Machine Learning Engineer Nanodegree

Capstone Project Proposal

Bertlesmen-Arvato Customer Segementation Project

J. P. Bedran

1. PROJECT OVERVIEW

Machine learning techniques are widely used these days to target potential customers by companies across all fields. Technique such as clustering has proven to be extremely helpful in this area. Arvato Financial Solutions has paired up with Udacity for their nanodegree program to create a customer segmentation report for them and predict which individuals are most likely to become a customer of their company.

The data that I will use has been provided by Udacity's partners at Bertelsmann Arvato Analytics, and represents a real-life data science task.

There are four data files associated with 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).

.

1.2 PROBLEM STATEMENT

As stated by Timo Reis, our main goal with this project is to acquire more customers efficiently for our client utilizing our technical expertise and the datasets provided by Arvato and German govt.

I will start by analyzing the first file, AZDIAS, which contains demographics for the general population. This process involves the cleaning and pre-preparation of the data. I will do the same for the customers data, although with not the same level of analysis. In AZDIAS file I will perform customer segmentation with PCA and KMeans, clustering the data so I can identify segments of the population that are compatible with the mail-order company.

I will then utilize supervised learning techniques to predict which individuals are most likely to become a customer of the company. As a part of this I will try out various classifiers like Logistic Regression, Random Forest and Gradient Boosting, the Regressor and Classifier of GB. To aid in the analysis of finding the best model I will use Grid Search.

1.3 Evaluation Metrics

The evaluation metric for this problem that I have chosen is AUC for the ROC (receiver operating characteristic curve) curve.

- True Positive Rate: This is also known as recall and is defined as follows:

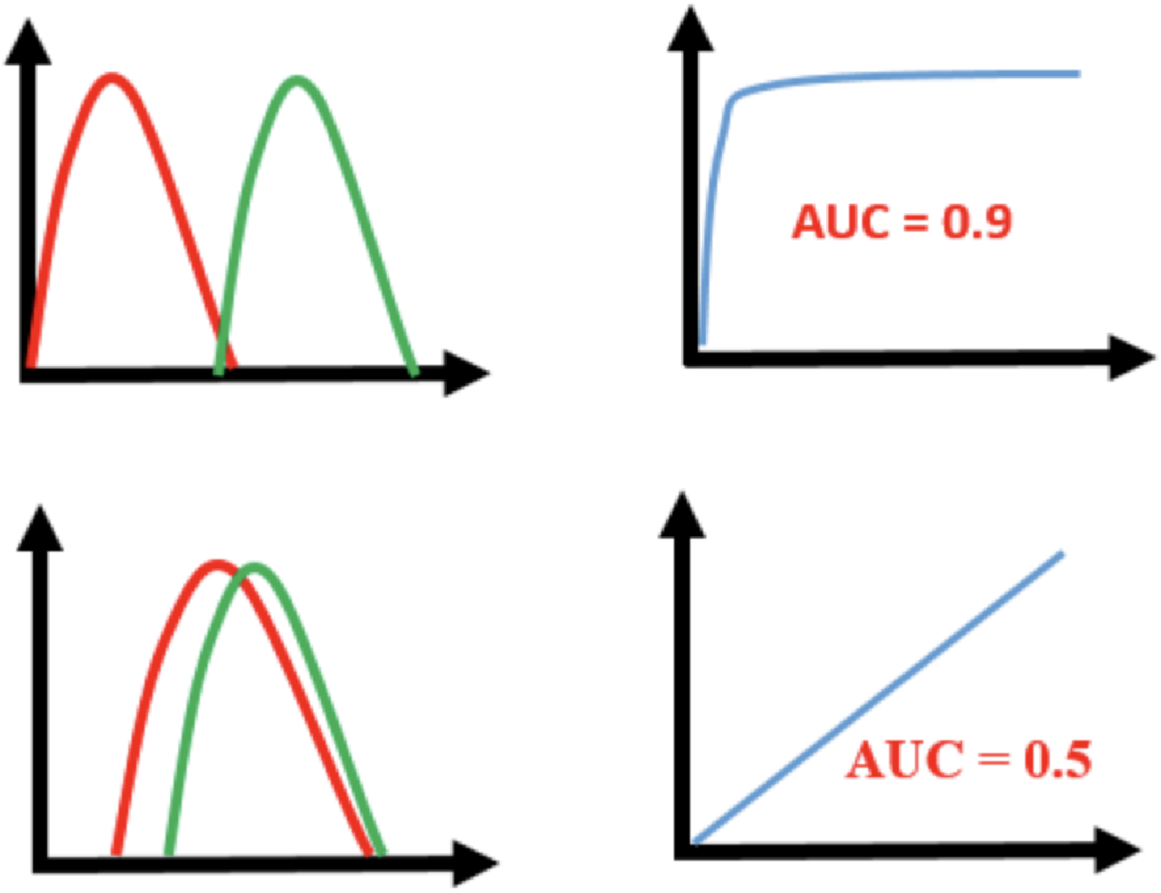
$$TPR = \frac{TP}{TP + FN}$$

- False Positive Rate: This is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

The ROC curve plots TPR vs FPR at different points of classification thresholds. Lowering the classification threshold marks more item as positive which increases both False Positives and True Positives.

AUC is the area under the ROC curve. This gives us a good idea about the performance of the model. Let us take a few examples to see how the AUC varies according to the performance of the model.



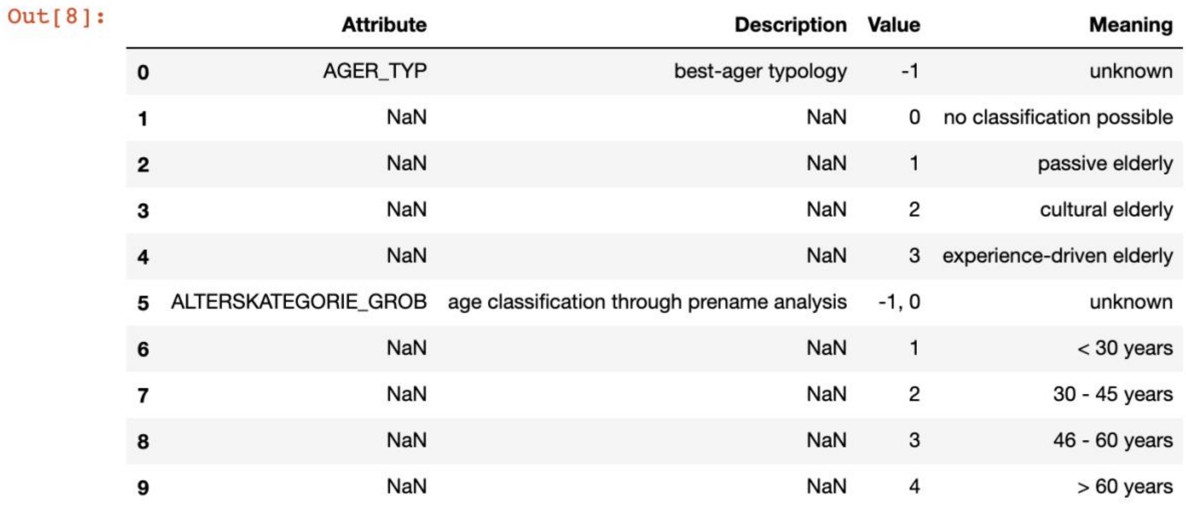
The red distribution in the example above represents the individual who did not respond to the campaign and the green distribution represents the individuals who responded to the campaign. The first model does a good job of distinguishing the positive and negative values. Therefore the AUC score is high as the area under the ROC curve is high. The second model does not distinguish the values and as a result it has overlapping values and therefore AUC score is low.

