TAYLOR'S UNIVERSITY
Wisdom · Integrity · Excellence

ITS69304 Data Analytics and Machine Learning

PRACTICAL TEST

HAND OUT DATE: MONDAY, 11 MARCH, 4:00PM
HAND IN DATE: TUESDAY, 12 MARCH 2024, 4:00PM
(24 HOURS SUBMISSION TIME WINDOW)

Instructions to students:

- The test should be attempted individually.
- Complete this cover sheet and attach it to your submission – this should be your first page.

Student declaration:	
I declare that: ▪ ***I understand what is meant by plagiarism.*** ▪ ***The implication of plagiarism and usage of AI generative tool have been explained to us by our lecturer.*** ***This project is all our work and I have acknowledged any use of the published or unpublished works of other people.***	
NAME	
Name	**Student ID**
Satoaki Ishihara	0354208

Avoid copy and paste job in your report and it is considered as plagiarism. Plagiarism in all forms is forbidden. Students who submit plagiarised document will deserve 0 marks.

Table of Contents

Question 1

Exploratory Data Analysis (Total: 10 marks)

You need to work on passenger class, sex, age, the number of siblings or spouses the passenger had, the number of parents or children the passenger had aboard the Titanic, and passenger fare to classify the passenger survival. Numerical and categorical data should be given a focus in this task. Conduct appropriate exploratory data analysis to get more insights about the dataset.

(10 marks)

【Descriptive Analysis】

To begin with the Exploratory Data Analysis on Titanic survival prediction data, descriptive analysis with statistical insights would come first.

The first method is the info function. pandas.DataFrame.**info**() provides us the information of all attributes in the dataset, with each name, count of non-null values and the data type of its attribute.

```
df_train.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
```

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
 11  Survived     418 non-null    int64
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

According to the results, both train and test data seem to have some missing values in “Age” and “Cabin” as common, “Embarked” for train and “Fare” for test. Both datasets are a mixture of int64, float64 and object type.

Next, dimension of train and test data is provided by the below source code execution. pandas.DataFrame.**shape** provides the information of the dataset's dimension in terms of tuple, number of rows as index of 0 and number of columns as index of 1.

```
print("The rows and columns of the train data is ", df_train.shape, ".", sep="")
print("The rows and columns of the test data is ", df_test.shape, ".", sep="")
```

```
The rows and columns of the train data is (891, 12).
The rows and columns of the test data is (418, 12).
```

According to the result, both train and test data are constructed by 12 columns, while train data has 891 rows and test data only contains 418 observations.

pandas.DataFrame.**describe** function represents a lot of statistical information attributes, such as mean, standard deviation, 1Q, 3Q, etc.

```
df_train.describe()
```

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

```
df_train.describe(include="O")
```

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Braund, Mr. Owen Harris	male	347082	B96 B98	S
freq	1	577	7	4	644

```
df_test.describe()
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Survived
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000	418.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188	0.363636
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576	0.481622
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800	0.000000
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200	0.000000
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000	1.000000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200	1.000000

```
df_test.describe(include="O")
```

	Name	Sex	Ticket	Cabin	Embarked
count	418	418	418	91	418
unique	418	2	363	76	3
top	Kelly, Mr. James	male	PC 17608	B57 B59 B63 B66	S
freq	1	266	5	3	270

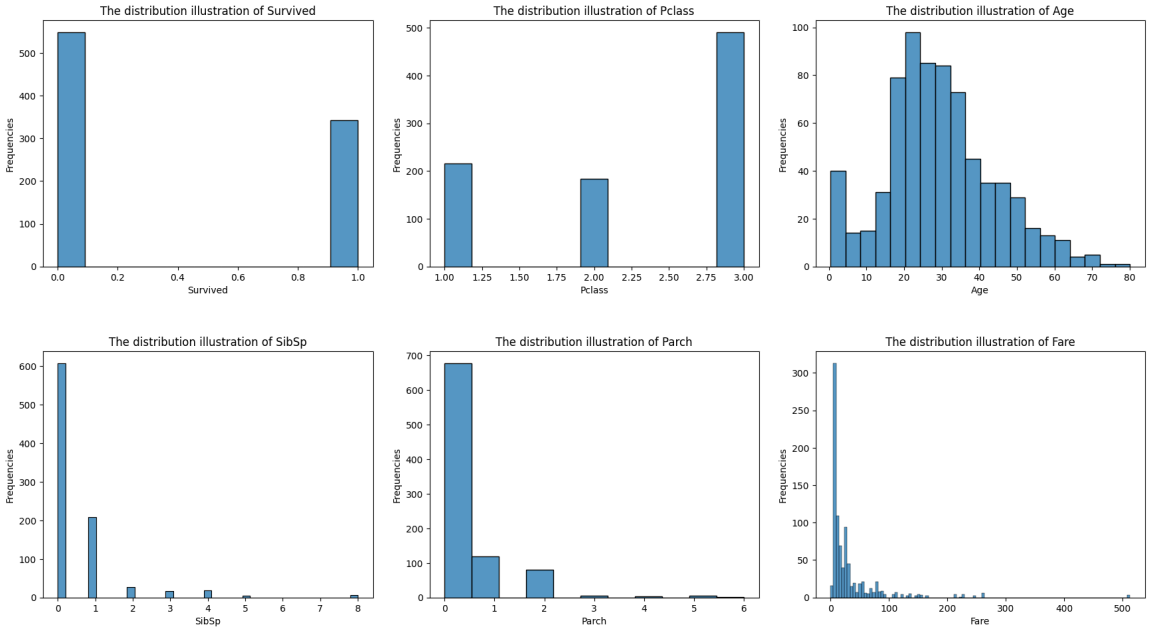
【Analysis with Visual Contexts】

Now proceed to the EDA with visualizations, using a number of plots for the analysis. Initially, drop the unnecessary attributes to not include in the analysis, such as “Name”, “PassengerId”, etc. since those features would not provide any valuable insights.

```
train_numerical = df_train.drop(columns = ["PassengerId", "Name", "Ticket", "Sex", "Embarked"])
train_categorical = df_train[["Sex", "Embarked"]]

test_numerical = df_test.drop(columns = ["PassengerId", "Name", "Ticket", "Sex", "Embarked"])
test_categorical = df_test[["Sex", "Embarked"]]
```

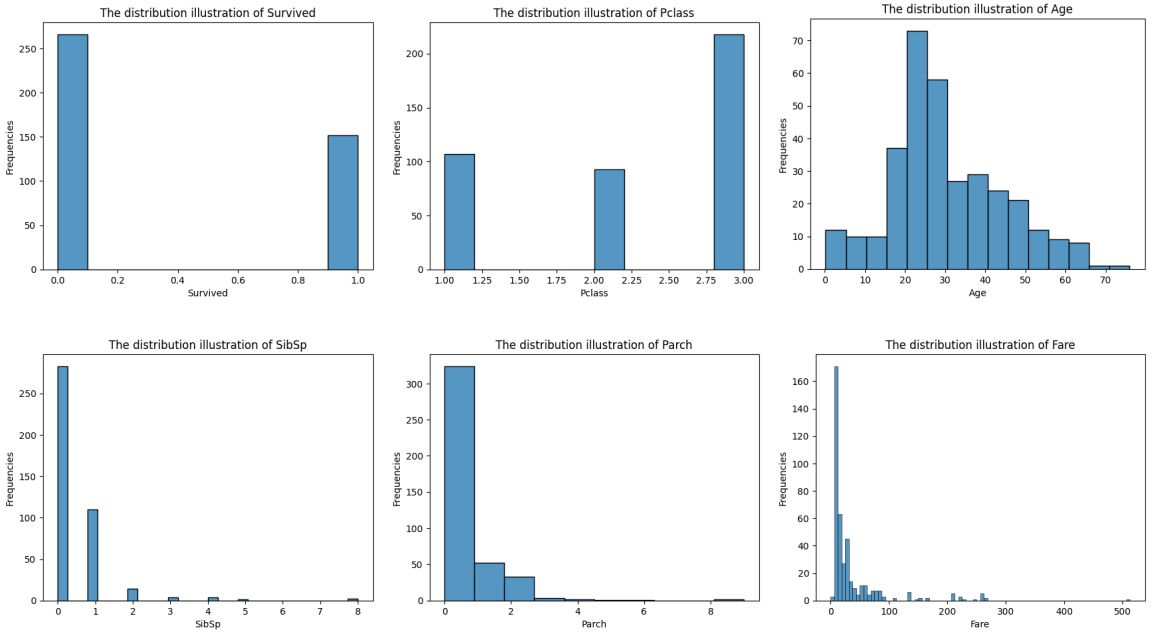
Histograms, Box plots, and Count plots are used to explore the data characteristics in depth.

```
# Histogram for numeric variables in train data set
for col in train_numerical:
    if df_train[col].dtype == int or df_train[col].dtype == float:
        sns.histplot(df_train[col])
        plt.xlabel(f"{col}")
        plt.ylabel(f"Frequencies")
        plt.title(f"The distribution illustration of {col}")
        plt.show()
```



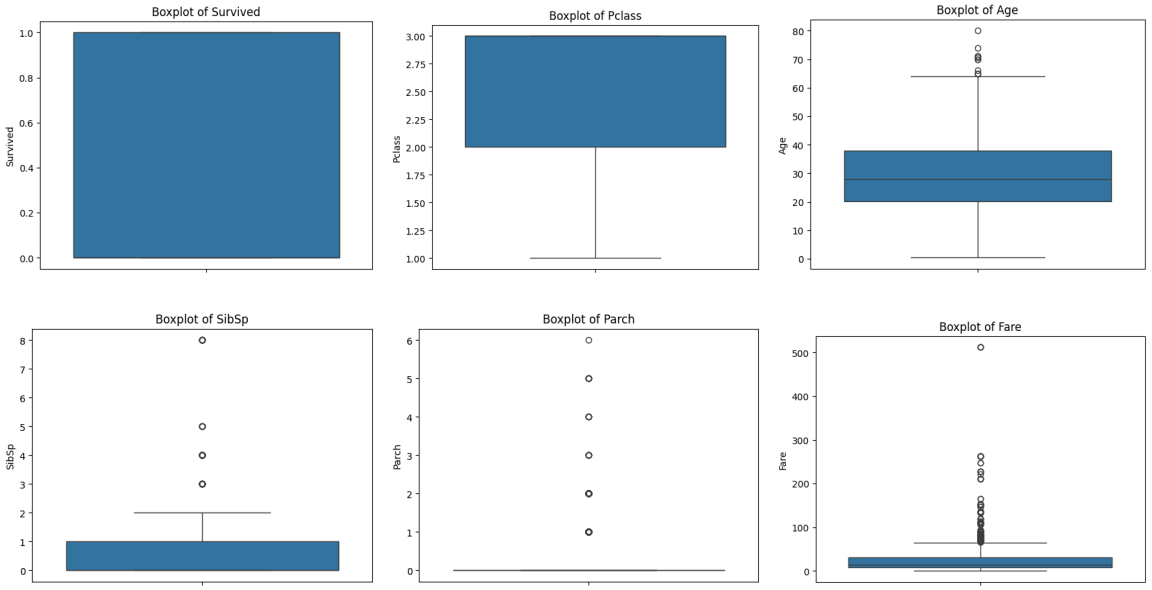
Figures: Histograms of numerical variables in train data.

```
# Histogram for numeric variables in test data set
for col in test_numerical:
    if df_test[col].dtype == int or df_test[col].dtype == float:
        sns.histplot(df_test[col])
        plt.xlabel(f"{col}")
        plt.ylabel(f"Frequencies")
        plt.title(f"The distribution illustration of {col}")
        plt.show()
```



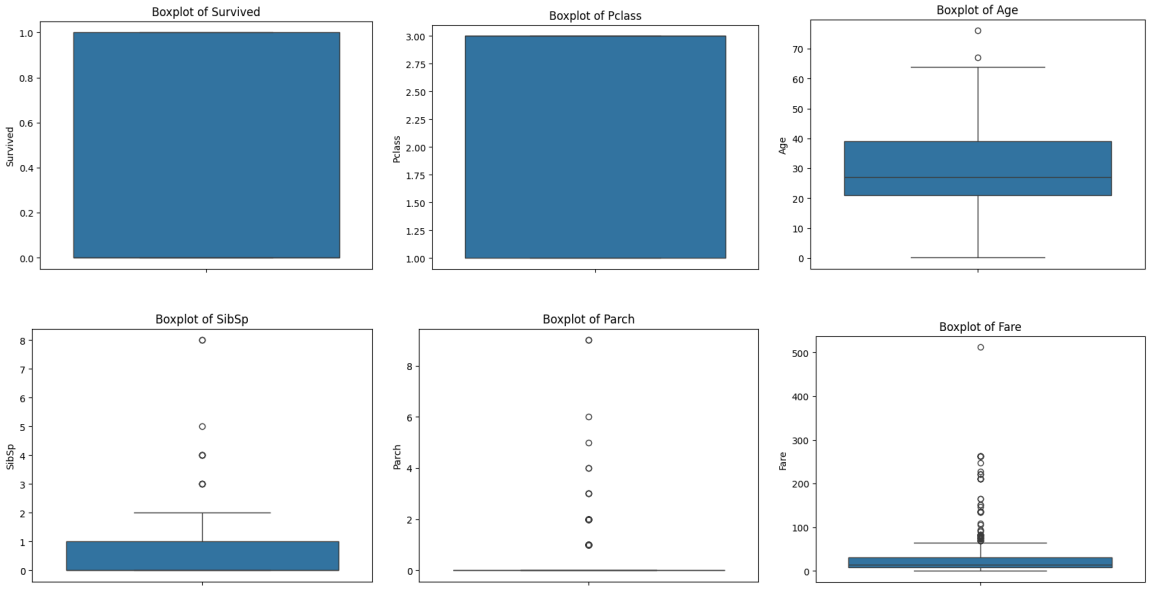
Figures: Histograms of numerical variables in test data.

