# Credit Card Churn Prediction Based on Spending Behaviors

### 0. Setup Google Drive Environment and Get Data

```
!pip install -U -q PyDrive

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
file = drive.CreateFile({'id':'1NWnmneo12ApYRv18uBYtFsNNbSDVWiz6'}) # replace the id with id of file you want to access
file.GetContentFile('data.csv')
```

```
import pandas as pd
import numpy as np

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

| | CLIENTNUM | Attrition_Flag | Customer_Age | Gender | Dependent_count | Education_Level | Marital_Statu |
|---|---|---|---|---|---|---|---|
| **0** | 768805383 | Existing Customer | 45 | M | 3 | High School | Marrie |
| **1** | 818770008 | Existing Customer | 49 | F | 5 | Graduate | Singl |
| **2** | 713982108 | Existing Customer | 51 | M | 3 | Graduate | Marrie |
| **3** | 769911858 | Existing Customer | 40 | F | 4 | High School | Unknow |
| **4** | 709106358 | Existing Customer | 40 | M | 3 | Uneducated | Marrie |

5 rows × 23 columns

### 1. Data Overview

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 23 columns):
 #   Column                                                                                                              Non-Nul
---  ------                                                                                                              -------
 0   CLIENTNUM                                                                                                           10127 n
 1   Attrition_Flag                                                                                                      10127 n
 2   Customer_Age                                                                                                        10127 n
 3   Gender                                                                                                              10127 n
 4   Dependent_count                                                                                                     10127 n
 5   Education_Level                                                                                                     10127 n
 6   Marital_Status                                                                                                      10127 n
 7   Income_Category                                                                                                     10127 n
 8   Card_Category                                                                                                       10127 n
 9   Months_on_book                                                                                                      10127 n
 10  Total_Relationship_Count                                                                                            10127 n
 11  Months_Inactive_12_mon                                                                                              10127 n
 12  Contacts_Count_12_mon                                                                                               10127 n
 13  Credit_Limit                                                                                                        10127 n
 14  Total_Revolving_Bal                                                                                                 10127 n
 15  Avg_Open_To_Buy                                                                                                     10127 n
 16  Total_Amt_Chng_Q4_Q1                                                                                                10127 n
 17  Total_Trans_Amt                                                                                                     10127 n
 18  Total_Trans_Ct                                                                                                      10127 n
 19  Total_Ct_Chng_Q4_Q1                                                                                                 10127 n
 20  Avg_Utilization_Ratio                                                                                               10127 n
 21  Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1  10127 n
```

```
 22  Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2  10127 n
dtypes: float64(7), int64(10), object(6)
memory usage: 1.8+ MB
```

```
df.nunique()
```

```
CLIENTNUM                                                                                                                     10127
Attrition_Flag                                                                                                                    2
Customer_Age                                                                                                                     45
Gender                                                                                                                            2
Dependent_count                                                                                                                   6
Education_Level                                                                                                                   7
Marital_Status                                                                                                                    4
Income_Category                                                                                                                   6
Card_Category                                                                                                                     4
Months_on_book                                                                                                                   44
Total_Relationship_Count                                                                                                          6
Months_Inactive_12_mon                                                                                                            7
Contacts_Count_12_mon                                                                                                             7
Credit_Limit                                                                                                                   6205
Total_Revolving_Bal                                                                                                            1974
Avg_Open_To_Buy                                                                                                                6813
Total_Amt_Chng_Q4_Q1                                                                                                           1158
Total_Trans_Amt                                                                                                                5033
Total_Trans_Ct                                                                                                                  126
Total_Ct_Chng_Q4_Q1                                                                                                             830
Avg_Utilization_Ratio                                                                                                           964
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1    1704
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2     640
dtype: int64
```

As shown above, this data set has 10127 rows and 22 columns from 4 categories:

**(1).Demographic Information**

CLIENTNUM: Unique identifier for each customer.

Customer_Age: Age of customer.

