International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

EDA in DataScience

Dharani.M

Student
Lady Doak College

What is Exploratory Data Analysis?

Exploratory Data Analysis (EDA) is an important first step in data science projects. It involves looking at and visualizing data to understand its main features, find patterns, and discover how different parts of the data are connected.

EDA helps to spot any unusual data or outliers and is usually done before starting more detailed statistical analysis or building models.

Why Exploratory Data Analysis is Important?

Exploratory Data Analysis (EDA) is important for several reasons, especially in the context of data science and statistical modeling. Here are some of the key reasons why EDA is a critical step in the data analysis process:

- Helps to understand the dataset, showing how many features there are, the type of data in each feature, and how the data is spread out, which helps in choosing the right methods for analysis.
- EDA helps to identify hidden patterns and relationships between different data points, which help us in and model building.
- Allows to spot errors or unusual data points (outliers) that could affect your results.
- Insights that you obtain from EDA help you decide which features are most important for building models and how to prepare them to improve performance.
- By understanding the data, EDA helps us in choosing the best modeling techniques and adjusting them for better results.

Types of Exploratory Data Analysis

There are various sorts of EDA strategies based on nature of the records. Depending on the number of columns we are analyzing we can divide EDA into three types:

1. Univariate
2. Bivariate
3. Multivariate.

1. Univariate Analysis

Univariate analysis focuses on studying one variable to understand its characteristics. It helps describe the data and find patterns within a single feature. Common methods include **histograms** to show data distribution, **box plots** to detect outliers and understand data spread, and **bar charts** for categorical data. **Summary statistics** like **mean**, **median**, **mode**, **variance**, and **standard deviation** help describe the central tendency and spread of the data.

2. Bivariate Analysis

Bivariate analysis focuses on exploring the relationship between two variables to find connections, correlations, and dependencies. It's an important part of exploratory data analysis that helps understand how two variables interact. Some key techniques used in bivariate analysis include **scatter plots**, which visualize the relationship between two continuous variables; **correlation coefficient**, which measures how strongly two variables are related, commonly using **Pearson's correlation** for linear relationships; and **cross-tabulation**, or **contingency tables**, which show the frequency distribution of two categorical variables and help understand their relationship.

Line graphs are useful for comparing two variables over time, especially in time series data, to identify trends or patterns. **Covariance** measures how two variables change together, though it's often supplemented by the correlation coefficient for a clearer, more standardized view of the relationship.

3. Multivariate Analysis

Multivariate analysis examines the relationships between two or more variables in the dataset. It aims to understand how variables interact with one another, which is crucial for most statistical modeling techniques. It include Techniques like **pair plots**, which show the relationships between multiple variables at once, helping to see how they interact. Another technique is **Principal Component Analysis (PCA)**, which reduces the complexity of large datasets by simplifying them, while keeping the most important information.

In addition to univariate and multivariate analysis, there are specialized EDA techniques tailored for specific types of data or analysis needs:

- **Spatial Analysis**: For geographical data, using maps and spatial plotting to understand the geographical distribution of variables.
- **Text Analysis**: Involves techniques like word clouds, frequency distributions, and sentiment analysis to explore text data.
- **Time Series Analysis:** This type of analysis is mainly applied to statistics sets that have a temporal component. Time collection evaluation entails inspecting and modeling styles, traits, and seasonality inside the statistics through the years. Techniques like line plots, autocorrelation analysis, transferring averages, and ARIMA (AutoRegressive Integrated Moving Average) fashions are generally utilized in time series analysis.

Steps for Performing Exploratory Data Analysis

Performing Exploratory Data Analysis (EDA) involves a series of steps designed to help you understand the data you're working with, uncover underlying patterns, identify anomalies, test hypotheses, and ensure the data is clean and suitable for further analysis.

Step 1: Understand the Problem and the Data

The first step in any data analysis project is to clearly understand the problem you're trying to solve and the data you have. This involves asking key questions such as:

- **What is the business goal or research question**?
- **What are the variables in the data** and what do they represent?
- **What types of data** (numerical, categorical, text, etc.) do you have?
- Are there any known **data quality issues** or **limitations**?
- Are there any **domain-specific concerns** or restrictions?

By thoroughly understanding the problem and the data, you can better plan your analysis, avoid wrong assumptions, and ensure accurate conclusions.

Step 2: Import and Inspect the Data

After clearly understanding the problem and the data, the next step is to import the data into your analysis environment (like **Python**, **R**, or a spreadsheet tool). At this stage, it's crucial to examine the data to get an initial understanding of its structure, variable types, and potential issues.

Here's what you can do:

- **Load the data** into your environment carefully to avoid errors or truncations.
- **Examine the size** of the data (number of rows and columns) to understand its complexity.
- **Check for missing values** and see how they are distributed across variables, since missing data can impact the quality of your analysis.
- **Identify data types** for each variable (like numerical, categorical, etc.), which will help in the next steps of data manipulation and analysis.
- **Look for errors** or inconsistencies, such as invalid values, mismatched units, or outliers, which could signal deeper issues with the data.

By completing these tasks, you'll be prepared to clean and analyze the data more effectively.

Step 3: Handle Missing Data

Missing data is common in many datasets and can significantly affect the quality of your analysis. During **Exploratory Data Analysis (EDA)**, it's important to identify and handle missing data properly to avoid biased or misleading results.

Here's how to handle it:

- **Understand the patterns** and possible reasons for missing data. Is it **missing completely at random** (MCAR), **missing at random** (MAR), or **missing not at random** (MNAR)? Knowing this helps decide how to handle the missing data.
- **Decide whether to remove** missing data (listwise deletion) or **impute** (fill in) the missing values. Removing data can lead to biased outcomes, especially if the missing data isn't MCAR. Imputing values helps preserve data but should be done carefully.
- Use appropriate **imputation methods** like **mean/median imputation**, **regression imputation**, or machine learning techniques like **KNN** or **decision trees** based on the data's characteristics.
- **Consider the impact** of missing data. Even after imputing, missing data can cause uncertainty and bias, so interpret the results with caution.

Properly handling missing data improves the accuracy of your analysis and prevents misleading conclusions.

Step 4: Explore Data Characteristics

After addressing missing data, the next step in **EDA** is to explore the characteristics of your data by examining the **distribution**, **central tendency**, and **variability** of your variables, as well as identifying any **outliers** or anomalies. This helps in selecting appropriate analysis methods and spotting potential data issues. You should calculate **summary statistics** like **mean**, **median**, **mode**, **standard deviation**, **skewness**, and **kurtosis** for numerical variables. These provide an overview of the data's distribution and help identify any irregular patterns or issues.

Step 5: Perform Data Transformation

Data transformation is an essential step in **EDA** because it prepares your data for accurate analysis and modeling. Depending on your data's characteristics and analysis needs, you may need to transform it to ensure it's in the right format.

Common transformation techniques include:

- **Scaling or normalizing** numerical variables (e.g., **min-max scaling** or **standardization**).
- **Encoding categorical variables** for machine learning (e.g., **one-hot encoding** or **label encoding**).

- Applying **mathematical transformations** (e.g., **logarithmic** or **square root**) to correct skewness or non-linearity.

- **Creating new variables** from existing ones (e.g., calculating ratios or combining variables).
- **Aggregating or grouping** data based on specific variables or conditions.

Step 6: Visualize Data Relationship

Visualization is a powerful tool in the **EDA** process, helping to uncover relationships between variables and identify patterns or trends that may not be obvious from summary statistics alone.

- For categorical variables, create **frequency tables**, **bar plots**, and **pie charts** to understand the distribution of categories and identify imbalances or unusual patterns.
- For numerical variables, generate **histograms**, **box plots**, **violin plots**, and **density plots** to visualize distribution, shape, spread, and potential outliers.
- To explore relationships between variables, use **scatter plots**, **correlation matrices**, or statistical tests like **Pearson's correlation coefficient** or **Spearman's rank correlation.**

Step 7: Handling Outliers

Outliers are data points that significantly differ from the rest of the data, often caused by errors in measurement or data entry. Detecting and handling outliers is important because they can skew your analysis and affect model performance. You can identify outliers using methods like **interquartile range (IQR), Z-scores, or domain-specific rules**. Once identified, outliers can be removed or adjusted depending on the context. Properly managing outliers ensures your analysis is accurate and reliable.

Step 8: Communicate Findings and Insights

The final step in **EDA** is to communicate your findings clearly. This involves summarizing your analysis, pointing out key discoveries, and presenting your results in a clear and engaging way.

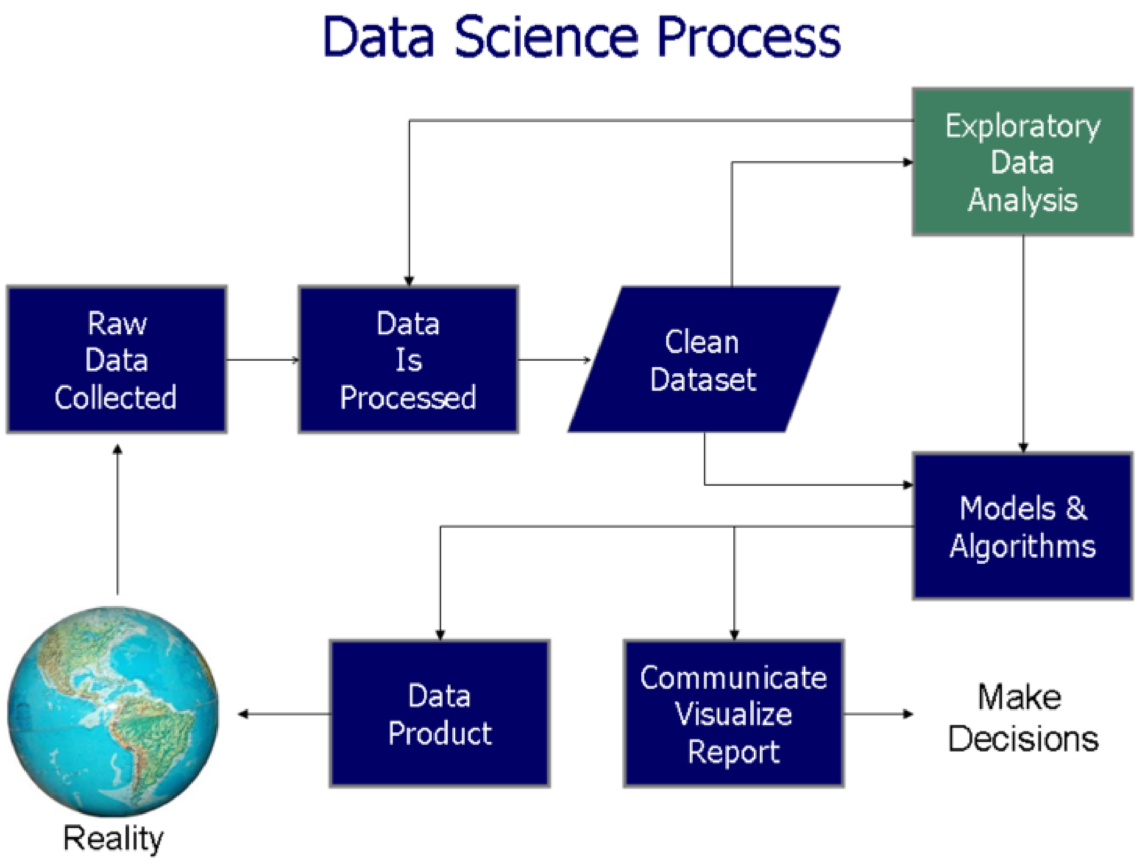
- Clearly state the **goals** and **scope** of your analysis.
- Provide **context** and background to help others understand your approach.
- Use **visualizations** to support your findings and make them easier to understand.
- Highlight key **insights**, **patterns**, or **anomalies** discovered.
- Mention any **limitations** or challenges faced during the analysis.
- Suggest **next steps** or areas that need further investigation.

Effective conversation is critical for ensuring that your EDA efforts have a meaningful impact and that your insights are understood and acted upon with the aid of stakeholders.

Exploratory Data Analysis (EDA) can be performed using a variety of tools and software, each offering features that deal to different data and analysis needs.

In **Python**, libraries like **Pandas** are essential for data manipulation, providing functions to clean, filter, and transform data. **Matplotlib** is used for creating basic static, interactive, and animated visualizations, while **Seaborn**, built on top of Matplotlib, allows for the creation of more attractive and informative statistical plots. For interactive and advanced visualizations, **Plotly** is an excellent choice

In **R**, packages like **ggplot2** are powerful for creating complex and visually appealing plots from data frames. **dplyr** helps in data manipulation, making tasks like filtering and summarizing easier, and **tidyr** ensures your data is in a tidy format, making it easier to work with.



Typical graphical techniques used in EDA are:

- Box plot
- Histogram
- Multi-vari chart
- Run chart
- Pareto chart
- Scatter plot (2D/3D)
- Stem-and-leaf plot

- Parallel coordinates
- Odds ratio
- Targeted projection pursuit
- Heat map
- Bar chart
- Horizon graph
- Glyph-based visualization methods such as PhenoPlot and Chernoff faces
- Projection methods such as grand tour, guided tour and manual tour
- Interactive versions of these plots.

Why is EDA important in data science?

The main purpose of EDA is to help look at data before making any assumptions. It can help identify obvious errors, as well as better understand patterns within the data, detect outliers or anomalous events, find interesting relations among the variables.

Data scientists can use exploratory analysis to ensure the results they produce are valid and applicable to any desired business outcomes and goals. EDA also helps stakeholders by confirming they are asking the right questions. EDA can help answer questions about standard deviations, categorical variables, and confidence intervals. Once EDA is complete and insights are drawn, its features can then be used for more sophisticated data analysis or modeling, including machine learning.

