Project

PURPOSE OF ANALYSIS

The purpose of this analysis to accurately predict the likelihood of customer taking loan. For the purpose of the this analysis we will look at the target_flag to tranform it into our target variable. The analysis will follow the following pattern :

1.Exploring data using summmary statistics

2.Reducing the # of features by looking at their variance, # of 0 values and categorical variables.

3.Exploring data using visualizations

4.Tranforming the feature using Min-Max Transformation

5.Evaluating SVM , Logistic Regression And Decision Tree

6.Conclusions in light of evaluation metrics (Accuracy , SSE and Conformance Matrix)

IMPORTING LIBRARIES

In [1]:

```
from pandas import DataFrame, read_csv
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns

file_name = "MNP_Training.xlsx"
sheet_title = "MNP_Training"
df = pd.read_excel(file_name ,  sheet_name =  sheet_title , index_col= False)
```

In [2]:

```
shape = tuple(df.shape)
print ("Rows in the Dataset - " + str(shape[0]))
print ("Columns in the Dataset - " + str(shape[1]))
```

```
Rows in the Dataset - 19000
Columns in the Dataset - 899
```

In [3]:

```
new_data = df
target_column = "target_flag"
print("SUMMARY STATISTICS - target_flag \n")
print(new_data.target_flag.describe())
```

```
SUMMARY STATISTICS - target_flag

count    19000.000000
mean         0.424842
std          0.494332
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          1.000000
Name: target_flag, dtype: float64
```

In [4]:

```
target_transform = df.target_flag
```

In [5]:

```
print("Target Flag with value 1        - " + str(target_transform.sum()))
print("Target Flag with value 0 - " + str( 19000 - int(target_transform.sum())) )
```

```
Target Flag with value 1        - 8072
Target Flag with value 0 - 10928
```

Dropping Target Flag from Data.

In [6]:

```
new_data = new_data.drop(columns=["target_flag"])
```

Dropping U_ID from Data

In [7]:

```
new_data = new_data.drop(columns=["U_ID"])

summary_statistics = new_data.describe()
summary_statistics.loc["Var"] = new_data.var()
summary_statistics.loc["Type"] = new_data.dtypes
summary_statistics = summary_statistics.transpose()
summary_statistics.sort_values(by = "Var").head(10)
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max	Var	Type
M3_U_OB_VC_CHINA_CNT_T6	19000	0	0	0	0	0	0	0	0	int64
M2_U_OB_VC_BAHRAIN_CNT_T6	19000	0	0	0	0	0	0	0	0	int64
M2_U_OB_VC_BAHRAIN_DUR_T6	19000	0	0	0	0	0	0	0	0	int64
M2_U_OB_VC_OMAN_CNT_T6	19000	0	0	0	0	0	0	0	0	int64
M2_U_OB_VC_OMAN_DUR_T6	19000	0	0	0	0	0	0	0	0	int64
M2_U_OB_VC_KUWAIT_CNT_T6	19000	0	0	0	0	0	0	0	0	int64
M2_U_OB_VC_KUWAIT_DUR_T6	19000	0	0	0	0	0	0	0	0	int64
M1_U_OB_FNF_SMS_CNT_T6	19000	0	0	0	0	0	0	0	0	int64
M1_U_OB_FNF_SMS_REV_T6	19000	0	0	0	0	0	0	0	0	int64
M2_U_RBT_USSD_REV_T6	19000	0	0	0	0	0	0	0	0	int64

DIMENSIONALITY REDUCTION

There are 897 features after removing the identifier U_ID and Target variable. All of these features don't contain meaningful information. Therefore, we will remove some of variables which fullfill the following criteria :

Having Variance < 5 = 373 features

Having Highly Correlated colums (Correlation Threshold of > ||0.9||) = 264 features

In [8]:

```
columns_to_drop = list( summary_statistics.loc[summary_statistics["Var"] <=5 ].index )
new_data = new_data.drop(columns = columns_to_drop)
print("Numbers of columns having variance < 5 - " + str(len(columns_to_drop) ))

corr_matrix = new_data.corr().abs()
corr_matrix_upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
to_drop = [column for column in corr_matrix_upper.columns if any(corr_matrix_upper[column] > 0.9)]
print("Numbers of columns having correlation > +0.90 - " + str(len(to_drop) ))
```

```
new_data = new_data.drop(columns=to_drop)
```

```
Numbers of columns having variance < 5 - 373
Numbers of columns having correlation > +0.90 - 260
```

In [11]:

```
print("\nFinal Numbers of Features - " + str(len(list(new_data))))

summary_statistics.loc[list(new_data)]
```

```
Final Numbers of Features - 264
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max	Va
M1_U_OB_DAY_VC_CC_CNT_T6	19000	22.0178	46.1617	0	0	3	21	574	2130.9
M1_U_OB_VC_DUR_T6	19000	194.836	523.252	0	0	12	141	12352	273793
M1_U_OB_VC_REV_T6	19000	49.5022	109.501	0	0	10.157	55.0637	2203.7	11990.4
M1_U_OB_NGT_VC_CNT_T6	19000	4.06747	12.9211	0	0	0	2	543	166.955
M1_U_OB_NGT_VC_DUR_T6	19000	13.8701	75.9367	0	0	0	3	3284	5766.39
M1_U_OB_NGT_VC_REV_T6	19000	3.73483	13.3196	0	0	0	2.3	758.795	177.411
M1_U_OB_FNF_VC_CNT_T6	19000	0.677789	8.21036	0	0	0	0	443	67.4099
M1_U_OB_FNF_VC_CC_CNT_T6	19000	0.291632	2.84421	0	0	0	0	118	8.08953
M1_U_OB_FNF_VC_DUR_T6	19000	4.21342	66.7319	0	0	0	0	3434	4453.15
M1_U_OB_FNF_VC_REV_T6	19000	0.451316	6.14731	0	0	0	0	314.888	37.7895
M1_U_OB_ONT_VC_REV_T6	19000	26.1783	54.399	0	0	2.449	29.6673	1131.44	2959.25
M1_U_OB_OFNT_VC_CC_CNT_T6	19000	7.32437	21.8787	0	0	0	5	540	478.677
M1_U_OB_OFNT_VC_DUR_T6	19000	16.9394	68.9611	0	0	0	8.5	2311	4755.63
M1_U_OB_OFNT_VC_REV_T6	19000	21.4386	71.6454	0	0	0	13.652	1987.5	5133.06
M1_U_OB_OFNT_LND_VC_REV_T6	19000	0.500514	7.1697	0	0	0	0	545.518	51.4046
M1_U_OB_OFNT_LND_VC_CNT_T6	19000	0.275211	3.30143	0	0	0	0	319	10.8994
M1_U_OB_OFNT_LND_VC_DUR_T6	19000	0.583158	8.05379	0	0	0	0	617	64.8636
M1_U_OB_OFNT_MBLNK_VC_CNT_T6	19000	4.56753	17.3208	0	0	0	2	511	300.009
M1_U_OB_OFNT_MBLNK_VC_DUR_T6	19000	7.32395	36.6135	0	0	0	2.5	1571	1340.55
M1_U_OB_OFNT_MBLNK_VC_REV_T6	19000	9.08075	37.4595	0	0	0	4.451	1596.77	1403.21
M1_U_OB_OFNT_UFN_VC_CNT_T6	19000	2.15911	10.3961	0	0	0	0	490	108.079
M1_U_OB_OFNT_UFN_VC_DUR_T6	19000	3.75626	24.0799	0	0	0	0	1105	579.843
M1_U_OB_OFNT_UFN_VC_REV_T6	19000	4.47774	24.505	0	0	0	0	912.238	600.494
M1_U_OB_OFNT_WRD_VC_CNT_T6	19000	0.972316	7.32186	0	0	0	0	480	53.6096
M1_U_OB_OFNT_WRD_VC_REV_T6	19000	1.98033	14.8244	0	0	0	0	802.5	219.763
M1_U_OB_OFNT_ZNG_VC_CNT_T6	19000	2.43932	10.223	0	0	0	1	451	104.509
M1_U_OB_OFNT_ZNG_VC_REV_T6	19000	5.18835	23.9069	0	0	0	0	614.86	571.54
M1_U_OB_OFNT_INT_VC_DUR_T6	19000	0.289947	5.01546	0	0	0	0	286	25.1548
M1_U_OB_OFNT_INT_VC_REV_T6	19000	1.87024	25.5758	0	0	0	0	1680.96	654.121
M1_U_OB_OFNT_NGT_VC_CNT_T6	19000	0.563684	2.50556	0	0	0	0	86	6.27785
...	...	...	...	...	...	...	...	...	...
M3_RECH_250_300_AMT	19000	2.69064	33.1724	0	0	0	0	1000	1100.41
M3_RECH_300_500_AMT	19000	11.8365	75.1966	0	0	0	0	1600	5654.53

