ABOUT THE DATASET 'TITANIC'

Variable Notes

1. pclass: A proxy for socio-economic status (SES)
   - 1st = Upper
   - 2nd = Middle
   - 3rd = Lower
2. age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5
3. sibsp: The dataset defines family relations in this way
   - Sibling = brother, sister, stepbrother, stepsister
   - Spouse = husband, wife (mistresses and fiancés were ignored)
4. parch: The dataset defines family relations in this way
   - Parent = mother, father
   - Child = daughter, son, stepdaughter, stepson Some children travelled only with a nanny, therefore parch=0 for them.

DATA COLLECTION AND EXPLORATION

In [137…

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

In [110…

```
# LOAD THE DATASET
data = pd.read_csv('tested.csv')
data
```

Out[110]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875
...	...	...	...	...	...	...	...	...	...	...
413	1305	0	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500
414	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000
415	1307	0	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500
416	1308	0	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500
417	1309	0	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583

418 rows × 12 columns

In [111…

```
# INSPECT THE DATA
data.head()
```

Out[111]:

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

In [112…

```
data.tail()
```

Out[112]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
413	1305	0	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500
414	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000
415	1307	0	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500
416	1308	0	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500
417	1309	0	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583

In [113…

```
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         418 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
```

In [114…

```
data.describe()
```

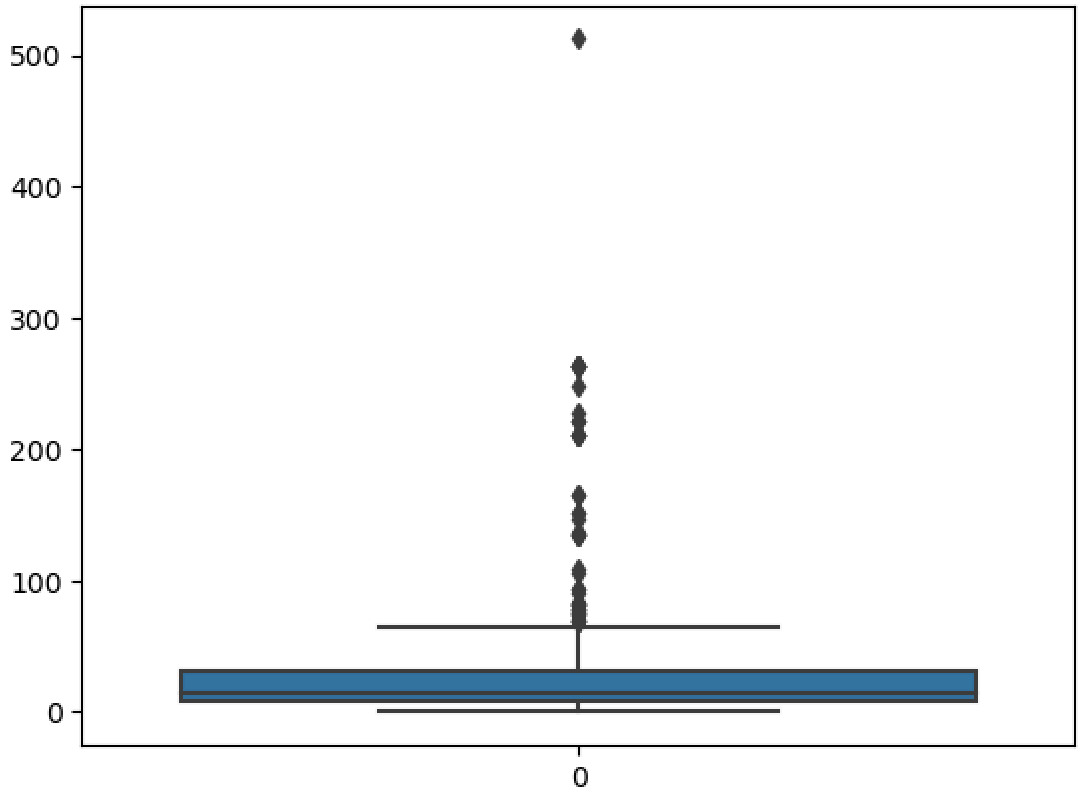
Out[114]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	418.000000	332.000000	418.000000	418.000000	418.000000
mean	1100.500000	0.363636	2.265550	30.272590	0.447368	0.392344	35.605760
std	120.810458	0.481622	0.841838	14.181209	0.896760	0.981429	55.842219
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	0.000000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	0.000000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	1.000000	3.000000	39.000000	1.000000	0.000000	31.471875
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.329200

