vivrgwlfz

January 27, 2025

[ ]:
```
# FINAL PROJECT REPORT:


#  Project Title: Decision Tree Classifier for Titanic Survival Prediction


# Dataset: Titanic Survival Dataset from Kaggle


# Objective: Predict survival of passengers based on features like age, sex,␣
 ↪class, etc.


# Methodology:

# 1. Data Importing and Preprocessing
#  2. Feature Scaling and Encoding
# 3. Decision Tree Classifier Modeling
# 4. Model Evaluation and Visualization


# Results:

#- Accuracy: 80.97%
#- Classification Report:
   # - Precision: 87.0%
    #- Recall: 82.0%
   # - F1-Score: 84.0%
#- Confusion Matrix:
   # - True Positives: 139
   # - False Positives: 30
   # - True Negatives: 21
    #- False Negatives: 78


#Conclusion:
```

```
#The Decision Tree Classifier model achieved satisfactory results in predicting␣
 ↪Titanic passenger survival.
#Feature scaling and encoding improved model performance.
#Further tuning and ensemble methods could enhance accuracy.
```

[26]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
data = pd.read_csv(r"C:\Users\91703\Downloads\ML - TITANIC.csv")
```

[3]:
```
data.head()
```

[3]:

	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
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

[4]:
```
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
```

[5]:
```
data.describe()
```

[5]:

	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

[6]:
```
data.isnull().sum()
```

[6]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
```

```
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

[7]:
```
#we are handling the null values by filling these methods
data['Age'] = data['Age'].fillna(data['Age'].median())
data['Embarked'] = data['Embarked'].fillna(data['Embarked'].mode()[0])
data['Cabin'] = data['Cabin'].fillna('Unknown')
```

[8]:
```
data.isnull().sum()
```

[8]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

[9]:
```
#Dropping unwanted columns next...
#We will drop 'PassengerId', 'Ticket', 'Name'
data.drop(['PassengerId', 'Ticket', 'Name'], axis=1, inplace=True)
```

[10]:
```
data.head()
```

[10]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	male	22.0	1	0	7.2500	Unknown	S
1	1	1	female	38.0	1	0	71.2833	C85	C
2	1	3	female	26.0	0	0	7.9250	Unknown	S
3	1	1	female	35.0	1	0	53.1000	C123	S
4	0	3	male	35.0	0	0	8.0500	Unknown	S

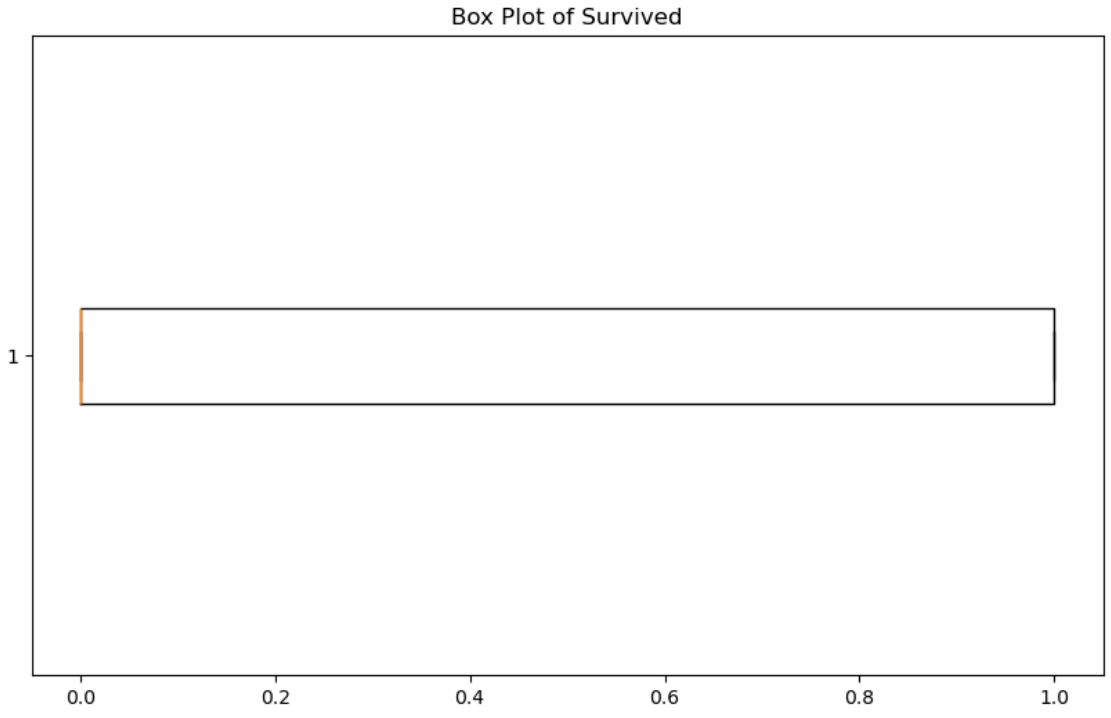
[13]:
```
#Convert categorical columns to numerical
le = LabelEncoder()
data['Sex'] = le.fit_transform(data['Sex'])
data['Embarked'] = le.fit_transform(data['Embarked'])
data['Cabin'] = le.fit_transform(data['Cabin'])
```

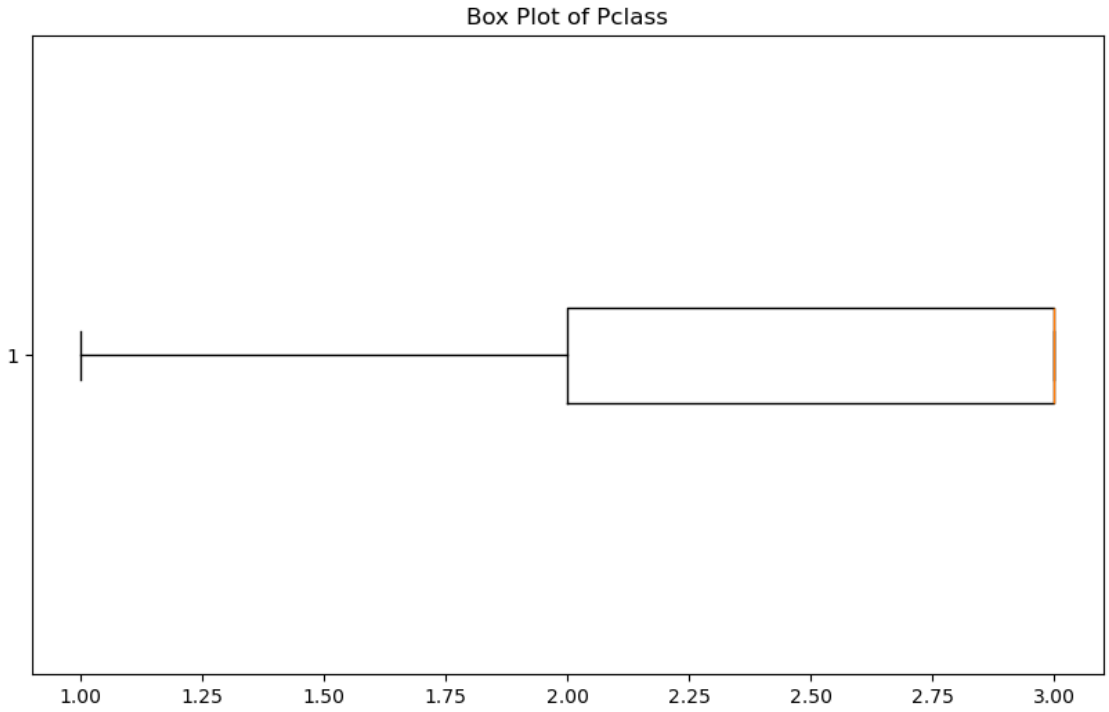
