Untitled5

December 18, 2023

1 Predicting Titanic Survival: A Data Science Project

This project involves developing a predictive model to determine the likelihood of survival for passengers on the Titanic using data science techniques in Python. The dataset encompasses various features such as socio-economic status, age, gender, and family size, providing valuable insights into factors influencing survival rates.

Dataset Columns:

PassengerId: Unique identifier for each passenger.

Survived: Binary indicator (1 or 0) for passenger survival.

Pclass: Ticket class representing socio-economic status (1st, 2nd, 3rd).

Name: Full name, including titles.

Sex: Gender of the passenger (male or female).

Age: Age of the passenger.

SibSp: Number of siblings/spouses aboard.

Parch: Number of parents/children aboard.

Ticket: Ticket number.

Fare: Passenger fare.

Cabin: Cabin number (with missing values).

Embarked: Port of embarkation (C, Q, S).

This project aims to leverage machine learning algorithms to analyze historical data and make predictions on passenger survival. By understanding the relationships between these features and survival outcomes, we can uncover patterns and insights that contribute to the overall narrative of the Titanic disaster.

1.1 Step 1: Import Libraries

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
```

1.2 Step 2: Load and Explore the Dataset

[3]:
```python
titanic_data = pd.read_csv('titanic.csv')
titanic_data.head()
titanic_data.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
```

[4]:
```python
titanic_data.describe()
```

[4]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000
```

