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:

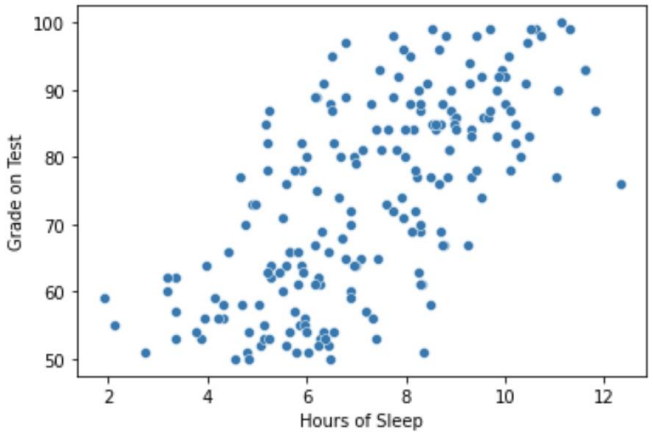
- Get a better sense of the data by viewing it from multiple angles
- Discover patterns, relationships, and insights from the data

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)
```

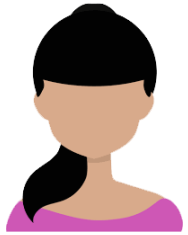
	Name	Date	Sales
0	Anna	3/7/23	9
5	Bob	4/1/23	33

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:

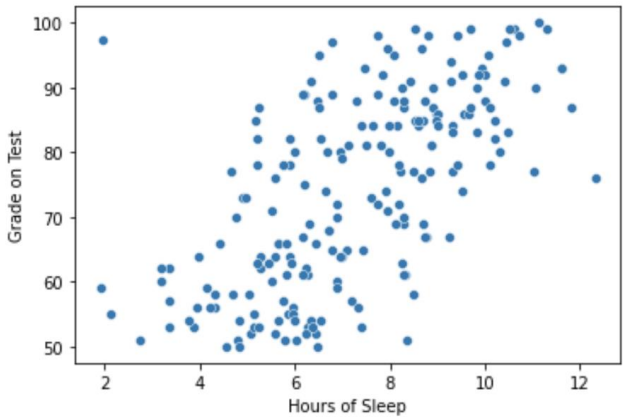
- More easily identify patterns and trends in the data
- Quickly spot anomalies to further investigate

```
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

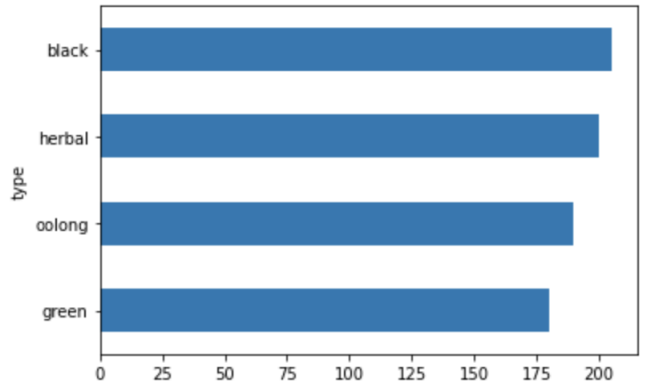
```
# tea temperatures by tea type
tea_temps
```

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

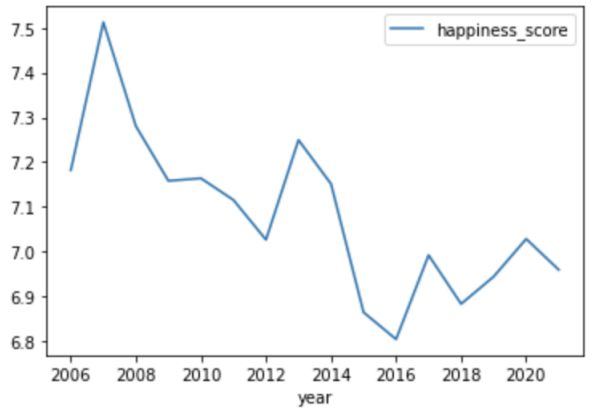
```
# happiness scores for the US
us_happiness.head()
```

