MACHINE LEARNING ENGINEER NANODEGREE

CAPSTONE PROJECT REPORT

Arvato-Bertelsmann Customer Acquisition

UDACITY

Ahmad Shapiro
ahmad.shapiro@alexu.edu.eg

Domain Background

Bertelsmann found its origins as a publishing house in 1835 (Schuler, 2010), and through steady growth and development made its way to the software and hardware distribution market in the 80’s (Computerwoche, 1983). By 1999 the company received its current name Arvato Bertelsmann (Neuer Name, neue Ziele, 1999) and over the next decade fully entered the domain of high-tech, information technology, and ecommerce services (Paperlein, 2012).

Arvato offers financial solutions in the form of diverse segments, from payment processing to risk management activities. It is in this domain that this capstone project will be developed. Arvato is looking to use its available datasets to support a client (mail-order company selling organic products) in identifying the best data founded way to acquire a new client base.

To achieve this goal I will explore Arvato’s existing datasets to identify attributes and demographic features that can help segment customers of interest for this particular client.

Customer centric marketing is a growing field that benefits greatly from accurate segmentation, with the help of machine learning hidden patterns can be found in volumes that could easily be missed without computational help, requiring very little maintenance or human intervention, leading to an improved experience from customer seekers and customers alike.

Problem Statement

The problem statement for this project is “How can a client – mail order company selling organic products – acquire new clients in a more efficient way?”

All of the projects I've done , I've been following a great strategy that always helped me , “Divide and Conquer” , so according to this strategy we will divide our project into two phases.

Phase 1:

We will use the population data with an unsupervised learning approach to identify the population different clusters and then apply them to the customers data set to see if we can detect any sort of pattern.

Phase 2:

After learning about customer's clusters we will be switching into a supervised learning approach using a dataset with demographics information for the target customers for the advertising campaign and predict which individuals would be more likely to convert to company customers.

Datasets and Input

All the datasets were provided by Arvato in the context of the Udacity Machine Learning Engineer Nanodegree, on the subject of Customer Acquisition / Targeted Advertising prediction models.

There are 4 datasets to be explored in this project:

- ❏ **Udacity_AZDIAS_052018.csv**: Demographics data for the general population of Germany; 891,211 persons (rows) x 366 features (columns)
- ❏ **Udacity_CUSTOMERS_052018.csv**: Demographics data for customers of a mail-order company; 191,652 persons (rows) x 369 features (columns).
- ❏ **Udacity_MAILOUT_052018_TRAIN.csv**: Demographics data for individuals who were targets of a marketing campaign; 42,982 persons (rows) x 367 (columns).
- ❏ **Udacity_MAILOUT_052018_TEST.csv**: Demographics data for individuals who were targets of a marketing campaign; 42,833 persons (rows) x 366 (columns).

And **2 metadata** files associated with these datasets:

- ❏ **DIAS Information Levels — Attributes 2017.xlsx**: list of attributes and descriptions, organized by informational category
- ❏ **DIAS Attributes — Values 2017.xlsx**: a detailed mapping of data values for each feature in alphabetical order.

Evaluation Metrics

Phase 1 (Unsupervised Learning) :-

1. PCA variance ratio for the dimensionality reduction .
2. Sum of squared error for the K-Means Clustering algorithm to determine the best k according to elbow method.

Phase 2 (Supervised Learning) :-

1. Area under ROC Curve :

   Before explaining this metric , we should first explain its components.

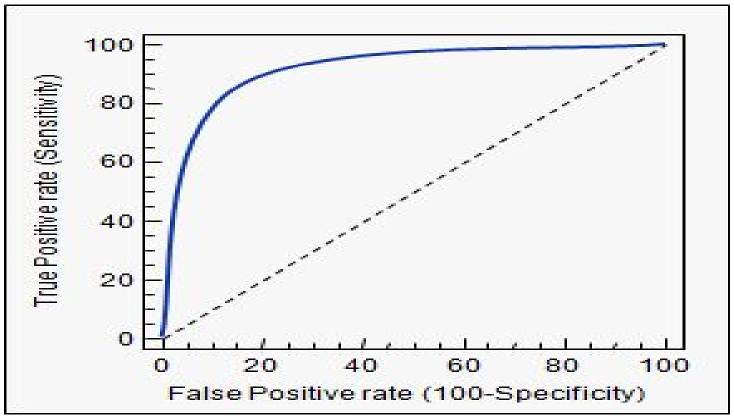
   As we are doing a binary classification task. We have four possibilities of any prediction.

   1. Predicted : Positive , Real : Positive which is TRUE Positive.
   2. Predicted : Positive , Real : Negative which is False Positive.
   3. Predicted : Negative , Real : Negative which is TRUE Negative.
   4. Predicted : Negative , Real : Positive which is False Negative.

After Calculating the above numbers , we are left with two other metrics :-

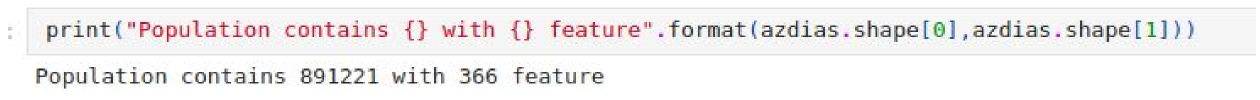
1. Sensitivity : Which is true positive rate = true positives / true positives + false negatives.
2. Specificity : Which is true negatives rate = true negatives / true negatives + false positives.
3. False Positive Rate : 1 - Specificity

The ROC curve is simply a plot of Sensitivity against the false positive rate for different cutoffs variables, each point on the curve is a pair of both specificity and sensitivity corresponding to a threshold , a good result will have 100% in both respictly which results in area of 1 under the curve, but the worse may have 0.5 area under the curve as illustrated below



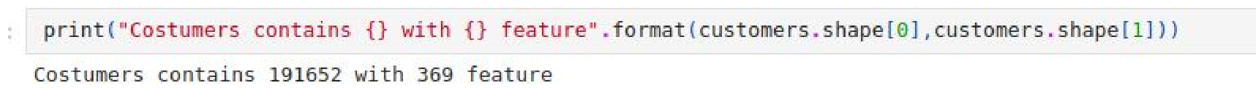
Data Wrangling :-

First we loaded both datasets : Customers and Azdias (population) .



```
print("Population contains {} with {} feature".format(azdias.shape[0],azdias.shape[1]))
```

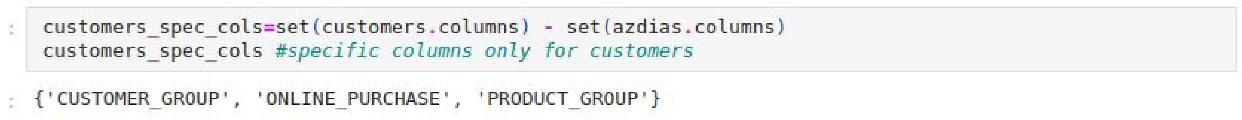
```
Population contains 891221 with 366 feature
```



```
print("Costumers contains {} with {} feature".format(customers.shape[0],customers.shape[1]))
```

```
Costumers contains 191652 with 369 feature
```

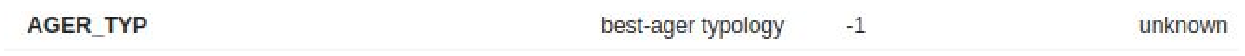
After that we noticed that there are 3 more features included in the customers data set which aren't included in azdias , so we checked they turned out to be some features that only needed for customers.



```
customers_spec_cols=set(customers.columns) - set(azdias.columns)
customers_spec_cols #specific columns only for customers
```

