TITANIC SURVIVAL PREDICTION

1. **Data Preprocessing**
    a. Load the dataset.
    b. Execute head() to see the first 5 rows of the dataset.

```
[1]  #JaiShreeRam

     #TitanicSurvivalPrediction

     import pandas as pd

     data = pd.read_csv("tested.csv")
```

```
     print(data.head())
```

```
   PassengerId  Survived  Pclass  \
0          892         0       3
1          893         1       3
2          894         0       2
3          895         0       3
4          896         1       3

                                          Name     Sex   Age  SibSp  Parch  \
0                              Kelly, Mr. James    male  34.5      0      0
1              Wilkes, Mrs. James (Ellen Needs)  female  47.0      1      0
2                     Myles, Mr. Thomas Francis    male  62.0      0      0
3                              Wirz, Mr. Albert    male  27.0      0      0
4  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female  22.0      1      1

    Ticket     Fare Cabin Embarked
0   330911   7.8292   NaN        Q
1   363272   7.0000   NaN        S
2   240276   9.6875   NaN        Q
3   315154   8.6625   NaN        S
4  3101298  12.2875   NaN        S
```

c. Execute describe() to see the summary statistics of the numerical column.

```
print(data.describe())
```

```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   418.000000  418.000000  418.000000  332.000000  418.000000
mean   1100.500000    0.363636    2.265550   30.272590    0.447368
std     120.810458    0.481622    0.841838   14.181209    0.896760
min     892.000000    0.000000    1.000000    0.170000    0.000000
25%     996.250000    0.000000    1.000000   21.000000    0.000000
50%    1100.500000    0.000000    3.000000   27.000000    0.000000
75%    1204.750000    1.000000    3.000000   39.000000    1.000000
max    1309.000000    1.000000    3.000000   76.000000    8.000000

             Parch        Fare
count  418.000000  417.000000
mean     0.392344   35.627188
std      0.981429   55.907576
min      0.000000    0.000000
25%      0.000000    7.895800
50%      0.000000   14.454200
75%      0.000000   31.500000
max      9.000000  512.329200
```

d. Check the data types data.info().

```
print(data.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   Survived     418 non-null    int64  
 2   Pclass       418 non-null    int64  
 3   Name         418 non-null    object 
 4   Sex          418 non-null    object 
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object 
 11  Embarked     418 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
None
```

e. Check the missing values.

```
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)
```

```
Missing Values:
 PassengerId      0
Survived          0
Pclass            0
Name              0
Sex               0
Age              86
SibSp             0
Parch             0
Ticket            0
Fare              1
Cabin           327
Embarked          0
dtype: int64
```

There are 86 missing values in the age column which means the information of the age of 86 passengers is missing. There are 327 passengers missing from the cabin.

f. Fill the missing values in the Age and Cabin column by median and unknown.

```
[6]  data['Age'].fillna(data['Age'].median(), inplace=True)
```

```
[7]  data['Cabin'].fillna('Unknown', inplace=True)
```

g. Remove duplicates.

```
[8]  data.drop_duplicates(inplace=True)
```

h. Save the preprocessed dataset.

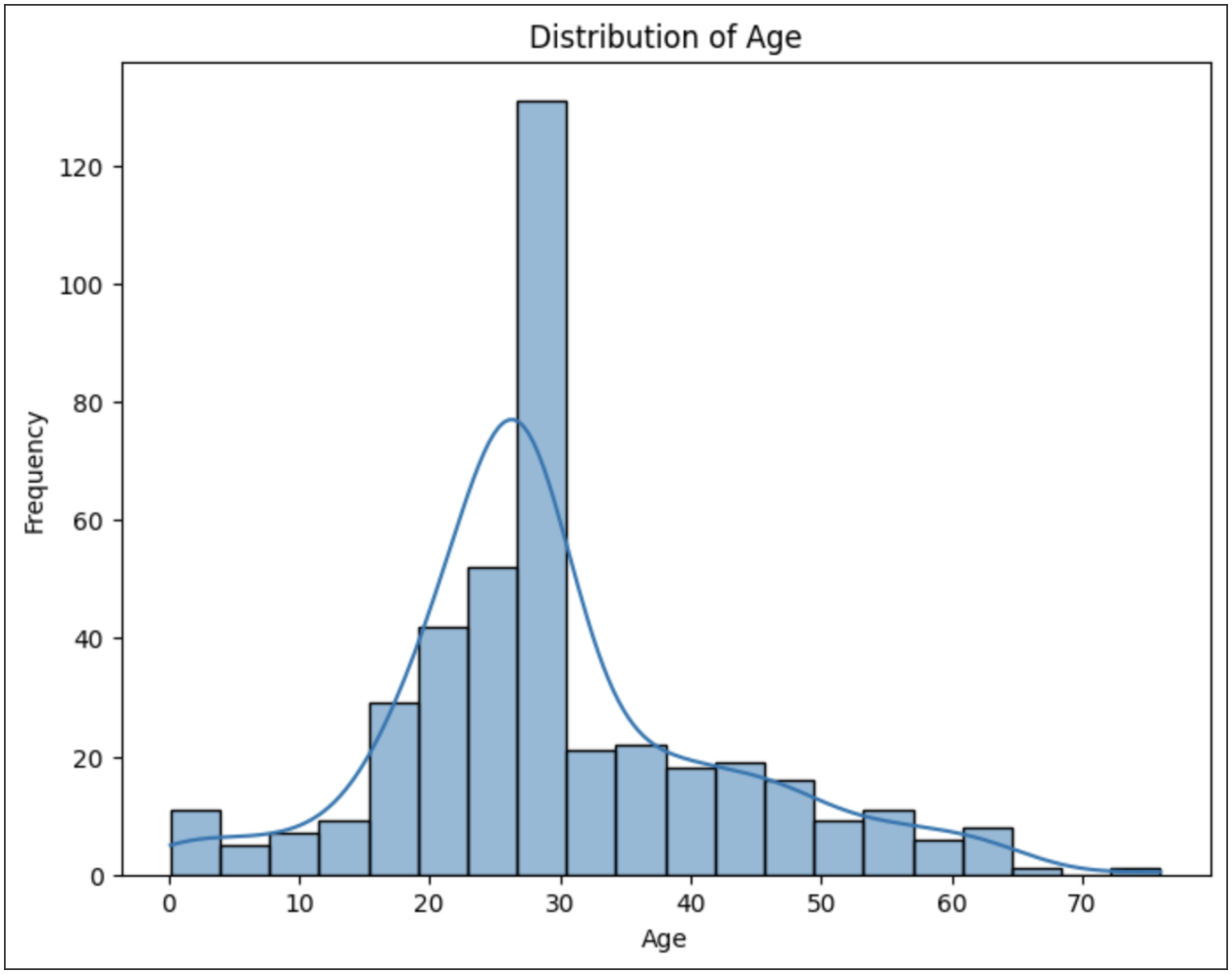
```
data.to_csv('preprocessed_data.csv', index=False)
```

