Case study Report

Objective: To predict if user will buy an item or not. (Classification Problem)

Programming Language: Python (Jupyter Notebook)

Name: Harshad Punghera

Submitted on: 02/09/2020

Dataset:

- Data consists of 2 files BuyAffinity_Train.txt & BuyAffinity_Test.txt.
- BuyAffinity_Train.txt converted to Data Frame using Pandas.

```
data = pd.read_csv(r'BuyAffinity_Train.txt',delim_whitespace=True)
data
```

Index	F1	F2	F3	F4	F5	F6	F7	F8	F9	...	F14	F15	F16	F17	F18	F19	F20	F21	F22	C
1	0.224506	0.500340	0.489860	0.902413	7934	-6970	-5714	9982	-5697	...	-3433637453	10/4/1986	9/6/1992	2	1	706	305	1	2	0
2	0.321128	0.281119	0.907283	0.772159	-8238	1219	1663	1287	-3658	...	609277486	2/24/1979	1/5/1983	1	1	423	206	18	7	1
3	0.893441	0.622005	0.998776	0.098386	8540	5266	-9377	-3504	-4511	...	-8977995005	1/12/1989	11/22/1986	2	1	703	315	1	4	0
4	0.320641	0.957234	0.346000	0.646479	-7772	-383	9681	-8661	3474	...	4868760308	2/18/1982	6/10/1992	1	1	122	304	15	1	0
5	0.475961	0.623008	0.544988	0.159709	1571	-8039	-7961	-2385	4407	...	9757408267	4/10/1987	10/19/1985	1	1	486	240	1	1	0
6	0.922726	0.600115	0.616261	0.339285	-6554	8770	1065	-9720	5801	...	-6662571037	6/28/1990	1/23/1998	4	1	806	157	6	5	0
7	0.858156	0.546053	0.066203	0.998563	-9455	-9937	4079	8178	-663	...	-7236244398	10/22/1989	10/29/1991	1	2	448	702	5	1	0
8	0.707213	0.302135	0.686451	0.747126	7089	2404	3157	5484	-2829	...	-6408783500	12/29/1984	4/1/1983	1	1	187	123	3	1	0
9	0.971173	0.715137	0.748127	0.783115	554	-3388	1279	4381	-8957	...	1802050857	10/14/1992	4/14/1992	2	1	701	34	5	1	0
10	0.133155	0.314039	0.919551	0.143820	8952	6923	3112	-7115	-1413	...	-9215966088	1/21/1988	5/30/1991	1	2	502	706	1	1	0
11	0.422358	0.881814	0.585969	0.722192	-2034	9904	7229	2207	3252	...	-7562842865	1/25/1993	2/15/1993	2	1	701	145	8	1	0
12	0.460761	0.547355	0.101228	0.863503	-8053	-6331	-4702	8602	9779	...	-1170915872	12/17/1984	1/10/1992	1	2	489	703	9	2	0
13	0.284544	0.769097	0.645292	0.247550	8523	-8575	8589	681	-505	...	8383637633	4/26/1995	11/20/1989	1	1	587	229	4	6	1

- It consists of 101180 rows * 24 columns

```
data.columns
```

```
Index(['Index', 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10',
       'F11', 'F12', 'F13', 'F14', 'F15', 'F16', 'F17', 'F18', 'F19', 'F20',
       'F21', 'F22', 'C'],
```

- Dependent Variable: 22
- Target Variable: 1 (C)

Missing Values/Null Values:

- Dataset contains no missing values.