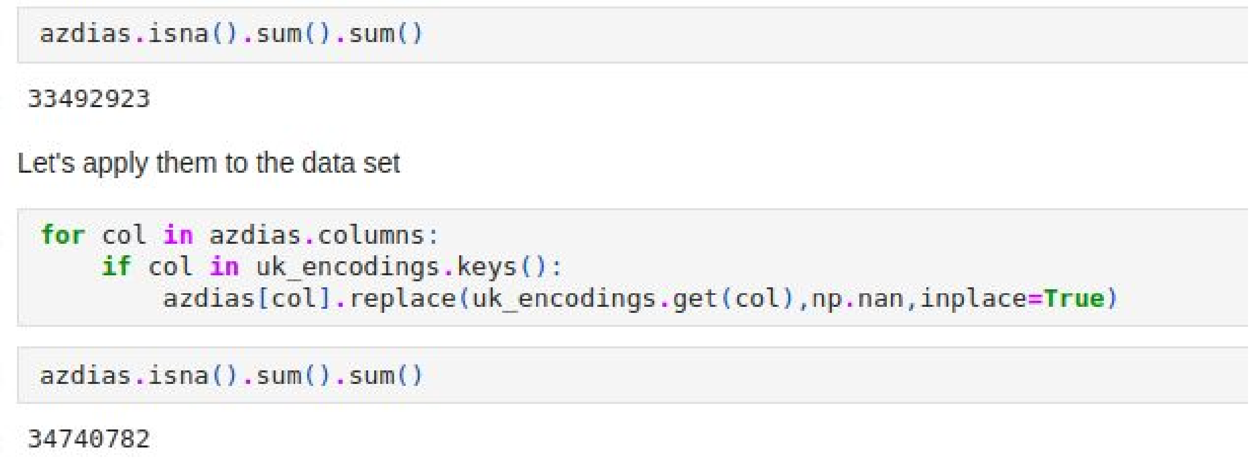
```
{'CUSTOMER_GROUP', 'ONLINE_PURCHASE', 'PRODUCT_GROUP'}
```

After that we loaded “**DIAS Attributes - Values 2017.xlsx”** to check the attributes of each feature and the range of it’s values .



AGER_TYP	best-ager typology	-1	unknown

Data is read from pandas and missing values are encoded into np.nan data type, but since we may have some entries that are unknowns rather than missing , we used the above sheet to make a dictionary to replace each value that meant either unknown or missing for each feature to np.nan.



```
azdias.isna().sum().sum()
```

```
33492923
```

Let's apply them to the data set

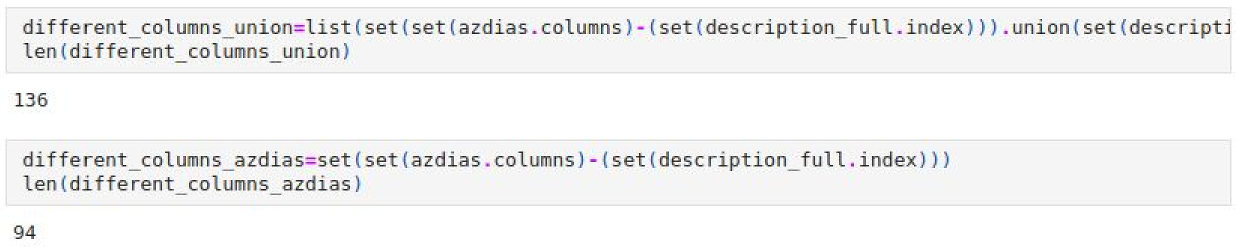
```
for col in azdias.columns:
    if col in uk_encodings.keys():
        azdias[col].replace(uk_encodings.get(col),np.nan,inplace=True)
```

```
azdias.isna().sum().sum()
```

```
34740782
```

The number of missing values increased approximately 1,250,000 with more missing values in the population data set and 260,000 for the customer’s data set.

Those were the explicit missing/unknown values but we may have some outliers or data entry errors, but fortunately we have all possible values of some features in the Attribute Description excel sheet.



```
different_columns_union=list(set(set(azdias.columns)-(set(description_full.index))).union(set(descripti
len(different_columns_union)
```

```
136
```

```
different_columns_azdias=set(set(azdias.columns)-(set(description_full.index)))
len(different_columns_azdias)
```

```
94
```

We can see from above that we have **94** feature in azdias that are not in the description sheet and **136** feature in total that are in azdias and not in description or vice versa

Due to above , we will only work with the intersection between both to identify potential outliers

```
inter_cols=list(set(azdias.columns).intersection(set(description_full.index)))
len(inter_cols)
```

```
272
```

The intersection between both datasets was 272 columns and we named them inter_cols.

Then for the columns that aren't included in the description dataset we checked the proportion of the missing values in them, using the summarize_columns helper function in tools.py script ,and those above the threshold we specified earlier will be dropped later.

```
nan_df.sort_values(by="Nan_prop",ascending=False,inplace=True)
nan_df[nan_df.Nan_prop >= 0.5]
```

	Nan_prop
ALTER_KIND4	0.998648
ALTER_KIND3	0.993077
ALTER_KIND2	0.966900
ALTER_KIND1	0.909048
EXTSEL992	0.733996
KK_KUNDENTYP	0.655967

After that , we made a dictionary with keys as column names and values are a list of all possible values that we extracted from the description data set.

```
{'KBA05_MOD8': [0, 1, 2, 3],
 'KBA13_CCM_2000': [0, 1, 2, 3, 4, 5],
 'TITEL_KZ': [1, 2, 3, 4, 5],
 'KBA13_SEG_GELAENDEWAGEN': [0, 1, 2, 3, 4, 5],
 'ONLINE_AFFINITAET': [0, 1, 2, 3, 4, 5],
 'KBA13_HALTER_45': [0, 1, 2, 3, 4, 5],
 'KBA13_SEG_VAN': [0, 1, 2, 3, 4, 5],
```

After that we looped over all columns in the azdias and customers data sets and any value that weren't in the values dictionary we replaced it with np.nan (missing value) which increased the number of nans in the azdias data set by 3,144,445 missing value.

After identifying all potential missing values , it was the time to drop columns with proportion of missing values more than the threshold we set earlier , but before this we had to take the decision to drop the additional 3 columns in the customers data set because they won't be a key in any further analysis , after dropping them both azdias and customers data sets had the same number of columns which is 366 feature.

Now for identifying columns with a high proportion of missing values.

We constructed a data frame with columns feature , class , proportion (missing values proportion) as following :-

	Feature	Class	Propotion
0	ALTER_KIND4	Population	0.998648
1	ALTER_KIND3	Population	0.993077
2	TITEL_KZ	Population	0.997576
3	ALTER_KIND2	Population	0.966900
4	ALTER_KIND1	Population	0.909048
...	...	...	...
727	SEMIO_TRADV	Customers	0.000000
728	SEMIO_VERT	Customers	0.000000
729	ZABEOTYP	Customers	0.000000
730	ANREDE_KZ	Customers	0.000000
731	ALTERSKATEGORIE_GROB	Customers	0.000000

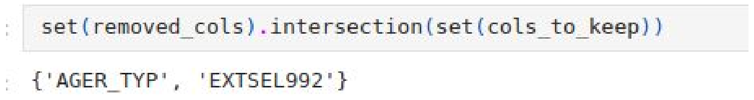
732 rows × 3 columns

Now it's time for selecting those with proportions above threshold in both data sets.

```
threshold=0.5
removed_cols=list(set(feature_nans[~(feature_nans.Propotion < threshold)].Feature.values))
removed_cols
```

```
['TITEL_KZ',
 'ALTER_KIND1',
 'ALTER_KIND3',
 'AGER_TYP',
 'ALTER_KIND4',
 'KBA05_BAUMAX',
 'KK_KUNDENTYP',
 'ALTER_KIND2',
 'CAMEO_DEUG_2015',
 'EXTSEL992']
```

