1. Load datasets and preview

Load the CSV files and preview the first rows to get a sense of the data. Ask students what t

```python
import pandas as pd
eda_path = "eda_dataset.csv"
plot_path = "plot_dataset.csv"

df = pd.read_csv(eda_path)
df_plot = pd.read_csv(plot_path)

print("EDA shape:", df.shape)
print("\nEDA preview:")
display(df.head(8))

print("\nPlot dataset preview:")
display(df_plot.head())
```

[13] ✓ 0.0s

```
EDA shape: (63, 9)

EDA preview:
```

	ID	Name	Age	City	Salary	Duration	Calories	Maxpulse	Date
0	1	Student_1	26.0		40679.0	50	236	118	2025-11-01
1	2	Student_2	27.0		57207.0	46	193	166	2025-11-02
2	3	Student_3	27.0		53725.0	58	850	104	2025-11-03
3	4	Student_4	26.0	Sulaimani	37710.0	35	187	73	11/05/2025
4	5	Student_5	25.0	Erbil	61906.0	77	260	181	2025-11-05
5	6	Student_6	27.0		300000.0	113	247	130	2025-11-06
6	7	Student_7	25.0	Erbil	59430.0	45	684	98	2025-11-07
7	8	Student_8	NaN	Baghdad	48561.0	40	555	196	2025-11-08

1.1 Inspecting the data (methods to teach)

- `df.info()` — shows dtypes and non-null counts.
- `df.describe()` — summary statistics for numeric columns.
- `df.isnull().sum()` — counts of missing values.
- `df.columns`, `df.shape`, `df.dtypes` — quick checks.

```python
# Basic inspection
print("Info:")
display(df.info())

print("\nDescribe (numeric):")
display(df.describe())

print("\nMissing counts per column:")
display(df.isnull().sum())

print("\nColumns and dtypes:")
print(df.dtypes)
```

[14] ✓ 0.0s

```
Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63 entries, 0 to 62
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   ID        63 non-null     int64  
 1   Name      63 non-null     object 
 2   Age       55 non-null     float64
 3   City      63 non-null     object 
 4   Salary    62 non-null     float64
 5   Duration  63 non-null     int64  
 6   Calories  63 non-null     int64  
 7   Maxpulse  63 non-null     int64  
 8   Date      62 non-null     object 
dtypes: float64(2), int64(4), object(3)
memory usage: 4.6+ KB
```

```
None
```

2. Handling missing values — teach these methods

- **Detect** missing values (`isnull`).
- **Drop** rows/columns with `dropna()` when appropriate.
- **Fill** missing values with constants or statistics (`fillna(mean/median/mode)`).
- **Replace** blank-only strings (like ' ') with `pd.NA` before filling.

Classroom activity: Ask students to propose which columns should use `mean` vs `median` vs `mode` and why.

```python
# Detect missing values
display(df.isnull().sum())
```

[15] ✓ 0.0s

```
ID          0
Name        0
Age         8
City        0
Salary      1
Duration    0
Calories    0
Maxpulse    0
Date        1
dtype: int64
```

```python
# Example strategies:
df_missing_demo = df.copy()

# Fill Age with mean (numeric)
df_missing_demo['Age'] = pd.to_numeric(df_missing_demo['Age'], errors='coerce')
df_missing_demo['Age'].fillna(df_missing_demo['Age'].mean(), inplace=True)

# Fill Salary with median
df_missing_demo['Salary'] = pd.to_numeric(df_missing_demo['Salary'], errors='coerce')
df_missing_demo['Salary'].fillna(df_missing_demo['Salary'].median(), inplace=True)

# Fill City blanks with 'Unknown'
df_missing_demo['City'].fillna('Unknown', inplace=True)

display(df_missing_demo.head(8))
display(df_missing_demo.isnull().sum())
```