Gender: Gender of customer.

Dependent_count: Number of dependents that customer has.

Education_Level: Education level of customer.

Marital_Status: Marital status of customer.

Income_Category: Income category of customer.

**(2).Relationship with Card Provider**

Card_Category: Type of card held by customer.

Months_on_book: How long customer has been on the books.

Total_Relationship_Count: Total number of relationships customer has with the credit card provider.

Months_Inactive_12_mon: Number of months customer has been inactive in the last twelve months.

Contacts_Count_12_mon: Number of contacts customer has had in the last twelve months.

Credit_Limit: Credit limit of customer.

**(3).Spending Behavior**

Total_Revolving_Bal: Total revolving balance of customer.

Avg_Open_To_Buy: Average open to buy ratio of customer.

Total_Amt_Chng_Q4_Q1: Total amount changed from quarter 4 to quarter 1.

Total_Trans_Amt: Total transaction amount.

Total_Trans_Ct: Total transaction count.

Total_Ct_Chng_Q4_Q1: Total count changed from quarter 4 to quarter 1.

Avg_Utilization_Ratio: Average utilization ratio of customer.

**(4).Information for Prediction**

Attrition_Flag: Flag indicating whether or not the customer has churned out.

Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon:

Naive Bayes classifier for predicting whether or not someone will churn

**2. Spending Behavior Preprocessing**

Pick out spending behavior features from data

```
df_sp = df[['Total_Revolving_Bal', 'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt', 'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg
df_sp.head()
```

| | Total_Revolving_Bal | Avg_Open_To_Buy | Total_Amt_Chng_Q4_Q1 | Total_Trans_Amt | Total_Trans_Ct | Tot |
|---|---|---|---|---|---|---|
| **0** | 777 | 11914.0 | 1.335 | 1144 | 42 | |
| **1** | 864 | 7392.0 | 1.541 | 1291 | 33 | |
| **2** | 0 | 3418.0 | 2.594 | 1887 | 20 | |
| **3** | 2517 | 796.0 | 1.405 | 1171 | 20 | |
| **4** | 0 | 4716.0 | 2.175 | 816 | 28 | |

```
y = df['Attrition_Flag']
y.replace({'Existing Customer': 0, 'Attrited Customer': 1}, inplace=True)
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: Attrition_Flag, dtype: int64
```

We can see that features in spending behaviors are all described by numeric values. Let's take a look at them.

```
df_sp.isnull().sum()
```

```
Total_Revolving_Bal      0
Avg_Open_To_Buy          0
Total_Amt_Chng_Q4_Q1     0
Total_Trans_Amt          0
Total_Trans_Ct           0
Total_Ct_Chng_Q4_Q1      0
Avg_Utilization_Ratio    0
dtype: int64
```

```
df_sp.describe()
```

| | Total_Revolving_Bal | Avg_Open_To_Buy | Total_Amt_Chng_Q4_Q1 | Total_Trans_Amt | Total_Trans_Ct |
|---|---|---|---|---|---|
| **count** | 10127.000000 | 10127.000000 | 10127.000000 | 10127.000000 | 10127.000000 |
| **mean** | 1162.814061 | 7469.139637 | 0.759941 | 4404.086304 | 64.858695 |
| **std** | 814.987335 | 9090.685324 | 0.219207 | 3397.129254 | 23.472570 |
| **min** | 0.000000 | 3.000000 | 0.000000 | 510.000000 | 10.000000 |
| **25%** | 359.000000 | 1324.500000 | 0.631000 | 2155.500000 | 45.000000 |
| **50%** | 1276.000000 | 3474.000000 | 0.736000 | 3899.000000 | 67.000000 |
| **75%** | 1784.000000 | 9859.000000 | 0.859000 | 4741.000000 | 81.000000 |
| **max** | 2517.000000 | 34516.000000 | 3.397000 | 18484.000000 | 139.000000 |

