees for?

Decision trees are a very widely used algorithm for making predictions. In fact, a flowchart of this form is used in some contexts even when not learned from data.

18.7.5. How is the tree created?

We'll talk about this a little bit on Friday, but it's mostly out of scope for this course. It is something you could learn about independently for your portfolio, or it will be covered in depth in CSC461.

The `sklearn` docs include a whole [algorithms](#) section that describes everything including their history and detailed mathematical formulation.

18.7.6. Do I use a gnb when I know the distribution is gaussian?

Yes, or at least close enough. Fitting the model to the data is a good practice. We'll learn about overfitting soon and by matching, you're less likely to face this problem.

19. Decision Tree Setting and more Evaluation

```
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
sns.set(palette = 'colorblind') # this improves contrast
from sklearn.metrics import confusion_matrix, classification_report
```

19.1. Review

```
corner_data = 'https://raw.githubusercontent.com/rhodyprog4ds/06-naive-
bayes/f425ba121cc0c4dd8bcfa7ebb2ff0b40b0b03bff/data/dataset6.csv'
df6= pd.read_csv(corner_data,usecols=[1,2,3])
iris_df = sns.load_dataset('iris')
```

```
df6.columns
```

```
Index(['x0', 'x1', 'char'], dtype='object')
```

set up the same splits again

```
X_train, X_test, y_train, y_test = train_test_split(df6[['x0','x1']],
                                                    df6['char'],
                                                    random_state=34)
```

Fit a baseline model with default settings

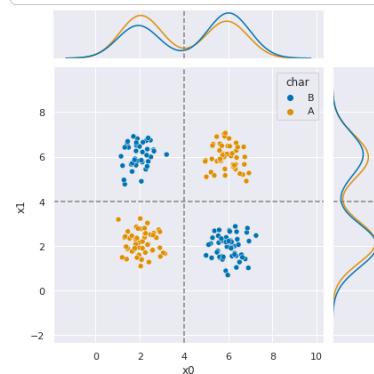
```
dt = tree.DecisionTreeClassifier()
dt.fit(X_train,y_train)
dt.score(X_test,y_test)
```

1.0

it does well, but let's examine it more closely. First, we'll look back at the data.

```
g = sns.JointGrid(data=df6, x='x0', y='x1', hue='char')
g.plot_joint(sns.scatterplot)
g.plot_marginals(sns.kdeplot)
g.refline(x=4, y=4)
```

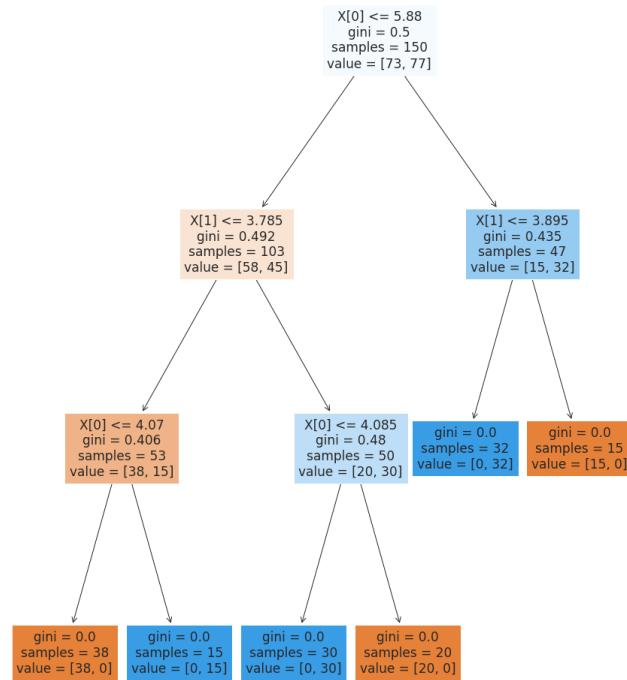
<seaborn.axisgrid.JointGrid at 0x7fd5083b6040>



In this, the dashed lines represent boundaries that separate the two classes. Next, we can look at what it learned. Using `filled=True` makes the nodes shaded, so we can examine it better.

```
plt.figure(figsize=(15,20))
tree.plot_tree(dt,filled=True)
```

```
[Text(0.5909090909090909, 0.875, 'X[0] <= 5.88\ngini = 0.5\nsamples = 150\nvalue = [73, 77]',  
Text(0.36363636363636365, 0.625, 'X[1] <= 3.785\ngini = 0.492\nsamples = 103\nvalue = [58, 45]',  
Text(0.181818181818182, 0.375, 'X[0] <= 4.07\ngini = 0.406\nsamples = 53\nvalue = [38, 15]',  
Text(0.09090909090909091, 0.125, 'gini = 0.0\nsamples = 38\nvalue = [38, 0]',  
Text(0.2727272727272727, 0.125, 'gini = 0.0\nsamples = 15\nvalue = [0, 15]',  
Text(0.5454545454545454, 0.375, 'X[0] <= 4.085\ngini = 0.48\nsamples = 50\nvalue = [20, 30]',  
Text(0.4545454545454543, 0.125, 'gini = 0.0\nsamples = 30\nvalue = [0, 30]',  
Text(0.6363636363636364, 0.125, 'gini = 0.0\nsamples = 20\nvalue = [20, 0]',  
Text(0.8181818181818182, 0.625, 'X[1] <= 3.895\ngini = 0.435\nsamples = 47\nvalue = [15, 32]',  
Text(0.7272727272727273, 0.375, 'gini = 0.0\nsamples = 32\nvalue = [0, 32]',  
Text(0.9090909090909091, 0.375, 'gini = 0.0\nsamples = 15\nvalue = [15, 0]'))
```



Each node in the tree shows the threshold that was compared, the gini score and the number of samples from each class that pass through that node.

we can see from the graph what happened, but this is still not finding what we know would be the best performing decision tree.

Let's try limiting when it can split based on the share of the data.

```

dt_large_split = tree.DecisionTreeClassifier(min_samples_split=.2,
                                             max_depth=2)
dt_large_split.fit(X_train,y_train)
dt_large_split.score(X_test,y_test)

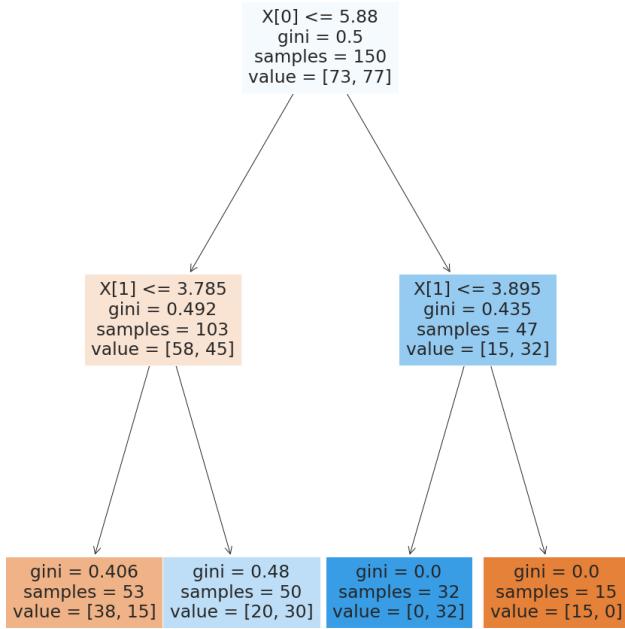
```

0.8

```

plt.figure(figsize=(15,20))
tree.plot_tree(dt_large_split,filled=True);

```



```

dt_large_split = tree.DecisionTreeClassifier(min_samples_split=.4,
                                             max_depth=2)
dt_large_split.fit(X_train,y_train)
dt_large_split.score(X_test,y_test)

```

0.66

We can also limit based on how much data has to be in each leaf.

```

dt_large_leaf = tree.DecisionTreeClassifier(min_samples_leaf=.2)
dt_large_leaf.fit(X_train,y_train)
dt_large_leaf.score(X_test,y_test)

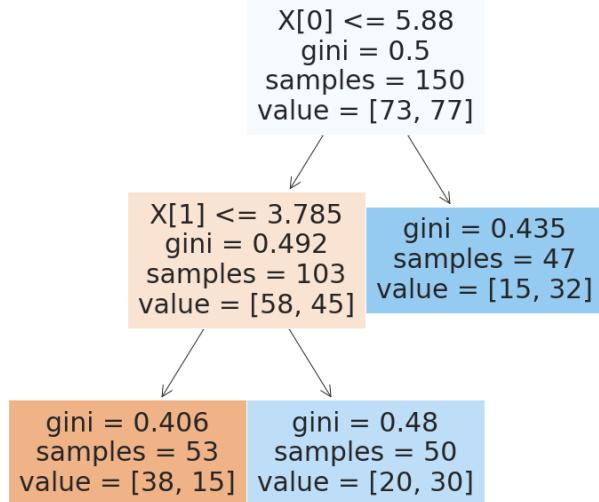
```

0.66

```

plt.figure(figsize=(12,12))
tree.plot_tree(dt_large_leaf,filled=True);

```



This one gets the right number of levels, but still does not have good performance.

19.2. More evaluation

To do more detailed evaluation than the main score, we have to get the predictions.

```

y_pred_ll = dt_large_leaf.predict(X_test)
print(classification_report(y_test,y_pred_ll))

```

	precision	recall	f1-score	support
A	0.76	0.57	0.65	28
B	0.59	0.77	0.67	22
accuracy			0.66	50
macro avg	0.67	0.67	0.66	50
weighted avg	0.68	0.66	0.66	50

We can also get the confusion matrix out

```

confusion_matrix(y_test,y_pred_ll)

```

```

array([[16, 12],
       [ 5, 17]])

```

We can unpack those values into individual elements, that match the [true negative](#), [false positive](#), [false negative](#), [true positive labels](#) using [ravel](#) flattens a 2d numpy array; default is 'C' row major order; can change with order param

```

tn, fp, fn, tp = confusion_matrix(y_test,y_pred_ll).ravel()

```

We can compute other metrics from the confusion matrix:

Accuracy is the true ones/ total number of samples

```

(tp + tn)/(tp+fp+fn+tn)

```

```

0.66

```

19.3. Parsing Assignment 7

What does data good for classification look like?

- there has to be a categorical variable to be the target
- there has to be other variables as the features where it makes sense to predict the target from those variables

Datasets for Machine Learning: the [UCI repository](#).

The new [beta](#) has a nicer interface for finding data.

You can filter by task and type of data. So far, we've only worked with tabular data and studied classification.

Further Reading

More on confusion matrices, use both to match up and figure out how you can calculate other metrics from these.

- [sklearn](#)
- [wiki](#)

Options

Clear

Dataset

Characteristics*

- Tabular 261
- Sequential 44
- Time-Series 87
- Text 50
- Image 3
- Other 111

Subject Area



Associated Tasks*



- Classification 261
- Regression 83
- Clustering 70
- Other 14

Attributes



Instances



19.4. Questions after class

19.4.1. Is there some kind of rule of thumb to make it go faster?

Not exactly. The more you understand the models you'll build intuition that help you decide faster and there are ways to use search algorithm to find the best set of parameters. We'll see those in a couple of weeks. For now, try a little experimentation and we'll consider more then.

19.5. More Practice

1. Write a function that uses if, else to implement the predict function of a decision tree
2. Compute the metrics from the confusion matrix (accuracy, precision, recall)
3. Apply Decision tree to the iris data

20. Linear 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')
```

20.1. Setting up a linear regression

```

tips = sns.load_dataset("tips")
tips.head(1)

total_bill    tip     sex   smoker   day      time   size
0       16.99  1.01  Female     No  Sun Dinner      2

```

We're going to predict `tip` from `total_bill` using 80% of the data for training. This is a regression problem because the target, `tip` is 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.

```

# sklearn requires 2D object of features even for 1 feature
tips_X = tips[['total_bill']].values
tips_X = tips_X[:, np.newaxis] # add an axis
tips_y = tips['tip']

tips_X_train, tips_X_test, tips_y_train, tips_y_test = train_test_split(
    tips_X,
    tips_y,
    train_size=.8,
    random_state=0)

```

To see what that new bit of code did, we can examine the shapes:

```

tips_X.shape

(244, 1)

```

what we ended up is 2 dimensions (there are two numbers) even though the second one is 1.

```

tips[['total_bill']].values.shape

(244,)

```

this, without the `newaxis` is one dimension, we can see that because there is no number after the comma.

Now that our data is ready, we create the linear regression estimator object

```

regr = linear_model.LinearRegression()

```

Now we fit the model.

```

regr.fit(tips_X_train, tips_y_train)

LinearRegression()

```

We can examine the coefficients and intercept.

```

regr.coef_, regr.intercept_

(array([0.0968534]), 1.0285439454607272)

```

These define a line ($y = mx+b$) `coef` is the slope.

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 $-0.10 + 1.028$. (it's actually $9.68\% + 1.028$)

To interpret this, we can apply it for a single value. We trained this to predict the tip from the total bill. So, we can put in any value that's a plausible total bill and get the predicted tip.

```

my_bill = np.asarray([17.78]).reshape(1, -1)
regr.predict(my_bill)

array([2.75059744])

```

We can also apply the function, as usual.

```

tips_y_pred = regr.predict(tips_X_test)

```

This gives a vector of values.

```

tips_y_pred

array([2.7321953 , 2.79999268, 2.91621676, 1.73073111, 2.60434881,
       1.58545101, 2.76415692, 3.28813383, 2.7864332 , 4.38451435,
       3.47699796, 3.47021823, 2.39127132, 2.28763818, 2.32831661,
       3.97288739, 1.83726986, 2.38449158, 2.84745685, 3.26585755,
       3.93995723, 3.05471713, 2.57819839, 2.48521912, 2.33763342,
       2.61693975, 2.26628132, 3.91477534, 3.4779665 , 2.55592121,
       2.45519457, 2.23727441, 2.52262341, 2.65422148, 2.79999268,
       2.32541101, 2.66827205, 2.02903959, 5.7094689 , 2.57626132,
       1.85954614, 2.23243174, 2.54817383, 3.91961801, 2.26439336,
       2.67214619, 2.79545001, 3.11864037, 2.68183153])

```

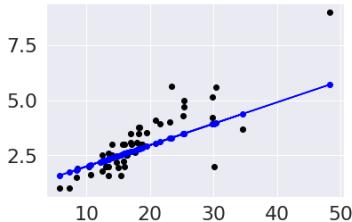
To visualize in more detail, we'll plot the data as black points and the predictions as blue points. To highlight that this is a perfectly linear prediction, we'll also add a line for the prediction.

```

plt.scatter(tips_X_test,tips_y_test, color='black')
plt.plot(tips_X_test,tips_y_pred, color='blue')
plt.scatter(tips_X_test,tips_y_pred, color='blue')

```

<matplotlib.collections.PathCollection at 0x7fe60e45eeb0>



20.2. Evaluating Regression - Mean Squared Error

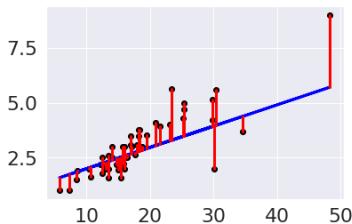
From the plot, we can see that there is some error for each point, so accuracy that we've been using, won't work. One idea is to look at how much error there is in each prediction, we can look at that visually first.

```

plt.scatter(tips_X_test, tips_y_test, color='black')
plt.plot(tips_X_test, tips_y_pred, color='blue', linewidth=3)

# draw vertical lines from each data point to its predict value
[plt.plot([x,x],[yp,yt], color='red', linewidth=3)
 for x, yp, yt in zip(tips_X_test, tips_y_pred, tips_y_test)];

```



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, tips_y_pred)
```

0.821309064276629

We can get back to the units being dollars, by taking the square root.

```
np.sqrt(mean_squared_error(tips_y_test, tips_y_pred))
```

0.9062610353957787

This is equivalent to using absolute value instead

```
np.mean(np.abs(tips_y_test - tips_y_pred))
```

0.6564074900962107

20.3. Evaluating Regression - R2

We can also use the R^2 regression coefficient.

```
r2_score(tips_y_test, tips_y_pred)
```

0.5906895098589039

This is a bit harder to interpret, but we can use some additional plots to visualize. This code simulates data by randomly picking 20 points, spreading them out and makes the "predicted" y values by picking a slope of 3. Then I simulated various levels of noise, by sampling noise and multiplying the same noise vector by different scales and adding all of those to a data frame with the column name the r score for if that column of target values was the truth.

Then I added some columns of y values that were with different slopes and different functions of x. These all have the small amount of noise.

Tip

[Facet Grids](#) allow more customization than the figure level plotting functions we have used otherwise, but each of those combines a FacetGrid with a particular type of plot.

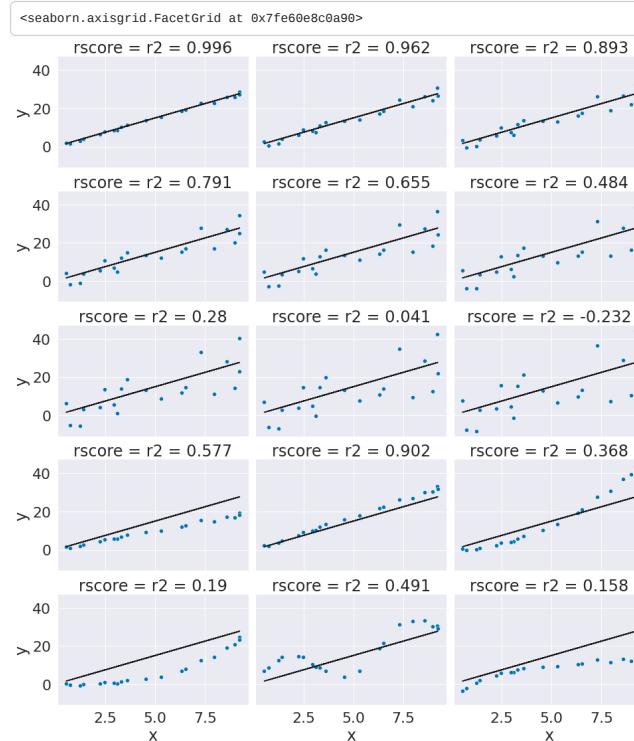
```

x = 10*np.random.random(20)
y_pred = 3*x
ex_df = pd.DataFrame(data = x,columns = ['x'])
ex_df['y_pred'] = y_pred
n_levels = range(1,18,2)
# sample 0 mean noise
noise = (np.random.random(20)-.5)*2
# add varying noise levels
for n in n_levels:
    # add noise, scaled
    y_true = y_pred + n* noise
    # compute the r2 in the column name, assign the "true" (data) here
    ex_df['r2 = '+ str(np.round(r2_score(y_pred,y_true),3))] = y_true

# add functions
f_x_list = [2*x,.35*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
    # compute the r2 in the column name, assign the "true" (data) here
    ex_df['r2 = '+ str(np.round(r2_score(y_pred,y_true),3))] = y_true

# melt the data frame for plotting
xy_df = ex_df.melt(id_vars=['x','y_pred'],var_name='rscore',value_name='y')
# create a FacetGrid so that we can add two types of plots per subplot
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",)

```



20.4. Multivariate Regression

We can also load data from Scikit learn.

This dataset includes 10 features measured on a given date and a measure of diabetes disease progression measured one year later. The predictor we can train with this data might be something a doctor uses to calculate a patient's risk.

```

diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y = True)

diabetes_X.shape

(442, 10)

diabetes_X_train, diabetes_X_test, diabetes_y_train, diabetes_y_test =
train_test_split(
    diabetes_X, diabetes_y)
regr_diabetes = linear_model.LinearRegression()

regr_diabetes.fit(diabetes_X_train,diabetes_y_train)

LinearRegression()
LinearRegression()

```

20.5. What score does linear regression use?

```

regr_diabetes.score(diabetes_X_test,diabetes_y_test)

0.3535771867545594

diabetes_y_pred = regr_diabetes.predict(diabetes_X_test)

r2_score(diabetes_y_test,diabetes_y_pred)

```

```
0.3535771867545594
```

```
mean_squared_error(diabetes_y_test,diabetes_y_pred)
```

```
3116.1282586999246
```

It uses the R2 score.

This model predicts what lab measure a patient will have one year in the future based on lab measures in a given day. Since we see that this is not a very high r2, we can say that this is not a perfect predictor, but a Doctor, who better understands the score would have to help interpret the core.

20.6. Questions After class

20.6.1. How I should use these with data most effectively? What is the proper use of these methods?

To answer continuous prediction tasks, like the ones we saw today. The notes above include more interpretation than we discussed in class, so read carefully for that.

20.6.2. Why is that even when random state is set to 0 numbers are still a little different compared to yours and my neighbor even

`random_state` sets the seed that's used internally and should work to [control the randomness](#) and produce reproducible results. If your results are just a little different, like that it could be a rounding error, maybe you somehow set a default for display that's different.

See for example [these options](#)

21. Interpreting Regression

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import itertools as itr
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
sns.set_theme(font_scale=2, palette='colorblind')
```

we'll return to the same data we used on Monday, first.

```
tips = sns.load_dataset("tips").dropna()
```

```
tips.shape
```

```
(244, 7)
```

```
tips.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

Again, we'll prepare the data.

```
# sklearn requires 2D object of features even for 1 feature
tips_X = tips[['total_bill']].values
tips_X = tips_X[:,np.newaxis] # add an axis
tips_y = tips['tip']

tips_X_train,tips_X_test, tips_y_train, tips_y_test = train_test_split(
    tips_X,
    tips_y,
    train_size=.8,
    random_state=0)
```

Next, we'll fit the model

```
regr_tips = linear_model.LinearRegression()
regr_tips.fit(tips_X_train,tips_y_train)
regr_tips.score(tips_X_test,tips_y_test)
```

```
0.5906895098589039
```

This doesn't perform all that well, but let's investigate it further. We'll start by looking at the residuals

```
tips_y_pred = regr_tips.predict(tips_X_test)
```

21.1. Examining Residuals

The error, the difference between the predictions and the truth is called the residual.

```
tips_y_pred - tips_y_test
```

```

64    0.092195
63   -0.960007
55   -0.593783
111   0.730731
225   0.104349
92    0.585451
76    -0.315843
181   -2.361866
188   -0.713567
180    0.704514
73    -1.523602
187   -0.819782
150   -0.108729
198    0.287638
224    0.748317
44    -1.627113
145    0.337270
110   -0.615508
243   -0.152549
189   -0.734142
210    1.939957
104   -1.025283
138    0.578198
8     0.525219
199    0.337033
263    0.116949
220    0.006281
125   -0.285225
5     -1.232034
22    0.325922
74    0.255195
124   -0.282726
12    0.952023
168   0.444221
45    -0.200007
158   -0.284589
37    -0.401728
136    0.029946
212   -3.290531
223   -0.423739
222   -0.060454
118    0.432432
231   -0.451826
155   -1.220382
155    0.034393
209   -0.827854
188   -0.964850
15    -0.801360
71    -0.318168
Name: tip, dtype: float64

```

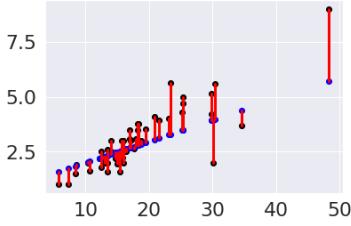
To examine these, we can plot them on the data:

```

pit.scatter(tips_X_test,tips_y_test, color='black')
pit.scatter(tips_X_test,tips_y_pred, color='blue')

[plt.plot([x,x],[yp,yp], color='red', linewidth=3)
 for x, yp, yt in zip(tips_X_test, tips_y_pred, tips_y_test)];

```



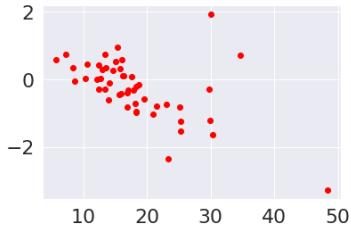
We can plot them as a scatter plot as well.

```

tips_residuals = tips_y_pred - tips_y_test
pit.scatter(tips_X_test,tips_residuals, color='red')

```

```
<matplotlib.collections.PathCollection at 0x7efca4cb3e20>
```



One thing we notice is that the residuals are smaller for some values of the `total_bill` and larger for others. This suggests that there is more information left. A good fit, would have residuals that are evenly distributed, not correlated with the feature(s) in this case, the total bill.

21.2. Polynomial regression

Polynomial regression is still a linear problem. Linear regression solves for the (β_i) for a (d) dimensional problem.

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d = \sum_i \beta_i x_i$$

Quadratic regression solves for

$$y = \beta_0 + \sum_i \beta_i x_i + \sum_j \beta_j x_j^2 + \dots + \sum_k \beta_k x_k^k$$

This is still a linear problem, we can create a new (X) matrix that has the polynomial values of each feature and solve for more (β) values.

We use a transformer object, which works similarly to the estimators, but does not use targets. First, we instantiate.

```

poly = PolynomialFeatures(include_bias=False)

```

Then we apply it

```
tips_X2_train = poly.fit_transform(tips_X_train)
tips_X2_test = poly.fit_transform(tips_X_test)
```

We can see what it did by looking at the shape.

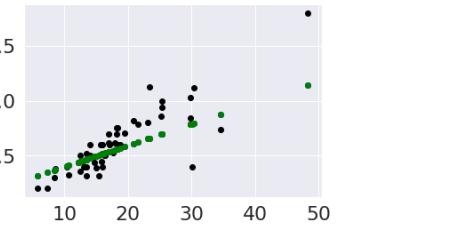
```
tips_X_train.shape, tips_X2_train.shape
((195, 1), (195, 2))
tips_X2_train[:5,1]
array([ 722.5344, 1067.9824, 320.0521, 419.8401, 2320.3489])
tips_X_train[:5]
array([[26.88],
       [32.68],
       [17.89],
       [20.49],
       [48.17]])
tips_X2_train[:5,0]
array([26.88, 32.68, 17.89, 20.49, 48.17])
```

Now, we can fit a linear model on this data, which learns a weight for the data and its squared value.

```
regr2_tips = linear_model.LinearRegression()
regr2_tips.fit(tips_X2_train, tips_y_train)
tips2_y_pred = regr2_tips.predict(tips_X2_test)
```

Then we can plot it.

```
plt.scatter(tips_X_test, tips_y_test, color='black')
plt.scatter(tips_X_test, tips_y_pred, color='blue')
plt.scatter(tips_X_test, tips2_y_pred, color='green')
<matplotlib.collections.PathCollection at 0x7efca2c268e0>
```

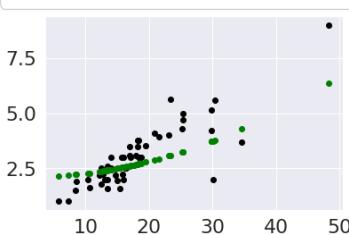


We can see that this is somewhat better, the residuals are more uniformly distributed, but it doesn't look very nonlinear.

We will examine this further in the next step, but first we will drop the linear column to see the quadratic more clearly.

```
poly = PolynomialFeatures()
tips_Xq_train = poly.fit_transform(tips_X_train)[:, ::2]
tips_Xq_test = poly.fit_transform(tips_X_test)[:, ::2]
regr_qu_tips = linear_model.LinearRegression(fit_intercept=False)
regr_qu_tips.fit(tips_Xq_train, tips_y_train)
tips2_q_pred = regr_qu_tips.predict(tips_Xq_test)
```

```
plt.scatter(tips_X_test, tips_y_test, color='black')
plt.scatter(tips_X_test, tips2_q_pred, color='green')
<matplotlib.collections.PathCollection at 0x7efca2c35580>
```



💡 Try it Yourself

How would you make it cubic? what about 4th dimension?

21.3. Examining Coefficients

Now we can compare the coefficients. We saw above that the quadratic didn't help much, so let's look at those.

```
regr2_tips.coef_
array([ 9.70620903e-02, -4.18198822e-06])
```

The second parameter is very very small, so that explains why it didn't change the fit much. We can use the features to figure out how important each feature is to the prediction. Large numbers strongly influence the prediction smaller ones influence it less.

```
{ regr_tips.coef_
```

```
array([ 0.0968534])
```

21.4. Sparse Regression

An extreme is for some coefficients to be zero. The LASSO model, constrains some of the coefficients to be 0, so it learns simultaneously how to combine the features to predict the target and which subset of the features to use.

Further Reading

For the mathematical formulation see the sklearn [User Guide Section on LASSO](#) and the code in [LASSO docs](#)

Thinking Ahead

LASSO is not required for assignment 8, but is one way you could earn level 3. Here is a preview, but you can investigate it further on your own.

```
tips_lasso = linear_model.Lasso(alpha=.0025)
tips_lasso.fit(tips_all_X_train,tips_all_y_train)
tips_lasso_y_pred = tips_lasso.predict(tips_all_X_test, )
tips_lasso.score(tips_all_X_test,tips_all_y_test)
```

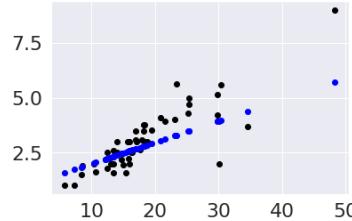
```
NameError: Traceback (most recent call last)
Input In [23], in <cell line: 2>()
    1 tips_lasso = linear_model.Lasso(alpha=.0025)
----> 2 tips_lasso.fit(tips_all_X_train,tips_all_y_train)
    3 tips_lasso_y_pred = tips_lasso.predict(tips_all_X_test, )
    4 tips_lasso.score(tips_all_X_test,tips_all_y_test)

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

```
plt.scatter(tips_X_test,tips_y_test, color='black')
plt.scatter(tips_X_test,tips_y_pred, color='blue')
plt.scatter(tips_X_test,tips_lasso_y_pred, color='green')
```

```
NameError: Traceback (most recent call last)
Input In [24], in <cell line: 3>()
    1 plt.scatter(tips_X_test,tips_y_test, color='black')
    2 plt.scatter(tips_X_test,tips_y_pred, color='blue')
----> 3 plt.scatter(tips_X_test,tips_lasso_y_pred, color='green')

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



```
{ sum(tips_lasso.coef_ ==0)/len(tips_lasso.coef_)
```

```
AttributeError: Traceback (most recent call last)
Input In [25], in <cell line: 1>()
----> 1 sum(tips_lasso.coef_ ==0)/len(tips_lasso.coef_)

AttributeError: 'Lasso' object has no attribute 'coef_'
```

```
{ tips_onehot.shape, tips_interacion.shape
```

```
NameError: Traceback (most recent call last)
Input In [26], in <cell line: 1>()
----> 1 tips_onehot.shape, tips_interacion.shape

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

The transform changed our data from 10 columns to 55.

```
{ tips_interacion.head
```

```
NameError: Traceback (most recent call last)
Input In [27], in <cell line: 1>()
----> 1 tips_interacion.head

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

21.5. Questions After Class

21.5.1. When do we do regression?

we do regression, when we want to predict a continuous value

21.5.2. What should I look for in datasets to know whether a linear model or non-linear model is best?

If you know a reason to choose one from domain knowledge, always use that. From data alone, a reasonable thing to do is to fit a linear model and then examine the residuals and use a more complex model if that makes sense.

21.5.3. How can we tell if a dataset is going to be useful through tweaking or is just not worth it?

This is a **very** good question, but does not have a simple answer. In some cases, a moderate fit quality is enough, because there's low risk of making errors. In other cases, a really high quality fit is required because of the risk.

22. Clustering

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn import datasets
from sklearn.cluster import KMeans
import string
import pandas as pd
```

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 $\{N + K \cdot d\}$ 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.

22.1. How does Kmeans work?

We will start with some synthetic data and then see how the clustering works.

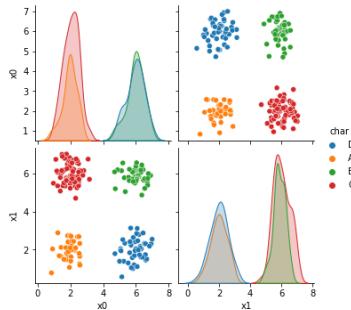
```
C = 4
N = 200
classes = list(string.ascii_uppercase[:C])
mu = {c: i for c, i in zip(classes, [[2,2], [6,6], [2,6],[6,2]])}
sigma = {c: i*.5 for c, i in zip(classes,np.random.random(4))}
sigma

target5 = np.random.choice(classes,N)
data5 = [np.random.multivariate_normal(mu[c], .25*np.eye(2)) for c in target5]
df5 = pd.DataFrame(data = data5,columns = ['x' + str(i) for i in range(2)]).round(2)

df5['char'] = target5

sns.pairplot(data = df5, hue='char')
```

<seaborn.axisgrid.PairGrid at 0x7f97a5c54ca0>



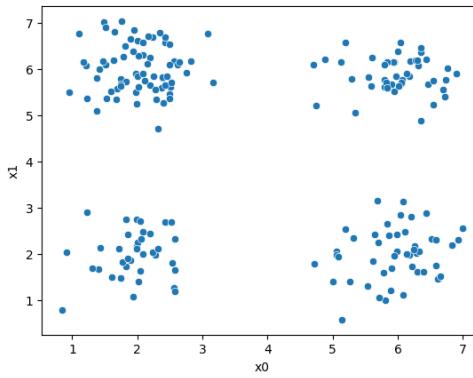
First, we'll pick just the data columns.

```
df = df5[['x0','x1']]
df.head()
```

	x0	x1
0	6.28	2.01
1	5.32	2.34
2	2.57	1.65
3	5.89	1.20
4	5.94	6.15

```
sns.scatterplot(data=df,x='x0',y='x1')
```

```
<AxesSubplot:xlabel='x0', ylabel='x1'>
```



Next, we'll pick 4 random points to be the starting points as the means.

```
K = 4  
mu0 = df.sample(n=K).values
```

```
array([[1.94, 6.39],  
       [2.25, 5.84],  
       [1.48, 7.02],  
       [2.5 , 6.09]])
```

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-mu_i)**2).sum(axis=1) for mu_i in mu0]
```

```
[0    38.0200  
1    27.8269  
2    22.8645  
3    42.5386  
4    16.0576  
...  
195   0.0549  
196   1.5793  
197   24.9065  
198   16.8805  
199   22.5056  
Length: 200, dtype: float64,  
0    30.9098  
1    21.6749  
2    17.6585  
3    34.7792  
4    13.7122  
...  
195   0.5585  
196   1.2913  
197   19.7665  
198   12.4225  
199   18.7762  
Length: 200, dtype: float64,  
0    48.1401  
1    36.6480  
2    30.0250  
3    53.3205  
4    20.6485  
...  
195   0.5746  
196   2.8232  
197   31.8766  
198   23.1842  
199   28.8077  
Length: 200, dtype: float64,  
0    30.9348  
1    22.0149  
2    19.7185  
3    35.4042  
4    11.8372  
...  
195   0.3985  
196   2.1713  
197   22.2265  
198   14.1425  
199   17.0612  
Length: 200, dtype: float64]
```

This gives us a list of 4 data DataFrames, one for each mean (μ), 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-mu_i)**2).sum(axis=1) for mu_i in mu0],axis=1).head()
```

	0	1	2	3
0	38.0200	30.9098	48.1401	30.9348
1	27.8269	21.6749	36.6480	22.0149
2	22.8645	17.6585	30.0250	19.7185
3	42.5386	34.7792	53.3205	35.4042
4	16.0576	13.7122	20.6485	11.8372

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-mu_i)**2).sum(axis=1) for mu_i in  
mu0],axis=1).idxmin(axis=1).head()
```

```

0    1
1    1
2    1
3    1
4    3
dtype: int64

```

We'll save all of this in a column named '`0`'. Since it is our 0th iteration.

```

df['0'] = pd.concat([(df-mu_i)**2).sum(axis=1) for mu_i in
mu0],axis=1).idxmin(axis=1)

df.head()

```

	x0	x1	0
0	6.28	2.01	1
1	5.32	2.34	1
2	2.57	1.65	1
3	5.89	1.20	1
4	5.94	6.15	3

Here, we'll set up some helper code.

```

data_cols = ['x0','x1']
def mu_to_df(mu,i):
    mu_df = pd.DataFrame(mu,columns=data_cols)
    mu_df['iteration'] = str(i)
    mu_df['class'] = ['M'+str(i) for i in range(K)]
    mu_df['type'] = 'mu'
    return mu_df

# color maps, we select every other value from this palette that has 8 values & is
# paired
cmap_pt = sns.color_palette('tab20',8)[1::2] # starting from 1
cmap_mu = sns.color_palette('tab20',8)[0::2] # starting from 0

```

Further Reading

For more on [color palettes](#) see the seaborn docs

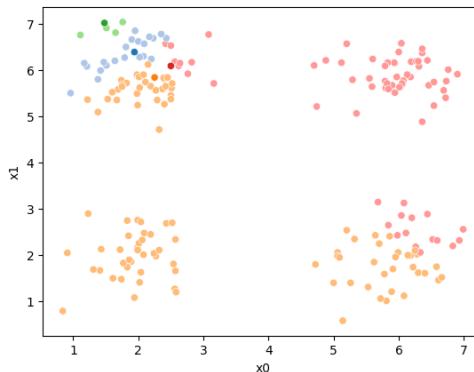
Now we can plot the data, save the axis, and plot the means on top of that. Seaborn plotting functions return an axis, by saving that to a variable, we can pass it to the `ax` parameter of another plotting function so that both plotting functions go on the same figure.

```

sfig = sns.scatterplot(data=df,x='x0',y='x1', hue='0',
                       palette=cmap_pt,legend=False)
mu_df = mu_to_df(mu0,0)
sns.scatterplot(data=mu_df, x='x0',y='x1',hue='class',
                 palette=cmap_mu,legend=False, ax=sfig)

```

```
<AxesSubplot: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.

