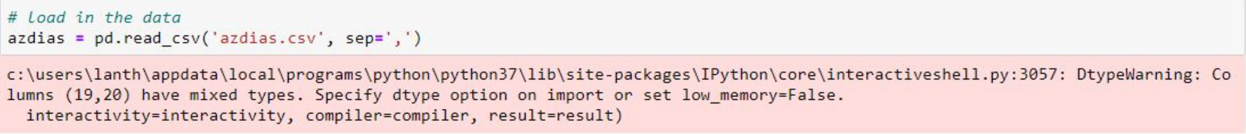
# UDACITY MACHINE LEARNING ENGINEER NANODEGREE

## BERTELSMANN-ARVATO

## CAPSTONE REPORT

# 1. Data Exploration

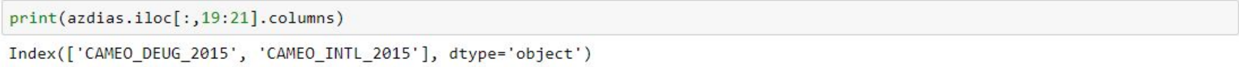
When we load the azdias and customers dataset we raise a mixed type warning :



```
# load in the data
azdias = pd.read_csv('azdias.csv', sep=',')
```

```
c:\users\lanth\appdata\local\programs\python\python37\lib\site-packages\IPython\core\interactiveshell.py:3057: DtypeWarning: Co
lumns (19,20) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

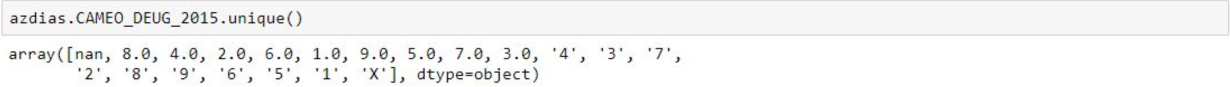
Going forward we need to know what are the names of columns 19 and 20 :



```
print(azdias.iloc[:,19:21].columns)
```

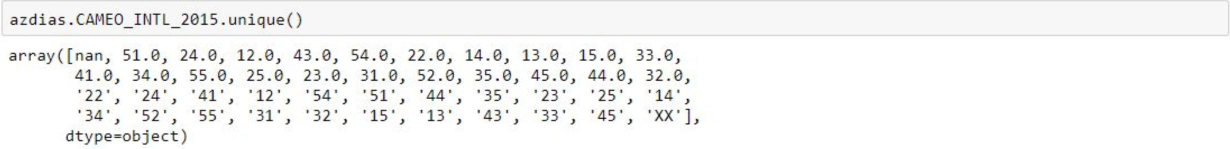
```
Index(['CAMEO_DEUG_2015', 'CAMEO_INTL_2015'], dtype='object')
```

Now we look at what they hold:



```
azdias.CAMEO_DEUG_2015.unique()
```

```
array([nan, 8.0, 4.0, 2.0, 6.0, 1.0, 9.0, 5.0, 7.0, 3.0, '4', '3', '7',
       '2', '8', '9', '6', '5', '1', 'X'], dtype=object)
```



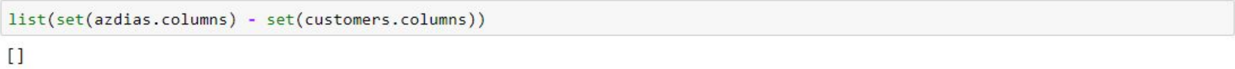
```
azdias.CAMEO_INTL_2015.unique()
```

```
array([nan, 51.0, 24.0, 12.0, 43.0, 54.0, 22.0, 14.0, 13.0, 15.0, 33.0,
       41.0, 34.0, 55.0, 25.0, 23.0, 31.0, 52.0, 35.0, 45.0, 44.0, 32.0,
       '22', '24', '41', '12', '54', '51', '44', '35', '23', '25', '14',
       '34', '52', '55', '31', '32', '15', '13', '43', '33', '45', 'XX'],
      dtype=object)
```

We can see that there are mainly numerical values : int, float and string
But there are also some 'X' and 'XX', corresponding to missing values.
So I have made a function cameo_fix which simply replaces the X and XX by np.nan and sets all to float.

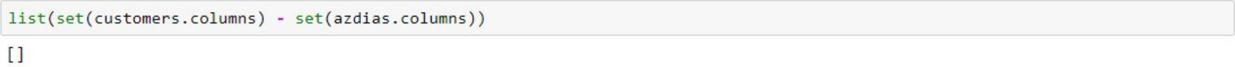
We know that customers dataset holds 3 columns that are not in azdias : 'PRODUCT_GROUP', 'CUSTOMER_GROUP' and 'ONLINE_PURCHASE', therefore we will drop these columns as we want to have the same columns in each dataset.

Then we check if we have the same columns in azdias and customers :



