Assignment-I: Titanic Datasets

1. **Exploratory Data Analysis (EDA)**
    - **Describe the dataset: number of rows, columns, and data types.**
        - **Code:**

```
In [1]: import pandas as pd

In [2]: import numpy as np

In [3]: import matplotlib.pyplot as plt

In [4]: from sklearn.preprocessing import MinMaxScaler
```

Here in the above code, we have imported different libraries of python for the basic data analysis of the data set.

```
In [5]: df = pd.read_csv('titanic.csv') #load data-set
```

In the 5 no line code we have loaded the 'titanic.csv' which is the titanic data set containing numerical data.

```
In [6]: df.shape #number of rows and column
Out[6]: (891, 12)

In [7]: df.shape[0] #number of rows
Out[7]: 891

In [8]: df.shape[1] #number of columns
Out[8]: 12
```

In the line no 6,7,8 we can see that the number of rows and column present in the data-set are displayed as output.

```
In [9]: df.dtypes  #datatypes
```

```
Out[9]: PassengerId      int64
        Survived         int64
        Pclass           int64
        Name            object
        Sex             object
        Age            float64
        SibSp            int64
        Parch            int64
        Ticket          object
        Fare           float64
        Cabin           object
        Embarked        object
        dtype: object
```

In line no 9, the data types of the attributes or columns in the data -set are displayed using the python function

- **Summarize numerical features: mean, median, mode, range. Standard deviation**
  - **Code:**

```
In [14]: summary_stats = df.describe()
         summary_stats
```

Out[14]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	202.000000	202.000000	202.000000	202.000000	202.000000	202.000000	202.000000
mean	455.495050	0.663366	1.198020	35.112392	0.445545	0.440594	76.103301

Here in the line no 14, the description of the data-set is displayed which is kept in summary_stats.

```
In [15]: numeric_df = df.select_dtypes(include='number') #excludes non numeric columns
```

```
In [18]: mean_values = numeric_df.mean() #mean value of the numerical data
         mean_values
```

```
Out[18]: PassengerId    455.495050
         Survived         0.663366
         Pclass           1.198020
         Age             35.112392
         SibSp            0.445545
         Parch            0.440594
         Fare            76.103301
         dtype: float64
```

The data set contains string values as well as numeric values so to calculate the numerical features we have included only the numerical values in line no 15.

In line no 18, we have calculated mean value from the numerical data extracted from the data-set.

```
In [17]: median_values = numeric_df.median()
         median_values

Out[17]: PassengerId    457.5
         Survived         1.0
         Pclass           1.0
         Age             33.5
         SibSp            0.0
         Parch            0.0
         Fare            55.0
         dtype: float64
```

In line no 17, we have calculated median value from the numerical data of the titanic data-set.

```
In [22]: #mode calculation
         #In case there are multiple modes, this selects the first one
         mode_values = numeric_df.mode().iloc[0] #mode value of the numerical data
         mode_values

Out[22]: PassengerId     2.000000
         Survived        1.000000
         Pclass          1.000000
         Age            29.699118
         SibSp           0.000000
         Parch           0.000000
         Fare           26.550000
         Name: 0, dtype: float64
```

Here in line no 22, we have calculated mode value but if there are multiple modes then 'df.mode().iloc[0]' selects the first one value.

```
In [20]: range_values = numeric_df.max() - numeric_df.min() #range values of the numerical data
         range_values

Out[20]: PassengerId    888.0000
         Survived         1.0000
         Pclass           2.0000
         Age             79.0800
         SibSp            3.0000
         Parch            4.0000
         Fare           512.3292
         dtype: float64
```

Here in line no 20, we have calculated range value of the data set. This part of code calculates the maximum value and minimum value for each column in the DataFrame. The 'max()' function in pandas returns the maximum value along a specified axis. Similarly 'min()' calculates the minimum value for each column in the DataFrame.

In [21]:
```
std_deviation = numeric_df.std()
std_deviation
```

Out[21]:
```
PassengerId    249.704228
Survived         0.473732
Pclass           0.528205
Age             14.988475
SibSp            0.630490
Parch            0.732294
Fare            74.759941
dtype: float64
```

Here in line no 21, the standard deviation is calculated using 'std()' function.

