WHAT'S YOUR EDA?

A beginner's guide to data exploration and visualization

Meet the Team!

Charles

Y2 DSA major + AI & Econs minor

Rachel

Y3 DSA major + Econs minor

Akshata

Y3 DSA major + IMD minor

Road Map

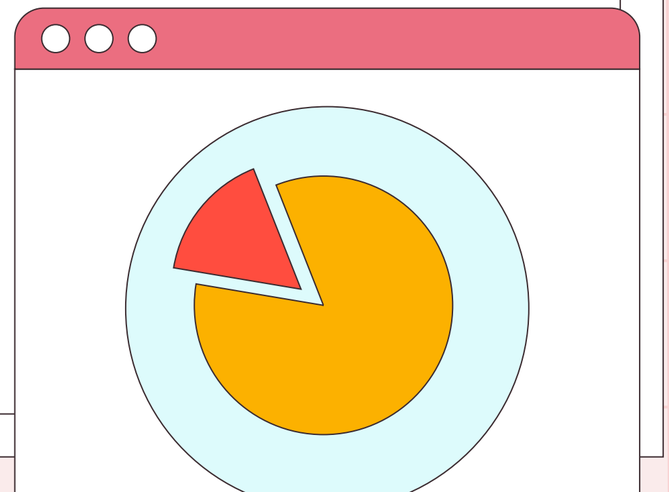
BUILD-UP APPROACH

Beginner	*Intermediate*	*Advanced*
Math: Linear Algebra, Calculus, Mathematical Analysis	Machine Learning	Deep Learning
Statistics, Probability	Web Scraping, API	Tensorflow, Pytorch
Programming Languages (Python, R)	Advanced SQL	Computer Vision, Natural Language Processing
Data Cleaning	Feature engineering	LLM
EDA and Data Visualization	Interpretability of ML models (Eg. SHAP, LIME)	Inferential Statistics
Dashboards	Time Series Analysis	Bayesian Statistics
SQL and Databases		MLOps

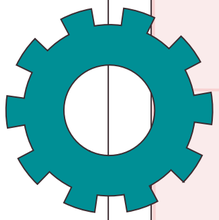
NOTE: NOT EXHAUSTIVE AND SUBJECT TO OWN JOURNEY!!!

Exploratory Data Analysis

01



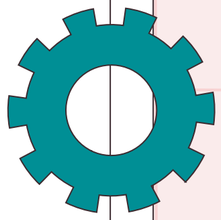
What is EDA?



Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods.

– *IBM*

Why do we need EDA?



01 What are you trying to solve?

02 Know what you are working with

03 Reliably discover patterns in data or perform machine learning

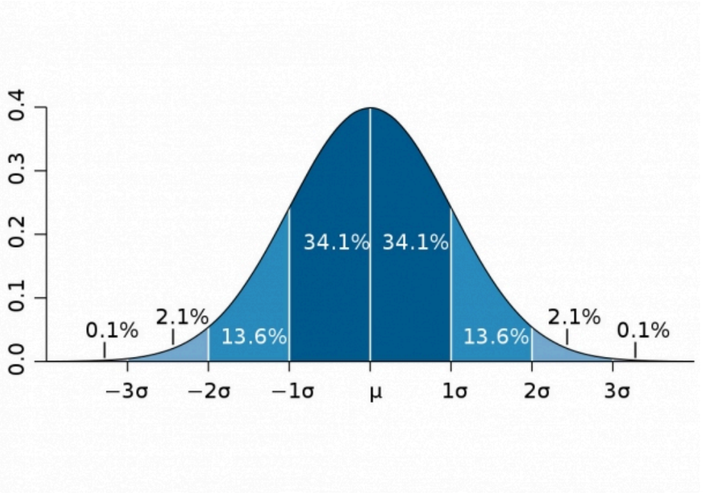
What are you trying to solve?

What is the business goal or research problem?

- Predict an outcome
- Understand what is driving sales down
- How can we reduce operational costs?
- What is the correlation between product price and time spent on the company’s social media page before the purchase?

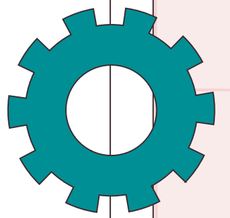
Know what you are working with

1. Features
    a. Variables, distributions
    b. Relationships between variables
2. Problems
    a. Duplicated values, missing values



Most important features for ML

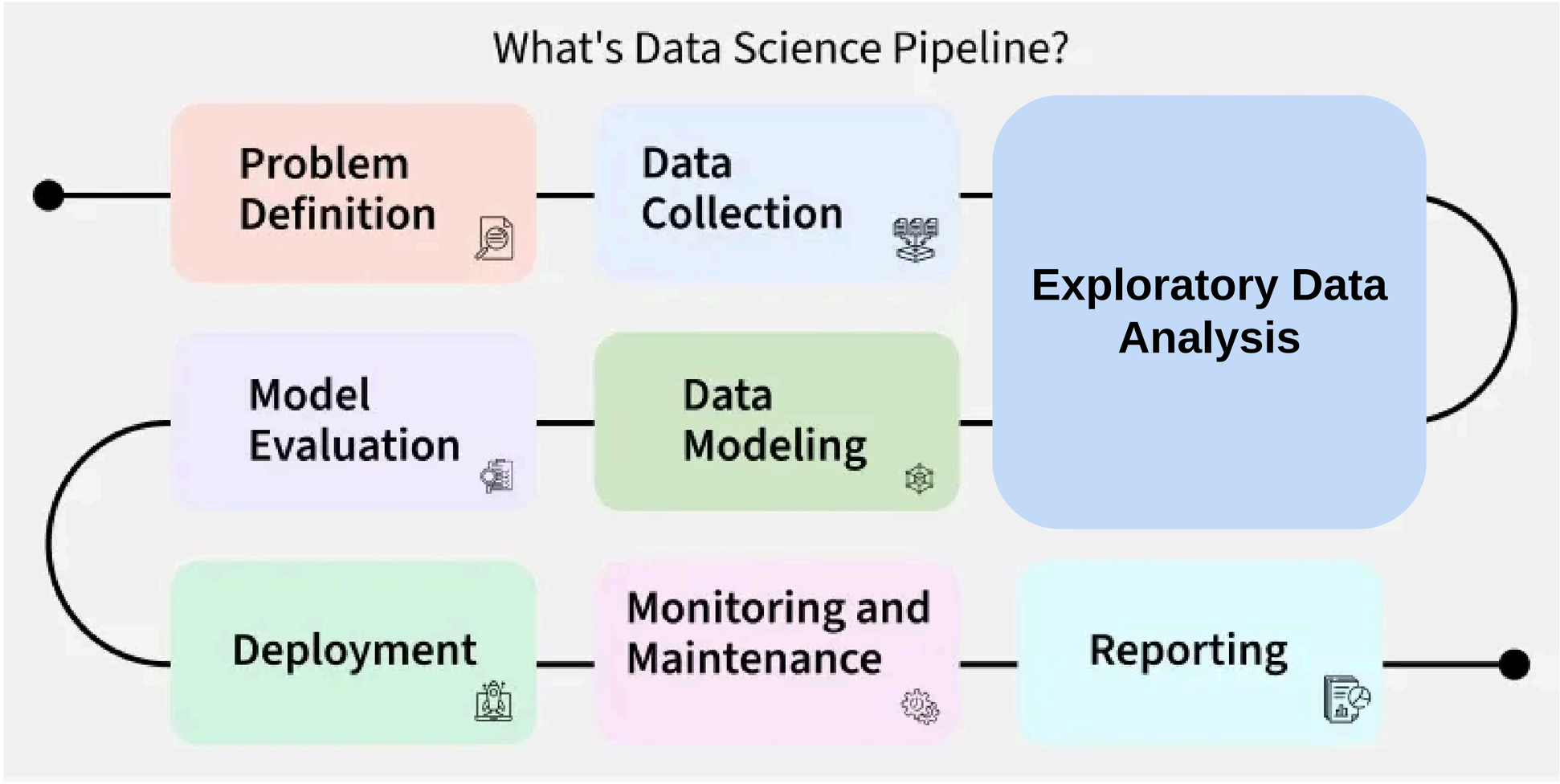
Dashboard creation

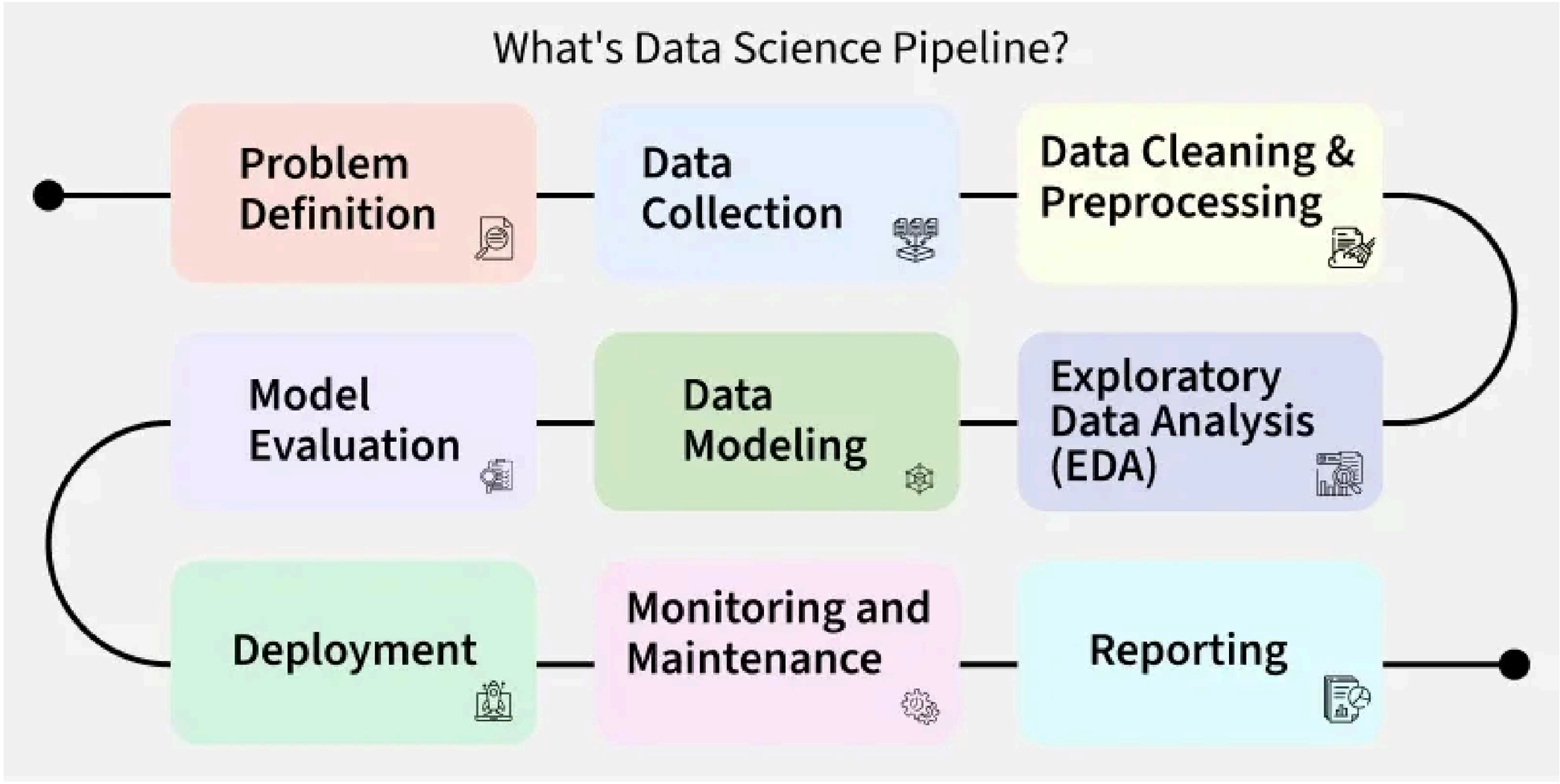


Insight

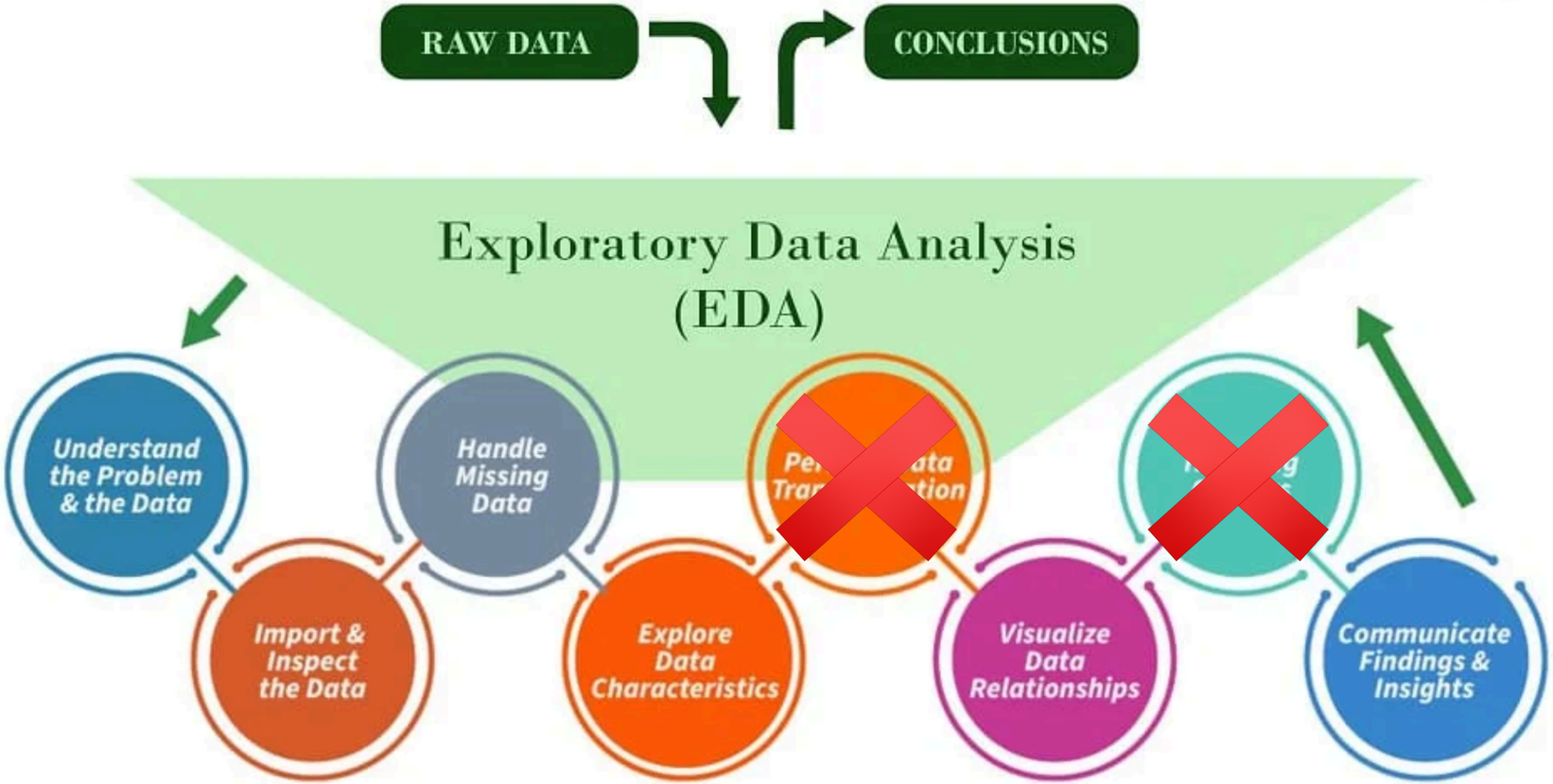
Suitable evaluation metrics

Choice of models


What's Data Science Pipeline?
Problem Definition
Data Collection
Exploratory Data Analysis
Model Evaluation
Data Modeling
Deployment
Monitoring and Maintenance
Reporting

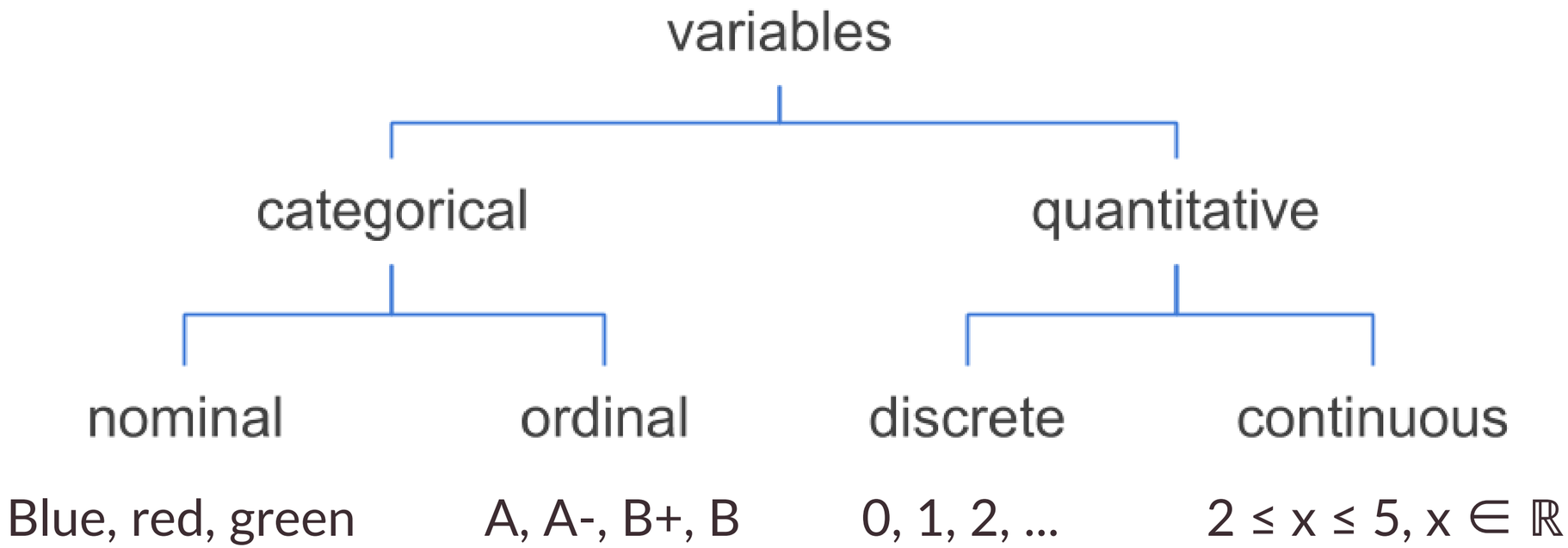

What's Data Science Pipeline?
Problem Definition
Data Collection
Data Cleaning & Preprocessing
Exploratory Data Analysis (EDA)
Data Modeling
Model Evaluation
Deployment
Monitoring and Maintenance
Reporting

EDA Pipeline



Foundational Concepts

Variables


variables
categorical
quantitative
nominal
ordinal
discrete
continuous
Blue, red, green
A, A-, B+, B
0, 1, 2, ...
$2 \leq x \leq 5, x \in \mathbb{R}$

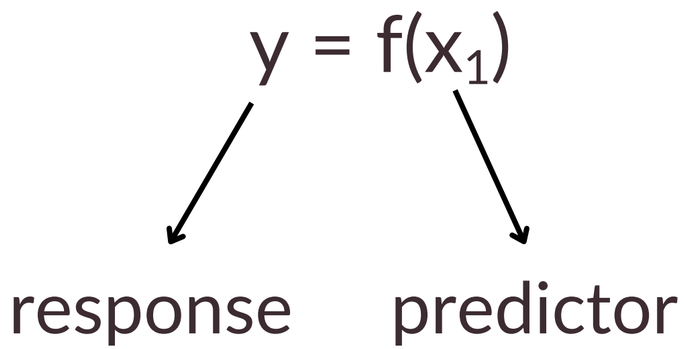
Variables

Explanatory / Predictor Variables:

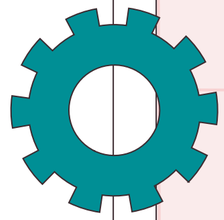
- Controlled/Changed in experiment
- Input variable for a ML model

Response Variable:

- Outcome of experiment
- Output variable



Python Libraries



NumPy

pandas

Python Libraries

Super fast calculator

Data Structure:
ndarray (n-dimensional array)

Python Libraries

pandas

Data Structures:
Series (1D array-like object)
Dataframes (spreadsheet)

Dataset

Dataset

This dataset contains information about employees in a company, including their educational backgrounds, work history, demographics, and employment-related factors.