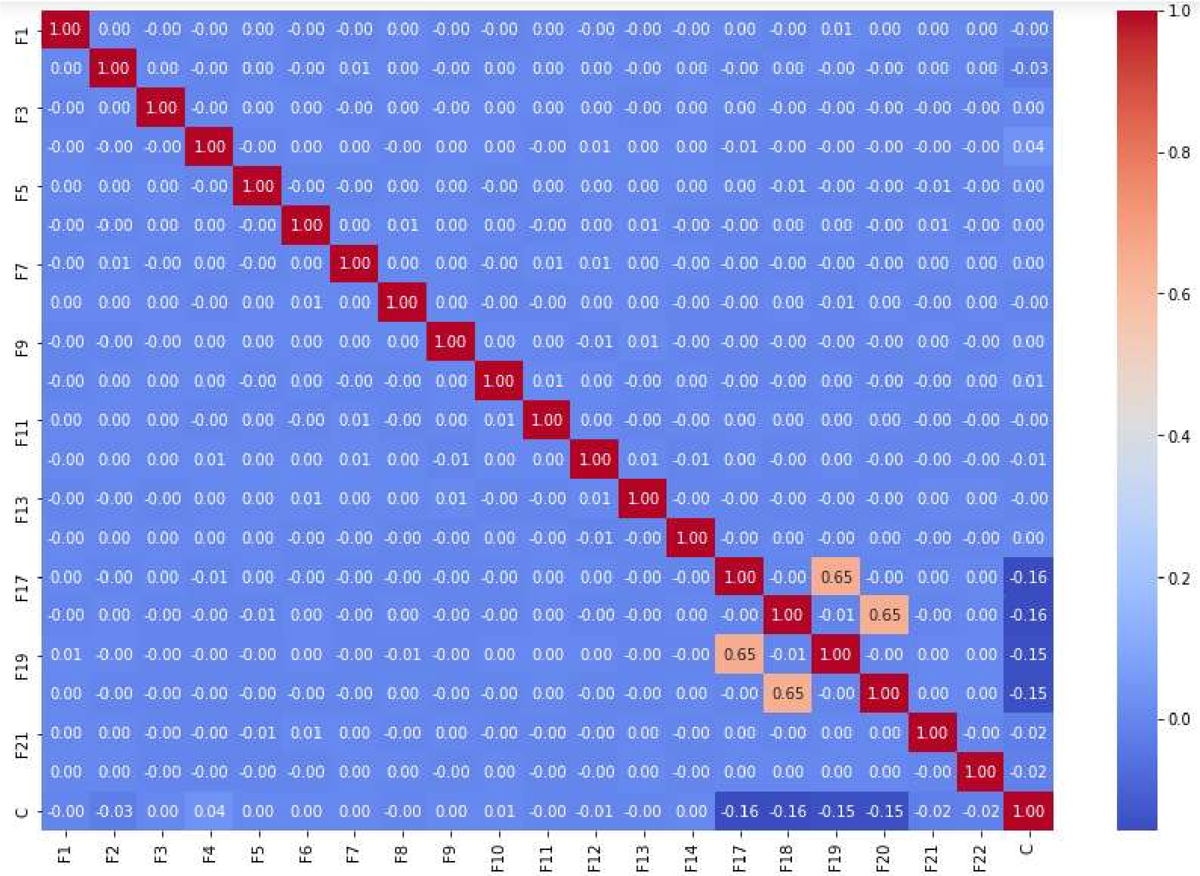
```
1  data_1.isnull().sum()    #total missing values in each column
```

```
F1      0
F2      0
F3      0
F4      0
F5      0
F6      0
F7      0
F8      0
F9      0
F10     0
F11     0
F12     0
F13     0
F14     0
F17     0
F18     0
F19     0
F20     0
F21     0
F22     0
C       0
dtype: int64
```

- Every column shows 0 missing values
- This is a clean dataset
- No need to work for any Data Imputation.

Co-relation Metrix:

- Heatmap using Seaborn and Matplotlib



- Very small amount of co-relation can be observed in F2, F4, F10, F12, F17, F18, F19, F20, F21 & F22.

Statistical Details:

```
1  data_1.describe()
```

	F1	F2	F3	F4	F5	F6
count	101180.000000	101180.000000	101180.000000	101180.000000	101180.000000	101180.000000
mean	0.502348	0.501497	0.499886	0.499839	-29.742617	1.511000
std	0.288058	0.289017	0.288875	0.288729	5781.829379	5796.594007
min	0.000018	0.000004	0.000002	0.000006	-10000.000000	-10000.000000
25%	0.253819	0.251115	0.248818	0.250501	-5045.000000	-5012.000000
50%	0.501802	0.501095	0.499820	0.501387	-46.000000	-11.500000
75%	0.753598	0.752404	0.750281	0.748803	4978.000000	5050.000000
max	0.999986	0.999990	0.999985	0.999977	10000.000000	10000.000000

Columns Dropped:

- 'Index' columns being all unique and F15, F16 being date columns can be dropped since they are not equidistance and cannot contribute much to the model.