2. DATA ANALYSIS

2.1 DATA EXPLORATION AND VISUALIZATION

2.1.1 DIAS Attributes - Values 2017.xlsx

Provides detailed description of the values in the features in AZDIAS file



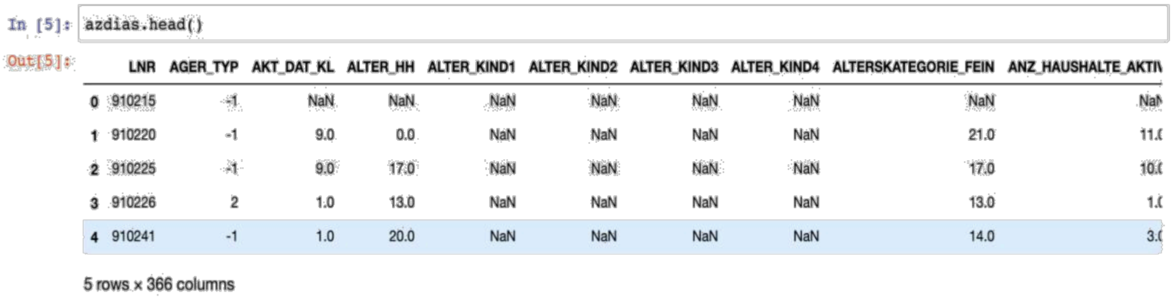
Out[8]:

	Attribute	Description	Value	Meaning
0	AGER_TYP	best-ager typology	-1	unknown
1	NaN	NaN	0	no classification possible
2	NaN	NaN	1	passive elderly
3	NaN	NaN	2	cultural elderly
4	NaN	NaN	3	experience-driven elderly
5	ALTERSKATEGORIE_GROB	age classification through prename analysis	-1, 0	unknown
6	NaN	NaN	1	< 30 years
7	NaN	NaN	2	30 - 45 years
8	NaN	NaN	3	46 - 60 years
9	NaN	NaN	4	> 60 years

We can see that there are many values in the attribute column that are NaN. I will later use ffill method to fill these values with the previous value.

2.1.2 Udacity_AZDIAS_052018.csv

This is demographics data for the general population of Germany which has 891211 persons (rows) x 366 features (columns). Below is sample of this dataset.



```
In [5]: azdias.head()
```

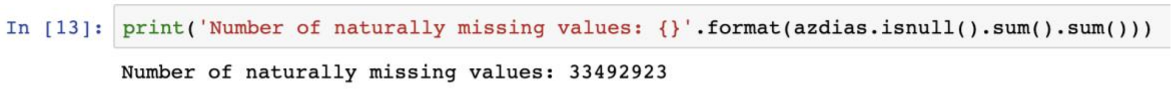
Out[5]:

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTI
0	910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	21.0	11.
2	910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	17.0	10.
3	910226	2	1.0	13.0	NaN	NaN	NaN	NaN	13.0	1.
4	910241	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0	3.

5 rows × 366 columns

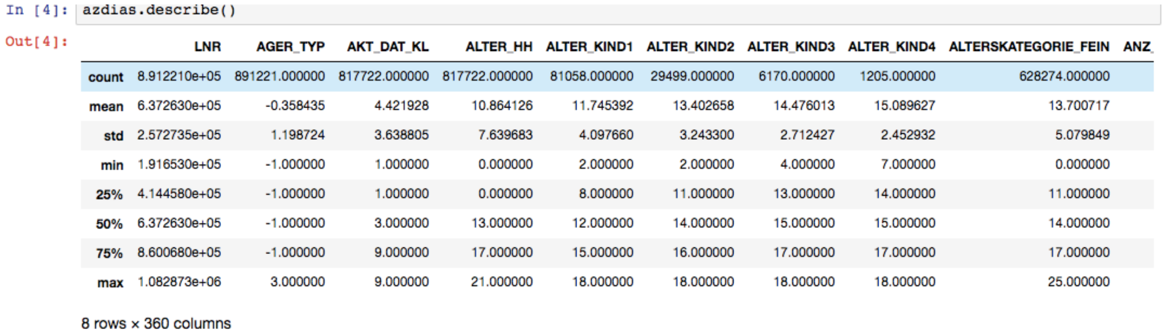
In a initial analysis, we can see that values that are marked as unkown or missing in DIAS ATTRIBUTES are not labeled as NaN in the AZDIAS file. I will later on create a Dataframe as to be able to apply a function so that we can alter these values throught the dataset, therefore giving the true number of missing values in the dataset

Initially there are already 33492923 values that are missing from AZDIAS.



```
In [13]: print('Number of naturally missing values: {}'.format(azdias.isnull().sum().sum()))

         Number of naturally missing values: 33492923
```