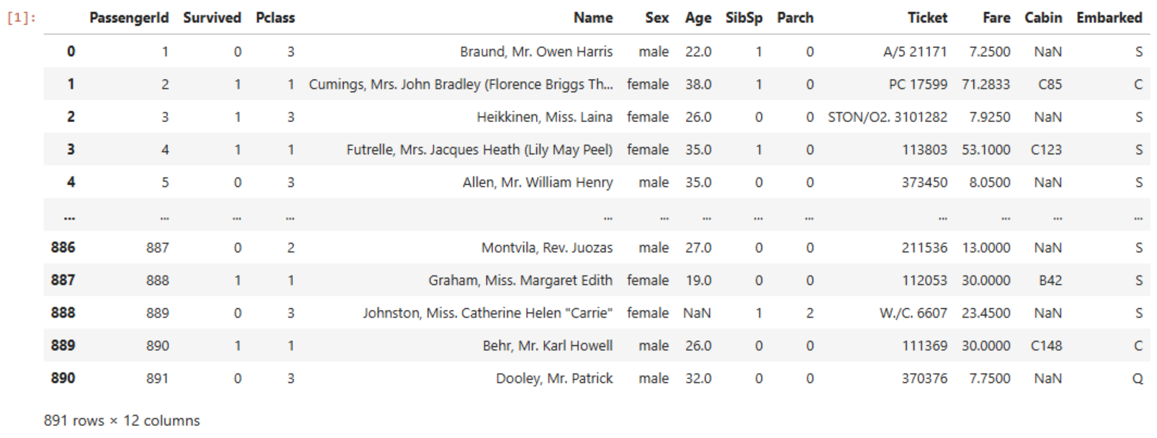
I have conducted an Exploratory Data Analysis (EDA) on the **train.csv** dataset, which is presented below.

Dataset from Kaggle : https://www.kaggle.com/datasets/shuofxz/titanic-machine-learning-from-disaster?select=train.csv

```
import pandas as pd
df=pd.read_csv('train.csv')
df
```

Interpretation:

This Python code imports the **pandas** library and reads a CSV file named 'train.csv' into a **DataFrame** named df using pd.read_csv('train.csv'). This function automatically detects column names, data types, and missing values, allowing structured data to be efficiently loaded for analysis. The dataset contains **891 records** with attributes: **PassengerId**, a unique identifier for each passenger; **Survived**, a binary indicator (0 = No, 1 = Yes) showing survival status; **Pclass**, representing the passenger class (1st, 2nd, or 3rd); **Name**, the full name of the passenger; **Sex**, indicating gender; **Age**, representing the passenger's age in years (which may have missing values); **SibSp**, the number of siblings/spouses aboard; **Parch**, the number of parents/children aboard; **Ticket**, the assigned ticket number; **Fare**, the price paid for the ticket; **Cabin**, the assigned cabin number (with possible missing values); and **Embarked**, denoting the port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton). By displaying df, users can view the first and last few rows of the dataset in a structured tabular format. This Titanic dataset is widely used in machine learning and statistical analysis for survival prediction, classification tasks, and exploratory data analysis.

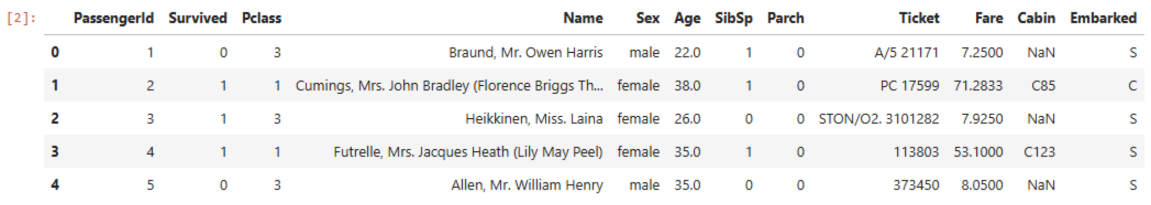


```
[1]:      PassengerId  Survived  Pclass                                               Name     Sex   Age  SibSp  Parch             Ticket     Fare  Cabin  Embarked
      0             1         0       3                            Braund, Mr. Owen Harris    male  22.0      1      0          A/5 21171   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
    ...           ...       ...     ...                                                ...     ...   ...    ...    ...                ...      ...    ...       ...
    886           887         0       2                              Montvila, Rev. Juozas    male  27.0      0      0             211536  13.0000    NaN         S
    887           888         1       1                       Graham, Miss. Margaret Edith  female  19.0      0      0             112053  30.0000    B42         S
    888           889         0       3           Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1      2         W./C. 6607  23.4500    NaN         S
    889           890         1       1                              Behr, Mr. Karl Howell    male  26.0      0      0             111369  30.0000   C148         C
    890           891         0       3                                Dooley, Mr. Patrick    male  32.0      0      0             370376   7.7500    NaN         Q

    891 rows × 12 columns
```

df.head()

Interpretation:

The command df.head() returns the first five rows of the DataFrame, providing a quick overview of the dataset. In the **Titanic dataset**, it displays key attributes such as PassengerId, a unique identifier for each passenger; Survived, indicating survival status (0 = Did not survive, 1 = Survived); Pclass, representing the passenger's ticket class (1st, 2nd, or 3rd); Name, showing the passenger's full name; Sex, specifying gender; Age, denoting the passenger's age, which may contain missing values; SibSp, the number of siblings/spouses aboard; Parch, the number of parents/children aboard; Ticket, the ticket number assigned; Fare, representing the price paid for the ticket; Cabin, indicating the assigned cabin (which may have missing values); and Embarked, showing the port of embarkation (C for Cherbourg, Q for Queenstown, S for Southampton).The Highest Fare in the five records is 71.2833. Moreover, The people booked the 3rd class in Pclass.This command helps in quickly inspecting the dataset, identifying missing values, and understanding the data structure before further analysis or preprocessing.



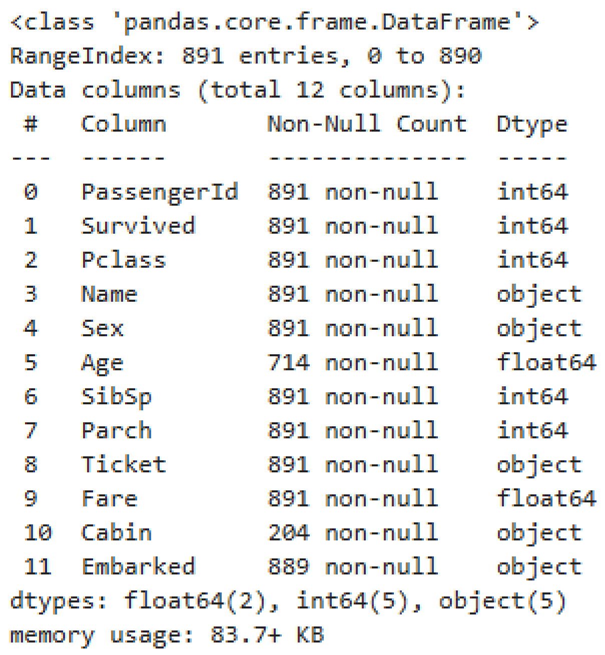
```
[2]:    PassengerId  Survived  Pclass                                               Name     Sex   Age  SibSp  Parch            Ticket     Fare  Cabin  Embarked
     0            1         0       3                            Braund, Mr. Owen Harris    male  22.0      1      0         A/5 21171   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
```

import pandas as pd
df=pd.read_csv('train.csv')
df.info()

Interpretation:

The given Python code imports the **pandas** library and reads the 'train.csv' file into a DataFrame named df using pd.read_csv('train.csv'). The df.info() function provides a concise summary of the dataset, displaying key details such as the total number of entries (rows), the number of columns, column names, data types, and non-null counts for each column. This function is particularly useful for identifying missing values and understanding the data structure. Given the **Titanic dataset**, the output of df.info() would look something like this:

This output shows that there are **891 records (entries)** and **12 columns**, with different data types such as int64 (for numerical values like PassengerId and Pclass), float64 (for values like Age and Fare), and object (for categorical/text values like Name and Sex). It also highlights missing values in columns like Age, Cabin, and Embarked, which need to be handled before analysis. This function is crucial for understanding the dataset structure and preparing it for further processing.

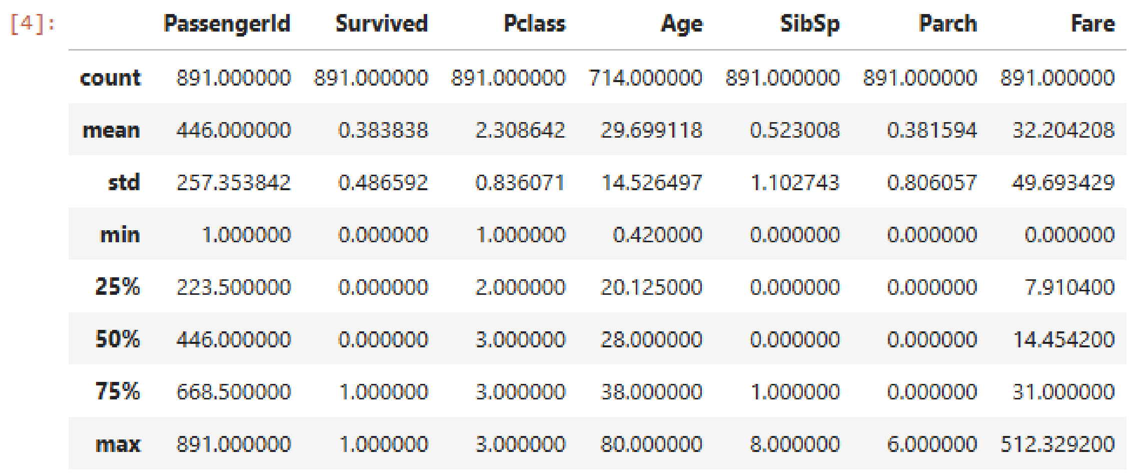


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    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
```

df.describe()

Interpretation :

The command df.describe() provides a statistical summary of all numerical columns in the dataset, giving insights into the distribution, central tendency, and spread of the data. It displays key metrics such as count, which shows the number of non-null values in each column (e.g., Age has 714 values, indicating 177 missing entries), and mean, representing the average value for each attribute (e.g., the average age is approximately 29.7 years, while the average fare is around 32.2). The std (standard deviation) measures data spread, revealing high variation in fare prices with a standard deviation of 49.69. The min and max values indicate the smallest and largest values in each column, such as a minimum fare of 0 and a maximum of 512.33. The 25%, 50% (median), and 75% percentiles describe the distribution, showing that 25% of passengers were aged 20 or younger, while 50% were aged 28. This summary helps in detecting outliers, understanding the dataset's structure, and identifying missing values before performing data preprocessing and analysis.

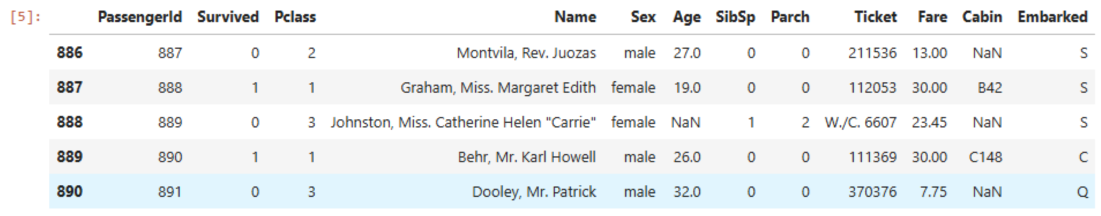

[4]:

	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

df.tail()

Interpretation:

The command df.tail() retrieves the last five rows of the dataset, providing insight into the final entries. In the Titanic dataset, it displays details such as PassengerId, which uniquely identifies each passenger, and Survived, where 0 indicates the passenger did not survive and 1 means they did. For example, passenger 887 (Rev. Juozas Montvila), a 27-year-old male, did not survive, while passenger 888 (Miss. Margaret Graham), a 19-year-old female, survived. The Pclass column represents the ticket class, where passengers like Karl Howell Behr (Passenger 889) traveled in 1st class, while Patrick Dooley (Passenger 891) traveled in 3rd class and did not survive. The Sex and Age columns indicate gender and age, though some values, like Age for Passenger 888 (Johnston, Miss.), are missing. The SibSp and Parch columns denote the number of siblings/spouses and parents/children aboard, respectively; for instance, Passenger 888 had one sibling and two parents onboard. The Ticket column contains ticket numbers such as 112053 for Miss. Graham and 370376 for Mr. Dooley, while Fare shows the price paid, which varies significantly, with Mr. Behr paying 30.00 and Mr. Dooley paying only 7.75. The Cabin column is mostly empty, but some passengers, like Karl Howell Behr, had cabin C148 assigned. Lastly, the Embarked column shows the port of boarding, where most passengers embarked from Southampton (S), but Behr boarded from Cherbourg (C) and Dooley from Queenstown (Q). This function helps check data consistency, missing values, and the dataset's structure before analysis.

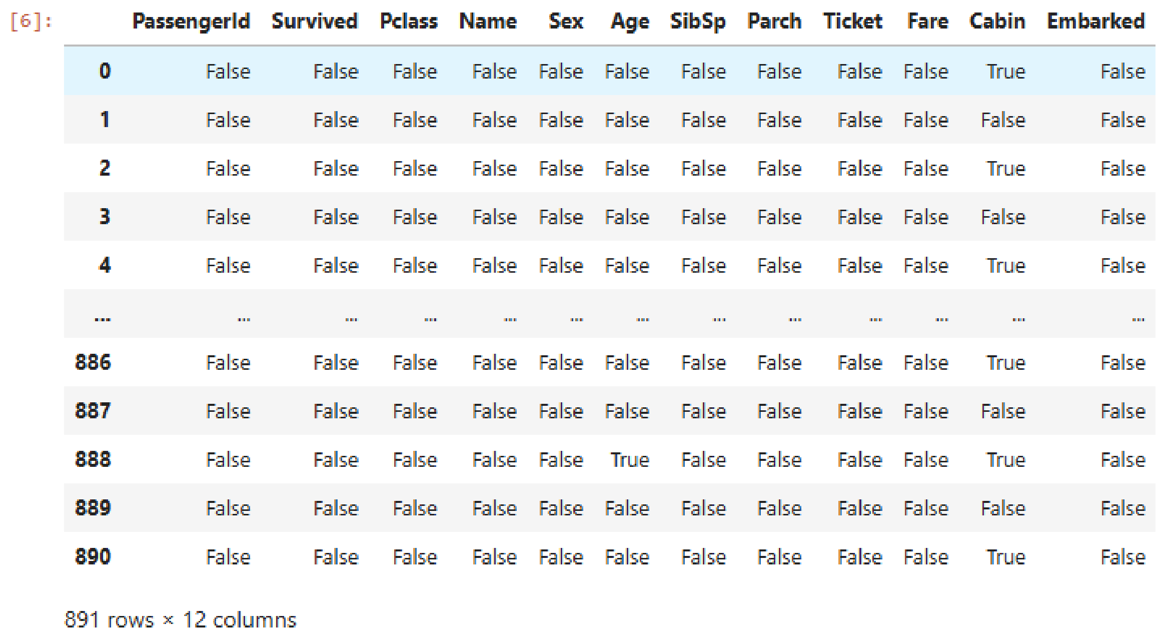

[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

df.isnull()

Interpretation:

The command df.isnull() generates a DataFrame of the same shape as the original dataset, where each value is either True or False, indicating whether a particular cell contains a missing (null) value. In the **Titanic dataset**, this function helps identify missing values across different attributes. For instance, the Cabin column has many True values, indicating a significant number of missing entries, with **687 out of 891 records lacking cabin information**. The Age column also has missing values, with **177 passengers having unknown ages**, while the Embarked column has only **two missing values**. Other attributes, such as PassengerId, Survived, Pclass, Name, Sex, SibSp, Parch, Ticket, and Fare, have no missing values. To obtain a summary of missing values per column, the command df.isnull().sum() is used, which confirms that Age, Cabin, and Embarked require attention before further data analysis or model training. This function is crucial for handling missing data, allowing for appropriate imputation strategies or column exclusions.



[6]:

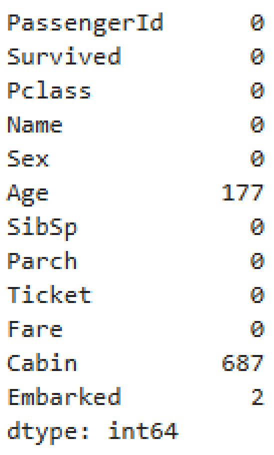
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	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
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	True	False
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	False	False	True	False	False	False	False	True	False
889	False	False	False	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	False	False	True	False

891 rows × 12 columns

print(df.isnull().sum())

Interpretation:

The command print(df.isnull().sum()) provides a summary of missing values in each attribute of the Titanic dataset by counting the number of NaN (null) values per attribute. The output reveals that the **Cabin** attribute has the highest number of missing values, with **687 out of 891 records lacking cabin information**, followed by the **Age** attribute, which has **177 missing values**, indicating that a significant portion of passengers' ages were not recorded. The **Embarked** attribute has only **two missing values**, which can be easily filled using the most common embarkation port. Other attributes, such as PassengerId, Survived, Pclass, Name, Sex, SibSp, Parch, Ticket, and Fare, are fully populated without any missing values. Identifying missing data is crucial in data preprocessing, as it allows for appropriate handling strategies such as imputation, attribute exclusion, or filling missing values with suitable replacements to ensure accurate analysis and model performance.

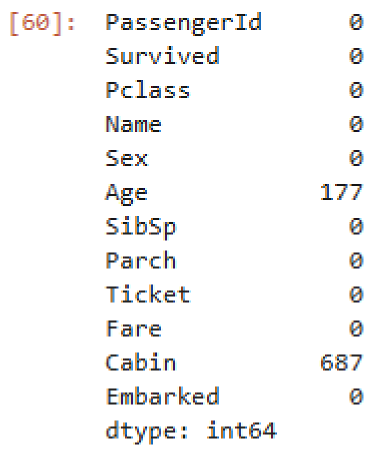


