DATA SCIENCE IN PYTHON

PART 1:

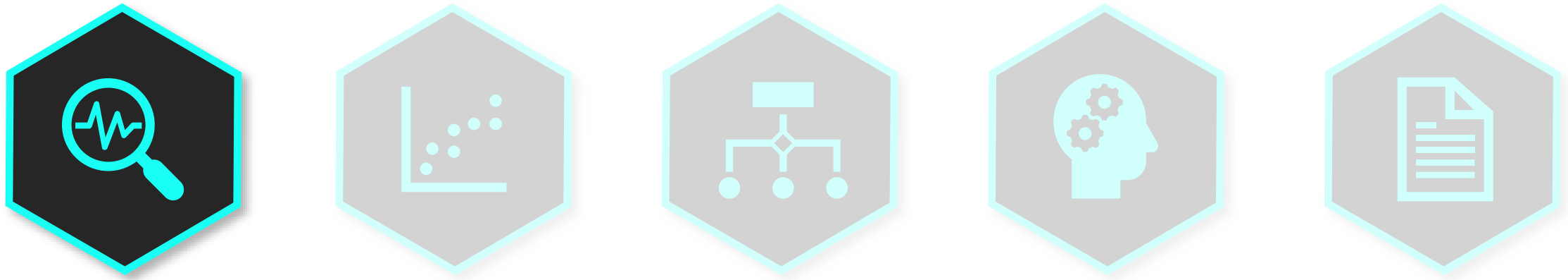
Data Prep & Exploratory Data Analysis

★★★★★ *With Expert Data Science Instructor Alice Zhao*

MAVEN ANALYTICS

ABOUT THIS SERIES

This is **Part 1** of a **5-Part series** designed to take you through several applications of data science using Python, including **data prep & EDA**, **regression**, **classification**, **unsupervised learning** & **NLP**



PART 1
Data Prep & EDA

PART 2
Regression

PART 3
Classification

PART 4
Unsupervised Learning

PART 5
Natural Language Processing

COURSE OUTLINE

1	**Intro to Data Science**	*Introduce the field of data science, review essential skills, and introduce each phase of the data science workflow*
2	**Scoping a Project**	*Review the process of scoping a data science project, including brainstorming problems and solutions, choosing techniques, and setting clear goals*
3	**Installing Jupyter Notebook**	*Install Anaconda and introduce Jupyter Notebook, the user-friendly coding environment we'll use for writing Python code*
4	**Gathering Data**	*Read flat files into a Pandas DataFrame in Python, and review common data sources & formats, including Excel spreadsheets and SQL databases*
5	**Cleaning Data**	*Identify and convert data types, find and fix common data issues like missing values, duplicates, and outliers, and create new columns for analysis*

COURSE OUTLINE

6	**Exploratory Data Analysis**	*Explore datasets to discover insights by sorting, filtering, and grouping data, then visualize it using common chart types like scatterplots & histograms*
7	**MID-COURSE PROJECT**	*Put your skills to the test by cleaning, exploring and visualizing data from a brand-new data set containing Rotten Tomatoes movie ratings*
8	**Preparing for Modeling**	*Structure your data so that it's ready for machine learning models by creating a numeric, non-null table and engineering new features*
9	**FINAL COURSE PROJECT**	*Apply all the skills learned throughout the course by gathering, cleaning, exploring, and preparing multiple data sets for Maven Music*

INTRODUCING THE COURSE PROJECT

THE SITUATION

You've just been hired as a Jr. Data Scientist for **Maven Music**, a streaming service that's been losing more customers than usual the past few months and would like to use data science to figure out how to reduce customer churn

THE ASSIGNMENT

You'll have access to data on Maven Music's customers, including subscription details and music listening history

Your task is to **gather, clean, and explore the data** to provide insights about the recent customer churn issues, then **prepare it for modeling** in the future

THE OBJECTIVES

1. **Scope** the data science project
2. **Gather** the data in Python
3. **Clean** the data
4. **Explore** & visualize the data
5. **Prepare** the data for modeling

MAVEN MUSIC

SINCE 1996

SETTING EXPECTATIONS

This course covers **data gathering**, **cleaning** and **exploratory data analysis**

- *We'll review common techniques for gathering, cleaning and analyzing data with Python, but will not cover more complex data formats or advanced statistical tools*

We will **NOT** be applying machine learning models in this course

- *This course will focus on preparing raw data for deeper analysis and modeling; we will introduce and apply supervised and unsupervised machine learning algorithms in-depth later in this series*

We'll use **Jupyter Notebook** as our primary coding environment

- *Jupyter Notebook is free to use, and the industry standard for conducting data analysis with Python*

You do **NOT** need to be a Python expert to take this course

- *We strongly recommended completing the Maven Analytics Data Analysis with Python and Pandas course before this one, or having some familiarity working with Pandas DataFrames*

INTRO TO DATA SCIENCE

INTRO TO DATA SCIENCE

In this section we'll **introduce the field of data science**, discuss how it compares to other data fields, and walk through each phase of the data science workflow

TOPICS WE'LL COVER:

- What is Data Science?
- Essential Skills
- What is Machine Learning?
- Data Science Workflow

GOALS FOR THIS SECTION:

- Compare roles under the broader data analytics umbrella
- Discuss essential skills of a data scientist
- Compare data science and machine learning
- Introduce supervised and unsupervised learning, and commonly used algorithms
- Review each phase of the data science workflow

WHAT IS DATA SCIENCE?

Data science is about *using data to make smart decisions*

Wait, isn't that **business intelligence** ?

Yes! The differences lie in the **types of problems** you solve, and **tools and techniques** you use to solve them:

What *happened*?

- Descriptive Analytics
- Data Analysis
- Business Intelligence

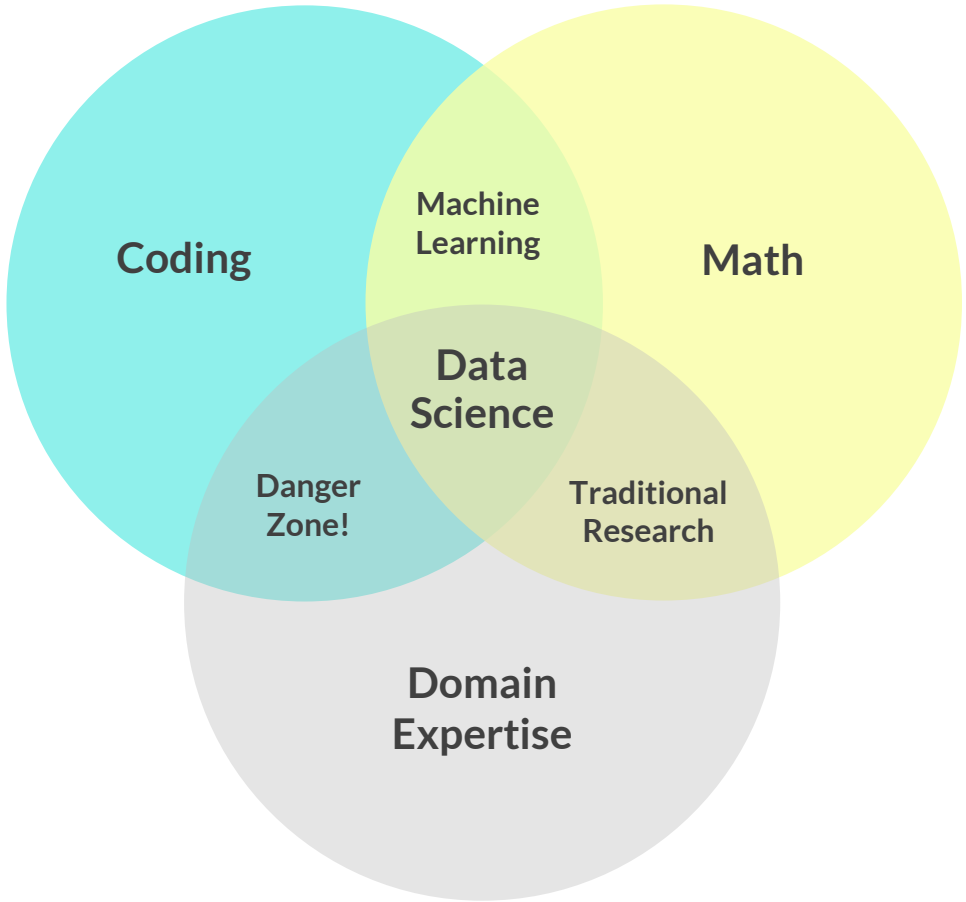
What's *going to* happen?

- Predictive Analytics
- Data Mining
- **Data Science**

DATA SCIENCE SKILL SET

- What is Data Science?
- Essential Skills
- What is Machine Learning?
- Data Science Workflow

Data science requires a blend of **coding**, **math**, and **domain expertise**



The key is in applying these along with soft skills like:

- Communication
- Problem solving
- Curiosity & creativity
- Grit
- Googling prowess

Data scientists & analysts approach problem solving in similar ways, but data scientists will often work with larger, more complex data sets and utilize advanced algorithms

WHAT IS MACHINE LEARNING?

- What is Data Science?
- Essential Skills
- What is Machine Learning?
- Data Science Workflow

Machine learning enables *computers to learn and make decisions from data*

How can a computer learn?

By using **algorithms**, which is a set of instructions for a computer to follow

How does this compare with data science?

Data scientists know how to apply algorithms, meaning they're able to tell a computer how to learn from data

SUPERVISED VS. UNSUPERVISED LEARNING

Machine learning algorithms fall into two broad categories:
supervised learning and **unsupervised learning**

Supervised Learning

Using historical data to predict the future

What will house prices look like for the next 12 months?

How can I flag suspicious emails as spam?

Unsupervised Learning

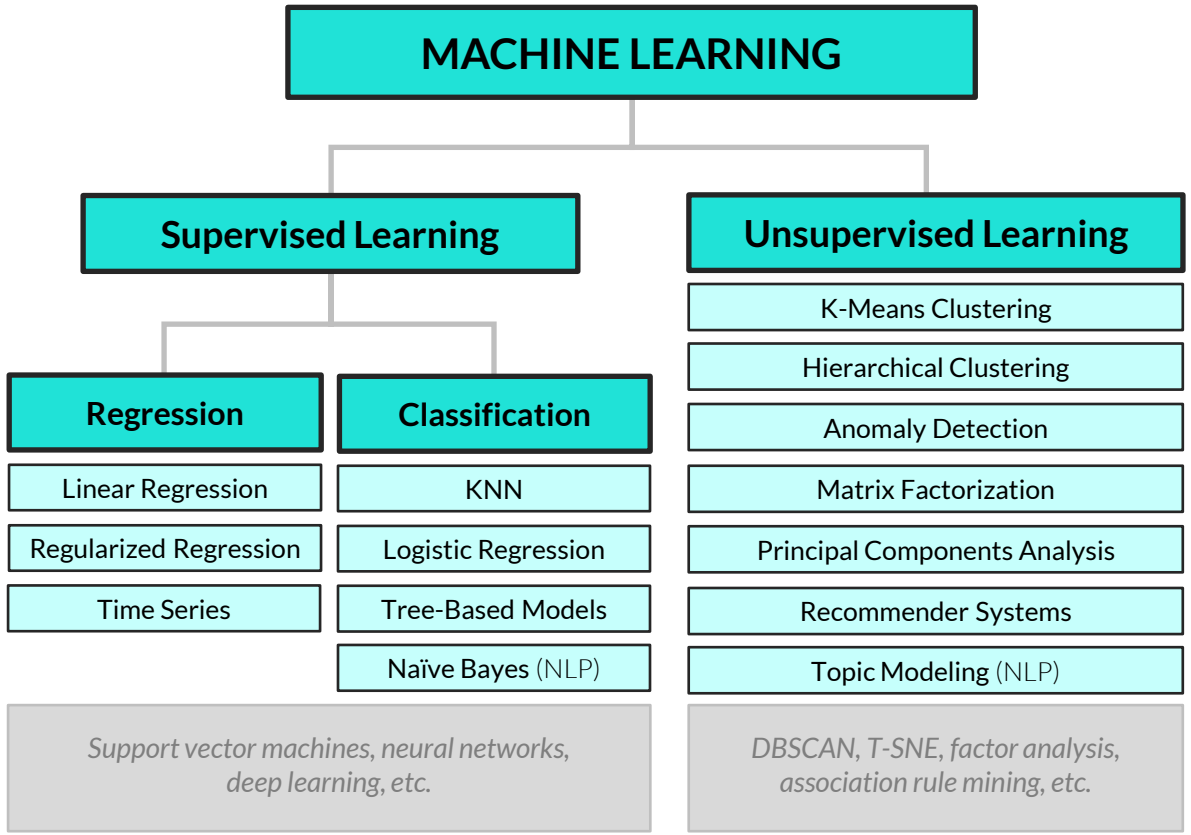
Finding patterns and relationships in data

How can I segment my customers?

Which TV shows should I recommend to each user?

COMMON ALGORITHMS

These are some of the most **common machine learning algorithms** that data scientists use in practice



Another category of machine learning is called **reinforcement learning**, which is more commonly used in robotics and gaming

Other fields like **deep learning** and **natural language processing** utilize both supervised and unsupervised learning techniques

DATA SCIENCE WORKFLOW

- What is Data Science?
- Essential Skills
- What is Machine Learning?
- Data Science Workflow

The **data science workflow** consists of scoping a project, gathering, cleaning and exploring the data, building models, and sharing insights with end users

This is not a linear process! *You'll likely go back to further gather, clean and explore your data*

STEP 1: SCOPING A PROJECT

1 → 2 ⇄ 3 ⇄ 4 ⇄ 5 → 6

Scoping a Project | Gathering Data | Cleaning Data | Exploring Data | Modeling Data | Sharing Insights

Projects don't start with *data*, they start with a **clearly defined scope**:

- Who are your end users or stakeholders?
- What business problems are you trying to help them solve?
- Is this a supervised or unsupervised learning problem? *(do you even need data science?)*
- What data do you need for your analysis?

STEP 2: GATHERING DATA

A project is only as strong as the underlying data, so **gathering the right data** is essential to set a proper foundation for your analysis

Data can come from a variety of sources, including:

- Files *(flat files, spreadsheets, etc.)*
- Databases
- Websites
- APIs

STEP 3: CLEANING DATA

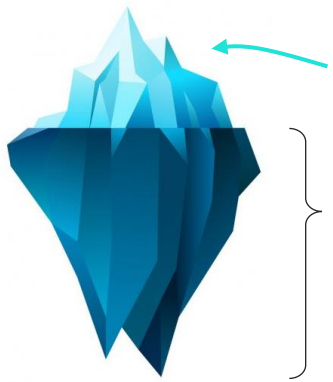
1 → 2 ⇄ 3 ⇄ 4 ⇄ 5 → 6

Scoping a Project | Gathering Data | **Cleaning Data** | Exploring Data | Modeling Data | Sharing Insights

A popular saying within data science is *"garbage in, garbage out"*, which means that **cleaning data** properly is key to producing accurate and reliable results

Data cleaning tasks may include:

- Correcting data types
- Imputing missing data
- Dealing with data inconsistencies
- Reformatting the data



Building models
The flashy part of data science

Cleaning data
Less fun, but very important

(Data scientists estimate that around 50-80% of their time is spent here!)

STEP 4: EXPLORING DATA

1 → 2 ⇄ 3 ⇄ 4 ⇄ 5 → 6

Scoping a Project | Gathering Data | Cleaning Data | **Exploring Data** | Modeling Data | Sharing Insights

Exploratory data analysis (EDA) is all about exploring and understanding the data you're working with before building models

EDA tasks may include:

- Slicing & dicing the data
- Summarizing the data
- Visualizing the data

A good number of the **final insights** that you share will come from the exploratory data analysis phase!

STEP 5: MODELING DATA

Modeling data involves structuring and preparing data for specific modeling techniques, and applying algorithms to make predictions or discover patterns

Data modeling tasks may include:

- Restructuring the data
- Feature engineering *(adding new fields)*
- Applying machine learning algorithms

With fancy new algorithms introduced every year, you may feel the need to learn and apply the latest and greatest techniques

In practice, **simple is best**; businesses & leadership teams appreciate solutions that are easy to understand, interpret and implement

STEP 6: SHARING INSIGHTS

1 → 2 ⇄ 3 ⇄ 4 ⇄ 5 → 6

Scoping a Project | Gathering Data | Cleaning Data | Exploring Data | Modeling Data | **Sharing Insights**

The final step of the workflow involves summarizing your key findings and **sharing insights** with end users or stakeholders:

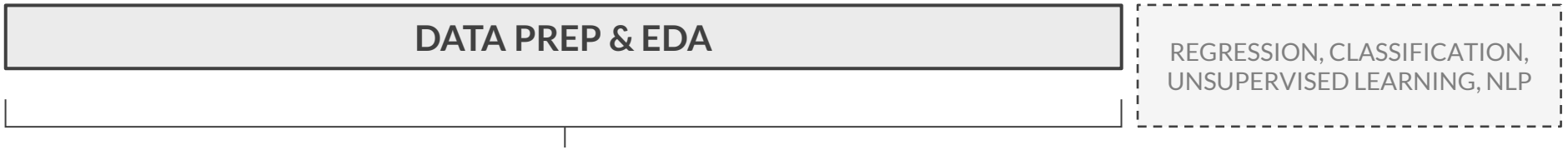
- Reiterate the problem
- Interpret the results of your analysis
- Share recommendations and next steps
- Focus on potential impact, not technical details

Even with all the technical work that's been done, it's important to remember that the focus here is on **non-technical solutions**

NOTE: Another way to share results is to deploy your model, or put it into production

DATA PREP & EDA

- What is Data Science?
- Essential Skills
- What is Machine Learning?
- Data Science Workflow



Data prep and EDA is a critical part of every data science project, and should *always* come first before applying machine learning algorithms

KEY TAKEAWAYS

Data science is about using data to make smart decisions

- *Supervised learning techniques use historical data to predict the future, and unsupervised learning techniques use algorithms to find patterns and relationships*

Data scientists have both **coding** and **math** skills along with **domain expertise**

- *In addition to technical expertise, soft skills like communication, problem-solving, curiosity, creativity, grit, and Googling prowess round out a data scientist's skillset*

The **data science workflow** starts with defining a clear scope

- *Once the project scope is defined, you can move on to gathering and cleaning data, performing exploratory data analysis, preparing data for modeling, applying algorithms, and sharing insights with end users*

Much of a data scientist's time is spent **cleaning & preparing** data for analysis

- *Properly cleaning and preparing your data ensures that your results are accurate and meaningful (garbage in, garbage out!)*

SCOPING A PROJECT

SCOPING A PROJECT

In this section we'll discuss the process of **scoping a data science project**, from understanding your end users to deciding which tools and techniques to deploy

TOPICS WE'LL COVER:

- Project Scoping Steps
- Thinking Like an End User
- Problems & Solutions
- Modeling Techniques
- Data Requirements
- Summarizing the Scope

GOALS FOR THIS SECTION:

- Outline the key steps for defining a clear and effective data science project scope
- Learn how to collaborate with end users to understand the business context, explore potential problems and solutions, and align on goals
- Identify which types of solutions would require supervised vs. unsupervised techniques
- Review common data requirements, including structure, features, sources and scope

SCOPING A PROJECT

Scoping a data science project means clearly defining the goals, techniques, and data sources you plan to use for your analysis

Scoping steps:

1. Think like an end user
2. Brainstorm problems
3. Brainstorm solutions
4. Determine the techniques
5. Identify data requirements
6. Summarize the scope & objectives

Project scoping is one of the **most difficult yet important steps** of the data science workflow

If a project is not properly scoped, a lot of time can be wasted going down a path to create a solution that solves the wrong problem

MAVEN MUSIC
SINCE 1996

We'll be **scoping the course project** in this section together to set you up for success throughout the rest of the course

THINK LIKE AN END USER

An **end user** (also known as a stakeholder) is a person, team, or business that will ultimately benefit from the results of your analysis

When introduced to a new project, start by asking questions that help you:

- **Empathize** with the end user – what do they care about?
- Focus on **impact** – what metrics are important to them?

Data Scientist

Debbie Dayda
Data Science Team

What's the situation?

People keep cancelling their streaming music subscriptions

Why is this a major issue?

Our monthly revenue growth is slowing

What would success look like for you?

Decreasing cancellation rate by 2% would be a big win

End User

Carter Careswell
Customer Care Team

BRAINSTORM PROBLEMS

Before thinking about potential solutions, it helps to **brainstorm problems** with the end user to identify improvements and opportunities

- The ideas do not have to be data science related
- The sky is the limit – don't think about your data or resource limitations
- You want to start by thinking big and then whittle it down from there

Data Scientist

Debbie Dayda
Data Science Team

Is our product differentiated from competitors?

Are technical bugs or limitations to blame?

Is our product too expensive?

Do we need to expand our music library?

Do we even know WHY customers are cancelling?

That's a good one – let's focus on that problem first!

End User

Carter Careswell
Customer Care Team

BRAINSTORM SOLUTIONS

Once you settle on a problem, the next step is to **brainstorm potential solutions**

Data science can be *one* potential solution, but it's not the only one:

- **PROS**: Solutions are backed by data, may lead to hidden insights, let you make predictions
- **CONS**: Projects take more time and specialized resources, and can lead to complex solutions

POTENTIAL SOLUTIONS

*Ask the product team to **add a survey** to capture cancellation feedback*

***Conduct customer interviews** to gather qualitative data and insights*

*Suggest that the leadership team **speak with other leaders** in the space to compare notes*

***Speak with customer reps** about any changes they've noticed in recent customer interactions*

***Research external factors** that may be impacting cancellations (competitive landscape, news, etc.)*

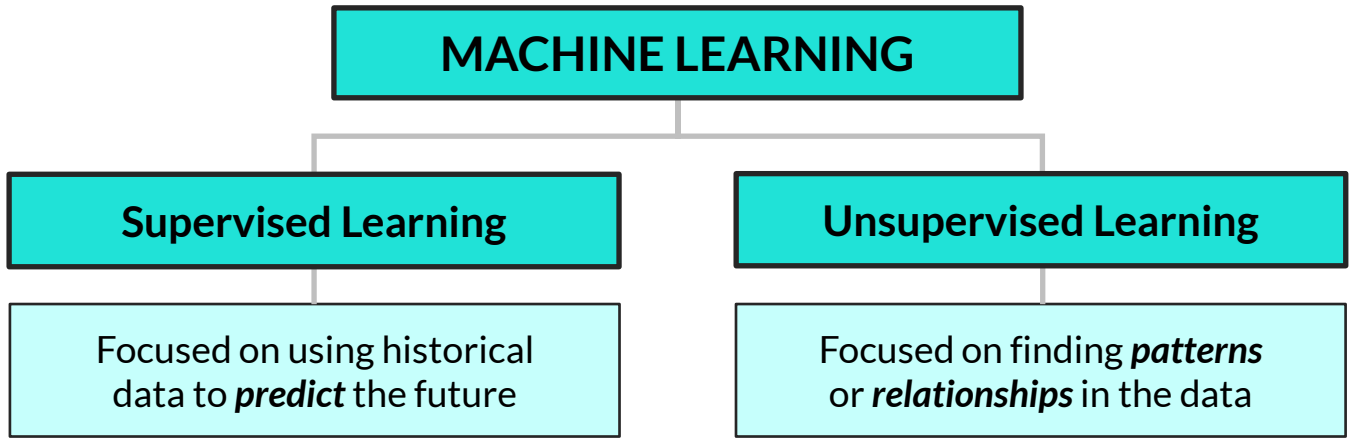
*Use data to **identify the top predictors** for account cancellations*

This is the only data science solution!

DETERMINE THE TECHNIQUES

If you decide to take a data science approach to solving the problem, the next step is to **determine which techniques** are most suitable:

- Do you need supervised or unsupervised learning?