```
# Box plot for numeric variables in train data set
for col in train_numerical:
    if df_train[col].dtype == int or df_train[col].dtype == float:
        sns.boxplot(df_train[col])
        plt.title(f"Boxplot of {col}")
        plt.show()
```



Figures: Box plots of numerical variables in train data.

```
# Box plot for numeric variables in test data set
for col in test_numerical:
    if df_test[col].dtype == int or df_test[col].dtype == float:
        sns.boxplot(df_test[col])
        plt.title(f"Boxplot of {col}")
        plt.show()
```

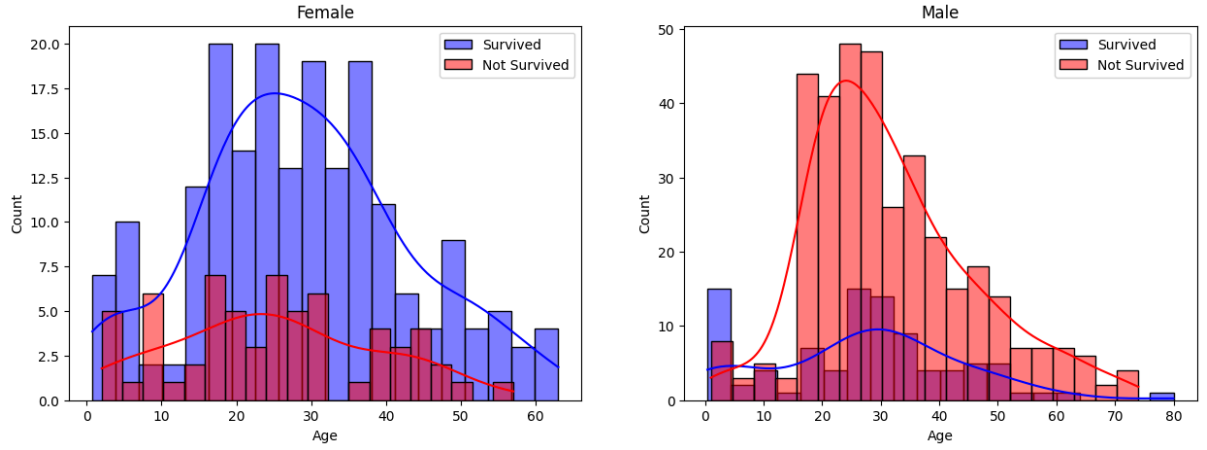


Figures: Box plots of numerical variables in test data.

The histograms of survival distribution for each gender and age bin are depicted below with the source code.

```
survived_label = 'Survived'
not_survived_label = 'Not Survived'
gender_categories = ['female', 'male']
colors = ['blue', 'red']
```

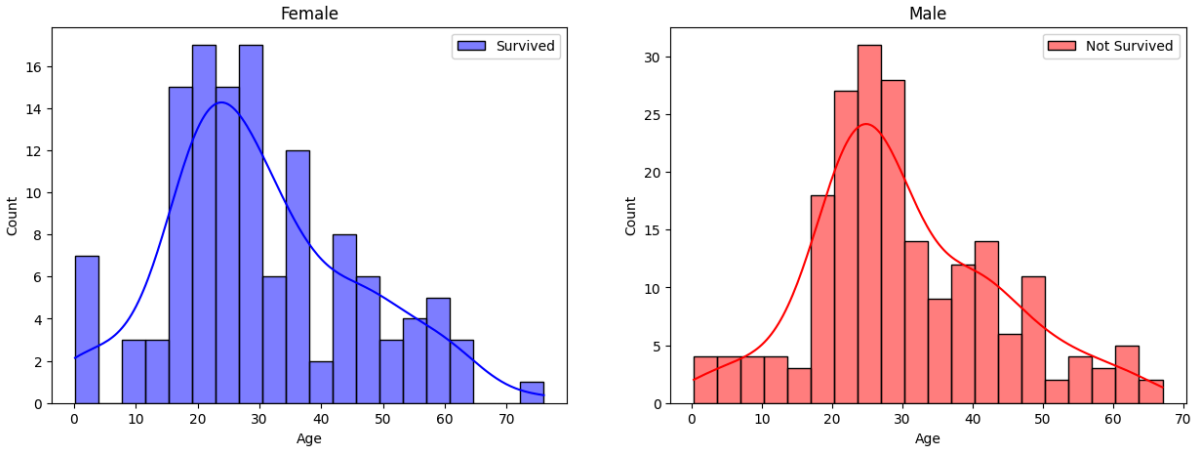
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

# Loop through each gender and plot
for i, gender in enumerate(gender_categories):
    subset = df_train[df_train['Sex'] == gender]

    for survived, color in zip([1, 0], colors):
        sns.histplot(subset[subset['Survived'] == survived]['Age'].dropna(), bins=20, label=f"{survived_label if survived else not_survived_label}", ax=axes[i], color=color, kde=True)

    axes[i].legend()
    axes[i].set_title(f"{gender.capitalize()}")

plt.show()
```



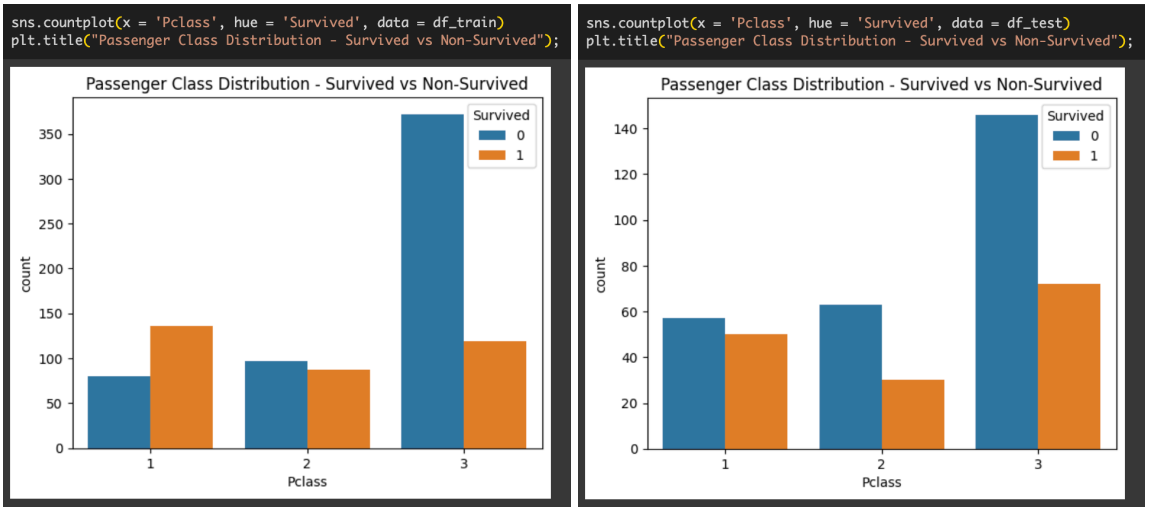
These histograms provided depict the age distribution of passengers on the Titanic, separated by gender and survival status, with 'Survived' shown in blue and 'Not Survived' shown in red.

For females, the histogram shows that the survival rate is higher across most age groups compared to the non-survival rate. There is a particularly high survival rate in the 20-30 age range, which could indicate that younger women had a higher chance of survival. The non-survival rate for women appears to be more evenly distributed but is visibly lower than the survival rate in almost all age brackets. In contrast, the male histogram indicates that the non-survival rate is significantly higher across nearly all age groups, with a particularly high number of non-survivors in the 15-30 age range. There is a small peak of male survivors in the younger age groups (notably under 10), suggesting that young boys had a higher survival rate. However, the chances of survival for men dramatically lessen as age increases.

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

# Loop through each gender and plot
for i, gender in enumerate(gender_categories):
    subset = df_test[df_test['Sex'] == gender]

    for survived, color in zip([1, 0], colors):
        sns.histplot(subset[subset['Survived'] == survived]['Age'].dropna(), bins=20, label=f"{survived_label if survived else not_survived_label}", ax=axes[i], color=color, kde=True)

    axes[i].legend()
    axes[i].set_title(f"{gender.capitalize()}")

plt.show()
```



The histograms suggest a stark contrast in survival outcomes between genders. Women have a relatively high survival count, indicative of the "women and children first" policy that was reportedly followed during the Titanic's evacuation. Conversely, men have a high non-survival count, particularly in the age group that would be traditionally considered of fighting age or responsible for helping others, which may have contributed to their lower survival rates.



The count plots depicted above display the distribution of passengers who survived versus those who did not, categorized by their ticket class on the Titanic. This visualization aids in interpreting the impact of socio-economic status, as represented by the ticket class, on the likelihood of survival.

Within the left count plot, we observe that passengers in the first class had a higher count of survivors compared to those who did not survive. This implies that first-class passengers had a better survival rate, potentially due to higher priority for lifeboats or more accessible escape routes. In contrast, the third class shows a significantly higher count of non-survivors

compared to survivors. The large number of non-surviving third-class passengers could be attributed to lower priority for lifeboats, less accessible escape routes, or possibly more limited awareness of the emergency due to cabin location. The second class appears to have a more balanced survival outcome, with a slightly higher count of non-survivors compared to survivors. The differences are not as stark as in the first or third class, suggesting that while second-class passengers faced disadvantages compared to first class, their survival rate was not as adversely affected as that of the third class.

On the other hand, the right countplot displays the distribution of passengers in a bit different form with the left one. In the first-class group, we observe a nearly even distribution between survivors and non-survivors, indicating that first-class passengers had a relatively high survival rate. This could be due to the better location of their cabins and easier access to lifeboats. The second-class passengers exhibit a slightly lower number of survivors compared to non-survivors. While their survival rates are lower than those in the first class, they are not as low as the third class, suggesting that their intermediary socio-economic status afforded them a better chance of survival than the third class but not as high as the first class. For the third-class passengers, the count of non-survivors is significantly higher than that of survivors. This group represents the largest discrepancy between survival and non-survival counts and underscores the harsh reality that third-class passengers had the lowest survival rate, possibly due to their cabins' location farthest from the lifeboats and lower prioritization during rescue operations.

Question 2

Data Pre-Processing (Total: 20 marks)

Part (a)

When embarking on data pre-processing:

- Initiate the process by thoroughly examining your dataset to uncover and subsequently address missing values in both categorical and numerical attributes, utilizing appropriate imputation techniques reflective of the data's nature. Transition categorical variables into a format amenable to machine learning algorithms via encoding strategies, selecting between One-Hot and Ordinal Encoding based on the data's characteristics.
- Similarly, consider the application of feature scaling techniques such as standardization or normalization to numerical variables, ensuring uniform contribution across features and preventing scale disparities from skewing the model. Think about the impact of splitting the data into train and test.

(10 marks)

【Data Cleaning】

For the first step of preprocessing on the dataset, I have dropped the unnecessary attributes like “PassengerId”.

```
df_train = df_train.drop(columns = ["PassengerId", "Name", "Ticket"])
df_test = df_test.drop(columns = ["PassengerId", "Name", "Ticket"])
```

Then I have performed **isna().sum()** function to recognize the number of missing values in each attribute, on both train data and test data.

```
# Check the number of missing values of each attribute belongs to the dataset
df_train.isna().sum()
```

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

```
# Check the number of missing values of each attribute belongs to the dataset
df_test.isna().sum()
```

```
Pclass        0
Sex           0
Age          86
SibSp         0
Parch         0
Fare          1
Cabin       327
Embarked      0
Survived      0
dtype: int64
```

Seems the **Cabin** attribute has more than 50% missing values, might not be valuable to keep in the datasets. Below source code is implemented to remove those attributes with critically a lot of missing values, specifically more than 50%.