I will evaluate statistical data from the dataset



```
In [4]: azdias.describe()
Out[4]:
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_
count	8.912210e+05	891221.000000	817722.000000	817722.000000	81058.000000	29499.000000	6170.000000	1205.000000	628274.000000	
mean	6.372630e+05	-0.358435	4.421928	10.864126	11.745392	13.402658	14.476013	15.089627	13.700717	
std	2.572735e+05	1.198724	3.638805	7.639683	4.097660	3.243300	2.712427	2.452932	5.079849	
min	1.916530e+05	-1.000000	1.000000	0.000000	2.000000	2.000000	4.000000	7.000000	0.000000	
25%	4.144580e+05	-1.000000	1.000000	0.000000	8.000000	11.000000	13.000000	14.000000	11.000000	
50%	6.372630e+05	-1.000000	3.000000	13.000000	12.000000	14.000000	15.000000	15.000000	14.000000	
75%	8.600680e+05	-1.000000	9.000000	17.000000	15.000000	16.000000	17.000000	17.000000	17.000000	
max	1.082873e+06	3.000000	9.000000	21.000000	18.000000	18.000000	18.000000	18.000000	25.000000	

8 rows × 360 columns

From the evaluation above we can come to a conclusion that the data is not Normally Distributed, we can see that by the discrepancies of values in STD and MEAN between the features. I will have to process the thoroughly as to normalize it.

I will be identifying columns with over 20% missing data and will be dropping them. Over 20% represents a good statistical value, and I will still not be dropping too many columns as to create a statistical misrepresentation of data.. Dropping columns should be as least as possible as to maintain variance and originality of data as close to original as possible.

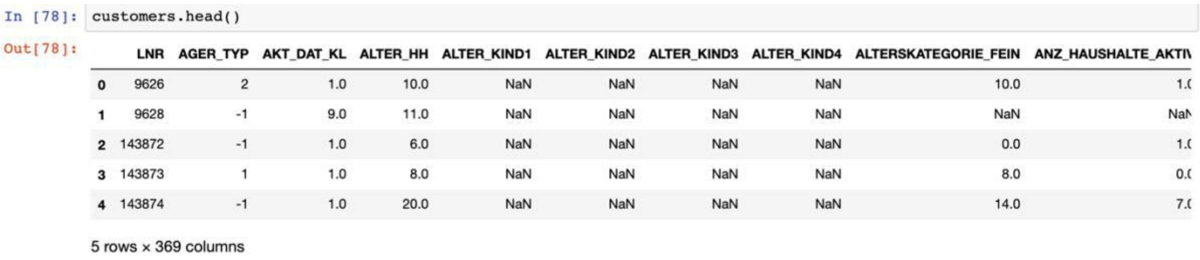
To deal with multi and mixed values columns, I will create specific functions for each after the data has been cleaned and statistical analysis applied to it. There are general models and algorithms to deal with these data problems, but our data is to specific and misrepresentation and miss interpretation of the data is plausible. To avoid this, I will manually create the functions as to maintain the data as original as possible.

There are some columns that are not in feat info, but most of them (a) have more than 20% missing data and (b), as the data provided by Arvato is real, any

Attribute that should or could appear later in customer segmentation and comparison of Gen. population with customers, can be found online.

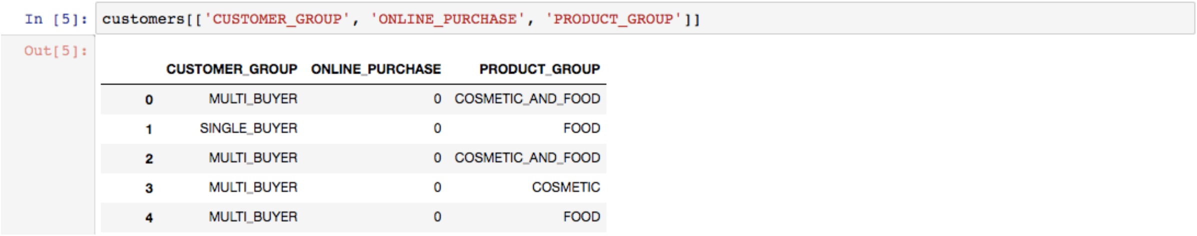
2.1.3 Udacity_CUSTOMERS_052018.csv

This is demographics data for customers of a mail-order company which has 191652 persons (rows) x 369 features (columns). Below is a sample of the dataset:



```
In [78]: customers.head()

