Faculty of Media Engineering and Technology
German University in Cairo
Dr. Nourhan Ehab

CSEN 903: Advanced Computer Lab, Winter 2025
Lab 1 Manual: Data Wrangling and Visualization

Part 1: Read and Watch Before the Lab

Data is the heart of AI. Before we train models, we need to *clean*, *organize*, and *understand* the data. This process is called *data wrangling* (or *data preprocessing*). It involves tasks like:

- Loading data from files (e.g., CSV)
- Inspecting the data
- Handling missing values and duplicates
- Converting data types (e.g., strings to numbers)
- Calculating basic statistics (mean, median, correlations)

Once the data is clean, *visualization* helps us uncover patterns, trends, and anomalies. We can use plots like:

- *Histograms*(to see distributions)
- *Scatter plots*(to explore relationships)
- *Heatmaps* (to visualize correlations)

These tools are essential because *"garbage in, garbage out"*. If your data is messy, your AI models will fail!

Required Readings:

1. Pandas Documentation (Getting Started)
2. Matplotlib Pyplot Tutorial
3. Seaborn Cheat Sheet

Required Watch List:

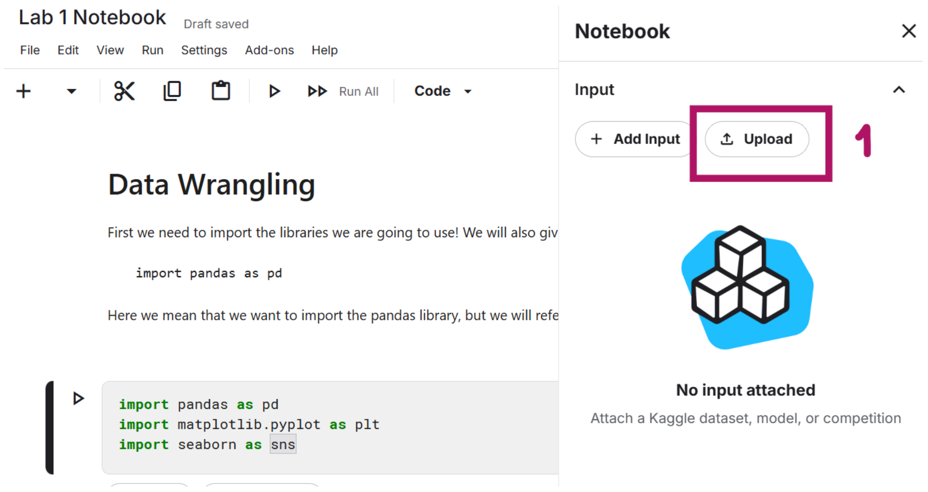
1. What is Data Wrangling?
2. Pandas in Python for Beginners (freeCodeCamp)
3. Data Visualization in Python (ProgrammingKnowledge)

Part 2: In-Lab Task

1. Set Up Your Environment. Create a new notebook and import the following libraries.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Load the Dataset. Either upload it manually or use the Add Input feature on Kaggle if the dataset is readily available on the platform.
   a. Manually:
      i. In the input tab, click 'Upload'



Data Wrangling

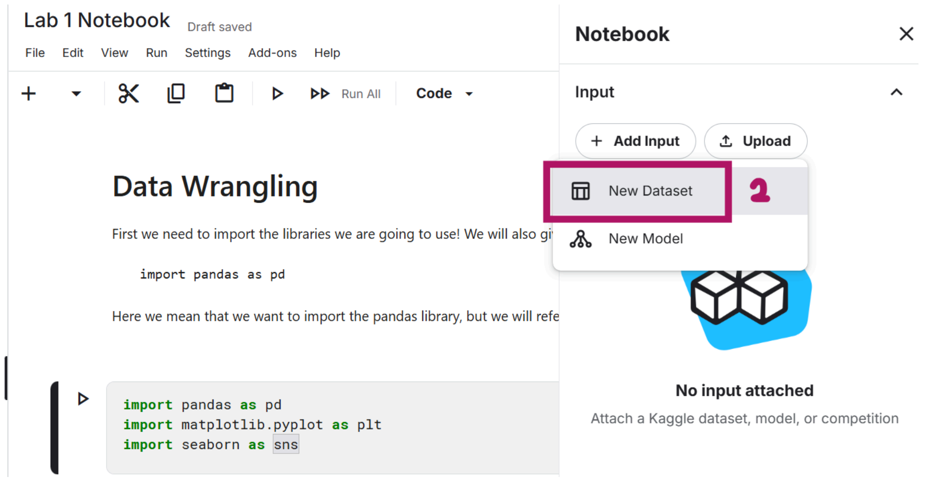
First we need to import the libraries we are going to use! We will also giv

```
import pandas as pd
```

