

## Introduction to pandas —

### 1: What is pandas?

Pandas is an open-source library in Python used for data cleaning, manipulation and analysis. It provides the data structures and functions needed to work efficiently with tabular or structured data (such as CSV files, Excel sheets, SQL databases, etc.).

#### 2: Pandas installation

```
pip install pandas
```

### 3: Using Python modules:

```
import pandas as pd # To import pandas
```

### Example: Titanic dataset

We will use the Titanic dataset from pandas as an example. It contains information about passengers aboard the Titanic, including their survival status, age, gender, ticket class, and fare.

survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	female	38.0	1	0	71.2833	C	First	Woman	False	C	Cherbourg	yes	False
1	3	female	26.0	0	0	7.9250	S	Third	Woman	False	NaN	Southampton	yes	True
1	1	female	35.0	1	0	53.1000	S	First	Woman	False	C	Southampton	yes	False
0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	yes	True

### Pandas Data Structures — •

#### 1: Series:

- A series is a one-dimensional labeled array capable of holding data of any type (integer, float, string, etc.).
- Creating series:
  - From lists, arrays, dictionaries pd.Series (data)
- Accessing elements: By position, indexes/labels series.loc[label]
  - By position: series.iloc[index]
  - By indexes/labels: series.loc[label]

```
# Creating a series from a list
age = [22.0, 38.0, 26.0, 35.0, 35.0]
s = pd.Series(age)

# Accessing elements by position
print(s.iloc[0])  # Output: 22.0
print(s.iloc[2])  # Output: 26.0

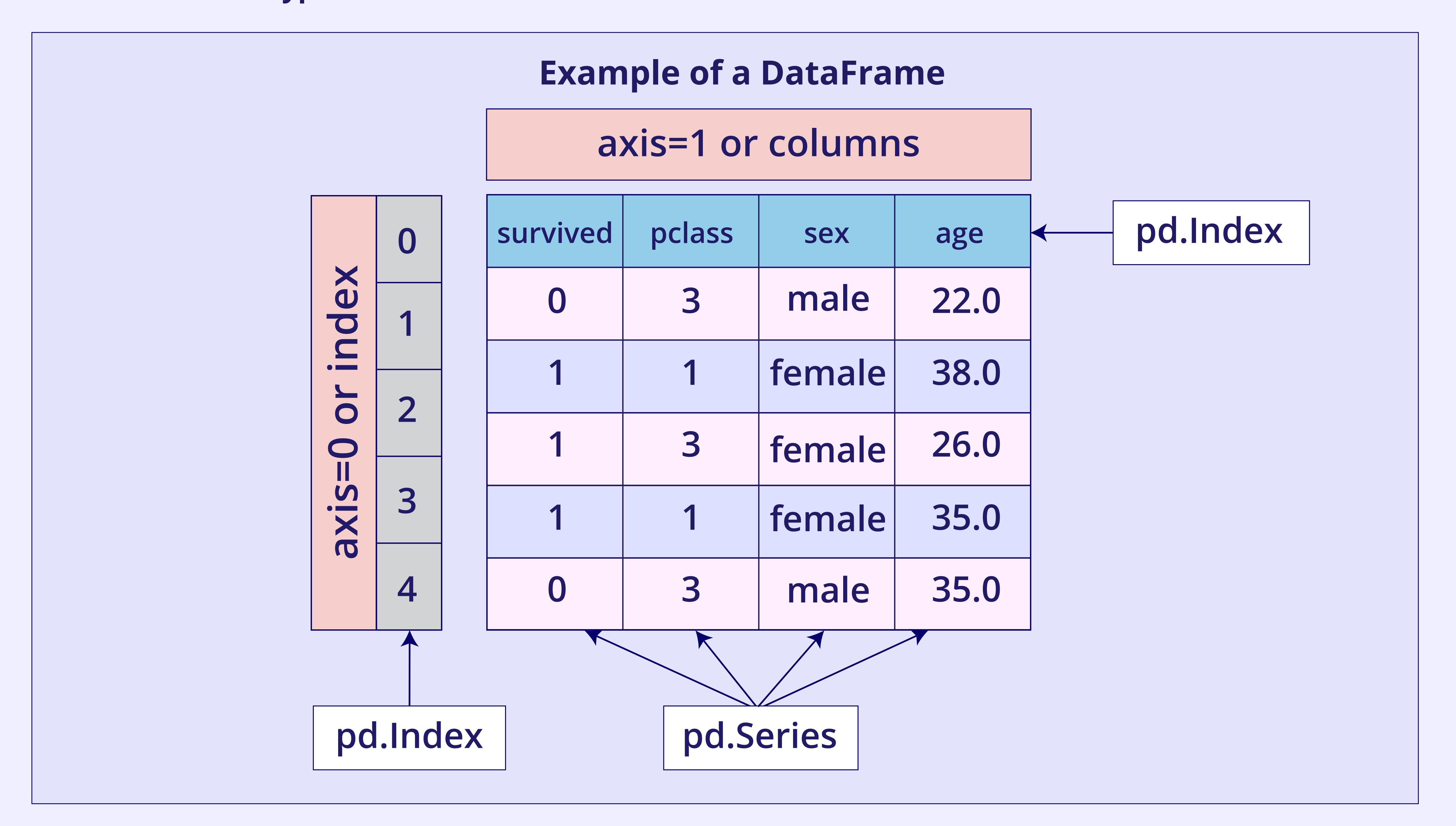
# Accessing elements by index/labels
print(s.loc[0])  # Output: 22.0
print(s.loc[2])  # Output: 26.0

# Creating a series from a dictionary
data_dict = {'survived': 0, 'pclass': 3, 'sex': 'male', 'age': 22.0, 'sibsp': 1, 'parch':0}
series_from_dict = pd.Series(data_dict)
```



