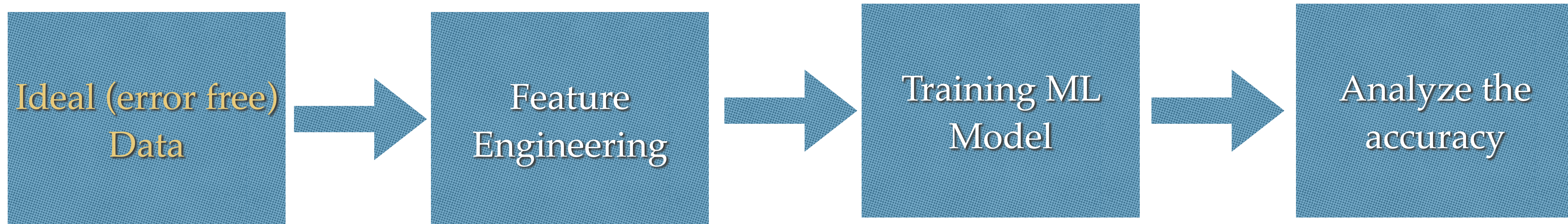

Exploratory Data Analysis (EDA)

What You Learned So Far

Step 1



Step 2



Exploratory Data Analysis (EDA)

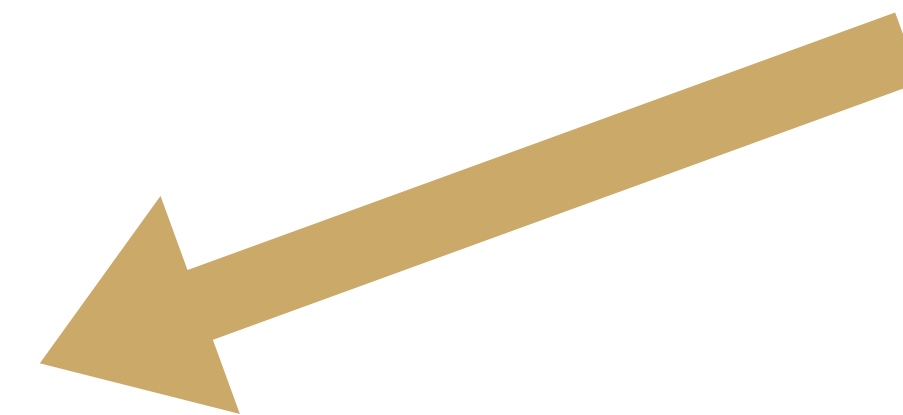
- **Goals of EDA:**

1. Identify data quality issues
2. Uncover interesting patterns or trends
3. Understand variable relationships
4. Generate hypotheses for further analysis



- **Techniques used in EDA:**

1. Summary statistics
2. Data cleaning and preprocessing
3. Graphical representations
4. Correlation analysis
5. Data transformation



- **EDA helps in:**

1. Selecting appropriate modeling techniques
2. Refining research questions
3. Leading to more informed decision-making
4. Gaining insights from the data.

Content

1. Data cleaning and preprocessing:

- This step involves handling missing values, removing duplicates, and transforming the data into a suitable format. It lays the foundation for accurate and meaningful analysis.

2. Summary statistics:

- Once the data is cleaned and preprocessed, it is beneficial to compute summary statistics such as mean, median, standard deviation, and quartiles. These statistics provide a concise overview of the data distribution and key insights.

3. Graphical representations:

- Creating visualizations, such as histograms, box plots, scatter plots, and bar charts, helps to explore the data visually. Visualizations can reveal patterns, trends, outliers, and relationships among variables.

4. Correlation analysis:

- Analyzing the correlation between variables helps understand the strength and direction of their relationships. Techniques such as correlation matrices, scatter plots, and heatmaps can provide insights into dependencies among variables.

5. Data transformation:

- Data transformation involves modifying or creating new variables to improve the analysis or meet specific requirements. Examples include feature scaling, log transformations, or creating derived variables.

Example: Titanic Dataset

We use titanic dataset to learn how to deal with missing data.

You can download the dataset from <https://www.kaggle.com/competitions/titanic/overview>

```
#Importing Pandas Python library
import pandas as pd
```

```
#Importing Numpy Python Library
import numpy as np
```

```
#Read the dataset.
#You can download this from (https://www.kaggle.com/competitions/titanic/overview)
df = pd.read_csv('titanic/train.csv')
```

```
#Show the first 10 rows
df.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	54.0	1	0	237736	30.0708	NaN	C

Example: Titanic Dataset

Data Dictionary

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Above table is from the Kaggle website.

Purpose of having dataset is to predict passengers' survival status based on the passenger information.

- 0: Indicates that the passenger did not survive.
- 1: Indicates that the passenger survived.

Example: Titanic Dataset

```
#Check the number of columns and rows  
df.shape  
  
(891, 12)
```

The `shape` attribute in Pandas DataFrame returns a tuple representing the dimensions of the DataFrame.

It provides the number of rows and columns in the DataFrame.

The syntax to access the shape attribute is `dataframe.shape`.

In the code, it shows that the training subset of the titanic dataset has 891 columns and 12 rows.

That means, there're 891 instances and 11 features and one target variable.

Data Types

Numerical Data:

- Continuous: Data that can take any numerical value within a range. Examples include height, weight, temperature, etc.

Categorical Data:

- Nominal: Data that represents categories without any inherent order or ranking. Examples include gender, color, country, etc.

Text Data:

- Textual data that represents unstructured text, such as sentences, paragraphs, or documents.

Date and Time Data:

- Data that represents specific dates, times, or timestamps. It can be represented in various formats, such as "YYYY-MM-DD" for dates or "HH:MM:SS" for timestamps.

Image Data:

- Data that represents images, typically in the form of pixels with color or grayscale information. Image data requires specialized preprocessing and handling techniques.

Time Series Data:

- Data that is recorded over time at regular intervals. Examples include stock prices, temperature measurements, sensor readings, etc.

Data Types

In this session, we focus on Numerical and Categorical data types


	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

Numerical

Categorical

Data Cleaning and Preprocessing

The data cleaning step of EDA involves

1. Handling missing values,
 2. Removing duplicates,
 3. Dealing with Outliers,
 4. Data Formatting and Standardization
 5. Data Validation and Integrity
 6. Data Transformation and Feature Engineering
- 
- data cleaning

Data cleaning is a critical step in the data analysis process.

It involves identifying and correcting or removing errors, inconsistencies, and inaccuracies in a dataset to ensure data quality and reliability.

Handling Missing Values

How to deal with NaN ("NaN" stands for "Not a Number" / missing values)?

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

- In the table, The Age and Cabin columns have NaN.
- If a data element is missing in a cell of a table, pandas assign NaN to that cell.
- When dealing with NaN in a column of a dataset, you have several options to consider depending on the nature of the data and the analysis you are performing.

Handling Missing Values

How to deal with NaN (missing values)?

Identifying Missing Values using isnull() function

```
df.isnull().head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0		False	False	False	False	False	False	False	False	False	True	False
1		False	False	False	False	False	False	False	False	False	False	False
2		False	False	False	False	False	False	False	False	False	True	False
3		False	False	False	False	False	False	False	False	False	False	False
4		False	False	False	False	False	False	False	False	False	True	False
5		False	False	False	False	True	False	False	False	False	True	False
6		False	False	False	False	False	False	False	False	False	False	False
7		False	False	False	False	False	False	False	False	False	True	False
8		False	False	False	False	False	False	False	False	False	True	False
9		False	False	False	False	False	False	False	False	False	True	False

Handling Missing Values

How to deal with NaN (missing values)?

Identifying Missing Values using `isna()` function

```
df.isna().head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False
5	False	False	False	False	False	True	False	False	False	False	True	False
6	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	True	False
8	False	False	False	False	False	False	False	False	False	False	True	False
9	False	False	False	False	False	False	False	False	False	False	True	False

Handling Missing Values

How to deal with NaN (missing values)?

Count the missing values

```
df.isna().sum()
```

```
PassengerId      0
Survived          0
Pclass           0
Name             0
Sex              0
Age             177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin           687
Embarked         2
dtype: int64
```


Handling Missing Values

Drop the rows with NaN:

- If the NaN values are relatively few and randomly distributed, you can choose to drop the rows containing NaN values using the `dropna()` method in Pandas.
- This approach can be suitable when the NaN values do not significantly impact the analysis or when you have sufficient data remaining after dropping the rows.

```
# Drop rows with any NaN value
df_dropped = df.dropna()
```

```
#Show the first 10 rows
df_dropped.head(10)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
21	22	1	2	Beesley, Mr. Lawrence	male	34.0	0	0	248698	13.0000	D56	S
23	24	1	1	Sloper, Mr. William Thompson	male	28.0	0	0	113788	35.5000	A6	S
27	28	0	1	Fortune, Mr. Charles Alexander	male	19.0	3	2	19950	263.0000	C23 C25 C27	S
52	53	1	1	Harper, Mrs. Henry Sleeper (Myna Haxtun)	female	49.0	1	0	PC 17572	76.7292	D33	C
54	55	0	1	Ostby, Mr. Engelhart Cornelius	male	65.0	0	1	113509	61.9792	B30	C

```
df_dropped.shape
```

```
(183, 12)
```

Handling Missing Values

```
# Drop rows with any NaN value
df_dropped = df.dropna()
```

```
#Show the first 10 rows
df_dropped.head(10)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
21	22	1	2	Beesley, Mr. Lawrence	male	34.0	0	0	248698	13.0000	D56	S
23	24	1	1	Sloper, Mr. William Thompson	male	28.0	0	0	113788	35.5000	A6	S
27	28	0	1	Fortune, Mr. Charles Alexander	male	19.0	3	2	19950	263.0000	C23 C25 C27	S
52	53	1	1	Harper, Mrs. Henry Sleeper (Myna Haxtun)	female	49.0	1	0	PC 17572	76.7292	D33	C
54	55	0	1	Ostby, Mr. Engelhart Cornelius	male	65.0	0	1	113509	61.9792	B30	C

```
df_dropped.shape
```

```
(183, 12)
```

`dropna()`: This method drops any rows that contain at least one NaN value, resulting in a new DataFrame (`df_dropped`) without those rows.

- The dataset exhibits a significant number of missing values.
- This leads to a reduction in the number of columns from 891 to 183.
- As a result, we are left with a limited amount of data to train our machine learning model.
- Simply dropping NaN values may not be a viable solution due¹⁶ to the high volume of missing data.

Handling Missing Values

Fill NaN values with a specific value.

You can use the following methods.

- Fill NaN values with mean value of the column
- Fill NaN values with median value of the column
- Fill NaN values with nearest non-null value
- Fill NaN values with most frequent non-null value
- Fill NaN values with a desired value

Handling Missing Values

Filling NaN Values with Mean Values

```
#Creating a copy of df
df_mean = df.copy()

# Identify numerical columns
numerical_columns = df.select_dtypes(include='number').columns

# Fill NaN values with mean for numerical columns
df_mean[numerical_columns] = df_mean[numerical_columns].fillna(df_mean[numerical_columns].mean())

#Show the first 10 rows
df_mean.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	29.699118	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.000000	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.000000	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.000000	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.000000	1	0	237736	30.0708	NaN	C

We can replace the NaN values with the mean value of the column.

We use **Pandas DataFrame's fillna()** function.

The argument of the fillna() function is the mean value of the column.

Note that we can apply this only for numerical columns.

We can not apply this for string columns.

Handling Missing Values

Filling NaN Values with Median Values

```
#Creating a copy of df
df_median = df.copy()

# Identify numerical columns
numerical_columns = df.select_dtypes(include='number').columns

# Fill NaN values with mean for numerical columns
df_median[numerical_columns] = df_median[numerical_columns].fillna(df_median[numerical_columns].mean())
```

```
numerical_columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare'], dtype='object')
```

```
#Show the first 10 rows
df_median.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	29.699118	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.000000	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.000000	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.000000	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.000000	1	0	237736	30.0708	NaN	C

We can replace the NaN values with the **median** value of the column.

We use **Pandas DataFrame's fillna()** function.

The argument of the fillna() function is the median value of the column.

Note that we can apply this only for numerical columns.

We can not apply this for string columns.

Handling Missing Values

Filling NaN Values with Most frequent Values in String Columns (Categorical)

```
#Creating a copy of df
df_mf = df_mean.copy()

# Identify string columns
string_columns = df_mf.select_dtypes(include='object').columns
```

```
string_columns

Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

```
for column in string_columns:
    df_mf[column].fillna(df_mf[column].mode()[0], inplace=True)
```

```
#Show the first 10 rows
df_mf.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	B96 B98	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	B96 B98	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.0500	B96 B98	S
5	6	0	3	Moran, Mr. James	male	29.699118	0	0	330877	8.4583	B96 B98	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.000000	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.000000	3	1	349909	21.0750	B96 B98	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.000000	0	2	347742	11.1333	B96 B98	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.000000	1	0	237736	30.0708	B96 B98	C

We can replace the NaN values with the **most frequent** value of the column.

We use **Pandas DataFrame's fillna()** function.

The argument of the fillna() function is the **mode** value of the column.

The inplace argument in the fillna() method of determines whether the changes made by the method should be applied directly to the DataFrame itself or if a modified copy of the DataFrame should be returned.

When inplace=True, the fillna() method modifies the DataFrame in place, meaning it replaces the NaN values with the specified fill values directly within the original DataFrame without creating a new copy.

Handling Missing Values

Filling NaN Values with a Desired Value in String Columns (Categorical)

```
#Creating a copy of df
df_d = df_mean.copy()

# Identify string columns
string_columns = df_mf.select_dtypes(include='object').columns
```

```
# Fill NaN values with a desired value for string columns
desired_string = 'no_value'
df_d[string_columns] = df_d[string_columns].fillna(desired_string)
```

```
#Show the first 10 rows
df_d.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	no_value	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	no_value	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.0500	no_value	S
5	6	0	3	Moran, Mr. James	male	29.699118	0	0	330877	8.4583	no_value	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.000000	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.000000	3	1	349909	21.0750	no_value	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.000000	0	2	347742	11.1333	no_value	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.000000	1	0	237736	30.0708	no_value	C

We can replace the NaN values with a desired value of the column.

We use **Pandas DataFrame's fillna()** function.

The argument of the fillna() function is a user given value.

Handling the Duplicates

1. **Bias in Training:** Duplicates can bias the training process and it can lead the model to assign higher importance to duplicated instances.
2. **Incorrect Performance Evaluation:** If duplicates are present in the dataset and not handled properly, it can result in overestimating the model's performance.
3. **Increased Training Time:** Duplicate instances unnecessarily increase the computational resources required for model training.
4. **Incorrect Statistical Analysis:** Duplicates can distort statistical analysis and lead to incorrect conclusions.

