UDACITY DATA SCIENTIST NANODEGREE

Capstone Project Report

Customer Segmentation – Arvato Financial Solutions

Thuy Pham

January 30th, 2021

Project Overview

This is final project of the Udacity Data Scientist Nanodegree. The goal of the project is to use appropriate data analytics tools and methodologies to predict potential customers for Arvato's mail order organic products.

Historical data on general population demographics, on customers and on client response to previous campaign have been provided to assist in predicting which ones from general population are likely to be good responders to Arvato's marketing campaign.

There are 3 major sections in this project:

1. Understanding business problem and data characteristics
2. Using Unsupervised Machine Learning to identify what segments of general population that match Arvato's existing customer segments
3. Using Supervised Machine Learning to predict which customers are likely to response to company's marketing campaign.

All the supporting analysis and documentation can be found at Github

Problem Statement

Marketing is crucial for the growth and sustainability of the business as it helps build company's brand, engage customers, grow revenues and increase sales. One of the key pain point of business is to understand customers and identify their needs in order to tailor campaigns to customer segments most likely to purchase products. Customer segmentation helps business plan marketing campaigns easier, focusing on certain customer groups instead of targeting the mass market, therefore more efficient in terms of time, money and other resources.

- What are the relationship between demographics of the company's existing customers and the general population of Germany?
- Which parts of the general population that are more likely to be part of the mail-order company's main customer bases, and which parts of the general population are less so
- How historical demographic data can help business to build prediction model, therefore be able to identify potential customers.

Fortunately, those business questions can be solved using analytics by involving appropriate data analytics tools and methodologies.

The approach to solve problem is to use an unsupervised machine learning algorithm KMeans to segment customers based on their demographic features, and then employ a binary classification machine learning model DecisionTreeClassifier to predict which ones of the general population are likely to be Arvato's potential customers.

Datasets and Inputs

4 datasets provided by Arvato will be explored in this project:

1. Udacity_AZDIAS_052018.csv: Demographics data for the general population of Germany; 891 211 persons (rows) x 366 features (columns).
2. Udacity_CUSTOMERS_052018.csv: Demographics data for customers of a mail-order company; 191 652 persons (rows) x 369 features (columns).
3. Udacity_MAILOUT_052018_TRAIN.csv: Demographics data for individuals who were targets of a marketing campaign; 42 982 persons (rows) x 367 (columns).
4. 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:

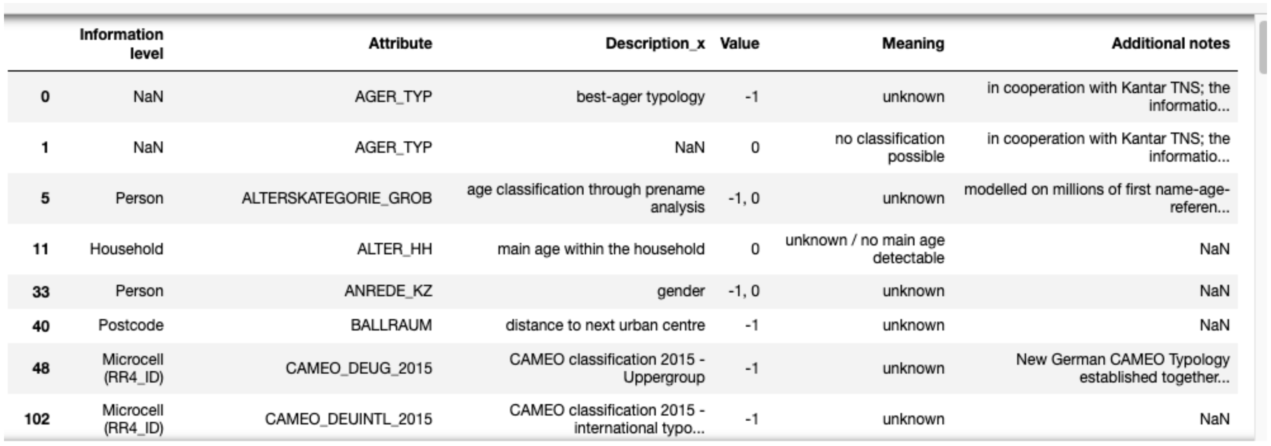
1. DIAS Information Levels - Attributes 2017.xlsx is a top-level list of attributes and descriptions, organized by informational category.
2. DIAS Attributes - Values 2017.xlsx is a detailed mapping of data values for each feature in alphabetical order

Evaluation Metrics

This is a two-class classification problem. Due to large output class imbalance, where most individuals did not respond to the mailout, the most appropriate evaluation metric is the Area Under the Curve Receiver Operating Characteristics (ROC-AUC). The curve represents a degree or measure of separability and, the higher the score the better the model is performing.

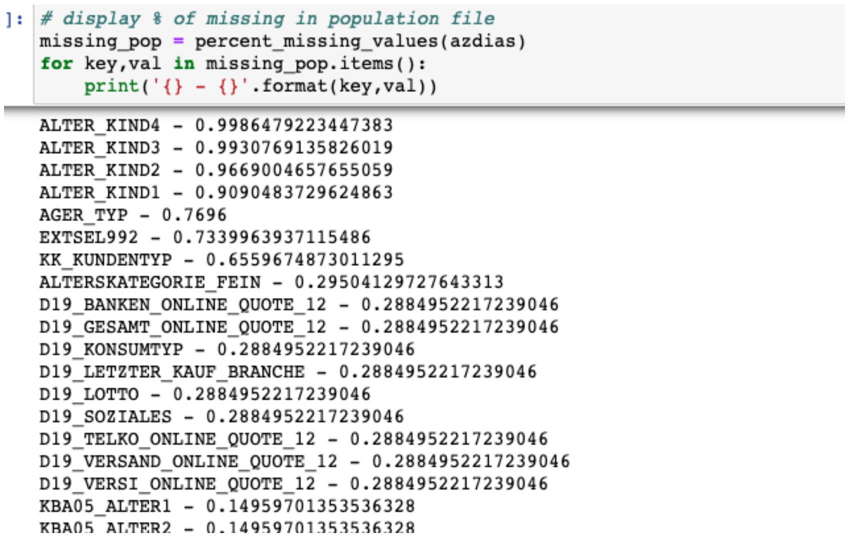
Data Exploration and Visualization

I’ve merged 2 metadata files DIAS Information Levels - Attributes 2017.xlsx and DIAS Attributes - Values 2017.xlsx to form a data dictionary for Arvato’s general population and customers demographic files.
It is interesting to see how NaN values are coded in major attributes of datasets