#### 2: DataFrame

• A DataFrame is a two-dimensional labeled data structure with columns of potentially different types.



- Creating DataFrames:
  - From a dictionary: pd.DataFrame(data)
  - CSV file: pd.read\_csv('Titanic.csv')
  - A list of lists: pd.DataFrame(data, columns=['survived', 'pclass', 'sex' 'age',...])
- Accessing elements:
  - By position: df.iloc[row\_index, col\_index]
  - By indexes/labels: df.loc[row label, col label]

```
Example
# Creating a DataFrame from a dictionary
data = {
    'survived': [0, 1, 1, 1, 0],
    'pclass': [3, 1, 3, 1, 3],
    'sex': ['male', 'female', 'female', 'female', 'male'],
    'age': [22.0, 38.0, 26.0, 35.0, 35.0],
    'sibsp': [1, 1, 0, 1, 0],
    'parch': [0, 0, 0, 0, 0]
df = pd.DataFrame(data)
# Accessing elements by position
print("Accessing by position:")
print(df.iloc[0, 1]) # Output: 25 (Age of first row)
print(df.iloc[2, 2]) # Output: Chicago (City of third row)
print()
# Accessing elements by index/labels
print("Accessing by index/labels:")
print(df.loc[0, 'Name']) # Output: Alice (Name of first row)
print(df.loc[3, 'City']) # Output: Houston (City of fourth row)
```



# Importing and Storing Data —

We can import and export data using efficient pandas functions.

- Reading and Writing Data
  - To a CSV file → CSV [Reading and writing data]

```
df = pd.read_csv('Titanic.csv')
df.to_csv('Titanic.csv')
```

To a SQL database → SQL [Reading and Writing Data]

```
pd.read_sql('SELECT * FROM table', connection)
df.to_sql('table', connection)
```

Parquet [Reading and Writing Data]

```
df = pd.read_parquet('Titanic.parquet')
df.to_parquet('Titanic.parquet')
```

• HDF5 [Reading and Writing Data]

```
df = pd.read_hdf('Titanic.h5')
df.to_hdf('Titanic.h5', key='df', mode='w')
```

(key to specify the key of the HDF5 group to write to, and mode to specify the mode of writing ('w' for write, 'a' for append))

To an Excel file → Excel [Reading and Writing Data]

```
pd.read_excel('Titanic.xlsx', sheet_name='Sheet1')
df.to_excel('Titanic.xlsx', sheet_name='Sheet1')
```

• To a JSON file  $\longrightarrow$  JSON [Reading and Writing Data]

```
pd.read_json('Titanic.json')
df.to_json('Titanic.json')
```