Handling the Duplicates

Adult Dataset

```
#Read the dataset.
# Load the dataset
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'
df = pd.read_csv(url, header=None)

# Set column names
column_names = ['age', 'workclass', 'fnlwgt', 'education', 'education-num',
                'marital-status', 'occupation', 'relationship', 'race', 'sex',
                'capital-gain', 'capital-loss', 'hours-per-week',
                'native-country', 'income']
df.columns = column_names
```

```
#Show the first 10 rows
df.head(4)
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K

Predict the likelihood of income exceeding \$50K per year using census data. This dataset is commonly referred to as the "Census Income" dataset.

Handling the Duplicates

```
df.duplicated(keep=False).head(20)
```

```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
10   False
11   False
12   False
13   False
14   False
15   False
16   False
17   False
18   False
19   False
dtype: bool
```

- You can isolate the duplicates of the DataFrame using `duplicated` function.
- `duplicated(keep=False)` all occurrences of duplicates are marked True.

Handling the Duplicates

```
#Show the duplicates  
#The keep=False parameter ensures that all occurrences of duplicates are marked as True.  
duplicates = df.duplicated(keep=False)  
duplicated_rows = df[duplicates]  
  
print(duplicated_rows)
```

	age	workclass	fnlwgt	education	education-num	\
2303	90	Private	52386	Some-college	10	
3917	19	Private	251579	Some-college	10	
4325	25	Private	308144	Bachelors	13	
4767	21	Private	250051	Some-college	10	
4881	25	Private	308144	Bachelors	13	
4940	38	Private	207202	HS-grad	9	
5104	90	Private	52386	Some-college	10	
5579	27	Private	255582	HS-grad	9	
5805	20	Private	107658	Some-college	10	
5842	25	Private	195994	1st-4th	2	
6990	19	Private	138153	Some-college	10	
7053	49	Self-emp-not-inc	43479	Some-college	10	
7920	49	Private	31267	7th-8th	4	
8080	21	Private	243368	Preschool	1	
8679	28	Private	274679	Masters	14	
9171	21	Private	250051	Some-college	10	
10367	42	Private	204235	Some-college	10	
11631	20	Private	107658	Some-college	10	
11865	40	Private	122616	Some-college	10	

Handling the Duplicates

Dropping the Duplicates and Finding the Unique Rows

```
# Check for duplicates  
duplicates = df.duplicated()  
num_duplicates = duplicates.sum()
```

```
# Remove duplicates  
df_unique = df.drop_duplicates()
```

```
# Display the number of duplicates and unique rows  
print("Number of duplicates:", num_duplicates)  
print("Number of unique rows:", len(df_unique))
```

```
Number of duplicates: 24  
Number of unique rows: 32537
```

- The `duplicated()` method is then used to identify duplicate rows, and the number of duplicates is calculated using `sum()`.
- Finally, the duplicates are removed using `drop_duplicates()`, and the resulting DataFrame (`df_unique`) contains only the unique rows.

One-Hot-Encoding

- Most machine learning algorithms and statistical models are designed to work with numerical data.
- Categorical variables, which represent qualitative characteristics, cannot be directly used as inputs for these models.
- One-hot encoding converts categorical variables into numerical representations, allowing them to be used as features in various modeling techniques.

Example: We have a categorical variable called **Color** with three categories Red, Blue and Green

```
import pandas as pd

# Create a DataFrame with a categorical variable
data = pd.DataFrame({'Color': ['Red', 'Blue', 'Green', 'Red', 'Green']})

print(data)
```

```
   Color
0    Red
1   Blue
2  Green
3    Red
4  Green
```

One-Hot-Encoding

We perform One-Hot-Encoding using `get_dummies` function in Pandas DataFrame

```
# Perform one-hot encoding using get_dummies()
encoded_data = pd.get_dummies(data)

print(encoded_data)
```

	Color_Blue	Color_Green	Color_Red
0	0	0	1
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

One-Hot-Encoding

Color			
Red	0	0	1
Green	0	1	0
Blue	1	0	0

- One-hot encoding is a commonly used technique in machine learning and data analysis to represent categorical variables as binary vectors.
- You can see that the Color variable has been transformed into three binary columns: `Color_Blue`, `Color_Green`, and `Color_Red`.
- Each column represents a unique category, and the value is 1 if an observation belongs to that category, and 0 otherwise.

One-Hot-Encoding

```
#Importing Pandas Python library
import pandas as pd

#Importing Numpy Python Library
import numpy as np

#Read the dataset.
df = pd.read_csv('titanic/train.csv')

#Creating a copy of df
df_mf = df.copy()
df_mf = df_mf.drop('Name', axis=1)

# Identify string columns
categorical_columns = df_mf.select_dtypes(include='object').columns

# Perform one-hot encoding using get_dummies()
encoded_data = pd.get_dummies(df_mf, columns=categorical_columns)

# Print the encoded DataFrame
print(encoded_data.head())
```


One-Hot-Encoding

```
0 PassengerId Survived Pclass Age SibSp Parch Fare Sex_female \
1      1      0      3  22.0      1      0  7.2500      0
2      2      1      1  38.0      1      0 71.2833      1
3      3      1      3  26.0      0      0  7.9250      1
4      4      1      1  35.0      1      0 53.1000      1
5      5      0      3  35.0      0      0  8.0500      0

0 Sex_male Ticket_110152 ... Cabin_F G73 Cabin_F2 Cabin_F33 Cabin_F38 \
1      1      0      ...      0      0      0      0
2      0      0      ...      0      0      0      0
3      0      0      ...      0      0      0      0
4      0      0      ...      0      0      0      0
5      1      0      ...      0      0      0      0

0 Cabin_F4 Cabin_G6 Cabin_T Embarked_C Embarked_Q Embarked_S
1      0      0      0      0      0      1
2      0      0      0      1      0      0
3      0      0      0      0      0      1
4      0      0      0      0      0      1
5      0      0      0      0      0      1
```

[5 rows x 840 columns]

Summary Statistics

Summary statistics provide a concise summary of the main characteristics of a dataset.

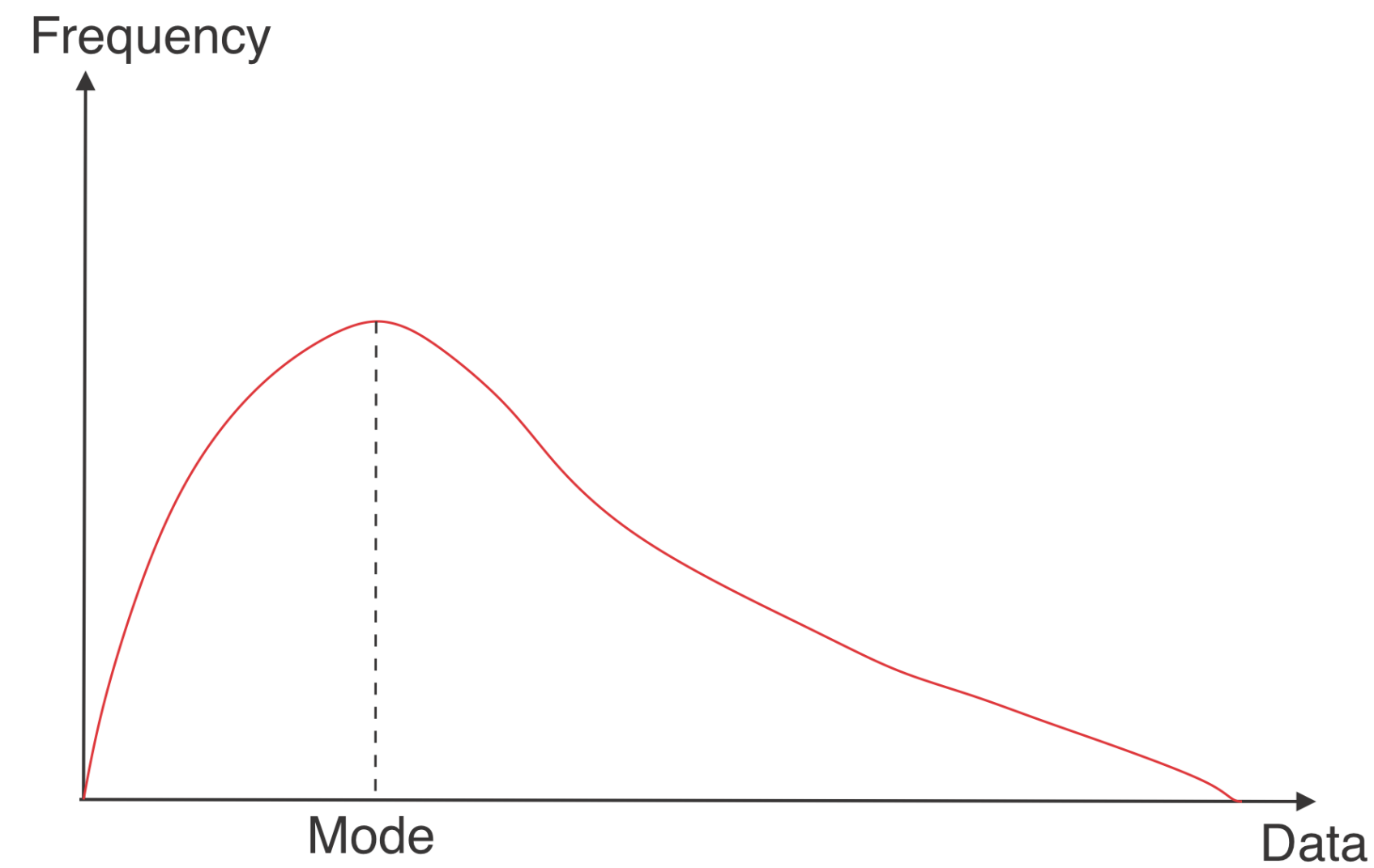
They help understand the central tendency, variability, and distribution of the data.

Common summary statistics include mean, median, mode, variance, standard deviation, minimum, maximum, and count.

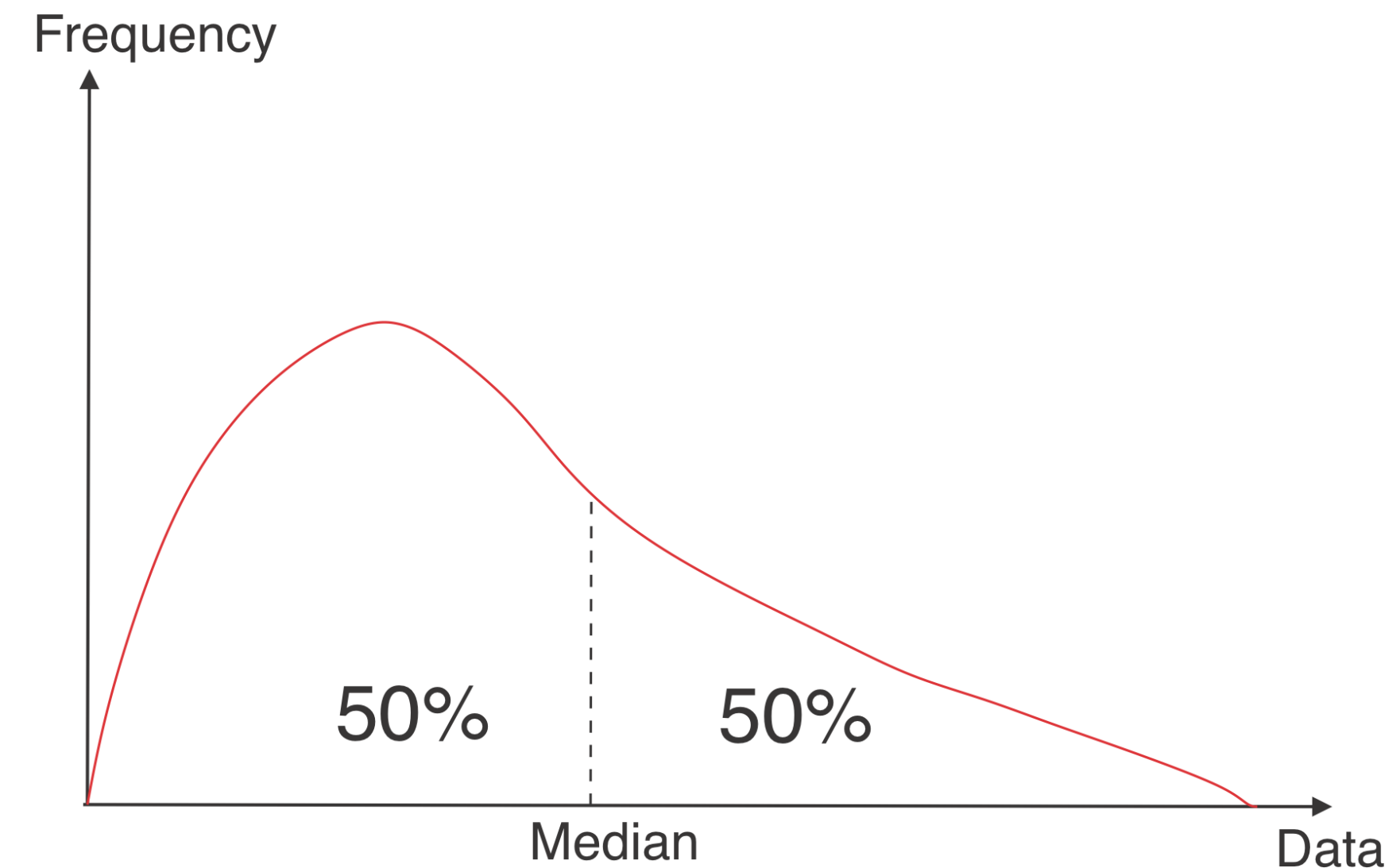
Summary statistics typically include measures such as:

- Mean: the average value of the dataset.
- Median: the middle value that separates the dataset into two equal halves.
- Mode: the most frequently occurring value(s) in the dataset.
- Variance: a measure of how spread out the values in the dataset are from the mean.
- Standard deviation: the square root of the variance, indicating the average amount of variation from the mean.
- Minimum: the smallest value in the dataset.
- Maximum: the largest value in the dataset.
- Count: the number of non-null values in the dataset.

Summary Statistics



- **Mode:** the most frequently occurring value(s) in the dataset.



- **Median:** the middle value that separates the dataset into two equal halves.

Summary Statistics

If you have N number of data elements in a column, the average value of that column is given by

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

Mathematical expression for calculating the variance of a data in a column is

$$\text{Variance} = \frac{1}{N} \sum_{i=1}^n (x_i - \bar{x})^2$$

Mathematical expression for calculating the standard deviation of a data in a column is

$$\text{Standard Deviation} = \sqrt{\text{Variance}}$$

X	Y	Z
x_0	y_0	z_0
x_1	y_1	z_1
x_2	y_2	z_2
x_3	y_3	z_4

Example: Titanic Dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  -
0   PassengerId     891 non-null   int64
1   Survived        891 non-null   int64
2   Pclass          891 non-null   int64
3   Name            891 non-null   object
4   Sex             891 non-null   object
5   Age            714 non-null   float64
6   SibSp           891 non-null   int64
7   Parch          891 non-null   int64
8   Ticket          891 non-null   object
9   Fare           891 non-null   float64
10  Cabin           204 non-null   object
11  Embarked        889 non-null   object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

The `info()` method in Pandas DataFrame provides a concise summary of the DataFrame's structure and content.

It provides information about the column names, data types, and the number of non-null values in each column.

The syntax to call the `info()` method is `dataframe.info()`.

Example: Titanic Dataset

```
df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

The `describe()` method in Pandas DataFrame provides descriptive statistics for the numerical columns in the DataFrame.