	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

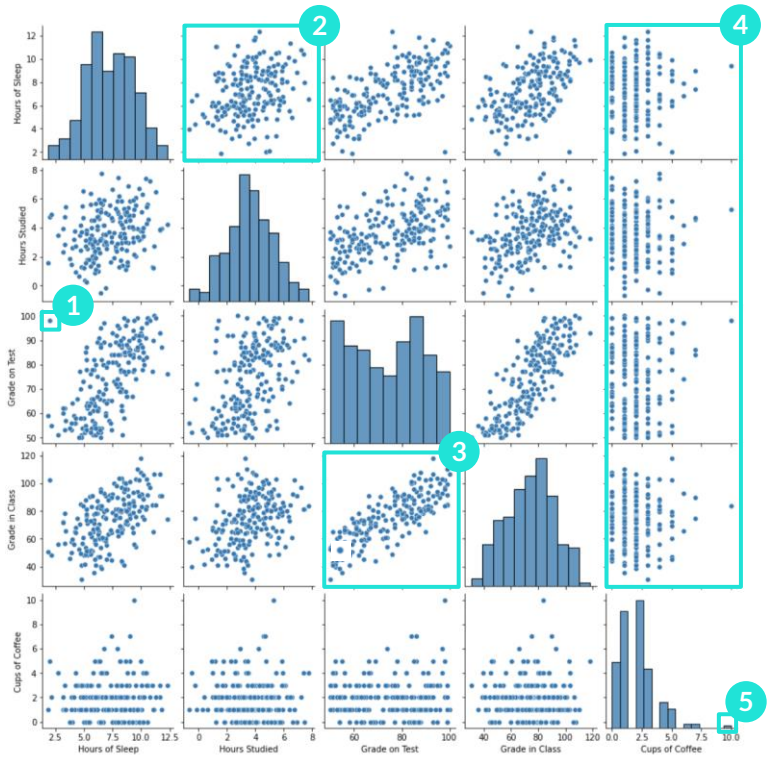
Exploring Data

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

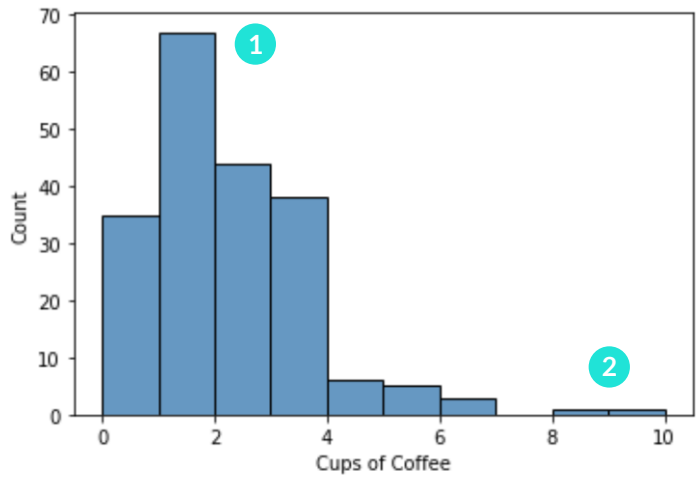
It can be shown in two ways:

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

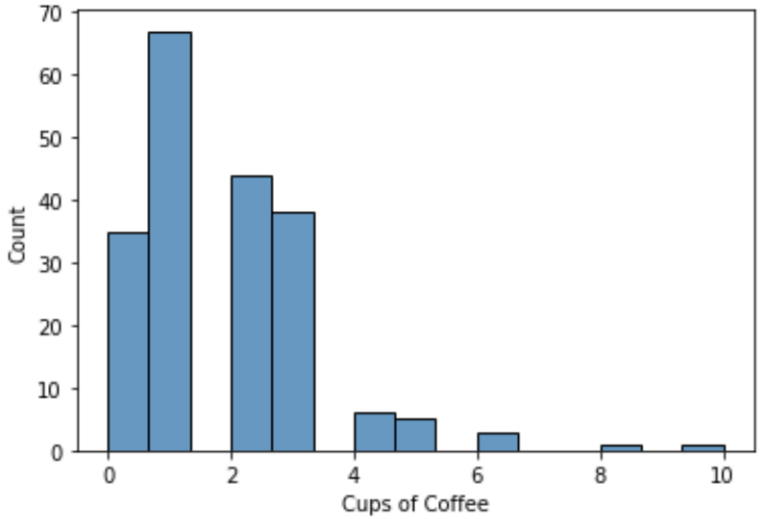
Visualizing Data

EDA Tips

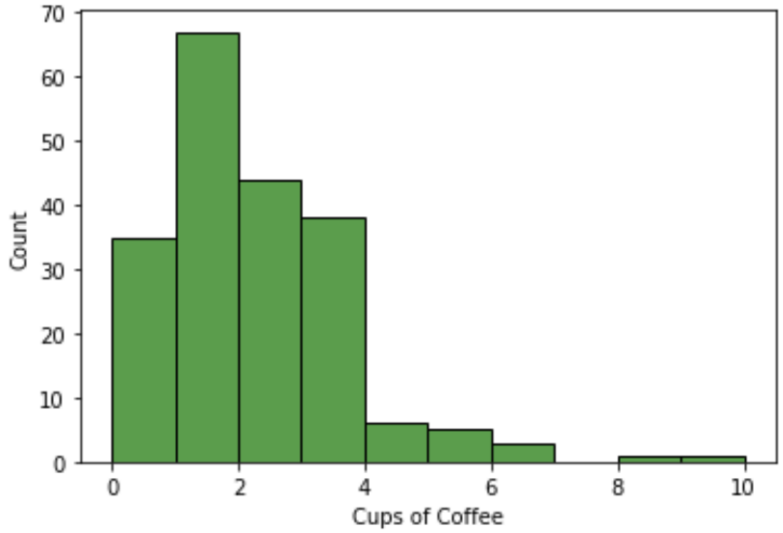
You can create a histogram by passing a DataFrame column to **sns.histplot()**

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

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

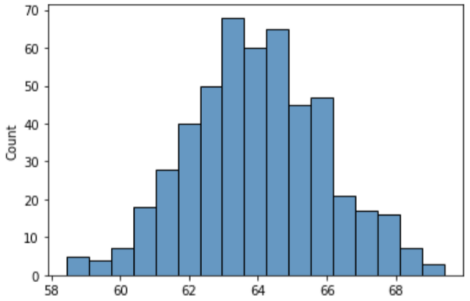



```
sns.histplot(student_data['Cups of Coffee'],
             bins=10, color='green');
```