PaymentTier → different salary tiers
EverBenched → indicates if an employee has ever been temporarily without assigned work

https://www.kaggle.com/datasets/tawfikelmetwally/employee-dataset

Dataset

LeaveOrNot → response variable

https://www.kaggle.com/datasets/tawfikelmetwally/employee-dataset

Dataset

Employee Retention

https://www.kaggle.com/datasets/tawfikelmetwally/employee-dataset

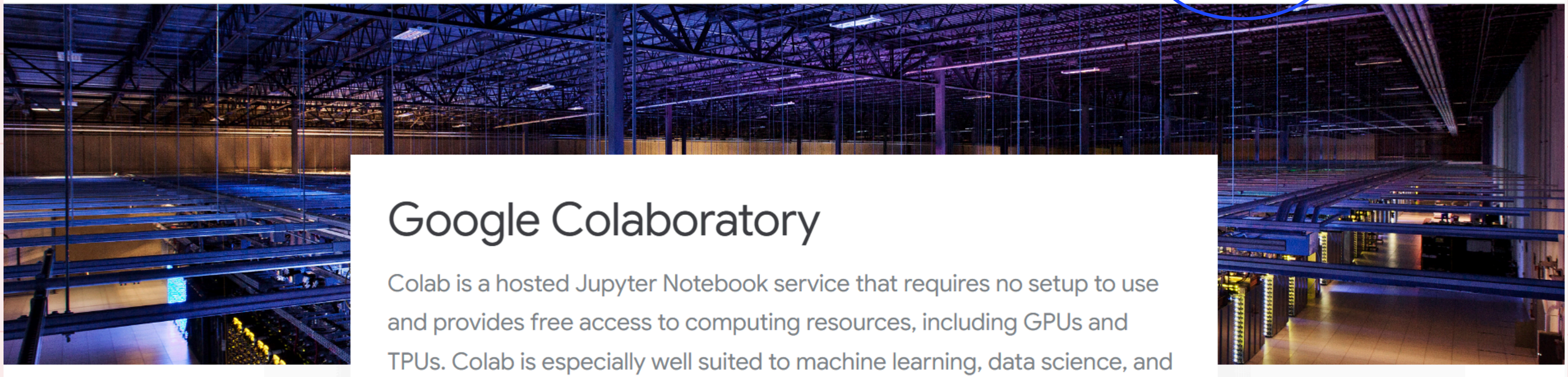
Google Colab Walkthrough

colab.google

Google colab

Blog Release Notes Notebooks Resources

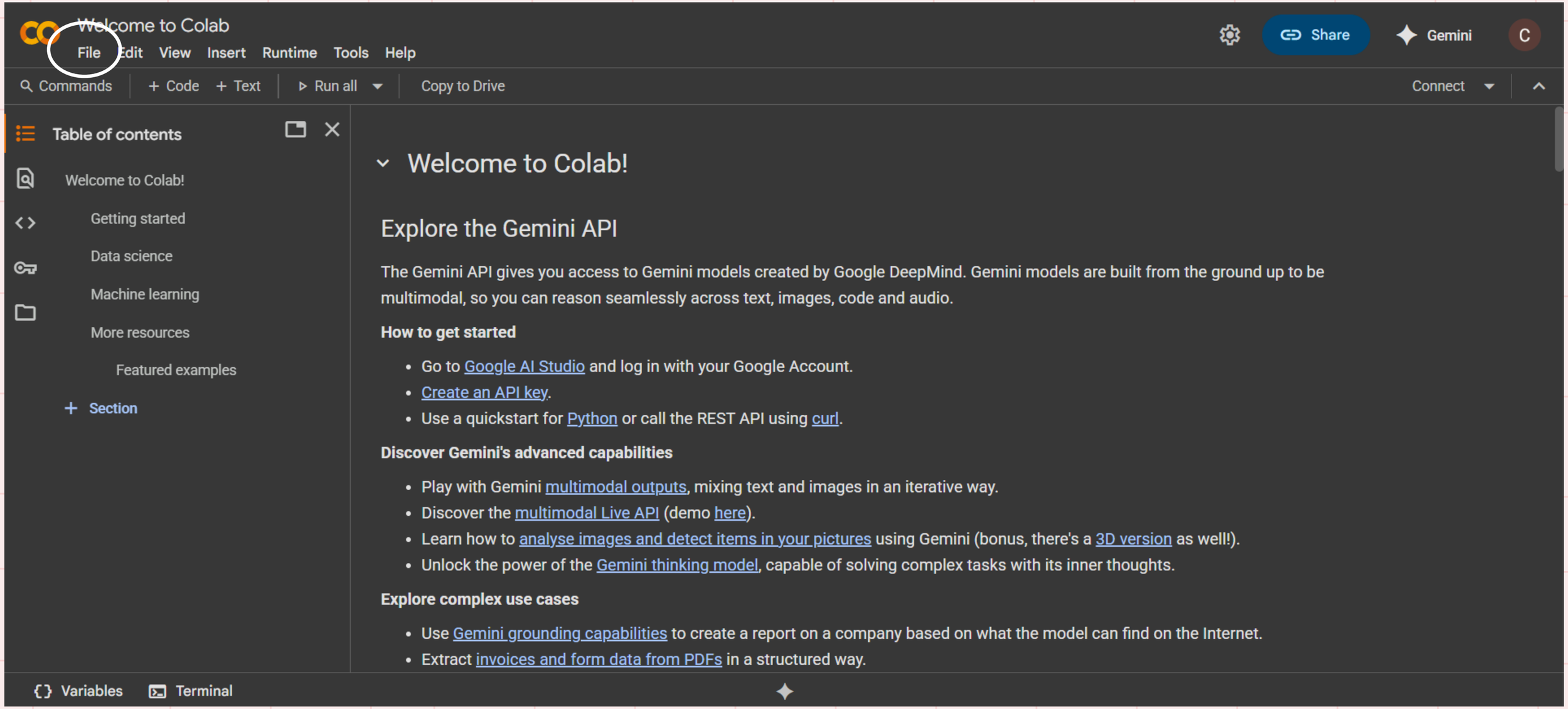
Open Colab New Notebook Sign Up

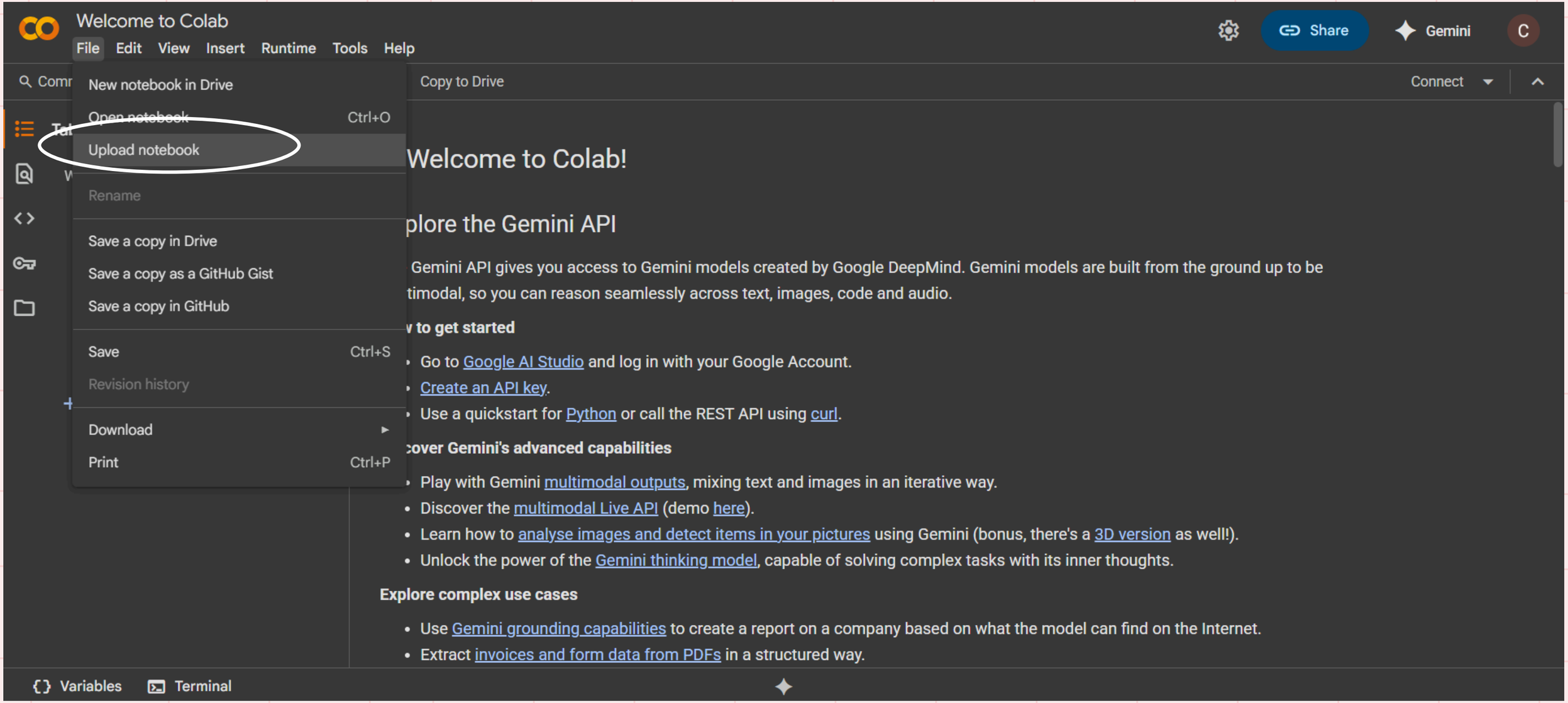


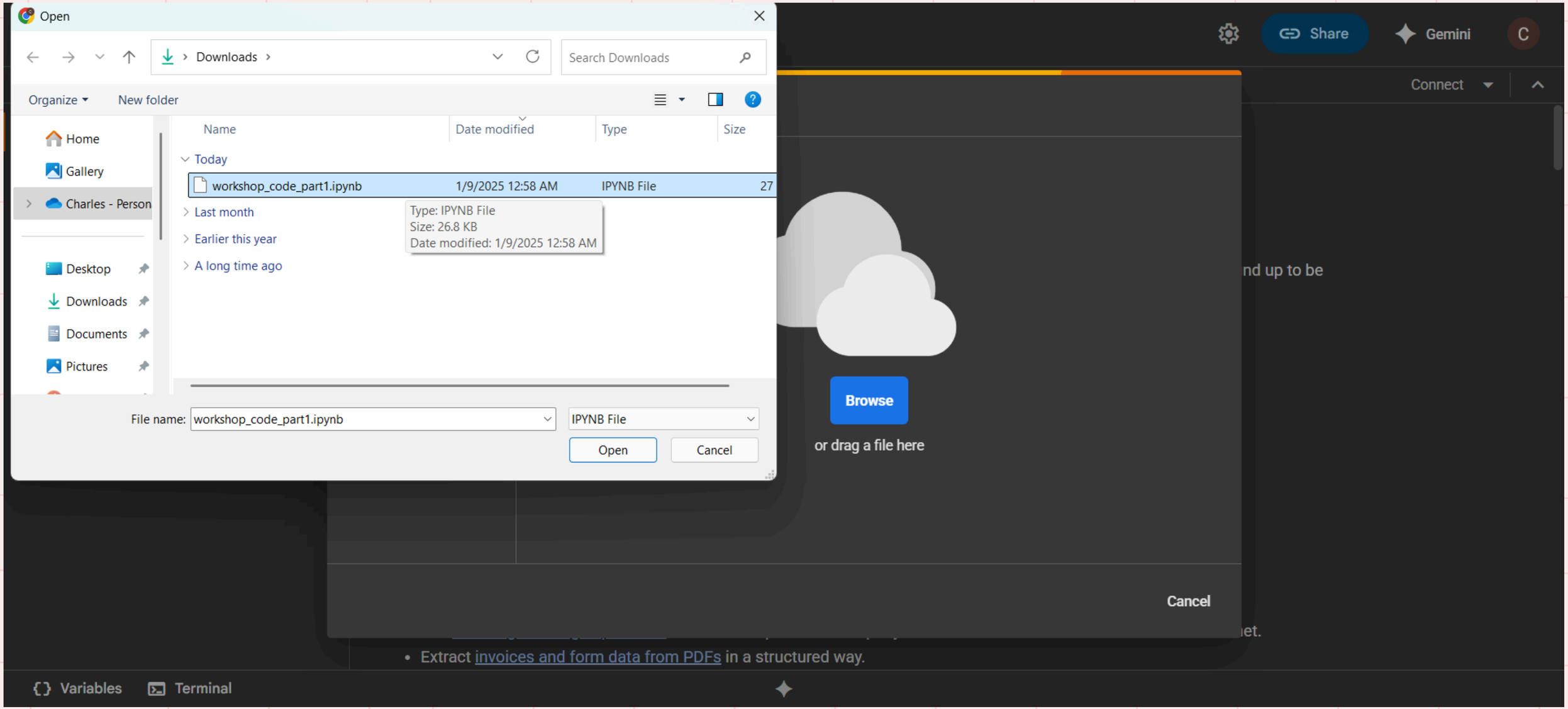
Google Colaboratory

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.

Open Colab New Notebook


Welcome to Colab
File Edit View Insert Runtime Tools Help
Share
Gemini
C
Commands
+ Code
+ Text
Run all
Copy to Drive
Connect
Table of contents
Welcome to Colab!
Getting started
Data science
Machine learning
More resources
Featured examples
Section
Welcome to Colab!
Explore the Gemini API
The Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be multimodal, so you can reason seamlessly across text, images, code and audio.
How to get started
Go to Google AI Studio and log in with your Google Account.
Create an API key.
Use a quickstart for Python or call the REST API using curl.
Discover Gemini's advanced capabilities
Play with Gemini multimodal outputs, mixing text and images in an iterative way.
Discover the multimodal Live API (demo here).
Learn how to analyse images and detect items in your pictures using Gemini (bonus, there's a 3D version as well!).
Unlock the power of the Gemini thinking model, capable of solving complex tasks with its inner thoughts.
Explore complex use cases
Use Gemini grounding capabilities to create a report on a company based on what the model can find on the Internet.
Extract invoices and form data from PDFs in a structured way.
Variables
Terminal


Welcome to Colab
File Edit View Insert Runtime Tools Help
Share
Gemini
C
New notebook in Drive
Open notebook Ctrl+O
Upload notebook
Rename
Save a copy in Drive
Save a copy as a GitHub Gist
Save a copy in GitHub
Save Ctrl+S
Revision history
Download
Print Ctrl+P
Copy to Drive
Connect
Welcome to Colab!
plore the Gemini API
Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be
timodal, so you can reason seamlessly across text, images, code and audio.
to get started
Go to Google AI Studio and log in with your Google Account.
Create an API key.
Use a quickstart for Python or call the REST API using curl.
cover Gemini's advanced capabilities
Play with Gemini multimodal outputs, mixing text and images in an iterative way.
Discover the multimodal Live API (demo here).
Learn how to analyse images and detect items in your pictures using Gemini (bonus, there's a 3D version as well!).
Unlock the power of the Gemini thinking model, capable of solving complex tasks with its inner thoughts.
Explore complex use cases
Use Gemini grounding capabilities to create a report on a company based on what the model can find on the Internet.
Extract invoices and form data from PDFs in a structured way.
Variables
Terminal


Open
Downloads
Search Downloads
Organize
New folder
Home
Gallery
Charles - Person
Desktop
Downloads
Documents
Pictures
Name
Date modified
Type
Size
Today
workshop_code_part1.ipynb
1/9/2025 12:58 AM
IPYNB File
27
Last month
Earlier this year
A long time ago
Type: IPYNB File
Size: 26.8 KB
Date modified: 1/9/2025 12:58 AM
File name:
workshop_code_part1.ipynb
IPYNB File
Open
Cancel
Share
Gemini
Connect
Browse
or drag a file here
Cancel
Extract invoices and form data from PDFs in a structured way.
Variables
Terminal

workshop_code_part1.ipynb

File Edit View Insert Runtime Tools Help

Share Gemini C

Commands + Code + Text Run all Connect

Table of Contents

Files

What's Ur EDA? | Exploratory Data Analysis I

Hello! Welcome to your first ipynb file for this workshop! The language we are using is Python. Here, we will cover data inspection and data cleaning techniques. After this segment, you will learn how to:

1. Read a csv file into the Python environment
2. Investigate the features of your dataset
3. Deal with duplicated values, inconsistent labelling, missing and erroneous values, and outliers
4. Generate a Profile Report with YData as an alternative

Please ensure that you have downloaded "employee_dirty.csv" and uploaded them (right click) on the side under "Files". You can use the "Table of Contents" tab at the side to help with navigation.

The original csv file can be found from https://www.kaggle.com/datasets/tawfikelmetwally/employee-dataset.

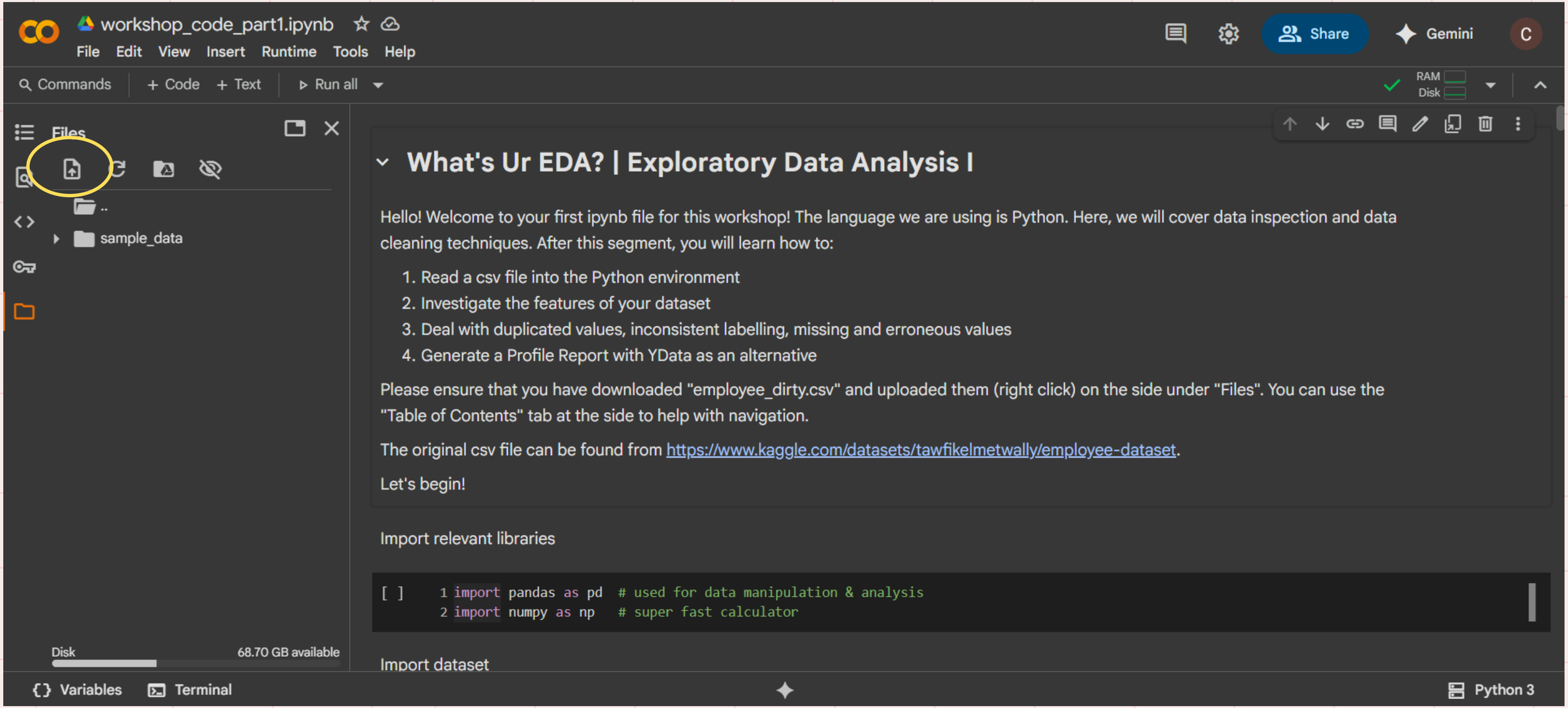
Let's begin!

Import relevant libraries

```
[ ]  1 import pandas as pd
     2 import numpy as np
```

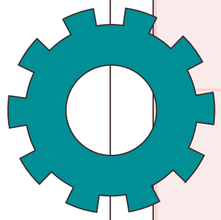
Import dataset

Variables Terminal


workshop_code_part1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
Files
..
sample_data
Disk
68.70 GB available
What's Ur EDA? | Exploratory Data Analysis I
Hello! Welcome to your first ipynb file for this workshop! The language we are using is Python. Here, we will cover data inspection and data cleaning techniques. After this segment, you will learn how to:
1. Read a csv file into the Python environment
2. Investigate the features of your dataset
3. Deal with duplicated values, inconsistent labelling, missing and erroneous values
4. Generate a Profile Report with YData as an alternative
Please ensure that you have downloaded "employee_dirty.csv" and uploaded them (right click) on the side under "Files". You can use the "Table of Contents" tab at the side to help with navigation.
The original csv file can be found from https://www.kaggle.com/datasets/tawfikelmetwally/employee-dataset.
Let's begin!
Import relevant libraries
[ ]
1 import pandas as pd # used for data manipulation & analysis
2 import numpy as np # super fast calculator
Import dataset
Variables
Terminal
Python 3

code timeee

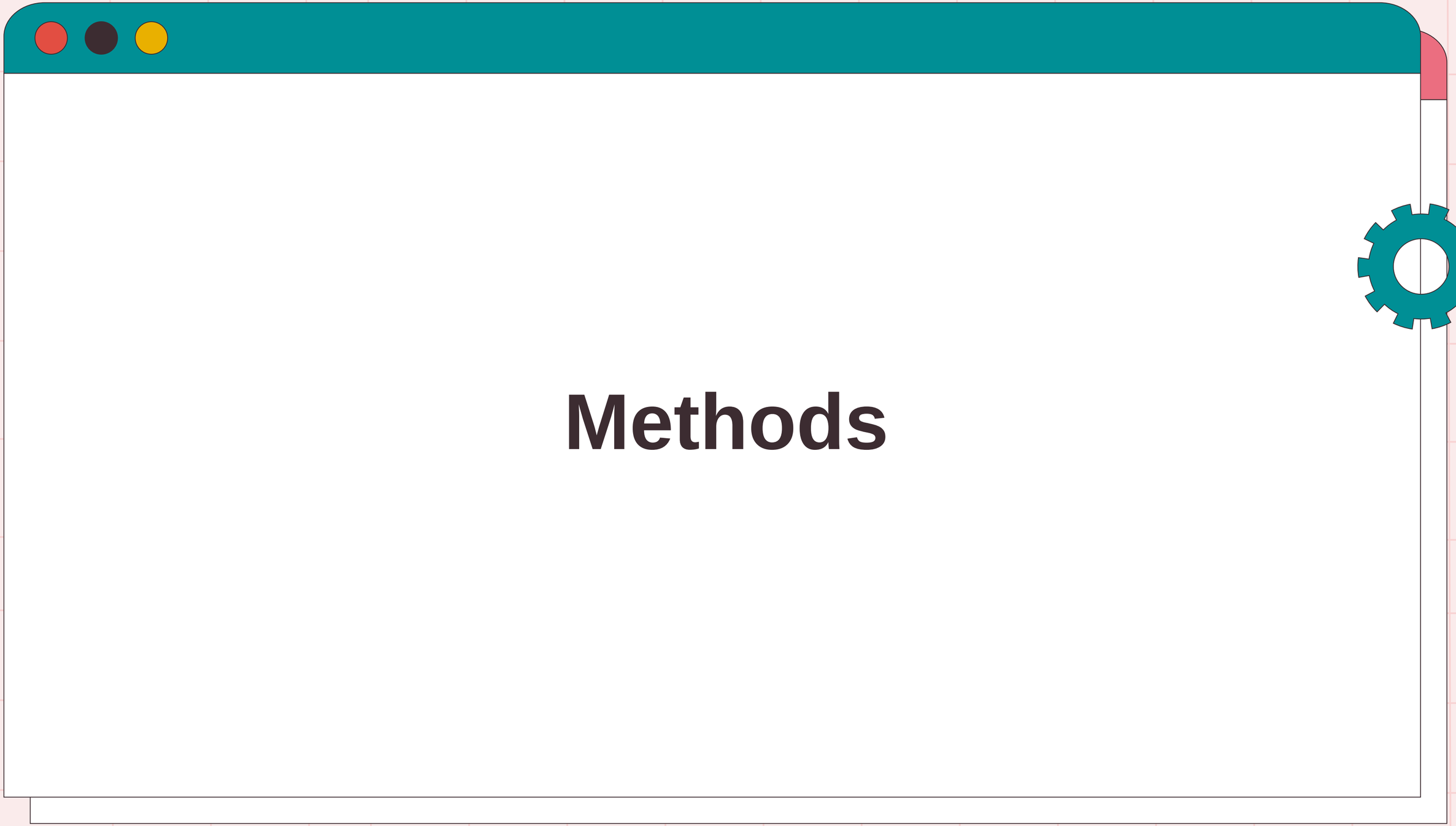
Useful Stuff!



```
# comment
```

```
'''
comments with
many lines
'''
```

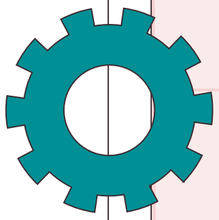
Please don’t click the “run all” button. We will run the code blocks together.