[16] ✓ 0.0s

3. Duplicates — detect and handle

- `df.duplicated()` to find duplicates; `df.drop_duplicates()` to remove.
- Teaching tip: show duplicated(keep=False) to see all copies, then decide which to keep.

```python
# Duplicates
print("Number of duplicate rows:", df.duplicated().sum())
display(df[df.duplicated(keep=False)].sort_values(by=list(df.columns)).head(10))
# Remove duplicates in practice
df_nodup = df.drop_duplicates().reset_index(drop=True)
print("\nAfter drop_duplicates shape:", df_nodup.shape)
```

[17] ✓ 0.0s

	ID	Name	Age	City	Salary	Duration	Calories	Maxpulse	Date
0	1	Student_1	26.0		40679.0	50	236	118	2025-11-01
60	1	Student_1	26.0		40679.0	50	236	118	2025-11-01
1	2	Student_2	27.0		57207.0	46	193	166	2025-11-02
61	2	Student_2	27.0		57207.0	46	193	166	2025-11-02
2	3	Student_3	27.0		53725.0	58	850	104	2025-11-03
62	3	Student_3	27.0		53725.0	58	850	104	2025-11-03

```
After drop_duplicates shape: (60, 9)
Number of duplicate rows: 3
```

4. Wrong formats (dates and numeric columns)

- Use `pd.to_datetime(..., errors='coerce')` to parse dates and convert invalid strings to `NaT`.
- Use `pd.to_numeric(..., errors='coerce')` for numbers; then handle NaNs.

```
# Fix Date column
df_dates = df.copy()
df_dates['Date_fixed'] = pd.to_datetime(df_dates['Date'], errors='coerce')
display(df_dates[['Date','Date_fixed']].head(12))
print("\nInvalid date count:", df_dates['Date_fixed'].isna().sum())
```

[18] ✓ 0.0s

	Date	Date_fixed
0	2025-11-01	2025-11-01
1	2025-11-02	2025-11-02
2	2025-11-03	2025-11-03
3	11/05/2025	NaT
4	2025-11-05	2025-11-05
5	2025-11-06	2025-11-06
6	2025-11-07	2025-11-07
7	2025-11-08	2025-11-08
8	2025.11.10	NaT
9	2025-11-10	2025-11-10
10	2025-11-11	2025-11-11
11	2025-11-12	2025-11-12

```
Invalid date count: 3
```

5. Outlier detection — methods to teach

- **IQR (Interquartile Range)**: robust to non-normal data.
- **Z-score**: useful for near-normal distributions.
- **Percentile thresholding** (e.g., top 1% or values > 99th percentile).

Classroom exercise: compute IQR outliers for `Salary` and discuss whether to remove or cap them.

```
# IQR method for Salary
df_num = df.copy()
df_num['Salary'] = pd.to_numeric(df_num['Salary'], errors='coerce')

Q1 = df_num['Salary'].quantile(0.25)
Q3 = df_num['Salary'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR

outliers_iqr = df_num[(df_num['Salary'] < lower) | (df_num['Salary'] > upper)]
print("IQR outliers count:", outliers_iqr.shape[0])
display(outliers_iqr[['ID','Name','Salary']].head(10))
```

[19] ✓ 0.0s

```
IQR outliers count: 2
```

	ID	Name	Salary
5	6	Student_6	300000.0
17	18	Student_18	150000.0

```
# Z-score method (manual fallback)
s = df_num['Salary'].dropna()
z = (s - s.mean()) / s.std()
print("Z-score outliers (>3):", (abs(z) > 3).sum())
```

[20] ✓ 0.0s

```
Z-score outliers (>3): 1
```

6. Correlation analysis

- Use `df.corr()` on numeric columns.
- Teaching tip: explain correlation coefficients: -1 (perfect negative) to +1 (perfect positive), and 0 = no linear correlation.

```python
# Correlation matrix for numeric fields
df_corr = df.select_dtypes(include=['number'])
display(df_corr.head())
display(df_corr.corr())
```