KNOWLEDGE CHECK

- Project Scoping Steps
- Thinking Like an End User
- Problems & Solutions
- Modeling Techniques
- Data Requirements
- Summarizing the Scope

Estimating how many customers will visit your website on New Year's Day

Looking at the products purchased by the highest-spend customers

Visualizing cancellation rate over time for various customer segments

Identifying the main themes mentioned in customer reviews

Clustering customers into different groups based on their preferences

Flagging which customers are most likely to cancel their membership

Supervised Learning

Unsupervised Learning

KNOWLEDGE CHECK

- Project Scoping Steps
- Thinking Like an End User
- Problems & Solutions
- Modeling Techniques
- Data Requirements
- Summarizing the Scope

Looking at the products purchased by the highest-spend customers

Visualizing cancellation rate over time for various customer segments

Supervised Learning

Estimating how many customers will visit your website on New Year's Day

Flagging which customers are most likely to cancel their membership

Unsupervised Learning

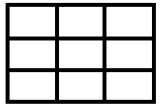
Identifying the main themes mentioned in customer reviews

Clustering customers into different groups based on their preferences

IDENTIFY DATA REQUIREMENTS

- Project Scoping Steps
- Thinking Like an End User
- Problems & Solutions
- Modeling Techniques
- **Data Requirements**
- Summarizing the Scope

After deciding on a technique, the next step is to **identify data requirements**, including structure, features, sources and scope

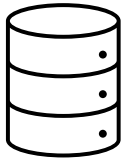


Structure

You will need to structure your data in different ways, whether you are using supervised or unsupervised techniques

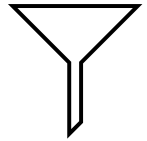
Features

Brainstorm specific columns or "features" that might provide insight (these will be good inputs for your models)



Sources

Determine which additional data sources (if any) are required to create or "engineer" those features



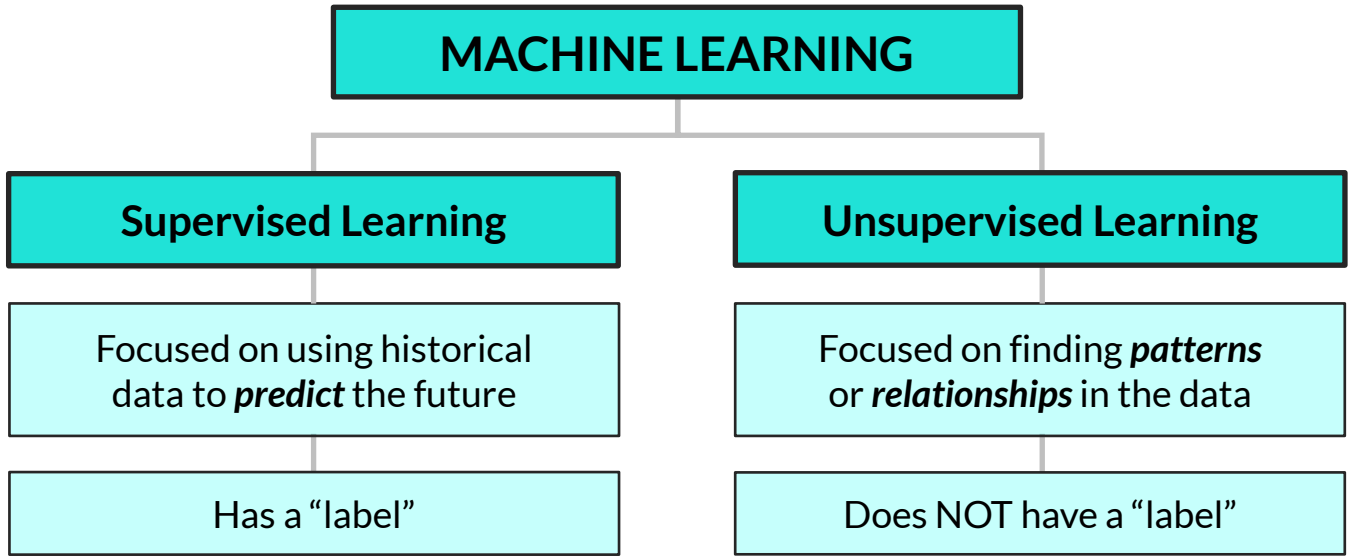
Scope

Narrow down your data to just what you need to get started (you can expand and iterate from there)

DATA STRUCTURE

How you **structure your data** often depends on which technique you're using:

- A **supervised** learning model takes in **labeled** data *(the outcome you want to predict)*
- An **unsupervised** learning model takes in **unlabeled** data



*A "label" is an **observed variable which you are trying to predict***

(house price ($), spam or not (1/0), chance of failure (probability), etc.)

DATA STRUCTURE

How you **structure your data** often depends on which technique you're using:

- A **supervised** learning model takes in **labeled** data *(the outcome you want to predict)*
- An **unsupervised** learning model takes in **unlabeled** data

EXAMPLE | ***Supervised Learning**: Predicting which customers are likely to cancel*

x variables = inputs = features
(what goes into a model)

y variable = output = target
(what you are trying to predict)

Customer	Months Active	Monthly Payment	Listened to Indie Artists?	Cancelled?
Aria	25	$9.99	Yes	No
Chord	2	$0	No	No
Melody	14	$14.99	No	Yes
When shown a new customer, predict whether they will cancel or not				
Rock	12	$9.99	Yes	**???**

*Each row represents a **customer***

*Each cancellation value is called the **label** for the row*

DATA STRUCTURE

How you **structure your data** often depends on which technique you're using:

- A **supervised** learning model takes in **labeled** data *(the outcome you want to predict)*
- An **unsupervised** learning model takes in **unlabeled** data

EXAMPLE | ***Unsupervised Learning***: *Clustering customers based on listening behavior*

x variables = inputs = features
(what goes into a model)

Each row represents a ***customer*** →

Customer	Daily Listening Hours	# Pop Songs	# Soundtracks	# Podcasts
Aria	10	50	3	4
Chord	3	28	0	0
Melody	2	0	0	3

When shown a new customer, figure out which customers it's most similar to

Rock	1	4	0	2

MODEL FEATURES

To identify relevant **model features**, brainstorm any potential variables that might be useful as model inputs based on your goal, for example:

- Supervised learning – features that will do a good job *predicting* cancellations
- Unsupervised learning – features that will do a good job *differentiating* customers

Data Scientist

Debbie Dayda
Data Science Team

What factors might help us predict cancellations?

Well, we recently increased the monthly subscription rate

It could also have something to do with auto-renewals

I wonder if certain demographics are more likely to cancel

Maybe a competitor launched a new offer or promotion?

End User

Carter Careswell
Customer Care Team

DATA SOURCES

Once you've identified a list of relevant features, **brainstorm data sources** you can use to create or engineer those features

Features:	Potential Sources:	Ease of Access:
Monthly rate	Customer subscription history, customer listening history	Easy *(internal data)*
Auto-renew		
Age	Customer demographics	
Urban vs. Rural		
Competitor Promotion	Customer's other subcriptions, competitor promotion history	Hard *(external data)*

DATA SCOPE

Next, **narrow the scope of your data** to remove sources that are difficult to obtain and prioritize the rest

- Remember that more data doesn't necessarily mean a better model!

Data Sources:

- Customer subscription history
- Customer listening history

Timeframe:
Past three months of data

Customers:
Only customers who actively subscribed, not those who were grandfathered in

PRO TIP: Aim to produce a **minimum viable product** (MVP) at this stage – if you go too deep without feedback, you risk heading down unproductive paths or over-complicating the solution

SUMMARIZE THE SCOPE & OBJECTIVES

The final step is to **clearly summarize the project scope and objectives**:

- What techniques and data do you plan to leverage?
- What specific impact are you trying to make?

Data Scientist

Debbie Dayda
Data Science Team

*We plan to use **supervised learning** to predict which customers are likely to cancel their subscription, using **the past three months of subscription and listening history**. This will allow us to:*

- *Identify the **top predictors for cancellation** and figure out how to address them*
- *Use the model to **flag customers who are likely to cancel** and take proactive steps to keep them subscribed*

*Our goal is to **reduce cancellations by 2%** over the next year.*

End User

Carter Careswell
Customer Care Team

Sweet!

KEY TAKEAWAYS

When scoping a project, start by **thinking like an end user**

- *Take time to sit down with end users or stakeholders to understand the situation and business context, brainstorm potential problems and solutions, and align on key objectives*

Don't limit yourself when brainstorming ideas for problems, solutions, or data

- *Start by thinking big and whittling ideas down from there, and keep in mind that many potential solutions likely won't require data science*

Supervised & unsupervised models require **different data structures**

- *Supervised models use labeled data which includes a target variable for the model to predict, while unsupervised models use unlabeled data to find patterns or relationships*

Leverage the **MVP approach** when working on data science projects

- *Start with something relatively quick and simple as a proof of concept, then continuously iterate to refine and improve your model once you know you're on the right track*

INSTALLING JUPYTER NOTEBOOK

INSTALLING JUPYTER NOTEBOOK

In this section we'll install Anaconda and introduce **Jupyter Notebook**, a user-friendly coding environment where we'll be coding in Python

TOPICS WE'LL COVER:

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

GOALS FOR THIS SECTION:

- Install Anaconda and launch Jupyter Notebook
- Get comfortable with the Jupyter Notebook environment and interface

WHY PYTHON?

Python is the most popular programming language used by data scientists around the world due to its:

Scalability

Unlike some data tools or self-service platforms, Python is **open source**, **free**, and **built for scale**

Versatility

With powerful libraries and packages, Python can add value at **every stage of the data science workflow**, from data prep to data viz to machine learning

Automation

Python can **automate workflows & complex tasks out of the box**, without complicated integrations or plug-ins

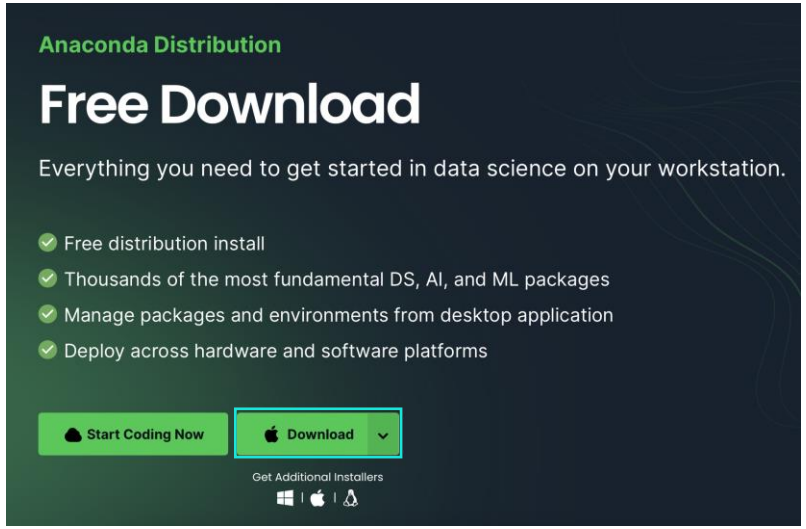
Community

Become part of a **large and active Python user community**, where you can share resources, get help, offer support, and connect with other users

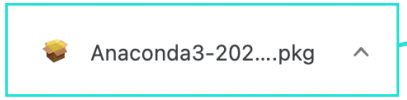
INSTALL ANACONDA (MAC)

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

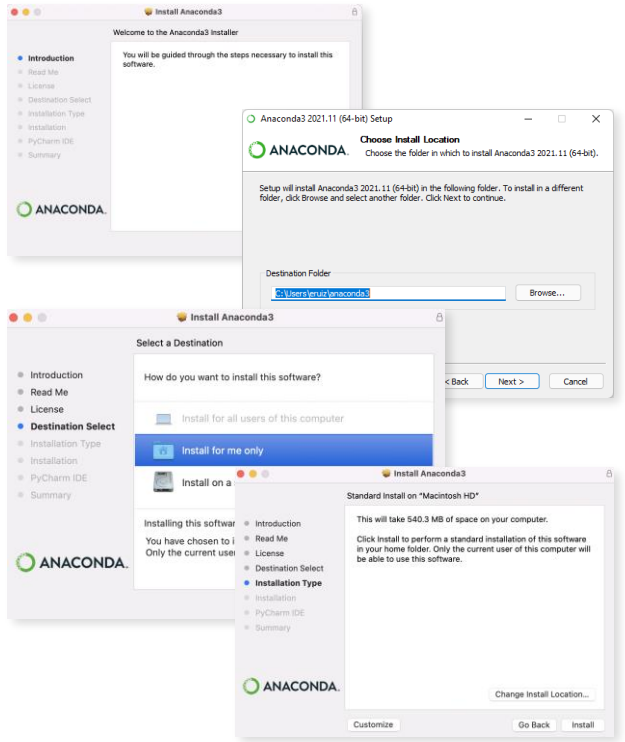
1) Go to **anaconda.com/products/distribution** and click 



2) Launch the downloaded Anaconda **pkg** file

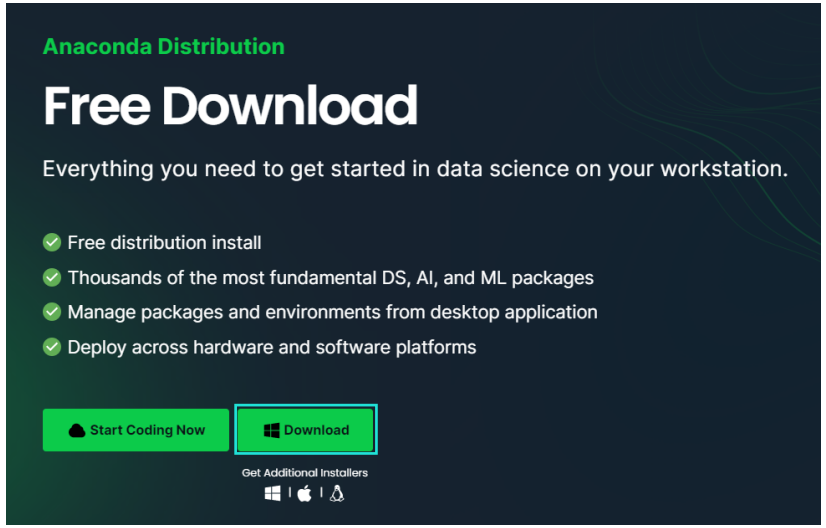


3) Follow the **installation steps**
(default settings are OK)

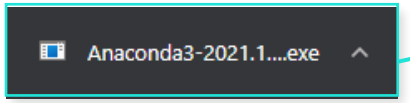


INSTALL ANACONDA (PC)

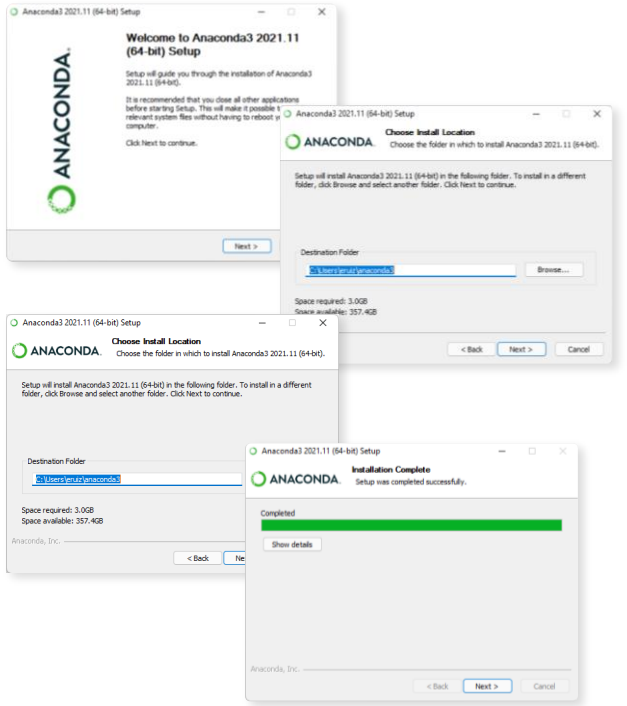
1) Go to **anaconda.com/products/distribution** and click **Download**



2) Launch the downloaded Anaconda **exe** file



3) Follow the **installation steps**
(default settings are OK)

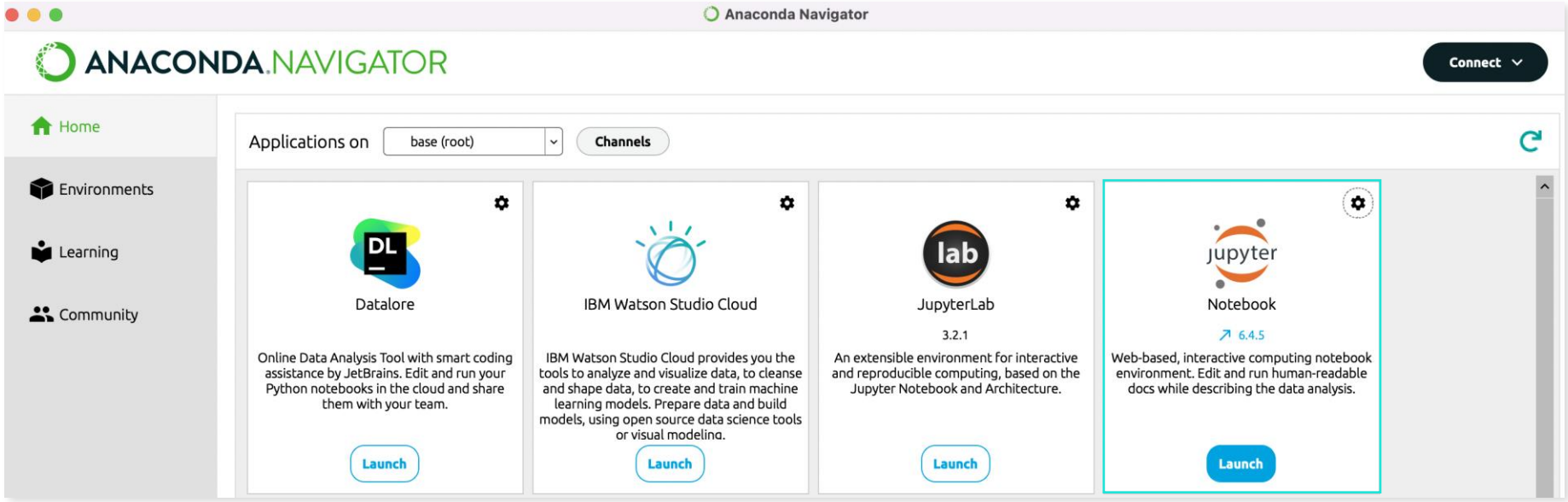


LAUNCHING JUPYTER NOTEBOOK

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

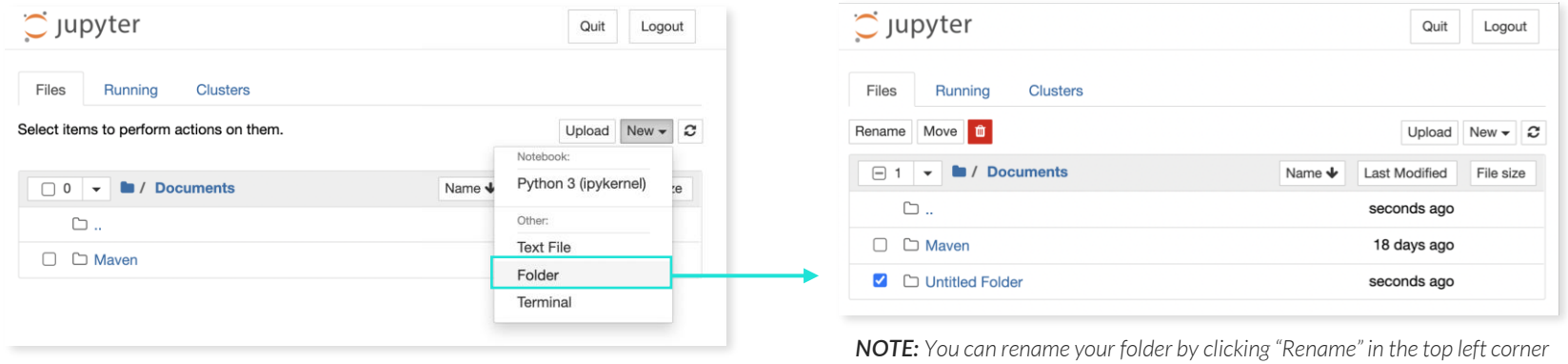
1) Launch **Anaconda Navigator**

2) Find **Jupyter Notebook** and click **Launch**



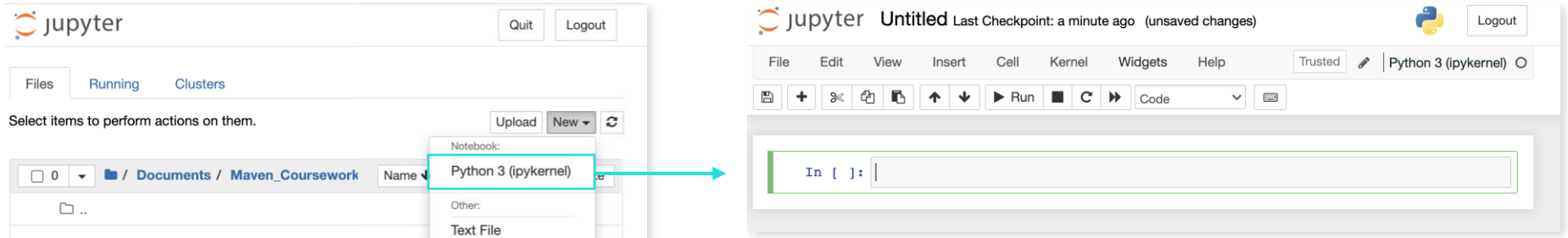
YOUR FIRST JUPYTER NOTEBOOK

1) Once inside the Jupyter interface, **create a folder** to store your notebooks for the course



NOTE: *You can rename your folder by clicking "Rename" in the top left corner*

2) Open your new coursework folder and **launch your first Jupyter notebook**!

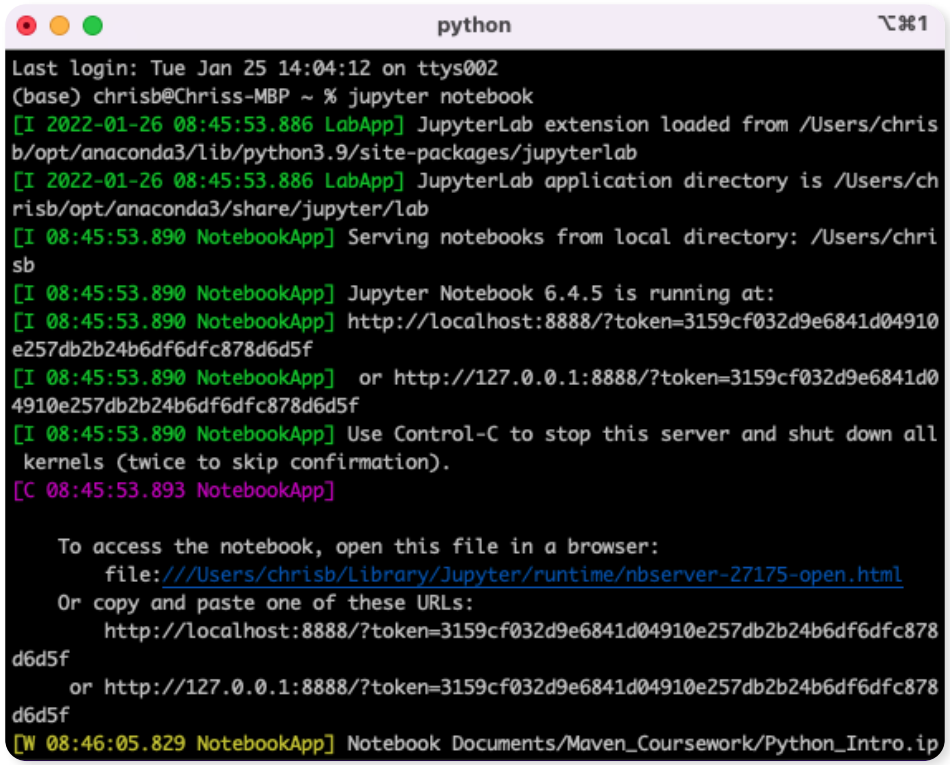


NOTE: *You can rename your notebook by clicking on the title at the top of the screen*

THE NOTEBOOK SERVER

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

NOTE: When you launch a Jupyter notebook, a terminal window may pop up as well; this is called a **notebook server**, and it powers the notebook interface



If you close the server window, **your notebooks will not run!**

Depending on your OS, and method of launching Jupyter, one may not open. As long as you can run your notebooks, don't worry!