Methods

Import dataset

```
[ ]    1 df = pd.read_csv("employee_dirty.csv") # Check if the first row is a header. If not, header=None
```

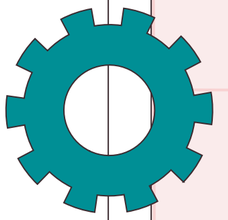


.read_csv() is a **method of the Pandas library**

- function that belongs to the entire library
- used to perform a task
- tools in a toolbox

Import dataset

```
[ ]    1 df = pd.read_csv("employee_dirty.csv") # Check if the first row is a header. If not, header=None
```



$$y = f(x_1, x_2)$$
$$\| \quad \|$$
$$2 \quad 3$$

Import dataset

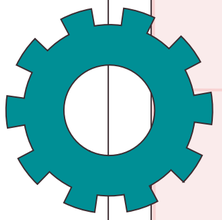
```
[ ]    1 df = pd.read_csv("employee_dirty.csv") # Check if the first row is a header. If not, header=None
```

header is a parameter in .read_csv()

- variable to be filled

“header = None” is the argument for .read_csv()

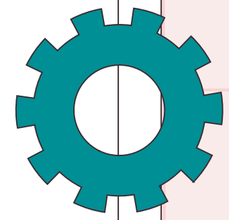
```
[ ]     1 # Shows the first 5 rows of dataframe & column headers
        2 df.head()
```



.head() is a **method of the dataframe object**

- Function specific to the dataframe
- Shows the first few rows

In general



pd.method()

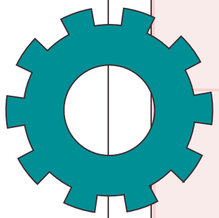
df.method()

Data Quality Issues

```
Education
['Bachelors' 'Masters' nan 'PHD']
```

```
Age
[34. 28. 38. 27. nan 23. 37. 32. 39. 29. 30. 22. 36. 31. 25. 26. 40. 35.
 24. -9. -1. 33. 41.]

Gender
['Male' 'Female' 'm' 'F']
```

```
ExperienceInCurrentDomain
[  0   3   2   5   1   4 200   7   6]
```



Duplicated Values

Data Quality Issues

Duplicated Values

	Pet	Color	Eyes
0	Cat	Brown	Black
1	Dog	Golden	Black
2	Dog	Golden	Black
3	Dog	Golden	Brown
4	Cat	Black	Green

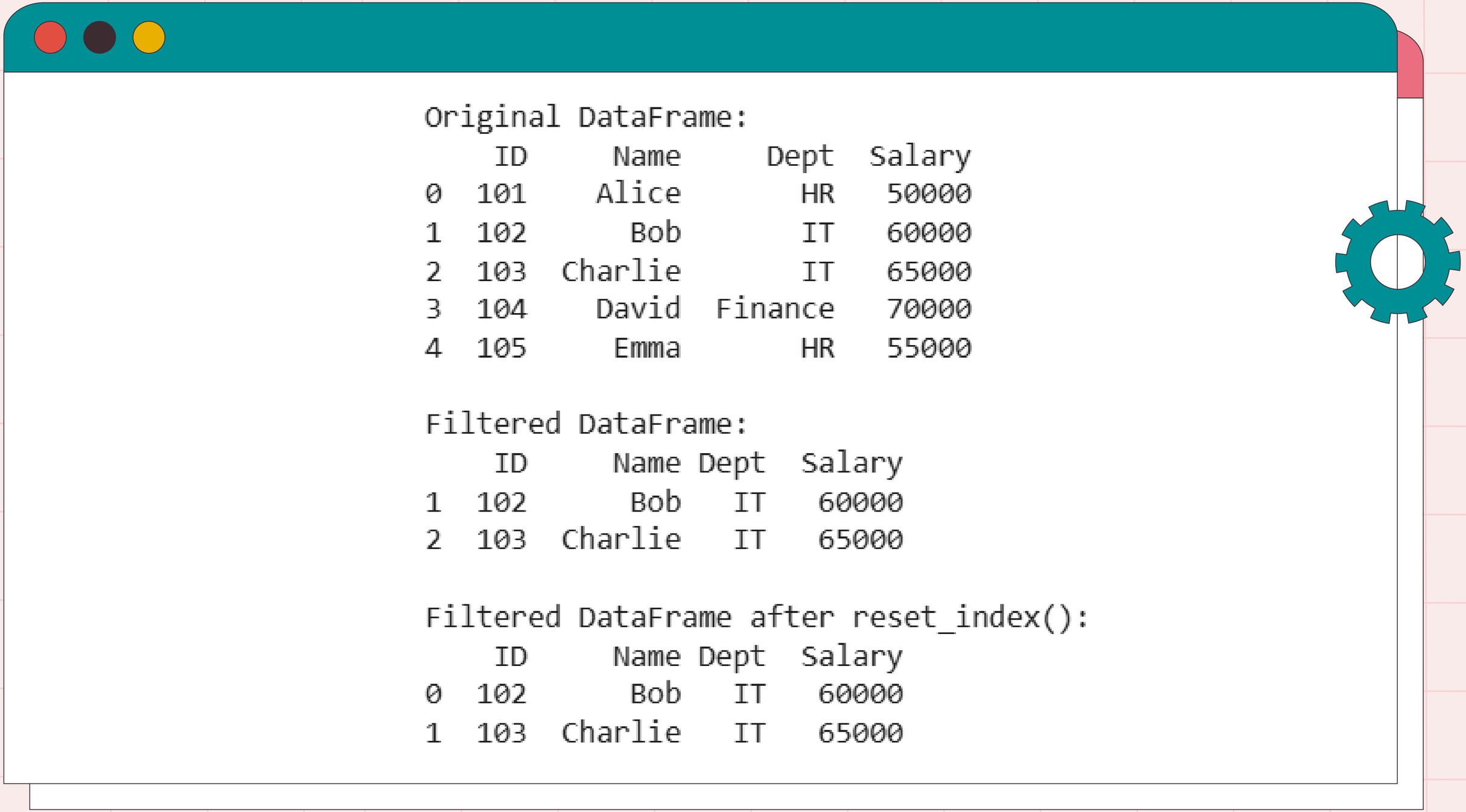
	Pet	Color	Eyes
0	Cat	Brown	Black
1	Dog	Golden	Black
3	Dog	Golden	Brown
4	Cat	Black	Green

Drop duplicates

Repeated data will give more weight to the data point it represents.

Reset Index

```
<class 'pandas.core.frame.DataFrame'>
Index: 3109 entries, 0 to 4651
Data columns (total 9 columns):
```



```
Original DataFrame:
    ID     Name     Dept  Salary
0  101    Alice       HR   50000
1  102      Bob       IT   60000
2  103  Charlie       IT   65000
3  104    David  Finance   70000
4  105     Emma       HR   55000

Filtered DataFrame:
    ID     Name Dept  Salary
1  102      Bob   IT   60000
2  103  Charlie   IT   65000

Filtered DataFrame after reset_index():
    ID     Name Dept  Salary
0  102      Bob   IT   60000
1  103  Charlie   IT   65000
```

Inconsistent Labelling

Data Quality Issues

Inconsistent Labelling

The same category is counted more than once.

- Different representation of Females = Undercounting in any variant of “Female”
- Incorrect statistics

Inconsistent Labelling

loc vs iloc

Feature	.loc	.iloc
Syntax	df.loc[row_indexer, column_indexer]	df.iloc[row_indexer, column_indexer]
Indexing Method	Label-based indexing	Position-based indexing
Used for Reference	Row and column labels (names)	Numerical indices of rows and columns (starting from 0)

Accessing Rows...

	CustomerID	Name	Age	Region	Income Strata	Sales
101	X1000	John	30	North	High	250
102	X1010	Ann	19	North	Medium	5000
103	X1020	Joe	25	South	Medium	132
104	X1030	Alice	53	East	Low	400
105	X1040	Susan	38	South	Medium	780
106	X1050	Bill	68	West	High	223

Dealing with missing / erroneous values

Dealing with missing values

Methods:

1. Deleting rows with missing values
2. Use mean/mode/median imputation
3. Advanced imputation techniques like kNN imputation

Imputation for Categorical Variables

Methods:

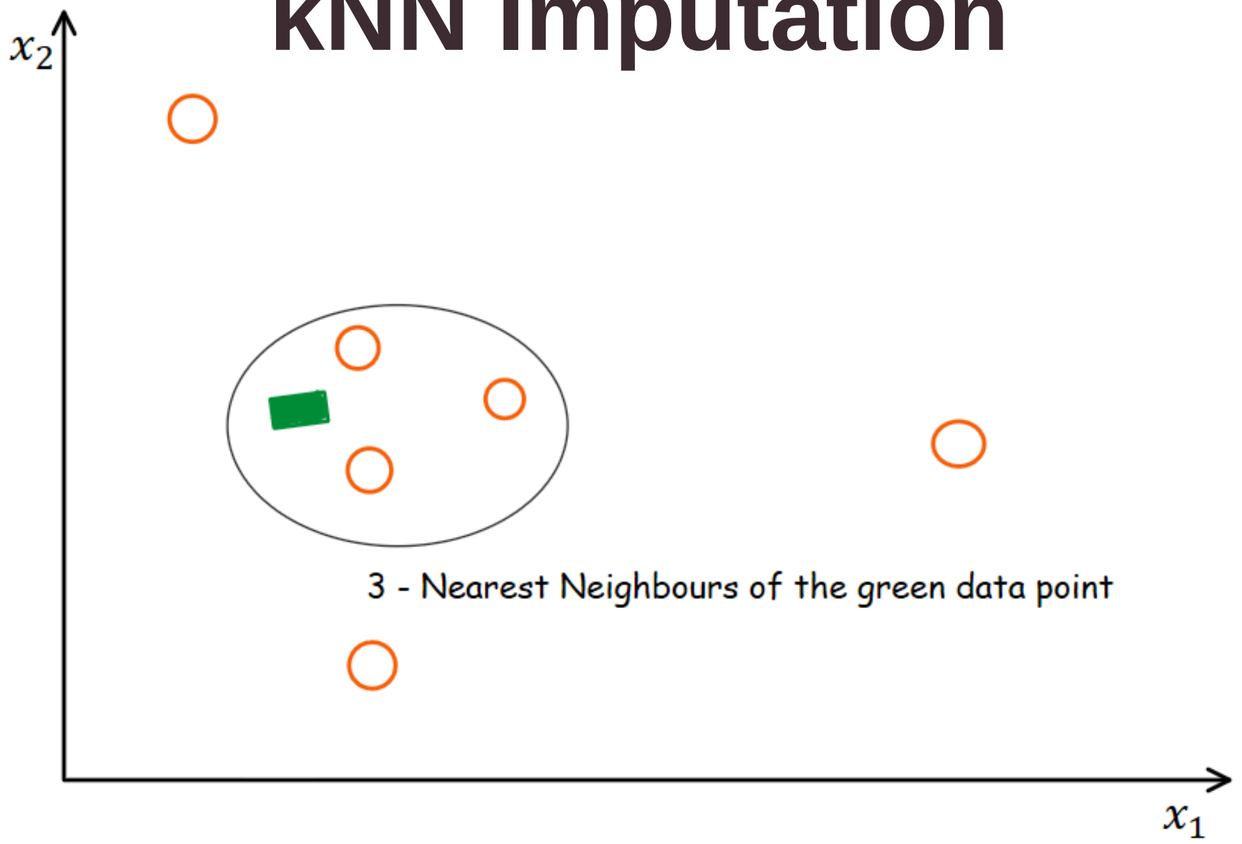
1. Impute using the most frequent category
2. Impute the value “Missing”
    - MissingIndicator from scikitlearn

Imputation for Categorical Variables

```
df_clean["Education"] = imputer.fit_transform(df_clean[["Education"]]).ravel()
```

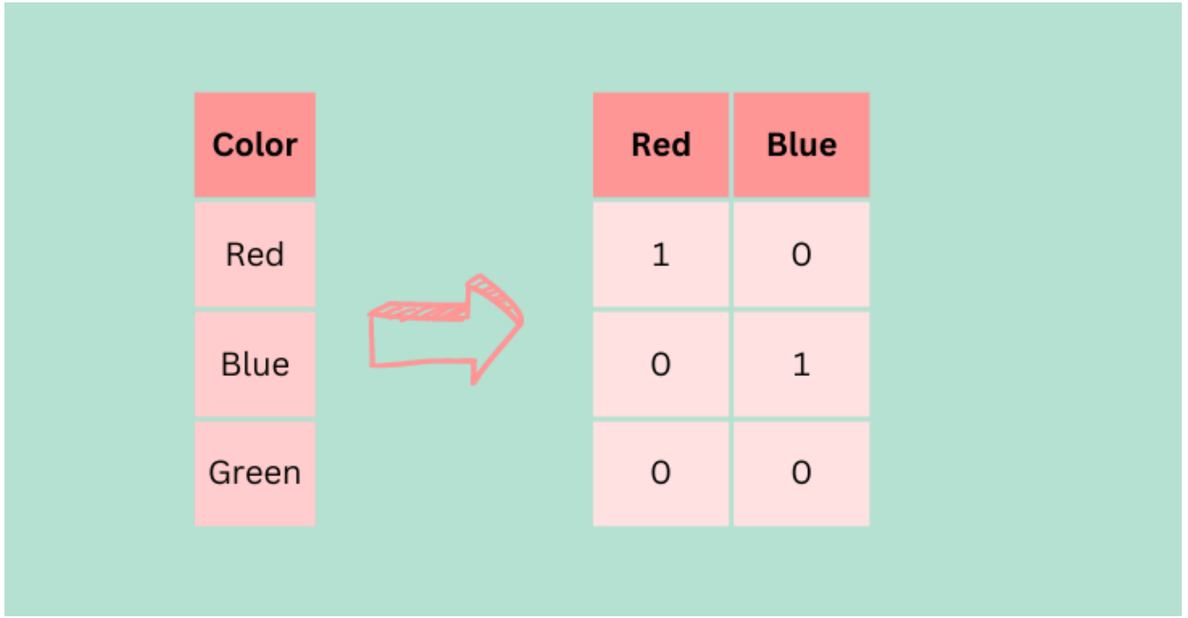
- fit: looks at column to find most frequent value
- transform: transform NaN with the value found
- [[]]: ensures that we have “Education” column has dataframe and not pandas Series
- .ravel(): takes 2D numPy array output → 1D array

kNN Imputation



The algorithm finds *k* other values that are most similar (by distance) and average them to fill the missing value

One-hot Encoding



One-hot encoding turns words (categories) into numbers the computer can understand

Scaling

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \in [0, 1]$$

Min-max normalisation

$$z = \frac{x_i - \mu}{\sigma}$$

Standardisation

Standardisation

	Age	Salary
0	0.758874	7.494733e-01
1	-1.711504	-1.438178e+00
2	-1.275555	-8.912655e-01
3	-0.113024	-2.532004e-01
4	0.177609	6.632192e-16
5	-0.548973	-5.266569e-01
6	0.000000	-1.073570e+00
7	1.340140	1.387538e+00
8	1.630773	1.752147e+00
9	-0.258340	2.937125e-01

Max-Min Normalization

	Age	Salary
0	0.739130	0.685714
1	0.000000	0.000000
2	0.130435	0.171429
3	0.478261	0.371429
4	0.565217	0.450794
5	0.347826	0.285714
6	0.512077	0.114286
7	0.913043	0.885714
8	1.000000	1.000000
9	0.434783	0.542857

Scaling

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \in [0, 1]$$

Min-max normalisation

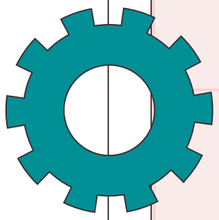
Color
Red
Blue
Green

Red	Blue
1	0
0	1
0	0

Categorical variables have range 0 to 1

What you have learnt

1. Types of variables
2. Pandas & NumPy
3. Methods
4. How to use Google Colab
5. How to explore the dataset
6. How to deal with data quality issues
7. Data scaling techniques



Learn more on EDA!

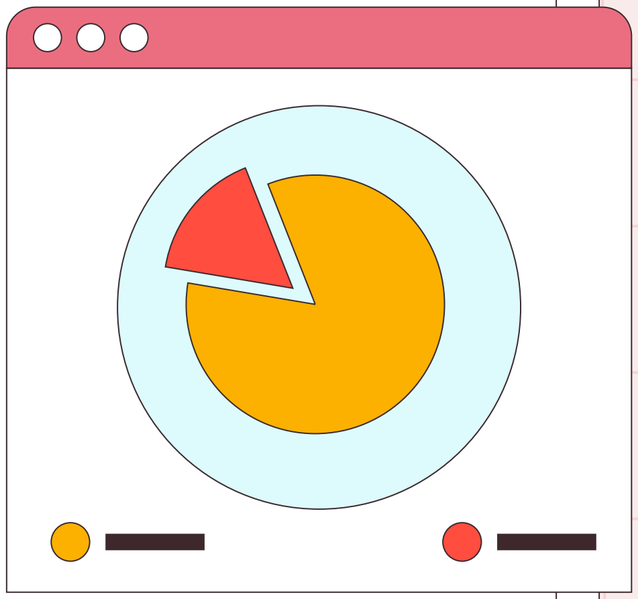
geeksforgeeks

medium

analyticsvidhya

End of Part 1!

5 min break

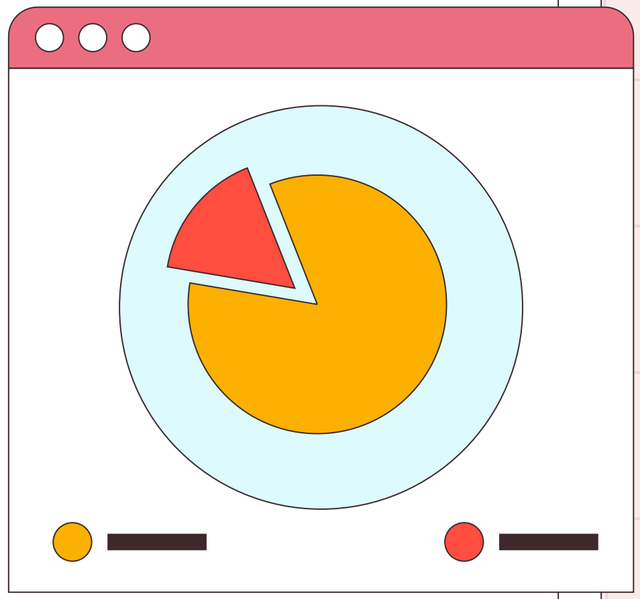


Welcome Back to Part 2!

1. Please download employee_cleaned.csv and upload it onto workshop_code_part2.ipynb!
2. Kindly do not run all the cells. We will run through them chunk by chunk

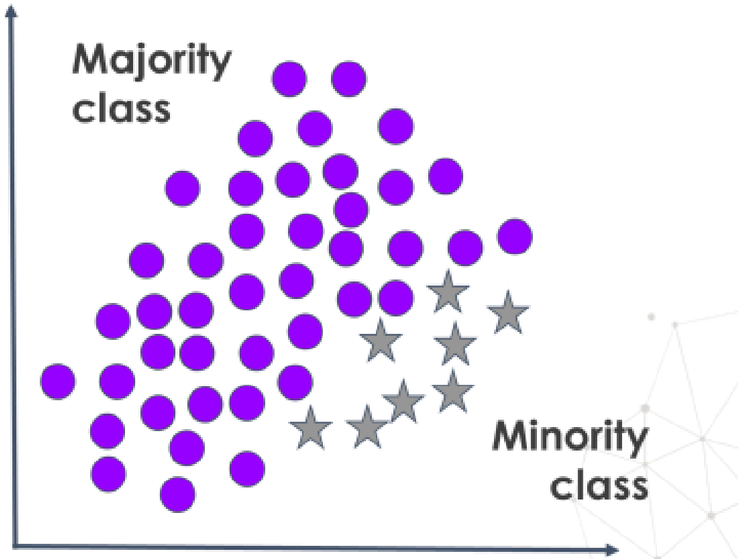
02

Imbalanced data

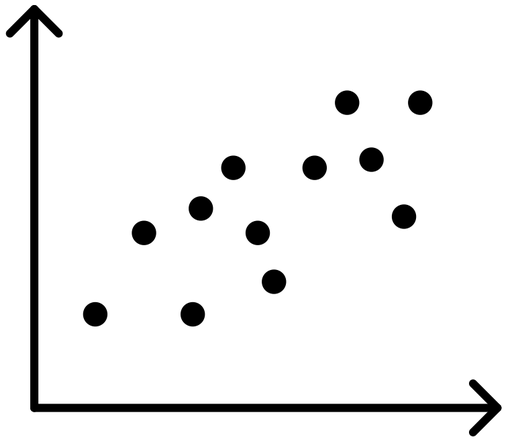


What is an imbalanced data?