```
list(set(azdias.columns) - set(customers.columns))
```

```
[]
```



```
list(set(customers.columns) - set(azdias.columns))
```

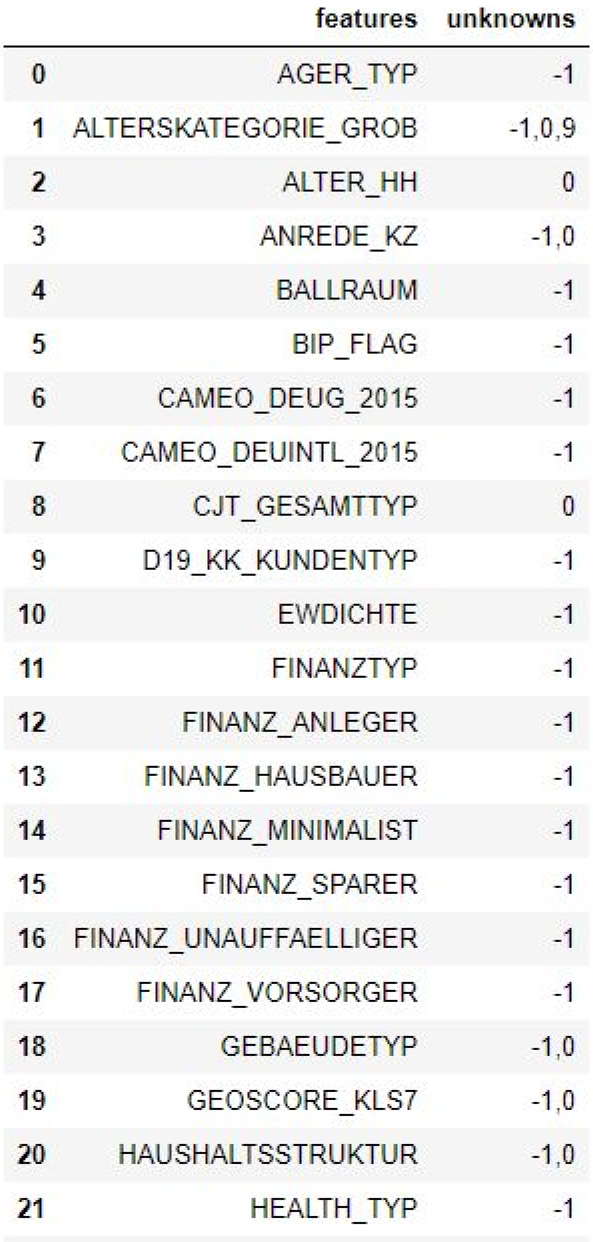
```
[]
```

# 2. Data Preprocessing

## 2.1 Data Cleaning

We will start the preprocessing process by implementing a missing_pct function to calculate the proportion of missing values in each columns

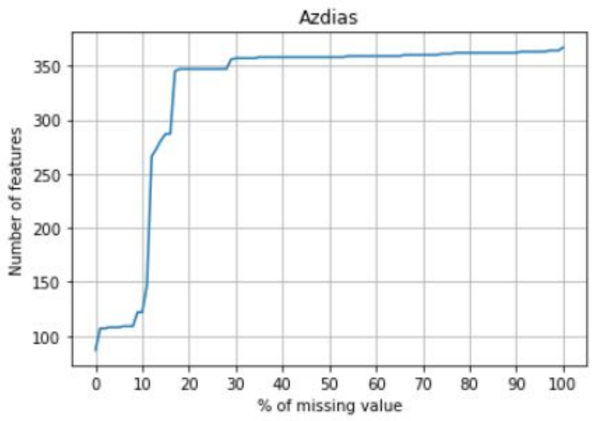
Next we open DIAS Attributes - Values 2017.xlsx to see that it hold missing/unknown value for each column. So we put these into a dataframe to be able to replace the corresponding values by np.nan in each columns

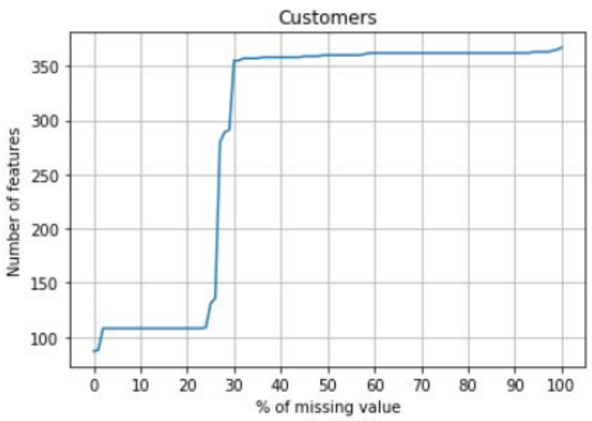


| | features | unknowns |
|---|---|---|
| 0 | AGER_TYP | -1 |
| 1 | ALTERSKATEGORIE_GROB | -1,0,9 |
| 2 | ALTER_HH | 0 |
| 3 | ANREDE_KZ | -1,0 |
| 4 | BALLRAUM | -1 |
| 5 | BIP_FLAG | -1 |
| 6 | CAMEO_DEUG_2015 | -1 |
| 7 | CAMEO_DEUINTL_2015 | -1 |
| 8 | CJT_GESAMTTYP | 0 |
| 9 | D19_KK_KUNDENTYP | -1 |
| 10 | EWDICHTE | -1 |
| 11 | FINANZTYP | -1 |
| 12 | FINANZ_ANLEGER | -1 |
| 13 | FINANZ_HAUSBAUER | -1 |
| 14 | FINANZ_MINIMALIST | -1 |
| 15 | FINANZ_SPARER | -1 |
| 16 | FINANZ_UNAUFFAELLIGER | -1 |
| 17 | FINANZ_VORSORGER | -1 |
| 18 | GEBAEUDETYP | -1,0 |
| 19 | GEOSCORE_KLS7 | -1,0 |
| 20 | HAUSHALTSSTRUKTUR | -1,0 |
| 21 | HEALTH_TYP | -1 |

To do so I write a missing_to_nans function and proceed to replace them and get the new missing_pct values.

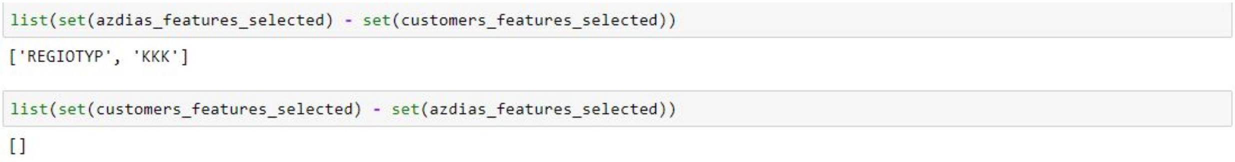
Then I implement a feature_cap function that returns the list of features that have less missing value proportion than a desired number
This way I can plot the amount of features retained given a specific % of missing maximum :





We can see that we could have most of the azdias's columns with a cap fixed around 20%. But that would have a tremendous consequence on customers since we can see that at 20% less than a third of the customers's columns would be kept. So we need to choose the number based on customers and not azdias.
Therefore as 30% seems to fit with most of the columns in both dataset we will keep this number.

Once we create the features_selected list for each dataset we compare to see which features are not in both :