THE NOTEBOOK INTERFACE

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

Menu Bar

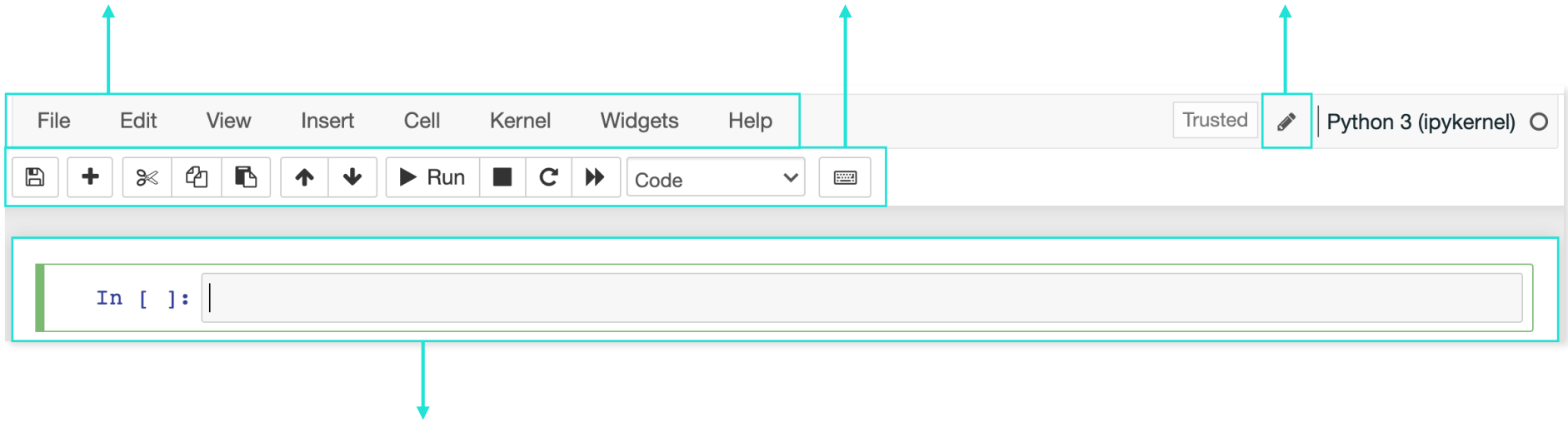
Options to manipulate the way the notebook functions

Toolbar

Buttons for the most-used actions within the notebook

Mode Indicator

*Displays whether you are in **Edit** Mode or **Command** Mode*



Code Cell

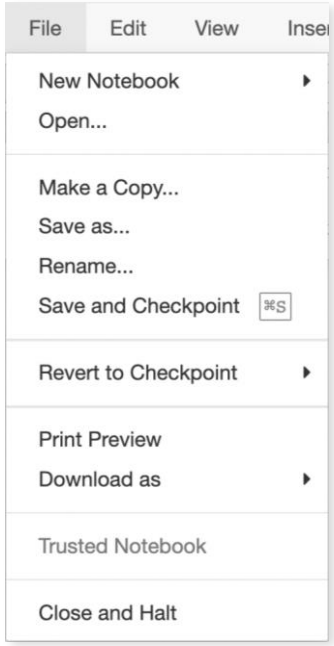
Input field where you will write and edit new code to be executed

MENU OPTIONS

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

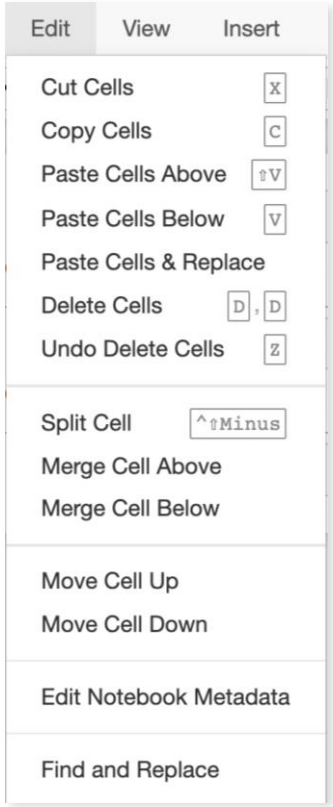
File

Save or revert, make a copy, open a notebook, download, etc.



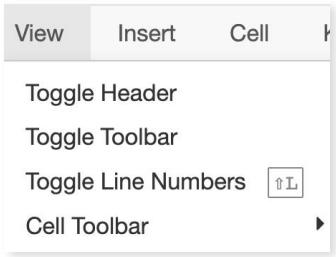
Edit

Edit cells within your notebook (while in command mode)



View

Edit cosmetic options for your notebook.



Insert

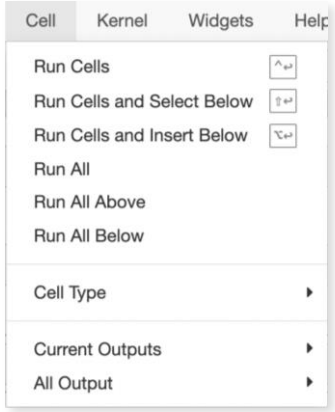
Insert new cells into your notebook

MENU OPTIONS

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

Cell

Access options for running the cells in your notebook

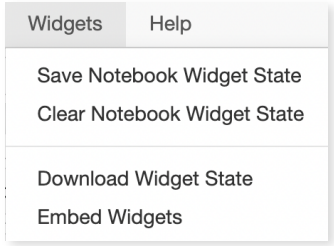


Kernel

Interact with the instance of Python that runs your code

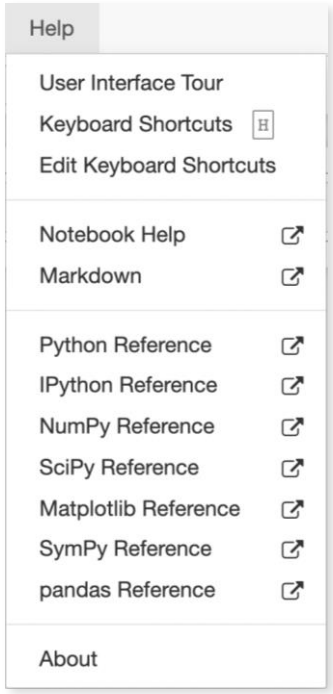
Widgets

Manage interactive elements, or 'widgets' in your notebook



Help

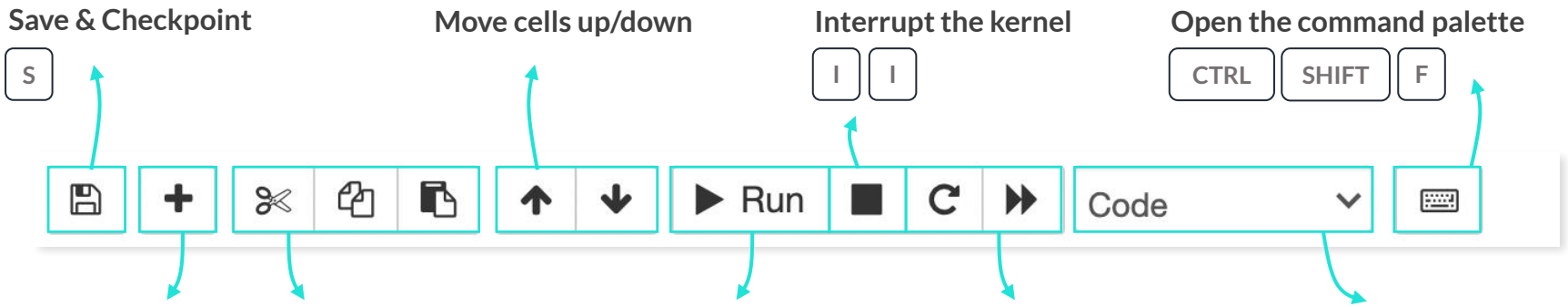
View or edit keyboard shortcuts and access Python reference pages



THE TOOLBAR

The **toolbar** provides easy access to the most-used notebook actions

- These actions can also be performed using hotkeys *(keyboard shortcuts)*

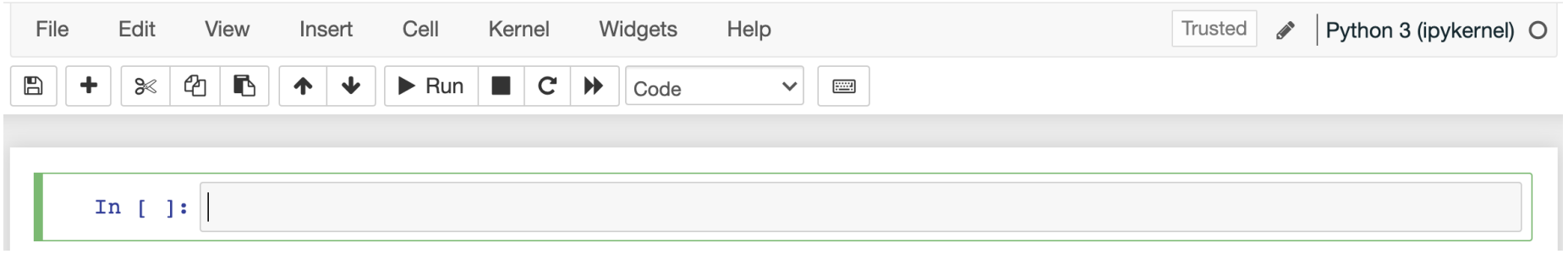


Shortcuts may differ depending on **which mode you are in**

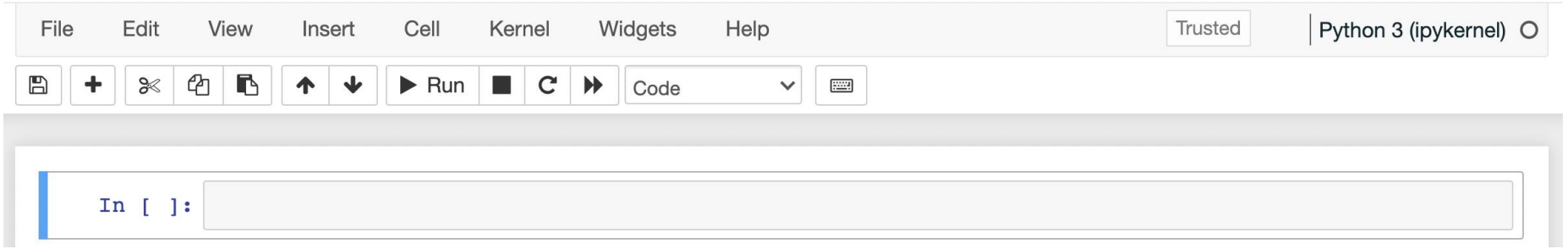
EDIT & COMMAND MODES

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

EDIT MODE is for editing **content within cells**, and is indicated by **green** highlights and a icon

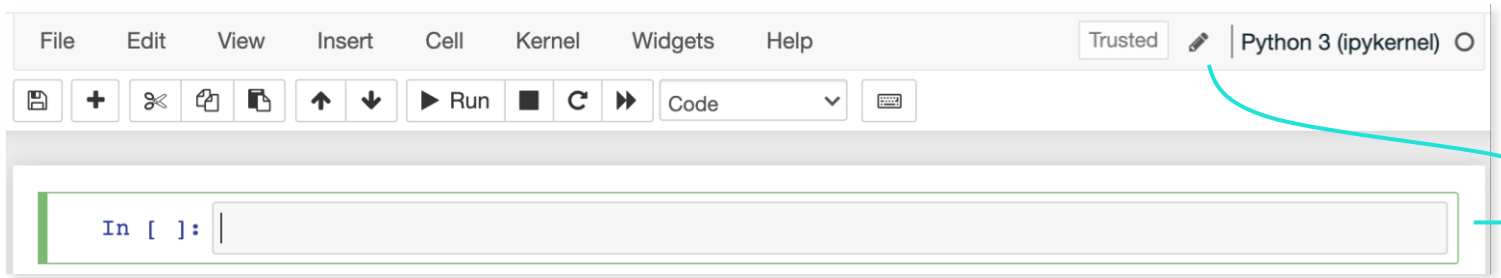


COMMAND MODE is for editing **the notebook**, and is indicated by **blue** highlights and no icon

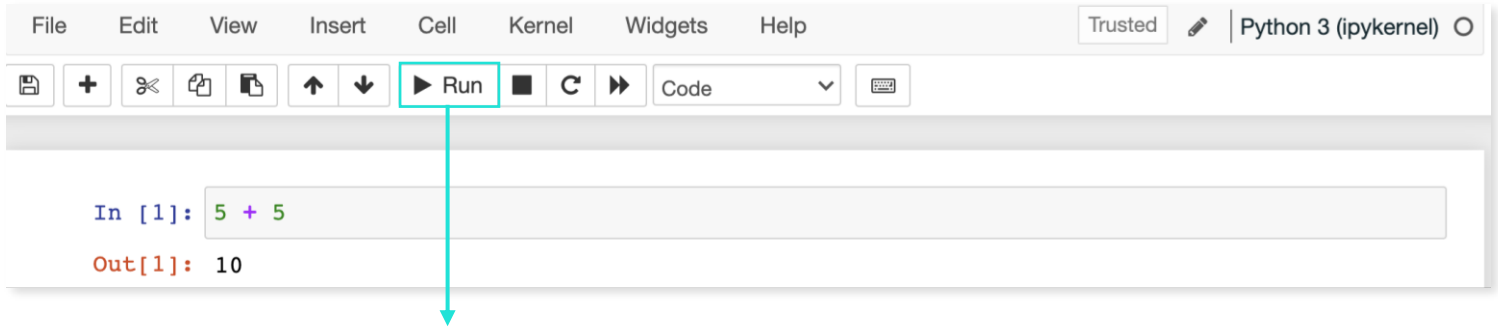


THE CODE CELL

The **code cell** is where you'll write and execute Python code



In **edit mode**, the cell will be highlighted **green** and a pencil icon will appear



Type some code, and click **Run** to execute

- `In[]:` Our code *(input)*
- `Out[]:` What our code produced *(output)**

**Note: not all code has an output!*

THE CODE CELL

The **code cell** is where you'll write and execute Python code

```
In [1]: 5 + 5
Out[1]: 10
```

Click back into the cell (or use the up arrow) and press **SHIFT + ENTER** to rerun the code

```
In [2]: 5 + 5
Out[2]: 10
```

Note that our output hasn't changed, but the number in the brackets increased from **1** to **2**.

This is a **cell execution counter**, which indicates how many cells you've run in the current session.

If the cell is still processing, you'll see `In[*]`

```
In [2]: 5 + 5
Out[2]: 10

In [3]: 5 - 5
Out[3]: 0
```

The cell counter will continue to increment as you run additional cells

COMMENTING CODE

Comments are lines of code that start with '**#**' and do not run

- They are great for explaining portions of code for others who may use or review it
- They can also serve as reminders for yourself when you revisit your code in the future

```
In [4]: # I'm subtracting five from five. Add a space between your hash and comment.
        5 - 5

Out[4]: 0

In [5]: 5 - 5 # change the second 5 to a 6 tomorrow

Out[5]: 0
```

Think about your audience when commenting your code (you may not need to explain basic arithmetic to an experienced Python programmer)

```
# This notebook is about teaching the basics of Jupyter notebook.
# Should i define what a jupyter notebook is here?
# I'm subtracting five from five. Add a space between your hash and comment.
5 - 5  # 5 is the fifth integer greater than zero. It is also the number of fingers on our hand
# 5 is a very interesting number
# so is 0, which is the output
```

```
0
```

Be conscious of over-commenting, which can actually make your code even more difficult to read

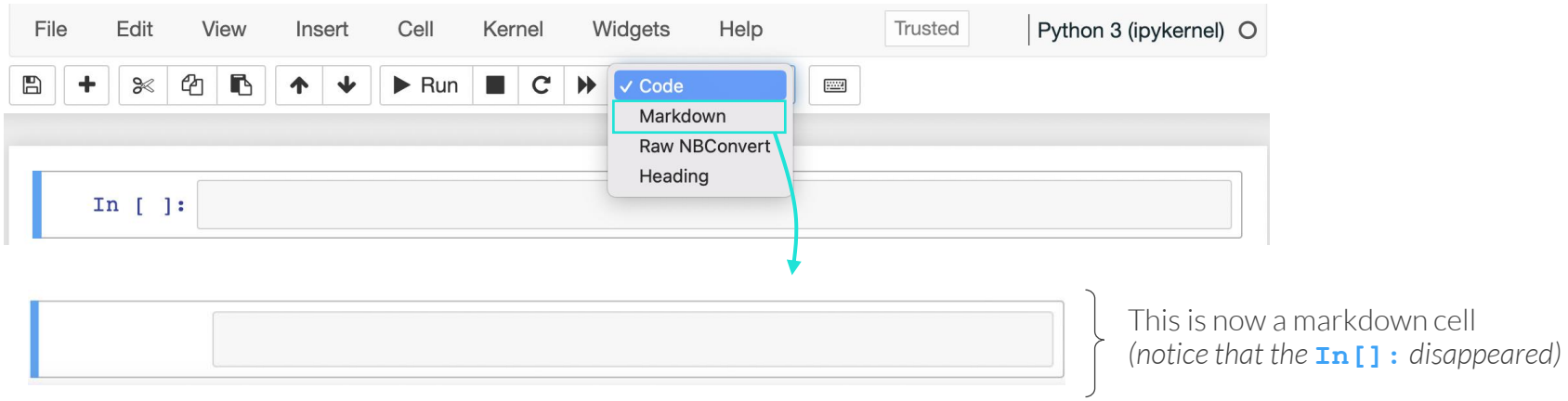
Comments should explain **individual cells or lines of code**, NOT your entire workflow – we have better tools for that!

THE MARKDOWN CELL

Markdown cells let you write structured text passages to explain your workflow, provide additional context, and help users navigate the notebook

To create a markdown cell:

1. Create a new cell above the top cell *(press **A** with the cell selected)*
2. Select "**Markdown**" in the cell menu *(or press **M**)*



MARKDOWN SYNTAX

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

Markdown cells use a special **text formatting syntax**

```
# Jupyter Notebook Intro

## Section 1: Markdown Basics


This is body text. I can use this area to provide more in depth explanations of my:

* Thought process
* Overall workflow
* etc

Anything that would be too much text for comments.

To create bulleted lists, begin a line with *.

Numbered lists can be created by beginning a line with 1., 2., and so on.

Markdown has a **lot** of capabilities, and could be a course on its own. You will learn more as you build more
notebooks and look at the work of others.

The Essentials to get started are:

1. Create headers with # (one is biggest, six is the smallest header)
2. **Bold**, *italicize*, ***Bold AND Italic***
3. Creating bulleted or numbered lists (begin a new line with * for bullets 1. for numbers).
4. Code highlighting e.g. `my_variable = 5`. Use the backtick, not apostrophe.

To explore further, I highly recommend [this guide.](https://www.markdownguide.org/basic-syntax/)
```

MARKDOWN SYNTAX

- Why Python?
- Installation & Setup
- Notebook Interface
- Code vs Markdown Cells
- Helpful Resources

Markdown cells use a special **text formatting syntax**

Jupyter Notebook Intro

Section 1: Markdown Basics

This is body text. I can use this area to provide more in depth explanations of my:

- Thought process
- Overall workflow
- etc

Anything that would be too much text for comments.

To create bulleted lists, begin a line with *.

Numbered lists can be created by beginning a line with 1., 2., and so on.

Markdown has a **lot** of capabilities, and could be a course on its own. You will learn more as you build more notebooks and look at the work of others.

The Essentials to get started are:

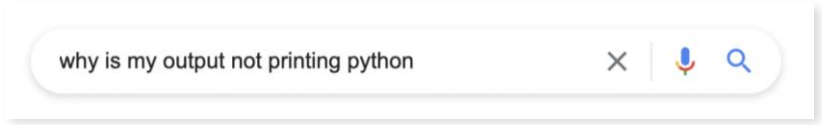
1. Create headers with # (one is biggest, six is the smallest header)
2. **Bold**, *italicize*, ***Bold AND Italic***
3. Creating bulleted or numbered lists (begin a new line with * for bullets 1. for numbers).
4. Code highlighting e.g. `my_variable = 5`. Use the backtick, not apostrophe.

To explore further, I highly recommend this guide.

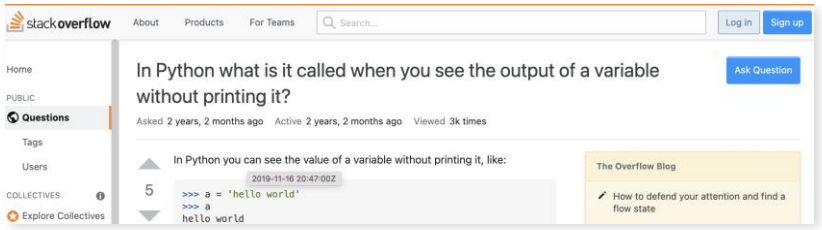
```
In [4]: 5 + 5

Out[4]: 10
```

HELPFUL RESOURCES

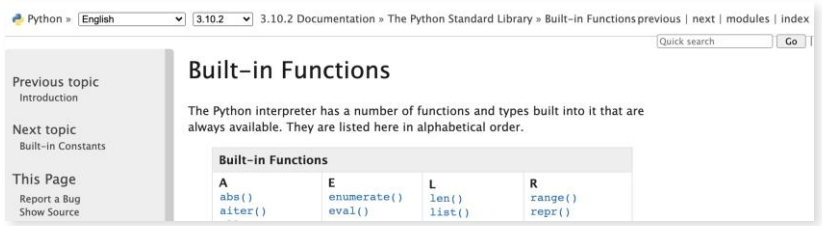


Google your questions – odds are someone else has asked the same thing and it has been answered *(include Python in the query!)*



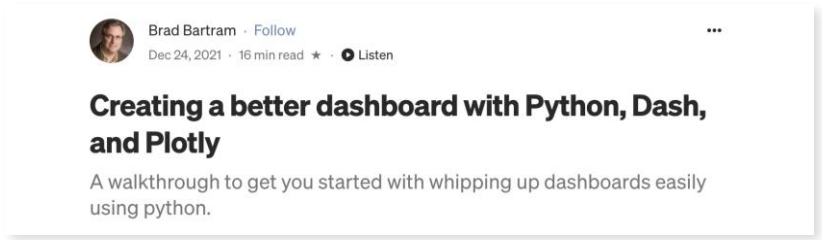
Stack Overflow is a public coding forum that will most likely have the answers to most of the questions you'll search for on Google

https://stackoverflow.com/



The **Official Python Documentation** is a great "cheat sheet" for library and language references

https://docs.python.org/3/



There are many quality **Python** & **Analytics Blogs** on the web, and you can learn a lot by subscribing and reviewing the concepts and underlying code

https://towardsdatascience.com/

KEY TAKEAWAYS

Jupyter Notebook is a user-friendly coding environment

- *Jupyter Notebook is popular among data scientists since they allow you to create and document entire machine learning workflows and render outputs and visualizations on screen*

Code cells are where you write and execute Python code

- *Make sure that you know how to run, add, move, and remove cells, as well as how to restart your kernel or stop the code from executing*

Use **comments** and **markdown** cells for documentation

- *Comments should be used to explain specific portions of your code, and markdown should be used to document your broader workflow and help users navigate the notebook*

GATHERING DATA

GATHERING DATA

In this section we'll cover the steps for **gathering data**, including reading files into Python, connecting to databases within Python, and storing data in Pandas DataFrames

TOPICS WE'LL COVER:

- Data Gathering Process
- Reading Files Into Python
- Connecting to a Database
- Exploring DataFrames

GOALS FOR THIS SECTION:

- Understand the data gathering process and the various ways that data can be stored and structured
- Identify the characteristics of a Pandas DataFrame
- Use Pandas to read in flat files and Excel spreadsheets, and connect to SQL databases
- Quickly explore a Pandas DataFrame

DATA GATHERING PROCESS

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

The **data gathering process** involves finding data, reading it into Python, transforming it if necessary, and storing the data in a Pandas DataFrame

1) Find the data

```
{
  "Name": ["Jim", "Dwight",
           "Angela", "Tobi"],
  "Age":  [26, 28, 27, 33],
  "Department": ["Sales", "Sales",
                 "Accounting",
                 "Human Resources"]
}
```

Raw data can come in many shapes and forms

2) Read in the data

Apply transformations using Python, if necessary

3) Store the data

	Name	Age	Department
0	Jim	26	Sales
1	Dwight	28	Sales
2	Angela	27	Accounting
3	Tobi	33	Human Resources

*Tables in Python are stored as **Pandas DataFrames** (more on this later!)*

DATA SOURCES

Data Gathering Process

Reading Files into Python

Connecting to a Database

Exploring DataFrames

Data can come from a variety of **sources**:

Local files

You can read data from a file stored on your computer

Common examples:

- Flat files *(.csv, .txt, etc.)*
- Spreadsheets *(.xlsx, etc.)*

Databases

You can connect to a database and write queries

Common examples:

- SQL databases
- NoSQL databases

Web access

You can programmatically extract data from websites

Common examples:

- Web scraping *(.html, etc.)*
- API *(.json, .xml, etc.)*

Python is a great tool for data gathering due to its ability to read in and transform data coming from a **wide variety of sources** and formats

STRUCTURED VS UNSTRUCTURED DATA

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

Structured

Already stored as tables

.xlsx

.db

Semi-Structured

Easily converted to tables

.csv

.json

Unstructured

Not easily converted to tables

.pdf

.jpg

To do analysis, data needs to be **structured as a table** with rows and columns

- Most data sources are already organized in this way and are ready for analysis
- Unstructured data sources require additional transformations before doing analysis

THE PANDAS DATAFRAME

Data Gathering Process

Reading Files into Python

Connecting to a Database

Exploring DataFrames

To do analysis in Python, data must reside within a **Pandas DataFrame**

- DataFrames look just like regular tables with rows and columns, but with additional features
- External structured and semi-structured data can be read directly into a DataFrame

*The **row index** assigns a unique ID to each row*

(The index starts at 0)

	Course ID	Course Name	Instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao
3	301	Algorithms	Chris Bruehl
4	331	Natural Language Processing	Alice Zhao

*Each column contains data of a **single data type***

(integer, text, etc.)