	count	mean	std	min	25%	50%	75%	max	Va
M3_RECH_500_750_AMT	19000	6.03953	71.9135	0	0	0	0	4000	5171.56
M3_RECH_750_1000_AMT	19000	0.434472	18.9973	0	0	0	0	950	360.896
M3_RECH_GT_1000_AMT	19000	6.27021	144.289	0	0	0	0	10602	20819.3
M3_SMARTSHARE_AMT	19000	0.552324	9.24417	0	0	0	0	590.23	85.4547
M3_LOAN_AMT	19000	18.6355	38.741	0	0	0	15	570	1500.86
M3_EC350_RECH_AMT	19000	8.91579	62.2848	0	0	0	0	1400	3879.4
M3_EC600_RECH_AMT	19000	1.98947	35.575	0	0	0	0	1200	1265.58
M3_EL_170_RECH_AMT	19000	0.662105	17.1657	0	0	0	0	680	294.661
M1_OG_CALLS	19000	188.767	364.558	0	3	49	214	14590	132902
M1_OG_FAILURES	19000	6.89132	23.4881	0	0	1	6	1635	551.689
M1_OCQ	19000	77.412	38.6216	0	87.5	97.1	100	200	1491.63
M2_OG_CALLS	19000	167.399	328.999	0	0	35	186	8427	108240
M2_OG_FAILURES	19000	5.90795	21.589	0	0	0	5	1152	466.084
M2_OCQ	19000	69.5925	43.8283	0	0	96.7	99.5	198	1920.92
M2_CST	19000	3.20692	2.37287	0	0	3.78	5.21	22.59	5.63053
M3_OG_CALLS	19000	144.587	302.512	0	0	22	154	5377	91513.2
M3_OG_FAILURES	19000	5.41774	23.8153	0	0	0	4	2030	567.169
M3_OCQ	19000	65.2547	45.6328	0	0	96.2	99.5	200	2082.35
M3_CST	19000	3.00823	2.42358	0	0	3.42	5.15	14.93	5.87376
M1_ARPU	19000	59.8811	214.06	0	0	0	7.23465	4409.33	45821.7
M1_GPRS_REV	19000	8.72038	38.4456	0	0	0	0.052775	2032.29	1478.06
M1_GPRS_VOL	19000	611588	2.55272e+06	0	0	0	3819	8.99962e+07	6.51636e+1
M2_ARPU	19000	58.1483	233.739	0	0	0	0	5341.74	54633.8
M2_GPRS_REV	19000	6.88532	34.2992	0	0	0	0	2311.97	1176.43
M2_GPRS_VOL	19000	583871	2.46374e+06	0	0	0	576	8.28538e+07	6.07002e+1
M3_ARPU	19000	56.4232	230.691	0	0	0	0	4733.53	53218.3
M3_GPRS_REV	19000	6.71406	38.2885	0	0	0	0	2281.22	1466.01
M3_GPRS_VOL	19000	562154	2.39461e+06	0	0	0	124.25	9.91186e+07	5.73415e+1

264 rows × 10 columns

FINAL FEATURE SET

Based on the features removal techniques, we now have a total of 264 variables.

And there are no missing values.

In [12]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
col_names = list(new_data.columns)
new_data[col_names] = scaler.fit_transform(new_data[col_names])
print("\nImputed and MIX Max Tranformation Variable Summary")
new_data.describe().transpose().head(10)
```

```
Imputed and MIX Max Tranformation Variable Summary
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
M1_H_OR_MAX_VC_CC_CNT_T6	19000.0	0.038259	0.080421	0.0	0.0	0.005236	0.036585	1.0

	count	mean	std	min	25%	50%	75%	max
M1_U_OB_DAY_VC_CC_CNT_T6	19000.0	0.038359	0.080421	0.0	0.0	0.005226	0.036585	1.0
M1_U_OB_VC_DUR_T6	19000.0	0.015774	0.042362	0.0	0.0	0.000972	0.011415	1.0
M1_U_OB_VC_REV_T6	19000.0	0.022463	0.049689	0.0	0.0	0.004609	0.024987	1.0
M1_U_OB_NGT_VC_CNT_T6	19000.0	0.007491	0.023796	0.0	0.0	0.000000	0.003683	1.0
M1_U_OB_NGT_VC_DUR_T6	19000.0	0.004224	0.023123	0.0	0.0	0.000000	0.000914	1.0
M1_U_OB_NGT_VC_REV_T6	19000.0	0.004922	0.017554	0.0	0.0	0.000000	0.003031	1.0
M1_U_OB_FNF_VC_CNT_T6	19000.0	0.001530	0.018534	0.0	0.0	0.000000	0.000000	1.0
M1_U_OB_FNF_VC_CC_CNT_T6	19000.0	0.002471	0.024103	0.0	0.0	0.000000	0.000000	1.0
M1_U_OB_FNF_VC_DUR_T6	19000.0	0.001227	0.019433	0.0	0.0	0.000000	0.000000	1.0
M1_U_OB_FNF_VC_REV_T6	19000.0	0.001433	0.019522	0.0	0.0	0.000000	0.000000	1.0