It computes various statistical measures, such as count, mean, standard deviation, minimum, quartiles, and maximum values.

The syntax to call the `describe()` method is `dataframe.describe()`.

Example: Titanic Dataset

```
#Writing a function to generate the summary statistics
def get_stat_report(df1):
    # Compute the mean of each column
    mean_values = df1.mean()

    # Compute the median of each column
    median_values = df1.median()

    # Compute the mode of each column
    mode_values = df1.mode().iloc[0] # Mode can have multiple values, so we select the first one

    # Compute the variance of each column
    variance_values = df1.var()

    # Compute the standard deviation of each column
    std_dev_values = df1.std()

    # Compute the minimum value of each column
    min_values = df1.min()

    # Compute the maximum value of each column
    max_values = df1.max()

    # Compute the count of non-null values in each column
    count_values = df1.count()

    # Combine the results into a summary DataFrame
    summary_stats = pd.DataFrame({
        'Mean': mean_values,
        'Median': median_values,
        'Mode': mode_values,
        'Variance': variance_values,
        'Std Dev': std_dev_values,
        'Min': min_values,
        'Max': max_values,
        'Count': count_values
    })

    # Display the summary statistics
    print(summary_stats)
```

By running this function, you will obtain a DataFrame that shows the summary statistics for each column in your dataset.

Here, we provide the data frame with the desired columns of the dataset as the argument of the function.

Summary statistics provide valuable insights into the dataset's characteristics and are computed using the functions of the Pandas DataFrame, such as `mean()`, `median()`, `mode()`, `var()`, `std()`, `min()`, `max()`, and `count()`.

Example: Titanic Dataset

```
#Selecting only the data of Age, Parch and Fare columns.
```

```
df1 =df[['Age','Fare']]
```

```
get_stat_report(df1)
```

	Mean	Median	Mode	Variance	Std Dev	Min	Max	Count
Age	29.699118	28.0000	24.00	211.019125	14.526497	0.42	80.0000	714
Fare	32.204208	14.4542	8.05	2469.436846	49.693429	0.00	512.3292	891

```
#Selecting only the numeric columns in the DataFrame
```

```
df2 = df.select_dtypes(include=np.number)
```

```
#Reporting the Summary Statistics for all the numerical columns
```

```
get_stat_report(df2)
```

	Mean	Median	Mode	Variance	Std Dev	Min	\
PassengerId	446.000000	446.0000	1.00	66231.000000	257.353842	1.00	
Survived	0.383838	0.0000	0.00	0.236772	0.486592	0.00	
Pclass	2.308642	3.0000	3.00	0.699015	0.836071	1.00	
Age	29.699118	28.0000	24.00	211.019125	14.526497	0.42	
SibSp	0.523008	0.0000	0.00	1.216043	1.102743	0.00	
Parch	0.381594	0.0000	0.00	0.649728	0.806057	0.00	
Fare	32.204208	14.4542	8.05	2469.436846	49.693429	0.00	

	Max	Count
PassengerId	891.0000	891
Survived	1.0000	891
Pclass	3.0000	891
Age	80.0000	714
SibSp	8.0000	891
Parch	6.0000	891
Fare	512.3292	891

In the upper cell, we print the summary statistics of only the Age and Fare columns.

In the second cell, we print that for all the columns with numerical data. Here, we use Numpy's number to check whether the columns have the numerical data.

numpy.number is a parent class for all the numeric data types in the NumPy library. It represents a generic numeric data type that is the base class for more specific numeric types such as integers, floating-point numbers, and complex numbers.

Example: Titanic Dataset

```
#Finding the mean age of males and females  
df[['Sex', 'Age']].groupby('Sex').mean()
```

	Age
Sex	
female	27.915709
male	30.726645

- The code you provided is correct for finding the mean age of males and females in a DataFrame (df).
- By using the groupby method in combination with the mean function, you can calculate the mean age for each unique value in the 'Sex' column.
- df[['Sex', 'Age']] selects only the 'Sex' and 'Age' columns from the DataFrame df.
- groupby('Sex') groups the data based on the unique values in the 'Sex' column.
- mean() calculates the mean of the 'Age' column for each group, which in this case is males and females.

Graphical Data Analysis

Some key reasons why Graphical Data Analysis is important:

1. Data visualization
2. Pattern identification
3. Communication of findings
4. Data quality assessment
5. Decision support

Graphical Data Analysis techniques:

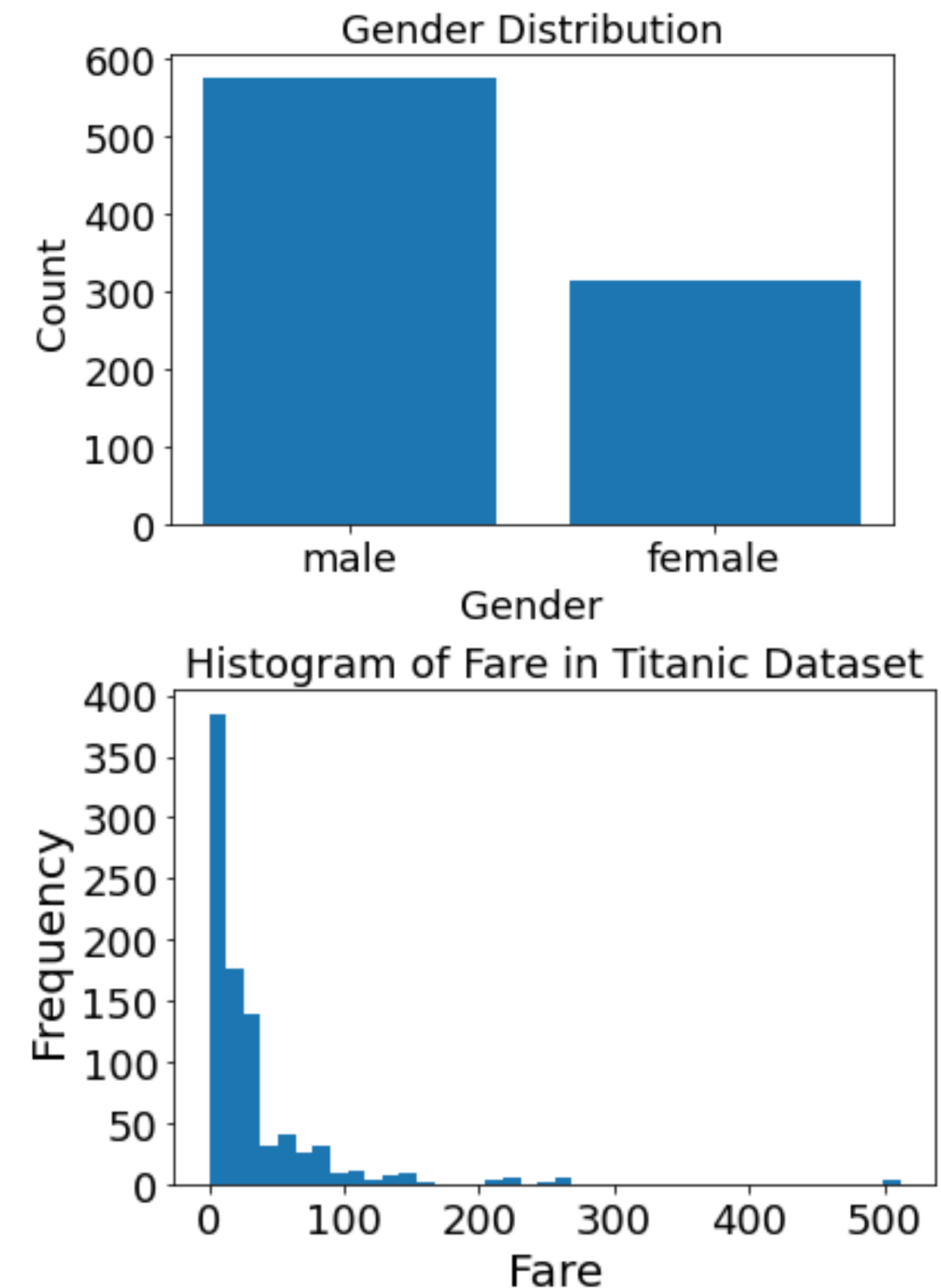
1. Bar charts
2. Histograms
3. Line charts
4. Scatter plots
5. Pie chart
6. Box plots
7. Heatmaps

Bar Charts Vs Histograms

Histograms and bar charts are both graphical representations used to visualize data, but they serve different purposes and are suitable for different types of data.

Key Differences:

- 1.**Data Type:** Histograms are used for **continuous or numerical data** (E.g.: fare), while bar charts are used for **categorical or discrete data** (E.g.: male/female).
- 2.**Representation:** Histograms represent the distribution of values, while bar charts represent the comparison or frequency of categories.
- 3.**Axis:** Histograms have a continuous axis (e.g., representing the range of values), while bar charts have a categorical axis (e.g., representing distinct categories).
- 4.**Bar Width:** In histograms, the width of each bar represents a range of values, whereas in bar charts, the width of each bar is typically uniform.



Bar Chart

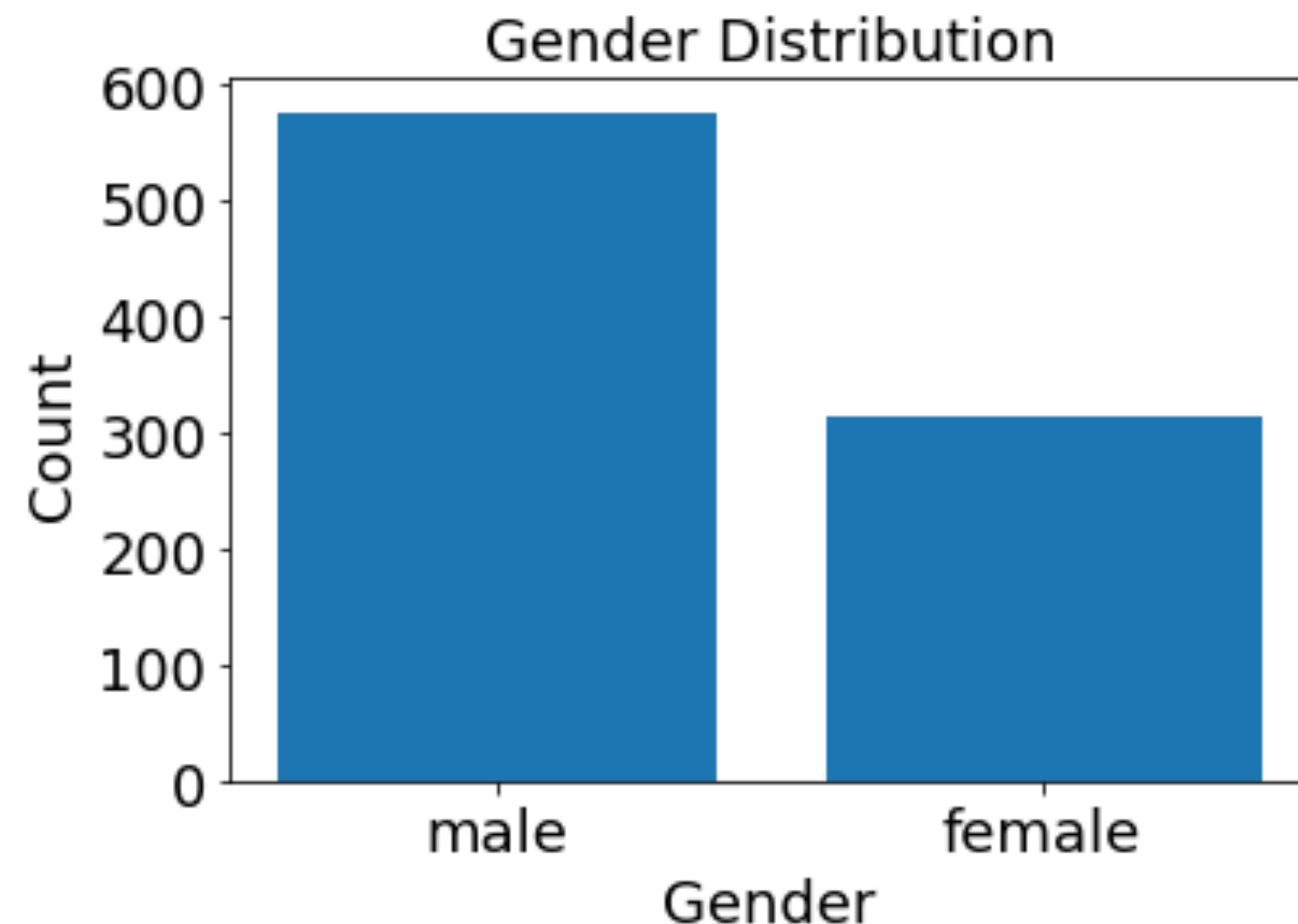
```
# Calculate the count of passengers for each gender
gender_counts = df['Sex'].value_counts()

# Create a bar chart with larger font size
plt.bar(gender_counts.index, gender_counts.values)
plt.xlabel('Gender', fontsize=18)
plt.ylabel('Count', fontsize=18)

# Set the title with larger font size
plt.title('Gender Distribution', fontsize=18)

# Increase the size of tick labels
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)

# Display the chart
plt.show()
```



we first calculate the number of passengers by gender using the `value_counts()` function on the 'Sex' column of the Titanic DataFrame.

Then, we use `plt.bar()` to create a bar chart, providing the gender categories as the x-axis values (`gender_counts.index`) and the corresponding counts as the bar heights (`gender_counts.values`).

we set the labels for the x-axis, y-axis, and title of the chart using `plt.xlabel()`, `plt.ylabel()`, and `plt.title()`, respectively.

And lastly, `plt.show()` is used to display the chart.