They turned out to be 10 columns which are listed above ,

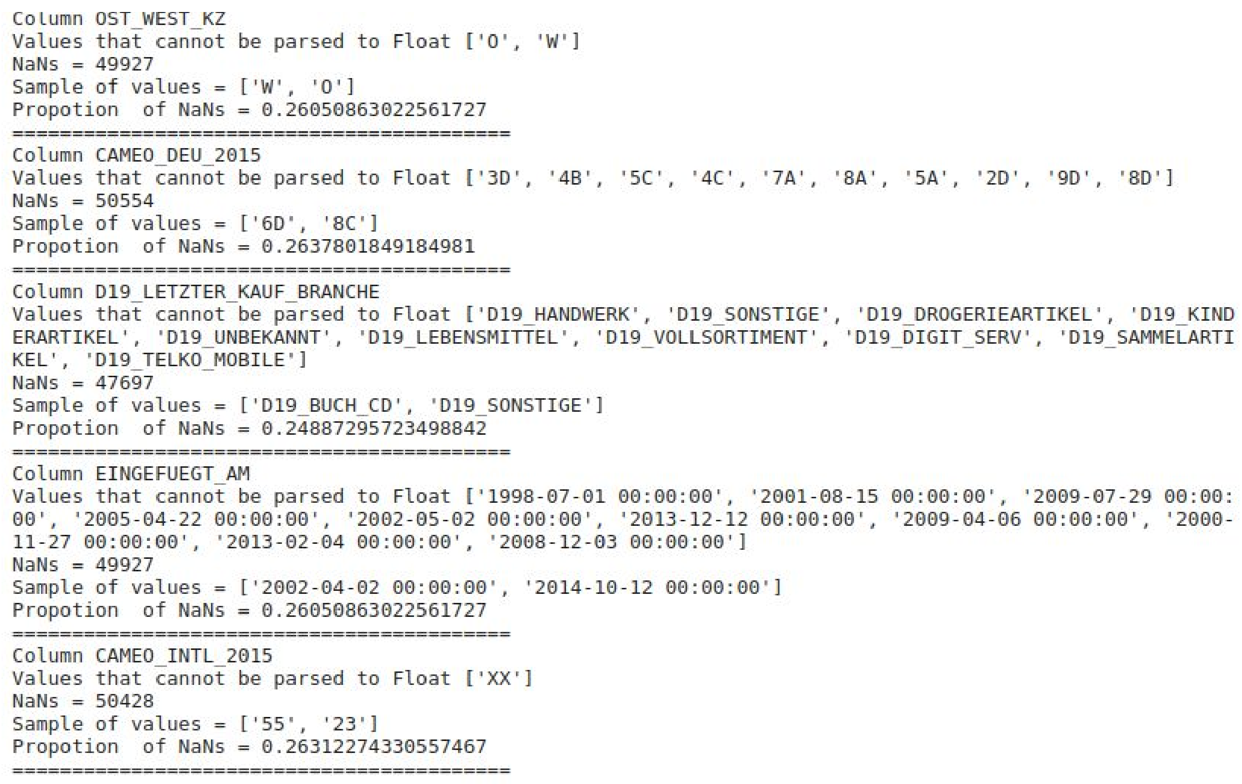


```
set(removed_cols).intersection(set(cols_to_keep))
```

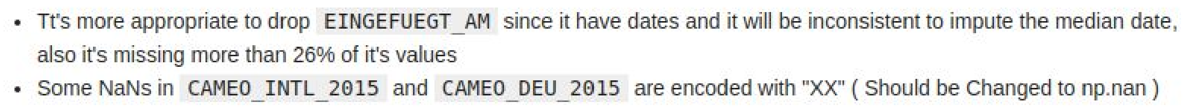
```
{'AGER_TYP', 'EXTSEL992'}
```

But two of them weren't shared between two data sets so 8 columns are only removed , to reach a final dimensionality of 358 features.

Now it was time to identify string categorical columns because we won’t be able to impute them unless they’re encoded. After identifying them we found them to be 5 columns among both dataframes, so we had to check them manually to decide if we can drop any of them for any reason or not.



```
Column OST_WEST_KZ
Values that cannot be parsed to Float ['O', 'W']
NaNs = 49927
Sample of values = ['W', 'O']
Propotion  of NaNs = 0.26050863022561727
===========================================
Column CAMEO_DEU_2015
Values that cannot be parsed to Float ['3D', '4B', '5C', '4C', '7A', '8A', '5A', '2D', '9D', '8D']
NaNs = 50554
Sample of values = ['6D', '8C']
Propotion  of NaNs = 0.2637801849184981
===========================================
Column D19_LETZTER_KAUF_BRANCHE
Values that cannot be parsed to Float ['D19_HANDWERK', 'D19_SONSTIGE', 'D19_DROGERIEARTIKEL', 'D19_KIND
ERARTIKEL', 'D19_UNBEKANNT', 'D19_LEBENSMITTEL', 'D19_VOLLSORTIMENT', 'D19_DIGIT_SERV', 'D19_SAMMELARTI
KEL', 'D19_TELKO_MOBILE']
NaNs = 47697
Sample of values = ['D19_BUCH_CD', 'D19_SONSTIGE']
Propotion  of NaNs = 0.24887295723498842
===========================================
Column EINGEFUEGT_AM
Values that cannot be parsed to Float ['1998-07-01 00:00:00', '2001-08-15 00:00:00', '2009-07-29 00:00:
00', '2005-04-22 00:00:00', '2002-05-02 00:00:00', '2013-12-12 00:00:00', '2009-04-06 00:00:00', '2000-
11-27 00:00:00', '2013-02-04 00:00:00', '2008-12-03 00:00:00']
NaNs = 49927
Sample of values = ['2002-04-02 00:00:00', '2014-10-12 00:00:00']
Propotion  of NaNs = 0.26050863022561727
===========================================
Column CAMEO_INTL_2015
Values that cannot be parsed to Float ['XX']
NaNs = 50428
Sample of values = ['55', '23']
Propotion  of NaNs = 0.26312274330557467
===========================================
```



- Tt's more appropriate to drop `EINGEFUEGT_AM` since it have dates and it will be inconsistent to impute the median date, also it's missing more than 26% of it's values
- Some NaNs in `CAMEO_INTL_2015` and `CAMEO_DEU_2015` are encoded with "XX" (Should be Changed to np.nan)

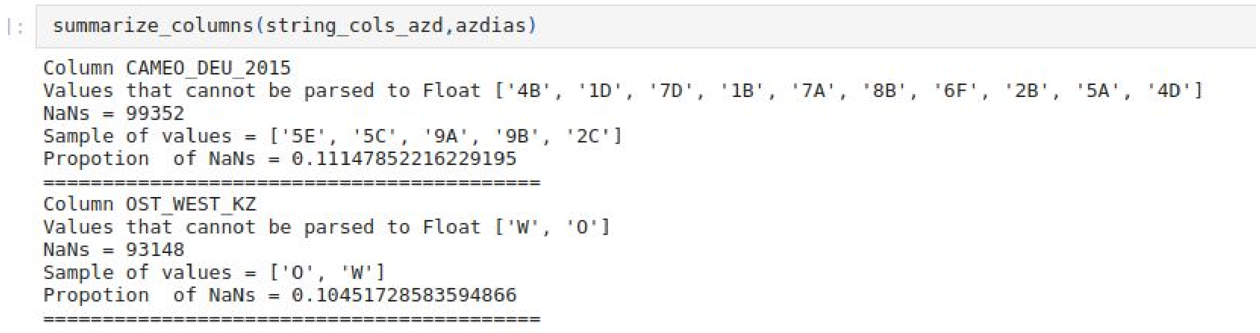
After that we discovered that we missed something in the previous encoding missing value step , in both columns CAMEO_INT_2015 and CAMEO_DEU_2015 with unknown values encoded as “XX” , after changing it to np.nan we found out the column CAMEO_INT_2015 turned to be from numerical categorical columns not a string one.

After all of the previous steps we reached a dimensionality of 357 columns in both data sets.

Now we only have three strings categorical columns which are :

‘'D19_LETZTER_KAUF_BRANCHE', ‘CAMEO_DEU_2015’ and ‘OST_WEST_KZ’

The first one’s values are only combination of other column’s names so we can drop it , to be left with only 2 string categorical columns .



```
[ ]: summarize_columns(string_cols_azd,azdias)

Column CAMEO_DEU_2015
Values that cannot be parsed to Float ['4B', '1D', '7D', '1B', '7A', '8B', '6F', '2B', '5A', '4D']
NaNs = 99352
Sample of values = ['5E', '5C', '9A', '9B', '2C']
Propotion  of NaNs = 0.11147852216229195
===========================================
Column OST_WEST_KZ
Values that cannot be parsed to Float ['W', 'O']
NaNs = 93148
Sample of values = ['O', 'W']
Propotion  of NaNs = 0.10451728583594866
===========================================
```

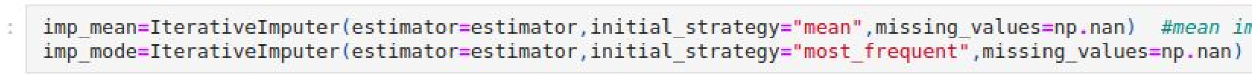
After summarizing both columns as illustrated in the figure above, they should be encoded to integers to impute the their missing values, we used sklearn’s OrdinalEncoder to encode CAMEO_DEU_2015 , and we hard encoded the OST_WEST_KZ because it had only two levels “W” and “O” which had been encoded into 1 , 0 respectively.