In [5]:
```
data_1=data.drop(['Index','F15','F16'],axis=1)  #drop irrevalent columns
data_1
```

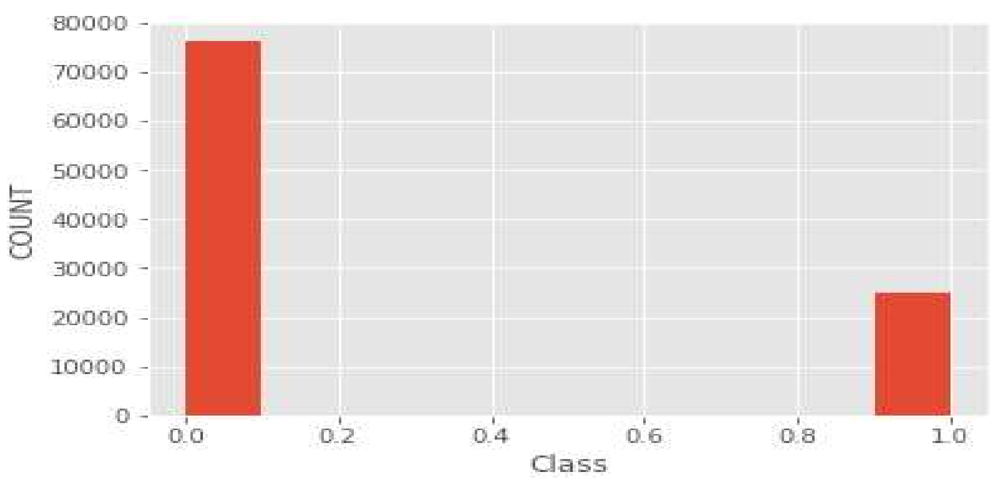
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F12	F13	F14	F17	F18	F19	F20	F21	F22	C
0	0.224506	0.500340	0.489860	0.902413	7934	-6970	-5714	9982	-5697	4227810299	...	316195953	6176861823	-3433637453	2	1	706	305	1	2	0
1	0.321128	0.281119	0.907283	0.772159	-8238	1219	1663	1287	-3658	-1146724819	...	1378635942	-9031507610	609277486	1	1	423	206	18	7	1
2	0.893441	0.622005	0.998776	0.098386	8540	5266	-9377	-3504	-4511	5947184989	...	-9921889287	-5610051842	-8977995005	2	1	703	315	1	4	0
3	0.320641	0.957234	0.346000	0.646479	-7772	-383	9681	-8661	3474	-5724795826	...	6550322883	-4697085930	4868760308	1	1	122	304	15	1	0
4	0.475961	0.623008	0.544988	0.159709	1571	-8039	-7961	-2385	4407	-3097637172	...	759031103	9984692447	9757408267	1	1	486	240	1	1	0
5	0.922726	0.600115	0.616261	0.339285	-6554	8770	1065	-9720	5801	6730646544	...	6027284059	5986948546	-6662571037	4	1	806	157	6	5	0
6	0.858156	0.546053	0.066203	0.998563	-9455	-9937	4079	8178	-663	7329897533	...	6276436738	-3715357603	-7236244398	1	2	448	702	5	1	0
7	0.707213	0.302135	0.686451	0.747126	7089	2404	3157	5484	-2829	5675038456	...	-3044219552	4876942469	-6408783500	1	1	187	123	3	1	0
8	0.971173	0.715137	0.748127	0.783115	554	-3388	1279	4381	-8957	-9988371423	...	-8488127003	561162925	1802050857	2	1	701	34	5	1	0

Class Imbalance:

- Since it is a Classification problem one of the most crucial step is to check for imbalance data i.e. the ratio of 0's and 1's.

