STARBUCKS CAPSTONE PROJECT REPORT

Introduction

This project aims to fulfill the requirement to finalize the nanodegree Machine Learning Engineer at Udacity and also to build a decision making tool based on Machine learning algorithms fed with data containing offers, customer demographics and transactions provided by Starbucks.

DOMAIN BACKGROUND

Starbucks is a world reference in the coffee business that is known by almost everyone. They have created a data set that simulates customer activity in their app. This information should help Starbucks to optimize the interaction with its customers, optimize marketing campaigns and increase sales.

I consider this project to be very important because it proposes finding data-based solutions to some questions that many retail businesses are interested in solving, such as: How do I get my offers to the right destination? Can I predict if an offer will become a purchase?

PROBLEM STATEMENT.

The objective of this project is to create a model, based on demographic, offers and transactional information that allows Starbucks to predict whether or not a customer belonging to a demographic group will complete or not a received offer.

DATASETS

The data set has been provided by Starbucks and contains three data domains:

- Descriptive information on the offers created and sent to customers
  file: **portfolio.json**
  information details: Basically offer metadata. Include type of offer and also reward, duration and difficulty.

- Customer demographics. Information of all customers that use the app
  file: **profile.json**
  information details: age, income, gender, date when customer created an app account.

- Transactional information on customer activity. This includes offers received, offers viewed and offers completed
file: **transcript.json**
information details: Event type (offer received, offer viewed, etc) , customer id, amount and time in hours since start of test

Since the objective is to create a binary classification model that allows Starbucks to predict whether or not a customer belonging to a demographic group will complete or not a received offer , it is important to mention that for this type of model to work well, there must be a balance in the number of positive and negative results (in this case, offer completed or not). In this case an exploratory analysis of the data shows that there is a reasonable balance in the number of transactional data for completed and uncompleted offers.

PROJECT DESIGN

The project will be implemented in the following phases

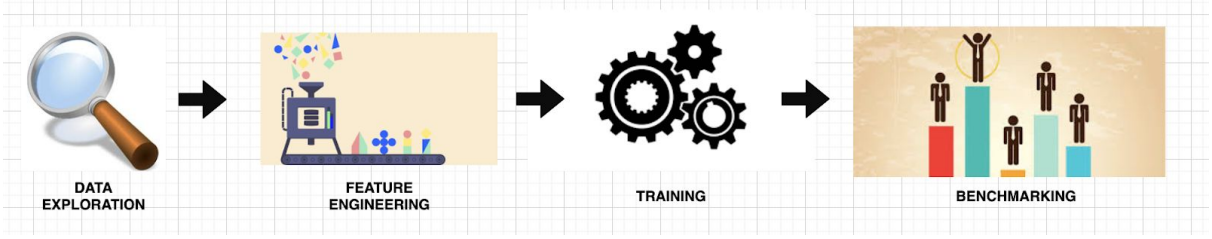
Data exploration: The idea is to understand and become familiar with the data set provided. Generate graphs and determine the correlation of characteristics.

Data preparation and cleaning: Identify and create new data sets suitable for the models, based on the existing ones. The data will also be cleaned

Creation and training of the models:
The models will be created and adjusted with the best hyperparameters.
You will train the models using the data sets.

Benchmarking
The results of all the models are compared and the best one is chosen.

THE PROCESS



Data Exploration

The objective of this phase of the project is to understand the data sets, identify the distribution of the most relevant data, missing data, garbage information, the outliers and which features need to be converted into another representation. All with the intention of identifying clues to create the most appropriate model.

PORTFOLIO DATASET

Contains offer ids and meta data about each offer (duration, type, etc.)

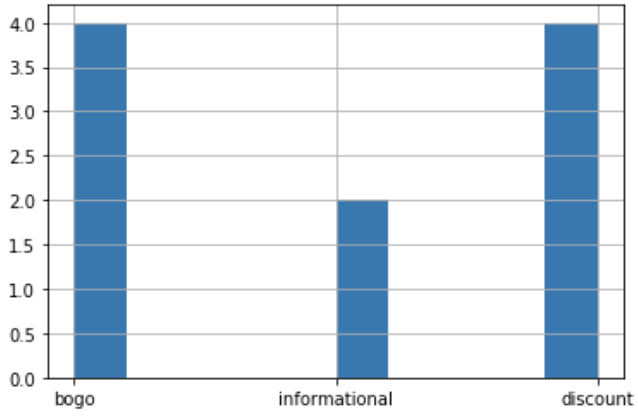
	reward	channels	difficulty	duration	offer_type	id
0	10	[email, mobile, social]	10	7	bogo	ae264e3637204a6fb9bb56bc8210ddfd
1	10	[web, email, mobile, social]	10	5	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0
2	0	[web, email, mobile]	0	4	informational	3f207df678b143eea3cee63160fa8bed
3	5	[web, email, mobile]	5	7	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9
4	5	[web, email]	20	10	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7

This dataset contains only 10 records, (shape (10,6)), and there is no missing values in this dataset:

```
portfolio.isna().sum()
```

```
reward        0
channels      0
difficulty    0
duration      0
offer_type    0
id            0
dtype: int64
```

This is the offer type distribution:



Details of the dataset

The unit of the columns **rewards** and **difficulty** is dollars, the name *difficulty* could be a bit confusing because it conveys the minimum value to be spent to complete the offer, and not how difficult it is to be completed.

There are no missing or negative values on these columns.

It is interesting that **discounts** offers are sent only by email and web, not by the mobile app.

PROFILE DATASET

This dataset contains demographic data for each customer.

The shape of the dataset is (17000, 5)

	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0
4	None	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN

Missing values

There are **2175** values missing in the gender and income columns, and it seems that age is **118** when those two column values are missing. Since the information in these records is inconsistent and of little value to the final dataset from which important business conclusions are expected I will drop those records from the dataset.

```
profile.isna().sum()
```

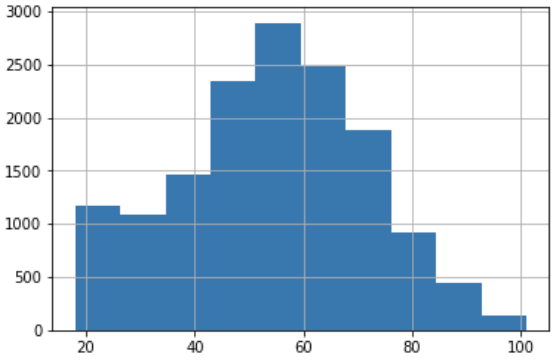
```
gender              2175
age                    0
id                     0
became_member_on       0
income              2175
dtype: int64
```

Data distribution

Age distribution

```
profile['age'].describe()
```

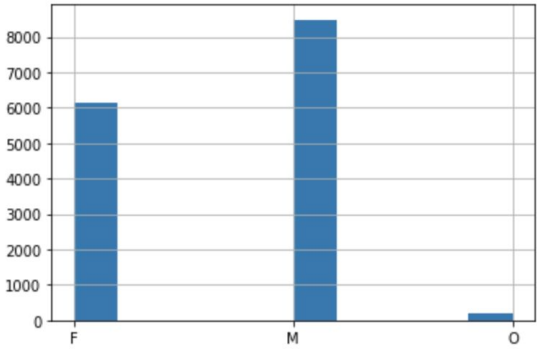
```
count    14825.000000
mean        54.393524
std         17.383705
min         18.000000
25%         42.000000
50%         55.000000
75%         66.000000
max        101.000000
Name: age, dtype: float64
```