	Information level	Attribute	Description_x	Value	Meaning	Additional notes
0	NaN	AGER_TYP	best-ager typology	-1	unknown	in cooperation with Kantar TNS; the informatio...
1	NaN	AGER_TYP	NaN	0	no classification possible	in cooperation with Kantar TNS; the informatio...
5	Person	ALTERSKATEGORIE_GROB	age classification through prename analysis	-1, 0	unknown	modelled on millions of first name-age-referen...
11	Household	ALTER_HH	main age within the household	0	unknown / no main age detectable	NaN
33	Person	ANREDE_KZ	gender	-1, 0	unknown	NaN
40	Postcode	BALLRAUM	distance to next urban centre	-1	unknown	NaN
48	Microcell (RR4_ID)	CAMEO_DEUG_2015	CAMEO classification 2015 - Uppergroup	-1	unknown	New German CAMEO Typology established together...
102	Microcell (RR4_ID)	CAMEO_DEUINTL_2015	CAMEO classification 2015 - international typo...	-1	unknown	NaN

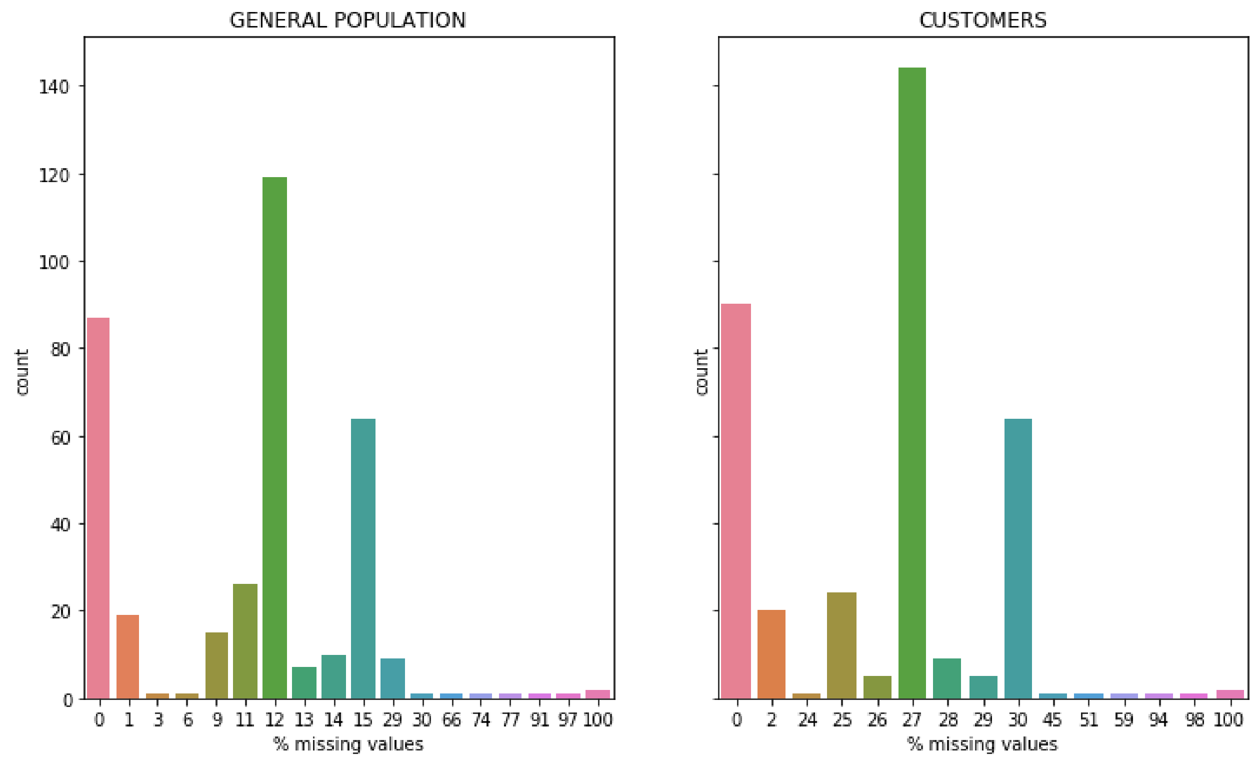
Value -1, 0 and even 9 (not displayed in screenshot above) represent ‘unknown’ .



```
]: # display % of missing in population file
   missing_pop = percent_missing_values(azdias)
   for key,val in missing_pop.items():
       print('{} - {}'.format(key,val))
```

```
ALTER_KIND4 - 0.9986479223447383
ALTER_KIND3 - 0.9930769135826019
ALTER_KIND2 - 0.9669004657655059
ALTER_KIND1 - 0.9090483729624863
AGER_TYP - 0.7696
EXTSEL992 - 0.7339963937115486
KK_KUNDENTYP - 0.6559674873011295
ALTERSKATEGORIE_FEIN - 0.29504129727643313
D19_BANKEN_ONLINE_QUOTE_12 - 0.2884952217239046
D19_GESAMT_ONLINE_QUOTE_12 - 0.2884952217239046
D19_KONSUMTYP - 0.2884952217239046
D19_LETZTER_KAUF_BRANCHE - 0.2884952217239046
D19_LOTTO - 0.2884952217239046
D19_SOZIALES - 0.2884952217239046
D19_TELKO_ONLINE_QUOTE_12 - 0.2884952217239046
D19_VERSAND_ONLINE_QUOTE_12 - 0.2884952217239046
D19_VERSI_ONLINE_QUOTE_12 - 0.2884952217239046
KBA05_ALTER1 - 0.14959701353536328
KBA05_ALTER2 - 0.14959701353536328
```