Imputing Missing Values :

We used an iterative imputer which simply fits an estimator to estimate a features missing values from the rest of the features.

We’ve chosen a Bayesian Ridge estimator , but as we mentioned earlier we had two types of columns : numerical and categorical .

For the numerical column we set the imputing strategy to mean “Average” , but for the Categorical columns we went for the most-frequent “mode” As illustrated below.



```
imp_mean=IterativeImputer(estimator=estimator,initial_strategy="mean",missing_values=np.nan)  #mean im
imp_mode=IterativeImputer(estimator=estimator,initial_strategy="most_frequent",missing_values=np.nan)
```

After that we Imputed the missing values and then decoded the encoded categorical variables again to their initial state without any missing values and then we saved azdias and customers data sets into their clean version , and the same exact pipeline has been done to test and training data set with only one exception that the training data had only an extra column which was the response variable.

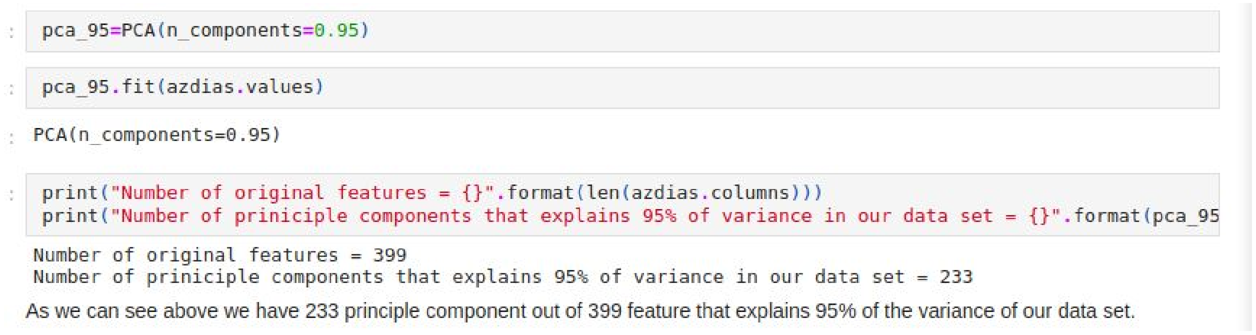
Unsupervised Learning :-

All of the upcoming unsupervised methods will be fitted on the population dataframe (Azdias) because it contains the most variability and highest number of observations.

PCA :-

Principal components analysis are one of the best dimensionality reduction algorithms ,

But before applying it we one-hot encoded our string categorical variables , after that we fitted a PCA object that targets to explain 95% of the variability in the data set on the population dataset , which computed 95% variance explaining features from a combination of the original 399 features (after one-encoding) As illustrated below.



```
pca_95=PCA(n_components=0.95)
```

```
pca_95.fit(azdias.values)
```

```
PCA(n_components=0.95)
```

```
print("Number of original features = {}".format(len(azdias.columns)))
print("Number of priniciple components that explains 95% of variance in our data set = {}".format(pca_95
```

```
Number of original features = 399
Number of priniciple components that explains 95% of variance in our data set = 233
```

As we can see above we have 233 principle component out of 399 feature that explains 95% of the variance of our data set.

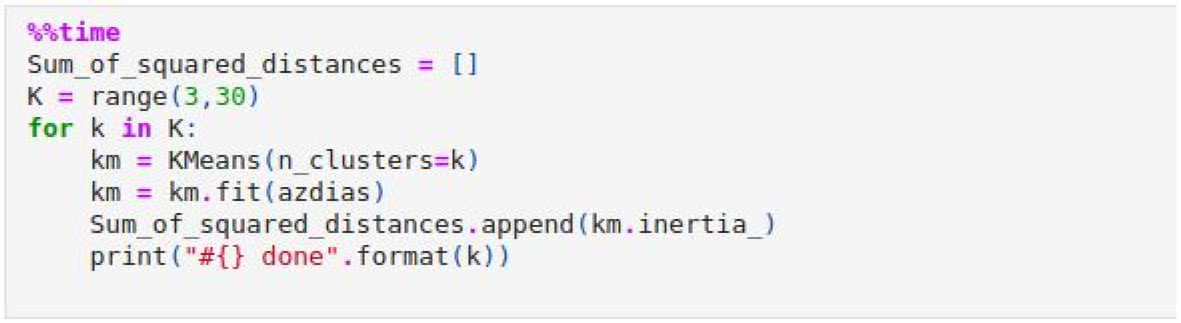
After that , the PCA object has been saved to be used on the customers data set as well .

After transforming the customers dataset to it's principal components we went to the last part of Phase 1 which is the segmentation step.

K-Means CLustering :-

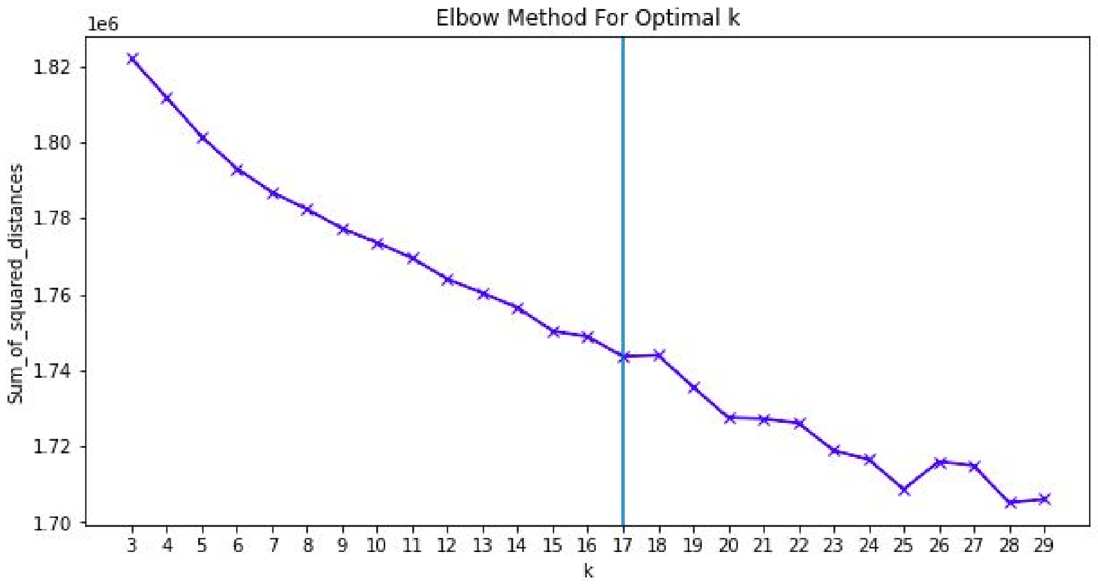
K-means is the most famous clustering algorithm in which the whole idea behind it is randomly fitting k centers in the data and iteratively adjusting them so that the distance between each point and it's cluster center is minimized.

As we are segmenting population demographics data we looped over a range from K=3 to K=29 to determine which K is better according to some an evaluation metric called elbow method.

