AIG150- Week 1

Introduction To Data Wrangling Using Python

Reading Text:

Chap 01,Ch 02:Pandas for everyone

Chap 01-03:Data Wrangling with Python

Agenda

- Data Wrangling
- Python & Data Science
- Introduction to different data analysis tools in Python
- DataFrame
- Data Series

What is Data?

- Definition: Data is raw facts and figures collected from different sources.
- Types of Data in Real Life:
  - Numbers: sales amount, age, temperature
  - Text: names, addresses, product descriptions
  - Dates/Time: appointment times, order dates
  - Categories: Male/Female, Yes/No, Paid/Unpaid
- Example: A shopping website collects data:
  - Customer Name: Alice
  - Age: 25
  - Purchase Amount: $50
  - Date: Sept 7, 2025

Common Ways to View Data

Tables (most common)Like an Excel spreadsheet: rows = records, columns = attributes.

Name	Age	Purchase
Alice	25	50
Bob	30	75
Cathy	NaN	20

Tab-Separated or Comma-Separated Values

- A .tsv file stands for Tab-Separated Values file.

- It’s a plain text file used to store tabular data (rows and columns).
- Each row = one record.
- Each column = separated by a tab (\t) instead of a comma.

- Difference from .csv
- CSV (Comma-Separated Values): columns separated by commas → Alice,20,A
- TSV (Tab-Separated Values): columns separated by tabs → Alice 20 A

- Both are used for data exchange between software (Excel, databases, Python, R, etc.).

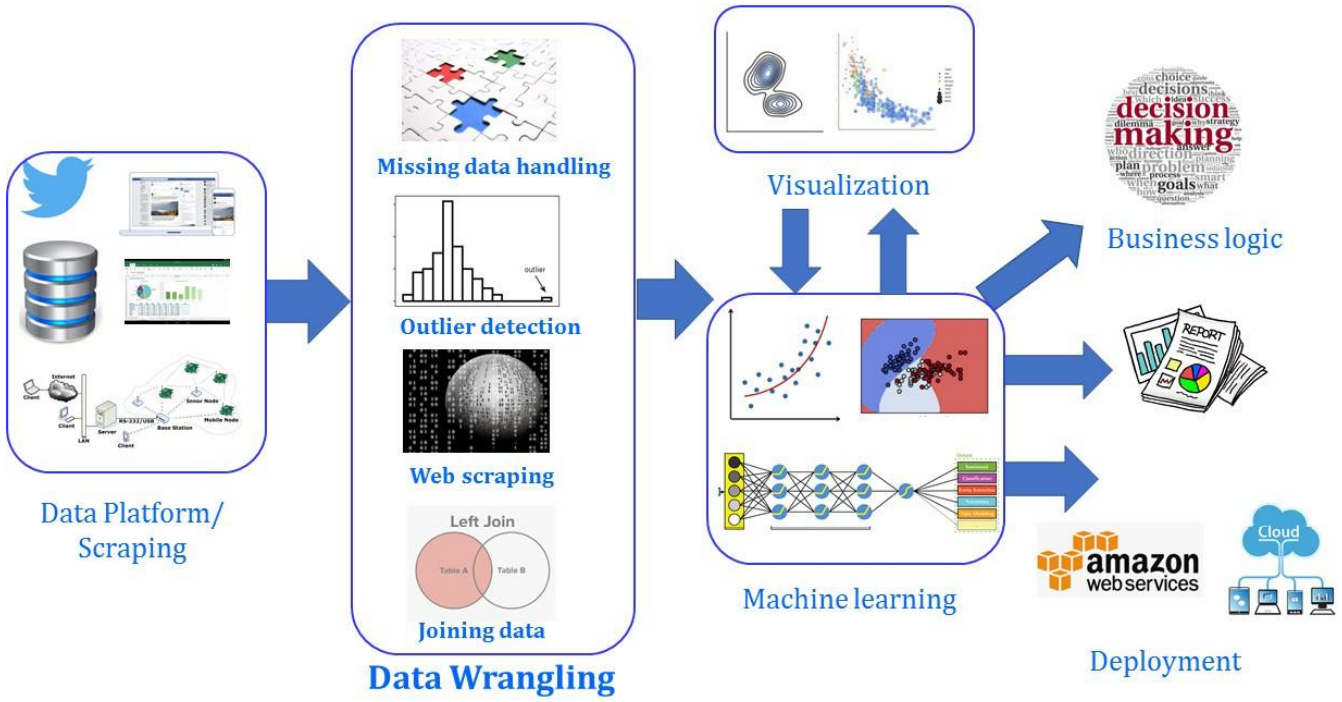
Data Wrangling

- The process that ensures that data is clean, accurate, formatted and ready to be use in data analysis
- Another word used for wrangling is ***"munging"***
- It is the first step of data analytical pipeline after selection of data sources
- Data is coming from different sources such as conventional databases, online feeds, smart devices, satellite imagery and many more
- Real-world data is messy → Wrangling makes it usable.

Data Wrangling --- The Process

- Data Collection → Gather raw data from multiple sources (websites, APIs, databases).
- Data Transformation → Clean and reformat (fix missing values, convert types, add new columns).
- Error Handling → Correct typos, wrong values, data types.
- Outlier Detection → Identify unusual values (too high/low).
- Data Visualization → Use charts/plots to detect issues and understand data.

Data Wrangling --- The Process



Ref: Chap 01, Data Wrangling With Python

1. Data Collection

- Definition: Gathering raw data from multiple sources.