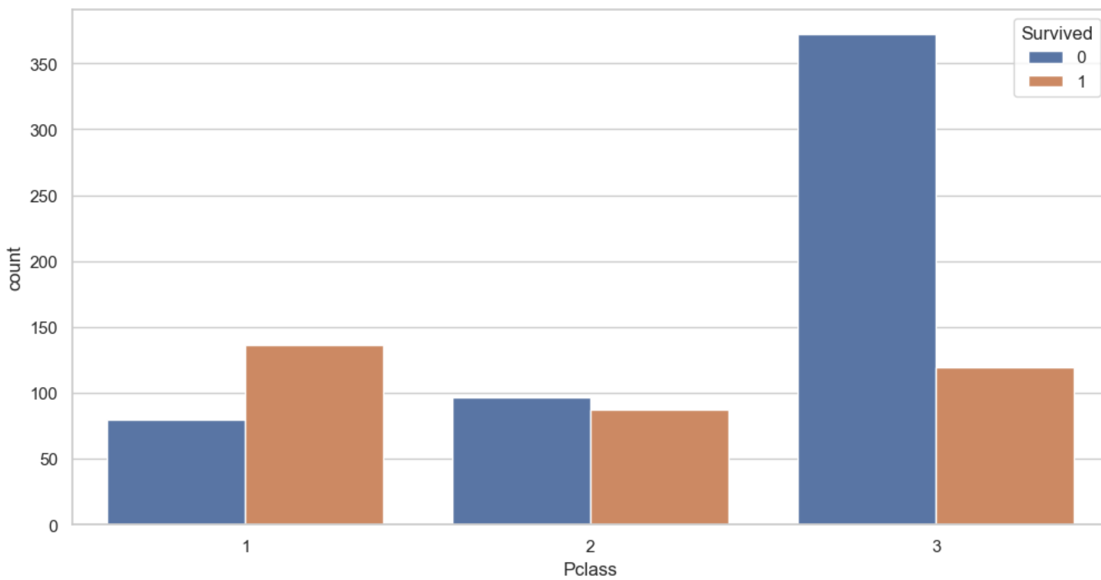
```
          Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

1.3 Step 3: Data Visualization

2 - Count Plot

[5]:
```
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))
sns.countplot(x='Pclass', hue='Survived', data=titanic_data)
```

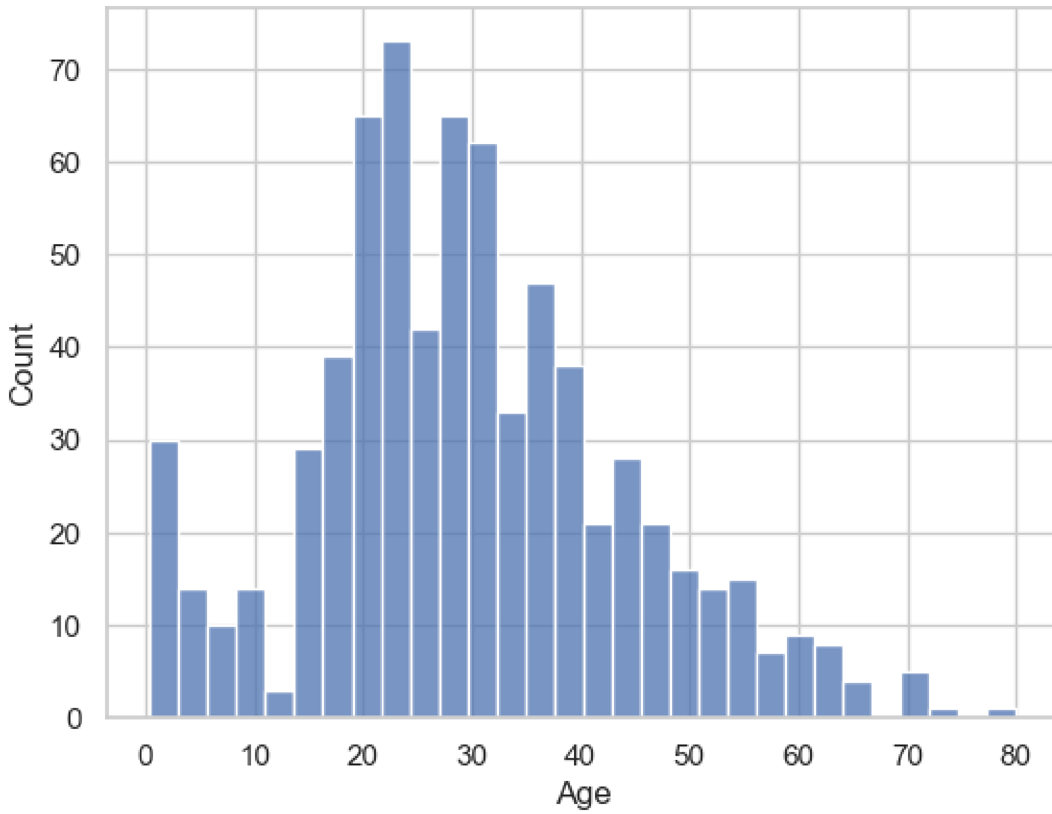
[5]:
```
<Axes: xlabel='Pclass', ylabel='count'>
```



3 - Hist Plot

