Assignment No. 1

Problem Statement: Exploratory data analysis - operations on dataset.

Objective: To conduct Exploratory Data Analysis (EDA) and preprocess a dataset, the objective is to examine its structure, identify inconsistencies, and make it suitable for machine learning applications. This involves managing missing values, evaluating relationships between variables, utilizing encoding methods, and representing data through visual elements like charts and heatmaps.

Prerequisite :

1. A Python environment set up with libraries like pandas, xml.etree.ElementTree, and requests (for web access).
2. Internet connection (for reading datasets from the web).
3. Text editor and basic knowledge of python and EDA

Theory :

Steps for Exploratory Data Analysis (EDA) and Preprocessing

1. Understanding the Dataset

Before performing any preprocessing, it is essential to analyze the dataset to detect inconsistencies and determine the necessary modifications. Key aspects to examine include:

- **Dataset Dimensions:**
  - Use `.shape` to get the number of rows (samples) and columns (features).
  - Large datasets may require feature selection to prevent overfitting, while small datasets might need augmentation techniques.
- **Data Types of Features:**
  - Columns can have numerical (integer/float) or categorical (string/object) values.
  - The `.info()` function provides an overview of the data types, helping to determine if encoding is needed for categorical variables.
- **Checking for Missing Values:**
  - Missing values can introduce bias in predictions.
  - Use `.isnull().sum()` to count missing values in each column.
- **Statistical Summary:**
  - Key statistics such as mean, median, and standard deviation (`.describe()`) provide insights into data distribution.

    - If the distribution is skewed, transformations like log scaling may be required.

2. Handling Missing Data

To ensure the dataset remains useful for model training, missing values must be addressed. Common approaches include:

- **Removing Missing Data:**
    - Columns with more than **50-60% missing values** may be dropped if they lack sufficient information.
    - Rows with missing values can be removed, but only when the count is small to avoid significant data loss.
- **Imputation Techniques:**
    - **For Numerical Data:**
        - Use **mean** imputation for normally distributed data.
        - Use **median** imputation if the data is skewed.
    - **For Categorical Data:**
        - Replace missing values with the **mode** (most frequently occurring category).

3. Correlation Analysis

Understanding the relationship between numerical features helps in detecting redundant variables, which can cause multicollinearity and impact model performance.

- **Pearson's Correlation Coefficient:**
    - **+1**: Strong positive correlation (both variables increase together).
    - **-1**: Strong negative correlation (one increases while the other decreases).
    - **0**: No correlation.
- **Heatmap Visualization:**
    - A heatmap visually represents correlations, allowing the identification of highly correlated features that may need removal or merging.

4. Encoding Categorical Features

Since machine learning models only work with numerical data, categorical variables must be transformed into numerical representations.

- **Encoding Methods:**
    - **Label Encoding:**
        - Assigns a unique integer to each category.
        - Suitable for **ordinal data** (e.g., Low < Medium < High).
    - **One-Hot Encoding (OHE):**
        - Creates separate binary columns for each category.

- Best for **nominal data** (e.g., gender, city names).

5. Data Visualization

Visualizing data helps identify patterns, trends, and anomalies.

- **Common Visualization Techniques:**
  - **Histograms:** Display numerical data distribution.
  - **Boxplots:** Highlight outliers.
  - **Scatter Plots:** Show relationships between numerical variables.

6. Feature Scaling and Normalization

Scaling numerical features ensures they are on a similar scale, improving model efficiency and accuracy.

- **Standardization (Z-score Normalization):**
  - Transforms data to **zero mean** and **unit variance**.
  - Formula:$X'=X-\mu\sigma X'=\sigma X-\mu$
  - Suitable for algorithms like **linear regression, logistic regression, and PCA**.
- **Min-Max Scaling:**
  - Rescales values between **0 and 1**.
  - Formula:$X'=X-XminXmax-XminX'=Xmax-XminX-Xmin$
  - Commonly used for models like **KNN and neural networks**.
- **Robust Scaling:**
  - Uses **median** and **interquartile range (IQR)** to manage outliers.
  - Formula:$X'=X-MedianIQRX'=IQRX-Median$
  - Ideal for datasets containing extreme values.