```

mu1 = df.groupby('0')[data_cols].mean().values
mu1

```

```

array([[1.80545455, 6.34318182],
       [3.19078431, 3.07215686],
       [1.502        , 6.908        ],
       [5.58774648, 5.21830986]])

```

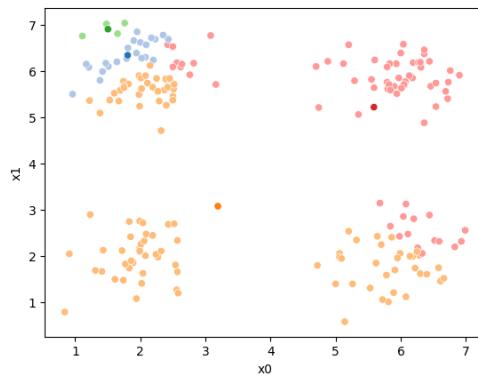
We can plot these again, the same data, but with the new means.

```

sfig = sns.scatterplot(data=df,x='x0',y='x1', hue='0',
                       palette=cmap_pt,legend=False)
mu_df = mu_to_df(mu1,0)
sns.scatterplot(data=mu_df, x='x0',y='x1',hue='class',
                 palette=cmap_mu,legend=False, ax=sfig)

```

```
<AxesSubplot: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.

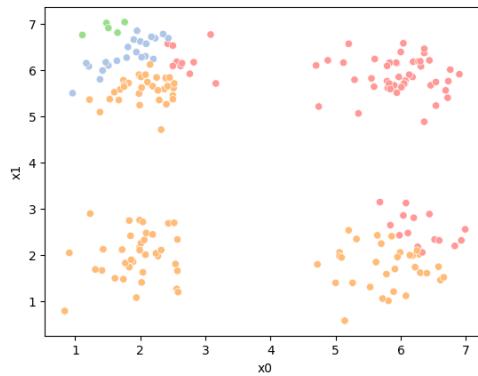
So, again we can update the assignments.

```
df['1'] = pd.concat([(df[data_cols]-mu_i)**2).sum(axis=1) for mu_i in mu],axis=1).idxmin(axis=1)
```

And plot again.

```
sfig = sns.scatterplot(data=df,x='x0',y='x1', hue='0',
                       palette=cmap_pt,legend=False)
mu_df = mu_to_df(mu,0)
sns.scatterplot(data=mu_df, x='x0',y='x1',hue='class',
                 palette=cmap_mu,legend=False, ax=sfig)
```

```
<AxesSubplot: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.

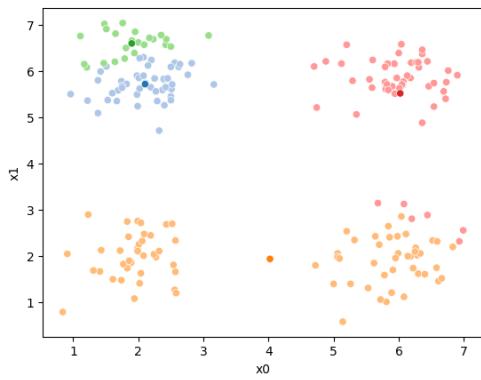
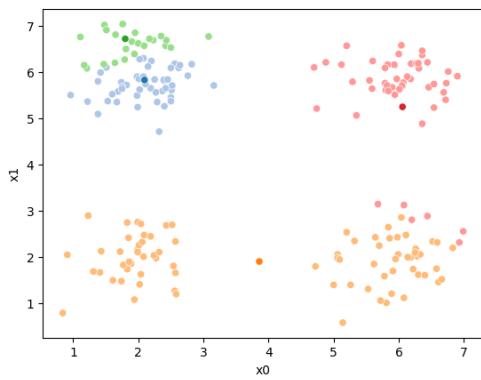
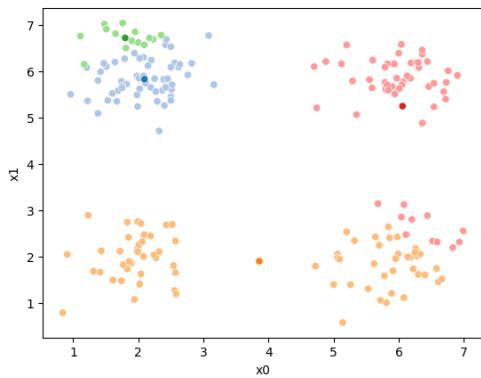
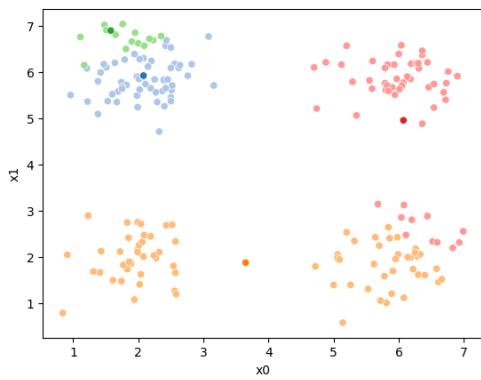
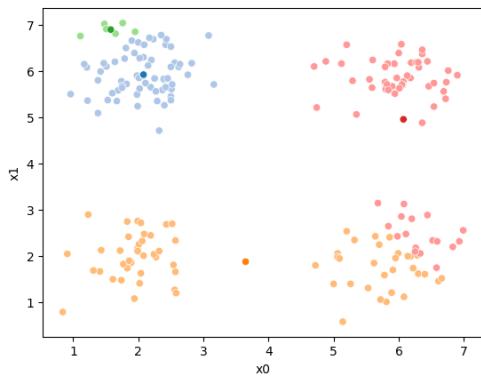
My previous version had a different *initialization* (the above part) so it was okay for the steps to be in the other order.

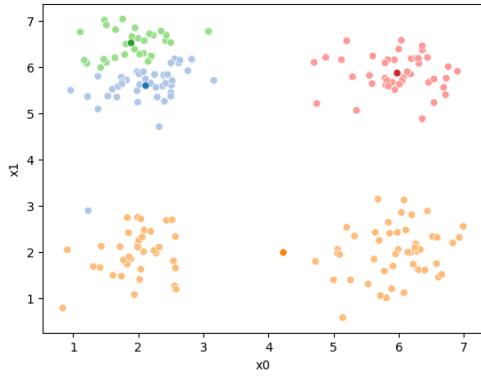
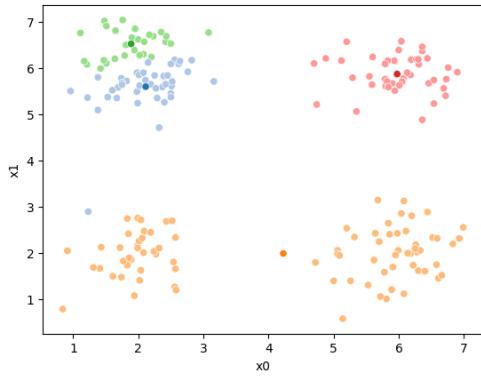
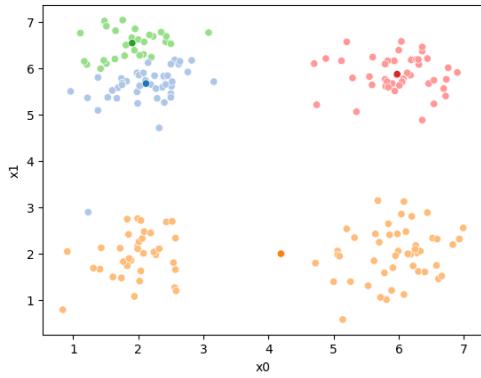
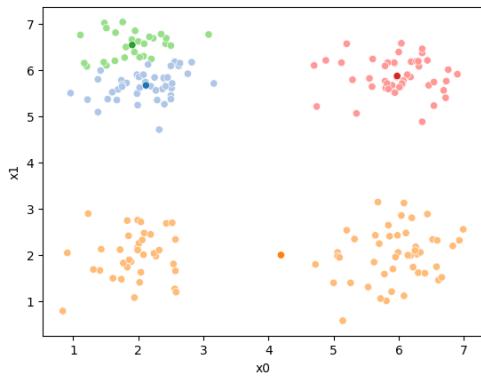
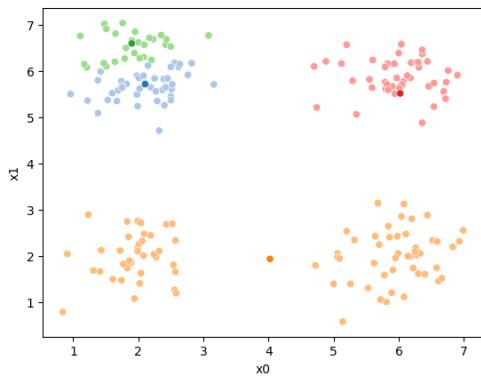
```
i =1
mu_list = [mu_to_df(mu0,i), mu_to_df(mu1,i)]
cur_old = str(i-1)
cur_new = str(i)
while sum(df[cur_old] != df[cur_new]) >0:
    cur_old = cur_new
    i +=1
    cur_new = str(i)
    # update the means and plot with current generating assignments
    mu = df.groupby(cur_old)[data_cols].mean()
    mu_df = mu_to_df(mu,i)
    mu_list.append(mu_df)

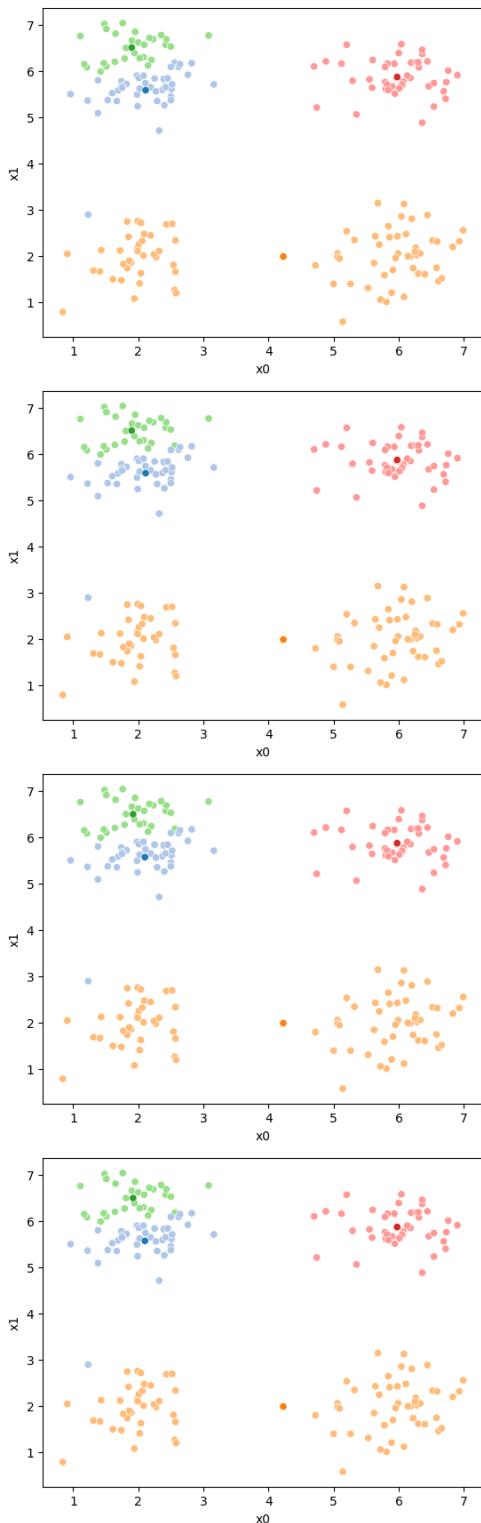
    fig = plt.figure()
    sfig = sns.scatterplot(data=df,x='x0',y='x1',hue=cur_old,palette=cmap_pt,legend=False)
    sns.scatterplot(data=mu_df,x='x0',y='x1',hue='class',palette=cmap_mu,ax=sfig,legend=False)

    # update the assignments 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)

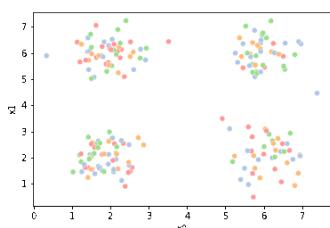
n_iter = i
```

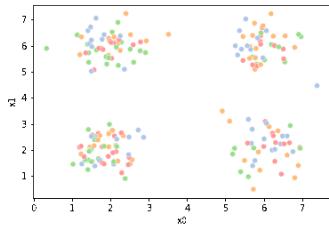
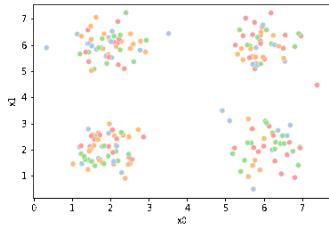
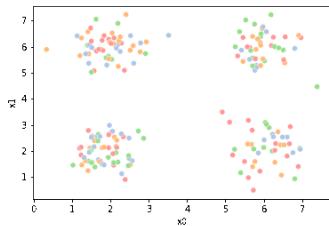






This algorithm is random. So each time we run it, it looks a little different.





22.2. Questions After Class

Ram token Opportunity

Contribute a question as an issue or contribute a solution to one of the try it yourself above.

23. Clustering

23.1. KMeans review

Review the notes from [Monday](#), including the gifs at the bottom for more illustration of what kmeans does under the hood. Ask any open questions on Prismia

23.2. Clustering with Sci-kit Learn

Read through this notebook and answer questions either by adding code or typing in markdown cells.

For hints about code, switch the markdown cells with the prompt to edit mode. There are hints in them stored as `html(markdown)` comments.

```
import seaborn as sns
import numpy as np
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
sns.set_theme(palette='colorblind')

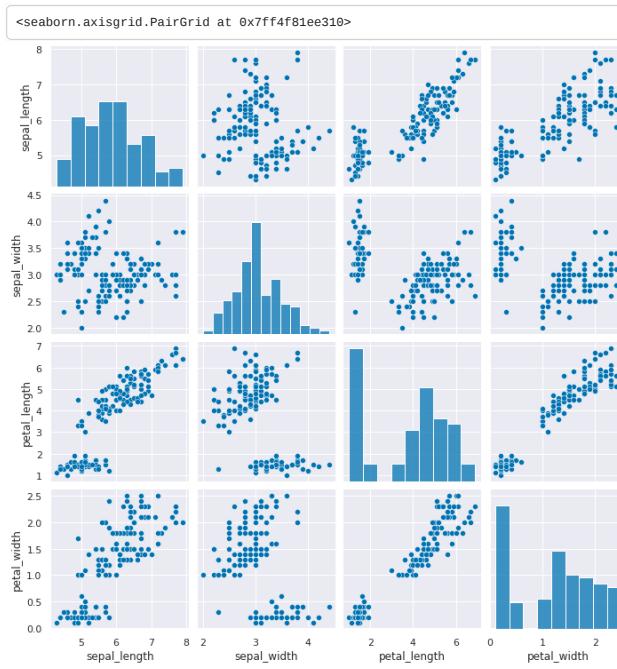
# set global random seed so that the notes are the same each time the site builds
np.random.seed(1103)
```

Load the iris data from seaborn

```
iris_df = sns.load_dataset('iris')
```

Create a grid of scatterplots of the data, without coloring the points differently

```
sns.pairplot(iris_df)
```



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.

Hint

We can either pick the measurements out or drop the species column. remember most data frame operations return a copy of the data frame.

We'll do this by picking out the measurement columns, but we could also drop the species for now.

```
measurement_cols = ['sepal_length','petal_length','sepal_width','petal_width']
iris_X = iris_df[measurement_cols]
# iris_X = iris_df.drop(columns=['species']) # equivalent to above
```

Create a Kmeans estimator object with 3 clusters, since we know that the iris data has 3 species of flowers. We refer to these three groups as classes in classification (the goal is to label the classes...) and in clustering we typically 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.

1 Question

How do we know there are three classes? didn't we just drop them?

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 flowers.

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

```
km = KMeans(n_clusters=3)
```

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. On Monday, that would be the last column of the data frame, the one with the labels.

Healthcare workers are at high risk of infection due to their close contact with patients.

This gives the labeled cluster by index, or the assignment, of each point

These are similar to the outputs in classification, except that in classification, it's able to tell us a specific species for each. Here it can only say clust 0, 1, or 2. It can't match those groups to the species of flower.

Now that we know what these are, we can give them to a variable.

```
cluster assignments = km.fit_predict(iris_X)
```

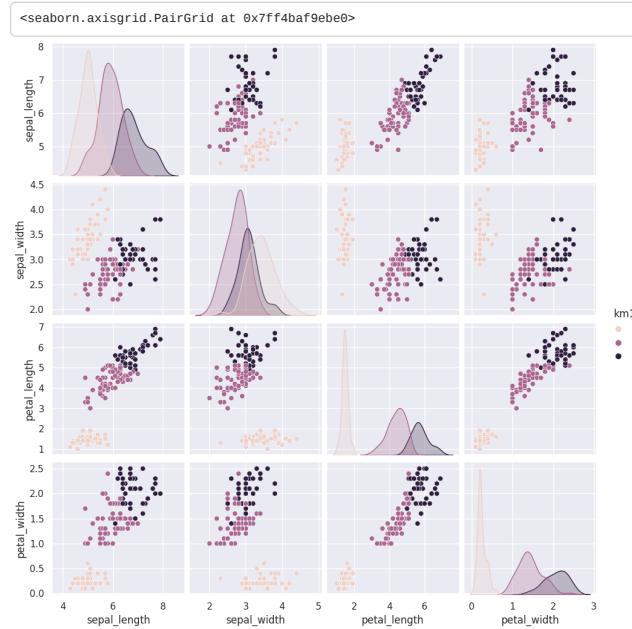
Use the `get_params` method to look at the parameters. Read the documentation to see what they mean.

```
{'algorithm': 'lloyd',
'copy_x': True,
'init': 'k-means++',
'max_iter': 300,
'n_clusters': 3,
'n_init': 10,
'random_state': None,
'tol': 0.0001,
'verbose': 0}
```

23.3. Visualizing the outputs

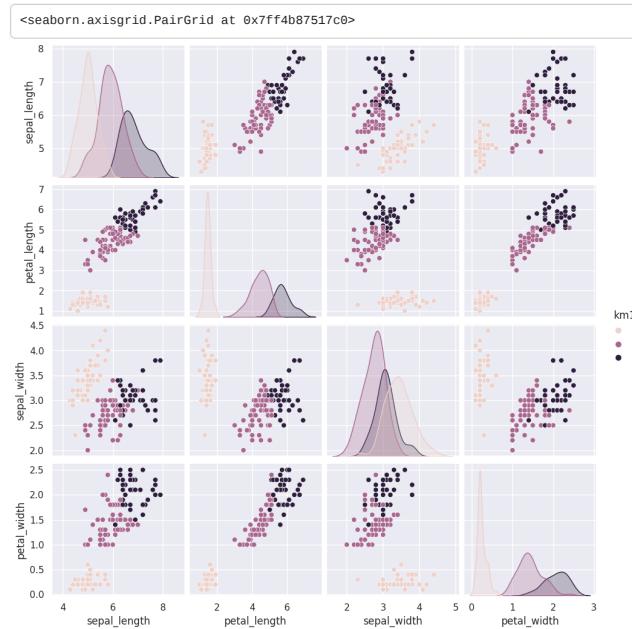
Add the predictions as a new column to the original `iris_df` and make a `pairplot` with the points colored by what the clustering learned.

```
iris_df['km1'] = cluster_assignments
sns.pairplot(data=iris_df, hue='km1')
```



We can use the `vars` parameter to plot only the measurement columns and not the cluster labels. We didn't have to do this before, because `species` is strings, but the cluster predictions are also numerical, so by default seaborn plots them.

```
iris_df['km1'] = cluster_assignments
sns.pairplot(data=iris_df, hue='km1', vars=measurement_cols)
```

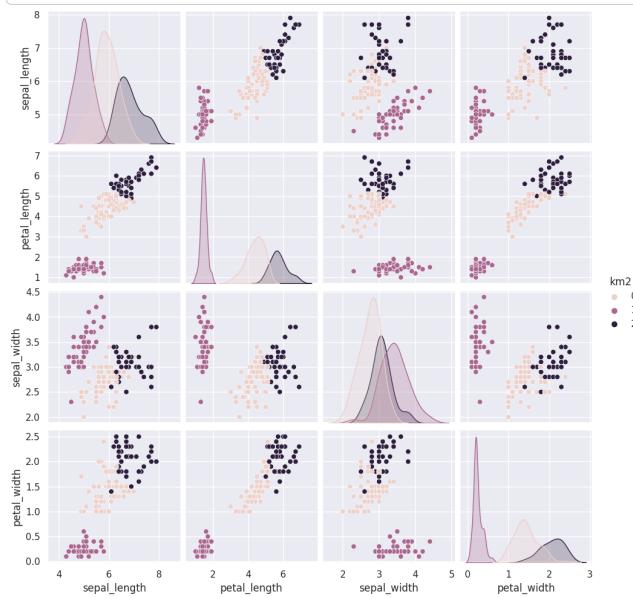


23.4. Clustering Persistence

We can run kmeans a few more times and plot each time and/or compare with a neighbor/ another group.

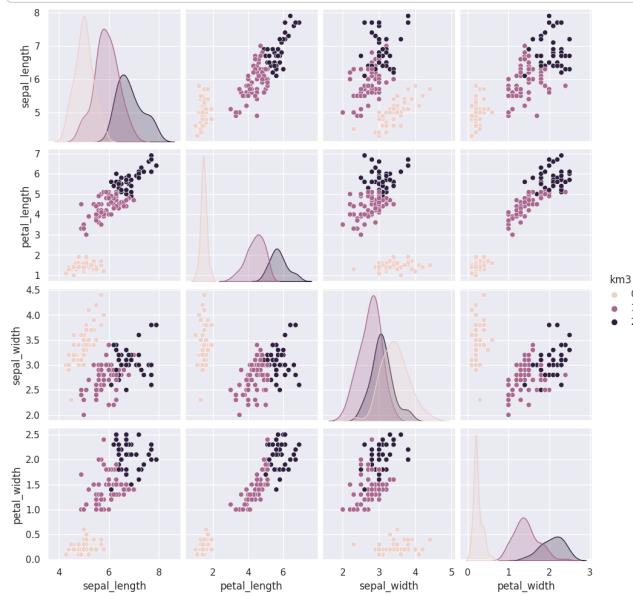
```
iris_df['km2'] = km.fit_predict(iris_X)
sns.pairplot(data=iris_df, hue='km2', vars=measurement_cols)
```

```
<seaborn.axisgrid.PairGrid at 0x7ff4afb2fac0>
```



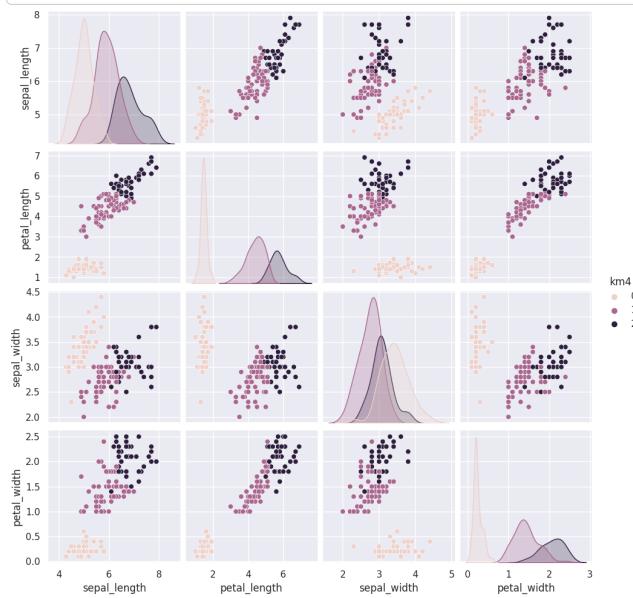
```
iris_df['km3'] = km.fit_predict(iris_X)
sns.pairplot(data=iris_df, hue='km3', vars=measurement_cols)
```

```
<seaborn.axisgrid.PairGrid at 0x7ff4afad9310>
```



```
iris_df['km4'] = km.fit_predict(iris_X)
sns.pairplot(data=iris_df, hue='km4', vars=measurement_cols)
```

```
<seaborn.axisgrid.PairGrid at 0x7ff4afb0b550>
```



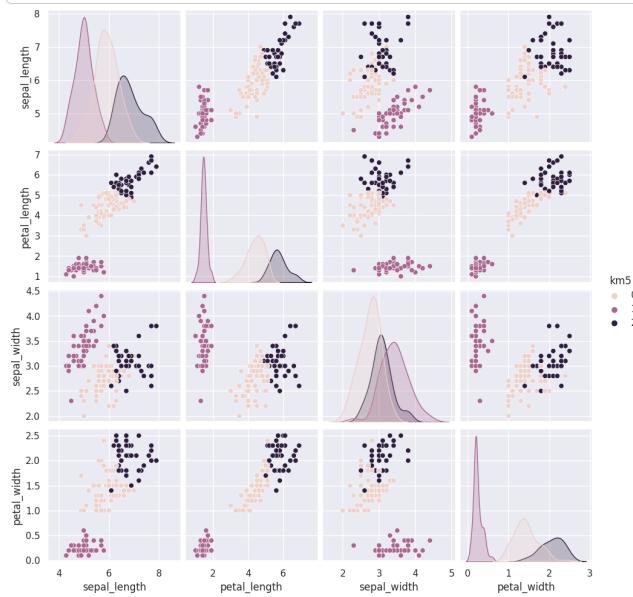
We could also use a loop (or list comprehension) to repeat kmeans multiple times.

```
for i in [5,6,7]:  
    iris_df['km' + str(i)] = km.fit_predict(iris_X)  
sns.pairplot(data=iris_df, hue='km5', vars=measurement_cols)
```

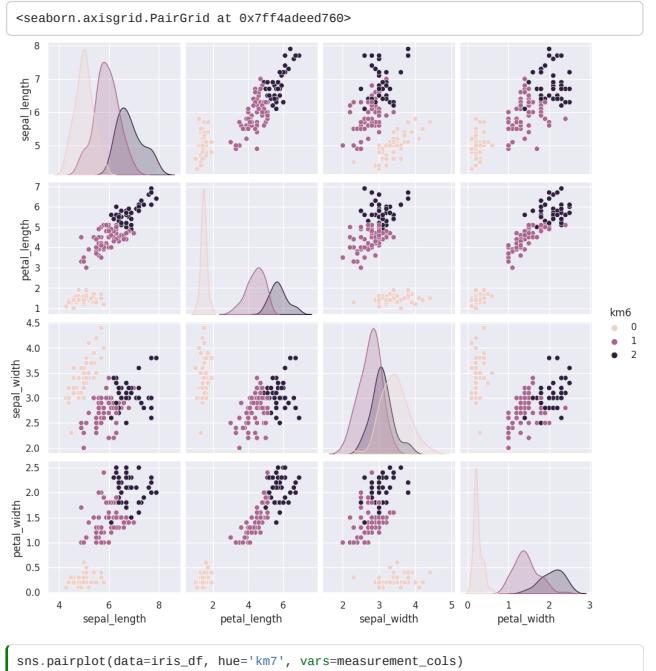
Tip

using the `i` as a loop variable here makes sense since we're actually just repeating for the sake of repeating

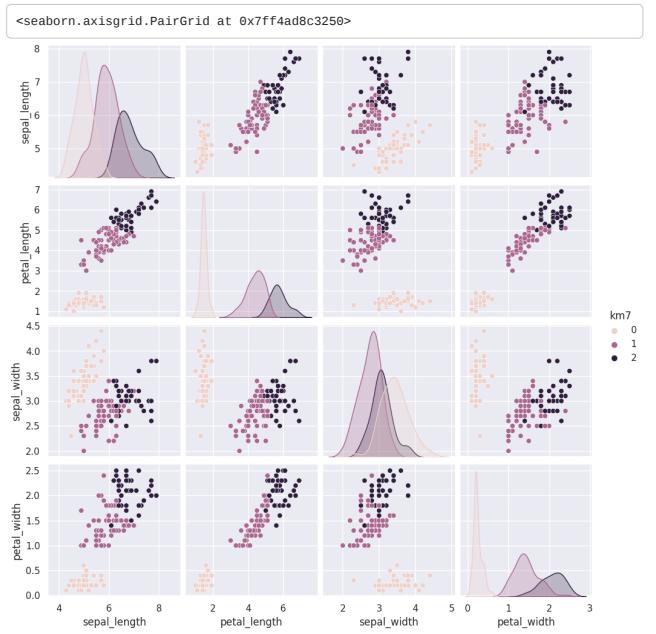
```
<seaborn.axisgrid.PairGrid at 0x7ff4ae7476d0>
```



```
sns.pairplot(data=iris_df, hue='km6', vars=measurement_cols)
```



```
sns.pairplot(data=iris_df, hue='km6', vars=measurement_cols)
```



The grouping of the points stay the same across different runs, but which color each group gets assigned to changes. Look at the 5th time compared to the ones before and 6 compared to that. Which blob is which color changes.

Today, we saw that the clustering solution was pretty similar each time in terms of which points were grouped together, but the labeling of the groups (which one was each number) was different each time. We also saw that clustering can only number the clusters, it can't match them with certainty to the species. This makes evaluating clustering somewhat different, so we need new metrics.

What might be our goal for evaluating clustering? We'll start from evaluating clustering on Friday.

23.5. Questions Before and After Class

23.5.1. How can I do linear regression with multiple features

We did that once, with the diabetes dataset, but basically its the same as how above we pick out the measurement columns as the X for clustering.

23.5.2. How can we do multiple iterations of KMeans easily?

I did two ways above, but we'll also see more ways to run different iterations of models next week, when we learn optimizing models. I wanted to cover a few different types before we optimize though, so that you can practice optimizing with any type you're most interested in.

23.5.3. If my results do not form good distinct clusters what does it mean?

This could be that the data is not very separable, or it could be that the dimensions that you're looking at are not the best directions to separate the clusters or it could be the wrong number of clusters.

It's important to consider that it could be a visual artifact of 2d screens and high dimensional data. We'll see scores that can help us check to differentiate.

23.5.4. Are there more ways to tell how accurate the model is

Yes! We'll talk about them Friday. If you want to get a head start, see the [clustering evaluation] section of the scikit learn documentation.

24. Evaluating Clustering

```
import seaborn as sns
import string
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn import metrics
import pandas as pd
```

We'll continue with the iris dataset because it's visually clear.

```
measurement_cols = ['sepal_length', 'petal_length', 'sepal_width', 'petal_width']

iris_df = sns.load_dataset('iris')
iris_X = iris_df[measurement_cols]
```

24.1. Clustering with KMeans

How do we tell if this is good?

One way to intuitively think about a good clustering solution is that every point should be close to points in the same cluster and far from points in other clusters. By definition with Kmeans, they will always be closer to points in the same cluster, but we also want that the clusters aren't just touching, but actually spaced apart, if the clustering actually captures meaningfully different groups.

24.2. Silhouette Score

The [Silhouette score](#) computes a ratio of how close points are to points in the same cluster vs other clusters.

$\forall s \equiv \frac{b-a}{\max(a,b)}$

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.

We can calculate this score for each point and get the average.

If the cluster is really tight, all of the points are close, then σ will be small.

If the clusters are really far apart, b will be large. If both of these are true then $b-a$ will be close to the value of b and the denominator will be b, so the score will be 1.

If the clusters are spread out and close together, then a and b will be close in value, and the s will be close to 0. These are overlapping clusters, or possibly too many clusters for this data.

Let's check our clustering solution:

Further Reading

See the [user guide](#) for more detail on this score including citations to source materials and advantages and disadvantages.

```
metrics.silhouette_score(iris_X, km3.labels_)

0.5528190123564098

km2 = KMeans(n_clusters=2)
km4 = KMeans(n_clusters=4)
km2.fit(iris_X)
km4.fit(iris_X)

▼ KMeans
KMeans(n_clusters=4)

metrics.silhouette_score(iris_X, km2.labels_)

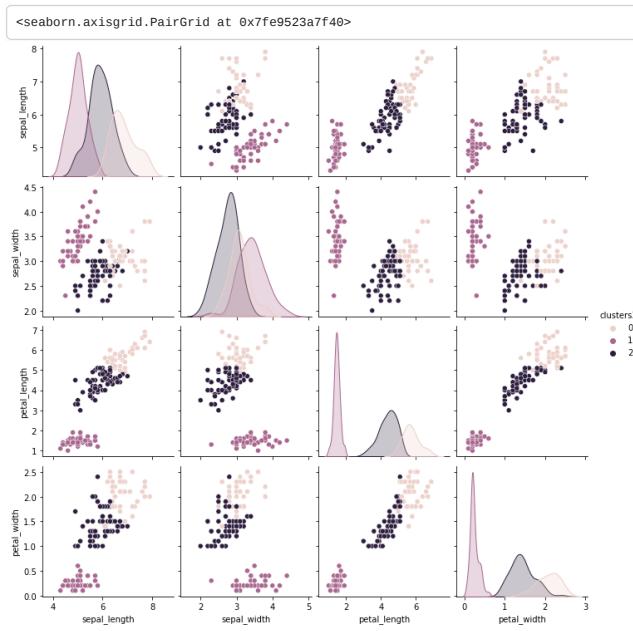
0.6810461692117464

metrics.silhouette_score(iris_X, km4.labels_)

0.49805050499728815

iris_df['clusters3'] = km3.labels_

sns.pairplot(data= iris_df,hue= 'clusters3',
# read docs to figure out hwy it didnt' plot
```



24.3. What's a good Silhouette Score?

To think through what a good silhouette score is, we can apply the score to data that represents different scenarios.

First, I'll re-sample some data like we used on Monday, but instead of applying K-means and checking the score of the actual algorithm, we'll add a few different scenarios and add that score.

Further Reading

Other types of clustering: [sklearn overview](#)

[classifier comparison](#)

```
K = 4
N = 200
classes = list(string.ascii_uppercase[:K])
mu = {c: i for c, i in zip(classes, [[2,2], [6,6], [2,6],[6,2]])}

# sample random cluster assignments
target = np.random.choice(classes,N)

# sample points with different means according to those assignments
data = [np.random.multivariate_normal(mu[c],.25*np.eye(2)) for c in target]
df = pd.DataFrame(data = data,columns = ['x'+ str(i) for i in range(2)])

# save the true assignments
df['true'] = target

# random assignments, right number of clusters
df['random'] = np.random.choice(classes,N)

# random assignments, way too many clusters
charsK4 = list(string.ascii_uppercase[:K*4])
df['random10'] = np.random.choice(charsK4,N)

# Kmeans with 2x number of clusters
kmrK2 = KMeans(K*2)
kmrK2.fit(df[['x0','x1']])
df['km' + str(K*2)] = kmrK2.labels_

# assign every point to its own cluster
df['id'] = list(range(N))

df.head()
```

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

```
sns.pairplot(data =df, hue='char')
```

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

Try it yourself

Compute the score for each and make plots.

24.4. 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.

Formally mutual information uses the joint distribution between the true labels and the cluster assignments to see how much they co-occur. If they're the exact same, then they have maximal mutual information. If they're completely and independent, then they have 0 mutual information. Mutual information is related to entropy in physics.

Further Reading

Sklearn provides many [mutual information based scores](#). See the user guide for definitions of each, pros and cons and examples.

The `mutual_info_score` method in the `metrics` module computes mutual information.