- Data can come from:
  - Web scraping – extracting data from websites (e.g., collecting prices from Amazon).
  - APIs – pulling structured data (e.g., Twitter API for tweets).
  - Databases – fetching customer data from SQL/Oracle.
  - Files – CSV, Excel, JSON, TSV.
- Example:
- A company wants to analyze housing prices.
  - Data collected from: real estate websites (scraping), government housing reports (CSV files), and APIs (mortgage rates).

2. Data Transformation

- Definition: Converting raw data into a clean, usable format for modeling.

- Common Steps:
- Handling Missing Values: Replace with average/median → Cathy's missing age = 28.
  - Or drop the row if it's unusable.
- Data Filtration: Remove irrelevant data (e.g., "null email addresses" from customer list).

- Adding New Data (Feature Engineering):
  - From "Date of Birth," calculate "Age."
  - From "Order Date," add "Day of Week."
- Data Conversions:
  - Convert "$50.00" → 50 (numeric).
  - Convert "Yes/No" → 1/0 (binary).
- Example:
- A hospital dataset has:
  - Missing patient ages → fill with average age.
  - Admission date in text → convert to YYYY-MM-DD format.

3. Error Handling

- Definition: Identifying and fixing data errors so analysis is not misleading.
- Common Errors:
- Typos in text: “Tornto” → “Toronto.”
- Wrong data type: Phone number stored as number (1234567890) instead of text (so leading zeros are lost).
- Invalid values: Age = -10, Salary = “abc.”
- Example:
  - If a salary dataset has $5,000, abc, and $6000, the abc must be fixed or removed before analysis.

4. Outlier Detection

- An outlier is a value that is very different (too small or too large) compared to the rest of the data.

- IQR = Interquartile Range
  It measures the spread of the middle 50% of the data.
- Steps:
- Arrange data in order.
- Find Q1 (25th percentile) → the value at the first quarter.
- Find Q3 (75th percentile) → the value at the third quarter.
- Compute IQR = Q3 – Q1.

Example dataset:

```
[10, 12, 14, 15, 18, 19, 20, 21, 30]
```

Q1 = 14
Q3 = 20
IQR = Q3 – Q1 = 20 – 14 = 6

- Statistical Rule (1.5 × IQR): Values outside [Q1 - 1.5x*IQR, Q3 + 1.5x*IQR].

Outlier Detection Using Z-Score

1. What is a Z-Score?

   A Z-score tells us how many standard deviations a value is away from the mean (average).

   Formula:

$$Z = \frac{(x - \text{mean})}{\text{standard deviation}}$$

   If a Z-score is very high (positive) or very low (negative), it means the value is far from the average.

2. Rule of Thumb for Outliers

   If |Z| > 3 (more than 3 standard deviations away from the mean), the value is considered an outlier.

5. Data Visualization

- Definition: Presenting data with charts and graphs to understand patterns.
- Why Important?
  - Makes data easy to interpret.
  - Helps spot missing values, outliers, and trends.
- Common Tools (in Python):
  - Matplotlib & Seaborn – bar charts, histograms, scatter plots.
  - Pandas – quick plotting from DataFrames.
- Examples:
  - Bar chart: Customer purchases by age group.
  - Histogram: Distribution of test scores.
  - Scatter plot: Income vs. Spending score to find clusters.

Python & Data Science

- One of the top choices for data analysis
- Enrich in libraries such as *NumPy*, *SciPy*, *Matplotlib* and *pandas*
- Multiprocessing on large datasets --- reducing processing time
- Specialized IDEs
- IDE = Integrated Development Environment
- A software that helps you write, run, and debug Python code in one place.

Notebook softwares

1. Kaggle (Kaggle Notebooks / Kaggle Kernels)

- **What it is:** A free cloud-based environment by Kaggle (owned by Google) mainly for data science, machine learning, and competitions.
- **Setup:** No installation needed. Everything runs in the browser.
- **Compute:** Free GPU/TPU access (time-limited). Limited RAM and disk space.
- **Data:** Easy access to Kaggle Datasets and Competitions (integrated). Built-in APIs for importing datasets.
- **Collaboration:** Share notebooks publicly, comment, and fork (like GitHub for data notebooks).
- **Best for:** Beginners, Kaggle competitions, quick experiments with datasets, reproducible notebooks.
- https://www.kaggle.com/code

2. Google Colab (Colaboratory)

- **What it is:** A free Jupyter-like notebook hosted by Google, runs in the browser.
- **Setup:** No setup needed. Runs on Google's cloud servers.
- **Compute:** Free GPU/TPU access (with limits). Pro version available with more resources.
- **Data:** Can easily connect to Google Drive, GitHub, or upload files.
- **Collaboration:** Works like Google Docs—multiple users can edit in real time.
- **Best for:** Students and researchers who want free GPU for ML/DL, integration with Google Drive, and collaboration.
- https://colab.research.google.com/

3. Jupyter Notebook (Local or JupyterLab)

- **What it is:** An open-source interactive notebook environment you install locally (or run on a server).
- **Setup:** Requires Python/Anaconda setup. More effort than Kaggle/Colab.
- **Compute:** Uses your **local machine's CPU/GPU** (or remote server if configured). No free cloud compute.
- **Data:** Full freedom to use local or remote datasets. Can integrate with any environment.
- **Collaboration:** Not as seamless—need GitHub, JupyterHub, or VSCode for team sharing.
- **Best for:** Serious projects, custom environments, enterprise work, or when you need full control over packages, libraries, and compute.