In [154…

```
sns.boxplot(data['Fare'])
```

Out[154]: <Axes: >

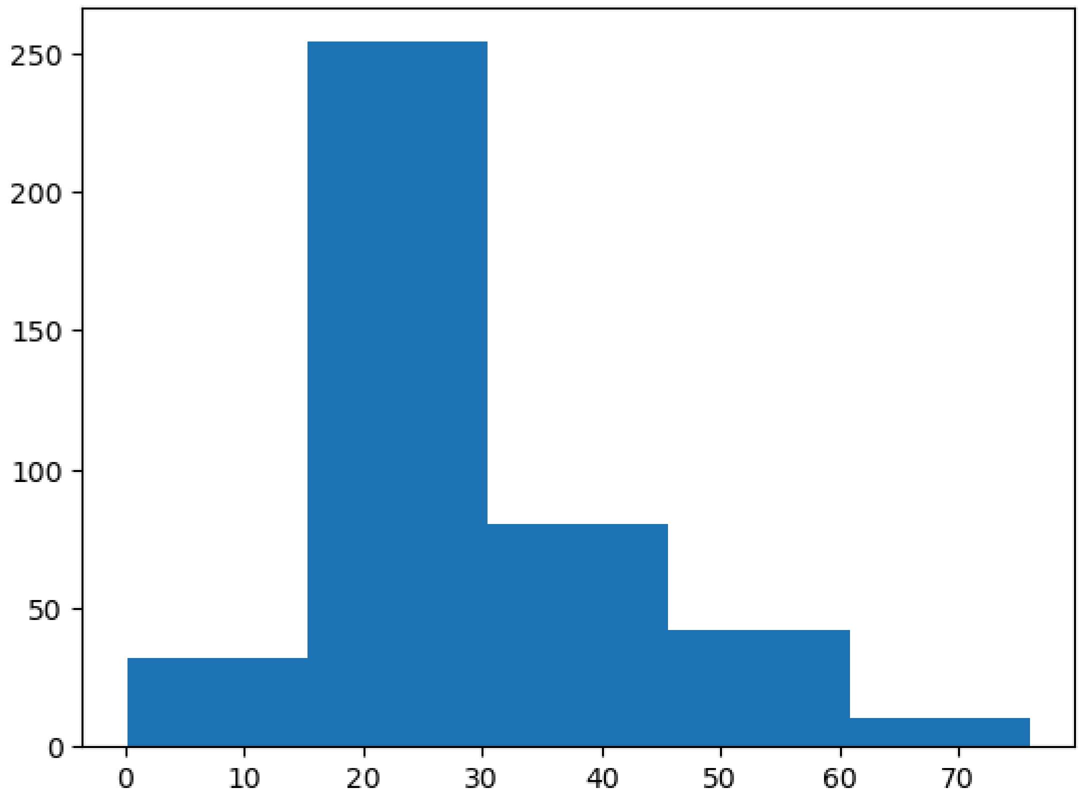


In [155…

```
plt.hist(data['Age'],bins=5)
```

Out[155]:

```
(array([ 32., 254.,  80.,  42.,  10.]),
 array([ 0.17 , 15.336, 30.502, 45.668, 60.834, 76.   ]),
 <BarContainer object of 5 artists>)
```