Gender distribution

```
profile['gender'].value_counts()
```

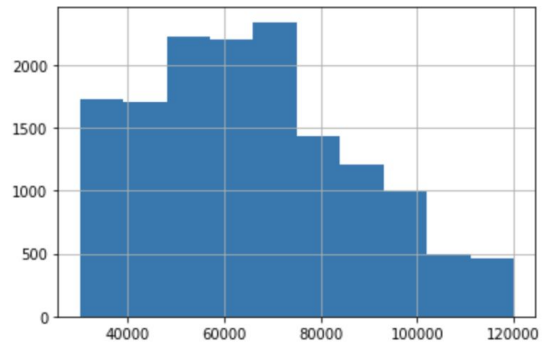
```
M    8484
F    6129
O     212
Name: gender, dtype: int64
```



Income distribution

```
profile['income'].describe()
```

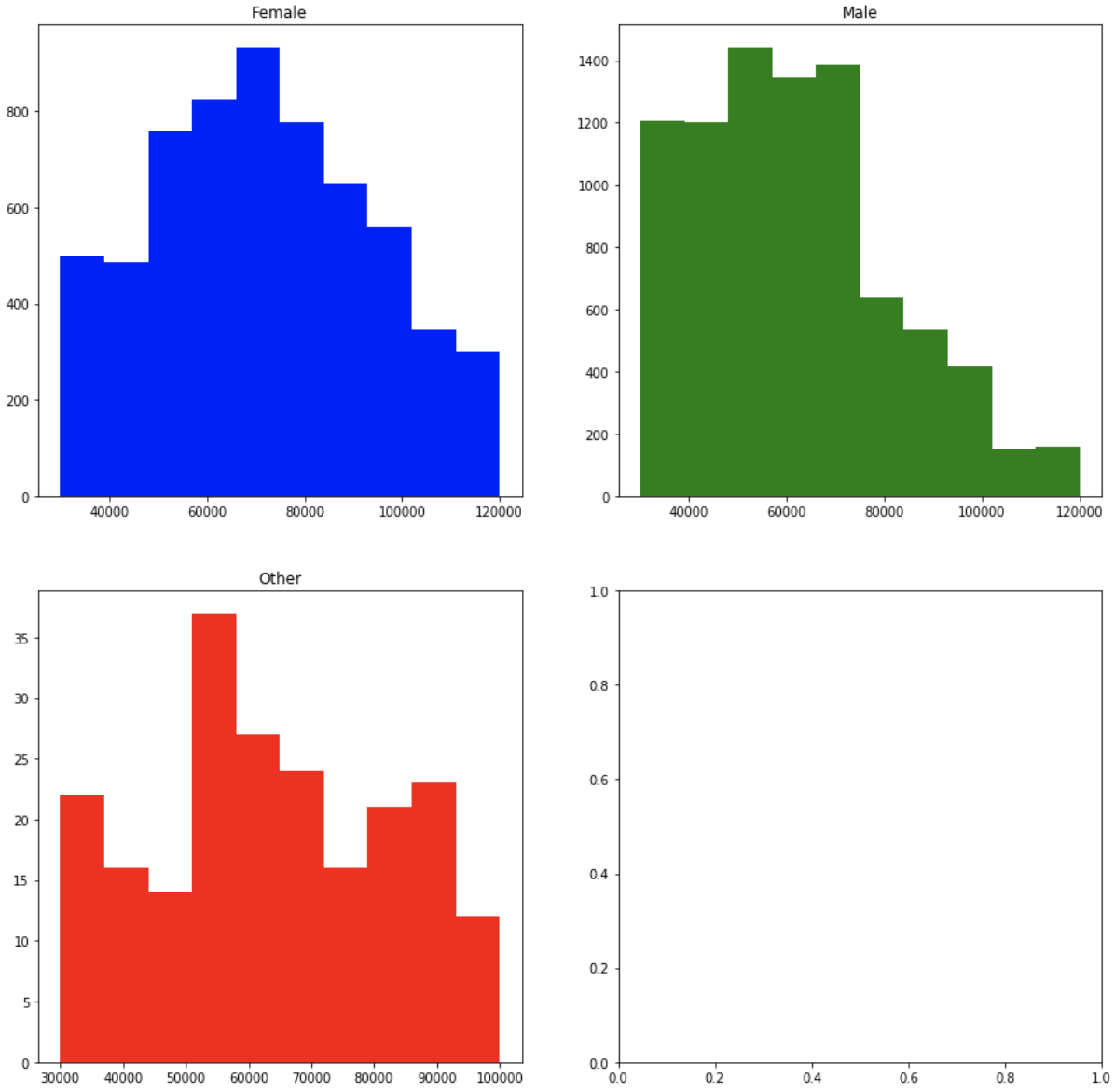
```
count     14825.000000
mean      65404.991568
std       21598.299410
min       30000.000000
25%       49000.000000
50%       64000.000000
75%       80000.000000
max      120000.000000
Name: income, dtype: float64
```



income by gender

```
genders = profile['gender'].unique()
for gender in genders:
    u = profile.loc[profile['gender']==gender,'income'].mean()
    print('{gender} Mean income is: {mean}'.format( gender=gender,mean=u))
```

```
F Mean income is: 71306.41213901126
M Mean income is: 61194.60160301744
O Mean income is: 63287.735849056604
```

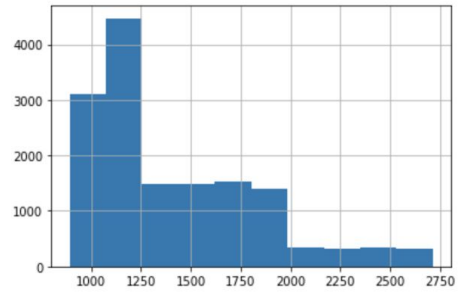


Empirically we could think that income could be a relevant feature for the classification model we are trying to create, however a later formal analysis of the correlation of features of the model will give us the answer. For now it would be a good idea to create a new column where to have the income value scaled.

It is important to analyze the distribution of the **became_member_on** column as this can be an important feature within the model that defines a client and could help predict whether the offer will be completed or not. Before that it is necessary to process this column in order to use a more appropriate format.

```
import datetime
from datetime import datetime as dt

profile['member_since'] = pd.to_datetime(profile['became_member_on'], format = '%Y%m%d')
profile['total_days_of_membership'] = (dt.today().date() - profile['member_since'].dt.date).dt.days
profile['total_days_of_membership'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11e910990>
```



Details of the dataset

The majority of clients are men (8484), followed by women (6129) and only 212 clients define their gender as other.

The average age of the clients is 54 years old. There are **3702** clients over 66 years old, which is **25%**.