```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

df['Embarked'].ffill(inplace=True)
df.isnull().sum()

Interpretation:

The command df['Embarked'].ffill(inplace=True) is used to fill the missing values in the **Embarked** attribute using **forward fill (ffill)**, which replaces any missing value with the previous non-null value in the dataset. Before applying this method, the Embarked attribute had **two missing values**. After executing the command, running df.isnull().sum() confirms that the missing values in Embarked have been successfully filled, reducing its count to **zero**. The updated summary now shows that only the Age attribute still has **177 missing values**, and the Cabin attribute remains the most incomplete, with **687 missing values**. Forward fill is particularly useful for categorical data like Embarked, where filling missing values with the most recent available data ensures continuity. However, for numerical columns like Age, alternative imputation methods such as mean or median substitution would be more appropriate to maintain data consistency and accuracy in further analysis.

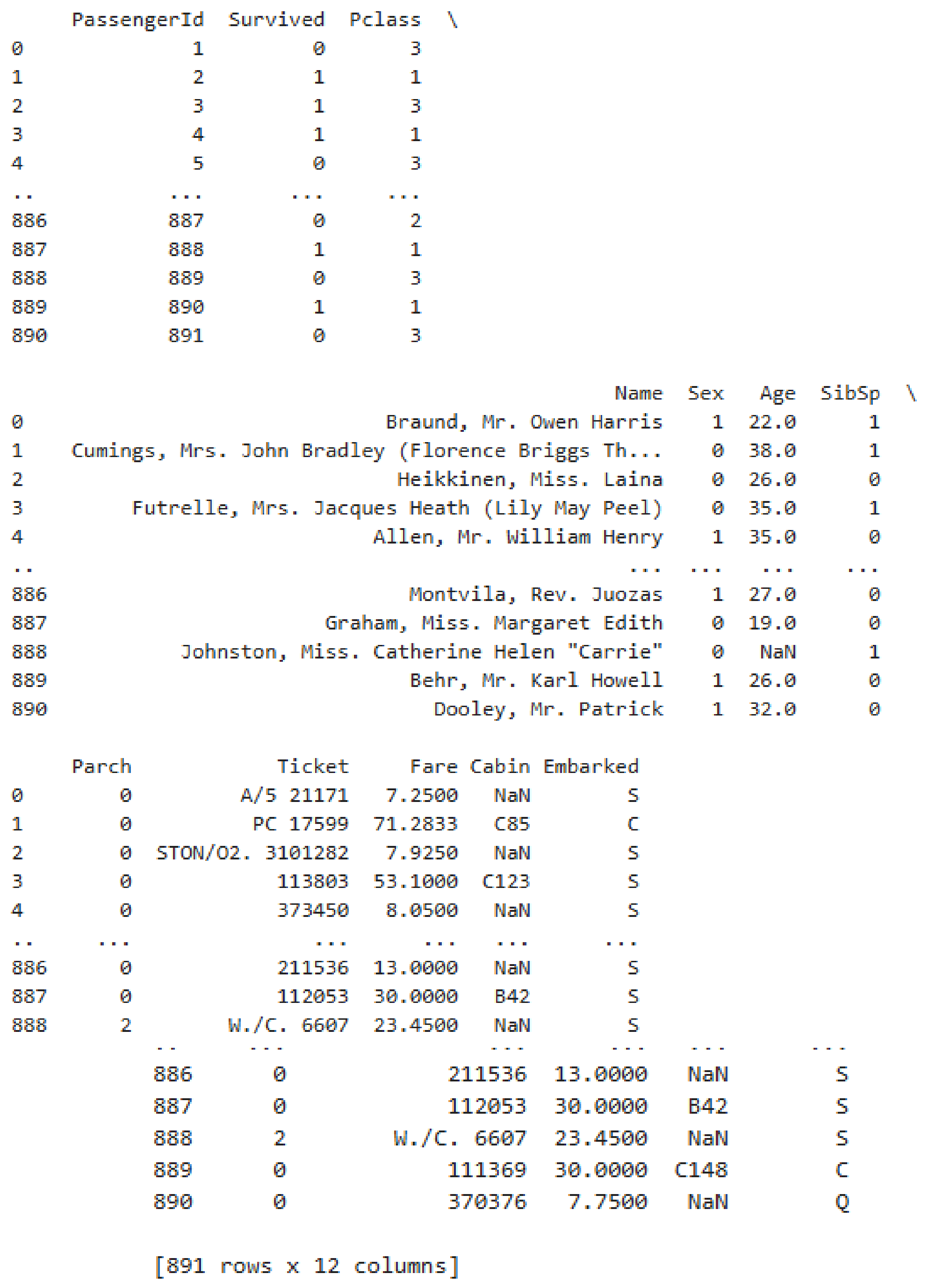

```
[60]: PassengerId      0
      Survived         0
      Pclass           0
      Name             0
      Sex              0
      Age            177
      SibSp            0
      Parch            0
      Ticket           0
      Fare             0
      Cabin          687
      Embarked       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

import pandas as pd
from sklearn import preprocessing
df = pd.read_csv('train.csv')
label_encoder = preprocessing.LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
print(df)

Interpretation:

This code reads the Titanic dataset (train.csv) using **pandas** and applies **Label Encoding** to the Sex attribute using **scikit-learn's preprocessing module**. Initially, the Sex attribute contains categorical values (male and female), which are not directly usable in numerical computations. To convert them into a machine-learning-friendly format, the **LabelEncoder** assigns numerical values where **'male' is encoded as 1** and **'female' is encoded as 0**. After executing the code, the dataset remains unchanged except for the

Sex attribute , which now contains numerical representations instead of text. The transformed dataset is then printed, showing that all occurrences of male have been replaced with 1 and female with 0. This encoding is particularly useful in machine learning applications, as numerical data is required for most algorithms to process features efficiently. From first five records three of them are females. Converting categorical variables into numerical format ensures that models can interpret and use the Sex column effectively in predictive analysis.

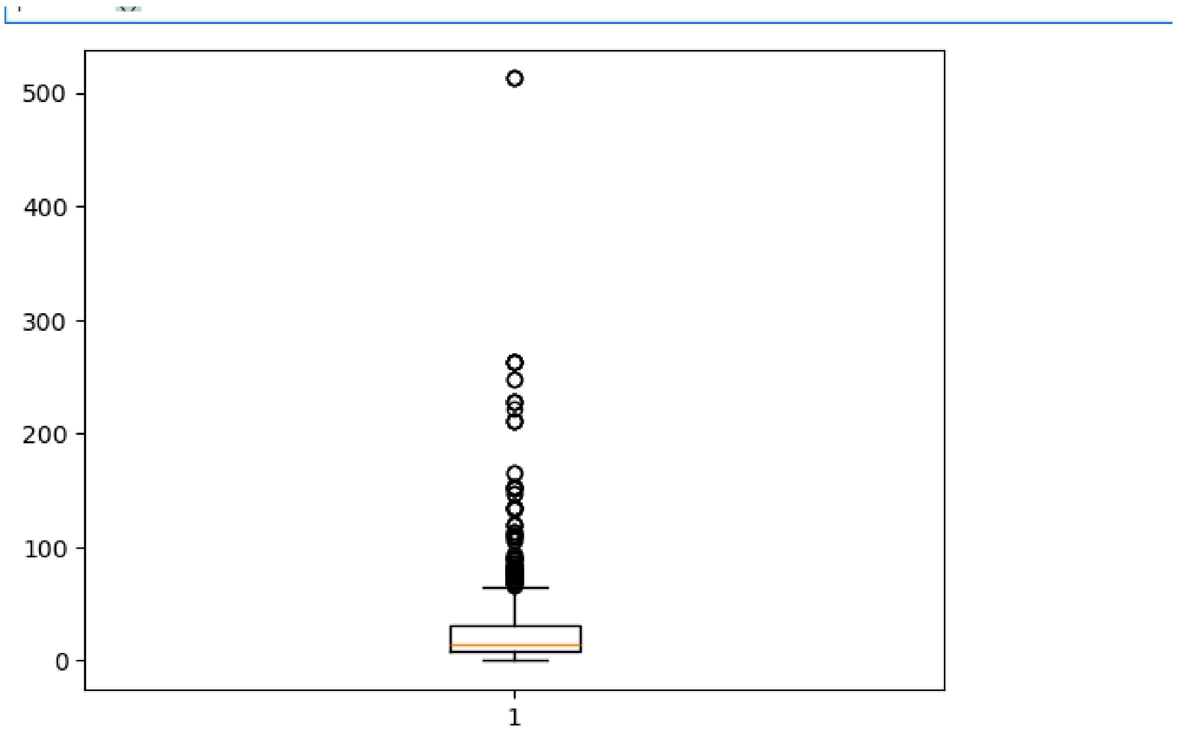


```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name  Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    1  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...    0  38.0      1
2                               Heikkinen, Miss. Laina    0  26.0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)    0  35.0      1
4                             Allen, Mr. William Henry    1  35.0      0
..                                                 ...  ...   ...    ...
886                              Montvila, Rev. Juozas    1  27.0      0
887                       Graham, Miss. Margaret Edith    0  19.0      0
888           Johnston, Miss. Catherine Helen "Carrie"    0   NaN      1
889                              Behr, Mr. Karl Howell    1  26.0      0
890                                Dooley, Mr. Patrick    1  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

```
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv('train.csv')
plt.boxplot(df['Fare'])
plt.show()
```

Interpretation:

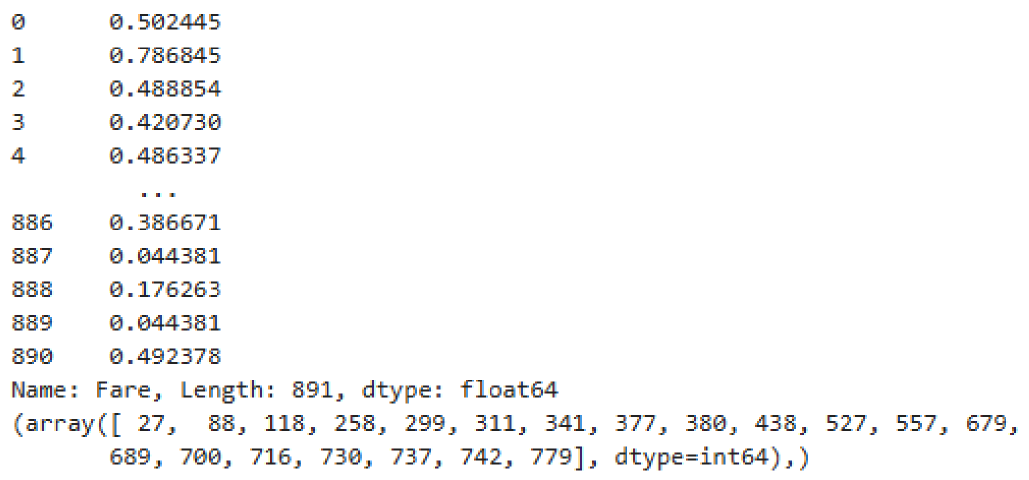
This code reads the **Titanic dataset (train.csv)** using **pandas** and generates a **box plot** for the Fare attribute using **Matplotlib**. A box plot is a powerful visualization tool that helps analyze the distribution, median, quartiles, and outliers in numerical data. The central line inside the box represents the **median fare**, while the edges of the box indicate the **interquartile range (IQR)**, covering the middle 50% of fare values. The whiskers extend to represent the range of non-outlier fares, whereas any individual points beyond the whiskers are considered **outliers**, which in this case likely correspond to **first-class passengers who paid extremely high ticket prices**. Given that Titanic fares varied significantly based on passenger class and ticket type, the plot is expected to show a **right-skewed distribution**, where most fares are relatively low, but a few high-priced tickets create extreme values. The highest outlier is abve 500. This visualization is useful for identifying skewness, fare disparities across different classes, and potential anomalies in the dataset.



```
import numpy as np
from scipy import stats
z=np.abs(stats.zscore(df.Fare))
print(z)
threshold=3
print(np.where(z>3))
```

Interpretation:

This code calculates the **Z-score** for the Fare attribute in the Titanic dataset and identifies **outliers** based on a threshold of 3 standard deviations. It first imports **NumPy** and **SciPy**, then computes the **absolute Z-score** for each value in the Fare attribute using stats.zscore(df.Fare). The **Z-score** represents how far each fare deviates from the **mean (μ) in terms of standard deviations (σ)**. Values close to **0** indicate fares near the average, while higher Z-scores indicate fares that are significantly different. The np.abs() function ensures that all Z-score values are positive, making deviation analysis straightforward. The threshold is set at **3**, meaning any fare with a **Z-score greater than 3** is considered an **outlier**. The function np.where(z > 3) returns the **indices of these outliers**, highlighting extreme fare values. Given that **Titanic ticket prices ranged from as low as Rs. 0 to over Rs.600**, expensive first-class tickets are expected to be detected as outliers. The analysis helps in **identifying unusually high fares** that might influence statistical models, and handling these extreme values properly can improve data-driven insights.