2. Exploratory Data Analysis

a. Univariate Analysis

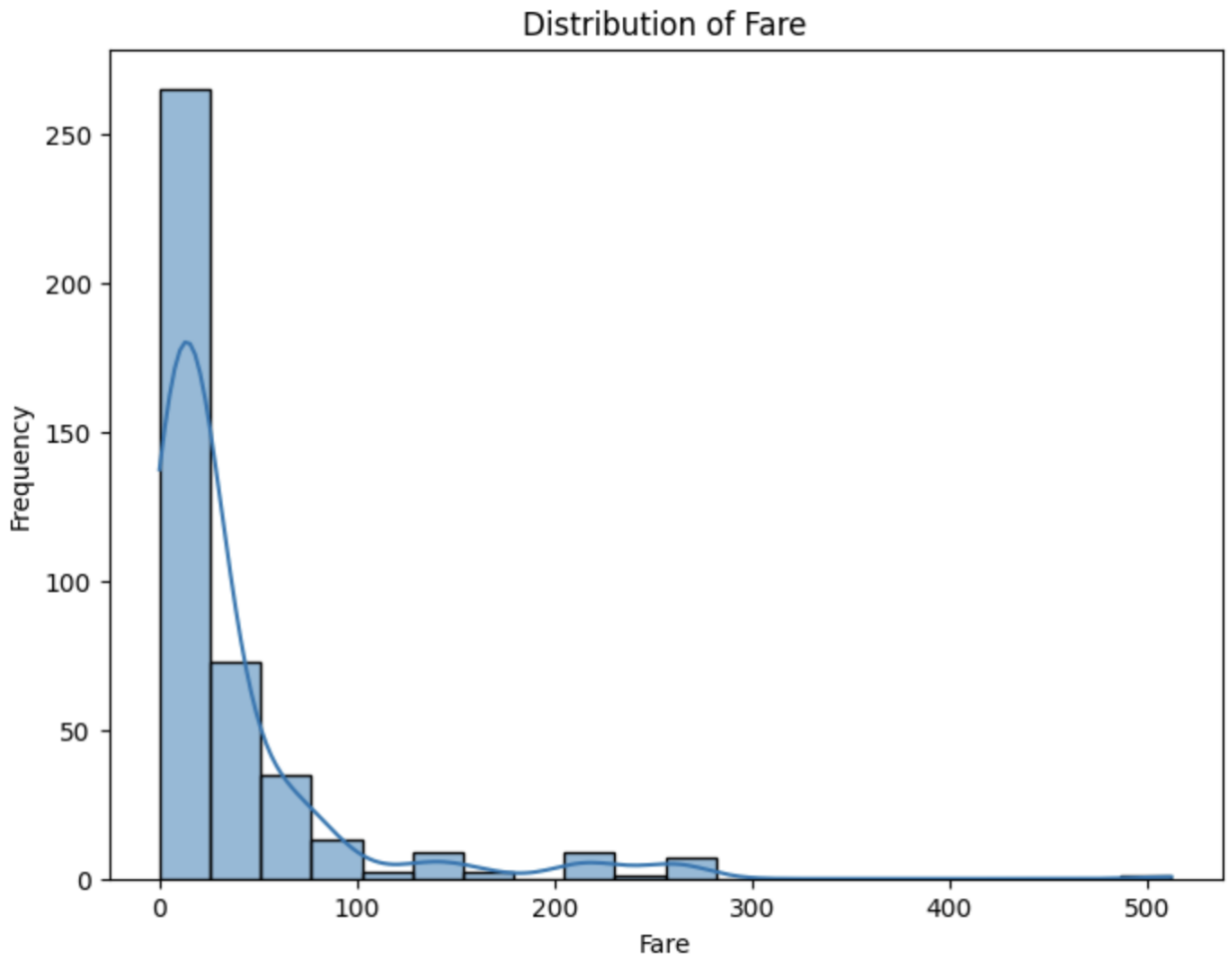
i. Age

```
#Exploratory Data Analysis

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.histplot(data['Age'], bins=20, kde=True)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Distribution of Age')
plt.show()
```



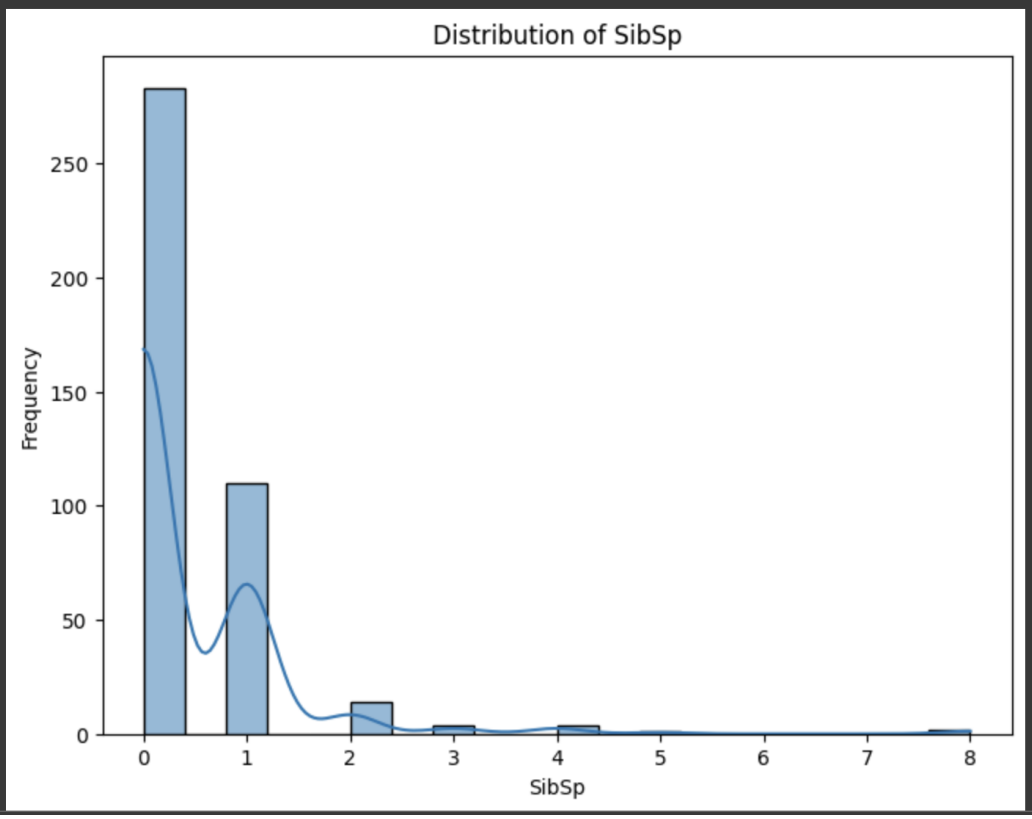
ii. Fare

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


plt.figure(figsize=(8, 6))
sns.histplot(data['Fare'], bins=20, kde=True)
plt.xlabel('Fare')
plt.ylabel('Frequency')
plt.title('Distribution of Fare')
plt.show()
```



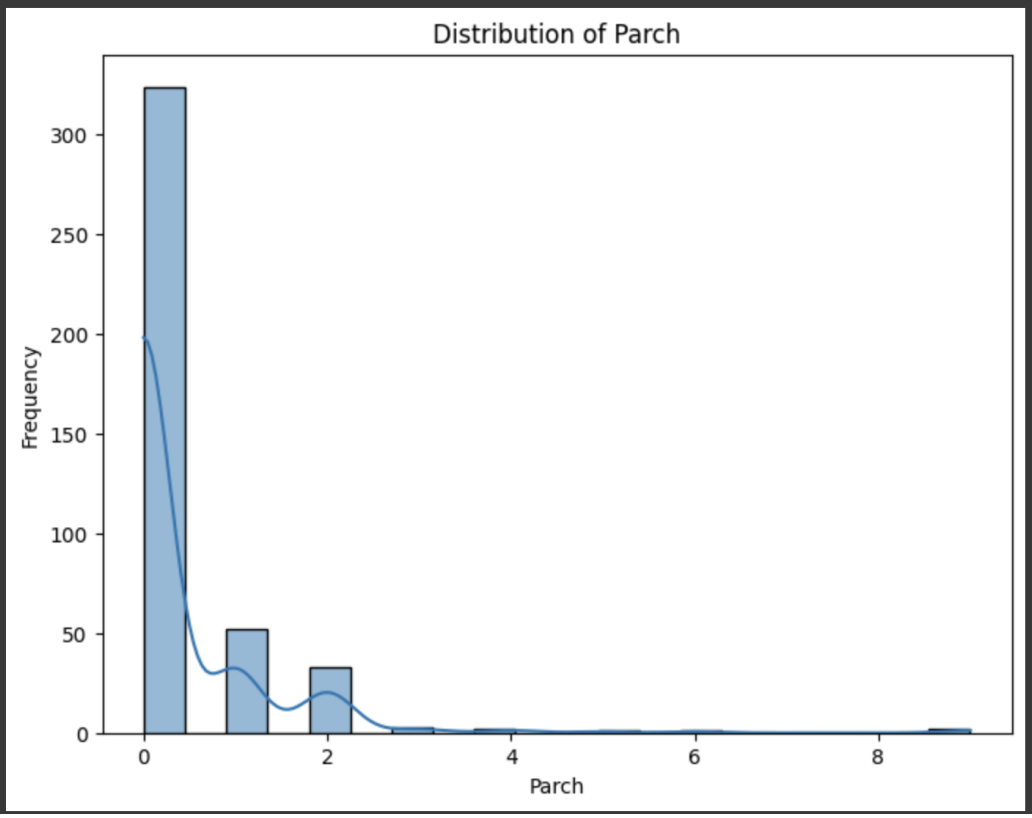
iii. SibSp