- A classification dataset where one class has significantly more data points than another class.
- A **common rule of thumb: 80~20**



In EDA, we will report any imbalances from the original dataset into visualisations (e.g. barplot).



Let’s Code!

Identifying imbalance

df[‘column’].**value_counts()**

- a Pandas function that counts how many times each unique value appears in a column of a DataFrame (frequency count)

```
# for 'EverBenched'
everbenched_counts = df['EverBenched'].value_counts()
print(everbenched_counts)
```

```
EverBenched
No      2726
Yes      383
```

Getting proportions

```
no_percent = round((everbenched_counts['No'] / len(df)) * 100, 2)
print("\nPercentage of employees who were never benched:", no_percent)

yes_percent = round((everbenched_counts['Yes'] / len(df)) * 100, 2)
print("\nPercentage of employees who were benched:", yes_percent)
```

```
Percentage of employees who were never benched: 87.68

Percentage of employees who were benched: 12.32
```

Simpler code!

```
# getting both counts & percentage in one line of code
percentage = df['EverBenched'].value_counts(normalize=True) * 100
percentage
```

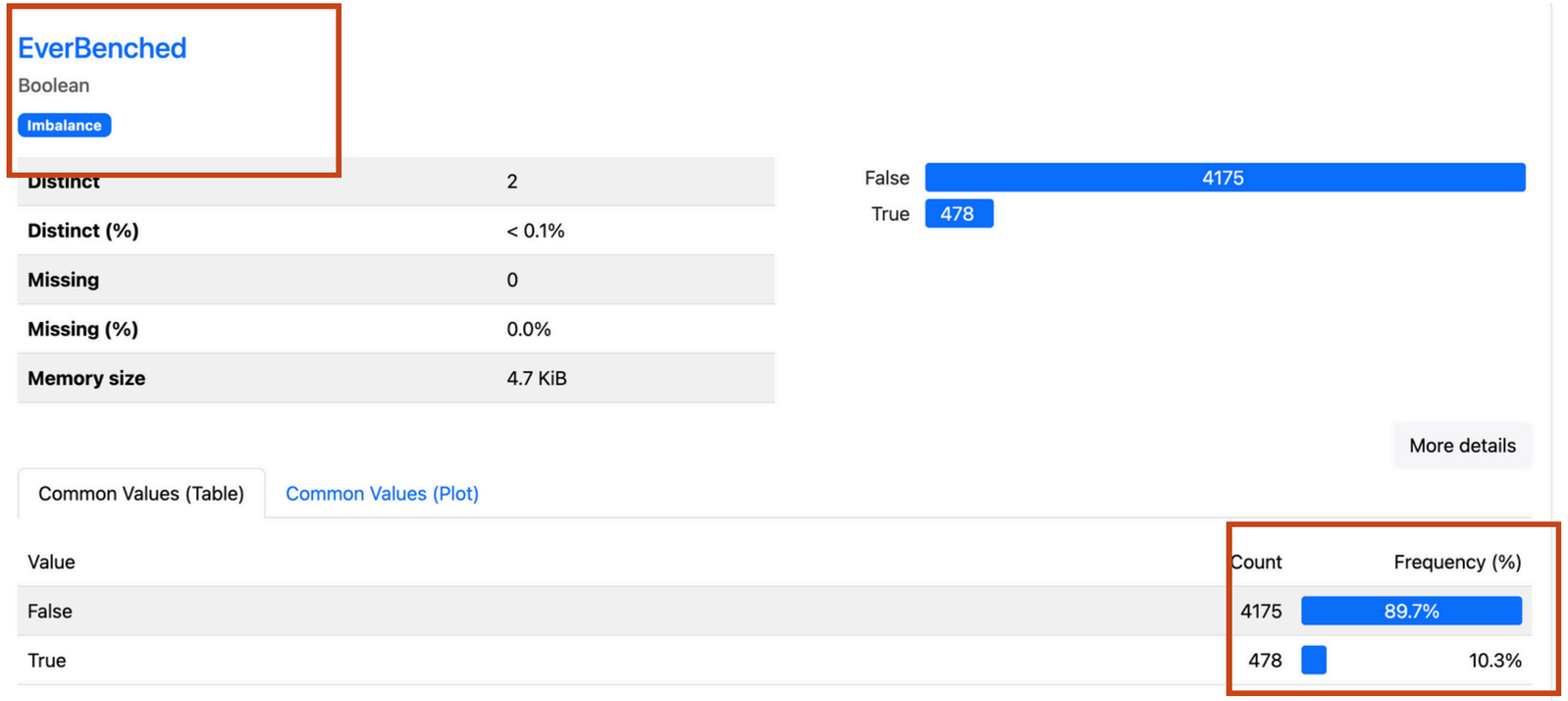
gives the relative frequencies of the unique values

EverBenched	proportion
No	87.680926
Yes	12.319074

imbalanced!

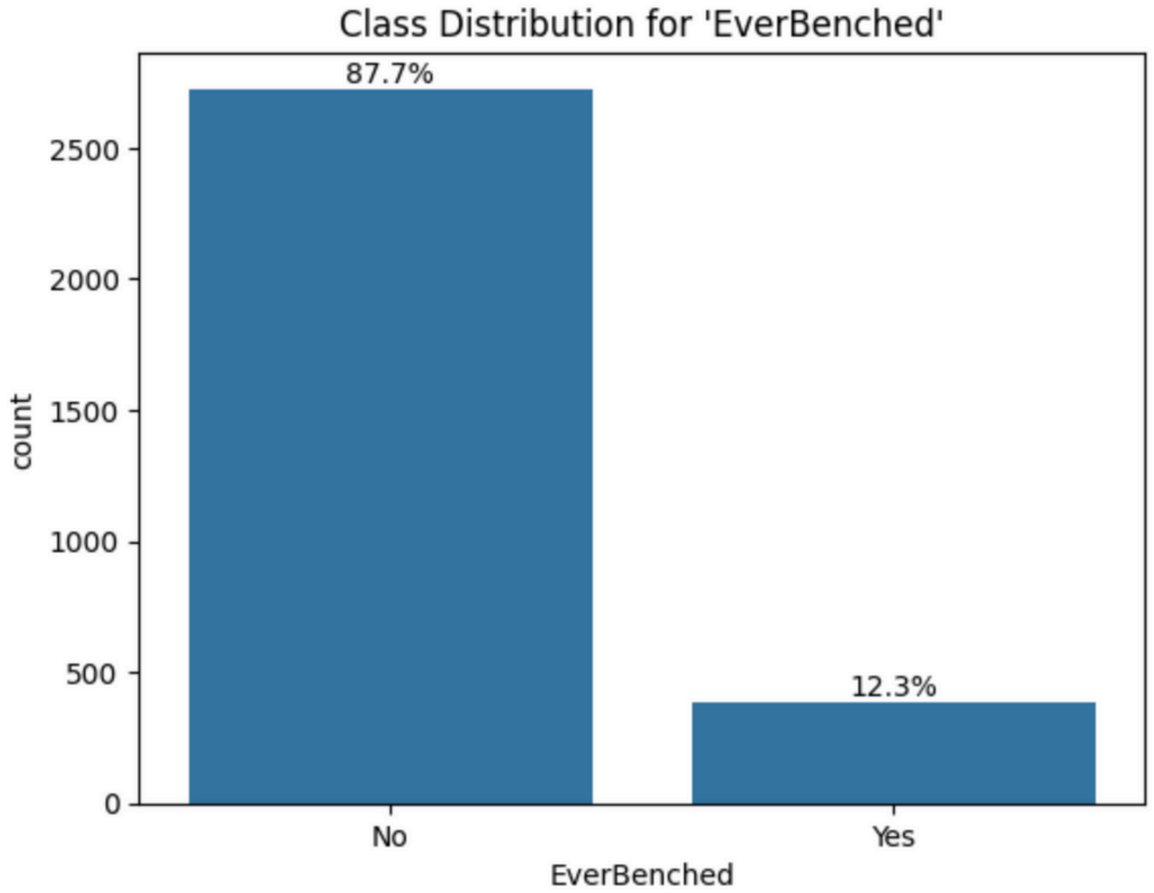
Identifying imbalance using ydata!

Note: This screenshot is not from the dataset that we have. This is only to show you what ydata can do.



ydata can point it out to you as well!

Visualising this finding



Note: We will go through how to code visualisation in part 3 of our workshop!

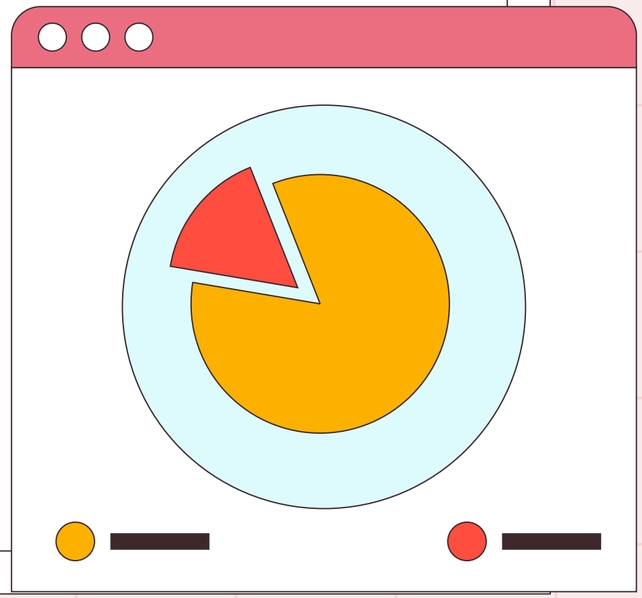
Recap~

EDA is the process of summarizing, visualizing, and understanding your data before building models and it helps identify patterns, trends, anomalies, and relationships in the dataset.

How can we derive useful insights from our cleaned data?

03

Analysis in EDA



Types of Analysis

Can you guess what each analysis is for?

Univariate

1 variable

Know your variables individually → data cleaning & summary stats

Bivariate

Finds the relationship between 2 variables

Multivariate

Finds the relationship of more than two variables

Univariate Analysis

To derive the data, define and summarize it,
and analyze the pattern present in it

Numerical Variables	Categorical Variables
• Central tendency: mean, median, mode. • Dispersion: variance, standard deviation, quartiles. • Visualization: boxplots, histogram	• Frequency counts, proportions. • Visualization: bar charts, pie charts

E.g: “What is the age distribution of employees in the company?”

Let’s code!

Numerical Variable Code

```
DataFrame.count
#Count number of non-NA/null observations.

DataFrame.max
#Maximum of the values in the object.

DataFrame.min
#Minimum of the values in the object.

DataFrame.mean
#Mean of the values.

DataFrame.std
#Standard deviation of the observations.

DataFrame.select_dtypes
#Subset of a DataFrame including/excluding columns based on their dtype.
```

BUT, there’s an easier way!

Using .describe( )

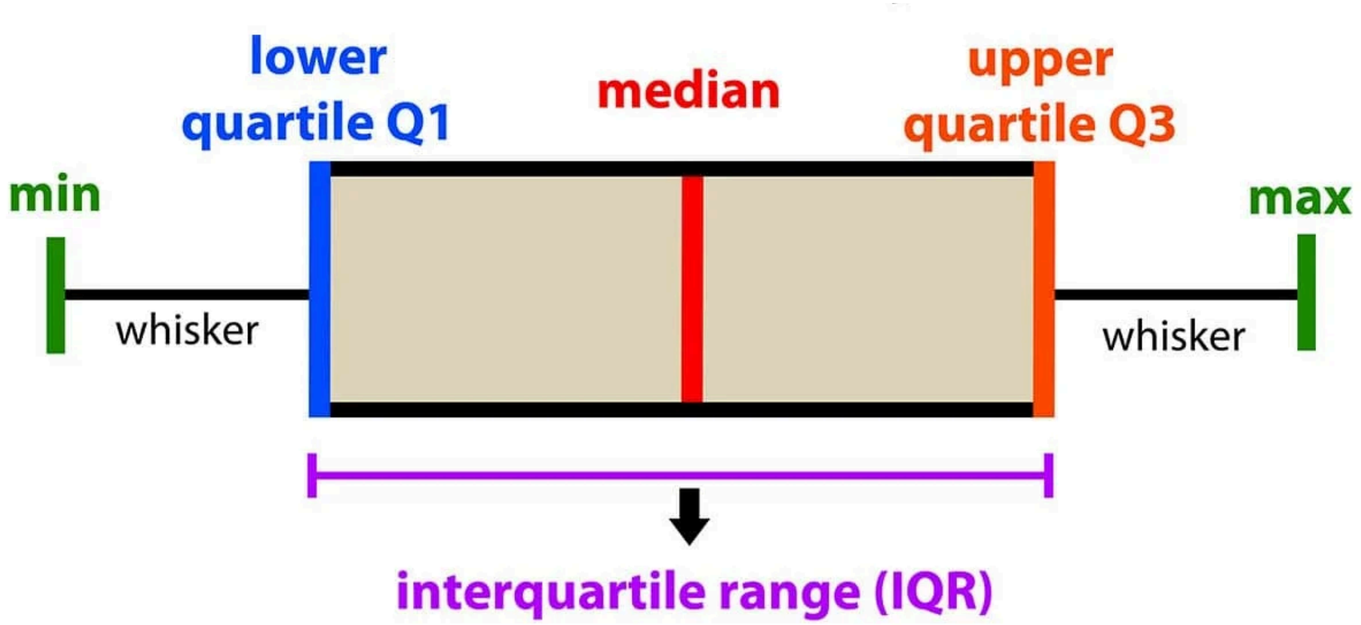
df.describe( )

- a quick summary tool
- gives important statistics about each column

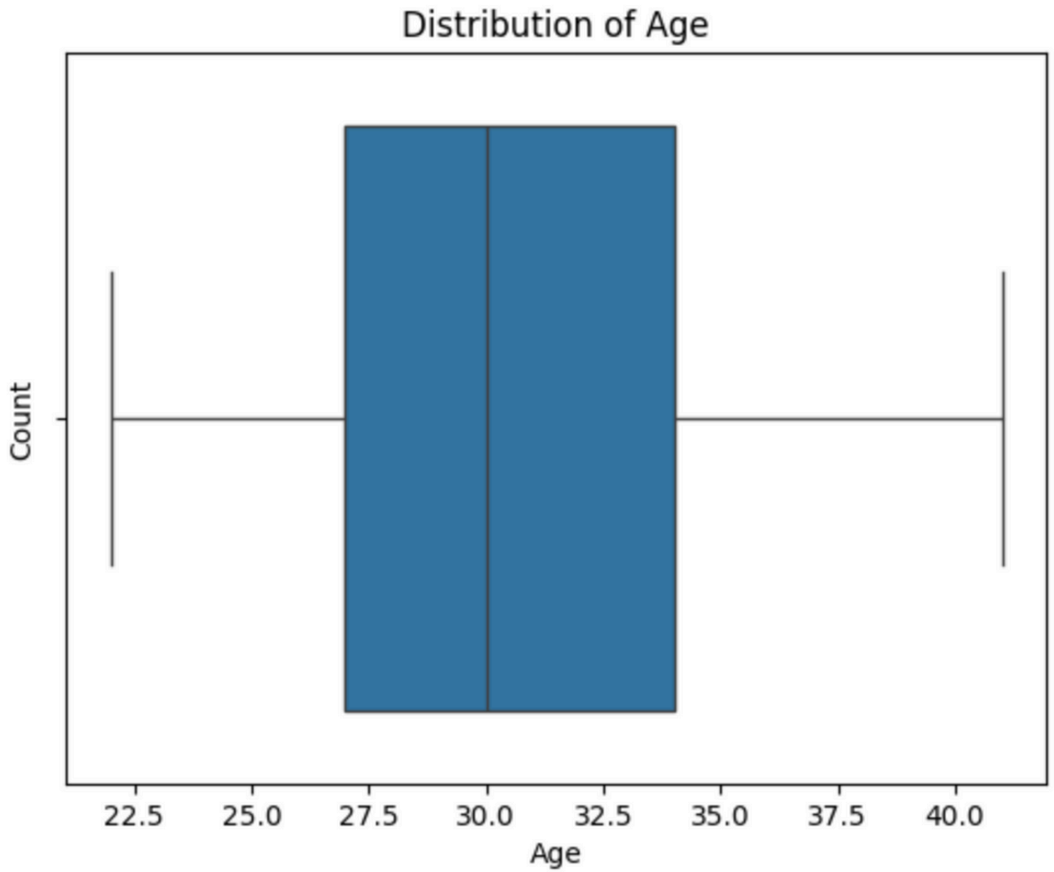
	JoiningYear	PaymentTier	Age	ExperienceInCurrentDomain	LeaveOrNot
count	3109.000000	3109.000000	3109.000000	3109.000000	3109.000000
mean	2015.099067	2.653586	30.589900	2.687038	0.385976
std	1.883810	0.608264	4.991051	1.593149	0.486903
min	2012.000000	1.000000	22.000000	0.000000	0.000000
25%	2013.000000	2.000000	27.000000	2.000000	0.000000
50%	2015.000000	3.000000	30.000000	3.000000	0.000000
75%	2017.000000	3.000000	34.000000	4.000000	1.000000
max	2018.000000	3.000000	41.000000	7.000000	1.000000

Boxplot

	Age
count	3109.000000
mean	30.589900
std	4.991051
min	22.000000
25%	27.000000
50%	30.000000
75%	34.000000
max	41.000000



Boxplot

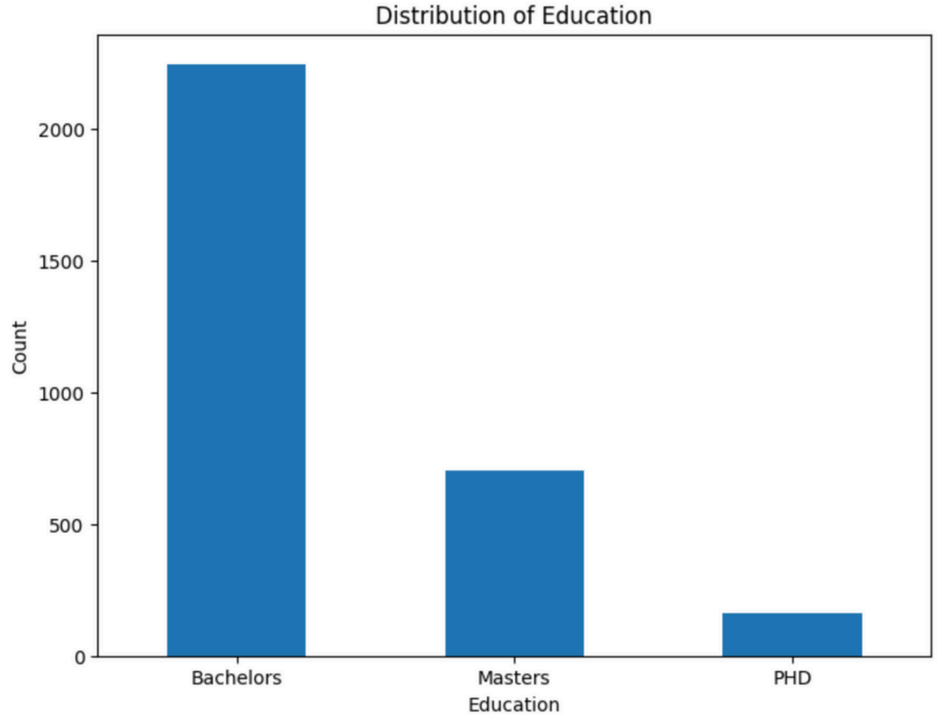


Categorical Variable Code

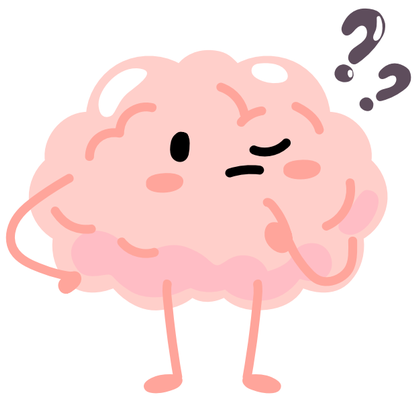
```
counts = df['Education'].value_counts()
proportions = df['Education'].value_counts(normalize=True) * 100
```

```
--- Education ---
Frequency Counts:
Education
Bachelors    2244
Masters       704
PHD           161
Name: count, dtype: int64

Proportions (%):
Education
Bachelors    72.177549
Masters      22.643937
PHD           5.178514
Name: proportion, dtype: float64
----------
```

Frequency count



Visualisation



Bivariate Analysis

WHY are bivariate analysis useful?

- Helps detect trends and patterns
- Guides decisions in marketing, research, and data analysis
- Supports building predictive models

E.g: “Does the employee’s experience in the domain affect whether they leave the company or not?”

Bivariate Analysis

2 numerical variables	2 categorical variables	1 numerical + 1 categorical
• Linear correlation • Scatterplot	• Contingency table • Chi square test	• T test

Bivariate Analysis

2 numerical variables	2 categorical variables	1 numerical + 1 categorical
• Linear correlation • Scatterplot	• Contingency table • Chi square test	• T test

What is linear correlation?