The median income is 65400 dollars

Women's average income (71,306) is higher than men's (61,195) and than the average in the data set (65405)

The income range is from 30,000 to 120,000

The average days of membership is 1408 days

THE TRANSCRIPT DATASET

This dataset contains records for transactions, offers received, offers viewed, and offers completed. Its shape is (306534, 4) and there are no missing values.

	person	event	value	time
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
1	a03223e636434f42ac4c3df47e8bac43	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	0
3	8ec6ce2a7e7949b1bf142def7d0e0586	offer received	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	0
4	68617ca6246f4fbc85e91a2a49552598	offer received	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	0

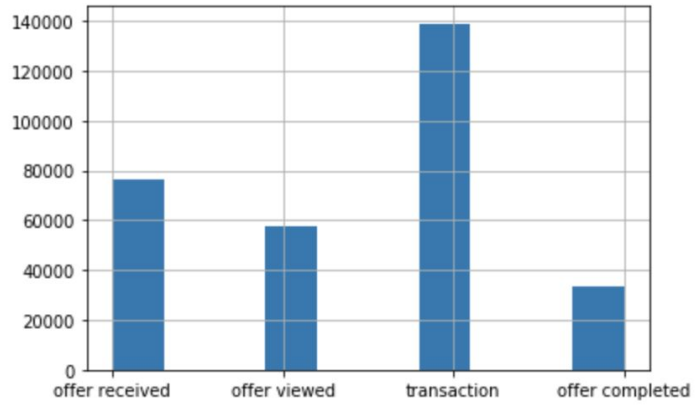
The **value** column is a dictionary, we need to know what keys are used.

```
#lets see what are the keys in transcript.value

for event_name in transcript['event'].unique():
    subset = transcript[transcript['event']==event_name]
    if (type (subset.iloc[0,2]) is dict ):
       print("{}  :  {}".format(event_name,subset.iloc[0,2].keys()))
```

```
offer received  :  dict_keys(['offer id'])
offer viewed  :  dict_keys(['offer id'])
transaction  :  dict_keys(['amount'])
offer completed  :  dict_keys(['offer_id', 'reward'])
```

Data distribution

Event distribution



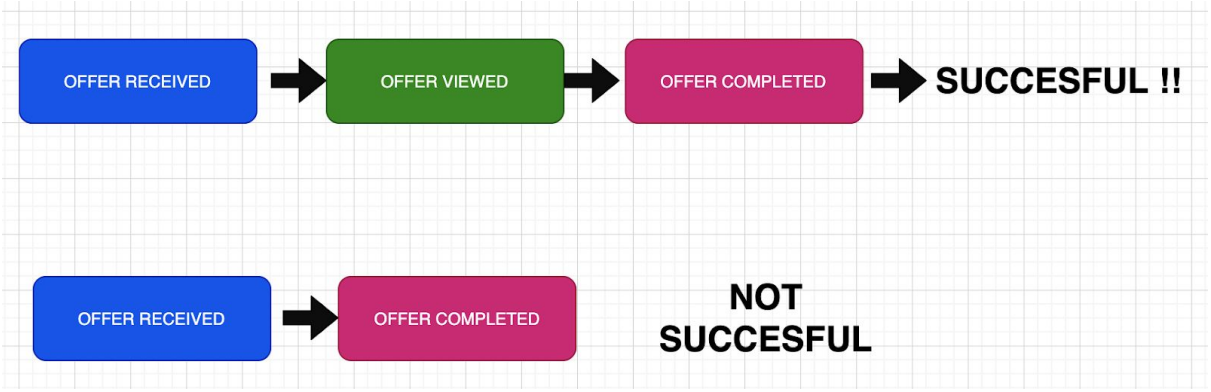
In order to prepare the training dataset, a column will be added to the transcript dataframe for each key that is in the Value column, we already know that they are (amount, reward, offer_id and offer id)

The time feature

The **time** feature indicates the time of occurrence of an event since the beginning of the test. This feature can be important to implement heuristics or to calculate statistics such as the average time for customers to complete an offer. It can also be used to create a regression model with which to predict the customer response time to complete an offer. In the case of this particular project the idea is to create a classification model that predicts whether an offer will be completed or not, we will eliminate this feature from the training data set being created as we consider it not relevant to the model.

How to determine successful offers? a key element in drawing valuable conclusions

A key element in being able to do a powerful data analysis and implement a successful model is being able to determine successful offers, i.e. those that were received, viewed and completed.



The following function has that objective. Here we assume that for an offer to be successful, (for a particular client and offer, to be exact) it must have three successive events in the following **strict** order:

offer received | offer viewed | offer completed

```
#The objective of this function is to mark the offers received that were successfully completed
def determine_offer_completed(person,offer_id,time):

    result = training_df.loc[  (training_df['person']   == person)
                             & (training_df['offer_id'] == offer_id)
                             & (training_df['time']>time), ['event','time']
                            ].sort_values(['time'],ascending=True).head(2)['event']

    completed = (len(result)==2) and (result.iloc[[0][0]]=='offer viewed') and (result.iloc[[1][0]]=='offer completed')
    return (1.0 if completed else 0.0)
```

So the algorithm starts creating a new column called completed, initialized with the value 0.0. It then scans all the offers received and updates this value to 1.0 or 0.0 depending on whether the offer was completed by the customer or not.

```
training_df['completed'] = 0.0

# we will process all the offers received to determine whether or not they were successfully completed

training_df['completed'] = ( training_df.loc[training_df['event']=='offer received']
                            .apply(lambda x: determine_offer_completed(x.person, x.offer_id, x.time), axis=1) )
training_df.fillna(0.0, inplace = True)
```

```
training_df = training_df.loc[training_df['event']=='offer received'].copy()

total_completed = len(training_df.loc[ (training_df['event']=='offer received') & (training_df['completed']==1.0) ])
percentage_completed = (total_completed / len(training_df.loc[ (training_df['event']=='offer received')])) * 100
print('Total offers completed {total}'.format(total=total_completed))
print('Percentage offers Completed {total}'.format(total=percentage_completed))
```

```
Total offers completed 16173
Percentage offers Completed 24.319935038570847
```

After processing the data, this is the result:

Total offers completed 16173
Percentage offers Completed 24.32

It is clear that our dataset has a significant degree of imbalance between completed and uncompleted offers, with completed offers representing only approximately 25% of the total.

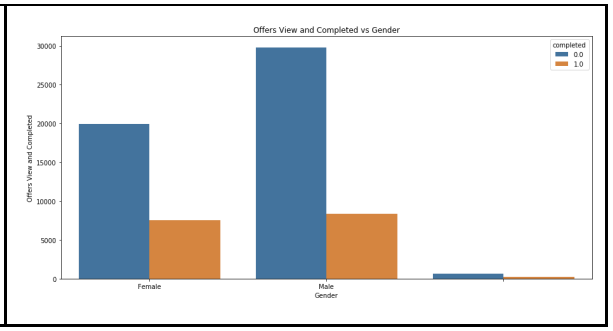
Class imbalance affects **accuracy** negatively, so something must be done so that model performance can still be evaluated using accuracy. This indicates that we should use a strategy to manage imbalanced classes, either by some method of increasing the minority class samples or decreasing the majority class samples or by using a classification model that is little affected by the class imbalance.

Some key data found when analyzing the data

Offers completed by Gender

```
training_df.groupby(['gender','completed'])['completed'].count()
```

```
gender  completed
F       0.0          19904
        1.0           7552
M       0.0          29768
        1.0           8361
O       0.0            656
        1.0            260
Name: completed, dtype: int64
```

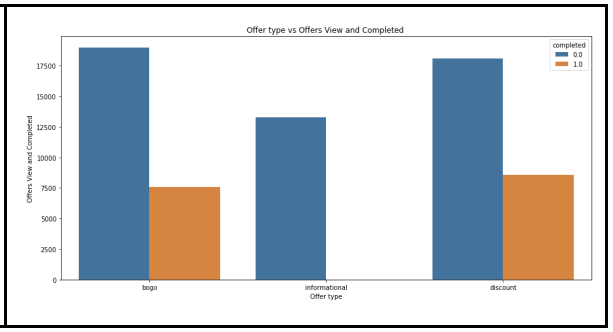


Women complete more offers than men.

