Experiment No. 2
Analyze the Titanic Survival Dataset and Apply appropriate Regression Technique
Date of Performance:21/07/2024
Date of Submission: 05/08/2024

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:

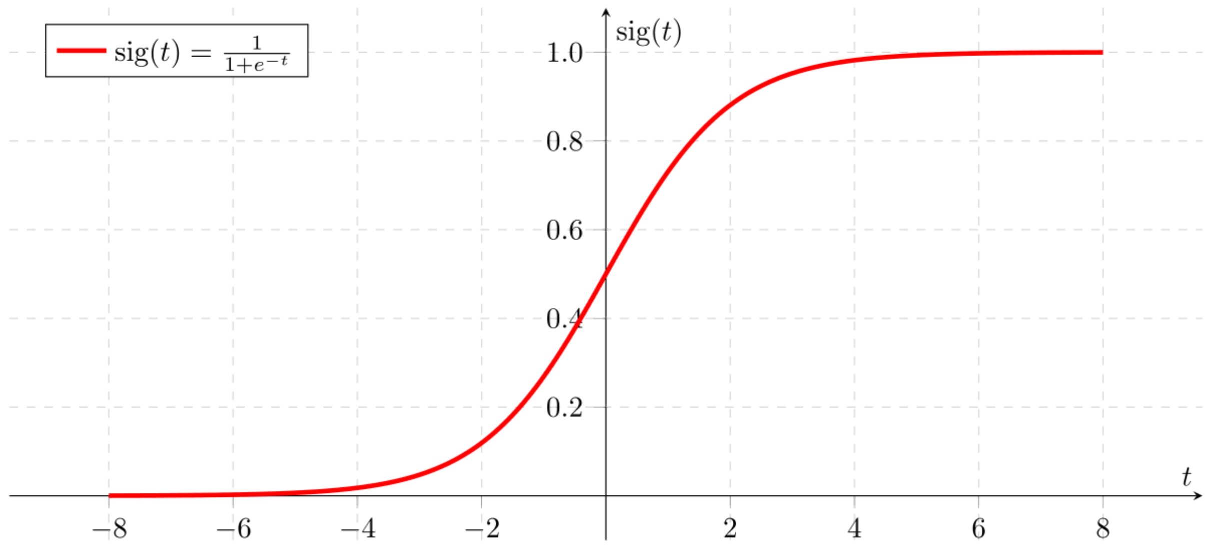
Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical and is binary in nature. In order to perform binary classification the logistic regression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

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

Conclusion:

The model predicts a 91.6% chance of survival for a 30-year-old woman traveling in first class on the Titanic, which makes sense given the historical context. During the Titanic disaster, women, especially those in first class, were given priority for lifeboats due to the "women and children first" policy. This means that your model has learned from the data that gender and social class were crucial factors in determining who survived. The high survival probability reflects the model's understanding that being a woman in a higher class greatly increased the chances of survival during that tragic event.

Open in Colab

Load and prepare the dataset

The first step is to load the dataset and prepare it for training a machine-learning model. One of the reasons the Titanic dataset is popular is that it provides ample opportunity for data scientists to practice their data-cleaning skills.

```
import pandas as pd

df = pd.read_csv('titanic.csv')
df.head()
```

	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

Determine whether any of the columns in the dataset are missing values.

```
df.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
```

Filter the dataset to remove columns that won't be used, one-hot-encode the "Sex" and "Pclass" columns, and remove rows with missing values.

```
df = df[['Survived', 'Age', 'Sex', 'Pclass']]
df = pd.get_dummies(df, columns=['Sex', 'Pclass'])
df.dropna(inplace=True)
df.head()
```

	Survived	Age	Sex_female	Sex_male	Pclass_1	Pclass_2	Pclass_3
0	0	22.0	False	True	False	False	True
1	1	38.0	True	False	True	False	False
2	1	26.0	True	False	False	False	True
3	1	35.0	True	False	True	False	False
4	0	35.0	False	True	False	False	True

The next step is to split the data into two datasets: one for training and one for testing. We'll use a stratified split to create a balanced distribution of samples in the training dataset and the testing dataset.

```
from sklearn.model_selection import train_test_split

x = df.drop('Survived', axis=1)
y = df['Survived']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y, random_
```

Build and train a model

Our classifier will use logistic regression. Logistic regression not only makes predictions, it yields probabilities as well.

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state=0)
model.fit(x_train, y_train)
```

```
▾       LogisticRegression
LogisticRegression(random_state=0)
```

Get an accuracy score.