```python
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.histplot(data['SibSp'], bins=20, kde=True)
plt.xlabel('SibSp')
plt.ylabel('Frequency')
plt.title('Distribution of SibSp')
plt.show()
```



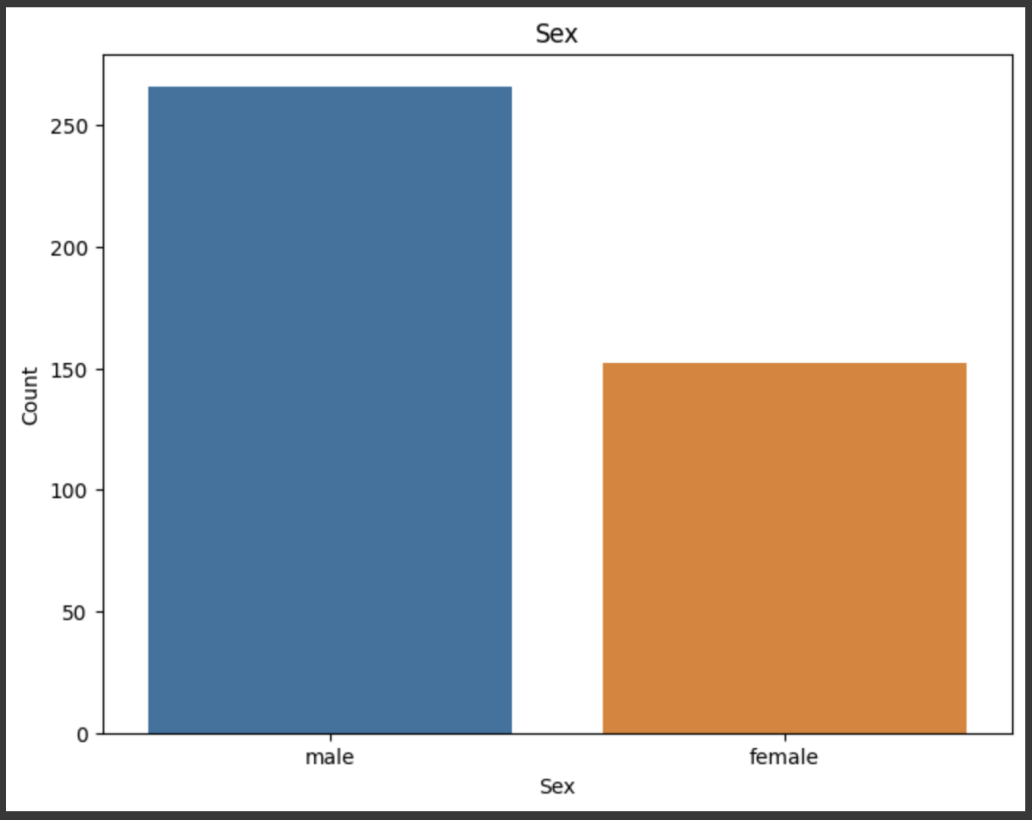
iv. Parch

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.histplot(data['Parch'], bins=20, kde=True)
plt.xlabel('Parch')
plt.ylabel('Frequency')
plt.title('Distribution of Parch')
plt.show()
```



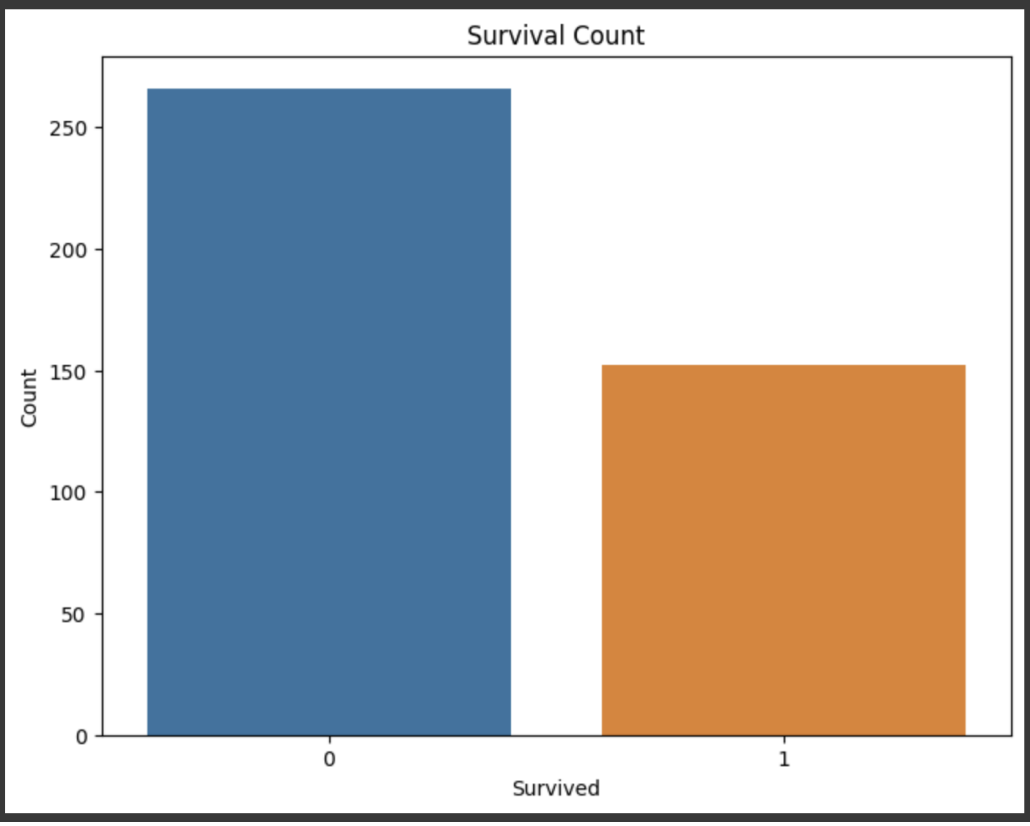
v. Sex

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Sex', data=data)
plt.xlabel('Sex')
plt.ylabel('Count')
plt.title('Sex')
plt.show()
```



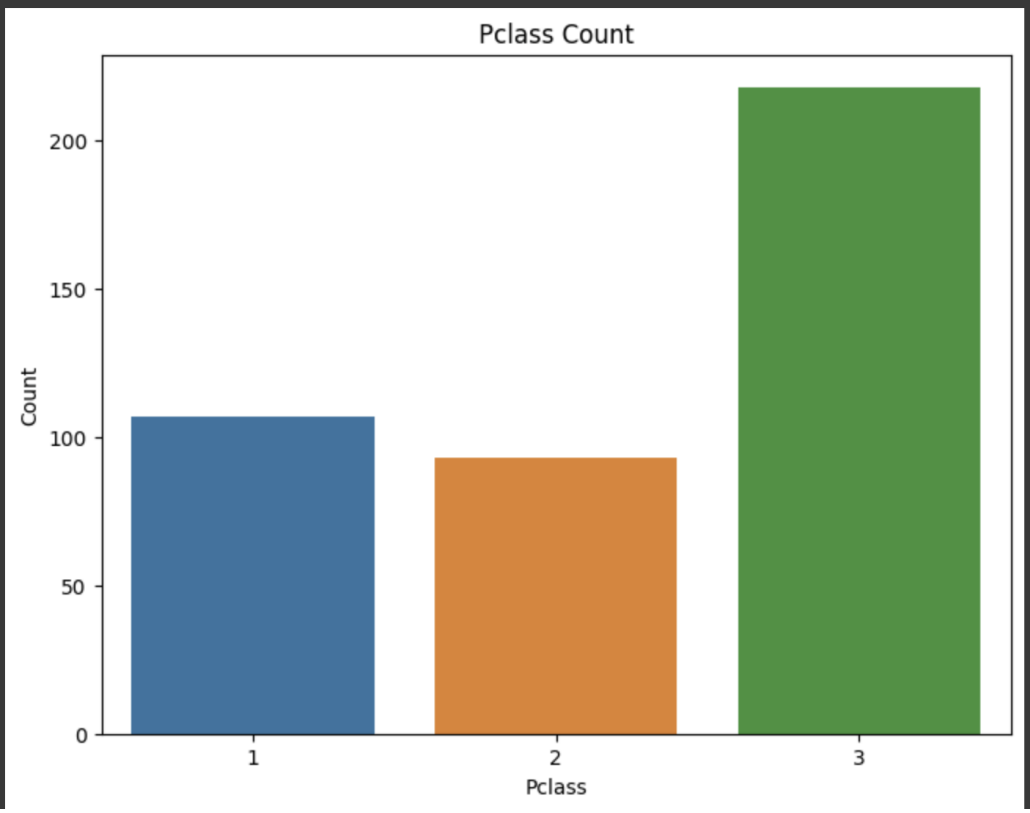
vi. Survived

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Survived', data=data)
plt.xlabel('Survived')
plt.ylabel('Count')
plt.title('Survival Count')
plt.show()
```