```
# Drop the column if the column has more than 50% of missing values
for col in df_train:
    if df_train[col].isna().sum() >= (df_train.shape[0] / 2):
        df_train = df_train.drop(columns = col)

for col in df_test:
    if df_test[col].isna().sum() >= (df_test.shape[0] / 2):
        df_test = df_test.drop(columns = col)
```

```
# Check whether all those critically missing attributes are demolished in train data
df_train.isna().sum()
```

```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

```
# Check whether all those critically missing attributes are demolished in test data
df_test.isna().sum()
```

```
Pclass       0
Sex          0
Age         86
SibSp        0
Parch        0
Fare         1
Embarked     0
Survived     0
dtype: int64
```

Cabin attribute is removed from the dataset, and now we only have **Age** and **Embarked** in train data, **Age** and **Fare** in test data to impute missing values.

```
"""
If, the attribute's data type is:
- object: fill in N/A with mode
- int64 or float64: fill in N/A with mean value
"""
for col in df_train:
    if df_train[col].isna().sum() == 0:
        continue
    else:
        if df_train[col].dtype == object:
            df_train[col].fillna(df_train[col].mode()[0], inplace = True)

# Fill in missing values in "Age" attribute in train data
df_train["Age"].fillna(df_train["Age"].mean(), inplace = True)

for col in df_test:
    if df_test[col].isna().sum() == 0:
        continue
    else:
        if df_test[col].dtype == object:
            df_test[col].fillna(df_test[col].mode()[0], inplace = True)

# Fill in missing values in "Age" and "Fare" attributes in test data
df_test["Age"].fillna(df_test["Age"].mean(), inplace = True)
df_test["Fare"].fillna(df_test["Fare"].median(), inplace = True)
```

Embarked attribute's N/A attributes are imputed with mode value (because the attribute is categorical) and mean value is adopted for **Age** attribute (because the attribute is numerical). **Fare** attribute in test data is filled with its median value. The entire missing values are successfully imputed and hence the dataset does not contain any missing values in itself, proofed by the below source code and output.

```
# Check whether the entire N/A values are successfully filled in train data
df_train.isna().sum()
```

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

```
# Check whether the entire N/A values are successfully filled in train data
df_test.isna().sum()
```

```
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
Survived    0
dtype: int64
```

【Encoding Categorical Variables】

- **One-Hot Encoding**: One-hot encoding is the simplest and most basic method. It is called 'one-hot' because it is set to 1 if it takes a certain category value and 0 otherwise. The method assigns a binary (binary) feature to each category value. The number of features produced by one-hot encoding is equal to the number of categories in the categorical attribute. Nevertheless, one-hot encoding might not be ideal in case that the specific categorical attribute contains inherent hierarchy among values. Furthermore, one-hot encoding can lead to a high-dimensional feature space if the categorical variable has many unique values.

- **Ordinal Encoding**: Ordinal encoding is a method of creating sequentially numbered or designated numbered columns corresponding to the number of items in a category variable. The behavior is similar to Label encoding, but the converted values can also

be specified. It has the advantage that encoding can be done in a sequential manner with solving the problems of Label encoding, but its disadvantage is that it is not an effective method for items where the difference between ranks is not constant, as it makes the difference uniform.

The choice of encoding method varies among the characteristics of each attribute we are going to encode for model training. Below list explains each attribute's suitable encoding method.

- **Sex**: This categorical variable is ideal for one-hot encoding since order is not crucial for this attribute.
- **Embarked**: As a nominal variable with few categories, it could be suitable for one-hot encoding.
- **Pclass**: Although it's numeric, it represents class categories and could be treated as ordinal and the hierarchy has a purpose, making ordinal encoding suitable.

Ought to these points, I would perform One-Hot Encoding for "Sex" and "Embarked", Ordinal Encoding for "Pclass" variable. Below source code is implemented for the overall encoding.

```
# Create an object of One-Hot Encoder and Ordinal Encoder
ohe = OneHotEncoder()
oe = OrdinalEncoder()
```

```
## Encode each attribute with its own suitable encoder (Train data)

# One-Hot Encoding ("Sex", "Embarked")
ohe_data = pd.DataFrame(ohe.fit_transform(df_train[["Sex", "Embarked"]]).toarray())
df_train = df_train.join(ohe_data).drop(columns = ["Sex", "Embarked"])

# Ordinal Encoding ("Pclass")
df_train[["Pclass"]] = oe.fit_transform(df_train[["Pclass"]])

## Encode each attribute with its own suitable encoder (Test data)

# One-Hot Encoding ("Sex", "Embarked")
ohe_data = pd.DataFrame(ohe.fit_transform(df_test[["Sex", "Embarked"]]).toarray())
df_test = df_test.join(ohe_data).drop(columns = ["Sex", "Embarked"])

# Ordinal Encoding ("Pclass")
df_test[["Pclass"]] = oe.fit_transform(df_test[["Pclass"]])
```

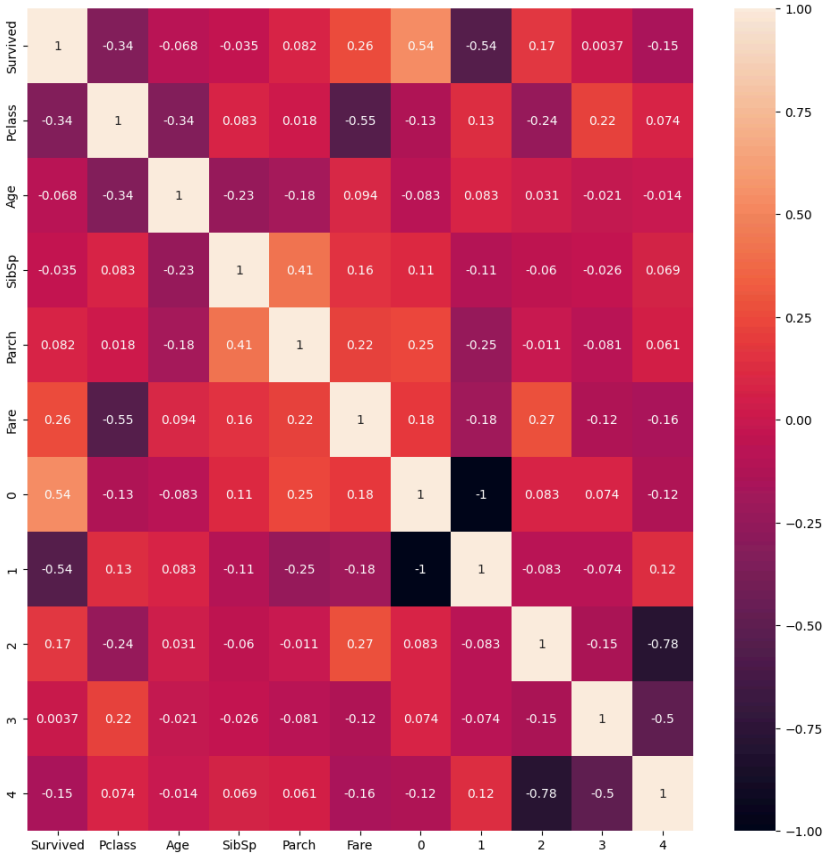
After the encoding is completed, all float data type attributes including the one generated during encoding must be converted to the integer type, to perform model training without any potential issues later.

```
# Convert the all column names data type into string (Train data)
df_train.columns = df_train.columns.astype(str)

# Convert the float type features into integer
for col in df_train:
    if df_train[col].dtype == float:
        df_train[col] = df_train[col].astype(int)

# Convert the all column names data type into string (Test data)
df_test.columns = df_test.columns.astype(str)

# Convert the float type features into integer
for col in df_test:
    if df_test[col].dtype == float:
        df_test[col] = df_test[col].astype(int)
```

Part (b)

Since the entire preprocessing phase is completed, I am going to generate the correlation matrix heatmap with the below code execution, generate the heatmap attached below.

```
# Heatmap illustrates the entire correlation matrix
plt.subplots(figsize = (12, 12))
sns.heatmap(df_train.corr(), annot = True)
plt.show()
```



According to the heatmap, following insights could be gained.

The **Fare** attribute shows a positive correlation with "Survived," suggesting that passengers who paid higher fares were more likely to survive. This could be because higher fares might correspond to higher classes, which had better access to lifeboats.

The **Pclass** attribute has a negative correlation with "Survived." This indicates that passengers in higher classes (where 1st class is the highest and 3rd class is the lowest) were more likely to survive, which is consistent with historical accounts of the Titanic disaster.

The **0** and **1** attributes, which are the “Sex” attribute components encoded with One-Hot Encoder, would likely be very significant based on historical accounts. This could be because female passengers had a higher survival rate.

The **2** and **4** attributes are also having relatively higher correlations with “Survived”.

Therefore,

- Fare
- Pclass
- 0
- 1
- 2
- 4

would be deployed to predict the outcome of **Survived**. Test dataset features would be selected same as the above selection, to consistently keep the features used for prediction.

Question 3

Part (a)

a) Choose 3 out of these classification algorithms: Logistic Regression, KNN, Kernel SVM rbf, Naïve Bayes, Decision Tree, and Random Forest for model development using Python.

(30 marks)

I would choose the following three classification algorithms.

- Logistic Regression
- Random Forest
- KNN

Logistic Regression is used when the dependent variable(target) is categorical. It estimates the probabilities using a logistic function, which is the cumulative logistic distribution. In this algorithm, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function. It's a linear model for binary classification that can be extended to multiclass classification, e.g., via the One-vs-Rest (OvR) scheme.

```
# Initializing the Logistic Regression model
LR = LogisticRegression(max_iter = 10000)

# Fitting the model with training data
LR.fit(X_train, y_train)

# Making predictions on the test data
lr_pred = LR.predict(X_test)
```

Random Forest is an ensemble learning method for classification (and regression) that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. They are very handy as they have methods for balancing error in class population unbalanced data sets and can handle thousands of input variables without variable deletion.

```
# Initializing the Logistic Regression model
RFC = RandomForestClassifier()

# Fitting the model with training data
RFC.fit(X_train, y_train)

# Making predictions on the test data
rfc_pred = RFC.predict(X_test)
```

K-Nearest Neighbors (KNN) is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The KNN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. It stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition as a non-parametric technique.

```
# Initializing the Logistic Regression model
KNC = KNeighborsClassifier()

# Fitting the model with training data
KNC.fit(X_train, y_train)

# Making predictions on the test data
knc_pred = KNC.predict(X_test)
```

All these models would be trained in a consistent way of programming, provided below. ModelClassifier could be replaced with classifier names like LogisticRegression etc.

```
# Initializing the Logistic Regression model
model = ModelClassifier()

# Fitting the model with training data
model.fit(X_train, y_train)

# Making predictions on the test data
prediction = model.predict(X_test)
```

Part (b)

b) Provide justifications related to the selection of such algorithms.

(10 marks)

Logistic Regression is chosen for its robustness and efficiency, particularly when it comes to binary classification problems such as predicting survival. It is a probabilistic model that lends itself well to scenarios where the outcome is dichotomous, as in the case of survival or non-survival. The model outputs probabilities, which can be thresholded to classify observations, providing a clear framework for decision-making. Moreover, its interpretability is a significant asset, offering insights into the importance and impact of different features, such as passenger class or age, on the predicted outcome.

Random Forest is included due to its high accuracy and the capacity to model complex interactions between features without the need for feature scaling. As an ensemble method that combines multiple decision trees, it naturally handles overfitting better than individual trees and is well-suited for datasets with a mixture of categorical and numerical features. Furthermore, Random Forest can handle missing values and provide estimates of feature importance, which can be invaluable when attempting to draw conclusions about the factors influencing survival rates.

K-Nearest Neighbors is selected for its simplicity and effectiveness in capturing the structure of the data by considering the proximity of observations. KNN is a non-parametric method that makes no underlying assumptions about the distribution of the data, making it a flexible choice for classification. It is especially useful when the decision boundary is not linear, as KNN can adapt to any shape of the dataset, provided an appropriate distance metric and k-value are chosen.

In conclusion, these three algorithms offer a blend of simplicity, interpretability, and predictive power. They are capable of capturing linear relationships and complex patterns in the data while also providing insights into feature importance and classification probabilities. This multifaceted approach is conducive to developing a robust predictive model for passenger survival on the Titanic, as it accounts for the intricacies and varied nature of the dataset.

Part (c)

c) Evaluate the performance of your model on the testing set. Focus on evaluation metrics such as accuracy, precision, recall, and the F1-score to gauge your model's performance.

(10 marks)

To evaluate the performance of my models on the testing set, I would use five evaluation metrics: Recall, Precision, F1-score, Accuracy and AUROC based on the representation of the confusion matrix.

```
# Evaluate the Train set
print(classification_report(y_test, lr_pred))
lr_cm = confusion_matrix(y_test, lr_pred)
sns.set_context("notebook")
ConfusionMatrixDisplay(confusion_matrix = lr_cm, display_labels = LR.classes_).plot()
```

```
# Evaluate the Train set
print(classification_report(y_test, rfc_pred))
rfc_cm = confusion_matrix(y_test, rfc_pred)
sns.set_context("notebook")
ConfusionMatrixDisplay(confusion_matrix = rfc_cm, display_labels = RFC.classes_).plot()
```

```
# Evaluate the Train set
print(classification_report(y_test, knc_pred))
knc_cm = confusion_matrix(y_test, knc_pred)
sns.set_context("notebook")
ConfusionMatrixDisplay(confusion_matrix = knc_cm, display_labels = KNC.classes_).plot()
```

```
# Evaluation Scores
roc_auc = round(roc_auc_score(y_test, knc_pred), 4)
accuracy = round(accuracy_score(y_test, knc_pred), 4)
recall = round(recall_score(y_test, knc_pred), 4)
precision = round(precision_score(y_test, knc_pred), 4)
f_one = round(f1_score(y_test, knc_pred), 4)

# Return evaluation scores
print(f"=========================")
print(f" ROC-AUC Score: {roc_auc}")
print(f" Accuracy Score: {accuracy}")
print(f" Recall Score: {recall}")
print(f" Precision Score: {precision}")
print(f" F1-score: {f_one}")
print(f"=========================")
```

Below four outcomes would be crucial for evaluations.

$$TP\text{: } True\ Positive\ (Predicted\ Positive\ and\ also\ actually\ Positive)$$

$$TN\text{: } True\ Negative\ (Predicted\ Negative\ and\ also\ actually\ Negative)$$

$$FP\text{: } False\ Positive\ (Predicted\ Positive\ but\ actually\ they\ were\ Negative)$$

$$FN\text{: } False\ Negative\ (Predicted\ Negative\ but\ actually\ they\ were\ Positive)$$

Recall is the metric that is also known as Sensitivity. Recall is crucial because it measures the ability of the model to correctly identify the positive class (in this case, frauds would be the one).

$$Recall\ =\ \frac{TP}{TP+FN}$$

On the other hand, **Precision** measures the proportion of positive predictions that are actually positive.

$$Precision\ =\ \frac{TP}{TP+FP}$$

These two metrics might not always support each other's outputs. Improving recall might reduce precision and improving precision might reduce recall. Therefore, the **F1-Score** could be helpful, as it provides an information of balance between the previous two metrics, recall and precision.

$$F1 - Score = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

The **accuracy** of a model is a measure of its overall performance, or the number of correct predictions it could make for a given data point.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

AUROC is a critical evaluation parameter for assessing the effectiveness of any classification model. It has an advantage that the metric provides meaningful information on whether the model is actually acquiring knowledge from the data or simply anticipating.