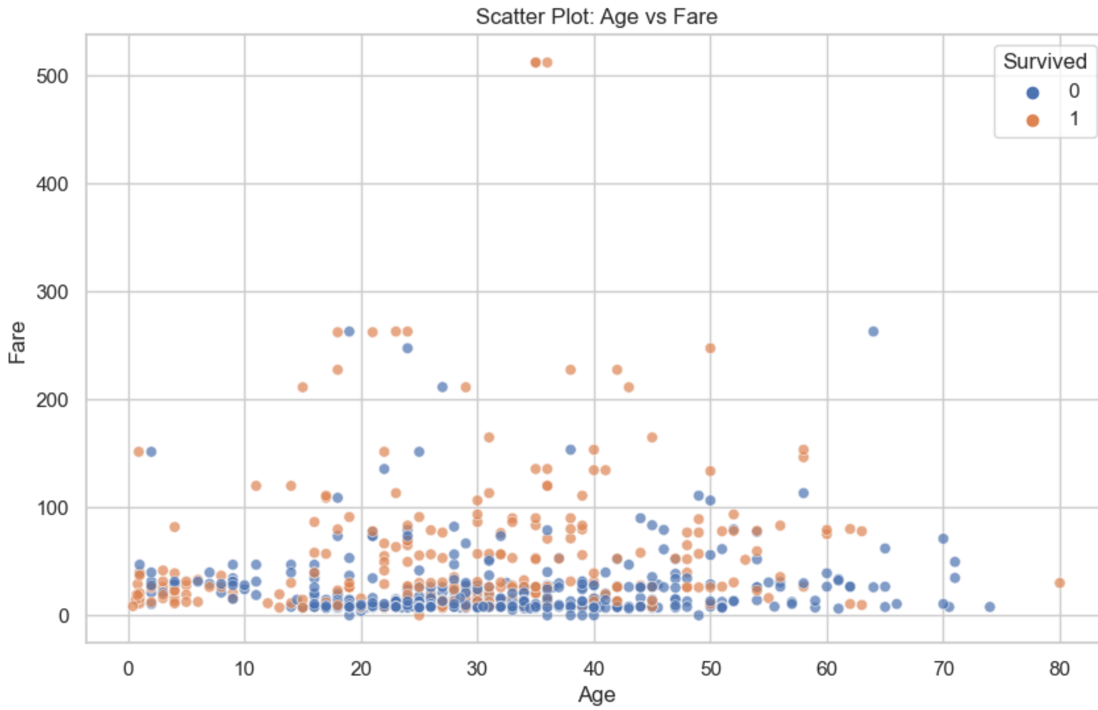
[6]:
```
sns.histplot(titanic_data['Age'].dropna(), kde=False, bins=30)
```

[6]:
```
<Axes: xlabel='Age', ylabel='Count'>
```



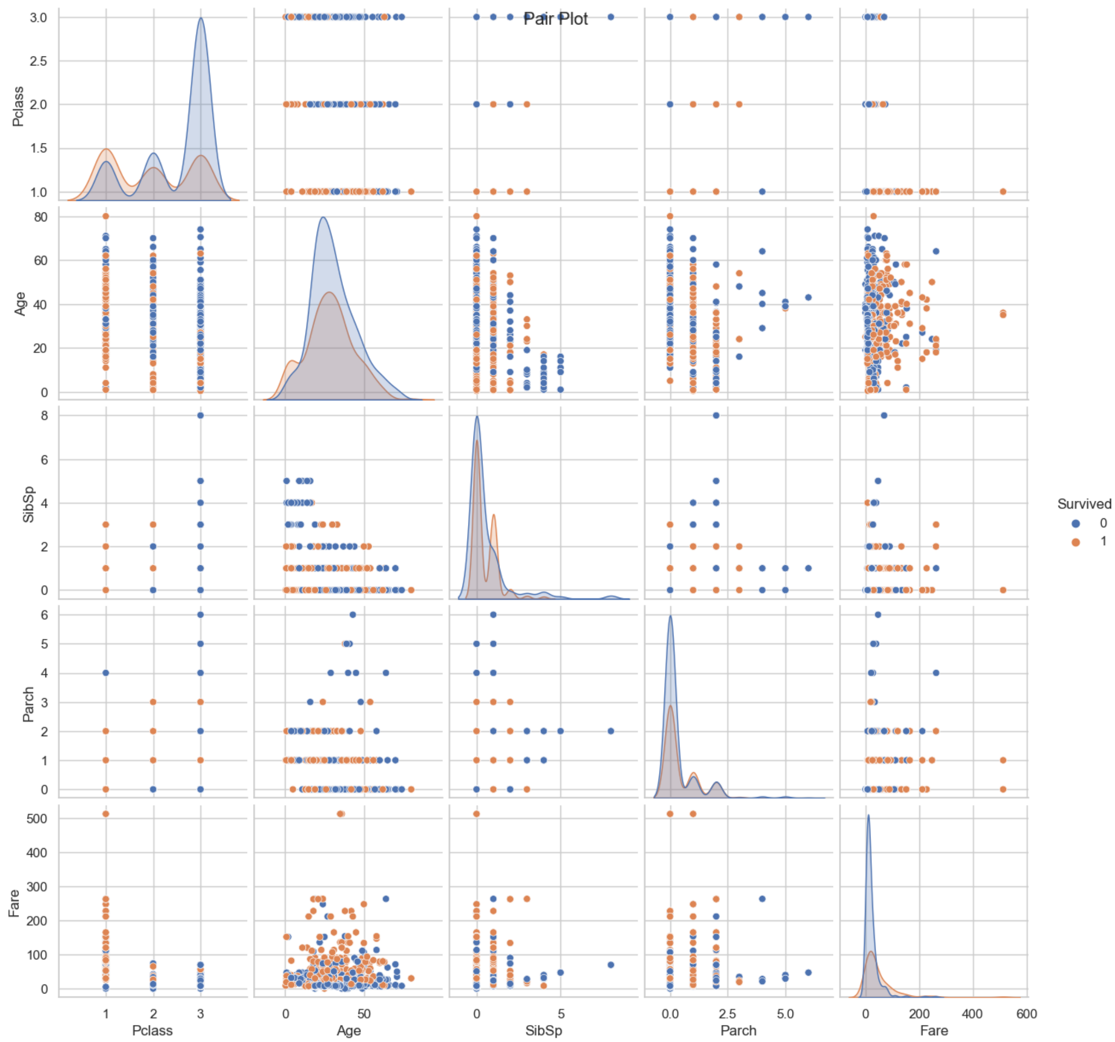
4 - Scatter Plot

[7]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Age', y='Fare', hue='Survived', data=titanic_data, alpha=0.7)
plt.title('Scatter Plot: Age vs Fare')
plt.show()
```