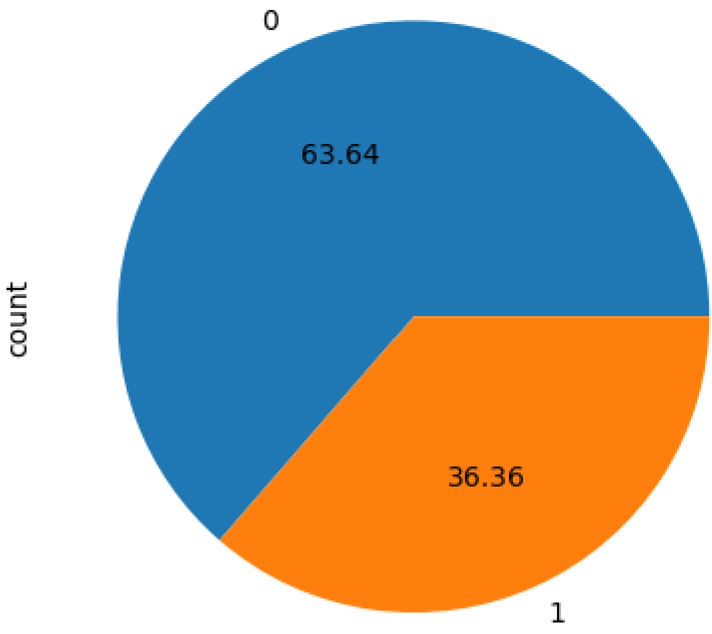


In [161…

```
data['Survived'].value_counts().plot(kind='pie', autopct='%.2f')
```

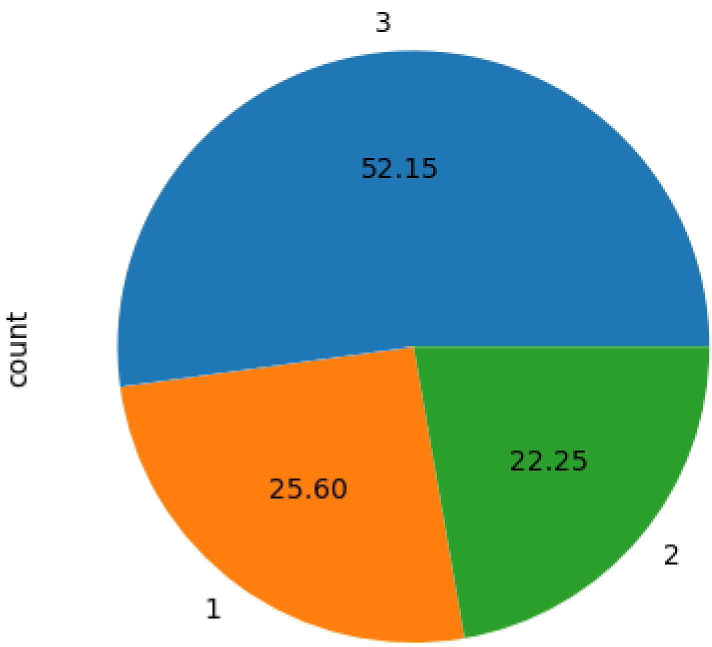
Out[161]:

```
<Axes: ylabel='count'>
```



In [164...
```
data['Pclass'].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[164]: <Axes: ylabel='count'>



DATA PREPROCESSING

HANDLE MISSING VALUES

As we can see there were 86 missing values in Age column so we are filling these missing values by the median value of Age column.

In [115…
```
data['Age'].fillna(data['Age'].median(), inplace=True)
data['Fare'].fillna(data['Fare'].median(), inplace=True)
```

In [116…
```
# Check for missing values
print(data.isnull().sum())
```

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

In [117…
```
# Handle missing values in 'Fare' column
data['Fare'].fillna(data['Fare'].median(), inplace=True)

# Verify that there are no missing values in the 'Fare' column
print(data.isnull().sum())
```

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

In [118…
```
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          418 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         418 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
```

FEATURE ENGINEERING

1. SibSp = Sibling/Spouse & Parch = Parent/Children

Here SibSp and Parch both are telling almost same thing so we can merge this 2 columns to 1 named as 'Total Members'

1. Here we are extracting the courtesy titles from Name column and making a new column named as 'Title'

In [119…
```
data['TotalMember'] = data['SibSp'] + data['Parch'] + 1
data['Title'] = data['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
```

In [120…
```
data.head()
```

Out[120]:

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

ENCODING CATEGORICAL VARIABLES

This function in Pandas is used to convert categorical variables into a format that can be provided to machine learning algorithms to do a better job in prediction.

In [121…
```
data = pd.get_dummies(data, columns=['Sex', 'Embarked', 'Title'], drop_first=True)
```

In [122…
```
data.head()
```

Out[122]:

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

5 rows × 22 columns

DROP UNWANTED COLUMNS

In [123…

