ML LAB MANUAL

FACULTY OF ENGINEERING AND TECHNOLOGY

BACHELOR OF TECHNOLOGY

Machine Learning (203105403)

6^{th} SEMESTER

COMPUTER SCIENCE ENGINEERING & TECHNOLOGY

CERTIFICATE

This is to certify that

Mr./Ms **Maru Anish J .** with enrolment no. **(2203031249006)** has successfully completed his/her laboratory experiments in the **MACHINE LEARNING (203105403)** from the department of **CSE-AI** during the academic year **2023-24.**

Date of Submission:.........................   Staff In charge:...........................

Head Of Department:..

TABLE OF CONTENT

Maru Anish J.
2203031249006

EXPERIMENT NO :- 1

AIM :- Use the Naive Bayes Classifier to differentiate between spam and non- spam (ham) messages.

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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df=pd.read_csv('spam.csv',encoding='latin')
df.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
df.v1.unique()
```

array(['ham', 'spam'], dtype=object)

```
df['spam']=df['v1'].apply(lambda x: 1 if x=='spam' else 0)
df.head(5)
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4	spam
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN	0
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN	1
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN	0
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN	0

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(df.v2,df.spam,test_size=0.2,random_state=42)

len(x_train)
```

4457

```
len(x_test)
```

1115

```
from sklearn.feature_extraction.text import CountVectorizer
v=CountVectorizer()
cv_messages = v.fit_transform(x_train.values)
cv_messages.toarray()[0:5]
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

```
from sklearn.naive_bayes import MultinomialNB
model=MultinomialNB()
```

```
model.fit(cv_messages,y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
email = [
         'Upto 30% discount on parking, exclusive offer just for yoy. Dont miss thi reward!',
         'Ok lar...joking wif u oni...'
]
email_count= v.transform(email)
model.predict(email_count)
```

array([1, 0])

```
x_test_count=v.transform(x_test)
model.score(x_test_count,y_test)
```

0.9838565022421525

```
from sklearn.pipeline import Pipeline
clf = Pipeline([
      ('vectorizer', CountVectorizer()),
      ('nb', MultinomialNB())
]
)
clf.fit(x_train,y_train)
```

```
email = [
        'Upto 30% discount on parking, exclusive offer just for yoy.
Dont miss thi reward!',
         'Ok lar...joking wif u oni...'
]
clf.predict(email)
```

array([1, 0])

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

0.9838565022421525

```
import joblib
joblib.dump(clf,'spam_model.pkl')
```

['spam_model.pkl']

```
# model is completed
```

Maru Anish J.
2203031249006

EXPERIMENT NO :- 2

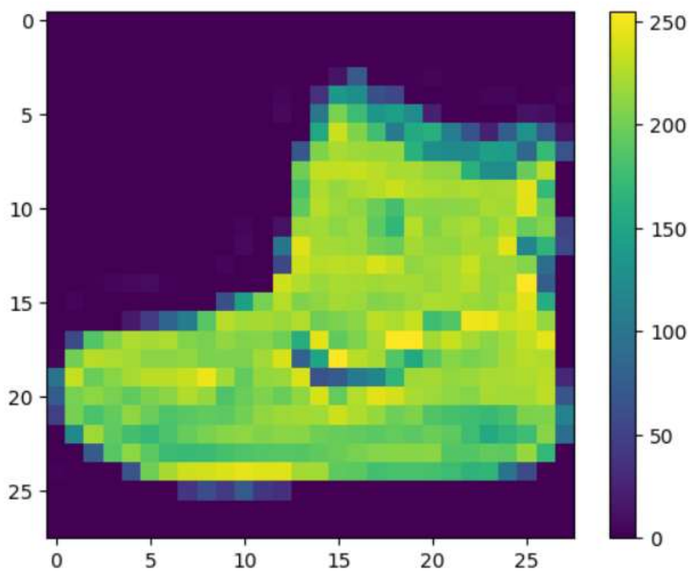
AIM :- Apply SVM to classify images of different fashion products.

```
# importing the necessary libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

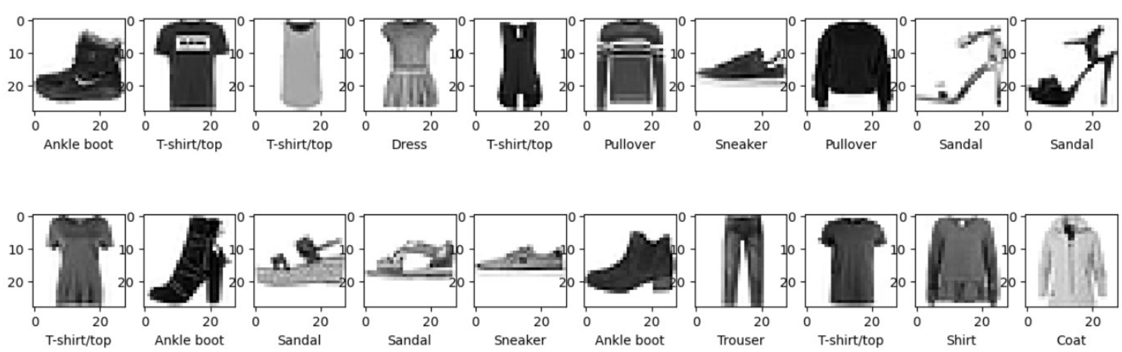
```
# storing the dataset path
clothing_fashion_mnist = tf.keras.datasets.fashion_mnist