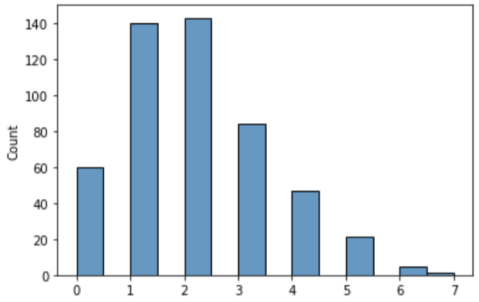
COMMON DISTRIBUTIONS

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



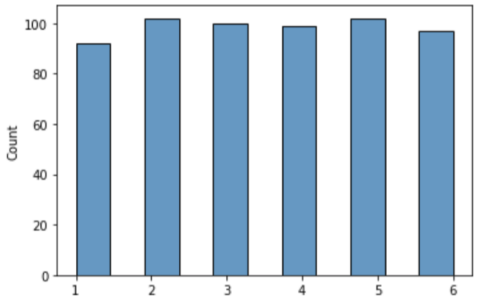
Normal distribution

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



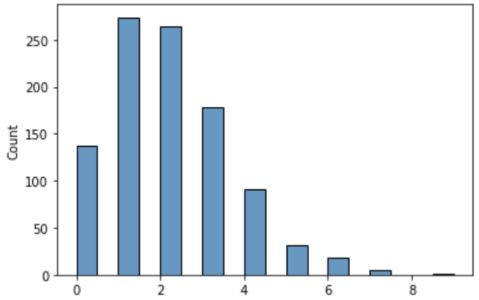
Binomial distribution

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

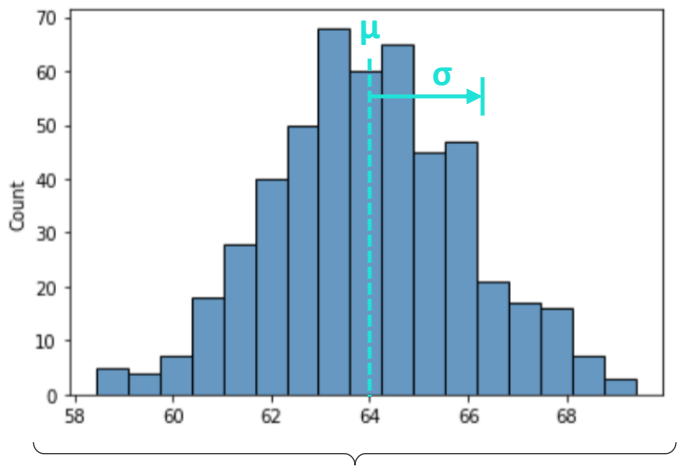
While it's not necessary to memorize the formulas for each distribution, being able to **recognize the shapes and data types** for these distributions will be helpful for future data science modeling and analysis

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

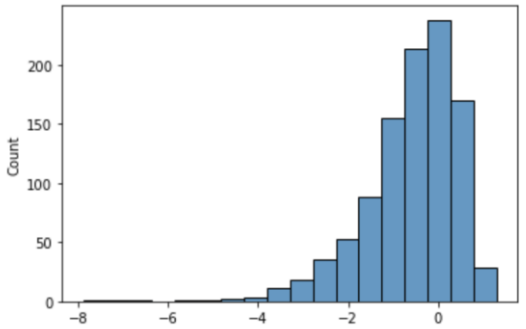
Visualizing Data

EDA Tips

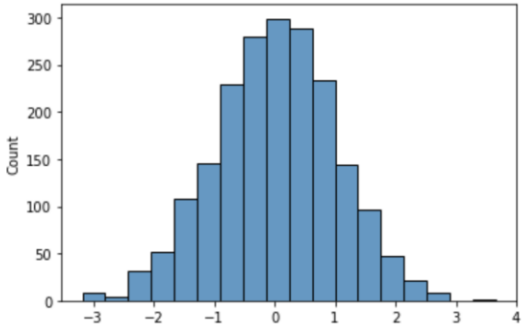
The **skew** represents the asymmetry of a normal distribution around its mean

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

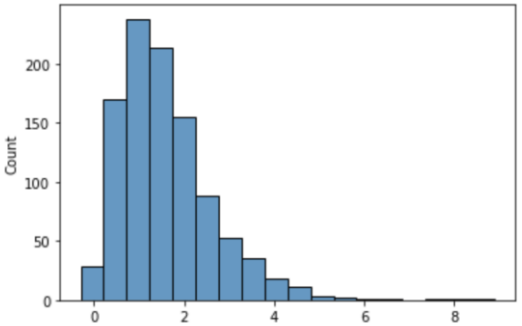
Left skew



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

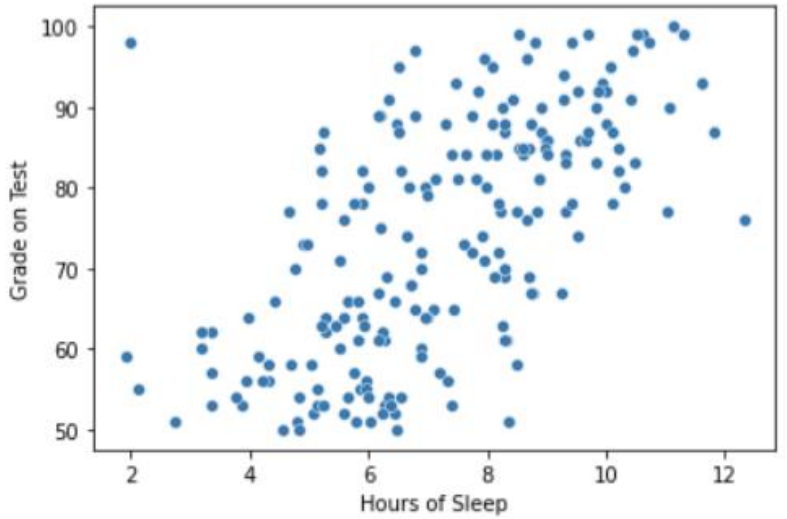
Scatterplots are used to visualize the relationship between numerical variables