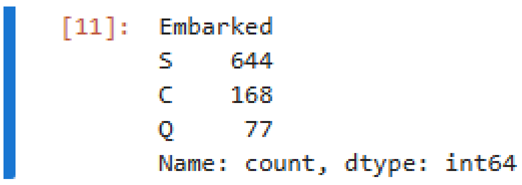


```
0      0.502445
1      0.786845
2      0.488854
3      0.420730
4      0.486337
         ...
886    0.386671
887    0.044381
888    0.176263
889    0.044381
890    0.492378
Name: Fare, Length: 891, dtype: float64
(array([ 27,  88, 118, 258, 299, 311, 341, 377, 380, 438, 527, 557, 679,
       689, 700, 716, 730, 737, 742, 779], dtype=int64),)
```

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

Interpretation:

The command df['Embarked'].value_counts() calculates the number of occurrences for each unique value in the **Embarked** attribute of the Titanic dataset, which represents the port from which each passenger boarded. The three possible values are **'S' for Southampton, 'C' for Cherbourg, and 'Q' for Queenstown**. Based on the output, **Southampton (S) had the highest number of passengers, with approximately 644 people boarding from there, followed by Cherbourg (C) with 168 passengers, and Queenstown (Q) with 77 passengers**. This analysis helps in understanding the distribution of passengers across different boarding locations, identifying the most common embarkation points, and detecting any potential missing values or anomalies in the dataset. Since most passengers embarked from Southampton, it suggests that the majority of the Titanic's journey originated from there, making it a significant boarding point in the dataset.



```
[11]:  Embarked
       S    644
       C    168
       Q     77
       Name: count, dtype: int64
```

df['Age'].max()

Interpretation:

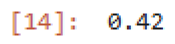
The command df['Age'].max() retrieves the maximum age from the Age attribute in the Titanic dataset. This function scans the entire attribute and returns the highest numerical value. Given the dataset's historical context, the oldest passenger on board was 80 years old. This result provides insight into the age distribution of passengers, highlighting that there were elderly individuals among the travelers. Analyzing this information can help in survival analysis, where age might have influenced the likelihood of survival. If there are missing values in the Age attribute, they will not affect the result since the .max() function automatically ignores NaN (null) values by default.

```
[13]:  80.0
```

df['Age'].min()

Interpretation:

The command df['Age'].min() retrieves the minimum age from the Age attribute in the Titanic dataset. This function scans the entire attribute and returns the smallest numerical value. Based on the dataset, the youngest passenger on board was 0.42 years old (approximately 5 months old), indicating that there were infants among the travelers. This information is useful for analyzing the age distribution of passengers, especially in survival analysis, as younger passengers, particularly children, might have had a higher priority for rescue. If there are missing values in the Age column, they do not affect the result, as the .min() function automatically ignores NaN (null) values by default.

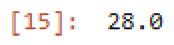


```
[14]:  0.42
```

df['Age'].median()

Interpretation:

The command df['Age'].median() calculates the median age of passengers in the Titanic dataset. The median is the middle value when all ages are arranged in ascending order, providing a measure of central tendency that is less affected by extreme values (outliers). Based on the dataset, the median age of passengers is 28 years, meaning that half of the passengers were younger than 28, and the other half were older. This is useful for understanding the age distribution of passengers and is often used to fill in missing values when handling incomplete data. Since the median is not affected by outliers, it provides a more robust measure of central tendency compared to the mean.

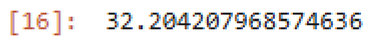


```
[15]:  28.0
```

df['Fare'].mean()

Interpretation:

The command df['Fare'].mean() calculates the average fare price paid by passengers in the Titanic dataset. The mean fare represents the sum of all fare values divided by the total number of passengers. Based on the dataset, the average fare is approximately 32.20. However, since fares varied significantly depending on class (1st, 2nd, and 3rd), this mean value may be influenced by high-priced first-class tickets, making the distribution right-skewed. To better understand the fare structure, it is useful to analyze the median fare, interquartile range, and outliers.



```
[16]:  32.204207968574636
```

df['Fare'].sum()

Interpretation:

The command df['Fare'].sum() calculates the total fare revenue from all passengers in the Titanic dataset by summing up all values in the Fare attribute. Based on the dataset, the total sum of fares is approximately 28,693.9493 This value represents the combined amount paid by all 891 passengers for their tickets. Since fares varied significantly based on passenger class, the sum includes a mix of high-cost first-class tickets and lower-cost third-class tickets. This metric is useful for understanding the overall revenue generated from ticket sales and analyzing the financial aspect of passenger distribution across different classes.

```
[17]:  28693.9493
```

import statistics as st
st.mode(df['Pclass'])

Interpretation:

The command st.mode(df['Pclass']) calculates the mode of the Pclass attribute in the Titanic dataset. The mode is the most frequently occurring value in a dataset. Since Pclass represents the passenger class (1st, 2nd, or 3rd), the mode helps determine which class had the highest number of passengers.Based on the dataset, the mode of Pclass is 3, meaning the majority of passengers traveled in 3rd class. This suggests that a significant portion of Titanic's passengers belonged to a lower economic class, as third-class tickets were the most affordable. Analyzing this information is useful for understanding passenger demographics and survival rate trends, as survival chances varied across different classes.

```
[18]:  3
```

print("Skewness: %f"% df['Age'].skew())

Interpretation:

The command print("Skewness: %f" % df['Age'].skew()) calculates and prints the **skewness** of the Age attribute in the Titanic dataset. **Skewness** measures the **asymmetry** of the data distribution:

- **Skewness > 0** → Right-skewed (positively skewed): More values are concentrated on the left, with a long tail on the right.
- **Skewness < 0** → Left-skewed (negatively skewed): More values are concentrated on the right, with a long tail on the left.

- **Skewness = 0** → Symmetrical distribution.

For the Titanic dataset, the **skewness of the Age attribute is approximately 0.38**, indicating a **slightly right-skewed distribution**. This means that **most passengers were younger, but a few older passengers (such as those above 60) create a longer right tail**. Understanding skewness helps in **data normalization, handling outliers, and making statistical inferences** about passenger demographics.

```
Skewness: 0.389108
```

print("Kurtosis: %f"% df['Age'].kurt())

Interpretation:

The command print("Kurtosis: %f" % df['Age'].kurt()) calculates and prints the kurtosis of the Age attribute in the Titanic dataset. Kurtosis measures the tailedness of the data distribution:

- Kurtosis > 0 (Leptokurtic) → Distribution has heavier tails than a normal distribution, meaning more extreme values (outliers).
- Kurtosis < 0 (Platykurtic) → Distribution has lighter tails, meaning fewer extreme values.
- Kurtosis = 0 (Mesokurtic) → The distribution is similar to a normal distribution.

The kurtosis of the Age attribute in the Titanic dataset is 0.17, indicating a distribution that is close to normal but slightly leptokurtic (having slightly heavier tails). This means that while the distribution is not extreme, it has a few more outliers (extreme age values) than a perfectly normal distribution.A kurtosis value of 0.17 suggests that there are some older passengers (e.g., above 60 or 70 years old) or very young infants that slightly increase the tail weight. However, the difference is not significant, meaning the age distribution is fairly normal with only a slight presence of extreme values. Understanding kurtosis helps in outlier detection, statistical modeling, an deciding whether data transformations are necessary.

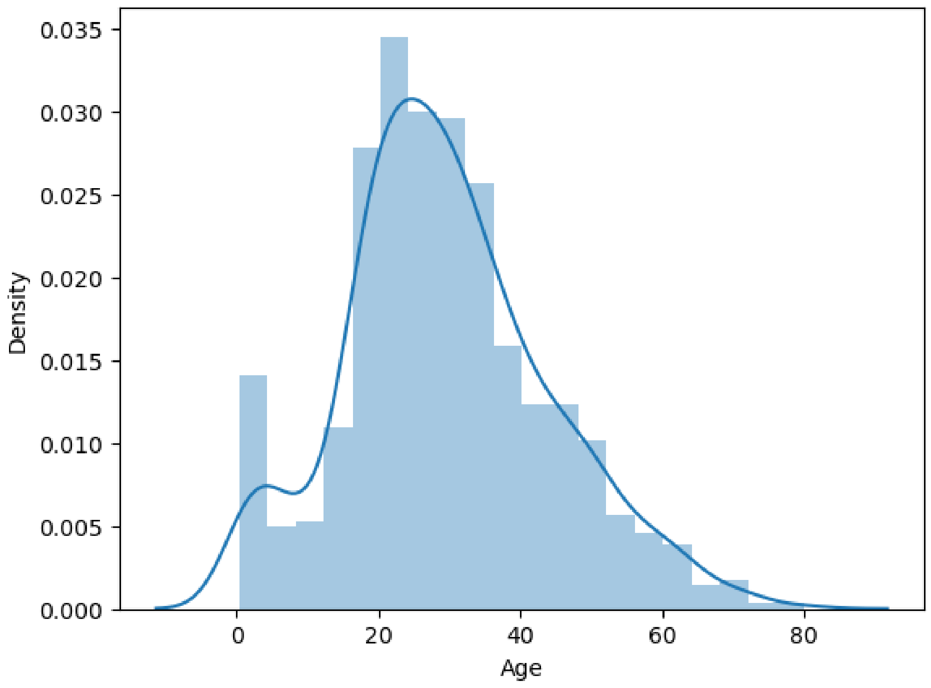
```
Kurtosis: 0.178274
```

import seaborn as sns
sns.distplot(df['Age'])

Interpretation:

The distribution plot of the Age attribute in the Titanic dataset, generated by sns.distplot(df['Age']), forms a bell-shaped curve, indicating a nearly normal distribution. The histogram bars represent the frequency of different age groups, while the Kernel Density Estimate (KDE) curve provides a smooth approximation of the age distribution. The peak around 20-40 years suggests that most passengers were young adults, while the slight right skewness (0.38) indicates a few older passengers (above 60 or 70 years) extending the tail on the right. Despite this minor skew, the distribution is approximately symmetrical, resembling a Gaussian (normal) distribution. This bell-shaped pattern suggests that most values cluster around the mean, with fewer passengers at the extreme age ranges. Such a distribution is useful in statistical analysis, as many machine learning algorithms assume normality in data for better performance and inference.