What are the types of offers most completed by clients?

```
training_df.groupby(['offer_type','completed'])['completed'].count()
```

```
offer_type     completed
bogo           0.0          18954
               1.0           7583
discount       0.0          18074
               1.0           8590
informational  0.0          13300
Name: completed, dtype: int64
```

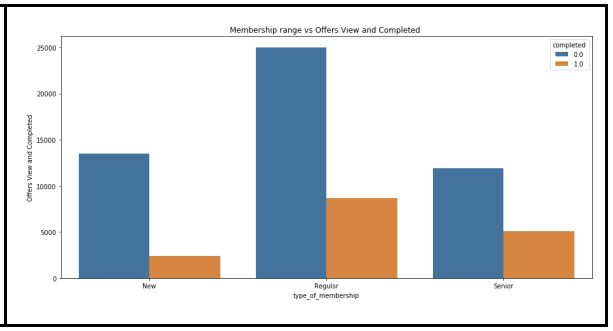


The most successful offer type is discount

How long a customer has been on the platform is related to successful offers?

```
training_df.groupby(['type_of_membership','completed'])['completed'].count()
```

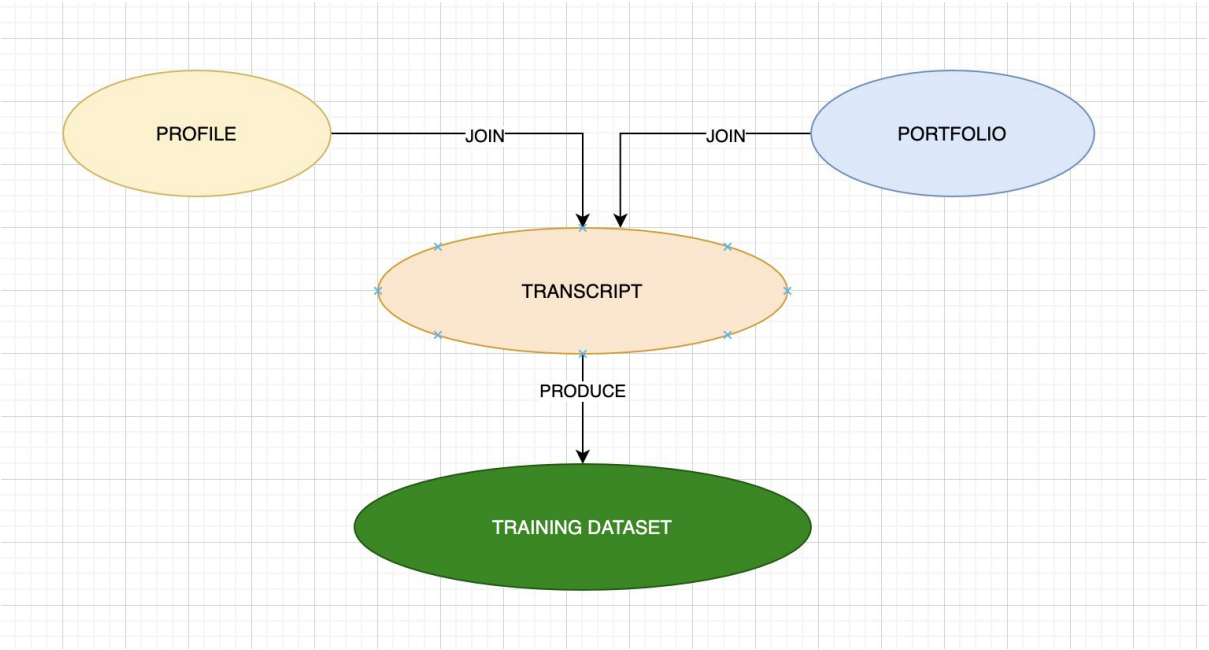
```
type_of_membership  completed
0                   0.0          13472
                    1.0           2414
1                   0.0          24973
                    1.0           8657
2                   0.0          11883
                    1.0           5102
```



The oldest customers on the platform are the ones who complete the most offers

Feature engineering

One of the main steps of this project is to create a data model based on demographic information composed of relevant information from the profile, portfolio and transcript datasets, which is the basis for a classification model with which to predict whether an offering will be successfully completed or not.



The following are the steps to follow:

Join transcript dataset with profile
Join transcript dataset with portfolio
scale income
scale age
one hot encode offer_type
one hot encode gender
create categories for total_days_of_membership
remove all the categorical features (person id, offer id, etc)

As our model is based on working with successful offers, i.e. those received, viewed and completed, ***we will not take into account transaction-type events***

Joining the datasets

```
training_df   =  pd.merge( training_df , profile[['person','gender', 'age','total_days_of_membership','income']],
                                                    on='person', how='right')
training_df = training_df.loc[training_df['event'].isin(['offer viewed','offer completed','offer received'])]
```

```
portfolio =  portfolio.rename(columns={'id':'offer_id'})
portfolio.set_index(['offer_id'])

training_df   =  pd.merge( training_df , portfolio[['offer_id','reward', 'difficulty','duration', 'offer_type']],
                                                    on='offer_id', how='left')
```

Ordinal representation or One Hot Encoding?

For the features **gender** and **offer type** one might consider converting them into an ordinal representation such as 1 = BOGO, 2 = INFORMATIONAL and 3 = DISCOUNT, however this might cause the model to learn that INFORMATIONAL and DISCOUNT are more similar than BOGO and DISCOUNT, since in reality there is no ordinal relationship between them. The same for the GENDER feature. Therefore it is better to make a One Hot Encoder representation for the features gender and offer_type.

```
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder


scaler = preprocessing.MinMaxScaler()
training_df['std_income'] = scaler.fit_transform(training_df[['income']])
training_df['std_age']    = scaler.fit_transform(training_df[['age']])

#one hot encode offer_type
enc = OneHotEncoder(handle_unknown='ignore')
enc_offer = pd.DataFrame(enc.fit_transform(  pd.DataFrame(training_df['offer_type'])  ).toarray())

feature_labels = enc.categories_
feature_labels = np.array(feature_labels).ravel()
enc_offer.columns = feature_labels

#one hot encode gender
enc_g = OneHotEncoder(handle_unknown='ignore')
enc_gender = pd.DataFrame(enc_g.fit_transform(pd.DataFrame(training_df['gender'])).toarray())

feature_labels = enc_g.categories_
feature_labels = np.array(feature_labels).ravel()
enc_gender.columns = feature_labels

training_df = training_df.join(enc_offer)
training_df = training_df.join(enc_gender)
training_df
```

Preprocessing of Total days of membership

Let's create categories for the column **total_days_of_membership** based on the interquartile ranges

1 - 1093 days : 0

1094 - 1683 days : 1

> 1683 : 2

Correlations

Now that we have a complete data set, it is now important to analyze the correlation between the different features.. Knowing, of course, that correlation does not imply causation