- https://jupyter.org/try-jupyter/lab/index.html

Fundamentals In Python -- Review

List

Sets

Dictionaries

Control Flow Tools

Basic File Operation

Import System

Click on the links for each topic to review

Core Functions of a Data Scientist

- Data Capture
- ETL – Extract, Transform and Load
- Python comes handy in extracting data from multiple sources --- CSV Files, DBMS, …
- Data from multiple sources can then be transformed into a common format and loaded to analyze using Python
  - Analysis
  - Presentation

Defining Variables with Different Types

```
# Integers
age = 25            # int

# Floating point
pi = 3.14159        # float

# String
name = "Alice"      # str

# Boolean
is_student = True   # bool
```

```
# List
scores = [90, 85, 88] # list

# Tuple
coordinates = (10, 20) # tuple

# Dictionary
student = {"name": "Alice", "age": 25} # dict

# Set
unique_nums = {1, 2, 3, 3}  # set → {1, 2, 3}
```

Data Types in Python

A data type tells Python what kind of value is stored.
Different operations work on different types.
Example: You can add numbers, but you can't add a number to text without converting it.

Data Type	Example	Description
int	`10, -5`	Whole numbers (no decimals).
float	`3.14, -2.7`	Numbers with decimals.
str	`"hello"`	Strings = text.
bool	`True, False`	Boolean values (logic).
list	`[1, 2, 3]`	Ordered, changeable collection.
tuple	`(1, 2, 3)`	Ordered, unchangeable collection.
dict	`{"a": 1, "b": 2}`	Key–value pairs.
set	`{1, 2, 3}`	Unordered, no duplicates.

Data Series

- A Series is a one-dimensional labeled array.
- It can hold any type of data:
- Integers → [1, 2, 3]
- Strings → ["apple", "banana", "cherry"]
- Floats → [3.14, 2.71, 1.41]
- Even Python objects → [{"a": 1}, "hello", 99]
- Think of it like a Python list but with labels (index) attached.

```
import pandas as pd

data = [10, 20, 30]
series = pd.Series(data, index=["a", "b", "c"])
print(series)
```

```
a    10
b    20
c    30
dtype: int64
```

Introduction to NumPy, Pandas & Matplotlib

- Numpy offers comprehensive mathematical functions, random number generators, linear algebra routines and many more
- NumPy arrays are optimized data structures for numerical analysis
- Fast, powerful and reliable open-source data analysis tool build on top of basic Python programming language. https://pandas.pydata.org/
- Both NumPy and Pandas have numerous built-in statistical and visualization methods available for data analysis.
- Matplotlib is the most powerful and versatile visualization library in Python.

Introduction to NumPy

```
# Create data with NumPy
# Use linspace to create 100 numbers between 0 and 10.
# Compute the sine of those numbers.
# Create evenly spaced numbers between 0 and 10

import numpy as np
x = np.linspace(0, 10, 100)
print("Evenly spaced numbers \n", x)
# Compute sine values for each
xy = np.sin(x)print("y = np.sin(x) \n",y)
```

```
Evenly spaced numbers
 [ 0.          0.1010101   0.2020202   0.3030303   0.4040404   0.50505051
  0.60606061  0.70707071  0.80808081  0.90909091  1.01010101  1.11111111
  1.21212121  1.31313131  1.41414141  1.51515152  1.61616162  1.71717172
  1.81818182  1.91919192  2.02020202  2.12121212  2.22222222  2.32323232
  2.42424242  2.52525253  2.62626263  2.72727273  2.82828283  2.92929293
  3.03030303  3.13131313  3.23232323  3.33333333  3.43434343  3.53535354
  3.63636364  3.73737374  3.83838384  3.93939394  4.04040404  4.14141414
  4.24242424  4.34343434  4.44444444  4.54545455  4.64646465  4.74747475
  4.84848485  4.94949495  5.05050505  5.15151515  5.25252525  5.35353535
  5.45454545  5.55555556  5.65656566  5.75757576  5.85858586  5.95959596
  6.06060606  6.16161616  6.26262626  6.36363636  6.46464646  6.56565657
  6.66666667  6.76767677  6.86868687  6.96969697  7.07070707  7.17171717
  7.27272727  7.37373737  7.47474747  7.57575758  7.67676768  7.77777778
  7.87878788  7.97979798  8.08080808  8.18181818  8.28282828  8.38383838
  8.48484848  8.58585859  8.68686869  8.78787879  8.88888889  8.98989899
  9.09090909  9.19191919  9.29292929  9.39393939  9.49494949  9.5959596
  9.6969697   9.7979798   9.8989899  10.        ]
y = np.sin(x)
 [ 0.          0.10083842  0.20064886  0.2984138   0.39313661  0.48385164
  0.56963411  0.64960951  0.72296256  0.78894546  0.84688556  0.8961922
  0.93636273  0.96698762  0.98775469  0.99845223  0.99897117  0.98930624
  0.96955595  0.93992165  0.90070545  0.85230712  0.79522006  0.73002623
  0.65739025  0.57805259  0.49282204  0.40256749  0.30820902  0.21070855
  0.11106004  0.01027934 -0.09060615 -0.19056796 -0.28858706 -0.38366419
 -0.47483011 -0.56115544 -0.64176014 -0.7158225  -0.7825875  -0.84137452
 -0.89158426 -0.93270486 -0.96431712 -0.98609877 -0.99782778 -0.99938456
 -0.99075324 -0.97202182 -0.94338126 -0.90512352 -0.85763861 -0.80141062
 -0.73701276 -0.66510151 -0.58640998 -0.50174037 -0.41195583 -0.31797166
 -0.22074597 -0.12126992 -0.0205576   0.0803643   0.18046693  0.27872982
  0.37415123  0.46575841  0.55261747  0.63384295  0.7086068   0.77614685
  0.83577457  0.8868821   0.92894843  0.96154471  0.98433866  0.99709789
  0.99969234  0.99209556  0.97438499  0.94674118  0.90944594  0.86287948
  0.8075165   0.74392141  0.6727425   0.59470541  0.51060568  0.42130064
  0.32770071  0.23076008  0.13146699  0.03083368 -0.07011396 -0.17034683
 -0.26884313 -0.36459873 -0.45663749 -0.54402111]
```

