PROGRAMMING SUPPLMENT(This is just an excerpt. The whole project can be found here-Github Profile Link)

```
import pandas as pd
import os
import numpy as np
import sklearn
import matplotlib.pyplot as plt
from IPython.display import display
import datetime
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import fbeta_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import make_scorer, precision_recall_curve,confusion_matrix
from sklearn import  metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn import linear_model
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import classification_report
from sklearn.model_selection import RandomizedSearchCV
import warnings
warnings.filterwarnings("ignore")
```

PROJECT: PREDICTING CHURNING CUSTOMERS
Objective:
1. To define metrics to identify disengaged customers, which will also include customer churned in that period.
2. Create a ML model to predict customers already churned.
3. Defining a strategy to target customers as soon as we identify them
Methodology:
i. Customer who don't have single transaction in last 4 months.
ii. Customers whose number of transactions are decreasing very month for continuously for 3 months.
iii. Customers whose transaction amount are decreasing very month for continuously for 3 months.
All the customers who qualify for any above metrics is considered as Disengaged customer. It will be a imbalanced class problem so we need to deal with that also if we don't get a good model in first scenarios. Also trying to understand which variables should be used as feature importance. We will also play around with probability cut-offs to increase the recall by compromising on precision. As we want to capture all the users, we will want a model which will have maximum recall. As cost of losing a customer is always higher than sending coupons or cashback offers.

```
groupby1 = trans_analysis.groupby(['user_id','month_year'])
trans_count = groupby1['transaction_id'].count()
amt_sum = groupby1['amount_usd'].sum()
dist_currency = groupby1['transactions_currency'].nunique()
dist_city = groupby1['ea_merchant_city'].nunique()
dist_country = groupby1['ea_merchant_country'].nunique()
dist_trans =  groupby1['transactions_type'].nunique()
trans_sum = pd.DataFrame({'trans_count':trans_count,'amt_sum':amt_sum,'dist_currency':dist_currency,'dist_city':dist_city,
                          'dist_country':dist_country,'dist_trans':dist_trans}).reset_index()
trans1 = trans_sum.set_index(['user_id', 'month_year'])
trans2 = trans1.unstack(level =1)
new_cols = [''.join(t) for t in trans2.columns]
trans2.columns = new_cols
trans2 =trans2.reset_index()
```

```
#Customer at risk or disengaged would have transaction amount going down for 3 months continuously
risky_avg_transaction = []
for idx,row in trans2.iterrows():
    if  (row['amt_sum20191'] > row['amt_sum20192']) & (row['amt_sum20192'] > row['amt_sum20193']) &
    (row['amt_sum20193'] > row['amt_sum20194']):
        risky_avg_transaction.append(idx)
trans2['risky_by_avg_tran'] = np.where(trans2.index.isin(risky_avg_transaction),1,0)

#Customer at risk flag or disengaged for customers whose transaction count is going down month by month
risky_count_transaction = []
for idx,row in trans2.iterrows():
    if  (row['trans_count20191'] > row['trans_count20192']) & (row['trans_count20192'] > row['trans_count20193']) &
    (row['trans_count20193'] > row['trans_count20194']):
        risky_count_transaction.append(idx)
trans2['risky_by_count_tran'] = np.where(trans2.index.isin(risky_count_transaction),1,0)

#Customer at risk flag or disengaged for customers who have no transaction in last 3 months
risky_no_transaction = []
for idx,row in trans2.iterrows():
    if  (row['trans_count20191'] == 0) & (row['trans_count20192'] ==0) & (row['trans_count20193'] == 0) &
    (row['trans_count20194'] == 0) & (row['trans_count201812'] > 0)  :
        risky_no_transaction.append(idx)
trans2['risky_by_no_transaction'] = np.where(trans2.index.isin(risky_no_transaction),1,0)

#Create target varibles for model by combining three flags created before
trans2['target'] = trans2.apply(lambda x: 1 if (x.risky_by_no_transaction==1) or (x.risky_by_count_tran==1) or
                                (x.risky_by_avg_tran == 1)  else 0,axis=1)
```

Summarizing the data by Transaction Month by User ID by creating the following variables at month level – Sum of USD Transactions Amount, Number of Transactions in a month, Number of distinct cities where the transactions were done, Number of distinct merchants with whom the transactions were done, Number of Distinct Transactions.