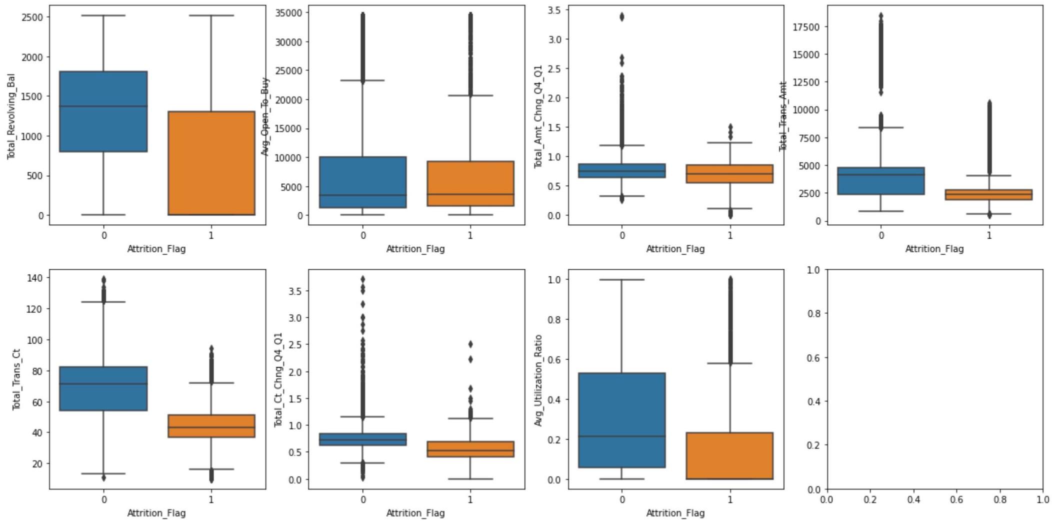
```
import matplotlib.pyplot as plt
import seaborn as sns

df_plt = df_sp.join(y)
df_plt.head()
```

| | Total_Revolving_Bal | Avg_Open_To_Buy | Total_Amt_Chng_Q4_Q1 | Total_Trans_Amt | Total_Trans_Ct | Tot |
|---|---|---|---|---|---|---|
| **0** | 777 | 11914.0 | 1.335 | 1144 | 42 | |
| **1** | 864 | 7392.0 | 1.541 | 1291 | 33 | |
| **2** | 0 | 3418.0 | 2.594 | 1887 | 20 | |
| **3** | 2517 | 796.0 | 1.405 | 1171 | 20 | |
| **4** | 0 | 4716.0 | 2.175 | 816 | 28 | |

```
# boxplot
_,axss = plt.subplots(2,4, figsize=[20,10])
sns.boxplot(x='Attrition_Flag', y ='Total_Revolving_Bal', data=df_plt, ax=axss[0][0])
sns.boxplot(x='Attrition_Flag', y ='Avg_Open_To_Buy', data=df_plt, ax=axss[0][1])
sns.boxplot(x='Attrition_Flag', y ='Total_Amt_Chng_Q4_Q1', data=df_plt, ax=axss[0][2])
sns.boxplot(x='Attrition_Flag', y ='Total_Trans_Amt', data=df_plt, ax=axss[0][3])
sns.boxplot(x='Attrition_Flag', y ='Total_Trans_Ct', data=df_plt, ax=axss[1][0])
sns.boxplot(x='Attrition_Flag', y ='Total_Ct_Chng_Q4_Q1', data=df_plt, ax=axss[1][1])
sns.boxplot(x='Attrition_Flag', y ='Avg_Utilization_Ratio', data=df_plt, ax=axss[1][2])
```

```
<AxesSubplot:xlabel='Attrition_Flag', ylabel='Avg_Utilization_Ratio'>
```