Introduction to NumPy & Matplotlib

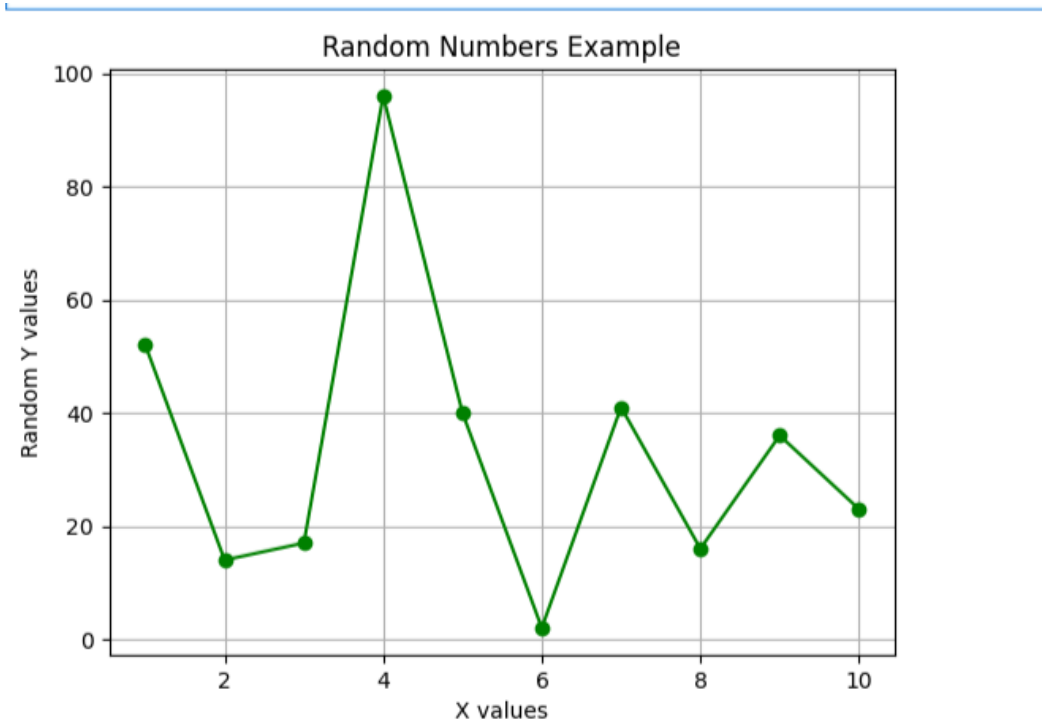
```
# Plot with Matplotlib
import numpy as np
import matplotlib.py

# Generate 10 random x values (1 to 10)
x = np.arange(1, 11)

# Generate 10 random y values (between 0 and 100)
y = np.random.randint(0, 100, size=10)

# Plot the random numbers
plot as plt

plt.plot(x, y, marker="o", linestyle="-", color="green")
plt.title("Random Numbers Example")
plt.xlabel("X values")
plt.ylabel("Random Y values")
plt.grid(True)
plt.show()
```



DataFrame

- The primary pandas data structure designed to work with relational or labeled data
- Two-dimensional, size-mutable, potentially heterogeneous tabular data structure also contains labeled axes (rows and columns)
- Performs arithmetic operations align on both row and column labels
- Can be thought of as a dictionary like container for Series objects
- For full details: pandas.DataFrame

DataFrame

- The primary pandas data structure designed to work with relational or labeled data
- Two-dimensional, size-mutable, potentially heterogeneous tabular data structure also contains labeled axes (rows and columns)
- Performs arithmetic operations align on both row and column labels
- Can be thought of as a dictionary like container for Series objects
- For full details: pandas.DataFrame

Panda Dataframe

```
# Print the version of Pandas used
import pandas as pd

print(pd.__version__)

2.2.3

# Upload TSV file and Read TSV file d

import pandas as pd
df = pd.read_csv('filename.tsv', sep='\t')

 # Upload CSV file and Read CSV file
 import pandas as pd
df = pd.read_csv("filename.csv")
```

```
# Save DataFrame as CSV file
df.to_csv("students.csv", index=False)

# Read back the CSV
csv_df = pd.read_csv("students.csv")
print(csv_df)

# Save DataFrame as TSV file
df.to_csv("students.tsv", sep="\t", index=False)

# Read back the TSV
tsv_df = pd.read_csv("students.tsv", sep="\t")
print(tsv_df)
```

Panda Dataframe contd..

```
import pandas as pd

data = {
   "Name": ["Alice", "Bob", "Cathy", "Dan"],
   "Age": [25, 30, 22, 35],
   "Purchase": [100, 200, 150, 300]
}
df = pd.DataFrame(data)
```

1. `.shape` Returns a tuple (rows, columns).

•print(df.shape)

•(4, 3) # 4 rows, 3 columns

2. head()

Shows the **first 5 rows** by default.

•print(df.head())

•Name Age Purchase

•0 Alice 25 100

•1 Bob 30 200

•2 Cathy 22 150

•3 Dan 35 300

3. .tail()

Shows the **last 5 rows** by default.

•print(df.tail())

Panda Dataframe contd..

```
import pandas as pd

data = {
   "Name": ["Alice", "Bob", "Cathy", "Dan"],
   "Age": [25, 30, 22, 35],
   "Purchase": [100, 200, 150, 300]
}
df = pd.DataFrame(data)
```

1. **`.shape`** Returns a tuple (rows, columns).

•print(df.shape)

•(4, 3) # 4 rows, 3 columns

2. head()

Shows the **first 5 rows** by default.