```
metrics.mutual_info_score(iris_df['species'], km2.labels_)
```

```
metrics.mutual_info_score(iris_df['species'], km3.labels_)
```

```
metrics.mutual_info_score(iris_df['species'], km4.labels_)
```

There's some random chance of getting things correct, so the adjusted mutual information corrects for that.

👉 Try it yourself

See how adjusted mutual information and mutual information compare on the iris data, the synthetic data above, or even use it in assignment 9.

24.5. Questions After Class

24.5.1. How I would use all of those different clustering algorithms

One nice thing about `sklearn` is that in code, they all work basically the same way. The [clustering overview](#) figure that was saw has the code for it in the `demo` section of the documentation. So you can download, run, and see all of them.

In a practical case, we choose by combining domain knowledge, exploratory data analyses (eg looking at the data) and the comparison techniques we'll see in two weeks.

24.5.2. How exactly do I find numbers of clusters

We'll look at strategies for finding the optimal number of clusters next week.

24.5.3. what do you mean by the data is represented in the 4th dimension?

the iris data has 4 *features* petal length, petal width, sepal length, sepal width. You can think of the features, or columns of the dataset as dimensions of the data. Each plot shows two of them, since it's a 2 dimensional plot.

24.5.4. How do we utilize these clusters once we have them?

Clustering can be used to find subroups in a dataset. For example one of my lab mate in grad school was collaborating with doctors to determine if COPD was actually many diseases with similar symptoms by clustering patients. With the results of the clusters, doctors could use those as hypotheses to look for underlying mechanisms of the disease or, they could use the cluster parameters to define classifier. This classifier would allow them to predict how one patient's disease would progress over time based on other patients that were similar to them.

We can also use clustering to just see if there are groups that are actually different or not. For example, we could use the NBA player stats data to see if there are different types of players, statistically. We know that players have a "position" that they play, but they may or may not be discretely different statistically, it could be sort of a smooth variation across all players. They could also have more "types" of players than there are positions.

24.5.5. How do we interpret the cluster labels?

We can't know for sure what clusters represent exactly. We can interpret them based on what we know about the features, or if we have labels we can use those. We can use visual inspection to figure it out.

24.5.6. How does Kmeans work in the simplest terms?

K means splits the data into K groups based on what points are close together (or most similar). If K = 3, then it will find 3 groups that are most similar.

💡 Important

if you don't see your question and the above don't answer it, then I misunderstood your question (the last two are my significant rephrasing of questions I had trouble with). If that's the case, **please** reach out to help me understand what you're confused about.

25. ML Task Review and Cross Validation

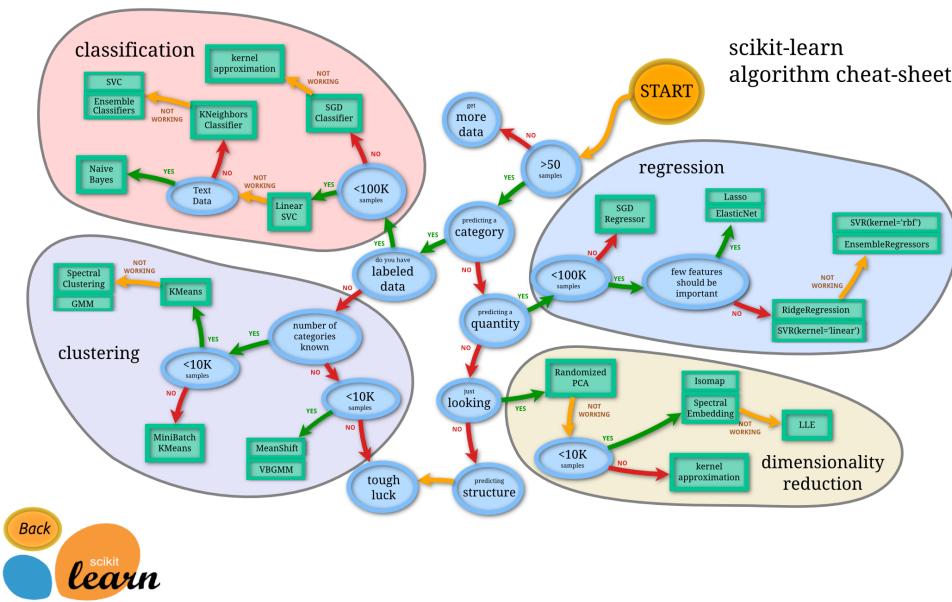
25.1. Relationship between Tasks

We learned classification first, because it shares similarities with each regression and clustering, while regression and clustering have less in common.

Classification is supervised learning for a categorical target.

Regression is supervised learning for a continuous target. Clustering is unsupervised learning for a categorical target.

Sklearn provides a nice flow chart for thinking through this.



Predicting a category is another way of saying categorical target. Predicting a quantity is another way of saying continuous target. Having labels or not is the difference between

The flowchart assumes you know what you want to do with data and that is the ideal scenario. You have a dataset and you have a goal. For the purpose of getting to practice with a variety of things, in this course we ask you to start with a task and then find a dataset. Assignment 9 is the last time that's true however. Starting with Assignment 10 and the last portfolios, you can choose and focus on a specific application domain and then choose the right task from there.

Thinking about this, however, you use this information to move between the tasks within a given type of data. For example, you can use the same data for clustering as you did for classification. Switching the task changes the questions though: classification evaluation tells us how separable the classes are given that classifiers decision rule. Clustering can find other subgroups or the same ones, so the evaluation we choose allows us to explore this in more ways.

Regression requires a continuous target, so we need a dataset to be suitable for that, we can't transform from the classification dataset to a regression one. However, we can go the other way and that's how some classification datasets are created.

The UCI [adult](#) Dataset is a popular ML dataset that was derived from census data. The goal is to use a variety of features to predict if a person makes more than \50k per year or not. While income is a continuous value, they applied a threshold (\50k) to it to make a binary variable. The dataset does not include income in dollars, only the binary indicator.

Further Reading

Recent work reconstructed the dataset with the continuous valued income. Their [repository](#) contains the data as well as links to their paper and a video of their talk on it.

25.2. Cross Validation

This week our goal is to learn how to optimize models. The first step in that is to get a good estimate of its performance.

We have seen that the test train splits, which are random, influence the performance.

```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn import metrics
```

We'll use the Iris data with a decision tree.

```
iris_df = sns.load_dataset('iris')
iris_X = iris_df.drop(columns=['species'])
iris_y = iris_df['species']

dt = tree.DecisionTreeClassifier()
```

We can split the data, fit the model, then compute a score, but since the splitting is a randomized step, the score is a random variable.

For example, if we have a coin that we want to see if it's fair or not. We would flip it to test. One flip doesn't tell us, but if we flip it a few times, we can estimate the probability it is heads by counting how many of the flips are heads and dividing by how many flips.

We can do something similar with our model performance. We can split the data a bunch of times and compute the score each time.

`cross_val_score` does this all for us.

It takes an estimator object and the data.

By default it uses 5-fold cross validation. It splits the data into 5 sections, then uses 4 of them to train and one to test. It then iterates through so that each section gets used for testing.

```
cross_val_score(dt, iris_X, iris_y,
```

```
array([0.96666667, 0.96666667, 0.9       , 1.       , 1.       ])
```

We get back a score for each section or “fold” of the data. We can average those to get a single estimate.

```
{ np.mean(cross_val_score(dt,iris_X,iris_y,))
```

```
0.9666666666666668
```

We can use more folds.

```
{ cross_val_score(dt,iris_X,iris_y,cv=10)
```

```
array([1.       , 0.93333333, 1.       , 0.93333333, 0.93333333, 0.86666667, 0.93333333, 1.       , 1.       , 1.       ])
```

💡 Try it yourself

What is the equivalent `train_size` for 5 fold? what about 10-fold?

```
{ np.mean(cross_val_score(dt,iris_X,iris_y,cv=10))
```

```
0.96
```

We can use *any* estimator object here.

```
{ km = KMeans(n_clusters=3)
```

```
{ cross_val_score(km,iris_X,)
```

```
array([-9.062      , -14.93195873, -18.93234207, -23.70894258, -19.55457726])
```

25.3. Notes

- Assignments 9 assesses up to level 2 for classification
- Heads up: Assignment 10 and 11 ask you to explore your work from 2 of (7,8,9) by optimizing the parameter and comparing different models for the same task. So the dataset selection problem is going away, little by little.

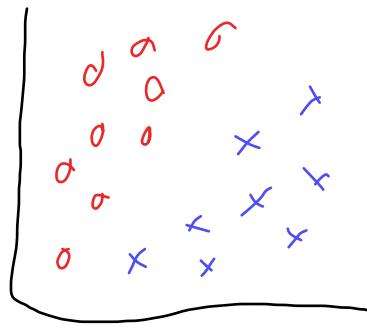
26. SVM and Parameter Optimizing

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn import cluster
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import tree
```

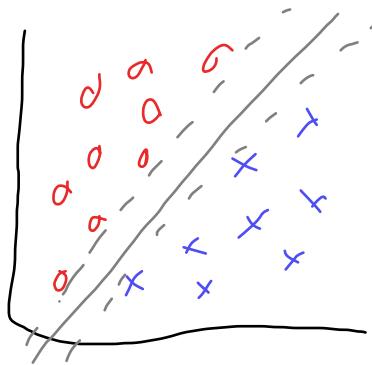
26.1. Support Vectors

26.1.1. Basic Idea

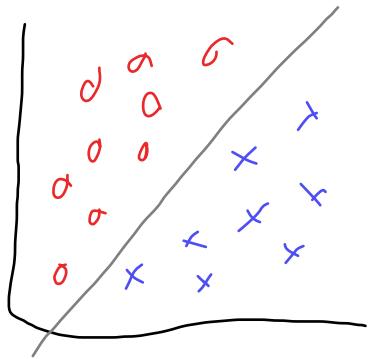
Imagine we have data that is like this



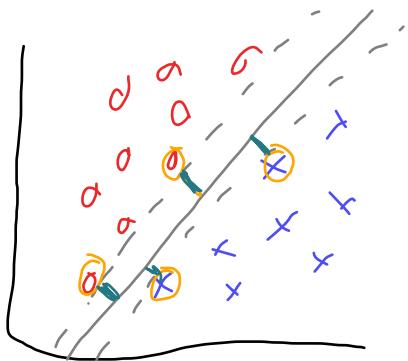
We might want to choose a decision boundary to separate it. We could choose any one of these three gray lines and get 100% training accuracy.



We could say that the best one is the solid one because it best separates the data.

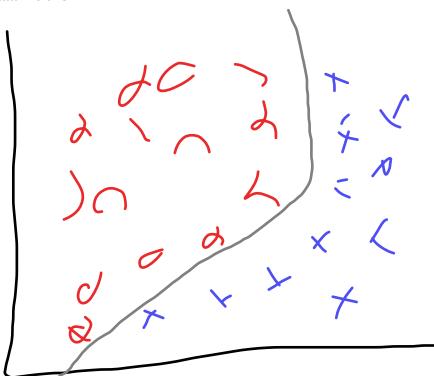


SVM does this, it finds the 'support vectors' which are the points of each class closer to the others and then finds the decision boundary that has the maximum margin, where the margin is the space between the boundary and each class.



When SVM is looking only for straight lines, it's called linear SVM, but SVM can look for different type of boundaries. We do this by changing the kernel function. A popular one is called the radial basis function or `rbf` it allows smooth curvy lines.

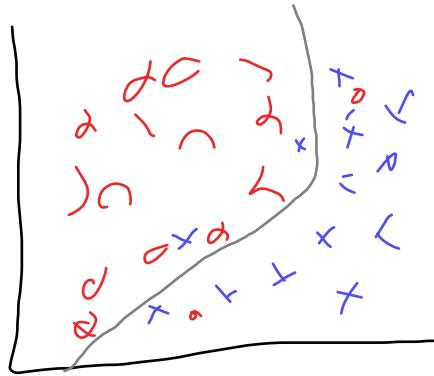
So that the SVM can work on data like this:



Note

Additional parameters control how smooth or wavy that line can be.

It can also handle data that is not perfectly separable like the following by minimizing the number of errors and maximizing the margin.



26.1.2. SVM in Sklearn

First we'll load the data and separate the features and target ($\setminus X$) and ($\setminus y$)

```
iris_df = sns.load_dataset('iris')
iris_X = iris_df.drop(columns='species')
iris_y = iris_df['species']
```

Next, we will split the data into test and train.

```
iris_X_train, iris_X_test, iris_y_train, iris_y_test =
train_test_split(iris_X,iris_y)
```

Fitting the model is just like other models we have seen:

1. instantiate the object
2. fit the model
3. score the model on the test dat

```
svm_clf = svm.SVC()
svm_clf.fit(iris_X_train, iris_y_train)
svm_clf.score(iris_X_test, iris_y_test)
```

```
0.9736842105263158
```

We see that this fits pretty well with the default parameters.

26.2. Grid Search Optimization

We can optimize, however to determine the different parameter settings.

A simple way to do this is to fit the model for different parameters and score for each and compare.

We'll focus on the kernel, which controls the type of line, and ($\setminus C$) which controls the regularization.

```
param_grid = {'kernel':['linear','rbf'], 'C':[.5, 1, 10]}
svm_opt = GridSearchCV(svm_clf,param_grid,)
```

The `GridSearchCV` object is constructed first and requires an estimator object and a dictionary that describes the parameter grid to search over. The dictionary has the parameter names as the keys and the values are the values for that parameter to test.

The `fit` method on the Grid Search object fits all of the separate models.

```
svm_opt.fit(iris_X_train,iris_y_train)
```

```
> GridSearchCV
> estimator: SVC
  > SVC
```

Then we can look at the output.

```
svm_opt.cv_results_
```

```

{'mean_fit_time': array([0.00156031, 0.00157475, 0.00147996, 0.00153718, 0.0014936
    , 0.00149393]),
 'std_fit_time': array([1.13108307e-04, 2.33343468e-05, 1.87443710e-05,
 1.03437400e-05,
 3.15339014e-05, 1.9236198e-05]),
 'mean_score_time': array([0.0011847, 0.00111156, 0.00105352, 0.00110064,
 0.00106411,
 0.00106015]),
 'std_score_time': array([1.10560266e-04, 3.41830720e-05, 2.39443483e-05,
 1.36733419e-05,
 3.86080298e-05, 2.13597500e-05]),
 'param_C': masked_array(data=[0.5, 0.5, 1, 1, 10, 10],
 mask=[False, False, False, False, False, False],
 fill_value='?',
 dtype=object),
 'param_kernel': masked_array(data=['linear', 'rbf', 'linear', 'rbf', 'linear',
 'rbf'],
 mask=[False, False, False, False, False, False],
 fill_value='?',
 dtype=object),
 'params': [{'C': 0.5, 'kernel': 'linear'},
 {'C': 0.5, 'kernel': 'rbf'},
 {'C': 1, 'kernel': 'linear'},
 {'C': 1, 'kernel': 'rbf'},
 {'C': 10, 'kernel': 'linear'},
 {'C': 10, 'kernel': 'rbf'}],
 'split0_test_score': array([0.95652174, 0.95652174, 0.95652174, 0.95652174,
 0.95652174,
 0.95652174]),
 'split1_test_score': array([1.        , 0.95652174, 1.        , 0.95652174,
 0.95652174,
 1.        ]),
 'split2_test_score': array([0.86363636, 0.81818182, 0.90909091, 0.86363636,
 0.90909091,
 0.95454545]),
 'split3_test_score': array([1., 1., 1., 1., 1., 1.]),
 'split4_test_score': array([1.        , 1.        , 1.        , 1.        ,
 0.95454545,
 1.        ]),
 'mean_test_score': array([0.96403162, 0.94624506, 0.97312253, 0.95533597,
 0.95533597,
 0.98221344]),
 'std_test_score': array([0.05294673, 0.06691876, 0.03617411, 0.04980237,
 0.02876428,
 0.02179296]),
 'rank_test_score': array([3, 6, 2, 4, 4, 1], dtype=int32)}

```

We note that this is a dictionary, so to make it more readable, we can make it a DataFrame.

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_kernel	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score
0	0.001560	0.000113	0.001118	0.000111	0.5	linear	{'C': 0.5, 'kernel': 'linear'}	0.956522	1.000000	0.863636	1
1	0.001575	0.000023	0.001112	0.000034	0.5	rbf	{'C': 0.5, 'kernel': 'rbf'}	0.956522	0.956522	0.818182	1
2	0.001480	0.000019	0.001054	0.000024	1	linear	{'C': 1, 'kernel': 'linear'}	0.956522	1.000000	0.909091	1
3	0.001537	0.000010	0.001101	0.000014	1	rbf	{'C': 1, 'kernel': 'rbf'}	0.956522	0.956522	0.863636	1
4	0.001494	0.000032	0.001064	0.000039	10	linear	{'C': 10, 'kernel': 'linear'}	0.956522	0.956522	0.909091	1
5	0.001494	0.000019	0.001060	0.000021	10	rbf	{'C': 10, 'kernel': 'rbf'}	0.956522	1.000000	0.954545	1

It also has a `best_estimator_` attribute, which is an estimator object.

```

type(svm_opt.best_estimator_)

sklearn.svm._classes.SVC

```

This is the model that had the best cross validated score among all of the parameter settings tested.

```

svm_opt.best_estimator_.score(iris_X_test,iris_y_test)

0.9736842105263158

```

We can then use this model on the test data.

Try it Yourself

Find the best criterion, max depth, and minimum number of samples per leaf

```

dt = tree.DecisionTreeClassifier()
params_dt = {'criterion':['gini','entropy'],'max_depth':[2,3,4],
'min_samples_leaf':list(range(2,20,2))}

```

To do this, we do just as we did above, instantiate and fit the model.

```

dt_opt = GridSearchCV(dt,params_dt)
dt_opt.fit(iris_X_train,iris_y_train)

> GridSearchCV
> estimator: DecisionTreeClassifier
>     DecisionTreeClassifier
.....
```

Then we can use the `best_params_` attribute to see the best parameter settings.

```
{ dt_opt.best_params_
{'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 2}
```

26.3. Questions after class

26.3.1. Can this be used on more types of machine learning than just decision trees and svm?

Yes, this can be used on any estimator in scikit learn. It can even be used on other models that adhere to the required [API](#).

GridSearchCV repeatedly:

- sets the parameter values from param_grid
- runs cross_val_score on the data

27. Model Comparison

To compare models, we will first optimize the parameters of two different models and look at how the different parameters settings impact the model comparison. Later, we'll see how to compare across models of different classes.

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn import cluster
from sklearn import svm
from sklearn import tree
# import the whole model selection module
from sklearn import model_selection
sns.set_theme(palette='colorblind')
```

We'll use the iris data again.

```
{ iris_X, iris_y = datasets.load_iris(return_X_y=True)
```

Remember, we need to split the data into training and test. The cross validation step will help us optimize the parameters, but we don't want *data leakage* where the model has seen the test data multiple times. So, we split the data here for train and test and the cross validation splits the training data into train and "test" again, but this test is better termed validation.

```
{ iris_X_train, iris_X_test, iris_y_train, iris_y_test =
model_selection.train_test_split(
    iris_X,iris_y, test_size=.2)
```

Then we can make the object, the parameter grid dictionary and the Grid Search object. We split these into separate cells, so that we can use the built in help to see more detail.

```
{ dt = tree.DecisionTreeClassifier()
params_dt = {'criterion':['gini','entropy'],
'max_depth':[2,3,4],
'min_samples_leaf':list(range(2,20,2))}

{ dt_opt = model_selection.GridSearchCV(dt,params_dt)
```

Then we fit the Grid search using the training data, and remember this actually resets the parameters and then cross validates multiple times.

```
{ dt_opt.fit(iris_X_train,iris_y_train)

> GridSearchCV
> estimator: DecisionTreeClassifier
> DecisionTreeClassifier
```

and look at the results

```
{ dt_opt.cv_results_
```

```
{'mean_fit_time': array([0.00043321, 0.00032129, 0.00031571, 0.00031662,
0.00031571,
 0.00031385, 0.00031204, 0.00030699, 0.00032072, 0.00034666,
 0.00032516, 0.00033627, 0.00032668, 0.0003222 , 0.00034633,
 0.00032625, 0.00031881, 0.00030398, 0.00032744, 0.00031311,
 0.0003274 , 0.0003212 , 0.00032439, 0.00031896, 0.00032763,
 0.00031948, 0.00030913, 0.00032988, 0.00032773, 0.00032196,
 0.00031948, 0.00031929, 0.00031438, 0.0003191 , 0.00032115,
 0.0003253 , 0.00034261, 0.0003449 , 0.00033703, 0.00033426,
 0.00033574 , 0.00033708, 0.00033274, 0.00032563, 0.00031972,
 0.0003511 , 0.00033922, 0.00033498, 0.00033283, 0.00033641,
 0.00033674, 0.00033231, 0.00032601, 0.00032163]),

'std_fit_time': array([1.90747499e-04, 9.55175172e-06, 1.05750399e-05,
8.49776282e-06,
 2.03254600e-05, 6.73944845e-06, 7.18807304e-06, 5.22697052e-06,
 1.24803871e-05, 2.69714541e-05, 7.41692571e-06, 1.39686432e-05,
 1.19983069e-05, 9.68531277e-06, 3.13034061e-05, 1.26362747e-05,
 7.45819656e-06, 8.23286661e-06, 6.32344325e-06, 1.31977982e-05,
 6.12768217e-06, 6.47937307e-06, 9.24326416e-06, 9.32018449e-06,
 1.15925380e-05, 7.07263802e-06, 7.68077951e-06, 9.42353769e-06,
 1.39694571e-05, 1.03700845e-05, 9.99312355e-06, 4.52719153e-06,
 6.05714588e-06, 4.90609593e-06, 1.02236822e-05, 1.37793563e-05,
 1.23573525e-05, 1.96365456e-05, 9.93905262e-06, 6.32595976e-06,
 1.25140459e-05, 1.60428600e-05, 1.06566437e-05, 5.79587671e-06,
 7.19534474e-06, 1.43658620e-05, 6.05864722e-06, 1.04415362e-05,
 1.07864317e-05, 6.04098294e-06, 8.55467657e-06, 6.07551170e-06,
 1.20646397e-05, 1.12588110e-05]),

'mean_score_time': array([0.00023766, 0.00020909, 0.00020151, 0.00020409,
0.0002039 ,
 0.00020552, 0.00020628, 0.00021095, 0.00021043, 0.00020976,
 0.0002068 , 0.00021029, 0.00020118, 0.00020847, 0.0002069 ,
```


We can reformat it into a dataframe for further analysis.

```
dt_df = pd.DataFrame(dt_opt.cv_results_)
dt_df.head(2)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_min_samples_leaf	params	split0_test_score	split1_test
0	0.000433	0.000191	0.000238	0.000048	gini	2		{'criterion': 'gini', 'max_depth': 2, 'min_sam...}	0.916667	
1	0.000321	0.000010	0.000209	0.000015	gini	2		{'criterion': 'gini', 'max_depth': 2, 'min_sam...}	0.916667	

Correction

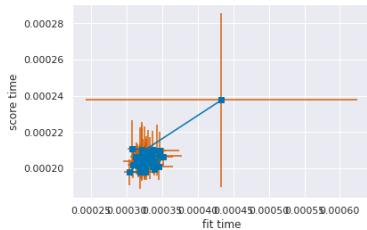
The parameters in this function were in the wrong order in this function in class

I changed the markers and the color of the error bars for readability.

```

plt.errorbar(x=dt_df['mean_fit_time'],y=dt_df['mean_score_time'],
             xerr=dt_df['std_fit_time'],yerr=dt_df['std_score_time'],
             marker='s',ecolor='r')
plt.xlabel('fit time')
plt.ylabel('score time')
# save the limits so we can reuse them
xmin, xmax, ymin, ymax = plt.axis()

```

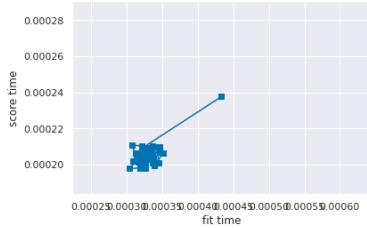


The “points” are at the mean fit and score times. The lines are the “standard deviation” or how much we expect that number to vary, since means are an estimate. Because the data shows an upward trend, this plot tells us that mostly, the models that are slower to fit are also slower to apply. This makes sense for decision trees, deeper trees take longer to learn and longer to traverse when predicting. Because the error bars mostly overlap the other points, this tells us that mostly the variation in time is not a reliable difference. If we re-ran the GridSearch, we could get them in different orders.

To interpret the error bar plot, let's look at a line plot of just the means, with the same limits so that it's easier to compare to the plot above.

```
plt.plot(dt_df['mean_fit_time'],
         dt_df['mean_score_time'], marker='s')
plt.xlabel('fit time')
plt.ylabel('score time')
# match the axis limits to above
plt.ylim(ymin, ymax)
plt.xlim(xmin, xmax)
```

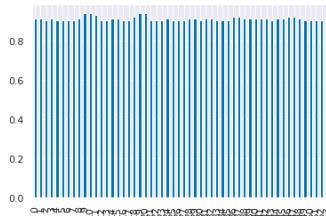
(0.00022338430927540292, 0.0006430288071796752)



this plot shows the mean times, without the error bars.

```
dt_df['mean_test_score'].plot(kind='bar')
```

<AxesSubplot:>



```
dt_df['mean_test_score']
```

```

0    0.916667
1    0.916667
2    0.908333
3    0.916667
4    0.908333
5    0.908333
6    0.908333
7    0.908333
8    0.916667
9    0.941667
10   0.941667
11   0.933333
12   0.908333
13   0.908333
14   0.916667
15   0.916667
16   0.908333
17   0.908333
18   0.925000
19   0.941667
20   0.941667
21   0.908333
22   0.908333
23   0.908333
24   0.916667
25   0.908333
26   0.908333
27   0.908333
28   0.916667
29   0.916667
30   0.908333
31   0.916667
32   0.916667
33   0.908333
34   0.908333
35   0.908333
36   0.925000
37   0.925000
38   0.916667
39   0.916667
40   0.916667
41   0.916667
42   0.916667
43   0.908333
44   0.916667
45   0.916667
46   0.925000
47   0.925000
48   0.916667
49   0.908333
50   0.908333
51   0.908333
52   0.908333
53   0.908333
Name: mean_test_score, dtype: float64

```

Now let's compare with a different model, we'll use the parameter optimized version for that model.

```

svm_clf = svm.SVC()
param_grid = {'kernel':['linear','rbf'], 'C':[.5, 1, 10]}
svm_opt = GridSearchCV(svm_clf,param_grid)

```

```

NameError                                 Traceback (most recent call last)
Input In [14], in <cell line: 3>()
      1 svm_clf = svm.SVC()
      2 param_grid = {'kernel':['linear','rbf'], 'C':[.5, 1, 10]}
----> 3 svm_opt = GridSearchCV(svm_clf,param_grid)

NameError: name 'GridSearchCV' is not defined

```

The error above is because we didn't import `GridSearchCV` directly today, we imported the whole `model_selection` module, so we have to use that in order to access the class.

```

svm_clf = svm.SVC()
param_grid = {'kernel':['linear','rbf'], 'C':[.5, .75, 1, 2, 5, 7, 10]}
svm_opt = model_selection.GridSearchCV(svm_clf,param_grid, cv=10)

```

```

type(model_selection)

```

```

module

```

```

dt_opt.__dict__

```

```
{'scoring': None,
'estimator': DecisionTreeClassifier(),
'n_jobs': None,
'refit': True,
'cv': None,
'verbose': 0,
'pre_dispatch': '2*n_jobs',
'error_score': nan,
'return_train_score': False,
'param_grid': {'criterion': ['gini', 'entropy'],
'max_depth': [2, 3, 4],
'min_samples_leaf': [2, 4, 6, 8, 10, 12, 14, 16, 18]},
'multimetric_': False,
'best_index_': 9,
'best_score_': 0.9416666666666667,
'best_params_': {'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 2},
'best_estimator_': DecisionTreeClassifier(max_depth=3, min_samples_leaf=2),
'refit_time_': 0.00032830238342285156,
'scorer_': <function sklearn.metrics._scorer_.passthrough_scorer(estimator,
*args, **kwargs)>,
'cv_results_': {'mean_fit_time': array([0.00043321, 0.00032129, 0.00031571,
0.00031662, 0.00031571,
0.00031385, 0.00031204, 0.00030699, 0.00032072, 0.00034666,
0.00032516, 0.00033627, 0.00032668, 0.00032222 , 0.00034633,
0.00032625, 0.00031881, 0.00030398, 0.00032744, 0.00033131,
0.0003274 , 0.0003212 , 0.00032439, 0.00031896, 0.00032763,
0.00031948, 0.00030913, 0.00032988, 0.00032773, 0.00032196,
0.00031948, 0.00032129, 0.00032438, 0.0003191 , 0.00032115,
0.0003253 , 0.00034261, 0.00034449 , 0.00033703, 0.00033426,
0.00033574, 0.00033706, 0.00033274, 0.00032563, 0.00031972,
0.0003511 , 0.00033922, 0.00033498, 0.00033283, 0.00033641,
0.00033674, 0.00033231, 0.00032601, 0.00032163]),
'std_fit_time_': array([1.90747499e-04, 9.55175172e-06, 1.05750399e-05,
8.40776282e-06,
2.03254600e-05, 6.73944845e-06, 7.18807304e-06, 5.22697052e-06,
1.24803871e-05, 2.69714541e-05, 7.41692571e-06, 1.39686432e-05,
```


This doesn't have attributes yet, even though they are the same type, because we have not fit it to data yet.

```
type(svm_opt), type(dt_opt)
```

```
(sklearn.model_selection._search.GridSearchCV,  
 sklearn.model_selection.GridSearchCV)
```

Now we can fit the model to the training data of this second model.

```
# fit the model and put the CV results in a dataframe
svm_opt.fit(iris_X_train,iris_y_train)
sv_df = pd.DataFrame(svm_opt.cv_results_)
```

```
sv df.head(2)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_kernel	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score
0	0.000471	0.000056	0.000252	0.000019	0.5	linear	{'C': 0.5, 'kernel': 'linear'}	1.0	0.916667	1.0	1.00000
1	0.000551	0.000013	0.000271	0.000007	0.5	rbf	{'C': 0.5, 'kernel': 'rbf'}	1.0	0.833333	1.0	0.91666

plt.errorbar(x=sv_df['mean_fit_time'], xerr=sv_df['std_fit_time'], y=sv_df['mean_score_time'], yerr=sv_df['std_score_time'])

<ErrorbarContainer object of 3 artists>

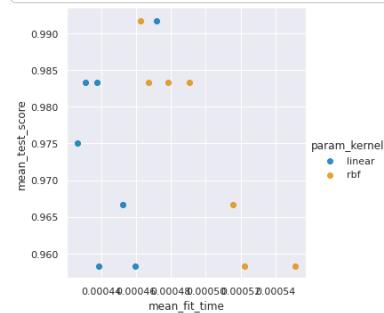
sv_df.columns

```
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_C', 'param_kernel', 'params', 'split0_test_score',
       'split1_test_score', 'split2_test_score', 'split3_test_score',
       'split4_test_score', 'split5_test_score', 'split6_test_score',
       'split7_test_score', 'split8_test_score', 'split9_test_score',
       'mean_test_score', 'std_test_score', 'rank_test_score'],
      dtype='object')
```

We can see if the models that take longer to fit or score perform better.

```
svm_time = sv_df.melt(id_vars=['param_C', 'param_kernel', 'params'],
                      value_vars=['mean_fit_time', 'std_fit_time',
                      'mean_score_time', 'std_score_time'])
sns.lmplot(data=sv_df, x='mean_fit_time', y='mean_test_score',
            hue='param_kernel', fit_reg=False)
```

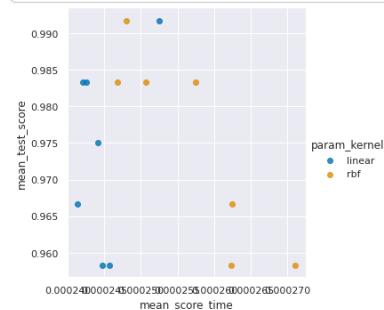
<seaborn.axisgrid.FacetGrid at 0x7fb76a9111f0>



This looks like mostly no.

```
sns.lmplot(data=sv_df, x='mean_score_time', y='mean_test_score',
            hue='param_kernel', fit_reg=False)
```

<seaborn.axisgrid.FacetGrid at 0x7fb76a9d08b0>



Again, for score time, the slower models don't appear to be better. Remember though the time differences weren't that different.

👉 Try it yourself

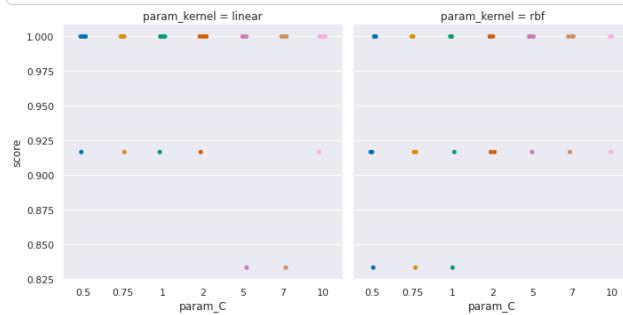
Try this same analysis for the decision tree, does it matter there?

```
sv_df_scores = sv_df.melt(id_vars=['param_C', 'param_kernel', 'params'],
                           value_vars=['split0_test_score',
                           'split1_test_score', 'split2_test_score', 'split3_test_score',
                           'split4_test_score'], value_name='score')
sv_df_scores.head()
```

	param_C	param_kernel	params	variable	score
0	0.5	linear	{'C': 0.5, 'kernel': 'linear'}	split0_test_score	1.0
1	0.5	rbf	{'C': 0.5, 'kernel': 'rbf'}	split0_test_score	1.0
2	0.75	linear	{'C': 0.75, 'kernel': 'linear'}	split0_test_score	1.0
3	0.75	rbf	{'C': 0.75, 'kernel': 'rbf'}	split0_test_score	1.0
4	1	linear	{'C': 1, 'kernel': 'linear'}	split0_test_score	1.0

```
sns.catplot(data=sv_df_scores,x='param_C',y='score',
            col='param_kernel')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb769fc5f10>
```

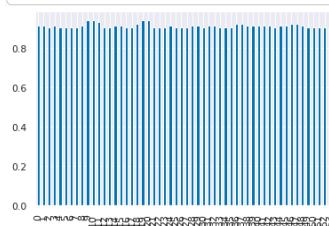


👉 Try it yourself

Try interpreting the plot above, what does it say? what can you conclude from it.

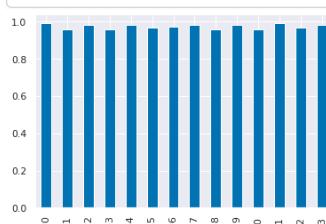
```
dt_df['mean_test_score'].plot(kind='bar')
```

```
<AxesSubplot:>
```



```
sv_df['mean_test_score'].plot(kind='bar')
```

```
<AxesSubplot:>
```



From these last two plots we see that the SVM performance is more sensitive to its parameters, where for the parameters tested, the decision tree is not impacted.

What can we say based on this? We'll pick up from here on Wednesday.

28. Model Selection

We'll pick up from where we left off on Monday.

💡 Further Reading

If you struggled to understand this code excerpt to fill in the comments, some generic strategies to understand code may help, beyond applying what we have covered in class.

The Programmer's brain is an overview of how brains work, as applied to programming, written for working developers. This means that it assumes you know most CS concepts and at least two programming languages. If you don't there may be some parts that do not make sense to you, but the general ideas should still make sense. The author is a professor who researchers how people learn programming and how to effectively teach it.

```

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn import cluster
from sklearn import svm
from sklearn import tree
# import the whole model selection module
from sklearn import model_selection
sns.set_theme(palette='colorblind')

# load and split the data
iris_X, iris_y = datasets.load_iris(return_X_y=True)
iris_X_train, iris_X_test, iris_y_train, iris_y_test =
model_selection.train_test_split(
    iris_X,iris_y, test_size = .2)

# create dt, set param grid & create optimizer
dt = tree.DecisionTreeClassifier()

params_dt = {'criterion':['gini','entropy'],
             'max_depth':[2,3,4,5,6],
             'min_samples_leaf':list(range(2,20,2))}

dt_opt = model_selection.GridSearchCV(dt,params_dt, cv=10)

# fit the model and optimize
dt_opt.fit(iris_X_train,iris_y_train)

# store the result in a dataframe
dt_df = pd.DataFrame(dt_opt.cv_results_)

# create svm, its parameter grid and optimizer
svm_clf = svm.SVC()
param_grid = {'kernel':['linear','rbf'], 'C':[.5, .75, 1, 2, 5, 7, 10]}
svm_opt = model_selection.GridSearchCV(svm_clf,param_grid, cv=10)

# fit the model and put the cv results in a dataframe
svm_opt.fit(iris_X_train,iris_y_train)
sv_df = pd.DataFrame(svm_opt.cv_results_)

```

We can compare how the best models fit during validation:

```
svm_opt.best_score_, dt_opt.best_score_
```

```
(0.9916666666666666, 0.983333333333332)
```

We see that the SVM does a little bit better.

We can also apply the best estimator from each model class (that is the best parameter settings for each SVM and Decision Tree) to the test data to get our final score.

```
svm_opt.best_estimator_.score(iris_X_test,iris_y_test)
```

```
0.9666666666666667
```

```
dt_opt.best_estimator_.score(iris_X_test,iris_y_test)
```

```
0.9333333333333333
```

We can also examine the results to think through this choice more clearly.

```
sv_df.head(2)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_kernel	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	
0	0.000460	0.000046	0.000249	0.000016	0.5	linear	{'C': 0.5, 'kernel': 'linear'}	1.0	1.0	1.0	1.0	1
1	0.000542	0.000018	0.000267	0.000007	0.5	rbf	{'C': 0.5, 'kernel': 'rbf'}	1.0	1.0	1.0	1.0	1

We can use EDA to understand how the score varied across all of the parameter settings we tried.

```
sv_df['mean_test_score'].describe()
```

```

count    14.000000
mean     0.984524
std      0.010770
min     0.958333
25%     0.983333
50%     0.991667
75%     0.991667
max     0.991667
Name: mean_test_score, dtype: float64

```

```
dt_df['mean_test_score'].describe()
```

```

count    90.000000
mean     0.968148
std      0.006193
min     0.958333
25%     0.966667
50%     0.966667
75%     0.966667
max     0.983333
Name: mean_test_score, dtype: float64

```

From this we see that in both cases the standard deviation (std) is really low. This tells us that the parameter changes didn't impact the performance much. Combined with the overall high accuracy this tells us that the data is probably really easy to classify. If the performance had been uniformly bad, it might have instead told us that we did not try a wide enough range of parameters.