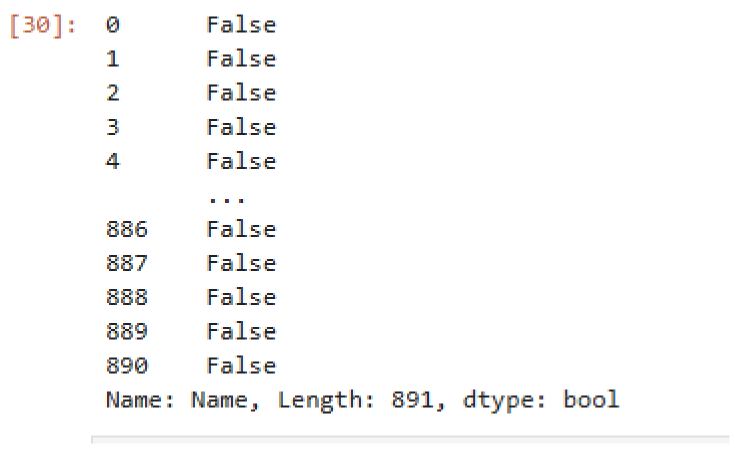
```
[25]:  <Axes: xlabel='Age', ylabel='Density'>
```



bf=df["Name"].duplicated()
df.drop_duplicates(keep=False,inplace=True)
bf

Interpretation:

The code first identifies duplicate names in the Titanic dataset using df["Name"].duplicated(), which returns a Boolean series where True indicates repeated names and False represents unique ones. This Boolean series is stored in bf. Then, df.drop_duplicates(keep=False, inplace=True) completely removes all occurrences of duplicate names, ensuring that only passengers with unique names remain in the dataset. Since the Titanic dataset contains **891 records** with attributes such as PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, and Embarked, this operation could significantly reduce the dataset size if multiple passengers share the same name. The bf output helps in identifying which names were marked as duplicates before removal. While this method effectively cleans the dataset, caution is required since some duplicate names might belong to different individuals rather than being actual data-entry errors.



```
[30]: 0      False
      1      False
      2      False
      3      False
      4      False
             ...
      886    False
      887    False
      888    False
      889    False
      890    False
      Name: Name, Length: 891, dtype: bool
```

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

columns = ['Age', 'Fare', 'Pclass']
for each in columns:
   fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(13, 5))

   # Box plot
   sns.boxplot(x=df[each], orient='v', ax=ax1)

   # Histogram
   ax2.hist(df[each])

   # Distplot
   sns.distplot(df[each], ax=ax3)

   plt.show()
```

Interpretation:

This code uses **Seaborn** and **Matplotlib** to generate three different visualizations (**box plot, histogram, and distribution plot**) for each of the selected numerical attributes —**Age, Fare, and Pclass**—from the Titanic dataset.

Breakdown of the Code:

1. The list columns = ['Age', 'Fare', 'Pclass'] specifies the attributes for visualization.

2. A for loop iterates over each column, creating three subplots using plt.subplots(1, 3, figsize=(13, 5)), arranging them **horizontally in a single row with three plots per figure**.
3. **Box Plot (ax1)**: Created using sns.boxplot(), it shows the **median, quartiles, and potential outliers** for each column.
4. **Histogram (ax2)**: Created using ax2.hist(), it displays the **frequency distribution** of the values in the column.
5. **Distribution Plot (ax3)**: Created using sns.distplot(), it provides a **smooth KDE curve** over the histogram to visualize the **probability distribution** of values.
6. plt.show() ensures that each set of plots is displayed before moving to the next column.

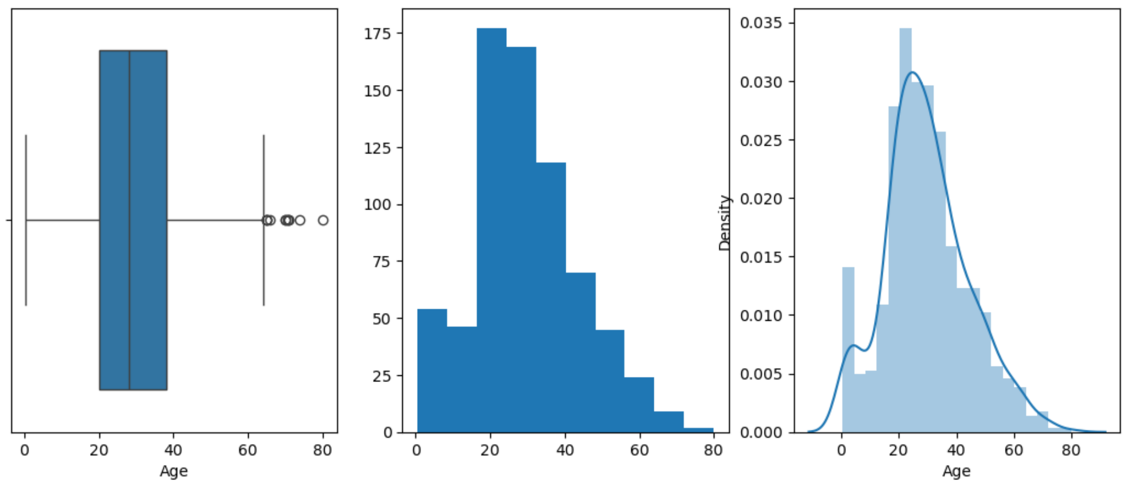
Interpretation of Each Plot:

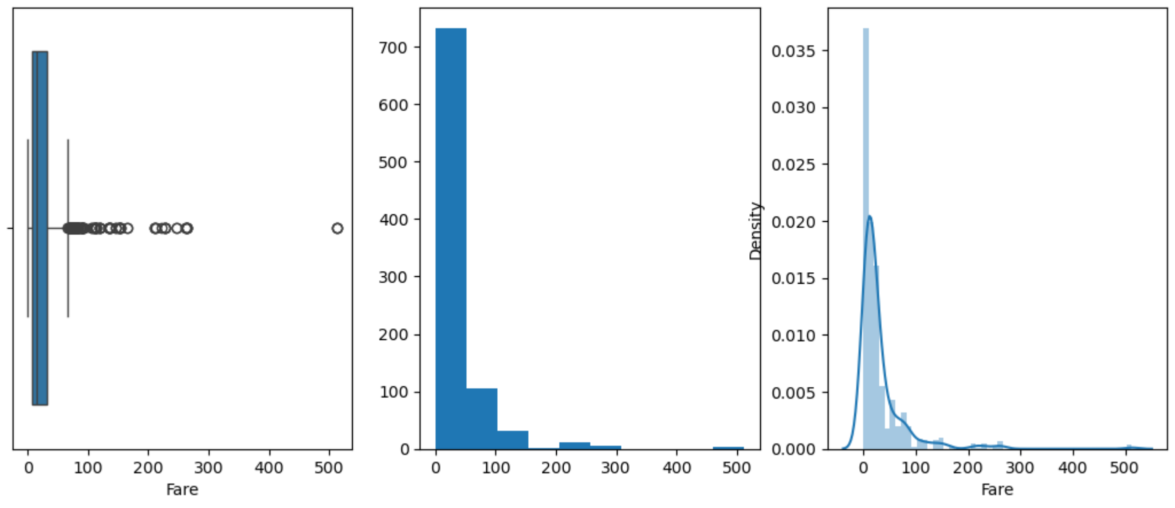
- **Box Plot:**
  - Helps detect **outliers** (e.g., Fare has a long right tail with extreme values).
  - Shows **median and interquartile range (IQR)**, which is useful for spotting skewness in data.
- **Histogram:**
  - Displays **frequency distributions** (e.g., Pclass is categorical, so bars will be discrete).
  - Helps in understanding **skewness and spread** of data.
- **Distribution Plot:**
  - Shows a **smooth KDE curve** over the histogram, making it easier to see **normality and skewness**.
  - If the **curve is bell-shaped**, the data is **normally distributed** (e.g., Age is nearly normal).
  - If there is a long tail, it suggests **skewness** (e.g., Fare has a right-skewed distribution).

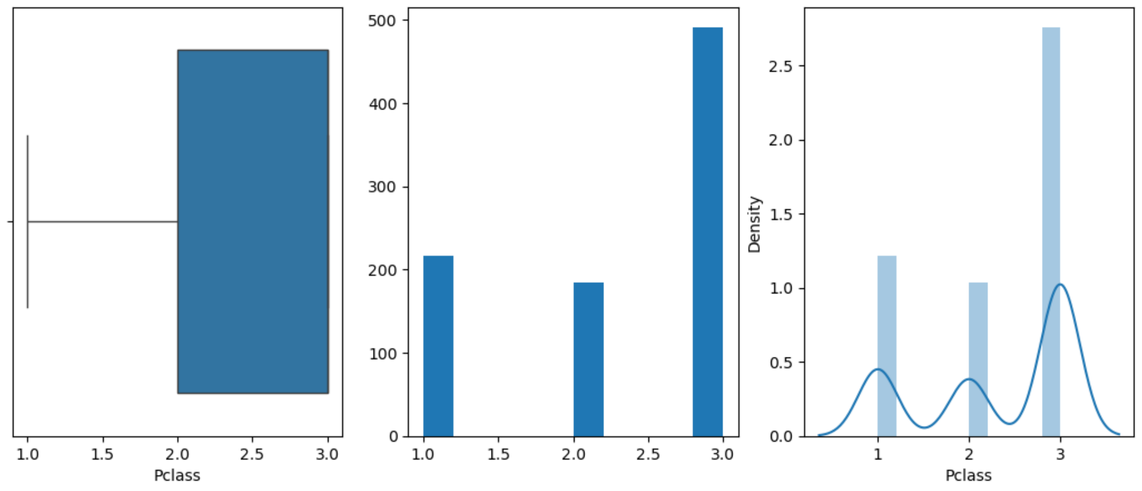
Possible Observations from the Titanic Dataset:

- **Age appears to be nearly normal**, with a slight right skew due to older passengers and it has outliers.
- **Fare is highly right-skewed**, meaning a few passengers paid significantly higher fares than the majority and it has outliers .
- **Pclass has distinct bars in the histogram**, as it is a categorical variable (1st, 2nd, and 3rd class).

This visualization helps in **understanding data distribution, identifying outliers, and preparing data for further analysis or machine learning models**.







```
import seaborn as sns
sns.boxplot(x='Survived',y='Age',data=df)
plt.show()
```

Interpretation:

This code uses **Seaborn** to create a **box plot** that visualizes the relationship between **survival status (Survived)** and **age (Age)** in the Titanic dataset. The **sns.boxplot(x='Survived', y='Age', data=df)** function plots Survived on the x-axis and Age on the y-axis, allowing us to compare the age distribution of passengers who **survived (Survived = 1)** versus those who **did not (Survived = 0)**.

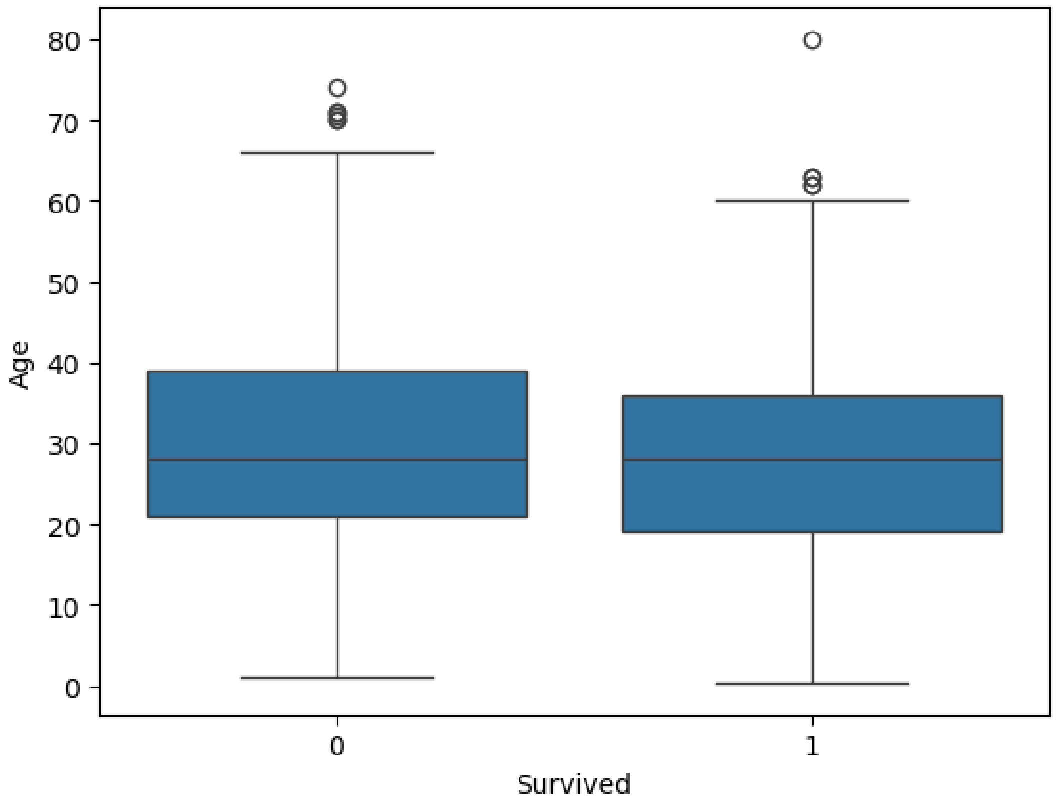
Interpretation of the Box Plot:

- Each box represents the **interquartile range (IQR)** (middle 50% of the data).
- The **median age** for both survived and non-survived groups is visible as a horizontal line inside each box.
- **Outliers**, such as very young or very old passengers, appear as individual points beyond the whiskers.
- If the box for Survived = 1 is higher than Survived = 0, it indicates **age might have influenced survival rates**.

Possible Observations from Titanic Data:

- **Younger passengers (children) might have a higher survival rate**, as seen from a lower median age for survivors.
- **Older passengers (above 50) are relatively fewer among survivors**, suggesting they had a lower survival rate.
- **The spread of age is wider in non-survivors**, implying a more diverse age range among those who did not survive.

This visualization helps in understanding whether **age had an impact on survival chances** and is useful for further exploratory data analysis.