Represents the strength of a linear relationship between the 2 variables

- It’s a number between -1 and +1:
- +1 → perfect positive relationship (if one goes up, the other goes up).
- -1 → perfect negative relationship (if one goes up, the other goes down).
- 0 → no relationship at all.

Example:

- Height vs Weight → positive, negative or no correlation?
- Exercise Time vs Body Fat % → positive, negative or no correlation?
- Shoe Size vs Salary → positive, negative or no correlation?

Linear correlation in code

df.corr( )

- Looks at all the numerical columns in your DataFrame.
- Calculates the linear correlation coefficient between each pair of columns.
- Returns a correlation matrix

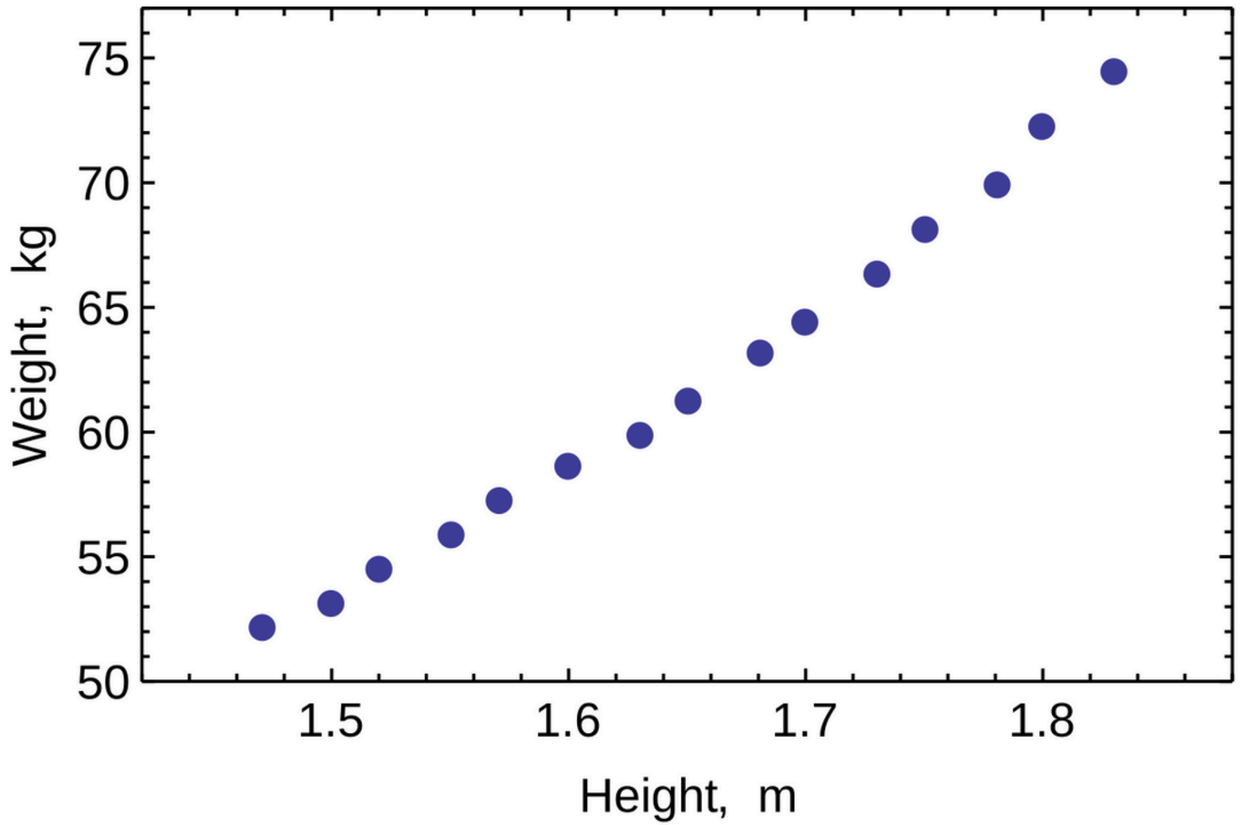
Linear correlation in code

```
# Select numerical columns
numerical_cols = df.select_dtypes(include=np.number).columns

# Linear Correlation
correlation_matrix = df[numerical_cols].corr()
print("Linear Correlation Matrix:")
display(correlation_matrix)
```

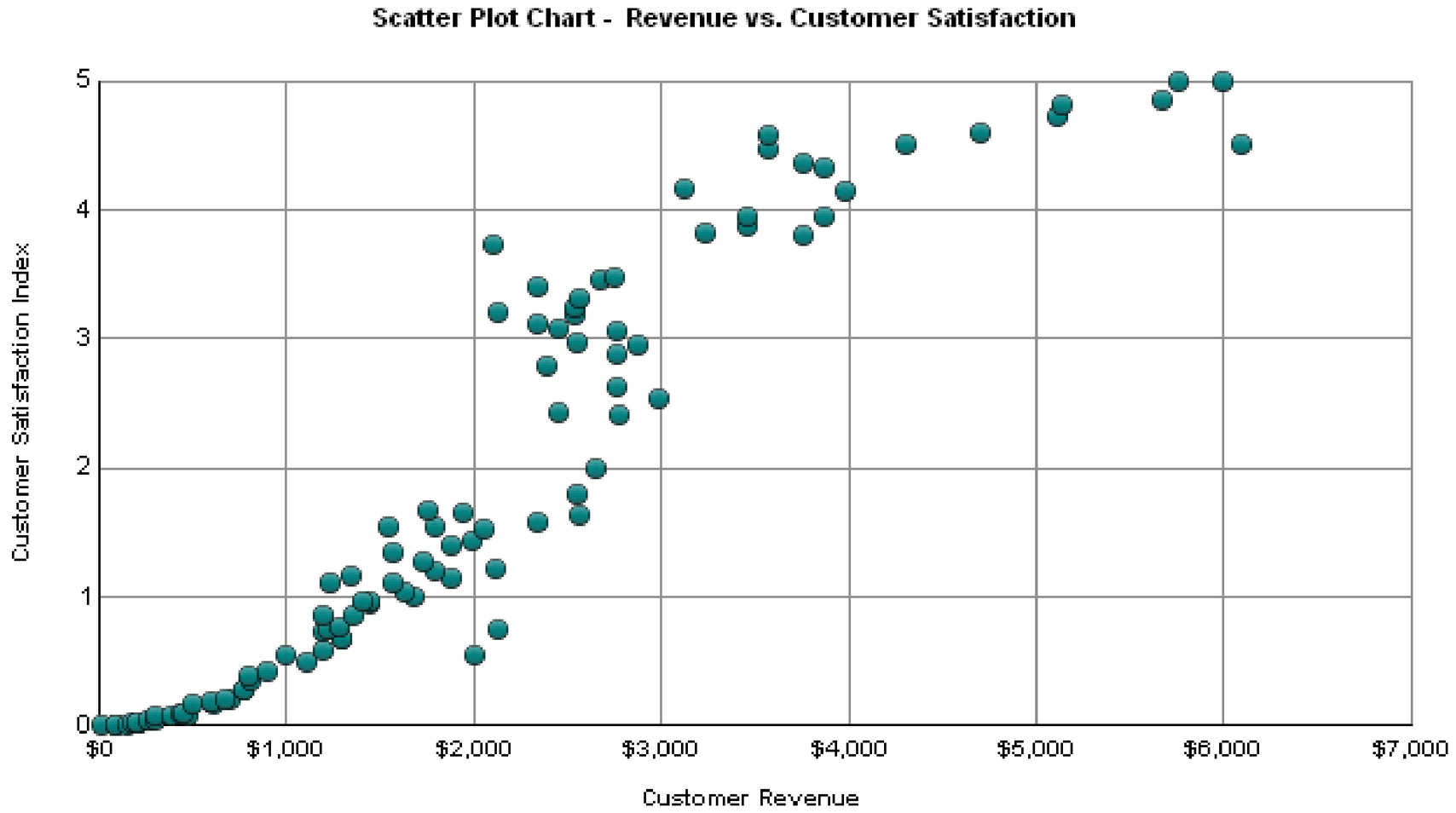
Linear Correlation Matrix:

	JoiningYear	PaymentTier	Age	ExperienceInCurrentDomain	LeaveOrNot
JoiningYear	1.000000	-0.063545	0.013699	-0.030512	0.161402
PaymentTier	-0.063545	1.000000	0.055145	-0.003672	-0.127437
Age	0.013699	0.055145	1.000000	-0.073686	-0.097562
ExperienceInCurrentDomain	-0.030512	-0.003672	-0.073686	1.000000	-0.016361
LeaveOrNot	0.161402	-0.127437	-0.097562	-0.016361	1.000000

Scatterplot



Scatterplot



Bivariate Analysis

2 numerical variables	2 categorical variables	1 numerical + 1 categorical
• Linear correlation • Scatterplot	• Contingency table • Chi square test	• T test

What is a contingency table?

- A contingency table shows the frequency of combinations of two categorical variables.
- Instead of just counting one column (like .value_counts( )), it counts how often values from two columns occur together.

Example:

- “How many men & women bought Product A vs Product B?”
- “How many students passed or failed by gender?”

How to code a contingency table?

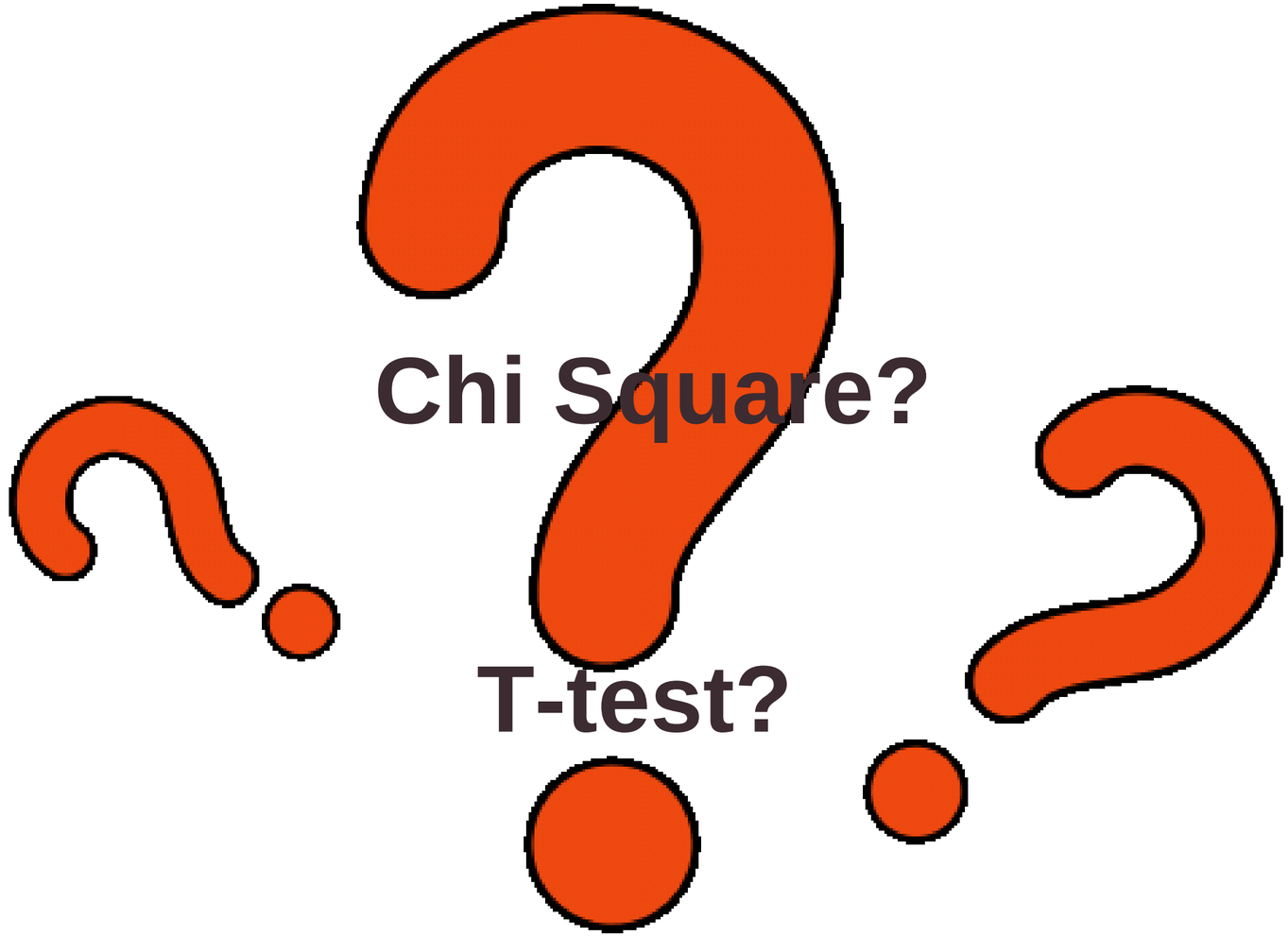
pd.crosstab(row_variable, column_variable)

- row_variable → goes on the rows of the table
- column_variable → goes on the columns of the table

```
# contingency table
contingency_table = pd.crosstab(df['Education'], df['Gender'])
print("Contingency Table:")
display(contingency_table)
```

```
Contingency Table:
```

Gender	Female	Male
Education		
Bachelors	973	1271
Masters	295	409
PHD	64	97


Chi Square?
T-test?

Basics of Statistics

1. WHAT IS **HYPOTHESIS TESTING**?

Hypothesis testing is how we check if our observations are real or just random

Null hypothesis (H_0)

- Nothing special is happening
- Example: “The new teaching method has no effect on students’ scores.”

Alternative hypothesis (H_1)

- Something is happening
- Example: “The new teaching method improves scores.”

Basics of Statistics

2. WHAT IS **SIGNIFICANCE LEVEL (α)**?

α : “How much risk of being wrong am I willing to take?”

Common choice: **α = 0.05 (5%)** → you’ll only reject the null hypothesis if there’s less than a 5% chance your result is random.

Basics of Statistics

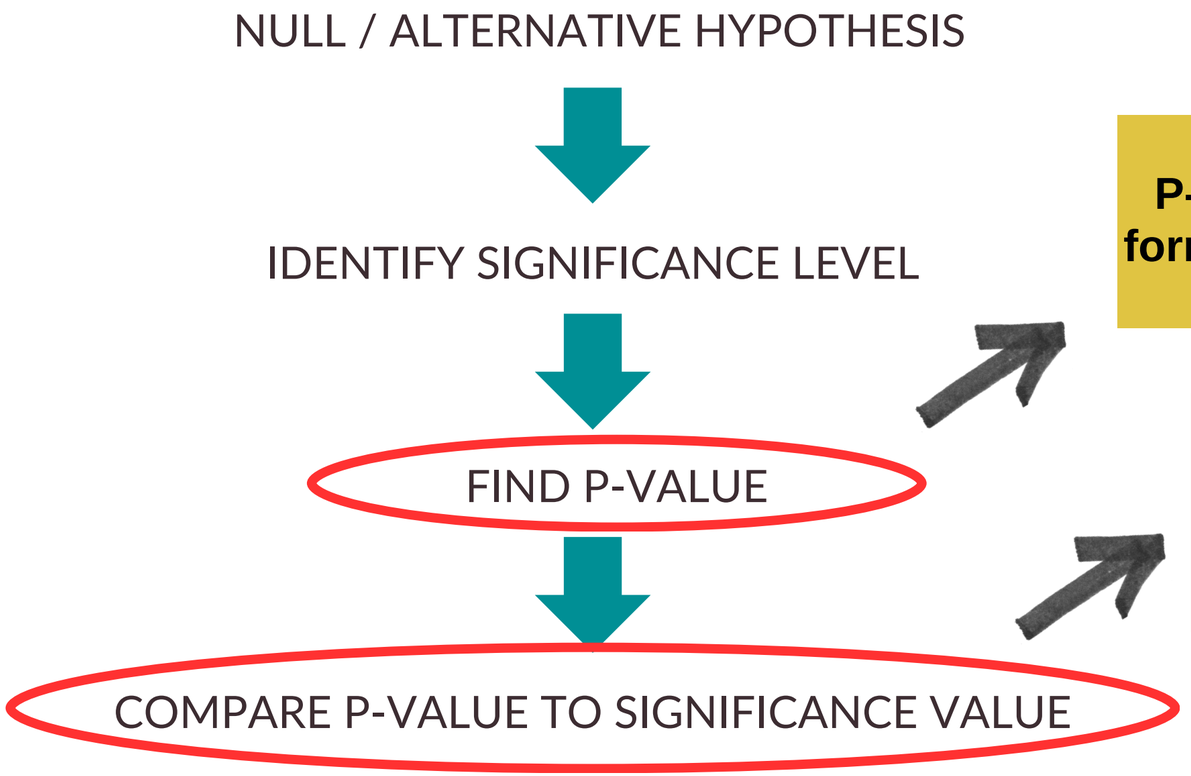
3. WHAT IS **P-VALUE**?

a number that helps you decide if your result is “real” or just happened by chance

- Small p-value → If H_0 were really true, getting this result (or something more extreme) would be very unlikely. So we have evidence against H_0.
- Large p-value → If H_0 were true, getting this result isn’t unusual. So there’s no strong evidence against H_0.

Basics of Statistics

4. HOW DOES ALL OF THIS RELATE TO EACH OTHER?



P-values are obtained by applying the formula of the statistical test to our data.

- **$p \leq \alpha \rightarrow$ reject H_0 (evidence supports alternative)**
- **$p > \alpha \rightarrow$ fail to reject H_0 (not enough evidence)**

Let’s now look at chi-square test

What do we use chi-square test for?:

- **Categorical** data (things in groups like “Yes/No” or “Male/Female”).
- Question it answers: “Are these two categories **related**, or are they **independent**?”

Motive of chi-square test

The aim of this chi-square test is to conclude whether the two variables are related to each other not.

- Null hypothesis: There is no relation between ‘Gender’ and ‘Education’.
- Alternate hypothesis: There is a significant relationship between ‘Gender’ and ‘Education’.

Chi-Square Test Code

```
#!pip install scipy

from scipy.stats import chi2_contingency #import package for chi-sq

# performing chi-sq
_, p, _, _ = chi2_contingency(contingency_table)
```

This is a function from scipy.stats that performs a Chi-square test on a **contingency table**

Contingency Table:

Gender	Female	Male
Education		
Bachelors	973	1271
Masters	295	409
PHD	64	97

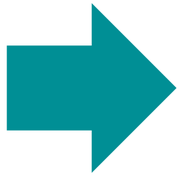
p: p-value (most impt!)

- If p ≤ 0.05 → likely the variables are related
- If p > 0.05 → likely the variables are independent

Chi-Square Test Code

Formatting your output

```
# performing chi-sq
_, p, _, _ = chi2_contingency(contingency_table)

print("\np-value:", p)

alpha = 0.05

print("\np value is " + str(p))
if p <= alpha:
    print('\nDependent (reject H0)')
else:
    print('\nIndependent (H0 holds true)')
```



```
p-value: 0.5691690269057504

p value is 0.5691690269057504

Independent (H0 holds true)
```

Insights from chi-square

```
p-value: 0.5691690269057504

p value is 0.5691690269057504

Independent (H0 holds true)
```

WHAT does this tell us?

- p > 0.05
- We cannot reject the null hypothesis => evidence supports null hypothesis
- **Conclusion: There is no relation between 'Gender' and 'Education'**

Bivariate Analysis

2 numerical variables	2 categorical variables	1 numerical + 1 categorical
• Linear correlation • Scatterplot	• Contingency table • Chi square test	• T test

What is T-test?

- Used for comparing the means (averages) of two groups.
- Question it answers: “Are the averages of these two groups really different, or is it just random?”

Example:

- Do **men** (group 1) and **women** (group 2) have different average heights?
- t-test compares the two averages and says if the difference is statistically significant.

T-Test on Age & LeaveOrNot

Motive: Performing independent t-test to compare the means of `Age` between employees who left and those who stayed.

- Null Hypothesis (H0): There is no significant difference in the mean of the two groups.
- Alternative Hypothesis (H1): There is a significant difference in the mean of the two groups.

T-Test Code

```
from scipy.stats import ttest_ind
```

```
left = df[df['LeaveOrNot'] == 1]
stayed = df[df['LeaveOrNot'] == 0]

# independent t-test for 'Age'
ttest_age = ttest_ind(left['Age'], stayed['Age'])
print(f"Independent t-test p-value for Age: {ttest_age.pvalue}")
```

What is the output?

```
Independent t-test p-value for Age: 5.0164133006641345e-08
```

- **p-value** → tells you if the difference is statistically significant:
  - p ≤ 0.05 → means the means are significantly different.
  - p > 0.05 → means the means are not significantly different.

T-Test Code

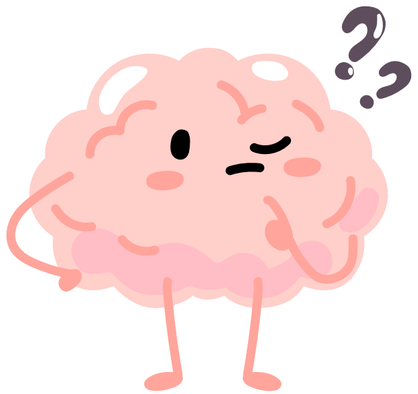
```
# output
alpha = 0.05

if ttest_age.pvalue <= alpha:
    print('Therefore, reject H0')
else:
    print('Therefore, H0 holds true')
```

*e-08 means x10^-8

```
p value for Age is 5.0164133006641345e-08
Therefore, reject H0
```

Insights from T-test

The statistically significant differences found by the t-tests suggest that age **are associated** with whether an employee leaves or stays in this dataset.

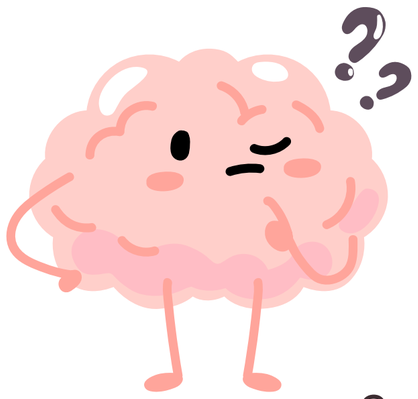


Multivariate Analysis

WHY do we need multivariate analysis?

