Titanic Survival Project

The goal of the project is to create a machine learning model that can predict whether or not a passenger will survive the historic Titanic shipwreck. The Titanic dataset from Kaggle, which includes details about the passengers like age, gender, and ticket class, will be used in our work. We will train a logistic regression model and use GridSearchCV to optimize its hyperparameters in order to attain the best possible performance after completing the required data preprocessing steps. To determine how well the model predicts passenger survival, a hold-out test set will be used to analyze its accuracy and classification report.

Section : K22BW

Team Members

1. *Indrajith M P || 12211823 || Roll number: 8*
2. *Devika E S || 12211824 || Roll number: 9*
3. *Milan Sihag || 12223432 || Roll number: 66*
4. *Sujal Verma || 12214775 || Roll number: 15*

This code block imports the necessary libraries and modules required for data manipulation, visualization, model building, and evaluation.

In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [2]:
```python
# load the data from csv file to Pandas DataFrame
titanic_data = pd.read_csv('train.csv')
```

In [3]:
```python
# printing the first 5 rows of the dataframe
titanic_data.head()
```

Out[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

Additional Points to note about the data

** *pclass*: A proxy for socio-economic status (SES)**
1st = Upper
2nd = Middle
3rd = Lower

***sibsp*: The dataset defines family relations in this way...**
Sibling = brother, sister, stepbrother, stepsister
Spouse = husband, wife (mistresses and fiancés were ignored)

---

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.

In [4]:
```python
titanic_data.describe() # provides a statistical summary of the numeric columns in the Titanic dataset
```

Out[4]:

	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

In [5]:
```python
# number of rows and Columns
titanic_data.shape
```

Out[5]:
```
(891, 12)
```

In [6]:
```python
# getting some informations about the data
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
```

In [7]:
```python
# check the number of missing values in each column
titanic_data.isnull().sum()
```

Out[7]:
```
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
```

In [8]:
```python
# drop the "Cabin" column from the dataframe
titanic_data = titanic_data.drop(columns='Cabin', axis=1)
```

Replacing the missing values in "Age" column with mean value

In [9]:
```python
titanic_data['Age'].fillna(titanic_data['Age'].mean(), inplace=True)
```

In [10]:
```python
# finding the mode value of "Embarked" column
print(titanic_data['Embarked'].mode())
```

```
0    S
Name: Embarked, dtype: object
```

In [11]:
```python
# replacing the missing values in "Embarked" column with mode value
titanic_data['Embarked'].fillna(titanic_data['Embarked'].mode()[0], inplace=True)
```

In [12]:
```python
# check the number of missing values in each column
titanic_data.isnull().sum()
```

Out[12]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

In [13]:
```python
# getting some statistical measures about the data
titanic_data.describe()
```

Out[13]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	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	13.002015	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	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

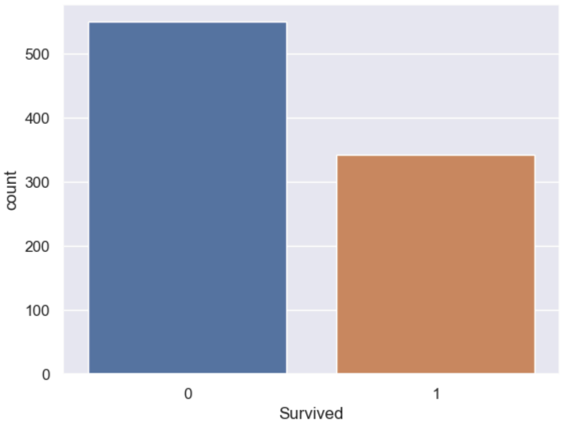
In [14]:
```python
# finding the number of people survived and not survived
titanic_data['Survived'].value_counts()
```

Out[14]:
```
Survived
0    549
1    342
Name: count, dtype: int64
```

In [15]:
```python
sns.set()
```

In [16]:
```python
# making a count plot for "Survived" column
sns.countplot(x='Survived', data=titanic_data)
```