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
data_1['C'].hist()
plt.xlabel('Class')
plt.ylabel('COUNT')
```

```
Text(0, 0.5, 'COUNT')
```



- Heavy Class Imbalance and be clearly seen using Bar Graph.
- 0 - 76353 & 1 - 24827
- This can lead to a model whose results are Bias toward class 0 since model will train more on Class 0 instance.
- To overcome this problem, we can apply Oversampling or Undersampling Technique.

Scaling:

- As the columns are present in the different units, we need to scaled down those features in order to achieve the better accuracy.

- Using sklearn StandardScaler.

```
from sklearn.preprocessing import StandardScaler
standardScaler=StandardScaler()
columns_to_scale=['F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10', 'F11',
       'F12', 'F13', 'F14', 'F17', 'F18', 'F19', 'F20', 'F21', 'F22']
data[columns_to_scale]=standardScaler.fit_transform(data[columns_to_scale])
```

```
data.head()
```

F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F12
-0.964539	-0.004003	-0.034707	1.394303	1.377381	-1.202697	-0.992435	1.723318	-0.988902	0.729604	...	0.053584
-0.629111	-0.762513	1.410295	0.943172	-1.419671	0.210036	0.287060	0.219716	-0.635582	-0.202013	...	0.237661
1.357696	0.416963	1.727018	-1.390420	1.482193	0.908208	-1.627760	-0.608779	-0.783390	1.027639	...	-1.720251
-0.630802	1.576862	-0.532711	0.507883	-1.339074	-0.066334	1.677733	-1.500565	0.600258	-0.995573	...	1.133700
-0.091602	0.420434	0.156131	-1.178031	0.276859	-1.387116	-1.382164	-0.415274	0.761929	-0.540183	...	0.130309

ws x 21 columns

Independent and Dependent variable:

- Declaring Input Variable 'X' with respective columns
- Declaring Output 'Y' with Categorical column

```
X = data_1[['F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10', 'F11',
        'F12', 'F13', 'F14', 'F17', 'F18', 'F19', 'F20', 'F21', 'F22']]
y = data_1['C']
```

Oversampling Technique:

- Since Data was heavily imbalance SMOTE is used.

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 0)
X_res, y_res = sm.fit_sample(X,y)
```

- After Oversampling

```
print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))
```

```
Original dataset shape Counter({0: 76353, 1: 24827})
Resampled dataset shape Counter({0: 76353, 1: 76353})
```

Train Test Split:

- Performing splitting of data into train and test with 80-20 split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_res,y_res,test_size = 0.2,random_state = 10)
```

Models

Logistic Regression:

- Logistic Model Algorithm works on sigmoid function to classify the instances into probability ranging between 0 and 1 with a default Threshold of 0.5
- Probability value below 0.5 is labelled as ‘0’
- Probability value above 0.5 is labelled as ‘1’
- 0.5 Threshold value can be change based on domain knowledge or industry requirement.

```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression(class_weight='balanced',random_state=10)
logmodel.fit(X_train,y_train)
```

```
LogisticRegression(class_weight='balanced', random_state=10)
```

- Predicting classes

```
y_pred = logmodel.predict(X_test)
```

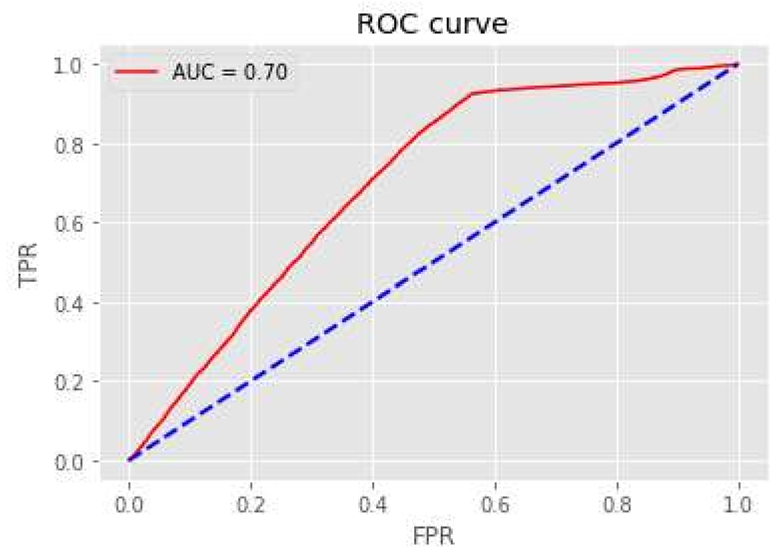
Logistic Regression Output:

```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

Model1 = pd.DataFrame([['Logistic Regression', acc,prec,rec, f1,roc]],
             columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
Model1
```

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Logistic Regression	0.678868	0.627137	0.88656	0.734618	0.678309

- **Logistic Regression ROC AUC Curve**



- **Logistic Regression Confusion Matrix**

```
Confusion Matrix:
[[ 7159  8071]
 [ 1737 13575]]
```

- It is clearly seen that Logistic model is not able to classify properly.
- As Imbalance classification problem we cannot depend upon Accuracy as our Evaluation Metrix.
- In such scenario ROC, precision, recall is considered.

Decision Tree:

- Decision tree is built and on the basis of Gini index value the tree is further splitted.
- It contain Root Node (Top) various Decision Node (Middle) and Terminal Node (Last).

```
from sklearn.tree import DecisionTreeClassifier

dct = DecisionTreeClassifier(random_state = 10)
dct.fit(X_train,y_train)
```

```
DecisionTreeClassifier(random_state=10)
```

- Predicting classes

```
y_pred = dct.predict(X_test)
```

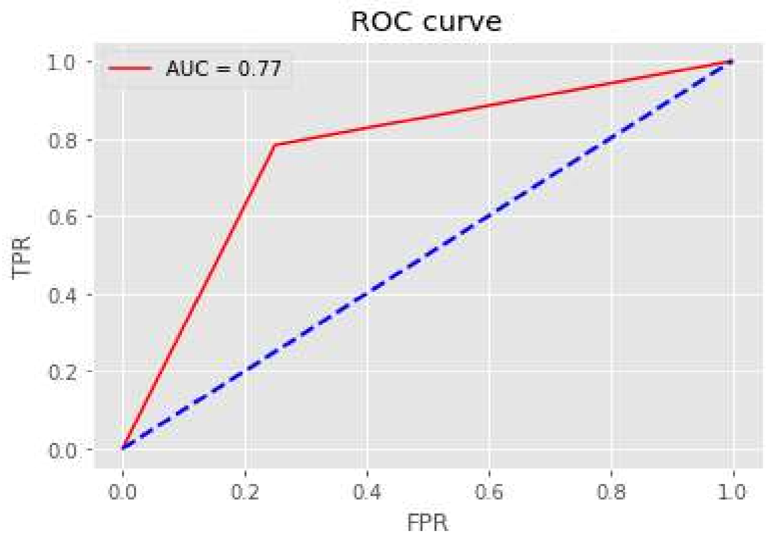
Decision Tree Output:

```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

Model2 = pd.DataFrame([['Decision Tree Classifier', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
Model2=Model1.append(Model2,ignore_index=True)
Model2
```

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Logistic Regression	0.678868	0.627137	0.886560	0.734618	0.678309
1	Decision Tree Classifier	0.766846	0.759225	0.783373	0.771110	0.766801

- **Decision Tree ROC AUC Curve**



- **Decision Tree Confusion Matrix**

```
Confusion Matrix:
[[11426  3804]
 [ 3317 11995]]
```

- Results shows clear improvement from previous model.

Random Forest:

- There are 2 types of ensemble technique Bagging and Boosting
- Random Forest comes under Bagging Technique
- In this Individual Models are built in parallel.
- Equal weights are given to all model.
- End results are calculated on the basis of Mode value for Classification problem.

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state = 10)
rfc.fit(X_train,y_train)
```

```
RandomForestClassifier(random_state=10)
```

- Predicting classes

```
y_pred = rfc.predict(X_test)
```

Random Forest Output:

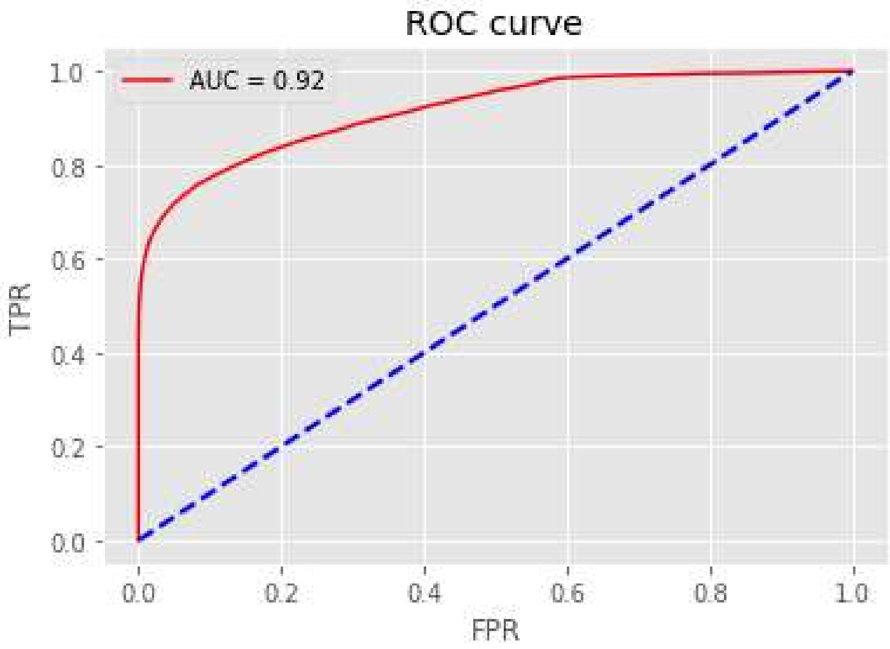
```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

Model3 = pd.DataFrame([['Random tree Classifier', acc,prec,rec, f1,roc]],
                      columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])

Model3=Model2.append(Model3,ignore_index=True)
Model3
```

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Logistic Regression	0.678868	0.627137	0.886560	0.734618	0.678309
1	Decision Tree Classifier	0.766846	0.759225	0.783373	0.771110	0.766801
2	Random tree Classifier	0.832002	0.857454	0.797479	0.826380	0.832095

- **Decision Tree ROC AUC Curve**



- **Decision Tree Confusion Matrix**

```
Confusion Matrix:
[[13200  2030]
 [ 3101 12211]]
```

Conclusion:

- Random Forest show promising result in terms of all Evaluation matrix

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Logistic Regression	0.678868	0.627137	0.886560	0.734618	0.678309
1	Decision Tree Classifier	0.766846	0.759225	0.783373	0.771110	0.766801
2	Random tree Classifier	0.832002	0.857454	0.797479	0.826380	0.832095

- Result can be generated on the basis of company requirement by Bias/Variance Trade-off using Hyperparameter tuning of the Random Forest.
- If company is more focused on Precision or Recall, hyperparameter tuning can be performed & desired results can be generated.

(Some of the Models were not applied due to lack of computational power.)

Output File:

- Same pre-processing is performed on BuyAffinity_Test.txt dataset and Classes are predicted.
- Result stored in Output.csv with Index and Predicted Class columns.

```
test_data=pd.read_csv(r'BuyAffinity_Test.txt',delim_whitespace=True)
test_data1=test_data.drop(['Index','F15','F16'],axis=1)  #drop irrevalent columns
columns_to_scale=['F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10', 'F11',
       'F12', 'F13', 'F14', 'F17', 'F18', 'F19', 'F20', 'F21', 'F22']
test_data1[columns_to_scale]=standardScaler.fit_transform(test_data1[columns_to_scale])

result=rfc.predict(test_data1) #predict

result_df= pd.DataFrame(result)  #predict df

test_index=pd.DataFrame(test_data['Index'])  # Index df

predicted_data=test_index.join(result_df) #join

predicted_data.to_csv('Output.csv')  #output file
```