# loading the dataset from tensorflow
(x_train, y_train),(x_test, y_test) =
clothing_fashion_mnist.load_data()

# displaying the shapes of training and testing dataset
print('Shape of training cloth images: ',x_train.shape)

print('Shape of training label: ',y_train.shape)

print('Shape of test cloth images: ',x_test.shape)

print('Shape of test labels: ',y_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
Shape of training cloth images:  (60000, 28, 28)
Shape of training label:  (60000,)
Shape of test cloth images:  (10000, 28, 28)
Shape of test labels:  (10000,)
```

```
# storing the class names as it is
# not provided in the dataset
label_class_names = ['T-shirt/top', 'Trouser',
          'Pullover', 'Dress', 'Coat',
          'Sandal', 'Shirt', 'Sneaker',
          'Bag', 'Ankle boot']

# display the first images
plt.imshow(x_train[0])
plt.colorbar() # to display the colourbar
plt.show()
```




```python
plt.figure(figsize=(15, 5)) # figure size
i = 0
while i < 20:
  plt.subplot(2, 10, i+1)

  # showing each image with colourmap as binary
  plt.imshow(x_train[i], cmap=plt.cm.binary)

  # giving class labels
  plt.xlabel(label_class_names[y_train[i]])
  i = i+1

plt.show() # plotting the final output figure
```



EXPERIMENT NO :- 3

AIM :- Build a linear regression model to predict house prices based on various features like size, location, year built.

```
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 LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Load the dataset from CSV
df = pd.read_csv('/content/house data.csv')
```

```
# Exploratory Data Analysis (EDA)
# Let's take a quick look at the first few rows of the dataset
print(df.head())
```

```
                  date      price  bedrooms  bathrooms  sqft_living  sqft_lot  \
0  2014-05-02 00:00:00   313000.0       3.0       1.50         1340      7912
1  2014-05-02 00:00:00  2384000.0       5.0       2.50         3650      9050
2  2014-05-02 00:00:00   342000.0       3.0       2.00         1930     11947
3  2014-05-02 00:00:00   420000.0       3.0       2.25         2000      8030
4  2014-05-02 00:00:00   550000.0       4.0       2.50         1940     10500

   floors  waterfront  view  condition  sqft_above  sqft_basement  yr_built  \
0     1.5           0     0          3        1340              0      1955
1     2.0           0     4          5        3370            280      1921
2     1.0           0     0          4        1930              0      1966
3     1.0           0     0          4        1000           1000      1963
4     1.0           0     0          4        1140            800      1976

   yr_renovated                    street       city  statezip country
0          2005      18810 Densmore Ave N  Shoreline  WA 98133     USA
1             0           709 W Blaine St    Seattle  WA 98119     USA
2             0  26206-26214 143rd Ave SE       Kent  WA 98042     USA
3             0           857 170th Pl NE   Bellevue  WA 98008     USA
4          1992         9105 170th Ave NE    Redmond  WA 98052     USA
```

```
# Summary statistics of the dataset
print(df.describe())
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot
count	4.600000e+03	4600.000000	4600.000000	4600.000000	4.600000e+03
mean	5.519630e+05	3.400870	2.160815	2139.346957	1.485252e+04
std	5.638347e+05	0.908848	0.783781	963.206916	3.588444e+04
min	0.000000e+00	0.000000	0.000000	370.000000	6.380000e+02
25%	3.228750e+05	3.000000	1.750000	1460.000000	5.000750e+03
50%	4.609435e+05	3.000000	2.250000	1980.000000	7.683000e+03
75%	6.549625e+05	4.000000	2.500000	2620.000000	1.100125e+04
max	2.659000e+07	9.000000	8.000000	13540.000000	1.074218e+06