```
data.drop(['Name', 'Ticket', 'Cabin', 'PassengerId'], axis=1, inplace=True)
```

In [124…

```
data
```

Out[124]:

	Survived	Pclass	Age	SibSp	Parch	Fare	TotalMember	Sex_male	Embarked_Q	Embar
0	0	3	34.5	0	0	7.8292	1	True	True	
1	1	3	47.0	1	0	7.0000	2	False	False	
2	0	2	62.0	0	0	9.6875	1	True	True	
3	0	3	27.0	0	0	8.6625	1	True	False	
4	1	3	22.0	1	1	12.2875	3	False	False	
...	...	...	...	...	...	...	...	...	...	
413	0	3	27.0	0	0	8.0500	1	True	False	
414	1	1	39.0	0	0	108.9000	1	False	False	
415	0	3	38.5	0	0	7.2500	1	True	False	
416	0	3	27.0	0	0	8.0500	1	True	False	
417	0	3	27.0	1	1	22.3583	3	True	False	

418 rows × 18 columns

DATA SPLITTING

In [125…

```
from sklearn.model_selection import train_test_split
```

1. X = are all the independent valriables [except Survived column]
2. Y = is the dependent variable [only Survived column]

random_state in train_test_split: It controls the random shuffling and splitting of data. Setting it to a specific value ensures the same split is obtained each time for reproducibility. Different values or None produce different random splits.

In [126…

```
X = data.drop('Survived', axis=1)
Y = data['Survived']

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_stat
```

In [127…

```
X_train
```

Out[127]:

	Pclass	Age	SibSp	Parch	Fare	TotalMember	Sex_male	Embarked_Q	Embarked_S	Title
336	2	32.0	0	0	13.0000	1	True	False	True	
31	2	24.0	2	0	31.5000	3	True	False	True	
84	2	27.0	0	0	10.7083	1	True	True	False	
287	1	24.0	1	0	82.2667	2	True	False	True	
317	2	19.0	0	0	10.5000	1	True	False	True	
...	...	...	...	...	...	...	...	...	...	
71	3	21.0	0	0	7.8958	1	True	False	True	
106	3	21.0	0	0	7.8208	1	True	True	False	
270	1	46.0	0	0	75.2417	1	True	False	False	
348	2	24.0	0	0	13.5000	1	True	False	True	
102	3	27.0	0	0	7.7500	1	True	True	False	

334 rows × 17 columns

In [128…

```
X_test
```

Out[128]:

	Pclass	Age	SibSp	Parch	Fare	TotalMember	Sex_male	Embarked_Q	Embarked_S	Titl
321	3	25.0	0	0	7.2292	1	True	False	False	
324	1	39.0	0	0	211.3375	1	False	False	True	
388	3	21.0	0	0	7.7500	1	True	True	False	
56	3	35.0	0	0	7.8958	1	True	False	True	
153	3	36.0	0	2	12.1833	3	False	False	True	
...	...	...	...	...	...	...	...	...	...	
57	3	25.0	0	0	7.6500	1	True	False	True	
126	3	22.0	0	0	7.7958	1	True	False	True	
24	1	48.0	1	3	262.3750	5	False	False	False	
17	3	21.0	0	0	7.2250	1	True	False	False	
66	3	18.0	0	0	7.8792	1	False	True	False	

84 rows × 17 columns

In [129…

```
Y_train
```

Out[129]:

```
336    0
31     0
84     0
287    0
317    0
      ..
71     0
106    0
270    0
348    0
102    0
Name: Survived, Length: 334, dtype: int64
```

In [130…

```
Y_test
```

Out[130]:

```
321    0
324    1
388    0
56     0
153    1
      ..
57     0
126    0
24     1
17     0
66     1
Name: Survived, Length: 84, dtype: int64
```

MODEL SELECTION AND TRAINING

Logistic regression is used here because it is a straightforward, interpretable algorithm well-suited for binary classification tasks. In this context, it helps predict whether a passenger survived (1) or not (0) based on various features. It effectively models the probability of a binary outcome.

In [131…

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
```

```
C:\Users\Nishita Bala\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.p
y:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[131]:

```
▾ LogisticRegression
LogisticRegression()
```

MODEL EVALUATION

In [133…

```
Y_pred = logreg.predict(X_test)
Y_pred
```

Out[133]:

```
array([0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1], dtype=int64)
```

In [136…

```
# Evaluate logistic regression model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print(f'Logistic Regression Accuracy: {accuracy_score(Y_test, Y_pred)}')
print(f'Logistic Regression Precision: {precision_score(Y_test, Y_pred)}')
print(f'Logistic Regression Recall: {recall_score(Y_test, Y_pred)}')
print(f'Logistic Regression F1 Score: {f1_score(Y_test, Y_pred)}')
```

```
Logistic Regression Accuracy: 1.0
Logistic Regression Precision: 1.0
Logistic Regression Recall: 1.0
Logistic Regression F1 Score: 1.0
```

SOME GRAPHICAL REPRESENTATION

Confusion Matrix

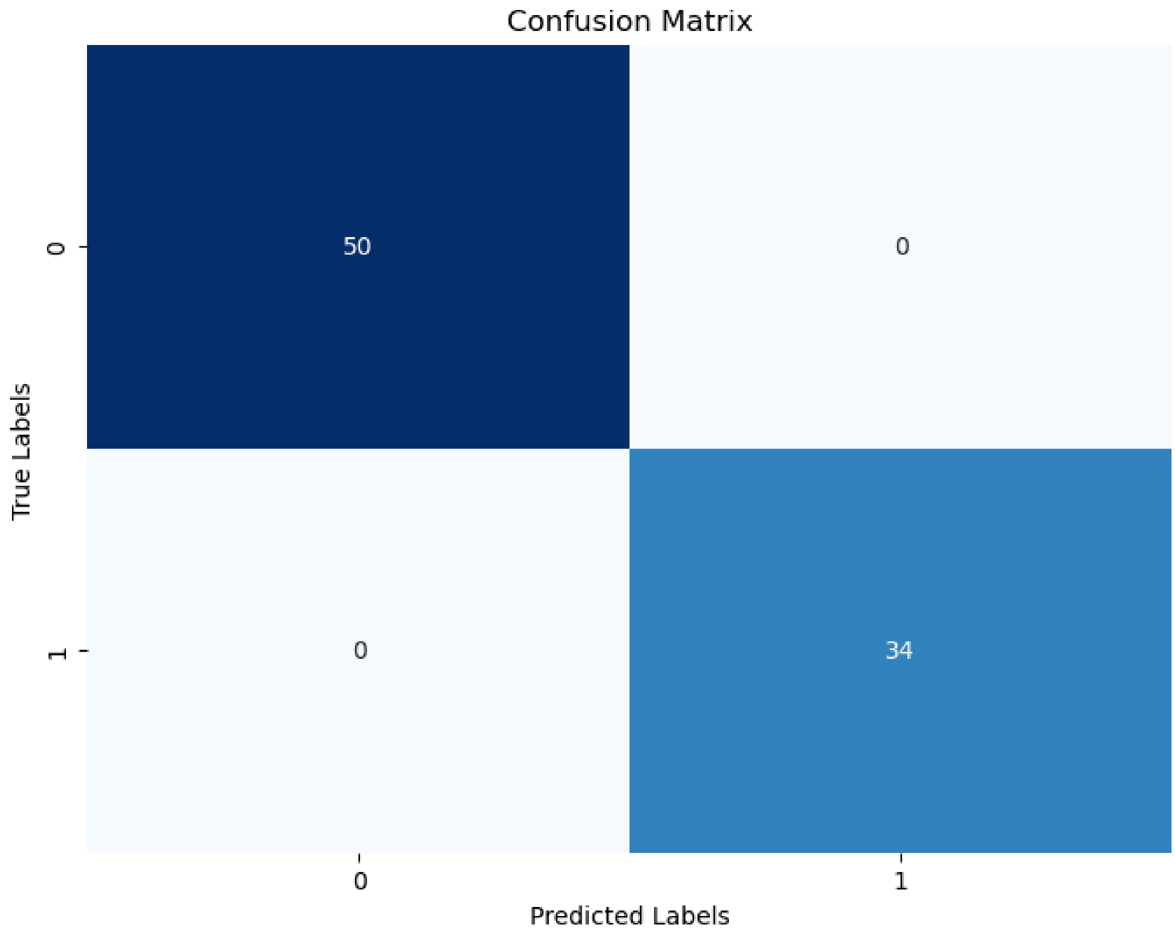
Shows the true positive, false positive, true negative, and false negative predictions of the model. Useful for understanding the model's performance in terms of correctly and incorrectly classified instances.

In [144…