Here we mean that we want to import the pandas library, but we will refe

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

      ii. Choose New 'Dataset'



Data Wrangling

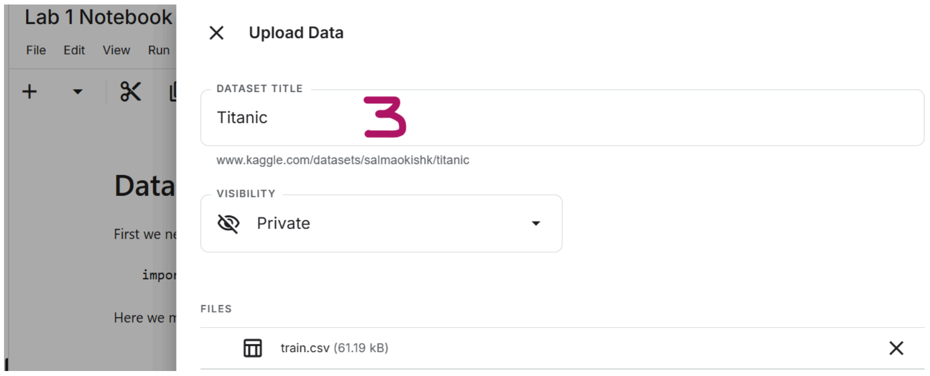
First we need to import the libraries we are going to use! We will also gi

```
import pandas as pd
```

Here we mean that we want to import the pandas library, but we will refe

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

iii. After dropping your files in, add a name for your dataset



DATASET TITLE: Titanic

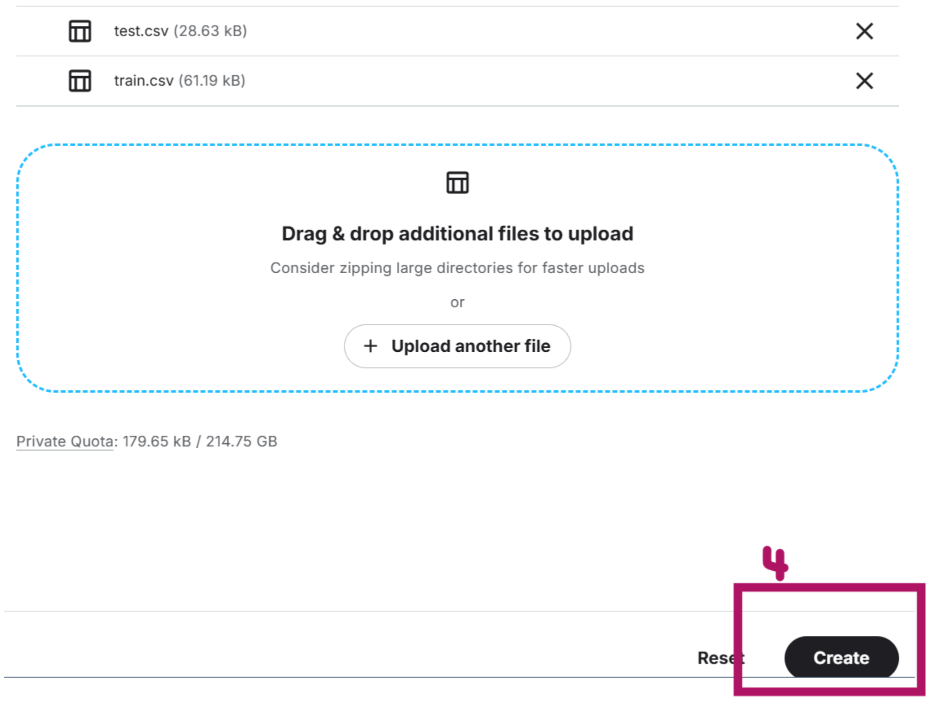
www.kaggle.com/datasets/salmaokishk/titanic