In conclusion, while accuracy is a useful metric for getting a quick understanding of the overall effectiveness of the model, it is not always sufficient, especially in cases where there is a class imbalance. Metrics like recall, precision, and the F1-score provide a more nuanced view by considering the type of errors the model makes.

Recall is particularly important when the consequences of missing a positive case are severe. Precision is key when the cost of a false positive is high. The F1-score offers a balance between precision and recall, and is especially useful when it's hard to decide which of the two is more important. Finally, the AUROC gives an aggregate measure of performance across all classification thresholds and is particularly useful for evaluating the discriminative ability of the model.

Together, these metrics provide a comprehensive assessment of a model's performance and help in making informed decisions when it comes to model selection and optimization for the task of predicting passenger survival on the Titanic.

Question 4

Part (a)

a) Introduce results heatmap that compares evaluation metrics across all selected algorithms.

(5 marks)

The results of each model evaluation are attached below. LR, RF and KNN from left.

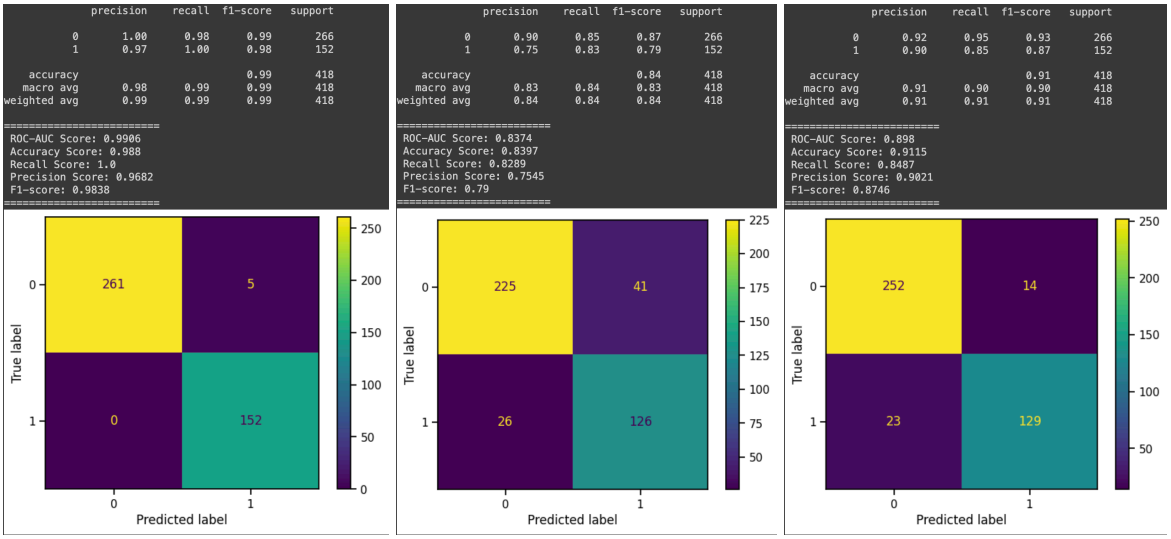


Figure: Logistic Regression, Random Forest and K-Nearest Neighbors confusion matrix heatmaps.

The Logistic Regression model has a high ROC-AUC score of 0.9906, indicating excellent discrimination between the positive and negative classes. It has a high overall accuracy of 0.988, with excellent precision and recall for class 0, but slightly lower (yet still high) for class 1.

For Random Forest, ROC-AUC score is 0.8374, which is lower than that of Logistic Regression but still indicates a good ability to discriminate between the classes. The accuracy is lower at 0.837, and both precision and recall are lower compared to Logistic Regression, especially for class 1.

The K-Nearest Neighbors model has a ROC-AUC score of 0.898, which is quite good, and the overall accuracy is 0.9115. The precision, recall, and F1-score are all high and are between the values for Logistic Regression and Random Forest.

Part (b)

b) Discuss and justify the evaluation metric results obtained in (a). The discussions must be specific towards the applicability of the selected algorithms in predicting passengers' survival status.

(10 marks)

Logistic Regression (LR) displays impressive performance with the highest precision, recall, and F1-scores across both classes, as well as the highest overall accuracy. Its ROC-AUC score is nearly perfect, indicating an excellent ability to distinguish between the survival statuses. This suggests that the relationship between the predictors and the outcome is relatively linear or that the boundaries separating the classes can be well described by a logistic function. The LR model's high recall for the survival class (1) means it is particularly adept at identifying most of the positive instances correctly, which is crucial for a scenario where missing out on a true positive could mean not recognizing a survivor.

On the other hand, Random Forest shows a decrease in performance across all metrics compared to LR. Its strength lies in handling complex interactions and nonlinear relationships, but this may also make it prone to overfitting compared to the simpler LR model. Despite this, it offers valuable insights into feature importance and model robustness. The lower scores in RF might be due to the randomness introduced in the tree-building process, which sometimes increases variance at the expense of bias. This suggests that RF might require further tuning of hyperparameters or a more extensive feature engineering process to improve its predictive performance.

K-Nearest Neighbors (KNN) performs well, with metrics closely trailing LR. It has the second-highest accuracy and a good balance between precision and recall, reflected in its F1-score. KNN relies on the assumption that similar cases with similar feature values are near each other; hence, it performs well if this assumption holds true for the given data. However, KNN can suffer if irrelevant or redundant features are present, or if the feature space is very high-dimensional.

In summary, it's evident that LR provided the most reliable model based on the metrics. Its high recall is particularly important for survival prediction, as it implies a low false negative rate—few survivors are overlooked by the model. The precision is also high, which is crucial to avoid wrongly predicting that a passenger would survive when they actually would not, potentially informing resource allocation during rescue operations.

Part (c)

c) Discuss about the main classification components that you have explored in this practical test and identify the most challenging part of it for future exploration.

(5 marks)

In this practical test, the main classification components explored are the selection of appropriate algorithms, evaluation metrics, and the interpretation of results within the context of the Titanic dataset's features to predict passenger survival. Three machine learning algorithms (LR, RF, KNN) are used for predictions, and Five classification evaluation metrics (Recall, Precision, Accuracy, F1-score, AUROC) are adopted for the model evaluation. Interpretation of the confusion matrices and these metrics was critical to understanding each model's strengths and weaknesses.

The most challenging part, which could be a focus for future exploration, is improving the less-performing models, especially the Random Forest classifier, which did not match the performance of Logistic Regression in this context. This could involve hyperparameter tuning, more advanced feature engineering, or exploring different ensemble techniques. Additionally, dealing with imbalanced data, which is common in survival prediction, could be a challenge. Techniques such as oversampling the minority class or using anomaly detection methods could be explored.

Further exploration could also be directed towards understanding the underlying data distributions and relationships more deeply, perhaps by employing more sophisticated dimensionality reduction techniques or advanced visualization tools. Understanding these relationships better could yield insights that allow for the creation of more nuanced features, which in turn might improve model performance.

Question 5

ITS69304_PracticalExam_0354208.ipynb

```
# Import Necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency # module for Chi-Squared Test

# Modules for pre-processing
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder # Encoder to convert object variables into numerical
from sklearn.preprocessing import StandardScaler # for dataset scaling

# Modules of Machine Learning Models
from sklearn.linear_model import LogisticRegression # Logistic Regression model class
from sklearn.ensemble import RandomForestClassifier # Random Forest Classifier class
from sklearn.neighbors import KNeighborsClassifier # K-Nearest Neighbors Classifier class

# Modules for Confusion Matrix plot generation
from sklearn.metrics import ConfusionMatrixDisplay, classification_report, confusion_matrix

# Modules for evaluation metrics
from sklearn.metrics import recall_score, precision_score, f1_score, accuracy_score, roc_auc_score

# Module to ignore the non-crucial warning messages on console
import warnings
warnings.filterwarnings("ignore")
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive', force_remount = True)
```

```
Mounted at /content/drive
```

Data Collection

```
# Load datasets and create a DataFrame object for each
df_train = pd.read_csv("/content/drive/MyDrive/Taylor's university lesson/Data Analytics and Machine Learning/Mid Term
df_test = pd.read_csv("/content/drive/MyDrive/Taylor's university lesson/Data Analytics and Machine Learning/Mid Term/
df_gs = pd.read_csv("/content/drive/MyDrive/Taylor's university lesson/Data Analytics and Machine Learning/Mid Term/ge
```

```
# Since df_test does not have target variable, combine the "Survived" in df_gs
df_test = pd.concat([df_test, df_gs["Survived"]], axis = 1, join = 'inner')
```

```
# Print the first 5 observations in df_train
df_train.head()
```

	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

```
# Print the first 5 observations in df_test
df_test.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	0
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	1
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	0
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	1

Question 1. Exploratory Data Analysis

Descriptive Analysis

```
# Attributes' names, non-null values counts, and data type in df_train
df_train.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
```

```
# Attributes' names, non-null values counts, and data type in df_test
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
 11  Survived     418 non-null    int64  
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

```
# Represents the dimension information of df_train
print("The rows and columns of the train data is ", df_train.shape, ".", sep="")
```

```
# Represents the dimension information of df_test
print("The rows and columns of the test data is ", df_test.shape, ".", sep="")
```

```
The rows and columns of the train data is (891, 12).
The rows and columns of the test data is (418, 12).
```

```
# Statistical information of numerical attributes in df_train
df_train.describe()
```

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

```
# Statistical information of categorical attributes in df_train
df_train.describe(include="O")
```