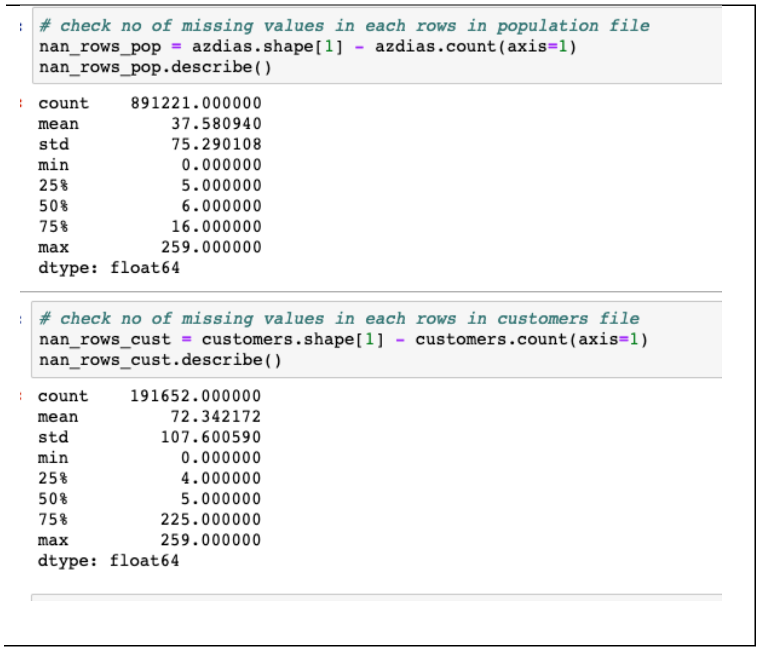
We can see that most of attributes have % of missing values less than 30%, 7 columns with more than 30% NaN are considered ‘outliers’ , therefore being removed.



Plotting the columns and associate % missing values side by side, we can see that most attributes have missing values 30% or less in both general population and customer files. Those attributes will be retained, else (greater than 30%) will be dropped from datasets.

One interesting observation from the charts is that both files have the same number of colums (~ 80) with no missing values. And the MODE % of missing values in Population file is 12%, whereas in Customer file is 27%

Let's have a look at NaN analysis at row level in the figure on the right. Population file seems to have less NaN than that in Customer file. The mean number of NaN at row level in Population is 37 whereas it is 72 in Customer file. Similarly at third IQR, it is 16 in Population file and 225 in Customer file.

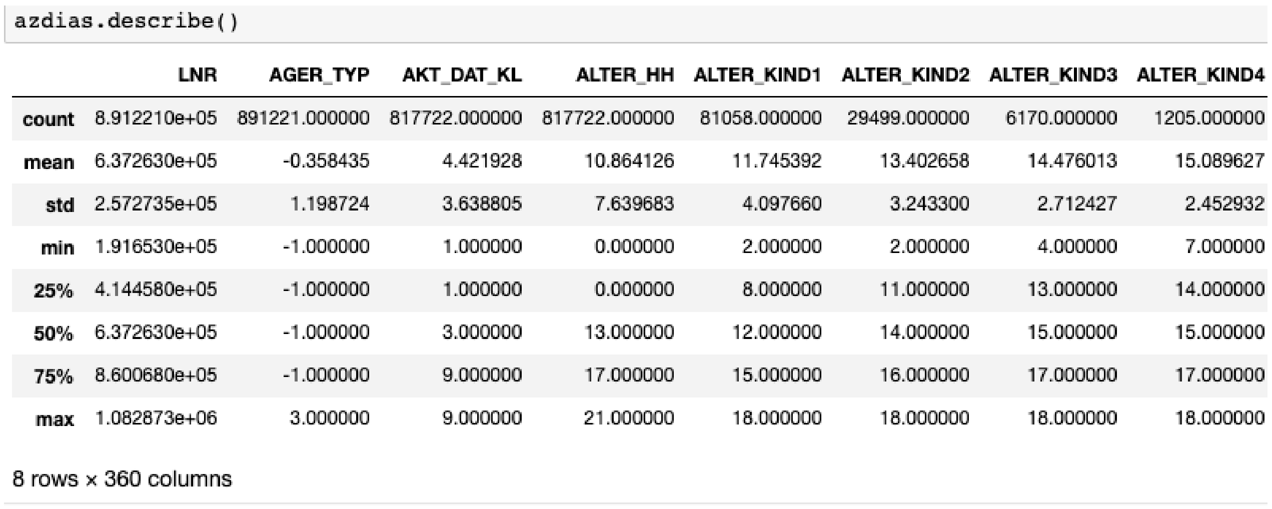


```
# check no of missing values in each rows in population file
nan_rows_pop = azdias.shape[1] - azdias.count(axis=1)
nan_rows_pop.describe()
```

```
count    891221.000000
mean         37.580940
std          75.290108
min           0.000000
25%           5.000000
50%           6.000000
75%          16.000000
max         259.000000
dtype: float64
```

```
# check no of missing values in each rows in customers file
nan_rows_cust = customers.shape[1] - customers.count(axis=1)
nan_rows_cust.describe()
```

```
count    191652.000000
mean         72.342172
std         107.600590
min           0.000000
25%           4.000000
50%           5.000000
75%         225.000000
max         259.000000
dtype: float64
```



```
azdias.describe()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4
