

## Assignment -2

Team ID	PNT2022TMID15174
Student Name	Manojkirubakaran
Student Roll Number	7179KCTKCTKCTKCTKCT19BEC208
Maximum Marks	2 Marks

# 1. Download the dataset: Dataset

Dataset Successfully download and uploaded in colab.

## 2 LOADING DATASET.

```
import numpy as np
import pandas as pd
```

```
ds=pd.read_csv("/content/Churn_Modelling.csv")
ds.head()
```

### Output

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

## 3. Perform Below Visualizations.

- Univariate Analysis • Bi - Variate Analysis • Multi - Variate Analysis

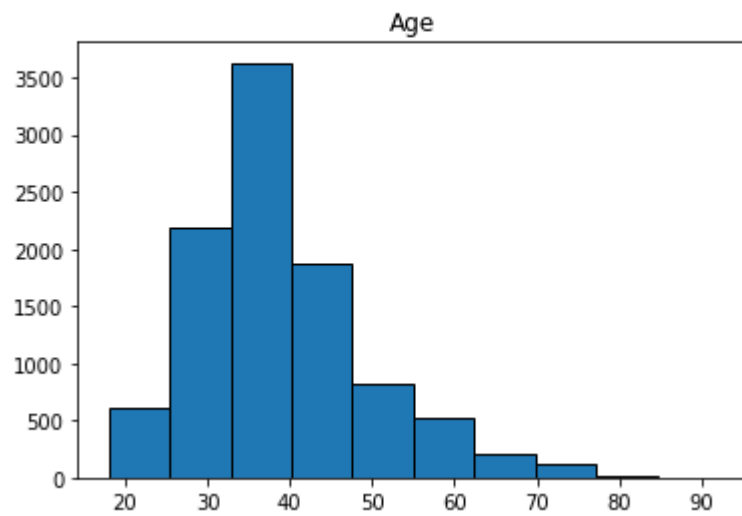
```
import matplotlib.pyplot as plt
import seaborn as sns
```

- Univariate Analysis

```
ds.hist(column='Age',grid=False,edgecolor='black')
```

## Output

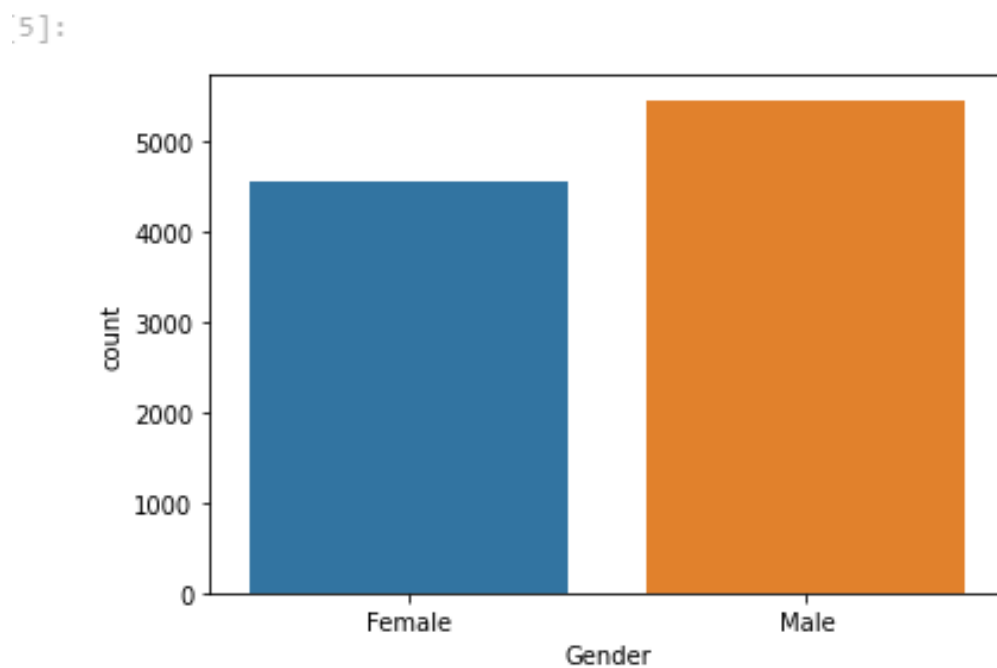
```
array([[ ]],  
      dtype=object)
```