- **Explore categorical features: frequency distribution, unique values.**
  - **Code:**

In [25]:
```
categorical_columns = df.select_dtypes(include='object').columns

for column in categorical_columns:
    frequency_distribution = df[column].value_counts()
    print(f"Frequency distribution for {column}:\n{frequency_distribution}\n")
```

```
Frequency distribution for Name:
Name
Cumings, Mrs. John Bradley (Florence Briggs Thayer)     1
Kimball, Mr. Edwin Nelson Jr                            1
Flynn, Mr. John Irwin ("Irving")                        1
Silvey, Mrs. William Baird (Alice Munger)               1
Thayer, Mrs. John Borland (Marian Longstreth Morris)    1
```

Here in line no 25, we have selected the columns from DataFrame that have data types classified as 'object' and stored in the categorical_columns. Then we have initiated a loop in categorical_columns where in each categorical columns, the frequency distribution of unique values is calculated, and then printed or displayed.

In [27]:
```
for column in categorical_columns:
    unique_values = df[column].unique()
    print(f"Unique values for{column}:\n{unique_values}\n")
```

```
Unique values forName:
['Cumings, Mrs. John Bradley (Florence Briggs Thayer)'
```

Here in line no 27, we have initiated a loop in categorical_columns where unique values are calculated using 'unique ()' function.

- **Visualize distributions: histograms, box plots.**
    - **Code:**

```
In [28]: import seaborn as sns
```

```
In [29]: #set style for seaborn
         sns.set(style="whitegrid")
```

Here in line 28, we have imported seaborn library and given alias name ‘sns’.

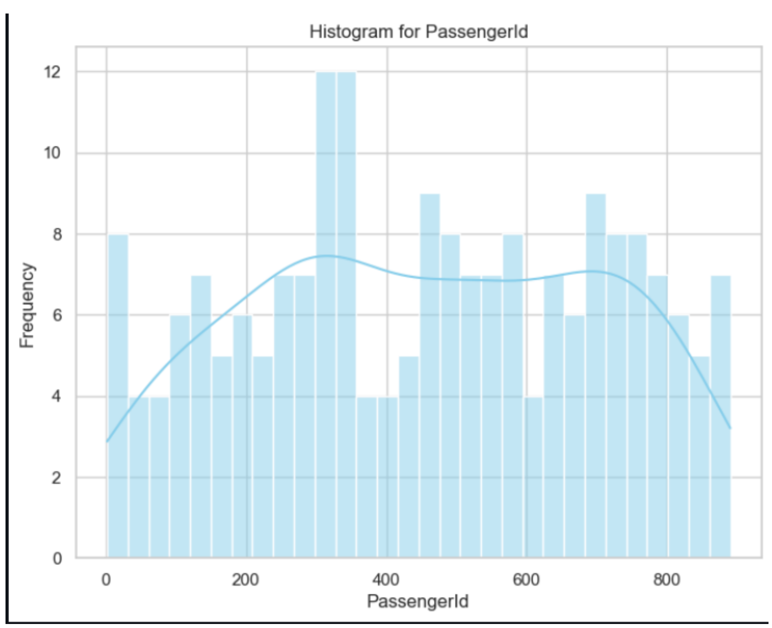
Here in line 29, we have set the overall style of seaborn plots as color white so that there will be more clarity.

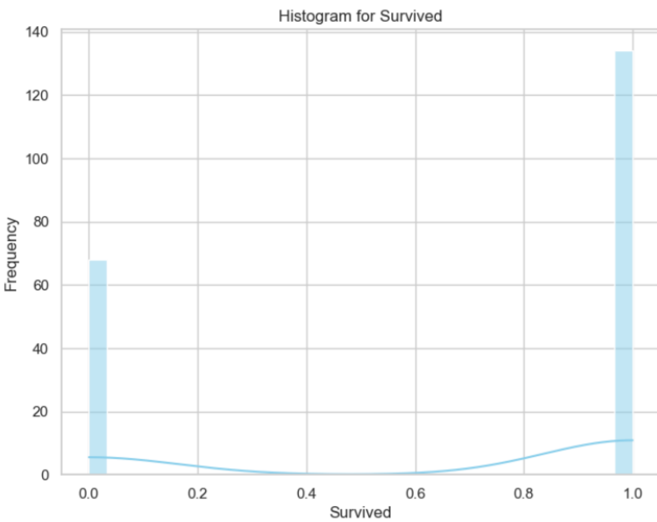
```
In [30]: numeric_columns = df.select_dtypes(include='number').columns
```