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Braund, Mr. Owen Harris	male	347082	B96 B98	S
freq	1	577	7	4	644

```
# Statistical information of numerical attributes in df_test
df_test.describe()
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Survived
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000	418.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188	0.363636
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576	0.481622
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800	0.000000
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200	0.000000
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000	1.000000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200	1.000000

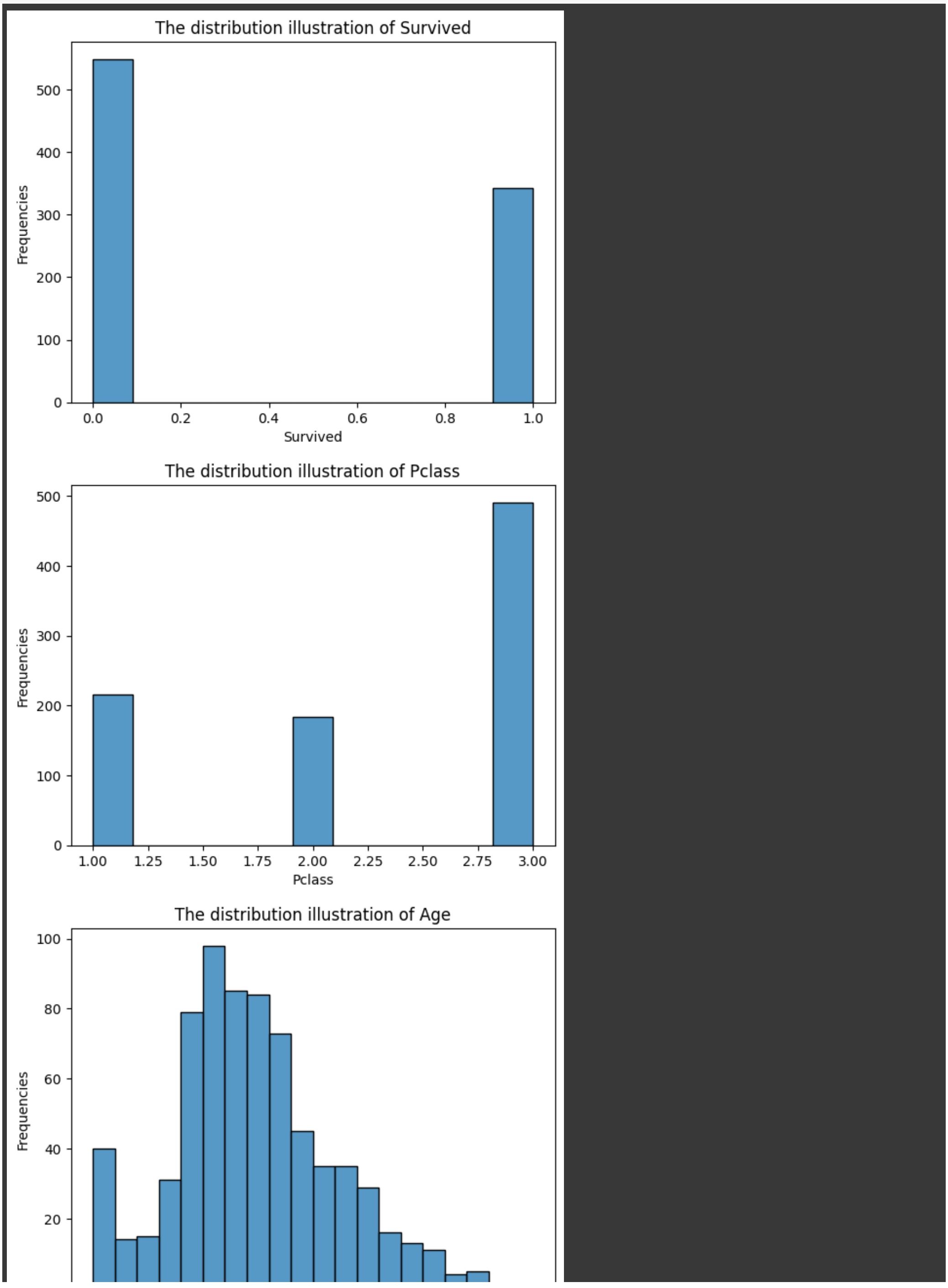
```
# Statistical information of categorical attributes in df_test
df_test.describe(include="O")
```

	Name	Sex	Ticket	Cabin	Embarked
count	418	418	418	91	418
unique	418	2	363	76	3
top	Kelly, Mr. James	male	PC 17608	B57 B59 B63 B66	S
freq	1	266	5	3	270

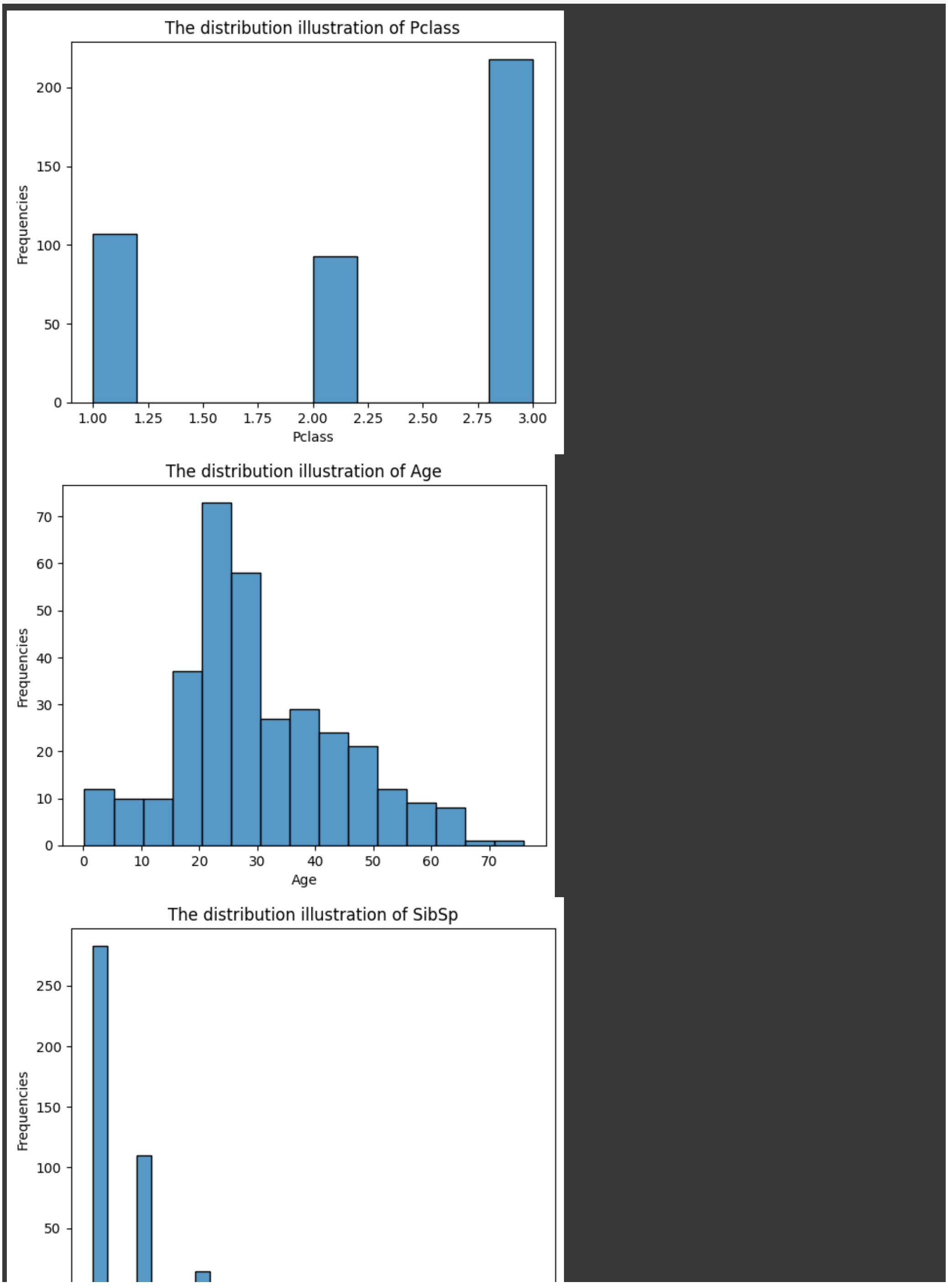
Analysis with Visualizations

```
# Determine numerical and categorical features in df_train for Visualization
train_numerical = df_train.drop(columns = ["PassengerId", "Name", "Ticket", "Sex", "Embarked"])
train_categorical = df_train[["Sex", "Embarked"]]

# Determine numerical and categorical features in df_test for Visualization
test_numerical = df_test.drop(columns = ["PassengerId", "Name", "Ticket", "Sex", "Embarked"])
test_categorical = df_test[["Sex", "Embarked"]]
```

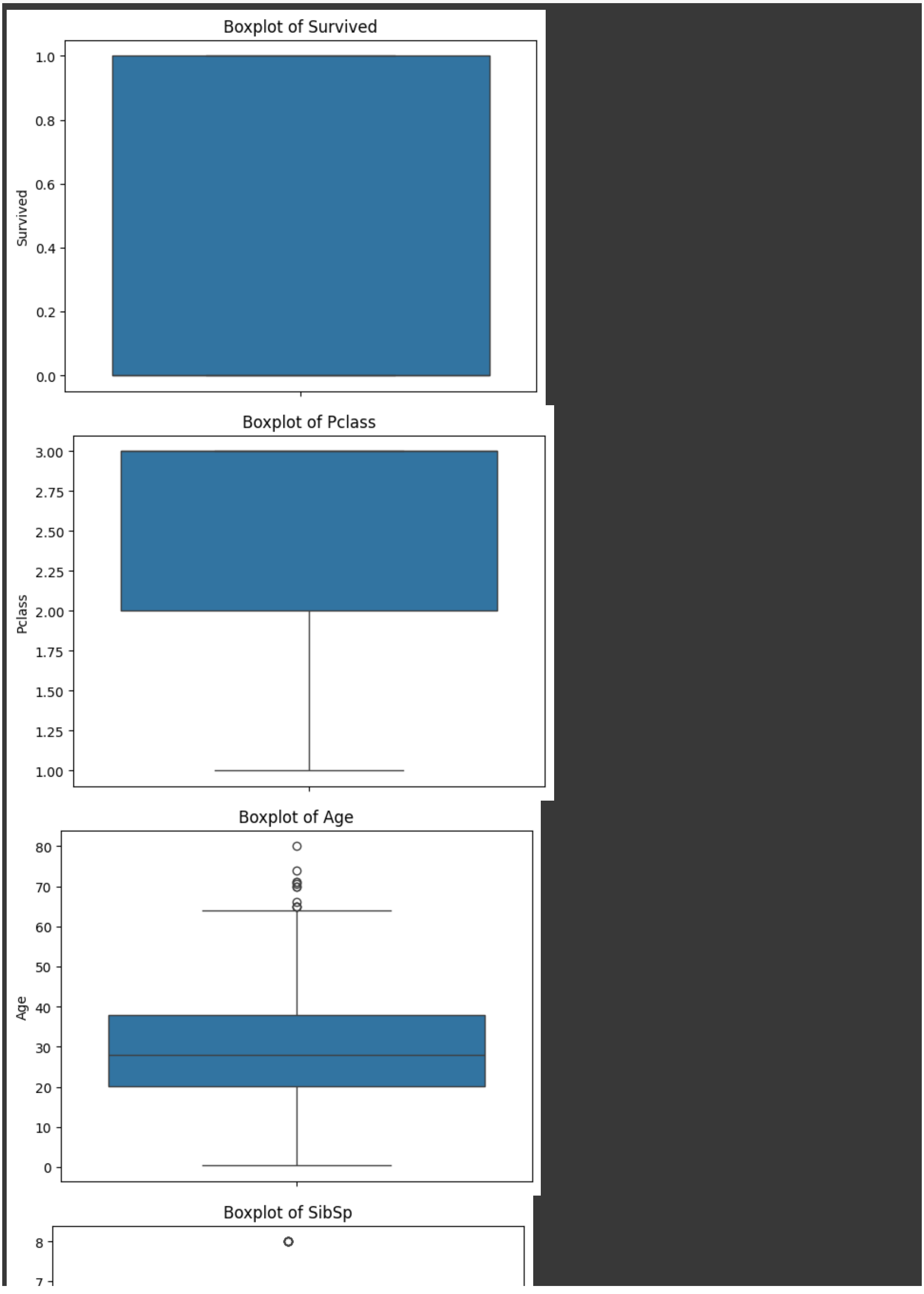
```
# Histogram for numeric variables in train data set
for col in train_numerical:
    if df_train[col].dtype == int or df_train[col].dtype == float:
        sns.histplot(df_train[col])
        plt.xlabel(f"{col}")
        plt.ylabel(f"Frequencies")
        plt.title(f"The distribution illustration of {col}")
        plt.show()
```



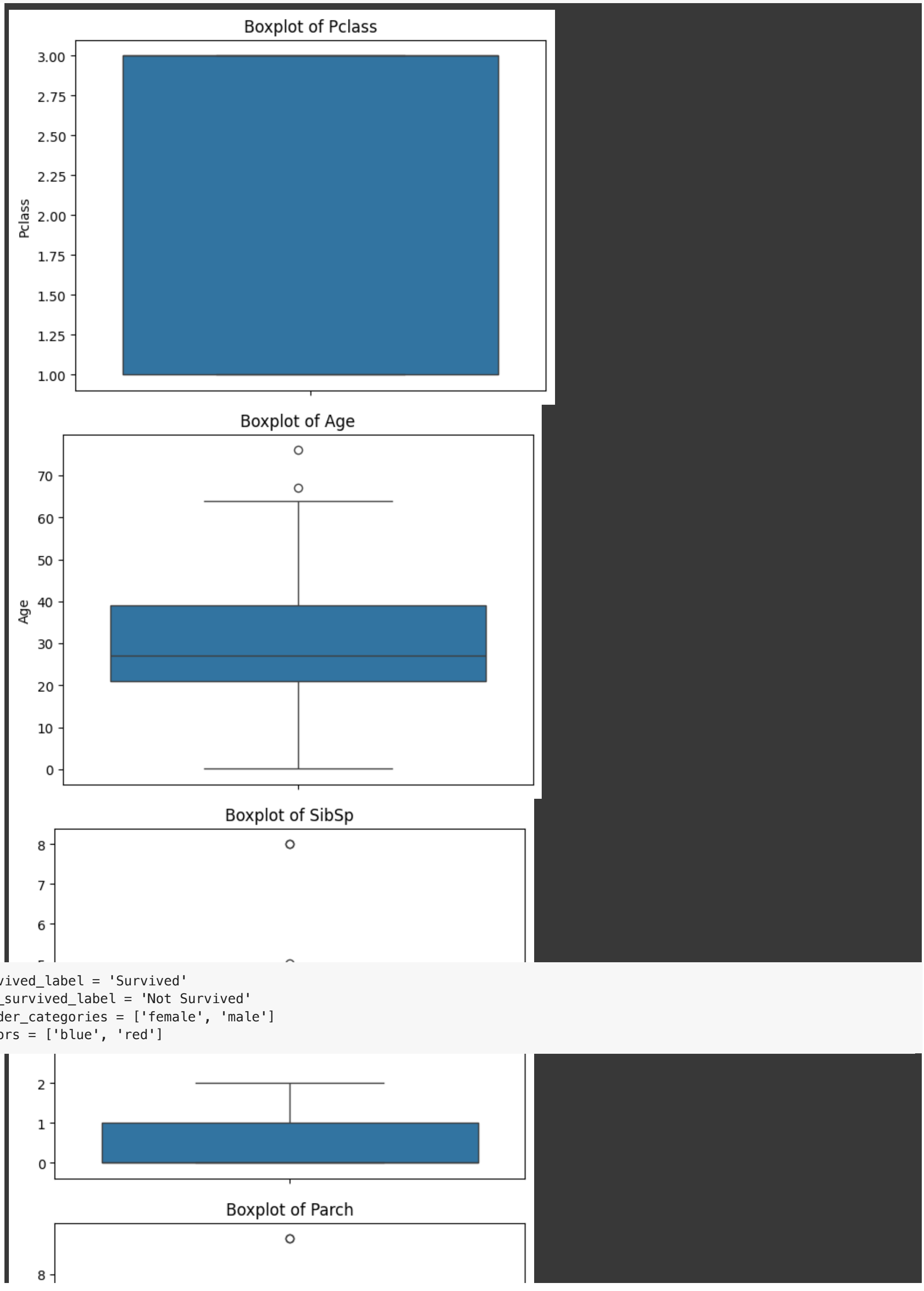
```
# Histogram for numeric variables in test data set
for col in test_numerical:
    if df_test[col].dtype == int or df_test[col].dtype == float:
        sns.histplot(df_test[col])
        plt.xlabel(f"{col}")
        plt.ylabel(f"Frequencies")
        plt.title(f"The distribution illustration of {col}")
        plt.show()
```