## Viewing and Other Operations —

• We can view the head and tail of the DataFrame by calling the following functions:

```
print(df.head()) # View the first few rows
print(df.tail()) # View the last few rows
```

 We can obtain statistical information, the length and shape of the DataFrame, information about columns, and a summary of the DataFrame by calling the following functions:

```
print(df.describe()) # Summary statistics
print(df.info()) # Information about the DataFrame
print(len(df)) # Length of the DataFrame
print(df.shape) # Shape of the DataFrame
print(df.columns) # Column names
print(df['column_name'].value_counts()) # Counts of unique
```



# Selecting and Filtering Data —•

We can select and filter data using pandas functions.

Selecting columns:

```
Df['age'] # Selecting one column
df[['survived', 'pclass', 'name']] #selecting multiple columns
```

• Selecting rows:

```
df.iloc[row_index], df.loc[row_label]
  df.iloc[10] # 11th row of DataFrame
  df.loc[df['age'] > 18] # Rows where age is greater than 18
```

• Filtering data based on specific criteria:

```
df[df['column'] > value], df.query('column > value')
print(df[df['age'] < 30])</pre>
```

• We can call advanced statistics and aggregation functions to analyze the data:

```
df.mean(), df.median(), df.mode(), df.var(), df.std()
print(df.groupby('sex').mean())
```

# Data Cleaning —

We can clean and preprocess data using pandas functions.

Handling missing values:

```
df['deck'].fillna('Unknown') # Filling missing values in the 'deck'
column of a DataFrame df with the string 'Unknown'.
```

Removing unnecessary columns/indices:

```
df.dropna(subset=['deck'], inplace=True) # Drop rows
where the 'deck' column has NaN values
df.dropna(subset=['embarked']) # Drop 'embarked' column
```

Handling duplicates:

```
df.drop_duplicates()
```

• Data type conversion:

```
# Convert 'Fare' column to float
df['Fare'] = df['Fare'].astype(float)
```

Data normalization:

```
def normalize_column(col):
    return (col - col.min()) / (col.max() - col.min())
# Normalize the 'Age' and 'Fare' columns
df['Age'] = normalize_column(df['Age'])
df['Fare'] = normalize_column(df['Fare'])
```

• Renaming columns:



### Data Transformation — •

We can transform data using various pandas functions.

Sorting data:

```
df.sort_values('col'), df.sort_index()
```

Merging and joining:

```
pd.merge(df1, df2, on='key')
other_df = pd.read_csv('other_data.csv')
pd.merge(df, other_df, on='Ticket')
```

Grouping by a column:

```
df.groupby('col').sum(), df.groupby('col').mean()

df.groupby('pclass')['survived'].mean()

# The result is a series where the index is the unique values in the 'pclass'
column (1, 2, 3 in this case), and the values are the average survival rates for each class.
```

Output:

Pclass					
1	0.65				
2	0.45				
3	0.25				

Arithmetic operations:

```
df['new_column'] = df['column1'] + df['column2']
df['FamilySize'] = df['sibSp'] + df['parch']
```

Reshaping:

```
df.pivot(index='index_col', columns='col')
df.pivot_table(values='survived', index='pclass', columns='sex', aggfunc='mean')
```

## Lambda Functions — •

We can apply customized functions with Lambda expressions to modify data in the DataFrame.

• The apply() function on one column

```
# Applying a Lambda function to one column

df['Fare_doubled'] = df['fare'].apply(lambda x: x * 2)
```

• The apply () function on multiple columns

```
# Applying a Lambda function to multiple columns
df['Family_size'] = df.apply(lambda x: x['sibsp'] + x['parch'] + 1, axis=1)
```

• The apply() function on one one row

```
# Applying a Lambda function to one row
row = df.iloc[0]
result = row.apply(lambda x: x * 2)
```



• The apply () function on multiple rows

```
# Applying a Lambda function to multiple rows
rows = df.iloc[1:3]
result = rows.apply(lambda x: x['fare'] * x['age'], axis=1)
```

• The assign() function on one column

```
# Using assign() to create a new column based on an existing column
df = df.assign(Age_doubled=df['age'] * 2)
```