Creating the 3 Customer Flags → Create Target Variable for model by combining 3 flags created.

```
#Define the variable list to be used in model along with the target variables
model_variables_list = ["user_id","trans_count20181","trans_count201810","trans_count201811","trans_count201812","trans_count2018
model_df =trans2[model_variables_list]
users_data = pd.read_csv(('C:\\Users\\ash\\Desktop\\data_revolut\\data\\rev-users.csv'))
model_df = pd.merge(model_df,users_data,how = 'inner',on = 'user_id')
users_device = pd.read_csv(('C:\\Users\\ash\\Desktop\\data_revolut\\data\\rev-devices.csv'))
model_df = pd.merge(model_df,users_device,how = 'inner',on = 'user_id')

#Feature Engineering
bins = [0,25, 35, 50, 59,60]
labels = ['less_25', '25-35', '35-50', '50-59', '60+']
model_df['agerange'] = pd.cut(2020 - model_df['birth_year'], bins, labels = labels,include_lowest = True)
model_df.drop(['birth_year'],axis = 1, inplace = True)
model_df['days_joined'] = (datetime.date(2018, 10, 31) - pd.DatetimeIndex(model_df['created_date']).date)
model_df['days_joined'] = model_df['days_joined'] / np.timedelta64(1, 'D')
bins = [0,30, 90, 180,999]
labels = ['less_month', 'month_quarter', 'quarter_halfyear', 'more_halfyear']
model_df['joinedrange'] = pd.cut(model_df['days_joined'], bins, labels = labels,include_lowest = True)
model_df.drop(['days_joined','created_date'],axis = 1, inplace = True)
```

Defining variable list and target variable to be used in the model → Feature Engineering

```
#Categorical variables one-hot dummy encoding
cat_variables_list = ['plan','brand','agerange','joinedrange','attributes_notifications_marketing_push','attributes_notifications
model_data = pd.get_dummies(model_df,columns = cat_variables_list)

#Balancing
target1 = len(model_data[model_data['target'] == 1]) * 2
target0 = model_data[model_data.target == 0].index
random_indices = np.random.choice(target0,target1 , replace=False)
target1 = model_data[model_data['target']==1].index
under_sample_indices = np.concatenate([target1,random_indices])
under_sample = model_data.loc[under_sample_indices]
target = under_sample['target']
under_sample.drop(['target'],inplace=True,axis=1)

#Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(under_sample, target, test_size = 0.2, random_state = 0,stratify = target)
```

Categorical Variables One-Hot Dummy encoding → Balancing → Train-Test Split of dataset

```python
# Building a train predictor to compare multiple models using F1-score and Accuracy as metrics.
# More prefrence given to F1-score as it is imbalanced class problem
def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    results = {}
    start = time.time()
    learner = learner.fit(X_train[:sample_size], y_train[:sample_size])
    end = time.time()
    results['train_time'] = end - start
    start = time.time()
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train)
    end = time.time()
    results['pred_time'] = end - start
    results['acc_train'] = accuracy_score(y_train, predictions_train)
    results['acc_test'] = accuracy_score(y_test, predictions_test)
    results['f_train'] = fbeta_score(y_train, predictions_train, 0.5,average='weighted')
    results['f_test'] = fbeta_score(y_test, predictions_test, 0.5,average='weighted')
    matrix =  confusion_matrix(y_test, predictions_test)
    TP =  matrix[0][0]
    FP = matrix[0][1]
    TN = matrix[1][1]
    FN = matrix[1][0]
    accuracy = metrics.accuracy_score(y_test, predictions_test)
    recall = float(TP )/  float(TP + FN)
    precision = float(TP )/ float(TP + FP)
    beta = 0.5
    fscore = (1 + 0.25) * (precision * recall) / ((0.25 * precision) + recall)
    dataframe = pd.DataFrame(predictions_train,y_train)
    dataframe.to_csv(learner.__class__.__name__+".txt")
    return results
```