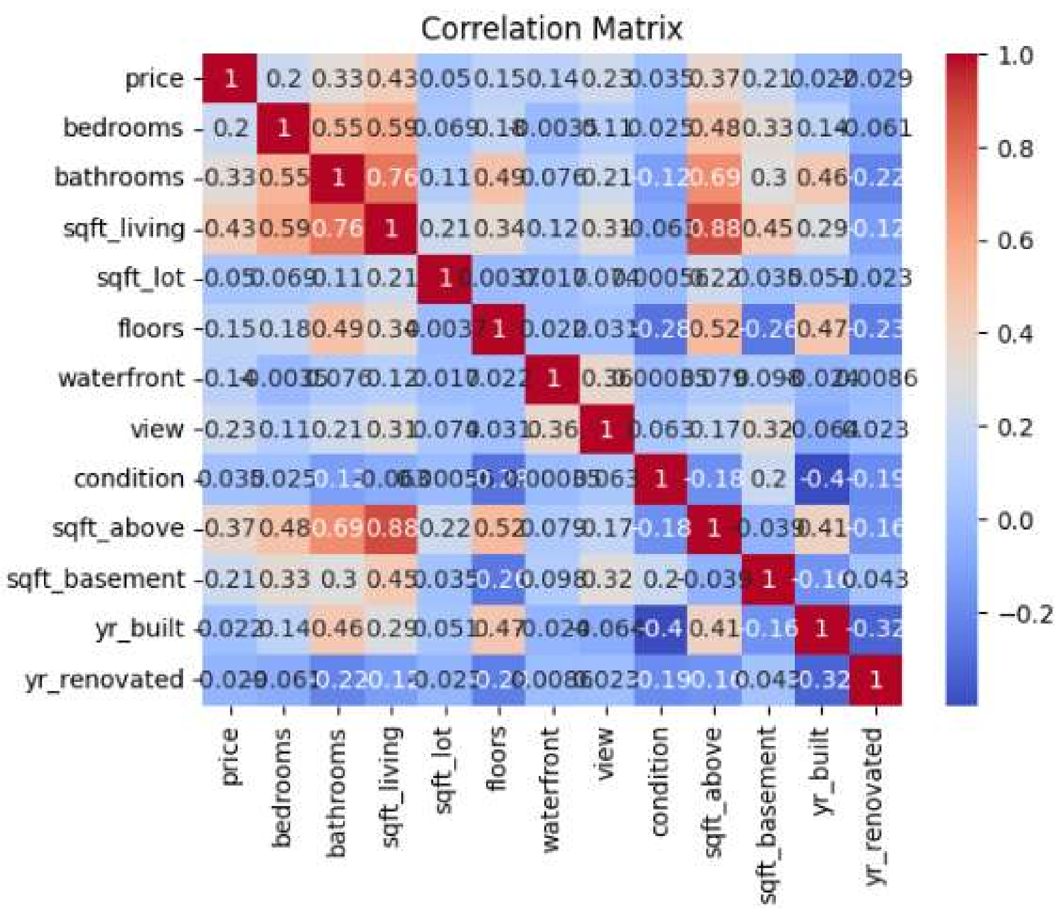
	floors	waterfront	view	condition	sqft_above
count	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000
mean	1.512065	0.007174	0.240652	3.451739	1827.265435
std	0.538288	0.084404	0.778405	0.677230	862.168977
min	1.000000	0.000000	0.000000	1.000000	370.000000
25%	1.000000	0.000000	0.000000	3.000000	1190.000000
50%	1.500000	0.000000	0.000000	3.000000	1590.000000
75%	2.000000	0.000000	0.000000	4.000000	2300.000000
max	3.500000	1.000000	4.000000	5.000000	9410.000000

	sqft_basement	yr_built	yr_renovated
count	4600.000000	4600.000000	4600.000000
mean	312.081522	1970.786304	808.608261
std	464.137228	29.731848	979.414536
min	0.000000	1900.000000	0.000000
25%	0.000000	1951.000000	0.000000
50%	0.000000	1976.000000	0.000000
75%	610.000000	1997.000000	1999.000000
max	4820.000000	2014.000000	2014.000000

```
# Check for missing values
print(df.isnull().sum())
```

```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

```
# Correlation matrix to understand feature relationships
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```




```
# Preprocessing: Selecting features and target variable
X = df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
'waterfront', 'view', 'condition']]
y = df['price']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
# Building the Linear Regression Model
model = LinearRegression()

# Fitting the model on the training data
model.fit(X_train, y_train)
```

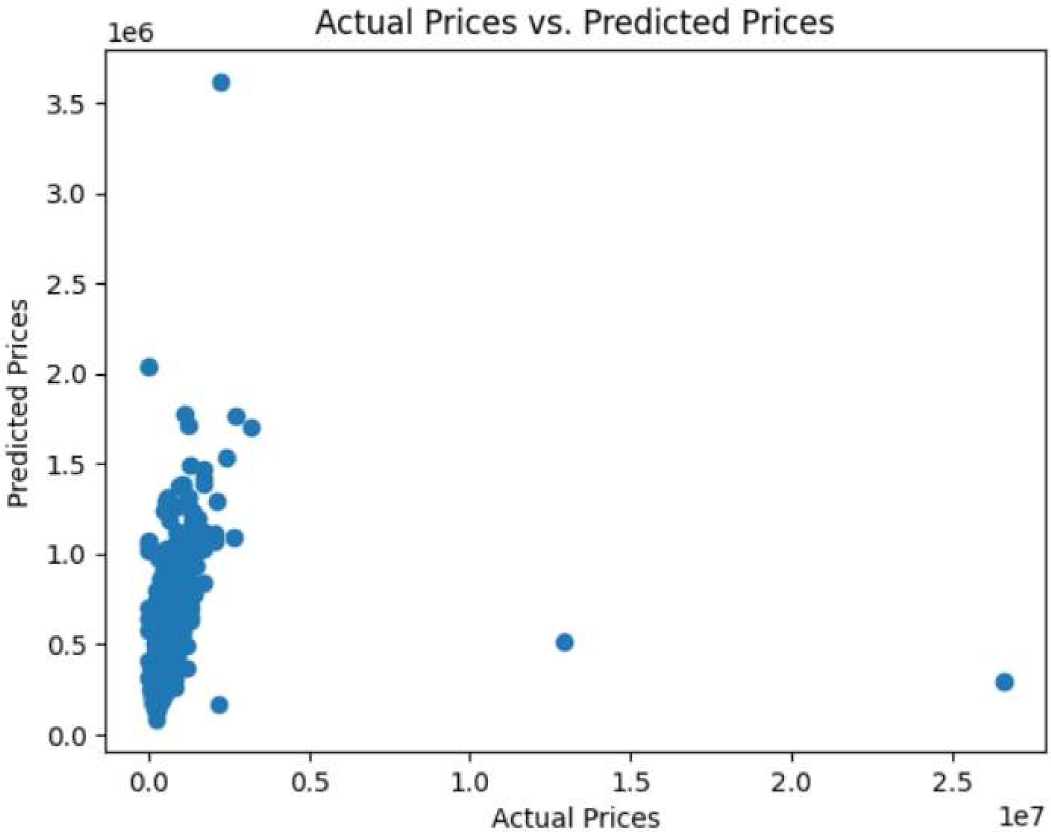
```
▾ LinearRegression
LinearRegression()
```

```
# Model Evaluation
y_pred = model.predict(X_test)

# Mean Squared Error and R-squared for model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 986869414953.98
R-squared: 0.03233518995632512
```

```
# Predictions and Visualization
# To visualize the predictions against actual prices, we'll use a
scatter plot
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs. Predicted Prices")
plt.show()


# Lastly, let's use the trained model to make predictions on new data
and visualize the results
new_data = [[3, 2, 1500, 4000, 1, 0, 0, 3]]
predicted_price = model.predict(new_data)

print("Predicted Price:", predicted_price[0])
```