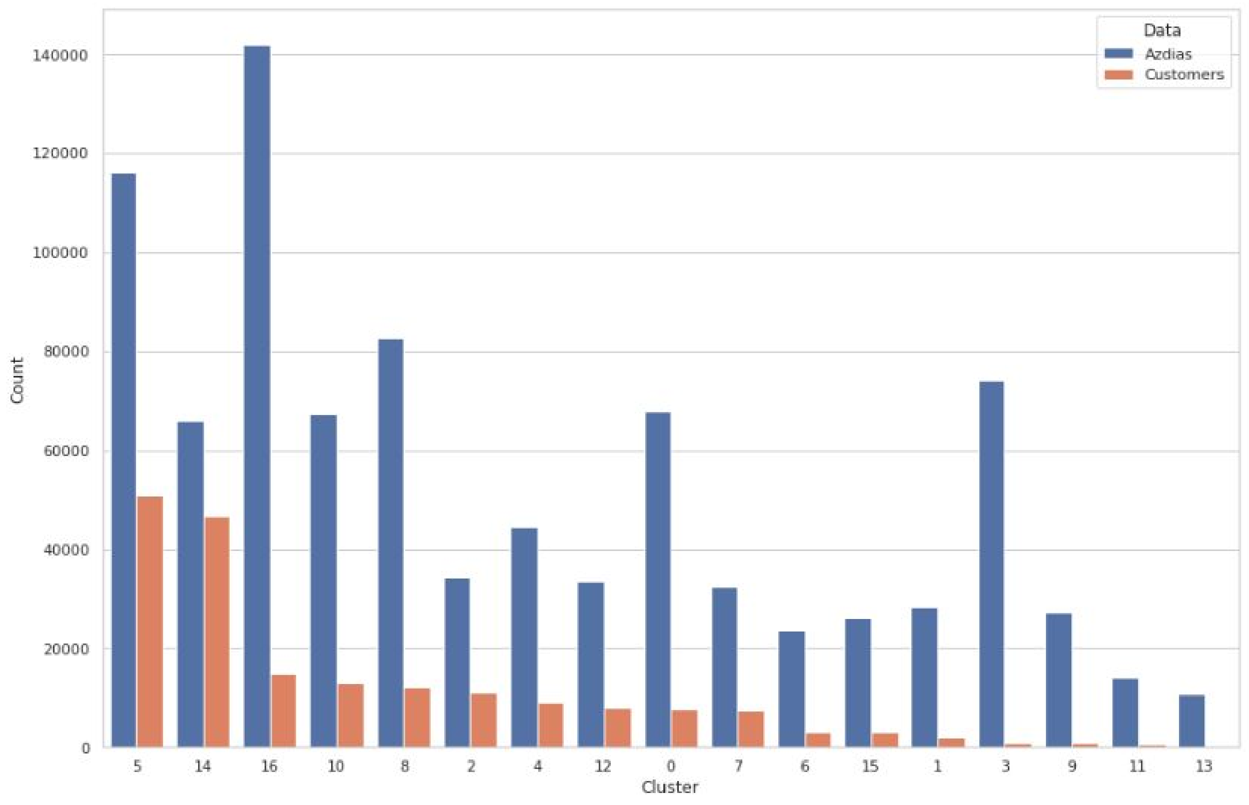


```
%%time
Sum_of_squared_distances = []
K = range(3,30)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(azdias)
    Sum_of_squared_distances.append(km.inertia_)
    print("#{} done".format(k))
```

Then after that we plotted the sum of squared distances on y axis and K on the x-axis and with help from kneed library we successfully located the most appropriate K which was 17 clusters.



After that we computed the clusters for the customers data set , and now it's time to do some comparisons.

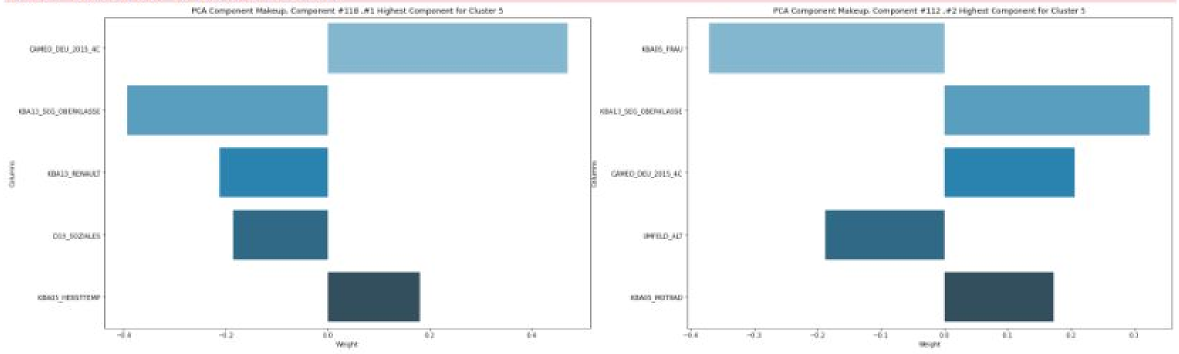


As we are seeing above that Clusters 5 and 14 are high among customers compared to other clusters , let's now check what features are contributing to those clusters, which will make us identify demographics of potential customers.

Cluster 5

```
plot_cluster(5,top_cluster_comp=2,top_pca_comp=5,plot_num=1)
```

```
/home/shapiro/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23: MatplotlibDeprecationWarni
ng: Passing non-integers as three-element position specification is deprecated since 3.3 and will be rem
oved two minor releases later.
/home/shapiro/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23: MatplotlibDeprecationWarni
ng: Passing non-integers as three-element position specification is deprecated since 3.3 and will be rem
oved two minor releases later.
```



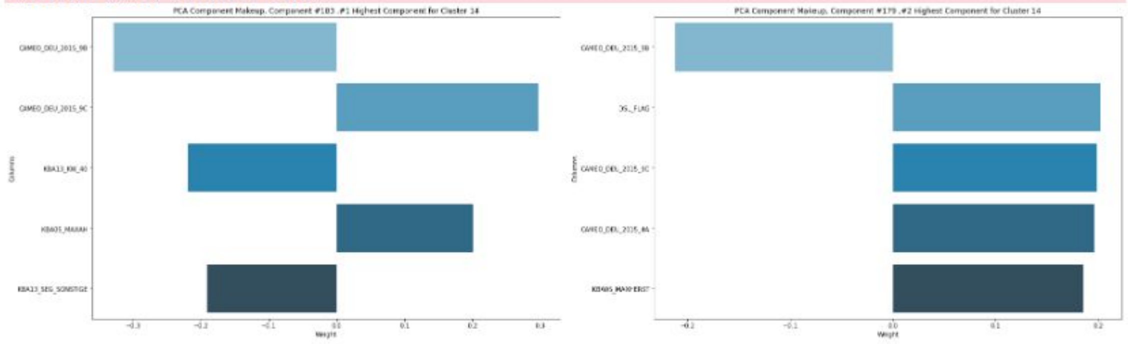
The CAMEO Classifications are a set of consumer classifications that are used internationally by organisations as part of their sales, marketing and network planning strategies.

As we can see above that cluster 5 two highest centroids components are positivily high reliable on a

- 1) 4C Category of `CAMEO_DEU_2015` which is : String Trimmer

```
plot_cluster(14,top_cluster_comp=2,top_pca_comp=5,plot_num=1)
```

```
/home/shapiro/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23: MatplotlibDeprecationWarni
ng: Passing non-integers as three-element position specification is deprecated since 3.3 and will be rem
oved two minor releases later.
/home/shapiro/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23: MatplotlibDeprecationWarni
ng: Passing non-integers as three-element position specification is deprecated since 3.3 and will be rem
oved two minor releases later.
```



As we can see above that cluster 5 two highest centroids components are positivily high reliable on a

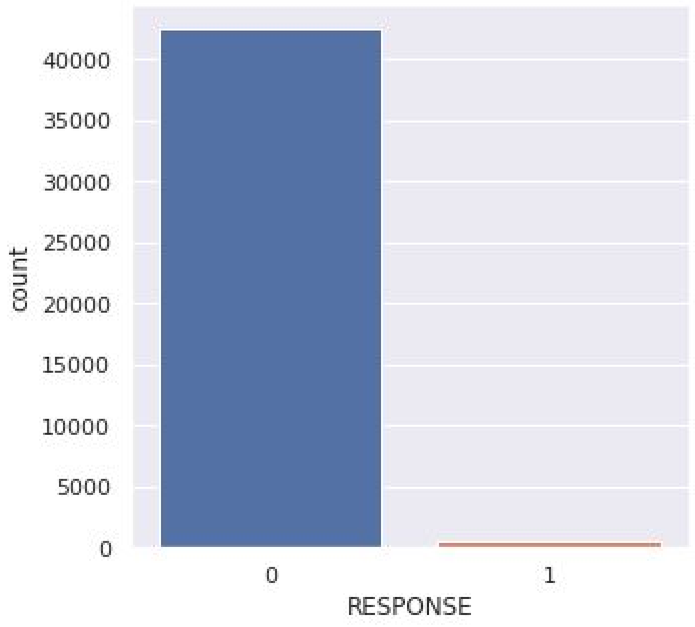
- 1) 9C Category of `CAMEO_DEU_2015` which is : Afternoon Talk Show
- 2) 4A Category of `CAMEO_DEU_2015` which is : Family Starter ### And negativly reliable on
- 1) 9B Category of `CAMEO_DEU_2015` which is : Temporary Workers

After segmenting customers , it’s the time for Phase 2, which is supervised learning to identify the potential respondents of the mail marketing campaign.