READING DATA INTO PYTHON

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

Pandas allows you to **read in data** from multiple file formats:

pd.read_csv	*Reads in data from a delimiter-separated flat file*	**pd.read_csv(*file path*, *sep*, *header*)**
pd.read_excel	*Reads in data from a Microsoft Excel file*	**pd.read_excel(*file path*, *sheet_name*)**
pd.read_json	*Reads in data from a JSON file*	**pd.read_json(*file path*)**

'pd' is the standard alias for the Pandas library

READING FLAT FILES

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

pd.read_csv() lets you read flat files by specifying the file path

```
pd.read_csv(file path, sep, header)
```

The file location, name and extension

Examples:

- *'data.csv'*
- *'sales/october.tsv'*

The column delimiter

Examples:

- *sep=',' (default)*
- *sep='\t'*

If the data has column headers in the first row

- *header='infer' (default)*
- *header=None (no headers)*

PRO TIP: Place the file in the same folder as the Jupyter Notebook so you don't have to specify the precise file location

For a full list of arguments, visit: ***https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html***

READING FLAT FILES

pd.read_csv() lets you read flat files by specifying the file path

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

course_offerings.csv

```
course_id,course_name,instructor
101,Intro to Python,Chris Bruehl
102,Intro to SQL,John Pauler
201,Exploratory Data Analysis,Alice Zhao
301,Algorithms,Chris Bruehl
331,Natural Language Processing,Alice Zhao
```

```
import pandas as pd

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

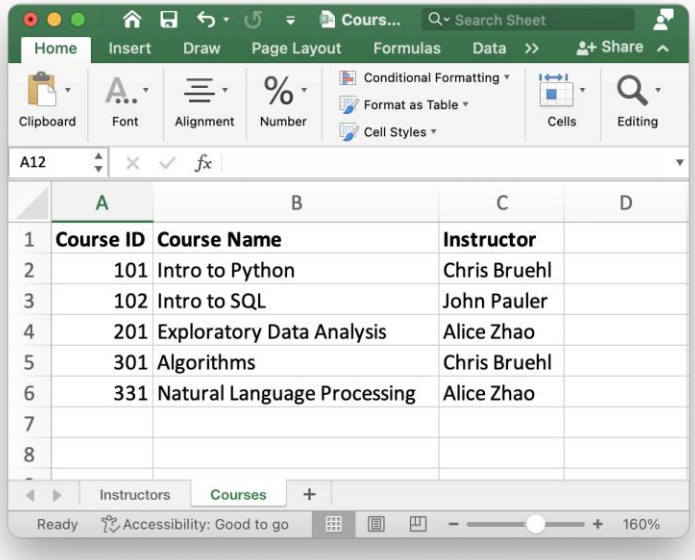
	course_id	course_name	instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao
3	301	Algorithms	Chris Bruehl
4	331	Natural Language Processing	Alice Zhao

READING EXCEL FILES

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

pd.read_excel() lets you read Excel files by specifying the file path

- You can use the "sheet_name" argument to specify the worksheet *(default is 0 – first sheet)*




```
import pandas as pd

pd.read_excel('course_offerings.xlsx', sheet_name=1)
```

	Course ID	Course Name	Instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao
3	301	Algorithms	Chris Bruehl
4	331	Natural Language Processing	Alice Zhao

CONNECTING TO A SQL DATABASE

To connect to a SQL database, import a database driver and specify the database connection, then use **pd.read_sql()** to query the database using SQL code


```
# Connect to a SQLite database

import sqlite3
conn = sqlite3.connect('maven.db')

pd.read_sql('SELECT * FROM courses', conn)
```

	course_id	course	instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao
3	301	Algorithms	Chris Bruehl
4	331	Natural Language Processing	Alice Zhao

SQL Software	Database Driver
SQLite	sqlite3
MySQL	mysql.connector
Oracle	cx_Oracle
PostgreSQL	psycopg2
SQL Server	pyodbc

QUICKLY EXPLORE A DATAFRAME

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

After reading data into Python, it's common to **quickly explore the DataFrame** to make sure the data was imported correctly

df.head()	*Display the first five rows of a DataFrame*
df.shape	*Display the number of rows and columns of a DataFrame*
df.count()	*Display the number of values in each column*
df.describe()	*Display summary statistics like mean, min and max*
df.info()	*Display the non-null values and data types of each column*

QUICKLY EXPLORE A DATAFRAME

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

After reading data into Python, it's common to **quickly explore the DataFrame** to make sure the data was imported correctly

```
df.head(3)
```

	course_id	course_name	instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao

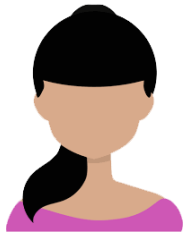
```
df.shape
```

```
(5, 3)
```

```
df.describe()
```

	course_id
count	5.000000
mean	207.200000
std	107.829495
min	101.000000
25%	102.000000
50%	201.000000
75%	301.000000
max	331.000000

ASSIGNMENT: READ A FILE INTO PYTHON



NEW MESSAGE

June 1, 2023

From: **Anna Analysis** *(Senior Data Scientist)*

Subject: **New Survey Data**

Hi there,

I hear you're the new junior data scientist on the team. Welcome!

Our polling partners just finished collecting survey data on the happiness levels in each country. I've attached the file.

Can you load the data into Python and confirm the number of rows in the data and the range of the happiness scores? Then we can walk through next steps together.

Thanks!
Anna

happiness_survey_data.csv

Reply Forward

Key Objectives

1. Read in data from a .csv file
2. Store the data in a DataFrame
3. Quickly explore the data in the DataFrame

KEY TAKEAWAYS

Python can **read data** from your computer, a database, or the internet

- *Flat files & spreadsheets are typically stored locally on a computer, but as a data scientist you'll likely need to connect to and query a SQL database, and potentially collect data through web scraping or an API*

Data needs to be organized into **rows** & **columns** for analysis

- *Flat files, spreadsheets and SQL tables are already in this format, but if your data is unstructured you would need to extract the relevant data and transform it into rows and columns yourself*

Python tables are known as Pandas **DataFrames**

- *DataFrames include a unique row index and each column must be the same data type (integer, text, etc.)*
- *There are several ways to quickly explore the characteristics of a DataFrame (head, shape, describe, etc.)*

CLEANING DATA

CLEANING DATA

In this section we'll cover the steps for **cleaning data**, including converting columns to the correct data type, handling data issues and creating new columns for analysis

TOPICS WE'LL COVER:

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

GOALS FOR THIS SECTION:

- Identify and convert between Python data types
- Find common "messy" data issues and become familiar with the different ways to resolve them
- Create new calculated columns from existing data

DATA CLEANING OVERVIEW

The goal of **data cleaning** is to get raw data into a format that's ready for analysis

This includes:

- Converting columns to the correct **data types** for analysis
- Handling **data issues** that could impact the results of your analysis
- **Creating new columns** from existing columns that are useful for analysis

The order in which you complete these data cleaning tasks will vary by dataset, but this is a good starting point

Even though there are automated tools available, doing some **manual data cleaning** provides a good opportunity to start understanding and getting a good feel for your data

DATA TYPES

When using Pandas to read data, columns are automatically assigned a **data type**

- Use the **.dtypes** attribute to view the data type for each DataFrame column
- Note that sometimes numeric columns *(int, float)* and date & time columns *(datetime)* aren't recognized properly by Pandas and get read in as text columns *(object)*

```
df = pd.read_csv('customers.csv')
df.head()
```

	Name	City	State	Income	Birthdate
0	Susan Rodriguez	Maplewood	NJ	$45,000	1970-09-12
1	Joseph Martinez	Portland	OR	$120,000	1960-12-10
2	Joseph Martinez	Portland	OR	$120,000	1960-12-10
3	Wayne Nielson	Dahlgren	VA	$75,000	1968-12-05
4	Beverly Nixon	Long Island	NY	$45,000,000	1962-03-05

Default data types

```
df.dtypes
```

```
Name          object
City          object
State         object
Income        object
Birthdate     object
dtype: object
```

*These need to be **converted** to be analyzed!*

Data Type	Description
bool	Boolean, True/False
int64	Integers
float64	Floating point, decimals
object	Text or mixed values
datetime64	Date and time values

PRO TIP: Use the **.info()** method as an alternative to show additional information along with the data types

CONVERTING TO DATETIME

Use **pd.to_datetime()** to convert object columns into datetime columns

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df.Birthdate
```

```
0     1970-09-12
1     1960-12-10
2     1960-12-10
3     1968-12-05
4     1962-03-05
5     1993-09-23
6     1996-03-12
7     1994-01-13
8     1967-08-27
9            NaN
10           NaN
11           NaN
12    1968-05-19
Name: Birthdate, dtype: object
```

```
# convert to a datetime column
df.Birthdate = pd.to_datetime(df.Birthdate)
df.Birthdate
```

```
0    1970-09-12
1    1960-12-10
2    1960-12-10
3    1968-12-05
4    1962-03-05
5    1993-09-23
6    1996-03-12
7    1994-01-13
8    1967-08-27
9           NaT
10          NaT
11          NaT
12   1968-05-19
Name: Birthdate, dtype: datetime64[ns]
```

Note that the missing values are now NaT instead of NaN (more on missing data later!)

PRO TIP: Pandas does a pretty good job at detecting the datetime values from an object if the text is in a standard format (like "YYYY-MM-DD"), but you can also manually specify the format using the format argument within the pd.to_datetime() function: **pd.to_datetime(dt_col, format='%Y-%M-%D')**

For a full list of formats, visit: ***https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior***

CONVERTING TO NUMERIC

Use **pd.to_numeric()** to convert object columns into numeric columns

- To remove non-numeric characters ($, %, etc.), use **str.replace()**

```
df.Income
```

```
0         $45,000
1        $120,000
2        $120,000
3         $75,000
4     $45,000,000
5         $50,000
6        $105,000
7         $62,000
8         $60,000
9             NaN
10        $80,000
11            NaN
12       $110,000
Name: Income, dtype: object
```

Currency data is read in as an object by Python due to the dollar sign ($) and comma separator (,)

```
# remove all dollar signs and commas
clean_income = df.Income.str.replace('$','').str.replace(',','')
clean_income
```

```
0        45000
1       120000
2       120000
3        75000
4     45000000
5        50000
6       105000
7        62000
8        60000
9          NaN
10       80000
11         NaN
12      110000
Name: Income, dtype: object
```

```
# convert to a numeric column
df.Income = pd.to_numeric(clean_income)
```

```
df.dtypes
```

```
Name                    object
City                    object
State                   object
Income                 float64
Birthdate    datetime64[ns]
dtype: object
```

An alternative is to use Series.astype() to convert to more specific data types like 'int', 'float', 'object' and 'bool', but **pd.to_numeric() can handle missing values** (NaN), while Series.astype() cannot

ASSIGNMENT: CONVERTING DATA TYPES

NEW MESSAGE

June 12, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Data in Python Request**

Hi there,

We just finished collecting survey data from a few thousand customers who've purchased our alarm clocks (see attached).

Can you read the data into Python and make sure the data type of each column makes sense? We'd like to do quite a few calculations using the data, so if a column can be converted to a numeric or datetime column, please do so.

Thanks!
Alan

Alarm Survey Data.xlsx

Reply Forward

Key Objectives

1. Read in data from the Excel spreadsheet and store it in a Pandas DataFrame
2. Check the data type of each column
3. Convert object columns into numeric or datetime columns as needed

DATA ISSUES OVERVIEW

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

Data issues need to be identified and corrected upfront in order to not impact or skew the results of your analysis

Common "messy" data issues include:

Duplicates

Inconsistent text & typos

Name	City	State	Income	Birthdate
Susan Rodriguez	Maplewood	NJ	$45,000	1970-09-12
Joseph Martinez	Portland	OR	$120,000	1960-12-10
Joseph Martinez	Portland	OR	$120,000	1960-12-10
Wayne Nielson	Dahlgren	VA	$75,000	1968-12-05
Beverly Nixon	Long Island	NY	$45,000,000	1962-03-05
Veronica Comerford	Tuskegee	AL	$50,000	1993-09-23
Ivan Layton	Albany	New York	$105,000	1996-03-12
Laura Hailey	Ephraim	Utah	$62,000	1994-01-13
John Depaul	Baton Rouge	LA	$60,000	1967-08-27
James Weiss	Houston	TX		
Addie Stevenson	Jacksonville		$80,000	
Daniel Long				
Queen Watson	Los Angeles	CA	$110,000	1968-05-19

Outliers

Missing data

MISSING DATA

There are various ways to represent **missing data** in Python

- **np.NaN** – NumPy's NaN is the most common representation *(values are stored as floats)*
- **pd.NA** – Pandas' NA is a newer missing data type *(values can be stored as integers)*
- **None** – Base Python's default missing data type *(doesn't allow numerical calculations)*

```
df = pd.read_csv("customers.csv")
df.tail()
```

	Name	City	State	Income	Age
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

*Missing values are treated as **np.NaN** when data is read into Pandas*

IDENTIFYING MISSING DATA

The easiest way to identify missing data is with the **.isna()** method

- You can also use **.info()** or **.value_counts(dropna=False)**

```
df.isna()
```

	Name	City	State	Income	Age
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False
6	False	False	False	False	False
7	False	False	False	False	False
8	False	False	False	False	False
9	False	False	False	True	True
10	False	False	True	False	True
11	False	True	True	True	True
12	False	False	False	False	False

*This returns **True** for any missing values*

```
df.isna().sum()
```

```
Name      0
City      1
State     2
Income    2
Age       3
dtype: int64
```

*Use **sum()** to return the missing values by column*

```
df[df.isna().any(axis=1)]
```

	Name	City	State	Income	Age
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN

*Or use **any(axis=1)** to select the rows with missing values*

HANDLING MISSING DATA

There are multiple ways to **handle missing data**:

- Keep the missing data as is
- Remove an entire row or column with missing data
- Impute missing numerical data with a 0 or a substitute like the average, mode, etc.
- Resolve the missing data based on your domain expertise

```
df[df.isna().any(axis=1)]
```

	Name	City	State	Income	Age
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN

Missing age values are likely ok to keep

We have no data on this customer

This is likely Jacksonville, FL

Can we calculate an approximate income?

There is no right or wrong way to deal with missing data, which is why it's important to **be thoughtful and deliberate** in how you handle it

KEEPING MISSING DATA

You can still perform calculations if you choose to **keep missing data**

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

```
df["Age"].mean()
```

```
48.5
```

```
df["Age"].count()
```

```
10
```

These ignore the missing values

REMOVING MISSING DATA

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

The **.dropna()** method removes rows with missing data

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

Our original data set with all of its rows, including all missing values

REMOVING MISSING DATA

The **.dropna()** method removes rows with missing data

Removes any rows with NaN values

```
df.dropna()
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
12	Queen Watson	Los Angeles	CA	110000.0	54.0

*Note that the row index is now skipping values, but you can reset the index with **df.dropna().reset_index()***

REMOVING MISSING DATA

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

The **.dropna()** method removes rows with missing data

Removes any rows with NaN values

```
df.dropna()
```

Removes rows that only have NaN values

```
df.dropna(how="all")
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

REMOVING MISSING DATA

The **.dropna()** method removes rows with missing data

Removes any rows with NaN values

```
df.dropna()
```

Removes rows that only have NaN values

```
df.dropna(how="all")
```

Removes rows that don't have at least "n" values

```
df.dropna(thresh=2)
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

REMOVING MISSING DATA

The **.dropna()** method removes rows with missing data

Removes any rows with NaN values

```
df.dropna()
```

Removes rows that only have NaN values

```
df.dropna(how="all")
```

Removes rows that don't have at least "n" values

```
df.dropna(thresh=2)
```

Removes rows with NaN values in a specified column

```
df.dropna(subset=["City"])
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

PRO TIP: KEEPING NON-MISSING DATA

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

You can use **.notna()** to keep non-missing data instead

```
df[df["City"].notna()]
```

Note that using .dropna() or .notna() to remove rows with missing data **does not make permanent changes** to the DataFrame, so you need to save the output to a new DataFrame (or the same one) or set the argument inplace=True

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

IMPUTING MISSING DATA

The **.fillna()** method imputes missing data with an appropriate value

- There are many ways to impute data in a column *(zero, mean, mode, etc.)*, so take a moment to decide the best one – if you're unsure, reach out to a subject matter expert

```
df[["Income"]]
```

	Income
0	45000.0
1	120000.0
2	120000.0
3	75000.0
4	45000000.0
5	50000.0
6	105000.0
7	62000.0
8	60000.0
9	NaN
10	80000.0
11	110000.0

What value can be used here?

```
df["Income"] = df["Income"].fillna(df["Income"].median())
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

Using the median removes the impact of the outlier

RESOLVING MISSING DATA

An alternative to imputing is to **resolve missing data** with domain expertise

- You can use the **.loc[]** accessor to update a specific value

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

```
df.loc[10,"State"] = "FL"
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	FL	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

ASSIGNMENT: MISSING DATA

NEW MESSAGE

June 13, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Missing Data Check**

Hi again,

Can you check the file I sent you yesterday for missing data?

Please use your best judgement when choosing how to handle the missing data and let me know what approaches you decide to take.

Thanks!
Alan

Reply Forward

Key Objectives

1. Find any missing data
2. Deal with the missing data

INCONSISTENT TEXT & TYPOS

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

Inconsistent text & typos in a data set are represented by values that are either:

- Incorrect by a few digits or characters
- Inconsistent with the rest of a column

```
df.head(8)
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0

These are full state names, while the rest of the column has abbreviations

Finding inconsistent text and typos within a large data set in Python is **not a straightforward approach**, as there is no function that will automatically identify these situations

IDENTIFYING INCONSISTENT TEXT & TYPOS

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

While there is no specific method to **identify inconsistent text & typos**, you can take the following two approaches to check a column depending on its data type:

Categorical data

Look at the unique values in the column

```
df["State"].value_counts()
```

```
OR          2
NJ          1
VA          1
NY          1
AL          1
New York    1
Utah        1
LA          1
TX          1
FL          1
CA          1
Name: State, dtype: int64
```

These represent the same thing!

Numerical data

Look at the descriptive stats of the column

```
df["Age"].describe()
```

```
count     10.000000
mean      48.500000
std       14.370108
min       27.000000
25%       34.750000
50%       54.000000
75%       59.500000
max       62.000000
Name: Age, dtype: float64
```

This looks like a realistic age range

HANDLING INCONSISTENT TEXT & TYPOS

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

You can fix **inconsistent text & typos** by using:

1. **.loc[]** to update a value at a particular location
2. **np.where()** to update values in a column based on a conditional statement
3. **.map()** to map a set of values to another set of values
4. **String methods** like str.lower(), str.strip() & str.replace() to clean text data

We've already covered the **loc[] accessor** to resolve missing data using domain expertise

UPDATE VALUES BASED ON A LOGICAL CONDITION

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

Use **np.where()** to update values based on a logical condition

```
np.where(condition, if_true, if_false)
```

Calls the NumPy library

A logical expression that evaluates to True or False

Value to return when the expression is True

Value to return when the expression is False

This is **different from the Pandas where method**, which has similar functionality, but different syntax

The NumPy function is used more often than the Pandas method because np.where is vectorized, meaning it executes faster

UPDATE VALUES BASED ON A LOGICAL CONDITION

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

Use **np.where()** to update values based on a logical condition

```
df
```

	Name	City	State	Income	Birthdate
0	Susan Rodriguez	Maplewood	NJ	45000.0	1970-09-12
1	Joseph Martinez	Portland	OR	120000.0	1960-12-10
2	Joseph Martinez	Portland	OR	120000.0	1960-12-10
3	Wayne Nielson	Dahlgren	VA	75000.0	1968-12-05
4	Beverly Nixon	Long Island	NY	45000000.0	1962-03-05
5	Veronica Comerford	Tuskegee	AL	50000.0	1993-09-23
6	Ivan Layton	Albany	New York	105000.0	1996-03-12
7	Laura Hailey	Ephraim	Utah	62000.0	1994-01-13

If a State value is equal to 'Utah', then replace it with 'UT',
Otherwise, keep the value that was already in the State column

```
import numpy as np

df.State = np.where(df.State == 'Utah', 'UT', df.State)
df
```

	Name	City	State	Income	Birthdate
0	Susan Rodriguez	Maplewood	NJ	45000.0	1970-09-12
1	Joseph Martinez	Portland	OR	120000.0	1960-12-10
2	Joseph Martinez	Portland	OR	120000.0	1960-12-10
3	Wayne Nielson	Dahlgren	VA	75000.0	1968-12-05
4	Beverly Nixon	Long Island	NY	45000000.0	1962-03-05
5	Veronica Comerford	Tuskegee	AL	50000.0	1993-09-23
6	Ivan Layton	Albany	New York	105000.0	1996-03-12
7	Laura Hailey	Ephraim	UT	62000.0	1994-01-13

MAP VALUES

Use **.map()** to map values from one set of values to another set of values


```
# map multiple state values to one abbreviation
state_mapping = {'AL': 'AL',
                 'Alabama': 'AL',
                 'NY': 'NY',
                 'New York': 'NY'}
```

You can pass a dictionary with existing values as the keys, and new values as the values

```
# use the map method to create a new column
df['State_Clean'] = df.State.map(state_mapping)
df
```

	Name	City	State	Income	Birthdate	State_Clean
4	Beverly Nixon	Long Island	NY	45000000.0	1962-03-05	NY
5	Veronica Comerford	Tuskegee	AL	50000.0	1993-09-23	AL
6	Ivan Layton	Albany	New York	105000.0	1996-03-12	NY

These states were mapped to these state abbreviations

This is similar to creating a **lookup table** in Excel and using VLOOKUP to search for a value in a column of the table and retrieve a corresponding value from another column

PRO TIP: CLEANING TEXT

String methods are commonly used to **clean text** and standardize it for analysis

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
run_times
```

	Name	Run Time	Warm Up Time	Location
0	Alexis	9.2343	3.5	“school”
1	Alexis	10.3842	3.5	School
2	Alexis	8.1209	3 min	“the gym”
3	David	7.2123	2.2	“school”
4	David	6.8342	2	“gym”

There is a lot more you can do to clean text data, which will be covered in the **Natural Language Processing** course

```
# strip away the leading and trailing characters
run_times.Location.str.strip('“”')
```

```
0     school
1     School
2    the gym
3     school
4        gym
Name: Location, dtype: object
```

```
# make everything lowercase
run_times.Location.str.strip('“”').str.lower()
```

```
0     school
1     school
2    the gym
3     school
4        gym
Name: Location, dtype: object
```

```
# replace or remove characters
run_times.Location.str.strip('“”').str.lower().str.replace('the ', '')
```

```
0    school
1    school
2       gym
3    school
4       gym
Name: Location, dtype: object
```

ASSIGNMENT: INCONSISTENT TEXT & TYPOS

NEW MESSAGE

June 14, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Inconsistent Text Check**

Hi again,

Thanks for your help with the missing data yesterday. I like how you decided to handle those missing values.

Can you check the same file for inconsistencies in the text and resolve any issues that you find?