```
list(set(azdias_features_selected) - set(customers_features_selected))
```

```
['REGIOTYP', 'KKK']
```

```
list(set(customers_features_selected) - set(azdias_features_selected))
```

```
[]
```

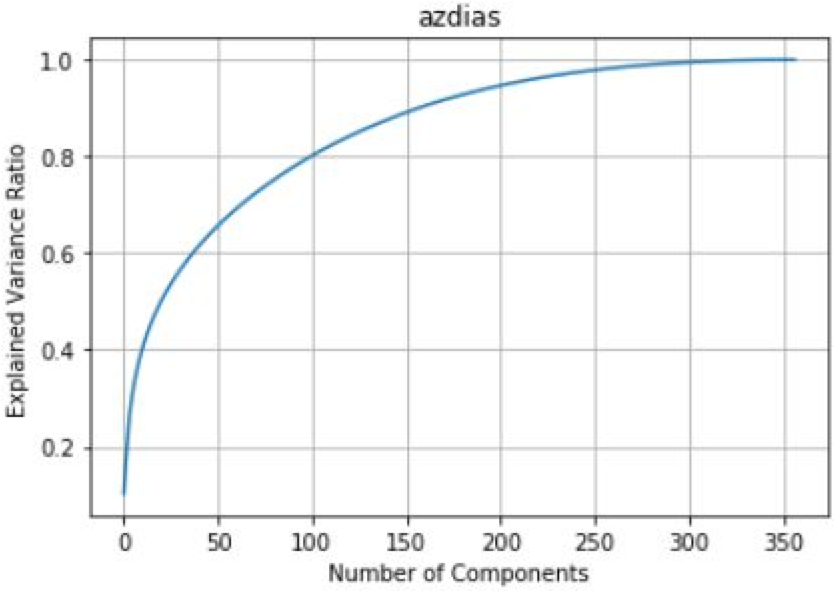
So we drop REGIOTYP and KKK from azdias.

Then I use a features_engineering to transform the features that are either of the wrong type. Those encoding form more than one thing, we split into 2 new features and drop the original one.
And finally it replaces every np.nan value by the most common value in the column.

## 2.2 Data Scaling/Dimensionality reduction

As there are very different ranges of values in each column, I implement a scaler_tool function that simply performs a MinMaxScaler.

Then I use Principal component analysis to achieve dimensionality reduction.
As I need to choose a targeted number of component, I use a plot of the sum of the explained variance for each number of components to decide :



From this plot I choose to go for 150 because it explains 90% of the variance because the gap statistic that i plan to use next is very long to compute and 200 components would make it even longer.

# 3. Customers segmentation

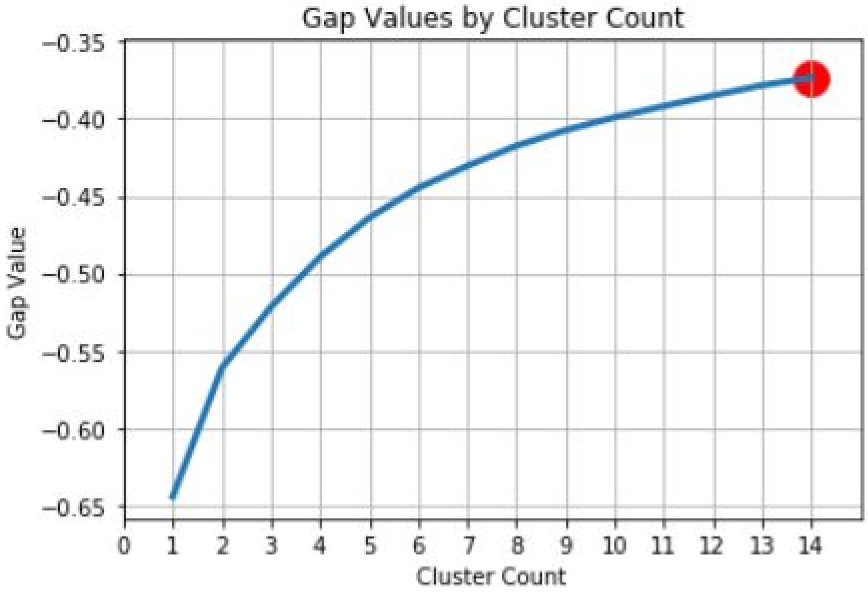
## 3.1 Definitions

“Customer segmentation is the process of dividing customers into groups based on common characteristics so companies can market to each group effectively and appropriately.”
From this definition I choose to use a clustering algorithm with the general demographic dataset (azdias), to then compare in which cluster the customers dataset will fit.