count	8.912210e+05	891221.000000	817722.000000	817722.000000	81058.000000	29499.000000	6170.000000	1205.000000
mean	6.372630e+05	-0.358435	4.421928	10.864126	11.745392	13.402658	14.476013	15.089627
std	2.572735e+05	1.198724	3.638805	7.639683	4.097660	3.243300	2.712427	2.452932
min	1.916530e+05	-1.000000	1.000000	0.000000	2.000000	2.000000	4.000000	7.000000
25%	4.144580e+05	-1.000000	1.000000	0.000000	8.000000	11.000000	13.000000	14.000000
50%	6.372630e+05	-1.000000	3.000000	13.000000	12.000000	14.000000	15.000000	15.000000
75%	8.600680e+05	-1.000000	9.000000	17.000000	15.000000	16.000000	17.000000	17.000000
max	1.082873e+06	3.000000	9.000000	21.000000	18.000000	18.000000	18.000000	18.000000

8 rows × 360 columns

Most attributes in general population and customer files are of numeric datatypes, only few categorical variables. As you can see in the screenshot of azdias file above, 360 out of 366 variables are numeric.

Data Pre-processing

Data cleansing

Data cleansing plays an important role as it improves data quality, therefore better prediction. The followings have been performed:

- drop rows with more than 75% missing values
- drop columns with more than 70% missing values
- drop customer id column
- drop categorical columns (only a few and not worth to employ Encoding technique)
- drop 3 columns exist in Customer file but not exist in Population file
- for numeric variables, replace NaN with values implying 'unknown' in data dictionary, in this case is -1

Feature scaling

With 98% of variables are numeric, feature scaling is essential preprocessing step, especially for KMeans. This distance-based algorithm is affected by the scale of variables.

There are many debates on StackExchange or StockOverflow about what Scaler should be employed. In this exercise I will use StandardScaler with default parameters:

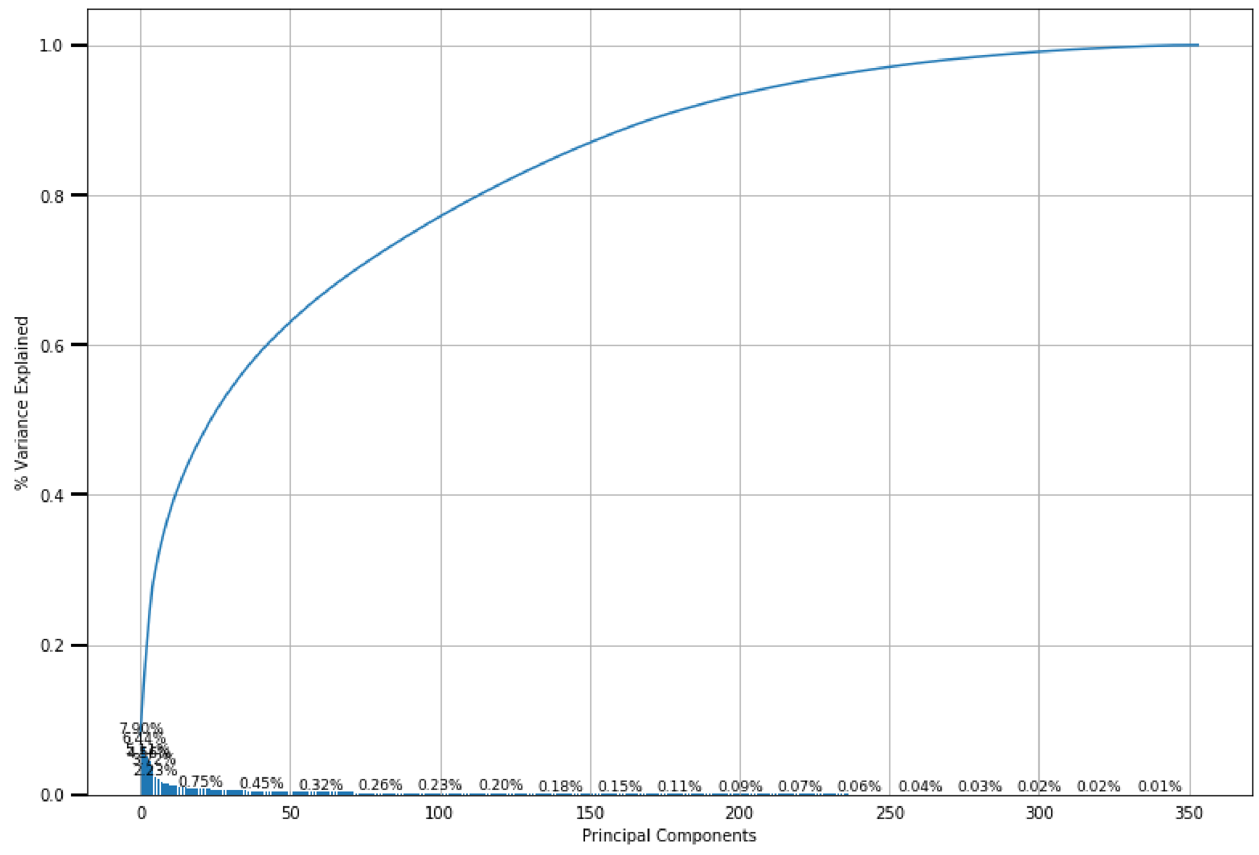
```
scaler = StandardScaler()