We can see that there are a lot of outliers in the data. Let's drop all the outliers and fill them with the median value of corresponding features.

```
def iqr_outlier_rm(dt_input):
    lq,uq=np.percentile(dt_input,[25,75])
    lower_l=lq - 1.5*(uq-lq)
    upper_l=uq + 1.5*(uq-lq)
    return dt_input[(dt_input >= lower_l) & (dt_input <= upper_l)]

df_sp_ws = iqr_outlier_rm(df_sp)
#df_plt_ws.dropna(axis = 0, how = 'any', inplace = True)
df_sp_ws.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Total_Revolving_Bal   10127 non-null  int64
 1   Avg_Open_To_Buy       5662 non-null   float64
 2   Total_Amt_Chng_Q4_Q1  10127 non-null  float64
 3   Total_Trans_Amt       6470 non-null   float64
 4   Total_Trans_Ct        10127 non-null  int64
 5   Total_Ct_Chng_Q4_Q1   10127 non-null  float64
 6   Avg_Utilization_Ratio 10127 non-null  float64
dtypes: float64(5), int64(2)
memory usage: 553.9 KB
```

```
df_sp_ws['Avg_Open_To_Buy'].fillna(df_sp_ws['Avg_Open_To_Buy'].median(), inplace = True)
df_sp_ws['Total_Trans_Amt'].fillna(df_sp_ws['Avg_Open_To_Buy'].median(), inplace = True)
df_sp_ws.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Total_Revolving_Bal   10127 non-null  int64
 1   Avg_Open_To_Buy       10127 non-null  float64
 2   Total_Amt_Chng_Q4_Q1  10127 non-null  float64
```

```
 3   Total_Trans_Amt         10127 non-null  float64
 4   Total_Trans_Ct          10127 non-null  int64
 5   Total_Ct_Chng_Q4_Q1     10127 non-null  float64
 6   Avg_Utilization_Ratio   10127 non-null  float64
dtypes: float64(5), int64(2)
memory usage: 553.9 KB
```

```
df_sp_ws.describe()
```

| | Total_Revolving_Bal | Avg_Open_To_Buy | Total_Amt_Chng_Q4_Q1 | Total_Trans_Amt | Total_Trans_Ct |
|---|---|---|---|---|---|
| **count** | 10127.000000 | 10127.000000 | 10127.000000 | 10127.000000 | 10127.000000 |
| **mean** | 1162.814061 | 1585.645463 | 0.759941 | 2237.537286 | 64.858695 |
| **std** | 814.987335 | 840.005231 | 0.219207 | 1050.033500 | 23.472570 |
| **min** | 0.000000 | 3.000000 | 0.000000 | 510.000000 | 10.000000 |
| **25%** | 359.000000 | 1324.500000 | 0.631000 | 1438.300000 | 45.000000 |
| **50%** | 1276.000000 | 1438.300000 | 0.736000 | 1658.000000 | 67.000000 |
| **75%** | 1784.000000 | 1521.500000 | 0.859000 | 3050.000000 | 81.000000 |
| **max** | 2517.000000 | 4391.000000 | 3.397000 | 4391.000000 | 139.000000 |

### 3. Model Training and Evaluation

### (1) Preparation

Split data to train set and test set

```
from sklearn import model_selection

x_train, x_test, y_train, y_test = model_selection.train_test_split(df_sp_ws, y, test_size=0.25, stratify = y, random_state=1)
#I want to generate the same report everytime I run this colab
x_train_temp = x_train.copy()
x_train.head()
```

| | Total_Revolving_Bal | Avg_Open_To_Buy | Total_Amt_Chng_Q4_Q1 | Total_Trans_Amt | Total_Trans_Ct |
|---|---|---|---|---|---|
| **9440** | 1583 | 1438.3 | 0.824 | 1438.3 | 112 |
| **959** | 1434 | 4127.0 | 1.423 | 1820.0 | 42 |
| **7737** | 2505 | 111.0 | 0.828 | 2576.0 | 42 |
| **7175** | 0 | 1438.3 | 0.809 | 2600.0 | 44 |
| **5844** | 0 | 1438.3 | 0.622 | 4333.0 | 84 |