I use K Means as a clustering algorithm, but then I need to choose the optimal K.

## 3.2 Clustering

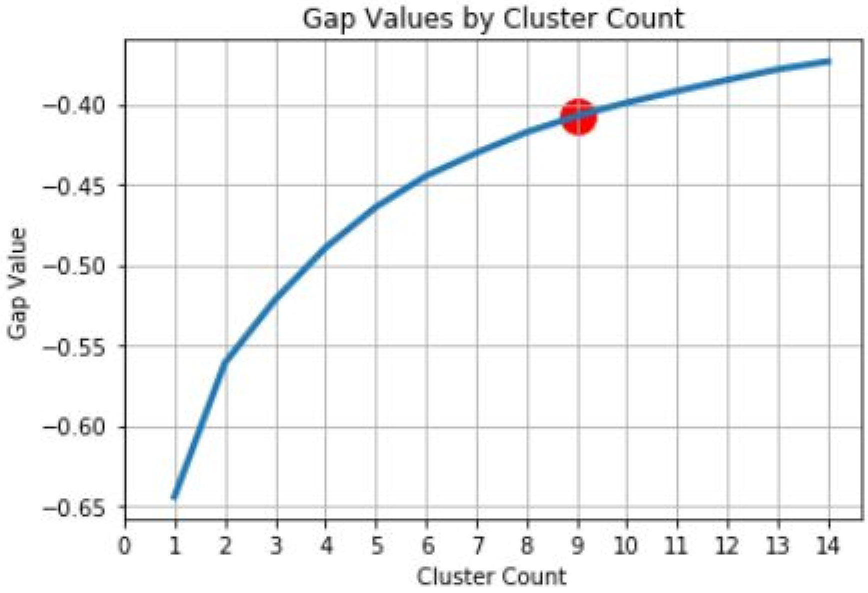
So I perform a gap statistic analysis (from Tibshirani, Walther, Hastie) with the function optimalK :



We can see though that the gap doesn't drop at some point, and will most likely keep increasing until we reach the number of row in the dataset.

In this case Tibshirani suggests the 1-standard-error method, which is implemented with the bestK function.

The result i k = 9



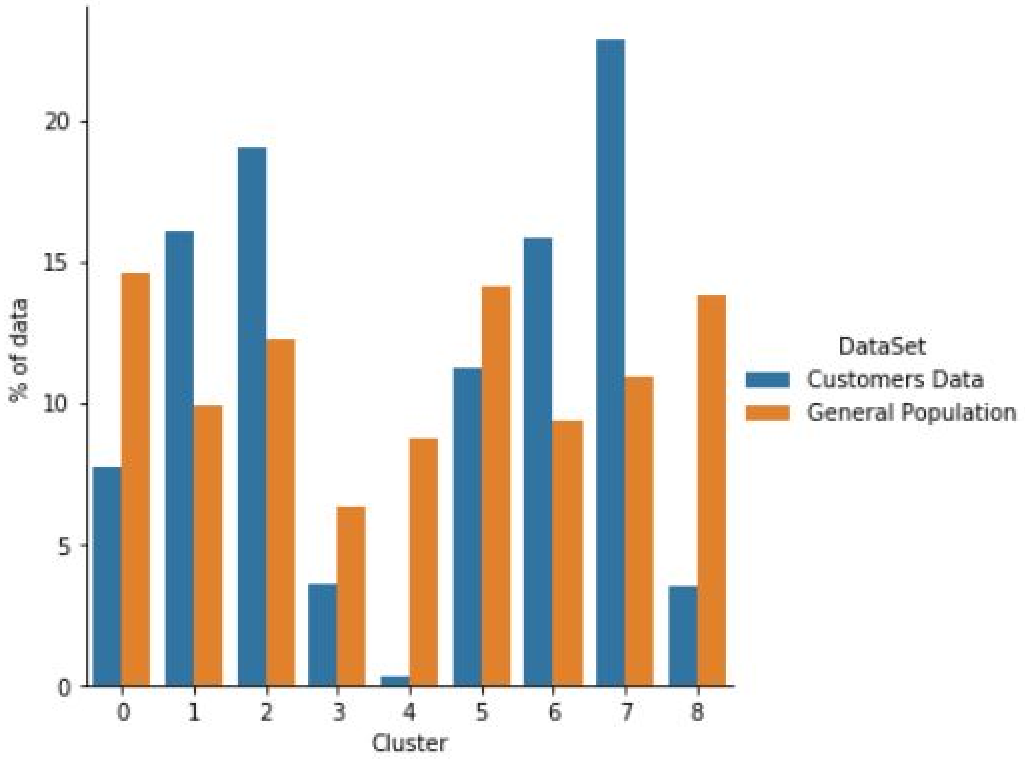
Then the azdias dataset that has been MinMaxed and then PCA(150) is now use to fit a KMeans model with K = 9

## 3.3 Segmentation

Now we have our model fitted we can transform the customers dataset through the same process to see where it ends :

Customers dataset -> preprocessing -> MinMaxScaler -> PCA(150)

Finally we can plot the resulting clusters proportion in each of the two datasets



From that plot I can say :

- Cluster 7 is the very best segment for customers
- Cluster 1, 2 and 6 are very good also
- Cluster 0, 3, 4 and 8 are very bad.

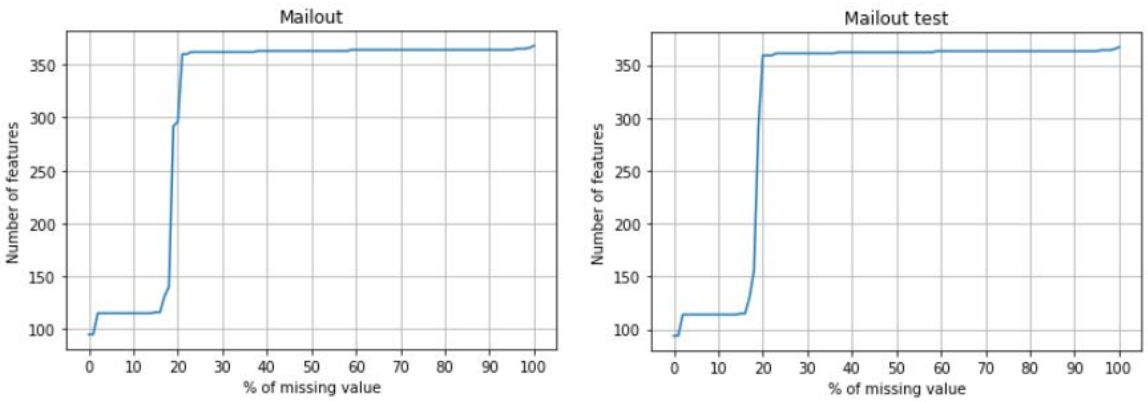
# 4. Supervised Learning

## 4.1 Cleaning/Preprocess

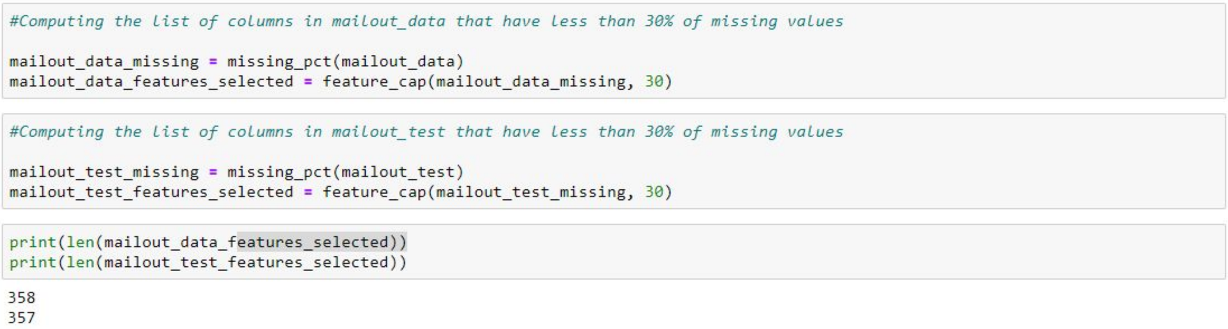
After loading mailout and mailout_test we will first preprocess it :

- cameo_fix
- missing_to_nans

As previously i will plot the amount of features kept by %of missing value cap :



We can see that about 22% would be enough but I will stay with 30% as the curve seems almost flat past 20%, it will not change much.