• The assign() function on multiple columns

```
# Using assign() to create multiple new columns

df = df.assign(Fare_age_ratio=df['fare'] / df['age'],

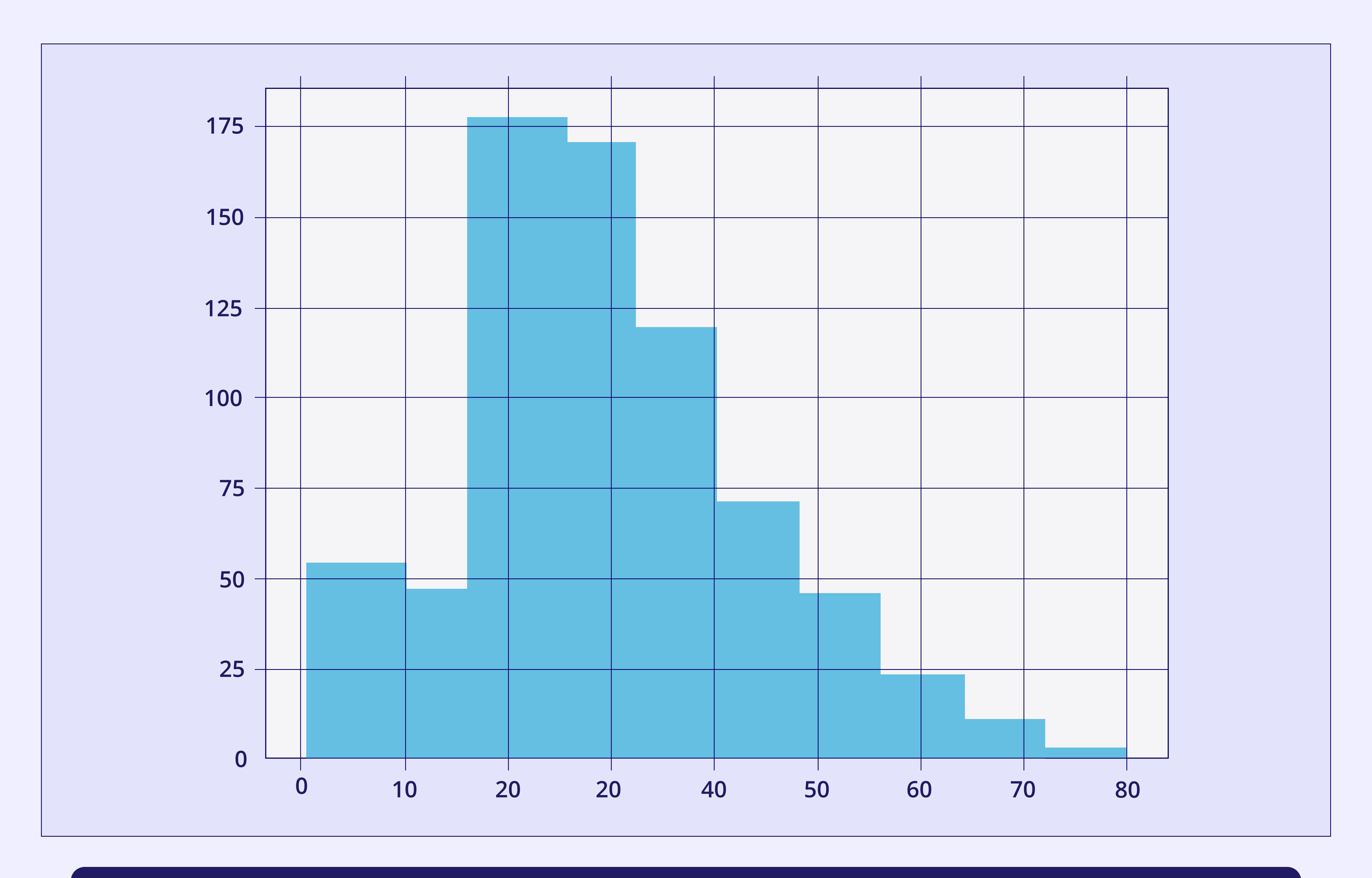
Fare_class_ratio=df['fare'] / df['pclass'])
```

### Visualization — •

- We can use pandas functions for data visualization.
  - hist()