[14]:
```
data.head()
```

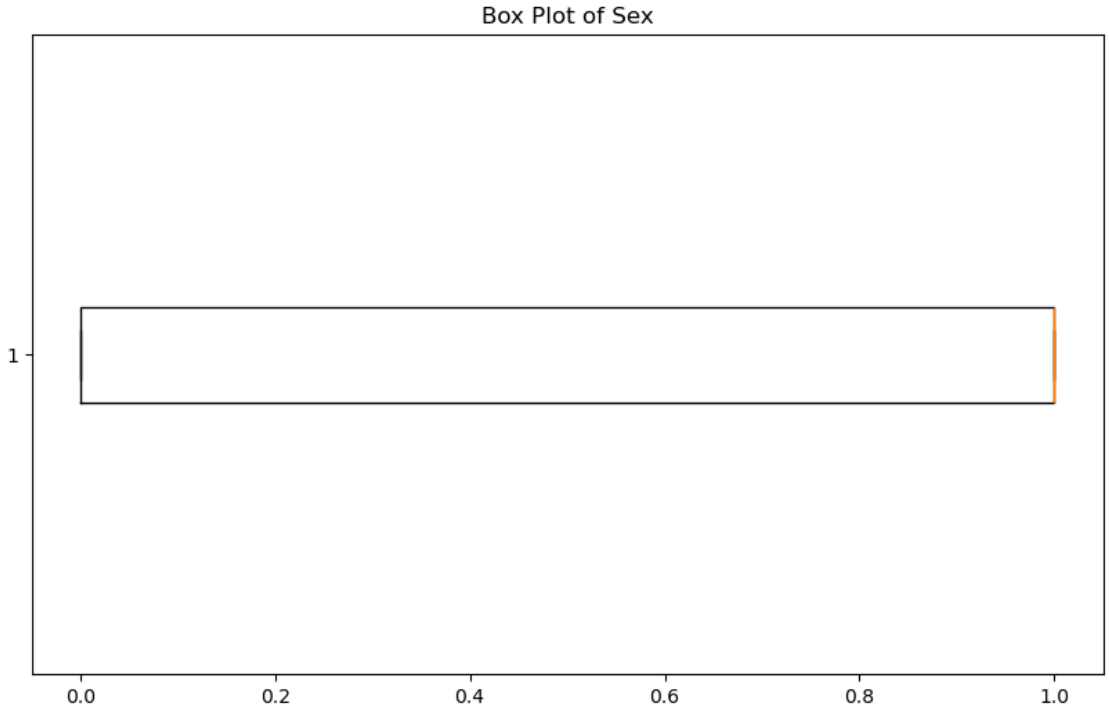
[14]:

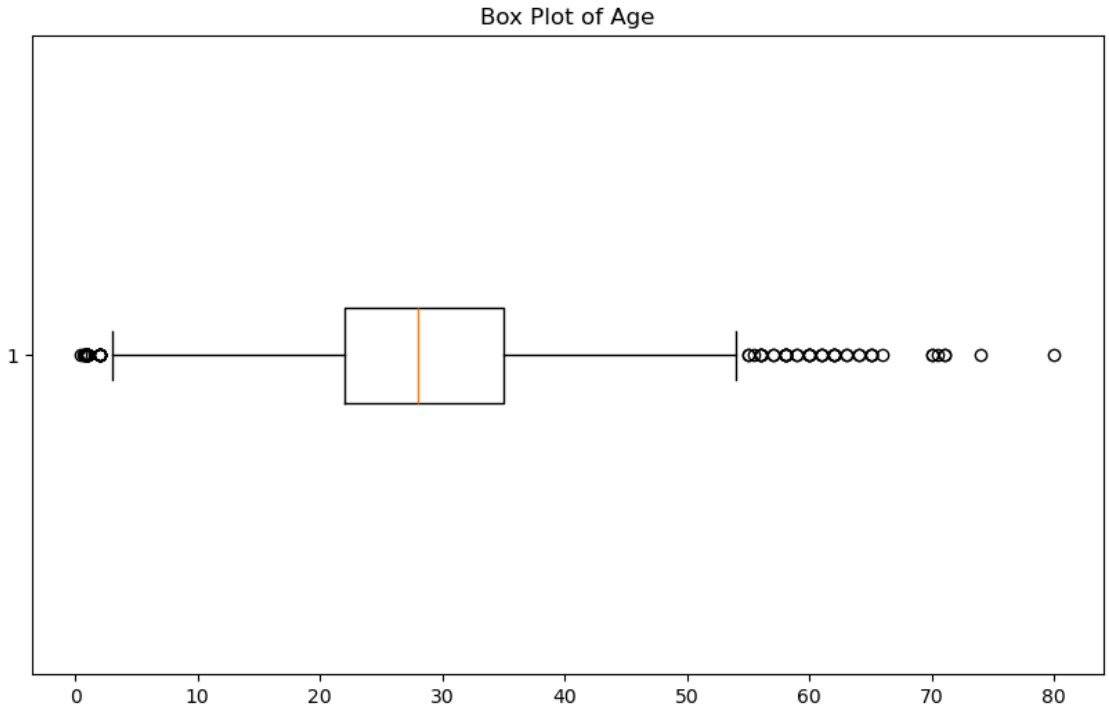
	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	1	22.0	1	0	7.2500	147	2
1	1	1	0	38.0	1	0	71.2833	81	0
2	1	3	0	26.0	0	0	7.9250	147	2
3	1	1	0	35.0	1	0	53.1000	55	2
4	0	3	1	35.0	0	0	8.0500	147	2

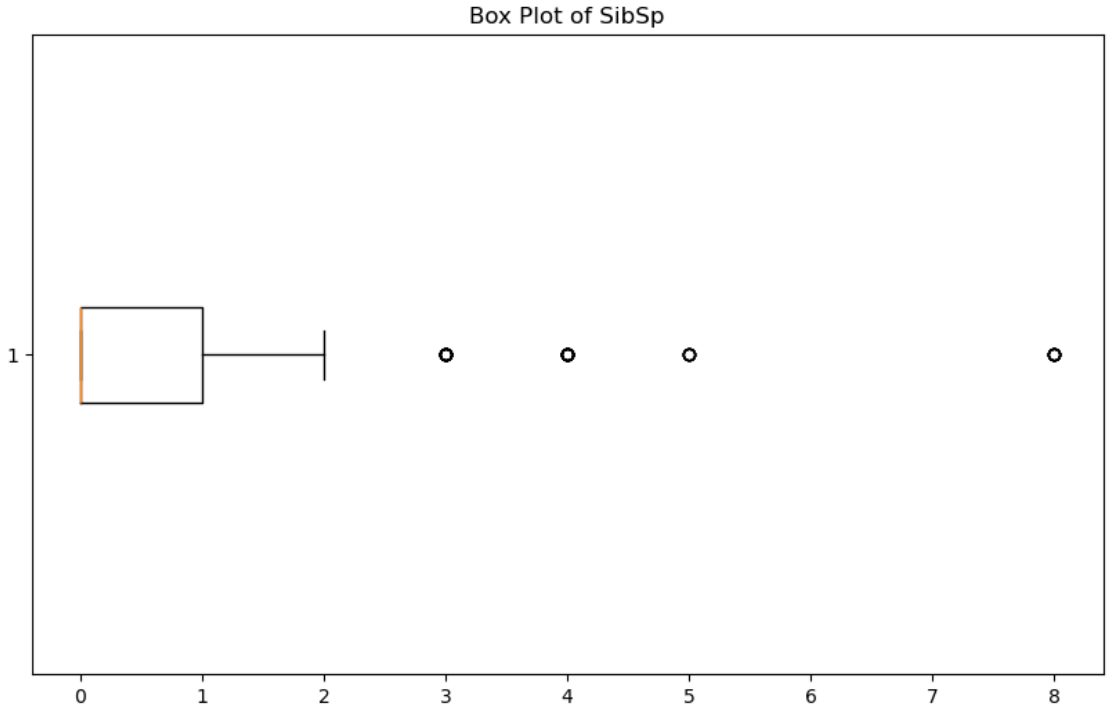
[21]:
```
for col in data.columns:
    plt.figure(figsize=(10,6))
    plt.subplot(1,1,1)
    plt.boxplot(data[col], vert=False)
    plt.title(f"Box Plot of {col}")
    plt.show()
```

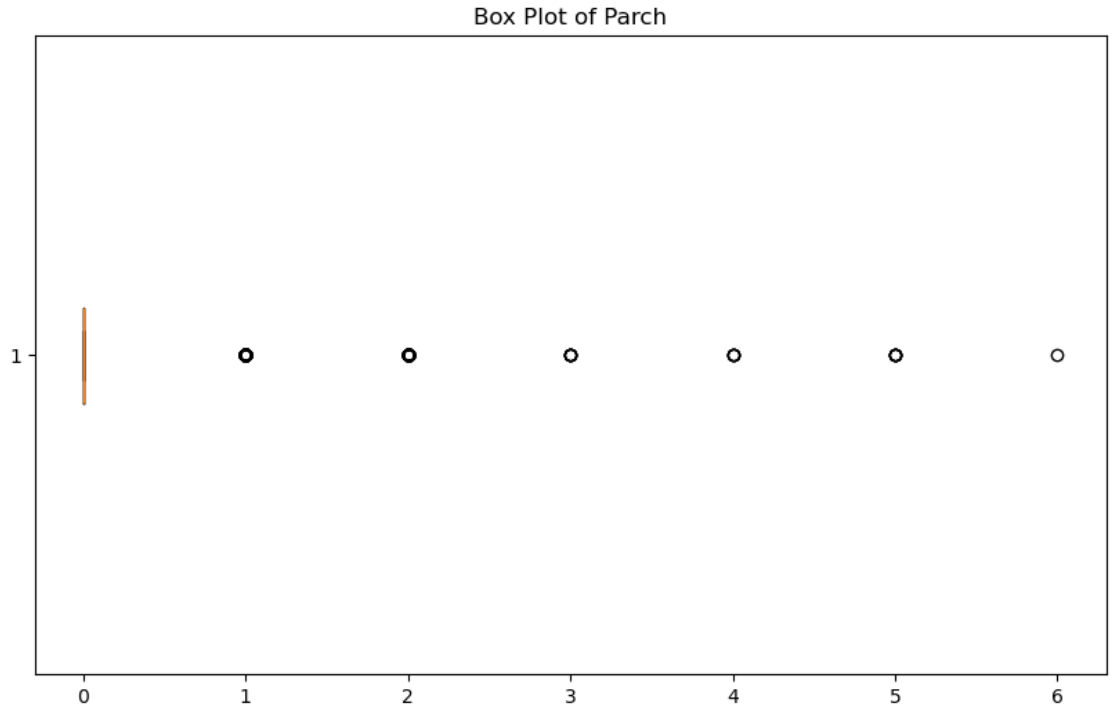


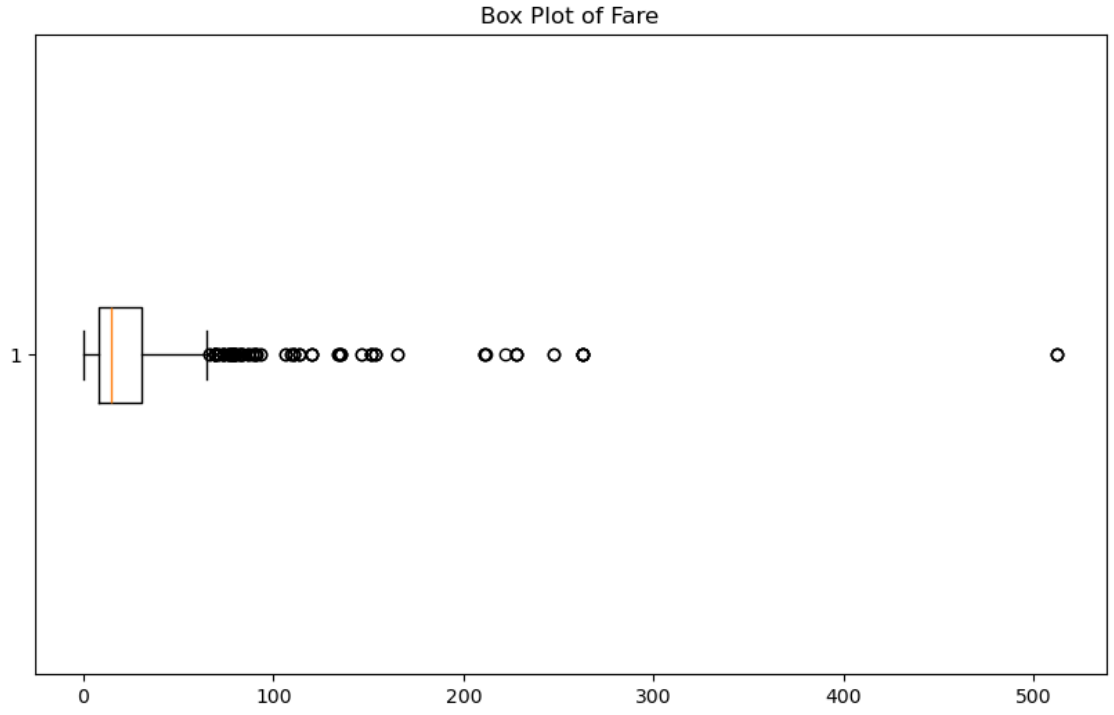

Box Plot of Pclass
1