```
#Computing the list of columns in mailout_data that have less than 30% of missing values

mailout_data_missing = missing_pct(mailout_data)
mailout_data_features_selected = feature_cap(mailout_data_missing, 30)
```

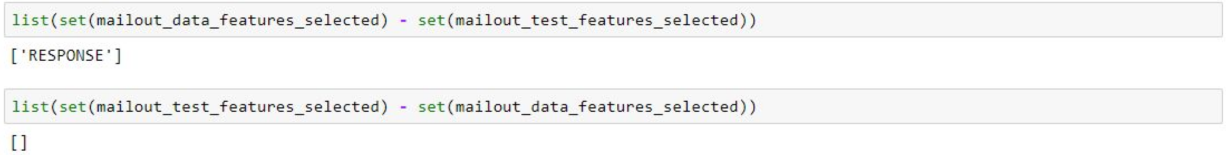
```
#Computing the list of columns in mailout_test that have less than 30% of missing values

mailout_test_missing = missing_pct(mailout_test)
mailout_test_features_selected = feature_cap(mailout_test_missing, 30)
```

```
print(len(mailout_data_features_selected))
print(len(mailout_test_features_selected))
```

```
358
357
```

We need to check if the columns match :



```
list(set(mailout_data_features_selected) - set(mailout_test_features_selected))
```