Maru Anish J.
2203031249006




```
Predicted Price: 331038.9687692916
```

Maru Anish J.
2203031249006

EXPERIMENT NO :- 4

AIM :- Given the actual & predicted outputs for a classification problem, implement the performance metrics in python from scratch: 1) Accuracy 2)Precision 3) F-1 Score 4) ROC.

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, f1_score,
roc_curve, auc,confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load the California housing dataset
ch = fetch_california_housing()
data = pd.DataFrame(ch.data, columns=ch.feature_names)
data['Price'] = ch.target
```

```
data.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
data.head(15)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25	2.992
7	3.1200	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25	2.414
8	2.0804	42.0	4.294118	1.117647	1206.0	2.026891	37.84	-122.26	2.267
9	3.6912	52.0	4.970588	0.990196	1551.0	2.172269	37.84	-122.25	2.611
10	3.2031	52.0	5.477612	1.079602	910.0	2.263682	37.85	-122.26	2.815
11	3.2705	52.0	4.772480	1.024523	1504.0	2.049046	37.85	-122.26	2.418
12	3.0750	52.0	5.322650	1.012821	1098.0	2.346154	37.85	-122.26	2.135
13	2.6736	52.0	4.000000	1.097701	345.0	1.982759	37.84	-122.26	1.913
14	1.9167	52.0	4.262903	1.009677	1212.0	1.954839	37.85	-122.26	1.592

```
data.head(487)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price	Target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526	1
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585	1
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521	1
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413	1
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422	1
...	...	...	...	...	...	...	...	...	...	...
482	3.1500	52.0	4.765714	1.062857	494.0	2.822857	37.86	-122.27	2.063	1
483	2.6914	52.0	4.846575	1.019178	849.0	2.326027	37.86	-122.27	2.188	1
484	1.8447	52.0	4.209854	1.063869	1127.0	2.056569	37.86	-122.27	1.982	0
485	1.6307	35.0	2.962687	1.001148	3276.0	1.880597	37.86	-122.26	2.536	1
486	2.9044	52.0	4.404282	1.047859	1493.0	1.880353	37.86	-122.26	2.574	1

487 rows × 10 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   Price       20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

```
data.shape
```

(20640, 9)

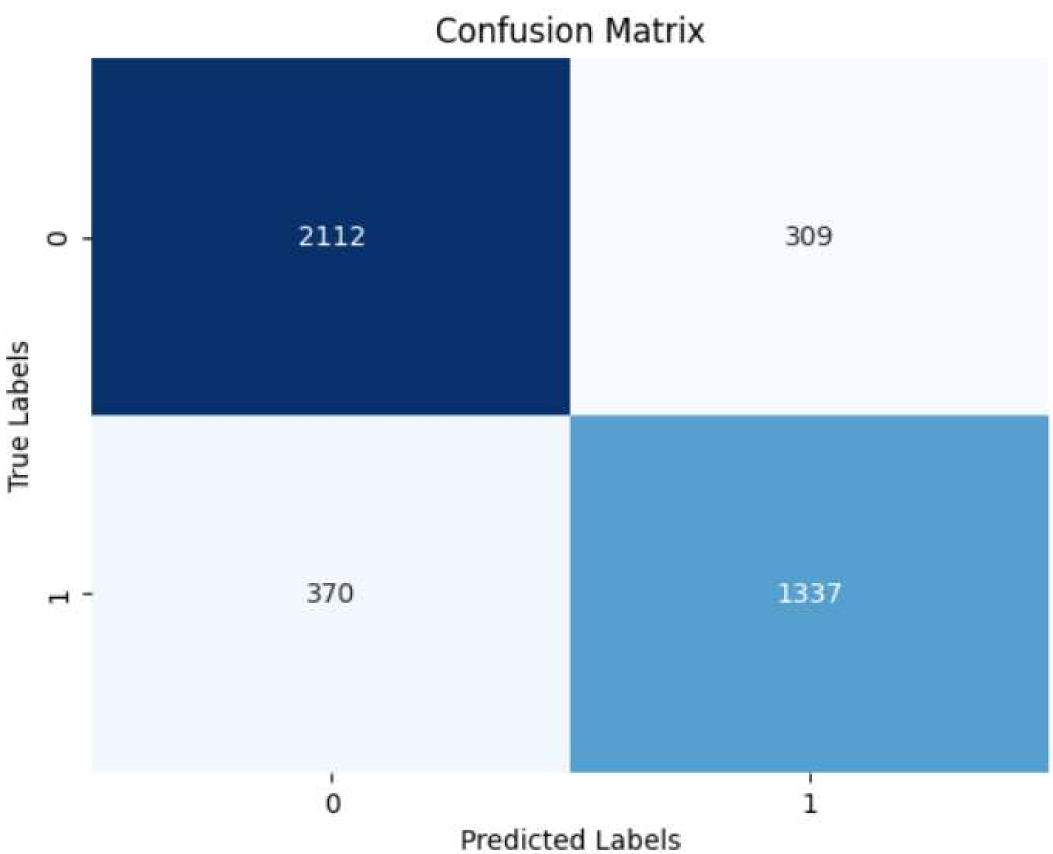
```
data.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35.631861	-119.569704	2.068558
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532	1.153956
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000	0.149990
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000	1.196000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000	1.797000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000	2.647250
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000	5.000010