To confirm how many parameter settings we have used we can check a couple different ways. First, above in the count of the describe.

We can also calculate directly from the parameter grids before we even do the fit.

```
n_combinations = 1
for param, vals in param_grid.items():
    n_combinations *= len(vals)
```

28.1. When do differences matter?

We can check calculate a confidence interval to determine more precisely when the performance of two models is meaningfully different.

This function calculates the 95% confidence interval. The range within which we are 95% confident the quantity we have estimated is truly within in. When we have more samples in the test set used to calculate the score, we are more confident in the estimate, so the interval is narrower.

```
def classification_confint(acc, n):
    """
    Compute the 95% confidence interval for a classification problem.
    acc -- classification accuracy
    n -- number of observations used to compute the accuracy
    Returns a tuple (lb,ub)
    """
    interval = 1.96*np.sqrt(acc*(1-acc)/n)
    lb = max(0, acc - interval)
    ub = min(1.0, acc + interval)
    return (lb,ub)
```

```
svm_opt.best_score_, dt_opt.best_score_
```

```
(0.9916666666666666, 0.983333333333333)
```

We can calculate the number of observations used to compute the accuracy using the size of the training data and the fact that we set it to 10-fold cross validation. That means that 10% (100/10) of the data was used for each fold and each validation set.

```
len(iris_X_train)*.1
```

```
12.0
```

```
split0_score = dt_df['split0_test_score'][dt_opt.best_index_]
classification_confint(split0_score ,len(iris_X_train)*.1)
```

```
(0.7602869057331314, 1.0)
```

Correction

since the best score is cross validated it actually uses the whole training data (by averaging 10 scores together).

```
classification_confint(dt_opt.best_score_,len(iris_X_train))
```

```
(0.9604278107904404, 1.0)
```

```
len(iris_y_test)
```

```
30
```

```
classification_confint(dt_opt.best_estimator_.score(iris_X_test,iris_y_test),len(iris_y_test))
```

```
(0.8440710066609742, 1.0)
```

If we take the exact value we can see how more samples narrows the interval.

```
classification_confint(.95,12)
```

```
(0.8266860375572443, 1.0)
```

```
classification_confint(.95,30)
```

```
(0.8720094022760863, 1.0)
```

We can further explore this by plotting directly from the calculation out of the function

```
n_list = list(range(5,200))
intervals = [1.96*np.sqrt(acc*(1-acc)/n) for n in n_list]
plt.plot(n_list, intervals)
```

```
Input In [18]
n_list = list(range(5,200))
^
SyntaxError: closing parenthesis ')' does not match opening parenthesis '('
```

28.2. Questions After Class

28.2.1. When would I choose to use an svm?

SVMs are often very accurate, and while in some cases a decision tree may be considered more interpretable, an SVM is not necessarily too hard to interpret.

If accuracy is important, the SVM is often a really good choice.

28.2.2. How can we tell how many parameter settings we tried?

We can calculate it in advance from the parameter grid or the length of the output DataFrame/lists in the dictionary.

29. Learning Curves

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn import cluster

from sklearn import naive_bayes
from sklearn import svm
from sklearn import tree
# import the whole model selection module
from sklearn import model_selection
sns.set_theme(palette='colorblind')
```

Today, we'll load a new dataset and use the default sklearn data structure for datasets. We get back the default data structure when we use a `load_` function without any parameters at all.

```
digits = datasets.load_digits()
```

This shows us that the type is defined by sklearn and they called it `bunch`:

```
type(digits)
```

```
sklearn.utils._bunch.Bunch
```

We can print it out to begin exploring it.

```
digits
```

```

{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
   [ 0.,  0.,  0., ..., 10.,  0.,  0.],
   [ 0.,  0.,  0., ..., 16.,  9.,  0.],
   ...,
   [ 0.,  0.,  1., ...,  6.,  0.,  0.],
   [ 0.,  0.,  2., ..., 12.,  0.,  0.],
   [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
'target': array([0, 1, 2, ..., 8, 9, 8]),
'frame': None,
'feature_names': ['pixel_0_0',
'pixel_0_1',
'pixel_0_2',
'pixel_0_3',
'pixel_0_4',
'pixel_0_5',
'pixel_0_6',
'pixel_0_7',
'pixel_1_0',
'pixel_1_1',
'pixel_1_2',
'pixel_1_3',
'pixel_1_4',
'pixel_1_5',
'pixel_1_6',
'pixel_1_7',
'pixel_2_0',
'pixel_2_1',
'pixel_2_2',
'pixel_2_3',
'pixel_2_4',
'pixel_2_5',
'pixel_2_6',
'pixel_2_7',
'pixel_3_0',
'pixel_3_1',
'pixel_3_2',
'pixel_3_3',
'pixel_3_4',
'pixel_3_5',
'pixel_3_6',
'pixel_3_7',
'pixel_4_0',
'pixel_4_1',
'pixel_4_2',
'pixel_4_3',
'pixel_4_4',
'pixel_4_5',
'pixel_4_6',
'pixel_4_7',
'pixel_5_0',
'pixel_5_1',
'pixel_5_2',
'pixel_5_3',
'pixel_5_4',
'pixel_5_5',
'pixel_5_6',
'pixel_5_7',
'pixel_6_0',
'pixel_6_1',
'pixel_6_2',
'pixel_6_3',
'pixel_6_4',
'pixel_6_5',
'pixel_6_6',
'pixel_6_7',
'pixel_7_0',
'pixel_7_1',
'pixel_7_2',
'pixel_7_3',
'pixel_7_4',
'pixel_7_5',
'pixel_7_6',
'pixel_7_7'],
'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
   [ 0.,  0., 13., ..., 15.,  5.,  0.],
   [ 0.,  3., 15., ..., 11.,  8.,  0.],
   ...,
   [ 0.,  4., 11., ..., 12.,  7.,  0.],
   [ 0.,  2., 14., ..., 12.,  0.,  0.],
   [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

   [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
   [ 0.,  0.,  0., ...,  9.,  0.,  0.],
```

```

[ 0.,  0.,  3., ...,  6.,  0.,  0.],
[ 0.,  0.,  1., ...,  6.,  0.,  0.],
[ 0.,  0.,  1., ...,  6.,  0.,  0.],
[ 0.,  0.,  0., ..., 10.,  0.,  0.]],

[[ 0.,  0.,  0., ..., 12.,  0.,  0.],
[ 0.,  0.,  3., ..., 14.,  0.,  0.],
[ 0.,  0.,  8., ..., 16.,  0.,  0.],
[ 0.,  9.,  16., ..., 0.,  0.,  0.],
[ 0.,  3.,  13., ..., 11.,  5.,  0.],
[ 0.,  0.,  0., ..., 16.,  9.,  0.]],

.....
[[ 0.,  0.,  1., ..., 1.,  0.,  0.],
[ 0.,  0.,  13., ..., 2.,  1.,  0.],
[ 0.,  0.,  16., ..., 16.,  5.,  0.],
[ 0.,  0.,  16., ..., 15.,  0.,  0.],
[ 0.,  0.,  2., ..., 6.,  0.,  0.]],

[[ 0.,  0.,  2., ..., 0.,  0.,  0.],
[ 0.,  0.,  14., ..., 15.,  1.,  0.],
[ 0.,  4.,  16., ..., 16.,  7.,  0.],
[ 0.,  0.,  0., ..., 16.,  2.,  0.],
[ 0.,  0.,  4., ..., 16.,  2.,  0.],
[ 0.,  0.,  5., ..., 12.,  0.,  0.]],

[[ 0.,  0.,  10., ..., 1.,  0.,  0.],
[ 0.,  2.,  16., ..., 1.,  0.,  0.],
[ 0.,  0.,  15., ..., 15.,  0.,  0.],
[ 0.,  4.,  16., ..., 16.,  6.,  0.],
[ 0.,  8.,  16., ..., 16.,  8.,  0.],
[ 0.,  1.,  8., ..., 12.,  1.,  0.]]],



'DESCR': """ _digits_dataset:\nOptical recognition of handwritten digits\n-----\n**Data Set Characteristics:**\n:Number of Instances: 1797 :Number of Attributes: 64\n:Attribute Information: 8x8 image of integer pixels in the range 0..16.\n:Missing Attribute Values: None\n:Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n:Date: July; 1998\n\nThis is a copy of the test set of the UCI ML hand-written digits\ndatasets\n-----\nThe data set contains images of hand-written digits: 10 classes\nwhere each class refers to a digit.\nPreprocessing programs made available by NIST were used to extract\nnormalized bitmaps of handwritten digits from a preprinted form.\nFrom a total of 43 people, 30 contributed to the training set and\ndifferent 13 to the test set. 32x32 bitmaps are divided into nonoverlapping\nblocks of 4x4 and the number of pixels are counted in each block.\nThis generates an input matrix of 8x8 where each element is an integer in the\nrange 0..16. This reduces dimensionality and gives invariance to\nsmall distortions.\nFor info on NIST preprocessing routines, see M. D. Garris,\nJ. L. Blue, G. nt. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet,\nand C. \nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR\n5469, 1994.\ntopic:: References\n- C. Kaynak (1995) Methods of Combining\nMultiple Classifiers and Their Applications to Handwritten Digit Recognition,\nMSc Thesis, Institute of Graduate Studies in Science and Engineering,\nBogazici University.\n- E. Alpaydin (1998) Cascading Classifiers,\nKybernetika.\n- Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai\nQin.\nLinear dimensionality reduction using relevance weighted LDA. School of\nElectrical and Electronic Engineering Nanyang Technological University.\n2005.\n- Claudio Gentile. A New Approximate Maximal Margin Classification\nAlgorithm. NIPS. 2000.\n"""

```

We note that it has key value pairs, and that the last one is called `DESCR` and is text that describes the data. If we send that to the print function it will be formatted more readably.

```

[ print(digits['DESCR'])

.. _digits_dataset:

Optical recognition of handwritten digits dataset
-----

**Data Set Characteristics:**\n:Number of Instances: 1797\n:Number of Attributes: 64\n:Attribute Information: 8x8 image of integer pixels in the range 0..16.\n:Missing Attribute Values: None\n:Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n:Date: July; 1998\n\nThis is a copy of the test set of the UCI ML hand-written digits datasets\nhttps://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits\n\nThe data set contains images of hand-written digits: 10 classes where\neach class refers to a digit.\n\nPreprocessing programs made available by NIST were used to extract\nnormalized bitmaps of handwritten digits from a preprinted form.\nFrom a total of 43 people, 30 contributed to the training set and different 13\nto the test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4 and the number of pixels are counted in each block.\nThis generates an input matrix of 8x8 where each element is an integer in the range\n0..16. This reduces dimensionality and gives invariance to small\ndistortions.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,\n1994.\n.. topic:: References\n- C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their\nApplications to Handwritten Digit Recognition, MSc Thesis, Institute of\nGraduate Studies in Science and Engineering, Bogazici University.\n- E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n- Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.\nLinear dimensionality reduction using relevance weighted LDA. School of\nElectrical and Electronic Engineering Nanyang Technological University.\n2005.\n- Claudio Gentile. A New Approximate Maximal Margin Classification\nAlgorithm. NIPS. 2000.

```

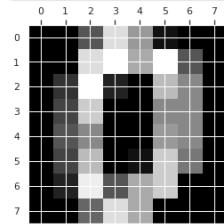
This tells us that we are going to be predicting what digit (0,1,2,3,4,5,6,7,8, or 9) is in the image.

To get an idea of what the images look like, we can use `matshow` which is short for `matrix show`. It takes a 2D matrix and plots it as a grayscale image. To get the actual color bar, we use the `matplotlib.pyplot`.

```
plt.gray()  
plt.matshow(digits.images[0])
```

```
<matplotlib.image.AxesImage at 0x7f2b6e9aeeb0>
```

```
<Figure size 432x288 with 0 Axes>
```



bunch objects are designed for machine learning, so they have the features as "data" and target explicitly identified.

```
digits_X = digits.data  
digits_y = digits.target
```

We can further check the type and shape of these

```
type(digits_y)
```

```
numpy.ndarray
```

Note

because this is the target, it's okay that this is one dimensional.

```
digits_y.shape
```

```
(1797, )
```

```
digits_X.shape
```

```
(1797, 64)
```

This has one row for each sample and has reshaped the 8x8 image into a 64 length vector. So we have one 'feature' for each pixel in the images.

We are going to do some model comparison, so we will instantiate estimator objects for two different classifiers.

```
svm_clf = svm.SVC(gamma=0.001)  
gnb_clf = naive_bayes.GaussianNB()
```

We're going to use a [ShuffleSplit](#) object to do Cross validation with 100 iterations to get smoother mean test and train score curves, each time with 20% data randomly selected as a validation set.

Further Reading

You can see visualization of different [cross validation](#) types in the sklearn documentation.

```
cv = model_selection.ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)
```

Note

This object has a `random_state` object, the `GridSearchCV` that we were using didn't have a way to control the random state directly, but it accepts not only integers, but also cross validation objects to the `cv` parameter. The `KFold` cross validation object also has that parameter, so we could repeat what we did in previous classes by creating a `KFold` object with a fixed random state.

We'll also create a linearly spaced list of training percentages and we'll also divide it into more jobs to be more computationally efficient.

```
train_sizes=np.linspace(0.1, 1.0, 5)  
n_jobs=4
```

Try it yourself

Try varying the `n_jobs` parameter and timing the execution using the `timeit magic`.

Now we can create the learning curve.

```
train_sizes_svm, train_scores_svm, test_scores_svm, fit_times_svm, score_times_svm  
= model_selection.learning_curve(  
    svm_clf,  
    digits_X,  
    digits_y,  
    cv=cv,  
    n_jobs=n_jobs,  
    train_sizes=train_sizes,  
    return_times=True, )
```

It returns the list of the counts for each training size (we input percentages and it returns counts)

```
train_sizes_svm
```

Try it yourself

Try using `matshow` without `plt.gray()`. How is it different? What might alternatives do? What other code in this notebook influences how plots look?

Tip

Removing a line from an excerpt of code can help you see what that line did and learn more about how each piece work.

```
array([ 143,  467,  790, 1113, 1437])
```

The other parameters, it returns a list for each length that's 100 long because our cross validation was 100 iterations.

```
fit_times_svm.shape
```

```
(5, 100)
```

We can save it in a DataFrame after averaging over the 100 trials.

```
svm_learning_df = pd.DataFrame(data = train_sizes_svm, columns = ['train_size'])
# svm_learning_df['train_size'] = train_sizes_svm
svm_learning_df['train_score'] = np.mean(train_scores_svm, axis=1)
svm_learning_df['test_score'] = np.mean(test_scores_svm, axis=1)
svm_learning_df['fit_time'] = np.mean(fit_times_svm, axis=1)
svm_learning_df['score_times'] = np.mean(score_times_svm, axis=1)
```

then we can look at the DataFrame

```
svm_learning_df.head()
```

	train_size	train_score	test_score	fit_time	score_times
0	143	0.999510	0.923611	0.005182	0.007035
1	467	0.999101	0.978278	0.025668	0.019760
2	790	0.998848	0.986278	0.048724	0.028412
3	1113	0.998913	0.989500	0.076667	0.035373
4	1437	0.998859	0.991306	0.105876	0.041529

We can use our skills in transforming data to make it easier to examine just a subset of the scores.

```
svm_learning_df_scores = svm_learning_df.melt(id_vars=['train_size'], value_vars=['train_score', 'test_score'])

svm_learning_df_scores.head(2)
```

	train_size	variable	value
0	143	train_score	0.999510
1	467	train_score	0.999101

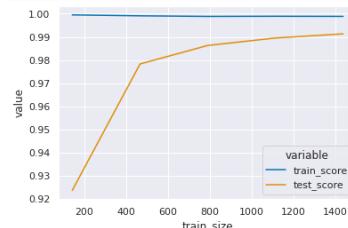
💡 Hint

This is one thing we can analyze, but there are others. To earn prepare on assignment 11, manipulate your results a different way.

This new DataFrame allows us to make convenient plots.

```
sns.lineplot(data = svm_learning_df_scores, x ='train_size',
y='value',hue='variable')
```

```
<AxesSubplot:xlabel='train_size', ylabel='value'>
```



💡 Tip

Getting used to thinking through these sorts of manipulations can take time, but is valuable. Investing time to learn these things will help you both write shorter, more readable, easy to examine, code (which is nicer to your co-workers) and help you develop flexible mental representation. The flexibility of your mental model of material is the way learning scientists distinguish competent practitioners from experts.

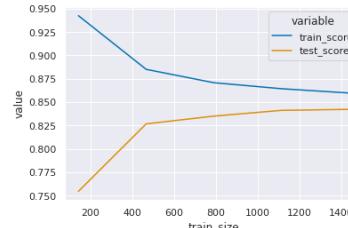
```
train_sizes_gnb, train_scores_gnb, test_scores_gnb, fit_times_gnb, score_times_gnb
= model_selection.learning_curve(
    gnb_clf,
    digits_X,
    digits_y,
    cv=cv,
    n_jobs=n_jobs,
    train_sizes=train_sizes,
    return_times=True,)
```

We can do the same for Gaussian Naive Bayes

```
gnb_learning_df = pd.DataFrame(data = train_sizes_gnb, columns = ['train_size'])
# gnb_learning_df['train_size'] = train_sizes_gnb
gnb_learning_df['train_score'] = np.mean(train_scores_gnb, axis=1)
gnb_learning_df['test_score'] = np.mean(test_scores_gnb, axis=1)
gnb_learning_df['fit_time'] = np.mean(fit_times_gnb, axis=1)
gnb_learning_df['score_times_gnb'] = np.mean(score_times_gnb, axis=1)
```

```
gnb_learning_scores = gnb_learning_df.melt(id_vars=['train_size'], value_vars=['train_score', 'test_score'])
sns.lineplot(data = gnb_learning_scores, x ='train_size',
y='value',hue='variable')
```

```
<AxesSubplot:xlabel='train_size', ylabel='value'>
```



These scores are overall not as high as the SVM (note the values on the y axis.)

29.1. Questions After Class

29.1.1. When would I use an SVM?

SVMs are a very powerful model and they're good when you need relatively quick training (and test) time, a model that takes a small amount of memory (eg running the prediction on a mobile device or smart device), with high accuracy.

Question in Class

This was a question after class, but the answer makes more sense inline here.

30. Intro to NLP- representing text data

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances
import pandas as pd

# %load http://drsmby.co/310read
# add an entry to the following dictionary with a name as the key and a sentence
# as the value
# share a sentence about how you're doing this week
# remember this will be python code, don't use
# You can remain anonymous (this page & the notes will be fully public)
# by attributing it to a celebrity or pseudonym, but include *some* sort of
# attribution
sentence_dict = {
    'Professor Brown': "I'm excited for Thanksgiving.",
    'Matt Langton': "I'm doing pretty good, I'll be taking the days off to catch up on
    various classwork.",
    'Evan': "I'm just here so my grade doesn't get fined",
    'Greg Bassett': "I'm doing well, my birthday is today. I'm looking forward to
    seeing my family this Thursday, I haven't seen a lot of them in a long time.",
    'Noah N': "I'm doing well! I can't wait to take opportunity of this long weekend to
    catch up on various HW's, projects, etc.",
    'Tuyetlinh': "I'm struggling to get all my work done before break, but I'm excited
    to have that time off when I'm all done.",
    'Kenza Bouabdallah': "I am doing good. How are you ?",
    'Chris Kerfoot': "I'm doing pretty good. I'm happy to have some days off this week
    because of Thanksgiving!",
    'Kang Liu': "New week, new start",
    'Aiden Hill': "I am very much enjoying this class.",
    'Muhammad S': "I am doing pretty well. I am looking forward to taking a few days
    off.",
    'Max Mastrorocco': "Cannot wait for a break.",
    'Daniel': "I am doing well. I am ready and excited for break!",
    'Nate': "I'm just vibing right now, ready for break",
    'Jacob': "I am going to eat Turkey.",
    'Anon': "nom nom nom"
}
```

How can we analyze these? All of the machine learning models we have seen only use numerical features organized into a table with one row per sample and one column per feature.

That's actually generally true. All ML models require numerical features, at some point. The process of taking data that is not numerical and tabular, which is called unstructured, into structured (tabular) format we require is called feature extraction. There are many, many ways to do that. We'll see a few over the course of the rest of the semester. Some more advanced models hide the feature extraction, by putting it in the same function, but it's always there.

30.1. Terms

- document: unit of text we're analyzing (one sample)
- token: sequence of characters in some particular document that are grouped together as a useful semantic unit for processing (basically a word)
- stop words: no meaning, we don't need them (like a, the, an.). Note that this is context dependent
- dictionary: all of the possible words that a given system knows how to process

We'll start by taking out one sentence and analyzing that:

```
s1 = sentence_dict['Professor Brown']
```

```
s1
```

```
"I'm excited for Thanksgiving."
```

30.2. Bag of Words Representation

We're going to learn a representation called the bag of words. It ignores the order of the words within a document. To do this, we'll first extract all of the tokens (tokenize) the documents and then count how many times each word appears. This will be our numerical representation of the data.

Then we initialize our transformer

```
counts = CountVectorizer()
```

We can use the fit transform method to fit the vectorizer model and apply it to this sentence.

```
counts.fit_transform([s1])
```

```
<1x3 sparse matrix of type '<class 'numpy.int64'>'  
with 3 stored elements in Compressed Sparse Row format>
```

Further Reading

[Transformers](#) are another broad class of sklearn objects. We've seen Estimators mostly so far. We're focusing on the [text feature extraction](#) for now.

To see the output better, we use the toarray method.

```
counts.fit_transform([s1]).toarray()
```

```
array([[1, 1, 1]])
```

We can also examine attributes of the object.

```
counts.vocabulary_
```

```
{'excited': 0, 'for': 1, 'thanksgiving': 2}
```

We see that what it does is creates an ordered (the values are the order) list of words as the parameters of this model (ending in `_` is an attribute of the object or parameter of the model)

⚡ Try it yourself

What other model parameters have we seen? How have we used model parameters in the past?

To see what happens a bit more, let's add a second sentence.

```
s2 = sentence_dict['Kang Liu']  
s2  
  
'New week, new start'  
  
counts.fit_transform([s1,s2])  
counts.vocabulary_  
  
{'excited': 0, 'for': 1, 'thanksgiving': 4, 'new': 2, 'week': 5, 'start': 3}
```

Now we can see that it puts the words in the `vocabulary_` attribute (aka the [dictionary](#)) in alphabetical order.

```
counts.fit_transform([s1,s2]).toarray()  
  
array([[1, 0, 0, 1, 0],  
       [0, 0, 2, 1, 0]])
```

From this we can see that the representation is the count of how many times each word appears.

Now we can apply it to all of the sentences, or our whole [corpus](#)

```
mat = counts.fit_transform(sentence_dict.values()).toarray()  
mat  
  
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, 1, 0, ..., 0, 0, 0],  
      [0, 0, 0, ..., 0, 0, 0]])
```

We can get the dictionary out in order using the `get_feature_names` method. This method has a generic name, not specific to text, because it's a property of transformers in general.

```
counts.get_feature_names()
```

💡 Tip

Notice that we keep using the same tools over and over to explore how things work. You can do this on your own, when you're learning new things. Example code is readily available online but not all of it is well documented or clearly explained.

Also, in a job, much, much, more of your time will be spent reading code than writing code from scratch. These strategies will help you get familiar with a new code base and get up to speed faster.

```
/opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will
be removed in 1.2. Please use get_feature_names_out instead.
    warnings.warn(msg, category=FutureWarning)
```

```
['all',
 'am',
 'and',
 'are',
 'be',
 'because',
 'before',
 'birthday',
 'break',
 'but',
 'can',
 'cannot',
 'catch',
 'class',
 'classwork',
 'days',
 'doesn',
 'doing',
 'done',
 'eat',
 'enjoying',
 'etc',
 'excited',
 'family',
 'few',
 'fined',
 'for',
 'forward',
 'get',
 'going',
 'good',
 'grade',
 'happy',
 'have',
 'haven',
 'here',
 'how',
 'hw',
 'in',
 'is',
 'just',
 'll',
 'long',
 'looking',
 'lot',
 'much',
 'my',
 'new',
 'nom',
 'now',
 'of',
 'off',
 'on',
 'opportuity',
 'pretty',
 'projects',
 'ready',
 'right',
 'seeing',
 'seen',
 'so',
 'some',
 'start',
 'struggling',
 'take',
 'taking',
 'thanksgiving',
 'that',
 'the',
 'them',
 'this',
 'thursday',
 'time',
 'to',
 'today',
 'turkey',
 'up',
 'various',
 'very',
 'vibing',
 'wait',
 'week',
 'weekend',
 'well',
 'when',
 'work',
 'you']
```

We can use a data frame again to see this more easily. We can put labels on both the index and the column headings.

```
sentence_df = pd.DataFrame(data = mat, columns =counts.get_feature_names(),
                           index=sentence_dict.keys())
sentence_df
```

	all	am	and	are	be	because	before	birthday	break	but	...	various	very	vibing	wait	week	weekend	well	when	work	you
Professor Brown	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Matt Langton	0	0	0	0	1	0	0	0	0	0	...	1	0	0	0	0	0	0	0	0	0
Evan	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Greg Bassett	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	1	0	0
Noah N	0	0	0	0	0	0	0	0	0	0	...	1	0	0	1	0	1	1	0	0	0
Tuyetlinh	2	0	0	0	0	0	0	1	0	1	1	...	0	0	0	0	0	0	0	1	1
Kenza Bouabdallah	0	1	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
Chris Kerfoot	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	1	0	0	0	0	0
Kang Liu	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	0	0
Aiden Hill	0	1	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	0
Muhammad S	0	2	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
Max Mastrorocco	0	0	0	0	0	0	0	0	1	0	...	0	0	0	1	0	0	0	0	0	0
Daniel	0	2	1	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	0	0
Nate	0	0	0	0	0	0	0	0	1	0	...	0	0	1	0	0	0	0	0	0	0
Jacob	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Anon	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

16 rows × 87 columns

30.3. How can we find the most commonly used word?

One guess

```
sentence_df.max()
```

```
all      2
am      2
and     1
are     1
be      1
.
.
.
weekend 1
well    1
when    1
work    1
you    1
Length: 87, dtype: int64
```

This is the maximum number of times each word appears in single "document", but it's also not sorted, it's alphabetical.

This shows the word that appears the most times.:

```
sentence_df.max().sort_values()
```

```
looking   1
start    1
some     1
so       1
seen     1
.
.
.
my       2
done     2
am       2
all     2
nom     3
Length: 87, dtype: int64
```

To get what we want we need to sum, which by default is along the columns, or per word.

```
sentence_df.sum()
```

```
all      2
am      7
and     1
are     1
be      1
.
.
.
weekend 1
well    4
when    1
work    1
you    1
Length: 87, dtype: int64
```

Again it's unsorted, but we can apply max

```
sentence_df.sum().max
```

```
<bound method NDFrame._add_numeric_operations.<locals>.max of all      2
am      7
and     1
are     1
be      1
.
.
.
weekend 1
well    4
when    1
work    1
you    1
Length: 87, dtype: int64>
```

this gives only the value though, we want the word. When we summed we got back a Series with the words in the index:

```
| sentence_df.sum().index
```

```
Index(['all', 'am', 'and', 'are', 'be', 'because', 'before', 'birthday',
       'break', 'but', 'can', 'cannot', 'catch', 'class', 'classwork', 'days',
       'doesn', 'doing', 'done', 'eat', 'enjoying', 'etc', 'excited', 'family',
       'few', 'fined', 'for', 'forward', 'get', 'going', 'good', 'grade',
       'happy', 'have', 'haven', 'here', 'how', 'hw', 'in', 'is', 'just', 'll',
       'long', 'looking', 'lot', 'much', 'my', 'new', 'nom', 'now', 'of',
       'off', 'on', 'opportunity', 'pretty', 'projects', 'ready', 'right',
       'seeing', 'seen', 'so', 'some', 'start', 'struggling', 'take', 'taking',
       'thanksgiving', 'that', 'the', 'them', 'this', 'thursday', 'time', 'to',
       'today', 'turkey', 'up', 'various', 'very', 'vibing', 'wait', 'week',
       'weekend', 'well', 'when', 'work', 'you'],
      dtype='object')
```

So we can use idxmax

```
| sentence_df.sum().idxmax()
```

```
'to'
```

30.4. Distances in text

We can now use a distance function to calculate how far apart the different sentences are.

```
| euclidean_distances(sentence_df)
```

```
array([[0.          , 4.24264069, 3.31662479, 5.19615242, 4.89897949,
       5.09901951, 3.          , 3.87298335, 3.          , 3.          ,
       4.12310563, 2.23606798, 3.16227766, 2.82842712, 2.82842712,
       3.46410162],
[4.24264069, 0.          , 4.79583152, 5.91607978, 4.69041576,
       5.83095189, 4.12310563, 4.12310563, 4.58257569, 4.58257569,
       4.12310563, 4.35889894, 4.89897949, 4.69041576, 4.24264069,
       4.89897949],
[3.31662479, 4.79583152, 0.          , 5.29150262, 5.38516481,
       5.38516481, 3.74165739, 4.69041576, 3.74165739, 3.74165739,
       4.69041576, 3.46410162, 4.35889894, 3.60555128, 3.60555128,
       4.12310563],
[5.19615242, 5.91607978, 5.29150262, 0.          , 5.56776436,
       6.244998 , 5.29150262, 5.47722558, 5.47722558, 5.29150262,
       5.29150262, 5.29150262, 5.56776436, 5.19615242,
       5.74456265],
[4.89897949, 4.69041576, 5.38516481, 5.56776436, 0.          ,
       6.164414 , 5.          , 5.          , 5.19615242, 5.          ,
       5.19615242, 4.79583152, 5.29150262, 5.29150262,
       4.69041576, 5.47722558],
[5.09901951, 5.83095189, 5.38516481, 6.244998 , 6.164414 ,
       0.          , 5.56776436, 5.56776436, 5.56776436,
       5.74456265, 5.19615242, 5.65685425, 5.47722558, 5.09901951,
       5.83095189],
[3.          , 4.12310563, 3.74165739, 5.29150262, 5.          ,
       5.56776436, 0.          , 4.          , 3.46410162, 3.16227766,
       3.74165739, 3.16227766, 3.31662479, 3.60555128, 3.          ,
       3.87298335],
[3.87298335, 4.12310563, 4.69041576, 5.47722558, 5.19615242,
       5.56776436, 4.          , 0.          , 4.24264069, 4.24264069,
       4.24264069, 4.24264069, 4.79583152, 4.58257569, 4.12310563,
       4.79583152],
[3.          , 4.58257569, 3.74165739, 5.29150262, 5.          ,
       5.56776436, 3.16227766, 4.24264069, 3.46410162, 0.          ,
       4.          , 3.16227766, 3.60555128, 3.60555128,
       3.87298335],
[3.87298335, 4.12310563, 4.69041576, 5.29150262, 5.19615242,
       5.74456265, 3.74165739, 4.24264069, 4.47213595, 4.          ,
       0.          , 4.24264069, 3.60555128, 4.58257569, 3.60555128,
       4.79583152],
[2.23606798, 4.35889894, 3.46410162, 5.29150262, 4.79583152,
       5.19615242, 3.16227766, 4.24264069, 3.16227766, 3.16227766,
       4.24264069, 0.          , 3.16662479, 2.64575131, 3.          ,
       3.60555128],
[3.16227766, 4.89897949, 4.35889894, 5.56776436, 5.29150262,
       5.65685425, 3.31662479, 4.79583152, 4.12310563, 3.60555128,
       3.60555128, 3.31662479, 0.          , 3.46410162, 3.46410162,
       4.47213595],
[2.82842712, 4.24264069, 3.60555128, 5.56776436, 5.29150262,
       5.47722558, 3.60555128, 4.58257569, 3.60555128, 3.60555128,
       4.58257569, 2.64575131, 3.46410162, 0.          , 3.46410162,
       4.          ],
[2.82842712, 4.24264069, 3.60555128, 5.19615242, 4.69041576,
       5.09901951, 3.          , 4.12310563, 3.31662479, 3.          ,
       3.60555128, 3.          , 3.46410162, 3.46410162, 0.          ,
       3.74165739],
[3.46410162, 4.89897949, 4.12310563, 5.74456265, 5.47722558,
       5.83095189, 3.87298335, 4.79583152, 3.87298335, 3.87298335,
       4.79583152, 3.60555128, 4.47213595, 4.          , 3.74165739,
       0.          ]])
```

We can make this easier to read by making it a Data Frame.

```
| dist_df = pd.DataFrame(data = euclidean_distances(sentence_df),
                           index= sentence_dict.keys(), columns= sentence_dict.keys())
dist_df
```

	Professor Brown	Matt Langton	Evan	Greg Bassett	Noah N	Tuyetlinh	Kenza Bouabdallah	Chris Kerfoot	Kang Liu	Aiden Hill	Muhammad S	Max Mastrorocco	Daniel	Nate
Professor Brown	0.000000	4.242641	3.316625	5.196152	4.898979	5.099020	3.000000	3.872983	3.000000	3.000000	4.123106	2.236068	3.162278	2.828427
Matt Langton	4.242641	0.000000	4.795832	5.916080	4.690416	5.830952	4.123106	4.123106	4.582576	4.582576	4.123106	4.358899	4.898979	4.690416
Evan	3.316625	4.795832	0.000000	5.291503	5.385165	5.385165	3.741657	4.690416	3.741657	3.741657	4.690416	3.464102	4.358899	3.605551
Greg Bassett	5.196152	5.916080	5.291503	0.000000	5.567764	6.244998	5.291503	5.477226	5.477226	5.291503	5.291503	5.291503	5.567764	5.567764
Noah N	4.898979	4.690416	5.385165	5.567764	0.000000	6.164414	5.000000	5.000000	5.196152	5.000000	5.196152	4.795832	5.291503	5.291503
Tuyetlinh	5.099020	5.830952	5.385165	6.244998	6.164414	0.000000	5.567764	5.567764	5.567764	5.567764	5.744563	5.196152	5.656854	5.477226
Kenza Bouabdallah	3.000000	4.123106	3.741657	5.291503	5.000000	5.567764	0.000000	4.000000	3.464102	3.162278	3.741657	3.162278	3.316625	3.605551
Chris Kerfoot	3.872983	4.123106	4.690416	5.477226	5.000000	5.567764	4.000000	0.000000	4.242641	4.242641	4.242641	4.242641	4.795832	4.582576
Kang Liu	3.000000	4.582576	3.741657	5.477226	5.196152	5.567764	3.464102	4.242641	0.000000	3.464102	4.472136	3.162278	4.123106	3.605551
Aiden Hill	3.000000	4.582576	3.741657	5.291503	5.000000	5.567764	3.162278	4.242641	3.464102	0.000000	4.000000	3.162278	3.605551	3.605551
Muhammad S	4.123106	4.123106	4.690416	5.291503	5.196152	5.744563	3.741657	4.242641	4.472136	4.000000	0.000000	4.242641	3.605551	4.582576
Max Mastrorocco	2.236068	4.358899	3.464102	5.291503	4.795832	5.196152	3.162278	4.242641	3.162278	3.162278	4.242641	0.000000	3.316625	2.645751
Daniel	3.162278	4.898979	4.358899	5.567764	5.291503	5.656854	3.316625	4.795832	4.123106	3.605551	3.605551	3.316625	0.000000	3.464102
Nate	2.828427	4.690416	3.605551	5.567764	5.291503	5.477226	3.605551	4.582576	3.605551	3.605551	4.582576	2.645751	3.464102	0.000000
Jacob	2.828427	4.242641	3.605551	5.196152	4.690416	5.099020	3.000000	4.123106	3.316625	3.000000	3.605551	3.000000	3.464102	3.464102
Anon	3.464102	4.898979	4.123106	5.744563	5.477226	5.830952	3.872983	4.795832	3.872983	3.872983	4.795832	3.605551	4.472136	4.000000

Who wrote the most similar question to me?

```
dist_df['Professor Brown'].drop('Professor Brown').idxmin()
```

'Max Mastrorocco'

30.5. Questions After Classroom

30.5.1. How can this be used for training a classifier?

To train a classifier, we would also need target variables, but the `mat` variable we had above can be used as the `x` for any `skLearn` estimator object. To train more complex tasks you would need appropriate data: for example labeled articles that are real and fake to train a fake news classifier (this is provided for a12).

30.5.2. How are ram tokens tracked?

Ram Tokens are tracked in the [Ram Token Bank](http://drsmb.co/ramtoken): <http://drsmb.co/ramtoken> form. You'll get e-mails when you earn or use them, however no one has submitted for any. You still can though (especially if you were advised to and forgot).

30.6. More Practice

1. Which two people wrote the most similar sentences?
 2. Do you think this representation captures all cases of similiy? Can you generate a case where it doesn't do well?
 3. Try

31. More NLP & Solving problems with ML

Imagine you work for a news agency, current articles are stored in a database in a table where they are indexed by a unique identifier. A separate table indicates the primary category for most articles, but some are missing. Your assignment is to create an automatic category generator to save editors time. Your manager suggests that an SVM or decision tree is probably best, but is unsure what text representations will perform best. You are expected to produce an accessible report with tables and plots that visualize the performance and impact of various modeling choices.