```
['RESPONSE']
```

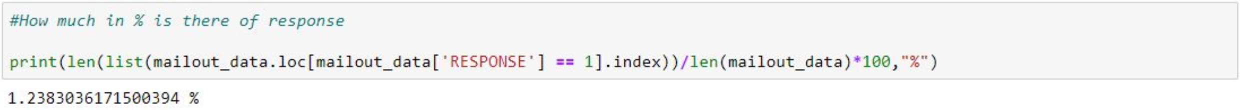
```
list(set(mailout_test_features_selected) - set(mailout_data_features_selected))
```

```
[]
```

Since we know that by definition mailout_test doesn't have the RESPONSE column, we can say that the columns will match after we separate RESPONSE from the mailout dataset.

- features_engineering
- MinMaxScaler

At this point the datasets have the same number of columns, we just do a quick check of how imbalanced the class are :



```
#How much in % is there of response

print(len(list(mailout_data.loc[mailout_data['RESPONSE'] == 1].index))/len(mailout_data)*100,"%")
```

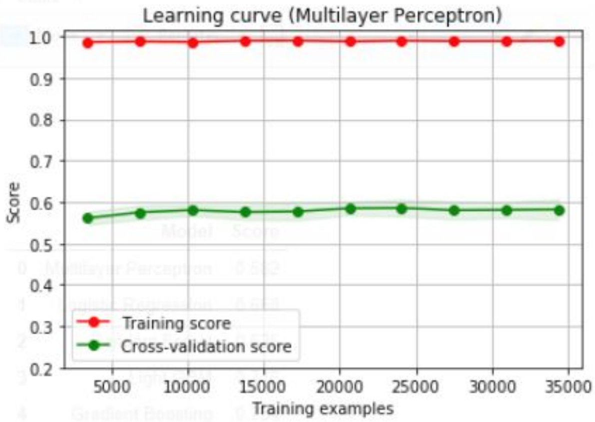
```
1.2383036171500394 %
```

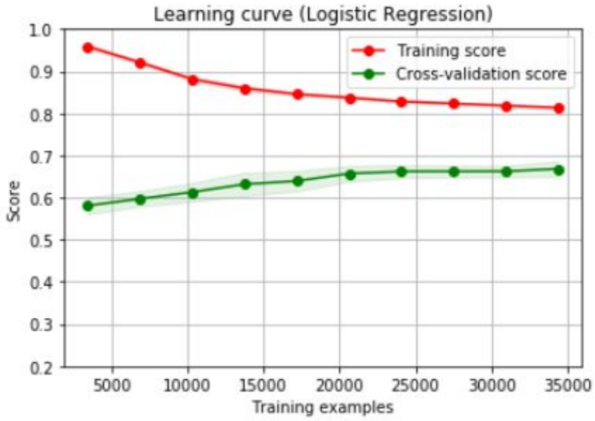
So the data is extremely imbalanced, we will need to address the issue.
I have tried to resample the mailout_data using imblearn TomekLinks, SMOTE, SMOTETomek and ClusterCentroids but it did not improve the quality of the model's prediction. So I've decided to not resample the data afterall.

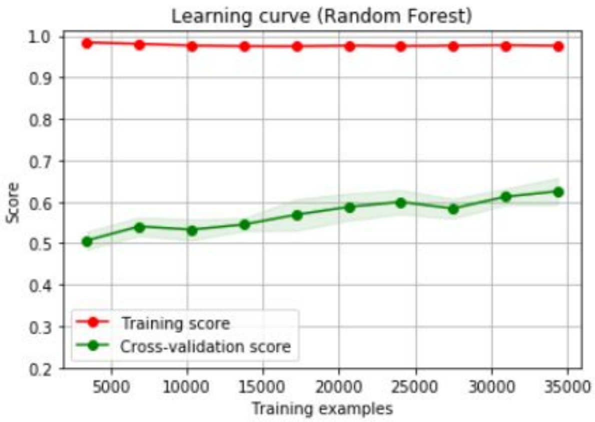
## 4.2 Choosing model

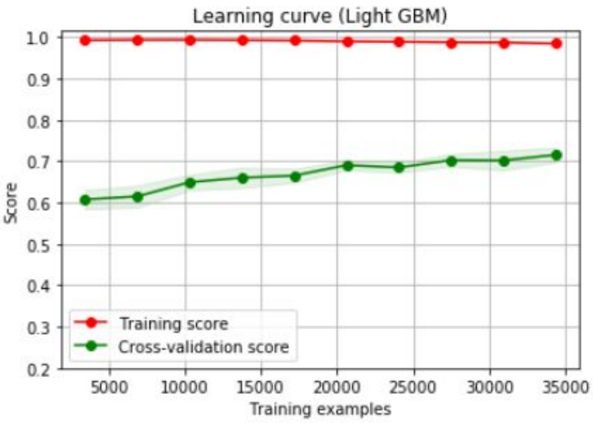
To choose which model I will use, I have decided to plot the learning curve of the following models:

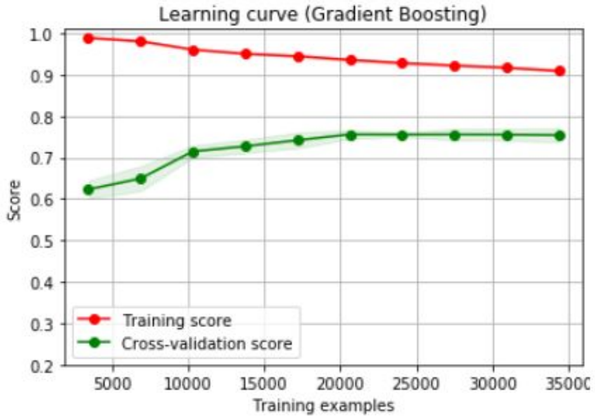