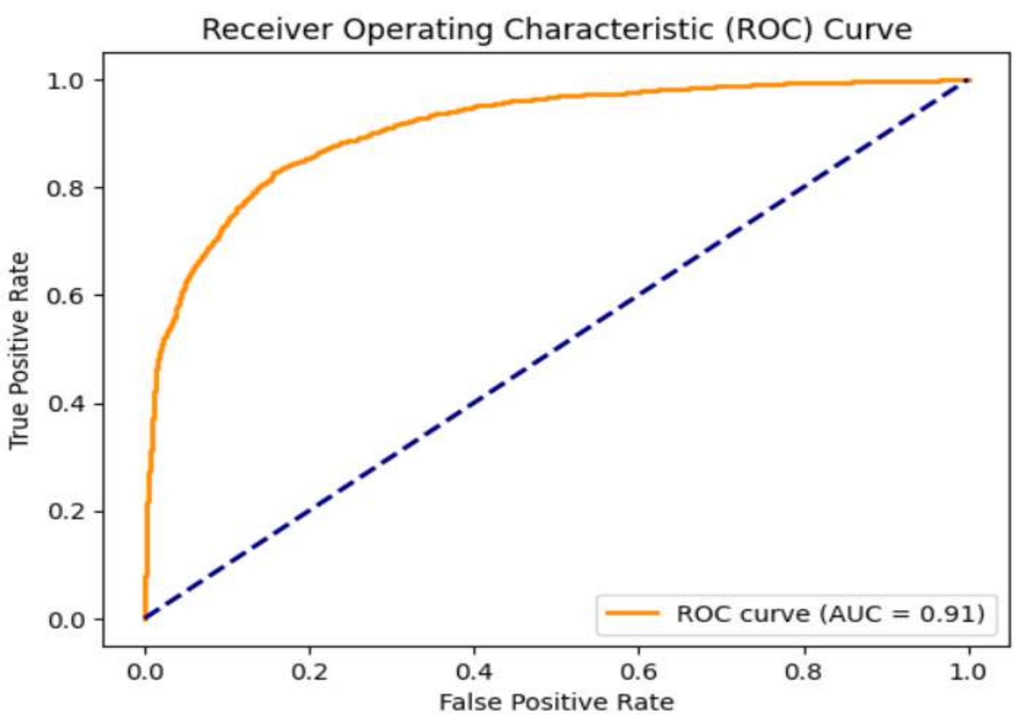
```
# Convert the regression problem into a binary classification problem
data['Target'] = (data['Price'] > 2.0).astype(int)   # Binary
classification based on a threshold

x = data.drop(['Price', 'Target'], axis=1)
y = data['Target']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
xt_scaled = scaler.fit_transform(x_train)
xte_scaled = scaler.transform(x_test)
```

```
# Train a logistic regression model
mdl = LogisticRegression()
mdl.fit(xt_scaled, y_train)

# Predictions
y_pred_proba = mdl.predict_proba(xte_scaled)[:, 1]
y_pred = (y_pred_proba > 0.5).astype(int)
```

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix using seaborn heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



```python
# Calculate classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()

# Print metrics
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'F-1 Score: {f1:.2f}')
```




```
Accuracy: 0.84
Precision: 0.81
F-1 Score: 0.80
```

EXPERIMENT NO :- 5

AIM :- Use logistic regression to identify fraudulent credit card transactions.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
df=pd.read_csv('/content/creditcard.csv')
```

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0.0
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0.0
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0.0
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0.0
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0.0

5 rows × 31 columns

```
df.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    1
Class     1
dtype: int64
```

```
df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
count	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	...	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	13954.000000	13953.000000	13953.000000
mean	10121.161817	-0.235676	0.284287	0.846047	0.302775	-0.118730	0.128013	-0.157297	-0.016148	0.989827	...	-0.068037	-0.157471	-0.035585	0.011158	0.117019	0.035544	0.012787	0.002122	63.882442	0.004013
std	7739.625811	1.720315	1.394804	1.561376	1.500595	1.289494	1.320395	1.238583	1.262024	1.214044	...	0.873942	0.621845	0.499807	0.587225	0.427953	0.558290	0.399550	0.255710	177.887021	0.063227
min	0.000000	-27.670569	-34.607649	-24.667741	-4.657545	-32.092129	-23.496714	-26.548144	-23.632502	-7.175097	...	-11.468435	-8.593642	-19.254328	-2.512377	-4.781606	-1.338556	-7.976100	-3.575312	0.000000	0.000000
25%	2984.250000	-0.969786	-0.282728	0.407297	-0.623141	-0.717155	-0.624025	-0.616307	-0.182270	0.288101	...	-0.271778	-0.549723	-0.173807	-0.339656	-0.135887	-0.374596	-0.076862	-0.014869	5.490000	0.000000
50%	9086.500000	-0.319439	0.252904	0.960978	0.220104	-0.191627	-0.144198	-0.111960	0.016945	0.971414	...	-0.132304	-0.122777	-0.045041	0.060208	0.155162	-0.035825	-0.000950	0.016238	15.690000	0.000000
75%	17103.500000	1.162662	0.884741	1.607031	1.198942	0.351255	0.508494	0.421830	0.265736	1.654184	...	0.018677	0.228997	0.069750	0.392734	0.393032	0.375271	0.098516	0.072497	52.150000	0.000000
max	24759.000000	1.960497	10.558600	4.101716	11.927512	34.099309	21.393069	34.303177	10.535558	10.392889	...	22.614889	4.534454	13.876221	3.200201	5.525093	3.517346	8.254376	4.860769	7712.430000	1.000000