Write the import statements you would need to solve this problem. Include whole libraries or modules as is most appropriate.

```
import pandas as pd
import sqlite3
from sklearn.feature_extraction import text
from sklearn import tree
from sklearn import svm
from sklearn import model_selection
import seaborn as sns
```

Given the following vocabulary,

['and', 'are', 'cat', 'cats', 'dogs', 'pets', 'popular', 'videos']

represent “Cats and dogs are pets” in vector format.

[1,1,0,1,1,1,0,0]

```

[ type(ng_y)
  numpy.ndarray

[ ng_y[:5]
  array([0, 0, 1, 0, 0])

[ ng_X[0]

[ "From: robert@cpuserver.acsc.com (Robert Grant)\nSubject: Virtual Reality for X on
the CHEAP!\nOrganization: USCA/CSC, Los Angeles\nLines: 187\nDistribution:
world\nReply-To: robert@cpuserver.acsc.com (Robert Grant)\nNNTP-Posting-Host:
cpuserver.acsc.com\nHi everyone,\nI thought that some people may be interested
in my VR\nsoftware on these groups:\n*****Announcing the release of
Multiverse-1.0.2*****\nMultiverse is a multi-user, non-immersive, X-Windows
based Virtual Reality\nsystem, primarily focused on
entertainment/research.\nFeatures:\nClient-Server based model, using
Berkeley Sockets.\nNo limit to the number of users (apart from performance).\n
Generic clients.\nCustomizable servers.\nHierarchical Objects (allowing
attachment of cameras and light sources).\nMultiple light sources (ambient,
point and spot).\nObjects can have extension code, to handle unique
functionality, easily\nattached.\nFunctionality:\nClient:\nThe
client is built around a 'fast' render loop. Basically it changes things\nwhen
told to by the server and then renders an image from the user's\nviewpoint. It
also provides the server with information about the user's\nactions - which can
then be communicated to other clients and therefore to\nother users.\n\nThe
client is designed to be generic - in other words you don't need to\ndevelop a
new client when you want to enter a new world. This means that\nresources can
be spent on enhancing the client software rather than adapting\nit. The
adaptations, as will be explained in a moment, occur in the servers.\n\nThis
release of the client software supports the following functionality:\n
o Hierarchical Objects (with associated addressing)\n
o Multiple Light Sources
and Types (Ambient, Point and Spot)\n
o User Interface Panels\n
o Colour
Polygonal Rendering with Phong Shading (optional wireframe for\nfaster frame
rates)\n
o Mouse and Keyboard Input\n
(Some people may be disappointed
that this software doesn't support the\nPowerGlove as an input device - this is
not because it can't, but because\nI don't have one! This will, however, be one
of the first enhancements!)\n
Server(s):\nThis is where customization can
take place. The following basic support is\nprovided in this release for
potential world server developers:\n
o Transparent Client Management\n
o Client Message Handling\n
This may not sound like much, but it takes away the
headache of accepting and\nterminating clients and receiving messages from
them - the application writer\ncan work with the assumption that things are
happening locally.\n
Things get more interesting in the object extension
functionality. This is\nwhat is provided to allow you to animate your
objects:\n
o Server Selectable Extension Installation:\n
What this
means is that you can decide which objects have extended\nfunctionality in
your world. Basically you call the extension\ninitialisers for an object
you basically write callback\nfunctions for the events that you want the
object to respond to.\n
(Current events supported: INIT, MOVE, CHANGE,
COLLIDE & TERMINATE)\n
o Collision Detection Registration:\n
If you
want your object to respond to collision events just provide\nsome basic
information to the collision detection management software.\n
Your callback
will be activated when a collision occurs.\n
This software is kept separate
from the worldserver applications because\nthe application developer wants to
build a library of extended objects\nfrom which to choose.\n
The
following is all you need to make a World Server application:\n
o Provide an
initWorld function:\n
This is where you choose what object extensions will
be supported, plus\nany initialization you want to do.\n
o Provide a
positionObject function:\n
This is where you determine where to place a new
client.\n
o Provide an installWorldObjects function:\n
This is where
you load the world (.wld) file for a new client.\n
o Provide a getWorldType
function:\n
This is where you tell a new client what persona they should
have.\n
o Provide an animateWorld function:\n
This is where you can go
wild! At a minimum you should let the objects\nmove (by calling a move
function) and let the server sleep for a bit\n(to avoid outrunning the
clients).\n
That's all there is to it! And to prove it here are the line
counts for the\nthree world servers I've provided:\n
generic - 81
lines\n
dactyl - 270 lines (more complicated collision detection due to
the\nstairs! Will probably be improved with future\nversions)\n
dogfight - 72 lines\n
Location:\n
This software is located
at the following site:\n
ftp.u.washington.edu\n
Directory:\n
pub/virtual-
worlds\n
File:\n
multiverse-1.0.2.tar.Z\n
Futures:\n
Client:\n
o Texture mapping.\n
o More realistic rendering: i.e. Z-Buffering (or similar),
Gouraud shading\n
o HMD support.\n
o Etc, etc...\n
Server:\n
o Physical Modelling (gravity, friction etc).\n
o Enhanced Object
Management/Interaction\n
o Etc, etc...\n
Both:\n
o Improved
Comms!!\n
I hope this provides people with a good understanding of the
Multiverse\nsoftware, unfortunately it comes with practically zero documentation,
and I'm not sure\nwhether that will ever be able to be rectified! :-(\n
I hope
people enjoy this software and that it is useful in our explorations of\nthe
Virtual Universe - I've certainly found fascinating developing it, and I\nwould
*LOVE* to add support for the PowerGlove...and an HMD :-)! Finally one major
disclaimer:\n
This is totally amateur code. By that I mean there is no support
for this code\nother than what I, out the kindness of my heart, or you, out of
pure\ndesperation, provide. I cannot be held responsible for anything good or
bad\nthat may happen through the use of this code - USE IT AT YOUR OWN
RISK!\n
Disclaimer over!\n
Of course if you love it, I would like to here from
you. And anyone with\nPOSITIVE contributions/criticisms is also encouraged to
contact me. Anyone who\nhates it: >
/dev/null!\n
*****\n
And if anyone wants to let me do this for a living: you know
where to\nwrite:\n
:-)\n
*****\n
Thanks,\n
Robert.\n
robert@acsc.com\n
*****\n"

```

We're going to instantiate the object and fit it to the whole dataset.

```

count_vec = text.CountVectorizer()
count_vec.fit(ng_X)

# CountVectorizer
CountVectorizer()

```

Important

I changed the following a little bit from class so that we can use the same test/train split for the two different types of transformation so that we can compare them more easily.

This also helps illustrate when using the fit and transform separately is helpful.

```
{ ng_X_train, ng_X_test, ng_y_train, ng_y_test = train_test_split(  
    ng_X, ng_y, random_state=0)
```

Now, we can use the transformation that we fit to the whole dataset to transform the train and test portions of the data separately.

The transform method also returns the sparse matrix directly so we no longer need the `toarray` method.

```
{ ng_vec_train = count_vec.transform(ng_X_train)  
ng_vec_test = count_vec.transform(ng_X_test)
```

```
{ ng_vec_train[0]
```

```
<1x24257 sparse matrix of type '<class 'numpy.int64'>'  
with 84 stored elements in Compressed Sparse Row format>
```

```
{ clf = MultinomialNB()
```

```
{ clf.fit(ng_vec_train,ng_y_train).score(ng_vec_test,ng_y_test)
```

```
0.9830508474576272
```

```
{ tfidf = text.TfidfTransformer()  
tfidf.fit_transform(ng_X)
```

```

ValueError                                Traceback (most recent call last)
Input In [11], in <cell line: 2>()
      1 tfidf = text.TfidfTransformer()
----> 2 tfidf.fit_transform(ng_X)

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/base.py:867, in TransformerMixin.fit_transform(self, X, y,
**fit_params)
    863 # non-optimized default implementation; override when a better
    864 # method is possible for a given clustering algorithm
    865 if y is None:
    866     # fit method of arity 1 (unsupervised transformation)
--> 867     return self.fit(X, **fit_params).transform(X)
    868 else:
    869     # fit method of arity 2 (supervised transformation)
    870     return self.fit(X, y, **fit_params).transform(X)

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/feature_extraction/text.py:1622, in TfidfTransformer.fit(self, X, y)
    1604 """Learn the idf vector (global term weights).
    1605
    1606 Parameters
    (...)

    1617     Fitted transformer.
    1618 """
    1619 # large sparse data is not supported for 32bit platforms because
    1620 # _document_frequency uses np.bincount which works on arrays of
    1621 # dtype NPY_INTP which is int32 for 32bit platforms. See #20923
-> 1622 X = self._validate_data(
    1623     X, accept_sparse="csr", "csc"), accept_large_sparse=not _IS_32BIT
    1624 )
    1625 if not sp.issparse(X):
    1626     X = sp.csr_matrix(X)

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/base.py:577, in BaseEstimator._validate_data(self, X, y, reset,
validate_separately, **check_params)
    575     raise ValueError("Validation should be done on X, y or both.")
    576 elif no_val_X and no_val_y:
--> 577     X = check_array(X, input_name="X", **check_params)
    578     out = X
    579 elif no_val_X and not no_val_y:

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/utils/validation.py:879, in check_array(array, accept_sparse,
accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd,
ensure_min_samples, ensure_min_features, estimator, input_name)
    877     # If input is 1D raise error
    878     if array.ndim == 1:
--> 879         raise ValueError(
    880             "Expected 2D array, got 1D array instead:\narray={}\n"
    881             "Reshape your data either using array.reshape(-1, 1) if "
    882             "your data has a single feature or array.reshape(1, -1) "
    883             "if it contains a single sample.".format(array)
    884         )
    885 if dtype_numeric and array.dtype.kind in "USV":
    886     raise ValueError(
    887         "dtype='numeric' is not compatible with arrays of bytes/strings."
    888         "Convert your data to numeric values explicitly instead."
    889     )

ValueError: Expected 2D array, got 1D array instead:
array=[{"From: robert@cpuserver.acsc.com (Robert Grant)\nSubject: Virtual Reality
for X on the CMEAPI\nOrganization: USCACSC, Los Angeles\nLines: 187\nDistribution:
world\nReply-To: robert@cpuserver.acsc.com (Robert Grant)\nNNTP-Posting-Host:
cpuserver.acsc.com\nHi everyone,\n\nI thought that some people may be interested
in my VR software on these groups:\n\n*****Announcing the release of
MultiVerse-1.0.2*****\n\nMultiVerse is a multi-user, non-immersive, X-Windows
based Virtual Reality system, primarily focused on
entertainment/research.\n\nFeatures:\n\nClient-Server based model, using
Berkeley Sockets.\nNo limit to the number of users (apart from performance).\n
Generic clients.\nCustomizable servers.\nHierarchical Objects (allowing
attachment of cameras and light sources).\nMultiple light sources (ambient,
point and spot).\nObjects can have extension code, to handle unique
functionality, easily.\nAttached.\nFunctionality:\n\nClient:\nThe
client is built around a 'fast' render loop. Basically it changes things\nwhen
told to by the server and then renders an image from the user's\nviewpoint. It
also provides the server with information about the user's\nactions - which can
then be communicated to other clients and therefore to\nother users.\n\nThe
client is designed to be generic - in other words you don't need to\ndevelop a
new client when you want to enter a new world. This means that\nresources can
be spent on enhancing the client software rather than adapting\nit. The
adaptations, as will be explained in a moment, occur in the servers.\n\nThis
release of the client software supports the following functionality:\n\n
o Hierarchical Objects (with associated addressing)\n
o Multiple Light Sources
and Types (Ambient, Point and Spot)\n
o User Interface Panels\n
o Colour
Polyangular Rendering with Phong Shading (optional wireframe for\nfaster frame
rates)\n
o Mouse and Keyboard Input\n\n(Some people may be disappointed
that this software doesn't support the\nPowerGlove as an input device - this is
not because it can't, but because\nI don't have one! This will, however, be one
of the first enhancements!)nn Server(s):\n\nThis is where customization can
take place. The following basic support is\nprovided in this release for
potential world server developers:\n
o Transparent Client Management\n
o

```

We can figure out why, we can compare what that method takes as input to what the count vectorizer takes as input. While working, the easiest way to do this is using the jupyter help (shift+tab, or ?) but for the notes, I'll print them out differently.

```
print('\n'.join(tfidf.fit_transform(doc).split('\n'))[:7]))
```

```
Learn the idf vector (global term weights).

Parameters
-----
X : sparse matrix of shape n_samples, n_features
    A matrix of term/token counts.
```

```
print('\n'.join(count_vec.fit.__doc__.split('\n')[7]))
```

```
Learn a vocabulary dictionary of all tokens in the raw documents.

Parameters
-----
raw_documents : iterable
    An iterable which generates either str, unicode or file objects.
```

We wanted the TfidfVectorizer, not the transformer, so that it accepts documents not features. We will again, instantiate the object and then fit on the whole dataset.

```
tfidf = text.TfidfVectorizer()
tfidf.fit(ng_X)

▼ TfidfVectorizer
TfidfVectorizer()
```

We can see this works, because the code runs, but for completeness , we can also check the input again to compare with the above.

```
print('\n'.join(tfidf.fit.__doc__.split('\n')[6]))
```

```
Learn vocabulary and idf from training set.

Parameters
-----
raw_documents : iterable
    An iterable which generates either str, unicode or file objects.
```

This now takes documents as we wanted. Since we split the data before transforming, we can then apply the new fit using the transform on the train/test splits.

```
ng_tfidf_train = tfidf.transform(ng_X_train)
ng_tfidf_test = tfidf.transform(ng_X_test)
```

Since these splits were made before we can use the same targets we used above.

```
clf.fit(ng_tfidf_train, ng_y_train).score(ng_tfidf_test, ng_y_test)
```

```
0.9288135593220339
```

31.1. Comparing representations

To start, we will look at one element from each in order to compare them.

```
ng_tfidf_train[0]
```

```
<1x24257 sparse matrix of type '<class 'numpy.float64'>'  
with 84 stored elements in Compressed Sparse Row format>
```

```
ng_vec_train[0]
```

```
<1x24257 sparse matrix of type '<class 'numpy.int64'>'  
with 84 stored elements in Compressed Sparse Row format>
```

To start they both have 84 elements, since it is two different representations of the same document, that makes sense. We can check a few others as well

```
ng_tfidf_train[1]
```

```
<1x24257 sparse matrix of type '<class 'numpy.float64'>'  
with 202 stored elements in Compressed Sparse Row format>
```

```
ng_vec_train[1]
```

```
<1x24257 sparse matrix of type '<class 'numpy.int64'>'  
with 202 stored elements in Compressed Sparse Row format>
```

```
(ng_vec_train[4]>0).sum() == (ng_tfidf_train[4]>0).sum()
```

```
True
```

Let's pick out a common word so that the calculation is meaningful and do the tfidf calcuation. To find a common word in the dictionary, we'll first filter the vocabulary to keep only the words that occur at least 300 times in the training set. We sum along the columns of the matrix, transform it to an array, then iterate over the sum, enumerated (assigning the number to each element of the sum) and use that to get the word out, if its total is over 300. I saw that this is actually a sort of long list, so I chose to only print out the first 25. We print them out with the index so we can use it for the one we choose.

```
[(count_vec.get_feature_names()[i], i) for i, n in
    enumerate(np.asarray(ng_vec_train.sum(axis=0))[0])
    if n>300][:25]
```

```
/opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will
be removed in 1.2. Please use get_feature_names_out instead.
    warnings.warn(msg, category=FutureWarning)
```

```
[('about', 3326),
 ('all', 3819),
 ('also', 3874),
 ('an', 3964),
 ('and', 4003),
 ('any', 4115),
 ('are', 4263),
 ('article', 4347),
 ('as', 4363),
 ('at', 4456),
 ('available', 4612),
 ('be', 4851),
 ('been', 4887),
 ('bit', 5096),
 ('but', 5582),
 ('by', 5605),
 ('can', 5779),
 ('chip', 6191),
 ('clipper', 6393),
 ('com', 6568),
 ('computer', 6786),
 ('could', 7180),
 ('cs', 7426),
 ('data', 7687),
 ('db', 7730)]
```

Let's use computer.

```
computer_idx = 6786
count_vec.get_feature_names()[computer_idx]
```

```
'computer'
```

```
ng_vec_train[:,computer_idx].toarray()[:10].T
```

```
array([[0, 0, 0, 5, 0, 0, 0, 0, 0, 0]])
```

```
ng_tfidf_train[:,computer_idx].toarray()[:10].T
```

```
array([[0.          , 0.          , 0.          , 0.06907742, 0.          ,
         0.          , 0.          , 0.          , 0.          , 0.        ]])
```

So, we can see they have non zero elements in the same places, meaning that in both representations the column refers to the same thing.

We can compare the untransformed to the count vectorizer:

```
len(ng_X_train[0].split())
```

```
100
```

```
ng_vec_train[0].sum()
```

```
100
```

We see that it is just a count of the number of total words, not unique words, but total

```
ng_tfidf_train[0].sum()
```

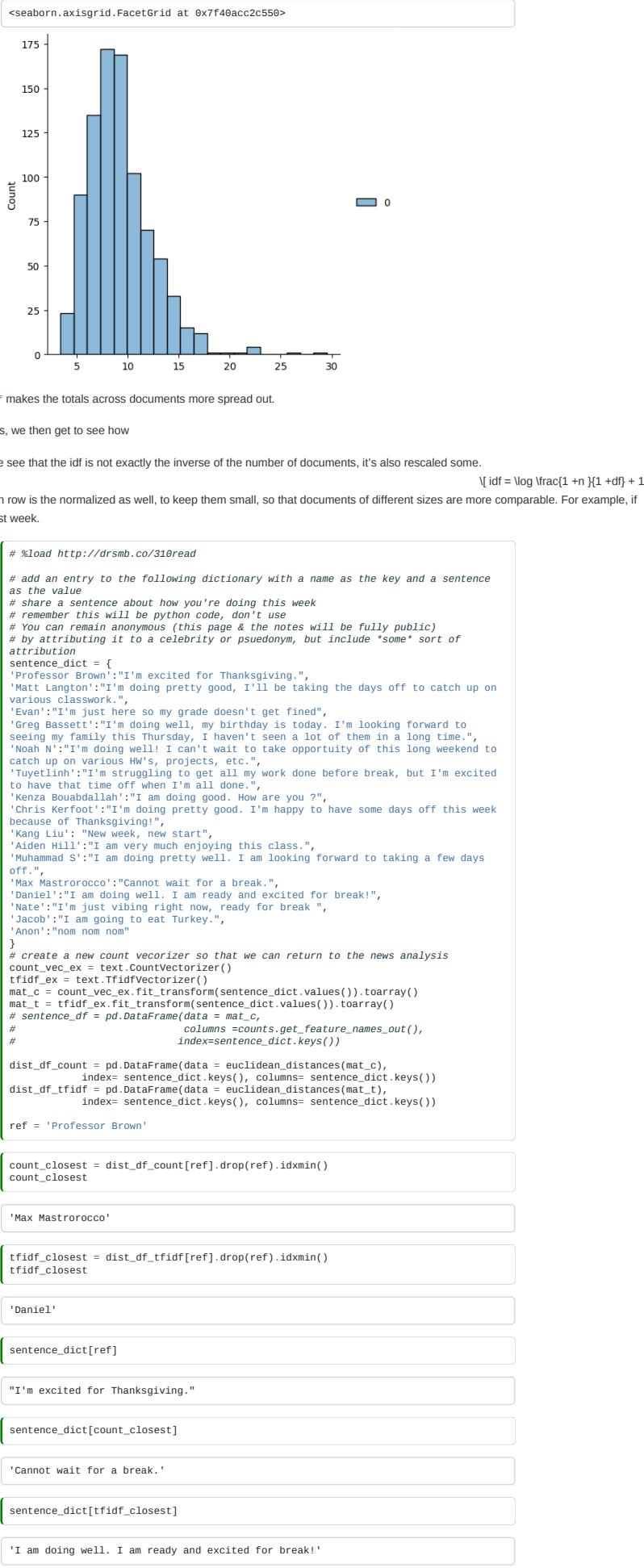
```
7.558255873996122
```

The tf-idf matrix, however is normalized to make the sums smaller. Each row is not the same, but it is more similar.

```
sns.displot(ng_vec_train.sum(axis=1),bins=20)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f40c24efb20>
```

```
sns.displot(ng_tfidf_train.sum(axis=1),bins=20)
```



We can see that the `tf-idf` makes the totals across documents more spread out.

When we sum across words, we then get to see how

From the documentation we see that the idf is not exactly the inverse of the number of documents, it's also rescaled some.

$$\text{idf} = \log \frac{1+n}{1+df} + 1$$

In this implementation, each row is the normalized as well, to keep them small, so that documents of different sizes are more comparable. For example, if we return to what we did last week.

```
# %load http://drsmrb.co/310read
# add an entry to the following dictionary with a name as the key and a sentence
# as the value
# share a sentence about how you're doing this week
# remember this will be python code, don't use
# You can remain anonymous (this page & the notes will be fully public)
# by attributing it to a celebrity or pseudonym, but include *some* sort of
# attribution
sentence_dict = {
    'Professor Brown': "I'm excited for Thanksgiving.",
    'Matt Langton': "I'm doing pretty good, I'll be taking the days off to catch up on
various classwork.",
    'Evan': "I'm just here so my grade doesn't get fined",
    'Greg Bassett': "I'm doing well, my birthday is today. I'm looking forward to
seeing my family this Thursday, I haven't seen a lot of them in a long time.",
    'Noah N': "I'm doing well! I can't wait to take opportunity of this long weekend to
catch up on various HW's, projects, etc.",
    'Tuyetlinh': "I'm struggling to get all my work done before break, but I'm excited
to have that time off when I'm all done.",
    'Kenze Bouabdallah': "I am doing good. How are you ?",
    'Chris Kerfoot': "I'm doing pretty good. I'm happy to have some days off this week
because of Thanksgiving!",
    'Kang Liu': "New week, new start",
    'Aiden Hill': "I am very much enjoying this class.",
    'Muhammad S': "I am doing pretty well. I am looking forward to taking a few days
off.",
    'Max Mastrorocco': "Cannot wait for a break.",
    'Daniel': "I am doing well. I am ready and excited for break!",
    'Nate': "I'm just vibing right now, ready for break",
    'Jacob': "I am going to eat Turkey.",
    'Anon': "nom nom nom"
}
# create a new count vectorizer so that we can return to the news analysis
count_vec_ex = text.CountVectorizer()
tfidf_ex = text.TfidfVectorizer()
mat_c = count_vec_ex.fit_transform(sentence_dict.values()).toarray()
mat_t = tfidf_ex.fit_transform(sentence_dict.values()).toarray()
# sentence_df = pd.DataFrame(data = mat_c,
#                             columns = counts.get_feature_names_out(),
#                             index=sentence_dict.keys())
# sentence_df['ref'] = ref
dist_df_count = pd.DataFrame(data = euclidean_distances(mat_c),
                             index= sentence_dict.keys(), columns= sentence_dict.keys())
dist_df_tfidf = pd.DataFrame(data = euclidean_distances(mat_t),
                             index= sentence_dict.keys(), columns= sentence_dict.keys())
ref = 'Professor Brown'

count_closest = dist_df_count[ref].drop(ref).idxmin()
count_closest

'Max Mastrorocco'

tfidf_closest = dist_df_tfidf[ref].drop(ref).idxmin()
tfidf_closest

'Daniel'

sentence_dict[ref]

"I'm excited for Thanksgiving."

sentence_dict[count_closest]

'Cannot wait for a break.'

sentence_dict[tfidf_closest]

'I am doing well. I am ready and excited for break!'
```

Now, the closest sentence actually shares a word that's similar.

```
[ dist_df_count[ref].drop(ref).sort_values()
```

```
Max Mastrocco      2.236068
Nate              2.828427
Jacob             2.828427
Kenza Bouabdallah 3.000000
Kang Liu          3.000000
Aiden Hill        3.000000
Daniel            3.162278
Evan              3.316625
Anon              3.464102
Chris Kerfoot     3.872983
Muhammad S        4.123106
Matt Langton      4.242641
Noah N            4.898979
Tuyetlinh         5.099820
Greg Bassett      5.196152
Name: Professor Brown, dtype: float64
```

```
[ dist_df_tfidf[ref].drop(ref).sort_values()
```

```
Daniel           1.153404
Max Mastrocco   1.248393
Chris Kerfoot    1.279299
Nate             1.299066
Tuyetlinh        1.341626
Noah N           1.414214
Kenza Bouabdallah 1.414214
Aiden Hill       1.414214
Muhammad S       1.414214
Jacob             1.414214
Anon              1.414214
Matt Langton     1.414214
Evan              1.414214
Greg Bassett     1.414214
Kang Liu          1.414214
Name: Professor Brown, dtype: float64
```

Now, the distances are all much smaller and Chris who also talked about Thanksgiving is much higher on the list too.

```
[ mat_c.sum(axis=1)
```

```
array([ 3, 15,  8, 22, 19, 19,  6, 14,  4,  6, 12,  4,  9,  7,  5,  3])
```

```
[ mat_t.sum(axis=1)
```

```
array([1.72586983, 3.81967092, 2.81780765, 4.4761216 , 4.1953457 ,
       3.75971487, 2.39375957, 3.68712845, 1.61398516, 2.41468677,
       3.22271608, 1.97816206, 2.73725829, 2.62109557, 2.17780895,
       1. ])
```

The sums are again much closer so the total length isn't as much of the signal. Since it's smaller, we can also sum along words.

```
[ sent_word_df = pd.DataFrame(data = count_vec_ex.get_feature_names(),
                               columns = ['word'])
sent_word_df['count_tot'] = mat_c.sum(axis=0)
sent_word_df['tfidf_tot'] = mat_t.sum(axis=0)
sent_word_df['doc_count'] = (mat_c>0).sum(axis=0)
```

```
/opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will
be removed in 1.2. Please use get_feature_names_out instead.
warnings.warn(msg, category=FutureWarning)
```

```
[ sent_word_df.sort_values(by='doc_count', ascending=False)
```

	word	count_tot	tfidf_tot	doc_count
73	to	9	1.513881	7
17	doing	7	1.326396	7
1	am	7	1.943291	5
51	off	4	0.860791	4
70	this	4	0.899345	4
...
32	happy	1	0.327365	1
31	grade	1	0.374658	1
29	going	1	0.517461	1
25	fined	1	0.374658	1
86	you	1	0.479913	1

87 rows × 4 columns

31.2. Sparse Matrices

To understand the sparse matrices, we can examine the size of the objects as a sparse matrix and after exporting to array.

```
[ getsizeof(ng_vec_train[0])
```

```
48
```

```
[ getsizeof(ng_vec_train[0].toarray())
```

```
194184
```

This is another reason it is better to use the sparse matrices (as returned by `transform`, but not by `fit_transform`)

```

ng_tfidf_train[0].toarray()
array([[0., 0., 0., ..., 0., 0., 0.]])

```

31.3. More Practice

- On the sentence dataset, try implementing the tf-idf calculation.

32. Neural Networks

We started thinking about machine learning with the idea that the basic idea is that we assume that our target variable (y_i) is related to the features (\mathbf{x}_i) by some function (for sample i):

$$y_i = f(\mathbf{x}_i)$$

But we don't know that function exactly, so we assume a type of f (a decision tree, a boundary for SVM, a probability distribution) that has some parameters (θ) and then use a machine learning algorithm (A) to estimate the parameters for f . In the decision tree the parameters are the thresholds to compare to, in GaussianNB the parameters are the mean and variance, in SVM it's the support vectors that define the margin.

$$\theta = A(X, y)$$

That we can use to test on our test data:

$$\hat{y}_i = f(x_i|\theta)$$

A neural net allows us to not assume a specific form for f first, it does universal function approximation. For one hidden layer and a binary classification problem:

$$f(x) = W_2g(W_1^T x + b_1) + b_2$$

where the function g is called the activation function. so we approximate some unknown, complicated function f by taking a weighted sum of all of the inputs, and passing those through another, known function.

```

from sklearn.neural_network import MLPClassifier
from sklearn import svm
import pandas as pd
import sklearn

from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn import model_selection
import numpy as np

```

We're going to use the digits dataset again.

```

digits = datasets.load_digits()
digits_X = digits.data
digits_y = digits.target
X_train, X_test, y_train, y_test =
model_selection.train_test_split(digits_X, digits_y)

digits.images[0]

```

```

array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0., 13., 10.,  0.,  0.,  0.]])

```

Sklearn provides an estimator for the MLP. We can see one with one layer to start.

```

mlp = MLPClassifier(
    hidden_layer_sizes=16,
    max_iter=500,
    alpha=1e-4,
    solver="lbfgs",
    verbose=10,
    random_state=1,
    learning_rate_init=0.1,
)

```

```

mlp.fit(X_train,y_train)
mlp.score(X_test,y_test)

```

```
RUNNING THE L-BFGS-B CODE
* * *
Machine precision = 2.220D-16
N =           1210      M =          10
At X0       0 variables are exactly at the bounds
At iterate   0    f=  9.21394D+00  |proj g|=  6.96113D+00
At iterate   1    f=  8.02949D+00  |proj g|=  7.20151D+00
At iterate   2    f=  3.17892D+00  |proj g|=  1.90047D+00
At iterate   3    f=  2.37772D+00  |proj g|=  3.92032D-01
At iterate   4    f=  2.26518D+00  |proj g|=  2.39824D-01
At iterate   5    f=  2.12881D+00  |proj g|=  3.01494D-01
At iterate   6    f=  1.99075D+00  |proj g|=  2.80028D-01
At iterate   7    f=  1.74864D+00  |proj g|=  9.20891D-01
At iterate   8    f=  1.67175D+00  |proj g|=  7.35058D-01
At iterate   9    f=  1.57395D+00  |proj g|=  3.62738D-01
At iterate  10    f=  1.49096D+00  |proj g|=  3.70005D-01
At iterate  11    f=  1.41542D+00  |proj g|=  2.41860D-01
At iterate  12    f=  1.27716D+00  |proj g|=  5.45331D-01
At iterate  13    f=  1.16193D+00  |proj g|=  3.01562D-01
At iterate  14    f=  1.10269D+00  |proj g|=  4.23750D-01
```

At iterate	15	f= 1.04333D+00	proj g = 3.37038D-01
At iterate	16	f= 1.00828D+00	proj g = 2.49417D-01
At iterate	17	f= 9.77489D-01	proj g = 1.86955D-01
At iterate	18	f= 9.31590D-01	proj g = 2.27472D-01
At iterate	19	f= 8.95976D-01	proj g = 4.06987D-01
At iterate	20	f= 8.59700D-01	proj g = 6.32608D-01
At iterate	21	f= 7.63536D-01	proj g = 3.91407D-01
At iterate	22	f= 7.20586D-01	proj g = 2.28068D-01
At iterate	23	f= 6.82277D-01	proj g = 2.87723D-01
At iterate	24	f= 6.34411D-01	proj g = 1.28163D-01
At iterate	25	f= 6.08008D-01	proj g = 6.66831D-01
At iterate	26	f= 5.88159D-01	proj g = 2.56877D-01
At iterate	27	f= 5.73585D-01	proj g = 1.81511D-01
At iterate	28	f= 5.49799D-01	proj g = 2.50269D-01
At iterate	29	f= 5.35880D-01	proj g = 5.76846D-01
At iterate	30	f= 5.20464D-01	proj g = 3.56778D-01
At iterate	31	f= 4.95145D-01	proj g = 2.19841D-01
At iterate	32	f= 4.67316D-01	proj g = 2.45132D-01
At iterate	33	f= 4.39916D-01	proj g = 3.17734D-01
At iterate	34	f= 4.15319D-01	proj g = 1.27921D-01
At iterate	35	f= 3.94491D-01	proj g = 4.06433D-01
At iterate	36	f= 3.59498D-01	proj g = 2.27178D-01
At iterate	37	f= 3.43046D-01	proj g = 1.07692D+00
At iterate	38	f= 3.20384D-01	proj g = 1.79385D-01
At iterate	39	f= 3.17130D-01	proj g = 1.53941D-01
At iterate	40	f= 3.03833D-01	proj g = 3.28224D-01
At iterate	41	f= 2.89769D-01	proj g = 2.39816D-01
At iterate	42	f= 2.52962D-01	proj g = 4.17232D-01
At iterate	43	f= 2.25673D-01	proj g = 2.80079D-01
At iterate	44	f= 2.10442D-01	proj g = 1.22768D-01
At iterate	45	f= 2.00895D-01	proj g = 8.13992D-02
At iterate	46	f= 1.90053D-01	proj g = 1.35335D-01
At iterate	47	f= 1.81659D-01	proj g = 1.87329D-01
At iterate	48	f= 1.78592D-01	proj g = 1.71960D-01
At iterate	49	f= 1.68644D-01	proj g = 1.52082D-01
At iterate	50	f= 1.58898D-01	proj g = 5.45019D-02
At iterate	51	f= 1.49575D-01	proj g = 1.42574D-01
At iterate	52	f= 1.42930D-01	proj g = 3.29864D-01
At iterate	53	f= 1.33162D-01	proj g = 1.19325D-01
At iterate	54	f= 1.27400D-01	proj g = 4.45197D-02
At iterate	55	f= 1.22236D-01	proj g = 8.16185D-02
At iterate	56	f= 1.15283D-01	proj g = 2.47361D-01
At iterate	57	f= 1.03659D-01	proj g = 1.56306D-01
At iterate	58	f= 9.95361D-02	proj g = 2.83905D-01
At iterate	59	f= 8.94936D-02	proj g = 5.53279D-02
At iterate	60	f= 8.72876D-02	proj g = 4.40299D-02
At iterate	61	f= 8.20028D-02	proj g = 6.74452D-02
At iterate	62	f= 7.71450D-02	proj g = 6.25825D-02
At iterate	63	f= 7.22496D-02	proj g = 4.79554D-02
At iterate	64	f= 6.91726D-02	proj g = 4.39867D-02
At iterate	65	f= 6.54796D-02	proj g = 3.78230D-02
At iterate	66	f= 6.25983D-02	proj g = 6.21870D-02
At iterate	67	f= 5.95564D-02	proj g = 3.63085D-02
At iterate	68	f= 5.73791D-02	proj g = 2.67269D-02
At iterate	69	f= 5.55690D-02	proj g = 2.47738D-02
At iterate	70	f= 5.21131D-02	proj g = 3.03027D-02
At iterate	71	f= 5.06966D-02	proj g = 6.22612D-02
At iterate	72	f= 4.86920D-02	proj g = 2.27376D-02
At iterate	73	f= 4.62419D-02	proj g = 1.97917D-02
At iterate	74	f= 4.49030D-02	proj g = 3.50337D-02
At iterate	75	f= 4.33872D-02	proj g = 3.88594D-02
At iterate	76	f= 4.13018D-02	proj g = 3.20638D-02
At iterate	77	f= 3.87933D-02	proj g = 2.47388D-02
At iterate	78	f= 3.73743D-02	proj g = 2.62607D-02
At iterate	79	f= 3.61676D-02	proj g = 1.96904D-02

```
At iterate  80  f=  3.50509D-02  |proj g|=  1.99354D-02
At iterate  81  f=  3.41404D-02  |proj g|=  1.99970D-02
At iterate  82  f=  3.23345D-02  |proj g|=  3.03628D-02
At iterate  83  f=  3.01240D-02  |proj g|=  4.26369D-02
At iterate  84  f=  2.79662D-02  |proj g|=  3.40368D-02
At iterate  85  f=  2.68383D-02  |proj g|=  2.12147D-02
At iterate  86  f=  2.63630D-02  |proj g|=  1.69560D-02
At iterate  87  f=  2.53514D-02  |proj g|=  1.10637D-02
At iterate  88  f=  2.39804D-02  |proj g|=  3.89764D-02
At iterate  89  f=  2.21337D-02  |proj g|=  2.36642D-02
At iterate  90  f=  2.05953D-02  |proj g|=  1.24658D-02
At iterate  91  f=  1.93152D-02  |proj g|=  1.44193D-02
At iterate  92  f=  1.86624D-02  |proj g|=  2.44650D-02
At iterate  93  f=  1.69807D-02  |proj g|=  2.26765D-02
At iterate  94  f=  1.58075D-02  |proj g|=  1.19084D-02
At iterate  95  f=  1.46978D-02  |proj g|=  9.22824D-03
At iterate  96  f=  1.38885D-02  |proj g|=  6.99225D-03
At iterate  97  f=  1.24851D-02  |proj g|=  1.93524D-02
At iterate  98  f=  1.13885D-02  |proj g|=  1.06720D-02
At iterate  99  f=  1.06152D-02  |proj g|=  1.63657D-02
```

This problem is unconstrained.