- Multi-layer Perceptron classifier
- Logistic Regression
- Random Forest
- Light GBM
- Gradient Boosting Classifier
- XGBoost

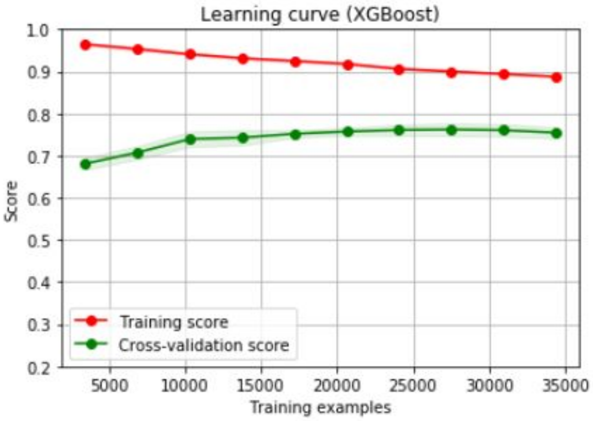




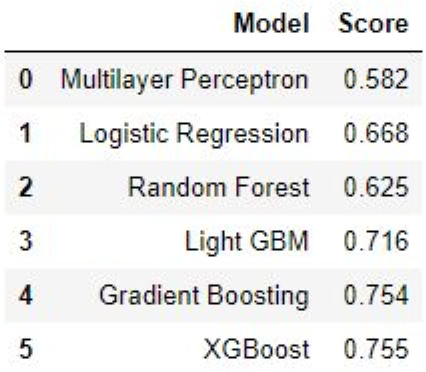








The plot_learning_curve function also return the average ROC_AUC score for the tests set:



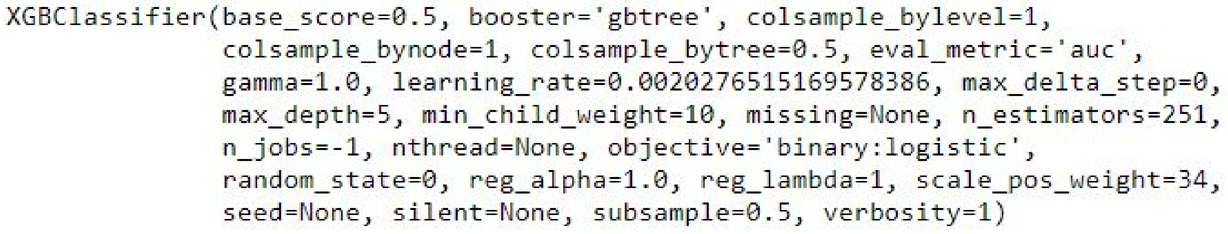
|   | Model | Score |
|---|---|---|
| 0 | Multilayer Perceptron | 0.582 |
| 1 | Logistic Regression | 0.668 |
| 2 | Random Forest | 0.625 |
| 3 | Light GBM | 0.716 |
| 4 | Gradient Boosting | 0.754 |
| 5 | XGBoost | 0.755 |

As the a result we can see that XGBoost is the best performing model from scratch (though being closely followed up by Light GBM)
So I will go further with XGBoost.

## 4.3 Hyperparameters tuning

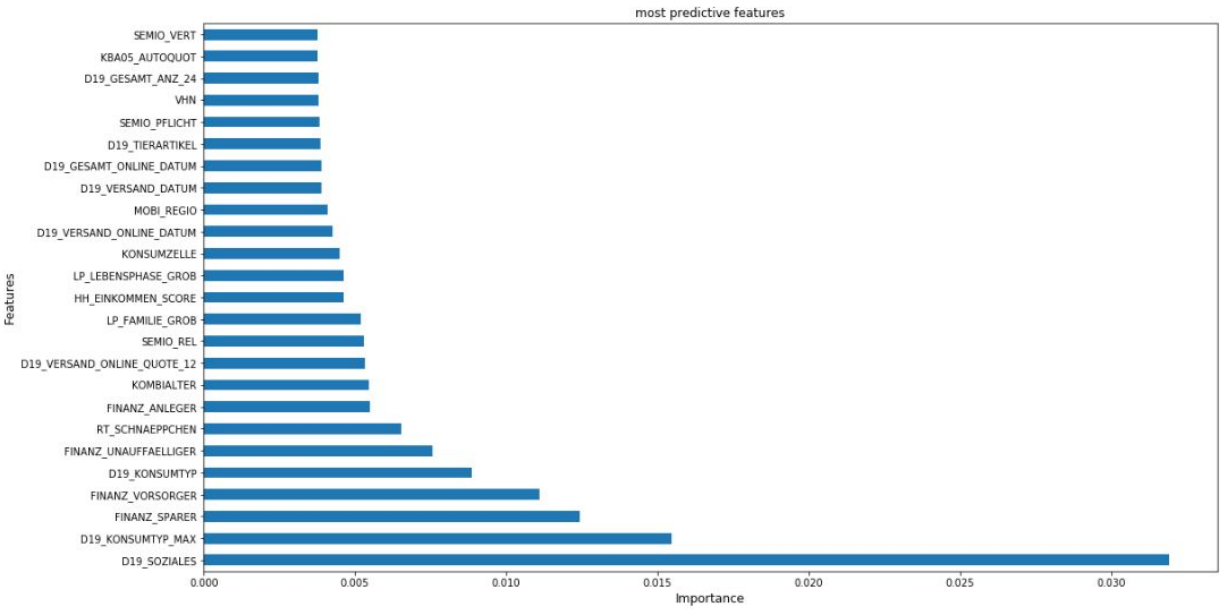
At this point we have only to find the hyperparameters to use for our XGBoost model, in order to have the best rank possible in the kaggle competition.
I have found a similar problem resolved on kaggle : Bayesian Optimization of xgBoost which suggests the use of the BayesSearchCV class from scikit-learn optimize. Though it is extremely time consuming, the results were very good.

The best classifier I have found with this optimization process :

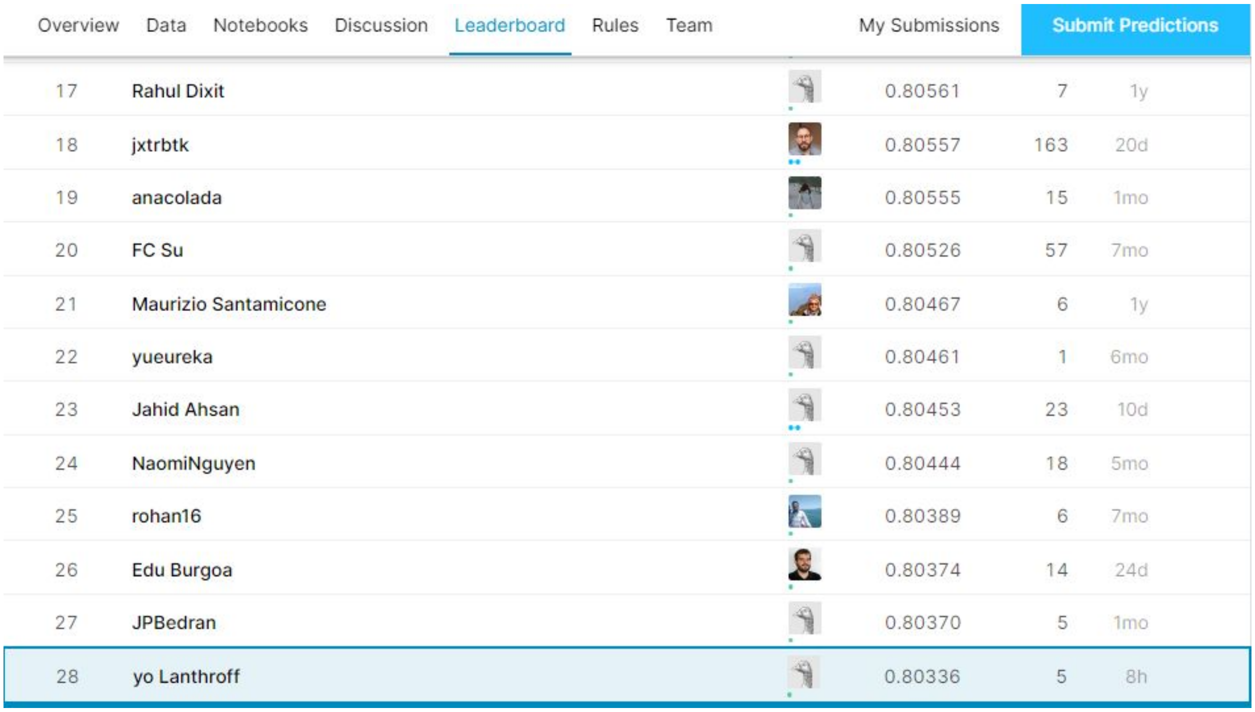