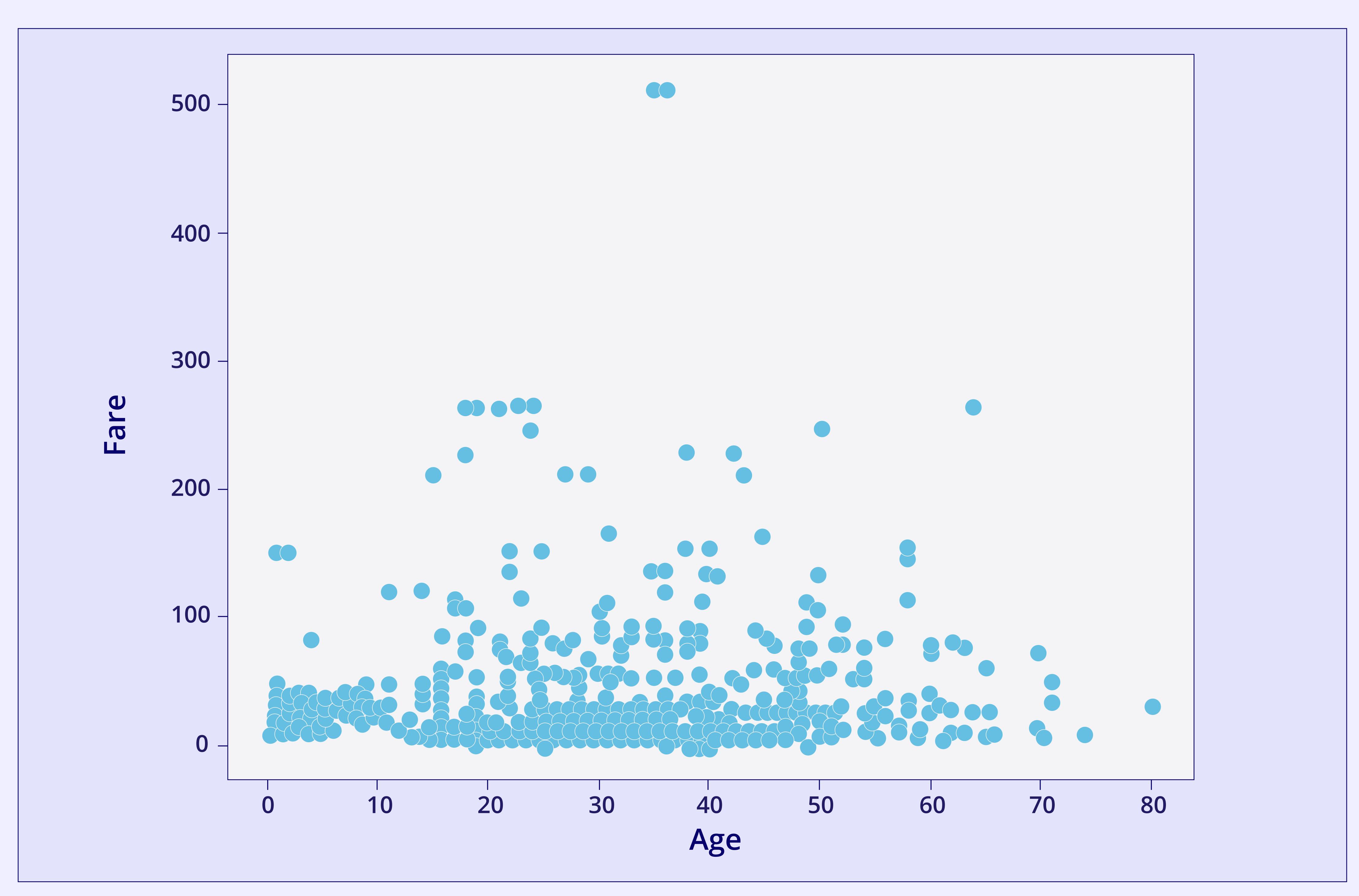
```
df['age'].hist()
```

We used the hist() function to plot a histogram for the distribution of ages among Titanic passengers.