Building a Train Predictor to compare multiple models using F1-score and Accuracy as metrics. More preference given to F1-score as it is an imbalanced class problem.

```python
#Trying out various classification models to pick the best for tuning
clf_A =  linear_model.LogisticRegressionCV(solver='lbfgs',random_state=40)
clf_B = KNeighborsClassifier()
clf_C = RandomForestClassifier(random_state=40)
samples_100 = len(y_train)
samples_10 = int(samples_100 * 0.1)
samples_1 = int(samples_100 * 0.01)
results = {}
for clf in [clf_A, clf_B, clf_C]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_1,samples_10,samples_100]):
        results[clf_name][i]=train_predict(clf, samples, X_train, y_train, X_test, y_test)
```

```
Test fscore with beta 0.5 :  0.8430097538318626
LogisticRegressionCV trained on 24 samples.
Test fscore with beta 0.5 :  0.7440039158100832
LogisticRegressionCV trained on 240 samples.
Test fscore with beta 0.5 :  0.8735632183908046
LogisticRegressionCV trained on 2409 samples.
Test fscore with beta 0.5 :  0.9037238873751136
KNeighborsClassifier trained on 24 samples.
Test fscore with beta 0.5 :  0.7414634146341463
KNeighborsClassifier trained on 240 samples.
Test fscore with beta 0.5 :  0.7898516036381044
KNeighborsClassifier trained on 2409 samples.
Test fscore with beta 0.5 :  0.8372310570626753
RandomForestClassifier trained on 24 samples.
Test fscore with beta 0.5 :  0.7041540020263424
RandomForestClassifier trained on 240 samples.
Test fscore with beta 0.5 :  0.8353394318728937
RandomForestClassifier trained on 2409 samples.
```

Trying out various classification models to pick the best for tuning → Random Forest and KNN are the best models, but the former will be used as it is faster and scalable on Big Data.

```python
#HyperParameter Tuning using RandomizedSearchCV
n_estimators = [200,400,600,800,1000]
max_features = ['auto', 'sqrt']
max_depth = [4,5,6,7,8,9,10]
min_samples_split = [2, 5, 10,20,50]
min_samples_leaf = [4,8,12,20,40]
bootstrap = [True, False]
class_weight = ['balanced','balanced_subsample']
random_grid = {'n_estimators': n_estimators,'max_features': max_features,'max_depth': max_depth,'min_samples_split':
               min_samples_split,'min_samples_leaf': min_samples_leaf,'bootstrap': bootstrap,'class_weight':class_weight}
rf =  RandomForestClassifier()
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 500, cv = 2, verbose=2,
                               random_state=42, n_jobs = -1)
rf_random.fit(X_train, y_train)
print (rf_random.best_params_)
best_random = rf_random.best_estimator_
best_predictions = best_random.predict(X_test)
print ("\nOptimized Model\n------")
print(classification_report(y_test, best_predictions))
```

Hyper-Parameter Tuning using RandomizedSearchCV

```
Optimized Model
------
              precision    recall  f1-score   support

           0       0.84      0.57      0.68       402
           1       0.48      0.79      0.60       201

    accuracy                           0.64       603
   macro avg       0.66      0.68      0.64       603
weighted avg       0.72      0.64      0.65       603
```

```python
# Feature Ranking and Visualization
importances = best_random.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf.estimators_],axis=0)
indices = np.argsort(importances)[::-1]
probabilities = clf.predict_proba(X_test)[:, 1]
print("Feature ranking:")
for f in range(X_train.shape[1]):
    print("%d. feature %s (%f)" % (f + 1, X_train.columns[indices[f]], importances[indices[f]]))
plt.figure(figsize=(12,8))
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X_train.shape[1]), indices)
plt.xlim([-1, X_train.shape[1]])
plt.xticks(rotation='vertical')
plt.show()
```

Feature Ranking and Visualization

```
Feature ranking:
1. feature amt_sum201812 (0.121308)
2. feature trans_count201812 (0.101498)
3. feature dist_trans201812 (0.098876)
4. feature dist_currency201812 (0.096600)
5. feature num_contacts (0.039067)
6. feature dist_city201812 (0.029899)
7. feature dist_country201812 (0.029831)
8. feature amt_sum201811 (0.025613)
9. feature amt_sum201810 (0.021535)
10. feature trans_count201811 (0.021101)
11. feature amt_sum20189 (0.017929)
12. feature amt_sum20188 (0.017550)
13. feature trans_count201810 (0.017325)
14. feature trans_count20189 (0.015397)
15. feature dist_city201811 (0.014307)
```