```
import seaborn as sns
sns.boxplot(x='Sex',y='Age',data=df)
plt.show()
```

Interpretation:

This code uses Seaborn to create a box plot that visualizes the age distribution across different sexes (Male and Female) in the Titanic dataset. The function sns.boxplot(x='Sex', y='Age', data=df) plots Sex on the x-axis and Age on the y-axis, allowing us to compare the age distribution of male and female passengers. The box plot provides insights into the median age, interquartile range (IQR), and outliers for each gender.

Interpretation of the Box Plot:

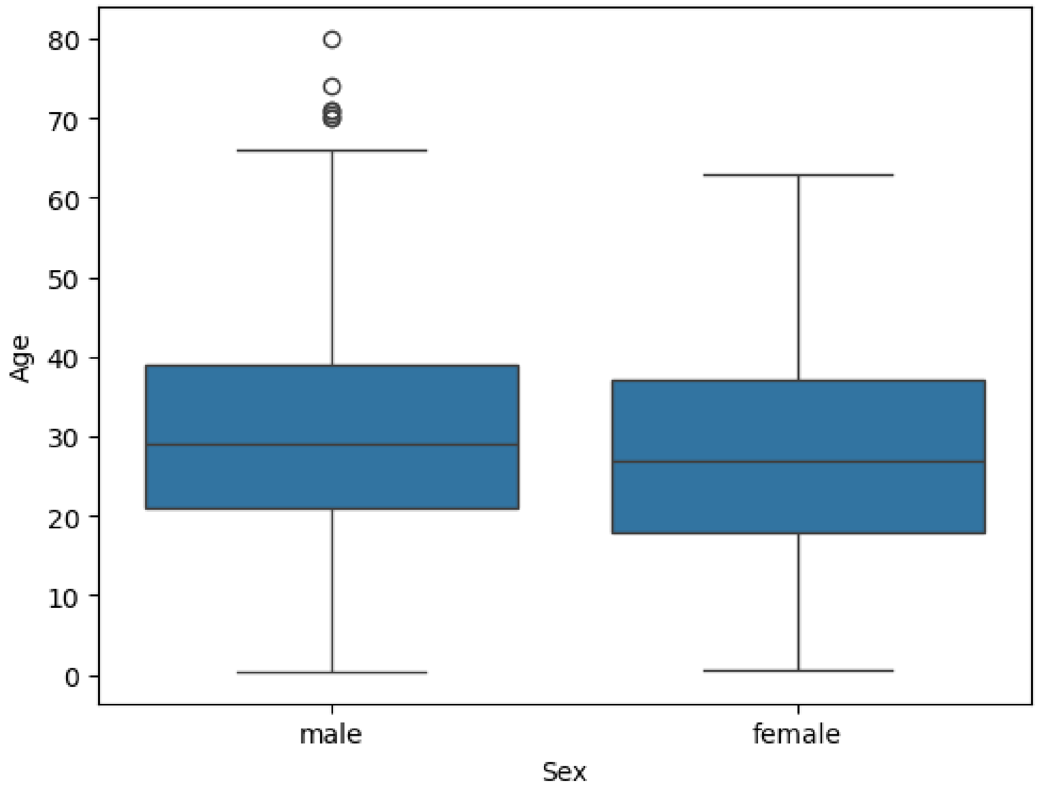
- The median age (horizontal line inside each box) for males and females can be compared.
- The interquartile range (IQR) shows the middle 50% of ages for each gender.
- Outliers, such as very young or very old passengers, appear as individual points beyond the whiskers.

Possible Observations from Titanic Data:

- The median age of male and female passengers appears similar, likely around 28-30 years.

- The spread of ages is slightly larger for males, suggesting a broader age distribution among male passengers.
- There are more visible outliers among male passengers, indicating the presence of older men or young boys.
- Female passengers may have a slightly lower IQR, suggesting their ages are more concentrated around the median.

This visualization helps analyze the age distribution by gender and can be used for further statistical comparisons or feature engineering in machine learning models.



import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv('train.csv')
sns.violinplot(x='Sex',y='Age',data=df,color='y')
plt.show()

Interpretation:

This code uses **Seaborn** to create a **violin plot** that visualizes the distribution of **age across different sexes (Male and Female)** in the Titanic dataset. The function **sns.violinplot(x='Sex', y='Age', data=df, color='y')** plots Sex on the x-axis and Age on the y-axis, with a violin-shaped plot that shows the

distribution, density, and probability of different age values for each gender. The color='y' parameter sets the violin plot to **yellow**.

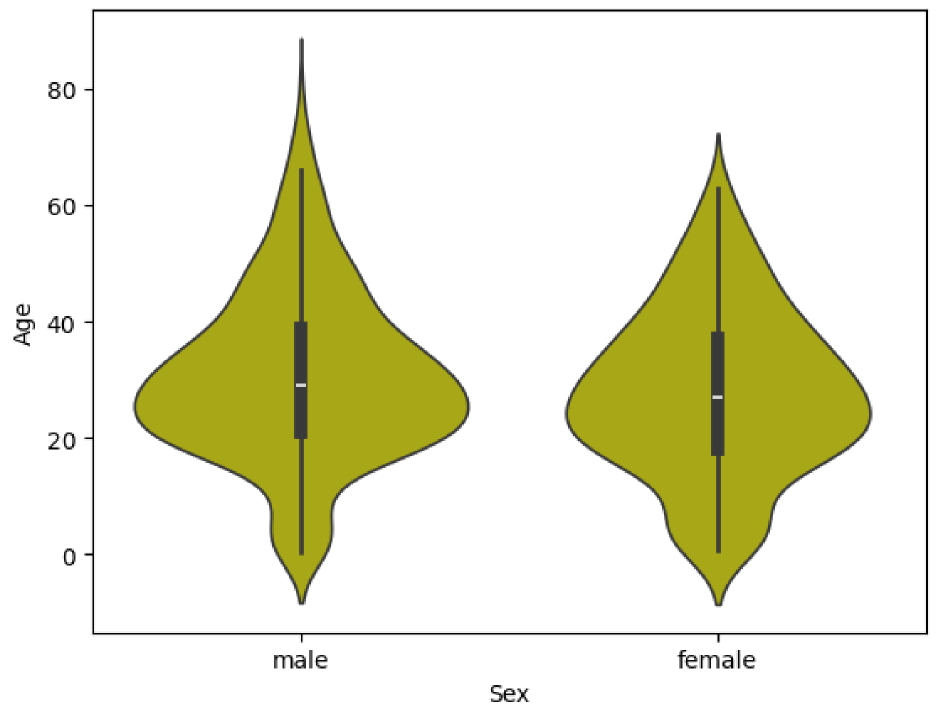
Interpretation of the Violin Plot:

- The **width of the violin** at different ages represents the **density** (how many passengers fall into that age range).
- The **thicker regions** indicate where most of the data points are concentrated.
- The **median and interquartile range (IQR)** are visible as white lines inside each violin.
- The **upper and lower tails** extend toward outliers, showing the range of ages for both genders.

Possible Observations from Titanic Data:

- **Males have a wider distribution of ages**, meaning their ages are more spread out compared to females.
- **Females have a higher concentration of younger passengers**, as the density is more compact in the lower age range.
- **There are more young male passengers**, as indicated by the greater width in the lower age range.
- **The age distribution for females appears more symmetric**, while males show more variation.

This violin plot provides **a more detailed view than a box plot** by showing both the **distribution shape and density**, helping in better understanding age patterns among male and female passengers.



data = pd.concat([df['Age'], df['Fare']], axis=1)
data.plot.scatter(x='Age', y='Fare', ylim=(0,200));

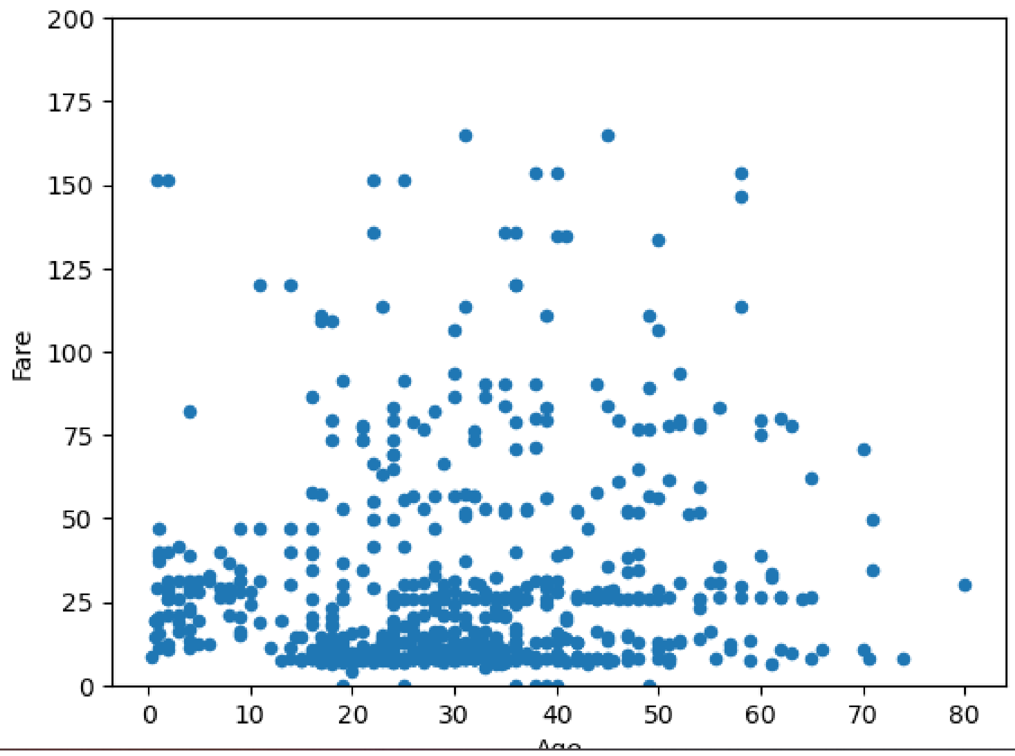
Interpretation:

This code creates a **scatter plot** to visualize the relationship between **Age** and **Fare** in the Titanic dataset. The pd.concat() function combines the Age and Fare columns, and the plot.scatter(x='Age', y='Fare', ylim=(0,200)) method generates a scatter plot with Age on the x-axis and Fare on the y-axis. The ylim=(0,200) parameter restricts the fare values to a range between **0 and 200** to remove extremely high fare values that could skew the visualization.

Interpretation of the Scatter Plot:

- **Each point represents a passenger**, with their age and fare plotted on the respective axes.
- **Passengers below 20 years** generally paid lower fares, as seen in the cluster of points in the lower-left region.
- **There is a wide variation in fares among older passengers**, with some paying significantly higher fares.
- **A few outliers can be seen where older passengers paid very high fares**, indicating they might have traveled in first class.
- **No strong linear relationship** between age and fare is observed, suggesting that ticket price was more influenced by class rather than age.

This scatter plot helps identify **trends, outliers, and clusters** within the data, providing insights into how fare prices varied across different age groups.



import matplotlib.pyplot as plt
import seaborn as sns

```
# Select only the numeric columns
numeric_df = df.select_dtypes(include=['number'])
# Correlation matrix
corrmat = numeric_df.corr()
# Plot the heatmap
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=1, square=True)
plt.show()
```

Interpretation:

This code generates a **heatmap** to visualize the correlation between **numeric variables** in the Titanic dataset. The function df.select_dtypes(include=['number']) filters only the **numeric columns**, and numeric_df.corr() computes the **correlation matrix**, which measures how strongly two variables are related. The **Seaborn heatmap** is then plotted using sns.heatmap(corrmat, vmax=1, square=True), where:

- vmax=1 ensures that the color scale ranges from **-1 to 1** (full correlation range).
- square=True makes the heatmap **square-shaped** for better readability.
- f, ax = plt.subplots(figsize=(12,9)) sets the figure size for clarity.

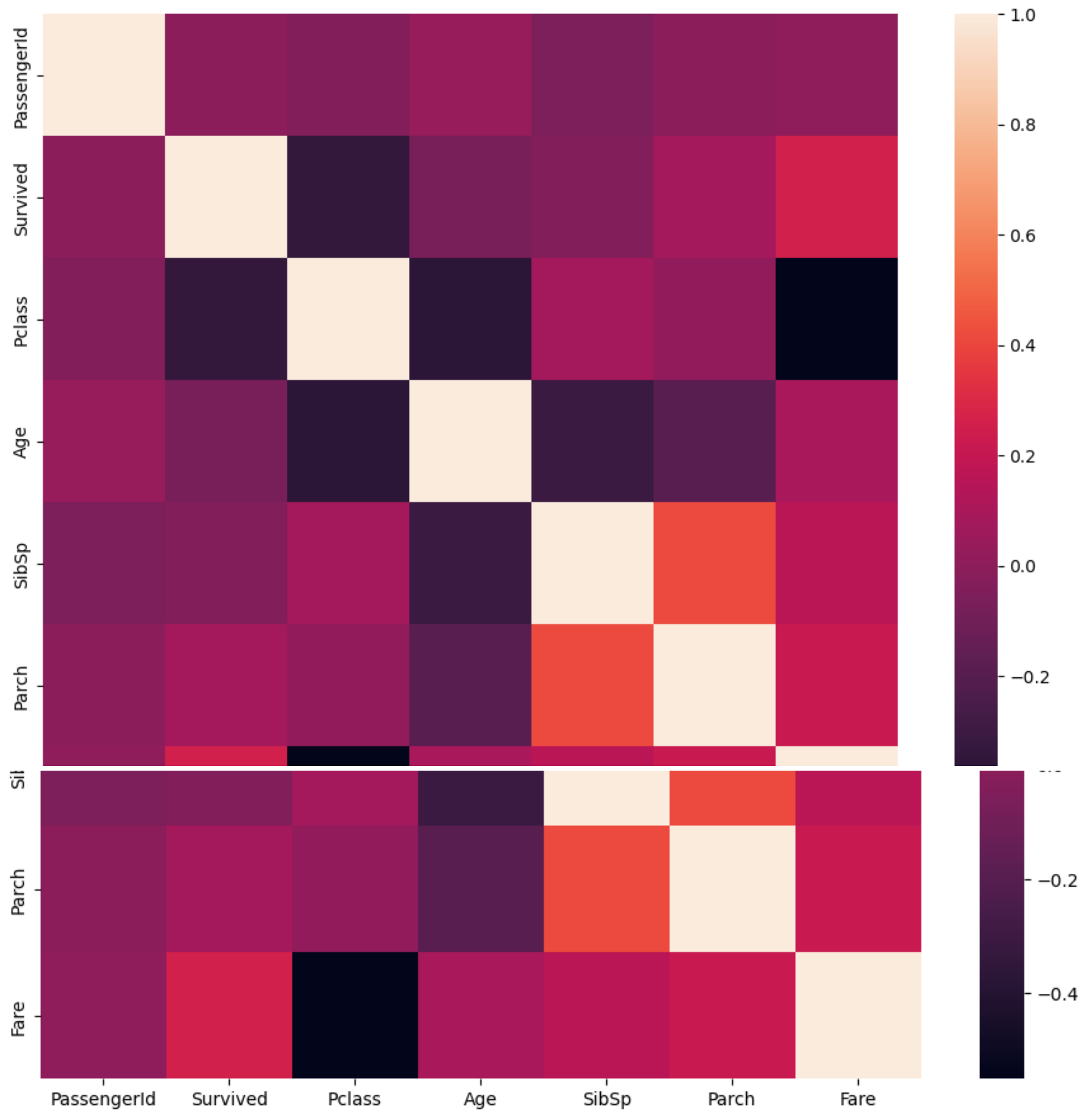
Interpretation of the Heatmap:

- **Values close to +1 (light colors)** indicate a **strong positive correlation** (e.g., if one variable increases, the other also increases).
- **Values close to -1 (dark colors)** indicate a **strong negative correlation** (e.g., if one variable increases, the other decreases).
- **Values around 0 (neutral colors)** indicate **no significant correlation**.

Possible Observations from Titanic Data:

- **Fare and Pclass show a strong negative correlation**, meaning higher-class passengers (Pclass=1) generally paid more.
- **Survival (Survived) is positively correlated with Fare**, suggesting that passengers who paid higher fares had a better chance of survival.
- **Age has little to no correlation with Survived**, indicating age alone was not a strong determinant of survival.
- **SibSp (Siblings/Spouses Aboard) and Parch (Parents/Children Aboard) are positively correlated**, meaning families tended to travel together.

This heatmap is a powerful tool for **feature selection and analysis** in machine learning, helping identify key relationships between variables in the Titanic dataset.



numeric_df.corr()

Interpretation:

The function **numeric_df.corr()** computes the **correlation matrix** for all **numeric columns** in the Titanic dataset. Correlation values range from **-1 to 1**, where:

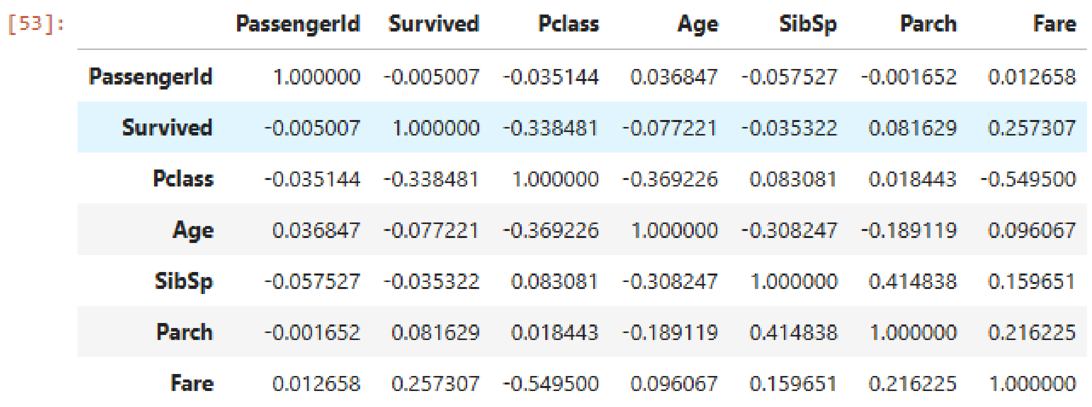
- **+1** indicates a **strong positive correlation** (as one variable increases, the other also increases).
- **-1** indicates a **strong negative correlation** (as one variable increases, the other decreases).
- **0** indicates **no correlation** between the variables.

Key Insights from Correlation Matrix:

- **Fare and Pclass are strongly negatively correlated (-0.55)**, meaning first-class passengers (Pclass=1) generally paid more.

- **Survival is positively correlated with Fare (0.26)**, suggesting passengers who paid higher fares had better chances of survival.
- **Pclass and Age have a negative correlation (-0.37)**, meaning older passengers were more likely to be in lower classes.
- **SibSp and Parch have a positive correlation (0.41)**, indicating that families traveled together.
- **Survived has a weak correlation with Age (-0.08)**, meaning age alone was not a major factor in survival.

This correlation matrix helps in **feature selection and analysis** for predictive modeling in machine learning.



[53]:

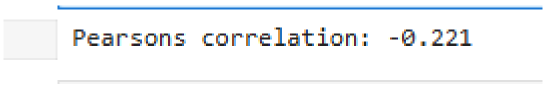
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