1. Real-world problems are complex, not one-dimensional

- In reality, many factors interact at the same time.
- Example: Whether an employee leaves a company isn’t just about salary. It could also depend on age, workload, career growth, work environment, and education all together.

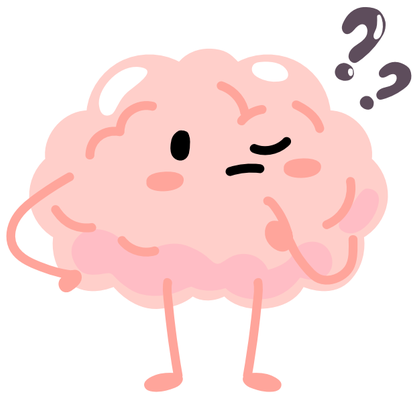


Multivariate Analysis

WHY do we need multivariate analysis?

2. Looking at only one factor can be misleading

- If we only check salary, we might wrongly assume “higher pay = no one leaves.”
- But maybe younger employees leave more often regardless of pay, or education level influences loyalty.



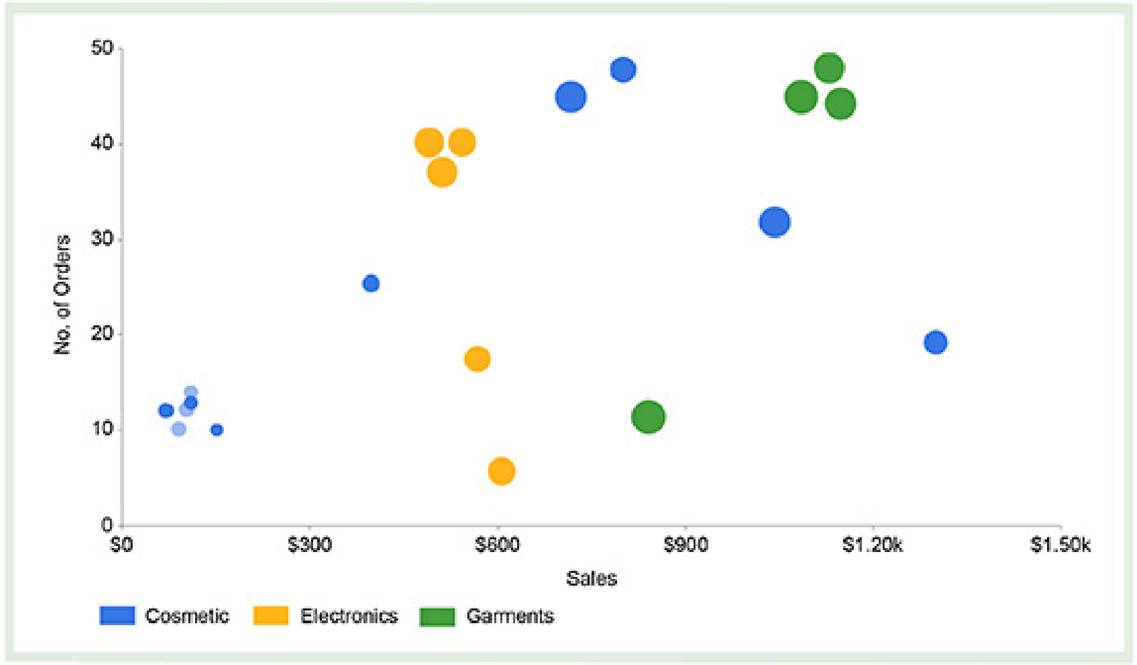
Multivariate Analysis

WHY do we need multivariate analysis?

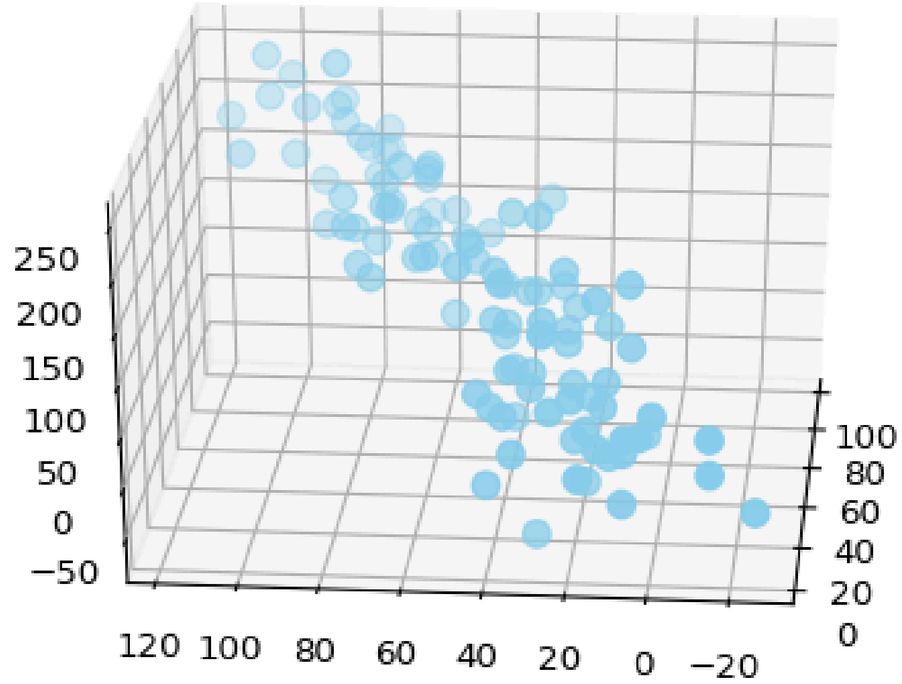
Imagine you’re judging a cooking competition. If you only look at presentation, you might crown the wrong winner. To be fair, you need to consider taste, creativity, texture, and presentation together.

That’s exactly what multivariate analysis does — it makes sure you’re not missing important pieces.

Multivariate Analysis

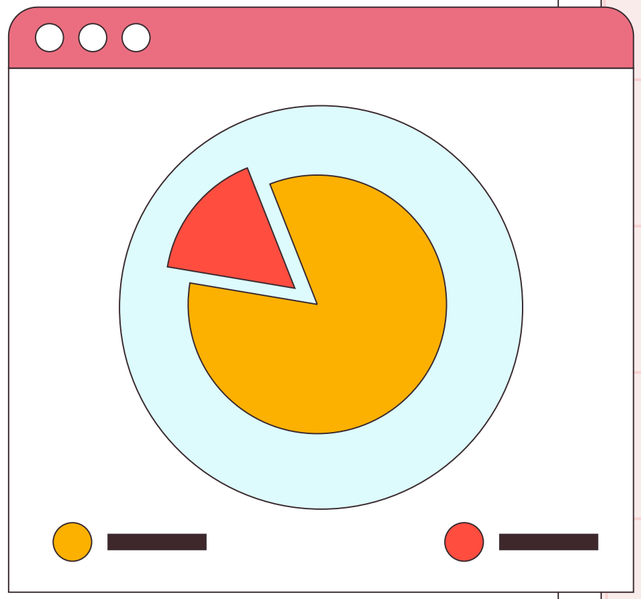


2D



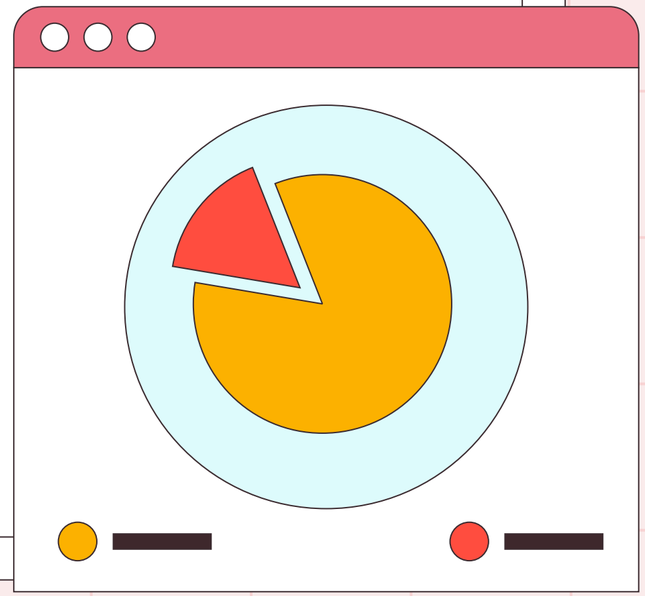
3D

End of Part 2!



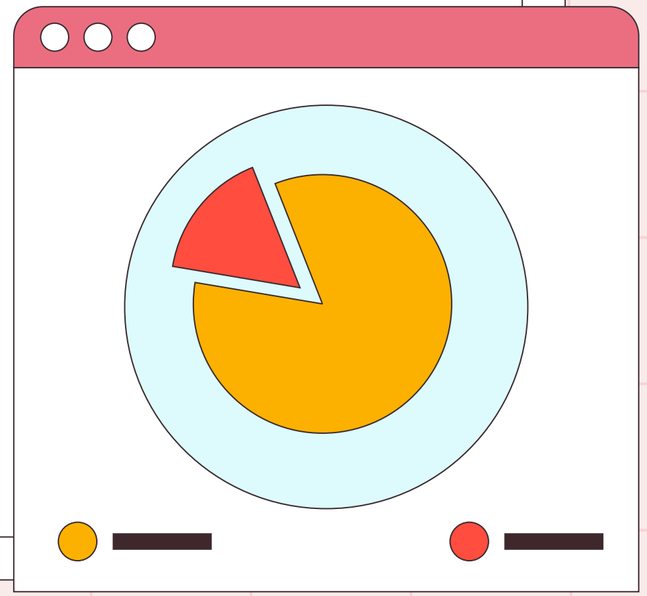
04

Data Visualization



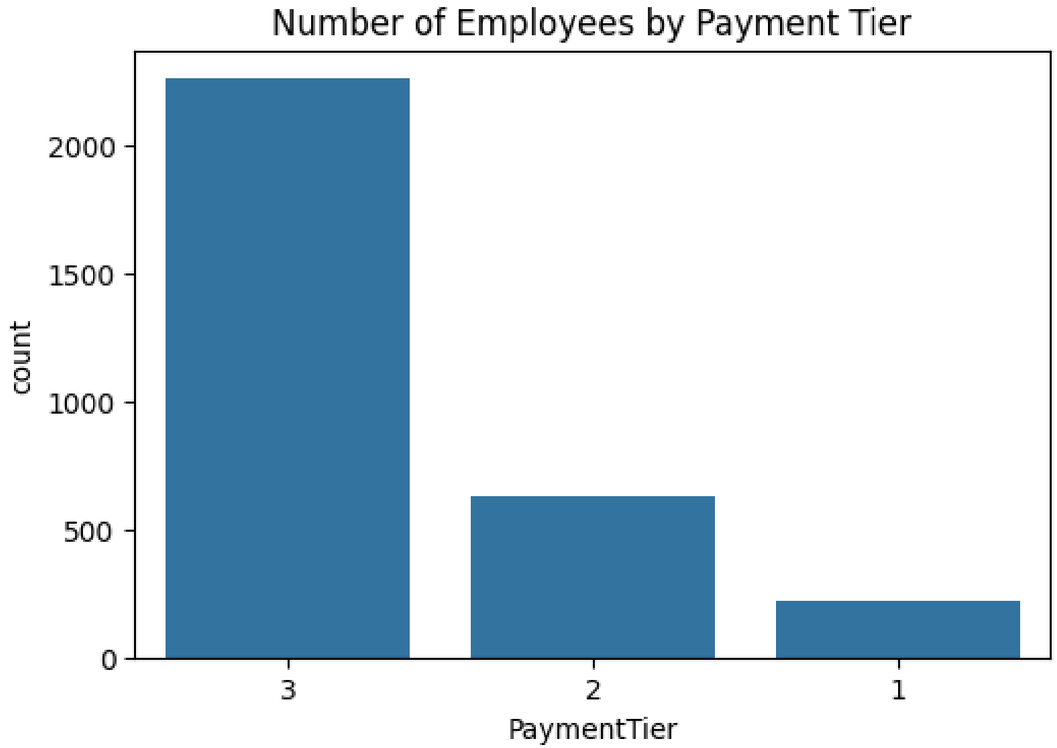
DataViz

1. We’ll continue to use employee_cleaned.csv for this part, please upload it into workshop_code_part3
2. Kindly do not run all the cells. We will run through them chunk by chunk



Contents

Why do you need visualizations?



- See Distribution → understand how variables look (skewness, outliers)
- Spot anomalies → detect errors or rare cases
- Compare groups → Explore categorical differences
- Reveal relationships → Identify correlations, trend
- Communicate stories → Make results easy for others to digest

Choosing the RIGHT Visual

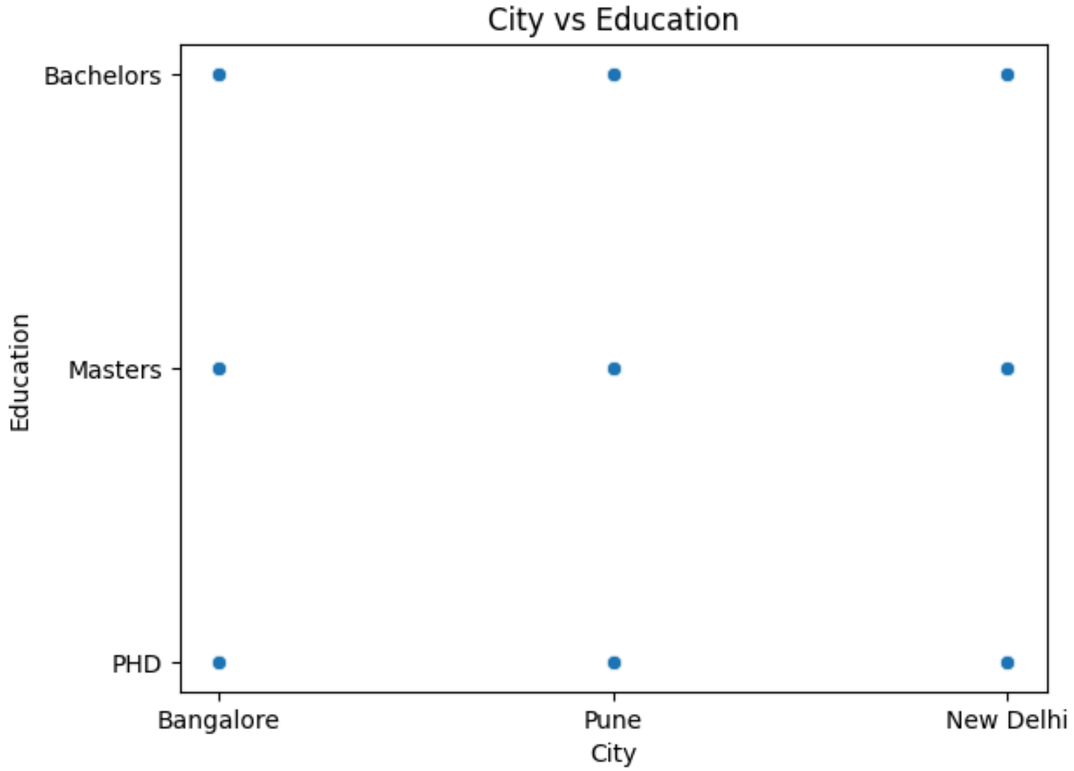
CHART MUST MATCH THE QUESTION + DATA TYPE

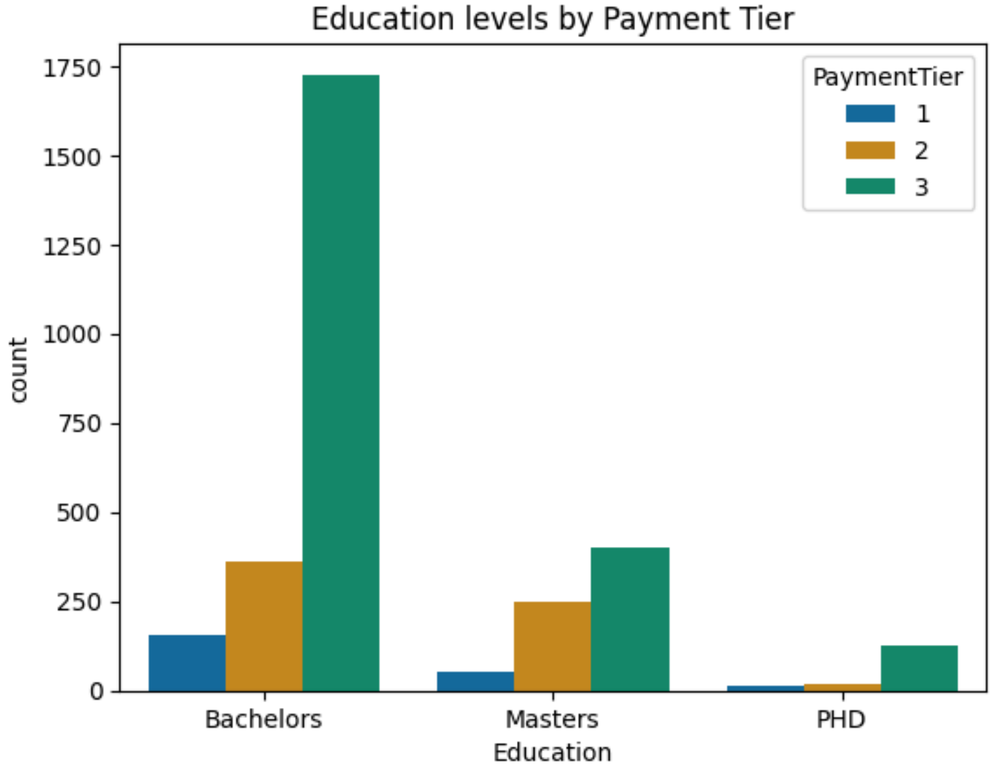
rly cool github for bad practices

Data Type	Best Visuals	WORST Visuals
Numerical → Numerical	Scatterplot, Correlation Plot, Heatmap	Pie chart, Bar chart of every value
Categorical → Numerical	Boxplot, Violin Plot, Barplot	Scatterplot
Categorical → Categorical	Grouped bar, Mosaic, Heatmap	Line Chart
Univariate	Histogram, Boxplot, Countplot	3D plots for no reason
Multivariate	Bubble chart, facet plot, scatterplot matrices	stacker bar chart, pie chart