8 rows × 31 columns

```
df_fraud = df[df['Class'] == 1]
number_fraud = len(df[df.Class == 1])
number_no_fraud = len(df[df.Class == 0])
```

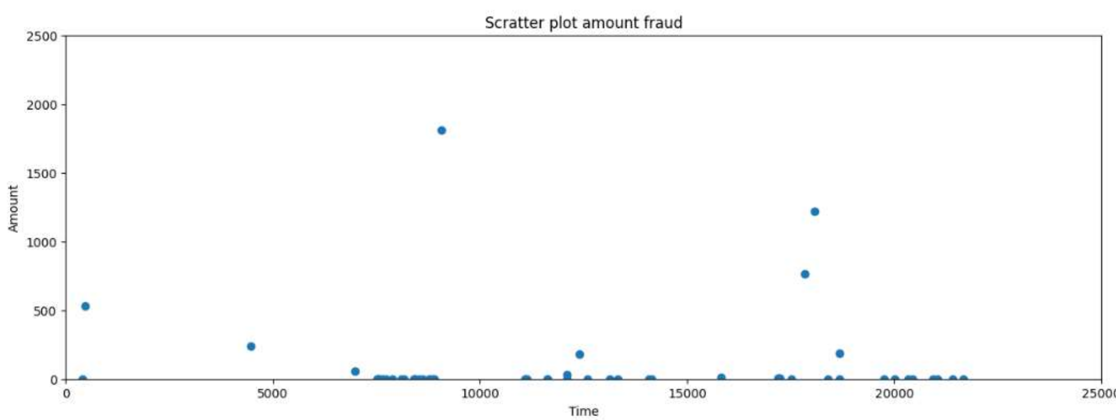
```
number_fraud
```

56

```
number_no_fraud
```

13897

```
plt.figure(figsize=(15,5))
plt.scatter(df_fraud['Time'], df_fraud['Amount'])
plt.title('Scratter plot amount fraud')
plt.xlabel('Time')
plt.ylabel('Amount')
plt.xlim([0,25000])
plt.ylim([0,2500])
plt.show()
```




```
df['V23'].fillna(value=df['V23'].mode()[0],inplace=True)
df['V24'].fillna(value=df['V24'].mode()[0],inplace=True)
df['V25'].fillna(value=df['V25'].mode()[0],inplace=True)
df['V26'].fillna(value=df['V26'].mode()[0],inplace=True)
df['V27'].fillna(value=df['V27'].mode()[0],inplace=True)
df['V28'].fillna(value=df['V28'].mode()[0],inplace=True)
df['Amount'].fillna(value=df['Amount'].mode()[0],inplace=True)
```

```python
df['Class'].fillna(value=df['Class'].mode()[0],inplace=True)

X=df.iloc[:,:-1]
y=df['Class']
# Assuming X contains features and y contains labels (fraud or not
fraud)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Instantiate logistic regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```python
# Predictions on the test set
y_pred = model.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[2779    2]
 [   4    6]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2781
         1.0       0.75      0.60      0.67        10

    accuracy                           1.00      2791
   macro avg       0.87      0.80      0.83      2791
weighted avg       1.00      1.00      1.00      2791
```

```
from sklearn.model_selection import GridSearchCV

# Define hyperparameters to tune
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

# Instantiate logistic regression model
model = LogisticRegression()

# Grid search for hyperparameter tuning
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best hyperparameters
best_params = grid_search.best_params_
print(f"Best Hyperparameters: {best_params}")

# Evaluate the model with the best hyperparameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Best Hyperparameters: {'C': 1}
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2781
         1.0       0.75      0.60      0.67        10

    accuracy                           1.00      2791
   macro avg       0.87      0.80      0.83      2791
weighted avg       1.00      1.00      1.00      2791
```

Maru Anish J.
2203031249006

EXPERIMENT NO :- 6

AIM :- Apply XGBoost and compare the performance for the loan default estimation.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
```

```
pip install xgboost scikit-learn
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
```

```
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.model_selection import train_test_split
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df=pd.read_csv('/content/drive/MyDrive/credit_risk_dataset.csv')
df.head()
```

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_default_on_file	cb_person_cred_hist_length
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	1	0.59	Y	3
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0	0.10	N	2
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1	0.57	N	3
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	1	0.53	N	2
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	1	0.55	Y	4

```
df.tail()
```

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_default_on_file	cb_person_cred_hist_length
32576	57	53000	MORTGAGE	1.0	PERSONAL	C	5800	13.16	0	0.11	N	30
32577	54	120000	MORTGAGE	4.0	PERSONAL	A	17625	7.49	0	0.15	N	19
32578	65	76000	RENT	3.0	HOMEIMPROVEMENT	B	35000	10.99	1	0.46	N	28
32579	56	150000	MORTGAGE	5.0	PERSONAL	B	15000	11.48	0	0.10	N	26
32580	66	42000	RENT	2.0	MEDICAL	B	6475	9.99	0	0.15	N	30