By following these steps, the dataset becomes well-structured, free from inconsistencies, and ready for further analysis or machine learning applications.

Code & Output :

[3…

```
import pandas as pd
df= pd.read_csv("/Users/pranavashokdivekar/this_mac/Foundations Of Data Science/Titanic-Dataset.csv")
```

[31…

```
print(df.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      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
```

[32…

```
# shape of the data
df.shape
```

[32…

```
(891, 12)
```

[33…

```
df.tail(10)
```

[33…

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embark
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958	NaN	
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167	NaN	
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000	NaN	
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500	NaN	
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250	NaN	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	

[34…
```
#data information
df.info()
```

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

[35…
```
# describing the data
df.describe()
```

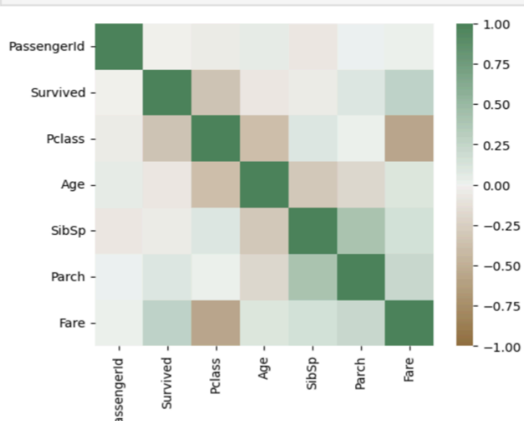
[35…

	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

[36…
```
Corr_Matrix = round(df.select_dtypes(include=[float, int]).corr(), 2)
print(Corr_Matrix)
```

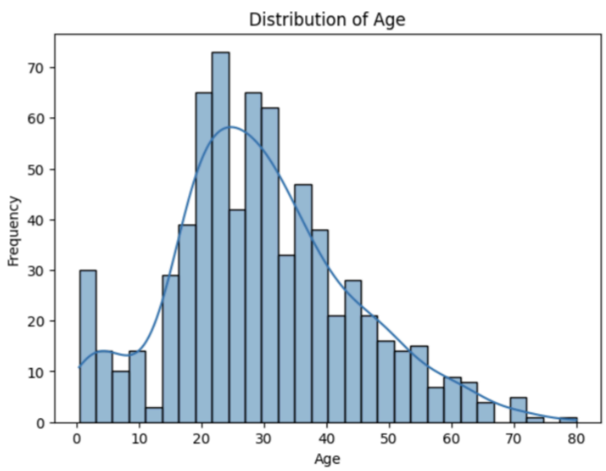
```
             PassengerId  Survived  Pclass   Age  SibSp  Parch  Fare
PassengerId         1.00     -0.01   -0.04  0.04  -0.06  -0.00  0.01
Survived           -0.01      1.00   -0.34 -0.08  -0.04   0.08  0.26
Pclass             -0.04     -0.34    1.00 -0.37   0.08   0.02 -0.55
Age                 0.04     -0.08   -0.37  1.00  -0.31  -0.19  0.10
SibSp              -0.06     -0.04    0.08 -0.31   1.00   0.41  0.16
Parch              -0.00      0.08    0.02 -0.19   0.41   1.00  0.22
Fare                0.01      0.26   -0.55  0.10   0.16   0.22  1.00
```