[21] ✓ 0.0s

	ID	Age	Salary	Duration	Calories	Maxpulse
0	1	26.0	40679.0	50	236	118
1	2	27.0	57207.0	46	193	166
2	3	27.0	53725.0	58	850	104
3	4	26.0	37710.0	35	187	73
4	5	25.0	61906.0	77	260	181

	ID	Age	Salary	Duration	Calories	Maxpulse
ID	1.000000	-0.297173	-0.190850	0.238520	0.125927	0.136887
Age	-0.297173	1.000000	0.202093	-0.166854	0.058233	-0.224686
Salary	-0.190850	0.202093	1.000000	0.142486	-0.046687	0.018035
Duration	0.238520	-0.166854	0.142486	1.000000	-0.128274	0.077524
Calories	0.125927	0.058233	-0.046687	-0.128274	1.000000	-0.240477
Maxpulse	0.136887	-0.224686	0.018035	0.077524	-0.240477	1.000000

7. Feature engineering & transformations

- **Scaling (min-max)** and **binning** as examples.
- Ask students: when do we scale? when do we bin?

```python
# Min-max scaling and age groups
df_fe = df.copy()
df_fe['Salary'] = pd.to_numeric(df_fe['Salary'], errors='coerce')
min_s = df_fe['Salary'].min()
max_s = df_fe['Salary'].max()
df_fe['Salary_minmax'] = (df_fe['Salary'] - min_s) / (max_s - min_s)

df_fe['Age'] = pd.to_numeric(df_fe['Age'], errors='coerce')
df_fe['AgeGroup'] = pd.cut(df_fe['Age'], bins=[0,24,29,35,100], labels=['<=24','25-29','30-35','36+'])

display(df_fe[['ID','Age','AgeGroup','Salary','Salary_minmax']].head(10))
```

0.0s

	ID	Age	AgeGroup	Salary	Salary_minmax
0	1	26.0	25-29	40679.0	0.011320
1	2	27.0	25-29	57207.0	0.074334
2	3	27.0	25-29	53725.0	0.061058
3	4	26.0	25-29	37710.0	0.000000
4	5	25.0	25-29	61906.0	0.092249
5	6	27.0	25-29	300000.0	1.000000
6	7	25.0	25-29	59430.0	0.082809
7	8	NaN	NaN	48561.0	0.041370
8	9	NaN	NaN	41434.0	0.014198
9	10	25.0	25-29	58436.0	0.079019

Save cleaned dataset (example)

This shows how to save a cleaned CSV for later use in modeling or visualization.

```
cleaned_path = "eda_dataset_cleaned.csv"
# We'll create a simple cleaned version based on earlier steps
df_clean = df.copy()
df_clean['City'] = df_clean['City'].replace(r'^\s*$', pd.NA, regex=True)
df_clean['Age'] = pd.to_numeric(df_clean['Age'], errors='coerce')
df_clean['Age'].fillna(df_clean['Age'].mean(), inplace=True)
df_clean['Salary'] = pd.to_numeric(df_clean['Salary'], errors='coerce')
df_clean['Salary'].fillna(df_clean['Salary'].median(), inplace=True)
df_clean['City'].fillna('Unknown', inplace=True)
df_clean.drop_duplicates(inplace=True)
df_clean.to_csv(cleaned_path, index=False)
print("Saved cleaned dataset to:", cleaned_path)
display(df_clean.head())
```

23] ✓ 0.0s

PART B — Matplotlib (visualization)