```
df.describe()
```

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_cred_hist_length
count	32581.000000	3.258100e+04	31686.000000	32581.000000	29465.000000	32581.000000	32581.000000	32581.000000
mean	27.734600	6.607485e+04	4.789686	9589.371106	11.011695	0.218164	0.170203	5.804211
std	6.348078	6.198312e+04	4.142630	6322.086646	3.240459	0.413006	0.106782	4.055001
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	0.000000	0.000000	2.000000
25%	23.000000	3.850000e+04	2.000000	5000.000000	7.900000	0.000000	0.090000	3.000000
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.990000	0.000000	0.150000	4.000000
75%	30.000000	7.920000e+04	7.000000	12200.000000	13.470000	0.000000	0.230000	8.000000
max	144.000000	6.000000e+06	123.000000	35000.000000	23.220000	1.000000	0.830000	30.000000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   person_age                  32581 non-null  int64
 1   person_income               32581 non-null  int64
 2   person_home_ownership       32581 non-null  object
 3   person_emp_length           31686 non-null  float64
 4   loan_intent                 32581 non-null  object
 5   loan_grade                  32581 non-null  object
 6   loan_amnt                   32581 non-null  int64
 7   loan_int_rate               29465 non-null  float64
 8   loan_status                 32581 non-null  int64
 9   loan_percent_income         32581 non-null  float64
 10  cb_person_default_on_file   32581 non-null  object
 11  cb_person_cred_hist_length  32581 non-null  int64
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

```
df.shape
```

```
(32581, 12)
```

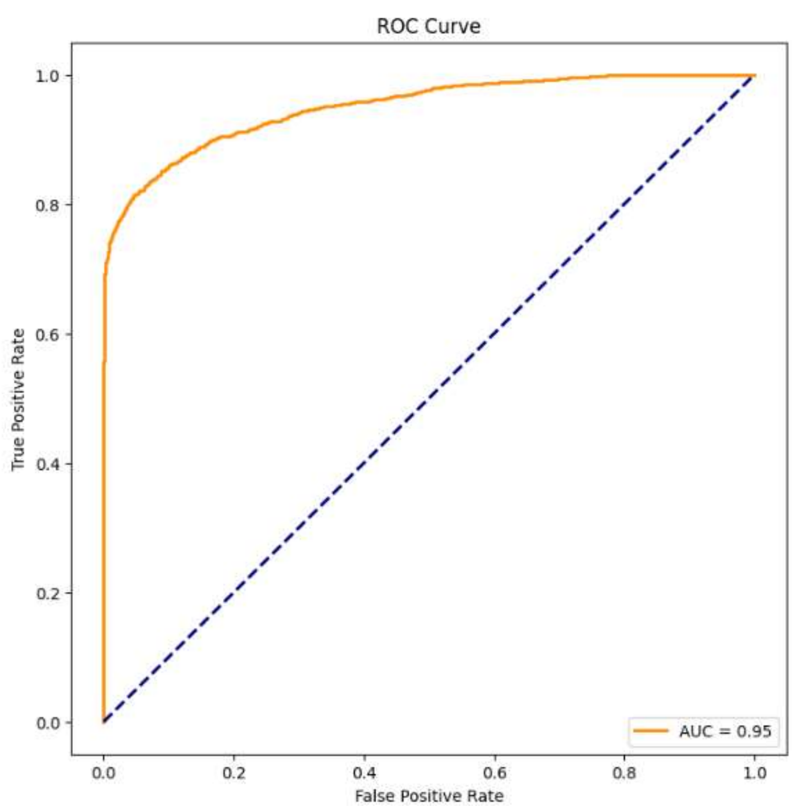
```
from sklearn.preprocessing import OneHotEncoder

# Assuming your dataset is stored in a DataFrame named 'df'
X = df[['person_age', 'person_income', 'person_home_ownership',
'person_emp_length',
        'loan_intent', 'loan_grade', 'loan_amnt', 'loan_int_rate',
        'loan_percent_income', 'cb_person_default_on_file',
'cb_person_cred_hist_length']]

y = df['loan_status']

# One-hot encode categorical variables
X_encoded = pd.get_dummies(X, columns=['person_home_ownership',
'loan_intent', 'loan_grade', 'cb_person_default_on_file'])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y,
test_size=0.2, random_state=42)

# Initialize and train the XGBoost classifier
```

```
model = xgb.XGBClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix:\n{confusion_mat}')
print(f'Classification Report:\n{classification_rep}')
```

```
Accuracy: 0.9349393892895504
Confusion Matrix:
[[5018   54]
 [ 370 1075]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      5072
           1       0.95      0.74      0.84      1445

    accuracy                           0.93      6517
   macro avg       0.94      0.87      0.90      6517
weighted avg       0.94      0.93      0.93      6517
```

```
from sklearn.metrics import roc_curve, roc_auc_score


# Assuming you already have X_train, X_test, y_train, y_test, and the
trained XGBoost model

# Make predictions on the test set
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC score
auc_score = roc_auc_score(y_test, y_pred_proba)

# Plot the ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC =
{auc_score:.2f}')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
```

```
plt.legend(loc='lower right')
plt.show()
from sklearn.model_selection import GridSearchCV

param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'n_estimators': [50, 100, 200]
}

grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=3)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print(f'Best Hyperparameters: {best_params}')