Standarlize data sets

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(x_train)
x_train = pd.DataFrame(scaler.transform(x_train))
x_test = pd.DataFrame(scaler.transform(x_test))
x_train.columns = df_sp.columns
x_test.columns = df_sp.columns
x_train.head()
```

| | Total_Revolving_Bal | Avg_Open_To_Buy | Total_Amt_Chng_Q4_Q1 | Total_Trans_Amt | Total_Trans_Ct | Tot |
|---|---|---|---|---|---|---|
| **0** | 0.518958 | -0.167468 | 0.292157 | -0.761188 | 2.014456 | |
| **1** | 0.336068 | 3.037316 | 3.018093 | -0.397718 | -0.974368 | |
| **2** | 1.650670 | -1.749538 | 0.310360 | 0.322175 | -0.974368 | |
| **3** | -1.424100 | -0.167468 | 0.223895 | 0.345029 | -0.888973 | |
| **4** | -1.424100 | -0.167468 | -0.627106 | 1.995260 | 0.818926 | |

### (2)Random Forest model

Train model

```
from sklearn.ensemble import RandomForestClassifier

classifier_RF = RandomForestClassifier()
classifier_RF.fit(x_train, y_train)
classifier_RF.predict(x_test)
classifier_RF.score(x_test, y_test)
```

    0.938783570300158

### Use Grid Search to Find Optimal Hyperparameters Using Cross Validation(CV)

```
from sklearn.model_selection import GridSearchCV

# helper function for printing out grid search results
def print_grid_search_metrics(gs):
        print ("Best score: " + str(gs.best_score_))
        print ("Best parameters set:")
        best_parameters = gs.best_params_
        for param_name in sorted(best_parameters.keys()):
                print(param_name + ':' + str(best_parameters[param_name]))

parameters = {
        'n_estimators' : [60,80,100],
        'max_depth': [1,5,10]
}
Grid_RF = GridSearchCV(RandomForestClassifier(),parameters, cv=5)
Grid_RF.fit(x_train, y_train)
print_grid_search_metrics(Grid_RF)
```

```
Best score: 0.9324555628703095
Best parameters set:
max_depth:10
n_estimators:80
```

```
# best random forest model
best_RF_model = Grid_RF.best_estimator_
best_RF_model.score(x_test, y_test)
```

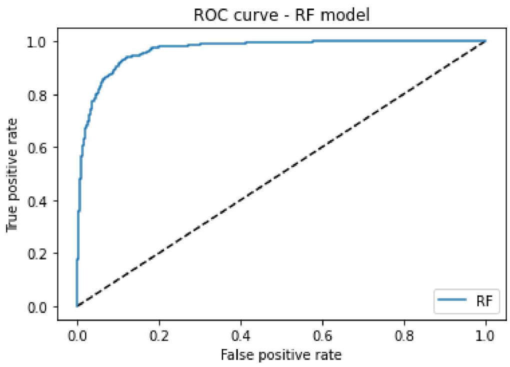
    0.9293048973143759

### ROC & AUC Evaluation of Random Forest Model

```
from sklearn.metrics import roc_curve
from sklearn import metrics

y_pred_rf = best_RF_model.predict_proba(x_test)[:, 1]
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--') # the diagnol
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve - RF model')
plt.legend(loc='best')
plt.show()
```




```
metrics.auc(fpr_rf,tpr_rf)
```

    0.9672276340511635

Feature importance of Random Forest Model

```
importances = best_RF_model.feature_importances_
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature importance ranking by Random Forest Model:")
for ind in range(len(indices)):
    print ("{0} : {1}".format(x_train_temp.columns[indices[ind]],round(importances[indices[ind]], 4)))
```