```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, eval_metric='auc',
              gamma=1.0, learning_rate=0.0020276515169578386, max_delta_step=0,
              max_depth=5, min_child_weight=10, missing=None, n_estimators=251,
              n_jobs=-1, nthread=None, objective='binary:logistic',
              random_state=0, reg_alpha=1.0, reg_lambda=1, scale_pos_weight=34,
              seed=None, silent=None, subsample=0.5, verbosity=1)
```

After a long computation, the resulting model gives us the following features importance's overview :



## 4.4 Kaggle competition



| | | | | |
|---|---|---|---|---|
| 17 | Rahul Dixit | 0.80561 | 7 | 1y |
| 18 | jxtrbtk | 0.80557 | 163 | 20d |
| 19 | anacolada | 0.80555 | 15 | 1mo |
| 20 | FC Su | 0.80526 | 57 | 7mo |
| 21 | Maurizio Santamicone | 0.80467 | 6 | 1y |
| 22 | yueureka | 0.80461 | 1 | 6mo |
| 23 | Jahid Ahsan | 0.80453 | 23 | 10d |
| 24 | NaomiNguyen | 0.80444 | 18 | 5mo |
| 25 | rohan16 | 0.80389 | 6 | 7mo |
| 26 | Edu Burgoa | 0.80374 | 14 | 24d |
| 27 | JPBedran | 0.80370 | 5 | 1mo |
| 28 | yo Lanthroff | 0.80336 | 5 | 8h |

When I checked the leaderboard first, I saw that a lot of people achieved a score just above 0.80 and only one did break the 0.81 limit. So my primary goal was also to be in the same range of 0.80+

I am glad I could achieve that.