Out[78]:
       LNR  AGER_TYP  AKT_DAT_KL  ALTER_HH  ALTER_KIND1  ALTER_KIND2  ALTER_KIND3  ALTER_KIND4  ALTERSKATEGORIE_FEIN  ANZ_HAUSHALTE_AKTI
0     9626         2         1.0      10.0          NaN          NaN          NaN          NaN                  10.0                  1.
1     9628        -1         9.0      11.0          NaN          NaN          NaN          NaN                   NaN                  Na
2   143872        -1         1.0       6.0          NaN          NaN          NaN          NaN                   0.0                  1.
3   143873         1         1.0       8.0          NaN          NaN          NaN          NaN                   8.0                  0.
4   143874        -1         1.0      20.0          NaN          NaN          NaN          NaN                  14.0                  7.

5 rows × 369 columns
```

There are 3 columns in customers that are not in AZDIAS

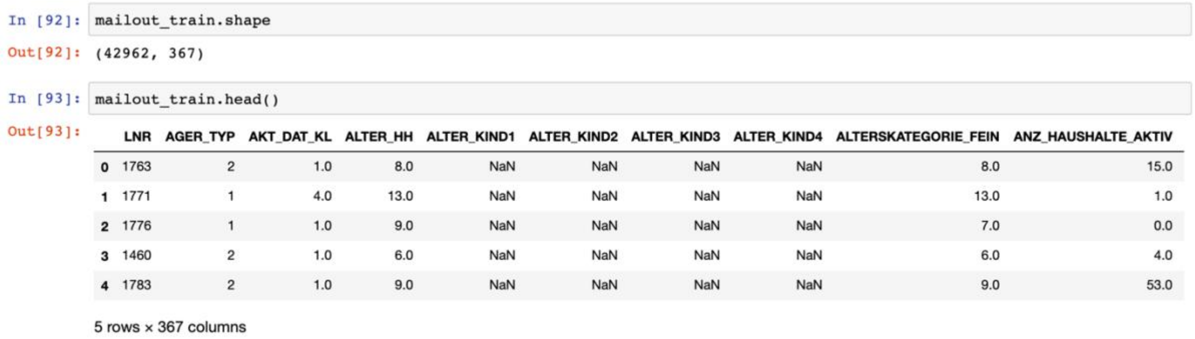


```
In [5]: customers[['CUSTOMER_GROUP', 'ONLINE_PURCHASE', 'PRODUCT_GROUP']]

Out[5]:
   CUSTOMER_GROUP  ONLINE_PURCHASE      PRODUCT_GROUP
0     MULTI_BUYER                0  COSMETIC_AND_FOOD
1    SINGLE_BUYER                0               FOOD
2     MULTI_BUYER                0  COSMETIC_AND_FOOD
3     MULTI_BUYER                0           COSMETIC
4     MULTI_BUYER                0               FOOD
```

I will be dropping these columns later as they do not contain any demographic data.. I will also be applying all the steps in AZDIAS to customers.

2.1.4 Udacity_MAILOUT_052018_TRAIN.csv

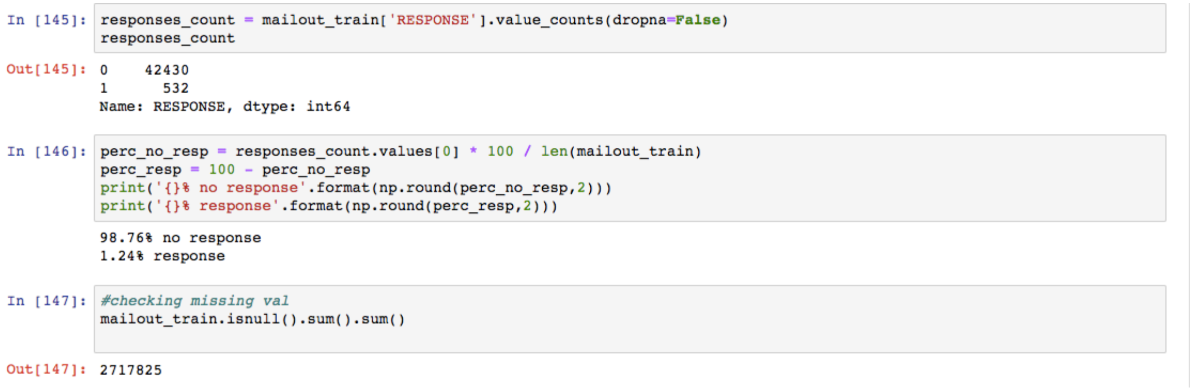
This is demographics data for individuals who were targets of a marketing campaign. It has 42962 persons (rows) x 367 (columns). It will be used to train the supervised learning model in the second half of the project.