Let's first eliminate the categorical columns and the redundant ones after scaling and one hot encoding.

```
training_df = training_df.drop(columns=['person', 'event', 'offer_id', 'gender', 'ag
training_df.columns
```

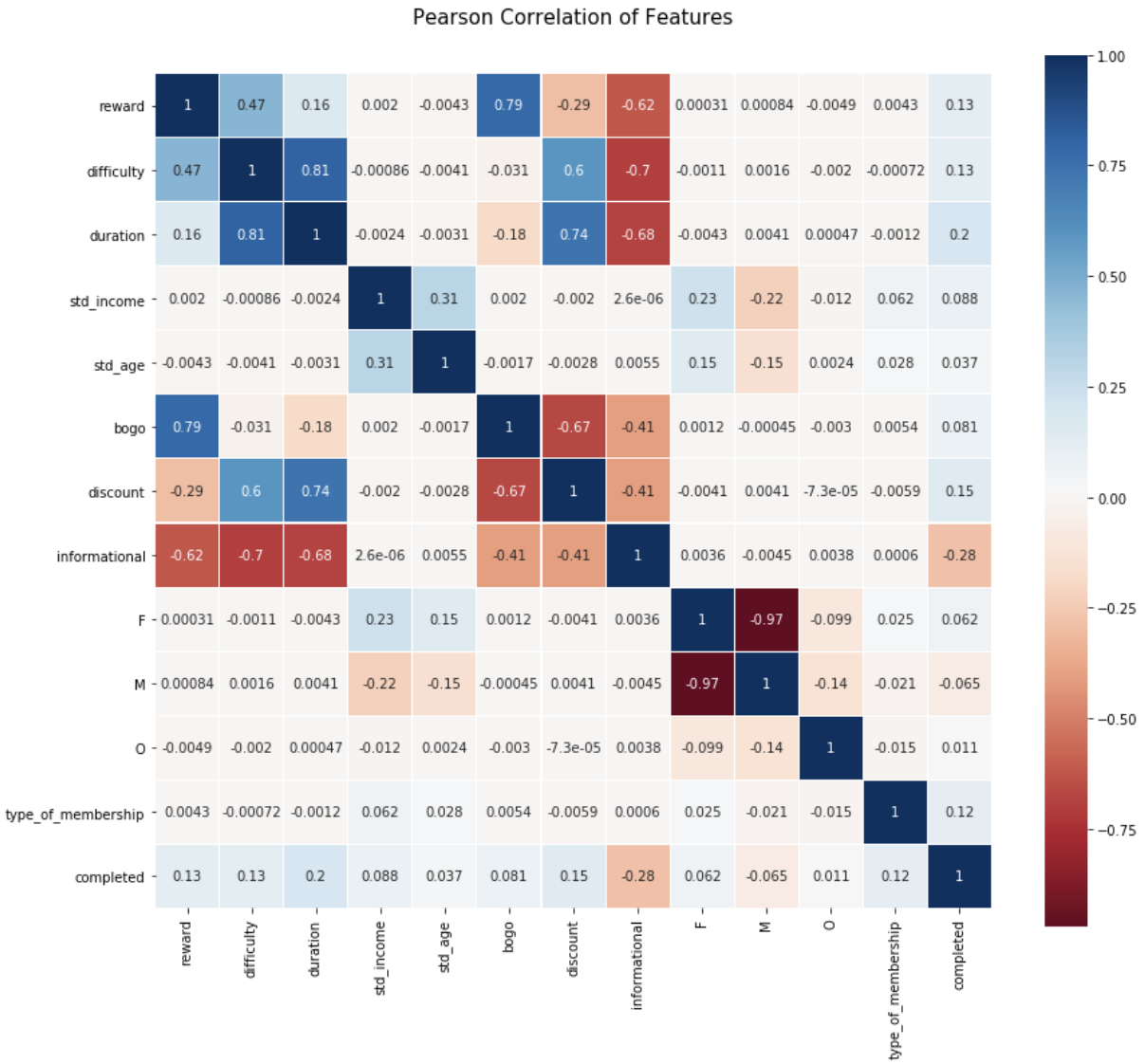
```
Index(['reward', 'difficulty', 'duration', 'std_income', 'std_age', 'bogo',
       'discount', 'informational', 'F', 'M', 'O', 'type_of_membership',
       'completed'],
      dtype='object')
```

And now lets see the correlation

```
correlation_matrix = training_df.corr()
correlation_matrix['completed'].sort_values(ascending=False)
```

```
completed              1.000000
duration               0.196006
discount               0.150572
difficulty             0.129000
reward                 0.125238
type_of_membership     0.120718
std_income             0.088242
bogo                   0.080825
F                      0.062272
std_age                0.036803
O                      0.011196
M                     -0.064629
informational         -0.283437
Name: completed, dtype: float64
```

it can be seen that the features most correlated to a successful offer(completed=1.0) are (in descending order): **duration,discount,difficulty,reward and type of membership (which was derived from total_days_of_membership)**



Dealing with imbalanced classes

It is clear that our model has a significant degree of imbalance between successful and un successful offers, with successful offers representing only approximately 25% of the total.

```
total_completed = len(training_df.loc[ (training_df['completed']==1.0)])
percentage_completed = (total_completed / len(training_df) ) * 100

print('Total offers completed {total}'.format(total=total_completed))
print('Percentage offers Completed {total}'.format(total=percentage_completed))

print('Total offers NO completed {total}'.format(total=len(training_df.loc[ (training_df['completed']==0.0)])))
```

```
Total offers completed 16173
Percentage offers Completed 24.319935038570847
Total offers NO completed 50328
```

Class imbalance affects accuracy negatively, so something must be done so that model performance can still be evaluated using accuracy. In this particular case

we are going to randomly increase the number of completed offer samples using the resample module of scikit learn

```
from sklearn.utils import resample

df_uncompleted = training_df[training_df.completed==0.0] # most of the records are unco
df_completed   = training_df[training_df.completed==1.0] # completed offers

#here we are aumenting the number of samples of completed offers
df_completed_augmented = resample(df_completed, replace=True,n_samples=50328, random_st

training_df_upsampled = pd.concat([df_uncompleted, df_completed_augmented])
training_df_upsampled.completed.value_counts()
```

```
1.0    50328
0.0    50328
Name: completed, dtype: int64
```

```
training_df_upsampled.shape
```

```
(100656, 13)
```

Create a training and test set

Before splitting the data set into training and testing, it is important to look for a method of division that can be more effective than a purely random one. In this case I consider that it would be good that the training and test data set has an equal data proportion based on the column **type_of_membership** (which is derived from the age of the client in the platform) and according to the correlation matrix, it has a moderate positive correlation with the label 'completed'.