•print(df.head())

•Name Age Purchase

•0 Alice 25 100

•1 Bob 30 200

•2 Cathy 22 150

•3 Dan 35 300

3. .tail()

Shows the **last 5 rows** by default.

•print(df.tail())

Panda Dataframe contd..

```
import pandas as pd

data = {
   "Name": ["Alice", "Bob", "Cathy", "Dan"],
   "Age": [25, 30, 22, 35],
   "Purchase": [100, 200, 150, 300]
}
df = pd.DataFrame(data)
```

4. .columns

Lists the **column names**.

```
print(df.columns)
```

```
Index(['Name', 'Age', 'Purchase'], dtype='object')
```

5. .describe()

Gives **summary statistics** for numeric columns.

```
print(df.describe())
```

```
        Age    Purchase
count   4.000000    4.000000
mean   28.000000  187.500000
std     5.477226   86.602540
min    22.000000  100.000000
25%    24.250000  137.500000
50%    27.500000  175.000000
75%    31.250000  225.000000
max    35.000000  300.000000
```

Panda Dataframe contd..

6. .keys()
Returns the column labels of the DataFrame (like dictionary keys).

```
import pandas as pd
df = pd.DataFrame({"A": [1,2], "B":[3,4]})
print(df.keys())   # Output: Index(['A', 'B'], dtype='object’)
```

7. .DESCR
Not a standard Pandas attribute — this is usually found in **scikit-learn datasets** (e.g., load_iris().DESCR), which shows the dataset description.
In Pandas, you’d use .info() or .describe() instead.

Panda Dataframe contd..

8. .dtypes
Shows the data type of each column.
print(df.dtypes)

```
Name        object
Age          int64
Purchase     int64
dtype: object
```

9. .loc[]
Label-based indexing: select rows/columns using labels.

```
print(df.loc[0, "Age"])      # value at row 0, column "Age"
print(df.loc[:, “Purchase"])      # all rows, column Purchase
```

```
25

0    100
1    200
2    150
3    300
Name: Purchase, dtype: int64
```

10. .iloc[]
Position-based indexing (integer indexes).

```
print(df.iloc[0, 0])       # first row, first column     Alice
print(df.iloc[:, 1])       # all rows, second column All rows, column index 1 (Age):
```

Panda Dataframe contd..

11. Subset DataFrame
Select multiple columns/rows.

```
print(df[["A"]])          # only column A
print(df.loc[0:1, ["A"]])   # rows 0–1, column A
```

```
   Age
0   25
1   30
2   22
3   35
```

```
   Age
0   25
1   30
```

12. .info()
Summary of DataFrame: index, column names, non-null counts, data types, and memory usage.

```
df.info()
```

Panda Dataframe contd..

13. .sort_values()

Sort by a column.

```
df_sorted = df.sort_values(by="B", ascending=False)
```

14. .unique()

Returns unique values of a Series (not entire DataFrame).

```
print(df["A"].unique())   # unique values in column A
```

Panda Dataframe contd..

```
import pandas as pd

data = {
   "Name": ["Alice", "Bob", "Cathy", "Dan"],
   "Age": [25, 30, 22, 35],
   "Purchase": [100, 200, 150, 300]
}
df = pd.DataFrame(data)
```

15. .sum()
Adds up values **column-wise (default)** or **row-wise**.

```
print(df["Purchase"].sum())

750# 100 + 200 + 150 + 300
```

df.sum(axis=0) → sums column values.
df.sum(axis=1) → sums row values.

16. .min(), .max(), .mean()
Summary statistics per column.

```
print(df.min())    # min of each column
print(df.max())    # max of each column
print(df.mean())   # mean of each column
```

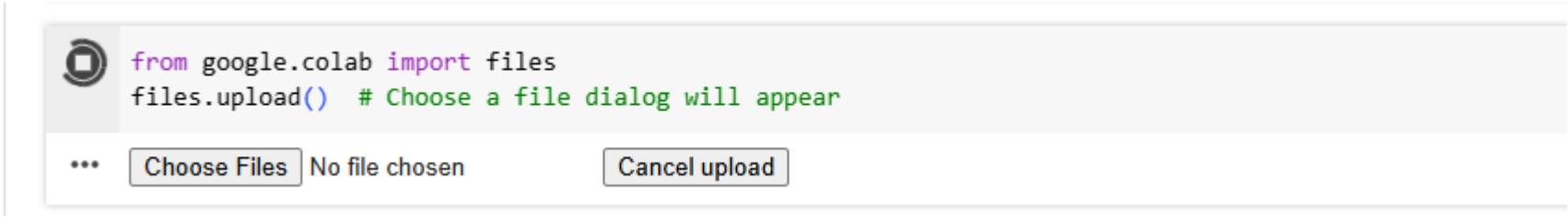
```
Age         22
Purchase   100
dtype: int64
```

```
Age         35
Purchase   300
dtype: int64
```

```
Age         28.0
Purchase   187.5
dtype: float64
```

Google Colab features

upload a file from your computer into Colab
from google.colab
import filesfiles.upload()
# Choose a file dialog will appear