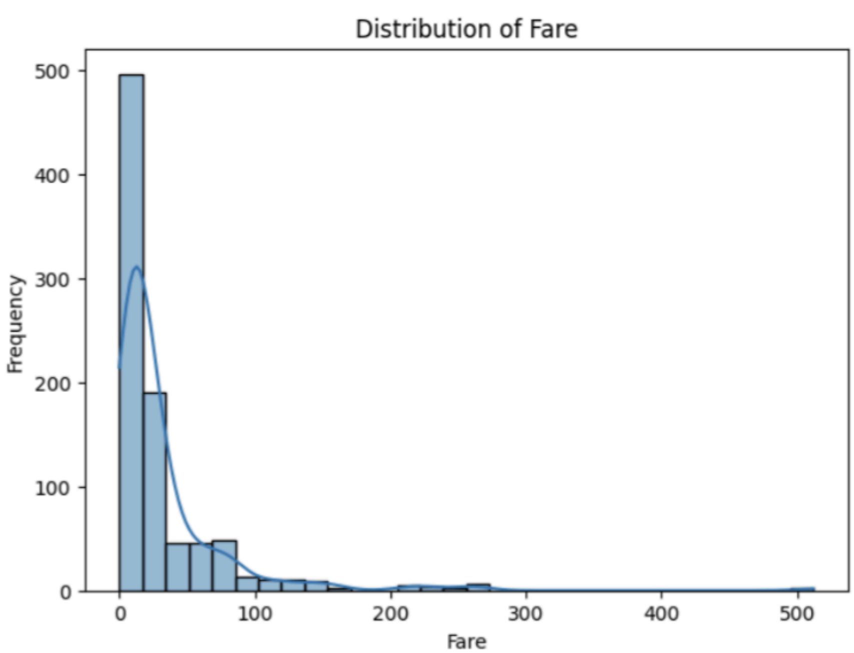
[37…
```
import matplotlib.pyplot as plt
import seaborn as sns

axis_corr = sns.heatmap(
Corr_Matrix,
vmin=-1, vmax=1, center=0,
cmap=sns.diverging_palette(50, 500, n=500),
square=True
)

plt.show()
```



[38…

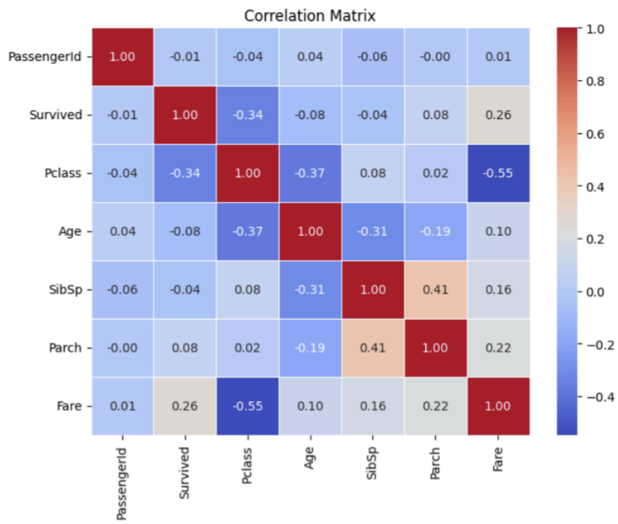
```
import seaborn as sns
import matplotlib.pyplot as plt

# Distribution of Age
plt.figure(figsize=(7, 5))
sns.histplot(df['Age'], kde=True, bins=30)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()

# Distribution of Fare
plt.figure(figsize=(7, 5))
sns.histplot(df['Fare'], kde=True, bins=30)
plt.title('Distribution of Fare')
plt.xlabel('Fare')
plt.ylabel('Frequency')
plt.show()
```