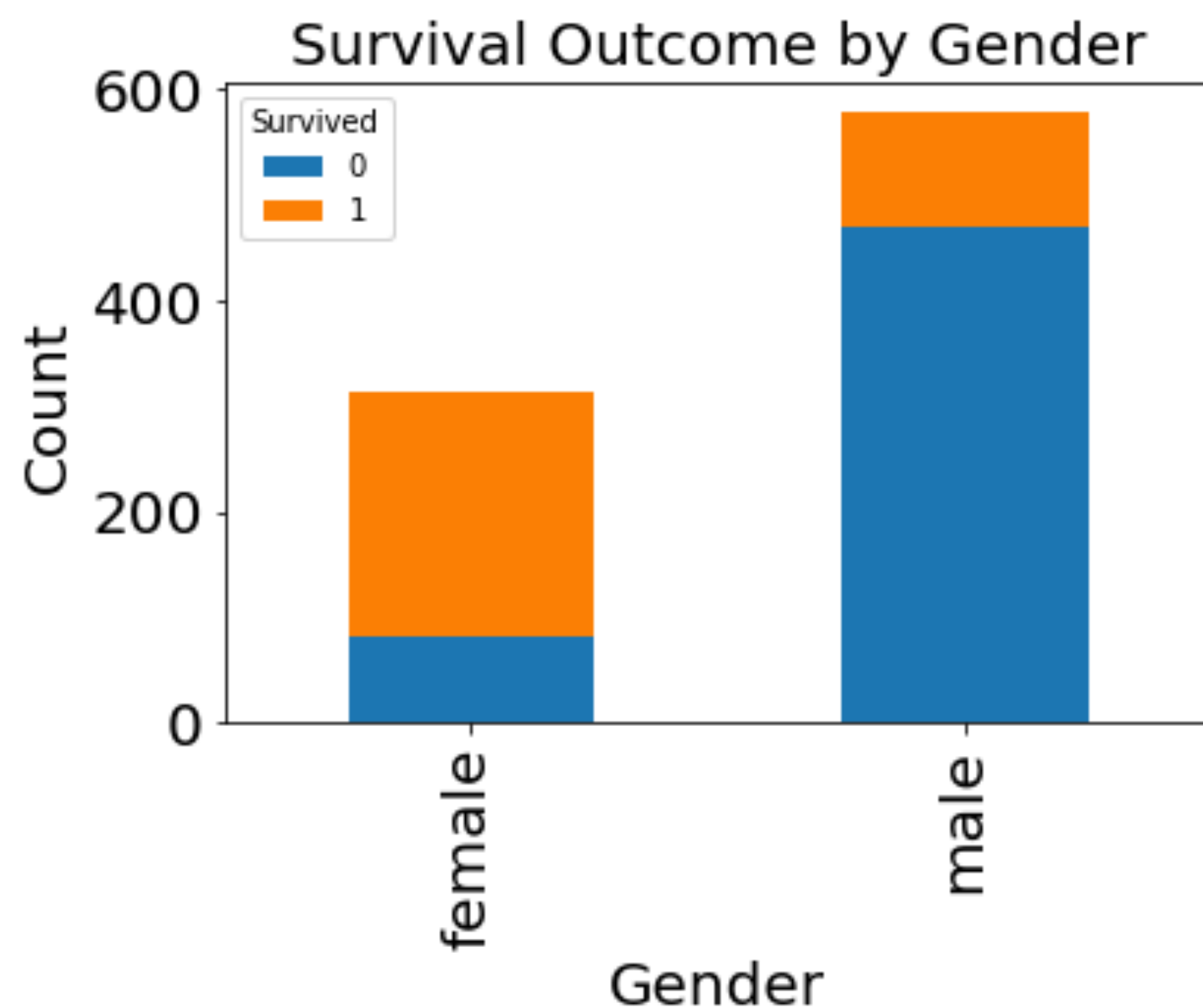
Bar chart

```
# Count the number of survivors and non-survivors by gender
survival_counts = df.groupby(['Sex', 'Survived']).size().unstack()

# Create a bar chart
ax=survival_counts.plot(kind='bar', stacked=True)

# Set the labels and title using the plot object
ax.set_xlabel('Gender', fontsize=20)
ax.set_ylabel('Count', fontsize=20)
ax.set_title('Survival Outcome by Gender', fontsize=20)

# Set the font size of the tick labels
ax.tick_params(axis='x', labelsz=20)
ax.tick_params(axis='y', labelsz=20)
```



In this example, we use the **groupby()** function on the 'Sex' and 'Survived' columns to group the data by gender and survival outcome.

Then, we use the **size()** function to calculate the count of survivors and non-survivors for each gender.

Next, we directly call `plot(kind='bar', stacked=True)` on the `survival_counts` DataFrame to create a stacked bar chart.

The **kind='bar'** parameter specifies the type of plot as a bar chart,

stacked=True stacks the bars for different survival outcomes within each gender category.

Histogram

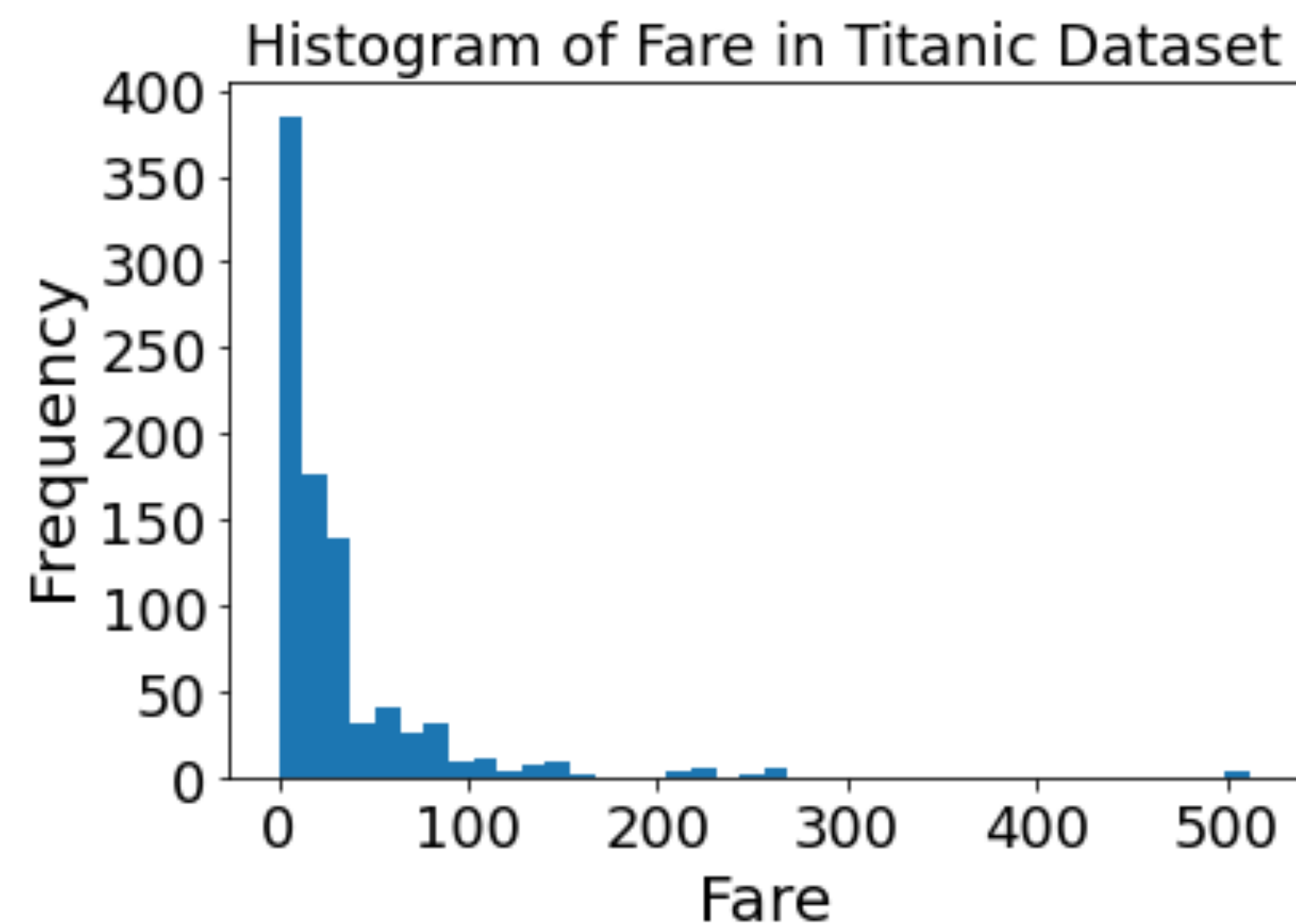
```
# Filter out missing fare values
fare_data = df['Fare'].dropna()

# Create a histogram
plt.hist(fare_data, bins=40)

# Set the labels and title
plt.xlabel('Fare', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
plt.title('Histogram of Fare in Titanic Dataset', fontsize=18)

# Increase the size of tick labels
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)

# Display the chart
plt.show()
```



We first extract the fare data from the 'Fare' column of the Titanic DataFrame and drop any missing values using the `dropna()` function.

We create a histogram using `plt.hist()`, passing the fare data and the number of bins (20 in this case) as arguments.

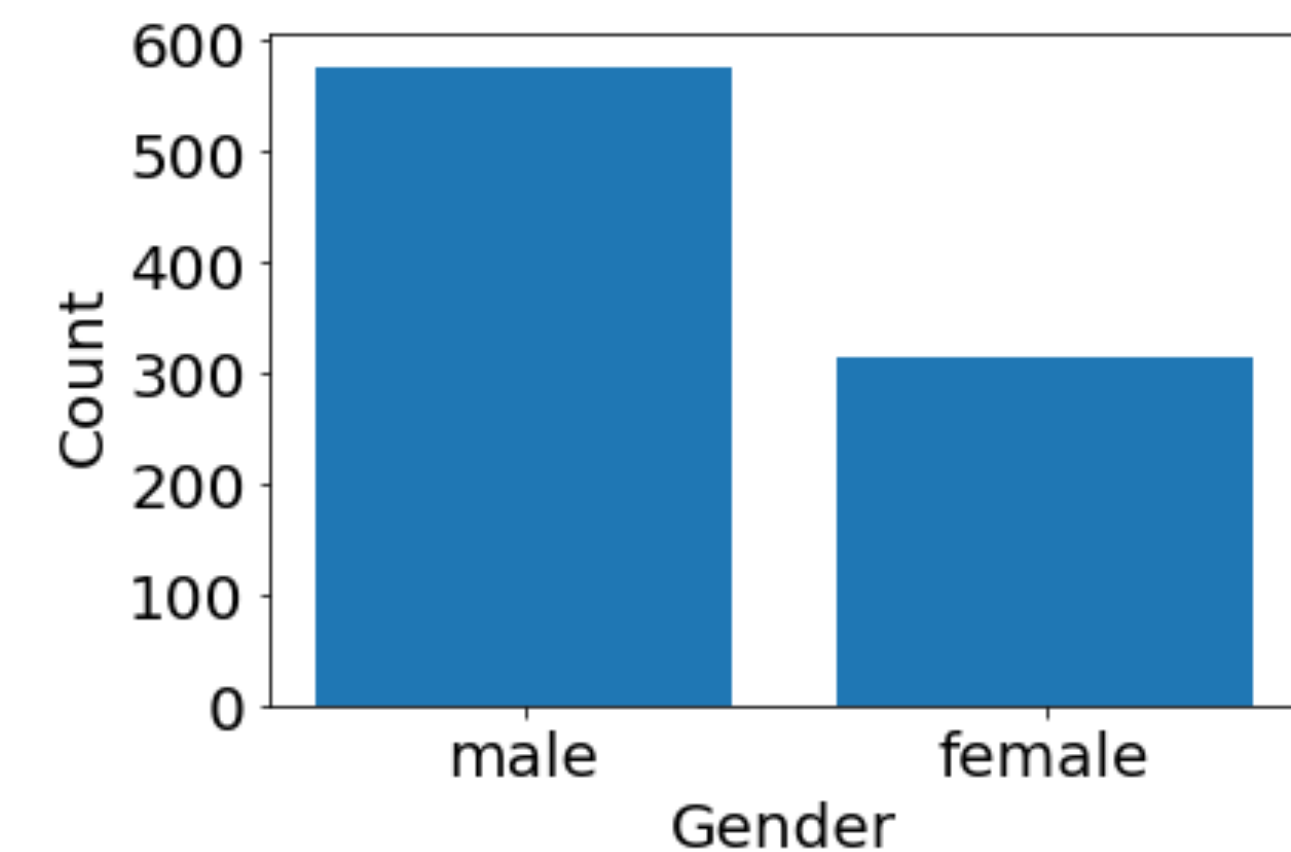
The number of bins determines the number of intervals the data will be divided into for the histogram.

Pie Chart

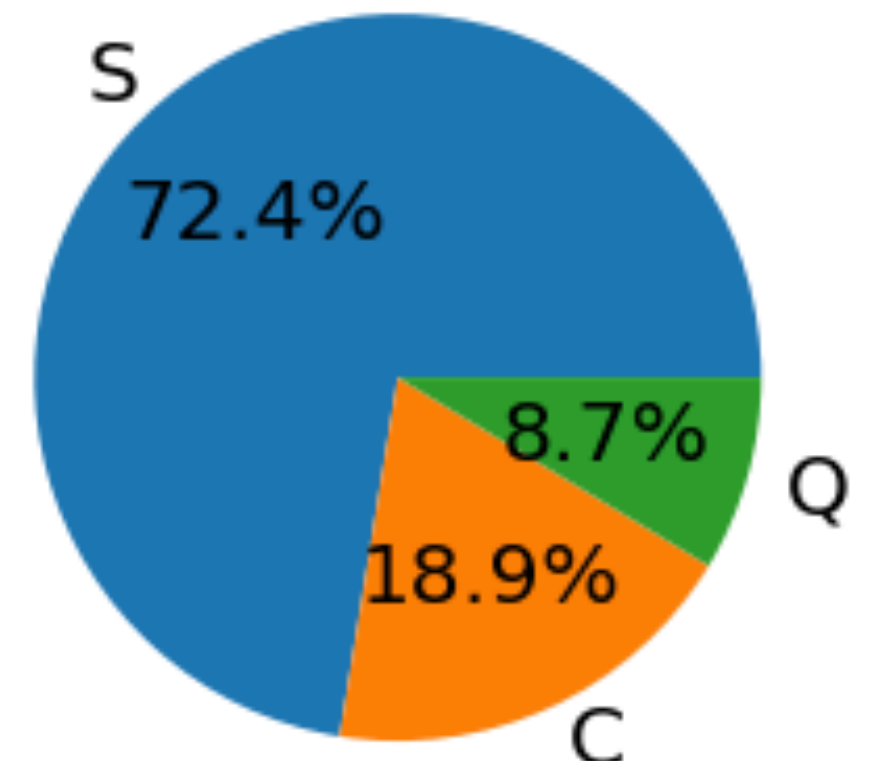
- Pie charts are circular representations used to show the proportion or percentage of different categories within a whole.
- Each slice represents a category, and the size of the slice corresponds to its proportion of the whole.
- **Key differences between Bar charts and Pie charts**

1. **Data Representation:** Pie charts represent proportions or percentages within a whole, while bar charts represent values or frequencies of different categories or groups.
2. **Comparison:** Pie charts focus on comparing the relative sizes or distributions of categories within a whole, while bar charts allow for direct comparison of values between different categories or groups.
3. **Visualization:** Pie charts provide a visual representation of proportions using slices, while bar charts use bars to represent values or frequencies.