```
# Box plot for numeric variables in train data set
for col in train_numerical:
    if df_train[col].dtype == int or df_train[col].dtype == float:
        sns.boxplot(df_train[col])
        plt.title(f"Boxplot of {col}")
        plt.show()
```

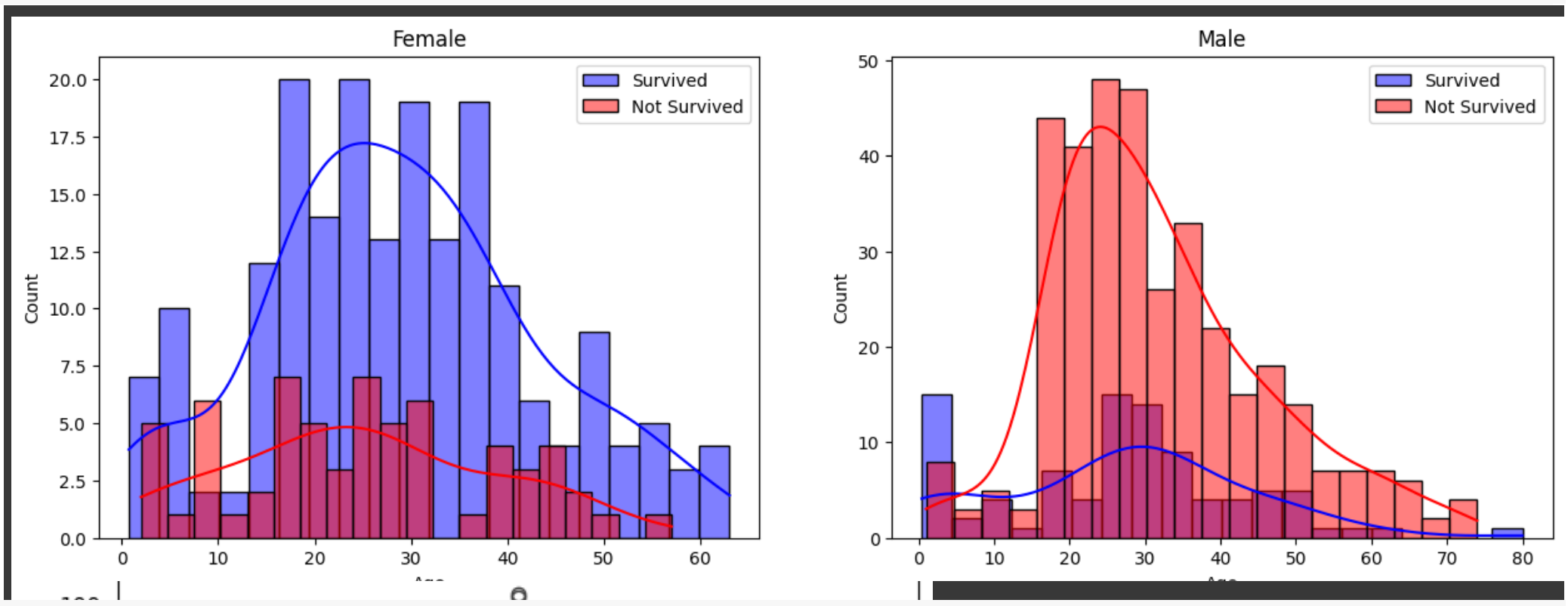


```
# Box plot for numeric variables in test data set
for col in test_numerical:
    if df_test[col].dtype == int or df_test[col].dtype == float:
        sns.boxplot(df_test[col])
        plt.title(f"Boxplot of {col}")
        plt.show()
```

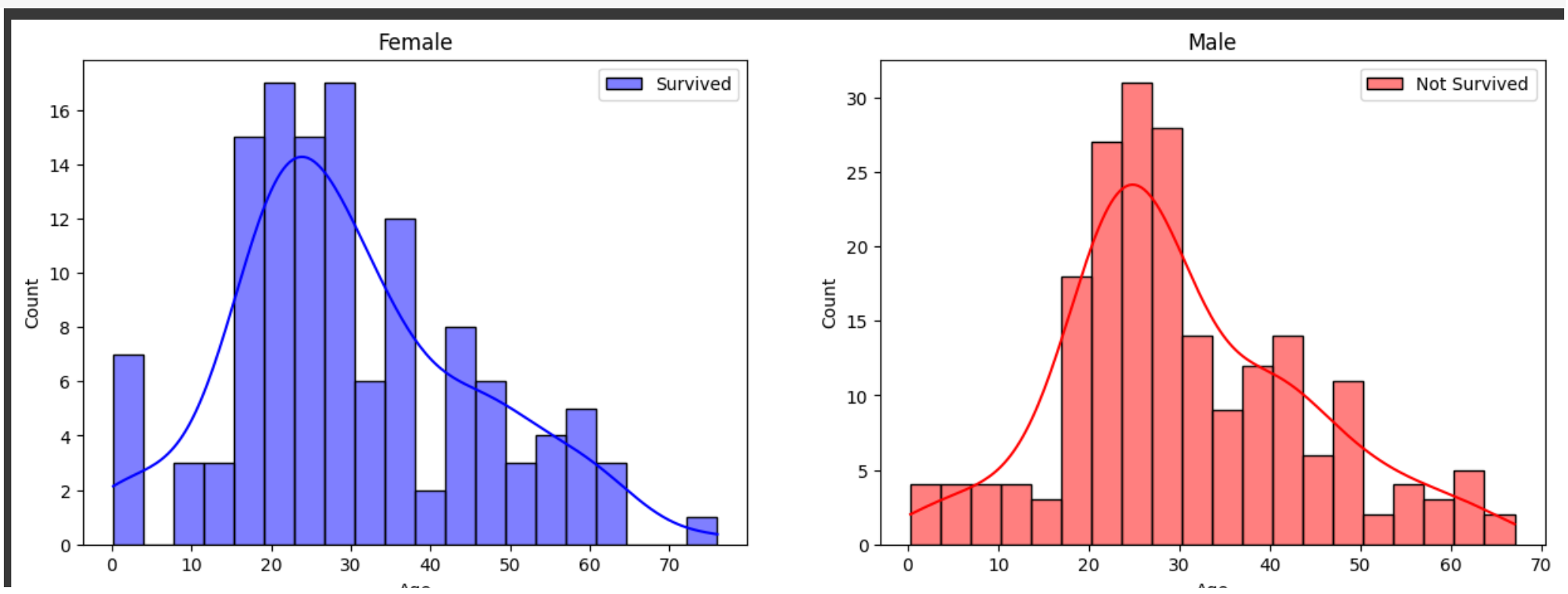



```
survived_label = 'Survived'
not_survived_label = 'Not Survived'
gender_categories = ['female', 'male']
colors = ['blue', 'red']
```

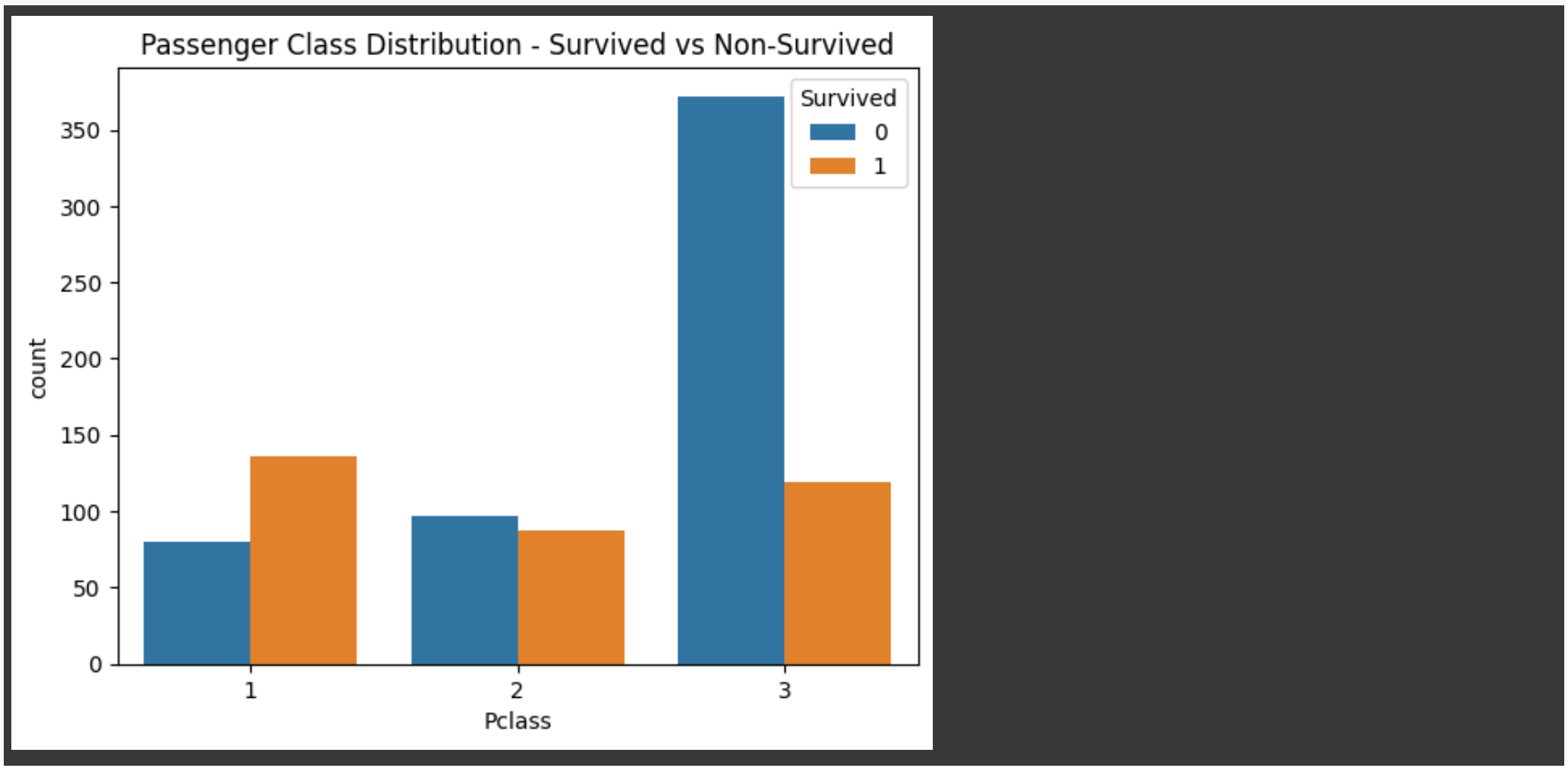
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

# Loop through each gender and plot
for i, gender in enumerate(gender_categories):
    subset = df_train[df_train['Sex'] == gender]

    for survived, color in zip([1, 0], colors):
        sns.histplot(subset[subset['Survived'] == survived]['Age'].dropna(), bins=20, label=f"{survived_la

    axes[i].legend()
    axes[i].set_title(f"{gender.capitalize()}")

plt.show()
```