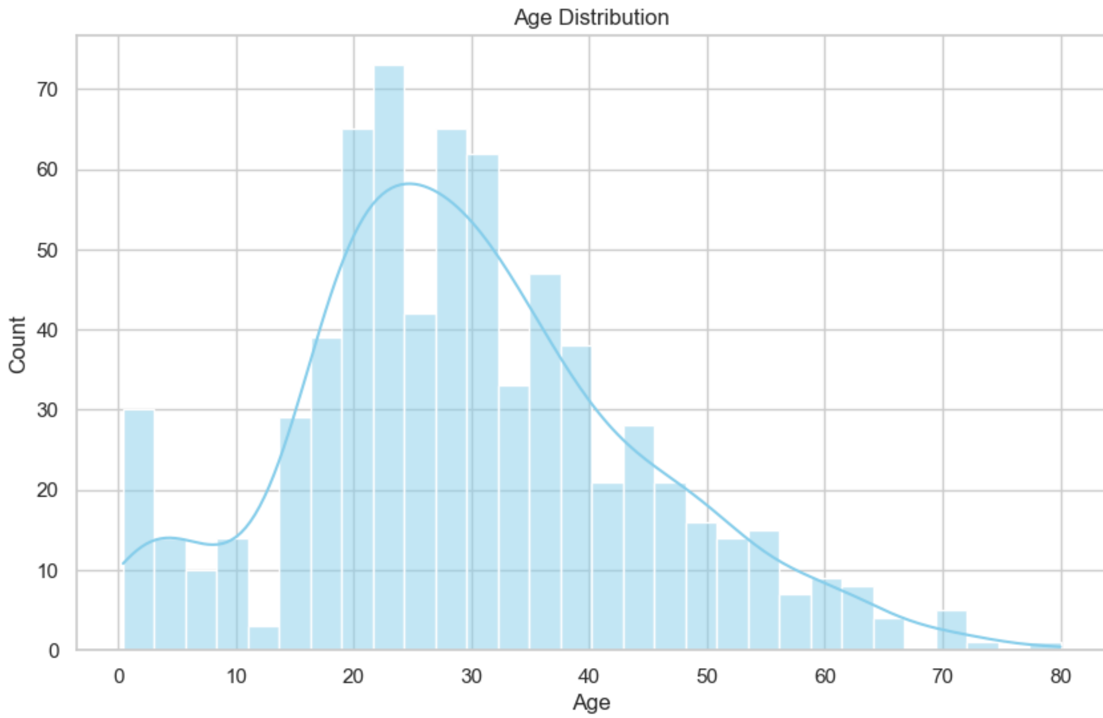
5 - Pair Plot

[8]:
```
sns.pairplot(titanic_data[['Survived', 'Pclass', 'Age', 'SibSp', 'Parch',␣
↪'Fare']], hue='Survived')
plt.suptitle('Pair Plot')
plt.show()
```