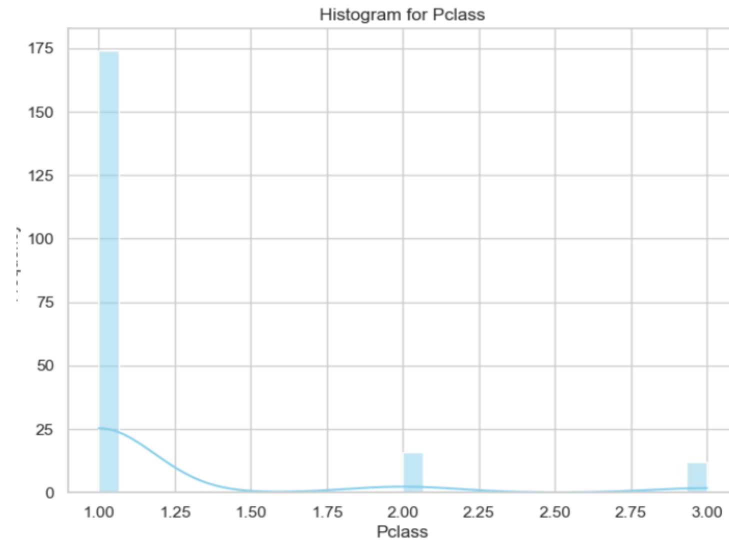
```
In [34]: for column in numeric_columns:
             plt.figure(figsize=(8,6))
             sns.histplot(data=df, x=column,kde=True,bins=30,color='skyblue')
             plt.title(f'Histogram for {column}')
             plt.xlabel(column)
             plt.ylabel('Frequency')
             plt.show()
```

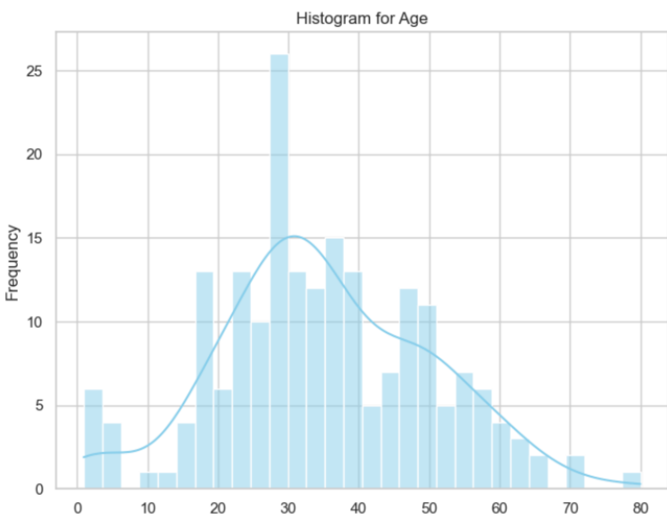
Herre in line no 30, the numerical columns are chosen which is essential for the plots.

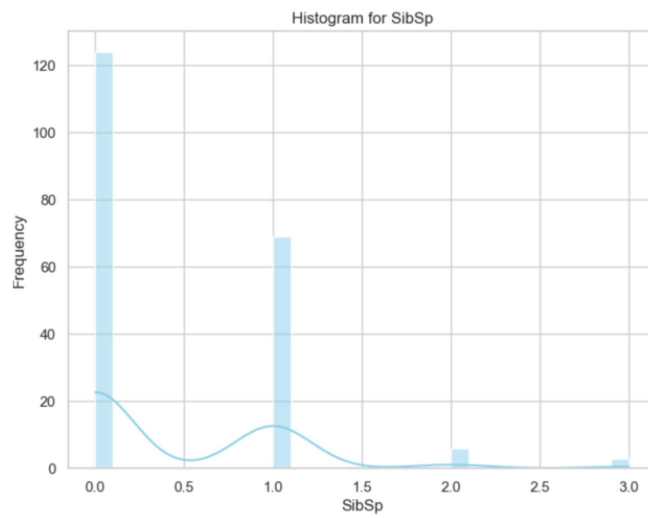
Here in the line no 34, we have initiated a loop in numeric_columns where the figure size is (8,6) and so on. The histogram visualization is created then. It also includes title, xlabel, ylabel and so on. The output is:







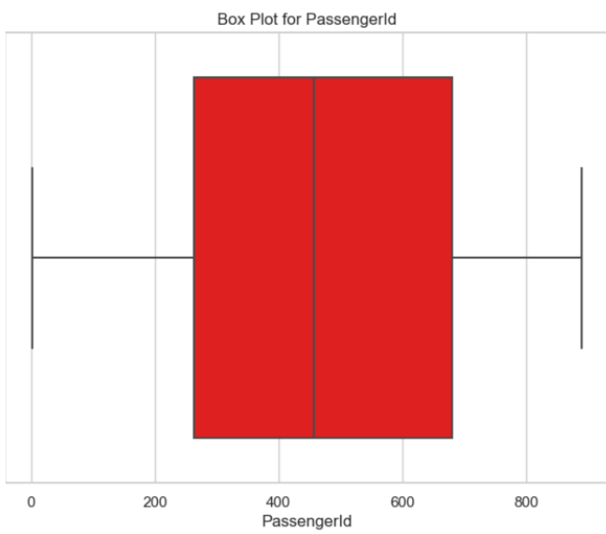


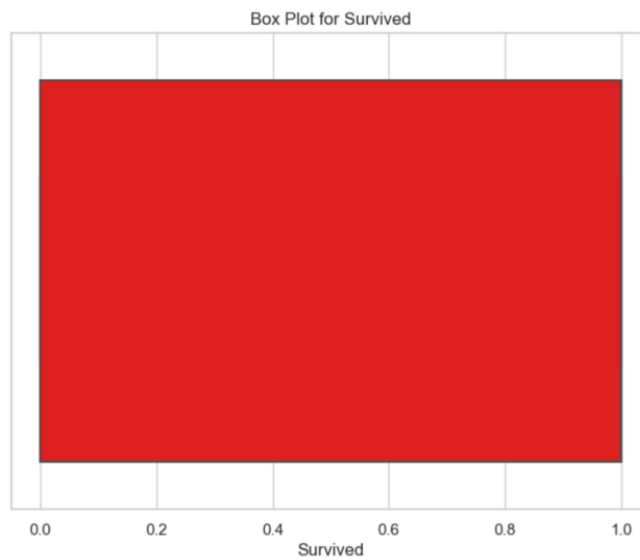


These are the histogram plots of the given data from the data-set.

In [37]:
```
for column in numeric_columns:
    plt.figure(figsize =(8,6))
    sns.boxplot(x=df[column],color='red')
    plt.title(f'Box Plot for {column}')
    plt.xlabel(column)
    plt.show()
```

Here in the line no 347 we have initiated a loop in numeric_columns where the figure size is (8,6) and so on. The Box Plot visualization is created then. It also includes title, xlabel, and so on. The output is:





These are the Box plots of the given data from the data-set.

- **Identify outliers and handle them appropriately.**
  - **Code:**

In [41]:
```
from scipy.stats import zscore
```

In [43]:
```
numeric_columns = df.select_dtypes(include='number').columns
numeric_df = df[numeric_columns]
```

In [44]:
```
z_scores = zscore(numeric_df)
outliers = (abs(z_scores)>3).all(axis=1)

#handle outliers
df_no_outliers = df[~outliers]
```

Here in line no 41, we have imported zscore to identify the outlier. Then from the numerical data we have calculated the z-scores for each element in the DataFrame. The z_scores is the new data set with the same shape as numeric_df, where each element represents the z-score of the corresponding element in 'numeric_df'.

The line ‘outliers = (abs(z_scores)>3).all(axis=1) creates a Boolean series where each row is marked as ‘True’ if any z-score in that row is greater than 3 (which indicated outlier), and ‘False’ otherwise. The ‘all(axis=1)’ ensures that all conditions must be true along each row for that row to be marked as an outlier.

The line ‘df_no_outliers = df[~outliers]’ created a new DataFrame by excluding rows identified as outliers. The ~ operator is used to negate the Boolean values, so only rows without outliers are included in the new DataFrame.

- **Conclusion**
  - At last, from this assignment I have learned many things such as basic commands to calculate the numerical data from the given data-set. The frequency distribution and unique values also give an information about the data-set. The visual representation gives more clarity to the study.