Thanks!
Alan

Reply Forward

Key Objectives

1. Find any inconsistent text and typos
2. Deal with the inconsistent text and typos

DUPLICATE DATA

Duplicate data represents the presence of one or more redundant rows that contain the same information as another, and can therefore be removed

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	NY	105000.0	27.0
7	Laura Hailey	Ephraim	UT	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	FL	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

This is duplicate data on the same customer

These are the same values, but they're not considered duplicate data

IDENTIFYING DUPLICATE DATA

Use **.duplicated()** to identify duplicate rows of data

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	NY	105000.0	27.0
7	Laura Hailey	Ephraim	UT	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	FL	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

```
df.duplicated()
```

```
0     False
1     False
2      True
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
dtype: bool
```

This returns True for every row that is a duplicate of a previous row

```
df.duplicated().sum()
```

```
1
```

*You can use **keep=False** to return all the duplicate rows*

```
df[df.duplicated(keep=False)]
```

	Name	City	State	Income	Age
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0

REMOVING DUPLICATE DATA

Use **.drop_duplicates()** to remove duplicate rows of data

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df.drop_duplicates()
```

You may need to reset the index

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	NY	105000.0	27.0
7	Laura Hailey	Ephraim	UT	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	FL	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

ASSIGNMENT: DUPLICATE DATA

NEW MESSAGE

June 15, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Duplicate Data Check**

Hi again,

I know the last task around finding inconsistent text was a bit tricky. This should be more straightforward!

Can you check the same file for duplicate data and resolve any issues?

Thanks!
Alan

Reply Forward

Key Objectives

1. Find any duplicate data
2. Deal with the duplicate data

OUTLIERS

An **outlier** is a value in a data set that is much bigger or smaller than the others

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Wayne Nielson	Dahlgren	VA	75000.0	54.0
3	Beverly Nixon	Long Island	NY	45000000.0	61.0
4	Veronica Comerford	Tuskegee	AL	50000.0	29.0
5	Ivan Layton	Albany	NY	105000.0	27.0
6	Laura Hailey	Ephraim	UT	62000.0	29.0
7	John Depaul	Baton Rouge	LA	60000.0	55.0
8	James Weiss	Houston	TX	80000.0	NaN
9	Addie Stevenson	Jacksonville	FL	80000.0	NaN
10	Queen Watson	Los Angeles	CA	110000.0	54.0

*Average income (**including** outlier) = **$4.1M***

*Average income (**excluding** outlier) = **$82K***

If outliers are not identified and dealt with, they can have a **notable impact** on calculations and models

IDENTIFYING OUTLIERS

Data Cleaning Overview

Data Types

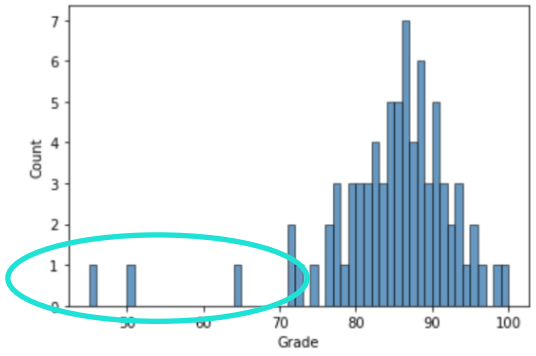
Data Issues

Creating New Columns

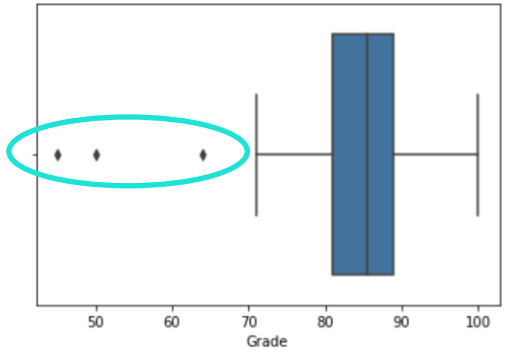
You can **identify outliers** in different ways using plots and statistics

EXAMPLE | *Identifying outliers in student grades from a college class*

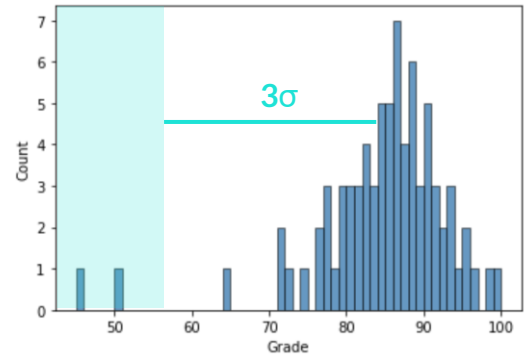
Histogram



Box Plot



Standard Deviation



It's also important to define what you'll consider an outlier in each scenario

Should we flag 3 or 2 outliers?

Use your domain expertise!

HISTOGRAMS

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

Histograms are used to visualize the distribution *(or shape)* of a numerical column

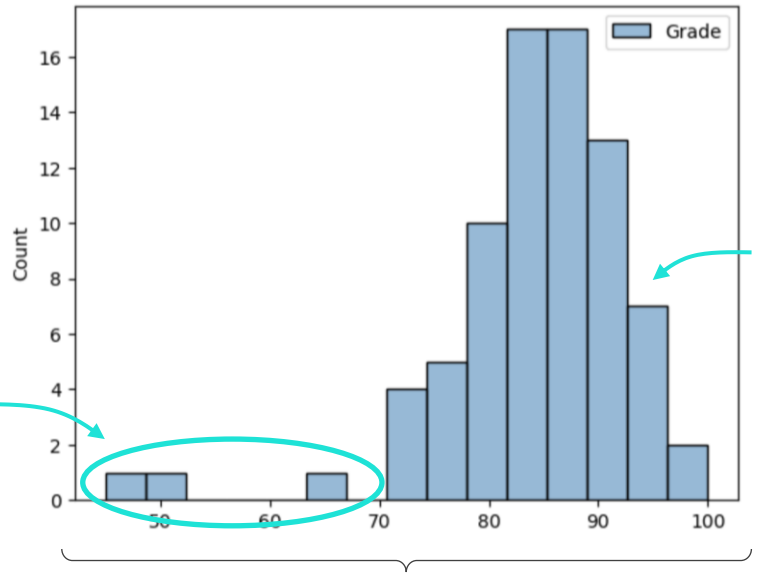
- They help identify outliers by showing which values fall outside of the normal range

```
df = pd.read_csv("student_grades.csv")
df
```

	Student	Grade
0	Emma	86
1	Olivia	86
2	Noah	86
3	Sophia	87
4	Liam	90
...	...	...
73	Zoey	91
74	Aaron	85
75	Charles	93
76	Connor	91
77	Riley	87

*You can create them with the **seaborn** library*

```
import seaborn as sns
sns.histplot(df);
```



These are the potential outliers

The height of each bar is how often the value occurs

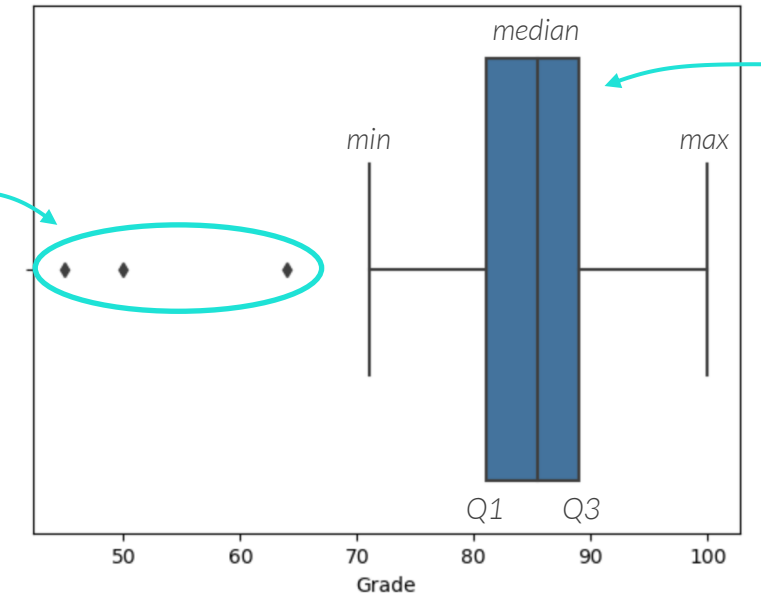
This is the range of values in the data set

BOXPLOTS

Boxplots are used to visualize the descriptive statistics of a numerical column

- They automatically plot outliers as dots outside of the min/max data range


```
sns.boxplot(x=df.Grade);
```



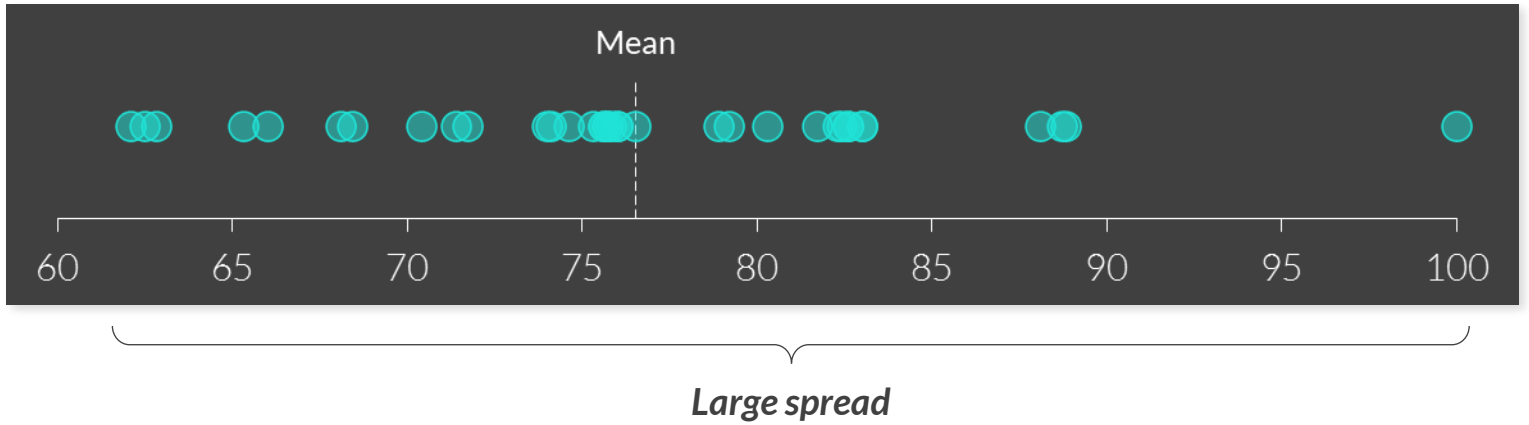
These are the outliers

*The width of the "box" is the **interquartile range** (IQR), which is the middle 50% of the data*

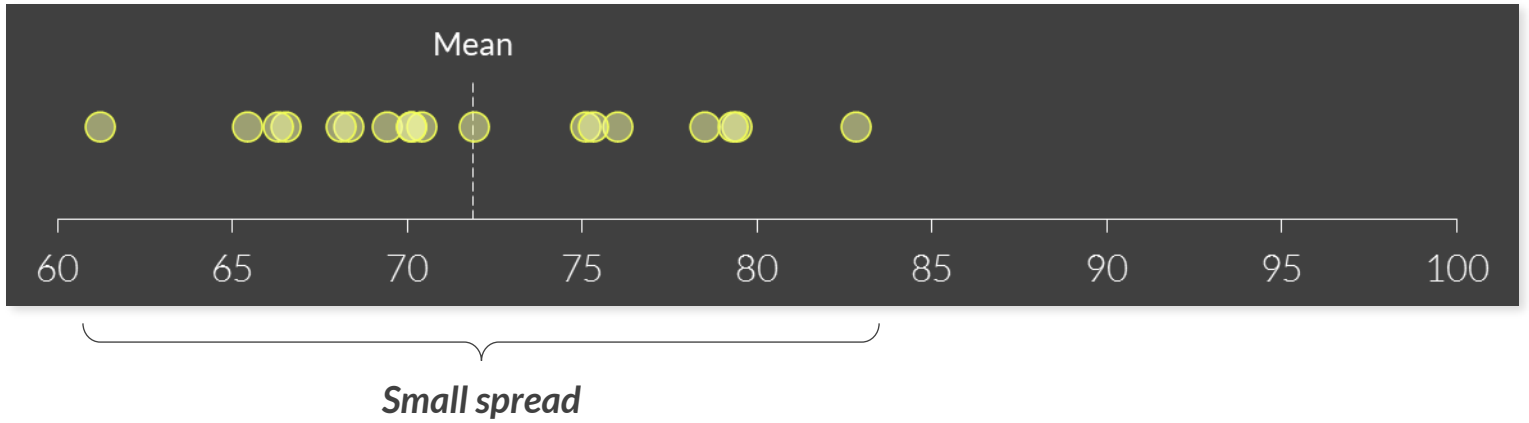
*Any value farther away than 1.5*IQR from each side of the box is considered an outlier*

STANDARD DEVIATION

The **standard deviation** is a measure of the spread of a data set from the mean



Standard Deviation = **8.2**



Standard Deviation = **5.8**

STANDARD DEVIATION

The **standard deviation** is a measure of the spread of a data set from the mean

Values at least 3 standard deviations away from the mean are considered outliers

- This is meant for normally distributed, or bell shaped, data
- The threshold of 3 standard deviations can be changed to 2 or 4+ depending on the data

```
import numpy as np

mean = np.mean(df.Grade)
sd = np.std(df.Grade)

mean, sd
```

```
(84.08987341772152, 8.723725033779411)
```

```
[grade for grade in df.Grade if (grade < mean - 3*sd) or (grade > mean + 3*sd)]
```

```
[50, 45]
```

This returns a list of values, or outliers, at least 3 standard deviations away from the mean

HANDLING OUTLIERS

Like with missing data, there are multiple ways to **handle outliers**:

- Keep outliers
- Remove an entire row or column with outliers
- Impute outliers with NaN or a substitute like the average, mode, max, etc.
- Resolve outliers based on your domain expertise

```
df.head()
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Wayne Nielson	Dahlgren	VA	75000.0	54.0
3	Beverly Nixon	Long Island	NY	45000000.0	61.0
4	Veronica Comerford	Tuskegee	AL	50000.0	29.0

How would you handle this?

ASSIGNMENT: OUTLIERS & REVIEW CLEANED DATA

NEW MESSAGE

June 16, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Outlier Check**

Hi again,

I have one last request for you and then I think our data is clean enough for now.

Can you check the file for outliers and resolve any issues?

Thanks for all your help this week – you rock!

Best,
Alan

Reply Forward

Key Objectives

1. Find any outliers
2. Deal with the outliers
3. Quickly explore the updated DataFrame. How do things look now after handling the data issues compared to the original DataFrame?

CREATING NEW COLUMNS

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

After cleaning data types & issues, you may still not have the exact data that you need, so you can **create new columns** from existing data to aid your analysis

- Numeric columns – calculating percentages, applying conditional calculations, etc.
- Datetime columns – extracting datetime components, applying datetime calculations, etc.
- Text columns – extracting text, splitting into multiple columns, finding patterns, etc.

Name	Cost	Date	Notes
Alexis	$0	4/15/23	Coach: great job!
Alexis	$0	4/22/23	Coach: keep it up
Alexis	$25	5/10/23	PT: add strength training
David	$20	5/1/23	Trainer: longer warm up
David	$20	5/10/23	Trainer: pace yourself

Add 8% tax *Extract the month* *Split into two columns*

Name	Cost + Tax	Month	Person	Note
Alexis	$0	April	Coach	great job!
Alexis	$0	April	Coach	keep it up
Alexis	$27.00	May	PT	add strength training
David	$21.60	May	Trainer	longer warm up
David	$21.60	May	Trainer	pace yourself

Data is ready for further analysis!

CALCULATING PERCENTAGES

To **calculate a percentage**, you can set up two columns with the numerator and denominator values and then divide them *(you can also multiply by 100 if desired)*

EXAMPLE *Finding the percentage of total spend for each item*

```
shopping_list
```

	Category	Item	Price
0	Fruit	Apple	1.29
1	Fruit	Banana	0.40
2	Fruit	Grapes	3.99
3	Vegetable	Carrots	1.50
4	Vegetable	Celery	1.99

*This will be the **numerator***

CALCULATING PERCENTAGES

To **calculate a percentage**, you can set up two columns with the numerator and denominator values and then divide them *(you can also multiply by 100 if desired)*

EXAMPLE *Finding the percentage of total spend for each item*

```
shopping_list
```

```
# calculate the total amount spent
shopping_list['Total Spend'] = shopping_list['Price'].sum()
shopping_list
```

	Category	Item	Price	Total Spend
0	Fruit	Apple	1.29	9.17
1	Fruit	Banana	0.40	9.17
2	Fruit	Grapes	3.99	9.17
3	Vegetable	Carrots	1.50	9.17
4	Vegetable	Celery	1.99	9.17

*This will be the **denominator***

***sum** = 9.17*

CALCULATING PERCENTAGES

To **calculate a percentage**, you can set up two columns with the numerator and denominator values and then divide them *(you can also multiply by 100 if desired)*

EXAMPLE *Finding the percentage of total spend for each item*

```
shopping_list
```

```
# calculate the total amount spent
shopping_list['Total Spend'] = shopping_list['Price'].sum()
shopping_list
```

```
# calculate the percent spent
shopping_list['Percent Spend'] = shopping_list['Price'] / total_spend * 100
shopping_list
```

	Category	Item	Price	Total Spend	Percent Spend
0	Fruit	Apple	1.29	9.17	14.067612
1	Fruit	Banana	0.40	9.17	4.362050
2	Fruit	Grapes	3.99	9.17	43.511450
3	Vegetable	Carrots	1.50	9.17	16.357688
4	Vegetable	Celery	1.99	9.17	21.701200

*These add up to **100%***

CALCULATING BASED ON A CONDITION

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

Use **np.where()** to create a new column based on a logical condition

```
run_times
```

	Name	Run Time	Warm Up Time	Location	Run Date	Race Date	Rain	Fee
0	Alexis	9.2343	3.5	school	2023-04-15	2023-06-01	False	0.0
1	Alexis	10.3842	3.5	school	2023-04-22	2023-06-01	True	0.0
2	Alexis	8.1209	3	gym	2023-05-10	2023-06-01	False	2.5
3	David	7.2123	2.2	school	2023-05-01	2023-06-15	False	0.0
4	David	6.8342	2	gym	2023-05-10	2023-06-15	False	2.5

```
# update all gym fees to include tax
run_times['Fee with Tax'] = np.where(run_times.Location == 'gym', run_times.Fee * 1.08, run_times.Fee)
run_times
```

	Name	Run Time	Warm Up Time	Location	Run Date	Race Date	Rain	Fee	Fee with Tax
0	Alexis	9.2343	3.5	school	2023-04-15	2023-06-01	False	0.0	0.0
1	Alexis	10.3842	3.5	school	2023-04-22	2023-06-01	True	0.0	0.0
2	Alexis	8.1209	3	gym	2023-05-10	2023-06-01	False	2.5	2.7
3	David	7.2123	2.2	school	2023-05-01	2023-06-15	False	0.0	0.0
4	David	6.8342	2	gym	2023-05-10	2023-06-15	False	2.5	2.7

If Location is equal to 'gym', then increase the Fee by 8% in the Fee with Tax column

Otherwise, set it to the existing Fee

ASSIGNMENT: CREATE COLUMNS FROM NUMERIC DATA

NEW MESSAGE

July 5, 2023

From: **Peter Penn** *(Sales Rep)*

Subject: **Pen Sales Data**

Hello,

I've attached the data on our June pen sales.

Can you create two new columns?

- A "Total Spend" column that includes both the pen cost and shipping cost for each sale
- A "Free Shipping" column that says yes if the sale included free shipping, and no otherwise

Thanks!
Peter

Pen Sales Data.xlsx

Reply Forward

Key Objectives

1. Read data into Python
2. Check the data type of each column
3. Create a numeric column using arithmetic
4. Create a numeric column using conditional logic

EXTRACTING DATETIME COMPONENTS

Use **dt.component** to extract a component from a datetime value *(day, month, etc.)*


```
run_times
```

	Run Date	Race Date
0	2023-04-15 12:00:00	2023-06-01
1	2023-04-22 12:30:00	2023-06-01
2	2023-05-10 15:00:00	2023-06-01
3	2023-05-01 15:15:00	2023-06-15
4	2023-05-10 16:30:00	2023-06-15

```
# extract the day from the date
run_times['Run Date'].dt.day
```

```
0    15
1    22
2    10
3     1
4    10
Name: Run Date, dtype: int64
```

```
# extract the day of the week
run_times['Run Date'].dt.dayofweek
```

```
0    5
1    5
2    2
3    0
4    2
Name: Run Date, dtype: int64
```

```
# extract the time
run_times['Run Date'].dt.time
```

```
0    12:00:00
1    12:30:00
2    15:00:00
3    15:15:00
4    16:30:00
Name: Run Date, dtype: object
```

Component	Output
dt.date	Date *(without time component)*
dt.year	Year
dt.month	Numeric month *(1-12)*
dt.day	Day of the month
dt.dayofweek	Numeric weekday *(Mon=0, Sun=6)*
dt.time	Time *(without date component)*
dt.hour	Hour *(0-23)*
dt.minute	Minute *(0-59)*
dt.second	Second *(0-59)*

DATETIME CALCULATIONS

Datetime calculations between columns can be done using basic arithmetic

- Use **pd.to_timedelta()** to add or subtract a particular timeframe

```
# calculate the difference between two dates
run_times['Race Date'] - run_times['Run Date']
```

```
0   46 days 12:00:00
1   39 days 11:30:00
2   21 days 09:00:00
3   44 days 08:45:00
4   35 days 07:30:00
dtype: timedelta64[ns]
```

Note that the data type changed

```
# add two weeks to the race date
run_times['Race Date'] + pd.to_timedelta(2, unit='W')
```

```
0   2023-06-15
1   2023-06-15
2   2023-06-15
3   2023-06-29
4   2023-06-29
Name: Race Date, dtype: datetime64[ns]
```

Time delta units:

- *D = day*
- *W = week*
- *H = hour*
- *T = minute*
- *S = second*

ASSIGNMENT: CREATE COLUMNS FROM DATETIME DATA

NEW MESSAGE

July 12, 2023

From: **Peter Penn** *(Sales Rep)*

Subject: **Delivery Time of Pens?**

Hello again,

Using the data I sent over last week, can you calculate the number of days between the purchase and delivery date for each sale and save it as a new column called “Delivery Time”?

What were the average days from purchase to delivery?

Thanks!
Peter

Reply Forward

Key Objectives

1. Calculate the difference between two datetime columns and save it as a new column
2. Take the average of the new column

EXTRACTING TEXT

You can use **.str[start:end]** to extract characters from a text field

- Note that the position of each character in a string is 0-indexed, and the "end" is non-inclusive

```
run_notes
```

	notes
0	Day 1 - Starting off: Congrats on starting you...
1	Day 2 - Progressing: On your second day, you m...
2	Day 3 - Finding a rhythm: By day 3, you may ha...
3	Day 4 - Pushing yourself: On day 4, you may ha...
4	Day 5 - Consistency: On your final day, you sh...

```
# first 6 characters
run_notes.notes.str[:6]
```

```
0    Day 1
1    Day 2
2    Day 3
3    Day 4
4    Day 5
Name: notes, dtype: object
```

The blank starts from 0 by default

```
# last 11 characters
run_notes.notes.str[-11:]
```

```
0     endurance.
1    Keep it up!
2     your runs.
3     endurance.
4    Keep it up!
Name: notes, dtype: object
```

The negative grabs the "start" from the end of the list, and the blank "end" goes to the end of the text string

SPLITTING INTO MULTIPLE COLUMNS

Use **str.split()** to split a column by a delimiter into multiple columns


```
run_notes
```

	notes
0	Day 1 - Starting off: Congrats on starting you...
1	Day 2 - Progressing: On your second day, you m...
2	Day 3 - Finding a rhythm: By day 3, you may ha...
3	Day 4 - Pushing yourself: On day 4, you may ha...
4	Day 5 - Consistency: On your final day, you sh...

```
# split on the - character
two_fields = run_notes.notes.str.split('-')
two_fields
```

```
0    [Day 1 ,  Starting off: Congrats on starting y...
1    [Day 2 ,  Progressing: On your second day, you...
2    [Day 3 ,  Finding a rhythm: By day 3, you may ...
3    [Day 4 ,  Pushing yourself: On day 4, you may ...
4    [Day 5 ,  Consistency: On your final day, you ...
Name: notes, dtype: object
```

Splitting text returns a list

```
# turn list series into dataframe
pd.DataFrame(two_fields.to_list(),
             columns=['Day', 'Notes'])
```

	Day	Notes
0	Day 1	Starting off: Congrats on starting your runni...
1	Day 2	Progressing: On your second day, you may have...
2	Day 3	Finding a rhythm: By day 3, you may have foun...
3	Day 4	Pushing yourself: On day 4, you may have felt...
4	Day 5	Consistency: On your final day, you should fe...