Examples of line plots, multiple lines, bar charts, histograms, scatter plots with conditional coloring, and pie charts.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
%matplotlib inline

df_plot = pd.read_csv("plot_dataset.csv")
df_plot.head()
```

24] ✓ 0.0s

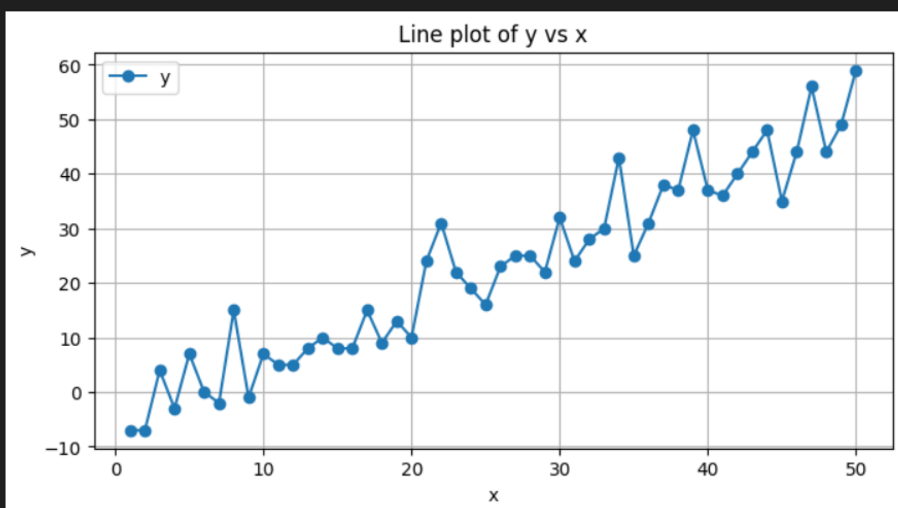
	x	y	y2
0	1	-7	-7
1	2	-7	-7
2	3	4	4
3	4	-3	-3
4	5	7	7

Line plot — explain axes, labels, and title

```python
x = df_plot['x']
y = df_plot['y']

plt.figure(figsize=(8,4))
plt.plot(x, y, label='y', marker='o')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Line plot of y vs x')
plt.legend()
plt.grid(True)
plt.show()
```

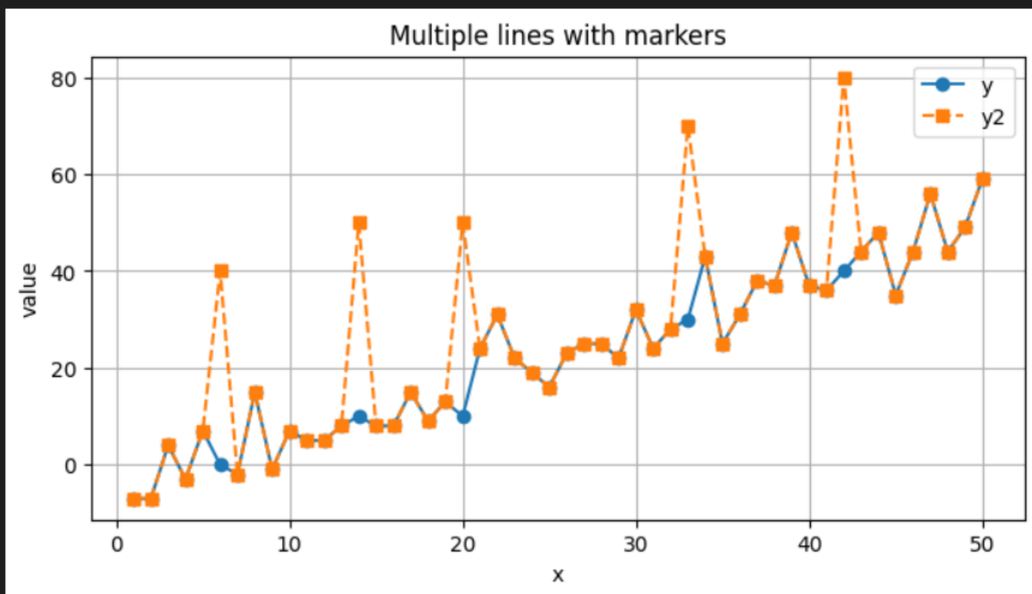
✓ 0.0s Python



Multiple lines and legends