Bar Chart



Pie Chart



Pie Chart

Example

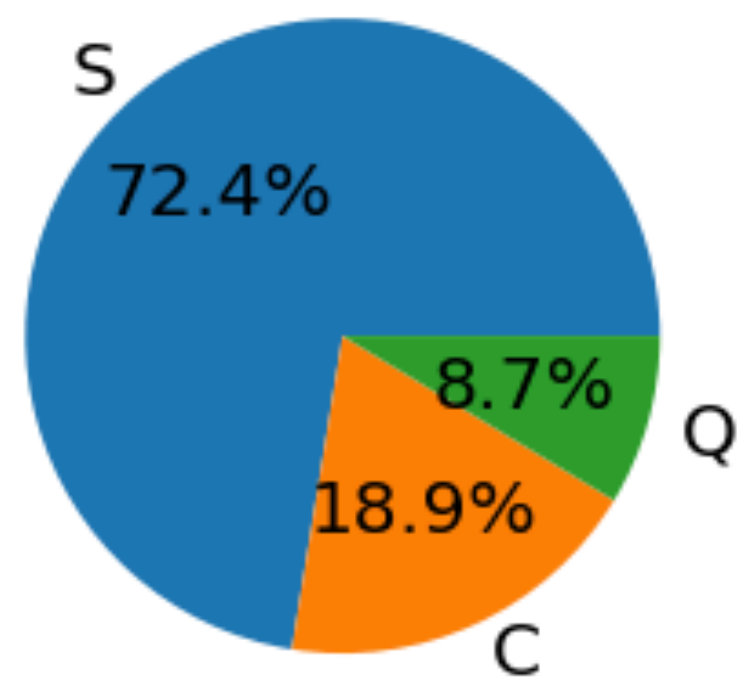
```
# Calculate the number of passengers from each embarked port
port_counts = df['Embarked'].value_counts()

# Create a pie chart
plt.pie(port_counts, labels=port_counts.index, autopct='%1.1f%%', textprops={'fontsize': 20})

# Set the title
plt.title('Embarked Port Distribution', fontsize=24)

# Display the chart
plt.show()
```

Embarked Port Distribution



Creating a pie chart to visualize the distribution of passengers across different embarked ports (e.g., Southampton, Cherbourg, Queenstown).

we first calculate the number of passengers from each embarked port using the `value_counts()` function on the 'Embarked' column of the Titanic DataFrame.

Then, we create a pie chart using `plt.pie()`, passing the `port_counts` as the data for the chart. The `labels` parameter is set to `port_counts`.

`index` to use the embarked ports as labels for each slice of the pie. The `autopct` parameter is set to `'%1.1f%%'` to display the percentage value on each slice.

Scatter Plot

Example

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

#Read the dataset.
#You can download the dataset from (https://www.kaggle.com/competitions/titanic/overview)
df = pd.read_csv('titanic/train.csv')

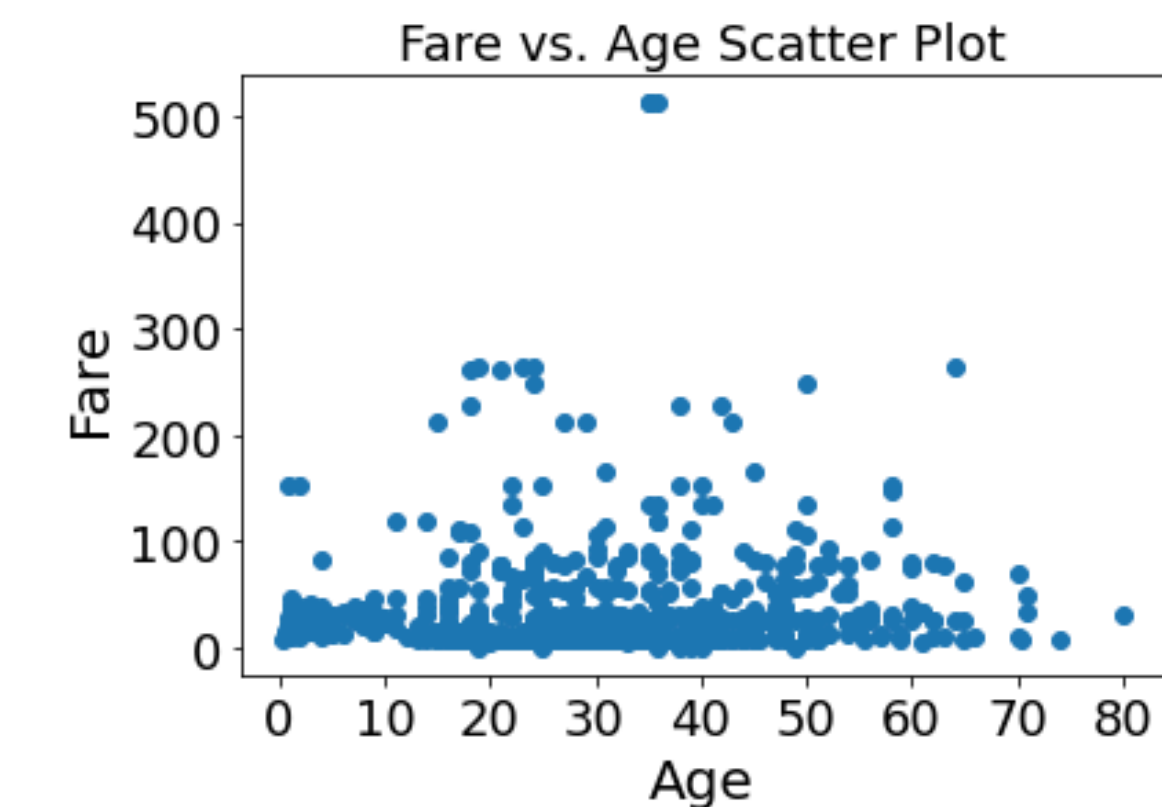
# Filter out missing values in Age and Fare columns
filtered_df = df.dropna(subset=['Age', 'Fare'])

# Create the scatter plot
plt.scatter(filtered_df['Age'], filtered_df['Fare'])

# Set the labels and title
plt.xlabel('Age', fontsize=20)
plt.ylabel('Fare', fontsize=20)
plt.title('Fare vs. Age Scatter Plot', fontsize=18)