```
Feature importance ranking by Random Forest Model:
Total_Trans_Ct : 0.2391
Total_Trans_Amt : 0.1989
Total_Ct_Chng_Q4_Q1 : 0.1659
Total_Revolving_Bal : 0.1568
Total_Amt_Chng_Q4_Q1 : 0.1015
Avg_Utilization_Ratio : 0.0986
Avg_Open_To_Buy : 0.0392
```

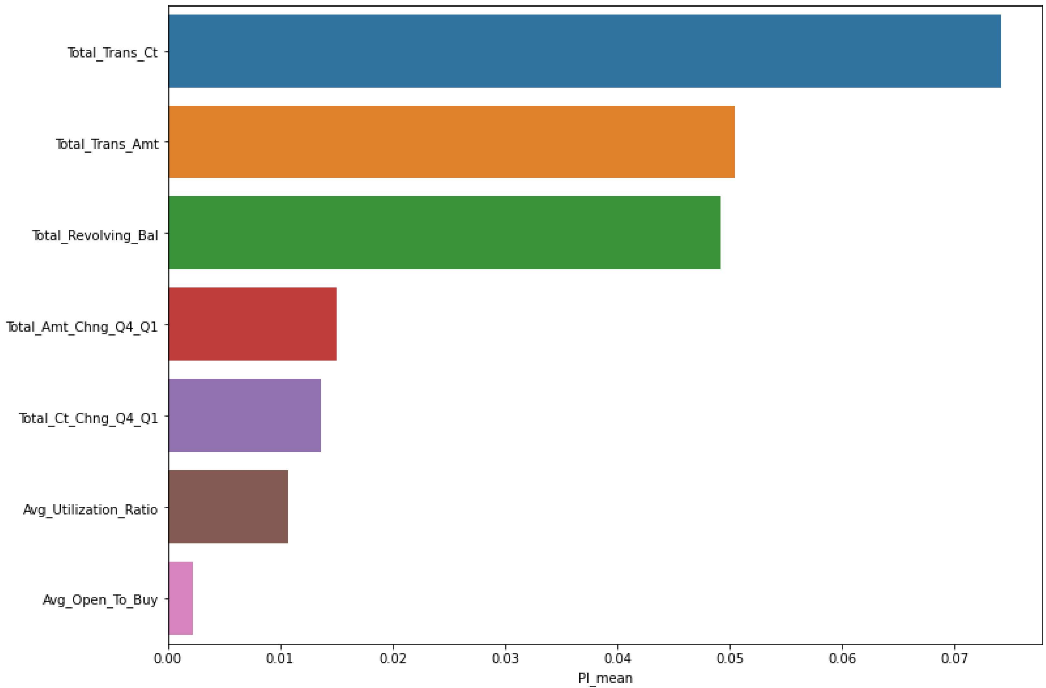
permutation Importance of Random Forest Model

```
from sklearn.inspection import permutation_importance

PI_RF = permutation_importance(best_RF_model, x_test, y_test, n_repeats=5, random_state=1)
PI_res = pd.DataFrame(data=np.transpose([PI_RF['importances_mean'],PI_RF['importances_std']]),
                      index = x_train.columns,columns=['PI_mean','PI_std'])
PI_res = PI_res.sort_values(by='PI_mean',ascending=False)
PI_res
```

| | PI_mean | PI_std |
|---|---|---|
| **Total_Trans_Ct** | 0.074092 | 0.003458 |
| **Total_Trans_Amt** | 0.050474 | 0.002907 |
| **Total_Revolving_Bal** | 0.049131 | 0.006401 |
| **Total_Amt_Chng_Q4_Q1** | 0.015008 | 0.001172 |
| **Total_Ct_Chng_Q4_Q1** | 0.013586 | 0.003743 |
| **Avg_Utilization_Ratio** | 0.010664 | 0.002902 |
| **Avg_Open_To_Buy** | 0.002212 | 0.001134 |