```
In [92]: mailout_train.shape
Out[92]: (42962, 367)

In [93]: mailout_train.head()
Out[93]:
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV
0	1763	2	1.0	8.0	NaN	NaN	NaN	NaN	8.0	15.0
1	1771	1	4.0	13.0	NaN	NaN	NaN	NaN	13.0	1.0
2	1776	1	1.0	9.0	NaN	NaN	NaN	NaN	7.0	0.0
3	1460	2	1.0	6.0	NaN	NaN	NaN	NaN	6.0	4.0
4	1783	2	1.0	9.0	NaN	NaN	NaN	NaN	9.0	53.0

5 rows × 367 columns

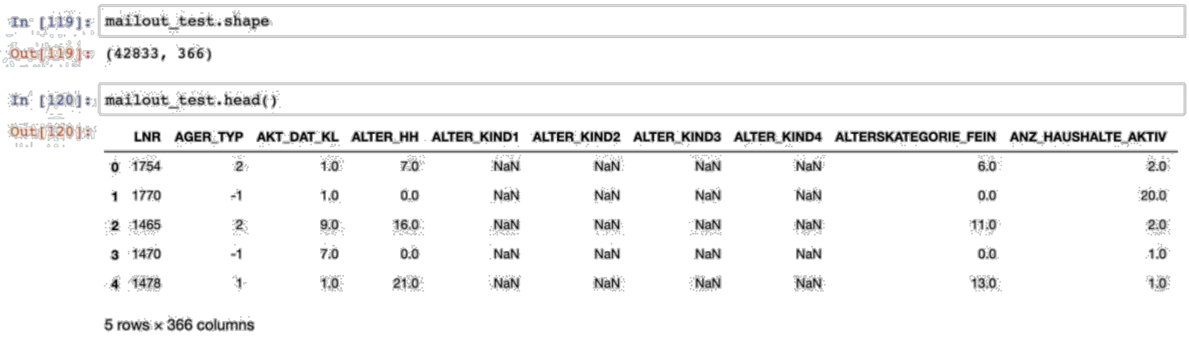


```
In [145]: responses_count = mailout_train['RESPONSE'].value_counts(dropna=False)
          responses_count

Out[145]: 0    42430
          1      532
          Name: RESPONSE, dtype: int64

In [146]: perc_no_resp = responses_count.values[0] * 100 / len(mailout_train)
          perc_resp = 100 - perc_no_resp
          print('{}% no response'.format(np.round(perc_no_resp,2)))
          print('{}% response'.format(np.round(perc_resp,2)))

          98.76% no response
          1.24% response

In [147]: #checking missing val
          mailout_train.isnull().sum().sum()

Out[147]: 2717825
```

We can see that there are only 532 responses out of 42430. Meaning less then 2% success on the campaign. There are also 2717825 missing values in the data. I will be applying preprocessing steps to this dataset as I did in AZDIAS and CUSTOMERS dataset.