- `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');
```

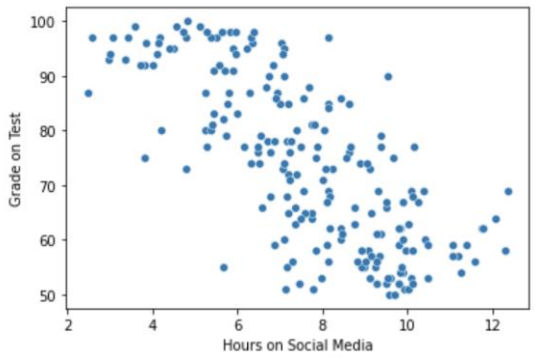


CORRELATION

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

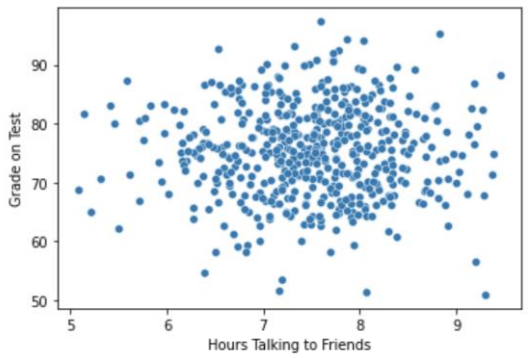
Correlation = –0.7



*This is a **negative correlation***

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

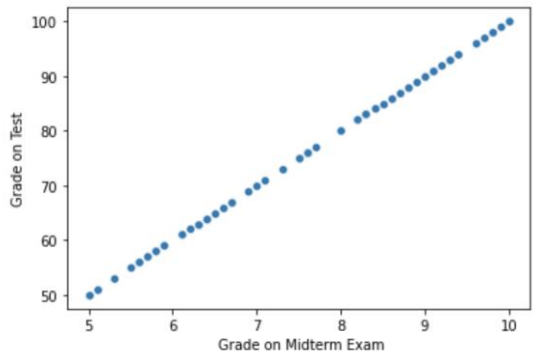
Correlation = 0



*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

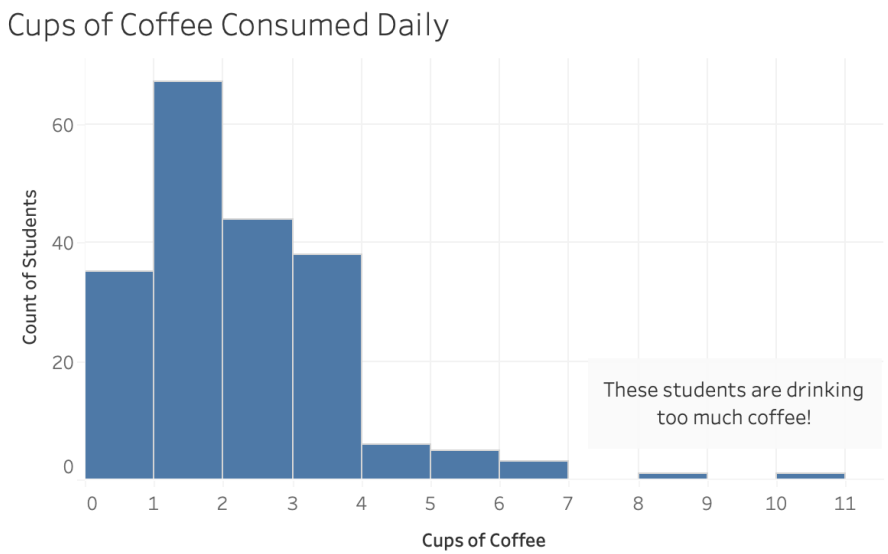
Key Objectives

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*

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**

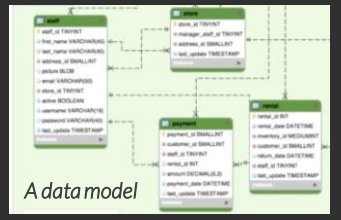
Data ready for EDA

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

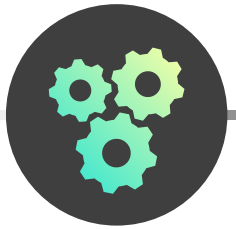
Data ready for modeling

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

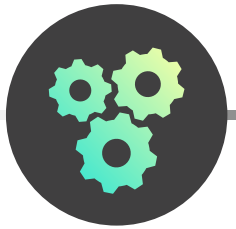
pd.concat(*[df1, df2, df3, ...]*)