```
plt_PI = sns.barplot(x="PI_mean", y=PI_res.index, data=PI_res)
plt_PI.figure.set_size_inches(12, 9)
```




**(3)K Nearest Neighbor model**

Train model

```
from sklearn.neighbors import KNeighborsClassifier

classifier_KNN = KNeighborsClassifier()
classifier_KNN.fit(x_train, y_train)
classifier_KNN.score(x_test, y_test)
```

    0.9210110584518167

Use Grid Search to Find Optimal Hyperparameters Using Cross Validation(CV)

```
parameters = {
        'n_neighbors':[1,3,5,7,9]
}
Grid_KNN = GridSearchCV(KNeighborsClassifier(),parameters, cv=5)
Grid_KNN.fit(x_train, y_train)
print_grid_search_metrics(Grid_KNN)
```

    Best score: 0.9208689927583936
    Best parameters set:
    n_neighbors:7

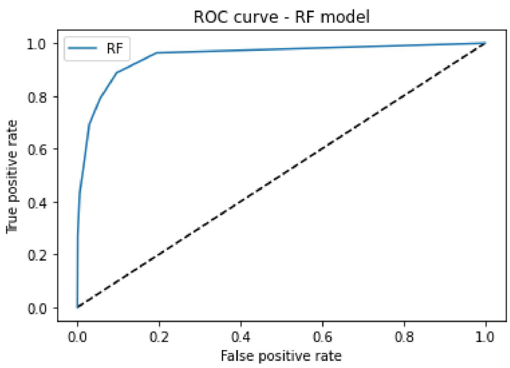
```
best_KNN_model = Grid_KNN.best_estimator_
best_KNN_model.score(x_test, y_test)
```

    0.9261453396524486

ROC & AUC Evaluation of K Nearest Neighbor Model

```
y_pred_rf = best_KNN_model.predict_proba(x_test)[:, 1]
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--') # the diagnol
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve - RF model')
plt.legend(loc='best')
plt.show()
```




```
metrics.auc(fpr_rf,tpr_rf)
```

    0.9502078335019513

**(4)Logistic Regression Model**

Train model

```
from sklearn.linear_model import LogisticRegression

classifier_LR = LogisticRegression()
classifier_LR.fit(x_train, y_train)
classifier_LR.score(x_test, y_test)
```

    0.8783570300157978

Use Grid Search to Find Optimal Hyperparameters Using Cross Validation(CV)

```
parameters = {
        'penalty':('l2','l1'),
        'C':(0.01, 0.05, 0.1, 0.2, 1)
}
Grid_LR = GridSearchCV(LogisticRegression(solver='liblinear'),parameters, cv=5)
Grid_LR.fit(x_train, y_train)
print_grid_search_metrics(Grid_LR)
```

```
Best score: 0.8812376563528638
Best parameters set:
C:0.1
penalty:l1
```

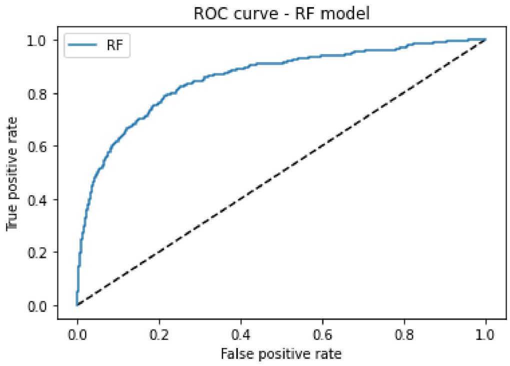
```
best_LR_model = Grid_LR.best_estimator_
best_LR_model.score(x_test, y_test)
```

```
0.8791469194312796
```

ROC & AUC Evaluation of Logistic Regression Model

```
y_pred_rf = best_LR_model.predict_proba(x_test)[:, 1]
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--') # the diagnol
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve - RF model')
plt.legend(loc='best')
plt.show()
```




```
metrics.auc(fpr_rf,tpr_rf)
```

```
0.8555502240208123
```