At iterate 100	f= 1.00794D-02	proj g = 1.75176D-02
At iterate 101	f= 9.71293D-03	proj g = 1.08156D-02
At iterate 102	f= 9.17202D-03	proj g = 6.38810D-03
At iterate 103	f= 8.68433D-03	proj g = 9.91086D-03
At iterate 104	f= 8.14962D-03	proj g = 1.40256D-02
At iterate 105	f= 7.61381D-03	proj g = 8.56425D-03
At iterate 106	f= 7.19185D-03	proj g = 9.61315D-03
At iterate 107	f= 6.56444D-03	proj g = 1.12913D-02
At iterate 108	f= 6.06836D-03	proj g = 2.18418D-02
At iterate 109	f= 5.41736D-03	proj g = 7.34607D-03
At iterate 110	f= 4.98156D-03	proj g = 6.25928D-03
At iterate 111	f= 4.46318D-03	proj g = 6.95943D-03
At iterate 112	f= 4.01820D-03	proj g = 6.92203D-03
At iterate 113	f= 3.69396D-03	proj g = 4.50527D-03
At iterate 114	f= 3.46068D-03	proj g = 3.26397D-03
At iterate 115	f= 3.09172D-03	proj g = 4.69429D-03
At iterate 116	f= 3.00082D-03	proj g = 5.58495D-03
At iterate 117	f= 2.87025D-03	proj g = 4.17833D-03
At iterate 118	f= 2.69155D-03	proj g = 3.75929D-03
At iterate 119	f= 2.51782D-03	proj g = 2.48791D-03
At iterate 120	f= 2.35152D-03	proj g = 2.56094D-03
At iterate 121	f= 2.25661D-03	proj g = 6.11287D-03
At iterate 122	f= 2.15062D-03	proj g = 2.34199D-03
At iterate 123	f= 2.11502D-03	proj g = 1.56548D-03
At iterate 124	f= 2.01610D-03	proj g = 2.47716D-03
At iterate 125	f= 1.88142D-03	proj g = 2.40020D-03
At iterate 126	f= 1.73070D-03	proj g = 3.65219D-03
At iterate 127	f= 1.58524D-03	proj g = 2.01177D-03
At iterate 128	f= 1.52637D-03	proj g = 4.48475D-03
At iterate 129	f= 1.45780D-03	proj g = 5.91255D-03
At iterate 130	f= 1.36442D-03	proj g = 1.17823D-03
At iterate 131	f= 1.31822D-03	proj g = 9.46583D-04
At iterate 132	f= 1.22873D-03	proj g = 1.71734D-03
At iterate 133	f= 1.13557D-03	proj g = 2.46574D-03
At iterate 134	f= 1.03305D-03	proj g = 1.93215D-03
At iterate 135	f= 9.68786D-04	proj g = 9.92793D-04
At iterate 136	f= 8.92769D-04	proj g = 1.16227D-03
At iterate 137	f= 8.29744D-04	proj g = 9.01028D-04
At iterate 138	f= 7.98302D-04	proj g = 8.86185D-04
At iterate 139	f= 7.73947D-04	proj g = 5.19081D-04
At iterate 140	f= 7.52566D-04	proj g = 4.68165D-04
At iterate 141	f= 7.27890D-04	proj g = 6.58627D-04
At iterate 142	f= 6.94529D-04	proj g = 4.60710D-04
At iterate 143	f= 6.77523D-04	proj g = 7.93142D-04
At iterate 144	f= 6.36941D-04	proj g = 1.06073D-03
At iterate 145	f= 6.13390D-04	proj g = 5.43111D-04
At iterate 146	f= 6.03222D-04	proj g = 5.11147D-04
At iterate 147	f= 5.58080D-04	proj g = 1.34224D-03
At iterate 148	f= 5.16415D-04	proj g = 1.12333D-03
At iterate 149	f= 5.03268D-04	proj g = 4.12541D-04
At iterate 150	f= 4.88449D-04	proj g = 3.39703D-04
At iterate 151	f= 4.62255D-04	proj g = 1.83025D-03
At iterate 152	f= 4.32271D-04	proj g = 4.58479D-04
At iterate 153	f= 3.99419D-04	proj g = 2.91894D-04
At iterate 154	f= 3.75600D-04	proj g = 3.63013D-04
At iterate 155	f= 3.57935D-04	proj g = 6.62849D-04
At iterate 156	f= 3.36780D-04	proj g = 5.39031D-04
At iterate 157	f= 3.11109D-04	proj g = 6.87490D-04
At iterate 158	f= 3.05801D-04	proj g = 7.09437D-04
At iterate 159	f= 3.01148D-04	proj g = 6.85326D-04
At iterate 160	f= 3.01059D-04	proj g = 6.84953D-04
At iterate 161	f= 3.01059D-04	proj g = 6.84952D-04

* * *

Tit = total number of iterations
Tnf = total number of function evaluations
Tint = total number of segments explored during Cauchy searches
Skip = number of BFGS updates skipped

```

Nact = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

* * *

      N   Tit    Tnf   Tnint   Skip   Nact    Projg       F
1210    161    207      1      0      0  6.850D-04  3.011D-04
      F =  3.0105877981689177E-004

CONVERGENCE: REL_REDUCTION_OF_F_<= _FACTR*EPSMCH

```

```

Warning: more than 10 function and gradient
evaluations in the last line search. Termination
may possibly be caused by a bad search direction.

```

```

0.9022222222222223

```

We can also see what happens if we increase the size of the hidden layer.

```

mlp = MLPClassifier(
    hidden_layer_sizes=(64),
    max_iter=500,
    alpha=1e-4,
    solver="lbfgs",
    verbose=10,
    random_state=1,
    learning_rate_init=0.1,
)

```

```

mlp.fit(X_train,y_train)
mlp.score(X_test,y_test)

```

RUNNING THE L-BFGS-B CODE

```
* * *
Machine precision = 2.220D-16
N =      4810      M =          10
At X0      0 variables are exactly at the bounds
At iterate   0      f=  1.03698D+01      |proj g|=  8.21359D+00
At iterate   1      f=  9.89552D+00      |proj g|=  4.86534D+00
```

```

At iterate  2   f=  8.35623D+00  |proj g|=  4.39855D+00
At iterate  3   f=  6.69828D+00  |proj g|=  2.67936D+00
At iterate  4   f=  5.11330D+00  |proj g|=  2.12734D+00
At iterate  5   f=  3.60312D+00  |proj g|=  2.83417D+00
At iterate  6   f=  2.14808D+00  |proj g|=  1.02876D+00
At iterate  7   f=  1.34821D+00  |proj g|=  5.74396D-01
At iterate  8   f=  1.04227D+00  |proj g|=  6.15047D-01
At iterate  9   f=  7.36471D-01  |proj g|=  3.33989D-01
At iterate 10   f=  5.27506D-01  |proj g|=  1.96413D-01
At iterate 11   f=  3.79183D-01  |proj g|=  2.11529D-01
At iterate 12   f=  2.68873D-01  |proj g|=  1.34314D-01
At iterate 13   f=  2.22457D-01  |proj g|=  7.54018D-02
At iterate 14   f=  1.81703D-01  |proj g|=  6.61648D-02
At iterate 15   f=  1.48369D-01  |proj g|=  7.16455D-02
At iterate 16   f=  1.28007D-01  |proj g|=  4.29253D-02
At iterate 17   f=  1.08685D-01  |proj g|=  5.14468D-02
At iterate 18   f=  7.87950D-02  |proj g|=  6.68861D-02
At iterate 19   f=  5.74328D-02  |proj g|=  4.55264D-02
At iterate 20   f=  4.67714D-02  |proj g|=  2.98632D-02
At iterate 21   f=  3.78612D-02  |proj g|=  2.91199D-02
At iterate 22   f=  3.23081D-02  |proj g|=  5.09173D-02
At iterate 23   f=  2.63184D-02  |proj g|=  1.63397D-02
At iterate 24   f=  2.14007D-02  |proj g|=  1.69408D-02
At iterate 25   f=  1.68296D-02  |proj g|=  1.71434D-02
At iterate 26   f=  1.43154D-02  |proj g|=  2.48857D-02
At iterate 27   f=  1.17778D-02  |proj g|=  1.50679D-02
At iterate 28   f=  9.11650D-03  |proj g|=  1.03592D-02
At iterate 29   f=  7.70156D-03  |proj g|=  3.13526D-02
At iterate 30   f=  6.33229D-03  |proj g|=  8.93632D-03
At iterate 31   f=  5.45635D-03  |proj g|=  4.96107D-03
At iterate 32   f=  4.29434D-03  |proj g|=  9.01255D-03
At iterate 33   f=  3.79157D-03  |proj g|=  3.44570D-02
At iterate 34   f=  2.90241D-03  |proj g|=  8.73803D-03
At iterate 35   f=  2.60408D-03  |proj g|=  4.47301D-03
At iterate 36   f=  2.22855D-03  |proj g|=  5.15857D-03
At iterate 37   f=  1.82981D-03  |proj g|=  4.50291D-03
At iterate 38   f=  1.40092D-03  |proj g|=  6.39691D-03
At iterate 39   f=  1.01507D-03  |proj g|=  3.63847D-03
At iterate 40   f=  8.16133D-04  |proj g|=  1.50603D-03
At iterate 41   f=  6.08546D-04  |proj g|=  1.32486D-03
At iterate 42   f=  3.79811D-04  |proj g|=  1.16492D-03
At iterate 43   f=  2.78027D-04  |proj g|=  2.64965D-03
At iterate 44   f=  1.83863D-04  |proj g|=  7.17014D-04
At iterate 45   f=  1.51841D-04  |proj g|=  2.78337D-04
At iterate 46   f=  1.13005D-04  |proj g|=  2.28644D-04
At iterate 47   f=  7.75315D-05  |proj g|=  4.85824D-04
At iterate 48   f=  6.89936D-05  |proj g|=  1.34093D-03
At iterate 49   f=  3.86465D-05  |proj g|=  1.85248D-04
At iterate 50   f=  3.46442D-05  |proj g|=  1.09727D-04
At iterate 51   f=  2.59627D-05  |proj g|=  5.16781D-05
* * *
Tit  = total number of iterations
Tnf  = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip = number of BFGS updates skipped
Nact = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F    = final function value
* * *
N      Tit      Tnf  Tnint  Skip  Nact   Projg      F
4810      51       53       1     0     0  5.168D-05  2.596D-05
F = 2.5962701458240261E-005
CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<= PGtol

```

This problem is unconstrained.

0.9688888888888889

We can compare it to SVM:

```
svm_clf = svm.SVC(gamma=0.001)
svm_clf.fit(X_train, y_train)
svm_clf.score(X_test,y_test)
```

```
0.9955555555555555
```

We can also have multiple hidden layers:

```
mlp = MLPClassifier(
    hidden_layer_sizes=(64, 64),
    max_iter=500,
    alpha=1e-4,
    solver="lbfgs",
    verbose=10,
    random_state=1,
    learning_rate_init=0.1,
)
mlp.fit(X_train,y_train)
mlp.score(X_test,y_test)
```

RUNNING THE L-BFGS-B CODE

```
* * *
Machine precision = 2.2200E-16
N =      8970      M =         10
At X0      0 variables are exactly at the bounds
At iterate   0   f=  7.62355D+00   |proj g|=  5.20464D+00
At iterate   1   f=  6.54295D+00   |proj g|=  5.36564D+00
At iterate   2   f=  4.50180D+00   |proj g|=  2.15553D+00
At iterate   3   f=  3.29943D+00   |proj g|=  1.40946D+00
At iterate   4   f=  2.54324D+00   |proj g|=  1.23591D+00
At iterate   5   f=  1.89734D+00   |proj g|=  7.40883D-01
At iterate   6   f=  1.47412D+00   |proj g|=  6.39404D-01
At iterate   7   f=  1.09479D+00   |proj g|=  3.60860D-01
At iterate   8   f=  7.93742D-01   |proj g|=  2.80646D-01
At iterate   9   f=  5.60834D-01   |proj g|=  3.11080D-01
At iterate  10   f=  4.27864D-01   |proj g|=  2.51791D-01
At iterate  11   f=  3.54756D-01   |proj g|=  1.69336D-01
At iterate  12   f=  2.86905D-01   |proj g|=  1.20048D-01
At iterate  13   f=  2.50500D-01   |proj g|=  2.93602D-01
At iterate  14   f=  2.14021D-01   |proj g|=  1.27666D-01
At iterate  15   f=  1.94758D-01   |proj g|=  1.19318D-01
At iterate  16   f=  1.59668D-01   |proj g|=  1.33700D-01
At iterate  17   f=  1.39745D-01   |proj g|=  2.15761D-01
At iterate  18   f=  1.16795D-01   |proj g|=  5.80537D-02
At iterate  19   f=  1.05279D-01   |proj g|=  4.01591D-02
At iterate  20   f=  9.61984D-02   |proj g|=  4.67903D-02
At iterate  21   f=  8.33562D-02   |proj g|=  7.65066D-02
At iterate  22   f=  7.22347D-02   |proj g|=  1.11024D-01
At iterate  23   f=  6.14001D-02   |proj g|=  6.78444D-02
At iterate  24   f=  4.67901D-02   |proj g|=  3.67616D-02
At iterate  25   f=  3.91106D-02   |proj g|=  2.62545D-02
At iterate  26   f=  3.02623D-02   |proj g|=  5.67919D-02
At iterate  27   f=  2.81939D-02   |proj g|=  1.14898D-01
At iterate  28   f=  2.19821D-02   |proj g|=  2.20006D-02
At iterate  29   f=  2.00730D-02   |proj g|=  2.19508D-02
At iterate  30   f=  1.70421D-02   |proj g|=  3.02863D-02
At iterate  31   f=  1.64277D-02   |proj g|=  5.87695D-02
At iterate  32   f=  1.35969D-02   |proj g|=  1.17029D-02
At iterate  33   f=  1.29498D-02   |proj g|=  8.74430D-03
At iterate  34   f=  1.15333D-02   |proj g|=  1.78542D-02
At iterate  35   f=  1.02687D-02   |proj g|=  1.20929D-02
At iterate  36   f=  8.00548D-03   |proj g|=  3.22610D-02
At iterate  37   f=  6.63484D-03   |proj g|=  2.16319D-02
At iterate  38   f=  5.92750D-03   |proj g|=  1.09963D-02
At iterate  39   f=  5.15555D-03   |proj g|=  1.08891D-02
```

This problem is unconstrained.

```

At iterate  40   f=  4.63862D-03  |proj g|=  1.14842D-02
At iterate  41   f=  4.00801D-03  |proj g|=  1.15733D-02
At iterate  42   f=  3.05215D-03  |proj g|=  5.36168D-03
At iterate  43   f=  2.36358D-03  |proj g|=  7.14234D-03
At iterate  44   f=  2.03758D-03  |proj g|=  2.12563D-02
At iterate  45   f=  1.49506D-03  |proj g|=  5.11690D-03
At iterate  46   f=  1.26685D-03  |proj g|=  2.77705D-03
At iterate  47   f=  1.06285D-03  |proj g|=  1.88592D-03
At iterate  48   f=  8.85728D-04  |proj g|=  2.30460D-03
At iterate  49   f=  6.98372D-04  |proj g|=  5.35378D-03
At iterate  50   f=  5.59329D-04  |proj g|=  1.55352D-03
At iterate  51   f=  4.68438D-04  |proj g|=  1.25558D-03
At iterate  52   f=  3.77392D-04  |proj g|=  9.26127D-04
At iterate  53   f=  2.72551D-04  |proj g|=  1.10296D-03
At iterate  54   f=  1.99338D-04  |proj g|=  2.46813D-03
At iterate  55   f=  1.11941D-04  |proj g|=  5.16635D-04
At iterate  56   f=  9.23810D-05  |proj g|=  4.48153D-04
At iterate  57   f=  6.81568D-05  |proj g|=  2.70260D-04
At iterate  58   f=  4.55703D-05  |proj g|=  1.85431D-04
At iterate  59   f=  3.70315D-05  |proj g|=  5.47486D-04
At iterate  60   f=  2.49469D-05  |proj g|=  1.83485D-04
At iterate  61   f=  2.19028D-05  |proj g|=  9.76855D-05
* * *
Tit  = total number of iterations
Tnf  = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F    = final function value
* * *
N    Tit     Tnf   Tnint Skip   Nact     Projg      F
8970   61      63      1      0      0  9.769D-05  2.190D-05
F =  2.1902830198013581E-005
CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL

```

```
0.9733333333333334
```

We saw that the SVM performed a bit better, but this is a simple problem. We can also compare these based on much they store, the number of parameters is realted to the complexity.

```

svm_clf.support_vectors_.shape
(679, 64)
[c.shape for c in mlp.coefs_]
[(64, 64), (64, 64), (64, 10)]
np.prod(list(svm_clf.support_vectors_.shape))
43456
np.sum([np.prod(list(c.shape)) for c in mlp.coefs_])
8832

```

We see this is much smaler.

32.1. Questions after class

32.1.1. How can we use this in our assignment?

You do not have to, but you could try an MLP in your assignment, but all that is required is any classifier and a text representation.

32.1.2. How do we know how to change the parameters?

If it doesn't work well, trying more layers or bigger layers is a good idea.

Neural nets work like a black box, they're hard to interpret, so while there are good heuristics, there isn't as solid theory for how to know what do to with them.

They work well, but because they're hard to understand, that's a risk of using them.

33. Predicting with Neural Networks

```

from scipy.special import expit
from sklearn.datasets import make_classification
from sklearn.neural_network import MLPClassifier

from sklearn import svm
import pandas as pd
import numpy as np
import sklearn

from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn import model_selection
# from sklearn.model_selection import train_test_split

import seaborn as sns
sns.set_theme(palette='colorblind')

```

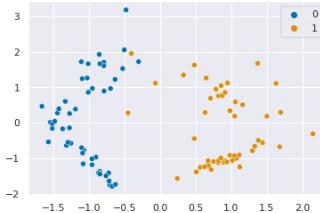
Today, we're going to use very simple data in order to examine how a neural network works.

```

X, y = make_classification(n_samples=100,
                           random_state=1, n_features=2, n_redundant=0)
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
                                                                    stratify=y,
                                                                    random_state=1)
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y)

```

<AxesSubplot:>



First, we'll train and score a tiny neural net: with 1 hidden layer of 1 neuron.

```

clf = MLPClassifier(
    hidden_layer_sizes=(1), # 1 hidden layer, 1 artificial neuron
    max_iter=100, # maximum 100 iterations in optimization
    alpha=1e-4, # regularization
    solver='lbfgs', # optimization algorithm
    verbose=10, # how much detail to print
    activation='identity' # how to transform the hidden layer before passing it to
    the next layer
)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)

```

RUNNING THE L-BFGS-B CODE

```

* * *

Machine precision = 2.220D-16
N = 5 M = 10
At X0      0 variables are exactly at the bounds
At iterate  0 f= 1.00933D+00  |proj g|= 4.21069D-01
At iterate  1 f= 5.94993D-01  |proj g|= 2.33065D-01
At iterate  2 f= 2.31377D-01  |proj g|= 1.94550D-01
At iterate  3 f= 7.77706D-02  |proj g|= 4.71313D-02
At iterate  4 f= 6.46813D-02  |proj g|= 2.78380D-02
At iterate  5 f= 5.67453D-02  |proj g|= 1.44718D-02
At iterate  6 f= 5.34348D-02  |proj g|= 1.36082D-02
At iterate  7 f= 5.18003D-02  |proj g|= 1.20832D-02
At iterate  8 f= 4.83562D-02  |proj g|= 2.52769D-03
At iterate  9 f= 4.80518D-02  |proj g|= 1.64091D-03
At iterate 10 f= 4.79184D-02  |proj g|= 1.18365D-03
At iterate 11 f= 4.79070D-02  |proj g|= 4.29842D-04

```

This problem is unconstrained.

1.0

```

At iterate 12 f= 4.79048D-02  |proj g|= 5.25402D-05
* * *
Tit = total number of iterations
Tnf = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip = number of BFGS updates skipped
Nact = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F   = final function value
* * *
N Tit Tnf Tnint Skip Nact Projg F
5 12 13 1 0 0 5.254D-05 4.790D-02
F = 4.7904815861860413E-002
CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL

```

Tip

This is actually equivalent to another classifier, called logistic Regression

Correction

I've removed the parameter `learning_rate_init=0.1` and `random_state=1` because `sklearn` actually only uses the learning rate and randomization for some of the optimization algorithms: adam and sgd, which we are not using.

These algorithms are good in high dimensional problems, our problem is simple, so we don't need the advanced parameters or algorithms.

Now we can see that it actually has another activation, that we didn't change the output layer still has a logistic activation layer, which we want. If we didn't then the output layer wouldn't be able to be interpreted as a probability, because probability always needs to be between 0 and 1.

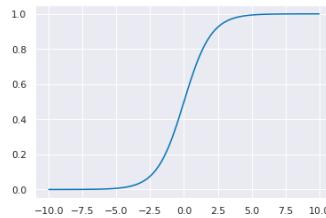
```
clf.out_activation_
```

```
'logistic'
```

The logistic function looks like this:

```
x_logistic = np.linspace(-10,10,100)
y_logistic = expit(x_logistic)
plt.plot(x_logistic,y_logistic)
```

```
[<matplotlib.lines.Line2D at 0x7f2a226cc640>]
```



The fit method learned the following weights:

```
clf.coefs_
```

```
[array([[ 4.4346835 ,  
       [-0.12768095]]),  
 array([[2.96383812]])]
```

and biases

```
clf.intercepts_
```

```
[array([1.75515035]), array([0.4937787])]
```

These are called coefficients and intercepts because the weights are multiplied by the inputs and the biases you can interpret as geometrically as shifting things, like a line intercept (recall $y=mx+b$)

33.1. Reconstructing the Predict method

we'll use an acutally new point, we can make one up

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

```
list
```

we want a numpy array so we will cast it

```
pt = np.array([-1,2])
```

```
type(pt)
```

```
numpy.ndarray
```

numpy's `matmul` does matrix multiplication (multiply columns by rows element wise and sum)

$$l\{f(x) = W_2g(W_1^T x + b_1) + b_2\}$$

the $l(g)$ is the activation function, which we set to identity $l(g(x) = x)$ so we don't have to do more

```
np.matmul(pt,clf.coefs_[0]) + clf.intercepts_[0])*clf.coefs_[1] +  
clf.intercepts_[1]
```

```
Input In [11]  
np.matmul(pt,clf.coefs_[0]) + clf.intercepts_[0])*clf.coefs_[1] +  
clf.intercepts_[1]  
^  
SyntaxError: unmatched '''
```

but we're not quite done, the output layer still transforms using the logistic function, which is also known as `expit` and we have imported from scipy.

```
expit((np.matmul(pt,clf.coefs_[0]) + clf.intercepts_[0])*clf.coefs_[1] +  
clf.intercepts_[1])
```

```
array([[0.00027327]])
```

We can compare this to the classifier's output. It outputs a probability for each class, we only computed the probabilt of the 1 class.

```
clf.predict_proba(pt)
```

```
array([[9.99726730e-01, 2.73270493e-04]])
```

and we can see how it predicts on that point.

```
clf.predict(pt)
```

```
array([0])
```

A single artificial neuron like the function below. where it has parameters that have to be determined before we can use it on an input vector.

```

def aritificial_neuron_template(activation,weights,bias,inputs):
    """
    simple artificial neuron

    Parameters
    -----
    activation : function
        activation function of the neuron
    weights : numpy array
        weights for summing inputs
    bias: numpy array
        bias term added to the weighted sum
    inputs : numpy array
        input to the neuron
    """

    return activation(np.matmul(inputs,weights) +bias)

# two common activation functions
identity_activation = lambda x: x
logistic_activation = lambda x: expit(x)

```

When we instantiate the multilayer perceptron object, `MLPClassifier`, we pick the activation function and when we give data to the `fit` method, we get the weights and biases.

A neural network passes the data to the hidden layer, and the output of the hidden layer to the output layer. In our neural network, we have just one neuron at each layer.

So the `predict_proba` method is the same as the following:

```

aritificial_neuron_template(logistic_activation,clf.coefs_[1],clf.intercepts_[1],
                            aritificial_neuron_template(identity_activation,clf.coefs_[0],
                            clf.intercepts_[0],pt))

array([[0.00027327]])

```

To make this easier to read, we can make the intermediate neurons their own lambda functions.

```

hidden_neuron = lambda x:
    aritificial_neuron_template(identity_activation,clf.coefs_[0],clf.intercepts_[0],x
)
output_neuron = lambda x:
    aritificial_neuron_template(expit,clf.coefs_[1],clf.intercepts_[1],x
)
output_neuron(hidden_neuron(pt))

array([[0.00027327]])

```

We can confirm that this works the same as the predict probability method:

```

clf.predict_proba(pt)

array([[9.99726730e-01, 2.73270493e-04]])

```

33.2. More Features and More Hidden Neurons

First, we'll sample more features and then train a new classifier

```

x, y = make_classification(n_samples=100,
                           random_state=1,n_features=4,n_redundant=0)
X_train, X_test, y_train, y_test = train_test_split(x, y, stratify=y,
                                                    random_state=5)
pt_4d = np.asarray([[-1,-2,2,-1],[1.5,0,.5,1]])
clf_4d = MLPClassifier(
    hidden_layer_sizes=(1),
    max_iter=5000,
    alpha=1e-4,
    solver="lbfgs",
    verbose=10,
    activation= 'identity'
)
clf_4d.fit(X_train, y_train)
clf_4d.score(X_test, y_test)

NameError: name 'train_test_split' is not defined

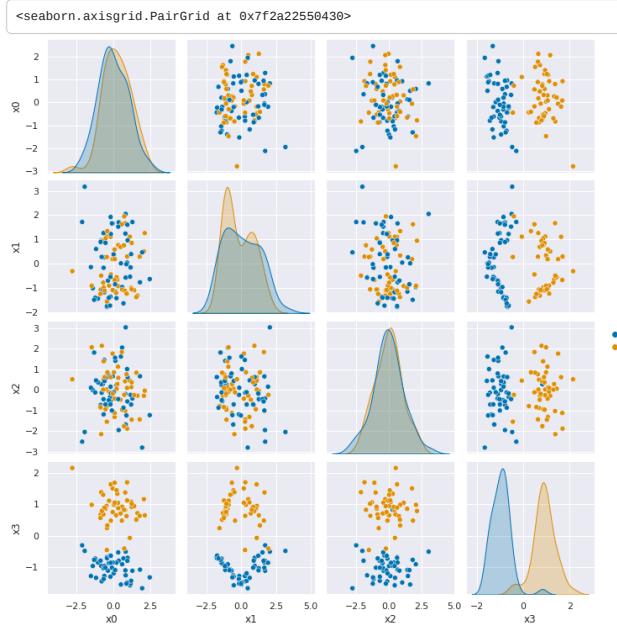
```

We can look at this data

```

df = pd.DataFrame(X,columns=['x0','x1','x2','x3'])
df['y'] = y
sns.pairplot(df,hue='y')

```



and based on this, we'll pick a new pair of points to test on:

```
pt_4d = np.asarray([-2, 2, 2, -2], [1.5, 0, -1, 3])
```

This neural network is just like the one before:

```
hidden_neuron_4d = lambda x: aritificial_neuron_template(identity_activation,
clf_4d.coefs_[0],clf_4d.intercepts_[0],x)
output_neuron_4d = lambda x: aritificial_neuron_template(logistic_activation,
clf_4d.coefs_[1],clf_4d.intercepts_[1],x)

output_neuron_4d(hidden_neuron_4d(pt_4d))
```

```
NameError: Traceback (most recent call last)
Input In [22], in <cell line: 7>()
      1 hidden_neuron_4d = lambda x:
aritificial_neuron_template(identity_activation,
      2
clf_4d.coefs_[0],clf_4d.intercepts_[0],x)
      3 output_neuron_4d = lambda x:
aritificial_neuron_template(logistic_activation,
      4
clf_4d.coefs_[1],clf_4d.intercepts_[1],x)
----> 7 output_neuron_4d(hidden_neuron_4d(pt_4d))

Input In [22], in <lambda>(x)
      1 hidden_neuron_4d = lambda x:
aritificial_neuron_template(identity_activation,
----> 2
clf_4d.coefs_[0],clf_4d.intercepts_[0],x)
      3 output_neuron_4d = lambda x:
aritificial_neuron_template(logistic_activation,
      4
clf_4d.coefs_[1],clf_4d.intercepts_[1],x)
----> 7 output_neuron_4d(hidden_neuron_4d(pt_4d))

NameError: name 'clf_4d' is not defined
```

```
clf_4d.predict_proba(pt_4d)
```

```
NameError: Traceback (most recent call last)
Input In [23], in <cell line: 1>()
----> 1 clf_4d.predict_proba(pt_4d)

NameError: name 'clf_4d' is not defined
```

However, remember this one was not as accurate:

```
clf_4d.score(X_test, y_test)
```

```
NameError: Traceback (most recent call last)
Input In [24], in <cell line: 1>()
----> 1 clf_4d.score(X_test, y_test)

NameError: name 'clf_4d' is not defined
```

To try improving it, we will add more layers and a different activation function:

```
clf_4d_4h = MLPClassifier(  
    hidden_layer_sizes=(4),  
    max_iter=500,  
    alpha=1e-4,  
    solver="lbfgs",  
    verbose=10,  
    activation='logistic'  
)  
  
clf_4d_4h.fit(X_train, y_train)  
  
clf_4d_4h.score(X_test, y_test)
```

```

RUNNING THE L-BFGS-B CODE
* * *
Machine precision = 2.220D-16
N =          17      M =          10
At X0          0 variables are exactly at the bounds
At iterate    0   f=  7.18499D-01   |proj g|=  9.74112D-02
At iterate    1   f=  6.72406D-01   |proj g|=  6.29961D-02
At iterate    2   f=  3.01475D-01   |proj g|=  9.56919D-02
At iterate    3   f=  1.38170D-01   |proj g|=  4.06613D-02
At iterate    4   f=  6.77357D-02   |proj g|=  9.78179D-03
At iterate    5   f=  6.51807D-02   |proj g|=  1.22121D-02
At iterate    6   f=  6.00664D-02   |proj g|=  1.28390D-02
At iterate    7   f=  5.75132D-02   |proj g|=  3.21291D-03
At iterate    8   f=  5.56862D-02   |proj g|=  5.58919D-03
At iterate    9   f=  5.20499D-02   |proj g|=  1.34643D-02
At iterate   10   f=  4.61949D-02   |proj g|=  7.23477D-03
At iterate   11   f=  4.35800D-02   |proj g|=  6.75115D-03
At iterate   12   f=  4.27648D-02   |proj g|=  4.83810D-03
At iterate   13   f=  4.10512D-02   |proj g|=  2.08123D-03
At iterate   14   f=  3.88969D-02   |proj g|=  7.42822D-03
At iterate   15   f=  3.75522D-02   |proj g|=  9.73559D-03
At iterate   16   f=  3.50586D-02   |proj g|=  3.91136D-03
At iterate   17   f=  3.34533D-02   |proj g|=  3.15949D-03
At iterate   18   f=  3.06723D-02   |proj g|=  5.01449D-03
At iterate   19   f=  2.92569D-02   |proj g|=  4.49681D-03
At iterate   20   f=  2.83902D-02   |proj g|=  4.04087D-03
At iterate   21   f=  2.78496D-02   |proj g|=  2.44612D-03
At iterate   22   f=  2.74076D-02   |proj g|=  1.79654D-03
At iterate   23   f=  2.71177D-02   |proj g|=  1.56286D-03
At iterate   24   f=  2.69760D-02   |proj g|=  5.05145D-03
At iterate   25   f=  2.64063D-02   |proj g|=  9.13487D-04
At iterate   26   f=  2.63157D-02   |proj g|=  4.85479D-04
At iterate   27   f=  2.62292D-02   |proj g|=  1.09661D-03
At iterate   28   f=  2.60554D-02   |proj g|=  2.03385D-03
At iterate   29   f=  2.56516D-02   |proj g|=  2.84952D-03
At iterate   30   f=  2.51594D-02   |proj g|=  1.68102D-03
At iterate   31   f=  2.49320D-02   |proj g|=  1.53555D-04
At iterate   32   f=  2.48248D-02   |proj g|=  6.36782D-04
At iterate   33   f=  2.46692D-02   |proj g|=  1.15171D-03
At iterate   34   f=  2.44561D-02   |proj g|=  1.53376D-03
At iterate   35   f=  2.42813D-02   |proj g|=  3.57333D-04
At iterate   36   f=  2.40270D-02   |proj g|=  1.29140D-03
At iterate   37   f=  2.39980D-02   |proj g|=  7.63292D-04
At iterate   38   f=  2.39071D-02   |proj g|=  3.66406D-04
At iterate   39   f=  2.38332D-02   |proj g|=  6.61710D-04
At iterate   40   f=  2.36665D-02   |proj g|=  1.12128D-03
At iterate   41   f=  2.35738D-02   |proj g|=  1.11525D-03
At iterate   42   f=  2.34903D-02   |proj g|=  7.14403D-04
At iterate   43   f=  2.34663D-02   |proj g|=  6.26687D-04
At iterate   44   f=  2.33466D-02   |proj g|=  9.94480D-04
At iterate   45   f=  2.33089D-02   |proj g|=  6.43085D-04
At iterate   46   f=  2.32698D-02   |proj g|=  7.43577D-04
At iterate   47   f=  2.31520D-02   |proj g|=  1.67045D-04
At iterate   48   f=  2.29391D-02   |proj g|=  5.82541D-04
At iterate   49   f=  2.28617D-02   |proj g|=  6.98080D-04
At iterate   50   f=  2.28150D-02   |proj g|=  6.08483D-04
At iterate   51   f=  2.28056D-02   |proj g|=  5.98656D-04
At iterate   52   f=  2.27790D-02   |proj g|=  2.05310D-04
At iterate   53   f=  2.27711D-02   |proj g|=  1.87164D-04
At iterate   54   f=  2.27396D-02   |proj g|=  3.75885D-04
At iterate   55   f=  2.26873D-02   |proj g|=  4.58487D-04
At iterate   56   f=  2.26358D-02   |proj g|=  5.87288D-04
At iterate   57   f=  2.24729D-02   |proj g|=  1.10490D-03
At itera

```

This problem is unconstrained.

1.0

	58	f=	2.22662D-02	proj g =	1.52310D-03
At iterate	59	f=	2.20527D-02	proj g =	1.39027D-03
At iterate	60	f=	2.18159D-02	proj g =	1.15635D-03
At iterate	61	f=	2.17452D-02	proj g =	9.99020D-04
At iterate	62	f=	2.16545D-02	proj g =	4.94297D-04
At iterate	63	f=	2.15798D-02	proj g =	1.18304D-04
At iterate	64	f=	2.15660D-02	proj g =	1.07849D-04
At iterate	65	f=	2.15462D-02	proj g =	1.41216D-04
At iterate	66	f=	2.15397D-02	proj g =	1.83221D-04
At iterate	67	f=	2.15312D-02	proj g =	1.49328D-04
At iterate	68	f=	2.15176D-02	proj g =	1.20358D-04
At iterate	69	f=	2.14852D-02	proj g =	2.52188D-04
At iterate	70	f=	2.14530D-02	proj g =	2.13058D-04
At iterate	71	f=	2.14324D-02	proj g =	3.17254D-04
At iterate	72	f=	2.13779D-02	proj g =	2.60563D-04
At iterate	73	f=	2.13483D-02	proj g =	1.79275D-04
At iterate	74				

we see some improvement.