Choosing the RIGHT Visual

CHART MUST MATCH THE QUESTION + DATA TYPE

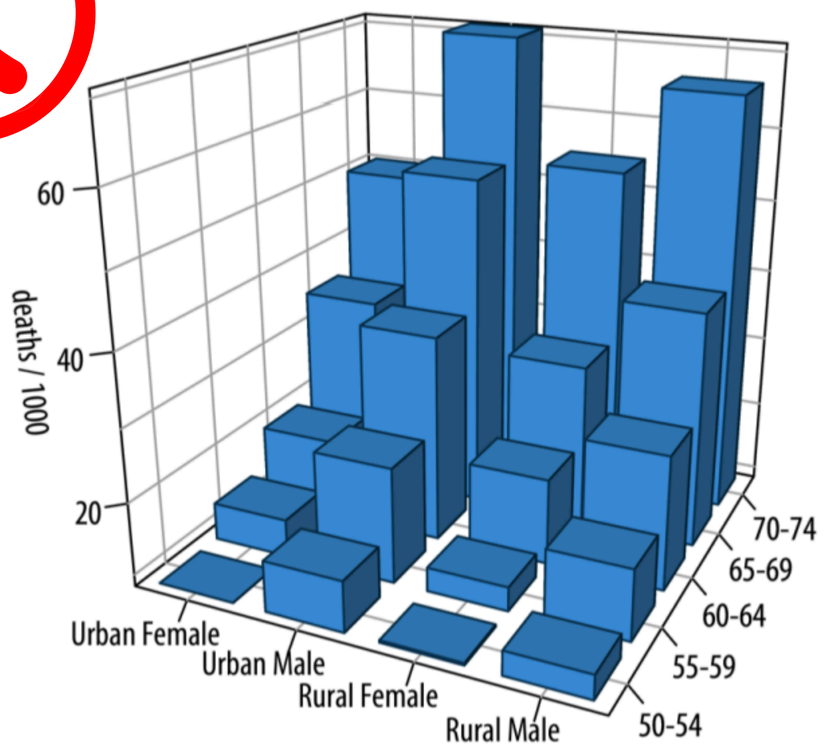


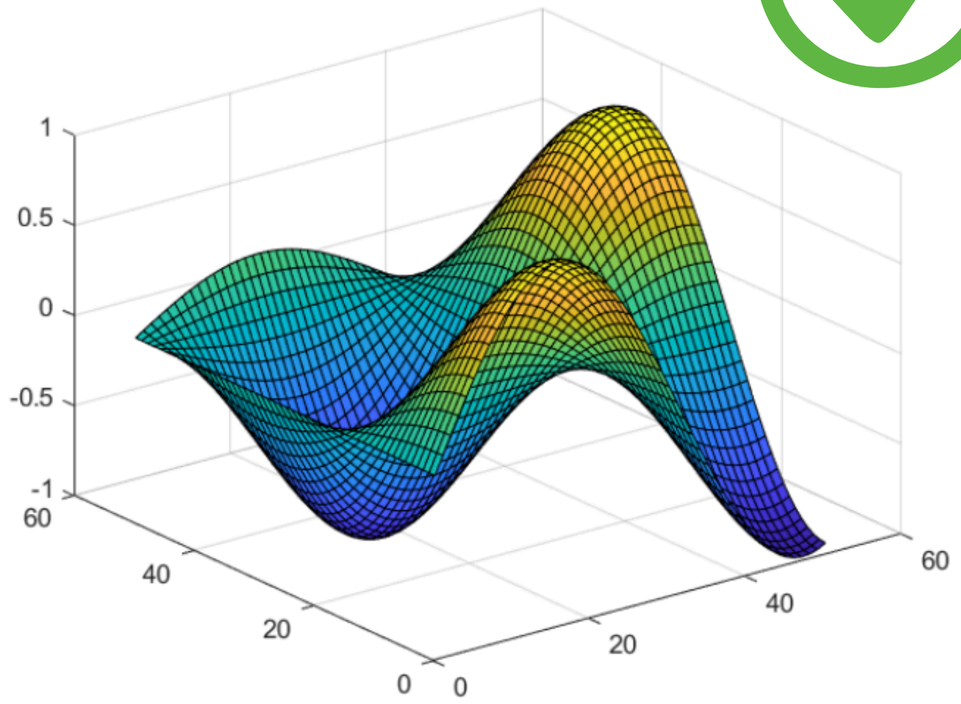


Choosing the RIGHT Visual

CHART MUST MATCH THE QUESTION + DATA TYPE

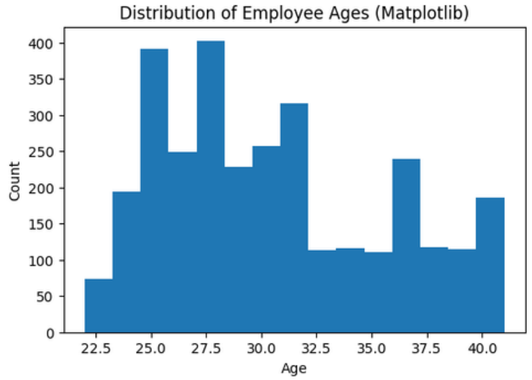
bad





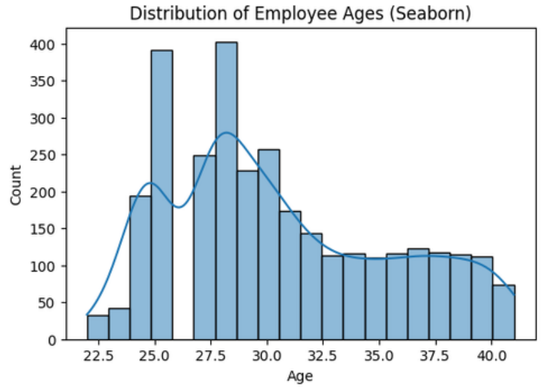
All about the Visuals

Libraries



Matplotlib

- base plotting library
- customizable but lots to remember



Seaborn

- built on Matplotlib
- easier (and prettier) for statistical plots

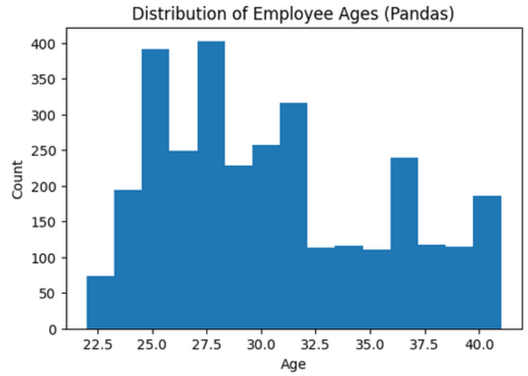
Pandas

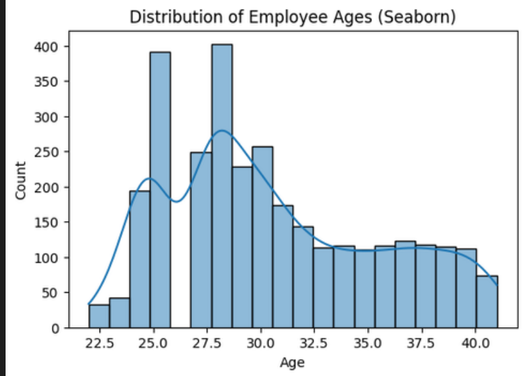
- quick plots directly from dataframes

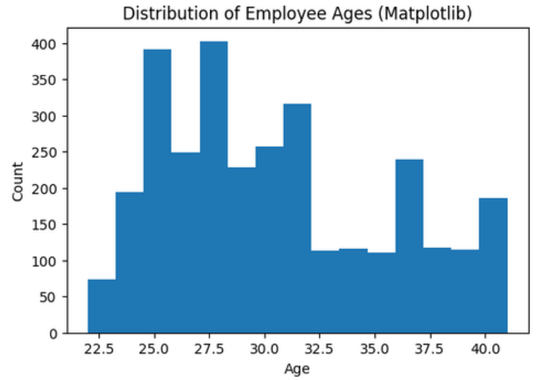
All about the Visuals

Libraries

```python
# matplotlib show
plt.figure(figsize=(6,4))
plt.hist(employees["Age"], bins=15)#, color="skyblue", edgecolor="black")
plt.title("Distribution of Employee Ages (Matplotlib)")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()

# seaborn show
plt.figure(figsize=(6,4))
sns.histplot(employees["Age"], kde=True)
plt.title("Distribution of Employee Ages (Seaborn)")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()

# pandas show
employees["Age"].plot(kind="hist", bins=15, figsize=(6,4))#, color="skyblue", edgecolor="black")
plt.title("Distribution of Employee Ages (Pandas)")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()
```

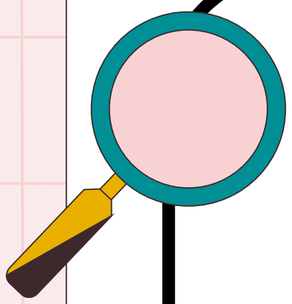






All about the Visuals

Histogram



What is it?

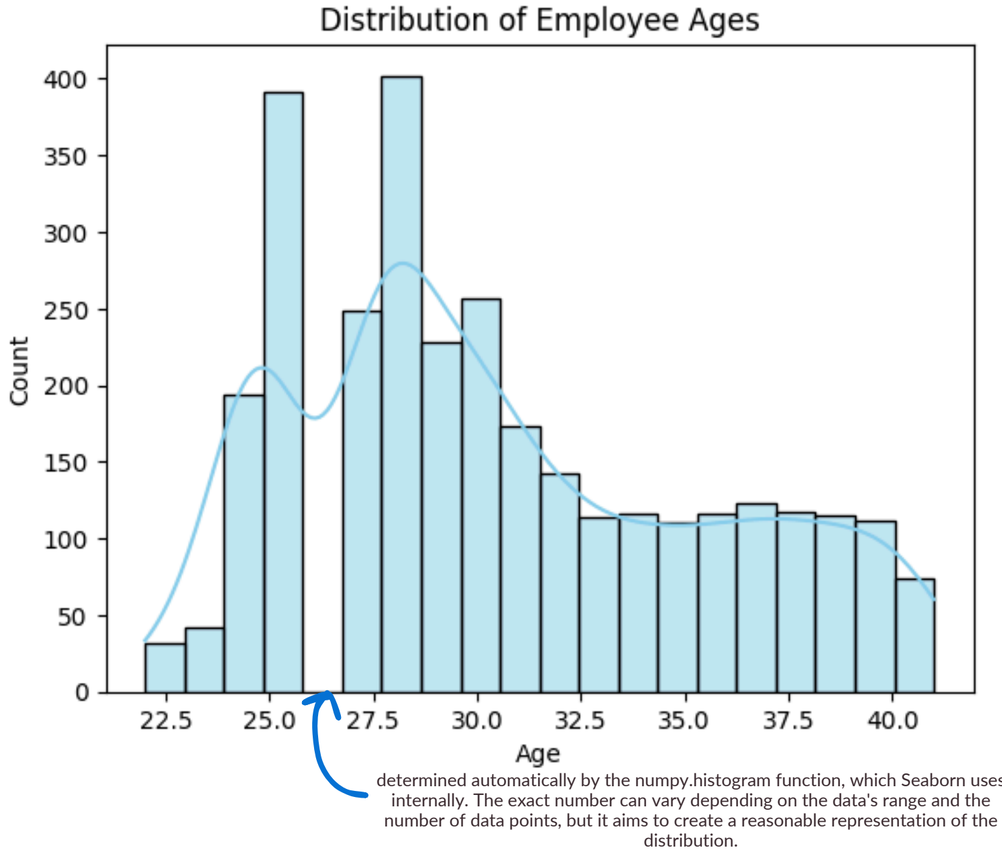
- a chart that groups values into bins and shows their frequency
- used to understand spread, skewness and outliers
- X-axis: value range
- Y-axis: frequency

```
1 # Histogram
2
3 sns.histplot(employees["Age"], kde=True, color="skyblue")
4 plt.title("Distribution of Employee Ages")
5 plt.xlabel("Age")
6 plt.ylabel("Count")
7 plt.show()
```

draws the histogram

smooth curve for distribution

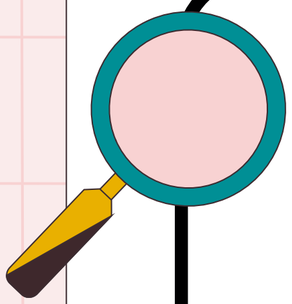
note that the y parameter is not needed



Insight = Most employees are in their late 20s”

All about the Visuals

Bar Chart/Countplot (for categories)



What is it?

- a chart that compares values across categories
- used to see diversity, balance or dominant groups
- X-axis: categories
- Y-axis: count

```
1 # Bar Chart
2
3 sns.countplot(x="Gender", data=employees, palette="pastel", hue="Gender")
4 plt.title("Gender Distribution of Employees")
5 plt.xlabel("Gender")
6 plt.ylabel("Count")
7 plt.show()
```

counts how many times each category appears

visual effect

note that the y parameter is not needed

Insight = Male employees outnumber female employees

All about the Visuals

Boxplot

What is it?

- a chart that shows the spread of numerical data across categories
- used to compare distributions, detect outlier and check medians
- Box= middle 50% of values
- Line inside = median
- Dots = Outliers


```
1 # Boxplot
2
3 sns.boxplot(x="PaymentTier", y="ExperienceInCurrentDomain", data=employees, palette="Set2")
4 plt.title("Experience by Payment Tier")
5 plt.xlabel("Payment Tier")
6 plt.ylabel("Years of Experience")
7 plt.show()
```

categories

numerical variable

Insight = Higher payment tiers dont always mean more experience

All about the Visuals

Grouped Barplot

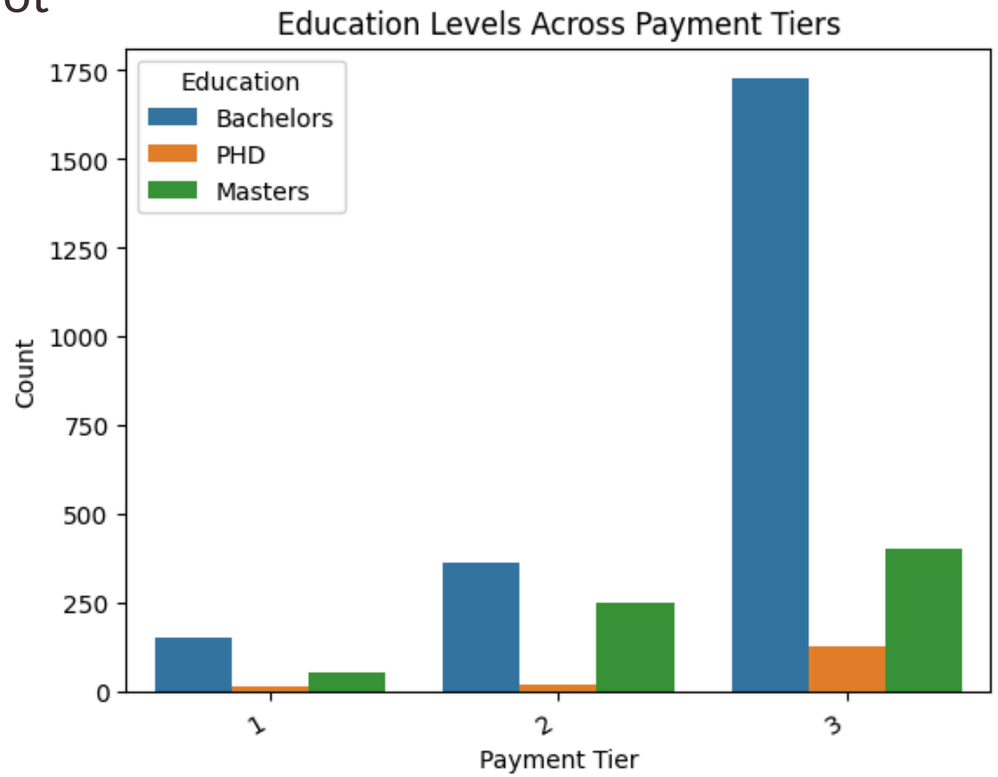
What is it?

- a chart that lets us compare counts across two categories
- used to see relationships between categorical features

```
1 # Grouped Barplot
2
3 sns.countplot(x="PaymentTier", hue="Education", data=employees)
4 plt.title("Education Levels Across Payment Tiers")
5 plt.xlabel("Payment Tier")
6 plt.ylabel("Count")
7 plt.xticks(rotation=30, ha="right")
8 plt.show()
```

split bars by education

rotate labels on x-axis



Insight = Those with a bachelors degree tend to dominate across tiers

All about the Visuals

Line Plot

What is it?

- a chart that connects individual data points with straight lines
- used to show trends over time or relationship between continuous variables
- X= time, Y=numeric value
- Connecting lines reveal trends or patterns

```
1 # Line Plot
2
3 employees_per_year = employees["JoiningYear"].value_counts().sort_index()
4 plt.figure(figsize=(6,4))
5 sns.lineplot(x=employees_per_year.index, y=employees_per_year.values, marker="o", color="skyblue")
6 plt.title("Employees Joined by Year (Seaborn)")
7 plt.xlabel("Joining Year")
8 plt.ylabel("Number of Employees")
9 plt.show()
```

counts employees per year sorted chronologically

marks each datapoint with a circle

Insight = Number of employees joining rises sharply in 2017 and drops in 2018

All about the Visuals

Pie Chart

What is it?

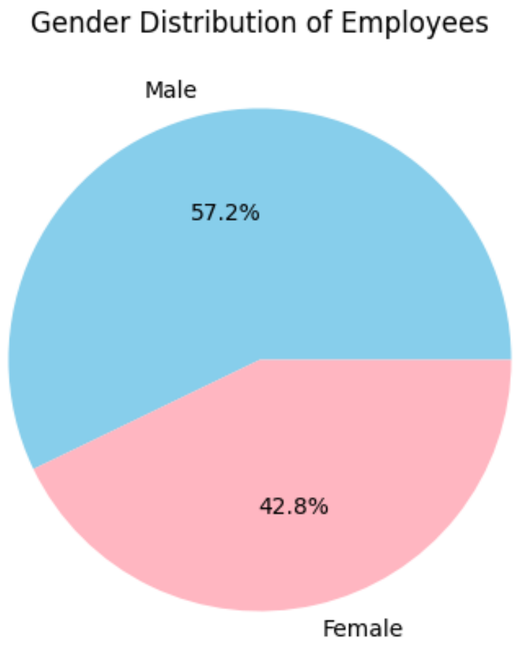
- a circular chart divided into slices, each slice representing the proportion of a category relative to the whole
- used to examine proportions
- each slice's angle = category's percentage
- best for 2-5 categories MAX

SEABORN DOES NOT HAVE A PIE CHART FUNCTION

```
# Pie Chart

gender_counts = employees["Gender"].value_counts()
plt.figure(figsize=(5,5))
plt.pie(gender_counts, labels=gender_counts.index, autopct="%1.1f%%", colors=["skyblue","lightpink"])
plt.title("Gender Distribution of Employees")
plt.show()
```

counts Male vs. Female

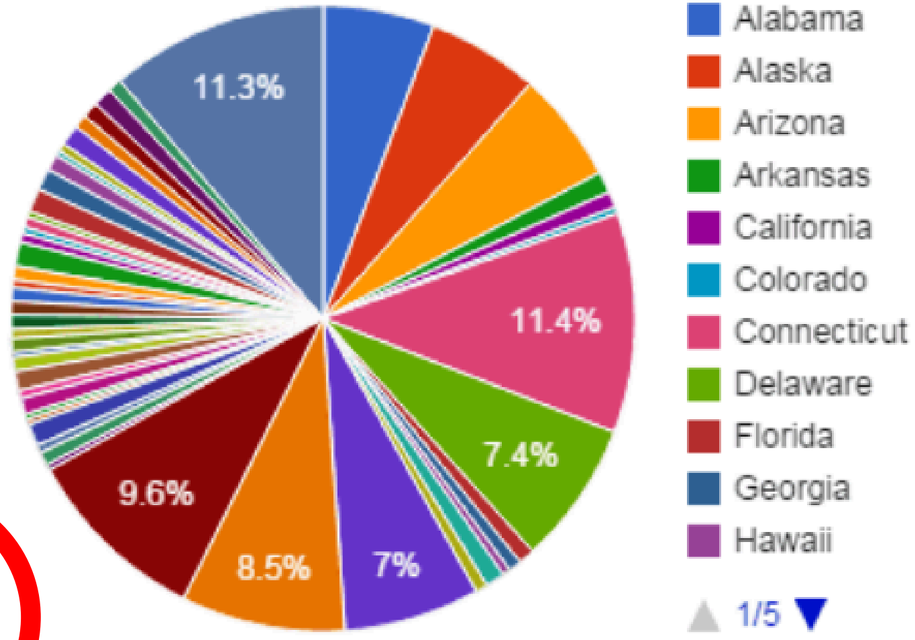
↳ show percentages with 1 decimal

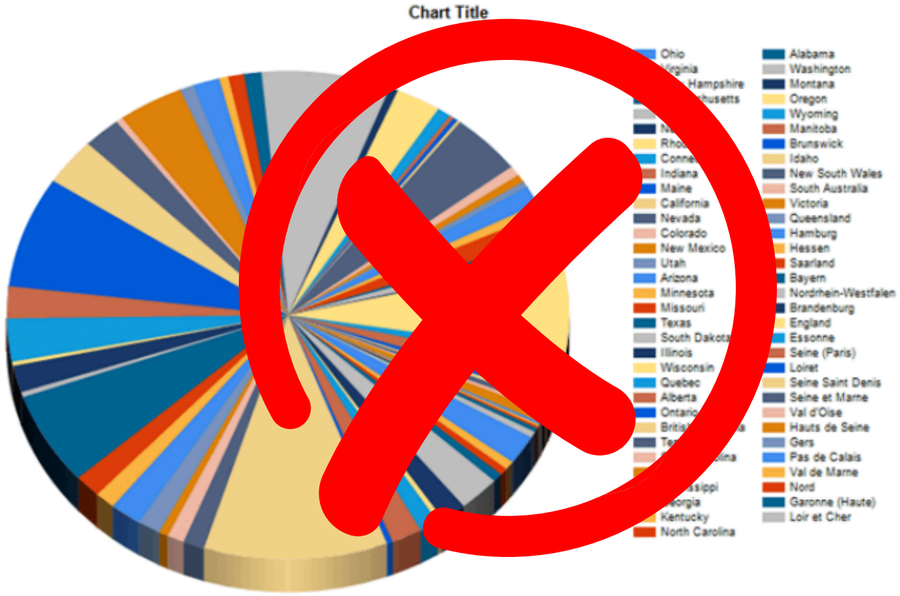


Insight = Male employees make up about 57% of the workforce while female employees make up about 43%, showing a gender imbalance

All about the Visuals

Pie Chart





Formatting for Clarity

Principles of Clean Visualization

Always label axes and add a descriptive title

Avoid clutter: remove chartjunk like unnecessary gridlines, 3D effects, etc

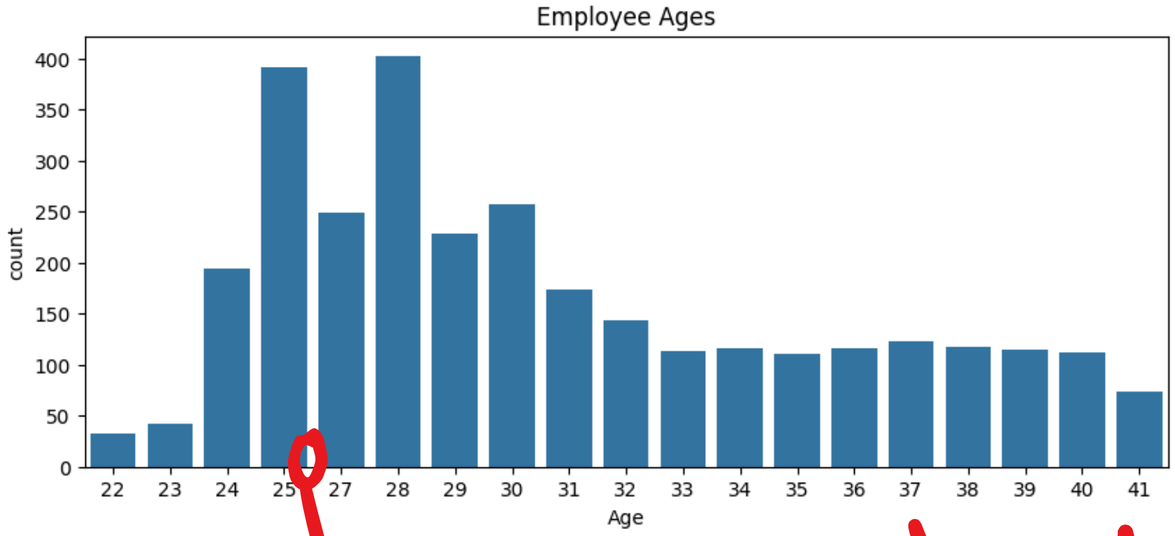
Keep colours consistent: one palette, avoid rainbows, colour-blind friendly

Highlight the key comparison (eg. annotate median line)