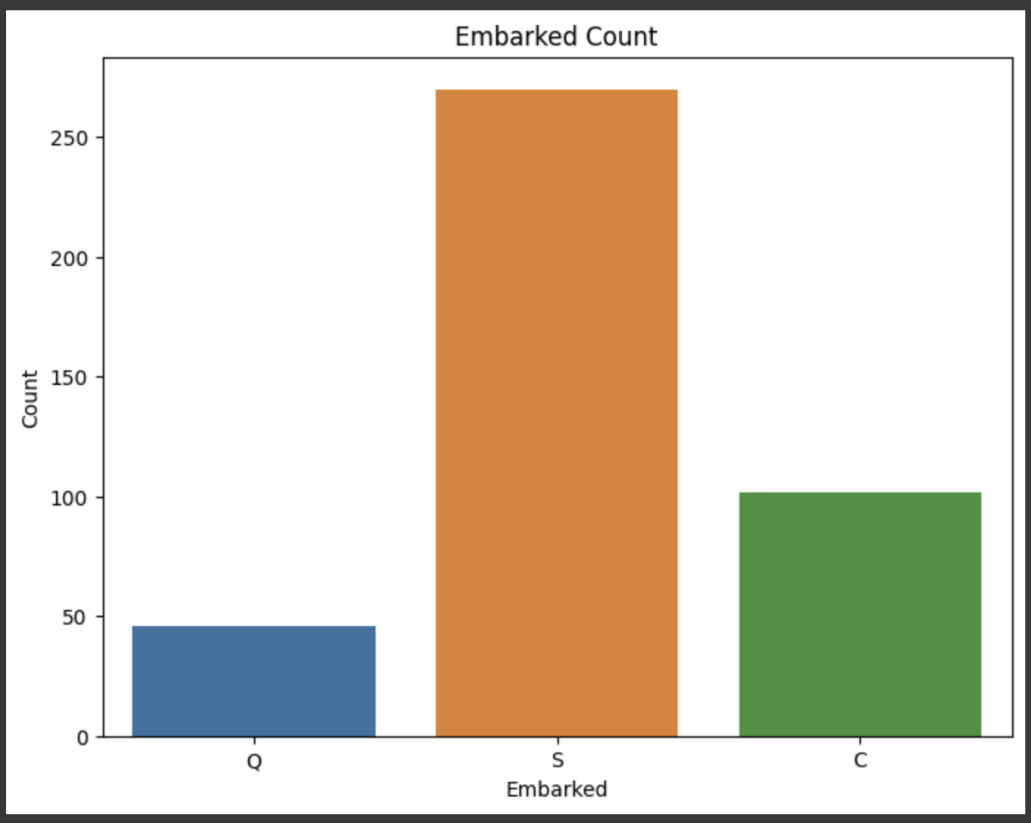
vii. Pclass

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Pclass', data=data)
plt.xlabel('Pclass')
plt.ylabel('Count')
plt.title('Pclass Count')
plt.show()
```



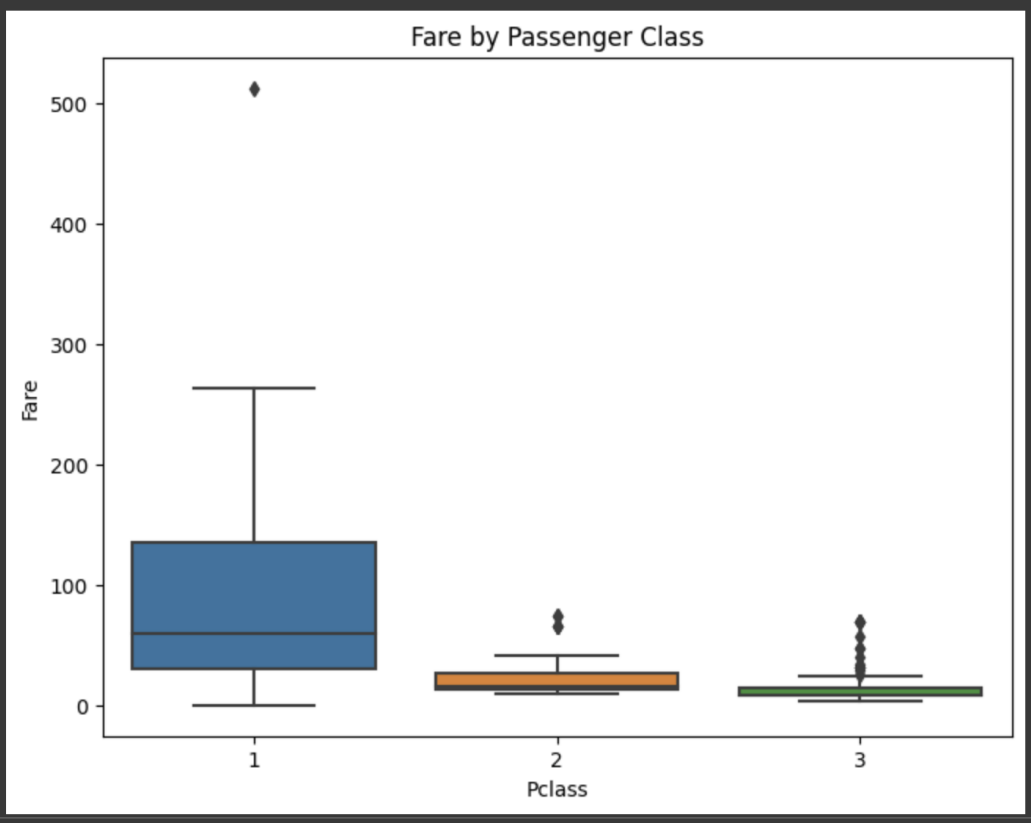
viii. Embarked

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Embarked', data=data)
plt.xlabel('Embarked')
plt.ylabel('Count')
plt.title('Embarked Count')
plt.show()
```



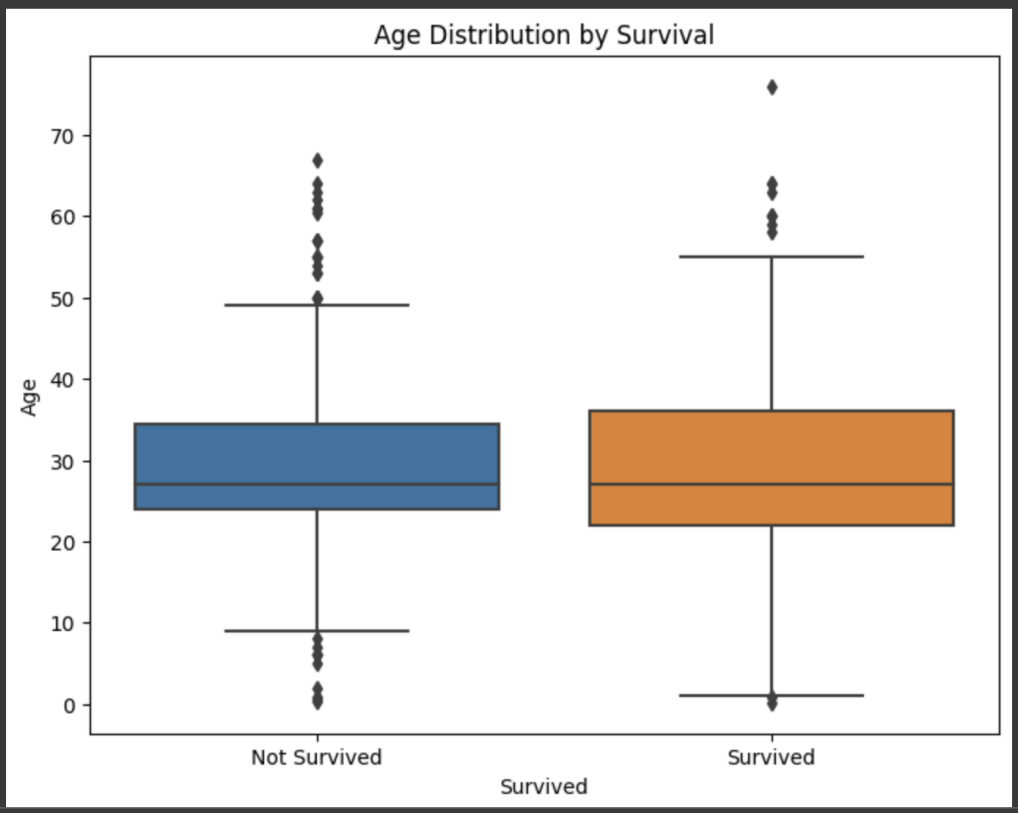
Bivariate Analysis
Pclass and Fare:

```
#Bivariate Analysis

plt.figure(figsize=(8, 6))
sns.boxplot(x='Pclass', y='Fare', data=data)
plt.xlabel('Pclass')
plt.ylabel('Fare')
plt.title('Fare by Passenger Class')
plt.show()
```