VISIBILITY: Private

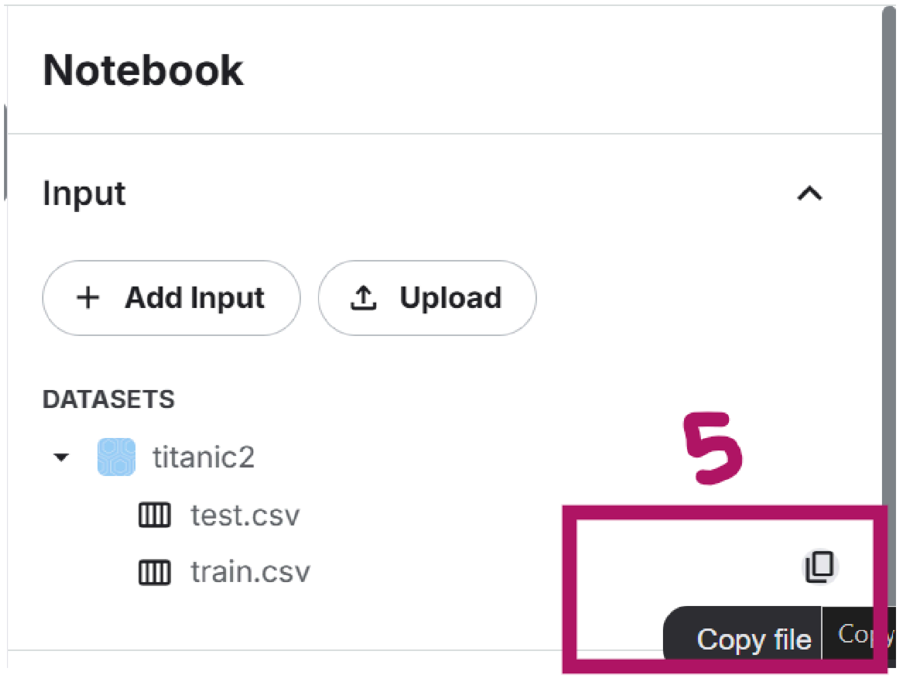
FILES

train.csv (61.19 kB)

iv. Make sure you have uploaded everything (if the dataset has multiple files, upload all of them), then click 'Create' at the bottom of the tab.

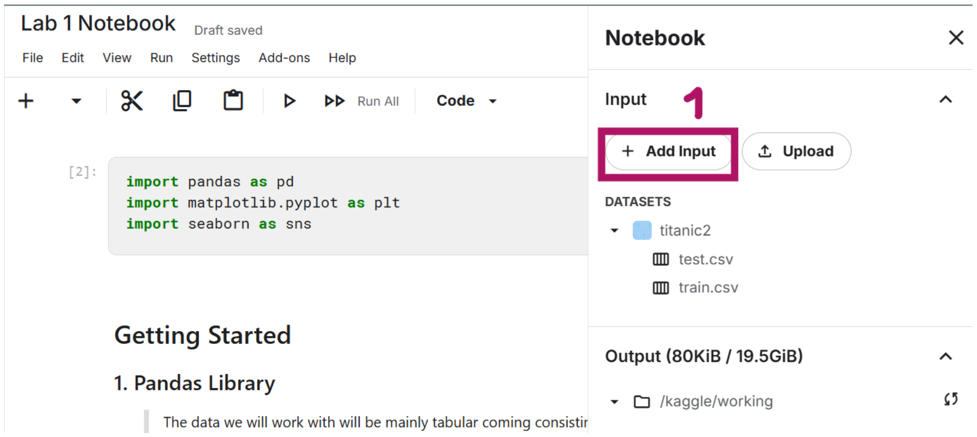

test.csv (28.63 kB)
train.csv (61.19 kB)
Drag & drop additional files to upload
Consider zipping large directories for faster uploads
or
Upload another file
Private Quota: 179.65 kB / 214.75 GB
4
Reset
Create

v. Copy the path of the file so we can read it



b. Add Input:

i. Click 'Add Input'



