Experiment No. 2
Analyze the Titanic Survival Dataset and apply appropriate regression technique
Date of Performance: 31-07-2023
Date of Submission: 10-08-2023

Aim: Analyze the Titanic Survival Dataset and apply appropriate Regression Technique.

Objective: Able to perform various feature engineering tasks, apply logistic regression on the given dataset and maximize the accuracy.

Theory:

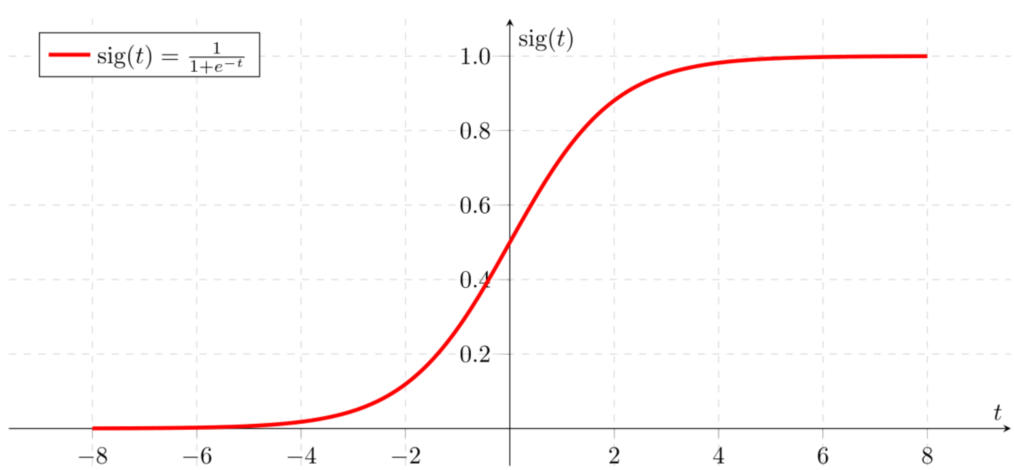
Logistic Regression was used in the biological sciences in the early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical and is binary in nature. In order to perform binary classification the logistic regression techniques make use of Sigmoid function.

For example,

To predict whether an email is spam (1) or (0)

Whether the tumor is malignant (1) or not (0)

Consider a scenario where we need to classify whether an email is spam or not. If we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, predicted continuous value 0.4 and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequences in real time.



From this example, it can be inferred that linear regression is not suitable for classification problems. Linear regression is unbounded, and this brings logistic regression into picture. Their value strictly ranges from 0 to 1.

Dataset:

The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the widely considered "unsinkable" RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren't enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew.

While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. In this challenge, we ask you to build a predictive model that answers the question: "what sorts of people were more likely to survive?" using passenger data (ie name, age, gender, socio-economic class, etc).

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Variable Notes

pclass: A proxy for socio-economic status (SES)

1st = Upper, 2nd = Middle, 3rd = Lower

age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

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 traveled only with a nanny, therefore parch=0 for them.

```
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
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
# load the data from csv file to Pandas DataFrame
titanic_data = pd.read_csv('/content/train.csv')
```

```
# printing the first 5 rows of the dataframe
titanic_data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embark
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th	female	38.0	1	0	PC 17599	71.2833	C85	

```
# number of rows and Columns
titanic_data.shape
```

```
(891, 12)
```

```
# 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
```

```
# check the number of missing values in each column
titanic_data.isnull().sum()
```

```
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
```

```
# 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
titanic_data['Age'].fillna(titanic_data['Age'].mean(), inplace=True)
```

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

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

```
print(titanic_data['Embarked'].mode()[0])
```

```
S
```

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

```
# check the number of missing values in each column
titanic_data.isnull().sum()
```

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

```
# getting some statistical measures about the data
titanic_data.describe()
```

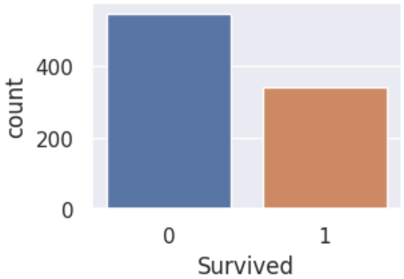
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	13.002015	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	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

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

```
0    549
1    342
Name: Survived, dtype: int64
```

```
sns.set()
```

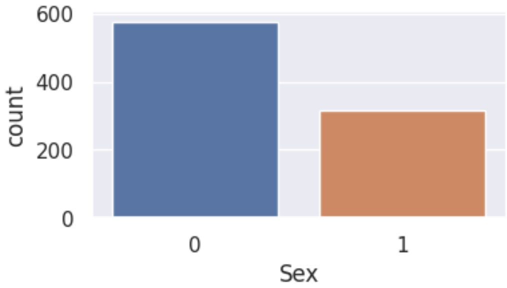
```
# making a count plot for "Survived" column
# Set the figure size
plt.figure(figsize=(3, 2))  # Adjust the width and height as needed
# Create the count plot
sns.countplot(x='Survived', data=titanic_data)
# Display the plot
plt.show()
```



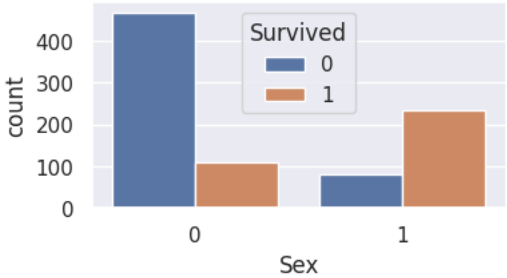