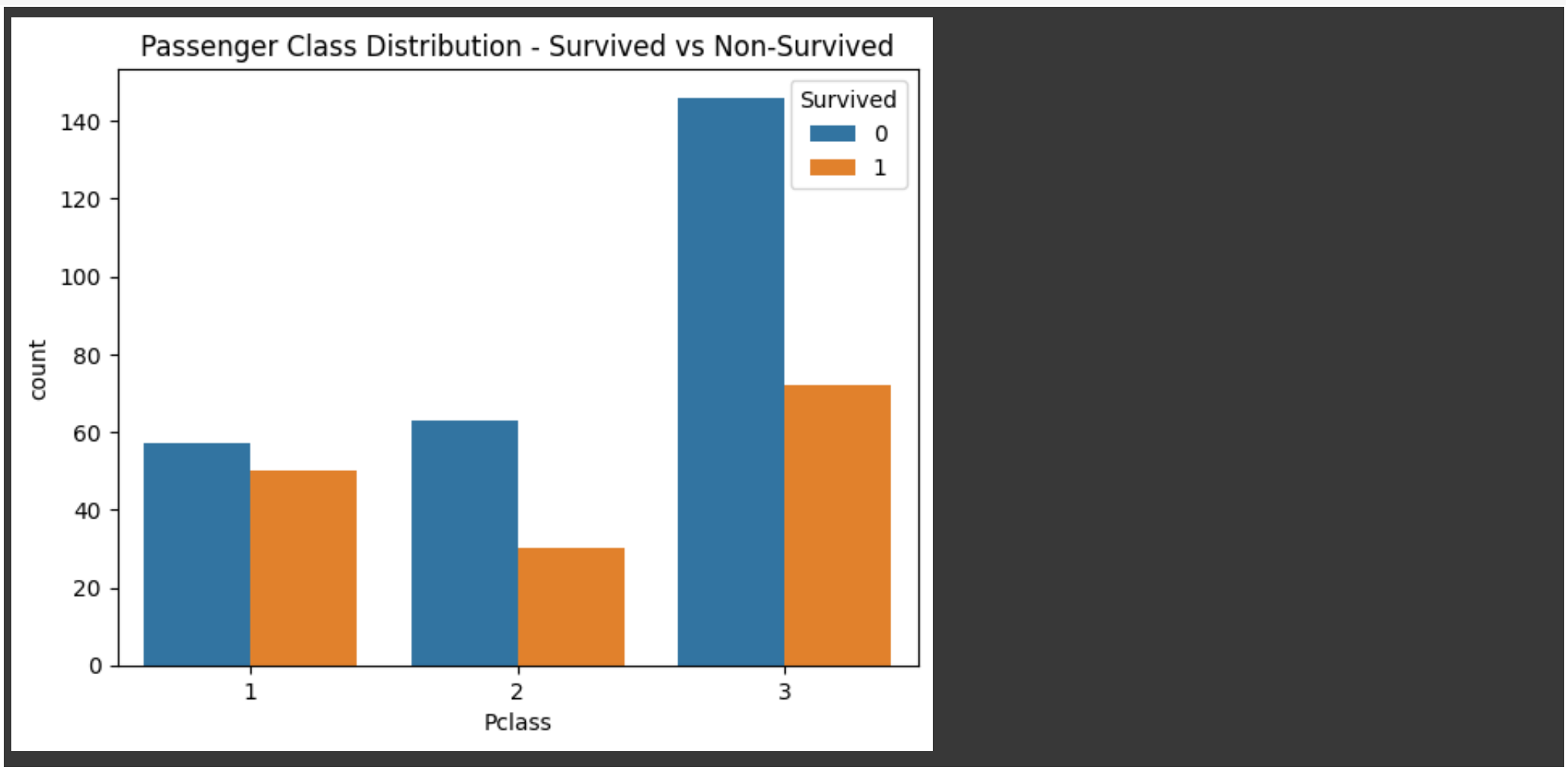

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

# Loop through each gender and plot
for i, gender in enumerate(gender_categories):
    subset = df_test[df_test['Sex'] == gender]

    for survived, color in zip([1, 0], colors):
        sns.histplot(subset[subset['Survived'] == survived]['Age'].dropna(), bins=20, label=f"{survived_la

    axes[i].legend()
    axes[i].set_title(f"{gender.capitalize()}")

plt.show()
```



```
# distribution of passengers survived versus non-survived, categorized by their ticket class on the Titanic in df_trai
sns.countplot(x = 'Pclass', hue = 'Survived', data = df_train)
plt.title("Passenger Class Distribution - Survived vs Non-Survived");
```




```
# distribution of passengers survived versus non-survived, categorized by their ticket class on the Titanic in df_test
sns.countplot(x = 'Pclass', hue = 'Survived', data = df_test)
plt.title("Passenger Class Distribution - Survived vs Non-Survived");
```



Question 2. Data Pre-Processing

(a) pre-process the data with tools

Data Cleaning

Remove unnecessary columns for the model training.

```
# Drop unnecessary attributes from both df_train and df_test
df_train = df_train.drop(columns = ["PassengerId", "Name", "Ticket"])
df_test = df_test.drop(columns = ["PassengerId", "Name", "Ticket"])
```

```
# Check the number of missing values of each attribute belongs to the dataset
df_train.isna().sum()
```

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

```
# Check the number of missing values of each attribute belongs to the dataset
df_test.isna().sum()
```

```
Pclass        0
Sex           0
Age          86
SibSp         0
Parch         0
Fare          1
Cabin       327
Embarked      0
Survived      0
dtype: int64
```

Impute N/A in Train Data Set since our EDA would be messed up without data cleaning

```
# Drop the column if the column has more than 50% of missing values in df_train
for col in df_train:
    if df_train[col].isna().sum() >= (df_train.shape[0] / 2):
        df_train = df_train.drop(columns = col)

# Drop the column if the column has more than 50% of missing values in df_test
for col in df_test:
    if df_test[col].isna().sum() >= (df_test.shape[0] / 2):
        df_test = df_test.drop(columns = col)
```

```
# Check whether all those critically missing attributes are demolished in train data
df_train.isna().sum()
```

```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

```
# Check whether all those critically missing attributes are demolished in test data
df_test.isna().sum()
```

```
Pclass       0
Sex          0
Age         86
SibSp        0
Parch        0
Fare         1
Embarked     0
Survived     0
dtype: int64
```

```
# If, the attribute's data type is object: fill in N/A with mode
for col in df_train:
    if df_train[col].isna().sum() == 0:
        continue
    else:
        if df_train[col].dtype == object:
            df_train[col].fillna(df_train[col].mode()[0], inplace = True)

# Fill in missing values in "Age" attribute in train data
df_train["Age"].fillna(df_train["Age"].mean(), inplace = True)

# If, the attribute's data type is object: fill in N/A with mode
for col in df_test:
    if df_test[col].isna().sum() == 0:
        continue
    else:
        if df_test[col].dtype == object:
            df_test[col].fillna(df_test[col].mode()[0], inplace = True)

# Fill in missing values in "Age" and "Fare" attributes in test data
df_test["Age"].fillna(df_test["Age"].mean(), inplace = True)
df_test["Fare"].fillna(df_test["Fare"].median(), inplace = True)
```

```
# Check whether the entire N/A values are successfully filled in train data
df_train.isna().sum()
```

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

```
# Check whether the entire N/A values are successfully filled in train data
df_test.isna().sum()
```

```
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
Survived    0
dtype: int64
```

Encoding Categorical Variables

```
# Create an object of One-Hot Encoder and Ordinal Encoder
ohe = OneHotEncoder()
oe = OrdinalEncoder()
```

```
## Encode each attribute with its own suitable encoder (Train data)

# One-Hot Encoding ("Sex", "Embarked")
ohe_data = pd.DataFrame(ohe.fit_transform(df_train[["Sex", "Embarked"]]).toarray())
df_train = df_train.join(ohe_data).drop(columns = ["Sex", "Embarked"])

# Ordinal Encoding ("Pclass")
df_train[["Pclass"]] = oe.fit_transform(df_train[["Pclass"]])

## Encode each attribute with its own suitable encoder (Test data)

# One-Hot Encoding ("Sex", "Embarked")
ohe_data = pd.DataFrame(ohe.fit_transform(df_test[["Sex", "Embarked"]]).toarray())
df_test = df_test.join(ohe_data).drop(columns = ["Sex", "Embarked"])

# Ordinal Encoding ("Pclass")
df_test[["Pclass"]] = oe.fit_transform(df_test[["Pclass"]])
```

```
# Convert the all column names data type into string (Train data)
df_train.columns = df_train.columns.astype(str)

# Convert the float type features into integer
for col in df_train:
    if df_train[col].dtype == float:
        df_train[col] = df_train[col].astype(int)

# Convert the all column names data type into string (Test data)
df_test.columns = df_test.columns.astype(str)

# Convert the float type features into integer
for col in df_test:
    if df_test[col].dtype == float:
        df_test[col] = df_test[col].astype(int)
```

```
# Confirm all existing attributes are entirely fulfilled, as integer type in df_train
df_train.info()
```

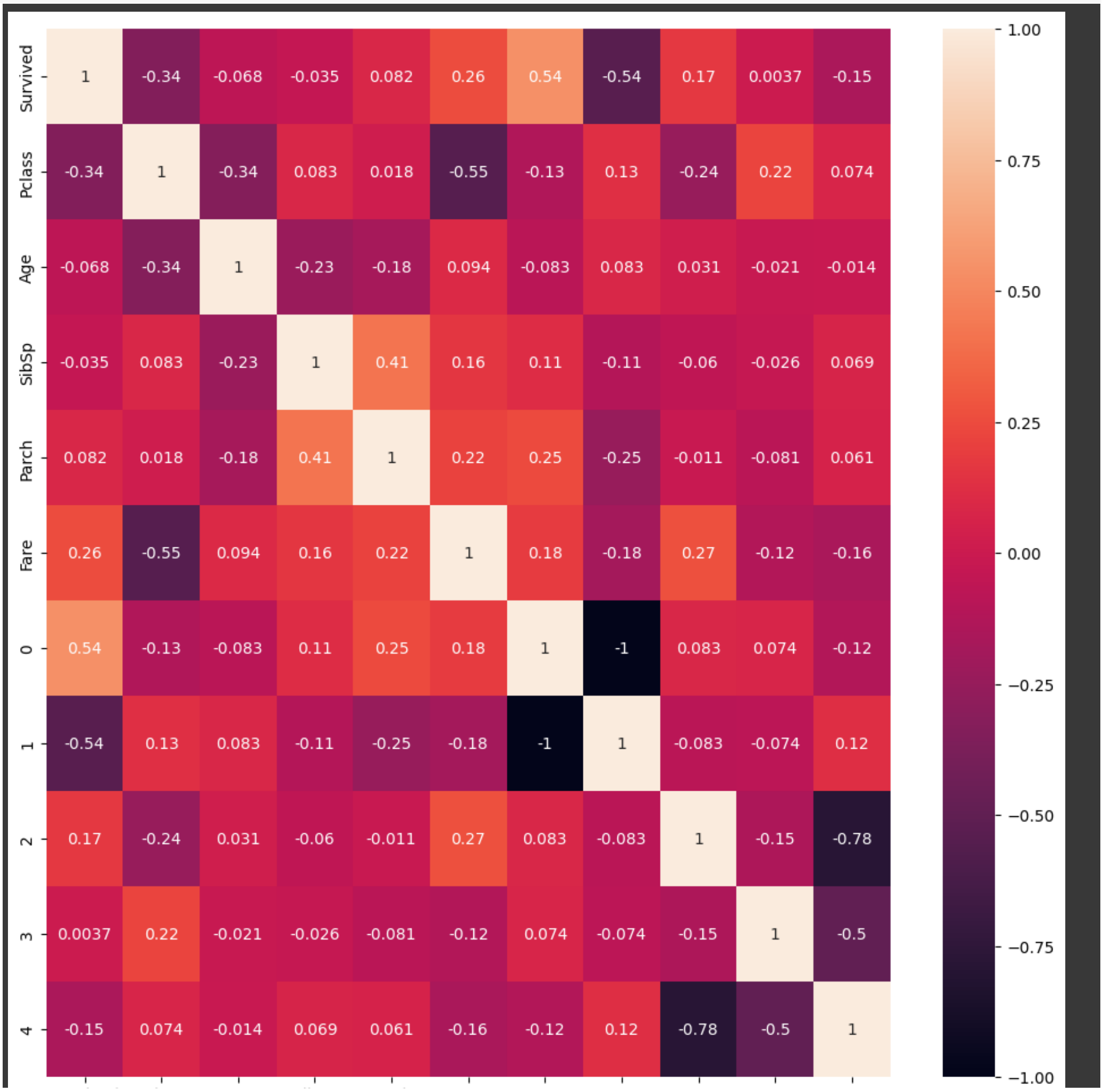
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Age       891 non-null    int64
 3   SibSp     891 non-null    int64
 4   Parch     891 non-null    int64
 5   Fare      891 non-null    int64
 6   0         891 non-null    int64
 7   1         891 non-null    int64
 8   2         891 non-null    int64
 9   3         891 non-null    int64
 10  4         891 non-null    int64
dtypes: int64(11)
memory usage: 76.7 KB
```

```
# Confirm all existing attributes are entirely fulfilled, as integer type in df_test
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    418 non-null    int64
 1   Age       418 non-null    int64
 2   SibSp     418 non-null    int64
 3   Parch     418 non-null    int64
 4   Fare      418 non-null    int64
 5   Survived  418 non-null    int64
 6   0         418 non-null    int64
 7   1         418 non-null    int64
 8   2         418 non-null    int64
 9   3         418 non-null    int64
 10  4         418 non-null    int64
dtypes: int64(11)
memory usage: 36.0 KB
```

(b) Create a correlation heatmap after pre-processing