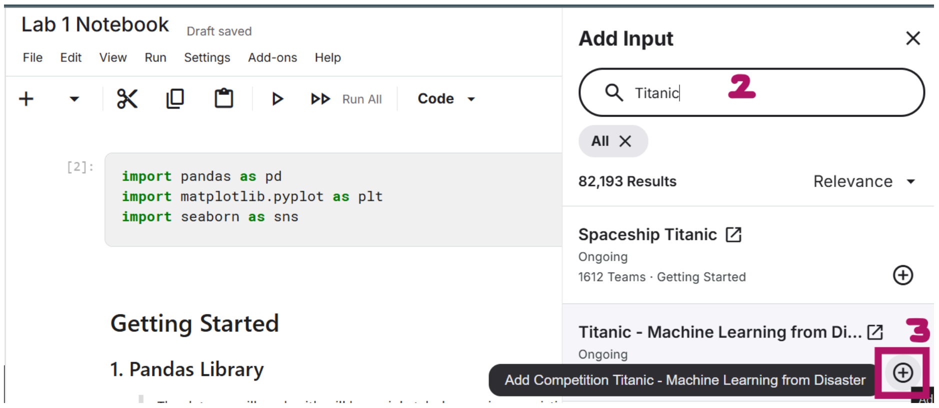
```
[2]: import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
```

Getting Started

1. Pandas Library

The data we will work with will be mainly tabular coming consistir

ii. Type in your dataset name and click on the add button


Lab 1 Notebook Draft saved

File Edit View Run Settings Add-ons Help

```
[2]: import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
```

Getting Started

1. Pandas Library

Add Input

Titanic

All

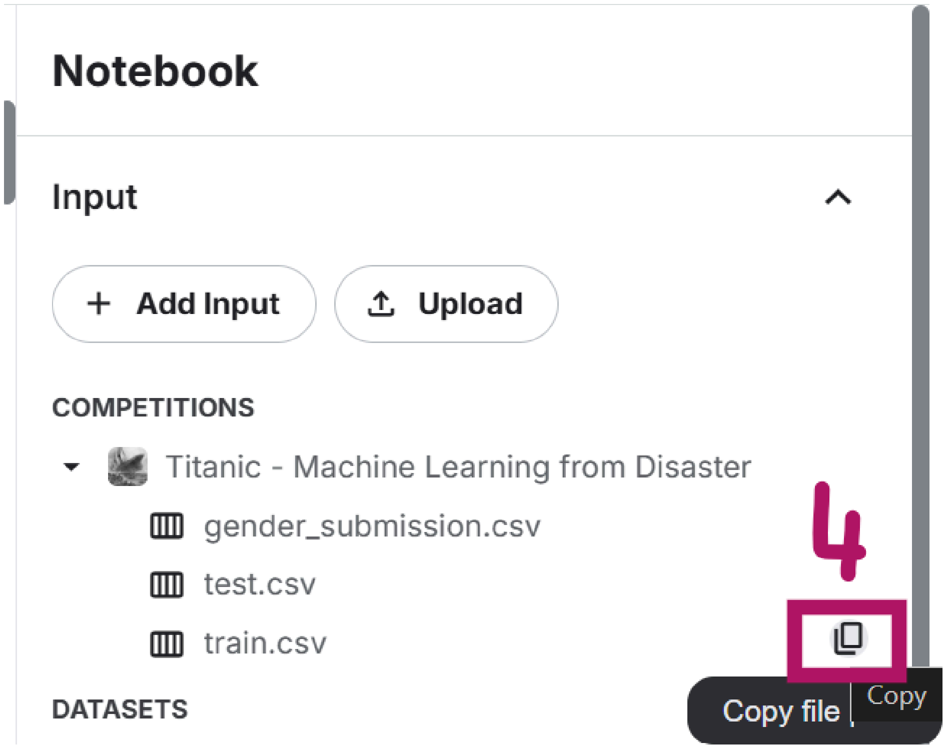
82,193 Results Relevance

Spaceship Titanic
Ongoing
1612 Teams · Getting Started

Titanic - Machine Learning from Di...
Ongoing

Add Competition Titanic - Machine Learning from Disaster

iii. Copy the file path and read the CSV


Notebook

Input

COMPETITIONS

Titanic - Machine Learning from Disaster
- gender_submission.csv
- test.csv
- train.csv

DATASETS

c. Read the CSV & check the first 5 rows

```
df = pd.read_csv('/kaggle/input/titanic2/train.csv')
# The line in red is the copied path^^
df.head() # This shows you the first 5 rows of the table!
```

Before we dive into cleaning and visualization, it's essential to understand the data we're working with, and that's where Pandas comes in.

