Experiment No. 2
Analyze the Titanic Survival Dataset and apply appropriate regression technique
Date of Performance: 31—07—23
Date of Submission: 7—08—23

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:

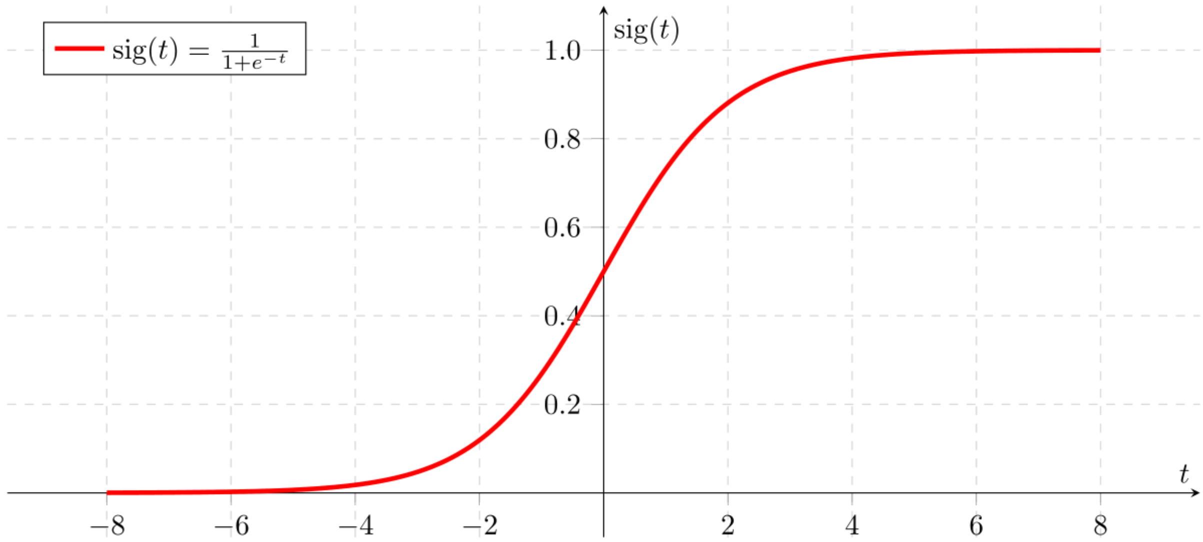
Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical and is binary in nature. In order to perform binary classification the logistic regression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



From this example, it can be inferred that linear regression is not suitable for classification problem. 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, socioeconomic 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 travelled only with a nanny, therefore parch=0 for them

Code:

Conclusion:

Passenger class (Pclass), gender (Sex), age (Age), number of siblings/spouses aboard (SibSp), and number of parents/children (Parch) are the features used for model construction. These characteristics are important because they may have an impact on survival rates and represent socioeconomic factors, such as giving higher-class passengers priority, prioritising women during evacuations, giving children and the elderly priority, allowing family members to travel, and relationships between departure port and socioeconomic backgrounds. An accuracy score of 0.8076 for the training data shows that the model correctly forecasts survival outcomes for this dataset. According to the test data accuracy score of 0.7821, the model performs well with fresh, untested data. These accuracy values indicate that the model successfully generalizes novel data.

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

```
data = pd.read_csv("/content/train (2).csv")
```

```
print(data)
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0      1
1                              Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                              Heikkinen, Miss. Laina  female  26.0      0
3                              Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                              Allen, Mr. William Henry    male  35.0      0   ..
                               ...     ...   ...    ...
886                               Montvila, Rev. Juozas    male  27.0      0
887                               Graham, Miss. Margaret Edith  female  19.0      0
888                               Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                               Behr, Mr. Karl Howell    male  26.0      0
890                               Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S  ..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

```
data.shape
```

```
(891, 12)
```

```
data.info()     # getting some informations about the data
```

```
<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
```

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

```
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
```

```
data = data.drop(columns='Cabin', axis=1)
```

```
data['Age'].fillna(data['Age'].mean(), inplace=True)  # replacing the missing values in "Age" column with mean value
```

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

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

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

```
S
```

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

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

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

```
data.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.0
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.2
std	257.353842	0.486592	0.836071	13.002015	1.102743	0.806057	49.6
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.0
25%	223.500000	0.000000	2.000000	22.000000	0.000000	0.000000	7.9
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.4
75%	668.500000	1.000000	3.000000	35.000000	1.000000	0.000000	31.0
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.3

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

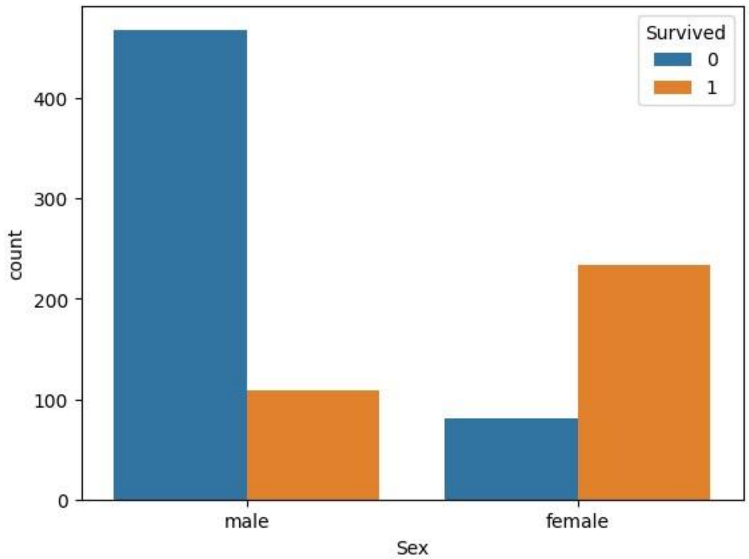
```
0    549
1    342
Name: Survived, dtype: int64
```

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

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

```
# number of survivors Gender wise
# 1st male and other female # 0 are the one who
did not survived sns.countplot(x='Sex',
hue='Survived', data=data)
```

```
<Axes: xlabel='Sex', ylabel='count'>
```




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

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

```
# converting categorical Columns
data.replace({'Sex':{'male':0,'female':1}, 'Embarked':{'S':0,'C':1,'Q':2}}, inplace=True)
```

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

```
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs	1	38.0	1	0	PC 17599	71.2

```
X.head()
```

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

```
Y.head()
```

```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

```
#Splitting the data into training data & Test data
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state=2)
print(X.shape, X_train.shape, X_test.shape)
```

```
(891, 7) (712, 7) (179, 7)
```

```
logr = LogisticRegression()
```

```
# training the Logistic Regression model with training data
logr.fit(X_train, Y_train) /usr/local/lib/python3.10/dist-
```

```
packages/sklearn/linear_model/_logistic.py:458: Con 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:
# accurac    y on training dat https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio a    n
X_train_prediction = logr.predict(X_train)  n_iter_i = _check_optimize_result(
training_data_accuracy = accuracy_score(Y_train, X▾LogisticRegression
        _train_prediction) print("Accuracy score of training data:",
training_data_accuracy) LogisticRegression()

    Accuracy score of training data: 0.8075842696629213
```

```
# accuracy on test data X_test_prediction =
logr.predict(X_test) test_data_accuracy =
accuracy_score(Y_test, X_test_prediction) print('Accuracy
score of test data : ', test_data_accuracy)

    Accuracy score of test data :  0.7821229050279329
```