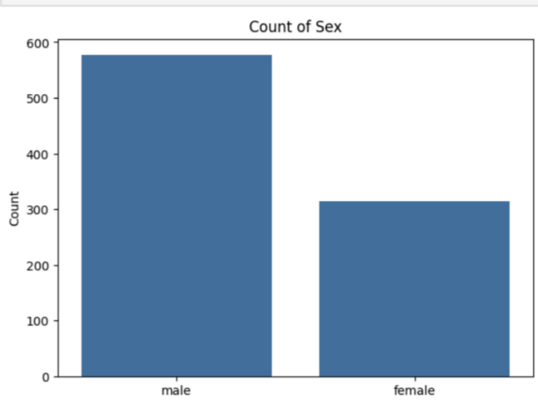
[39…

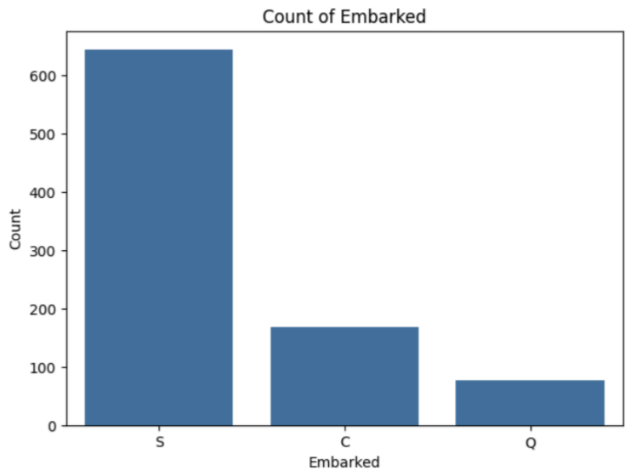
```python
# Correlation Matrix (only numerical columns)
corr_matrix = df.select_dtypes(include=[float, int]).corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

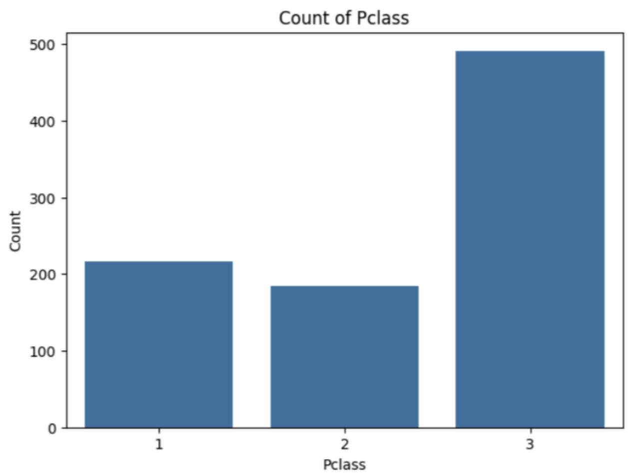


[40…

```python
# Count plot for Sex
plt.figure(figsize=(7, 5))
sns.countplot(x='Sex', data=df)
plt.title('Count of Sex')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.show()

# Count plot for Embarked
plt.figure(figsize=(7, 5))
sns.countplot(x='Embarked', data=df)
plt.title('Count of Embarked')
plt.xlabel('Embarked')
plt.ylabel('Count')
plt.show()

# Count plot for Pclass
plt.figure(figsize=(7, 5))
sns.countplot(x='Pclass', data=df)
plt.title('Count of Pclass')
plt.xlabel('Pclass')
plt.ylabel('Count')
plt.show()
```

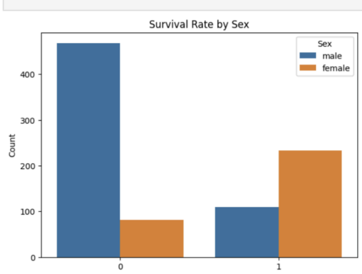


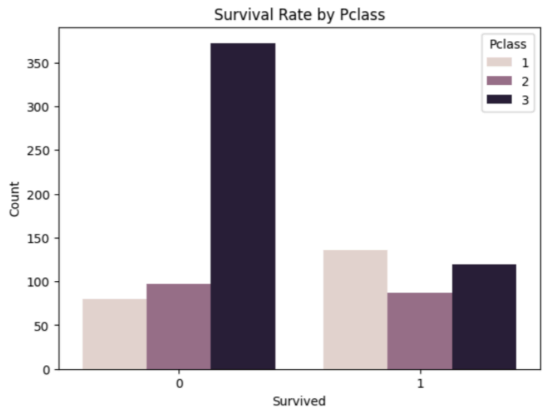


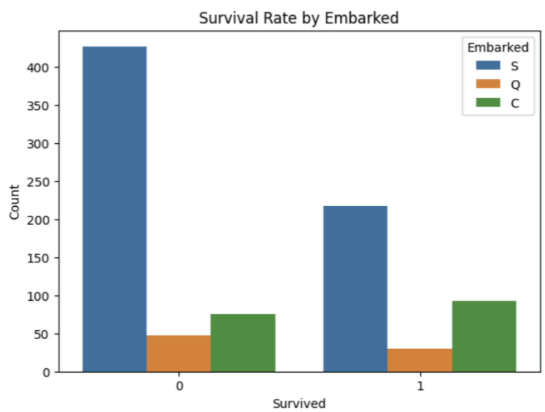


[41…

```
# Survival Rate by Sex
plt.figure(figsize=(7, 5))
sns.countplot(x='Survived', hue='Sex', data=df)
plt.title('Survival Rate by Sex')
plt.xlabel('Survived')
plt.ylabel('Count')
plt.show()