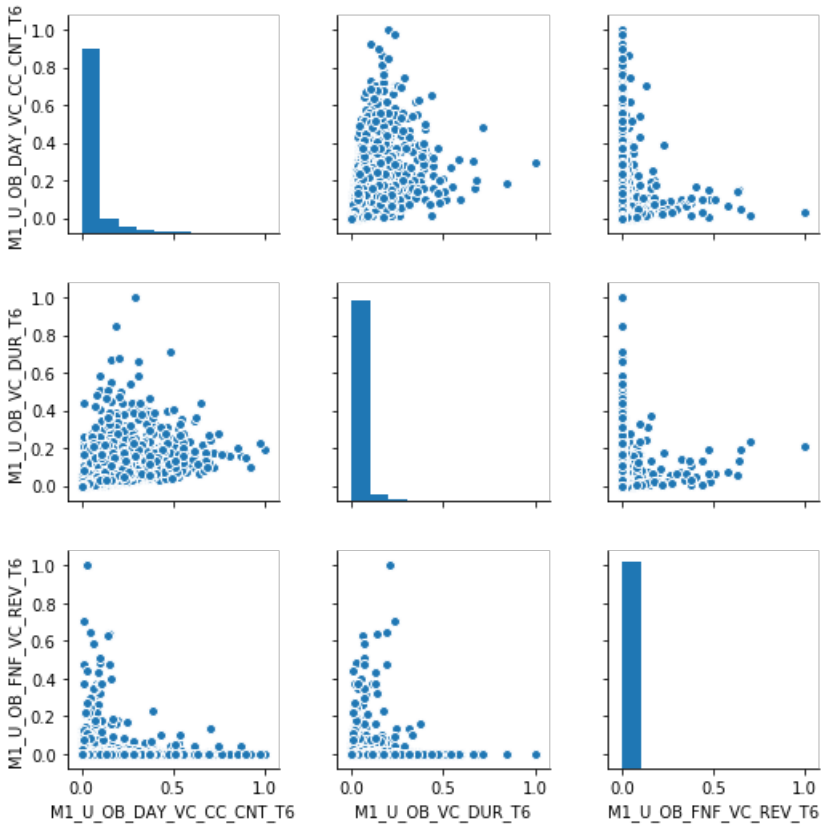
Visual Implimentations:

In [13]:

```
variables_of_interest = [ 'M1_U_OB_DAY_VC_CC_CNT_T6','M1_U_OB_VC_DUR_T6','M1_U_OB_FNF_VC_REV_T6']
sns.pairplot(new_data[variables_of_interest])
```

Out[13]:

```
<seaborn.axisgrid.PairGrid at 0x25f0911e4a8>
```



In [26]:

```
greater_percent = 100.0 * target_transform.sum() / 19000
accuracy = greater_percent / 100.0

# TODO: Calculate F-score using the formula above for beta = 0.5
beta = 0.5
recall = 1.0
fscore = (1 + beta**2) * accuracy * recall / (beta**2 * accuracy + recall)

# Print the results
print("Naive predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".format(accuracy, fscore))
```

```
Naive predictor: [Accuracy score: 0.4248, F-score: 0.4801]
```

SPLITTING THE DATASET

In [14]:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(new_data, target_transform, test_size = 0.3, ra
ndom_state = 0)
print("Train set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Train set has 13300 samples.
Testing set has 5700 samples.
```

```
C:\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was
deprecated in version 0.18 in favor of the model_selection module into which all the refactored cl
asses and functions are moved. Also note that the interface of the new CV iterators are different
from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

MODEL FITTING

If we consider everyone will apply for a customer loan , then benchmark or the Naive Model will have an accuracy of 43% , since total obsevations are 19000 and 8072 have applied for a loan.

Therfoer ,the goal is the identify a model which has a prediction accuracy of greater than 76% . For this purpose we will be fitting the training dataset on the following models :

Logistic Regression

LinearSVC

Decision Tree

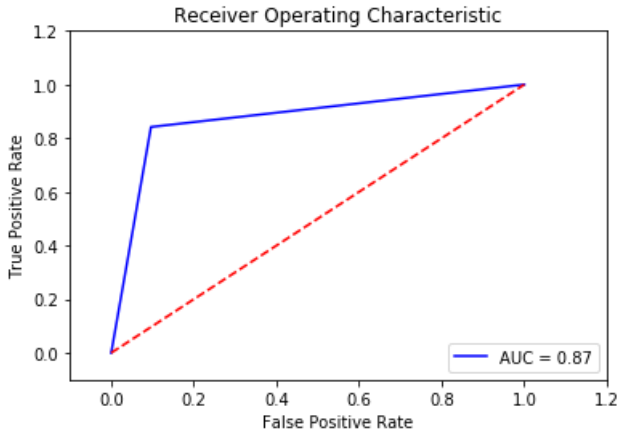
In [151]:

```
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import fbeta_score
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB

import matplotlib.pyplot as plt

clf_A = LogisticRegression(random_state=101)
clf_B = LinearSVC(random_state=101)
clf_C = DecisionTreeClassifier(random_state = 101)

print("\n\nFitting - Logistic Regression Model")
clf_A.fit(X_train,y_train)
print("\nTest Data - Accuracy")
print(clf_A.score(X_test, y_test))
print("\nTest Data - Confusion Matrix")
print(metrics.confusion_matrix(y_test, clf_A.predict(X_test)))
print("\nTest Data - SSE Matrix")
print(mean_squared_error(y_test, clf_A.predict(X_test)))
print("\nTest Data - FScore")
predictions_test = clf_A.predict(X_test)
print(fbeta_score(y_test, predictions_test, beta=beta))

print("\n ROC Curve")
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, clf_A.predict(X_test))
roc_auc = auc(false_positive_rate, true_positive_rate)
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, 'b',
label='AUC = %0.2f'% roc_auc)
```

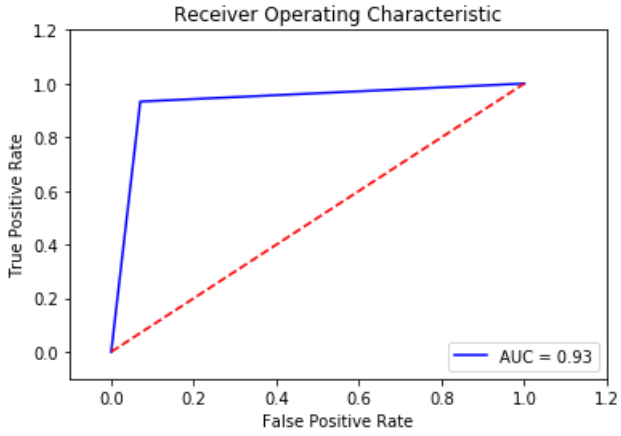
```
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print("\n\nFitting - Linear SVC Model")
clf_B.fit(X_train,y_train)
print("\nTest Data - Accuracy")
print(clf_B.score(X_test, y_test))
print("\nTest Data - Confusion Matrix")
print(metrics.confusion_matrix(y_test, clf_B.predict(X_test)))
print("\nTest Data - SSE Matrix")
print(mean_squared_error(y_test, clf_B.predict(X_test)))
print("\nTest Data - FScore")
predictions_test = clf_B.predict(X_test)
print(fbeta_score(y_test, predictions_test, beta=beta))

print("\n ROC Curve")
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, clf_B.predict(X_test))
roc_auc = auc(false_positive_rate, true_positive_rate)
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, 'b',
label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print("\n\nFitting - Decision Tree Model")
clf_C.fit(X_train,y_train)
print("\nTest Data - Accuracy")
print(clf_C.score(X_test, y_test))
print("\nTest Data - Confusion Matrix")
print(metrics.confusion_matrix(y_test, clf_C.predict(X_test)))
print("\nTest Data - SSE Matrix")
print(mean_squared_error(y_test, clf_C.predict(X_test)))
print("\nTest Data - FScore")
predictions_test = clf_C.predict(X_test)
print(fbeta_score(y_test, predictions_test, beta=beta))

print("\n ROC Curve")
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, clf_C.predict(X_test))
roc_auc = auc(false_positive_rate, true_positive_rate)
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, 'b',
label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()


import pickle
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