This type of division is called stratified division and for this we are going to use **StratifiedShuffleSplit** of scikit learn

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1 , test_size=0.2, random_state= 42 )

for train_index, test_index in split.split(training_df_upsampled,training_df_upsampled['type_of_membership']):
    strat_train_set = training_df_upsampled.iloc[train_index]
    strat_test_set  = training_df_upsampled.iloc[test_index]

print(strat_train_set.shape)
print(strat_test_set.shape)
```

```
(80524, 13)
(20132, 13)
```

```
strat_train_set['type_of_membership'].value_counts()/len(strat_train_set)
```

```
1    0.515250
2    0.277321
0    0.207429
Name: type_of_membership, dtype: float64
```

```
strat_test_set['type_of_membership'].value_counts()/len(strat_test_set)
```

```
1    0.515249
2    0.277320
0    0.207431
Name: type_of_membership, dtype: float64
```

From the previous image we can see that the train and test data sets have the same proportion of “type_of_membership”

```
X_train = strat_train_set.drop(columns=['completed'])
y_train = strat_train_set['completed']
print(X_train.shape)
print(y_train.shape)
X_train.columns
```

```
(80524, 12)
(80524,)

Index(['reward', 'difficulty', 'duration', 'std_income', 'std_age', 'bogo',
       'discount', 'informational', 'F', 'M', 'O', 'type_of_membership'],
      dtype='object')
```

```
X_test = strat_test_set.drop(columns=['completed'])
y_test = strat_test_set['completed']
print(X_test.shape)
print(y_test.shape)
X_test.columns
```

```
(20132, 12)
(20132,)
```

Training

The following algorithms will be used to train and validate different models: RandomForestClassifier , LogisticRegressor, XGBRegressor and KNeighborsClassifier.

Metrics to evaluate our models

AUROC

Intuitively, AUROC represents the likelihood of one classification model to distinguishing observations from two classes. In other words, if you randomly select one observation from each class, AUROC tell the probability that the model will be able to "rank" them correctly

Confusion Matrix

A confounding matrix is an N x N matrix that is used to evaluate the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us an overview of how good our model is and what types of errors it is making.

RandomForestClassifier

Random Forest is a flexible and easy-to-use machine learning algorithm that produces good results even without adjusting hyper parameters. We wanted to use this algorithm also because it is good for handling large data sets with a high dimensionality, although this particular model is not as large.

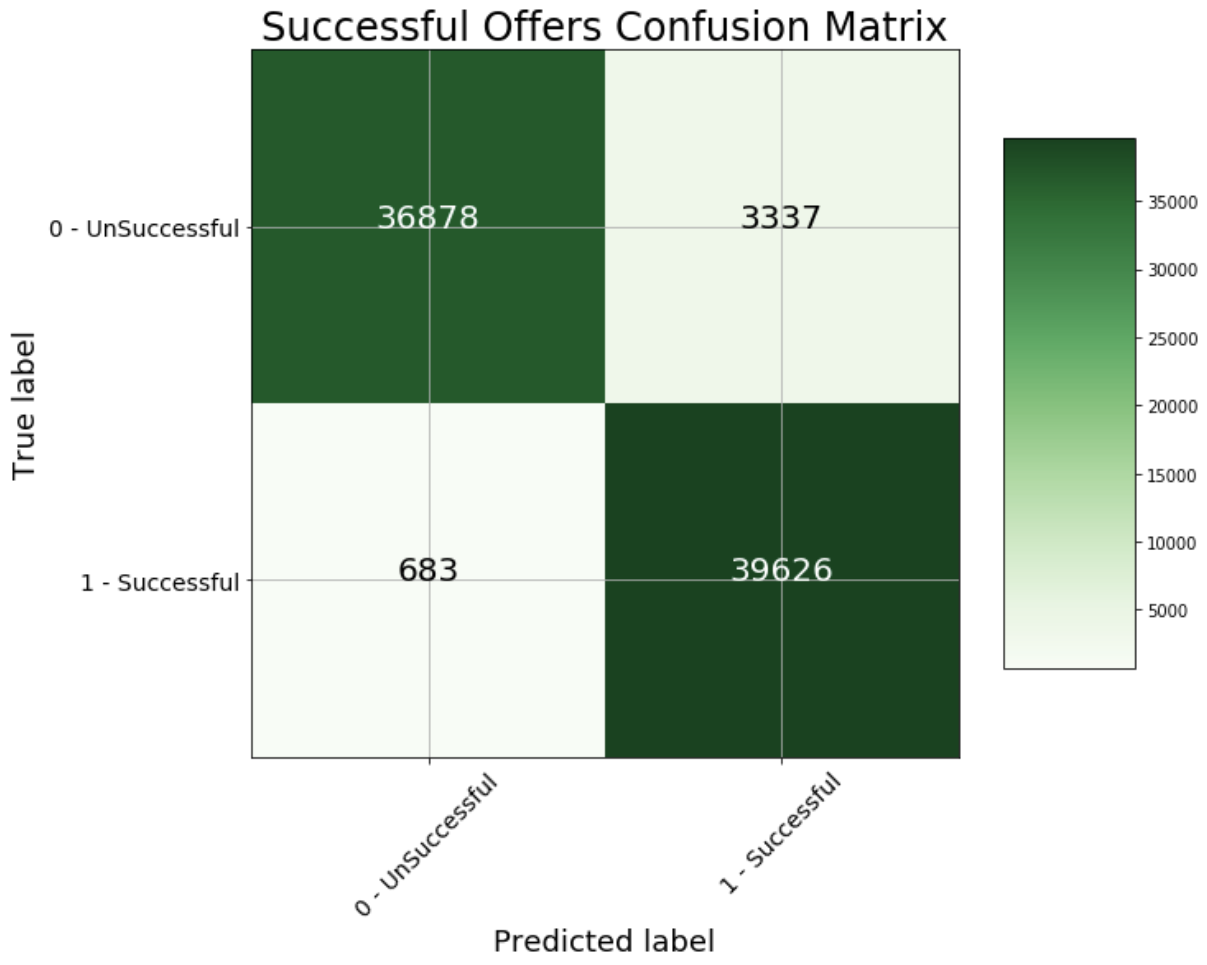
```
seed = 42

rf = RandomForestClassifier(n_estimators = 100, random_state = seed)

rf.fit(X_train, y_train)

# Predict on training set
train_pred = rf.predict(X_train)

print( "RandomForestClassifier Accuracy: " + str( round( accuracy_score( y_train,train_pred)*100,2)

# How many classes are our model predicting?
print( np.unique( train_pred ) )
```

```
RandomForestClassifier Accuracy: 95.01
[0. 1.]
```

The accuracy of this model is 95.01, which is very good. We need to validate that the model is not overfitting the training dataset.

```
train_probs =  rf.predict_proba(X_train)
train_probs = [p[1] for p in train_probs]

print( "AUROC: " + str(round(roc_auc_score(y_train, train_probs)*100,2)) )
```

```
AUROC: 99.0
```



The confusion matrix shows good results, having a slightly high number of false negatives. From a business point of view, we could mistakenly rule out some successful offers.

LogisticRegression

LogisticRegression