Supervised Learning :-

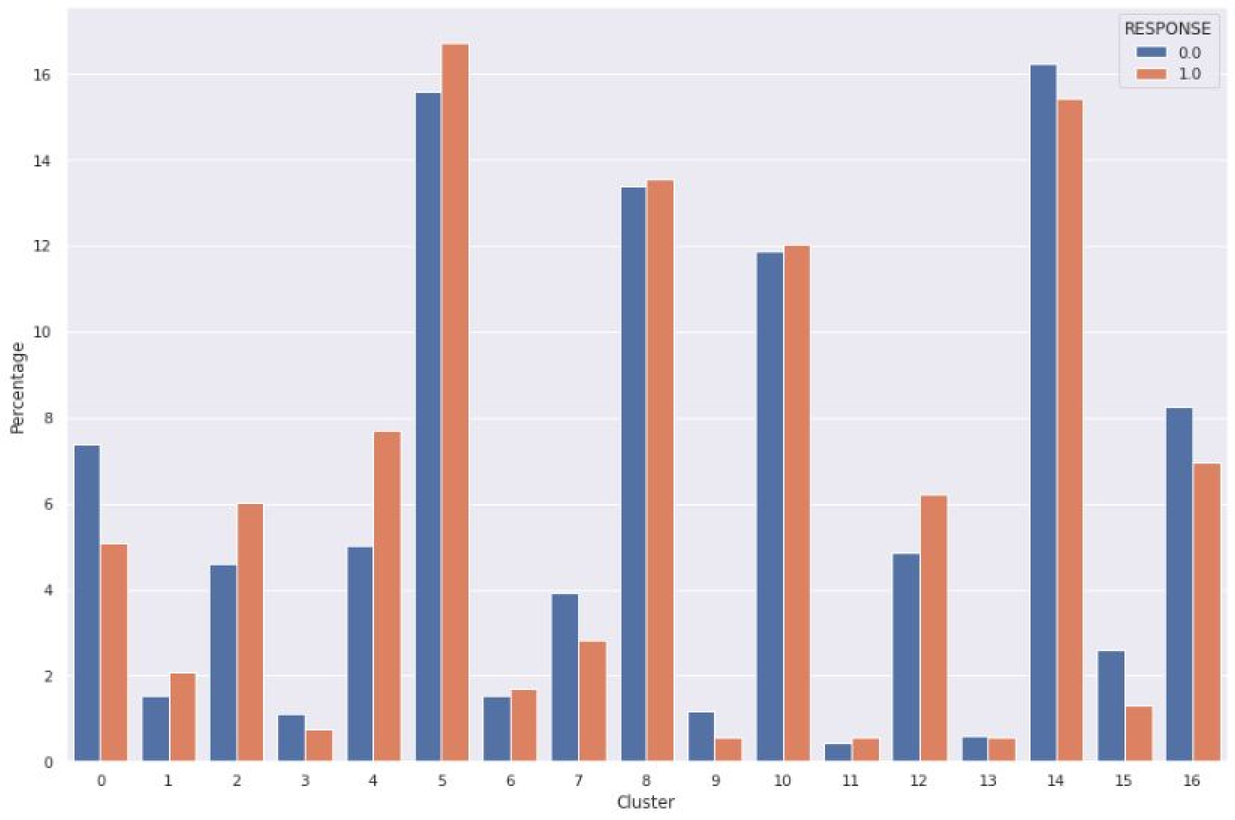
EDA :

After loading our training data we have done some exploratory data analysis , and we discovered the following .



That our training data is highly skewed ,using helper functions as transform_df and prepare training implemented in tools.py script , we did the same pipeline of PCA and K-means on training dataset as well.

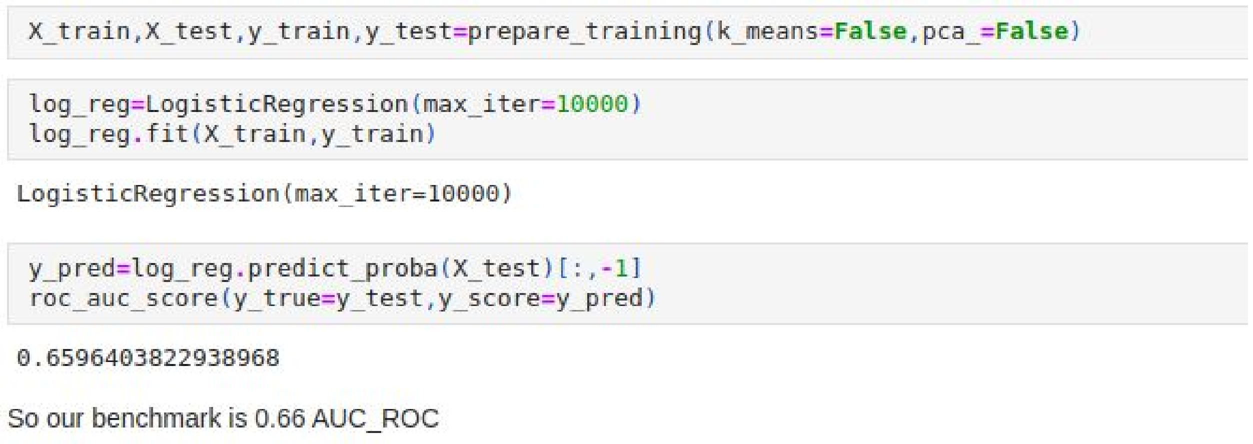
After computing the clusters in the training data, we discovered that clusters don't provide a good separation of whether the person will respond to the mail or not as we see below.



After observing the above results , we decided that any further supervised learning approach will be fitted on the original data because from previous experience i noticed that PCA and supervised learning combined for encoded categorical data isn't the best practice, also the clustering didn't add any information because as we can see above it's consistent among both the negative and positive examples with highest in cluster 14 and 5 as we observed earlier.

Classification:-

As we mentioned earlier in the proposal we fitted a logistic regression as a benchmark model which scored a 0.66 auc_roc which was our eval metric.



```
X_train,X_test,y_train,y_test=prepare_training(k_means=False,pca_=False)
```

```
log_reg=LogisticRegression(max_iter=10000)
log_reg.fit(X_train,y_train)
```

```
LogisticRegression(max_iter=10000)
```

```
y_pred=log_reg.predict_proba(X_test)[:,-1]
roc_auc_score(y_true=y_test,y_score=y_pred)
```

```
0.6596403822938968
```

So our benchmark is 0.66 AUC_ROC

As we mentioned earlier we've chosen a boosting classification algorithms because of how powerful they're also with a some slight tuning they could be robust to imbalanced data, we had the idea of oversampling in mind whether it's random or any of the SMOTs , but since the data is very high dimensional and from previous experiences SMOT don't add much information when data is high dimensional.