```
model.score(x_test, y_test)
```

```
0.8321678321678322
```

Cross-validate the model using five folds.

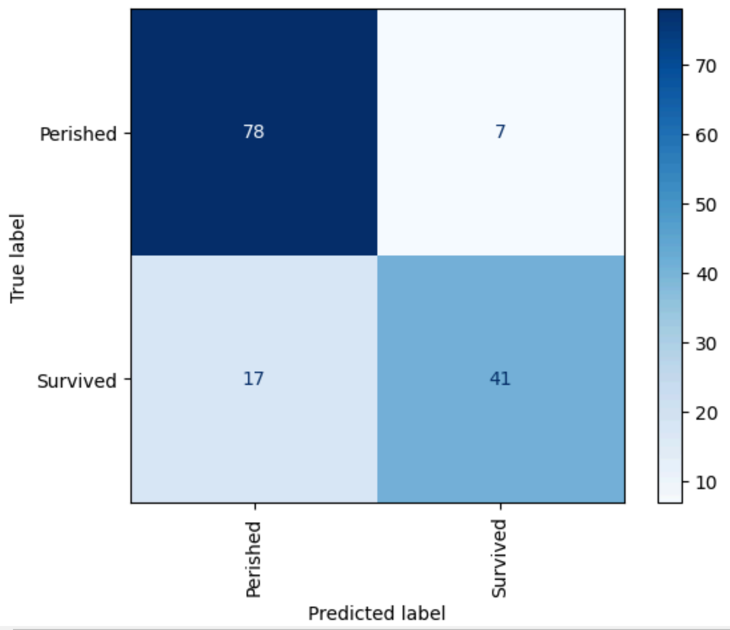
Use a confusion matrix to see how the model performed during testing.

```
from sklearn.metrics import confusion_matrix

y_predicted = model.predict(x_test)
confusion_matrix(y_test, y_predicted)
```

```
array([[78,  7],
       [17, 41]])
```

Print the confusion matrix again, but this time use `plot_confusion_matrix` for clarity.

```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

# Assuming you have already fitted your model and have predictions
y_pred = model.predict(x_test)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Create a ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=['Perished', 'Sur

# Plot the confusion matrix
disp.plot(cmap='Blues', xticks_rotation='vertical')

# Show the plot
plt.show()
```



Print a classification report to view precision, recall, and other metrics.

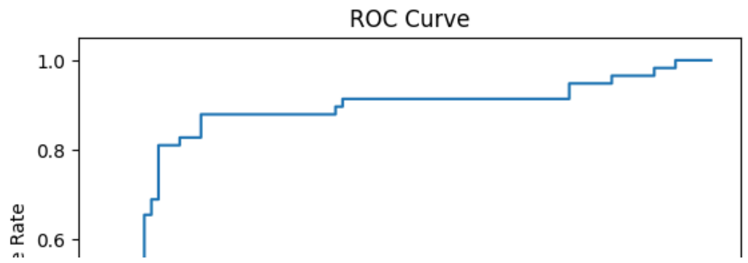
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.82      0.92      0.87        85
           1       0.85      0.71      0.77        58

    accuracy                           0.83       143
   macro avg       0.84      0.81      0.82       143
weighted avg       0.83      0.83      0.83       143
```

Finally, visualize the model's accuracy by plotting an ROC curve.

```
from sklearn.metrics import roc_curve, RocCurveDisplay
y_prob = model.predict_proba(x_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob)

# Create the ROC curve display
disp = RocCurveDisplay(fpr=fpr, tpr=tpr)
disp.plot()

# Add labels and title if desired
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are
```



Use the model to make predictions

Now use the model to predict whether a 30-year-old female traveling in first class will survive the voyage.

```python
female = [[30, 1, 0, 1, 0, 0]]
model.predict(female)[0]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LogisticRegressio
  warnings.warn(
1
```

More to the point, what is the probability that a 30-year-old female traveling in first class will survive?

```python
probability = model.predict_proba(female)[0][1]
print(f'Probability of survival: {probability:.1%}')
```

```
Probability of survival: 91.6%
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LogisticRegressio
  warnings.warn(
```

How about a 60-year-old male traveling in third class?

```python
male = [[60, 0, 1, 0, 0, 1]]
probability = model.predict_proba(male)[0][1]
print(f'Probability of survival: {probability:.1%}')
```

```
Start coding or generate with AI.
```