```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(Y_test, Y_pred)
```

In [145…

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



ROC Curve

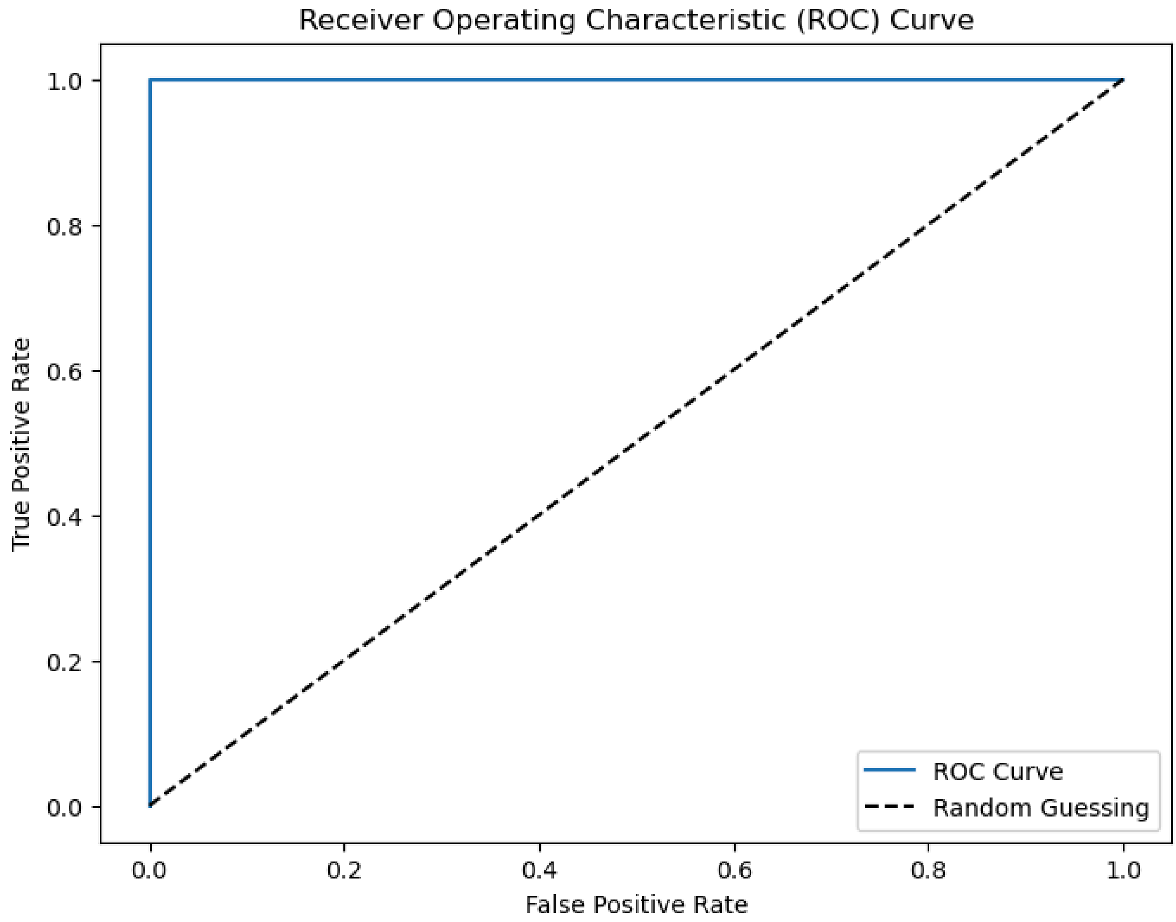
Illustrates the trade-off between sensitivity (true positive rate) and specificity (true negative rate) across different threshold values. AUC (Area Under the Curve) represents the model's ability to distinguish between positive and negative classes.

In [148…