This network is more complicated. It has 5 total neurons:

```

hidden_neuron_4d_h0 = lambda x: aritificial_neuron_template(logistic_activation,
                                                               clf_4d_4h.coefs_[0]
                                                               [:, 0],clf_4d_4h.intercepts_[0][0],x)
hidden_neuron_4d_h1 = lambda x: aritificial_neuron_template(logistic_activation,
                                                               clf_4d_4h.coefs_[0]
                                                               [:, 1],clf_4d_4h.intercepts_[0][1],x)
hidden_neuron_4d_h2 = lambda x: aritificial_neuron_template(logistic_activation,
                                                               clf_4d_4h.coefs_[0]
                                                               [:, 2],clf_4d_4h.intercepts_[0][2],x)
hidden_neuron_4d_h3 = lambda x: aritificial_neuron_template(logistic_activation,
                                                               clf_4d_4h.coefs_[0]
                                                               [:, 3],clf_4d_4h.intercepts_[0][3],x)
output_neuron_4d_h4 = lambda x: aritificial_neuron_template(logistic_activation,
                                                               clf_4d_4h.coefs_[1],clf_4d_4h.intercepts_[1],x)

```

f=	2.13314D-02	proj g =	1.17966D-04		
At iterate	75	f=	2.13240D-02	proj g =	1.00953D-04
At iterate	76	f=	2.13206D-02	proj g =	2.15596D-04
At iterate	77	f=	2.13069D-02	proj g =	1.56939D-04
At iterate	78	f=	2.12862D-02	proj g =	8.13704D-05

Tit = total number of iterations
 Tnf = total number of function evaluations
 Tnint = total number of segments explored during Cauchy searches
 Skip = number of BFGS updates skipped
 Nact = number of active bounds at final generalized Cauchy point
 Projg = norm of the final projected gradient
 F = final function value

```

    * * *
    N      Tit      Tnf      Tnint     Skip     Nact     Projg
    17      78       111        1         0         0   8.137D-
    F = 2.1286191548877748E-002

CONVERGENCE: NORM OF PROJECTED GRADIENT <= PCTOL
```

And we have to take the output of all 4 hidden neurons into the output neuron, because they are a single layer, not in sequence.

```

ValueError                                Traceback (most recent call last)
Input In [27], in <cell line: 1>()
----> 1 output_neuron_4d_4h(np.asarray([hidden_neuron_4d_h0(pt_4d),
  2             hidden_neuron_4d_h1(pt_4d),
  3             hidden_neuron_4d_h2(pt_4d),
  4             hidden_neuron_4d_h3(pt_4d)]).T)

Input In [26], in <lambda>(x):
----> 1 hidden_neuron_4d_h0 = lambda x:
aritificial_neuron_template(logistic_activation,
 2
clf_4d_4h.coefs_[0][:,0],clf_4d_4h.intercepts_[0][0],x)
 3 hidden_neuron_4d_h1 = lambda x:
aritificial_neuron_template(logistic_activation,
 4
clf_4d_4h.coefs_[0][:,1],clf_4d_4h.intercepts_[0][1],x)
 5 hidden_neuron_4d_h2 = lambda x:
aritificial_neuron_template(logistic_activation,
 6
clf_4d_4h.coefs_[0][:,2],clf_4d_4h.intercepts_[0][2],x)

Input In [15], in aritificial_neuron_template(activation, weights, bias, inputs)
 1 def aritificial_neuron_template(activation,weights,bias,inputs):
 2     """
 3         simple artificial neuron
 4
 5     ...
 6
 7     return activation(np.matmul(inputs,weights) +bias)

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with
gufunc signature (n?,k),(k,m?)->(n?,m?) (size 2 is different from 4)

```

And again, we see this is the probability of predicting 1:

```

clf_4d_4h.predict_proba(pt_4d)

ValueError                                Traceback (most recent call last)
Input In [28], in <cell line: 1>()
----> 1 clf_4d_4h.predict_proba(pt_4d)

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:1252, in
MLPClassifier.predict_proba(self, X)
1238 """Probability estimates.
1239
1240 Parameters
1241 -----
1242     model, where classes are ordered as they are in `self.classes_`.
1250 """
1251 check_is_fitted(self)
-> 1252 y_pred = self._forward_pass_fast(X)
1254 if self.n_outputs_ == 1:
1255     y_pred = y_pred.ravel()

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:160, in
BaseMultilayerPerceptron._forward_pass_fast(self, X)
144     def _forward_pass_fast(self, X):
145         """Predict using the trained model
146
147         This is the same as _forward_pass but does not record the activations
148
149         The decision function of the samples for each class in the model.
159     """
--> 160     X = self._validate_data(X, accept_sparse=["csr", "csc"], reset=False)
162     # Initialize first layer
163     activation = X

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/base.py:600, in BaseEstimator._validate_data(self, X, y, reset,
validate_separately, **check_params)
597     out = X, y
599 if not no_val_X and check_params.get("ensure_2d", True):
--> 600     self._check_n_features(X, reset=reset)
602 return out

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/sklearn/base.py:400, in BaseEstimator._check_n_features(self, X, reset)
397     return
399 if n_features != self.n_features_in_:
--> 400     raise ValueError(
401         f"X has {n_features} features, but {self.__class__.__name__} "
402         f"is expecting {self.n_features_in_} features as input."
403     )

ValueError: X has 4 features, but MLPClassifier is expecting 2 features as input.

```

33.3. (optional) What is a numerical optimization algorithm?

Numerical Optimization algorithms are at the core of many of the fit methods.

One way we can optimize a function is to take the derivative, set it equal to zero and solve for the parameter. If we know the function is convex (like a bowl or valley shape) then the place where the derivative (slope) is 0 is the bottom or lowest point of the valley.

Numerical optimization is for when we can't analytically solve that problem once we set it equal to zero. Optimization algorithms are sort of like search algorithms but can work in high dimensions and use strategy based on calculus.

The basic idea in many numerical optimization algorithms is to start at a point (initial setting of the coefficients in this case) and then compute the value of the function then change the coefficients a little and compute again. We can use those two points to see if the direction we "moved" or the way we changed the parameters made it better or worse. If it was better, we change them more in the same direction, (if we made both smaller then we make them both smaller again) if it got worse, we change in a different direction.

You can think of this like trying to find the bottom of a valley, without being able to see, just check your altitude. You take a step left, right, forward or back and then see if your altitude went up or down.

LBFGS actually uses the derivative, so it's like you can see the direction of the hill you're on, but you have to keep taking steps and then if you reach a point where you can't go down anymore you know you are done. When the algorithm finds it can't get better, that's called convergence.

Stochastic gradient descent works in high dimensions where it's too hard to do the derivative, but you can randomly move in different directions (or take the partial derivative in a small number of directions). Adam is a special version of that with better strategy.

Numerical optimization is a whole research area. In graduate school, I took a whole semester long course just learning different algorithms for this.

34. Review, IDEA, & Preparing for Deep Learning

34.1. Review for Level 1

Remember that to earn a C or better (or for you level 2 achievements to influence) your grade at all you have to earn all of the level 1s.

Review the [Achievement Definitions](#) to be sure you know what we are looking for.

34.2. Representation

We change the representation of data when it is not tabular. As we've seen with text and images, we have to transform the data into a tabular form for the algorithms we have seen so far. Deep learning combines learning a representation with learning a prediction rule.

34.3. Matching Tools to Steps in the ML Pipeline

pandas	loading data, cleaning data, summarizing
seaborn	visualization
BeautifulSoup	webscraping
sklearn	fitting models, evaluating models, transforming data for modeling

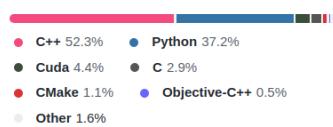
34.4. Deep Learning Ecosystem

While `sklearn` can do basic neural networks, it doesn't have many types of neurons or fancy layers.

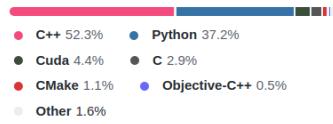
What makes deep learning work is its ability to transform data in new ways. We will see them in more detail.

Two popular Deep Learning Libraries are [Pytorch \(code\)](#) and [TensorFlow \(code\)](#), both are open source. These are both complex, high performance libraries with optimized code. We have used python in class because it is a user-friendly programming language, but it is not optimal, performance-wise, so much of the code in these libraries is in C++.

Languages



Languages



[Keras](#) is a high level library written on top of tensorflow in python. The goal of this library is to enable fast experimentation. You can think of the relationship between `tensorflow` and `keras` like the relationship between `matplotlib` and `seaborn`.

In `seaborn` we got high level plot figures with high level parameters like using variables in our dataset to create rows and columns of subplots. In `matplotlib`, we would have to separately create the subplot grid, then break the data apart, and assign a plot using a subset of the data for each subplot. However, under the hood, `seaborn` uses `matplotlib`. Just as we have mostly used `seaborn`, but occasionally use `matplotlib` to tweak things, we will use `keras`.

Deep learning is computationally expensive, which is why the code is so optimized. However this means installing deep learning libraries is harder. They require more dependencies than we have worked with to date. Also, the code is optimized in order to work on high performance hardware: GPUs and other parallel computing systems, which most laptops do not have. We will use [Google Colab](#) to do deep learning in class.

⚠ Warning

There is a whole semester long course on just Deep Learning (CSC541). We will not cover everything; the goal is just enough that you know where to start.

If you want to use deep learning after this class alone, you'll need to study substantially on your own, but past students have taken this foundation and made it work.

💡 Note

Note however that the relationship here is somewhat different, as `keras` is actually a submodule of `tensorflow`. Technically, the relationship is probably closer to `matplotlib` as a whole to `matplotlib.pyplot`, but we have not used base `matplotlib` at all.

34.5. IDEA

The IDEA feedback is important feedback for helping Faculty, Departments, and URI at large monitor how we are doing at teaching. Filling out this feedback is important, even though it does not impact your time in this class. You fill it out to help future students in this class and help instructors improve our overall teaching, and you could see any of us again.

[IDEA Feedback](#)

💡 Important

If the class gets to 70% response rate, you'll all be allowed to reply to portfolio 4 feedback and change earn achievements you were close on but didn't quite get. (whatever number is appropriate; still a deadline on 12/22, so you'll get ~24 hours to reply & change your grade)

34.6. More Practice

💡 Important

these questions will help you be prepared for questions to earn the last few achievements in class time.

1. What questions would you like to ask a Data scientist about how they do their work? Bring questions to the last two classes for our speakers (see bios from Prismia on 12/3).

2. How do different machine learning tasks (classification, regression, clustering) compare?
3. How do different models within a task compare?
4. What can your model's performance tell you?
5. Pick an ML application and think about what data, and tools you would need to use in order to replicate the system.
6. Review the import cells in different classes and be sure you know why we import each package, module, class, and function
7. What does each part of the output of a `GridSearchCV` experiment tell you?

35. Neural Networks with Keras

```
import numpy as np           # advanced math library
import matplotlib.pyplot as plt
import random               # for generating random numbers

from keras.datasets import mnist # MNIST dataset is included in Keras
from keras.models import Sequential # Model type to be used

from keras.layers.core import Dense, Dropout, Activation # Types of layers to be
used in our model
from keras.utils import np_utils      # NumPy related tools

from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D,
GlobalAveragePooling2D, Flatten
from keras.layers import BatchNormalization
```

```
-----
ModuleNotFoundError          Traceback (most recent call last)
Input In [1], in <cell line: 5>()
      2 import matplotlib.pyplot as plt
      3 import random               # for generating random numbers
--> 5 from keras.datasets import mnist # MNIST dataset is included in Keras
      6 from keras.models import Sequential # Model type to be used
      8 from keras.layers.core import Dense, Dropout, Activation # Types of layers
to be used in our model

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/keras/_init_.py:21, in <module>
    15 """Implementation of the Keras API, the high-level API of TensorFlow.
    16 
    17 Detailed documentation and user guides are available at
    18 [keras.io](https://keras.io).
    19 """
   20 # pylint: disable=unused-import
--> 21 from tensorflow.python import tf2
   22 from keras import distribute
   24 from keras import models

ModuleNotFoundError: No module named 'tensorflow'
```

First, we'll use `keras` to build similar networks to the ones we saw with `sklearn` it will be more complex, and we're using a different version of the digits data (larger images) but this will still be just learning the predictions, not the representation for now. On Friday, we'll learn how to add new layers that transform the data and learn the representation at the same time.

35.1. Preparing data for deep learning

The MNIST data is split between 60,000 28 x 28 pixel training images and 10,000 28 x 28 pixel images. It's realated to the digits data that we have seen before.

We will load the data and look at a random sample.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()

plt.rcParams['figure.figsize'] = (9,9) # Make the figures a bit bigger

for i in range(9):
    plt.subplot(3,3,i+1)
    num = random.randint(0, len(X_train))
    plt.imshow(X_train[num], cmap='gray', interpolation='none')
    plt.title("Class {}".format(y_train[num]))

plt.tight_layout()
```

```
NameError          Traceback (most recent call last)
Input In [2], in <cell line: 1>()
--> 1 (X_train, y_train), (X_test, y_test) = mnist.load_data()
      4 plt.rcParams['figure.figsize'] = (9,9) # Make the figures a bit bigger
      6 for i in range(9):

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

Next, we need to transform the data to be compatible by reshaping 28 x 28 matrices into vectors, then converting to floats and normalizing.

28*28

784

Now that we know the length, we can transform.

```
X_train = X_train.reshape(60000, 784) # 60,000 training samples
X_test = X_test.reshape(10000, 784) # 10,000 test samples

X_train = X_train.astype('float32') # change integers to 32-bit floating point
numbers
X_test = X_test.astype('float32')

X_train /= 255                  # normalize each value for each pixel for the entire
vector for each input
X_test /= 255

print("Training matrix shape", X_train.shape)
print("Testing matrix shape", X_test.shape)
```

```

NameError Traceback (most recent call last)
Input In [4], in <cell line: 1>()
----> 1 X_train = X_train.reshape(60000, 784) # 60,000 training samples
      2 X_test = X_test.reshape(10000, 784) # 10,000 test samples
      3 X_train = X_train.astype('float32') # change integers to 32-bit floating
      point numbers

NameError: name 'X_train' is not defined

```

We will use one-hot encoding for the target variable, since our neural net's output layer is actually the probability distribution over the 10 digits, not a value from 0 to 9.

```

nb_classes = 10 # number of unique digits
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

```

```

NameError Traceback (most recent call last)
Input In [5], in <cell line: 3>()
----> 1 nb_classes = 10 # number of unique digits
      2 Y_train = np_utils.to_categorical(y_train, nb_classes)
      3 Y_test = np_utils.to_categorical(y_test, nb_classes)

NameError: name 'np_utils' is not defined

```

```
y_train[0]
```

```

NameError Traceback (most recent call last)
Input In [6], in <cell line: 1>()
----> 1 y_train[0]

NameError: name 'y_train' is not defined

```

```
Y_train[0]
```

```

NameError Traceback (most recent call last)
Input In [7], in <cell line: 1>()
----> 1 Y_train[0]

NameError: name 'Y_train' is not defined

```

35.2. Building a neural network in Keras

We will start with a network similar to what we have seen in `sklearn`. It will have a number of layers and, when predicting pass data from one layer to the next sequentially.

```
model = Sequential()
```

```

NameError Traceback (most recent call last)
Input In [8], in <cell line: 1>()
----> 1 model = Sequential()

NameError: name 'Sequential' is not defined

```

Next we add layers. In `keras` what we saw as one neuron (connections + activation) before is treated as two separate layers, where the size of the layer is the number of neurons.

```

model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))

```

```

NameError Traceback (most recent call last)
Input In [9], in <cell line: 1>()
----> 1 model.add(Dense(512, input_shape=(784,)))
      2 model.add(Activation('relu'))

NameError: name 'model' is not defined

```

```

model.add(Dense(512))
model.add(Activation('relu'))

```

```

NameError Traceback (most recent call last)
Input In [10], in <cell line: 1>()
----> 1 model.add(Dense(512))
      2 model.add(Activation('relu'))

NameError: name 'model' is not defined

```

```

model.add(Dense(10))
model.add(Activation('softmax'))

```

```

NameError Traceback (most recent call last)
Input In [11], in <cell line: 1>()
----> 1 model.add(Dense(10))
      2 model.add(Activation('softmax'))

NameError: name 'model' is not defined

```

```
model.summary()
```

```

NameError Traceback (most recent call last)
Input In [12], in <cell line: 1>()
----> 1 model.summary()

NameError: name 'model' is not defined

```

Keras also has parameters about the optimization, like we saw before, but we set those with the `compile` method.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

NameError                                 Traceback (most recent call last)
Input In [13], in <cell line: 1>()
----> 1 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

NameError: name 'model' is not defined

model.fit(x_train, Y_train,
          batch_size=128, epochs=5,
          verbose=1)

NameError                                 Traceback (most recent call last)
Input In [14], in <cell line: 1>()
----> 1 model.fit(x_train, Y_train,
                  2      batch_size=128, epochs=5,
                  3      verbose=1)

NameError: name 'model' is not defined

score = model.evaluate(X_test, Y_test)
score

NameError                                 Traceback (most recent call last)
Input In [15], in <cell line: 1>()
----> 1 score = model.evaluate(X_test, Y_test)
      2 score

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

35.3. What kind of mistakes did it make?

```
predicted_prob = model.predict(X_test)
predicted_classes = np.argmax(predicted_prob, axis=1)

# Check which items we got right / wrong
correct_indices = np.nonzero(predicted_classes == y_test)[0]
incorrect_indices = np.nonzero(predicted_classes != y_test)[0]

plt.figure()
for i, correct in enumerate(correct_indices[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(X_test[correct].reshape(28,28), cmap='gray', interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[correct],
                                              y_test[correct]))

plt.tight_layout()

plt.figure()
for i, incorrect in enumerate(incorrect_indices[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(X_test[incorrect].reshape(28,28), cmap='gray', interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[incorrect],
                                              y_test[incorrect]))

plt.tight_layout()
```



```
NameError                                 Traceback (most recent call last)
Input In [16], in <cell line: 1>()
----> 1 predicted_prob = model.predict(X_test)
      2 predicted_classes = np.argmax(predicted_prob, axis=1)
      3 # Check which items we got right / wrong

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

35.4. Dropout for less overfitting

Dropout makes some of the neurons zero.

```
model_dropout = Sequential()

model_dropout.add(Dense(512, input_shape=(784,)))
model_dropout.add(Activation('relu'))
model_dropout.add(Dropout(0.2))

model_dropout.add(Dense(512))
model_dropout.add(Activation('relu'))
model_dropout.add(Dropout(0.2))

model_dropout.add(Dense(10))
model_dropout.add(Activation('softmax'))

model_dropout.summary()
```



```
NameError                                 Traceback (most recent call last)
Input In [17], in <cell line: 1>()
----> 1 model_dropout = Sequential()
      2 model_dropout.add(Dense(512, input_shape=(784,)))
      3 model_dropout.add(Activation('relu'))

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

Again, we set the optimization parameters and then we can fit.

```
model_dropout.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_dropout.fit(X_train, Y_train,
                  batch_size=128, epochs=5,
                  verbose=1)
```

```

NameError                                 Traceback (most recent call last)
Input In [18], in <cell line: 1>()
----> 1 model_dropout.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
  2 model_dropout.fit(X_train, Y_train,
  3 batch_size=128, epochs=5,
  4 verbose=1)

NameError: name 'model_dropout' is not defined

```

```

score = model_dropout.evaluate(X_test, Y_test)
score

```

```

NameError                                 Traceback (most recent call last)
Input In [19], in <cell line: 1>()
----> 1 score = model_dropout.evaluate(X_test, Y_test)
      2 score

NameError: name 'model_dropout' is not defined

```

Now we see the gap between the train and test performance is smaller and the test performance is higher.

35.5. Questions

35.5.1. "What is the difference between the different activation methods?"

36. Convolutional Neural Networks

```

import numpy as np                         # advanced math library
import matplotlib.pyplot as plt            # for generating random numbers
import random
from keras.datasets import cifar10
from keras.models import Sequential # Model type to be used

from keras.layers.core import Dense, Dropout, Activation # Types of layers to be
used in our model
from keras.utils import np_utils                      # NumPy related tools

from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D,
GlobalAveragePooling2D, Flatten
from keras.layers import BatchNormalization

```

```

ModuleNotFoundError                  Traceback (most recent call last)
Input In [1], in <cell line: 5>()
      2 import matplotlib.pyplot as plt
      3 import random                      # for generating random numbers
----> 5 from keras.datasets import cifar10
      6 from keras.models import Sequential # Model type to be used
      8 from keras.layers.core import Dense, Dropout, Activation # Types of layers
to be used in our model

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/keras/_init_.py:21, in <module>
    15 """Implementation of the Keras API, the high-level API of TensorFlow.
    16
    17 Detailed documentation and user guides are available at
    18 [keras.io](https://keras.io).
    19 """
   20 # pylint: disable=unused-import
--> 21 from tensorflow.python import tf2
   22 from keras import distribute
   24 from keras import models

ModuleNotFoundError: No module named 'tensorflow'

```

```

(c_X_train, c_y_train), (c_X_test, c_y_test) = cifar10.load_data()

```

```

NameError                                 Traceback (most recent call last)
Input In [2], in <cell line: 1>()
----> 1 (c_X_train, c_y_train), (c_X_test, c_y_test) = cifar10.load_data()

NameError: name 'cifar10' is not defined

```

```

c_X_test.shape

```

```

NameError                                 Traceback (most recent call last)
Input In [3], in <cell line: 1>()
----> 1 c_X_test.shape

NameError: name 'c_X_test' is not defined

```

```

nb_classes = 10

c_X_train_flat = c_X_train.reshape(50000, 3072) #add an additional dimension to
represent the single-channel
c_X_test_flat = c_X_test.reshape(10000, 3072)

c_X_train_flat = c_X_train_flat.astype('float32')      # change integers to 32-
bit floating point numbers
c_X_test = c_X_test.astype('float32')

c_X_train_flat /= 255                                # normalize each value for each
pixel for the entire vector for each input
c_X_test /= 255

c_Y_train = np_utils.to_categorical(c_y_train, nb_classes)
c_Y_test = np_utils.to_categorical(c_y_test, nb_classes)

```

```
NameError Traceback (most recent call last)
Input In [4], in <cell line: 3>()
      1 nb_classes = 10
----> 3 c_X_train_flat = c_X_train.reshape(50000, 3072) #add an additional
dimension to represent the single-channel
      4 c_X_test_flat = c_X_test.reshape(10000, 3072)
      7 c_X_train_flat = c_X_train_flat.astype('float32')      # change
integers to 32-bit floating point numbers

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

```
model_dropout = Sequential()
model_dropout.add(Dense(512, input_shape=(3072,)))
model_dropout.add(Activation('sigmoid'))
model_dropout.add(Dropout(0.2))

model_dropout.add(Dense(512))
model_dropout.add(Activation('relu'))
model_dropout.add(Dropout(0.2))

model_dropout.add(Dense(10))
model_dropout.add(Activation('softmax'))
```

```
NameError Traceback (most recent call last)
Input In [5], in <cell line: 1>()
----> 1 model_dropout = Sequential()
      3 model_dropout.add(Dense(512, input_shape=(3072,)))
      4 model_dropout.add(Activation('sigmoid'))

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

```
model_dropout.summary()
```

```
NameError Traceback (most recent call last)
Input In [6], in <cell line: 1>()
----> 1 model_dropout.summary()

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

```
model_dropout.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_dropout.fit(c_X_train_flat, c_Y_train,
                  batch_size=128, epochs=5,
                  verbose=1)

score = model_dropout.evaluate(c_X_test_flat, c_Y_test)
score
```

```
NameError Traceback (most recent call last)
Input In [7], in <cell line: 1>()
----> 1 model_dropout.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
      2 model_dropout.fit(c_X_train_flat, c_Y_train,
      3           batch_size=128, epochs=5,
      4           verbose=1)
      6 score = model_dropout.evaluate(c_X_test_flat, c_Y_test)

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

```
c_X_train = c_X_train.astype('float32')      # change integers to 32-bit
floating point numbers
c_X_train = c_X_train.astype('float32')

c_X_train /= 255                            # normalize each value for each
pixel for the entire vector for each input
c_X_train /= 255
```

```
NameError Traceback (most recent call last)
Input In [8], in <cell line: 1>()
----> 1 c_X_train = c_X_train.astype('float32')      # change integers to 32-
bit floating point numbers
      2 c_X_train = c_X_train.astype('float32')
      4 c_X_train /= 255                            # normalize each value for
each pixel for the entire vector for each input

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

```

model_cnn = Sequential()                                     # Linear stacking of
layers

# Convolution Layer 1
model_cnn.add(Conv2D(32, (3, 3), input_shape=(32,32,3))) # 32 different 3x3
kernels -- so 32 feature maps
model_cnn.add(BatchNormalization(axis=-1))                  # normalize each feature
map before activation
convLayer01 = Activation('relu')                           # activation
model_cnn.add(convLayer01)

# Convolution Layer 2
model_cnn.add(Conv2D(32, (3, 3)))                         # 32 different 3x3
kernels -- so 32 feature maps
model_cnn.add(BatchNormalization(axis=-1))                  # normalize each feature
map before activation
model_cnn.add(Activation('relu'))                          # activation
convLayer02 = MaxPooling2D(pool_size=(2,2))               # Pool the max values over a
2x2 kernel
model_cnn.add(convLayer02)

# Convolution Layer 3
model_cnn.add(Conv2D(64,(3, 3)))                         # 64 different 3x3
kernels -- so 64 feature maps
model_cnn.add(BatchNormalization(axis=-1))                  # normalize each feature
map before activation
convLayer03 = Activation('relu')                           # activation
model_cnn.add(convLayer03)

# Convolution Layer 4
model_cnn.add(Conv2D(64, (3, 3)))                         # 64 different 3x3
kernels -- so 64 feature maps
model_cnn.add(BatchNormalization(axis=-1))                  # normalize each feature
map before activation
model_cnn.add(Activation('relu'))                          # activation
convLayer04 = MaxPooling2D(pool_size=(2,2))               # Pool the max values over a
2x2 kernel
model_cnn.add(convLayer04)
model_cnn.add(Flatten())                                  # Flatten final 4x4x64
output matrix into a 1024-length vector

# Fully Connected Layer 5
model_cnn.add(Dense(512))                                # 512 FCN nodes
model_cnn.add(BatchNormalization())                      # normalization
model_cnn.add(Activation('relu'))                        # activation

# Fully Connected Layer 6
model_cnn.add(Dropout(0.2))                            # 20% dropout of randomly
selected nodes
model_cnn.add(Dense(10))                               # final 10 FCN nodes
model_cnn.add(Activation('softmax'))                   # Softmax activation

```

```

NameError
Input In [9], in <cell line: 1>()
----> 1 model_cnn = Sequential()                         Traceback (most recent call last)
of layers
      3 # Convolution Layer 1
      4 model_cnn.add(Conv2D(32, (3, 3), input_shape=(32,32,3))) # 32 different
      3x3 kernels -- so 32 feature maps

NameError: name 'Sequential' is not defined

```

```

model_cnn.summary()

```

```

NameError
Input In [10], in <cell line: 1>()
----> 1 model_cnn.summary()                            Traceback (most recent call last)

NameError: name 'model_cnn' is not defined

```

```

model_cnn.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_cnn.fit(c_X_train, c_Y_train,
              batch_size=128, epochs=5,
              verbose=1)
score = model_cnn.evaluate(c_X_test, c_Y_test)

```

```

NameError
Input In [11], in <cell line: 1>()
----> 1 model_cnn.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
      2 model_cnn.fit(c_X_train, c_Y_train,
      3         batch_size=128, epochs=5,
      4         verbose=1)
      6 score = model_cnn.evaluate(c_X_test, c_Y_test)

NameError: name 'model_cnn' is not defined

```

```

score_cnn = model_cnn.evaluate(c_X_test, c_Y_test)
score_cnn

```

```

NameError
Input In [12], in <cell line: 1>()
----> 1 score_cnn = model_cnn.evaluate(c_X_test, c_Y_test)          Traceback (most recent call last)
      2 score_cnn

NameError: name 'model_cnn' is not defined

```

1. Portfolio Setup, Data Science, and Python

Due: 2020-09-12

1.1. Objective & Evaluation

This assignment is an opportunity to earn level 2 achievements for the `process` and `python` and confirm that you have all of your tools setup, including your portfolio.

1.2. To Do

Important

If you have trouble, check the GitHub FAQ on the left before e-mailing

```
```{warning}
If you have trouble with the (*)d steps, don't worry, we can help work around these later. To help us out, document the errors as bugs on your repository.
````
```

Your task is to:

1. Install required software from the Tools & Resource page
2. Create your portfolio, by [accepting the assignment](#)
3. Learn about your portfolio from the README file on your repository.
4. edit `_config.yml` to set your name as author and change the logo if you wish
5. Fill in `about/index.md` with information about yourself(not evaluated, but useful) and your own definition of data science (graded for **level 1 process**)
6. (*) Install some additional python packages with: `pip install pip install -r requirements.txt` (this is a python operation, so use anaconda prompt on Windows, if the pip version doesn't work, try it with conda: `conda install --file requirements.txt`) form inside the portfolio folder
7. (*) Configure precommit to help keep your repo clean with `pre-commit install`. If this step doesn't work, see the portfolio README under "Using your Jupyter Book Portfolio"
8. Add a Jupyter notebook called `grading.ipynb` to the `about` folder and write a function that computes a grade for this course, with the following docstring. Include:
 - 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
9. Add the line `- file: about/grading` in your `_toc.yml` file.

Important

remember to add, commit, and push your changes so we can see them