```
Fitting - Logistic Regression Model

Test Data - Accuracy
0.8773684210526316

Test Data - Confusion Matrix
[[2956  315]
 [ 384 2045]]

Test Data - SSE Matrix
```

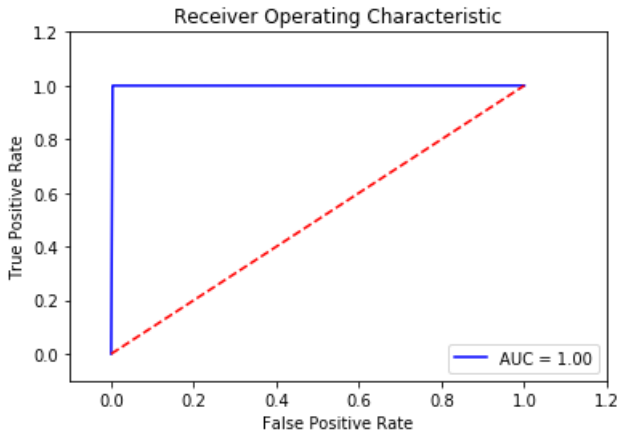
```
0.12263157894736842

Test Data - FScore
0.8614879096806807

 ROC Curve
```




```
Fitting - Linear SVC Model

Test Data - Accuracy
0.9310526315789474

Test Data - Confusion Matrix
[[3041  230]
 [ 163 2266]]

Test Data - SSE Matrix
0.06894736842105263

Test Data - FScore
0.9127527592040603

 ROC Curve
```




```
Fitting - Decision Tree Model

Test Data - Accuracy
0.9978947368421053

Test Data - Confusion Matrix
[[3260   11]
 [   1 2428]]

Test Data - SSE Matrix
0.002105263157894737

Test Data - FScore
0.9963069347558474

 ROC Curve
```



Results:

Metric	Logistic Regression	Linear SVC	Decision Tree
Accuracy Score	0.8773	0.9310	0.9978
F-score	0.8614	0.9127	0.9963

Decision Tree Model is the Best Model

Implementation: Model Tuning

Fine tune the chosen model. Use grid search (`GridSearchCV`) with at least one important parameter tuned with at least 3 different values. You will need to use the entire training set for this.

As there is Accuracy Score of almost 1 for decision tree, so we choose logistic regression for model tuning.

In [45]:

```
# TODO: Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer

# TODO: Initialize the classifier
clf = LogisticRegression(random_state=101)

# TODO: Create the parameters list you wish to tune
parameters = {'solver': ['newton-cg', 'lbfgs', 'sag'],
              'C': [0.01, 0.1, 1.0, 10.0, 100.0]
             }

# TODO: Make an fbeta_score scoring object
scorer = make_scorer(fbeta_score, beta=beta)

# TODO: Perform grid search on the classifier using 'scorer' as the scoring method
grid_obj = GridSearchCV(clf, parameters, scoring=scorer)

# TODO: Fit the grid search object to the training data and find the optimal parameters
grid_fit = grid_obj.fit(X_train, y_train)

# Get the estimator
best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and model
predictions = (clf.fit(X_train, y_train)).predict(X_test)
best_predictions = best_clf.predict(X_test)
# Report the before-and-afterscores
print ("Unoptimized model\n------")
print ("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions)))
print ("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta = 0.5)))
print ("\nOptimized Model\n------")
print ("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test, best_predic
tions)))
```

```
print ("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, bet
a = 0.5)))

# show best parameters
print ("\nBest Classifier\n------")
print (best_clf)
```

```
C:\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning: The max_iter w
as reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning: The max_iter w
as reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning: The max_iter w
as reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning: The max_iter w
as reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning: The max_iter w
as reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning: The max_iter w
as reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
Unoptimized model
------
Accuracy score on testing data: 0.8774
F-score on testing data: 0.8615

Optimized Model
------
Final accuracy score on the testing data: 0.9642
Final F-score on the testing data: 0.9472

Best Classifier
------
LogisticRegression(C=100.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=101, solver='newton-cg', tol=0.0001,
          verbose=0, warm_start=False)
```

Results:

Metric	Benchmark Predictor	Unoptimized Model	Optimized Model
Accuracy Score	0.4248	0.8774	0.9642
F-score	0.4801	0.8615	0.9472

CLASSIFICATION

In [51]:

```
# TODO: Import a supervised learning model that has 'feature_importances_'
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import fbeta_score, accuracy_score

# TODO: Train the supervised model on the training set
clf_a = GradientBoostingClassifier().fit(X_train, y_train)
clf_b = AdaBoostClassifier().fit(X_train, y_train)
clf_c = RandomForestClassifier().fit(X_train, y_train)


# show scores
predictions = clf_a.predict(X_test)
model_accuracy = accuracy_score(y_test, predictions)
model_fscore = fbeta_score(y_test, predictions, beta=0.5)
print ("\nGradient Boosting Model accuracy:", model_accuracy, ", fscore: ", model_fscore)
```

```
predictions = clf_b.predict(X_test)
model_accuracy = accuracy_score(y_test, predictions)
model_fscore = fbeta_score(y_test, predictions, beta=0.5)
print ("\nAdaptive Boosting Model accuracy:", model_accuracy, ", fscore: ", model_fscore)

predictions = clf_c.predict(X_test)
model_accuracy = accuracy_score(y_test, predictions)
model_fscore = fbeta_score(y_test, predictions, beta=0.5)
print ("\nRandom Forest Model accuracy:", model_accuracy, ", fscore: ", model_fscore)
```

```
Gradient Boosting Model accuracy: 0.9973684210526316 , fscore:  0.9953267196851684

Adaptive Boosting Model accuracy: 0.9980701754385964 , fscore:  0.996390187874313

Random Forest Model accuracy: 0.9963157894736843 , fscore:  0.9955551897275496
```

In [55]:

```
clf_b = AdaBoostClassifier(n_estimators=500,random_state=10).fit(X_train, y_train)

predictions = clf_b.predict(X_test)
model_accuracy = accuracy_score(y_test, predictions)
model_fscore = fbeta_score(y_test, predictions, beta=0.5)
print ("\nAdaptive Boosting Model accuracy:", model_accuracy, ", fscore: ", model_fscore)
```

```
Adaptive Boosting Model accuracy: 0.9982456140350877 , fscore:  0.996717275338531
```

This accuracy is more closer to 1 and we take it as the best boosting model.

Implementation - Extracting Feature Importance

Choose a scikit-learn supervised learning algorithm that has a feature *importance* attribute availble for it. This attribute is a function that ranks the importance of each feature when making predictions based on the chosen algorithm.

So we are extracting Feature importance from the Decision Tree Model.

In [52]:

```
DecisionTreeModel = clf_C

# TODO: Extract the feature importances
importances = DecisionTreeModel.feature_importances_

# show most importance features
a = np.array(importances)
factors = pd.DataFrame(data = np.array([importances.astype(float), new_data.columns]).T,
                       columns = ['importances', 'features'])
factors = factors.sort_values('importances', ascending=False)

print ("\n Top 10 important features")
display(factors[:10])
```

```
 Top 10 important features
```

	importances	features
255	0.564015	M1_ARPU
244	0.336369	M1_OG_CALLS
251	0.0561024	M3_OG_CALLS
257	0.0179411	M1_GPRS_VOL
184	0.01146	M1_RECH_AMT
247	0.00607037	M2_OG_CALLS
61	0.00442275	M1_U_VAS_REV_T6
204	0.000893415	M2_RECH_AMT

225	0.000318725	M3_RECH_Features
261	0.000310502	M3_ARPU

Scoring from scoring data

In [146]:

```
from pandas import DataFrame, read_csv
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns

file_name = "MNP_Scoring.xlsx"
sheet_title = "MNP_Scoring"
df1 = pd.read_excel(file_name ,  sheet_name =  sheet_title , index_col= False)
```

In [147]:

```
new_data1 = df1.drop(columns=["U_ID"])
```

In [148]:

```
summary_statistics1 = new_data1.describe()
summary_statistics1.loc["Var"] = new_data1.var()
summary_statistics1.loc["Type"] = new_data1.dtypes
summary_statistics1 = summary_statistics1.transpose()
summary_statistics1.sort_values(by ="Var").head(10)
```

Out[148]:

	count	mean	std	min	25%	50%	75%	max	Var	Type
M1_U_OB_VC_France_DUR_T6	2000	0	0	0	0	0	0	0	0	int64
M1_U_OB_HYBD_INT_VC_DUR_T6	2000	0	0	0	0	0	0	0	0	int64
M1_U_OB_VC_Germany_DUR_T6	2000	0	0	0	0	0	0	0	0	int64
M2_U_VAS_CC_SMS_CNT_T6	2000	0	0	0	0	0	0	0	0	int64
M1_U_VAS_PHNBOOK_SVC_REV_T6	2000	0	0	0	0	0	0	0	0	int64
M1_U_OB_VC_France_CNT_T6	2000	0	0	0	0	0	0	0	0	int64
M1_U_VAS_PHNBOOK_SVC_CNT_T6	2000	0	0	0	0	0	0	0	0	int64
M1_U_VAS_NEWS_RADIO_CNT_T6	2000	0	0	0	0	0	0	0	0	int64
M1_U_OB_VC_Norway_DUR_T6	2000	0	0	0	0	0	0	0	0	int64
M3_U_OB_VC_Denmark_CNT_T6	2000	0	0	0	0	0	0	0	0	int64

In [149]:

```
columns_to_drop1 = list( summary_statistics1.loc[summary_statistics1["Var"] <=3.5 ].index )
print("Numbers of columns having variance < 3.5 - " + str(len(columns_to_drop1) ))
new_data1 = new_data1.drop(columns = columns_to_drop1)

corr_matrix1 = new_data1.corr().abs()
corr_matrix_upper1 = corr_matrix1.where(np.triu(np.ones(corr_matrix1.shape), k=1).astype(np.bool))
to_drop1 = [column for column in corr_matrix_upper1.columns if any(corr_matrix_upper1[column] > 0.9
385)]
print("Numbers of columns having correlation > +0.9385 - " + str(len(to_drop1) ))
new_data1 = new_data1.drop(columns=to_drop1)

print("\nFinal Numbers of Features - " + str(len(list(new_data1))))
```

```
Numbers of columns having variance < 3.5 - 373
Numbers of columns having correlation > +0.9 - 260
```

Final Numbers of Features - 264

In [153]:

```
import pickle

filename_model = 'finalized_model.sav'

loaded_model = pickle.load(open(filename_model, 'rb'))

target1 = target_transform[0:2000]

from sklearn.cross_validation import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(new_data1, target1, test_size = 0.1, random
_state = 0)
print("Training set has {} samples.".format(X_train1.shape[0]))
print("Testing set has {} samples.".format(X_test1.shape[0]))
```

```
Training set has 1800 samples.
Testing set has 200 samples.
```

In [159]:

```
result1 = loaded_model.score(X_train1, y_train1)
result2 = loaded_model.score(X_test1, y_test1)
```

In [177]:

```
result = result1 + result2
print("Score =",result)
```

```
Score = 0.9477777777777778
```

In [163]:

```
prediction1 = loaded_model.predict(X_train1)
```

In [164]:

```
print(prediction1)
```

```
[1 1 1 ... 1 0 1]
```

In [174]:

```
print("Outcome")
print("-------")
for i,each in enumerate(prediction1,start=1):
    print ("{}. {}".format(i,each))
```