- Bi - Variate Analysis

## Output

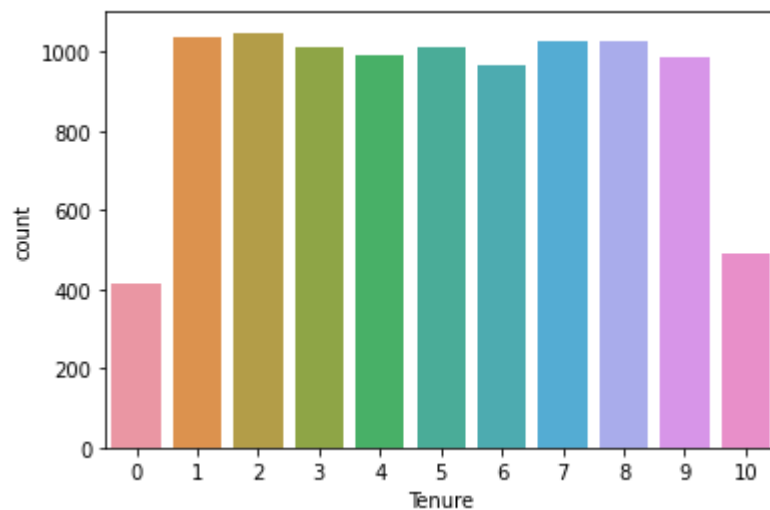
```
[5]: sns.countplot(x='Gender',data=ds)
```



- Multi - Variate Analysis

```
sns.countplot(x="Tenure",data=ds)
```

### Output



## 4. Perform descriptive statistics on the dataset.

```
ds.describe()
```

### Output

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

## 5. Handle the Missing values.

```
ds.info()
```

```
RangeIndex: 10000 entries, 0 to 9999
```

```
Data columns (total 14 columns):
```

#	Column	Non-Null Count	Dtype
0	RowNumber	10000 non-null	int64
1	CustomerId	10000 non-null	int64
2	Surname	10000 non-null	object
3	CreditScore	10000 non-null	int64
4	Geography	10000 non-null	object
5	Gender	10000 non-null	object
6	Age	10000 non-null	int64
7	Tenure	10000 non-null	int64
8	Balance	10000 non-null	float64
9	NumOfProducts	10000 non-null	int64
10	HasCrCard	10000 non-null	int64
11	IsActiveMember	10000 non-null	int64
12	EstimatedSalary	10000 non-null	float64
13	Exited	10000 non-null	int64

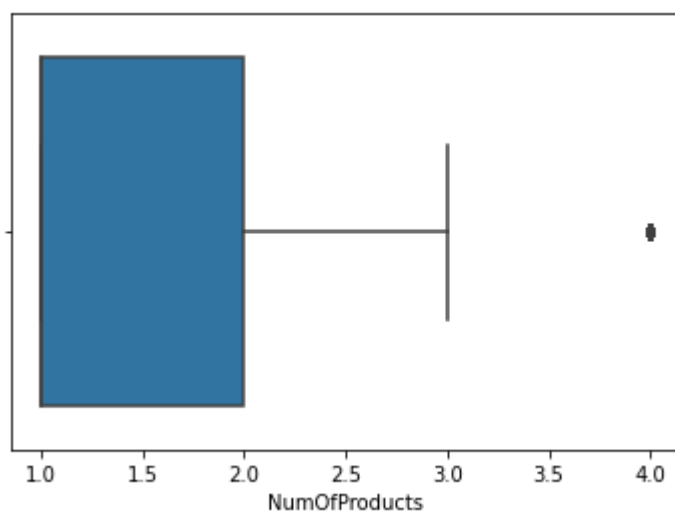
```
dtypes: float64(2), int64(9), object(3)
```

```
memory usage: 1.1+ MB
```