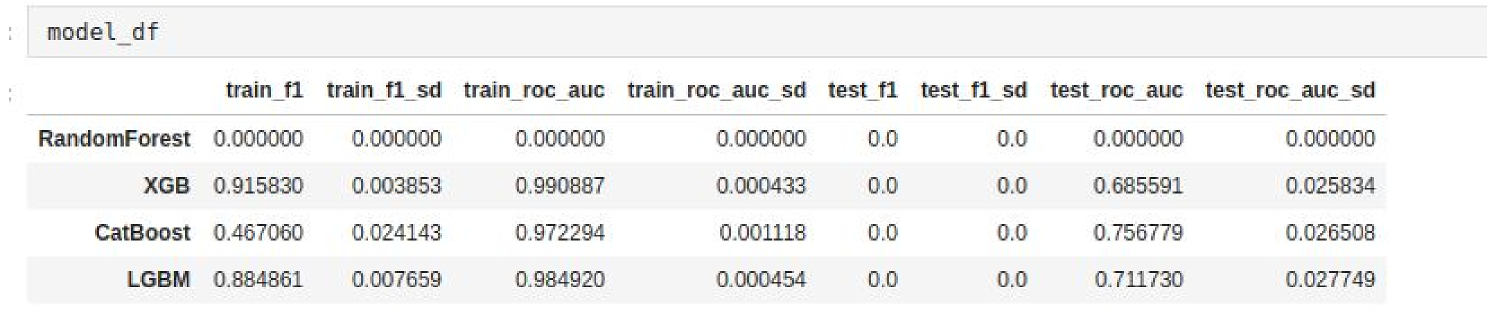
So , we had 4 Algorithms to test :-

1. XGBoost
2. CatBoost
3. LGBM
4. Autokeras StructredDataClassifier Neural Network.

We chose boosting algorithms because of how powerful they're and how appropriate for categorical data , and with a little bit of tuning they could be more powerful and robust of the imbalanced data.

So we tried all of them without tunning using cross validation on the training data.

And the results was as the following :-

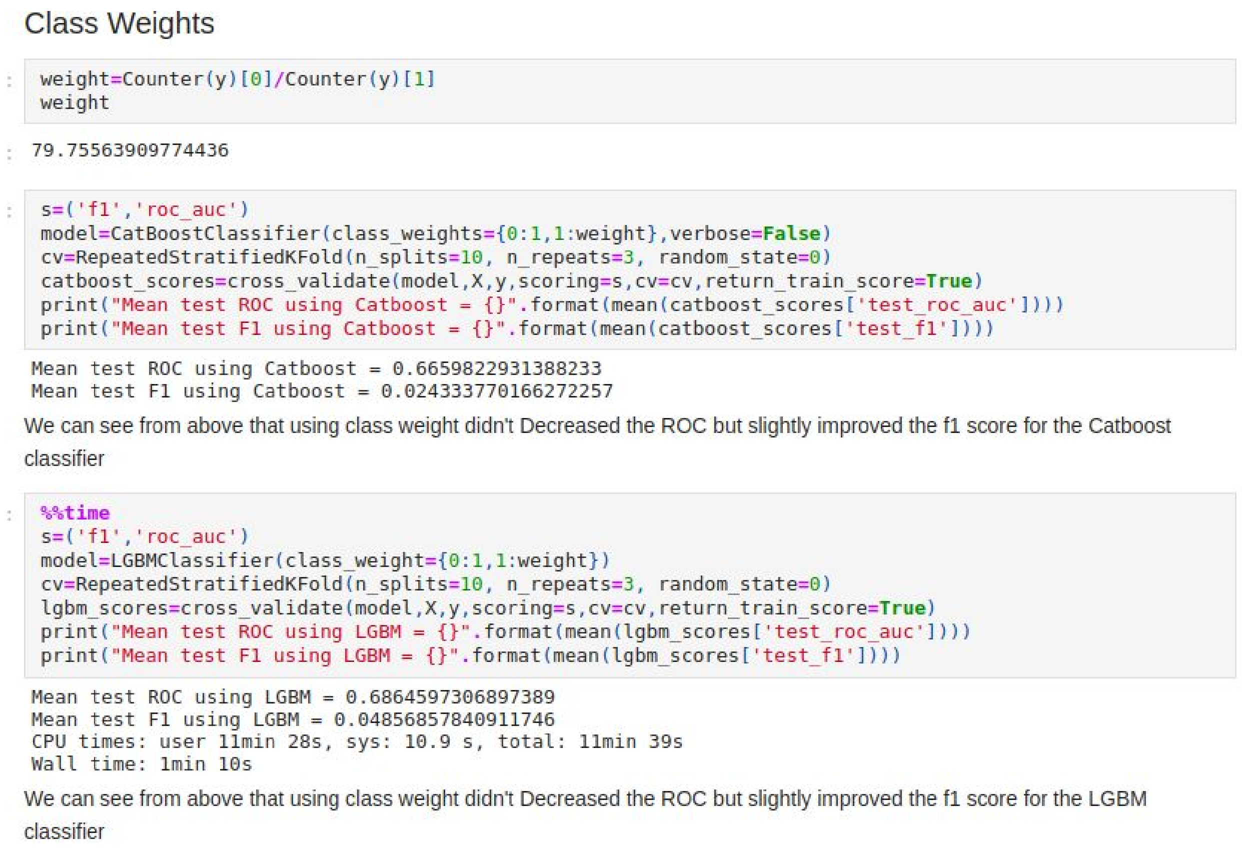


```
model_df
```

	train_f1	train_f1_sd	train_roc_auc	train_roc_auc_sd	test_f1	test_f1_sd	test_roc_auc	test_roc_auc_sd
RandomForest	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.000000
XGB	0.915830	0.003853	0.990887	0.000433	0.0	0.0	0.685591	0.025834
CatBoost	0.467060	0.024143	0.972294	0.001118	0.0	0.0	0.756779	0.026508
LGBM	0.884861	0.007659	0.984920	0.000454	0.0	0.0	0.711730	0.027749

Both Catboost and LGBM were our top performers , with acceptable test_roc_auc , and we should tune them later.

We tried to use class weight since the data is imbalanced but it didn't improve , on the contrary it made it worse.



Class Weights

```
weight=Counter(y)[0]/Counter(y)[1]
weight
```

```
79.75563909774436
```

```
s=('f1','roc_auc')
model=CatBoostClassifier(class_weights={0:1,1:weight},verbose=False)
cv=RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
catboost_scores=cross_validate(model,X,y,scoring=s,cv=cv,return_train_score=True)
print("Mean test ROC using Catboost = {}".format(mean(catboost_scores['test_roc_auc'])))
print("Mean test F1 using Catboost = {}".format(mean(catboost_scores['test_f1'])))
```

```
Mean test ROC using Catboost = 0.6659822931388233
Mean test F1 using Catboost = 0.024333770166272257
```

We can see from above that using class weight didn't Decreased the ROC but slightly improved the f1 score for the Catboost classifier

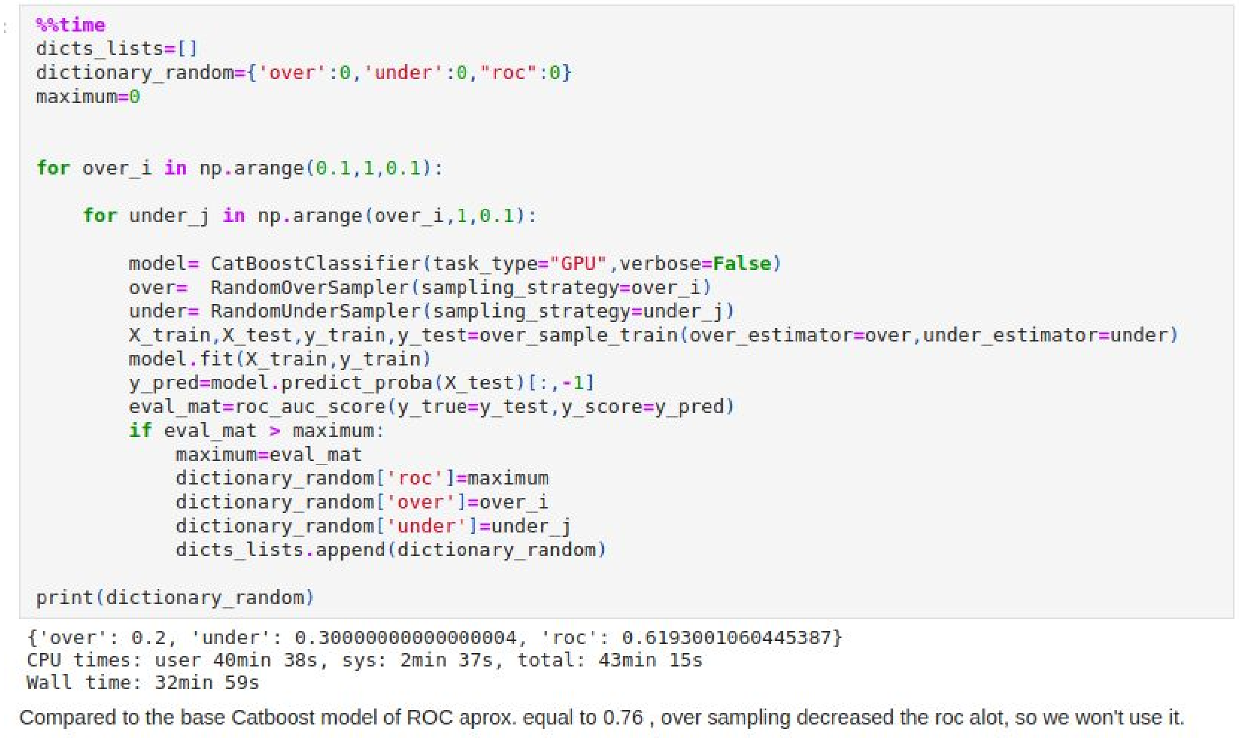
```
%%time
s=('f1','roc_auc')
model=LGBMClassifier(class_weight={0:1,1:weight})
cv=RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
lgbm_scores=cross_validate(model,X,y,scoring=s,cv=cv,return_train_score=True)
print("Mean test ROC using LGBM = {}".format(mean(lgbm_scores['test_roc_auc'])))
print("Mean test F1 using LGBM = {}".format(mean(lgbm_scores['test_f1'])))
```

```
Mean test ROC using LGBM = 0.6864597306897389
Mean test F1 using LGBM = 0.048568578409117 46
CPU times: user 11min 28s, sys: 10.9 s, total: 11min 39s
Wall time: 1min 10s
```