Out[16]:
```
<Axes: xlabel='Survived', ylabel='count'>
```

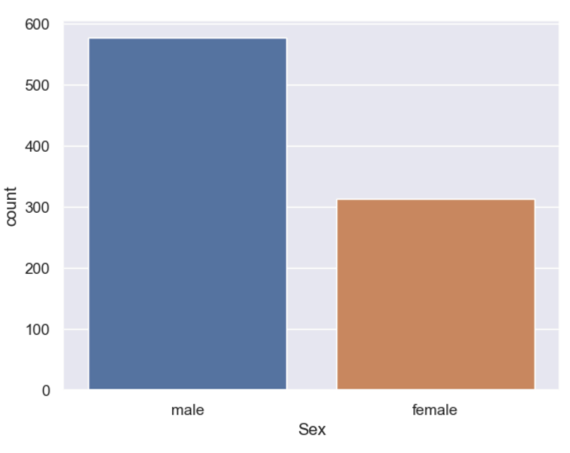


In [17]:
```python
titanic_data['Sex'].value_counts()
```

Out[17]:
```
Sex
male      577
female    314
Name: count, dtype: int64
```

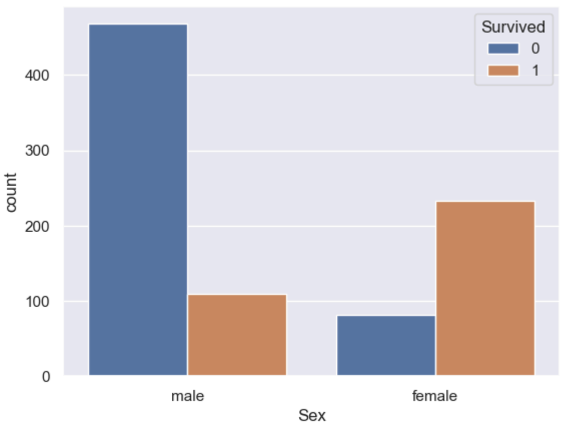
In [18]:
```python
# making a count plot for "Sex" column
sns.countplot(x='Sex', data=titanic_data)
```

Out[18]:
```
<Axes: xlabel='Sex', ylabel='count'>
```



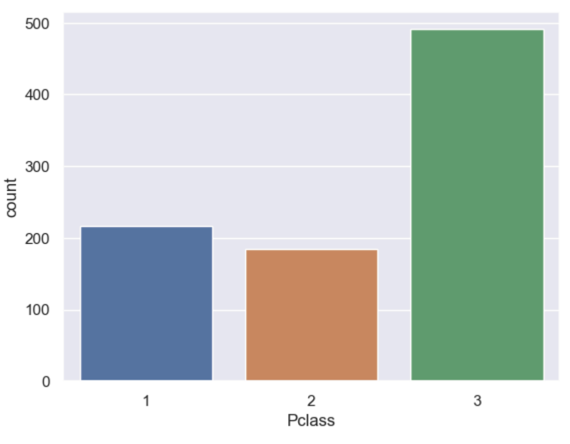
In [19]:
```python
# number of survivors Gender wise
sns.countplot(x='Sex', hue='Survived', data=titanic_data)
```

Out[19]:
```
<Axes: xlabel='Sex', ylabel='count'>
```



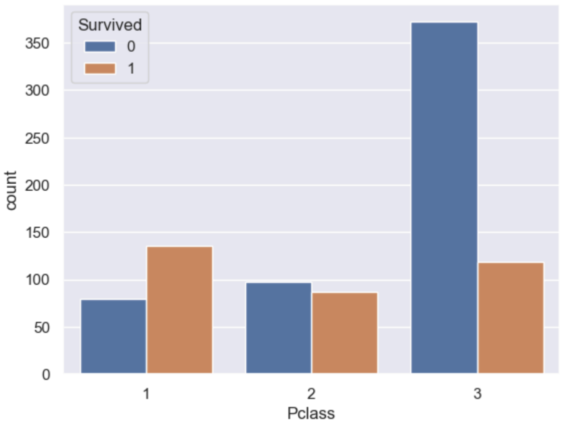
In [20]:
```python
# making a count plot for "Pclass" column
sns.countplot(x='Pclass', data=titanic_data)
```

Out[20]:
```
<Axes: xlabel='Pclass', ylabel='count'>
```



In [21]:
```python
sns.countplot(x='Pclass', hue='Survived', data=titanic_data)
```

Out[21]:
```
<Axes: xlabel='Pclass', ylabel='count'>
```



In [22]:
```python
titanic_data['Sex'].value_counts()
```

Out[22]:
```
Sex
male      577
female    314
Name: count, dtype: int64
```

In [23]:
```python
titanic_data['Embarked'].value_counts()
```

Out[23]:
```
Embarked
S    646
C    168
Q     77
Name: count, dtype: int64
```

Converting categorical Columns

In [24]:
```python
titanic_data.replace({'Sex':{'male':0,'female':1}, 'Embarked':{'S':0,'C':1,'Q':2}}, inplace=True)
```

In [25]:
```python
# getting some informations about the data
titanic_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 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    int64
 5   Age          891 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  Embarked     891 non-null    int64
dtypes: float64(2), int64(7), object(2)
memory usage: 76.7+ KB
```