This is now a DataFrame with two columns

FINDING PATTERNS

Use **str.contains()** to find words or patterns within a text field


```
# create a new column that says whether or not
# the note contains the term 'final'
run_notes['Contains final'] = run_notes.notes.str.contains('final')
run_notes
```

	notes	Contains final
0	Day 1 - Starting off: Congrats on starting you...	False
1	Day 2 - Progressing: On your second day, you m...	False
2	Day 3 - Finding a rhythm: By day 3, you may ha...	False
3	Day 4 - Pushing yourself: On day 4, you may ha...	False
4	Day 5 - Consistency: On your final day, you sh...	True

```
# contains great or congrats
run_notes.notes.str.contains('great|congrats', regex=True)
```

```
0    False
1    False
2     True
3     True
4    False
Name: notes, dtype: bool
```

Regex stands for **regular expression**, which is a way of finding patterns within text (more on this topic will be covered in the Natural Language Processing course)

ASSIGNMENT: CREATE COLUMNS FROM TEXT DATA

NEW MESSAGE

July 19, 2023

From: **Peter Penn** *(Sales Rep)*

Subject: **Pen Reviews**

Hello again,

You may have noticed that the data I sent over a few weeks ago also includes pen reviews.

Can you split the reviews on the "|" character to create two new columns: "User Name" and "Review Text"?

Can you also create a "Leak or Spill" column that flags the reviews that mention either "leak" or "spill"?

Thanks!
Peter

Reply Forward

Key Objectives

1. Split one column into multiple columns
2. Create a Boolean column (True/False) to show whether a text field contains particular words

KEY TAKEAWAYS

Always check that the **data type** for each column matches their intended use

- *Sometimes all columns are read in as objects into a DataFrame, so they may need to be converted into numeric or datetime columns to be able to do any appropriate calculations down the line*

It's important to resolve any **messy data issues** that could impact your analysis

- *When you're looking over a data set for the first time, check for missing data, inconsistent text & typos, duplicate data, and outliers, then be thoughtful and deliberate about how you handle each*

You can **create new columns** based on existing columns in your DataFrame

- *Depending on the data type, you can apply numeric, datetime, or string calculations on existing columns to create new columns that can be useful for your analysis*

You goal should be to make the data **clean enough** for your analysis

- *It's difficult to identify all the issues in your data in order to make it 100% clean (especially if working with text data), so spending extra time trying to do so can be counterproductive – remember the MVP approach!*

EXPLORATORY DATA ANALYSIS

EXPLORATORY DATA ANALYSIS

In this section we'll cover **exploratory data analysis** (EDA), which includes a variety of techniques used to better understand a dataset and discover hidden patterns & insights

TOPICS WE'LL COVER:

- EDA Overview
- Exploring Data
- Visualizing Data
- EDA Tips

GOALS FOR THIS SECTION:

- Learn Python techniques for exploring a new data set, like filtering, sorting, grouping, and visualizing
- Practice finding patterns and drawing insights from data by exploring it from multiple angles
- Understand that while EDA focuses on technical aspects, the goal is to get a good feel for the data

EXPLORATORY DATA ANALYSIS

- EDA Overview
- Exploring Data
- Visualizing Data
- EDA Tips

The **exploratory data analysis** (EDA) phase gives data scientists a chance to:

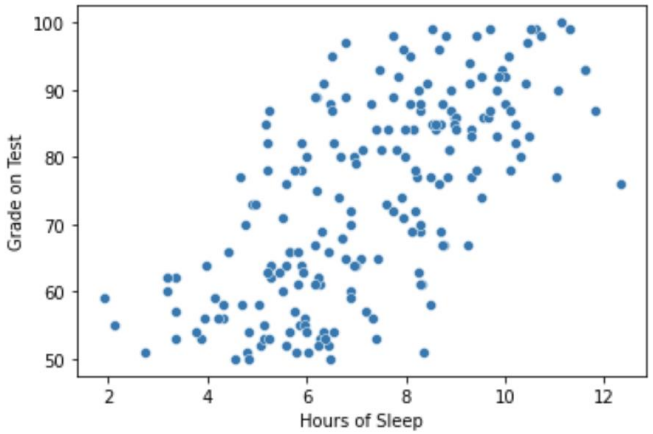
- Get a better sense of the data by viewing it from multiple angles
- Discover patterns, relationships, and insights from the data

From data…

	Hours of Sleep	Hours Studied	Grade on Test
0	10.602667	4.586892	99
1	8.172997	4.405120	72
2	6.430132	-0.519630	52
3	7.963793	5.004348	80
4	8.279421	2.984489	87

EDA

… to insights



Students with more sleep got higher grades!

COMMON EDA TECHNIQUES

These are some **common EDA techniques** used by data scientists:

Exploring data

Viewing & summarizing data from multiple angles

Examples:

- Filtering
- Sorting
- Grouping

Visualizing data

Using charts to identify trends & patterns

Examples:

- Histograms
- Scatterplots
- Pair plots

There is **no particular order** that these tasks need to be completed in; you are free to mix and match them depending on your data set

FILTERING

EDA Overview

Exploring Data

Visualizing Data

EDA Tips

You can filter a DataFrame by **passing a logical test** into the loc[] accessor

- Apply multiple filters by using the "&" and "|" operators *(AND/OR)*

```
tea
```

	type	name	temp
0	green	sencha	180
1	black	chai	210
2	black	ceylon	200
3	herbal	mint	212
4	herbal	ginger	212
5	herbal	chamomile	200
6	herbal	tumeric	175
7	oolong	green oolong	190

```
tea.loc[tea.type == 'herbal']
```

	type	name	temp
3	herbal	mint	212
4	herbal	ginger	212
5	herbal	chamomile	200
6	herbal	tumeric	175

This returns rows where "type" is equal to "herbal"

```
mask = (tea.type == 'herbal') | (tea.temp >= 200)

tea.loc[mask]
```

*Use a **Boolean mask** to apply multiple filters with complex logic*

	type	name	temp
3	herbal	mint	212
4	herbal	ginger	212
5	herbal	chamomile	200
6	herbal	tumeric	175

This returns rows where "type" is equal to "herbal" OR "temp" is greater than or equal to 200

SORTING

EDA Overview

Exploring Data

Visualizing Data

EDA Tips

You can sort a DataFrame by using the **.sort_values()** method

- This sorts in ascending order by default

```
tea
```

	type	name	temp
0	green	sencha	180
1	black	chai	210
2	black	ceylon	200
3	herbal	mint	212
4	herbal	ginger	212
5	herbal	chamomile	200
6	herbal	tumeric	175
7	oolong	green oolong	190

```
tea.sort_values('name')
```

	type	name	temp
2	black	ceylon	- 200
1	black	chai	210
5	herbal	chamomile	200
4	herbal	ginger	212
7	oolong	green oolong	190
3	herbal	mint	212
0	green	sencha	180
6	herbal	tumeric	+ 175

```
tea.sort_values('name', ascending=False)
```

	type	name	temp
6	herbal	tumeric	+ 175
0	green	sencha	180
3	herbal	mint	212
7	oolong	green oolong	190
4	herbal	ginger	212
5	herbal	chamomile	200
1	black	chai	210
2	black	ceylon	- 200

GROUPING

The **"split-apply-combine"** approach is used to group data in a DataFrame and apply calculations to each group

EXAMPLE | *Finding the average temperature for each type of tea*

	type	name	temp
0	green	sencha	180
1	black	chai	210
2	black	ceylon	200
3	herbal	mint	212
4	herbal	ginger	212
5	herbal	chamomile	200
6	herbal	tumeric	175
7	oolong	green oolong	190

***Split** the data by "type"*

0	green	sencha	180	*180*
1	black	chai	210	*205*
2	black	ceylon	200	
3	herbal	mint	212	*199.75*
4	herbal	ginger	212	
5	herbal	chamomile	200	
6	herbal	tumeric	175	
7	oolong	green oolong	190	*190*

***Apply** a calculation (average) on the "temp" in each group*

	type	temp
0	black	205.00
1	green	180.00
2	herbal	199.75
3	oolong	190.00

***Combine** the "type" and average "temp" for each group into a final table*

GROUPING

- EDA Overview
- Exploring Data
- Visualizing Data
- EDA Tips

You can group a DataFrame by using the **.groupby()** method

```
df.groupby(col)[col].aggregation()
```

The DataFrame to group

The column(s) to group by

*(unique values determine the **rows** in the output)*

The column(s) to apply the calculation to

*(these become the new **columns** in the output)*

The calculation(s) to apply for each group

*(these become the new **values** in the output)*

Examples:

- *mean()*
- *sum()*
- *min()*
- *max()*
- *count()*
- *nunique()*

GROUPING

EDA Overview

Exploring Data

Visualizing Data

EDA Tips

You can group a DataFrame by using the **.groupby()** method

```
tea
```

	type	name	temp
0	green	sencha	180
1	black	chai	210
2	black	ceylon	200
3	herbal	mint	212
4	herbal	ginger	212
5	herbal	chamomile	200
6	herbal	tumeric	175
7	oolong	green oolong	190

```
tea.groupby('type')['temp'].mean()
```

```
type
black      205.00
green      180.00
herbal     199.75
oolong     190.00
Name: temp, dtype: float64
```

This returns the average "temp" by "type"

```
tea.groupby('type')['temp'].mean().reset_index()
```

	type	temp
0	black	205.00
1	green	180.00
2	herbal	199.75
3	oolong	190.00

*Use **reset_index()** to return a DataFrame*

MULTIPLE AGGREGATIONS

- EDA Overview
- Exploring Data
- Visualizing Data
- EDA Tips

Chain the **.agg()** method to .groupby() to apply multiple aggregations to each group

```
tea.groupby('type')['temp'].agg(['min', 'max', 'count', 'nunique']).reset_index()
```

	type	min	max	count	nunique
0	black	200	210	2	2
1	green	180	180	1	1
2	herbal	175	212	4	3
3	oolong	190	190	1	1

This returns the minimum & maximum temperatures, the number of teas, and the unique temperatures for each tea type

You can also write the code this way!

```
(tea.groupby('type')['temp']
    .agg(['min', 'max', 'count', 'nunique'])
    .reset_index())
```

PRO TIP: When chaining multiple methods together, wrap the code in parenthesis so you can place each method on a separate line

(this makes reading the code easier!)

PRO TIP: HEAD & TAIL

In addition to aggregating grouped data, you can also use the **.head()** and **.tail()** methods to return the first or last "n" records for each group

```
sales
```

	Name	Date	Sales
0	Anna	3/7/23	9
1	Anna	3/10/23	28
2	Anna	3/13/23	14
3	Anna	3/16/23	8
4	Anna	3/19/23	27
5	Bob	4/1/23	33
6	Bob	4/2/23	9
7	Bob	4/3/23	20
8	Bob	4/4/23	31

Return the first row of each group

```
sales.groupby('Name').head(1)
```

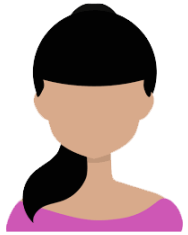
	Name	Date	Sales
0	Anna	3/7/23	9
5	Bob	4/1/23	33

Return the last 3 rows of each group

```
sales.groupby('Name').tail(3)
```

	Name	Date	Sales
2	Anna	3/13/23	14
3	Anna	3/16/23	8
4	Anna	3/19/23	27
6	Bob	4/2/23	9
7	Bob	4/3/23	20
8	Bob	4/4/23	31

ASSIGNMENT: EXPLORING DATA



NEW MESSAGE

August 1, 2023

From: **Anna Analysis** *(Senior Data Scientist)*

Subject: **Happiest Countries of the 2010s?**

Hi,

The marketing team would like to share out the **five happiest countries of the 2010s** on social media.

I've attached a notebook that another data scientist started with happiness data inside. I would recommend:

- Creating a list of each country's highest happiness score, and then sorting it from happiest to least happy country
- Creating a list of each country's average happiness score, and then sorting it from happiest to least happy country

Are there any differences between the two lists?

Thanks!

section06_exploring_data_assignment.ipynb

Reply Forward

Key Objectives

1. Filter out any data before 2010 and after 2019
2. Group the data by country and calculate the maximum happiness score for each one
3. Sort the grouped countries by happiness score and return the top five
4. Group the data by country and calculate the average happiness score for each one
5. Sort the grouped countries by happiness score and return the top five
6. Compare the two lists

VISUALIZING DATA

EDA Overview

Exploring Data

Visualizing Data

EDA Tips

Visualizing data as part of the exploratory data analysis process lets you:

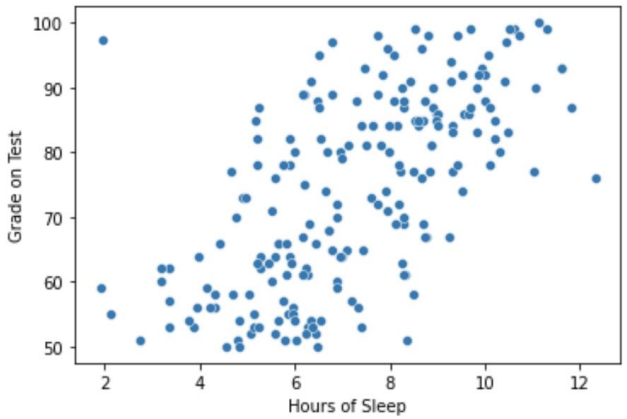
- More easily identify patterns and trends in the data
- Quickly spot anomalies to further investigate

```
student_data[['Hours of Sleep', 'Grade on Test']]
```

	Hours of Sleep	Grade on Test
0	10.602667	99
1	8.172997	72
2	6.430132	52
3	7.963793	80
4	8.279421	87
...	...	...
195	4.155300	59
196	4.306093	58
197	4.686733	58
198	9.407950	98
199	5.989777	54

200 rows × 2 columns

Scatter plot



Students with more sleep got higher grades

One student barely slept but tested very well

Data visualization is also used later in the data science process to communicate insights to stakeholders

DATA VISUALIZATION IN PANDAS

EDA Overview

Exploring Data

Visualizing Data

EDA Tips

You can use the **.plot()** method to create quick and simple visualizations directly from a Pandas DataFrame

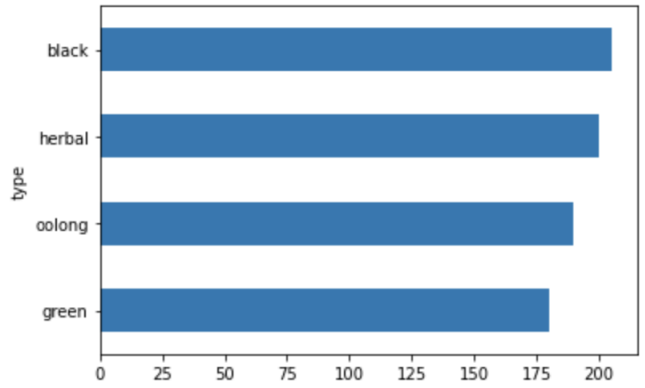
```
# tea temperatures by tea type
tea_temps
```

```
type
green     180.00
oolong    190.00
herbal    199.75
black     205.00
Name: temp, dtype: float64
```

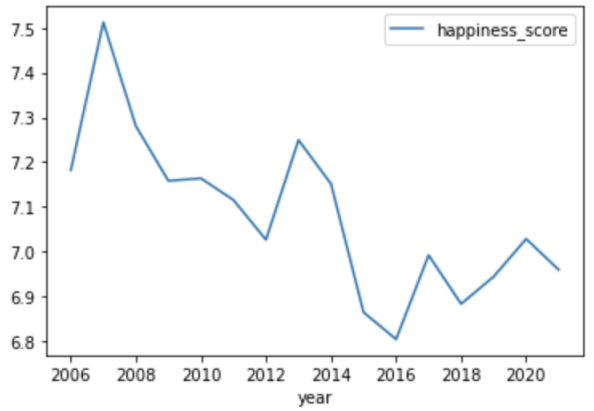
```
# happiness scores for the US
us_happiness.head()
```

	country_name	year	happiness_score
1967	United States	2006	7.181794
1968	United States	2007	7.512688
1969	United States	2008	7.280386
1970	United States	2009	7.158032
1971	United States	2010	7.163616

```
# create a bar chart from a dataframe
tea_temps.plot.barh();
```




```
# create a line chart from a dataframe
us_happiness.plot.line(x='year', y='happiness_score');
```



PRO TIP: PAIR PLOTS

EDA Overview

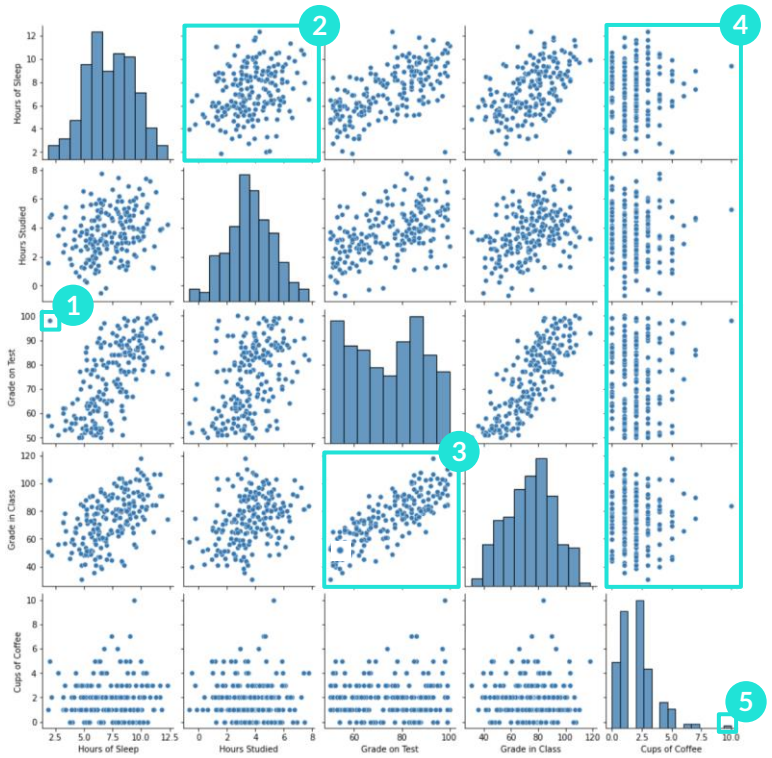
Exploring Data

Visualizing Data

EDA Tips

Use **sns.pairplot()** to create a pair plot that shows all the scatterplots and histograms that can be made using the numeric variables in a DataFrame

```
import seaborn as sns
sns.pairplot(student_data);
```



1. ***Outlier*** – *This student barely studied and still aced the test (look into them)*
2. ***Relationship*** – *The hours spent studying and sleeping don't seem related at all (this is a surprising insight)*
3. ***Relationship*** – *The test grade is highly correlated with the class grade (can ignore one of the two fields for the analysis)*
4. ***Data Type*** – *The cups of coffee field only has integers (keep this in mind)*
5. ***Outlier*** – *This student drinks a lot of coffee (check on them)*

PRO TIP: Create a pair plot as your first visual to identify general patterns that you can dig into later individually

DISTRIBUTIONS

A **distribution** shows all the possible values in a column and how often each occurs

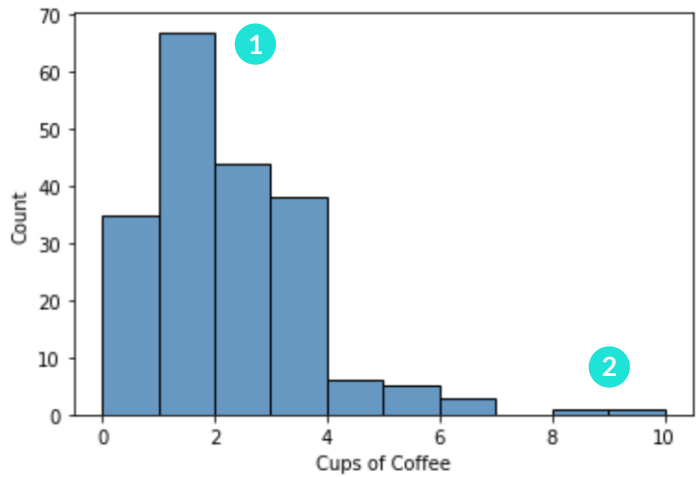
It can be shown in two ways:

Frequency table

	Cups of Coffee
0	35
1	67
2	44
3	38
4	6
5	5
6	3
8	1
10	1

Name: Cups of Coffee

Histogram



1. *Most students drink 1 cup of coffee daily (67)*
2. *A few students drink 8+ cups of coffee daily (2)*

PRO TIP: Distributions can be used to find inconsistencies and outliers

FREQUENCY TABLES

- EDA Overview
- Exploring Data
- Visualizing Data
- EDA Tips

You can create a frequency table with the **.value_counts()** method

```
student_data['Cups of Coffee']
```

```
0       5
1       3
2       4
3       1
4       0
       ..
195     2
196     1
197     1
198    10
199     2
Name: Cups of Coffee, Length: 200, dtype: int64
```

```
(student_data['Cups of Coffee'].value_counts()
                               .sort_index())
```

```
0     35
1     67
2     44
3     38
4      6
5      5
6      3
8      1
10     1
Name: Cups of Coffee, dtype: int64
```

HISTOGRAMS

EDA Overview

Exploring Data

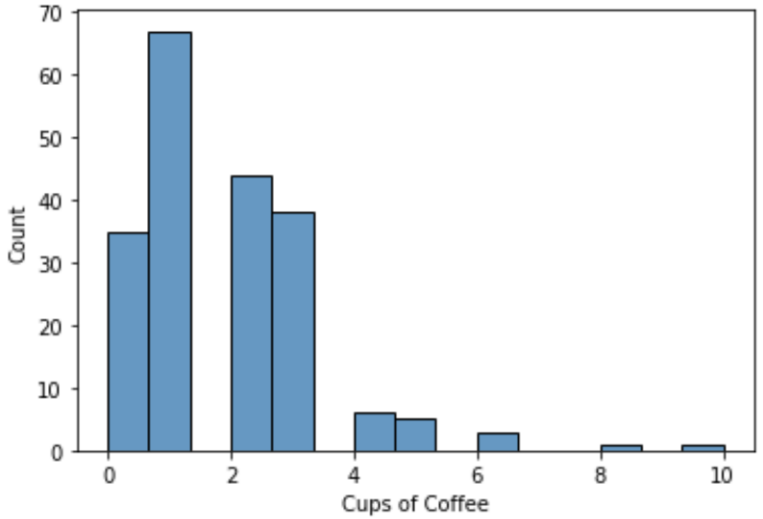
Visualizing Data

EDA Tips

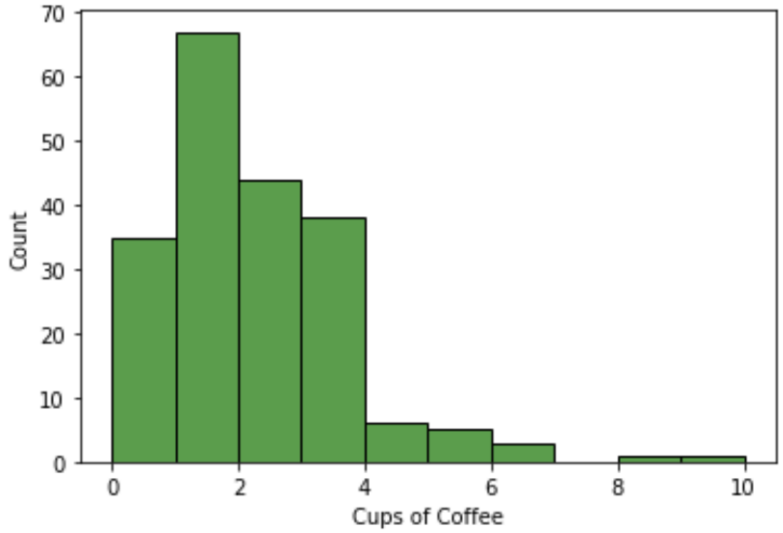
You can create a histogram by passing a DataFrame column to **sns.histplot()**

```
sns.histplot(student_data['Cups of Coffee'])
```

```
<AxesSubplot:xlabel='Cups of Coffee', ylabel='Count'>
```

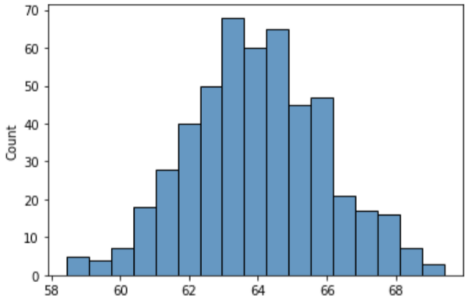



```
sns.histplot(student_data['Cups of Coffee'],
             bins=10, color='green');
```