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

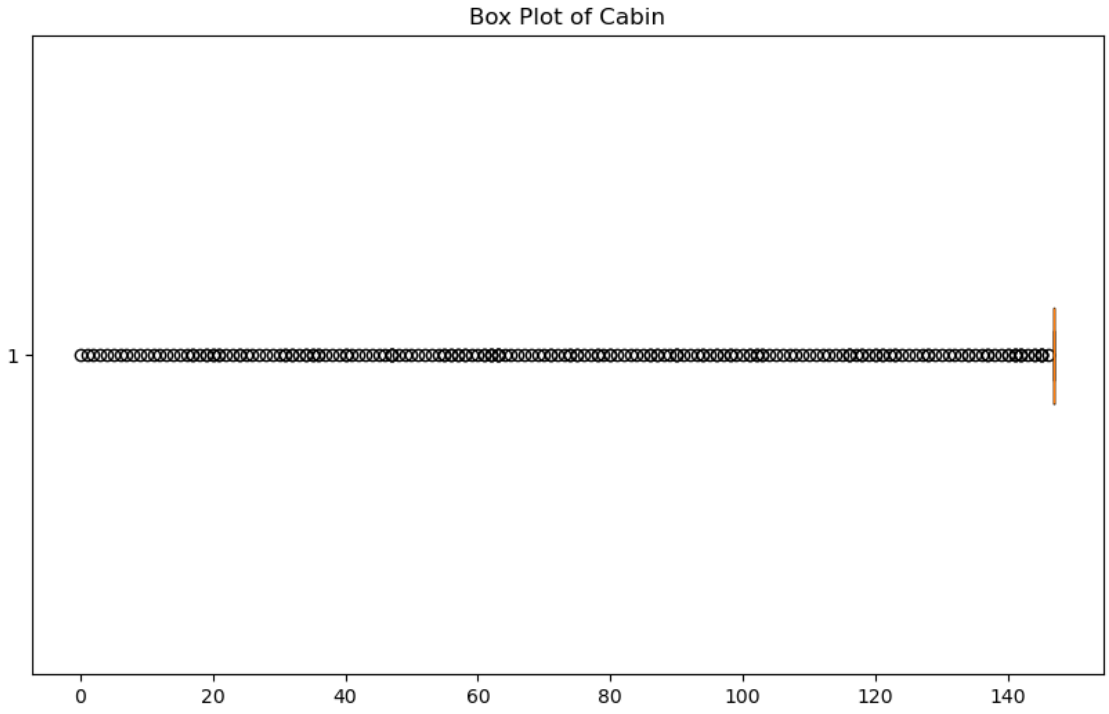

Box Plot of Sex
1
0.0
0.2
0.4
0.6
0.8
1.0

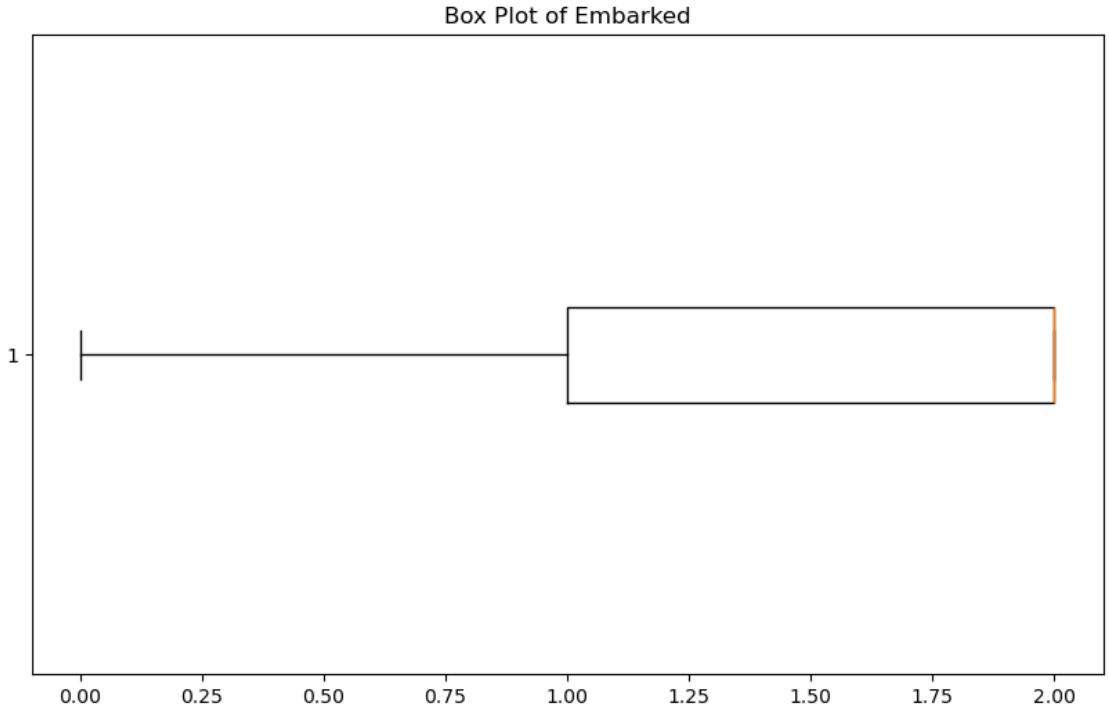

Box Plot of Age
1
0
10
20
30
40
50
60
70
80


Box Plot of SibSp
1
0
1
2
3
4
5
6
7
8


Box Plot of Parch
1
0
1
2
3
4
5
6


Box Plot of Fare
1
0
100
200
300
400
500


Box Plot of Cabin
1
0
20
40
60
80
100
120
140


Box Plot of Embarked
1
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

[24]:
```
#Checking outliers using Z-Score:
from scipy import stats
data[(np.abs(stats.zscore(data)) < 3).all(axis=1)]
#Outliers handled by not removing any data as z score is less than 3.