```
Outcome
-------
1. 1
2. 1
3. 1
4. 1
5. 1
6. 1
7. 1
8. 1
9. 0
10. 1
11. 0
12. 1
13. 1
14. 1
15. 1
16. 1
```

17. 1
18. 1
19. 1
20. 1
21. 1
22. 1
23. 1
24. 1
25. 1
26. 0
27. 1
28. 1
29. 1
30. 1
31. 1
32. 1
33. 1
34. 1
35. 1
36. 1
37. 1
38. 0
39. 1
40. 1
41. 1
42. 0
43. 1
44. 1
45. 1
46. 1
47. 1
48. 1
49. 1
50. 1
51. 0
52. 1
53. 0
54. 1
55. 1
56. 1
57. 1
58. 1
59. 1
60. 1
61. 1
62. 1
63. 1
64. 1
65. 0
66. 0
67. 1
68. 1
69. 1
70. 1
71. 1
72. 1
73. 1
74. 1
75. 1
76. 1
77. 1
78. 1
79. 0
80. 1
81. 1
82. 1
83. 1
84. 1
85. 1
86. 1
87. 1
88. 1
89. 1
90. 0
91. 1
92. 1
93. 1
94. 1

94. 1
95. 1
96. 1
97. 1
98. 1
99. 1
100. 1
101. 1
102. 1
103. 1
104. 1
105. 1
106. 1
107. 1
108. 1
109. 1
110. 1
111. 1
112. 1
113. 1
114. 1
115. 1
116. 0
117. 1
118. 1
119. 1
120. 1
121. 1
122. 0
123. 1
124. 1
125. 1
126. 1
127. 1
128. 1
129. 1
130. 1
131. 1
132. 1
133. 1
134. 1
135. 1
136. 1
137. 1
138. 1
139. 1
140. 1
141. 0
142. 1
143. 1
144. 1
145. 1
146. 1
147. 1
148. 1
149. 1
150. 1
151. 1
152. 1
153. 1
154. 0
155. 1
156. 1
157. 0
158. 1
159. 0
160. 1
161. 1
162. 0
163. 1
164. 0
165. 0
166. 1
167. 1
168. 1
169. 1
170. 0

171. 1
172. 1
173. 0
174. 1
175. 1
176. 1
177. 1
178. 0
179. 1
180. 1
181. 1
182. 1
183. 1
184. 1
185. 0
186. 0
187. 1
188. 1
189. 1
190. 1
191. 1
192. 1
193. 1
194. 1
195. 1
196. 1
197. 1
198. 1
199. 1
200. 1
201. 1
202. 0
203. 1
204. 1
205. 1
206. 1
207. 1
208. 1
209. 1
210. 1
211. 1
212. 0
213. 1
214. 1
215. 1
216. 1
217. 1
218. 1
219. 1
220. 0
221. 1
222. 1
223. 0
224. 1
225. 1
226. 1
227. 0
228. 0
229. 1
230. 1
231. 1
232. 0
233. 1
234. 1
235. 1
236. 1
237. 1
238. 1
239. 1
240. 1
241. 1
242. 1
243. 1
244. 1
245. 1
246. 1
247. 0

248. 1
249. 0
250. 1
251. 1
252. 1
253. 1
254. 1
255. 1
256. 1
257. 0
258. 0
259. 0
260. 1
261. 0
262. 1
263. 1
264. 1
265. 1
266. 1
267. 0
268. 1
269. 1
270. 1
271. 1
272. 1
273. 1
274. 0
275. 1
276. 1
277. 1
278. 1
279. 0
280. 1
281. 1
282. 1
283. 1
284. 0
285. 1
286. 1
287. 0
288. 1
289. 1
290. 1
291. 1
292. 1
293. 1
294. 0
295. 1
296. 1
297. 0
298. 1
299. 0
300. 1
301. 1
302. 0
303. 1
304. 1
305. 0
306. 1
307. 1
308. 1
309. 1
310. 1
311. 1
312. 1
313. 1
314. 1
315. 0
316. 0
317. 1
318. 1
319. 1
320. 1
321. 1
322. 1
323. 1
324. 1

325. 1
326. 0
327. 1
328. 1
329. 1
330. 1
331. 1
332. 1
333. 1
334. 0
335. 1
336. 1
337. 0
338. 1
339. 1
340. 1
341. 1
342. 0
343. 1
344. 1
345. 1
346. 1
347. 1
348. 1
349. 1
350. 1
351. 1
352. 1
353. 1
354. 1
355. 1
356. 1
357. 1
358. 0
359. 1
360. 1
361. 1
362. 1
363. 1
364. 1
365. 1
366. 1
367. 1
368. 1
369. 1
370. 1
371. 1
372. 0
373. 1
374. 1
375. 1
376. 1
377. 1
378. 1
379. 1
380. 1
381. 1
382. 1
383. 1
384. 1
385. 1
386. 1
387. 0
388. 0
389. 1
390. 1
391. 1
392. 0
393. 0
394. 1
395. 1
396. 0
397. 1
398. 1
399. 1
400. 0
401. 1

402. 1
403. 1
404. 1
405. 0
406. 0
407. 1
408. 1
409. 1
410. 1
411. 0
412. 1
413. 1
414. 1
415. 0
416. 1
417. 1
418. 1
419. 1
420. 1
421. 1
422. 0
423. 1
424. 1
425. 1
426. 1
427. 1
428. 1
429. 1
430. 1
431. 1
432. 1
433. 1
434. 1
435. 1
436. 1
437. 1
438. 0
439. 1
440. 0
441. 1
442. 1
443. 1
444. 0
445. 1
446. 1
447. 1
448. 1
449. 0
450. 0
451. 0
452. 0
453. 1
454. 1
455. 1
456. 0
457. 1
458. 1
459. 1
460. 0
461. 1
462. 0
463. 0
464. 1
465. 1
466. 1
467. 1
468. 1
469. 1
470. 0
471. 1
472. 1
473. 1
474. 1
475. 1
476. 1
477. 1
478. 1

479. 1
480. 1
481. 0
482. 1
483. 1
484. 1
485. 1
486. 1
487. 0
488. 1
489. 1
490. 1
491. 1
492. 1
493. 1
494. 1
495. 0
496. 1
497. 1
498. 1
499. 1
500. 1
501. 1
502. 1
503. 0
504. 1
505. 1
506. 1
507. 1
508. 1
509. 0
510. 1
511. 1
512. 1
513. 1
514. 1
515. 1
516. 1
517. 1
518. 1
519. 1
520. 1
521. 0
522. 1
523. 1
524. 1
525. 1
526. 1
527. 1
528. 1
529. 1
530. 1
531. 1
532. 1
533. 1
534. 0
535. 1
536. 0
537. 1
538. 1
539. 1
540. 1
541. 1
542. 1
543. 1
544. 1
545. 0
546. 1
547. 1
548. 1
549. 0
550. 0
551. 1
552. 1
553. 1
554. 1
555. 1

556. 1
557. 0
558. 0
559. 1
560. 1
561. 0
562. 1
563. 1
564. 0
565. 1
566. 1
567. 1
568. 1
569. 1
570. 1
571. 1
572. 0
573. 1
574. 1
575. 1
576. 1
577. 1
578. 1
579. 1
580. 1
581. 1
582. 1
583. 1
584. 1
585. 0
586. 1
587. 1
588. 0
589. 1
590. 0
591. 1
592. 1
593. 1
594. 1
595. 1
596. 1
597. 1
598. 1
599. 1
600. 1
601. 1
602. 0
603. 1
604. 1
605. 1
606. 1
607. 1
608. 0
609. 1
610. 1
611. 0
612. 1
613. 1
614. 0
615. 1
616. 1
617. 1
618. 1
619. 1
620. 1
621. 1
622. 1
623. 1
624. 0
625. 1
626. 1
627. 1
628. 1
629. 1
630. 1
631. 1
632. 1

633. 1
634. 1
635. 0
636. 1
637. 1
638. 1
639. 1
640. 1
641. 1
642. 1
643. 1
644. 1
645. 0
646. 1
647. 1
648. 1
649. 0
650. 1
651. 1
652. 1
653. 1
654. 1
655. 0