# Use the best model for predictions
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
```




```
# Let's create a new example for prediction
new_example = pd.DataFrame({
    'person_age': [28],
    'person_income': [60000],
```

```
    'person_home_ownership': ['OWN'],
    'person_emp_length': [3.0],
    'loan_intent': ['PERSONAL'],
    'loan_grade': ['B'],
    'loan_amnt': [12000],
    'loan_int_rate': [10.5],
    'loan_percent_income': [0.2],
    'cb_person_default_on_file': ['N'],
    'cb_person_cred_hist_length': [5]
})

# One-hot encode categorical variables
new_example_encoded = pd.get_dummies(new_example,
columns=['person_home_ownership', 'loan_intent', 'loan_grade',
'cb_person_default_on_file'])

# Ensure new_example_encoded has all the necessary columns
missing_columns = set(X_train.columns) -
set(new_example_encoded.columns)
for col in missing_columns:
    new_example_encoded[col] = 0

# Reorder the columns to match the order during training
new_example_encoded = new_example_encoded[X_train.columns]

# Make predictions
new_example_pred_proba = model.predict_proba(new_example_encoded)[:, 1]

# Print the predicted probability
print(f'Predicted Probability of Default:
{new_example_pred_proba[0]:.4f}')
```

```
Predicted Probability of Default: 0.0000
```

Maru Anish J.
2203031249006

EXPERIMENT NO :- 7

AIM :- Predict the onset of diabetes based on diagnostic measures using Logistic Regression.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df=pd.read_csv('/content/drive/MyDrive/diabetes.csv')
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.describe()
```

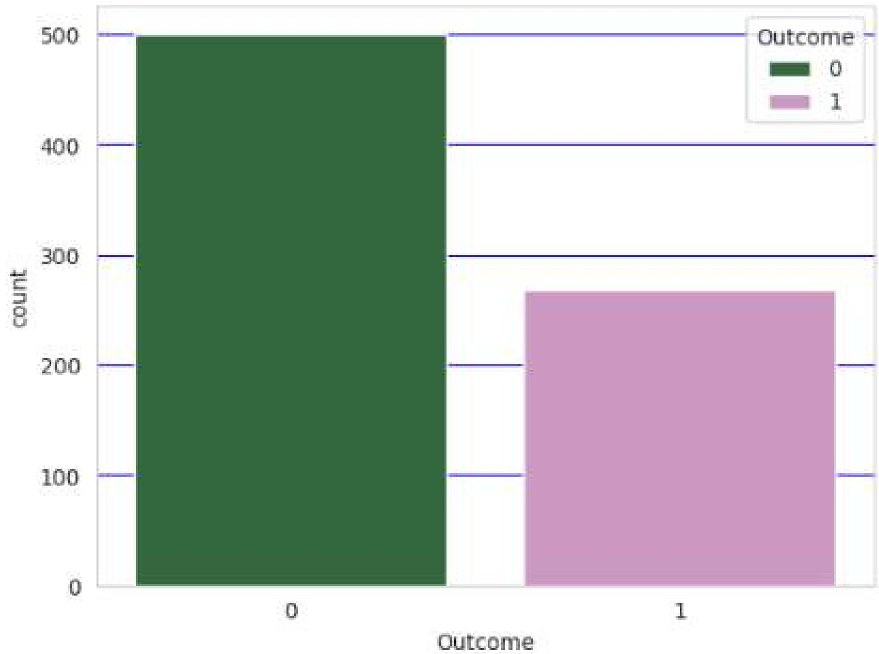
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
df.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
sns.set_style('whitegrid', {'grid.color': 'Blue'})
sns.countplot(x='Outcome',hue='Outcome',data=df,palette='cubehelix')
```

<Axes: xlabel='Outcome', ylabel='count'>




```
from sklearn.model_selection import train_test_split
x=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','B
MI','DiabetesPedigreeFunction','Age']
y=['Output']
X_train,X_test,y_train,y_test=train_test_split(df.drop('Outcome',axis=1
),df['Outcome'],test_size=0.20,random_state=101)
X_test.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
766	1	126	60	0	0	30.1	0.349	47
748	3	187	70	22	200	36.4	0.408	36
42	7	106	92	18	0	22.7	0.235	48
485	0	135	68	42	250	42.3	0.365	24
543	4	84	90	23	56	39.5	0.159	25

```
from sklearn.linear_model import LogisticRegression
LRModel=LogisticRegression(solver='lbfgs', max_iter=7600)
LRModel.fit(X_train,y_train)
```

▾ LogisticRegression

LogisticRegression(max_iter=7600)

```
predictions_diabetes=LRModel.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test,predictions_diabetes))
```

```
              precision    recall  f1-score   support

           0       0.83      0.86      0.85       103
           1       0.70      0.65      0.67        51

    accuracy                           0.79       154
   macro avg       0.77      0.76      0.76       154
weighted avg       0.79      0.79      0.79       154
```

```
# paitentid_54=pd.DataFrame([1,123,126,60,0,30.1,0.349,47],columns=x)
#Defining a sample data to test the model
x=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']
data=[0,170,126,60,35,30.1,0.649,78]
paitentid_54=pd.DataFrame([data],columns=x)
paitentid_54.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0	170	126	60	35	30.1	0.649	78

```
predictions_diabetes=LRModel.predict(paitentid_54)
```

```
print(predictions_diabetes)
```