```
'''
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 (+/-)
'''
```

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. Submission Instructions

Create a Jupyter Notebook with your function in your portfolio folder commit and push the changes.

In your browser, view the `gh-pages` branch to see your compiled submission, as `portfolio.pdf` or by viewing your website.

There will be a pull request on your repository that is made by GitHub classroom, [request a review](#) from @rhodypro4dg/fall21instructors.

Note

If you get stuck on any of this after accepting the assignment and creating a repository, you can create an issue on your repository, describing what you're stuck on and tag us: @rhodypro4dg/fall21instructors

To do this click Issues at the top, the green "New Issue" button and then type away.

Warning

your function can have a different name than `compute_grade`, but make sure it's your function name, with those parameter values in your tests.

Note

when the value of the expression after `assert` is `True`, it will look like nothing happened. `assert` is used for testing

2. Practicing Python and Accessing Data

due : 2020-09-21

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`.

In this assignment, you'll practice/ review python skills by manipulating datasets and extracting. The following table summarizes the grading. It supplements the skill definitions from the [Achievement Definitions](#).

| Task | Skills (max level) |
|--|--------------------|
| identify possible uses for data in a data science pipeline | [process (1)] |
| load data from one file format | [access (1)] |
| load data from at least two of (.csv, .tsv, .dat, database, .json) | [access (2)] |
| compare the data formats | [access (2)] |
| complete the assignment in python | python (1)] |
| use python data types (eg dictionaries) to prepare information about datasets | [python (2)] |
| use informative variable names, pythonic iteration, and other common PEP 8 conventions | [python (2)] |
| display DataFrame properties | [summarize (1)] |

Table 2.1 rubric for grading

First, [accept the assignment](#). It contains a notebook with some template structure (and will set you up for grading).

2.2. 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 notebook that includes:

- heading of the dataset's name
- a link to where someone can learn about the dataset
- a 1-2 sentence summary of what the dataset contains and why it was collected
- 1-2 questions you would like to answer with that dataset.

💡 Hint

The [Datasets page](#) has information about data for any assignment. For this assignment, the [Best for loading directly into a notebook](#) section is probably the best place to start.

2.3. Store them for loading

Create a list of dictionaries in `datasets.py`, so that there is one dictionary for each dataset with the url, a name, and what function should be used to load the data into a `pandas.DataFrame`.

💡 Hint

The goal here is to set up this list of dictionaries so that you can load data using different functions in each pass through the loop, without an `if` statement. You'll be iterating over the list of dictionaries, so the loop variable be a different dictionary each time.

see the where we used a lambda in a dictionary in [the class notes](#) for more information.

💡 Hint

Any `.py` file can become a [module](#)

💡 Tip

Urls are strings. The `string` class in python has a lot of helpful methods for manipulating strings, like `split`.

2.4. Make a dataset about your datasets

Import the list from the `datasets` module you created in the step above. Then `iterate` over the list of dictionaries, and:

1. save it to a local csv using the short name you provided for the dataset as the file name, without writing the index column to the file.
2. record attributes about the dataset as in the table below in a list of lists:
3. Use that to create a DataFrame with the following columns:

| | |
|---------------|--|
| 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 |

Table 2.2 Meta Data Description of the DataFrame to build

2.5. Manipulate your datasets

For one dataset that includes nonnumerical data:

- display the heading and the last 4 rows
- make and display a new data frame with only the numerical columns (select these programmatically)

For any other dataset:

- display the heading and the first three rows
- display the datatype for each column
- Are there any variables where pandas may have read in the data as a datatype that's not what you expect (eg a numerical column mistaken for strings)? If so, investigate and try to figure out why.

For the third dataset:

- display the first 3 odd rows (eg 1,3,5) of the data for two columns of your choice

2.6. Exploring data files

For each dataset, in a separate section of your notebook titled `when things go wrong`:

- try reading in data with the wrong `read_` function and make notes about what happens.
- was the format that the data was provided in a good format? why or why not?
- 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.

💡 Think Ahead

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](#) eg this [google scholar result](#) How could something like this impact your work as a data scientist?

3. Assignment 3: Exploratory Data Analysis

Due: 2020-09-28 11:59pm

[Template repo for submission](#)

3.1. Objective & Evaluation

This week your goal is to do a small exploratory data analysis for two datasets of your choice.

| task | skill (max level) |
|--|------------------------------|
| compute and display overall statistics | summarize (2) |
| compute and display individual statistics of a dataset | summarize (2) |
| group a data set by a variable and compute summary statistics | summarize (2) |
| plot two different pairwise relationships | visualize (2) |
| interpret the statistics and plots | summarize (2), visualize (2) |
| load data from at least two different file types | access (2) |
| compare and contrast the file types and/or sources | access (2) |
| match questions appropriate to the dataset and match plots and stats | process (1) |

Table 3.1 plot basic views of data and generate descriptive statistics and basic plots

3.2. 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.

3.2.1. Dataset 1 (d1)

must include at least:

- two continuous valued variables and
- one categorical variable.

3.2.2. Dataset 2 (d2)

must include at least:

- two categorical variables and
- one continuous valued variable

3.3. EDA

Use a separate notebook for each dataset, name them `dataset_01.ipynb` and `dataset_02.ipynb`.

For **each** dataset, in a dedicated notebook, complete the following:

1. Load the data to a notebook as a `DataFrame` from url or local path, if local, include the data in your repository.
2. Explore the dataset in a notebook enough to describe its structure use the heading `## Description`
 - shape
 - columns
 - variable types
 - overall summary statistics
3. Write a short description of what the data contains and what it could be used for
4. Ask and answer 4 questions using statistics, split-apply-combine, and visualizations. Make a heading for each question using a markdown cell and `H2:##`. Make sure your analyses meet the criteria in the check lists below. Interpret the answer from the analysis/plot.
5. Describe what, if anything might need to be done to clean or prepare this data for further analysis

3.3.1. Dataset 1 Checklist

1. Overall summary statistics grouped by a categorical variable
2. A single statistic grouped by a categorical variable
3. a scatter plot with the points colored by a categorical variable
4. a plot and summary table that convey the same information. This can be one statistic or many.

3.3.2. Dataset 2 Analysis

1. two individual summary statistics for one variable
2. one summary statistic grouped by two categorical variables
3. a figure with a grid of subplots that correspond to two categorical variables
4. a plot and summary table that convey the same information. This can be one statistic or many.

💡 Tip

Be sure to start early and use help hours to make sure you have a plan for all of these.

📝 Think Ahead

1. How could you make more customized summary tables?
2. Could you use any of the variables in this dataset to add more variables that would make interesting ways to apply split-apply-combine? (eg thresholding a continuous value to make a categorical value)

4. Assignment 4:

Due: 2020-10-05 11:59pm

[accept the assignment](#)

⚠️ Warning

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.

⚠️ Warning

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.

| task | skill (max level) |
|---|----------------------|
| drop nan rows from a dataset | prepare (2) |
| display parts of a dataframe | summarize (1) |
| impute a value to fill missing values | prepare (2) |
| filter data based on extreme values or other outliers | prepare (2) |
| convert a variable to one hot encoding | prepare (2) |
| add a new column computed from one or more other columns | prepare (2) |
| transform a dataset to tidy format | prepare (2) |
| compute overall and individual summary statistics | summarize (2) |
| use split-apply-combine paradigm | summarize (2) |
| generate at least two types of plots | visualize (2) |
| interpret statistics and plots | summarize, visualize |
| use list comprehensions or loops and pythonic conventions | python (2) |
| load data from at least two types | access (2) |
| compare data storage formats | access (2) |
| match EDA techniques to questions appropriately | process (1) |

Table 4.1 practice basic pandas by reshaping and organizing data

For this assignment, prepare the provided datasets. Your preparation needs to include the following steps and narrative description of how you're making decisions about your data cleaning.

The notebooks in the template have instructions for how to work with each dataset.

To earn prepare level 2, clean the data and do just enough exploratory data analysis to show that the data is usable (eg 1 stat and/or plot). For prepare level 2:

- travel_times AND one of:
- cs_degrees, airlines, and coffee

To earn summarize and visualize level 2, add extra exploratory data analyses meeting the criteria above.

To earn python level 2, make sure that you use a function or lambda and comprehension or pythonic loops somewhere. The CS degrees data will have that, but it's harder. The coffee data will be the easiest one to get all python level 2.

For access level 2 you must clean the airline data (to get data in a second file type).

💡 Hint

renaming thing is often done well with a dictionary comprehension or lambda.

5. Assignment 5: Constructing Datasets and Using Databases

[accept the assignment](#)

Due: 2020-10-12 11:59pm

| task | skill |
|---|----------------------|
| drop nan rows from a dataset | prepare (2) |
| impute a value to fill missing values | prepare (2) |
| filter data based on extreme values or other outliers | prepare (2) |
| convert a variable to one hot encoding | prepare (2) |
| add a new column computed from one or more other columns | prepare (2) |
| transform a dataset to tidy format | prepare (2) |
| append a dataset provided in pieces | construct (2) |
| merge data with a shared column | construct (2) |
| compute overall and individual summary statistics | summarize (2) |
| use split-apply-combine paradigm | summarize (2) |
| generate at least two types of plots | visualize (2) |
| interpret statistics and plots | summarize, visualize |
| use list comprehensions or loops and pythonic conventions | python (2) |

Table 5.1 access data from a database and merge multiple tables from a dataset

5.1. Constructing Datasets

Your goal is to programmatically construct two ready to analyze dataset from multiple sources.

- Each dataset must combine at least 2 source tables(4 total).
- At least one source table(of the 4) must come from an sqlite database or from web scraping.
- You should use at least two different joins(types of merges, or concat).

The notebook you submit should include:

- a motivating question for why you're combining the datasets in an introduction section
- code and description of how you built and prepared each dataset. For each step describe what you're about to do, the code with output, interpretation that leads into the next step.
- exploratory data analysis that shows why you built the data and confirms that is prepared enough to analyze.

For construct, this can be very minimal EDA.

You may build one dataset from three tables instead of two from two each if you'd like

5.2. Earning additional achievements

To earn additional achievements, you must do more cleaning and/or exploratory data analysis.

5.2.1. Prepare level 2

To earn level 2 for prepare, you must, either on component table(s) or the final dataset:

- 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

5.2.2. Summarize and Visualize level 2

To earn level 2 for summarize and/or visualize, include additional analyses after building the datasets. Include:

- compute overall summary statistics
- compute individual summary statistics
- use split-apply-combine with two categorical variables
- at least two types of plots for visualize
- use a categorical variable to modify the plot (color points or create subplots)

5.2.3. Python Level 2

Use pythonic naming conventions throughout, AND:

- Use pythonic loops and a list or dictionary OR
- use a list or dictionary comprehension

Thinking Ahead

Compare the level 2 skill definitions to level 3, how could you extend and adapt what you've done to meet level 3?

6. Assignment 6: Understanding Classification

[accept the assignment](#)

Due: 2020-10-19 11:59pm

| task | skill |
|--|--------------------|
| fit a naive bayes classifier | classification (1) |
| explain when the model works/does not and what to use to investigate | classification (2) |
| evaluate the performance of a classifier | evaluate (1) |

Table 6.1 demonstrate understanding of classification as a tasks and how to evaluate them

Important

You only need to do datasets 1,2, 5, and 6.

For each dataset, answer the following:

1. Do you expect Gaussian Naive Bayes to work well on this dataset, why or why not?
 - think about the assumptions of naive bayes and classification in general
 - explanation is essential here, because you can actually use the classifier to check
2. How well does a Gaussian Naive Bayes classifier work on this dataset?
 - check the overall performance
3. How does the actual performance compare to your prediction? If it performs much better or much worse than you expected, what might you use to figure out why?

Tip

you do not have to figure out why your predictions were not correct, just list tools you've learned in class that might help you figure that out

Think ahead

Do you think a different classifier might work better or do you think this data cannot be predicted any better than this?

- check the type of errors
- are the errors random or are some errors more common than others?

7. Assignment 7: Decision Trees

7.1. Quick Facts

- [accept the assignment](#)
- Recommended completion:
- Due: 2021-10-27 11:59pm

7.2. Related notes

- [2021-10-18](#)
- [2021-10-20](#)
- [2021-10-22](#)

7.3. Assessment

| task | skill |
|--|------------------------------|
| fit a decision tree | classification (2) |
| apply a decision tree to get predictions | classification (2) |
| interpret the model assumed by a decision tree | classification (2) |
| use multiple metrics evaluate performance | evaluate (2) |
| interpret how decisions (test/train size, model parameters) impact model performance | evaluate (2) |
| interpret the classifier performance in the context of the dataset | process (2) |
| analyze the impact of model parameters on model performance | process (2) |
| use loops and lists effectively | python (2) |
| use EDA techniques to examine the experimental results | summarize (2), visualize (2) |
| create a dataset by combining data from multiple sources | construct (2) |

Table 7.1 fit a decision tree

7.4. Instructions

Choose a datasets that is well suited for classification and that has only numerical features.

💡 Tip

A file can be a "comma separated file" and read in with `pd.read_csv` even if the file name does not end in ".csv". The part after the '.' in a file name is called the file *extension* and its a sort of metadata built into a file. CSV is a specification for how to write data to a file, or a file *format*. It's best practice to make the file extension match the file format, but it's very much not required. Especially the older files on the UCI repository, the extension is something else (eg dat, or data, or names), but the actual contents of the files are comma separated and compatible with `read_csv`

Practice using decision trees and exploring how classification works, and what evaluations mean in the following exercises.

💡 Hint

The Wisconsin Breast Cancer data from UCI is a good option as is the wine data set, which has the red & white wines separated, so you can earn prepare with this. You could also use the NSA data and try to predict who will makes the NBA 75 based on their game stats, this would require some manipulation and so would be a way to earn construct.

ℹ️ Note

If you want to use a dataset with nonnumerical features you will have to convert the categorical features to one hot encoding or drop them if there are enough continuous values in addition to the categorical.

7.4.1. Part 1: DT Basics

1. Include a basic description of the data(what the features are)
2. Write your own description of what the classification task is and why a decision tree is a reasonable model to try for this data.
3. Fit a decision tree with the default parameters on 50% of the data
4. Test it on 50% held out data and generate a classification report
5. Inspect the model to answer:
 - Does this model make sense?
 - Are there any leaves that are very small?
 - Is this an interpretable number of levels?
6. Repeat the split, train, and test steps 5 times.
 - Is the performance consistent enough you trust it?
7. Interpret the model and its performance in terms of the application. Some questions you might want to answer in order to do this include:
 - do you think this model is good enough to use for real?
 - is this a model you would trust?
 - do you think that a more complex model should be used?
 - do you think that maybe this task cannot be done with machine learning?

7.4.2. Part 2: Exploring Evaluation

Do an experiment to compare test set size vs performance:

1. Train decision tree with max depth 2 less than the depth it found above on 10%, 30%, ..., 90% of the data. Save the results of both test accuracy and training 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?

💡 Hint

The most important thing about the max depth here is that it's the same across all of the models. If you get an error, try making it smaller.

💡 Hint

use a loop for this part, possibly also a function

💡 Tip

The summary statistics and visualization we used before are useful for helping to investigate the performance of our model. We can try fitting a model with different settings to create a new "dataset" for our experiments. The same skills apply.

7.4.3. Part 3: DT parameters

Experiment with DT Parameters:

1. Choose one parameter to change in the training that you think might improve the model and say why, then train a second decision tree
2. Check the performance of the new decision tree with at least two performance metrics
3. Did changing the parameter do what you expected?
4. Choose a second parameter to change in the training that you think might improve the model and say why, then train a third decision tree
5. Validate your third decision tree with at least two performance metrics.
6. Did changing the parameter do what you expected?

Thinking Ahead

Repeat your experiment from Part 2 with cross validation and plot with error bars.

- What is the tradeoff to be made in choosing a test/train size?
- What is the best test/train size for this dataset?

Repeat the experiment in part 2 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

8. Assignment 8: Regression

8.1. Quick Facts

- [accept the assignment](#)
- Due: 2020-11-03 11:59pm

8.2. Related notes

- [2021-10-25](#)
- [2021-10-29](#)

8.3. Assessment

| task | skill |
|--|------------------------------|
| fit a linear regression model | regression (2) |
| evaluate fit of linear regression | evaluate (2) |
| use multiple metrics evaluate performance | evaluate (2) |
| interpret how decisions (test/train size, model parameters) impact model performance | evaluate (2) |
| interpret the model performance in the context of the dataset | process (2) |
| analyze the impact of model parameters on model performance | process (2) |
| use loops and lists effectively | python (2) |
| use EDA techniques to examine the experimental results | summarize (2), visualize (2) |
| create a dataset by combining data from multiple sources | construct (2) |

8.4. Instructions

Find a dataset suitable for regression. We recommend a dataset from the UCI repository. Complete the following in a single notebook.

8.4.1. Linear Regression Basics

Fit a linear regression model, measure the fit with two metrics, and make a plot that helps visualize the result.

1. Include a basic description of the data(what the features are)
 2. Write your own description of what the regression task is and why a linear model is a reasonable model to try for this data.
 3. Fit a linear model with 75% training data
 4. Test it on 25% held out test data and measure the fit with two metrics and one plot
 5. Inspect the model to answer:
 - Does this model make sense?
 - What to the coefficients tell you?
 - What to the residuals tell you?
 6. Repeat the split, train, and test steps 5 times.
 - Is the performance consistent enough you trust it?
 7. Interpret the model and its performance in terms of the application. Some questions you might want to answer in order to do this include:
 - do you think this model is good enough to use for real?
 - is this a model you would trust?
 - do you think that a more complex model should be used?
 - do you think that maybe this task cannot be done with machine learning?
1. Try fitting the model only on one feature. Justify your choice of feature based on the results above. Plot this result.

8.4.2. Part 2: Exploring Evaluation

Note

If you have the relevant level 2 achievements (evaluation, summarize, visualize) you can skip this part, but it might still be interesting.

Do an experiment to compare test set size vs performance:

1. Re-fit your regression model using 10%, 30%, ..., 90% of the data for training. Save the results of both test and train r2 and MSE for each size training data in a DataFrame with columns ['train_pct','n_train_samples','n_test_samples','train_r2','test_r2','train_mse','test_mse']
2. Plot the metrics vs training percentage in a line graph.
3. Interpret these results. How does training vs test size impact the model?

Thinking Ahead

1. Try these experiments with a different type of regression.
2. How do your evaluation experiment results compare in regression vs classification?

Tip

The summary statistics and visualization we used before are useful for helping to investigate the performance of our model. We can try fitting a model with different settings to create a new "dataset" for our experiments. The same skills apply.

9. Assignment 9: Clustering

9.1. Quick Facts

- [accept the assignment](#)
- Due: 2020-11-10 11:59pm

9.2. Related notes

- [2021-11-01](#)
- [2021-11-03](#)
- [2021-11-05](#)

9.3. Assessment

| task | skill |
|--|------------------------------|
| apply and interpret kmeans clustering | clustering (2) |
| use multiple metrics evaluate performance | evaluate (2) |
| interpret how decisions impact model performance | evaluate (2) |
| interpret the classifier performance in the context of the dataset | process (2) |
| analyze the impact of model parameters on model performance | process (2) |
| use EDA techniques to interpret the experimental results | summarize (2), visualize (2) |

9.4. Instructions

Use the same dataset you used for assignment 7, unless there was a problem, or pick one of the recommended ones for that assignment if you did not complete assignment 7.

1. Describe what question you'd be asking in applying clustering to this dataset.
2. Apply Kmeans using the known, correct number of clusters, $\backslash(K)$.
3. Evaluate how well clustering worked on the data:
 - using a true clustering metric
 - using visual inspection
 - using a clustering metric that uses the ground truth labels
4. Include a discussion of your results that addresses the following:
 - describes what the clustering means
 - what the metrics show
 - Does this clustering work better or worse than expected based on the classification performance (if you didn't complete assignment 7, also apply a classifier)
5. Repeat your analysis using a different number of clusters:
 - can you interpret the new clusters?
 - how do they relate to the original clusters? are they completely different, did one split? did some merge?
 - is there a reasonable explanation for more clusters than there are classes in this dataset?

💡 Hint

a true clustering metric works in an unsupervised way, it does not need to use the true labels, unlike the metrics we used for classification and regression. However, when we have the labels, we can see if the clustering algorithm recovers the same groups we know exist in the data.

See the notes on [Evaluating Clustering](#) for an example of each. There's also a margin note, with a link to sklearn docs with more. If you're curious you can try different metrics from the notes.

💡 Think Ahead

How can clustering be used to ask many different questions? What can you do with clustering results?

10. Assignment 10: Tuning Model Parameters

10.1. Quick Facts

- [accept the assignment](#)
- Due: 2020-11-17 11:59pm

10.2. Related notes

- [2021-11-08](#)
- [2021-11-12](#)

10.3. Asseessment

| task | skill |
|---|---|
| test the model on test data and interpret in context | classification, clustering, OR regression (2) |
| choose and justify appropriate parameters parameter grid for the context | classification, clustering, OR regression (2) AND process (2) |
| evaluate fit of the model while varying the cross validation parameters and interpret | evaluate (2) |
| optimize model parameter (s) and interpret | optimize (2) |
| ask relevant questions of the data domain and interpret results in context | process (2) |
| use EDA techniques to examine the experimental results | summarize (2), visualize (2) |

Table 10.1 Optimize model parameters

Note you can only earn one of classification, clustering, OR regression, but to earn one of those you must both interpret the model and the parameters.

For process you must situate your overall analysis in context (which you should do even if you already have the process achievement; you should *always* do this) and explain how you're picking parameters to evaluate. Your reasons only have to be reasonable, they don't have to be correct. It's okay if what you try doesn't improve the model, but then you have to interpret that.

10.4. Instructions

summary Extend the work you did in assignment 7,8, or 9, by optimizing the model parameters.

1. Choose your dataset, task, and a model. It can be any model we have used in class so far.
2. Fit the model and show some exploration to choose reasonable model parameter values for your parameter grid
3. Use grid search to find the best performing model parameters

4. Examine and interpret the cv results. How do they vary in terms of time? Is the performance meaningfully different or just a little?
 5. Try varying the cross validation parameters. Does this change your conclusions?

💡 Tip

this is best for regression or classification, but if you use clustering use the `scoring` parameter to pass better metrics than the default of the score method.

💡 Hint

Assignment 11 will be to optimize two models and then compare two models on the same task

🕒 Thinking Ahead

What other tradeoffs might you want to make in choosing a model? How could you present these results using your EDA skills?

11. Assignment 11: Model Comparison

11.1. Quick Facts

[accept the assignment](#)

Due: 2020-11-24 11:59pm

11.2. Related notes

- [2021-11-15](#)
- [2021-11-17](#)
- 2021-11-19

11.3. Assessment

| task | skill |
|---|---|
| determine the best model for a given dataset | compare (2) |
| test the model on test data and interpret in context | classification, clustering, OR regression (2) |
| choose and justify appropriate parameters parameter grid for the context | classification, clustering, OR regression (2) AND process (2) |
| evaluate fit of the model while varying the cross validation parameters and interpret | evaluate (2) |
| interpret the classifier performance in the context of the dataset | process (2) |
| analyze the impact of model parameters on model performance | process (2) |
| use EDA techniques to interpret the experimental results | summarize (2), visualize (2) |

Table 11.1 compare models and make a recommendation

Choose a dataset, it can be appropriate for classification, regression, or clustering. Fit at least two models for the same task and choose the appropriate metrics to compare the fit. Decide which model you would recommend based on a realistic setting for that dataset and include evidence justifying that choice. Summarize your findings with plots and tables as appropriate.

This will be easiest if you use a dataset you've used on for one of the previous assignments or choose another.

💡 Think Ahead

How would this decision making compare for a more complex model or in more realistic setting.

12. Assignment 12: Fake News

12.1. Quick Facts

- [accept the assignment](#)
- First feedback: 2021-12-02 6:00pm Final due date: **2021-12-08 11:59pm**

Submit by the first feedback deadline to get feedback and then continue revising your work. After the final due date that feedback will be final.

12.2. Evaluation

| task | skill |
|---|------------------------------|
| transform text data to a format compatible with ML | representation (2) |
| plan to solve a real world problem using the tools from class | workflow (2) |
| use EDA techniques to interpret the experimental results | summarize (2), visualize (2) |
| determine the best model for this task | compare (2) |
| determine the best parameter settings for this task | optimize (2) |

Table 12.1 use text to predict fake from real news

12.3. Instructions

Use the dataset in the assignment template repo to answer the following questions. The data includes variables:

- 'text': contents of an article
- 'label': whether it is real or fake news
- 'title': title of the article

1. Is the text or the title of an article more predictive of whether it is real or fake?
2. Are titles of real or fake news more similar to one another?

Consider what difference you can have in how you represent the data and how that might impact your model performance in order. Use summary statistics and visualizations appropriately in order to explain your results.

💡 Hint

The data set contains a large number of articles (takes a long time to train), you can downsample this to something like a 1,000 articles or so in order to speed up training and evaluation (hint: use shuffle).

Portfolio Dates and Key Facts

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, you can also earn level 1 or 2. 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.

❗ Important

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.

In 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](#)

Current: Check 3

The third submission will be graded on the following criteria* and due on December 20 :

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

- it can also be graded against the level 1 or 2 criteria from the syllabus/Data Science Achievements

Submission Checklist:

- update your gh action or precommit hook
- complete your KWL chart
- add notebooks or markdown files for your work

Submission Introductions

Your portfolio will be assessed both on your demonstration of skills through your chapters that are inspired by the prompts and that your summary demonstrates that you *know* you learned the skills.

Each portfolio submission, you will edit the corresponding [submission_x_intro.md](#) file. This is where you write your summary and reflect on your learning. This reflection does not need to be long, it shouldn't take a very long time. It's okay for it to be brief, mostly bullet points.

KWL Chart

One part of your introduction is a **KWL** or Know, Want to know, Learned, chart.

In your file it will look like this:(but longer)

```

``{list-table} Portfolio 1 KWL Chart
:header-rows: 1
:name: kw1

* - Skill
  - Know
  - Want to Know
  - Learned
* - python
  - basics
  - more efficient types
  -
* - access
  - that datasets are collated on kaggle
  - how to load data for analysis
  - how to load data from 3 different types and compare them
* - ...
  - ...
  - ...
  - ...
```

```

and when you build your portfolio it will render like this:

Skill	Know	Want to Know	Learned
python	basics	more efficient patterns	pep8 patterns and common conventions,
access	that datasets are collated on kaggle	how to load data for analysis	how to load data from 3 different types and compare them
...	...	...	...

Table 1 Portfolio 1 KWL Chart

## Overview

In the overview, you summarize the contents of your portfolio. Think of it as the introduction to the overall submission. Your goal is to help us know what to expect when grading your portfolio and to know that you know what you've learned.

Writing this out will help give you a space to confirm that you're on track by checking your own work against the [Achievement Definitions](#) table. If your work does not earn level 3 achievements, your summary will help us identify if you are on track or if you're not on track. If you're not on track it will help us distinguish between if it's because of a misunderstanding in the expectations or the material.

This summary helps us help you achieve your own goals and lets us help you accordingly. We want you succeed in the course and the best way to do that is to check in frequently.

### 💡 Learning Tip

This reflection process also help you learn better, in addition to being an accountability check. What you draw your attention to gets reinforced in your memory, so reflecting on what you've learned helps you learn better.

## Formatting Tips

### ⚠️ Warning

This is all based on you having accepted the portfolio assignment on github and having a cloned copy of the template. If you are not enrolled or the initial assignment has not been issued, you can view [the template on GitHub](#)

Your portfolio is a [jupyter book](#). This means a few things:

- it uses [myst markdown](#)
- it will run and compile Jupyter notebooks

This page will cover a few basic tips.

## Managing Files and version

You can either convert your ipynb files to earier to read locally or on GitHub.

The GitHub version means installing less locally, but means that after you push changes, you'll need to pull the changes that GitHub makes.

### To manage with a precommit hook jupytext conversion

change your [.pre-commit-config.yaml](#) file to match the following:

```

repos:
- repo: https://github.com/mwouts/jupytext
 rev: v1.10.0 # CURRENT_TAG/COMMIT_HASH
 hooks:
 - id: jupytext
 args: [--from_ipynb, --to_myst]

```

Run Precommit over all the files to actually apply that script to your repo.

```

pre-commit install
pre-commit run --all-files

```

If you do [git status](#) now, you should have a [.md](#) file for each [ipynb](#) file that was in your repository, now add and commit those.

Now, each time you commit, it will run jupytext first.

### To manage with a gh action jupytext conversion

create a file at [.github/workflows/jupytext.yml](#) and paste the following:

```

name: jupytext

Only run this when the master branch changes
on:
 push:
 branches:
 - main
 # If your git repository has the Jupyter Book within some-subfolder next to
 # unrelated files, you can make this run only if a file within that specific
 # folder has been modified.
 #
 # paths:
 # - some-subfolder/**

This job installs dependencies, build the book, and pushes it to `gh-pages`
jobs:
 jupytext:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v2

 # Install dependencies
 - name: Set up Python 3.7
 uses: actions/setup-python@v1
 with:
 python-version: 3.7

 - name: Install dependencies
 run:
 pip install jupytext
 - name: convert
 run:
 jupytext *.ipynb --to myst
 jupytext *.ipynb --to myst
 - uses: EndBug/add-and-commit@v4 # You can change this to use a specific version
 with:
 # 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 first to set it up
 # Default: '.'
 cwd: '.'

 # Whether to use the --force option on `git add`, in order to bypass eventual gitignores
 # Default: false
 force: true

 # Whether to use the --signoff option on `git commit`
 # Default: false
 signoff: true

 # The message for the commit
 # Default: 'Commit from GitHub Actions'
 message: 'convert notebooks to md'

 # Name of the branch to use, if different from the one that triggered the workflow
 # Default: the branch that triggered the workflow (from GITHUB_REF)
 ref: 'main'

 # Name of the tag to add to the new commit (see the paragraph below for more info)
 # Default: ''
 tag: "v1.0.0"

 env:
 # This is necessary in order to push a commit to the repo
 GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }} # Leave this line unchanged

```

## Organization

The summary of for the `part` or whole submission, should match the skills to the chapters. Which prompt you're addressing is not important, the prompts are a *starting point* not the end goal of your portfolio.

## Data Files

Also note that for your portfolio to build, you will have to:

- include the data files in the repository and use a relative path OR
- load via url

using a full local path(eg that starts with `///file:`) **will not work** and will render your portfolio unreadable.

## Structure of plain markdown

Use a heading like this:

```

Heading of page
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](#)

```
```{note}
Here is a note!
````
```

```
```{warning}
Here is a warning!
````
```

```
```{tip}
Here is a tip!
````
```

```
```{margin}
Here is a margin note!
````
```

For a complete list of options, see [the sphinx-book-theme documentation](#).

## Links

Markdown syntax for links

```
[text to show](path/or/url)
```

## Configurations

Things like the menus and links at the top are controlled as [settings](#), in `_config.yml`. The following are some things that you might change in your configuration file.

### Show errors and continue

To show errors and continue running the rest, add the following to your configuration file:

```
Execution settings
execute:
 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, but these are ideas for the structure.

### 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.

### Several shorter reflections/analyses

You could also submit a few shorter pieces that in total cover all of the skills. Some example formats:

#### Tutorial

Write a notebook that explains a concept with examples in a real dataset and with visuals or a toy dataset (minimal number of columns rows)

#### Cheatsheet

Make a detailed reference with code outputs on a topic or a few topics.

#### Blog post

Write a blog post styled Notebook that compares or analyzes something, for example:

- how do different ways of loading data compare
- describe best practices you've learned and show why they're good with examples

#### Correction & Reflection

If you had trouble with an assignment so far, you can revise what you submitted and resubmit it, with reflections and explanation of what you were confused about, what you tried initially, how you eventually figured it out, and explains the correct answer. Then go a little deeper in exploring the topic in that context to also earn level 3.

#### Practice Problems and Solutions

Based on the level 3 rubric descriptions, write practice problems that build off of the lecture notes. Include solutions and descriptions for each. These can be open ended or multiple choice questions with plausible distractors. A plausible distractor is an incorrect answer that represents a way that you think someone could misunderstand.

For example if the question is  $37 + 15 = ?$ , MCQ with plausible distractors might be:

- 52 (correct)

- 412 (didn't carry the one, correctly:  $7+5 = 12$ ,  $3+1 = 4$ )
- 42 (dropped the one  $7+5 = 12$ , ones place is 2,  $3+1 = 4$ )
- 43 (carried one into wrong column,  $7 + 5 = 12$ ,  $1+2 = 3$ ,  $3+1 = 3$ )

## Check 2 Ideas

For Check 2, all of the prompts from check 1 apply, plus the following additional prompts, since there are new skills.

If you have other ideas, you can also ask and those are likely possible.

### Level 1 Achievement Catchup

To make up level 1 achievements, include a detailed introduction file to your portfolio and one of the following (per skill):

- minor extensions to what we did in class
- answers to problems from the notes
- additional glossary terms
- psuedocode for one of the other prompts

### Extend Assignment 7, 8, or 9

Assignments 7-9 help you think through what machine learning tasks are. Extend those ideas by adding additional experiments based on your own questions or the questions in your feedback.

### Build a data set for Prediction

Build a dataset that works for prediction (classification, regression, or clustering) from other sources.

### Learn a new model

Repeat what you did in 7, 8, or 9, with a different model.

### Create datasets that fail

Create datasets that violate assumptions of a model we have learned. The [sklearn data generators](#) are a good place to start.

### Process level 3

Process level 3 is a little different than most of the others. You may be able to work it into an analysis notebook, but likely, you'll need to do one of the following.

#### Data Science Pipeline Comparisons

Find two different sources that describe the data science pipeline or lifecycle. Write a blog style post that discusses their differences and hypothesizes about why they may be different? Are they for different audiences? Is one domain specific? How do they emphasize different modeling tasks? Include a recommendation for when you think each one is better

#### Write a short story

Write a short story that explains the concepts of data science to demonstrate your understanding of process.

#### Media Review

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> 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 4 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.

This could be a concept map, a table that shows how you've traced how something works or any other sort of conceptual tool that helps convey your understanding.

### 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 Beibhinn (beibhinn)
- via Prismia.chat during class
- by creating an [issue](#)

## Syllabus and Grading FAQ

### How much does assignment x, class participation, or a portfolio check weigh in my grade?

There is no specific weight for any activities, because your grade is based on earning achievements for the skills listed in the [skills rubric](#).

However, if you do not submit (or earn no achievements from) assignments or portfolios, the maximum grade you can earn is a C. If you do not submit (or earn no achievements from) your portfolio, the maximum grade you can earn is a B.

### Can I submit this assignment late if ... ?

Late assignments are not accepted, however, your grade is based on the skills, not the assignments. All skills are assessed in at least two [assignments](#), so missing any one will not hurt your grade. If you need an accommodation because you cannot submit 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 push new changes there.

After you run

```
git pull
```

You'll probably have to [resolve a merge conflict](#)

### The content I added to my portfolio isn't in the pdf

There was an error in the original `_toc.yml` file, change yours to match the following:

```
format: jb-book
root: intro
parts:
- caption: About
 chapters:
 - file: about/index
 - file: about/grading
- caption: Check 1
chapters:
- file: submission_1_intro
```

uncomment the later lines and add any new files you add.

### My command line says I cannot use a password

GitHub has [strong rules](#) about authentication. You need to use SSH with a public/private key; HTTPS with a [Personal Access Token](#) 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 alternative in the portfolio 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 available 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

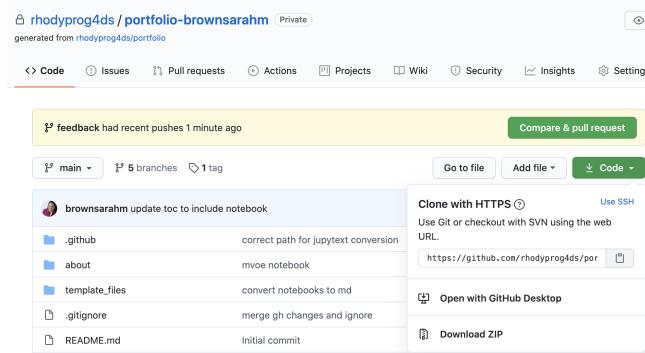
If there's an error your notebook it can't complete running. You can allow it to run if the error is on purpose by changing settings as mentioned on the formatting page.

## 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 instructions 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 shown



Next open a terminal or GitBash and type the following.

```
git clone
```

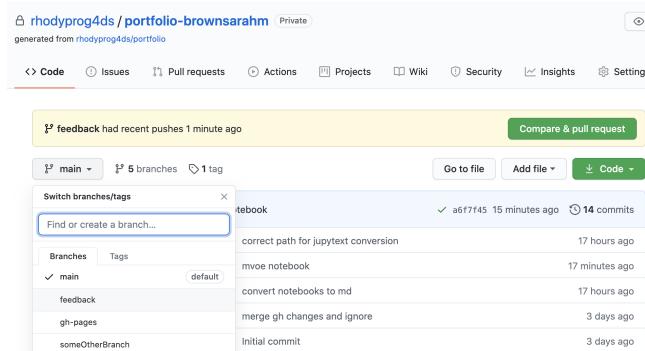
then past your url that you copied. It will look something like this, but the last part will be the current assignment repo and your username.

```
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.



Now view the list of all of the commits to this branch, by clicking on the clock icon with a number of commits

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.

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 822cfe51a70d356d448bcaede5b15282838a5028
```

If it works, your terminal will say something like

```
HEAD is now at 822cfe5 Setting up GitHub Classroom Feedback
```

but the number on yours will be different.

Now your local copy of the `feedback` branch is reverted back as if you had not merged the pull request and what's left to do is to push those changes to GitHub. By default, GitHub won't let you push changes unless you have all of the changes that have been made on their side, so we have to tell Git to force GitHub to do this.

Since we're about to do something with forcing, we should first check that we're doing the right thing.

```
git status
```

and it should show something like

```
On branch feedback
Your branch is behind 'origin/feedback' by 12 commits, and can be fast-forwarded.
(use "git pull" to update your local branch)
```

Your number of commits will probably be different but the important things to see here is that it says `On branch feedback` so that you know you're not deleting the `main` copy of your work and `Your branch is behind origin/feedback` to know that reverting worked.

Now to make GitHub match your reverted local copy.

```
git push origin -f
```

and you'll get something like this to know that it worked

```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
 + f301d90..822cfe5 feedback -> feedback (forced update)
```

Again, the numbers will be different and it will be your url, not mine.

Now back on GitHub, in your browser, click on the code tab. It should look something like this now. Notice that it says, "This branch is 11 commits behind main" your number will be different but it should be 1 less than the number you had when you checked `git status`. This is because we reverted the changes you made to main (11 for me) and the 1 commit for merging main into feedback. Also the last commit (at the top, should say "Setting up GitHub Classroom Feedback").

This branch is 11 commits behind main.

brownsarahm Setting up GitHub Classroom Feedback 822cfe5 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

Now, you need to recreate your Pull Request, click where it says pull request.

This branch is 11 commits behind main.

brownsarahm Setting up GitHub Classroom Feedback 822cfe5 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.

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

Show other branches

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 Request" interface.

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.

base: feedback ▾ compare: main ▾ ✓ Able to merge. These branches can be automatically merged.

Feedback

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

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/fall120instructors` for help.

## Code Errors

### Key Error

If you get a key error for a pandas operation, it means that the column name as you typed it is not in the DataFrame. Check the spelling, leading or trailing whitespace can be especially troubling.

## <bound method

You're probably missing `()` on a method, so Python returned the method itself as an object instead of calling it and returning the output.

# Glossary

### Ram Token Opportunity

Contribute glossary items and links for further reading using the suggest an edit button behind the GitHub menu at the top of the page.

### **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.

### **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

### **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 iterate over some data structure and typically uses the `for` keyword

### **iterable**

any object in python that can return its members one at a time. The most common example is a list, but there are others.

### **kernel**

in the jupyter environment, `the kernel` is a language specific computational engine

### **lambda**

they keyword used to define an anonymous function; lambda functions are defined with a compact syntax `<name> = lambda <parameters>: <body>`

### **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 called tracing back through the stack

### training accuracy

percentage of predictions that the model predict correctly, based on the training data

### 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 long columns will give length 2 (one per column). Setting up our toy dataset to not be a square was important so that we can use it to check which way is which.

```
df.sum(axis=0)
```

```
A 7
B 8
dtype: int64
```

```
df.sum(axis=1)
```

```
0 1
1 9
2 5
dtype: int64
```

```
{ df.apply(sum, axis=0)
```

```
A 7
B 8
dtype: int64
```

```
{ df.apply(sum, axis=1)
```

```
0 1
1 9
2 5
dtype: int64
```

## Indexing

```
{ df['A'][1]
```

```
5
```

```
{ df.iloc[0][1]
```

```
0
```

## Data Sources

This page is a semi-curated source of datasets for use in assignments. The different sections have datasets that are good for different assignments.

### Best for loading directly into a notebook

- [Tidy Tuesday](#) inside the folder for each year there is a README file with list of the datasets. These are .csv files
- [Json Datasets](#)
- [National Center for Education Statistics Digest 2019](#) These data tables are available for download as excel and visible on the page.
- Lots of wikipedia pages have tables in them.

### 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](#)

### Datasets with time

- [Superbowl commercials](#)

### Databases

- [SQLite Databases](#)

If you have others please share by creating a pull request or issue on this repo (from the GitHub logo at the top right, [suggest edit](#)).

## General Tips and Resources

This section is for materials that are not specific to this course, but are likely useful. They are not generally required readings or installs, but are options or advice I provide frequently.

### on email

- [how to e-mail professors](#)

## How to Study in this class

This is a programming intensive course and it's about data science. This course is designed to help you learn how to program for data science and in the process build general skills in both programming and using data to understand the world. Learning two things at once is more complex. In this page, I break down how I expect learning to work for this class.

Remember the goal is to avoid this:



## Why this way?

Learning to program requires iterative practice. It does not require memorizing all of the specific commands, but instead learning the basic patterns. Using reference materials frequently is a built in part of programming, most languages have built in help as a part of the language for this reason. This course is designed to have you not only learn the material, but also to build skill in learning to program. Following these guidelines will help you build habits to not only be successful in this class, but also in future programming.

A new book that might be of interest if you find programming classes hard is [The Programmers Brain](#). As of 2021-09-07, it is available for free by clicking on chapters at that linked table of contents section.

### ⓘ Where are your help tools?

In Python and Jupyter notebooks, what help tools do you have?

## Learning in class

### ⓘ Important

My goal is to use class time so that you can be successful with *minimal frustration* while working outside of class time.

Programming requires both practical skills and abstract concepts. During class time, we will cover the practical aspects and introduce the basic concepts. You will get to see the basic practical details and real examples of debugging during class sessions. Learning to debug something you've never encountered before and setting up your programming environment, for example, are *high frustration* activities, when you're learning, because you don't know what you don't know. On the other 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.

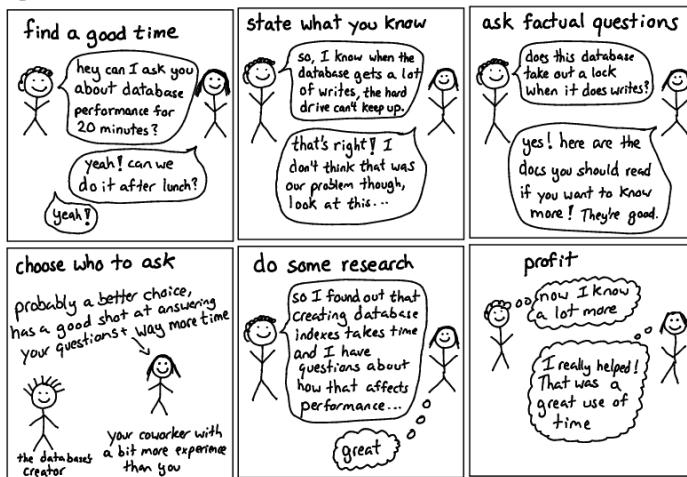
## Portfolios

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

## 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.

### Note

A fun version of this is [rubber duck debugging](#)

## Understanding Errors

Error messages from the compiler are not always straight forward.

The [TraceBack](#) can be a really long list of errors that seem like they are not even from your code. It will trace back to all of the places that the error occurred. It is often about how you called the functions from a library, but the compiler cannot tell that.

To understand what the traceback is, how to read one, and common examples, see [this post on Real Python](#).

One thing to try, is [friendly traceback](#) a python package that is designed to make that error message text more clear and help you figure out what to do next.

### Ram Token Opportunity

If you try out friendly traceback and find it helpful, add a testimonial here, using

```
```{epigraph}
```

Terminals and Environments

Why all this work?

Managing environments is **one of the hardest parts of programming** so, as instructors, we often design our courses around not having to do it. In this class, however, I'm choosing to take the risk and help you all through beginning to manage your own environments.

These issues will be the most painful in the course, I promise.

I think it's worth this type of pain though, because all 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.

Note

We know that we don't currently teach a lot of this in our department, so in Spring 22 I'm teaching a brand new course on Computer Systems, that will help you understand the underlying concepts that make all of this stuff make sense, instead of just following recipes and debugging here and there.

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](#)).

If, for example, you come to me in week 5 and have never got an environment working and you're trying for the first time, your grade will be hurt because you will be very far behind at that point. Ask for help early and often.

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[“ptrans] if you would like.

If you go to the main page of the [organization](#) you can search by your username (or the first few characters of it) and see only your repositories.

⚠ Warning

Don't try to work on a repository that does not end in your username; those are the template repositories for the course and you don't have edit permission on them.

Letters to Future students

This section is a place for students enrolled in Fall 2021 to write letters to future students taking this class with Professor Brown. The websites for future sections will link back here for them to read.

Contributed Notes

Portfolio Check Learning Curve Coming from a business background with not the best knowledge of statistics, data science came with quite a large learning curve. Just to get up to the level at which other students were performing, I had to put in extra hours week in and week out which is why it is smart to devote 6-10 hours each week as far as working through weekly assignments at the minimum. This may not include the time needed to work on portfolio checks as that is a whole separate time requirement of its own. Portfolio checks are also the reason I am writing this today for future students to read. I personally had a hard time with portfolio checks as they are vastly different from any other assignment. They are very open-ended unlike the assignments and sound much easier than they actually are. The most challenging part for me was combining the different learning achievements under one open-ended assignment. This took in-depth thought and preparation; unfortunately, there were times where I had quite a few hours of work put into a portfolio check only to have to restart because I got off track from the objective or in research. With that being said, take the time plan ahead, set goals, and give yourself time to do hours of research on your own if need be because many of the level three achievements call for material that is not learned in class (which is the whole point, to show you can devote X amount of time, research on your own, and apply it to similar methods we have worked on in class). Luckily you all have multiple attempts to obtain each achievement so there is a somewhat forgiving trial/error system, but do not let the seemingly long periods in between checks distract you because the semester moves quickly. My second recommendation is to thoroughly read the outline of what a portfolio check is and either ask questions in class or during office hours because as I stated, it is very different from the weekly assignments.

To contribute

Via GitHub directly:

1. Use the edit button above to add a note to this file following the example that's commented out
2. create a pull request