# Increase the size of tick labels
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)

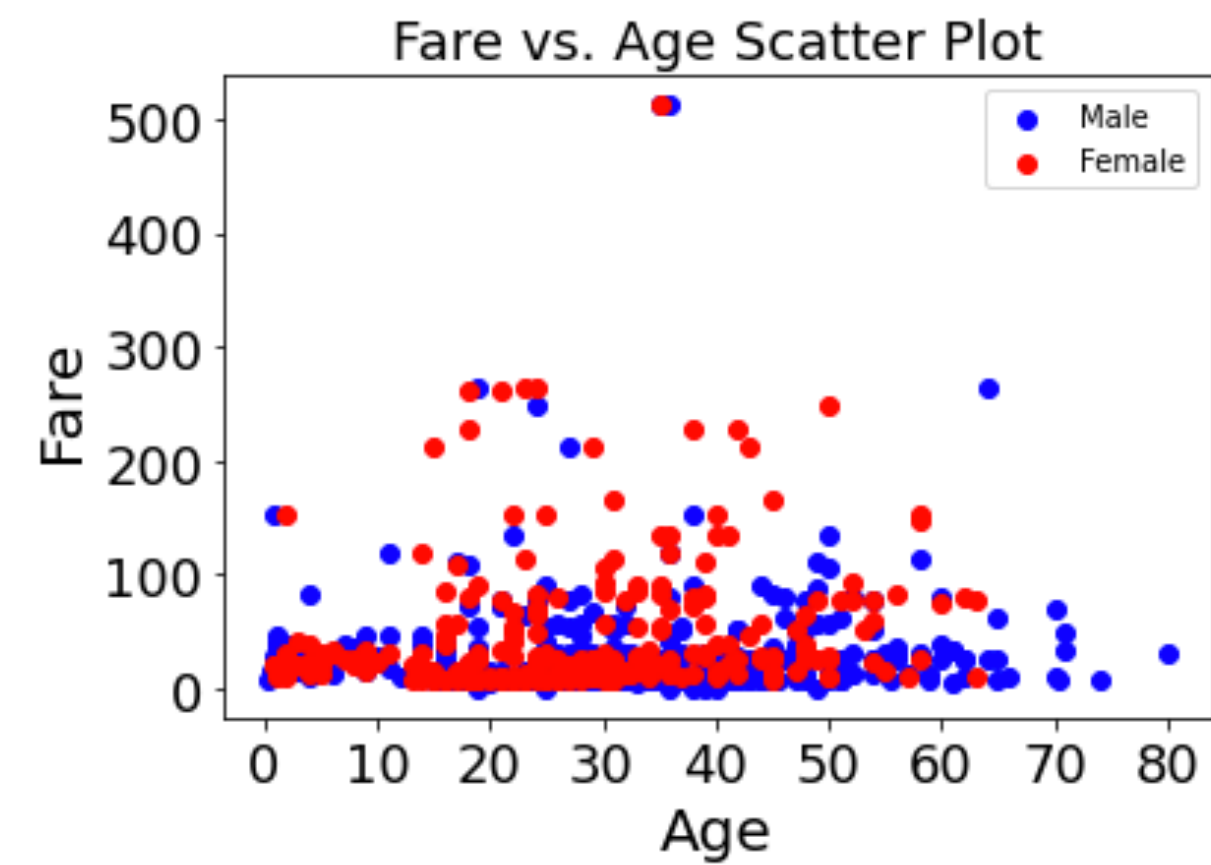
# Display the chart
plt.show()
```



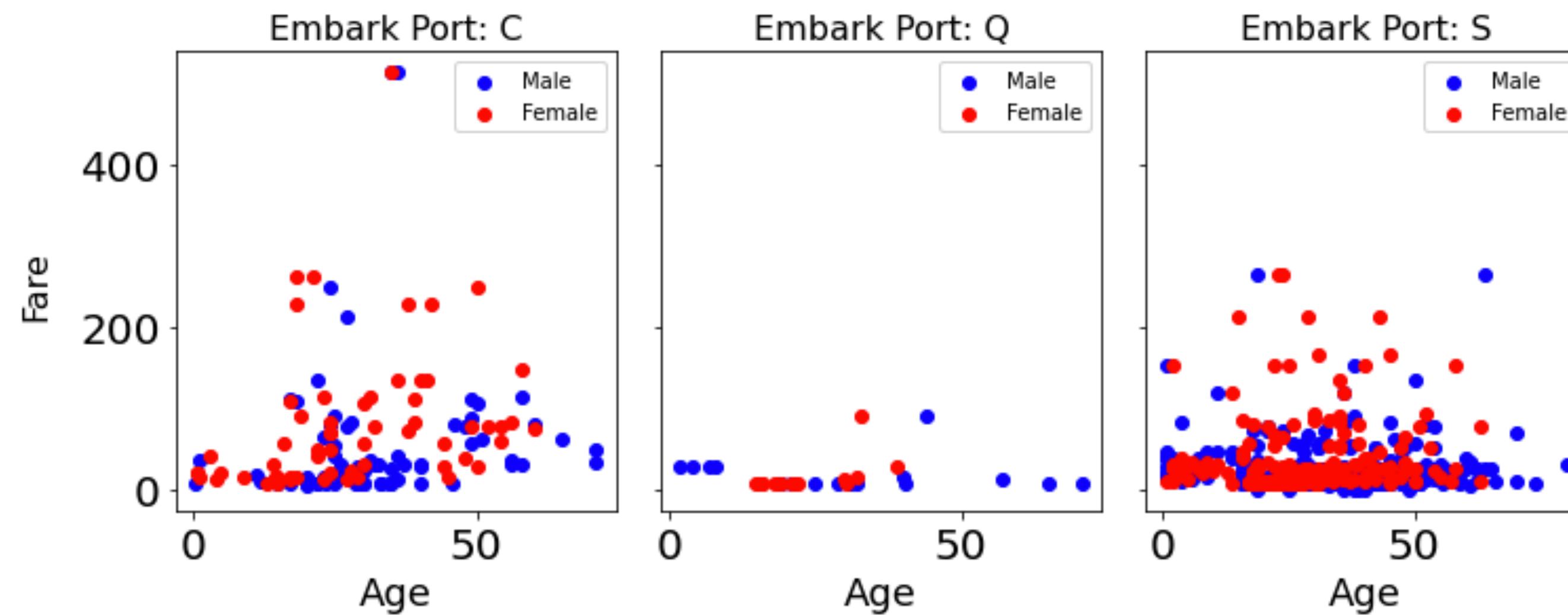
Scatter plots are particularly useful for:

1. Visualizing the relationship: Scatter plots provide a visual representation of the relationship between two variables. They allow you to observe patterns, trends, clusters, or outliers in the data.
2. Assessing correlation: Scatter plots can help assess the strength and direction of the correlation between two variables.
3. Detecting outliers: Scatter plots can help identify outliers, which are data points that deviate significantly from the overall pattern of the data.
4. Examining clusters or patterns: Scatter plots can reveal clusters or patterns in the data.

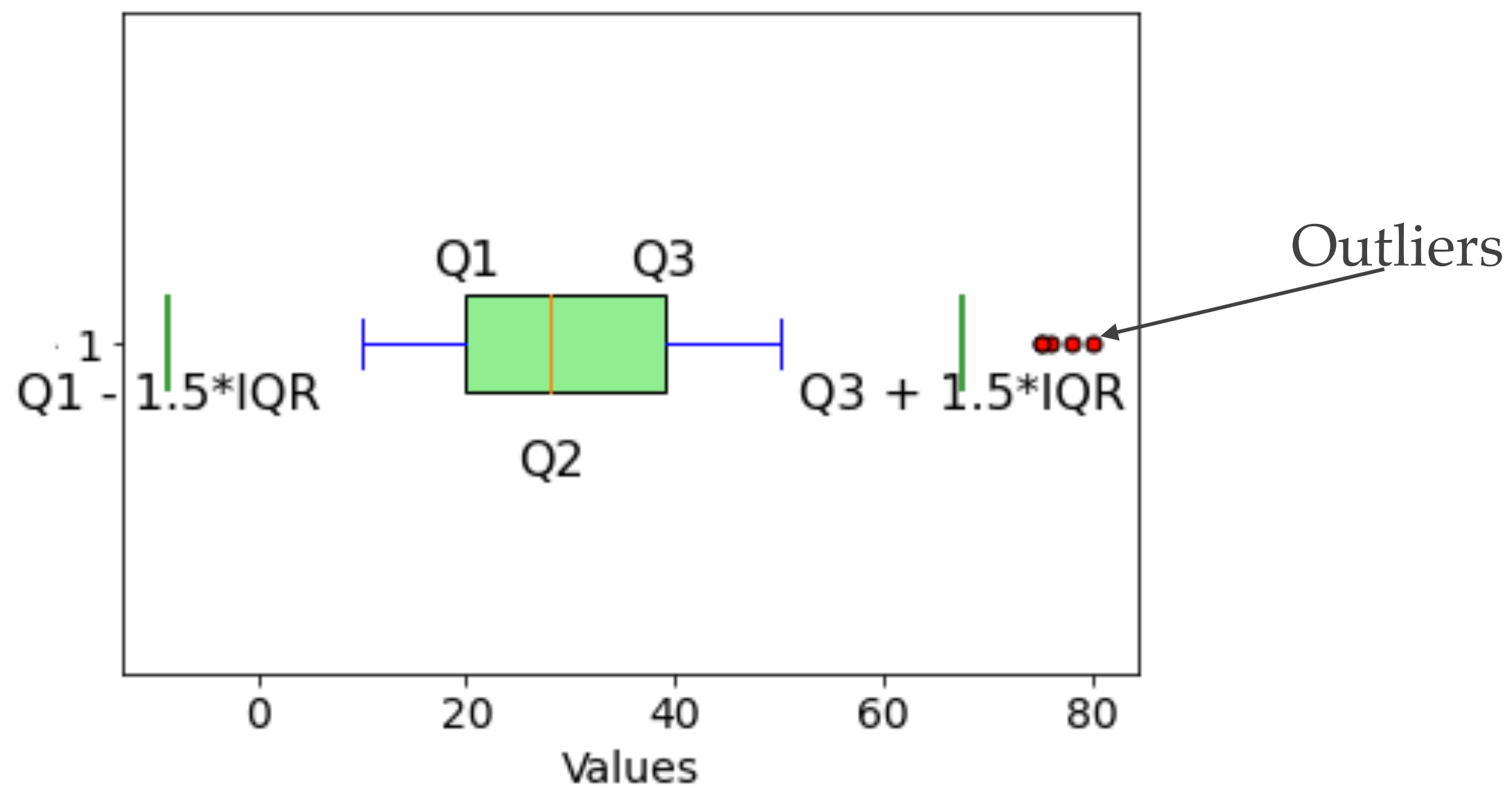
Scatter Plot



We can identify the patterns



Box Plot



Values below the lower bound ($Q1 - 1.5IQR$) are potential outliers. Similarly, the upper bound for potential outliers is calculated as $Q3 + 1.5IQR$.

It's worth noting that the choice of using 1.5 times the IQR as a threshold to identify outliers is a common convention, but it can vary depending on the specific dataset and the context of the analysis.

Different thresholds, such as 3 times the IQR, can be used based on the desired level of sensitivity to outliers.

Q1: First Quartile (25th percentile)

Q2: Median (50th percentile)

Q3: Third Quartile (75th percentile)

IQR: Interquartile range (Q3 - Q2)

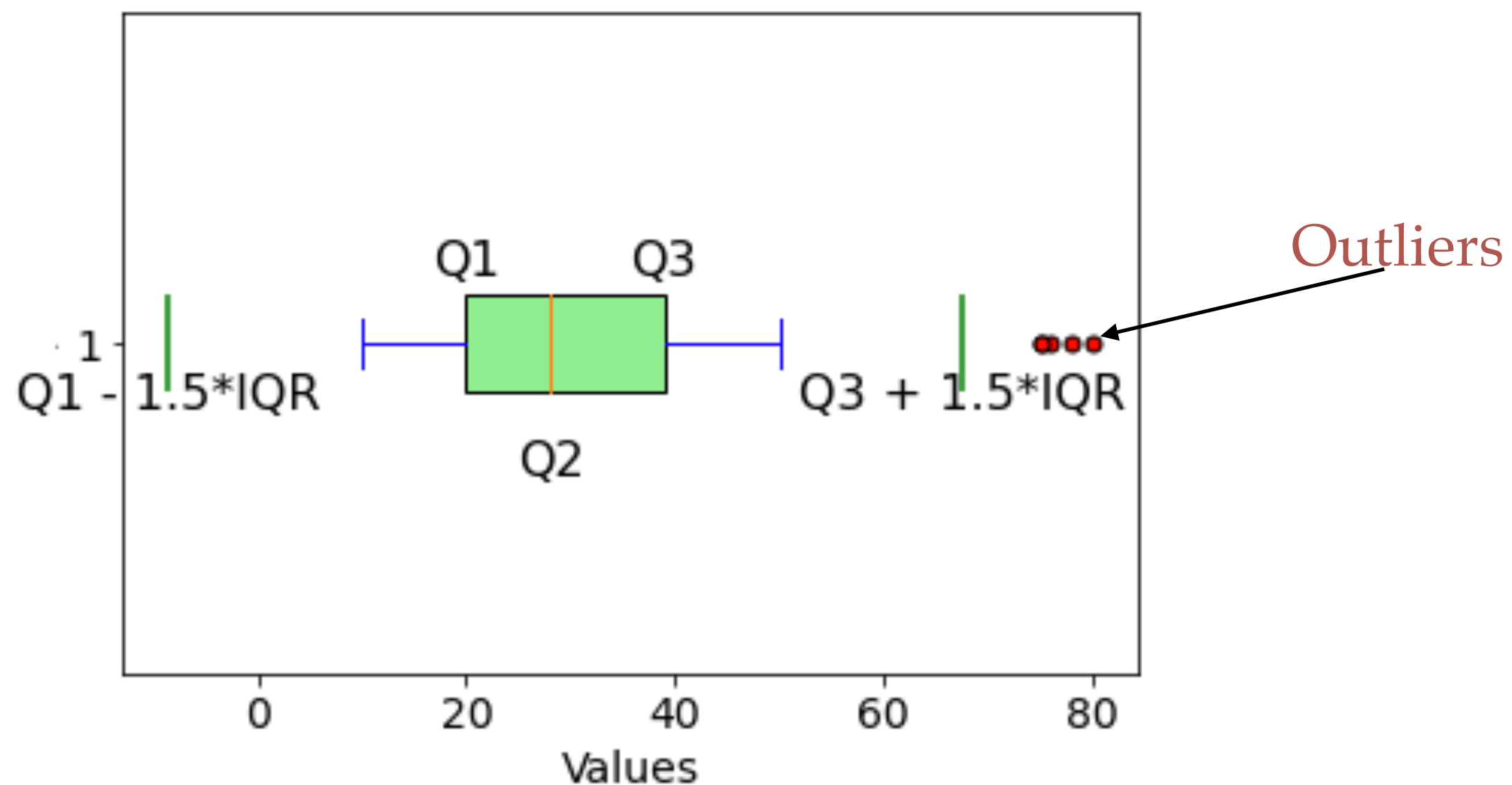
Whiskers: Shown in blue

Outliers: Shown in red

Lower bound: $Q1 - 1.5IQR$ (Shown as green line)

Upper bound: $Q3 + 1.5IQR$ (Shown as green line)

Box Plot



- **Minimum Value (Whisker):** The minimum value is determined as the smallest observed data point that is not considered an outlier.
- **Maximum Value (Whisker):** The maximum value is determined as the largest observed data point that is not considered an outlier.

Q1: First Quartile (25th percentile)

Q2: Median (50th percentile)

Q3: Third Quartile (75th percentile)

IQR: Interquartile range ($Q3 - Q1$)

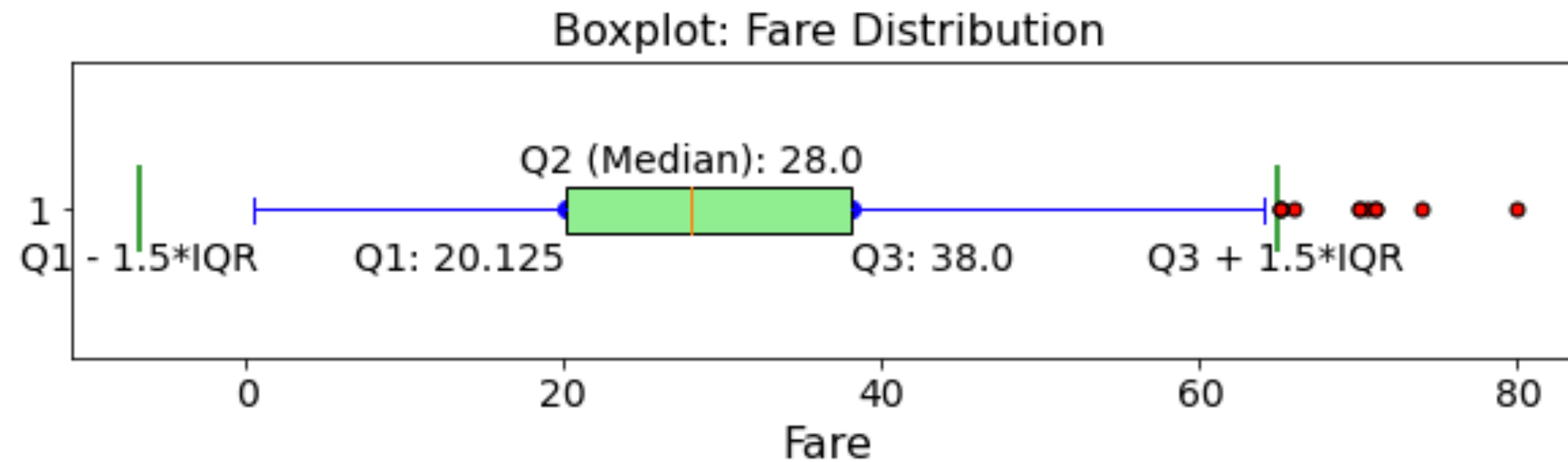
Whiskers: Shown in blue

Outliers: Shown in red

Lower bound: $Q1 - 1.5 \times IQR$ (Shown as green line)

Upper bound: $Q3 + 1.5 \times IQR$ (Shown as green line)

Box Plot



Above Box plot was produced for Fare column of the Titanic dataset.

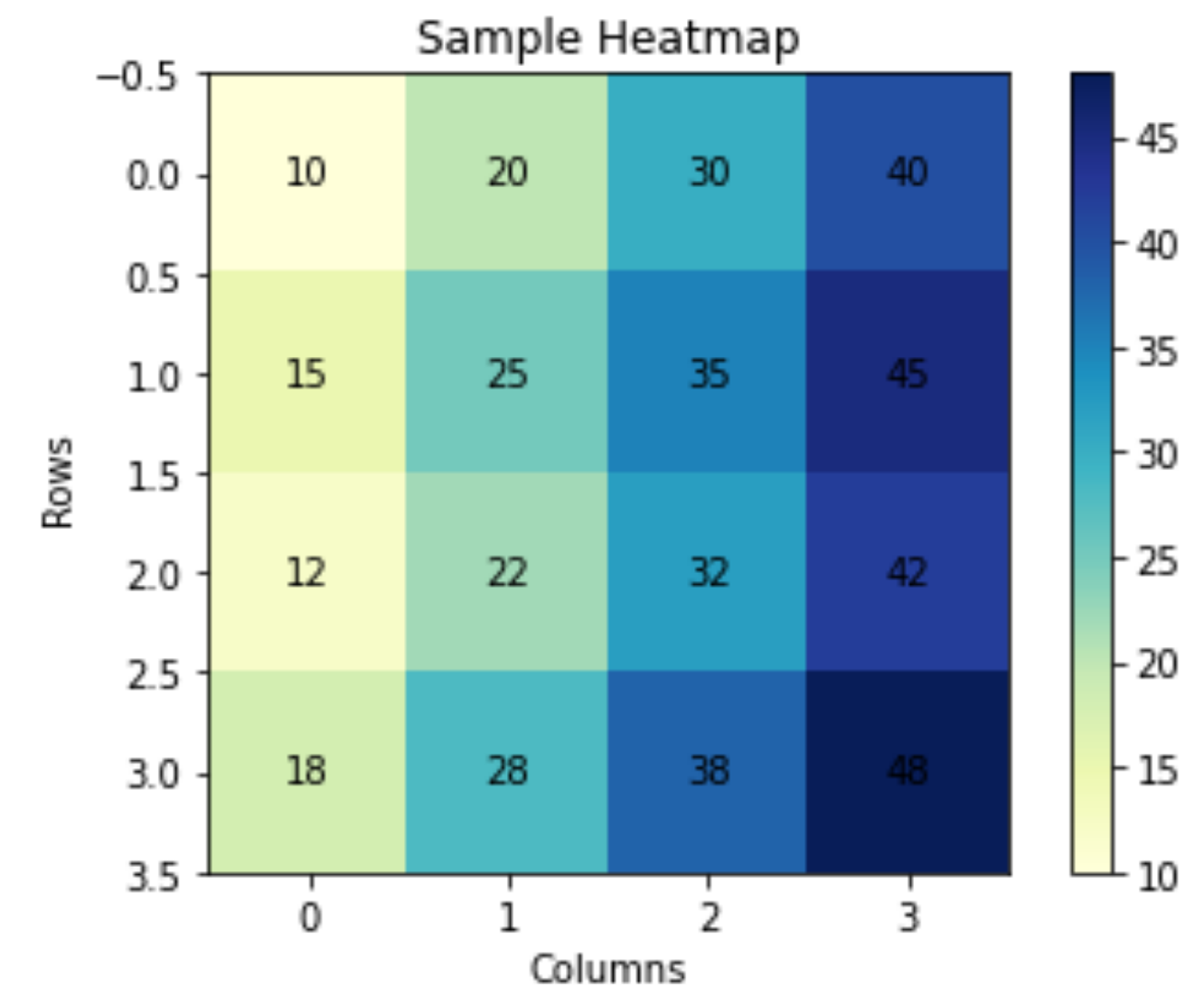
Median is at 28.0

Upper bound is at $Q3 + 1.5 \times IQR = 64.8125$

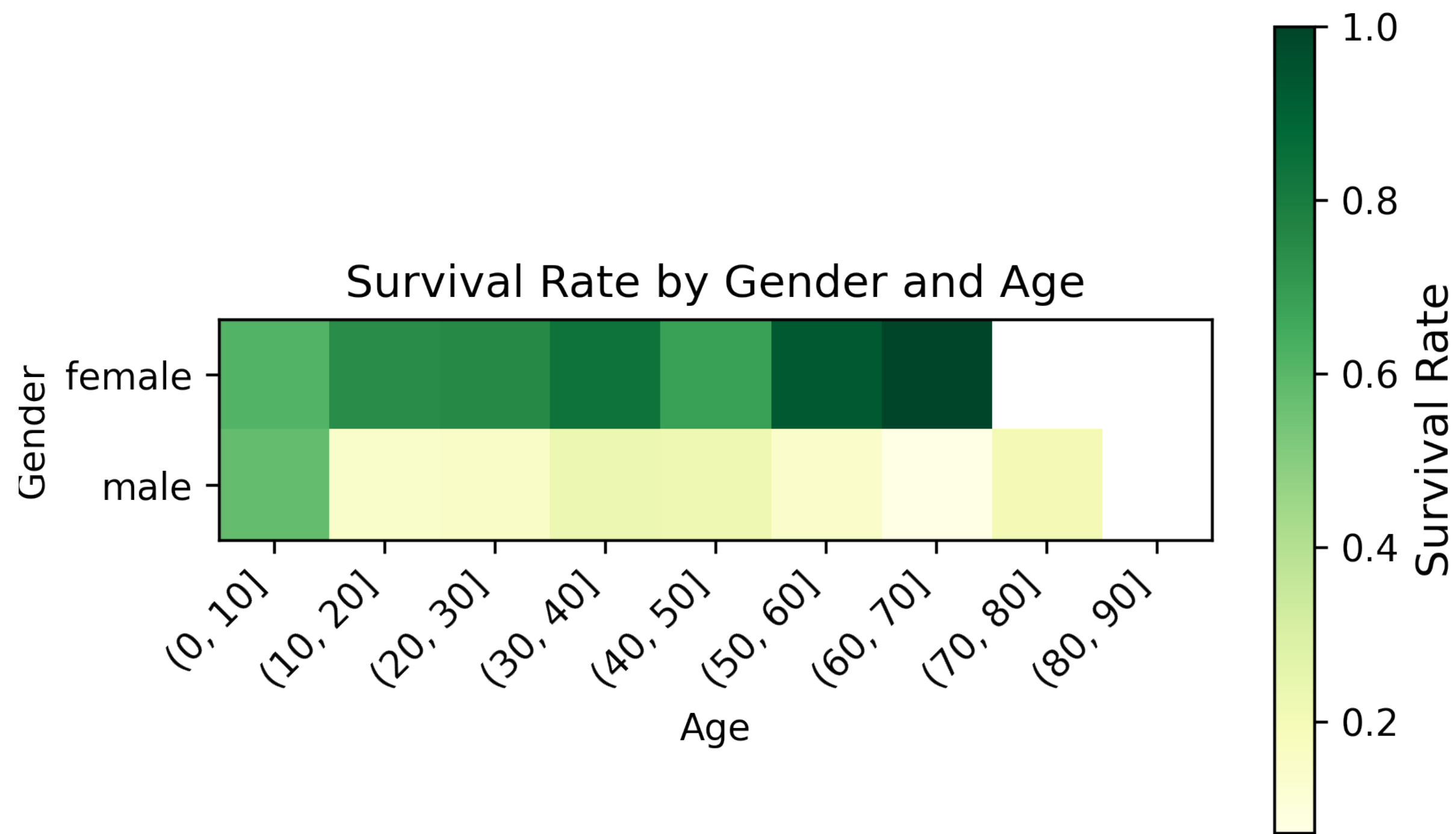
After 64.8125, outliers can be found (based on the conventional method).

Heat Maps

- Heatmaps use a color scale to represent the values of the dataset.
- Typically, a gradient of colors is used, where lighter or brighter colors represent lower values, and darker or cooler colors represent higher values.
- Heatmaps are often displayed as a grid of cells, where each cell corresponds to a specific value in the dataset.
- The rows and columns of the heatmap typically represent different variables or categories.
- Heatmaps can reveal patterns and relationships within the dataset.
- By examining the colors and intensities within the heatmap, one can infer the relative magnitude or intensity of the values.



Heat Maps



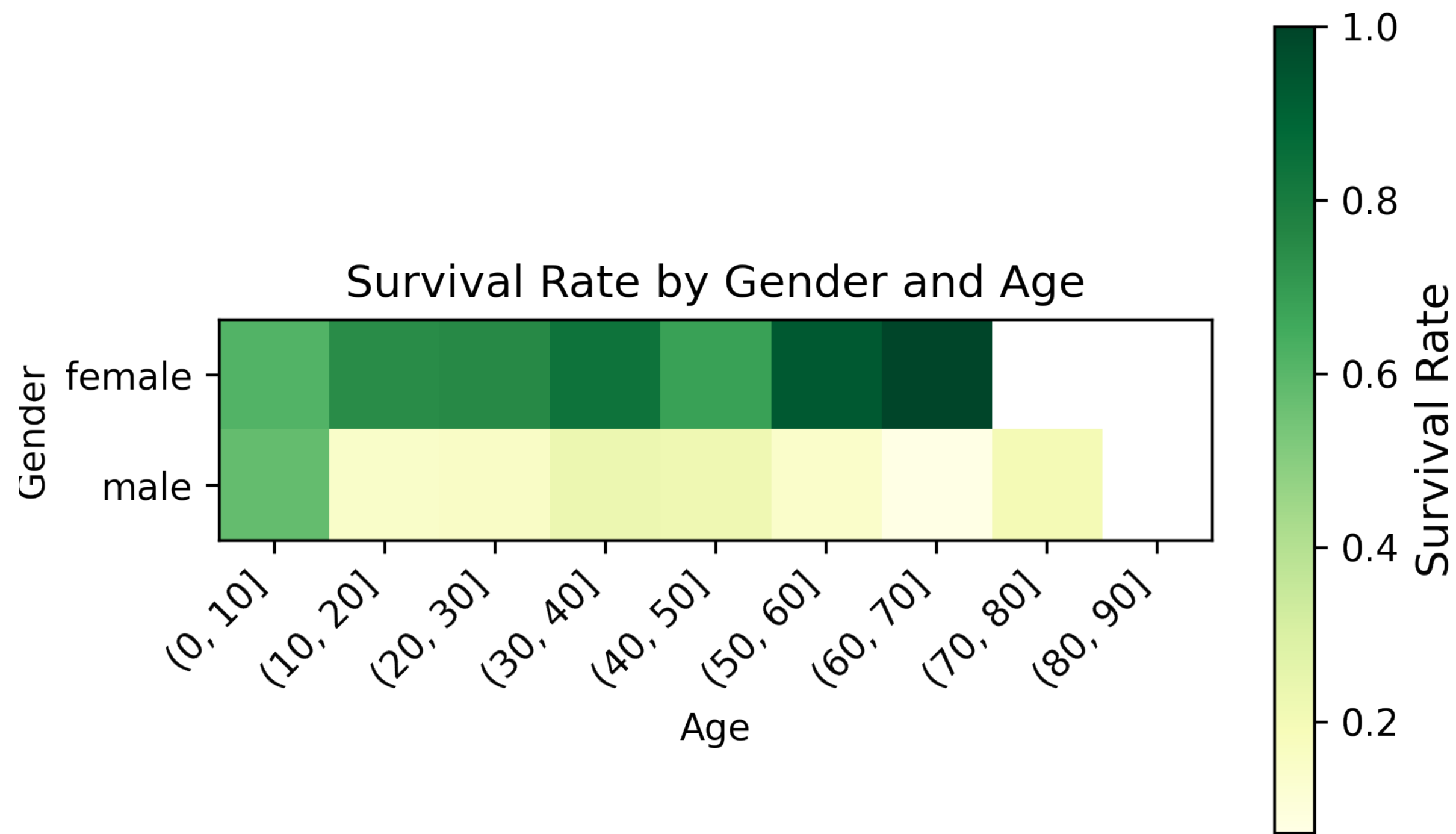
The heatmap represents the survival rate of passengers from the Titanic dataset, categorized by gender and age groups.