```
plt.figure(figsize=(8,4))
plt.plot(df_plot['x'], df_plot['y'], label='y', linestyle='-', marker='o')
plt.plot(df_plot['x'], df_plot['y2'], label='y2', linestyle='--', marker='s')
plt.xlabel('x')
plt.ylabel('value')
plt.title('Multiple lines with markers')
plt.legend()
plt.grid(True)
plt.show()
```

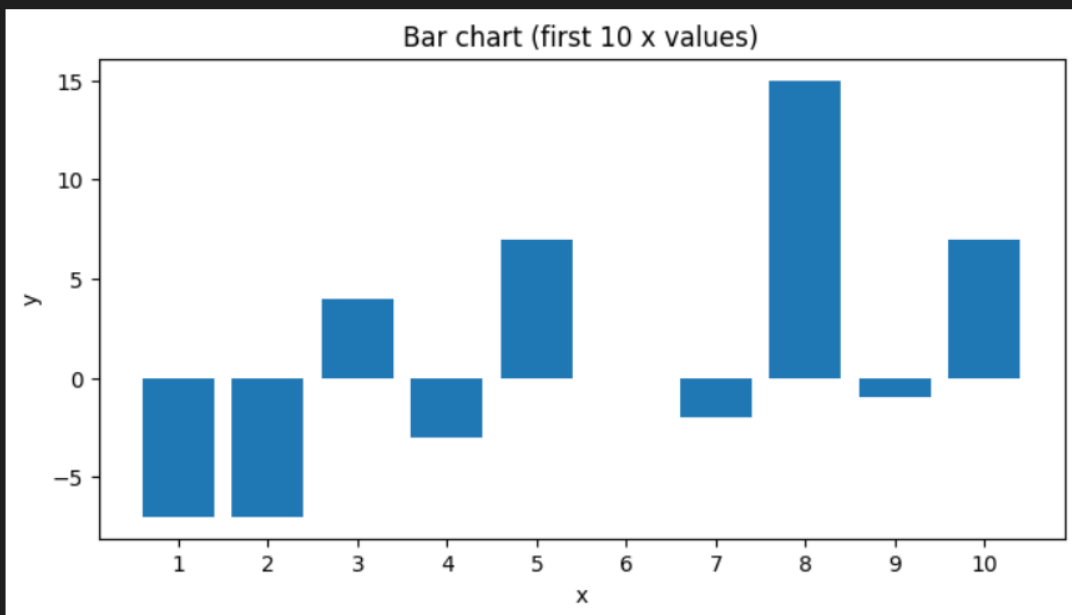
[4] ✓ 0.0s



Bar chart example

```
plt.figure(figsize=(8,4))
vals = df_plot['y'].head(10)
labels = df_plot['x'].head(10).astype(str)
plt.bar(labels, vals)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Bar chart (first 10 x values)')
plt.show()
```

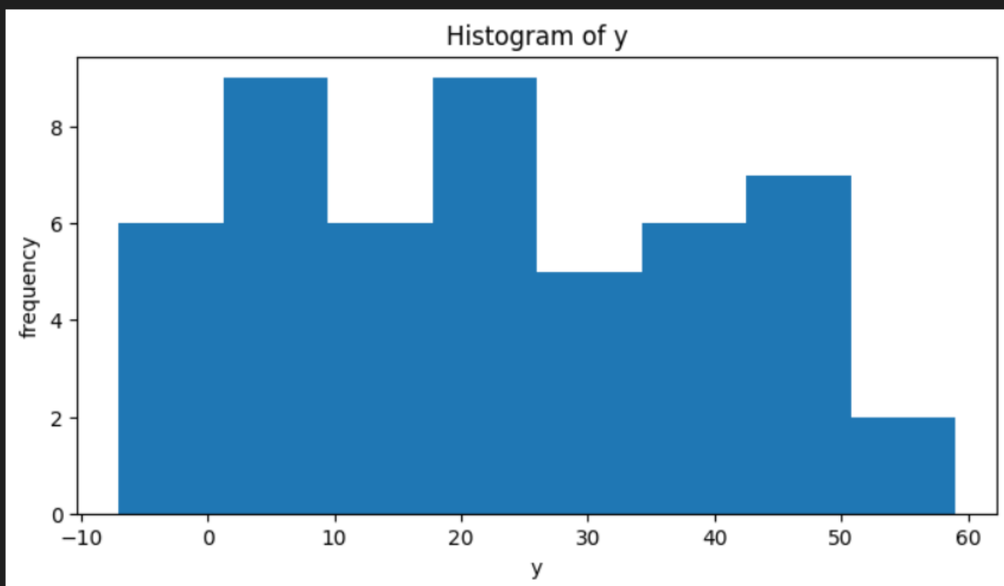
[5] ✓ 0.0s



Histogram example — bins and interpretation