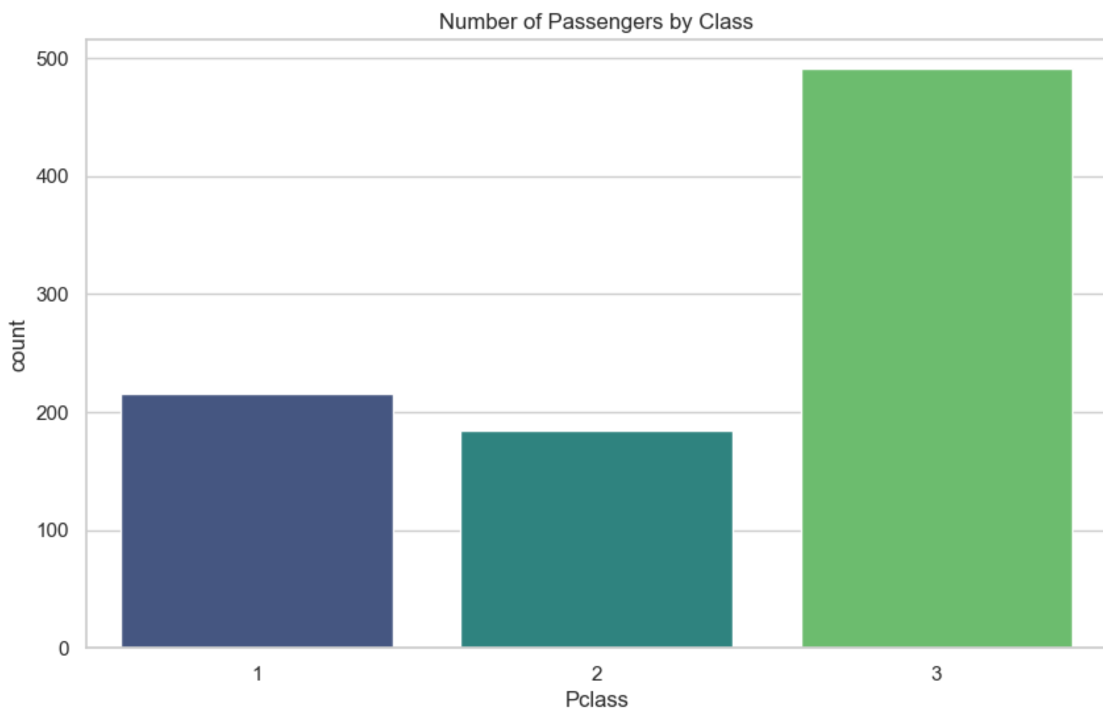
6 - Dist Plot

[9]:
```
plt.figure(figsize=(10, 6))
sns.histplot(titanic_data['Age'].dropna(), kde=True, bins=30, color='skyblue')
plt.title('Age Distribution')
plt.show()
```