```
logistic_regressor = LogisticRegression()

logistic_regressor.fit(X_train, y_train)

train_pred = logistic_regressor.predict(X_train)

print( "LogisticRegression Accuracy:" + str(round(accuracy_score( y_train, train_pred)*100,2)) )

print( np.unique( train_pred ) )
```

```
LogisticRegression Accuracy:67.94
[0. 1.]
```

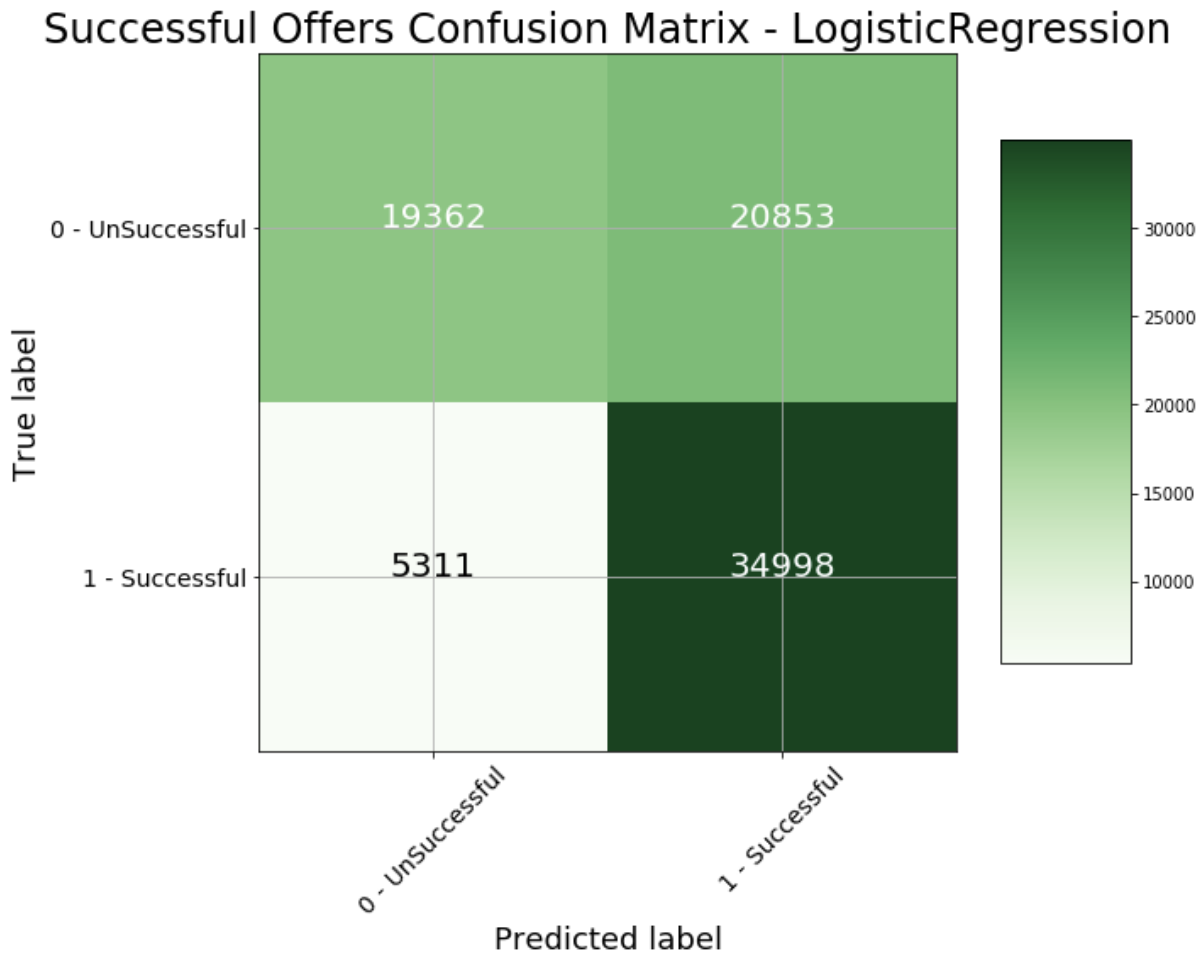
The accuracy of this model is 67.94, which is not very good. Other metrics

AUROC

```
prob_y_4 = logistic_regressor.predict_proba(X_train)
prob_y_4 = [p[1] for p in prob_y_4]

print( "AUROC: " + str(round(roc_auc_score(y_train, prob_y_4)*100,2)) )
```

```
AUROC: 74.51
```

AUROC is 74.51 which is not very good as well.



In general the results of this model are not very good

XGBRegressor

```
xg_reg = xgb.XGBRegressor(objective ='binary:logistic')

train_pred = xg_reg.fit(X_train,y_train)

print(f'Accuracy of Logistic regression classifier on training set: {round(xg_
```

```
[16:28:43] WARNING: /Users/runner/miniforge3/conda-bld/xgboost_1607604592557/w
GBoost 1.3.0, the default evaluation metric used with the objective 'binary:lo
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Accuracy of Logistic regression classifier on training set: 33.95%.
```

The accuracy of this model is approximately 34%, which is very low. Due to this poor performance I will not continue to evaluate other metrics from this model and will discard it.

KNeighborsClassifier

```
knc = KNeighborsClassifier()

knc.fit(X_train, y_train)

# Predict on training set
train_pred = knc.predict(X_train)

print( "KNeighborsClassifier Accuracy:" + str( round( accuracy_score( y_
```

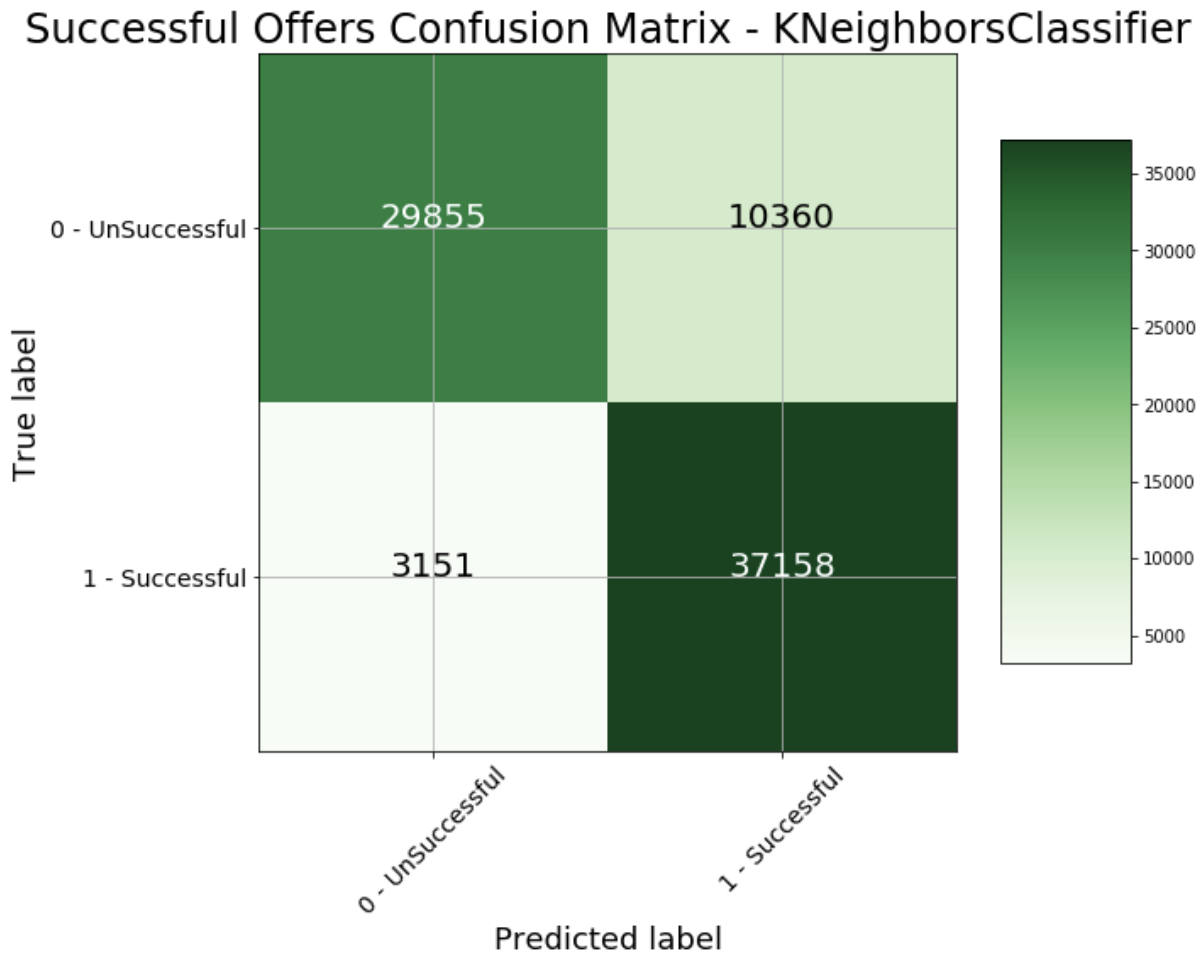
```
KNeighborsClassifier Accuracy:83.22
```

The accuracy of this model is approximately 83%, which is moderately good. Let's see other metrics of this model.