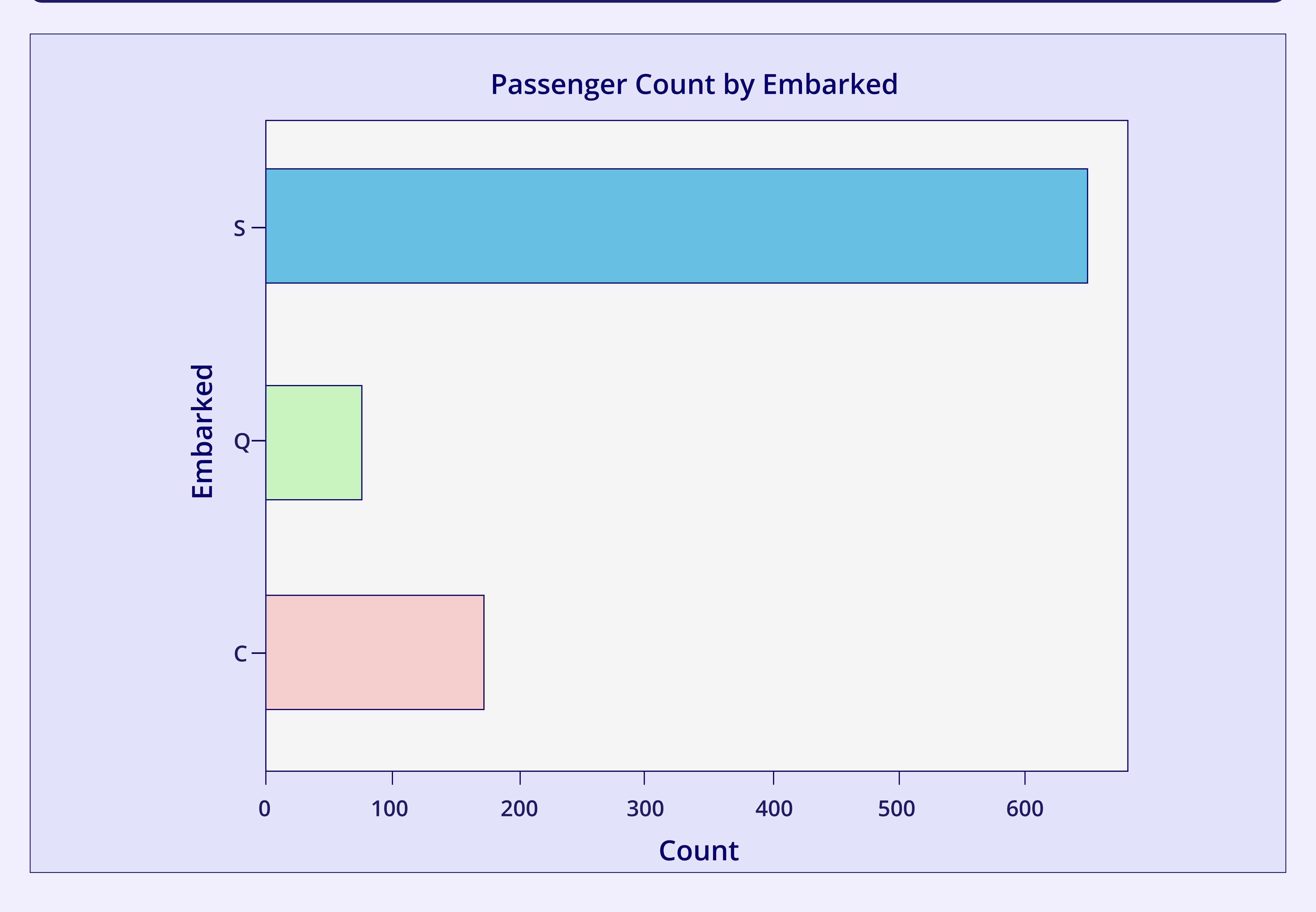
```
df.plot(kind='scatter', x='age', y='fare')
```

We used the plot() function to draw a scatter plot for presenting the relationship between age and fare paid by Titanic passengers.



The code below groups the DataFrame df by the 'embarked' column, calculates the occurrences of each unique value in the 'embarked' column (value\_counts() method), and plots a horizontal bar chart.





We can blend different functions, from pandas to create complex visualizations. Here, we used the plot() function to plot the bar chart in horizontal alignment to present the count of passengers at different locations ('C', 'Q', 'S').