The x-axis of the heatmap represents the age groups, divided into bins of 10 years.

The y-axis represents the gender categories, with "Male" and "Female" labels.

The color bar on the right side of the heatmap indicates the range of survival rates, with lighter shades corresponding to higher survival rates and darker shades corresponding to lower survival rates.

Heat Maps



What we can see in the graph

- Survival rate of female is higher than that of male.
- The highest survival rate is shown by females at 60 to 70 age group.
- Among the males, 0-10 age group has highest survival rate.

Correlation Analysis

- Correlation analysis measures and quantifies the relationship between variables.
- It identifies the degree to which changes in one variable are associated with changes in another variable.
- Correlation analysis helps in identifying patterns, dependencies, and potential associations.
- It assists in variable selection by assessing the correlations between variables and the target variable of interest.
- Correlation analysis facilitates data exploration and visualization through scatterplots and correlation matrices.
- It enhances decision-making by providing insights into the dependencies between variables.

Pearson's Correlation

Pearson's correlation coefficient is a measure of the linear relationship between two continuous variables.

It is denoted by the symbol " r " and can range from -1 to 1.

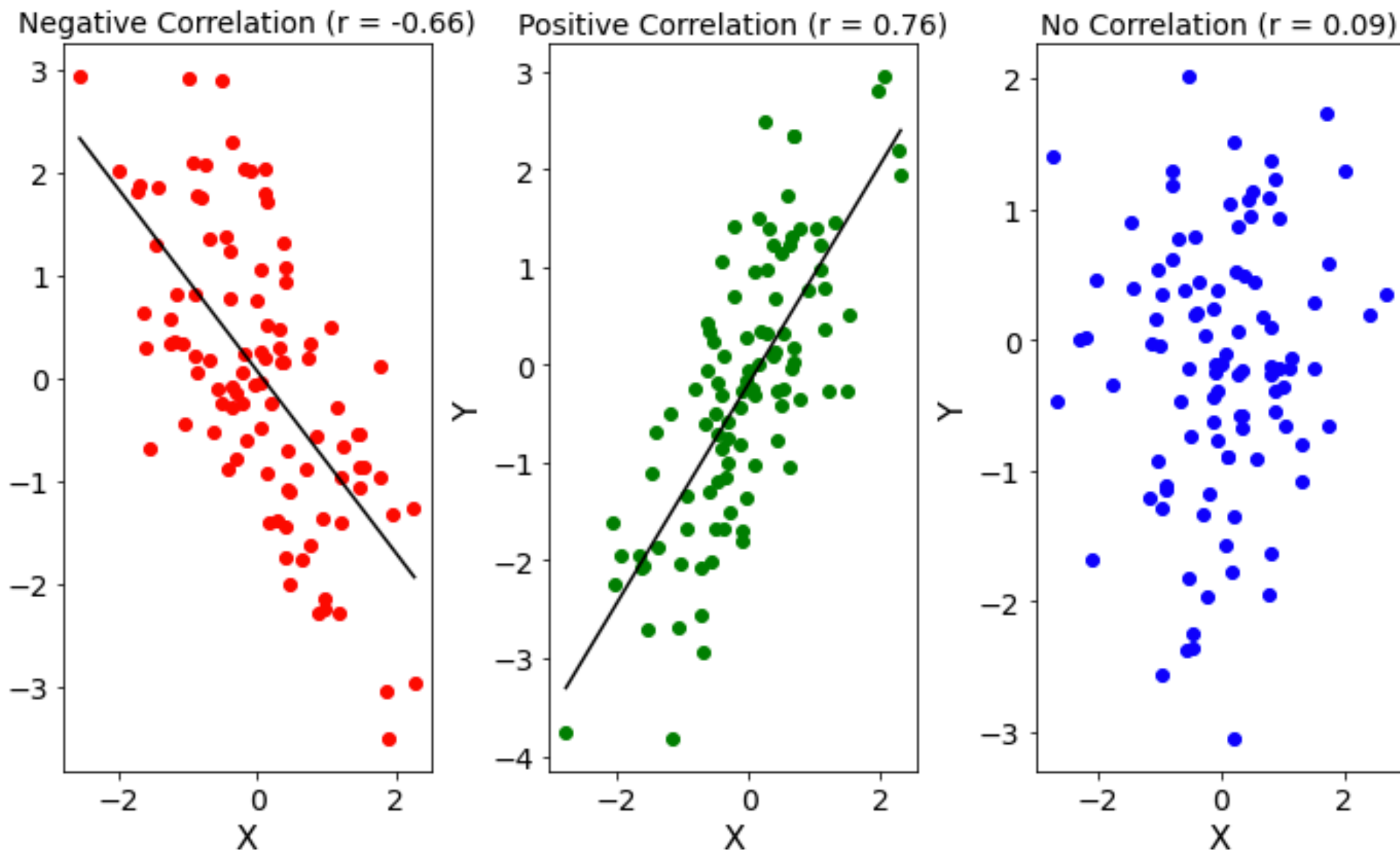
The formula to calculate Pearson's correlation coefficient is:

$$r = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2 \sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

where:

- X_i and Y_i are individual data points of the two variables.
- \bar{X} and \bar{Y} are the means of the two variables.
- \sum denotes the sum over all data points.

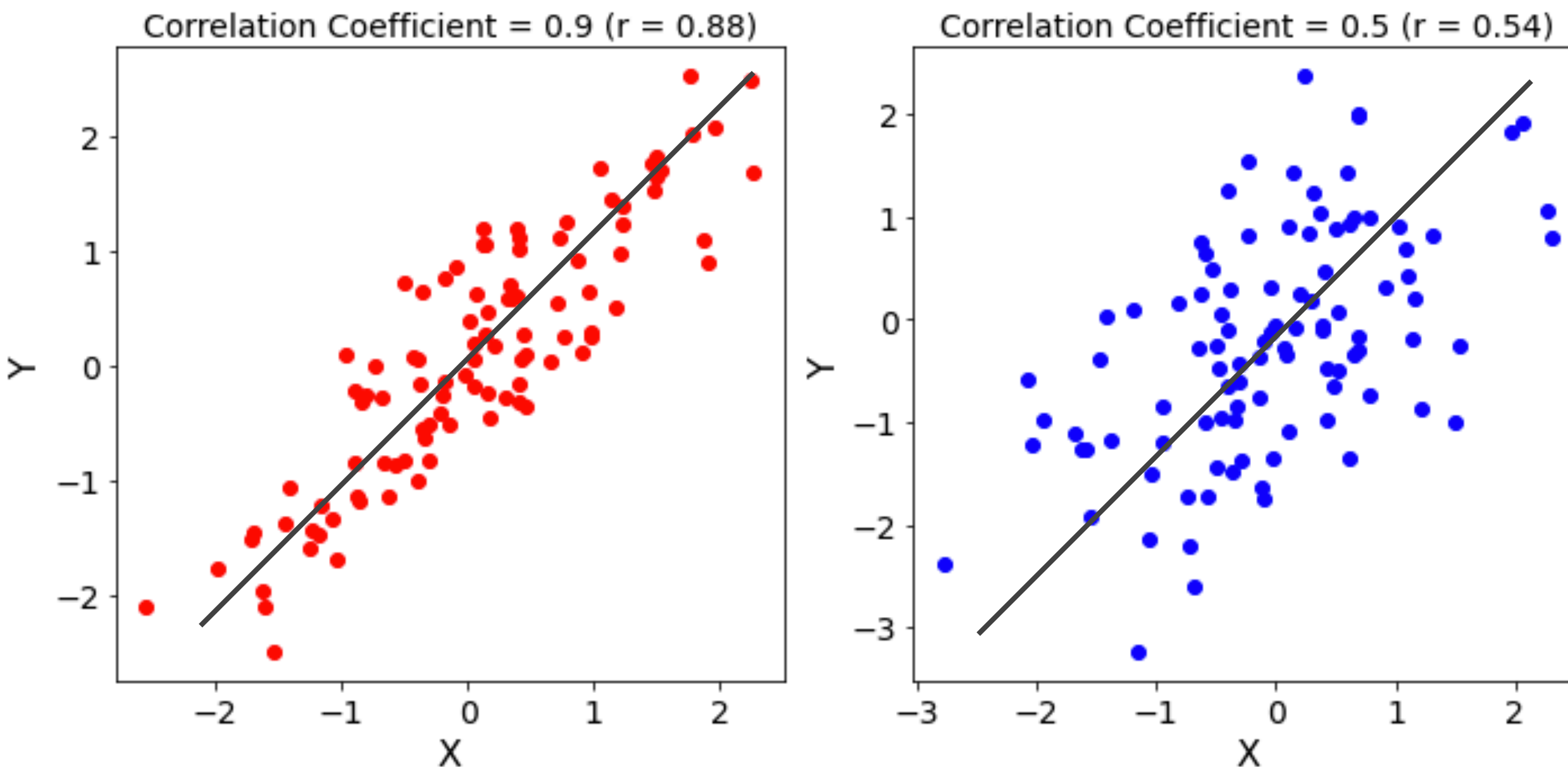
Pearson's Correlation



Interpretation of the correlation coefficient:

- If $r = 1$, it indicates a perfect positive linear relationship. As one variable increases, the other variable increases proportionally.
- If $r = -1$, it indicates a perfect negative linear relationship. As one variable increases, the other variable decreases proportionally.
- If $r = 0$, it indicates no linear relationship between the variables. There is no systematic linear pattern in their relationship.
- Values between -1 and 1 represent varying degrees of correlation strength. The closer the value is to -1 or 1, the stronger the correlation.
- The sign of the correlation coefficient (+ or -) indicates the direction of the relationship (positive or negative).

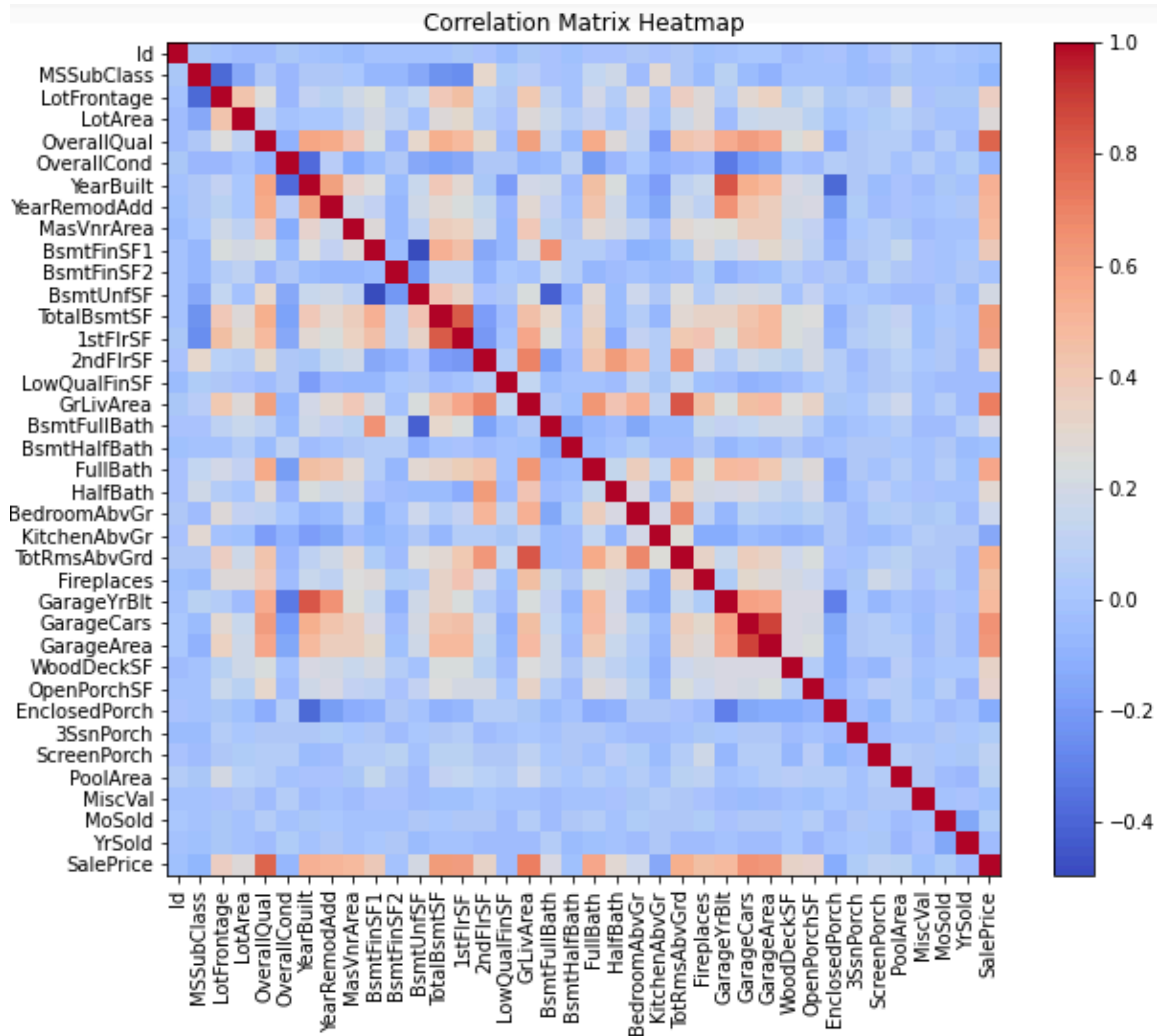
Pearson's Correlation



Interpreting the magnitude of the correlation coefficient:

- A correlation coefficient close to 1 or -1 indicates a strong relationship between the variables.
- A correlation coefficient close to 0 indicates a weak or no relationship between the variables.
- The absolute value of the correlation coefficient (ignoring the sign) represents the strength of the relationship. The closer it is to 1, the stronger the relationship.
- However, it's important to note that correlation does not imply causation. A strong correlation does not necessarily mean that one variable causes the changes in the other variable.

Correlation Matrix



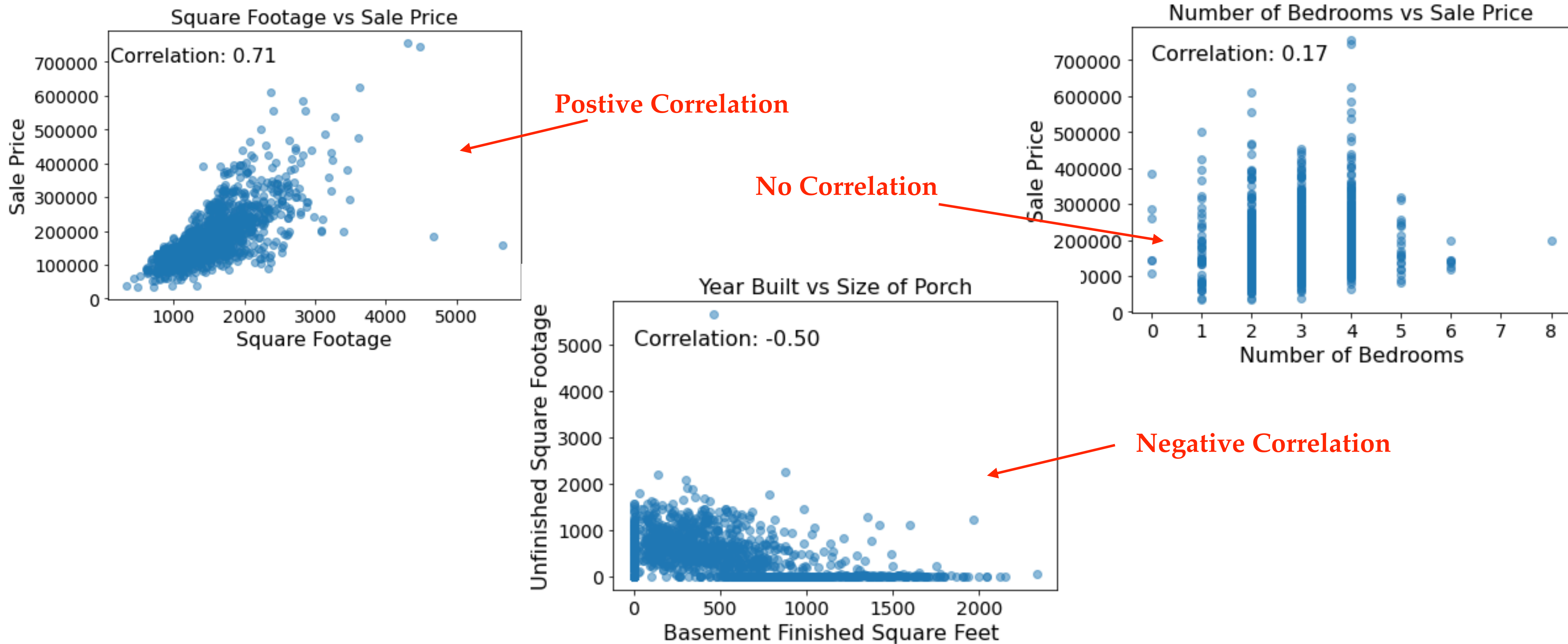
```
df = pd.read_csv('house-prices/train.csv')