Pandas is a powerful Python library used for data manipulation and analysis. It introduces the DataFrame, a 2-dimensional, table-like data structure that allows you to store, access, and manipulate data efficiently.

When working with real-world datasets, things are often messy; containing missing values, incorrect formats, or inconsistent categories. That's why it's critical to explore your data first.

3. Inspect the data.
    a. **Check the structure** → We need information such as the 'shape' of our data (number of rows & columns). We usually deal with 2 types of data:
        i. Numerical Data: represents measurable or countable quantities (like height or number of items).
        ii. Categorical Data: represents labels or names that group information into categories (like color or gender).

       We can also see things such as data usage, and the number of null values. This helps us get some insights on how to clean the data.
    b. **Get summary statistics** → Getting the statistical summary for the numerical values can be very beneficial in telling us whether there are outliers in our data, which can mislead our models.
    c. **Check for missing values** → These need to be dealt with as they can impact our model's performance or even create errors.
    d. **Explore categorical features** → These variables often need to be converted into numerical form, and exploring them early helps us plan how to handle them correctly, especially if there are inconsistencies (like "Male", "male", "M") or rare categories that may need special treatment.

```python
# Do each in a separate cell
df.info()
df.describe()
df.isnull().sum()

df['Sex'].value_counts()
df['Embarked'].value_counts()
```

4. Clean the data. From our exploration, we found that there are 177 missing values

in 'Age', 687 in 'Cabin', and 2 in 'Embarked'. When dealing with missing data, we have two options: **impute** or **remove**. **Imputation** involves filling in missing values with plausible, estimated ones. Depending on the number and distribution of the missing values, we decide whether to **fill them in** or simply **remove** the problematic rows or columns. We can also try to visualize **missing data** patterns!

Sometimes there is a pattern in the missing data that can help us make more informed decisions! So why not try visualizing where the missing values are before we start dealing with them?

```
import missingno as msno
msno.matrix(df)
```

a. Drop unnecessary columns (e.g., `Cabin` since we have over 77% missing): (Think about it! Does it make sense to "guess" the remaining 77% from the 23% we have? Is Cabin even that important of a feature?)

```
df = df.drop(columns=['Cabin'])
```

b. Fill missing `Age` values with **median**: With only 177 missing values and an important feature like Age, it's better to fill the missing entries with an appropriate estimate. We could've chosen the **mean** or the **mode**, but we went with the **median**. The median usually provides the most accurate estimate of the central value, as it isn't affected by **outliers** like the mean or by **frequent repetitions** like the mode.

```
df['Age'] = df['Age'].fillna(df['Age'].median())
```

c. Fill `Embarked` with **mode**: We can't use the mean for categorical values, nor the median, since this column isn't ordinal, and with only 2 missing entries, filling them with the **mode** (most frequent value) is the most appropriate choice.

```
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
```

d. Encode categorical features: This is the process of converting categorical variables into numerical form. The two main types of encoding are **Label Encoding & One Hot Encoding.**

i. **Label Encoding:** If we have a column named `Difficulty` with categories (difficulty ratings), we can transform each category into a number. Example: `['low','medium','high'] -> [0,1,2]`

ii. **One Hot Encoding:** If we have a column named Color with different colors, we can create binary columns for each color!

COLOR	RED	GREEN	BLUE
Red	1	0	0
Green	0	1	0
Blue	0	0	1

iii. **When to use which?**

1. **When the categories are truly ordered (ordinal)** → Use **Label Encoding** (since 0 < 1 < 2, this will imply that low < medium < high, which is what we want!)