```
plt.figure(figsize=(8,4))
plt.hist(df_plot['y'], bins=8)
plt.xlabel('y')
plt.ylabel('frequency')
plt.title('Histogram of y')
plt.show()
```

0.0s



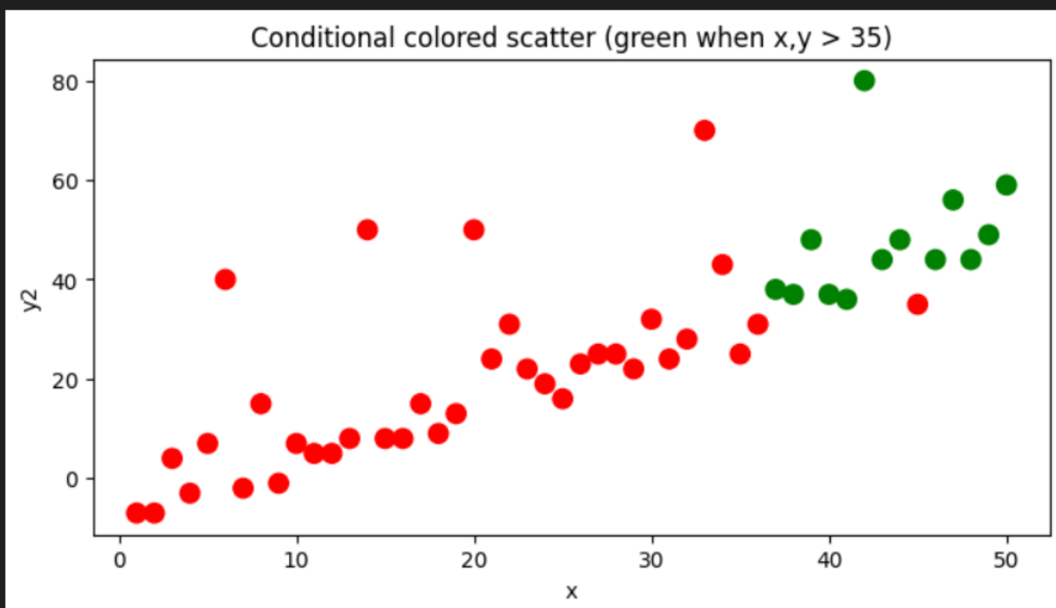
Scatter plot — conditional coloring (assignment)

Make points green when both x and y > 35; otherwise red. Discuss how to vectorize this.

```
x = df_plot['x']
y = df_plot['y2']
colors = ['green' if (x.iloc[i] > 35 and y.iloc[i] > 35) else 'red' for i in range(len(x))]

plt.figure(figsize=(8,4))
plt.scatter(x, y, c=colors, s=80)
plt.xlabel('x')
plt.ylabel('y2')
plt.title('Conditional colored scatter (green when x,y > 35)')
plt.show()
```

[7] ✓ 0.0s



Pie chart example

```python
activities = ['eat','sleep','work','play']
slices = [3,7,8,6]
plt.figure(figsize=(5,5))
plt.pie(slices, labels=activities, autopct='%1.1f%%', startangle=90, explode=(0,0,0.1,0))
plt.title('Pie chart example')
plt.show()
```

[8] ✓ 0.0s