656. 1
657. 1
658. 1
659. 1
660. 1
661. 1
662. 0
663. 1
664. 1
665. 1
666. 0
667. 1
668. 1
669. 1
670. 1
671. 1
672. 1
673. 1
674. 1
675. 1
676. 0
677. 1
678. 1
679. 1
680. 1
681. 1
682. 1
683. 1
684. 1
685. 1
686. 1
687. 1
688. 1
689. 0
690. 1
691. 0
692. 1
693. 0
694. 1
695. 1
696. 1
697. 1
698. 0
699. 1
700. 1
701. 1
702. 0
703. 1
704. 1
705. 1
706. 1
707. 1
708. 1
709. 0

710. 1
711. 1
712. 1
713. 1
714. 1
715. 1
716. 1
717. 1
718. 1
719. 1
720. 0
721. 1
722. 1
723. 1
724. 1
725. 1
726. 1
727. 1
728. 0
729. 1
730. 1
731. 1
732. 1
733. 1
734. 1
735. 1
736. 1
737. 1
738. 1
739. 1
740. 1
741. 1
742. 1
743. 1
744. 1
745. 1
746. 1
747. 1
748. 1
749. 0
750. 0
751. 0
752. 1
753. 1
754. 0
755. 0
756. 1
757. 1
758. 1
759. 1
760. 1
761. 1
762. 1
763. 1
764. 1
765. 1
766. 0
767. 1
768. 1
769. 1
770. 1
771. 0
772. 1
773. 1
774. 1
775. 0
776. 0
777. 1
778. 0
779. 1
780. 1
781. 0
782. 1
783. 0
784. 1
785. 1
786. 0

787. 1
788. 1
789. 1
790. 1
791. 1
792. 1
793. 0
794. 0
795. 0
796. 1
797. 1
798. 1
799. 1
800. 1
801. 1
802. 1
803. 1
804. 1
805. 1
806. 1
807. 1
808. 1
809. 1
810. 1
811. 1
812. 1
813. 1
814. 1
815. 1
816. 1
817. 1
818. 1
819. 0
820. 1
821. 0
822. 1
823. 1
824. 1
825. 0
826. 1
827. 1
828. 1
829. 1
830. 1
831. 1
832. 1
833. 1
834. 1
835. 1
836. 0
837. 1
838. 1
839. 1
840. 1
841. 0
842. 1
843. 1
844. 1
845. 1
846. 1
847. 1
848. 1
849. 1
850. 1
851. 1
852. 1
853. 1
854. 1
855. 1
856. 1
857. 1
858. 1
859. 1
860. 1
861. 1
862. 1
863. 1

864. 1
865. 0
866. 0
867. 1
868. 1
869. 1
870. 1
871. 1
872. 1
873. 1
874. 1
875. 1
876. 0
877. 0
878. 1
879. 0
880. 0
881. 0
882. 1
883. 0
884. 1
885. 1
886. 1
887. 0
888. 1
889. 1
890. 1
891. 1
892. 1
893. 1
894. 1
895. 1
896. 1
897. 0
898. 1
899. 1
900. 1
901. 1
902. 1
903. 0
904. 1
905. 1
906. 1
907. 1
908. 0
909. 1
910. 0
911. 0
912. 1
913. 1
914. 1
915. 1
916. 1
917. 1
918. 1
919. 1
920. 1
921. 0
922. 1
923. 0
924. 1
925. 1
926. 1
927. 1
928. 1
929. 1
930. 1
931. 0
932. 1
933. 1
934. 0
935. 1
936. 1
937. 1
938. 1
939. 0
940. 1

941. 0
942. 1
943. 1
944. 1
945. 1
946. 1
947. 0
948. 1
949. 1
950. 1
951. 1
952. 1
953. 1
954. 1
955. 1
956. 1
957. 1
958. 1
959. 0
960. 0
961. 1
962. 0
963. 1
964. 1
965. 1
966. 1
967. 0
968. 1
969. 1
970. 1
971. 1
972. 0
973. 1
974. 1
975. 1
976. 1
977. 1
978. 1
979. 1
980. 1
981. 1
982. 0
983. 1
984. 1
985. 1
986. 1
987. 1
988. 1
989. 0
990. 1
991. 0
992. 1
993. 1
994. 1
995. 1
996. 1
997. 1
998. 1
999. 0
1000. 1
1001. 1
1002. 0
1003. 1
1004. 1
1005. 1
1006. 1
1007. 1
1008. 1
1009. 1
1010. 1
1011. 0
1012. 1
1013. 1
1014. 1
1015. 1
1016. 1
1017. 0

1017. 0
1018. 1
1019. 1
1020. 1
1021. 1
1022. 1
1023. 1
1024. 1
1025. 1
1026. 1
1027. 0
1028. 1
1029. 1
1030. 1
1031. 1
1032. 1
1033. 1
1034. 1
1035. 1
1036. 1
1037. 0
1038. 1
1039. 0
1040. 1
1041. 1
1042. 1
1043. 1
1044. 1
1045. 1
1046. 1
1047. 0
1048. 0
1049. 1
1050. 1
1051. 1
1052. 0
1053. 0
1054. 1
1055. 1
1056. 1
1057. 1
1058. 1
1059. 1
1060. 1
1061. 1
1062. 1
1063. 1
1064. 1
1065. 0
1066. 0
1067. 1
1068. 0
1069. 1
1070. 1
1071. 0
1072. 0
1073. 1
1074. 1
1075. 1
1076. 1
1077. 1
1078. 1
1079. 1
1080. 1
1081. 1
1082. 1
1083. 1
1084. 1
1085. 1
1086. 1
1087. 0
1088. 1
1089. 1
1090. 1
1091. 0
1092. 0
1093. 1
1094. 1

1094. 1
1095. 1
1096. 1
1097. 1
1098. 1
1099. 0
1100. 0
1101. 0
1102. 1
1103. 1
1104. 1
1105. 1
1106. 1
1107. 1
1108. 1
1109. 1
1110. 1
1111. 0
1112. 1
1113. 1
1114. 1
1115. 1
1116. 1
1117. 1
1118. 1
1119. 1
1120. 0
1121. 1
1122. 1
1123. 1
1124. 0
1125. 1
1126. 0
1127. 1
1128. 0
1129. 1
1130. 1
1131. 1
1132. 1
1133. 1
1134. 1
1135. 1
1136. 1
1137. 1
1138. 1
1139. 1
1140. 1
1141. 1
1142. 1
1143. 1
1144. 0
1145. 1
1146. 1
1147. 1
1148. 1
1149. 1
1150. 1
1151. 0
1152. 1
1153. 1
1154. 1
1155. 1
1156. 1
1157. 1
1158. 0
1159. 0
1160. 1
1161. 0
1162. 0
1163. 1
1164. 0
1165. 1
1166. 1
1167. 1
1168. 0
1169. 1
1170. 0
1171. 1

1171. 1
1172. 1
1173. 1
1174. 1
1175. 1
1176. 1
1177. 1
1178. 1
1179. 1
1180. 0
1181. 1
1182. 0
1183. 1
1184. 1
1185. 1
1186. 1
1187. 1
1188. 1
1189. 0
1190. 0
1191. 1
1192. 1
1193. 1
1194. 1
1195. 1
1196. 1
1197. 1
1198. 1
1199. 1
1200. 1
1201. 1
1202. 0
1203. 1
1204. 0
1205. 1
1206. 1
1207. 0
1208. 1
1209. 0
1210. 0
1211. 1
1212. 1
1213. 1
1214. 1
1215. 0
1216. 1
1217. 1
1218. 1
1219. 1
1220. 1
1221. 1
1222. 1
1223. 1
1224. 1
1225. 1
1226. 1
1227. 1
1228. 1
1229. 1
1230. 1
1231. 1
1232. 1
1233. 1
1234. 1
1235. 1
1236. 1
1237. 1
1238. 1
1239. 0
1240. 1
1241. 0
1242. 1
1243. 1
1244. 1
1245. 1
1246. 1
1247. 0
1248. 1

1248. 1
1249. 1
1250. 0
1251. 0
1252. 1
1253. 1
1254. 1
1255. 1
1256. 1
1257. 1
1258. 0