## 6. Find the outliers and replace the outliers

```
sns.boxplot(ds['NumOfProducts'])
```

### Output



```
# Values above 3 are outliers
outliers=np.where(ds['NumOfProducts']>3)
outliers
```

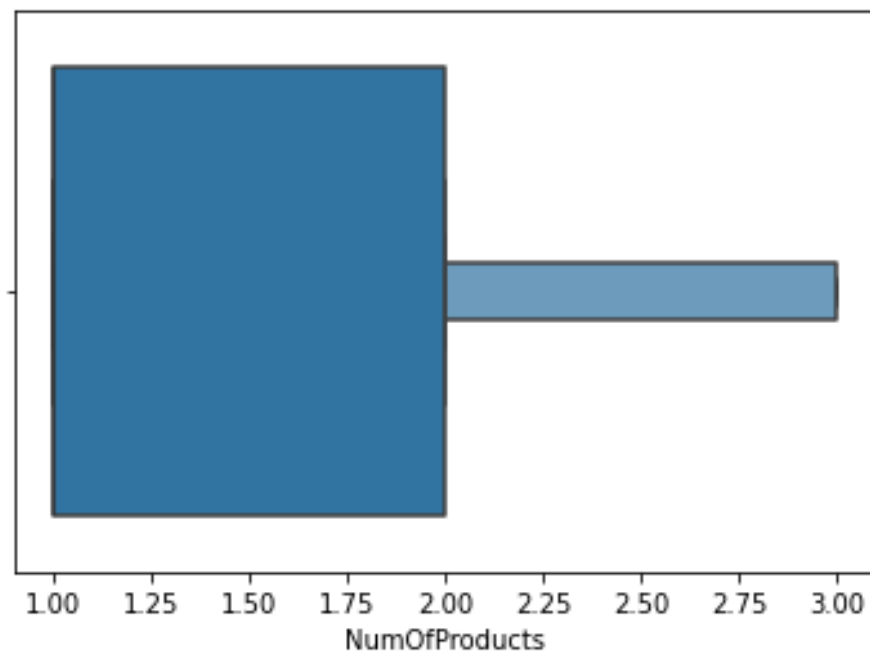
## Output

```
(array([ 7, 70, 1254, 1469, 1488, 1701, 1876, 2124, 2196, 2285, 2462,
        2499, 2509, 2541, 2614, 2617, 2872, 3152, 3365, 3841, 4013, 4014,
        4166, 4260, 4403, 4511, 4516, 4606, 4654, 4748, 4822, 5010, 5137,
        5235, 5386, 5700, 5904, 6150, 6172, 6279, 6750, 6875, 7257, 7457,
        7567, 7698, 7724, 7729, 8041, 8590, 8683, 8850, 8923, 9215, 9255,
        9323, 9370, 9411, 9540, 9565]),)
```

```
for i in outliers:
    ds['NumOfProducts'][i]=3 #replacing with previous value
```

```
sns.boxenplot(ds['NumOfProducts'])
```

## Output



## 7. Check for Categorical columns and perform encoding.

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

ds["Surname"]=le.fit_transform(ds["Surname"])
ds['Geography']=le.fit_transform(ds['Geography'])
ds['Gender']=le.fit_transform(ds['Gender'])

ds.head()
```

### Output

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	1115	619	0	0	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	1177	608	2	0	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	2040	502	0	0	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	289	699	0	0	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	1822	850	2	0	43	2	125510.82	1	1	1	79084.10	0

## 8. Split the data into dependent and independent variables.

```
x = ds.iloc[:, 3:13]
y = ds.iloc[:, 13]
```

## 9. Scale the independent variables

```
x=x.drop(['Geography','Gender'],axis=1)  
x
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	42	2	0.00	1	1	1	101348.88
1	608	41	1	83807.86	1	0	1	112542.58
2	502	42	8	159660.80	3	1	0	113931.57
3	699	39	1	0.00	2	0	0	93826.63
4	850	43	2	125510.82	1	1	1	79084.10
...	...	...	...	...	...	...	...	...
9995	771	39	5	0.00	2	1	0	96270.64
9996	516	35	10	57369.61	1	1	1	101699.77
9997	709	36	7	0.00	1	0	1	42085.58
9998	772	42	3	75075.31	2	1	0	92888.52
9999	792	28	4	130142.79	1	1	0	38190.78