df_scaled = pd.DataFrame(scaler.fit_transform(df), index=df.index, columns=df.columns)
```

The output of Sklearn.preprocessing.StandardScaler is an array, however I've converted it into Pandas dataframe as I need the column names for reverse engineering later.

Feature reduction

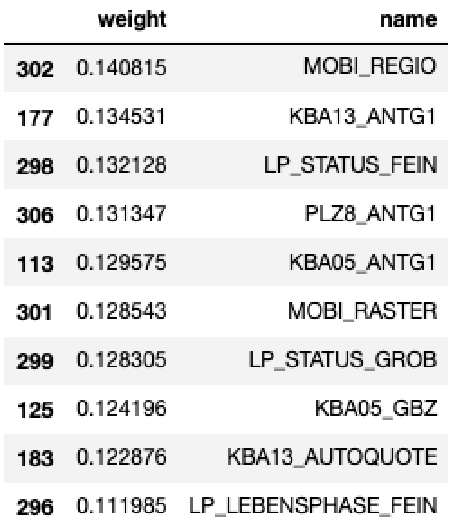
Due to the huge number of explanatory variables (350), and many of them may not contribute to prediction of target variable. I've uses sklearn.decomposition.PCA to limit the number of components being used for machine learning.

Let's have a look at the chart below. The choice of 150 principal components seems reasonable. It reduces more than half of features while still have more than 80% explanation power.



First 10 records of First PCA

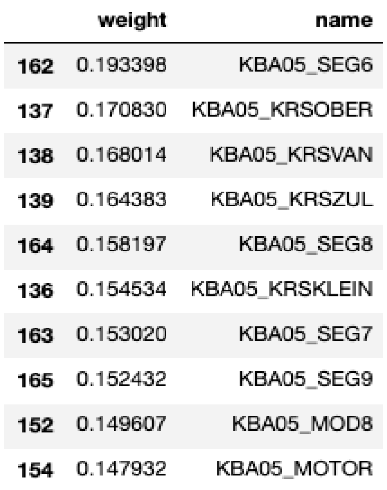
The first principal components refer to social status and lifestyles of individuals (Mobility, Social Status, 1-2 family houses, number of buildings, share of cars, lifestyles)



	weight	name
302	0.140815	MOBI_REGIO
177	0.134531	KBA13_ANTG1
298	0.132128	LP_STATUS_FEIN
306	0.131347	PLZ8_ANTG1
113	0.129575	KBA05_ANTG1
301	0.128543	MOBI_RASTER
299	0.128305	LP_STATUS_GROB
125	0.124196	KBA05_GBZ
183	0.122876	KBA13_AUTOQUOTE
296	0.111985	LP_LEBENSPHASE_FEIN

First 10 records of Second PCA

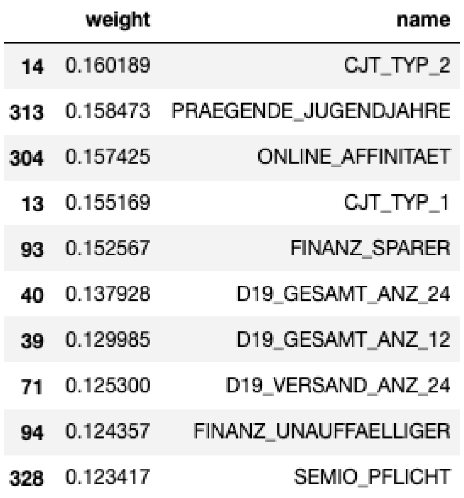
This Is interesting cluster where the attributes all start with ‘KBA05’ which are vehicle related features



	weight	name
162	0.193398	KBA05_SEG6
137	0.170830	KBA05_KRSOBER
138	0.168014	KBA05_KRSVAN
139	0.164383	KBA05_KRSZUL
164	0.158197	KBA05_SEG8
136	0.154534	KBA05_KRSKLEIN
163	0.153020	KBA05_SEG7
165	0.152432	KBA05_SEG9
152	0.149607	KBA05_MOD8
154	0.147932	KBA05_MOTOR

First 10 records of Third PCA

Unfortunately, many of features in this group are not in data dictionary, look like they are finance related features.



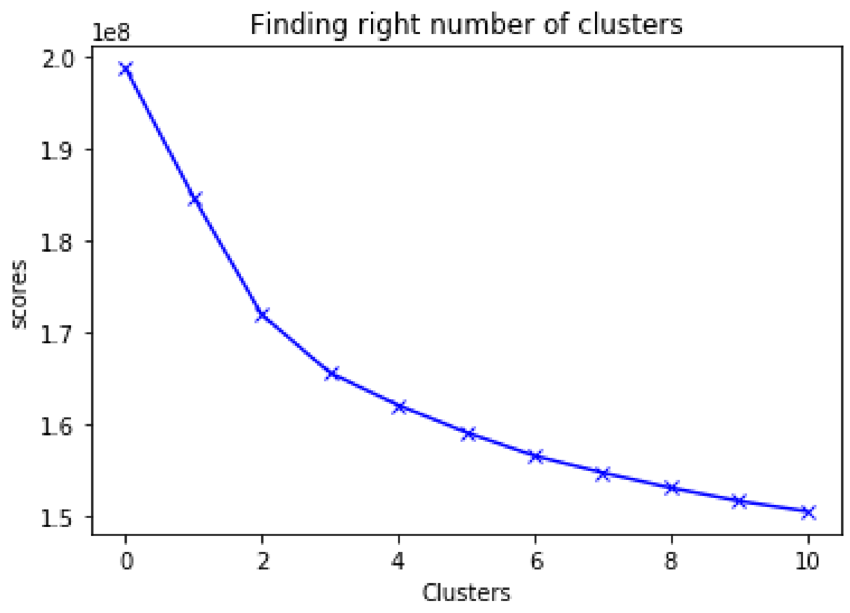
	weight	name
14	0.160189	CJT_TYP_2
313	0.158473	PRAEGENDE_JUGENDJAHRE
304	0.157425	ONLINE_AFFINITAET
13	0.155169	CJT_TYP_1
93	0.152567	FINANZ_SPARER
40	0.137928	D19_GESAMT_ANZ_24
39	0.129985	D19_GESAMT_ANZ_12
71	0.125300	D19_VERSAND_ANZ_24
94	0.124357	FINANZ_UNAUFFAELLIGER
328	0.123417	SEMIO_PFLICHT

Unsupervised Machine Learning

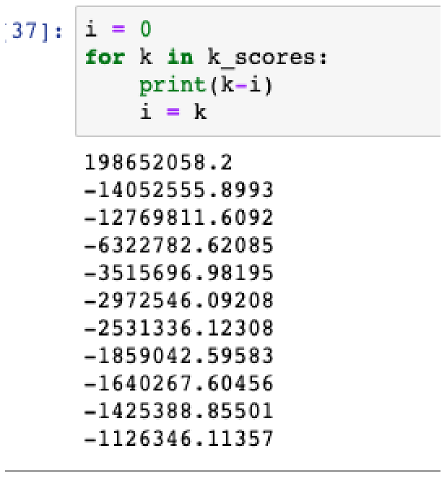
KMeans Algorithm

The reason I chose Kmeans algorithm is that it is relatively simple to implement and scale to large datasets.

Let’s start with finding the optimum number of clusters



The plot above suggests number of clusters between 4 and 6, let pick number of clusters using calculation

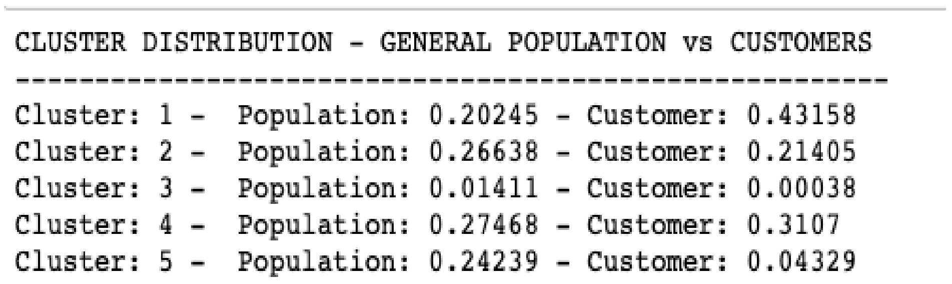