2. Handling Missing Values

- **Identify missing values: count missing values per column**
  - **Code:**

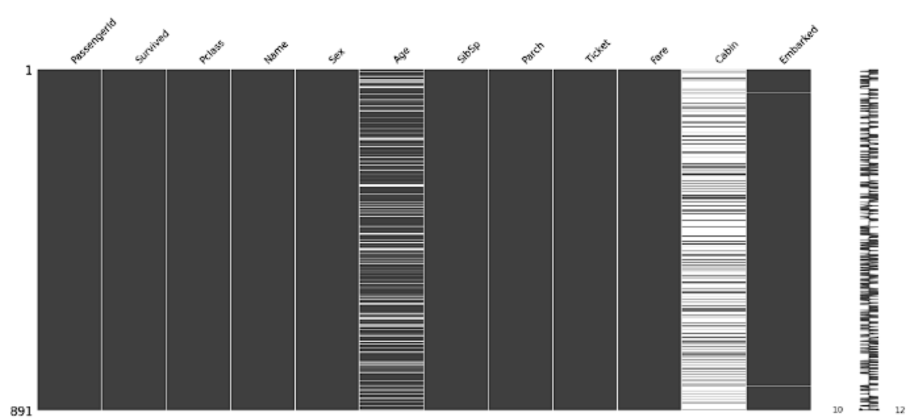
```
In [5]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import missingno as msno
        from sklearn.preprocessing import MinMaxScaler
```

Here in the above code, we have imported different libraries of python for the basic data analysis of the data set.

```
In [2]: df = pd.read_csv('titanic.csv') #load data-set
```

In the 2 no line code we have loaded the 'titanic.csv' which is the titanic data set containing numerical data.

```
In [8]: msno.matrix(df)
Out[8]: <Axes: >
```



Here in line no 8, the line generates a matrix plot using 'missingno' to visualize missing values in the DataFrame. The matrix function displays a grid where each row corresponds to a row in the DataFrame, and each column corresponds to a variable(feature) in the DataFrame. The white cells represent the non-missing values.

```
In [9]: missing_values = df.isnull().sum()
        missing_values
```

```
Out[9]: 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
```

In the line no 9, we can see that missing_values series provide a summary of the count of missing values for each column in the Data Frame. The line 'df.isnull().sum()' states that Boolean DataFrame is obtained which can be True or False then the 'sum()' is applied. Since 'True' is equivalent to 1 and 'False' is equivalent to 0 when summing, this operation counts the number of 'True' values (missing values in

```
In [14]: summary_stats = df.describe()
         summary_stats
```

Out[14]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	202.000000	202.000000	202.000000	202.000000	202.000000	202.000000	202.000000
mean	455.495050	0.663366	1.198020	35.112392	0.445545	0.440594	76.103301

each column. The result is a Series where the index corresponds to column names, and the values represent the count of missing values in each column.

- **Evaluate the impact of missing data: analyze patterns, reasons for missingness.**
  - **Code:**

    missing_info = pd.DataFrame({
    'MissingCount' : missing_values,
    *'MissingPercentage' :(missing_values / len(df)) * 100*
    })
    missing_info = missing_info[missing_info['MissingCount'] > 0].sort_values(by = 'MissingPercentage', ascending=False)
    missing_info

In the above given code, a column in created in missing_info which is *'MissingCount'*. Then the line *'MissingPercentage' : (missing_values / len (df)) * 100* creates another column in the missing_info, DataFrame named 'MissingPercentage', It calculated the percentage of missing values for each column by dividing the count of missing values by the total number of rows in the DataFrame and then multiplying by 100.

The line *'missing_info['MissingCount'] > 0'* condition ensures that only columns with missing values are included in ' missing_info'.

The line *'sort_valyes(by ='MissingPercentage', ascending=False)'* sorts DataFrame based on the 'MissingPercentage' column in descending order, so columns with a higher percentage of missing values appear first.

- **Steps to evaluate the impact of missing data**
  1. Analyze Patterns
     - Use visualization tools like heatmaps to visualize the distribution of missing values across the dataset.
     - Check if missing values occur randomly or if there are specific patterns or clusters.
  2. Reasons for Missingness
     - Understand the reasons behind missing data. It could be missing completely at random (MCAR), missing at random (MAR), or missing not at random (MNAR).
     - MCAR: Missing data occurs randomly and unrelated to other variables.
     - MAR: The probability of missing data depends on other observed variables.
     - MNAR: The missing data is related to its own values, which are unobserved.
  3. Imputation Strategies
     - Decide on an imputation strategy based on the nature and impact of missing data.
     - Impute missing values using statistical measures like mean, median, or mode.

- Consider advanced imputation methods such as K-nearest neighbors (KNN) imputation, regression imputation, or data-driven imputation techniques.

iv. Document Findings
   - Document your findings and the impact of missing data on your analysis.
   - Communicate uncertainties associated with missing data to stakeholders.

v. Sensitivity Analysis
   - Conduct sensitivity analysis to evaluate how different imputation methods or handling strategies impact your results.