```

[24]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	1	22.0	1	0	7.2500	147	2
1	1	1	0	38.0	1	0	71.2833	81	0
2	1	3	0	26.0	0	0	7.9250	147	2
3	1	1	0	35.0	1	0	53.1000	55	2
4	0	3	1	35.0	0	0	8.0500	147	2
..	…	…	…	…	…	…	…	…	…
886	0	2	1	27.0	0	0	13.0000	147	2
887	1	1	0	19.0	0	0	30.0000	30	2
888	0	3	0	28.0	1	2	23.4500	147	2
889	1	1	1	26.0	0	0	30.0000	60	0
890	0	3	1	32.0	0	0	7.7500	147	1

[798 rows x 9 columns]

[27]:
```
#Scaling data and selecting independent/dependent variables
#Scaling data using StandardScaler:
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

[28]:
```
data.head()
```

[28]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	1	22.0	1	0	7.2500	147	2
1	1	1	0	38.0	1	0	71.2833	81	0
2	1	3	0	26.0	0	0	7.9250	147	2
3	1	1	0	35.0	1	0	53.1000	55	2
4	0	3	1	35.0	0	0	8.0500	147	2

[51]:
```
#Selecting independent (X) and dependent (y) variables
y = data['Survived']
X = data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
```

[71]:
```
#split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=72)
```

[72]:
```
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

[72]:
```
DecisionTreeClassifier()
```

[73]:
```
y_pred = model.predict(X_test)
print(y_pred)
```

```
[1 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 1 0 1
 0 1 0 1 0 0 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0
 0 0 1 1 0 0 1 0 0 1 1 0 1 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0
 1 0 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0
 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 1 0 1 0 0 0 0 1 1