10000 rows × 8 columns

```
y
```

0	1
1	0
2	1
3	0
4	0
...	..
9995	0
9996	0
9997	1
9998	1
9999	0

Name: Exited, Length: 10000, dtype: int64

## 10. Split the data into training and testing

```
from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=0
)

xtrain.shape,xtest.shape

((7000, 8), (3000, 8))

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import StandardScaler

sc=StandardScaler()
xtrain=sc.fit_transform(xtrain)
xtest=sc.fit_transform(xtest)

classifier=Sequential()
classifier.add(Dense(8,activation='relu'))
classifier.add(Dense(16,activation='relu'))
classifier.add(Dense(24,activation='relu'))
classifier.add(Dense(8,activation='relu'))
classifier.add(Dense(1,activation='linear'))

:
classifier.compile(optimizer = 'adam', loss = 'mse', metrics = ['accuracy'])

classifier.fit(xtrain,ytrain,batch_size=10,epochs=100)
```

### Output

```
Epoch 1/100
700/700 [=====] - 2s 2ms/step - loss: 0.1511 - acc
uracy: 0.8004
Epoch 2/100
700/700 [=====] - 1s 2ms/step - loss: 0.1280 - acc
uracy: 0.8329
Epoch 3/100
700/700 [=====] - 1s 2ms/step - loss: 0.1164 - acc
uracy: 0.8480
Epoch 4/100
700/700 [=====] - 1s 2ms/step - loss: 0.1127 - acc
uracy: 0.8536
Epoch 5/100
```



700/700 [=====] - 1s 2ms/step - loss: 0.1111 - accuracy: 0.8537  
Epoch 6/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1100 - accuracy: 0.8550  
Epoch 7/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1091 - accuracy: 0.8559  
Epoch 8/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1089 - accuracy: 0.8550  
Epoch 9/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1088 - accuracy: 0.8560  
Epoch 10/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1086 - accuracy: 0.8583  
Epoch 11/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1078 - accuracy: 0.8574  
Epoch 12/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1073 - accuracy: 0.8587  
Epoch 13/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1068 - accuracy: 0.8596  
Epoch 14/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1067 - accuracy: 0.8569  
Epoch 15/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1067 - accuracy: 0.8584  
Epoch 16/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1070 - accuracy: 0.8584  
Epoch 17/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1062 - accuracy: 0.8574  
Epoch 18/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1062 - accuracy: 0.8597  
Epoch 19/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1056 - accuracy: 0.8597  
Epoch 20/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1055 - accuracy: 0.8594  
Epoch 21/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1052 - accuracy: 0.8589  
Epoch 22/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1056 - accuracy: 0.8579  
Epoch 23/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1052 - accuracy: 0.8594  
Epoch 24/100

700/700 [=====] - 1s 2ms/step - loss: 0.1049 - accuracy: 0.8599  
Epoch 25/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1049 - accuracy: 0.8609  
Epoch 26/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1049 - accuracy: 0.8604  
Epoch 27/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1045 - accuracy: 0.8586  
Epoch 28/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1041 - accuracy: 0.8619  
Epoch 29/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1045 - accuracy: 0.8607  
Epoch 30/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1039 - accuracy: 0.8609  
Epoch 31/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1038 - accuracy: 0.8616  
Epoch 32/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1038 - accuracy: 0.8646  
Epoch 33/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1040 - accuracy: 0.8614  
Epoch 34/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1035 - accuracy: 0.8617  
Epoch 35/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1041 - accuracy: 0.8604  
Epoch 36/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1034 - accuracy: 0.8627  
Epoch 37/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1035 - accuracy: 0.8614  
Epoch 38/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1030 - accuracy: 0.8631  
Epoch 39/100  
700/700 [=====] - 2s 2ms/step - loss: 0.1035 - accuracy: 0.8623  
Epoch 40/100  
700/700 [=====] - 2s 2ms/step - loss: 0.1031 - accuracy: 0.8634  
Epoch 41/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1027 - accuracy: 0.8627  
Epoch 42/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1027 - accuracy: 0.8641  
Epoch 43/100