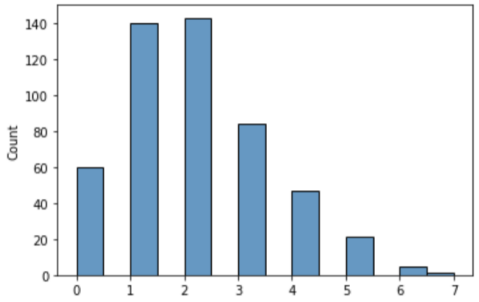
COMMON DISTRIBUTIONS

These are some **common distributions** you'll encounter as a data scientist



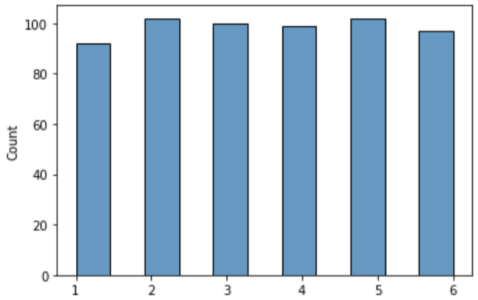
Normal distribution

If you collect the height of 500 women, most will be ~64" with fewer being much shorter or taller than that



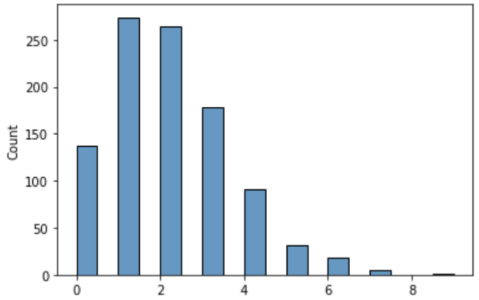
Binomial distribution

Knowing that 10% of all ad views result in a click, these are the clicks you'll get if you show an ad to 20 customers



Uniform distribution

If you roll a die 500 times, the chances of getting any of the 6 numbers is the same



Poisson distribution

Knowing that you typically get 2 cancellations each day, these are the number of cancellations you will see on any given day

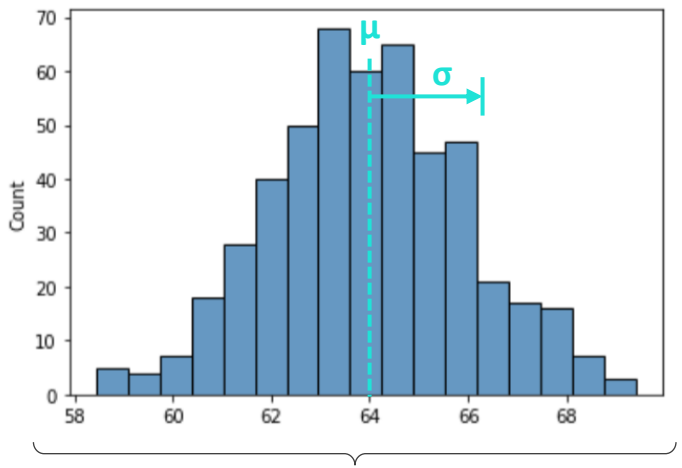
While it's not necessary to memorize the formulas for each distribution, being able to **recognize the shapes and data types** for these distributions will be helpful for future data science modeling and analysis

NORMAL DISTRIBUTION

Many numeric variables naturally follow a **normal distribution**, or "bell curve"

- Normal distributions are described by two values: the **mean (μ)** & **standard deviation (σ)**
- The standard deviation measures, on average, how far each value lies from the mean

EXAMPLE | *Women's Heights (in)*



A sample of 500 women shows a mean height of 5'4" (64 inches) and a standard deviation of 2.2 inches

The **empirical rule** outlines where most values fall in a normal distribution:

- **68%** fall within **1σ** from the mean
- **95%** fall within **2σ** from the mean
- **99.7%** fall within **3σ** from the mean

(this is why data points over 3σ away from the mean are considered outliers)

SKEW

EDA Overview

Exploring Data

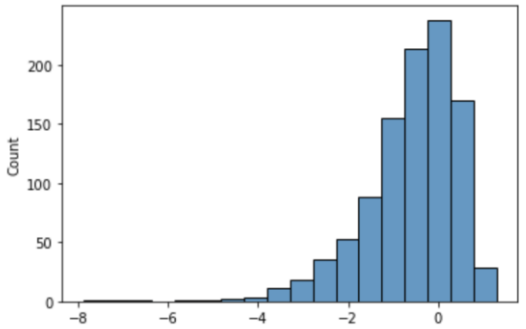
Visualizing Data

EDA Tips

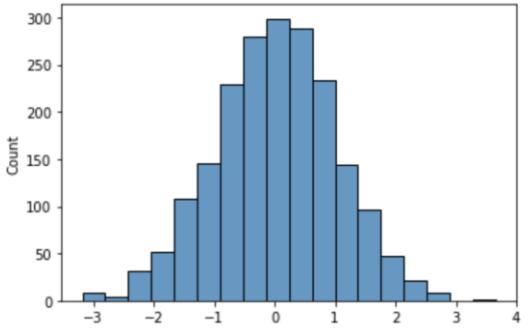
The **skew** represents the asymmetry of a normal distribution around its mean

- For example, data on household income is typically right skewed

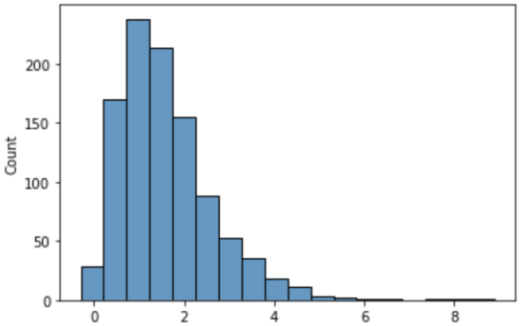
Left skew



Normal Distribution



Right skew



There are **techniques that can deal with skewed data**, such as taking the log of the data set, to turn it into normally distributed data; more on this will be discussed in the prep for modeling section

ASSIGNMENT: DISTRIBUTIONS

NEW MESSAGE

August 8, 2023

From: **Sarah Song** *(Music Analysis Team)*

Subject: **Musical Attribute Distributions**

Hi,

Our music analysis team has labeled 100 songs with their musical attributes, such as its danceability and energy.

Can you tell us more about the distributions of the musical attributes? I'm guessing that since most of the musical attributes are numeric, they should be normally distributed, but let me know if you see otherwise.

Thanks!

section06_visualizing_data_assignments.ipynb

Reply Forward

Key Objectives

1. Import the seaborn library
2. Create a pair plot
3. Interpret the histograms
    - Which fields are normally distributed?
    - Which fields have other distributions?
    - Do you see any skew?
    - Do you see any outliers?
    - Any other observations?
    - Do your interpretations make sense?

SCATTERPLOTS

EDA Overview

Exploring Data

Visualizing Data

EDA Tips

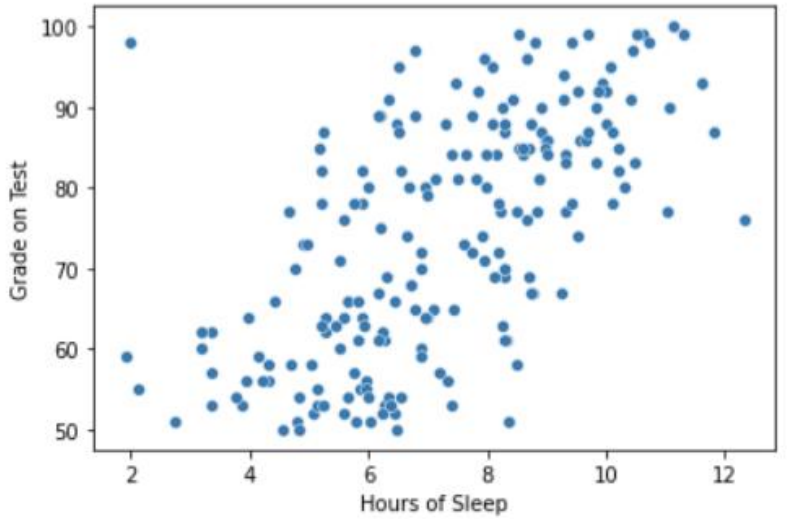
Scatterplots are used to visualize the relationship between numerical variables

- `sns.scatterplot(data=df, x="x axis column", y="y axis column")`

```
student_data.head(3)
```

	Hours of Sleep	Hours Studied	Grade on Test
0	10.602667	4.586892	99
1	8.172997	4.405120	72
2	6.430132	-0.519630	52

```
sns.scatterplot(data=student_data,
                x='Hours of Sleep',
                y='Grade on Test');
```

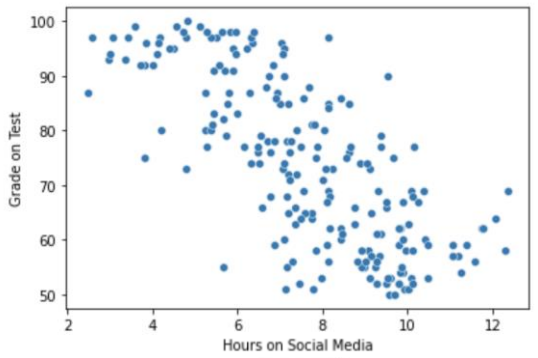


CORRELATION

A **correlation** describes the relationship between two numerical columns *(-1 to 1)*

- **-1** is a perfect negative correlation, **0** is no correlation, and **1** is a perfect positive correlation

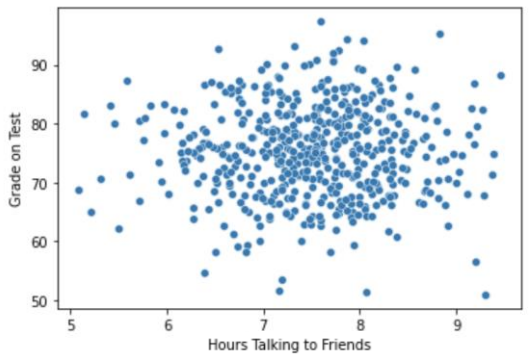
Correlation = –0.7



*This is a **negative correlation***

(Students who spent more time on social media got worse grades)

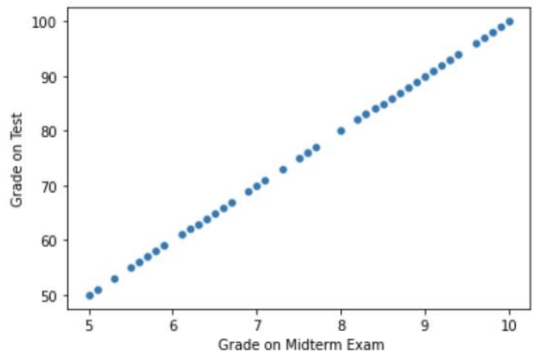
Correlation = 0



*This has **no correlation***

(No relationship exists between hours talking with friends and test grades)

Correlation = 1



*This is a **perfect positive correlation***

(This is likely an error and the grades are exact copies of each other)

Correlation does not imply causation! Just because two variables are related does not necessarily mean that changes to one cause changes to the other

CORRELATION

EDA Overview

Exploring Data

Visualizing Data

EDA Tips

Use the **.corr()** method to calculate the correlation coefficient between each pair of numerical variables in a DataFrame

- A correlation under 0.5 is weak, between 0.5 and 0.8 is moderate, and over 0.8 is strong

```
student_data.corr()
```

	Hours of Sleep	Hours Studied	Grade on Test	Grade in Class	Cups of Coffee
Hours of Sleep	1.000000	0.265404	0.637864	0.502397	-0.079643
Hours Studied	0.265404	1.000000	0.479153	0.374456	-0.020209
Grade on Test	0.637864	0.479153	1.000000	0.815721	-0.038804
Grade in Class	0.502397	0.374456	0.815721	1.000000	-0.035288
Cups of Coffee	-0.079643	-0.020209	-0.038804	-0.035288	1.000000

1. *The correlation between a variable with itself will always be 1 – **you can ignore these values across the diagonal***
2. *While this looks like a negative correlation, this value is very close to zero, meaning that hours studied and cups of coffee are **uncorrelated***
3. *The only strong correlation in this table is that the grade on the test is **positively correlated** with the grade in the class (keep this in mind for future modeling or insights)*

ASSIGNMENT: CORRELATIONS

NEW MESSAGE

August 9, 2023

From: **Sarah Song** *(Music Analysis Team)*

Subject: **Musical Attribute Relationships**

Hi again,

Thanks for looking into those distributions earlier. Can you also tell us more about the correlations of the musical attributes?

Can you tell us about any relationships between the attributes, like if some are positively or negatively correlated?

Thanks!

section06_visualizing_data_assignments.ipynb

Reply Forward

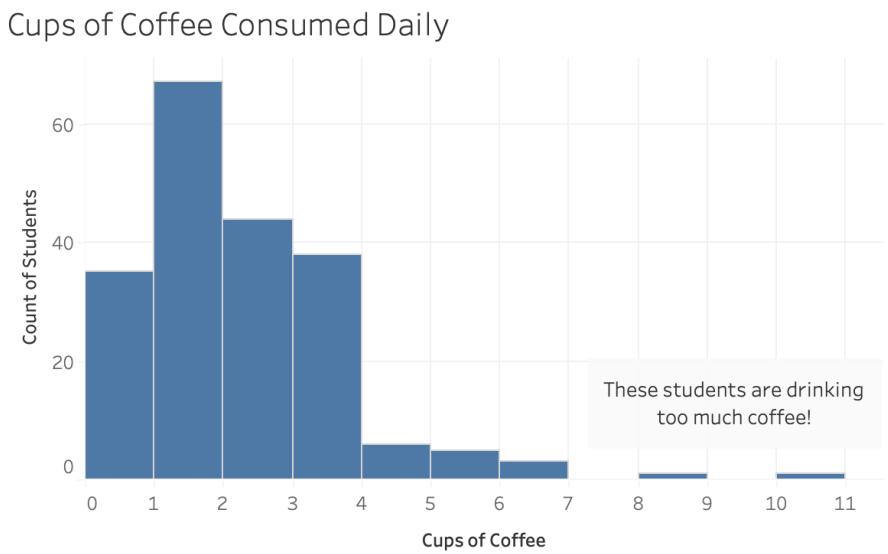
Key Objectives

1. Look at the previously created pair plot
2. Interpret the scatterplots
   - Which fields are highly correlated?
   - Which fields are uncorrelated?
   - Which fields have positive or negative correlations?
   - Any other observations?
   - Do your interpretations make sense?

DATA VISUALIZATION IN PRACTICE

Data visualizations can be **difficult to tweak and polish in Python**, so feel free to export the data as a CSV file and import it into another tool *(Excel, Tableau, etc.)*

```
student_data.to_csv('student_data.csv')
```



PRO TIP: Before sharing a visual, take a moment to think about what you want your audience to take away from it

If possible, modify, remove or highlight specific parts emphasize your points

EDA TIPS

EDA Overview

Exploring Data

Visualizing Data

EDA Tips

Before diving into EDA:

- Remind yourself of the original question(s) that you're trying to answer

As you go through EDA:

- Keep a running list of observations and questions for both yourself and your client
- Apply the techniques in any order that makes the most sense for your data
- You may have to go back to earlier steps and gather more data or further clean your data as you discover more about your data during EDA

You know you've completed your initial EDA once you've:

- Investigated any initial idea or question that comes to mind about the data and gathered some meaningful insights *(you can always come back to EDA after modeling!)*

Working with a large data set?
Look at a subset, apply EDA techniques, then extrapolate to the whole data set

Already answered your question?
Sometimes EDA is all you need, and you may not need to apply any algorithms

KEY TAKEAWAYS

EDA is all about looking at and **exploring data** from multiple angles

- *You can get a better understanding of your data just from filtering, sorting and grouping your data in Python*

It's often helpful to **visualize data** to more easily see patterns in the data

- *One of the first plots that data scientists create to visualize their data is a pair plot, which includes scatter plots for looking at correlations and histograms for looking at distributions*

By exploring and visualizing data, you'll start to **discover insights**

- *These insights can be saved for the end of the project to share with stakeholders, or they can be used as context when preparing the data for modeling*

MID-COURSE PROJECT

MID-COURSE PROJECT: MOVIE RATINGS

NEW MESSAGE

June 1, 2023

From: **Katniss Potter** *(Podcast Host)*

Subject: **Movie Ratings Exploration**

Hi there,

I'm the host of a movie reviews podcast and I'm currently making an episode about movie review aggregators.

I found this data set from Rotten Tomatoes (inside the .ipynb file that I've attached). Could you dig into the data and share any interesting insights that you find? My audience loves fun facts about movies.

Thank you!
KP

section07_midcourse_project.ipynb

Reply Forward

Key Objectives

1. Explore the data by filtering, sorting, and grouping the data
2. Create new columns to aid in analysis
3. Visualize the data
4. Interpret the aggregations & plots and share interesting insights

(detailed steps and questions in the Jupyter Notebook)

PREPARING FOR MODELING

PREPARING FOR MODELING

In this section we'll learn to **prepare data for modeling**, including merging data into a single table, finding the right row granularity for analysis, and engineering new features

TOPICS WE'LL COVER:

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

We will **NOT** cover any modeling techniques in depth
(these will be taught in the rest of the courses in this series)

GOALS FOR THIS SECTION:

- Switch from an exploratory to a modeling mindset
- Become familiar with key modeling terms
- Understand the data structures required as inputs for machine learning models
- Learn common feature engineering techniques

CASE STUDY: PREPARING FOR MODELING

MAVEN MEGA MART

You've just been hired as a Data Science Intern for **Maven Mega Mart**, a large ecommerce website that sells everything from clothing to pet supplies

From: **Candice Canine** *(Senior Data Scientist)*

Subject: **Email Targeting**

Hi!

We're launching a new dog food brand and would like to send out an email blast to a subset of customers that are most likely to purchase dog food.

Can you help me prep our data so I can identify those customers?

You'll need to:

1. Create a single table with the appropriate row & column formats
2. Engineer features

Reply Forward

Scope: Identify customers that are most likely to purchase dog food

Technique: Supervised learning

Label (y): Whether a customer has purchased dog food recently or not

Features (x):

- What other items a customer has purchased
- How much money a customer has spent
- etc.

DATA PREP: EDA VS MODELING

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

So far, we've gathered and cleaned data to **prepare for EDA**, but a few additional steps are required to **prepare for modeling**

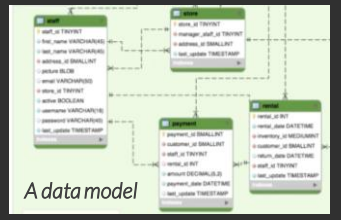
Data ready for EDA

Address	City	Color	Price ($)
200 N Michigan	Chicago	Blue	350,000
10 S State	Chicago	Blue	500,000
123 Main St	Evanston	White	180,000
50 Dempster	Evanston	Yellow	270,000

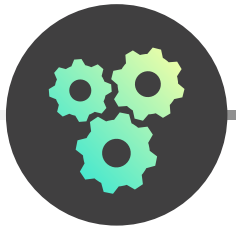
Data ready for modeling

Index	Walk Score	Blue	White	Price ($)
0	97	1	0	350,000
1	92	1	0	500,000
2	70	0	1	180,000
3	64	0	0	270,000

Here, "modeling" refers to **applying an algorithm**, which is different from "data modeling", where the goal is to visually show the relationship between the tables in a relational database



A data model



PREPARING FOR MODELING

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

To **prepare for modeling** means to transform the data into a structure and format that can be used as a direct input for a machine learning algorithm

You can prepare your data for modeling by:

1. Creating a **single table**
2. Setting the correct **row granularity**
3. Ensuring each **column** is non-null and numeric
4. **Engineering features** for the model

CREATING A SINGLE TABLE

There are two ways to combine multiple tables into a **single table**:

- **Appending** stacks the rows from multiple tables with the same column structure
- **Joining** adds related columns from one tables to another, based on common values

date	store	sales
2022-05-01	1	341
2022-05-01	2	291
2022-05-01	3	493
2022-05-01	4	428
2022-05-01	5	152

date	store	sales
2022-06-01	1	67
2022-06-01	2	144
2022-06-01	3	226
2022-06-01	4	397
2022-06-01	5	163

date	store	sales
2022-05-01	1	341
2022-05-01	2	291
2022-05-01	3	493
2022-05-01	4	428
2022-05-01	5	152
2022-06-01	1	67
2022-06-01	2	144
2022-06-01	3	226
2022-06-01	4	397
2022-06-01	5	163

Appending *these two tables with the same columns added the rows from one to the other*

date	store	sales
2022-05-01	1	341
2022-05-01	2	291
2022-05-01	3	493
2022-05-01	4	428
2022-05-01	5	152

store	region
1	North
2	North
3	East
4	East
5	West

date	store	sales	region
2022-05-01	1	341	North
2022-05-01	2	291	North
2022-05-01	3	493	East
2022-05-01	4	428	East
2022-05-01	5	152	West

Joining *these two tables added the region column based on the matching store values*

APPENDING

Use **pd.concat()** to append, or vertically stack, multiple DataFrames

- The columns for the DataFrames must be identical
- **pd.concat([df_1, df_2])** will stack the rows from "df_2" at the bottom of "df_1"

`sales_may`

	date	store	sales
0	2022-05-01	1	341
1	2022-05-01	2	291
2	2022-05-01	3	493

`sales_june`

	date	store	sales
0	2022-06-01	1	67
1	2022-06-01	2	144
2	2022-06-01	3	226

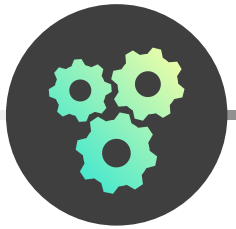
pd.concat(*[df1, df2, df3, ...]*)

```
pd.concat([sales_may, sales_june])
```

	date	store	sales
0	2022-05-01	1	341
1	2022-05-01	2	291
2	2022-05-01	3	493
0	2022-06-01	1	67
1	2022-06-01	2	144
2	2022-06-01	3	226

*Chain **.reset_index()** to the code to make the index go from 0 to 5*

PRO TIP: You can also use **.concat()** to combine DataFrames horizontally by setting axis = 1



JOINING

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

Use **.merge()** to join two DataFrames based on common values in a column(s)

- The DataFrames must have at least one column with matching values
- This is different from the Pandas .join() method, which joins DataFrames on their indices

```
left_df.merge(right_df,
              how,
              left_on,
              right_on)
```

"Right" DataFrame to join with "Left"

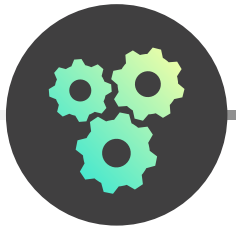
Type of join

Column(s) in "Left" DataFrame to join by

"Left" DataFrame to join with "Right"

Column(s) in "Right" DataFrame to join by

For a full list of arguments, visit: ***https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html***



JOINING

Use **.merge()** to join two DataFrames based on common values in a column(s)

- The DataFrames must have at least one column with matching values
- This is different from the Pandas .join() method, which joins DataFrames on their indices

```
sales_may
```

	date	store	sales
0	2022-05-01	1	341
1	2022-05-01	2	291
2	2022-05-01	3	493
3	2022-05-01	4	428
4	2022-05-01	5	152

```
regions
```

	store	region
0	2	North
1	3	East
2	4	West

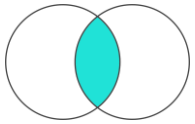
```
sales_may.merge(regions,
                how='left',
                left_on='store',
                right_on='store')
```

	date	store	sales	region
0	2022-05-01	1	341	NaN
1	2022-05-01	2	291	North
2	2022-05-01	3	493	East
3	2022-05-01	4	428	West
4	2022-05-01	5	152	NaN

This added the region from the "regions" table to the "sales" table based on the "store" field

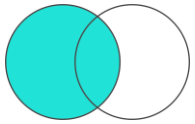
TYPES OF JOINS

These are the most common **types of joins** you can use with .merge()



how = 'inner'

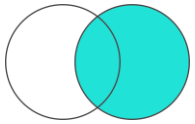
Returns records that exist in BOTH tables, and excludes unmatched records from either table



how = 'left'

Returns ALL records from the LEFT table, and any matching records from the RIGHT table

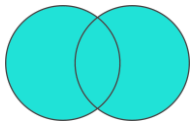
Most commonly used joins



how = 'right'

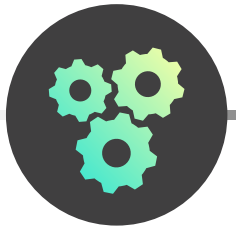
Returns ALL records from the RIGHT table, and any matching records from the LEFT table

Rarely used – in practice, switch the tables and use a left join instead



how = 'outer'

Returns ALL records from BOTH tables, including non-matching records



TYPES OF JOINS

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

Left Table

date	store	offer	sales
2022-05-01	1	1	341
2022-05-01	2		291
2022-05-01	3	1	493
2022-05-01	4	2	428
2022-05-01	5		152
2022-06-01	1	3	67
2022-06-01	2		144
2022-06-01	3	1	226
2022-06-01	4		397
2022-06-01	5	4	163

n=10

Right Table

offer	discount
1	5%
2	10%
3	15%
4	20%
5	50%

n=5

Left Join

date	store	offer	sales	discount
2022-05-01	1	1	341	5%
2022-05-01	2		291	
2022-05-01	3	1	493	5%
2022-05-01	4	2	428	10%
2022-05-01	5		152	
2022-06-01	1	3	67	15%
2022-06-01	2		144	
2022-06-01	3	1	226	5%
2022-06-01	4		397	
2022-06-01	5	4	163	20%

n=10

Inner Join

date	store	offer	sales	discount
2022-05-01	1	1	341	5%
2022-05-01	3	1	493	5%
2022-05-01	4	2	428	10%
2022-06-01	1	3	67	15%
2022-06-01	3	1	226	5%
2022-06-01	5	4	163	20%

n=6

Outer Join

date	store	offer	sales	discount
2022-05-01	1	1	341	5%
2022-05-01	2		291	
2022-05-01	3	1	493	5%
2022-05-01	4	2	428	10%
2022-05-01	5		152	
2022-06-01	1	3	67	15%
2022-06-01	2		144	
2022-06-01	3	1	226	5%
2022-06-01	4		397	
2022-06-01	5	4	163	20%
		5		50%

n=11

ASSIGNMENT: CREATING A SINGLE TABLE

NEW MESSAGE

July 3, 2023

From: **Brooke Reeder** *(Owner, Maven Books)*

Subject: **Combine data sets**

Hi there,

We just finished collecting our Q2 book sales data. Can you help us create one giant table that includes:

- April, May and June's book sales
- Customer data

The *customer_id* field links all the tables together.