```
# Heatmap illustrates the entire correlation matrix
plt.subplots(figsize = (12, 12))
sns.heatmap(df_train.corr(), annot = True)
plt.show()
```



The `Fare` attribute shows a positive correlation with "Survived," suggesting that passengers who paid higher fares were more likely to survive. This could be because higher fares might correspond to higher classes, which had better access to lifeboats.

The `Pclass` attribute has a negative correlation with "Survived." This indicates that passengers in higher classes (where 1st class is the highest and 3rd class is the lowest) were more likely to survive, which is consistent with historical accounts of the Titanic disaster.

The `0` and `1` attributes, which are the “Sex” attribute components encoded with One-Hot Encoder, would likely be very significant based on historical accounts. This could be because female passengers had a higher survival rate.

The 2 and 4 attributes are also having relatively higher correlations with “Survived”.

Therefore,

- Fare
- Pclass
- 0
- 1
- 2
- 4

would be deployed to predict the outcome of Survived.

Question 3. Model Training

Split the Data, and Scale with StandardScaler

```
# Determine the independent variables and target variable
X_train = df_train[["Fare", "Pclass", "0", "1", "2", "4"]]
y_train = df_train["Survived"]
X_test = df_test[["Fare", "Pclass", "0", "1", "2", "4"]]
y_test = df_test["Survived"]
```

```
# Feature scaling to bring all features to the same scale
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

Logistic Regression

```
# Initializing the Logistic Regression model
LR = LogisticRegression(max_iter = 10000)

# Fitting the model with training data
LR.fit(X_train, y_train)

# Making predictions on the test data
lr_pred = LR.predict(X_test)
```

Random Forest

```
# Initializing the Logistic Regression model
RFC = RandomForestClassifier()

# Fitting the model with training data
RFC.fit(X_train, y_train)

# Making predictions on the test data
rfc_pred = RFC.predict(X_test)
```

K-Nearest Neighbors

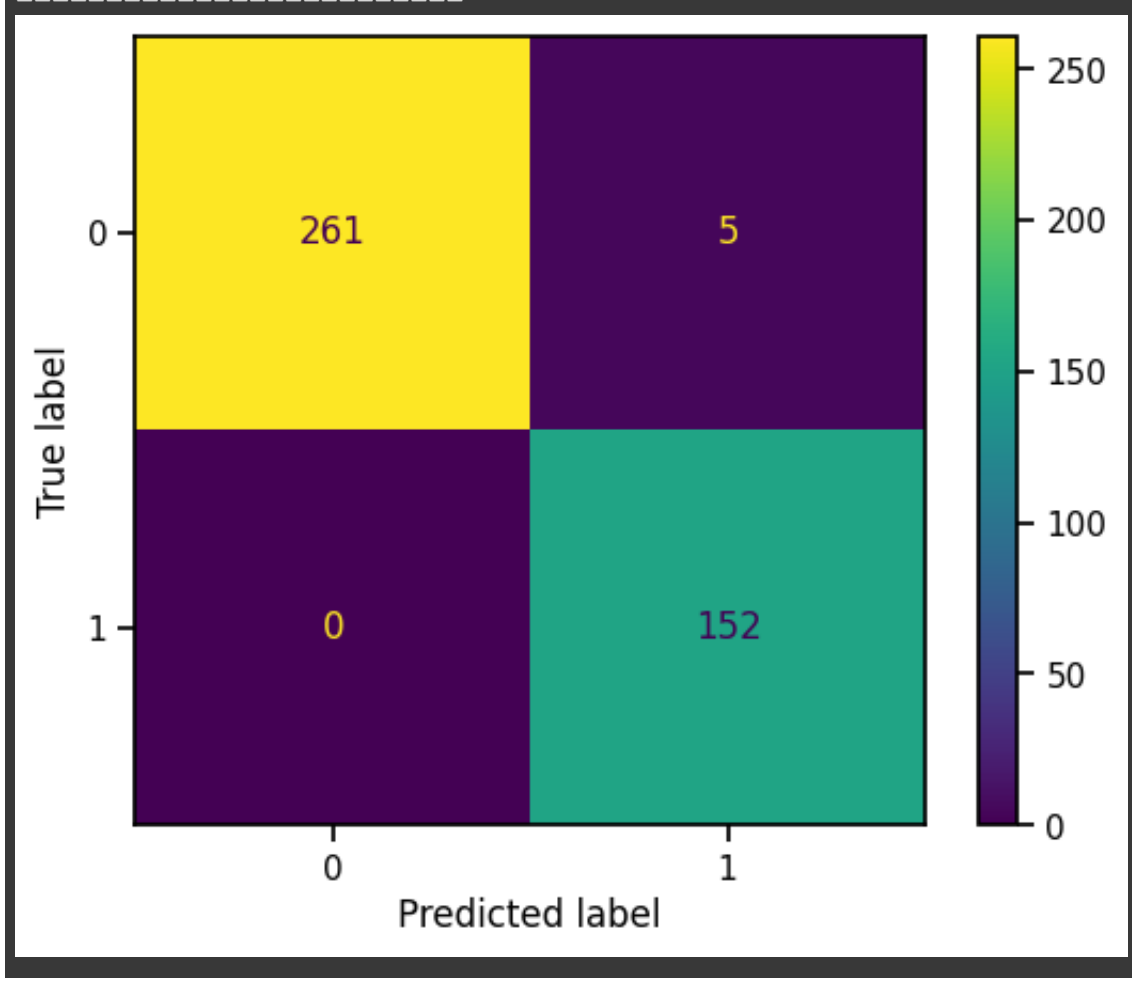
```
# Initializing the Logistic Regression model
KNC = KNeighborsClassifier()

# Fitting the model with training data
KNC.fit(X_train, y_train)

# Making predictions on the test data
knc_pred = KNC.predict(X_test)
```

Question 4. Evaluation and Future Perspective

Logistic Regression Evaluation

```
# Evaluate the Train set
print(classification_report(y_test, lr_pred))
lr_cm = confusion_matrix(y_test, lr_pred)
sns.set_context("notebook")
ConfusionMatrixDisplay(confusion_matrix = lr_cm, display_labels = LR.classes_).plot()

# Evaluation Scores
roc_auc = round(roc_auc_score(y_test, lr_pred), 4)
accuracy = round(accuracy_score(y_test, lr_pred), 4)
recall = round(recall_score(y_test, lr_pred), 4)
precision = round(precision_score(y_test, lr_pred), 4)
f_one = round(f1_score(y_test, lr_pred), 4)

# Return evaluation scores
print(f"=========================")
print(f" ROC-AUC Score: {roc_auc}")
print(f" Accuracy Score: {accuracy}")
print(f" Recall Score: {recall}")
print(f" Precision Score: {precision}")
print(f" F1-score: {f_one}")
print(f"=========================")
```

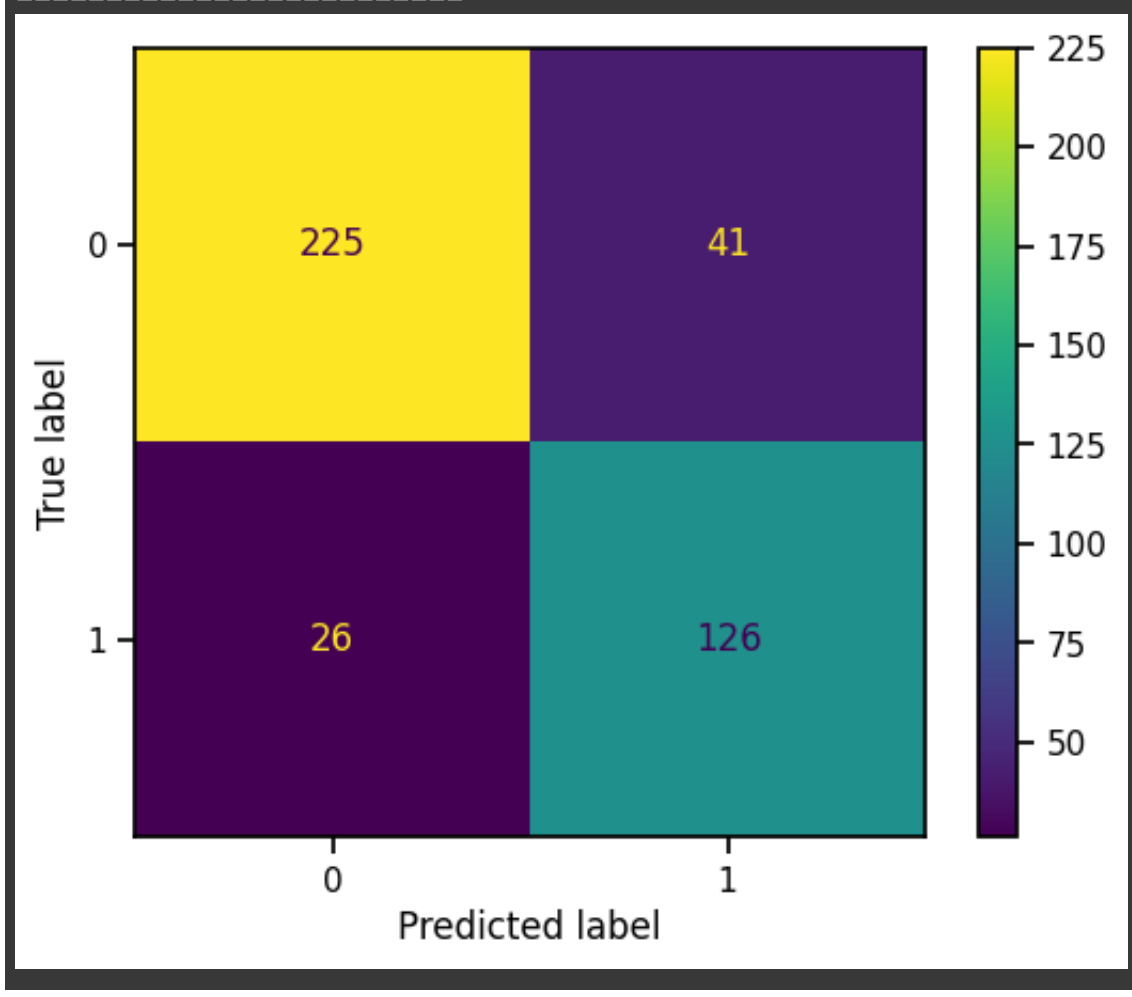
```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       266
           1       0.97      1.00      0.98       152

    accuracy                           0.99       418
   macro avg       0.98      0.99      0.99       418
weighted avg       0.99      0.99      0.99       418

=========================
 ROC-AUC Score: 0.9906
 Accuracy Score: 0.988
 Recall Score: 1.0
 Precision Score: 0.9682
 F1-score: 0.9838
=========================
```



Random Forest Evaluation

```
# Evaluate the Train set
print(classification_report(y_test, rfc_pred))
rfc_cm = confusion_matrix(y_test, rfc_pred)
sns.set_context("notebook")
ConfusionMatrixDisplay(confusion_matrix = rfc_cm, display_labels = RFC.classes_).plot()

# Evaluation Scores
roc_auc = round(roc_auc_score(y_test, rfc_pred), 4)
accuracy = round(accuracy_score(y_test, rfc_pred), 4)
recall = round(recall_score(y_test, rfc_pred), 4)
precision = round(precision_score(y_test, rfc_pred), 4)
f_one = round(f1_score(y_test, rfc_pred), 4)

# Return evaluation scores
print(f"=========================")
print(f" ROC-AUC Score: {roc_auc}")
print(f" Accuracy Score: {accuracy}")
print(f" Recall Score: {recall}")
print(f" Precision Score: {precision}")
print(f" F1-score: {f_one}")
print(f"=========================")
```

```
              precision    recall  f1-score   support

           0       0.90      0.85      0.87       266
           1       0.75      0.83      0.79       152

    accuracy                           0.84       418
   macro avg       0.83      0.84      0.83       418
weighted avg       0.84      0.84      0.84       418

=========================
 ROC-AUC Score: 0.8374
 Accuracy Score: 0.8397
 Recall Score: 0.8289
 Precision Score: 0.7545
 F1-score: 0.79
=========================
```



K-Nearest Neighbors Evaluation

```
# Evaluate the Train set
print(classification_report(y_test, knc_pred))
knc_cm = confusion_matrix(y_test, knc_pred)
sns.set_context("notebook")
ConfusionMatrixDisplay(confusion_matrix = knc_cm, display_labels = KNC.classes_).plot()

# Evaluation Scores
roc_auc = round(roc_auc_score(y_test, knc_pred), 4)
accuracy = round(accuracy_score(y_test, knc_pred), 4)
recall = round(recall_score(y_test, knc_pred), 4)
precision = round(precision_score(y_test, knc_pred), 4)
f_one = round(f1_score(y_test, knc_pred), 4)

# Return evaluation scores
print(f"=========================")
print(f" ROC-AUC Score: {roc_auc}")
print(f" Accuracy Score: {accuracy}")
print(f" Recall Score: {recall}")
print(f" Precision Score: {precision}")
print(f" F1-score: {f_one}")
print(f"=========================")
```

```
              precision    recall  f1-score   support

           0       0.92      0.95      0.93       266
           1       0.90      0.85      0.87       152

    accuracy                           0.91       418
   macro avg       0.91      0.90      0.90       418
weighted avg       0.91      0.91      0.91       418

=========================
 ROC-AUC Score: 0.898
 Accuracy Score: 0.9115
 Recall Score: 0.8487
 Precision Score: 0.9021
 F1-score: 0.8746
=========================
```