700/700 [=====] - 1s 2ms/step - loss: 0.1030 - accuracy: 0.8623  
Epoch 44/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1030 - accuracy: 0.8626  
Epoch 45/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1020 - accuracy: 0.8634  
Epoch 46/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1026 - accuracy: 0.8636  
Epoch 47/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1027 - accuracy: 0.8651  
Epoch 48/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1022 - accuracy: 0.8641  
Epoch 49/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1022 - accuracy: 0.8634  
Epoch 50/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1016 - accuracy: 0.8667  
Epoch 51/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1019 - accuracy: 0.8649  
Epoch 52/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1020 - accuracy: 0.8657  
Epoch 53/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1018 - accuracy: 0.8653  
Epoch 54/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1015 - accuracy: 0.8667  
Epoch 55/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1017 - accuracy: 0.8661  
Epoch 56/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1016 - accuracy: 0.8641  
Epoch 57/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1012 - accuracy: 0.8664  
Epoch 58/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1009 - accuracy: 0.8654  
Epoch 59/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1008 - accuracy: 0.8649  
Epoch 60/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1009 - accuracy: 0.8654  
Epoch 61/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1007 - accuracy: 0.8651  
Epoch 62/100

700/700 [=====] - 1s 2ms/step - loss: 0.1006 - accuracy: 0.8676  
Epoch 63/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1010 - accuracy: 0.8660  
Epoch 64/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1003 - accuracy: 0.8694  
Epoch 65/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1009 - accuracy: 0.8657  
Epoch 66/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1004 - accuracy: 0.8677  
Epoch 67/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1003 - accuracy: 0.8663  
Epoch 68/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1007 - accuracy: 0.8667  
Epoch 69/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1004 - accuracy: 0.8661  
Epoch 70/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1001 - accuracy: 0.8673  
Epoch 71/100  
700/700 [=====] - 1s 2ms/step - loss: 0.1001 - accuracy: 0.8669  
Epoch 72/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0997 - accuracy: 0.8671  
Epoch 73/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0995 - accuracy: 0.8700  
Epoch 74/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0999 - accuracy: 0.8671  
Epoch 75/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0998 - accuracy: 0.8684  
Epoch 76/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0995 - accuracy: 0.8674  
Epoch 77/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0995 - accuracy: 0.8673  
Epoch 78/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0997 - accuracy: 0.8674  
Epoch 79/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0995 - accuracy: 0.8679  
Epoch 80/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0992 - accuracy: 0.8686  
Epoch 81/100

700/700 [=====] - 1s 2ms/step - loss: 0.0988 - accuracy: 0.8681  
Epoch 82/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0989 - accuracy: 0.8701  
Epoch 83/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0989 - accuracy: 0.8699  
Epoch 84/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0991 - accuracy: 0.8691  
Epoch 85/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0989 - accuracy: 0.8699  
Epoch 86/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0982 - accuracy: 0.8711  
Epoch 87/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0987 - accuracy: 0.8699  
Epoch 88/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0985 - accuracy: 0.8694  
Epoch 89/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0985 - accuracy: 0.8697  
Epoch 90/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0982 - accuracy: 0.8706  
Epoch 91/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0989 - accuracy: 0.8694  
Epoch 92/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0984 - accuracy: 0.8683  
Epoch 93/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0982 - accuracy: 0.8683  
Epoch 94/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0980 - accuracy: 0.8686  
Epoch 95/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0979 - accuracy: 0.8697  
Epoch 96/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0977 - accuracy: 0.8696  
Epoch 97/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0979 - accuracy: 0.8696  
Epoch 98/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0978 - accuracy: 0.8710  
Epoch 99/100  
700/700 [=====] - 1s 2ms/step - loss: 0.0977 - accuracy: 0.8697  
Epoch 100/100

```
700/700 [=====] - 1s 2ms/step - loss: 0.0980 - accuracy: 0.8694
```

```
ypred=classifier.predict(xtest)
```

```
from sklearn.metrics import r2_score
```

```
r2_score(ytest,ypred)*100
```

### **Output**

```
29.498513365970247
```