 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1 1 0 0 1 0 1 0 1
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 1 1 0 1
 0 0 1 0 1 1 1 0 0]
```

[74]:
```
# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred))
```

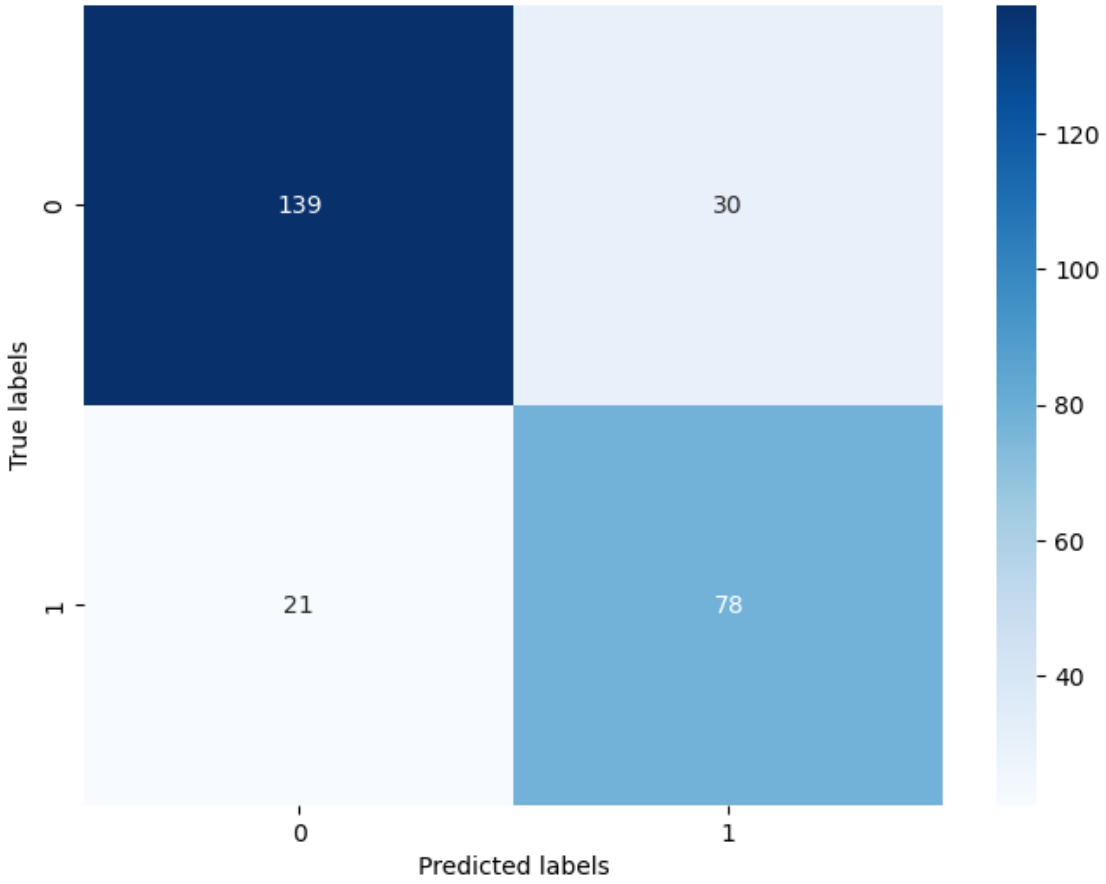
```
Accuracy: 0.8097014925373134
Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.82      0.84       169
           1       0.72      0.79      0.75        99

    accuracy                           0.81       268
   macro avg       0.80      0.81      0.80       268
weighted avg       0.81      0.81      0.81       268

Confusion Matrix: [[139  30]
 [ 21  78]]
```

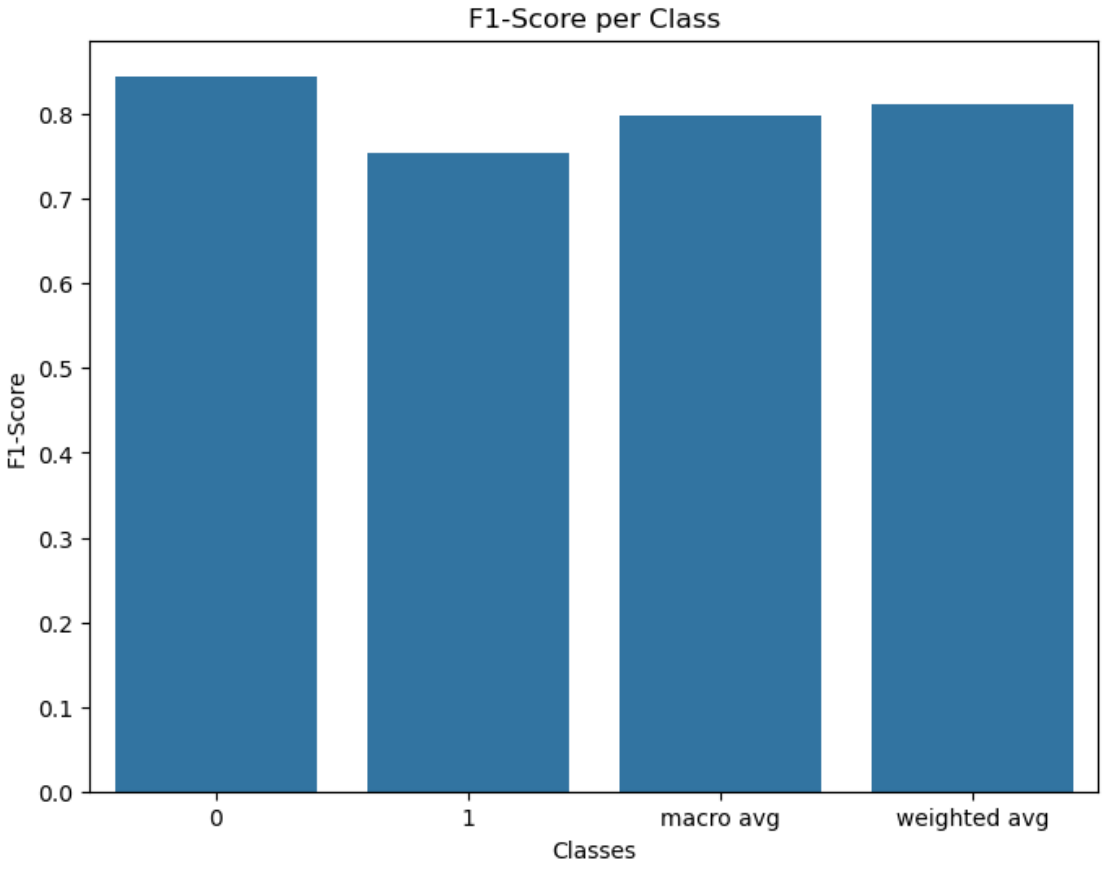
[75]:
```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues', fmt='g')
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.show()
```



[76]:
```
#Classification Report Bar Chart
# Assume y_test and y_pred are defined and correct
report = classification_report(y_test, y_pred, output_dict=True)

# Extract only class-specific keys (exclude 'accuracy', 'macro avg', 'weighted␣
↪avg')
class_keys = [key for key in report.keys() if isinstance(report[key], dict)]
f1_scores = [report[key]['f1-score'] for key in class_keys]

# Plot
plt.figure(figsize=(8, 6))
sns.barplot(x=class_keys, y=f1_scores)
plt.xlabel("Classes")
plt.ylabel("F1-Score")
plt.title("F1-Score per Class")
plt.show()
```


F1-Score per Class
F1-Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1
macro avg
weighted avg
Classes