# Survival Rate by Pclass
plt.figure(figsize=(7, 5))
sns.countplot(x='Survived', hue='Pclass', data=df)
plt.title('Survival Rate by Pclass')
plt.xlabel('Survived')
plt.ylabel('Count')
plt.show()

# Survival Rate by Embarked
plt.figure(figsize=(7, 5))
sns.countplot(x='Survived', hue='Embarked', data=df)
plt.title('Survival Rate by Embarked')
plt.xlabel('Survived')
plt.ylabel('Count')
plt.show()
```







[42…

```
# sum of missing values:
df.isnull().sum()
```

[42…

```
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
```

[43…

```
# Calculate the percentage of missing values for each column
missing_percentage = df.isnull().mean() * 100
print(missing_percentage)
```

```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
dtype: float64
```

[ ]:

```
#we can drop the cabin column because it has too much missing values, more than 70%
```

[44…

```
#checking duplicate values
df.nunique()
```

[44…

```
PassengerId    891
Survived         2
Pclass           3
Name           891
Sex              2
Age             88
SibSp            7
Parch            7
Ticket         681
Fare           248
Cabin          147
Embarked         3
dtype: int64
```

```
[46… # Fill missing values in 'Age' with the median of the column
df['Age'] = df['Age'].fillna(df['Age'].median())

# Drop 'Cabin' as it has too many missing values and we don't have enough data to fill them
df.drop(columns=['Cabin'], inplace=True, errors='ignore')

# Fill missing values in 'Embarked' with the mode of the column
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
```

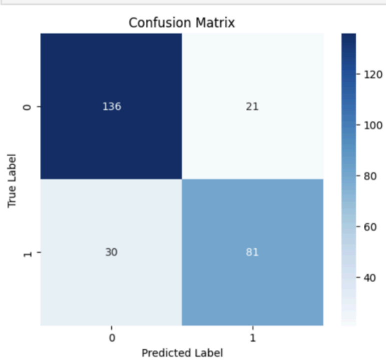
```
[47… df.isnull().sum()
```

```
[47… PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

```
[48… df.info()
```

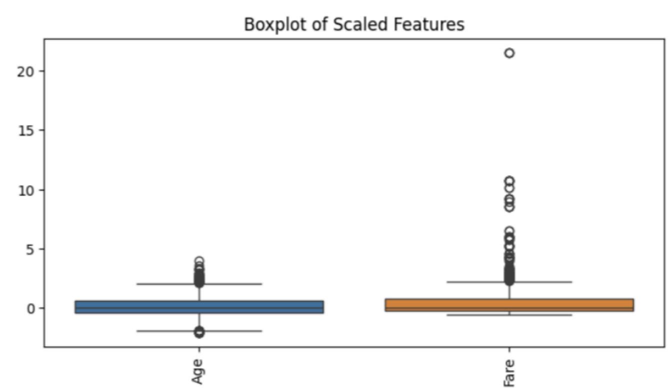
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          891 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  Embarked     891 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