2.1.5 Udacity_MAILOUT_052018_TEST.csv

This is demographics data for individuals who were targets of a marketing campaign; 42833 persons (rows) x 366 (columns). It will be used to test the supervised learning model.



```
In [119]: mailout_test.shape
Out[119]: (42833, 366)

In [120]: mailout_test.head()
Out[120]:
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV
0	1754	2	1.0	7.0	NaN	NaN	NaN	NaN	6.0	2.0
1	1770	-1	1.0	0.0	NaN	NaN	NaN	NaN	0.0	20.0
2	1465	2	9.0	16.0	NaN	NaN	NaN	NaN	11.0	2.0
3	1470	-1	7.0	0.0	NaN	NaN	NaN	NaN	0.0	1.0
4	1478	1	1.0	21.0	NaN	NaN	NaN	NaN	13.0	1.0

5 rows × 366 columns

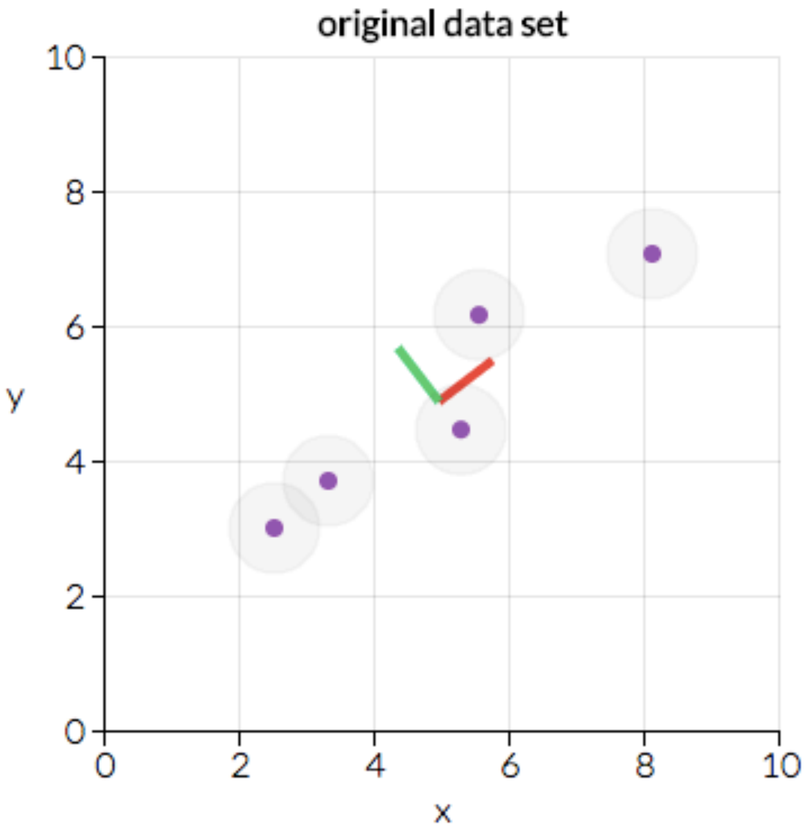
Same preprocessing steps will be applied to this dataset which will be applied to Udacity_MAILOUT_052018_TRAIN.csv

2.1.5 CONCLUSION OF ANALYSIS

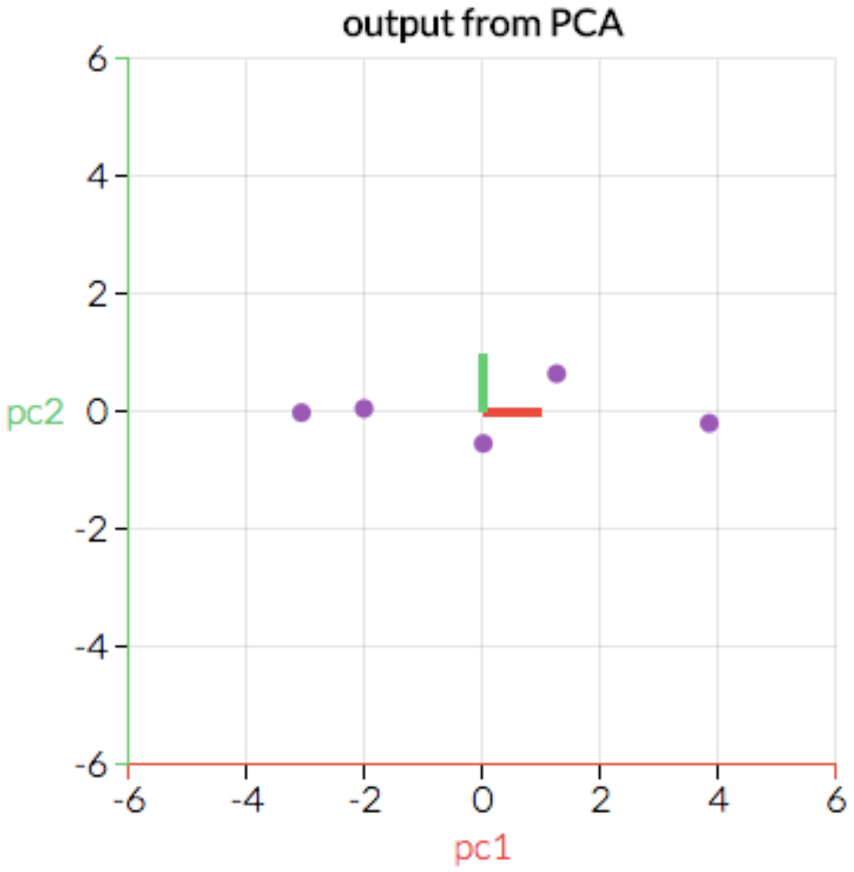
As the data has a lot of incomplete and missing values, it would probably not be normalized with 100% confidence, not Gaussian. I will have to clean the data, apply the steps described to each set. Generalizing, would be cleaning, imputing and scaling the data. That would provide a dataset with a normal distribution and statistically viable to pass through advanced models without having bias in the result. One great advantage of this data is its size. I will be able to drop values, not a lot, but I believe enough, to clean the data without causing Omitted Variable Bias. Meaning my finsished data will in all probability be BLUE – Best Linear Unbiased Estimator.
There is a lot of work to be done on the data, which is especially good, meaning the data is raw and real. If succeding in this, probably most important step, the results from the estimation from the models will be atleast satisfactory.

2.2 TECHNIQUES

As we are working with high dimensionality data I will be first using the unsupervised learning technique PCA (Principal component analysis) to reduce the Datasets dimension but maintaining its original variance, as to maintain the "originality" of the data and reduce the risk of over fitting.