```
from sklearn.metrics import roc_curve, roc_auc_score
Y_pred_proba = logreg.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(Y_test, Y_pred_proba)
```

In [149…

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='ROC Curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



Feature Importance

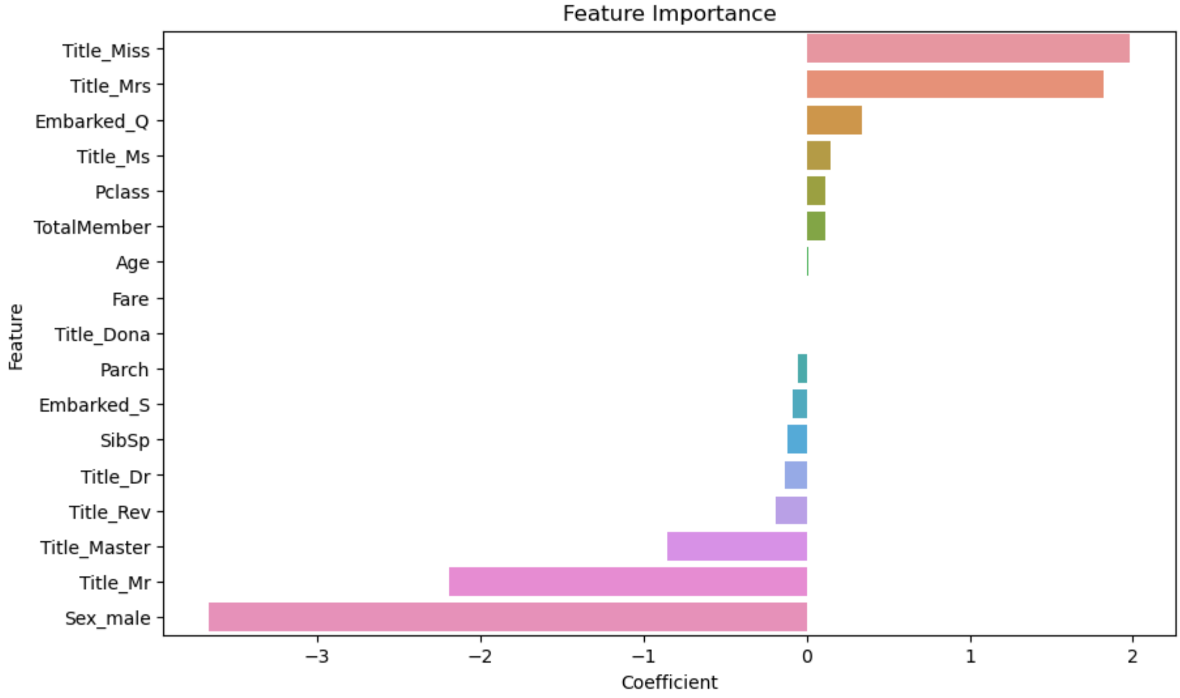
Displays the coefficients assigned to each feature by the logistic regression model. Positive coefficients indicate features that positively contribute to survival probability, while negative coefficients indicate features that negatively contribute.

In [150…

```
coef = logreg.coef_[0]
features = X.columns
feature_importance_df = pd.DataFrame({'Feature': features, 'Coefficient': coef})
feature_importance_df = feature_importance_df.sort_values(by='Coefficient', ascendi
```

In [151…

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Coefficient', y='Feature', data=feature_importance_df)
plt.xlabel('Coefficient')
plt.ylabel('Feature')
plt.title('Feature Importance')
plt.show()
```



CONCLUSION

- Socio-Economic Status: Features like fare price and class may have a significant positive impact on survival, as suggested by their positive coefficients in the feature importance plot.
- Age: Younger passengers may have a higher chance of survival, as indicated by the positive coefficient for age.
- Gender: Being female likely increases the likelihood of survival, as indicated by the positive coefficient for the 'Sex_male' feature.
- These visualizations and analysis provide insights into the factors influencing survival on the Titanic and help in building a system to predict survival likelihood.