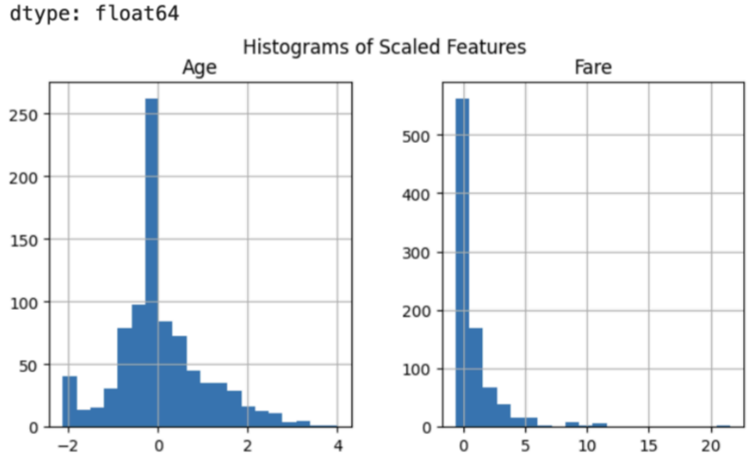
```
[53… #Confusion Matrix Heatmap
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

conf_matrix = np.array([[136, 21], [30, 81]])
class_names = ['0', '1']

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', xticklabels=class_names, yticklabels=class_
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



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

# 1. Check Descriptive Statistics
print("Descriptive Statistics after Scaling:\n", df[['Age', 'Fare']].describe())

# 2. Check Mean and Standard Deviation
print("\nMean Values:\n", df[['Age', 'Fare']].mean())
print("\nStandard Deviation:\n", df[['Age', 'Fare']].std())

# 3. Check Minimum and Maximum Values
print("\nMinimum Values:\n", df[['Age', 'Fare']].min())
print("\nMaximum Values:\n", df[['Age', 'Fare']].max())

# 4. Check Data Distribution Using Histograms
df[['Age', 'Fare']].hist(figsize=(8, 4), bins=20)
plt.suptitle("Histograms of Scaled Features")
plt.show()

# 5. Check Outliers Using Boxplots
plt.figure(figsize=(8, 4))
sns.boxplot(data=df[['Age', 'Fare']])
plt.xticks(rotation=90)
plt.title("Boxplot of Scaled Features")
plt.show()
```

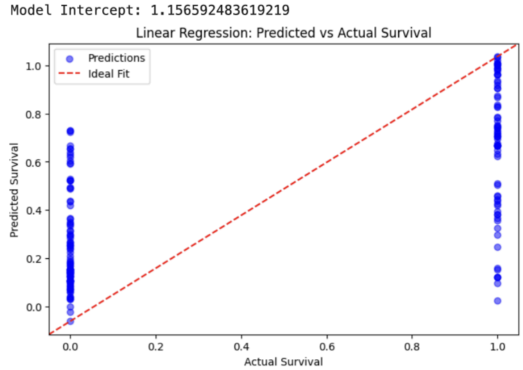
```
Descriptive Statistics after Scaling:
              Age        Fare
count  891.000000  891.000000
mean     0.104737    0.768745
std      1.001515    2.152200
min     -2.121538   -0.626005
25%     -0.461538   -0.283409
50%      0.000000    0.000000
75%      0.538462    0.716591
max      4.000000   21.562738

Mean Values:
 Age     0.104737
Fare    0.768745
dtype: float64

Standard Deviation:
 Age     1.001515
Fare    2.152200
dtype: float64

Minimum Values:
 Age    -2.121538
Fare   -0.626005
dtype: float64

Maximum Values:
 Age      4.000000
Fare    21.562738
dtype: float64
```





```python
#Linear regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# Select Features and Target
X = df[['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex_male', 'Embarked_Q', 'Embarked_S']]
y = df['Survived']

# Split dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Model Evaluation
train_mse = mean_squared_error(y_train, y_train_pred)
train_rmse = np.sqrt(train_mse)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)

print("Training MSE:", train_mse)
print("Training RMSE:", train_rmse)
print("Testing MSE:", test_mse)
print("Testing RMSE:", test_rmse)
print("Model Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)

# Scatter Plot: Predicted vs Actual Survival
plt.figure(figsize=(8,5))
plt.scatter(y_test, y_test_pred, alpha=0.5, color="blue", label="Predictions")
plt.plot([0, 1], [0, 1], transform=plt.gca().transAxes, color="red", linestyle="--", label="Ideal Fit")
plt.xlabel("Actual Survival")
plt.ylabel("Predicted Survival")
plt.title("Linear Regression: Predicted vs Actual Survival")
plt.legend()
plt.show()
```

```
Training MSE: 0.14460581250588436
Training RMSE: 0.3802707095029597
Testing MSE: 0.135074012314622
Testing RMSE: 0.3675241656199249
Model Coefficients: [-0.15450237 -0.06103188 -0.03885891 -0.01957555  0.00823382 -0.51402058
 -0.02452473 -0.07141985]
Model Intercept: 1.156592483619219
```



Github :- https://github.com/Pranav-Divekar/Machine-learning-

Conclusion:

This EDA task improved data quality for machine learning by handling missing values, encoding categorical features, detecting duplicates, and applying scaling. However, incorrect overwriting affected results, highlighting the need for proper transformations.