Thanks!
Brooke

Book_Sales_April.xlsx, Book_Sales_May.xlsx, Book_Sales_June.xlsx, Book_Customers.csv

Reply Forward

Key Objectives

1. Read all four files into a Jupyter Notebook
2. Append the May and June book sales to the April DataFrame
3. Join the newly created book sales DataFrame with the customers DataFrame on *customer_id*
    - Which type of join would work best here?

PREPARING ROWS FOR MODELING

To **prepare rows for modeling**, you need to think about the question you're trying to answer and determine what one row (observation) of your table will look like

- In other words, you need to determine the *granularity* of each row

GOAL | *Predict which customers are most likely to buy dog food in June*

	customer	item_id	purchase_date	item_description	price	category	rating
0	Ava	1011	2023-04-01	Paint	15.99	Arts & Crafts	3.5
1	Ava	1014	2023-04-01	Brush	1.99	Arts & Crafts	4.2
2	Ava	1015	2023-04-15	Paper	22.49	Arts & Crafts	4.5
3	Ava	1018	2023-05-01	Scissors	3.50	Arts & Crafts	4.6
4	Ben	2345	2023-04-15	Dog Food	29.99	Pet Supplies	4.9
5	Ben	2300	2023-04-20	Dog Leash	14.20	Pet Supplies	3.2
6	Ben	2345	2023-06-15	Dog Food	29.99	Pet Supplies	4.9
7	Chloe	3811	2023-05-01	Socks	7.50	Apparel	3.7
8	Chloe	3814	2023-05-01	Shirt	9.99	Apparel	4.1
9	Chloe	3828	2023-05-01	Shorts	12.47	Apparel	4.3
10	Chloe	1012	2023-05-02	Crayons	2.87	Arts & Crafts	4.7
11	Chloe	1018	2023-05-04	Scissors	3.50	Arts & Crafts	4.6
12	Chloe	2345	2023-06-06	Dog Food	29.99	Pet Supplies	4.9

Because we are predicting something for a customer, one row of data in table should represent ***one customer***

Group by customer

Number of pet supplies purchased before June

How much was spent on all items before June

	customer	Apparel	Arts & Crafts	Pet Supplies	Total Spend	Bought Dog Food in June?
0	Ava	0	4	0	43.97	0
1	Ben	0	0	2	44.19	1
2	Chloe	3	2	0	36.33	1

x variables
(features to be input into a model)

y variable
(label / output)

ASSIGNMENT: PREPARE ROWS FOR MODELING

NEW MESSAGE

July 5, 2023

From: **Brooke Reeder** *(Owner, Maven Books)*

Subject: **Please format data for analysis**

Hi again,

We're trying to predict which customers will purchase a book this month.

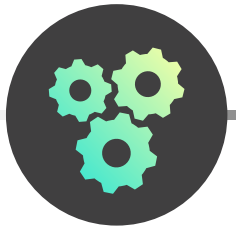
Can you reformat the data you compiled earlier this week so that it's ready to be input into a model, with each row representing a customer instead of a purchase?

Thanks!
Brooke

Reply Forward

Key Objectives

1. Determine the row granularity needed
2. Create a column called "June Purchases" that sums all purchases in June
3. Create a column called "Total Spend" that sums the prices of the books purchased in April & May
4. Combine the "June Purchases" and "Total Spend" columns into a single DataFrame for modeling

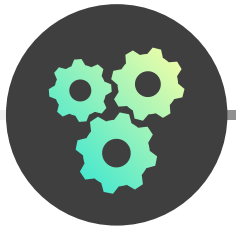


PREPARING COLUMNS FOR MODELING

Once you have the data in a single table with the right row granularity, you'll move on to **preparing the columns for modeling**:

1. All values should be **non-null**
   - Use df.info() or df.isna() to identify null values and either remove them, impute them, or resolve them based on your domain expertise
2. All values should be **numeric**
   - Turn text fields to numeric fields using dummy variables
   - Turn datetime fields to numeric fields using datetime calculations

PRO TIP: There are some algorithms that can handle null and non-numeric values, including tree-based models and some classification models, but it is still best practice to prepare the data this way



DUMMY VARIABLES

A **dummy variable** is a field that only contains zeros and ones to represent the presence (1) or absence (0) of a value, also known as one-hot encoding

- They are used to transform a categorical field into multiple numeric fields

House ID	Price	Color
1	$350,000	Blue
2	$500,000	Blue
3	$180,000	White
4	$270,000	Yellow
5	$245,000	White

*These dummy variables are **numeric representations** of the "Color" field*

House ID	Price	Color	Blue	White	Yellow
1	$350,000	Blue	1	0	0
2	$500,000	Blue	1	0	0
3	$180,000	White	0	1	0
4	$270,000	Yellow	0	0	1
5	$245,000	White	0	1	0

It only takes 2 columns to distinguish 3 values, so you might sometimes see one less column

DUMMY VARIABLES

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

Use **pd.get_dummies()** to create dummy variables in Python

```
houses
```

	House ID	Price	Color
0	1	350000	Blue
1	2	500000	Blue
2	3	180000	White
3	4	270000	Yellow
4	5	245000	White

```
# get dummy variables for all non-numeric fields
pd.get_dummies(houses)
```

	House ID	Price	Color_Blue	Color_White	Color_Yellow
0	1	350000	1	0	0
1	2	500000	1	0	0
2	3	180000	0	1	0
3	4	270000	0	0	1
4	5	245000	0	1	0

```
# drop one of the dummy variable columns
pd.get_dummies(houses, drop_first=True)
```

	House ID	Price	Color_White	Color_Yellow
0	1	350000	0	0
1	2	500000	0	0
2	3	180000	1	0
3	4	270000	0	1
4	5	245000	1	0

*Set **drop_first=True** to remove one of the dummy variable columns*

(it does not matter which column removed, as long as one is dropped)

PRO TIP: PREPARING DATETIME COLUMNS

To **prepare datetime columns** for modeling they need to be converted to numeric columns, but extracting datetime components isn't enough

	customer	purchase_date	month
0	Ava	2023-04-01	4
1	Ava	2023-04-01	4
2	Ava	2023-04-15	4
3	Ava	2023-05-01	5

The "month" field is not an appropriate numeric input because a model would interpret May (5) as being better than April (4)

Instead, you can prepare them using:

Dummy variables

	4	5	6
customer			
Aiden	4	4	1
Ava	3	1	0
Ben	2	0	1

Number of purchases by month

The days from "today"

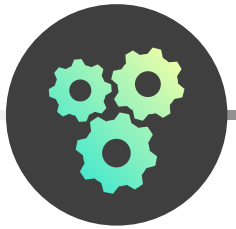
	days_passed
customer	
Aiden	16
Ava	50
Ben	5

Days since the latest purchase

The average time between dates

	days_between
customer	
Aiden	7.625
Ava	10.000
Ben	30.500

Average days between purchases



PRO TIP: PREPARING DATETIME COLUMNS

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

EXAMPLE

*Using **dummy variables** to prepare a date column for modeling*

`purchases`

	customer	purchase_date	month
0	Ava	2023-04-01	4
1	Ava	2023-04-01	4
2	Ava	2023-04-15	4
3	Ava	2023-05-01	5
4	Ben	2023-04-15	4
...	...	...	...
105	Jenny	2023-04-20	4
106	Jenny	2023-04-20	4
107	Jenny	2023-04-20	4
108	Jenny	2023-04-20	4
109	Jenny	2023-04-20	4

110 rows × 3 columns

```
# create dummy variables and join with customers
month_dummies = pd.get_dummies(purchases.month)

monthly_purchases = pd.concat([purchases.customer, month_dummies], axis=1)
monthly_purchases.head(3)
```

	customer	4	5	6
0	Ava	1	0	0
1	Ava	1	0	0
2	Ava	1	0	0

```
# group by customer and sum the purchases by month
monthly_purchases.groupby('customer').sum().head()
```

	4	5	6
customer			
Aiden	4	4	1
Ava	3	1	0
Ben	2	0	1
Bennett	5	0	0
Blake	0	1	1

PRO TIP: PREPARING DATETIME COLUMNS

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- **Preparing Columns**
- Feature Engineering
- Preview: Modeling

EXAMPLE | *Using the **days from "today"** to prepare a date column for modeling*

`purchases`

	customer	purchase_date
0	Ava	2023-04-01
1	Ava	2023-04-01
2	Ava	2023-04-15
3	Ava	2023-05-01
4	Ben	2023-04-15
...	...	...
105	Jenny	2023-04-20
106	Jenny	2023-04-20
107	Jenny	2023-04-20
108	Jenny	2023-04-20
109	Jenny	2023-04-20

110 rows × 2 columns

```
# group by customer to get their latest purchase date
last_purchase = purchases.groupby('customer').max()
last_purchase.head(2)
```

customer	purchase_date
Aiden	2023-06-04
Ava	2023-05-01

```
# assume today is the max purchase date
today = purchases.purchase_date.max()
today
```

```
Timestamp('2023-06-20 00:00:00')
```

```
# calculate the days passed between today and their latest purchase
last_purchase['days_passed'] = (today - last_purchase.purchase_date).dt.days
last_purchase.head(3)
```

customer	purchase_date	days_passed
Aiden	2023-06-04	16
Ava	2023-05-01	50
Ben	2023-06-15	5

PRO TIP: PREPARING DATETIME COLUMNS

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

EXAMPLE

*Using the **average time between dates** to prepare a date column for modeling*

```
purchases
```

	customer	purchase_date
0	Ava	2023-04-01
1	Ava	2023-04-01
2	Ava	2023-04-15
3	Ava	2023-05-01
4	Ben	2023-04-15
...	...	...
105	Jenny	2023-04-20
106	Jenny	2023-04-20
107	Jenny	2023-04-20
108	Jenny	2023-04-20
109	Jenny	2023-04-20

110 rows × 2 columns

```
purchases['days_between'] = purchases.groupby('customer').diff(1)
purchases.head()
```

	customer	purchase_date	days_between
0	Ava	2023-04-01	NaT
1	Ava	2023-04-01	0 days
2	Ava	2023-04-15	14 days
3	Ava	2023-05-01	16 days
4	Ben	2023-04-15	NaT

```
purchases['days_between'] = purchases['days_between'].dt.days
purchases.groupby('customer')['days_between'].mean().head()
```

	customer	days_between
0	Aiden	7.625
1	Ava	10.000
2	Ben	30.500
3	Bennett	0.750
4	Blake	31.000

A variation of **.diff()** is **.shift()**, which will shift all the rows of a column up or down

ASSIGNMENT: PREPARE COLUMNS FOR MODELING

NEW MESSAGE

July 7, 2023

From: **Brooke Reeder** *(Owner, Maven Books)*

Subject: **Please help make data numeric**

Hi again,

Thanks for your help earlier this week!

We just learned that we also have to make all the data numeric before inputting it into a predictive model.

Can you turn the "Audience" text field into a numeric field?

Thanks!
Brooke

Reply Forward

Key Objectives

1. Create dummy variables from the "Audience" field
2. Using the "Audience" dummy variables, create three new columns that contain the number of Adult, Children, and Teen books purchased by each customer
3. Combine the three new columns back with the customer-level data

FEATURE ENGINEERING

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

Feature engineering is the process of creating columns that you think will be helpful inputs for improving a model (*help predict, segment, etc.)*

- When preparing rows & columns for modeling you're already feature engineering!

```
original_data.head()
```

	customer	item_id	purchase_date	item_description	price	category	rating
0	Ava	1011	4/1/23	Paint	$15.99	Arts & Crafts	3.5
1	Ava	1014	4/1/23	Brush	$1.99	Arts & Crafts	4.2
2	Ava	1015	4/15/23	Paper	$22.49	Arts & Crafts	4.5
3	Ava	1018	5/1/23	Scissors	$3.50	Arts & Crafts	4.6
4	Ben	2345	4/15/23	Dog Food	$29.99	Pet Supplies	4.9

```
model_input.head()
```

	customer	total_spend	Pet Supplies	days_since_purchase	june_purchases
0	Aiden	222.16	8	13.0	1.0
1	Ben	44.19	2	42.0	1.0
2	Blake	25.55	1	22.0	1.0
3	Calvin	29.99	1	16.0	1.0
4	Chloe	36.33	0	28.0	1.0

Other feature engineering techniques:

- Transformations *(log transform)*
- Scaling *(normalization & standardization)*
- Proxy variables

Once you have your data in a single table and have prepared the rows & columns, it's ready for modeling

However, being deliberate about **engineering new features** can be the difference between a good model and a great one

TRANSFORMATIONS: LOG TRANSFORMS

Transformations require mapping a set of values to another in a consistent way

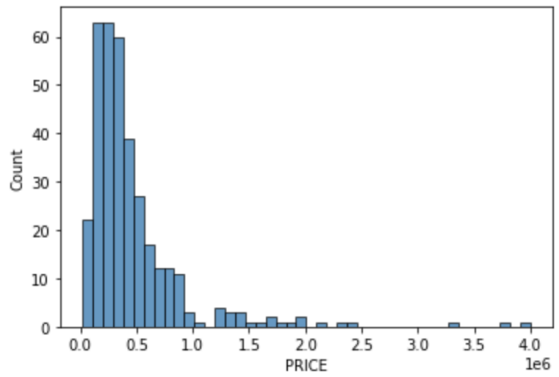
- **Log transforms** turn skewed data into more normally-distributed data
- You can use the **np.log()** function to apply a log transform to a Series

Right-skewed data

```
house_prices
```

```
0       399000
1        99900
2       539000
3       299000
4       320000
         ...
338     649000
339     265000
340      72000
341     245000
342     190000
```

```
# house prices histogram
sns.histplot(house_prices);
```

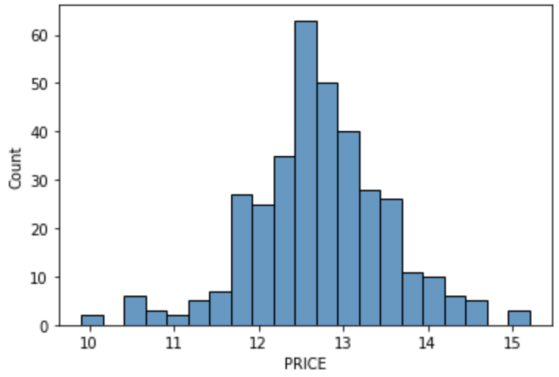


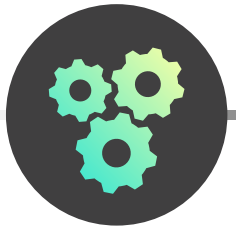
Normally-distributed data

```
# log of house prices
np.log(house_prices)
```

```
0       12.896717
1       11.511925
2       13.197471
3       12.608199
4       12.676076
          ...
338     13.383188
339     12.487485
340     11.184421
341     12.409013
342     12.154779
```

```
# log of house prices histogram
sns.histplot(np.log(house_prices));
```





SCALING: NORMALIZATION

Scaling, as its name implies, requires setting all input features on a similar scale

- **Normalization** transforms all values to be between 0 and 1 (or between -1 and 1)

```
houses
```

	PRICE	BEDS
0	399000	3.0
1	99900	2.0
2	539000	3.0
3	299000	2.0
4	320000	2.0
5	699900	4.0
6	295000	3.0
7	245900	1.0
8	480000	5.0
9	375000	5.0
10	770000	3.0

```
(houses - houses.min()) / (houses.max() - houses.min())
```

	PRICE	BEDS
0	0.446351	0.50
1	0.000000	0.25
2	0.655275	0.50
3	0.297120	0.25
4	0.328458	0.25
5	0.895389	0.75
6	0.291151	0.50
7	0.217878	0.00
8	0.567229	1.00
9	0.410536	1.00
10	1.000000	0.50

*You can also use the **.MinMaxScaler()** function from the **sklearn** library*

Normalization equation

$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

PRO TIP: Normalization is typically used when the distribution of the data is unknown

SCALING: STANDARDIZATION

Scaling, as its name implies, requires setting all input features on a similar scale

- **Normalization** transforms all values to be between 0 and 1 (or between -1 and 1)
- **Standardization** transforms all values to have a mean of 0 and standard deviation of 1

```
houses
```

	PRICE	BEDS
0	399000	3.0
1	99900	2.0
2	539000	3.0
3	299000	2.0
4	320000	2.0
5	699900	4.0
6	295000	3.0
7	245900	1.0
8	480000	5.0
9	375000	5.0
10	770000	3.0

```
(houses - houses.mean()) / houses.std()
```

	PRICE	BEDS
0	-0.061292	0.000000
1	-1.569570	-0.790569
2	0.644689	0.000000
3	-0.565564	-0.790569
4	-0.459667	-0.790569
5	1.456063	0.790569
6	-0.585735	0.000000
7	-0.833333	-1.581139
8	0.347168	1.581139
9	-0.182317	1.581139
10	1.809558	0.000000

*You can also use the **.StandardScaler()** function from the **sklearn** library*

Standardization equation

$$\frac{x - x_{mean}}{x_{std}}$$

PRO TIP: Standardization is typically used when the distribution is normal (bell curve)

PROXY VARIABLES

A **proxy variable** is a feature meant to approximately represent another

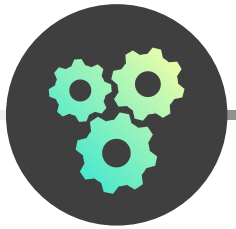
- They are used when a feature is either difficult to gather or engineer into a new feature

Zip codes may look numeric, but should not be input into a model (60202 is not better than 60201)

	ZIP CODE	PRICE	BEDS	MEDIAN INCOME
0	60202	399000	3.0	$92,433
1	60201	99900	2.0	$81,128
2	60201	539000	3.0	$81,128
3	60201	299000	2.0	$81,128
4	60202	320000	2.0	$92,433
5	60201	699900	4.0	$81,128
6	60201	295000	3.0	$81,128
7	60201	245900	1.0	$81,128
8	60201	480000	5.0	$81,128
9	60202	375000	5.0	$92,433

*Instead of turning the zip code into dummy variables, you can use a **proxy variable** like the median income for the zip code, or its distance from the city center*

You may not be able to engineer proxy variables from existing data, but they can be gathered from external sources



FEATURE ENGINEERING TIPS

1. Anyone can apply an algorithm, but only someone with **domain expertise** can engineer relevant features, which is what makes a great model

2. You want your data to be long, not wide (**many rows, few columns**)

3. If you're working with customer data, a popular marketing technique is to engineer features related to the **recency**, **frequency**, and **monetary value** (RFM) of a customer's transactions

4. Once you start modeling, you're bound to find things you missed during data prep and will **continue to engineer features** and gather, clean, explore, and visualize the data

ASSIGNMENT: FEATURE ENGINEERING

NEW MESSAGE

July 10, 2023

From: **Brooke Reeder** *(Owner, Maven Books)*

Subject: **Please create new features**

Hi again,

I have one final request for you.

As a reminder, our goal is to try and predict which customers will purchase a book this month.

Can you create new features that you think will do a good job making a prediction?

Thanks!
Brooke

Reply Forward

Key Objectives

1. Brainstorm features that would make good predictors for a model
2. Engineer two new features
3. Add them to the non-null, numeric DataFrame that is ready for modeling

PREVIEW: APPLYING ALGORITHMS

- Data Prep for Modeling
- Creating a Single Table
- Preparing Rows
- Preparing Columns
- Feature Engineering
- Preview: Modeling

You can input the prepared data into a supervised learning model to predict which customers are most likely to purchase dog food

x *y*

	Customer	Apparel	Arts & Crafts	Pet Supplies	Total Spend	Bought Dog Food in June?
0	Aiden	0	0	8	222.16	1
1	Bennett	0	5	0	27.73	0
2	Blake	0	0	1	25.55	1
3	Calvin	0	0	1	29.99	1
4	Daniel	0	0	0	17.46	0
5	Evelyn	6	0	0	66.19	0
6	Gavin	2	2	0	39.47	1
7	Henry	2	2	0	112.42	1
8	Isabel	0	0	0	2.79	1
9	Jenny	0	6	0	49.34	0
10	Kate	2	1	0	83.25	0
11	Lia	2	0	1	78.95	3
12	Lily	0	0	4	69.31	1
13	Madeline	6	1	0	122.63	0
14	Margaret	0	0	1	7.99	1
15	Maxwell	1	0	0	78.31	1
16	Nolan	1	2	0	67.51	1
17	Olivia	1	1	2	68.03	2
18	Sophie	0	0	0	2.57	0

```
# import the logistic regression algo
from sklearn.linear_model import LogisticRegression

# input the data into the algo
lr = LogisticRegression().fit(X, y)

# use the algo to make a prediction
lr.predict_proba(X_test)
```

x test

	Customer	Apparel	Arts & Crafts	Pet Supplies	Total Spend	Prediction
0	Xavier	0	4	0	43.97	0.391533
1	Yvonne	0	0	2	44.19	0.954803
2	Zev	3	2	0	36.33	0.144423

Whether a customer purchased pet supplies in recently is a good predictor of whether they will buy dog food

Yvonne is the most likely to buy dog food

KEY TAKEAWAYS

Preparing data for EDA is different than **preparing data for modeling**

- *The goal of EDA is exploration while the goal of modeling is to use an algorithm to answer a question, so to prep for modeling means to get the data in a format that can be directly input into a model*

All data needs to be in a **single table** with **non-null**, **numeric values** for modeling

- *Join together multiple tables with .merge and .concat, remove null values with .isna, create dummy variables to turn text into numeric values and use datetime calculations to turn datetimes into numeric values*

Engineering features turns good models into great models

- *Use the techniques learned and intuition built during EDA to create meaningful features for modeling as well as feature transformation, feature scaling and proxy variables*

FINAL PROJECT

RECAP: THE COURSE PROJECT

THE SITUATION

You've just been hired as a Jr. Data Scientist for **Maven Music**, a streaming service that's been losing more customers than usual the past few months and would like to use data science to figure out how to reduce customer churn

THE ASSIGNMENT

You'll have access to data on Maven Music's customers, including subscription details and music listening history

Your task is to **gather, clean, and explore the data** to provide insights about the recent customer churn issues, then **prepare it for modeling** in the future

THE OBJECTIVES

1. **Scope** the data science project
2. **Gather** the data in Python
3. **Clean** the data
4. **Explore** & visualize the data
5. **Prepare** the data for modeling

MAVEN MUSIC

SINCE 1996

FINAL PROJECT: MAVEN MUSIC

NEW MESSAGE

June 8, 2023

From: **Carter Careswell** *(Customer Care)*

Subject: **Customer Churn Data Prep**

Hi again,

Thanks for scoping the customer churn project with us earlier. We've identified the location of the data sets for the project that we talked about – customer and listening history.

Can you read the data into Python, clean it, explore it and prepare it for modeling? More details can be found in the attached Jupyter Notebook.

Thanks for all your help!
Carter

section09_final_project.ipynb

Reply | Forward

Key Objectives

1. Revisit the project scope
2. Read the data files into Python
3. Clean the data by converting data types, resolving data issues, and creating new columns
4. Explore the data independently, then join the tables for further exploration
5. Create a non-null, numeric DataFrame and engineer features that could be good predictors of customer churn
6. Visualize and interpret the data in the final DataFrame that is ready for modeling