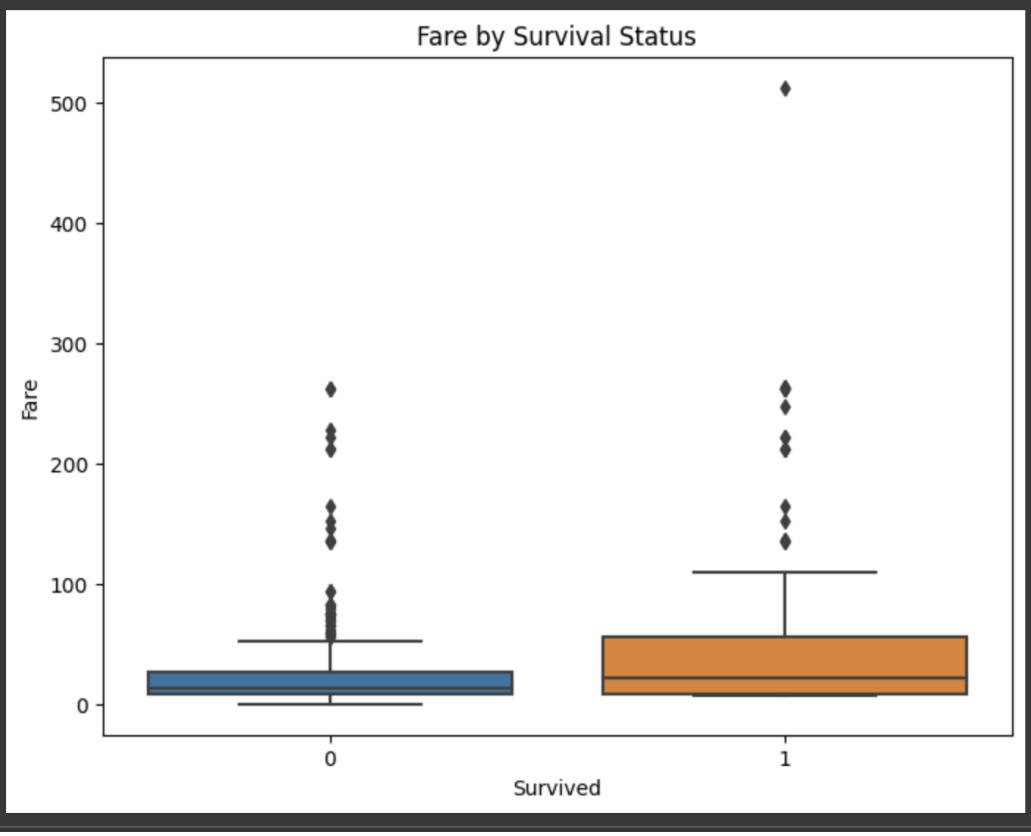
Survived and Age:

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Survived', y='Age', data=data)
plt.xlabel('Survived')
plt.ylabel('Age')
plt.title('Age Distribution by Survival')
plt.xticks([0, 1], ['Not Survived', 'Survived'])
plt.show()
```



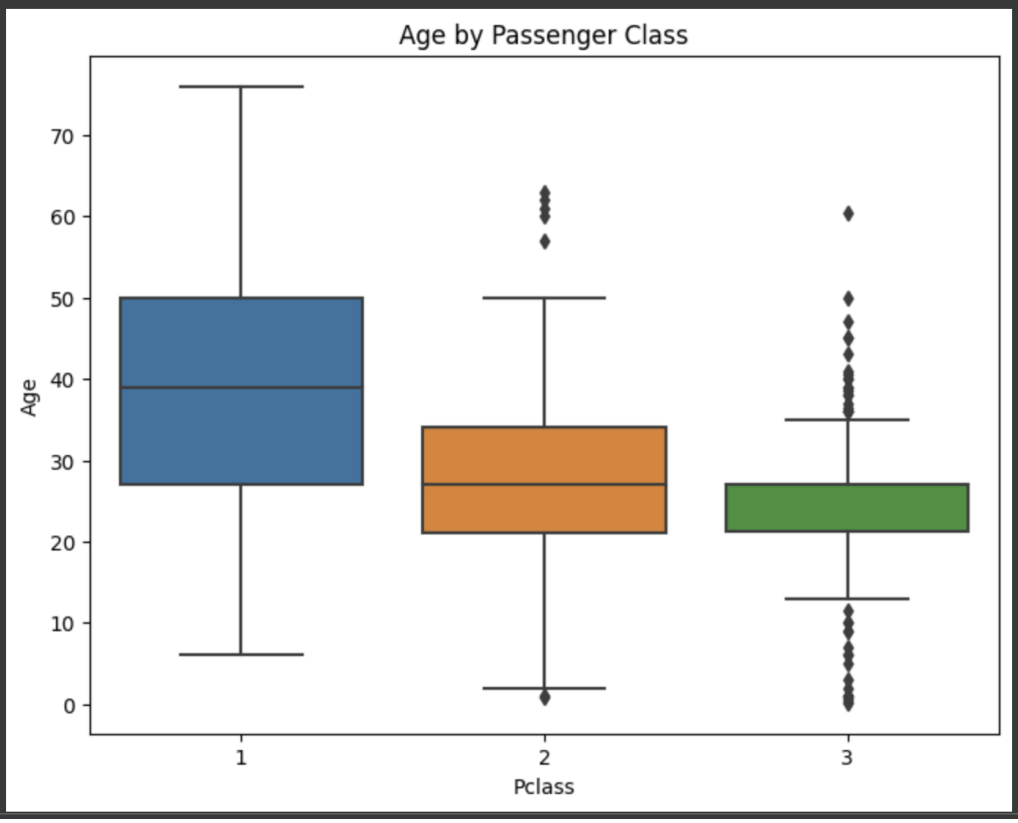
Survived and Fare:

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Survived', y='Fare', data=data)
plt.xlabel('Survived')
plt.ylabel('Fare')
plt.title('Fare by Survival Status')
plt.show()
```



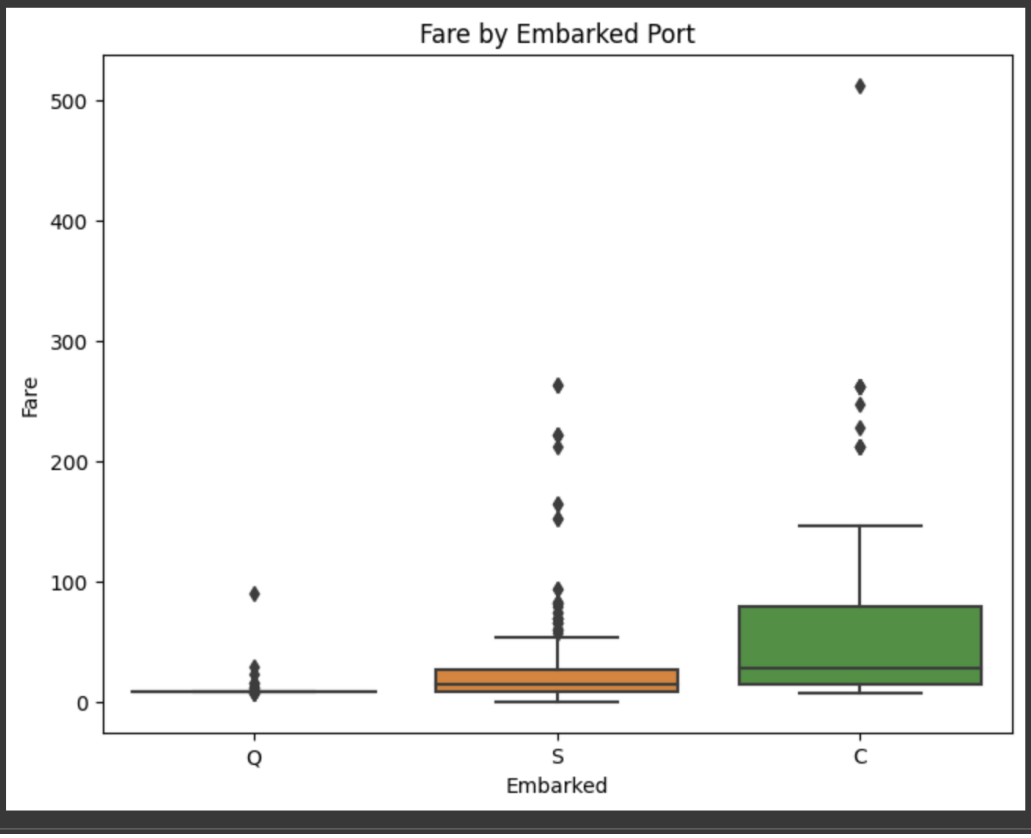
Age and Pclass:

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Pclass', y='Age', data=data)
plt.xlabel('Pclass')
plt.ylabel('Age')
plt.title('Age by Passenger Class')
plt.show()
```



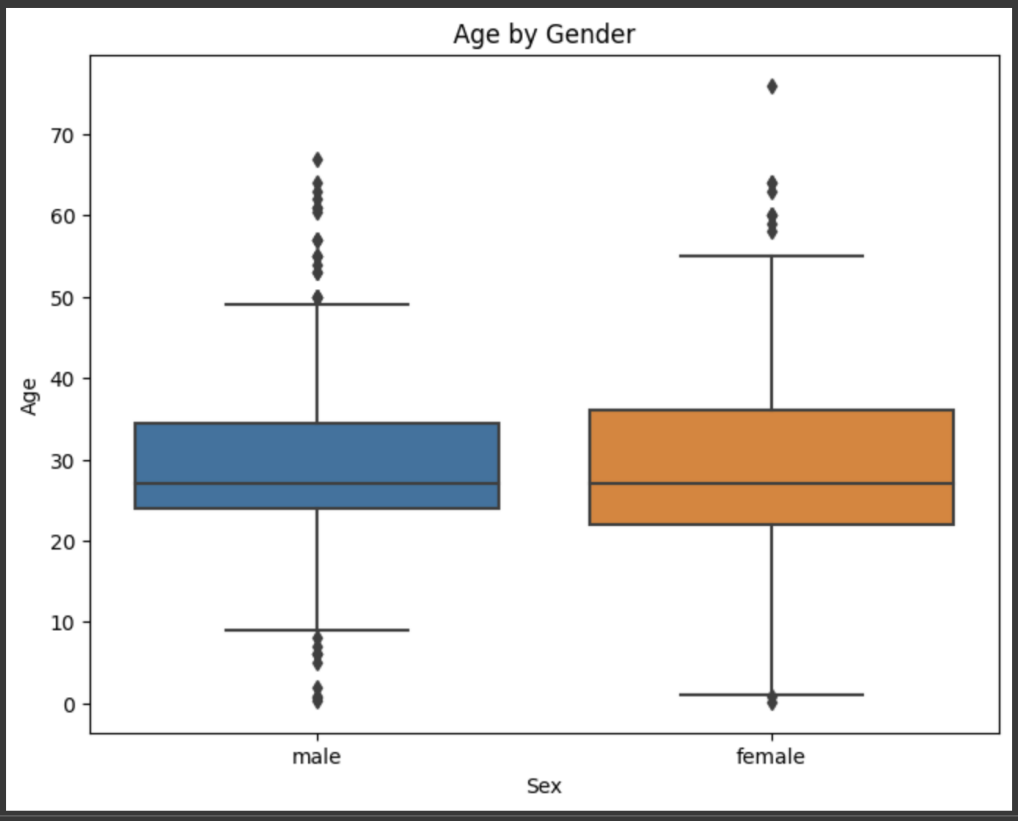
Fare and Embarked:

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Embarked', y='Fare', data=data)
plt.xlabel('Embarked')
plt.ylabel('Fare')
plt.title('Fare by Embarked Port')
plt.show()
```



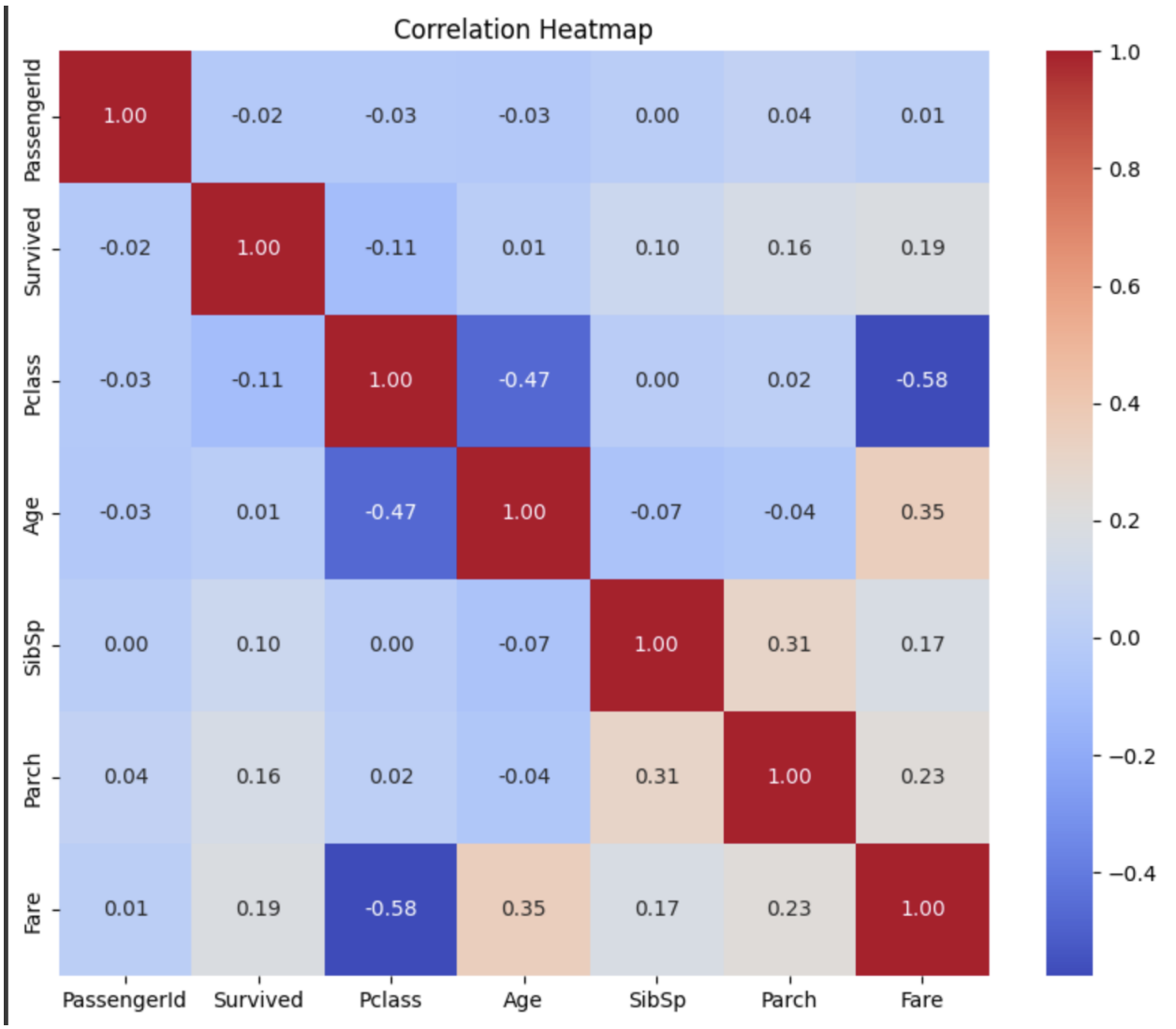
Age and Sex

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Sex', y='Age', data=data)
plt.xlabel('Sex')
plt.ylabel('Age')
plt.title('Age by Gender')
plt.show()
```



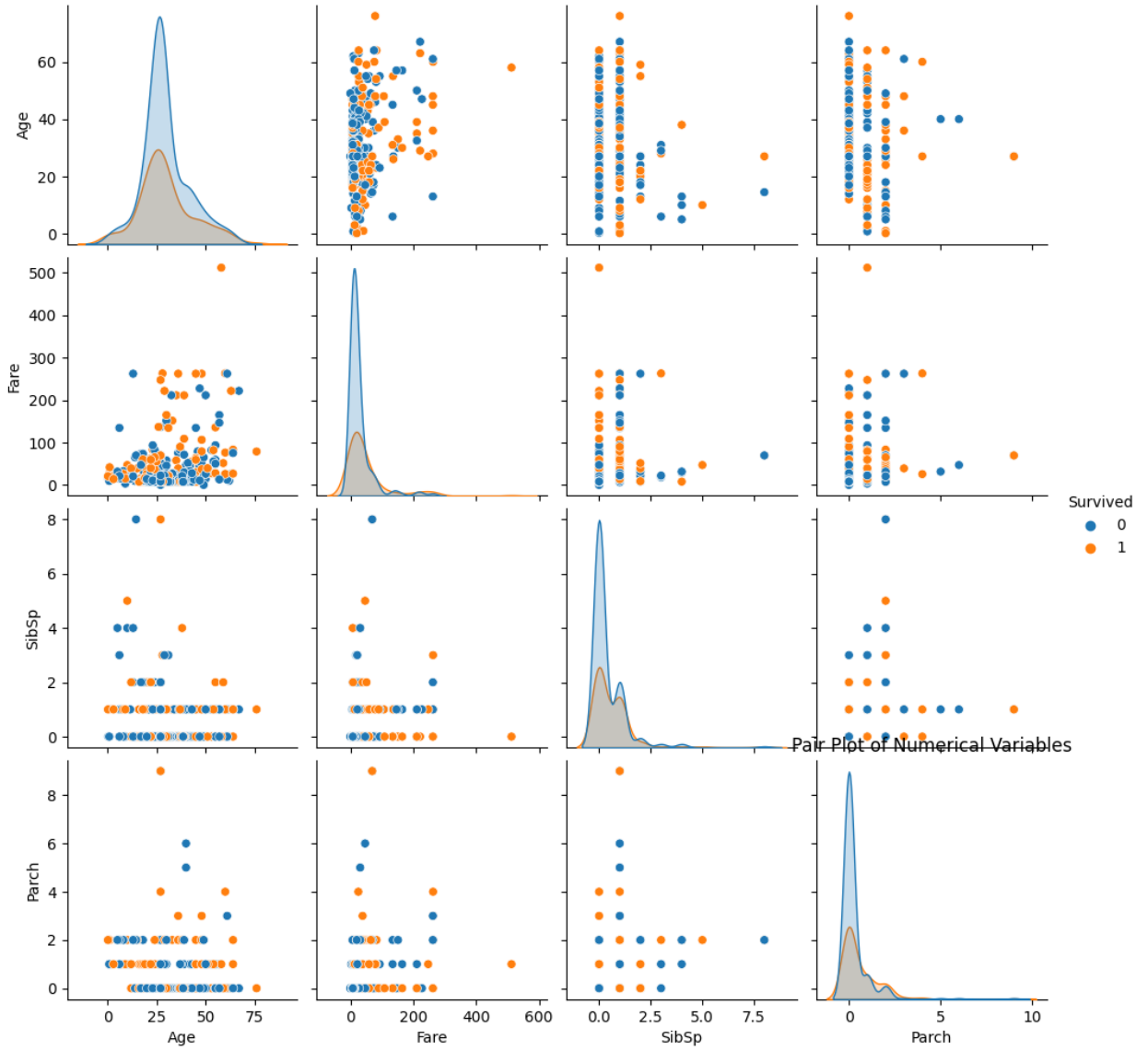
Correlation Analysis:

```python
#Correlation Analysis
correlation_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



Multivariate Analysis:

```
sns.pairplot(data[['Age', 'Fare', 'SibSp', 'Parch', 'Survived']], hue='Survived')
plt.title('Pair Plot of Numerical Variables')
plt.show()
```



Feature Engineering:

```
#Feature Engineering

# Create a 'FamilySize' feature
data['FamilySize'] = data['SibSp'] + data['Parch']
```

```
[33] # Extract titles from the 'Name' column
data['Title'] = data['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
```

Binning

```
# Binning 'Age' into categories
bins = [0, 18, 35, 60, 100]
labels = ['Child', 'Young Adult', 'Adult', 'Senior']
data['AgeCategory'] = pd.cut(data['Age'], bins=bins, labels=labels)

# Binning 'Fare' into categories
fare_bins = [0, 25, 50, 100, 1000]
fare_labels = ['Low', 'Medium', 'High', 'Very High']
data['FareCategory'] = pd.cut(data['Fare'], bins=fare_bins, labels=fare_labels)
```

Creating Dummy:

```
[35] # Create dummy variables for 'Sex' and 'Embarked'
data = pd.get_dummies(data, columns=['Sex', 'Embarked'], prefix=['Sex', 'Embarked'], drop_first=True)

# Create dummy variables for 'Pclass'
data = pd.get_dummies(data, columns=['Pclass'], prefix=['Pclass'], drop_first=False)
```

Feature Scaling:

```
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Scale numerical features
data[['Age', 'Fare']] = scaler.fit_transform(data[['Age', 'Fare']])
```