2. **When the categories are unordered (nominal)** → Use **One Hot Encoding** (if we use label encoding, the model might learn that red < green < blue, which doesn't make any sense!)

iv. So for our dataset, we will encode Sex and Embarked once using a mapping and the other using the predefined function get_dummies, which automatically creates the new columns for us. The drop_first parameter basically makes us return k-1 columns if we have k categories. (Since if I tell you it's not red nor green, you will automatically know it's blue!)

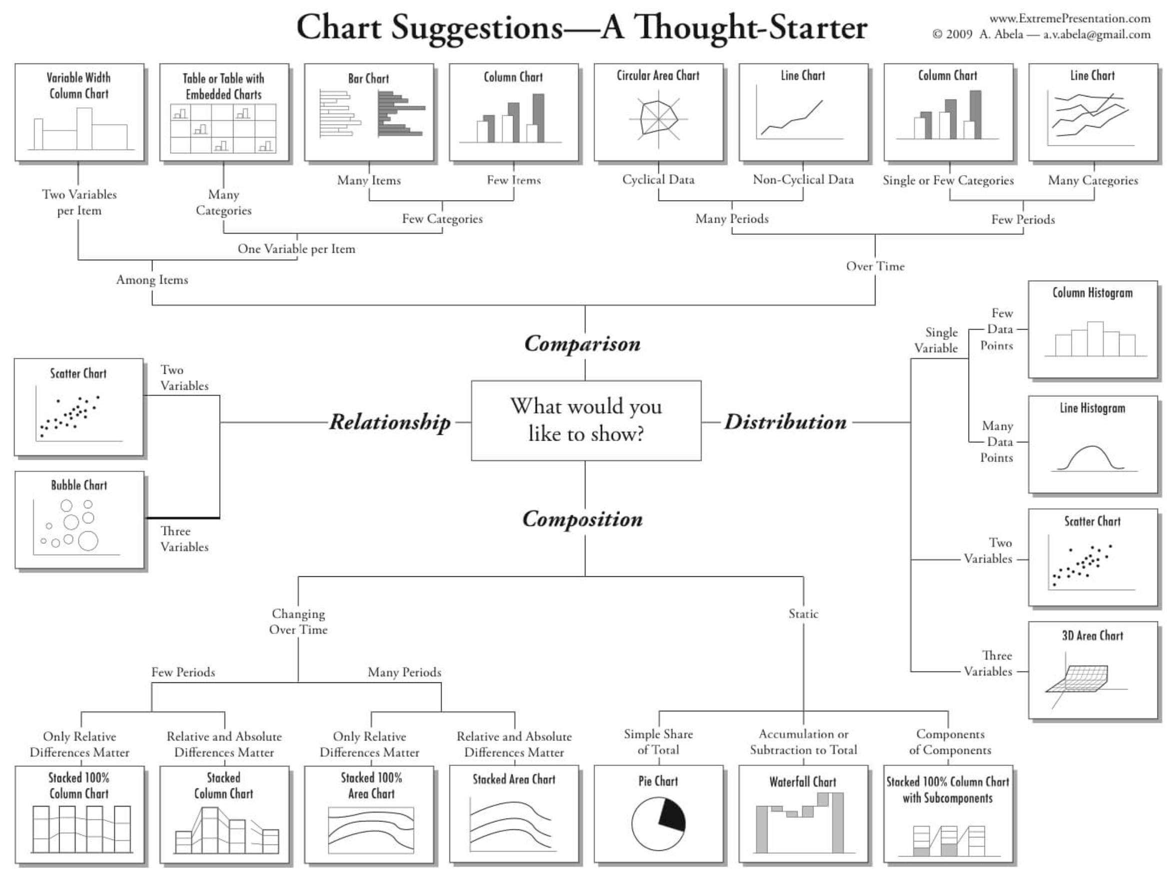
```
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
df = pd.get_dummies(df, columns=['Embarked'], drop_first=True)
```

5. Create Visualizations; We will explore two libraries here: **Matplotlib** and **Seaborn**.
   a. **Matplotlib Library:** Matplotlib is the core plotting library in Python, used to create a wide range of static, interactive, and animated visualizations. It gives you full control over every aspect of a plot, making it ideal for custom or complex charts.

b. **Seaborn Library:** Seaborn is a high-level data visualization library built on top of Matplotlib. It simplifies the process of creating attractive and informative statistical graphics, especially when working with Pandas DataFrames.