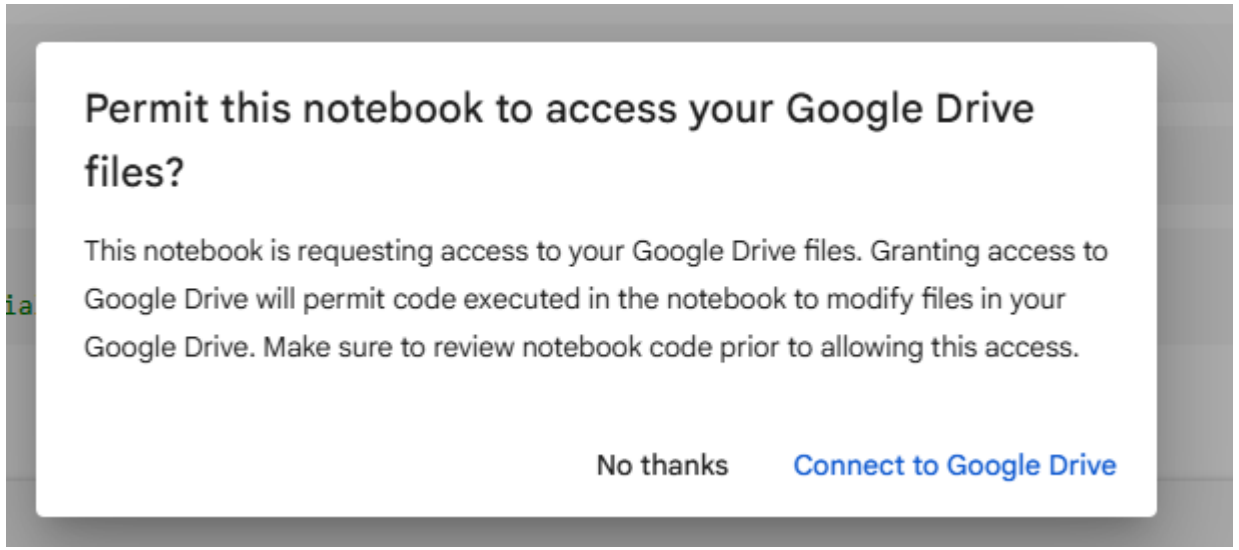
- **Temporary files stored in Colab**
- In the /content/ directory (deleted when the session ends).

# **Python version in Colab**
import sys
print(sys.version)

Google Colab features contd..

Mount Google Drive in Colab to access your files

```
from google.colab
import drivedrive.mount('/content/drive’)
 # Follow the auth link shown
```



Google Colab features contd..

1. CPU (Central Processing Unit)

- The default processor in your computer.Good for general tasks (running programs, basic Python).Not always the fastest for large-scale math or deep learning.

2. GPU (Graphics Processing Unit)

- Originally designed for rendering graphics (games, videos).
- Can process many calculations in parallel (thousands of small tasks at once).
- Great for: Machine Learning (ML),Deep Learning (DL),Image/Video processing
- Example: Training a neural network on images is **10–100x faster** on GPU than CPU.

3. TPU (Tensor Processing Unit)

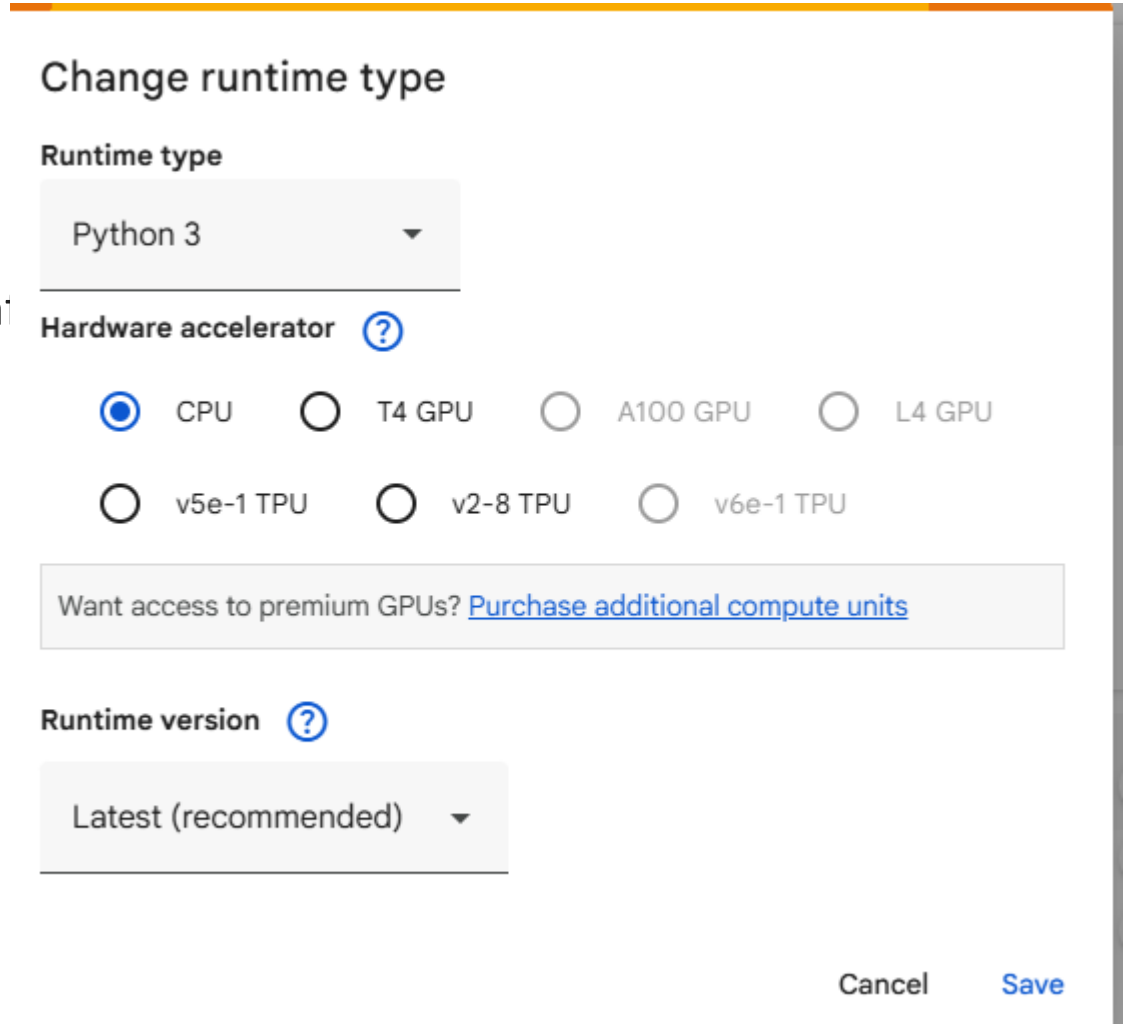
- A specialized chip built by Google, designed only for machine learning (especially TensorFlow).
- Even faster than GPU for certain ML tasks.Best for:
- Neural networks
- Tensor :Flow-based models. Very large datasets and training jobs
- Example: Training a large language model or image classifier can be faster and cheaper on TPU.