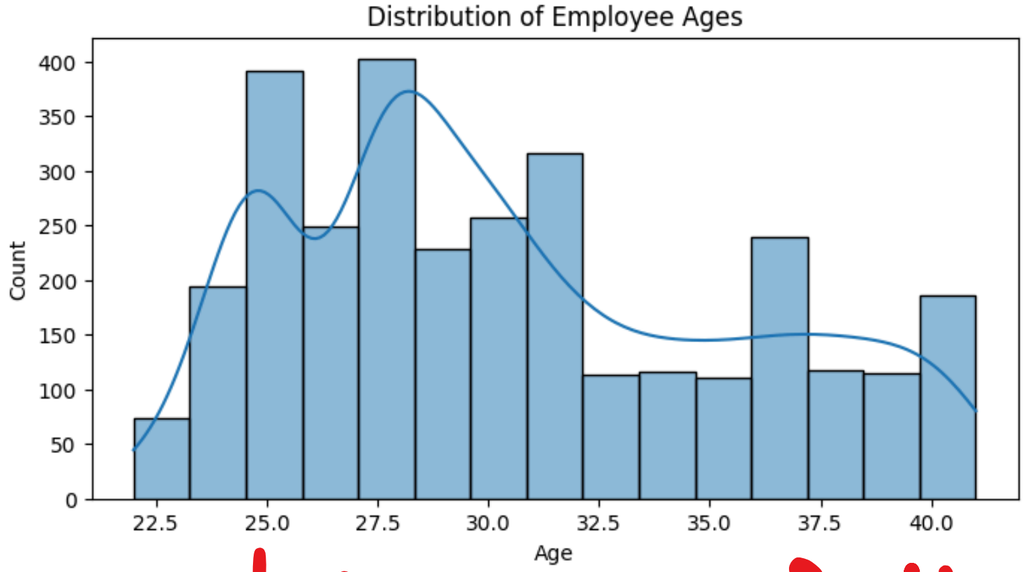
Use readable scales (don’t squash/expand axis artificially)

Formatting for Clarity

Principles of Clean Visualization



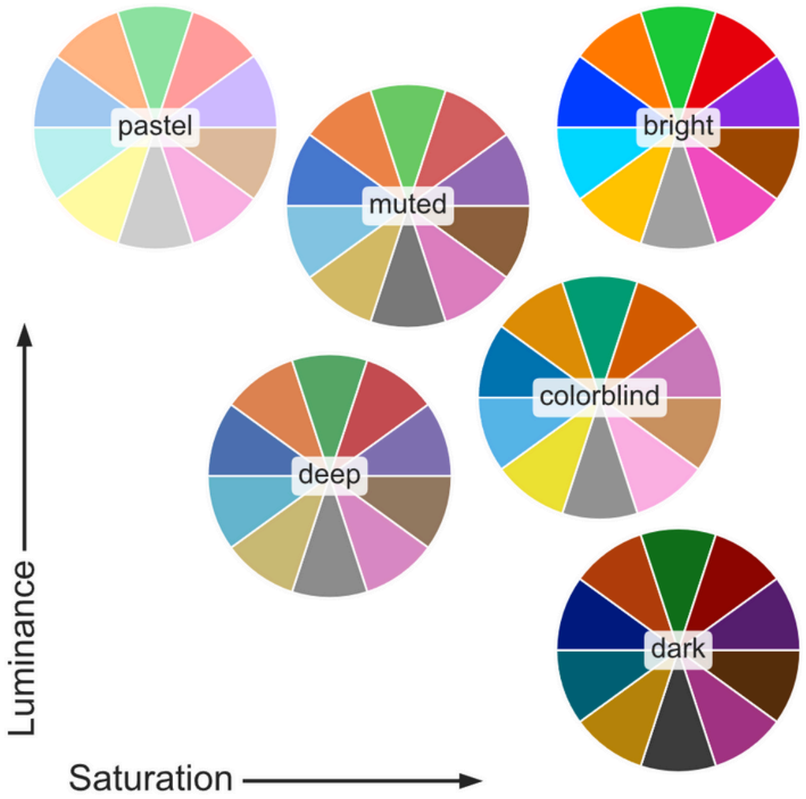
↳no 26 ⇒gap explained



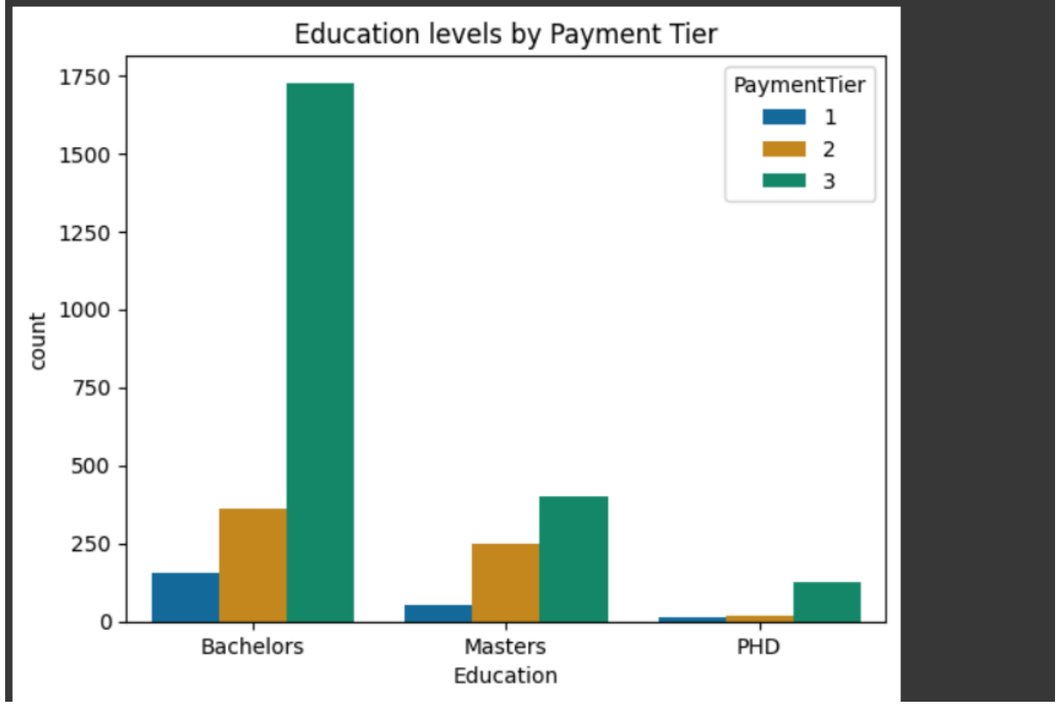
↳bins=15 specified here

Formatting for Clarity

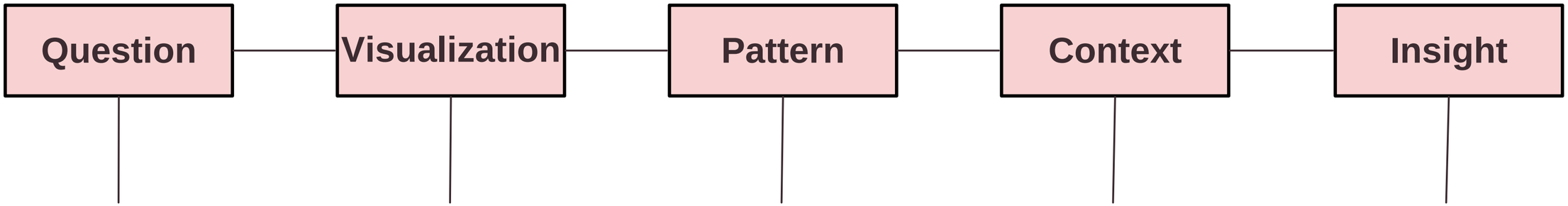
Principles of Clean Visualization




```python
# grouped barplot of education across payment tier
sns.countplot(x="Education",hue="PaymentTier",data=employees,palette="colorblind")
plt.title("Education levels by Payment Tier")
plt.show()
```



Building a Story from Visuals



Start with a question

Do higher payment tiers correspond to more domain experience?

Select the right chart

Boxplot → Experience in Current Domain vs Payment Tier

Describe the pattern

Payment tier isn't strongly separated by experience, employees with both low and high experience are present in any tier

Add Context

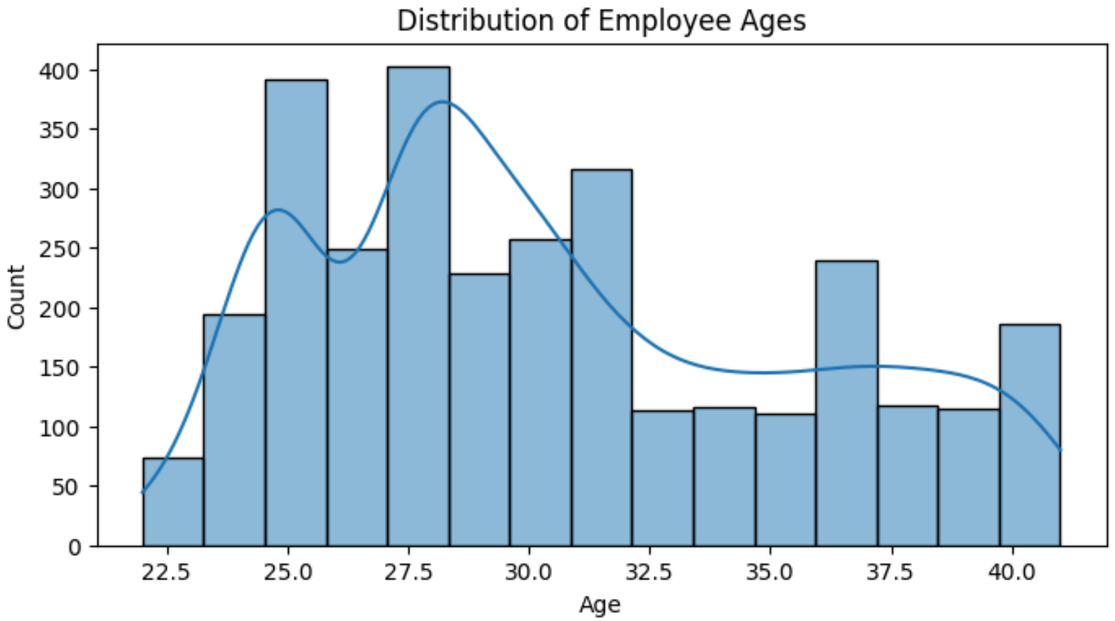
- Median Line is around 3 years across all tiers
- The spread looks pretty similar with some employees having 0 and 7 years

Conclude with an insight

Payment Tiers may depend on other factors → maybe performance, city or education

Visualization Interpretations

Move from chart → insight



Not enough to say

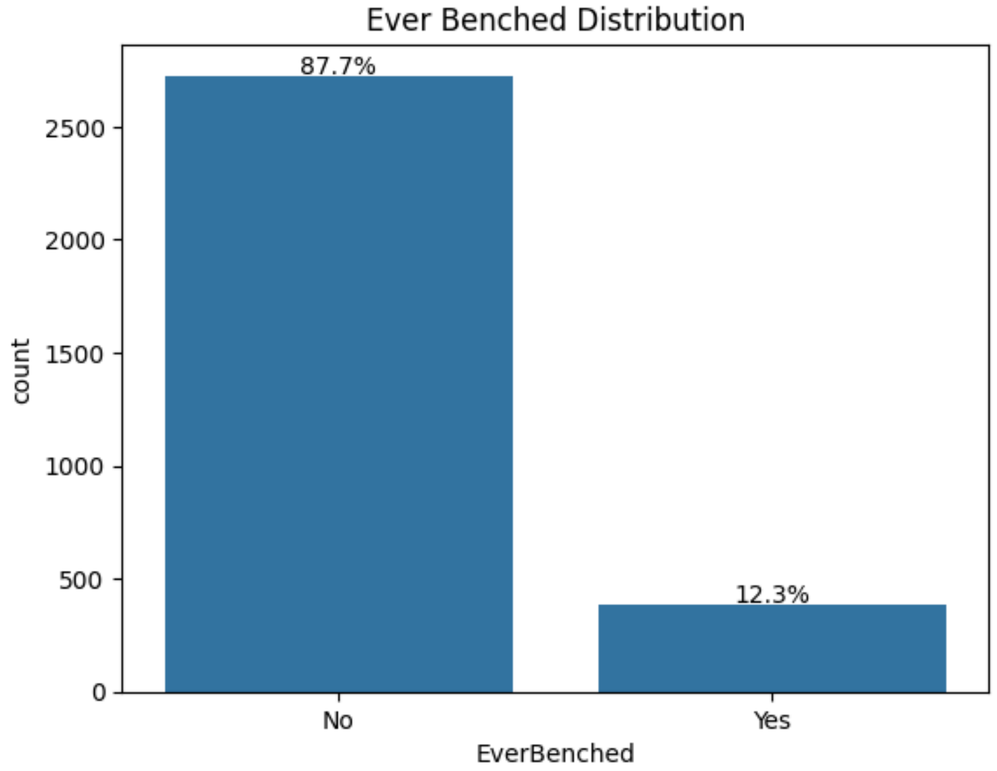
This is a histogram of Age

Instead say

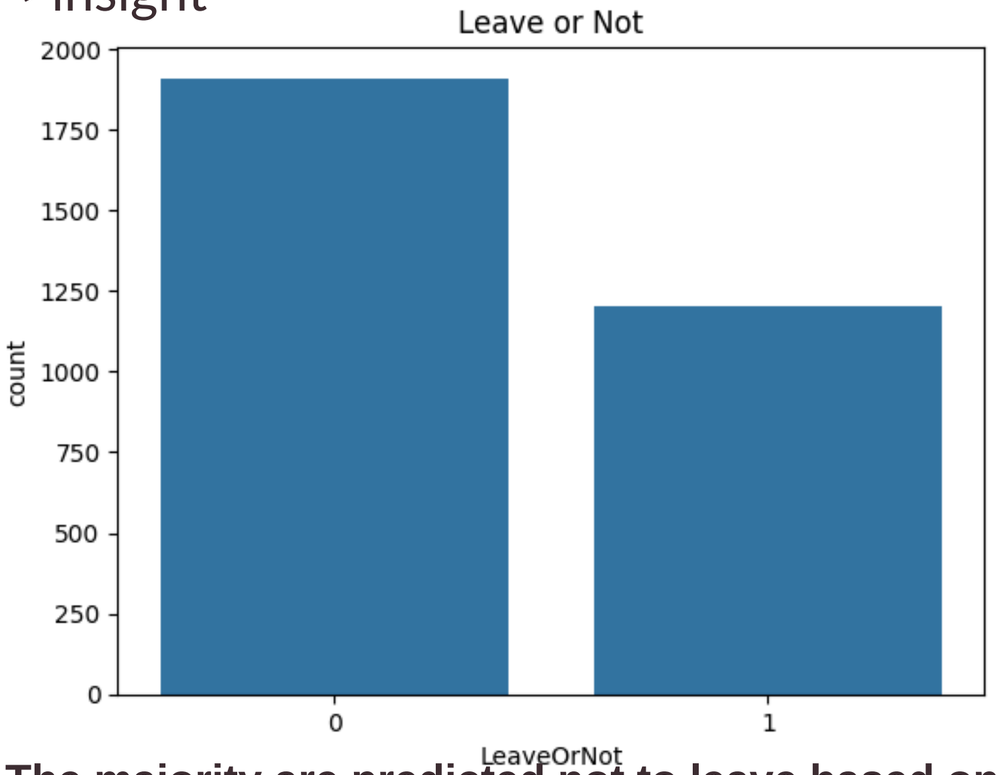
Most employees are in their late 20s-30s; very few older than 40 → the workforce is relatively young

Visualization Interpretations

Move from chart → insight



Around 12.3% have been benched → indicates periods of underutilization



The majority are predicted not to leave based on inference from the variables (since this is the target variable)

Netflix Challenge

1. Split into your groups
2. First 15 mins: Quick EDA (head, describe, missing values, distributions)
3. Broad question: “How has Netflix’s content evolved over time, by type, genre and geography?”
4. 5 mins: Brainstorm a sub-question that helps answer this question
5. Make a visualization that answers your question
6. Present your chart and 1-sentence story (inference)

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python **Interactively** at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> importmatplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

Axes

All plotting is done with respect to anAxes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them
Draw unconnected points, scaled or colored
Plot vertical rectangles (constant width)
Plot horiontal rectangles (constant height)
Draw a horizontal line across axes
Draw a vertical line across axes
Draw filled polygons
Fill between y-values and 0

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)
>>> axes[1,1].quiver(y,z)
>>> axes[0,1].streamplot(X,Y,U,V)
```

Add an arrow to the axes
Plot a 2D field of arrows
Plot a 2D field of arrows

Data Distributions

```
>>> ax1.hist(y)
>>> ax3.boxplot(y)
>>> ax3.violinplot(z)
```

Plot a histogram
Make a box and whisker plot
Make a violin plot

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

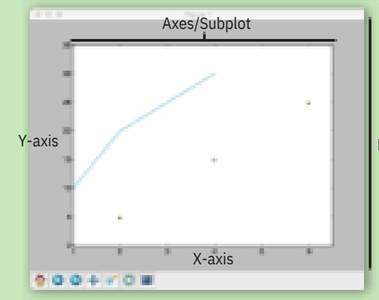
Colormapped or RGB arrays

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2]= ax.clabel(CS)
```

Pseudocolor plot of 2D array Pseudocolor plot of 2D array Plot contours Plot filled contours Label a contour plot

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,

            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8,     0),     xycoords='data',
                xytext=(10.5,                 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",

                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

Add padding to a plot
Set the aspect ratio of the plot to 1
Set limits for x-and y-axis
Set limits for x-axis

Legends

```
>>> ax.set(title='An Example Axes',
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')
```

Set a title and x-and y-axis labels

No overlapping plot elements

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',
                   direction='inout',
                   length=10)
```

Manually set x-ticks

Make y-ticks longer and go in and out

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,
hspace=0.3,
left=0.125,
right=0.9,
top=0.9,
bottom=0.1)
>>> fig.tight_layout()
```

Adjustthe spacing between subplots

Fitsubplot(s) in to the figure area

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Make the top axis line for a plot invisible
Move the bottom axis line outward

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

Clear an axis
Clear the entire figure
Close a window

DataCamp
Learn Python for Data Science Interactively

Python For Data Science *Cheat Sh*

Seaborn

Learn Data Science **Interactively** at www.DataCamp.com

Statistical Data Visualization With Seaborn

ThePython visualization library **Seaborn** isbased on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         Step 4
>>> plt.title("title")                                 Step 5
>>> plt.show(g)
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6))
```
Create a figure and one subplot

Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks",
                  {"xtick.major.size":8,
                   "ytick.major.size":8})
>>> sns.axes_style("whitegrid")
```

(Re)set the seaborn default
Set the matplotlib parameters
Set the matplotlib parameters

Return a dict of params or use with `with` to temporarily set the style

Context Functions

```
>>> sns.set_context("talk")
>>> sns.set_context("notebook",
                    font_scale=1.5,
                    rc={"lines.linewidth":2.5})
```

Set context to `"talk"`
Set context to `"notebook"`
Scale font elements and override param mapping

Color Palette

```
>>> sns.set_palette("husl",3)
>>> sns.color_palette("husl")
>>> flatui =["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui)
```

Define the color palette
Use with `with` to temporarily set palette
Set your own color palette

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist,"age")
```

Subplot grid for plotting conditional relationships

```
>>> sns.factorplot(x="pclass",
                   y="survived",
                   hue="sex",
                   data=titanic)
```

Draw a categorical plot onto a Facetgrid

```
>>> sns.lmplot(x="sepal_width",
               y="sepal_length",
               hue="species",
               data=iris)
```

Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
```

Subplot grid for plotting pairwise relationships

```
>>> sns.pairplot(iris) >>>
```

Plot pairwise bivariate distributions

```
i = sns.JointGrid(x="x",
                  y="y",
                  data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
```

Grid for bivariate plot with marginal univariate plots

```
>>> sns.jointplot("sepal_length",
                  "sepal_width",
                  data=iris,
                  kind='kde')
```

Plot bivariate distribution

Categorical Plots

Scatterplot

```
>>> sns.stripplot(x="species",
                  y="petal_length",
                  data=iris)
```

Scatterplot with one categorical variable

```
>>> sns.swarmplot(x="species",
                  y="petal_length",
                  data=iris)
```

Categorical scatterplot with non-overlapping points

Bar Chart

```
>>> sns.barplot(x="sex",
                y="survived",
                hue="class",
                data=titanic)
```

Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

```
>>> sns.countplot(x="deck",
                  data=titanic,
                  palette="Greens_d")
```

Show count of observations

Point Plot

```
>>> sns.pointplot(x="class",
                  y="survived",
                  hue="sex",
                  data=titanic,
                  palette={"male":"g",
                           "female":"m"},
                  markers=["^","o"],
                  linestyles=["-","--"])
```

Show point estimates and confidence intervals as rectangular bars

Boxplot

```
>>> sns.boxplot(x="alive",
                y="age",
                hue="adult_male",
                data=titanic)
```

Boxplot

```
>>> sns.boxplot(data=iris,orient="h")
```

Boxplot with wide-form data

Violinplot

```
>>> sns.violinplot(x="age",
                   y="sex",
                   hue="survived",
                   data=titanic)
```

Violin plot

Regression Plots

```
>>> sns.regplot(x="sepal_width",
                y="sepal_length",
                data=iris,
                ax=ax)
```

Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y,
                        kde=False,
                        color="b")
```

Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1)
```

Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived",
                      "Sex")
```

Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

```
>>> h.set(xlim=(0,5),
ylim=(0,5),
xticks=[0,2.5,5],
yticks=[0,2.5,5])
```

Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax,yticks=[0,5])
>>> plt.tight_layout()
```

Add plot title Adjust the label of the y-axis Adjust the label of the x-axis Adjust the limits of the y-axis Adjust the limits of the x-axis Adjust a plot property Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

```
>>>      plt.show()      >>>
plt.savefig("foo.png")  >>>
plt.savefig("foo.png",
            transparent=True)
```

Show the plot
Save the plot as a figure
Save transparent figure

Close & Clear

Also see Matplotlib

```
>>>    plt.cla()
>>>    plt.clf()
>>> plt.close()
```

Clear an axis
Clear an entire figure
Close a window

Materials are also on

Github

https://github.com/NUS-SDS-Workshops/AY-25-26-Public

Our next workshop:
Week 9: Machine Learning!

Also beginner friendly! If you did not request to receive email and want to for this workshop, tele pm @kaiironglee

Feedback Form

Your paragraph text

Thanks!