We can try to categorize the visualizations into 4 types, depending on what exactly it is you want to see or show.

1. **Comparison** – Shows differences between categories or groups.
2. **Distribution** – Reveals how data is spread across a range of values.
3. **Composition** – Displays parts of a whole and how they change.
4. **Relationship** – Highlights connections or correlations between variables.



c. **Histogram** of `Age`:

Histograms are used to show the distribution of a numerical variable, which helps us understand the shape of the data, such as its skewness, center, and spread.

- **X-axis:** Continuous intervals (bins).
- **Y-axis:** Frequency/count of observations in each bin.

```
plt.hist(df['Age'], bins=20)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

d. **Scatter plot** of `Age` vs `Fare`:

Scatter Plots are used to show the **relationship** between **two numerical variables**. It helps identify correlations and clusters.

- **X-axis:** Continuous numerical variable 1
- **Y-axis:** Continuous numerical variable 2

```
plt.scatter(df['Age'], df['Fare'])
plt.xlabel('Age')
plt.ylabel('Fare')
plt.title('Age vs Fare')
plt.show()
```

- Bubble Charts are similar to scatter plots in the sense that they show us **relationships,** but with an added dimension (bubble size) in order to accommodate having **three numerical variables.**

    - **X-axis:** Continuous numerical variable 1
    - **Y-axis:** Continuous numerical variable 2
    - **Size:** Continuous numerical variable 3

Hint: You can also add the color attribute to accommodate another variable.

e. **Heatmap** of correlations for `Sex`, `Pclass` & `Survived`

Heatmaps are powerful tools that help us visualize **patterns, relationships, and correlations** in data using color intensity. They're highly flexible and can work with **both categorical and numerical variables;** however, numerical variables will need binning.
Heatmaps are especially useful when trying to spot trends at a glance.

Depending on the context:

- **X-axis:** Categorical or numerical variable
- **Y-axis:** Categorical or numerical variable
- **Color intensity:** Represents a third numerical variable (e.g., frequency, correlation, aggregated function)

```
pivot = pd.pivot_table(df, index="Sex", columns="Pclass",
values="Survived", aggfunc="mean")

sns.heatmap(pivot, annot=True, cmap="coolwarm")
plt.show()
```

You'll notice we used something called a pivot table here! Pivot tables are like a "summary" of our original table. We passed in our original DataFrame, specified that rows should be grouped by `Sex`, columns by `Pclass`, and the cell values should show the mean survival rate. This basically means we wanted to find out for each gender in each class, what the survival rate was.

f. **Barplot** of survival by age group

Barplots (or Barcharts) are used to create a comparison between the quantities of categorical variables

- **X-axis:** Categorical variables.
- **Y-axis:** Value/Count for each category.

```
bins = [0, 10, 20, 30, 40, 50, 60, 70, 80]
labels = ['0-10', '11-20', '21-30', '31-40', '41-50', '51-60',
'61-70', '71-80']
```

```
df['AgeGroup'] = pd.cut(df['Age'], bins=bins, labels=labels)

sns.barplot(data=df, x='AgeGroup', y='Survived')
plt.title('Survival Rate by Age Group')
plt.show()
```

g. **Boxplot** to spot outliers.

Boxplots are used to show the **distribution** of a numerical variable and identify **outliers** or **skewness** across categories.

- **X-axis**: Categorical variables (e.g., groups or labels).
- **Y-axis**: Numerical values for each category

```
sns.boxplot(x='Pclass', y='Fare', data=df)
plt.title('Fare Distribution by Class')
plt.show()
```

Part 3: Graded Task

Task Description:

Using the *Titanic dataset*, complete the following:

1. Compare survival rates (`Survived`) by **gender** (`Sex`).
2. Visualize the distribution of **Fare** across different **passenger classes** (`Pclass`) and detect possible outliers.
3. Calculate the **average Fare** and **average Age** by **Pclass** and report the results.
4. Recreate a heatmap of all numeric features.
5. Visualize the percentage of survivors broken down by gender

Deliverables:

Submit an accessible notebook link containing all the completed tasks listed above via the form. The submission **deadline** is by the end of your lab.