# Calculate the correlation matrix
correlation_matrix = df.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
plt.imshow(correlation_matrix, cmap='coolwarm')
plt.colorbar()
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns, rotation=90)
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

- **Heat Maps** can be effectively used for visualizing the correlation.
- The `corr` function of pandas was used to calculate the correlation.
- The default method of `corr` function is Pearson.
- This code snippet uses the `imshow` function from Matplotlib to display the correlation matrix as a heatmap.
- The `cmap='coolwarm'` parameter sets the color map for the heatmap.

Visualizing Correlation



Data Normalization

- Normalizing the data brings all the features onto a common scale.
- Many machine learning algorithms are sensitive to the scale of the input features.
- If the features have different scales, those with larger values may dominate the learning process and overshadow the contributions of other features.
- Normalization ensures that all features contribute equally to the learning process and prevents any feature from dominating based solely on its scale.
- Normalizing the data can improve the performance and convergence of machine learning models.
- Normalized data provides a standardized framework for understanding the relative magnitudes and relationships between features.

Data Normalization

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

The numerical columns like SibSp, Parch have smaller values compared to the values in Age and Fare columns.

If we do not normalize the 'Fare' and Age columns, it may have a larger impact on the learning process compared to other features due to its higher magnitude.

Data Normalization

Types of Data Normalization Techniques we consider in this session:

- **Min-Max Scaling (Normalization):**
 - Scaling data to a specific range (e.g., $[0, 1]$) by subtracting the minimum value and dividing by the range.
- **Z-Score Standardization (Standardization):**
 - Standardizing data to have zero mean and unit variance by subtracting the mean and dividing by the standard deviation.
- **Robust Scaling:**
 - Scaling data using median and interquartile range (IQR) to handle outliers.

Min-Max Scaling

Min-Max scaling, also known as normalization, is a popular data normalization technique used in machine learning.

It scales the features of a dataset to a specific range, typically between 0 and 1.

The formula for Min-Max scaling is as follows:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

- X is the original value of a feature.
- X_{min} is the minimum value of that feature in the dataset.
- X_{max} is the maximum value of that feature in the dataset.
- X_{norm} is the normalized value of the feature.

This formula ensures that the minimum value of the feature becomes 0, and the maximum value becomes 1, while maintaining the relative distribution of the data.

Min-Max Scaling

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

#Read the dataset.
#You can download the dataset from (https://www.kaggle.com/competitions/titanic/overview)
df = pd.read_csv('titanic/train.csv')

# Identify numerical columns
numerical_columns = df.select_dtypes(include='number').columns

# Create a new DataFrame with only the numerical columns
df_numerical = df[numerical_columns]

# Create an instance of the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the numerical columns
df_normalized = pd.DataFrame(scaler.fit_transform(df_numerical), columns=numerical_columns)

# Print the normalized DataFrame
print(df_normalized.head())
```

We use **MinMaxScaler** in Scikit Learn Python package to normalize the numerical data.

Here, we show only the numerical data for convenience.

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	0.000000	0.0	1.0	0.271174	0.125	0.0	0.014151
1	0.001124	1.0	0.0	0.472229	0.125	0.0	0.139136
2	0.002247	1.0	1.0	0.321438	0.000	0.0	0.015469
3	0.003371	1.0	0.0	0.434531	0.125	0.0	0.103644
4	0.004494	0.0	1.0	0.434531	0.000	0.0	0.015713

Z-Score Scandalization (Scandalization)

Z-score standardization, also known as standardization or feature scaling, is a data normalization technique used in machine learning.

It transforms the features of a dataset to have zero mean and unit variance.

The formula for Z-score standardization is as follows:

$$X_{\text{std}} = \frac{X - \mu}{\sigma}$$

- X is the original value of a feature.
- μ is the mean of that feature in the dataset.
- σ is the standard deviation of that feature in the dataset.
- X_{std} is the normalized value of the feature.

This transformation centers the distribution of the feature around zero and scales it to have a standard deviation of one.

Z-Score Scandalization (Scandalization)

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

#Read the dataset.
#You can download the dataset from (https://www.kaggle.com/competitions/titanic/overview)
df = pd.read_csv('titanic/train.csv')

# Identify numerical columns
numerical_columns = df.select_dtypes(include='number').columns

# Create a new DataFrame with only the numerical columns
df_numerical = df[numerical_columns]

# Create an instance of the StandardScaler
scaler = StandardScaler()

# Fit and transform the numerical columns
df_normalized = pd.DataFrame(scaler.fit_transform(df_numerical), columns=numerical_columns)

# Print the normalized DataFrame
print(df_normalized.head())
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	-1.730108	-0.789272	0.827377	-0.530377	0.432793	-0.473674	-0.502445
1	-1.726220	1.266990	-1.566107	0.571831	0.432793	-0.473674	0.786845
2	-1.722332	1.266990	0.827377	-0.254825	-0.474545	-0.473674	-0.488854
3	-1.718444	1.266990	-1.566107	0.365167	0.432793	-0.473674	0.420730
4	-1.714556	-0.789272	0.827377	0.365167	-0.474545	-0.473674	-0.486337

We use **StandardScaler** in Scikit Learn Python package to scandalize the numerical data.

Here, we show only the numerical data for convenience.

Robust Scaling

Robust scaling, also known as robust standardization, is a method used to scale numerical data that is resistant to the presence of outliers and handles skewed distributions.

The formula for Robust Scaling is as follows:

$$X_{\text{scaled}} = \frac{(X - X_{\text{median}})}{\text{IQR}}$$

- X is the original value of a feature.
- X_{median} is the median of that feature in the dataset.
- IQR is the interquartile range, calculated as the difference between the 75th and 25th percentiles of the feature / column
- X_{scaled} is the scaled value of the feature.

By subtracting the median and dividing by the interquartile range (IQR), robust scaling normalizes the data to have a median of 0 and a spread based on the IQR.

Robust Scaling

```
import pandas as pd
from sklearn.preprocessing import RobustScaler

#Read the dataset.
#You can download the dataset from (https://www.kaggle.com/competitions/titanic/overview)
df = pd.read_csv('titanic/train.csv')

# Identify numerical columns
numerical_columns = df.select_dtypes(include='number').columns

# Create a new DataFrame with only the numerical columns
df_numerical = df[numerical_columns]

# Create an instance of the StandardScaler
scaler = RobustScaler()

# Fit and transform the numerical columns
df_normalized = pd.DataFrame(scaler.fit_transform(df_numerical), columns=numerical_columns)

# Print the normalized DataFrame
print(df_normalized.head())
```

We use **RobustScaler** in Scikit Learn Python package to scandalize the numerical data.

Here, we show only the numerical data for convenience.

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	-1.000000	0.0	0.0	-0.335664	1.0	0.0	-0.312011
1	-0.997753	1.0	-2.0	0.559441	1.0	0.0	2.461242
2	-0.995506	1.0	0.0	-0.111888	0.0	0.0	-0.282777
3	-0.993258	1.0	-2.0	0.391608	1.0	0.0	1.673732
4	-0.991011	0.0	0.0	0.391608	0.0	0.0	-0.277363