```
titanic_data['Sex'].value_counts()
```

```
male      577
female    314
Name: Sex, dtype: int64
```

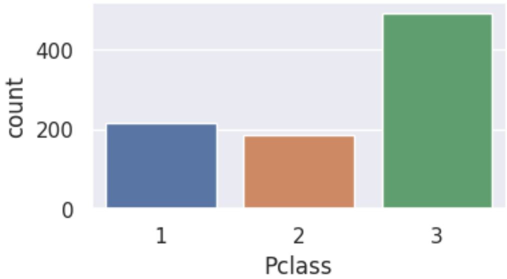
```
# making a count plot for "Sex" column
plt.figure(figsize=(4, 2))
sns.countplot(x='Sex', data=titanic_data)
plt.show()
```




```
# number of survivors Gender wise
plt.figure(figsize=(4, 2))
sns.countplot(x='Sex', hue='Survived', data=titanic_data)
plt.show()
```




```
# making a count plot for "Pclass" column
plt.figure(figsize=(4, 2))
sns.countplot(x='Pclass', data=titanic_data)
plt.show()
```




```
plt.figure(figsize=(4, 2))
sns.countplot(x='Pclass', hue='Survived', data=titanic_data)
plt.show()
```


```
titanic_data['Sex'].value_counts()
```

```
male      577
female    314
```

```
Name: Sex, dtype: int64
```

```
titanic_data['Embarked'].value_counts()
```

```
S    646
C    168
Q     77
Name: Embarked, dtype: int64
```

```
# converting categorical Columns

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

```
titanic_data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599	71.2833	1
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	0

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

```
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]
```

```
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
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.33, random_state=42)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(891, 7) (596, 7) (295, 7)
```

```
model = LogisticRegression()
```

```
# training the Logistic Regression model with training data
model.fit(X_train, Y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
# accuracy on training data
X_train_prediction = model.predict(X_train)
```

```
print(X_train_prediction)
```

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

```
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.8003355704697986
```

```
# accuracy on test data
X_test_prediction = model.predict(X_test)
```

```
print(X_test_prediction)
```

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

```
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print('Accuracy score of test data : ', test_data_accuracy)
```

```
Accuracy score of test data :  0.8135593220338984
```

```
from sklearn.metrics import confusion_matrix
pd.DataFrame(confusion_matrix(Y_test, X_test_prediction),columns=['Predicted No','Predicted Yes'],index=['Actual No','Actual Yes'])
```

	Predicted No	Predicted Yes
Actual No	153	22
Actual Yes	33	87

```
from sklearn.metrics import classification_report
print(classification_report(Y_test, X_test_prediction))
```

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.85       175
           1       0.80      0.72      0.76       120

    accuracy                           0.81       295
   macro avg       0.81      0.80      0.80       295
weighted avg       0.81      0.81      0.81       295
```

Conclusion:

The features that have been chosen to develop the model are as follows:

1. Pclass (Passenger Class): This is the class of the passenger's ticket (1st, 2nd, or 3rd class). It can be a valuable indicator as higher classes might have been given preference duringemergencies.

2. Sex: This feature represents the gender of the passenger Historically, during the Titanic disaster, the "women and children first" policy was followed during evacuation, potentially making gender a significant factor in survival prediction,

3. Age: The age of the passenger is an important consideration. It's known that children and elderly passengers might have had different survival rates compared to young adults.

4. SibSp (Number of Siblings/Spouses Aboard). This feature indicates the presence of family members (siblings or spouses) on board. It might influence the decision-making process during evacuation

5 Parch (Number of Parents/Children Aboardy Similar to 'SibSp', this feature represents the presence of parents or children on board, which could impact survival decisions.

6. Fare: The fare paid by the passenger might correlate with their class or accommodations. which could affect their access to lifeboats or safety measures.

7. Embarked: This feature represents the port of embarkation (S=Southampton, C= Cherbourg, Q = Queenstown). Different embarkation points might have socio-economic implications, potentially affecting survival rates.

Accuracy:-

1.Training Accuracy (80.03%): This means that your model correctly predicts the survival outcome for approximately 80.03% of the training data. In other words, out of all the instances in your training dataset, your model accurately predicted the survival outcome for about 80.03% of them.

2.Test Accuracy (81.36%): Similarly, this indicates that your model correctly predicts the survival outcome for around 81.36% of the test data. Out of all the instances in your test dataset, your model accurately predicted the survival outcome for about 81.36% of them.

The fact that the accuracy scores for both training and test data are relatively close suggests that your model is performing consistently on both datasets. A small difference between training and test accuracy indicates that the model is not significantly overfitting or underfitting the data. It's important to note that the actual values of the accuracy percentages are more important than the specific difference between them.