```
[37]: i = 0
      for k in k_scores:
          print(k-i)
          i = k

198652058.2
-14052555.8993
-12769811.6092
-6322782.62085
-3515696.98195
-2972546.09208
-2531336.12308
-1859042.59583
-1640267.60456
-1425388.85501
-1126346.11357
```

The score difference between k and (k-1) reducing as number of clusters increasing. From figure on the left, we can see starting from k=5 , the score difference (-3515696.98195) become smaller.

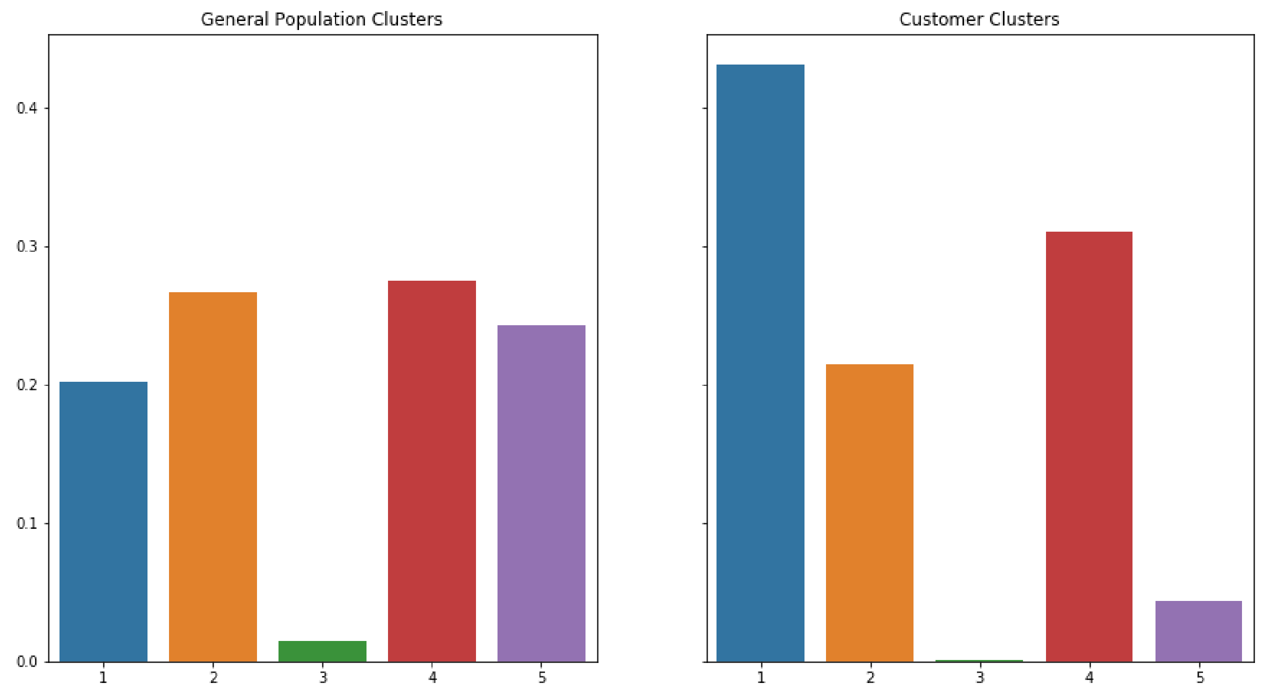
So the clusters = 5 seems to be the right choice.

Apply KMeans(5) on PCA datasets I have the results below:



```
CLUSTER DISTRIBUTION - GENERAL POPULATION vs CUSTOMERS
------------------------------------------------------
Cluster: 1 -  Population: 0.20245 - Customer: 0.43158
Cluster: 2 -  Population: 0.26638 - Customer: 0.21405
Cluster: 3 -  Population: 0.01411 - Customer: 0.00038
Cluster: 4 -  Population: 0.27468 - Customer: 0.3107
Cluster: 5 -  Population: 0.24239 - Customer: 0.04329
```

Customer Segmentation Report



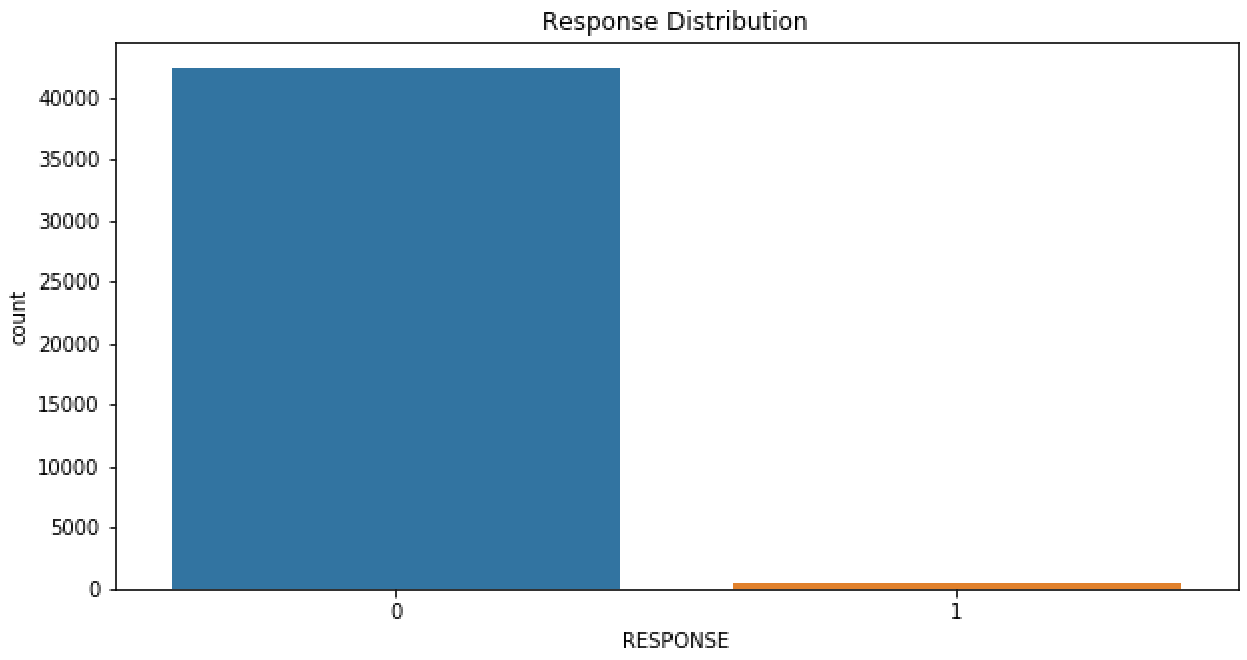
Comparing the proportion of persons in each cluster of the general population and Avarto's customers, we can see that there are big differences in Cluster 1 and Cluster 5.

The higher proportion of persons in a cluster for the customer data compared to that of general population suggests the people in that cluster are likely to be target audience for the company, because that population possess the characteristics of Arvato's customers. That means the population in Cluster 1 and Cluster4 are more likely to be part of the mail-order company's main customer base, the Cluster 3 and Cluster 5 of the general population are less so. There is not much difference in Cluster 2 , this suggest part of this population may be company's potential customers.

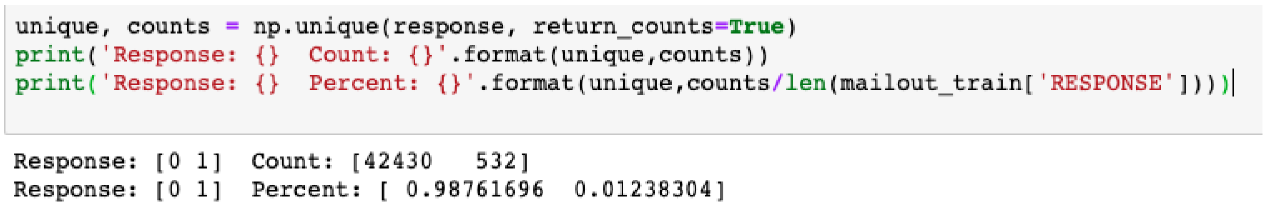
Supervised Machine Learning

Data Exploration

Given mailout_train and mailout_test are similar to 'customers' file which have been thoroughly analysed in previous sections, my focus now is on distribution of 'RESPONSE' values in mailout_train dataset



We can clearly see there is huge imbalance in domain values of attribute 'RESPONSE', let check out the exact counts and proportion in each class below:



```
unique, counts = np.unique(response, return_counts=True)
print('Response: {}  Count: {}'.format(unique,counts))
print('Response: {}  Percent: {}'.format(unique,counts/len(mailout_train['RESPONSE'])))
```

```
Response: [0 1]  Count: [42430   532]
Response: [0 1]  Percent: [ 0.98761696  0.01238304]
```

The percentage of individuals who has responded is 1.24% compared to 98.76% who did not respond.

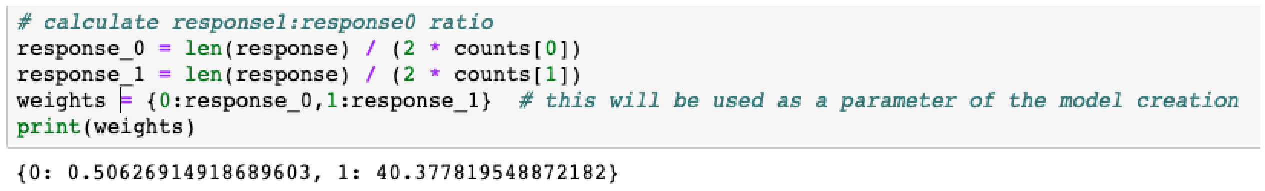
Data preparation like remove NaN, impute NaN, drop columns, scaling will be similar to what I've done for azdias and customers files.

Metric

As illustrated in previous section, the individual has not responded (0) is ~8 times more than ones who responded (1). Therefore, accuracy is a poor metric to use, and the Receiver Operating Characteristic Area Under the Curve (ROC AUC) will be used instead. The higher ROC AUC score, the better model performs. In other words, the closer ROC AUC is to 1, the better prediction who is likely to respond or not respond

Base Models

Given the imbalance of 'Response', I've calculated the class weight which will be input for my models where applicable

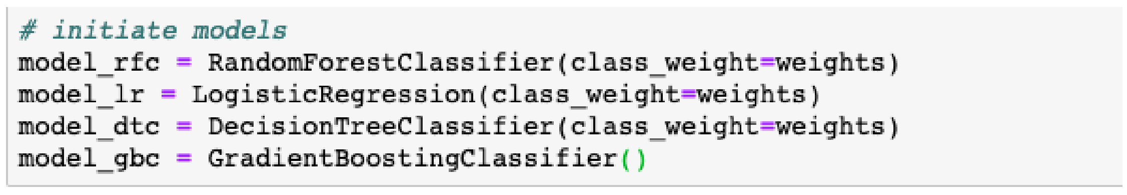


```
# calculate response1:response0 ratio
response_0 = len(response) / (2 * counts[0])
response_1 = len(response) / (2 * counts[1])
weights = {0:response_0,1:response_1}  # this will be used as a parameter of the model creation
print(weights)
```

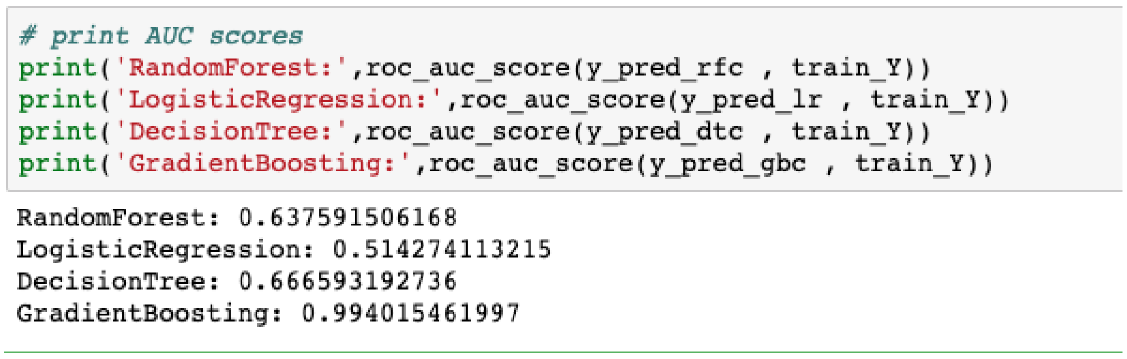
```
{0: 0.50626914918689603, 1: 40.377819548872182}
```

My selected algorithms are:

1. RandomForestClassifier
2. LogisticRegression
3. DecisionTreeClassifier
4. GradientBoostingClassifier