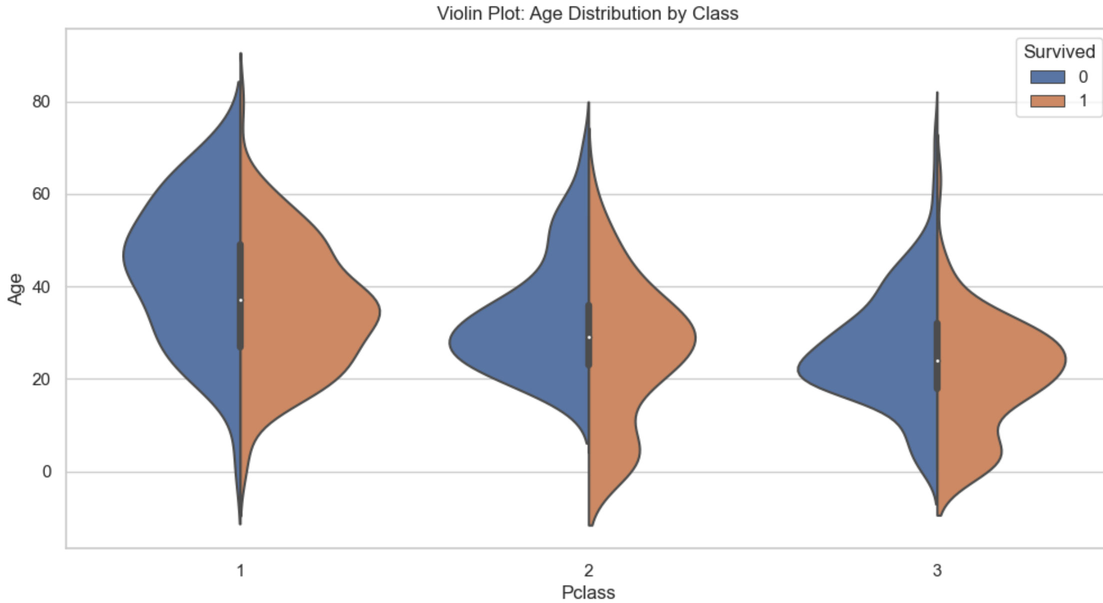
7 - Bar Plot

[10]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='Pclass', data=titanic_data, palette='viridis')
plt.title('Number of Passengers by Class')
plt.show()
```



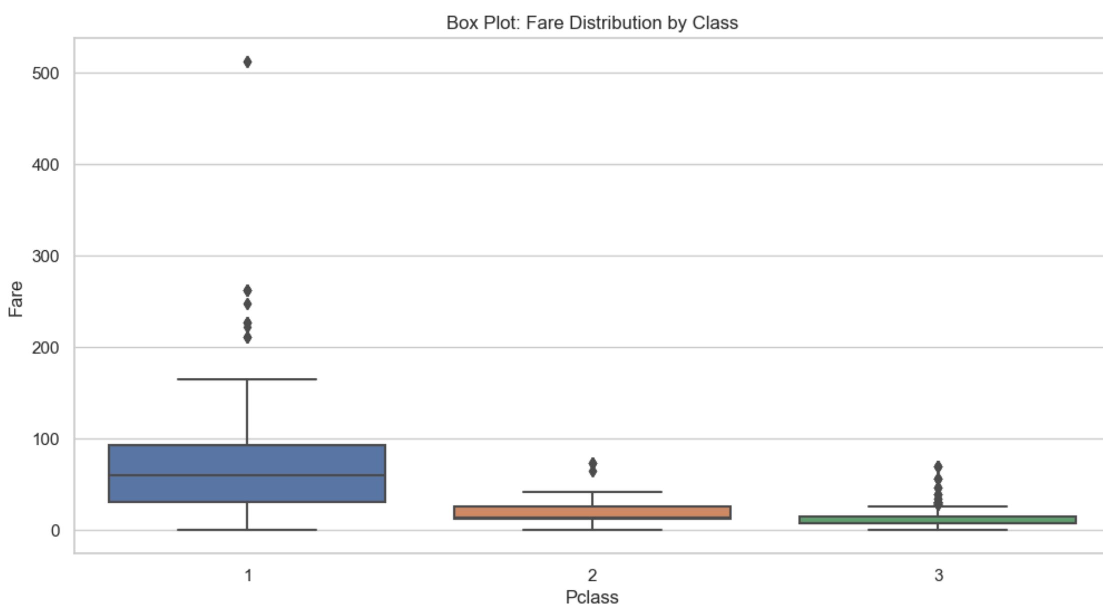
8 - Violin Plot

[11]:
```
plt.figure(figsize=(12, 6))
sns.violinplot(x='Pclass', y='Age', hue='Survived', data=titanic_data,␣
↪split=True)
plt.title('Violin Plot: Age Distribution by Class')
plt.show()
```



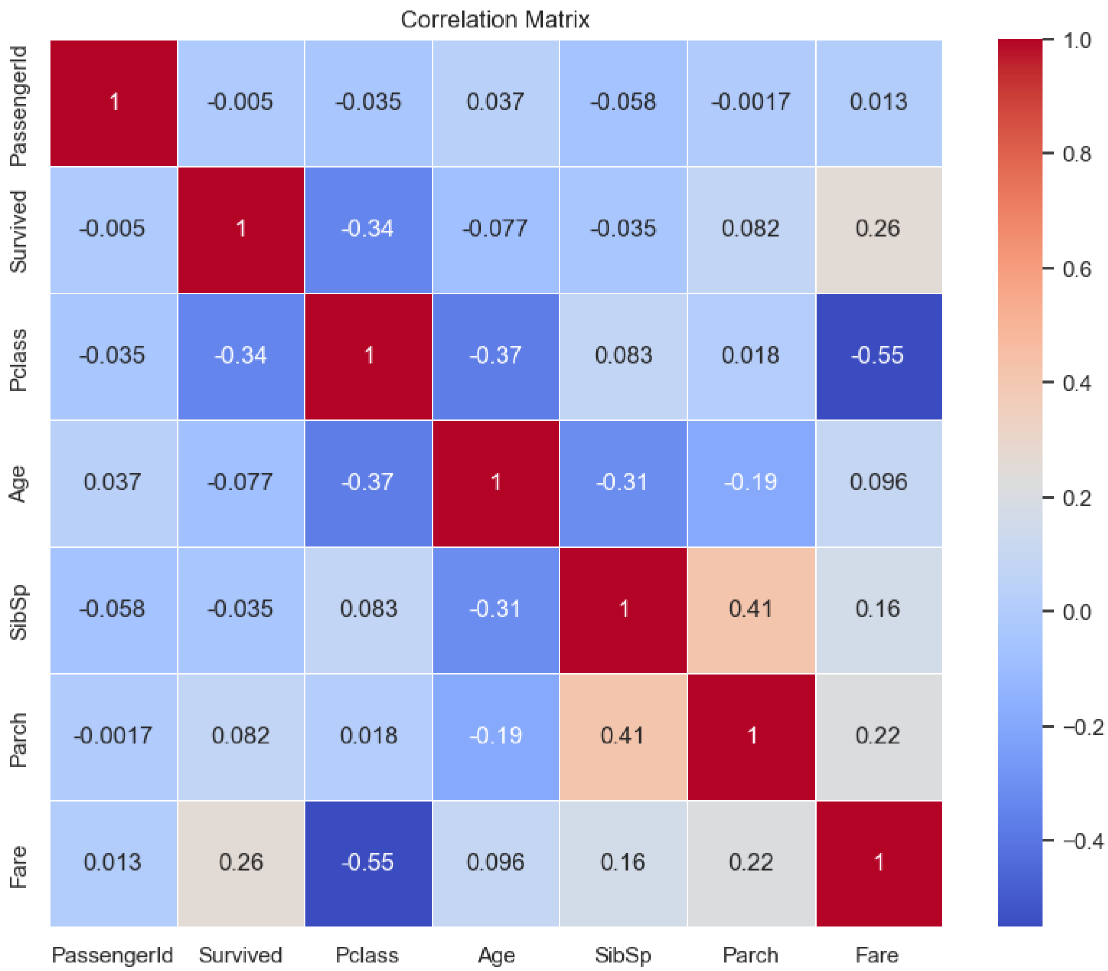
9 - Box Plot

[12]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='Pclass', y='Fare', data=titanic_data)
plt.title('Box Plot: Fare Distribution by Class')
plt.show()
```



10 - HeatMap

[13]:
```
correlation_matrix = titanic_data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```



10.1 Step 4: Data Preprocessing

[14]:
```
titanic_data.drop(['Cabin'], axis=1, inplace=True)
titanic_data['Age'].fillna(titanic_data['Age'].median(), inplace=True)
titanic_data['Embarked'].fillna(titanic_data['Embarked'].mode()[0],␣
↪inplace=True)
```