```
import pandas as pd
from sklearn import preprocessing
from scipy.stats import pearsonr

# Convert 'Embarked' column to numeric values using LabelEncoder
label_encoder = preprocessing.LabelEncoder()
df['Embarked'] = label_encoder.fit_transform(df['Embarked'])

# Extract the numeric columns for correlation
list1 = df['Fare']
list2 = df['Embarked']

# Apply the pearsonr() function
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

Interpretation:

The Pearson correlation coefficient between **Fare** and **Embarked** is **-0.221**, indicating a **weak negative correlation**. This means that as **Fare increases, the Embarked value tends to decrease slightly**. Since the LabelEncoder converted the **'Embarked'** column into numerical values (0, 1, 2 for

Cherbourg (C), Queenstown (Q), and Southampton (S), respectively), the negative correlation suggests that **passengers who embarked from ports with lower numerical values (e.g., Cherbourg)** generally **paid higher fares**, whereas those from **Southampton or Queenstown paid lower fares on average**. This aligns with historical passenger data, where Cherbourg had more **first-class passengers**, leading to **higher ticket fares**, while **Southampton, the primary departure point, had many third-class passengers, resulting in lower fares**. Although the correlation is not strong, it provides insights into how **ticket prices varied based on embarkation locations**.



```
Pearsons correlation: -0.221
```

sns.pairplot(df,hue="Survived",size=2)
plt.show()

Interpretation:

This code generates a **pairplot** using the **Seaborn** library to visualize relationships between numerical features in the dataset, with different colors based on the **'Survived'** attribute.

Code Breakdown:

1. sns.pairplot(df, hue="Survived", size=2):
    - Creates scatter plots for **each pair of numerical attributes** in the dataset.
    - Uses **'Survived'** as the hue to differentiate between **survivors (1) and non-survivors (0)** using color.
    - The diagonal plots will display **histograms** of individual features.
2. plt.show():
    - Displays the generated **pairplot**.

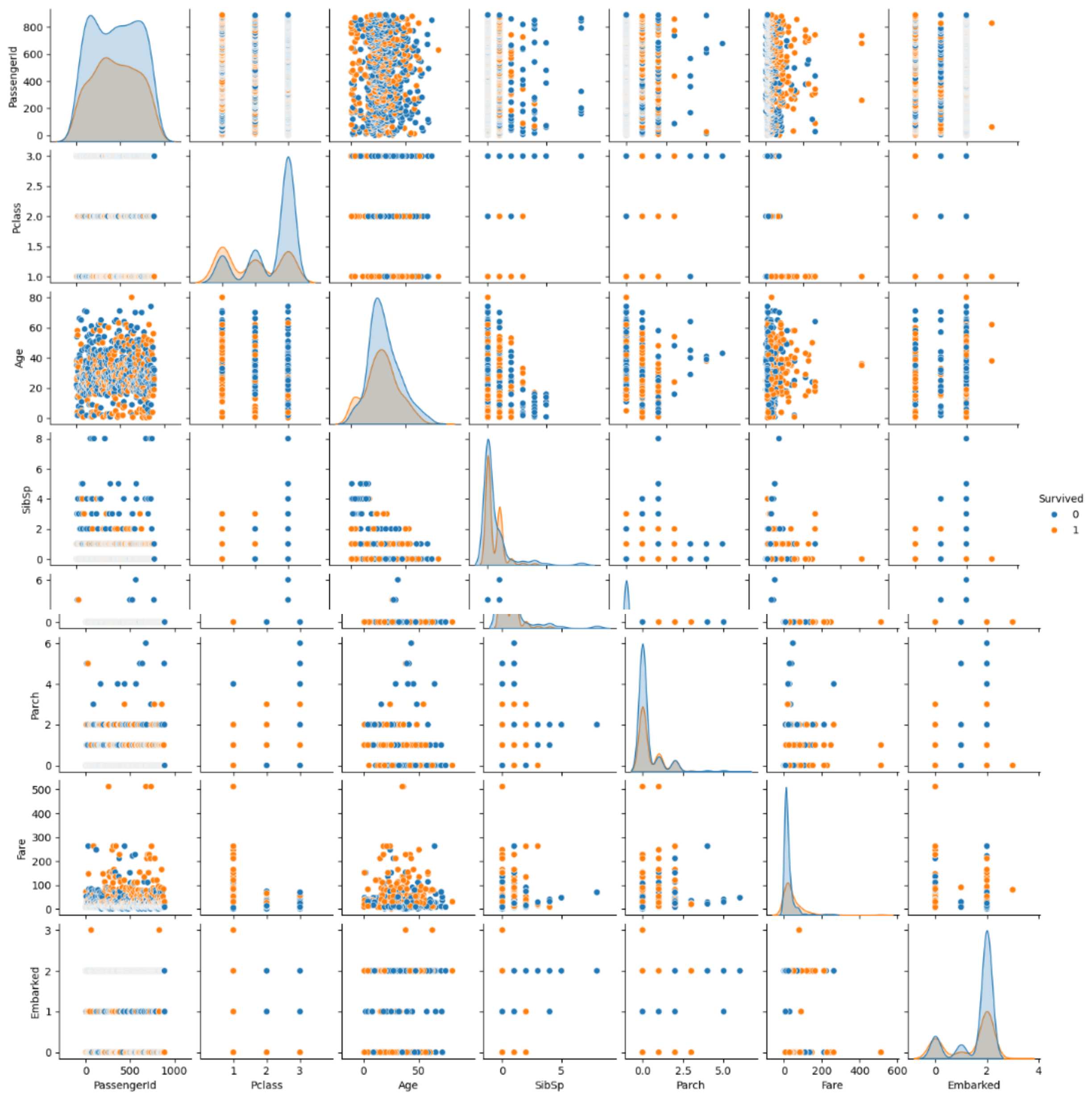
Interpretation:

- **Patterns between numerical features**:
    - Helps identify **correlations** between survival and other attributes like **Age, Fare, Pclass, and SibSp**.
    - For example, higher **Fare values** might be more associated with survivors, indicating that **first-class passengers had higher survival rates**.
- **Clusters and distributions**:
    - Different clusters in scatter plots can **reveal trends** (e.g., younger passengers in first class might have had better survival chances).

Expected Insights:

- **Higher fares** might correspond to a **higher survival rate**.
- **Passengers in lower Pclass** (3rd class) might show a **higher density of non-survivors**.
- **Age distributions** could indicate whether **younger passengers (e.g., children) had better survival rates**.

This visualization is useful for **exploratory data analysis (EDA)** and helps in **feature selection** for predictive modeling.



```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import uniform
train_df = pd.read_csv('train.csv')
column_name = 'Pclass'
data = train_df[column_name]
uniform_data = uniform.rvs(size=len(data), loc=min(data), scale=max(data) - min(data))
train_df['uniform_'+column_name] = uniform_data
sns.histplot(uniform_data, bins=50, kde=True, color='skyblue', alpha=0.7)
plt.xlabel('Uniform Distribution')
plt.ylabel('Frequency')
plt.title('Uniform Distribution for '+column_name)
plt.show()
```

Interpretation:

This code generates a **uniform distribution** based on the values of the 'Pclass'attribute in the Titanic dataset and visualizes it using a **histogram with a KDE (Kernel Density Estimate) curve**.

Code Breakdown & Execution Flow:

1. **Loading the Dataset**
    - train_df = pd.read_csv('train.csv')
    - Reads the Titanic dataset into a **Pandas DataFrame**.
2. **Extracting 'Pclass' Data**
    - column_name = 'Pclass'
    - data = train_df[column_name]
    - Extracts the **'Pclass'** attribute for processing.
3. **Generating a Uniform Distribution**
    - uniform.rvs(size=len(data), loc=min(data), scale=max(data) - min(data))
    - Uses the **Scipy uniform.rvs() function** to generate random values from a **uniform distribution**.
    - size=len(data): Ensures the number of generated values matches the dataset.
    - loc=min(data): The minimum value in 'Pclass' is set as the **starting point**.
    - scale=max(data) - min(data): The range is determined based on 'Pclass'.

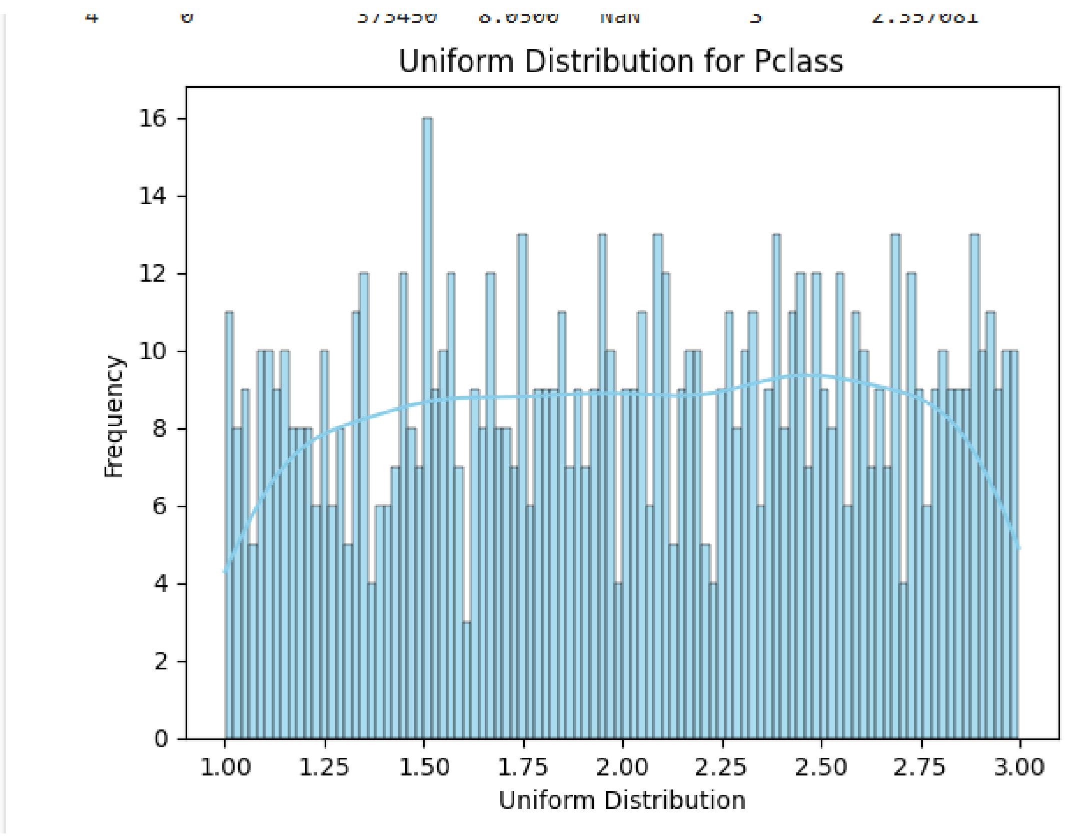
4. **Adding the New Uniformly Distributed Data to the DataFrame**
   - train_df['uniform_'+column_name] = uniform_data
   - Stores the generated **uniform distribution values** in a new attribute named 'uniform_Pclass'.
5. **Plotting the Histogram & KDE**
   - sns.histplot(uniform_data, bins=50, kde=True, color='skyblue', alpha=0.7)
   - Plots a histogram with **50 bins** to visualize the uniform distribution.
   - kde=True: Adds a **density curve** for smooth visualization.
   - color='skyblue', alpha=0.7: Sets the color & transparency for aesthetics.
6. **Setting Labels & Title**
   - Adds appropriate labels and a title:
     - plt.xlabel('Uniform Distribution')
     - plt.ylabel('Frequency')
     - plt.title('Uniform Distribution for Pclass')
7. **Displaying the Plot**
   - plt.show()

Interpretation of the Plot:

- **Uniform Distribution** means **each class (1st, 2nd, 3rd) has an equal probability** of being selected within the defined range.
- The **histogram** should show a **flat (or nearly flat) distribution**, where **all values are equally likely**.
- The KDE curve helps verify if the generated **random values** are **spread evenly** across the range.