- **Choose appropriate strategies for handling missing values: imputation, deletion**
  - **Explain:**
    For handling the missing values, I have chosen both imputation and deletion. While using Imputation I will impute the data using mean method. And delete the row and column if there exist any missing values.

- **Implement chosen strategies: impute missing values using mean, median, mode, or advanced methods like regression**
  - **Code:**

```
In [13]: #Impute missing numeric values with mean
         numeric_df = df.select_dtypes(include = 'number')
         numeric_df.fillna(numeric_df.mean() , inplace=True)
```

Here in line 13, we have chosen mean method for the imputation of the missing numeric data's.

```
In [14]: #Deletion
         #Remove Rows with Missing Values
         numeric_df.dropna(axis=0,inplace=True)
```

Here in line no 14, we have deleted the rows with missing values using the dropna function of python.

```
In [17]: #Remove Columns with Missing Values
         numeric_df.dropna(axis =1, inplace=True)
```

Here in line 17, we have removed the columns with missing values.

- **Validate the effectiveness of the chosen strategy: compare before and after results.**
  - **Code:**

In [21]:
```
#comparision between before and after results
numeric_df.describe()
```

Out[21]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	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	13.002015	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	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Here in line no 21, the before version of the data set is shown.

In [22]:
```
numeric_df_imputed = numeric_df.copy()
numeric_df_imputed.fillna(numeric_df.mean(),inplace=True)
```

In [23]:
```
numeric_df_imputed.describe()
```

Out[23]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	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	13.002015	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	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Here in line no 23, the after version of the data is shown, which shows the effectiveness

3. **Dealing with Duplicate Values**
   - **Detect duplicate rows: identify rows with identical values across all columns.**
     - **Code:**

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.preprocessing import MinMaxScaler
```

Here in the above code, we have imported different libraries of python

```
In [2]: df = pd.read_csv('titanic.csv') #load data-set
```

In the 2 no line code we have loaded the ‘titanic.csv’ which is the titanic data set containing numerical data.

```
In [3]: df_no_duplicates = df.drop_duplicates()
        df_no_duplicates
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age
0	1	0	3	Braund, Mr. Owen Harris	male	22.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0

In line no 3, we have deleted the duplicated rows from the Data Frame.

- **Analyze the impact of duplicated values: assess their frequency and distribution**
  - **Code:**

```
In [4]: #Assess frequency of duplicated values
        duplicate_counts = df.duplicated().sum()
        duplicate_counts

Out[4]: 0
```

Here in line no 4, the code calculated and stores the number of duplicated rows in the Data Frame. The 'duplicated ()' method is useful tool to identify and quantify the presence of duplicate rows in a Data Frame.

```
In [5]: #Assess dstribution of duplicated values acros columns
        column_duplicate_counts = df[df.duplicated()].count()
        column_duplicate_counts

Out[5]: PassengerId    0
        Survived       0
        Pclass         0
        Name           0
        Sex            0
        Age            0
        SibSp          0
        Parch          0
        Ticket         0
        Fare           0
        Cabin          0
        Embarked       0
        dtype: int64
```

Here in line no 5, the code provided the information about the distribution of duplicated values across columns. This information can be useful for understanding which columns have duplicated values in the context of duplicate rows and may aid in further data exploration or cleaning.

- **Decide on the treatment strategy: remove duplicated or keep one instance**
  - **Explain:**
    The decision taken for the treatment strategy is to remove the duplicated.

- **Implement chosen strategies: remove duplicated based on specific criteria, such as ticket number or passenger name.**
  - **Code:**

```
In [7]: duplicate_rows = df.duplicated(subset=['Ticket','Name'])
        df_no_duplicates = df[~duplicate_rows]
        df.duplicated().sum()
        df_no_duplicates.shape

Out[7]: (891, 12)
```

Here in line 7, the duplicated data in ‘Ticket’ and ‘Name’ column which are removed and the output is displayed.

- **Validate the effectiveness of the duplicate removal: assess the impact on data integrity.**
  - **Code:**

```
In [8]: summary_before = df.describe()
        summary_before
```

Out[8]:

	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

Here in line no 8, the before version of the data set is shown.

In [9]:
```
summary_after = df_no_duplicates.describe()
summary_after
```

Out[9]:

	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

Here in line no 9, the after version of the data is shown, which shows the effectiveness

- **Conclusion**
  - At last, from this assignment I have learned many things such as basic commands to calculate the numerical data from the given data-set. The frequency distribution and unique values also give an information about the data-set. The visual representation gives more clarity to the study.