```
titanic_data = pd.get_dummies(titanic_data, columns=['Sex', 'Embarked'],␣
↪drop_first=True)

X = titanic_data.drop(['Survived', 'PassengerId', 'Name', 'Ticket'], axis=1)
y = titanic_data['Survived']
```

10.2 Step 5: Train-Test Split

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

10.3 Step 6: Build and Evaluate Models

[16]:
```
def train_and_evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy

# Logistic Regression
logreg_model = LogisticRegression()
logreg_accuracy = train_and_evaluate_model(logreg_model, X_train, y_train,␣
↪X_test, y_test)

# K Nearest Neighbour
knn_model = KNeighborsClassifier()
knn_accuracy = train_and_evaluate_model(knn_model, X_train, y_train, X_test,␣
↪y_test)

# Random Forest
rf_model = RandomForestClassifier()
rf_accuracy = train_and_evaluate_model(rf_model, X_train, y_train, X_test,␣
↪y_test)

# Xtreme Boosting
xgb_model = XGBClassifier()
xgb_accuracy = train_and_evaluate_model(xgb_model, X_train, y_train, X_test,␣
↪y_test)

# Support Vector Machine
svm_model = SVC()
svm_accuracy = train_and_evaluate_model(svm_model, X_train, y_train, X_test,␣
↪y_test)

# Display accuracy scores
print("Logistic Regression Accuracy:", logreg_accuracy)
```

```
print("K Nearest Neighbour Accuracy:", knn_accuracy)
print("Random Forest Accuracy:", rf_accuracy)
print("Xtreme Boosting Accuracy:", xgb_accuracy)
print("Support Vector Machine Accuracy:", svm_accuracy)
```

```
Logistic Regression Accuracy: 0.7988826815642458
K Nearest Neighbour Accuracy: 0.7150837988826816
Random Forest Accuracy: 0.8156424581005587
Xtreme Boosting Accuracy: 0.8212290502793296
Support Vector Machine Accuracy: 0.6536312849162011
```

[17]:
```
best_model = max([(logreg_accuracy, 'Logistic Regression'),
                  (knn_accuracy, 'K Nearest Neighbour'),
                  (rf_accuracy, 'Random Forest'),
                  (xgb_accuracy, 'Xtreme Boosting'),
                  (svm_accuracy, 'Support Vector Machine')])

print(f"The best model is {best_model[1]} with an accuracy of {best_model[0]:.
 ↪2f}")
```

```
The best model is Xtreme Boosting with an accuracy of 0.82
```

10.4 Step 7: Make Predictions

[18]:
```
# Assuming you have chosen XGBClassifier as the best model
best_model_instance = XGBClassifier()
best_model_instance.fit(X_train, y_train)

# Make predictions on the test set
y_pred = best_model_instance.predict(X_test)
```

10.5 Step 8: Evaluate the Model

[19]:
```
columns_for_prediction = ['PassengerId', 'Pclass', 'Age', 'SibSp', 'Parch',␣
 ↪'Fare', 'Sex_male', 'Embarked_Q', 'Embarked_S']
titanic_data_for_prediction = titanic_data[columns_for_prediction]

likelihood_of_survival = best_model_instance.
 ↪predict(titanic_data_for_prediction.drop(['PassengerId'], axis=1))

print("Likelihood of Survival:", likelihood_of_survival)
```

```
Likelihood of Survival: [0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0
1 0 0 1 1 0 0 0 0
 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0
 1 0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 1 0
 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1
```

0 1 0 0 0 1 0 0 1 1 1 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0
0 0 1 0 0 0 0 0 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1
1 0 1 0 0 1 0 0 0 1 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 1 1 1 0 0
0 1 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1
1 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 1 0 1 0 0 1 0 0 1
0 1 1 1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0
0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 0 1 0
1 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 0 1 1
0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1
0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0
0 1 0 1 0 0 1 1 0 0 0 0 1 0 0 1 1 0 1 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 1 0
0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 1 0
0 0 0 1 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0
1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0
1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0
0 0 1 1 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1
0 1 0]