Use Case:

- This is useful in **simulation studies** to create synthetic datasets.
- Helps compare **real Titanic passenger distribution vs. randomly generated uniform data**.



```
4        0          373450    8.0500    NaN         3        2.337081
```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
train_df = pd.read_csv('train.csv')
column_name = 'PassengerId'
data = train_df[column_name]
mean, std_dev = data.mean(), data.std()
normal_data = np.random.normal(mean, std_dev, len(data))
train_df['normal_'+column_name] = normal_data
sns.histplot(normal_data, bins=50, kde=True, color='blue', alpha=0.7)
plt.xlabel('Normal Distribution')
plt.ylabel('Frequency')
plt.title('Normal Distribution for '+column_name)
plt.show()

Interpretation:

This code generates a normal (Gaussian) distribution based on the 'PassengerId' attribute in the Titanic dataset and visualizes it using a histogram with a KDE (Kernel Density Estimate) curve.

Code Breakdown & Execution Flow:

1. **Loading the Dataset**
   - **train_df = pd.read_csv('train.csv')**
   - **Reads the Titanic dataset into a Pandas DataFrame.**
2. **Extracting 'PassengerId' Data**
   - **column_name = 'PassengerId'**
   - **data = train_df[column_name]**
   - **Extracts the 'PassengerId' attribute for processing.**
3. **Calculating Mean & Standard Deviation**
   - **mean, std_dev = data.mean(), data.std()**
   - **Computes the mean (μ) and standard deviation (σ) of 'PassengerId'.**
4. **Generating Normally Distributed Data**
   - **normal_data = np.random.normal(mean, std_dev, len(data))**
   - **Uses np.random.normal() to generate normally distributed random values:**
     - **Mean (μ) = Computed from 'PassengerId'**
     - **Standard deviation (σ) = Computed from 'PassengerId'**
     - **Size = Matches the number of rows in the dataset**
5. **Adding the Generated Normal Data to the DataFrame**
   - **train_df['normal_'+column_name] = normal_data**
   - **Stores the new normally distributed values in a new column 'normal_PassengerId'.**
6. **Plotting the Histogram & KDE**
   - **sns.histplot(normal_data, bins=50, kde=True, color='blue', alpha=0.7)**
   - **Plots a histogram with 50 bins to visualize the normal distribution.**
   - **kde=True: Adds a density curve for smooth visualization.**
   - **color='blue', alpha=0.7: Sets the color & transparency for aesthetics.**
7. **Setting Labels & Title**
   - **Adds appropriate labels and a title:**
     - **plt.xlabel('Normal Distribution')**
     - **plt.ylabel('Frequency')**
     - **plt.title('Normal Distribution for PassengerId')**
8. **Displaying the Plot**
   - **plt.show()**

Interpretation of the Plot:

- **Normal Distribution (Bell Curve): The generated values should follow a symmetric bell-shaped curve centered around the mean PassengerId.**
- **Standard Deviation (σ): Determines how spread out the values are.**
- **Histogram: Shows frequencies of different values in the generated data.**
- **KDE Curve: Helps visualize the density of the distribution.**

Use Case:

Code Breakdown & Execution Flow:

1. **Loading the Dataset**
   - train_df = pd.read_csv('train.csv')
   - Reads the Titanic dataset into a **Pandas DataFrame**.
2. **Extracting 'Survived' Data**
   - column_name = 'Survived'
   - data = train_df[column_name]
   - Extracts the **'Survived'** attribute, which is binary (0 = Did not survive, 1 = Survived).
3. **Calculating Poisson Parameter (λ - Lambda)**
   - lambda_param = data.mean()
   - Computes the **mean survival rate**, which represents the Poisson λ **(lambda)** parameter.
4. **Generating Poisson-Distributed Data**
   - poisson_data = np.random.poisson(lam=lambda_param, size=len(data))
   - Uses np.random.poisson() to generate **Poisson-distributed random values**, where:
     - **Lambda (λ)** = Mean survival rate (probability of survival)
     - **Size** = Matches the number of passengers in the dataset.
5. **Storing Poisson Data in DataFrame**
   - train_df['poisson_'+column_name] = poisson_data
   - Stores the **generated Poisson data** in a new attribute 'poisson_Survived'.
6. **Plotting the Histogram**
   - plt.hist(data, bins=30, alpha=0.5, label='Original Data')
   - plt.hist(poisson_data, bins=30, alpha=0.5, label='Poisson Distribution')
   - Plots **two histograms**:
     - One for the **original 'Survived' attribute** .
     - One for the **Poisson-generated data**.
   - **Alpha (0.5)** ensures transparency for overlapping visualization.
7. **Adding Labels & Title**
   - plt.legend() → Adds a legend to differentiate between original and Poisson data.
   - plt.title('Poisson Distribution for Survived') → Titles the plot.
   - plt.xlabel('Values') → X-axis represents survival values.
   - plt.ylabel('Frequency') → Y-axis represents the frequency of occurrences.
8. **Displaying the Plot**
   - plt.show()

Interpretation of the Plot:

1. Poisson Distribution Overview:

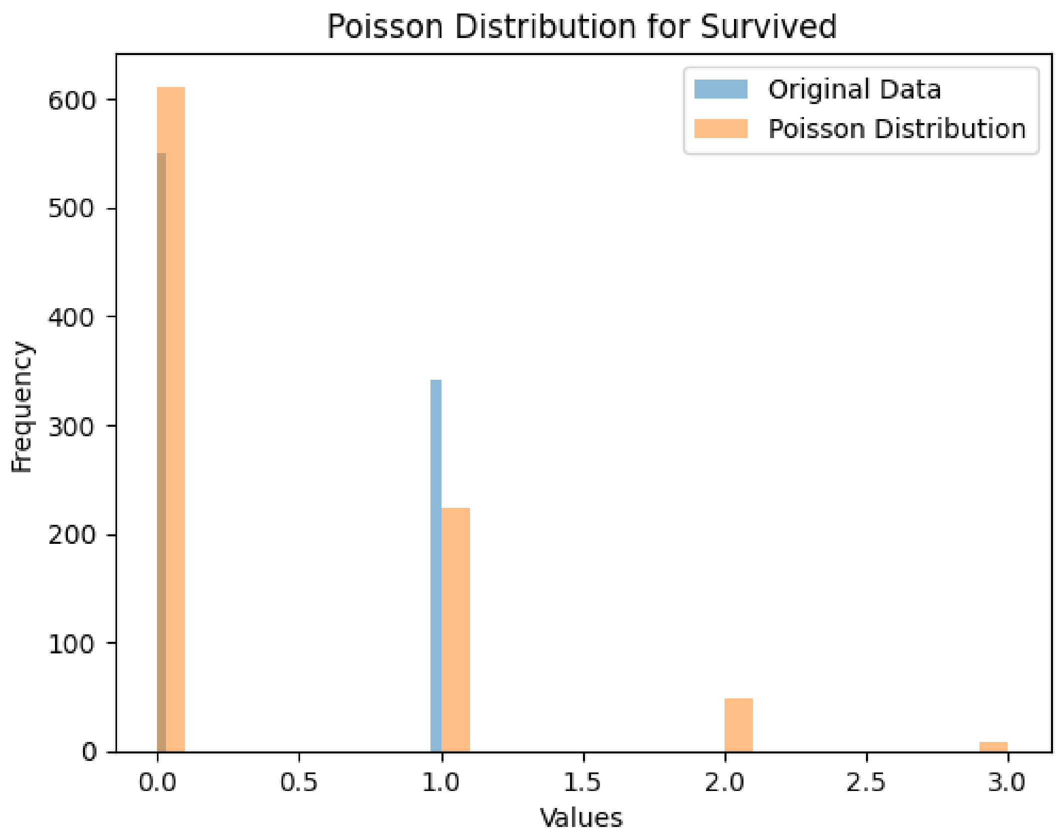
- A **Poisson distribution** models the **probability of a given number of events occurring in a fixed interval** (e.g., number of survivors per group of Titanic passengers).
- It is **discrete** and is commonly used for **count-based data**.

2. Comparison with Original Data:

- The **original 'Survived' column** contains only **0 or 1** values.
- The **Poisson distribution models event occurrences** and **may generate values >1**, though rare.
- **Poisson data closely follows the mean survival rate** but shows a broader spread.

Use Cases:

- **Survival Probability Modeling**: Poisson distribution is useful for predicting survival rates **under different conditions**.
- **Event Frequency Analysis**: It can estimate the likelihood of specific events happening (e.g., how many passengers survived per group).
- **Synthetic Data Generation**: Helps in **data augmentation** for machine learning models.



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
train_df = pd.read_csv('train.csv')
column_name = 'Fare'
data = train_df[column_name].dropna()
n = int(np.ceil(data.max()))
```

```
p = min(data.mean() / n, 0.99)
binomial_data = np.random.binomial(n, p, size=len(data))
sns.histplot(data, bins=30, color='blue', label='Original Data', alpha=0.5)
sns.histplot(binomial_data, bins=30, color='red', label='Binomial Distribution', alpha=0.5)
plt.legend()
plt.title('Binomial Distribution for'+ column_name)
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

Interpretation:

This code **simulates a Binomial distribution** based on the **'Fare'** column from the Titanic dataset and compares it with the original distribution using histograms.

Code Breakdown & Execution Flow with Specifications:

1. **Loading the Dataset**
   - train_df = pd.read_csv('train.csv')
   - Reads the Titanic dataset, which contains **891 records** with attributes like PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, and Embarked.
2. **Extracting the 'Fare' attribute**
   - column_name = 'Fare'
   - data = train_df[column_name].dropna()
   - Drops missing values (NaN) in the **'Fare'** attribute before analysis.
3. **Defining Binomial Distribution Parameters:**
   - **Number of Trials (n):**
     - n = int(np.ceil(data.max()))
     - The **maximum fare** value is taken as n and rounded up using ceil().
   - **Probability of Success (p):**
     - p = min(data.mean() / n, 0.99)
     - The probability is derived as **mean fare / max fare**, ensuring it does not exceed 0.99.
4. **Generating Binomial Distribution Data:**
   - binomial_data = np.random.binomial(n, p, size=len(data))
   - Creates **synthetic binomially distributed data** based on the calculated parameters.
5. **Plotting the Distribution:**
   - **Original 'Fare' Data:**
     - sns.histplot(data, bins=30, color='blue', label='Original Data', alpha=0.5)
   - **Simulated Binomial Data:**
     - sns.histplot(binomial_data, bins=30, color='red', label='Binomial Distribution', alpha=0.5)

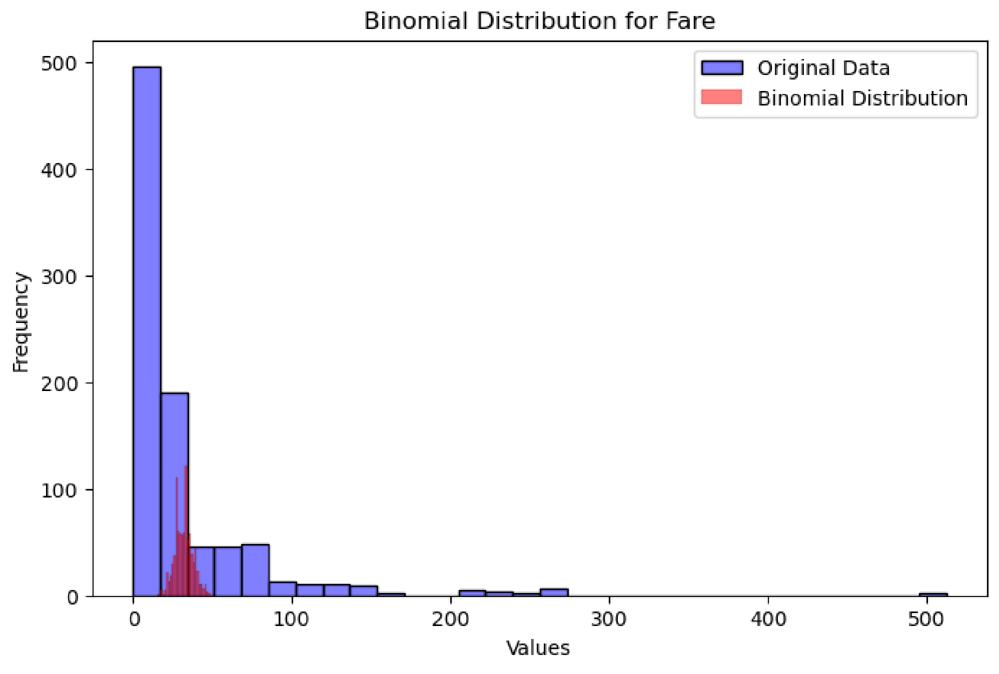
- ○ Labels, legends, and axes are added to ensure clarity.

6. **Displaying the Plot:**
   - ○ plt.legend(), plt.title('Binomial Distribution for ' + column_name), plt.xlabel('Values'), plt.ylabel('Frequency')
   - ○ The histogram visualizes the difference between the original **Fare distribution** and the simulated **Binomial distribution**.

Specifications & Observations:

- The **original 'Fare'** data is **continuous**, whereas a **Binomial distribution** is **discrete**.
- The **Fare distribution is positively skewed**, which might not fit well with a Binomial model.
- The **Binomial model** assumes **independent trials**, which may not accurately represent how fares were assigned in the Titanic dataset.

This method is useful for understanding the suitability of different probability distributions for real-world datasets.


Binomial Distribution for Fare
Original Data
Binomial Distribution
Frequency
Values