Google Colab features contd..

Python code to check if Colab is using a GPU.

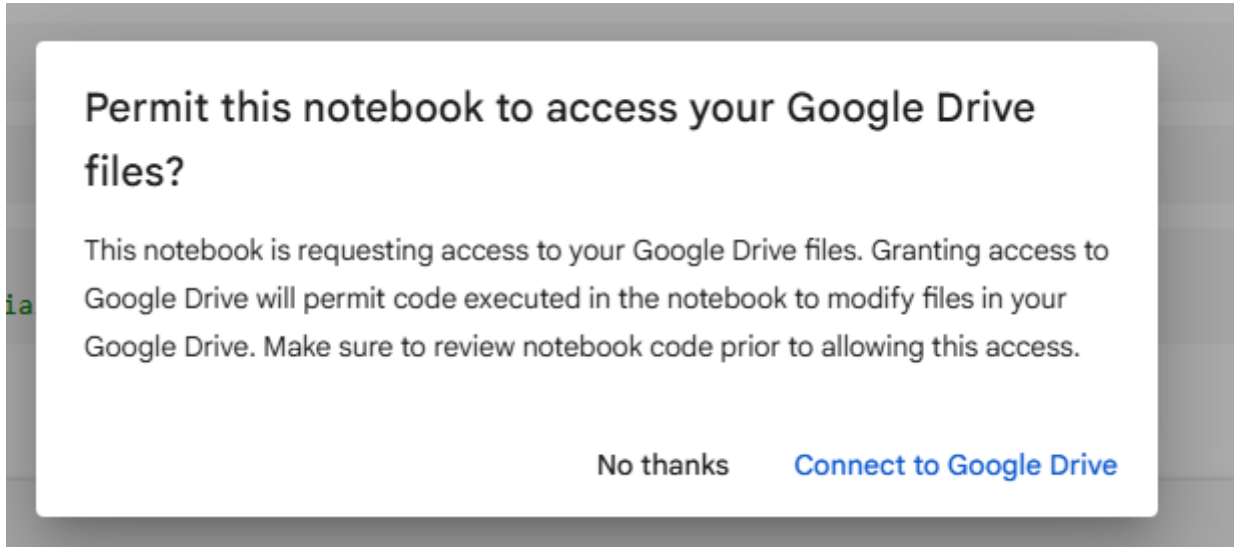
```
import tensorflow as tf
gpu_info = {} # Initialize the dictionary
gpu_info['tensorflow'] = [d.name for d in tf.con
print('GPU info (TF/PyTorch):', gpu_info)
```

Runtime → Change runtime type →
Hardware accelerator → GPU → Save



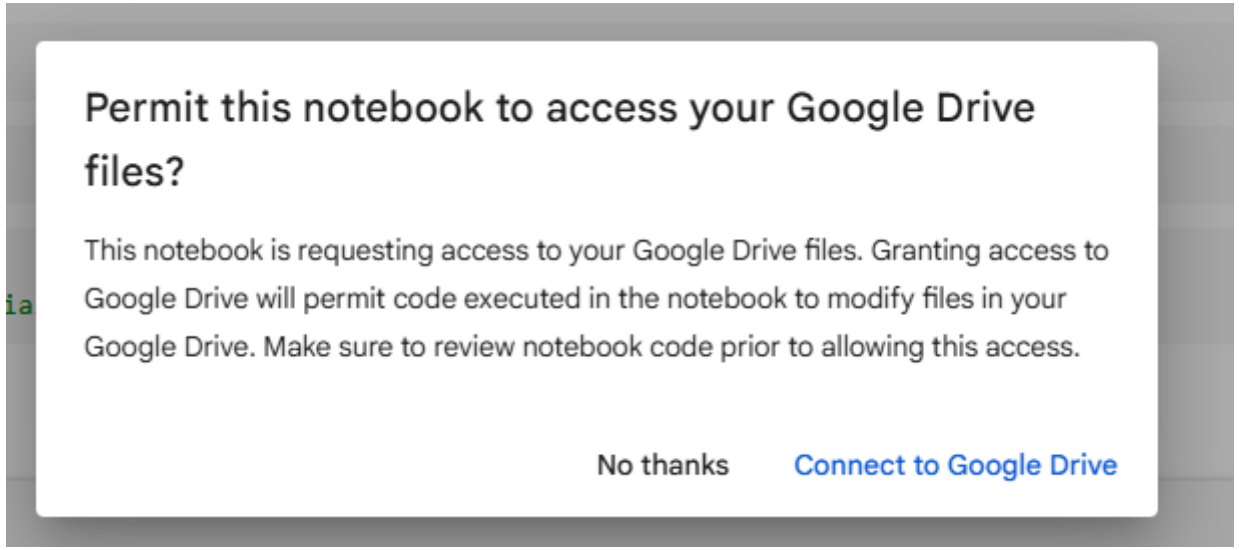
Google Colab features contd..

```
from google.colab
import drivedrive.mount('/content/drive’)
 # Follow the auth link shown
```