```
prob_y_4 = knc.predict_proba(X_train)
prob_y_4 = [p[1] for p in prob_y_4]

print( "AUROC: " + str(round(roc_auc_score(y_train, prob_y_4)*100,2)) )
```

```
AUROC: 92.45
```

AUROC is 92.45% which is a good value.



However, the confusion matrix does not show such good results.

Which algorithm should be chosen?

Metrics showed that the best performing algorithm was **RandomForestClassifier**, the cause can be found in that decision trees often perform well on classification models because their hierarchical structure allows them to learn signals from both classes. And if we use Tree ensembles like Random Forests the result will be better because the result of a Random Forests almost always outperform singular decision trees.

Let's see the parameters of this model

```
from sklearn.model_selection import RandomizedSearchCV

from pprint import pprint
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

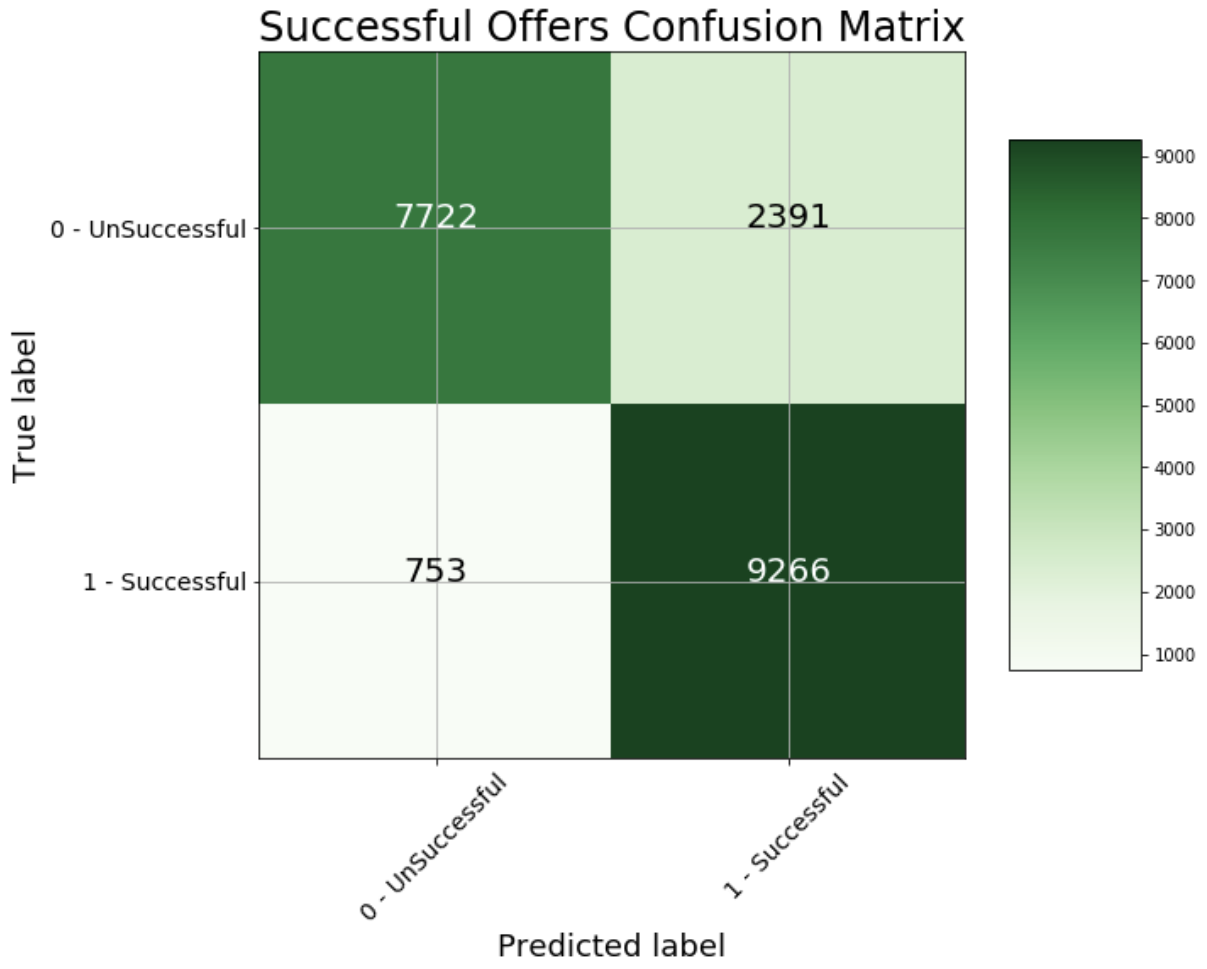
Evaluating the final model on the test set

```
final_predictions = rf.predict(X_test)

print( "Model Accuracy: " + str(round( accuracy_score(y_test,final_predictions)*100,2))
```

```
Model Accuracy: 84.38
```

```
prob_y_4 = rf.predict_proba(X_test)
prob_y_4 = [p[1] for p in prob_y_4]

print( "AUROC: " + str(round(roc_auc_score(y_test, prob_y_4)*100,2)) )
```

```
AUROC: 91.07
```



After validating the model with the training data, it can be seen that the model is generalizing in a reasonable way. An accuracy of 85%, while not the highest, can be a good start for more complex models.

An AUROC of 91.07 is also very good for this model

Conclusion

Working on this capstone project was really useful to be able to put into practice the fundamental concepts of data science and face the challenges of creating a model that is useful to solve a real life problem.

The most challenging part for me, and the one that took the most time, was to create a data set that would allow me to model the objectives stated in the problem statement.

It was also exciting to discover that there was a class imbalance in my data set, i.e. the number of successful offers only represented 25% of the total. This made me investigate and look for alternatives since the accuracy of a classification model depends on a balance in the number of samples of each label.

After so much playing with the data, I could understand that different models can be created from it, for example with a little effort you could create a regression model to predict how much a customer could spend on average after receiving an offer, another option could also be a regression model with which to predict the response time to an offer. And I'm sure that by taking more time other options can be discovered. This is really a very enriching project.

Improvement options?

I think there are opportunities for improvement, if I had more time I would try with Ensembling/Stacking algorithms. Another option would be to look at combining features to have a more simplified model.