We can see from above that using class weight didn't Decreased the ROC but slightly improved the f1 score for the LGBM classifier

Also , we tried to use Random Oversampling of the minority class and Random UnderSampling of the majority class .

Since Catboost was out top performer we used it to test them , but as we thought earlier they didn't improve anything.



```
%%time
dicts_lists=[]
dictionary_random={'over':0,'under':0,"roc":0}
maximum=0


for over_i in np.arange(0.1,1,0.1):

    for under_j in np.arange(over_i,1,0.1):

        model= CatBoostClassifier(task_type="GPU",verbose=False)
        over=  RandomOverSampler(sampling_strategy=over_i)
        under= RandomUnderSampler(sampling_strategy=under_j)
        X_train,X_test,y_train,y_test=over_sample_train(over_estimator=over,under_estimator=under)
        model.fit(X_train,y_train)
        y_pred=model.predict_proba(X_test)[:,-1]
        eval_mat=roc_auc_score(y_true=y_test,y_score=y_pred)
        if eval_mat > maximum:
            maximum=eval_mat
            dictionary_random['roc']=maximum
            dictionary_random['over']=over_i
            dictionary_random['under']=under_j
            dicts_lists.append(dictionary_random)

print(dictionary_random)
```

```
{'over': 0.2, 'under': 0.30000000000000004, 'roc': 0.6193001060445387}
CPU times: user 40min 38s, sys: 2min 37s, total: 43min 15s
Wall time: 32min 59s
```

Compared to the base Catboost model of ROC aprox. equal to 0.76 , over sampling decreased the roc alot, so we won't use it.

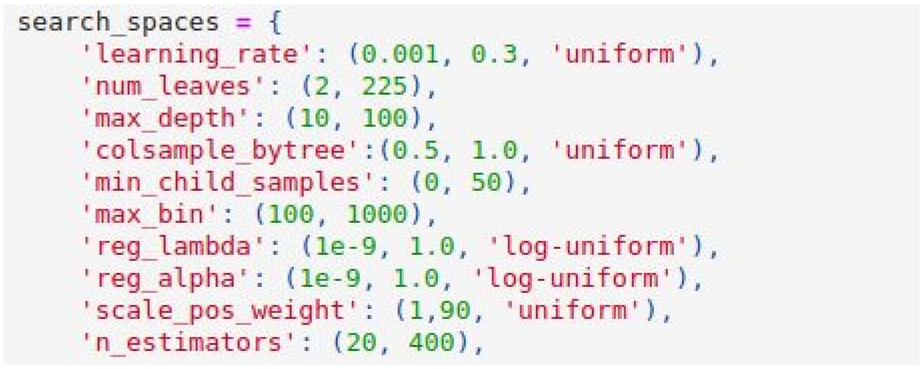
Autokeras Structured Data Classifier :-

After more than 5 hours of training and trying more than 33 different architectures , the autokeras model couldn't even get close to the CATBoost , with autokeras's best roc auc value around 0.74 and catboost 0.76.

Model Tuning :-

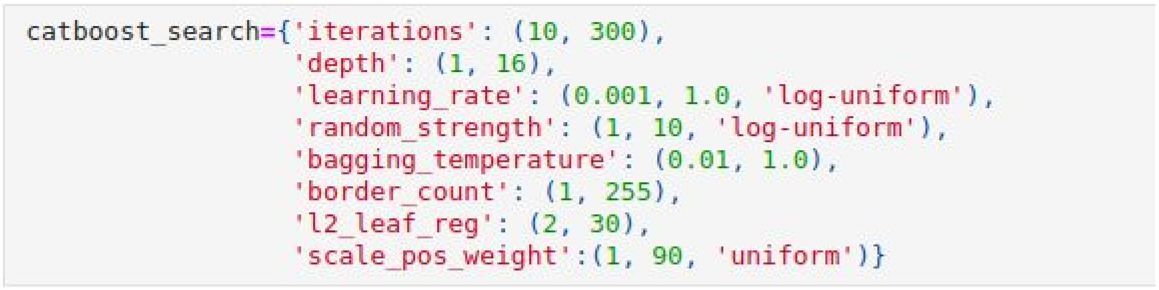
We used BayesearchCV in model tuning from skopt library, my personal opinion is tuning using bayesian methods is the best because parameters distributions are kept updating according to the posterior relations at every iterations , so when it's in the right place it keeps searching inside it.

1. We tuned LGBM and tried 100 Models , which improved the Area Under ROC Curve from 71 to 77 in only 100 Trials .



```
search_spaces = {
    'learning_rate': (0.001, 0.3, 'uniform'),
    'num_leaves': (2, 225),
    'max_depth': (10, 100),
    'colsample_bytree':(0.5, 1.0, 'uniform'),
    'min_child_samples': (0, 50),
    'max_bin': (100, 1000),
    'reg_lambda': (1e-9, 1.0, 'log-uniform'),
    'reg_alpha': (1e-9, 1.0, 'log-uniform'),
    'scale_pos_weight': (1,90, 'uniform'),
    'n_estimators': (20, 400),
```

2. After that we tuned the CATBoost model over 200 Models , but it slightly improved from 75.6 to 77.6



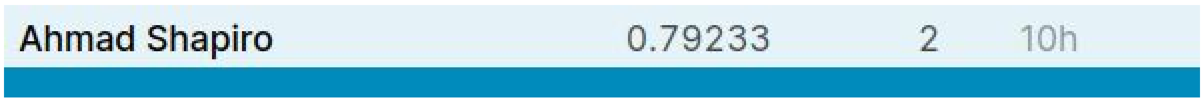
```
catboost_search={'iterations': (10, 300),
                 'depth': (1, 16),
                 'learning_rate': (0.001, 1.0, 'log-uniform'),
                 'random_strength': (1, 10, 'log-uniform'),
                 'bagging_temperature': (0.01, 1.0),
                 'border_count': (1, 255),
                 'l2_leaf_reg': (2, 30),
                 'scale_pos_weight':(1, 90, 'uniform')}
```

Model Selection :-

Model Name	Eval Metric (Aread Under ROC)
Logistics Regression (Benchmark)	0.66
Tuned CatBoost	0.776
Tuned LGBM	0.772

As we can see from above , both LGBM and CatBoost compared to the Benchmark Model (Logistic Regression) was significantly high.

Even the final performance of the CATBoost is slightly above LGBM but we went for the CATBOOST to submit the test results to kaggle.



```
Ahmad Shapiro                 0.79233        2     10h
```

And we scored 0.79 of the Area Under the ROC curve, but the evaluation is only done on some of the test dataset, so we still have room for improvement or not.

Ideas for improving the Model :-

1. Training LGBM for more iterations.
2. More feature engineering and one-hot encode the rest of the categorical features
3. Trying Google's Tab-net Neural Network.

THANK YOU !