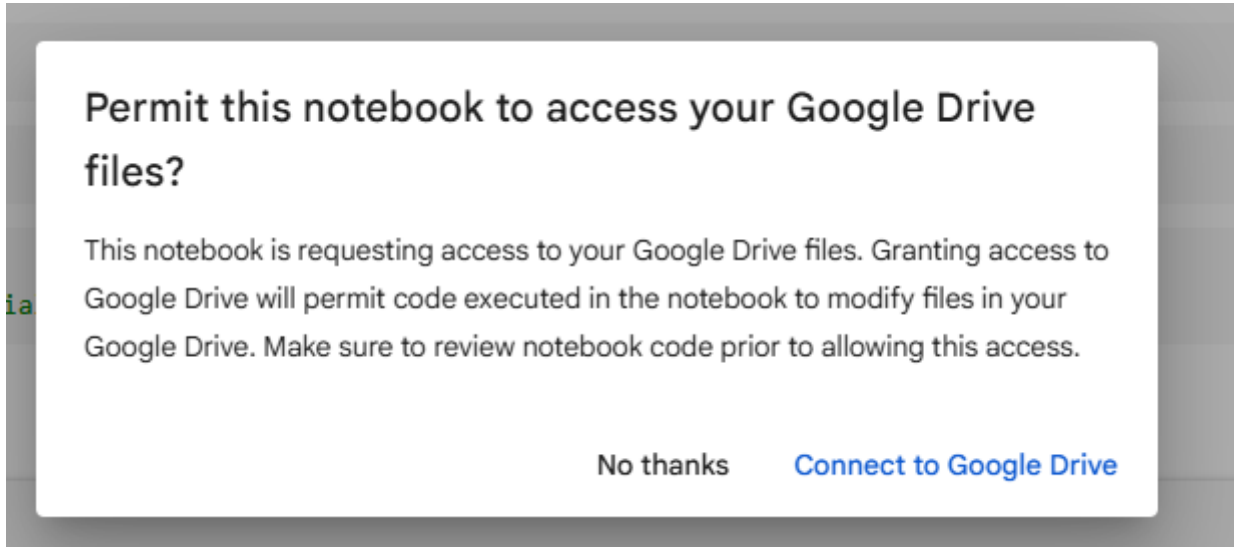
Google Colab features contd..

```
from google.colab
import drivedrive.mount('/content/drive’)
 # Follow the auth link shown
```



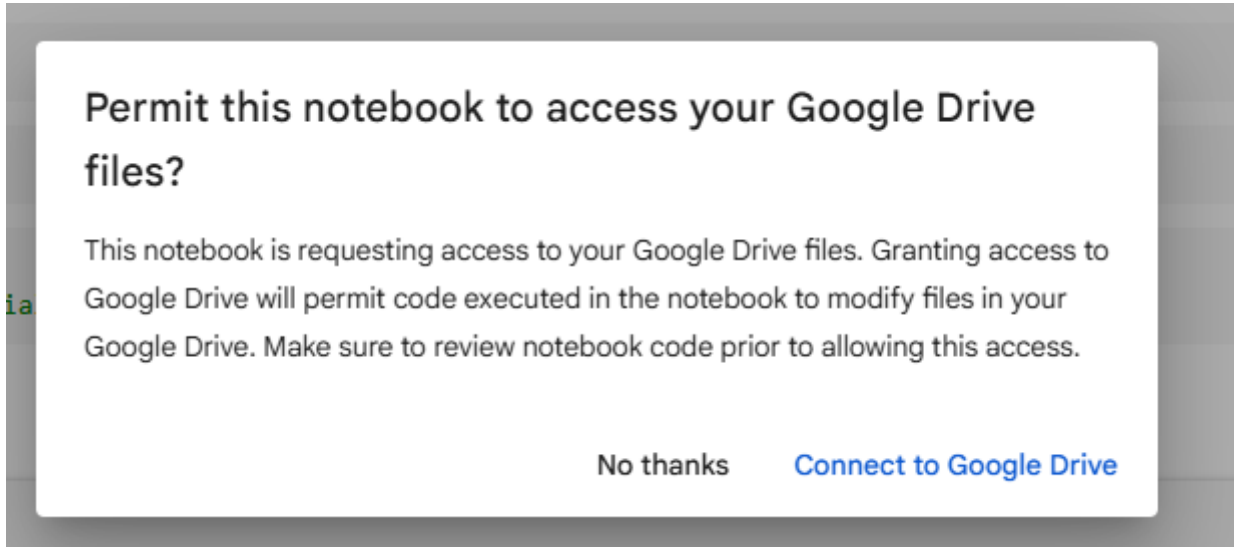
Google Colab features contd..

```
from google.colab
import drivedrive.mount('/content/drive’)
 # Follow the auth link shown
```



Google Colab features contd..

```
from google.colab
import drivedrive.mount('/content/drive’)
 # Follow the auth link shown
```



Summary of Python & Data Wrangling Basics

- **Data Wrangling**
- Cleaning, transforming, and organizing raw data into a usable format.
- **Steps:**
  - Collect data (CSV, TSV, databases).
  - Clean (fix missing values, duplicates, errors).
  - Transform (convert formats, add new features).
  - Handle outliers (IQR, Z-score methods).
  - Validate and visualize (charts, plots).
- **Pandas Series & DataFrame**
- **Series:** One-dimensional labeled array (like a Python list with labels).
- **DataFrame:** Two-dimensional table (like Excel). Each column = a Series.
- **Google Colab features**