Separating features & Target

In [26]:
```python
X = titanic_data.drop(columns = ['PassengerId','Name','Ticket','Survived'],axis=1)
Y = titanic_data['Survived']
```

In [27]:
```python
print(X)
```

```
     Pclass  Sex        Age  SibSp  Parch     Fare  Embarked
0         3    0  22.000000      1      0   7.2500         0
1         1    1  38.000000      1      0  71.2833         1
2         3    1  26.000000      0      0   7.9250         0
3         1    1  35.000000      1      0  53.1000         0
4         3    0  35.000000      0      0   8.0500         0
..      ...  ...        ...    ...    ...      ...       ...
886       2    0  27.000000      0      0  13.0000         0
887       1    1  19.000000      0      0  30.0000         0
888       3    1  29.699118      1      2  23.4500         0
889       1    0  26.000000      0      0  30.0000         1
890       3    0  32.000000      0      0   7.7500         2

[891 rows x 7 columns]
```

In [28]:
```python
print(Y)
```

```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

Principal Component Analysis (PCA) for dimensionality reduction on the features did not improve the model's accuracy, so it was decided not to use PCA in the final model.
Linear Discriminant Analysis (LDA), being a supervised technique, was able to capture discriminative information better than PCA by finding directions that maximize the separation between classes, leading to an increase in model accuracy.

In [29]:
```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Create an instance of the LDA classifier
lda = LinearDiscriminantAnalysis()

# Fit the LDA model to the data and transform X
X_lda = lda.fit_transform(X, Y)
```

This line of code splits the dataset into training and testing sets, with 20% of the data allocated for testing, ensuring a random state of 42 for reproducibility.

In [30]:
```python
X_train, X_test, Y_train, Y_test = train_test_split(X_lda,Y, test_size=0.2, random_state=42)
```

In [31]:
```python
print(X_lda.shape, X_train.shape, X_test.shape)
```

```
(891, 1) (712, 1) (179, 1)
```

Initializing a Logistic Regression model object, which will be used for training and making predictions on the Titanic dataset.

In [32]:
```python
model = LogisticRegression()
```

Defining the parameter grid for hyperparameter tuning of the Logistic Regression model, specifying the values to be evaluated for the regularization strength (C), the type of regularization penalty (l1 or l2), and the optimization solver (liblinear or saga).

In [33]:
```python
# Perform hyperparameter tuning
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga']
}
```

Initializing a GridSearchCV object with the Logistic Regression model and the specified parameter grid, and then fit the GridSearchCV object to the training data using 5-fold cross-validation and accuracy as the scoring metric, utilizing all available CPU cores for parallel processing.

In [34]:
```python
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, Y_train)
```

Out[34]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [35]:
```python
# Get the best hyperparameters
best_params = grid_search.best_params_
print('Best Hyperparameters:', best_params)
```

```
Best Hyperparameters: {'C': 0.01, 'penalty': 'l1', 'solver': 'saga'}
```

Creating the final Logistic Regression model using the best hyperparameters found during the grid search process and fit the model to the training data.

In [36]:
```python
final_model = grid_search.best_estimator_
final_model.fit(X_train, Y_train)
```

Out[36]:
```
▾ LogisticRegression
LogisticRegression(C=0.01, penalty='l1', solver='saga')
```

The trained final Logistic Regression model is used to make predictions on the test data, generating class labels (survived or not survived) for the test instances.

In [37]:
```python
Y_pred = final_model.predict(X_test)
```

In [38]:
```python
from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       105
           1       0.82      0.69      0.75        74

    accuracy                           0.81       179
   macro avg       0.81      0.79      0.80       179
weighted avg       0.81      0.81      0.81       179
```

In [39]:
```python
# accuracy on training data
X_train_prediction = final_model.predict(X_train)
```

In [40]:
```python
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.8132022471910112
```

In [41]:
```python
# accuracy on test data
X_test_prediction = final_model.predict(X_test)
```

In [42]:
```python
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print('Accuracy score of test data : ', test_data_accuracy)
```

```
Accuracy score of test data :  0.8100558659217877
```

To conclude, the Logistic Regression model with optimized hyperparameters through GridSearchCV achieved a reasonable performance on the Titanic dataset, predicting passenger survival with an overall accuracy of 0.81. Additionally, Principal Component Analysis (PCA) was explored for dimensionality reduction, but it reduced the model's accuracy to around 60%, hence it was not used in the final model, Instead (Linear Discriminant Analysis) LDA was used which further improved the accuracy from 79 to 81%