```
# initiate models
model_rfc = RandomForestClassifier(class_weight=weights)
model_lr = LogisticRegression(class_weight=weights)
model_dtc = DecisionTreeClassifier(class_weight=weights)
model_gbc = GradientBoostingClassifier()
```



```
# print AUC scores
print('RandomForest:',roc_auc_score(y_pred_rfc , train_Y))
print('LogisticRegression:',roc_auc_score(y_pred_lr , train_Y))
print('DecisionTree:',roc_auc_score(y_pred_dtc , train_Y))
print('GradientBoosting:',roc_auc_score(y_pred_gbc , train_Y))
```

```
RandomForest: 0.637591506168
LogisticRegression: 0.514274113215
DecisionTree: 0.666593192736
GradientBoosting: 0.994015461997
```

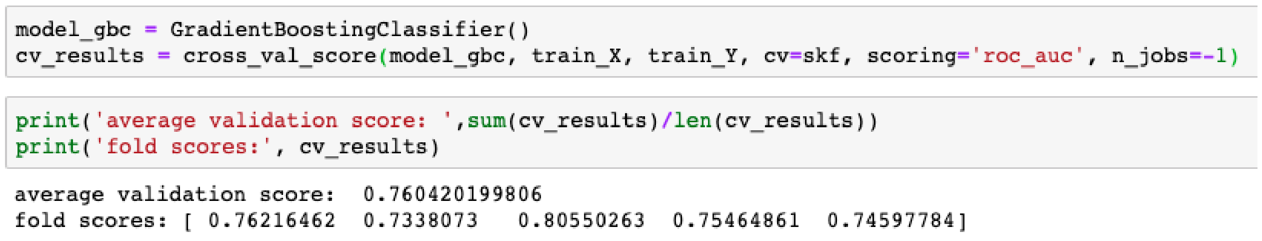
Model Evaluation and Validation

Given the AUC ROC traing score in precious section. I picked GradientBoostingClassifier to explore and further fine tuning
The approach for GradientBoostingClassifier evaluation and validation below:

1) validate base model using StratifiedKfold (5) validation
2) fine tune hyperparameters using GridSearchCV
3) retrain the model using best parameters from step above
4) apply StratifiedKfold validation on tuned model
5) compare validation before and after tuning.

The hyperparameters using for GridSearchCV below:
param_grid = {
'learning_rate: [0.05, 0.1, 0.15],
'max_depth':[3,5,8],
'n_estimators':['50,100,150],
'max_features':['log2','sqrt'] }

BASE MODEL VALIDATION RESULT



```
model_gbc = GradientBoostingClassifier()
cv_results = cross_val_score(model_gbc, train_X, train_Y, cv=skf, scoring='roc_auc', n_jobs=-1)
```

```
print('average validation score: ',sum(cv_results)/len(cv_results))
print('fold scores:', cv_results)
```

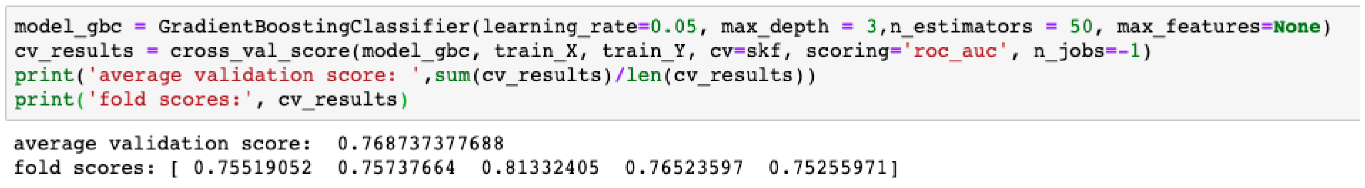
```
average validation score:  0.760420199806
fold scores: [ 0.76216462  0.7338073   0.80550263  0.75464861  0.74597784]
```

TUNED MODEL VALIDATION RESULT

The parameters that give the best validation scores when experimenting difference hyperparameter values are:

learning_rate = 0.05
max_depth = 3
n_estimators = 50
max_features = none

I will retrain the model using hyperparameters above. The validation scores below:



```
model_gbc = GradientBoostingClassifier(learning_rate=0.05, max_depth = 3,n_estimators = 50, max_features=None)
cv_results = cross_val_score(model_gbc, train_X, train_Y, cv=skf, scoring='roc_auc', n_jobs=-1)
print('average validation score: ',sum(cv_results)/len(cv_results))
print('fold scores:', cv_results)
```

```
average validation score:  0.768737377688
fold scores: [ 0.75519052  0.75737664  0.81332405  0.76523597  0.75255971]
```

From the scores above we can see that GradientBoostingClassifier give consistent scores across 5 folds

There is slight improvement after tuning parameters , so the retrained model will be used to predict 'RESPONSE' in mailout_test file

Justification

It is pain point when tuning hyperparameters using GridSearchCV, it took forever to run. So I had to kill it and manually train the model with different hyperparameters one by one. I believe the solution is reasonably adequate as I have experimented with different models, different hyperparameters, different validation techniques, thoroughly analyzing, wrangling and scaling data.
I think the validation score of 0.7687 is quite OK, in addition the consistency of scores across 5 folds give me confident that my model is robust against small perturbations in the training data, therefore reliable in predicting individuals' responses.

Reflection

The project gives me the opportunity to expand and apply my knowledge on real-world business problems. I've gone through the end-to-end solution journey starting from business understanding, data understanding and then using unsupervised machine learning to cluster customers, following by exploring 4 supervised learning classification models, experiment various validation and hyperparameter tuning techniques, and finally built a model that I am comfortable with. Of course, there are still lots of rooms for improvements. The first one is to use Pipeline for a more modular codes, another one could be feature engineering and lastly using more charts/diagrams to visualize results.

Works Cited

Brownlee, J. (2020, Aug 15). 8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset

Sklearn ROC AUC Score documentaion https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

Dogan, S. (2020 Apr 13). Why scree plot is important in PCA

KFold Cross Validation in Python https://www.askpython.com/python/examples/k-fold-cross-validation