```
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"

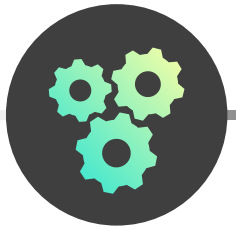
Type of join

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

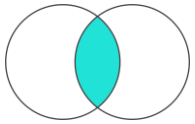
```
sales_may.merge(regions,
                how='left',
                left_on='store',
                right_on='store')
```

	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

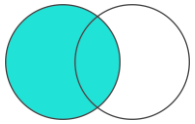
TYPES OF JOINS

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



how = 'inner'

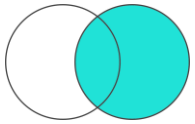
Returns records that exist in BOTH tables, and excludes unmatched records from either table



how = 'left'

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

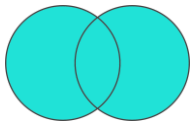
Most commonly used joins



how = 'right'

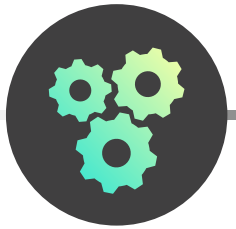
Returns ALL records from the RIGHT table, and any matching records from the LEFT table

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.

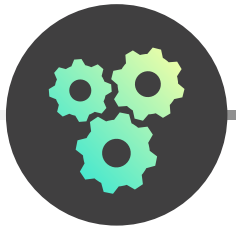
Can you reformat the data you compiled earlier this week so that it's ready to be input into a model, with each row representing a customer instead of a purchase?

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

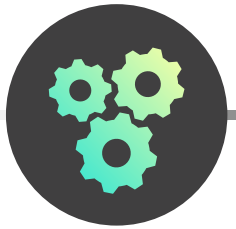


PREPARING COLUMNS FOR MODELING

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"

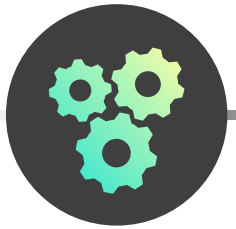
	days_passed
customer	
Aiden	16
Ava	50
Ben	5

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

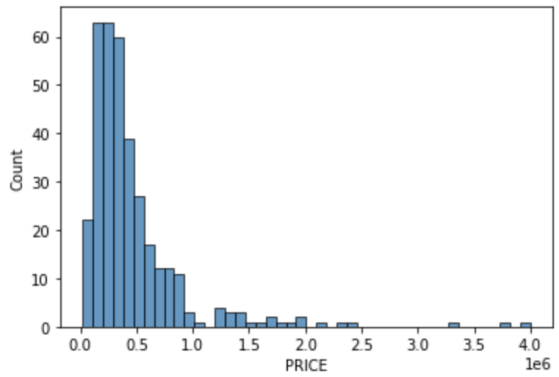
- **Log transforms** turn skewed data into more normally-distributed data
- You can use the **np.log()** function to apply a log transform to a Series

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);
```

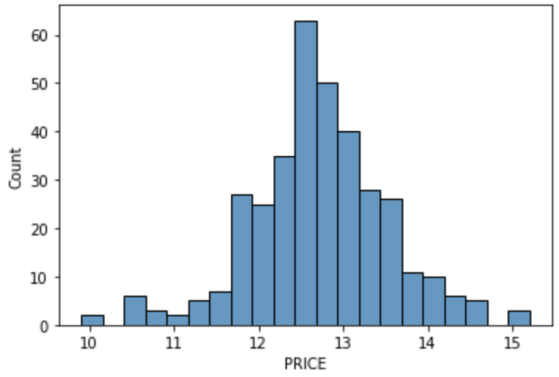


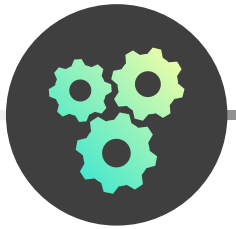
Normally-distributed data

```
# 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

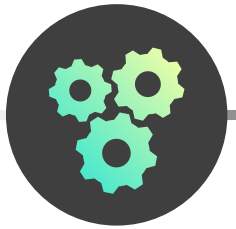
- They are used when a feature is either difficult to gather or engineer into a new feature

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*