1259. 0
1260. 1
1261. 1
1262. 1
1263. 1
1264. 1
1265. 0
1266. 1
1267. 1
1268. 1
1269. 1
1270. 1
1271. 0
1272. 0
1273. 1
1274. 1
1275. 1
1276. 1
1277. 1
1278. 1
1279. 1
1280. 0
1281. 1
1282. 1
1283. 0
1284. 0
1285. 1
1286. 0
1287. 1
1288. 0
1289. 1
1290. 1
1291. 1
1292. 1
1293. 1
1294. 1
1295. 1
1296. 1
1297. 1
1298. 1
1299. 1
1300. 0
1301. 1
1302. 1
1303. 1
1304. 1
1305. 1
1306. 1
1307. 1
1308. 1
1309. 1
1310. 1
1311. 1
1312. 0
1313. 1
1314. 1
1315. 1
1316. 1
1317. 1
1318. 0
1319. 1
1320. 1
1321. 1
1322. 0
1323. 1
1324. 1
1325. 1

1325. 1
1326. 1
1327. 1
1328. 1
1329. 1
1330. 1
1331. 1
1332. 1
1333. 1
1334. 0
1335. 1
1336. 1
1337. 1
1338. 1
1339. 1
1340. 1
1341. 1
1342. 1
1343. 1
1344. 1
1345. 1
1346. 1
1347. 0
1348. 1
1349. 1
1350. 1
1351. 1
1352. 1
1353. 1
1354. 1
1355. 1
1356. 1
1357. 1
1358. 1
1359. 1
1360. 1
1361. 0
1362. 1
1363. 0
1364. 0
1365. 1
1366. 1
1367. 0
1368. 0
1369. 1
1370. 0
1371. 1
1372. 1
1373. 1
1374. 1
1375. 1
1376. 1
1377. 1
1378. 1
1379. 0
1380. 1
1381. 1
1382. 0
1383. 1
1384. 1
1385. 1
1386. 1
1387. 1
1388. 1
1389. 1
1390. 1
1391. 1
1392. 1
1393. 1
1394. 1
1395. 1
1396. 1
1397. 0
1398. 1
1399. 0
1400. 1
1401. 1

1402. 1
1403. 1
1404. 1
1405. 0
1406. 0
1407. 1
1408. 1
1409. 1
1410. 1
1411. 1
1412. 1
1413. 1
1414. 1
1415. 1
1416. 1
1417. 1
1418. 1
1419. 1
1420. 1
1421. 0
1422. 1
1423. 1
1424. 1
1425. 1
1426. 1
1427. 1
1428. 1
1429. 1
1430. 1
1431. 1
1432. 1
1433. 1
1434. 1
1435. 1
1436. 1
1437. 1
1438. 1
1439. 1
1440. 1
1441. 1
1442. 1
1443. 1
1444. 1
1445. 1
1446. 0
1447. 1
1448. 0
1449. 1
1450. 1
1451. 0
1452. 1
1453. 0
1454. 1
1455. 1
1456. 0
1457. 1
1458. 0
1459. 1
1460. 0
1461. 1
1462. 1
1463. 0
1464. 0
1465. 1
1466. 1
1467. 1
1468. 1
1469. 1
1470. 1
1471. 1
1472. 1
1473. 1
1474. 0
1475. 1
1476. 1
1477. 1
1478. 0

1479. 0
1480. 1
1481. 1
1482. 1
1483. 1
1484. 1
1485. 1
1486. 1
1487. 1
1488. 1
1489. 1
1490. 1
1491. 1
1492. 0
1493. 1
1494. 0
1495. 1
1496. 1
1497. 1
1498. 1
1499. 1
1500. 1
1501. 1
1502. 1
1503. 1
1504. 1
1505. 0
1506. 1
1507. 1
1508. 0
1509. 0
1510. 1
1511. 1
1512. 1
1513. 1
1514. 1
1515. 1
1516. 1
1517. 1
1518. 1
1519. 0
1520. 1
1521. 1
1522. 1
1523. 1
1524. 1
1525. 1
1526. 0
1527. 1
1528. 1
1529. 1
1530. 1
1531. 1
1532. 1
1533. 1
1534. 1
1535. 1
1536. 1
1537. 1
1538. 1
1539. 0
1540. 1
1541. 1
1542. 1
1543. 1
1544. 1
1545. 1
1546. 0
1547. 0
1548. 1
1549. 1
1550. 1
1551. 1
1552. 1
1553. 0
1554. 1
1555. 1

1556. 1
1557. 1
1558. 0
1559. 0
1560. 1
1561. 1
1562. 1
1563. 1
1564. 1
1565. 1
1566. 1
1567. 1
1568. 0
1569. 1
1570. 1
1571. 1
1572. 1
1573. 0
1574. 0
1575. 1
1576. 1
1577. 1
1578. 1
1579. 0
1580. 1
1581. 1
1582. 1
1583. 1
1584. 1
1585. 1
1586. 1
1587. 1
1588. 1
1589. 1
1590. 0
1591. 1
1592. 1
1593. 1
1594. 1
1595. 1
1596. 1
1597. 1
1598. 1
1599. 1
1600. 1
1601. 1
1602. 0
1603. 1
1604. 1
1605. 0
1606. 1
1607. 0
1608. 1
1609. 1
1610. 1
1611. 0
1612. 1
1613. 0
1614. 1
1615. 1
1616. 1
1617. 0
1618. 1
1619. 1
1620. 1
1621. 1
1622. 1
1623. 1
1624. 1
1625. 1
1626. 1
1627. 1
1628. 1
1629. 1
1630. 1
1631. 1
1632. 1

1633. 1
1634. 0
1635. 1
1636. 1
1637. 1
1638. 1
1639. 1
1640. 1
1641. 1
1642. 0
1643. 1
1644. 1
1645. 1
1646. 1
1647. 1
1648. 1
1649. 0
1650. 1
1651. 0
1652. 1
1653. 0
1654. 1
1655. 1
1656. 0
1657. 1
1658. 1
1659. 1
1660. 1
1661. 1
1662. 1
1663. 0
1664. 1
1665. 0
1666. 1
1667. 1
1668. 1
1669. 1
1670. 1
1671. 1
1672. 1
1673. 1
1674. 1
1675. 1
1676. 1
1677. 1
1678. 1
1679. 1
1680. 1
1681. 1
1682. 1
1683. 1
1684. 1
1685. 0
1686. 1
1687. 1
1688. 1
1689. 1
1690. 0
1691. 1
1692. 1
1693. 1
1694. 1
1695. 1
1696. 1
1697. 1
1698. 1
1699. 1
1700. 1
1701. 1
1702. 1
1703. 0
1704. 1
1705. 0
1706. 1
1707. 1
1708. 1
1709. 1

1710. 1
1711. 1
1712. 1
1713. 0
1714. 0
1715. 1
1716. 1
1717. 1
1718. 0
1719. 1
1720. 1
1721. 1
1722. 1
1723. 1
1724. 1
1725. 1
1726. 1
1727. 1
1728. 0
1729. 1
1730. 1
1731. 1
1732. 1
1733. 0
1734. 0
1735. 1
1736. 1
1737. 1
1738. 1
1739. 1
1740. 1
1741. 0
1742. 1
1743. 1
1744. 0
1745. 1
1746. 1
1747. 1
1748. 1
1749. 1
1750. 1
1751. 1
1752. 1
1753. 0
1754. 0
1755. 0
1756. 1
1757. 1
1758. 1
1759. 1
1760. 0
1761. 1
1762. 1
1763. 1
1764. 1
1765. 0
1766. 1
1767. 1
1768. 0
1769. 1
1770. 1
1771. 1
1772. 1
1773. 1
1774. 1
1775. 0
1776. 1
1777. 1
1778. 0
1779. 1
1780. 1
1781. 1
1782. 1
1783. 1
1784. 1
1785. 1
1786. 1

```
1787. 1
1788. 1
1789. 1
1790. 1
1791. 1
1792. 0
1793. 1
1794. 1
1795. 1
1796. 0
1797. 1
1798. 1
1799. 0
1800. 1
```

In [179]:

```
print("value 1 - " + str(prediction1.sum()))
print("value 0 - " + str( 1800 - int(prediction1.sum())) )
```

```
value 1 - 1485
value 0 - 315
```