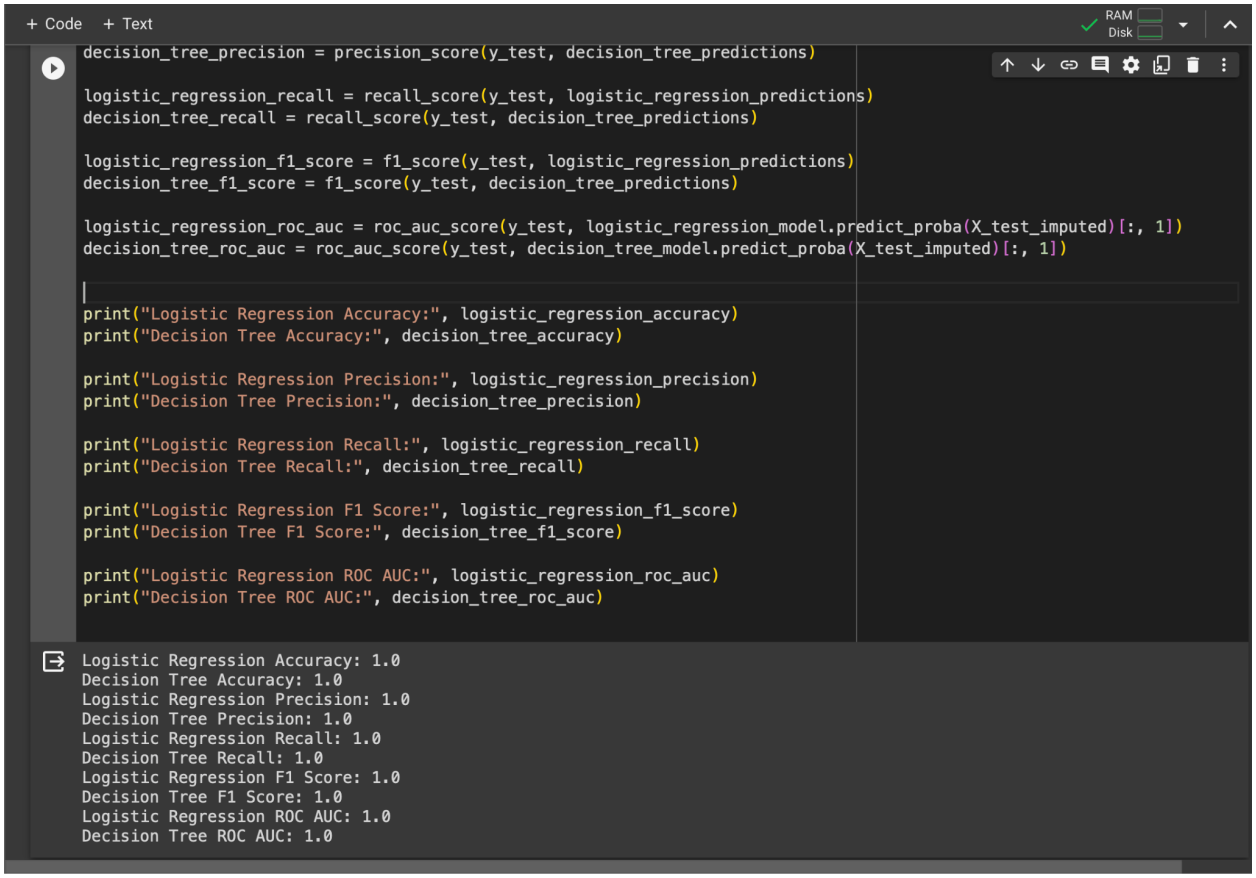
Data Splitting:

```
from sklearn.model_selection import train_test_split

X = data.drop(columns=['Survived'])  # Features
y = data['Survived']  # Target variable


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

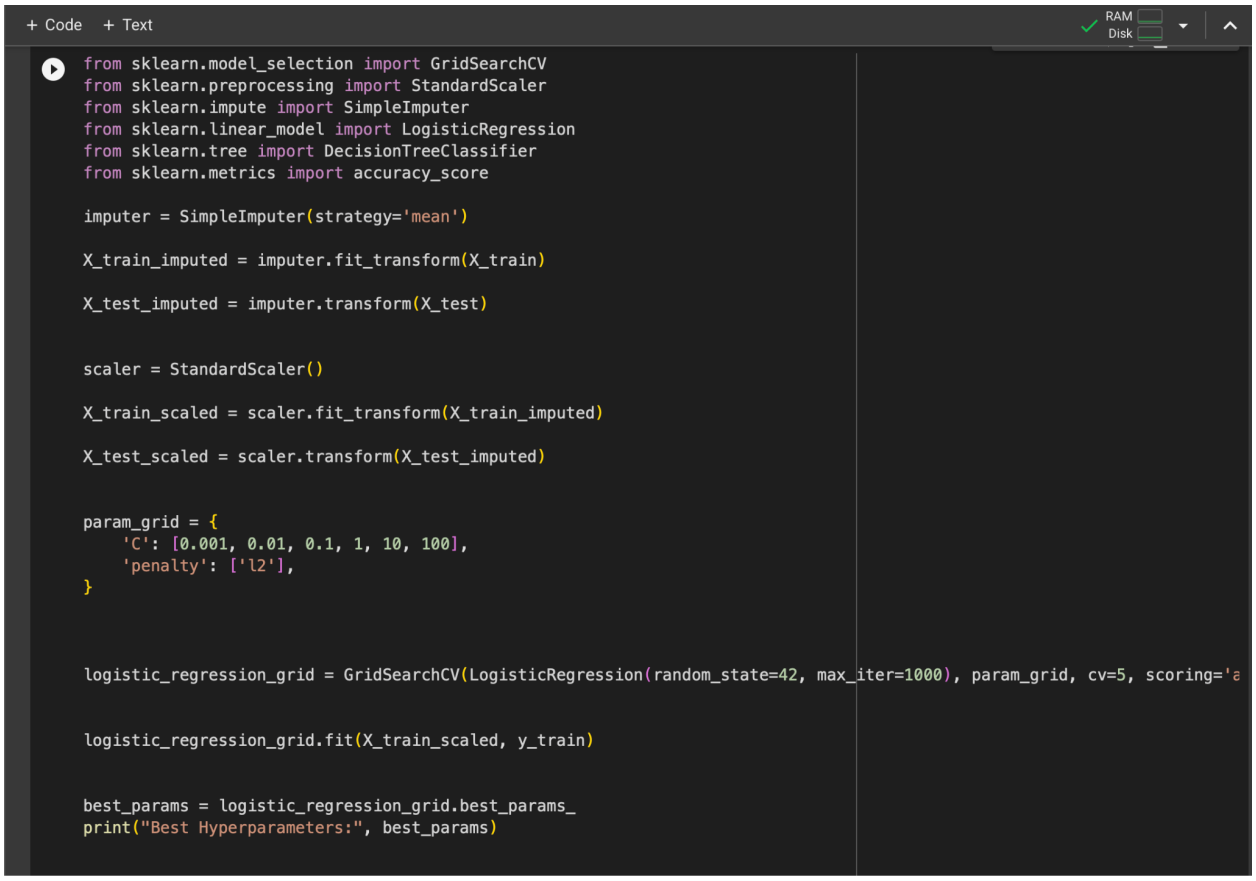
```
X_train shape: (334, 18)
X_test shape: (84, 18)
y_train shape: (334,)
y_test shape: (84,)
```

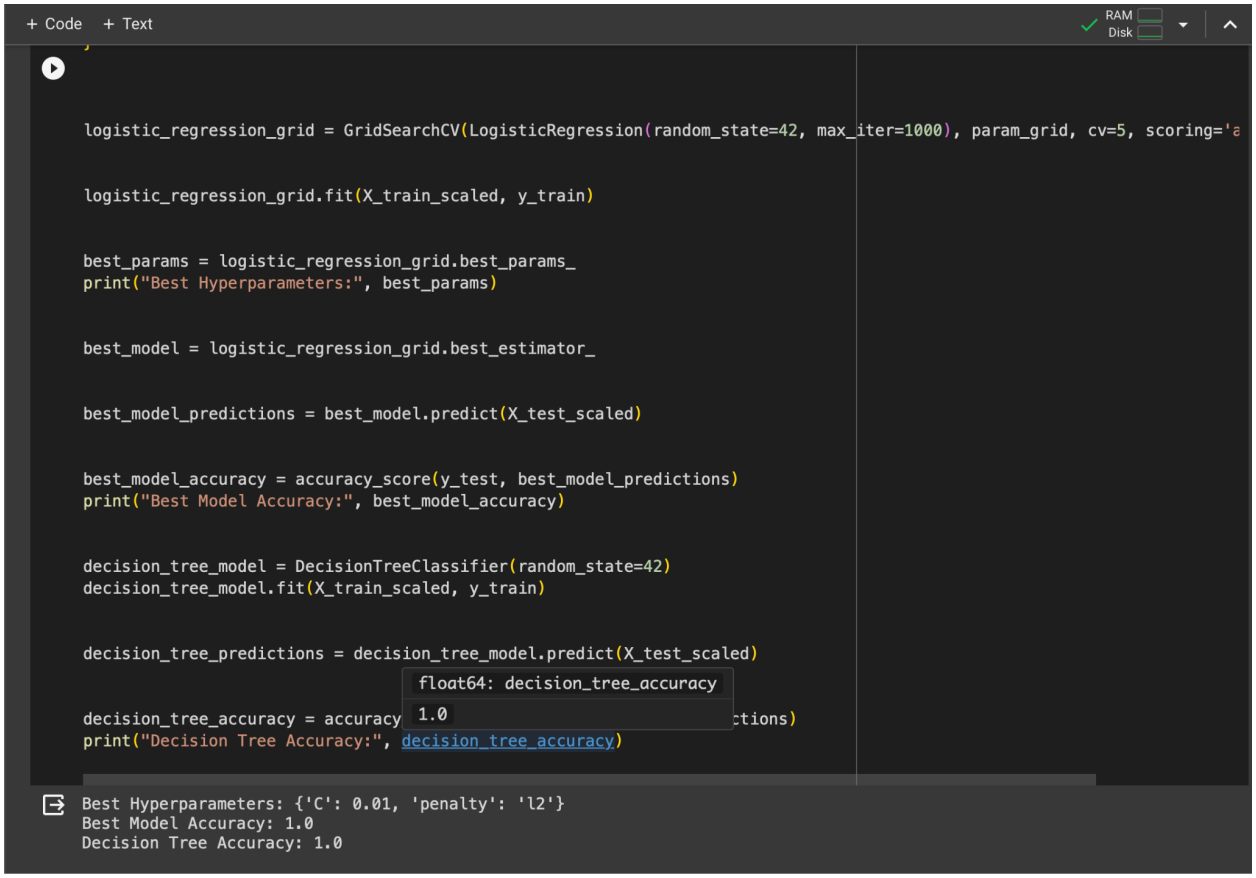
Model Training and Evaluation

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.impute import SimpleImputer


imputer = SimpleImputer(strategy='mean')

X_train_imputed = imputer.fit_transform(X_train)


X_test_imputed = imputer.transform(X_test)


logistic_regression_model = LogisticRegression(random_state=42, max_iter=1000)
logistic_regression_model.fit(X_train_imputed, y_train)


decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train_imputed, y_train)


logistic_regression_predictions = logistic_regression_model.predict(X_test_imputed)
decision_tree_predictions = decision_tree_model.predict(X_test_imputed)


logistic_regression_accuracy = accuracy_score(y_test, logistic_regression_predictions)
decision_tree_accuracy = accuracy_score(y_test, decision_tree_predictions)

logistic_regression_precision = precision_score(y_test, logistic_regression_predictions)
decision_tree_precision = precision_score(y_test, decision_tree_predictions)

logistic_regression_recall = recall_score(y_test, logistic_regression_predictions)
decision_tree_recall = recall_score(y_test, decision_tree_predictions)

logistic_regression_f1_score = f1_score(y_test, logistic_regression_predictions)
decision_tree_f1_score = f1_score(y_test, decision_tree_predictions)
```


+ Code
+ Text
RAM
Disk
decision_tree_precision = precision_score(y_test, decision_tree_predictions)
logistic_regression_recall = recall_score(y_test, logistic_regression_predictions)
decision_tree_recall = recall_score(y_test, decision_tree_predictions)
logistic_regression_f1_score = f1_score(y_test, logistic_regression_predictions)
decision_tree_f1_score = f1_score(y_test, decision_tree_predictions)
logistic_regression_roc_auc = roc_auc_score(y_test, logistic_regression_model.predict_proba(X_test_imputed)[:, 1])
decision_tree_roc_auc = roc_auc_score(y_test, decision_tree_model.predict_proba(X_test_imputed)[:, 1])
print("Logistic Regression Accuracy:", logistic_regression_accuracy)
print("Decision Tree Accuracy:", decision_tree_accuracy)
print("Logistic Regression Precision:", logistic_regression_precision)
print("Decision Tree Precision:", decision_tree_precision)
print("Logistic Regression Recall:", logistic_regression_recall)
print("Decision Tree Recall:", decision_tree_recall)
print("Logistic Regression F1 Score:", logistic_regression_f1_score)
print("Decision Tree F1 Score:", decision_tree_f1_score)
print("Logistic Regression ROC AUC:", logistic_regression_roc_auc)
print("Decision Tree ROC AUC:", decision_tree_roc_auc)
Logistic Regression Accuracy: 1.0
Decision Tree Accuracy: 1.0
Logistic Regression Precision: 1.0
Decision Tree Precision: 1.0
Logistic Regression Recall: 1.0
Decision Tree Recall: 1.0
Logistic Regression F1 Score: 1.0
Decision Tree F1 Score: 1.0
Logistic Regression ROC AUC: 1.0
Decision Tree ROC AUC: 1.0

Hyperparameter Tuning:




+ Code
+ Text
RAM
Disk
logistic_regression_grid = GridSearchCV(LogisticRegression(random_state=42, max_iter=1000), param_grid, cv=5, scoring='a
logistic_regression_grid.fit(X_train_scaled, y_train)
best_params = logistic_regression_grid.best_params_
print("Best Hyperparameters:", best_params)
best_model = logistic_regression_grid.best_estimator_
best_model_predictions = best_model.predict(X_test_scaled)
best_model_accuracy = accuracy_score(y_test, best_model_predictions)
print("Best Model Accuracy:", best_model_accuracy)
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train_scaled, y_train)
decision_tree_predictions = decision_tree_model.predict(X_test_scaled)
float64: decision_tree_accuracy
1.0
ctions)
print("Decision Tree Accuracy:", decision_tree_accuracy)
Best Hyperparameters: {'C': 0.01, 'penalty': 'l2'}
Best Model Accuracy: 1.0
Decision Tree Accuracy: 1.0