PCA in its simplest form is the identification of a linear subspace of lower dimensionality where the largest variance in the original dataset is maintained. This is accomplished by transforming the feature values in a transposed covariance matrix of the features, then calculating the eigenvalues and eigenvectors, and finally selecting the features with highest values.
Visually in a 2 feature dataset this would look like:



Our original data in the xy-plane. (Source.)



I will then use another unsupervised learning technique to describe the relationship between the demographics of the company's existing customers and the general population of Germany called Clustering. I will be using KMeans to accomplish this. Clustering is the process of grouping a data into different clusters. K-means accomplishes this task by minimizing the Euclidean distance between the objects and the centroids and obtaining as a result a certain number of clusters.. The Euclidean distance between points p and q is the length of the line segment connecting them. In Cartesian coordinates, if p = (p1, p2,..., pn) and q = (q1, q2,..., qn) are two points in Euclidean n-space, then the distance (d) from p to q, or
from q to p is given by the Pythagorean formula [4]:

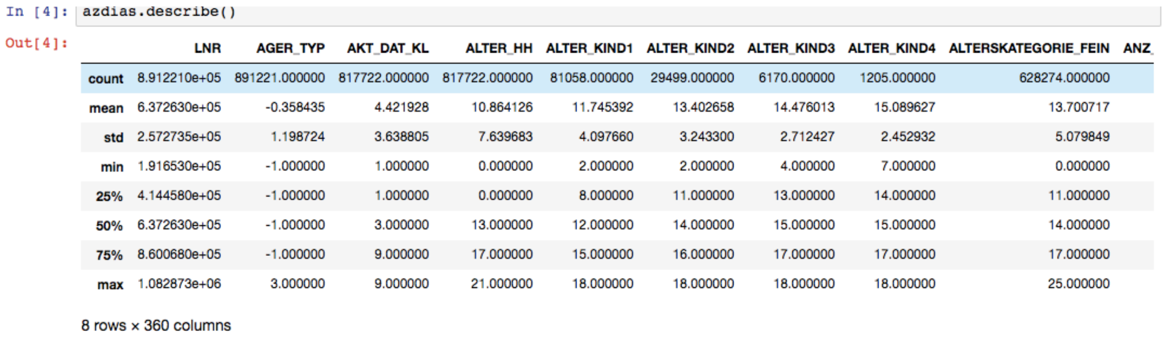
$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

In the preprocessing step I will be applying developing a function and transforming the DIAS Attributes file into a dataframe. In this dataframe there will be an added column missing_vals which will contain the values for each feature in a accessible dtype for pandas. This dataframe will be then used by the function to transform all of these values in their respectable features in the AZDIAS Dataset to NaN.

After all cleaning steps have been concluded I will create a sample version of AZDIAS with about 20% of the data. Statistically samples from large data taken at random, generate the same result as the data itself.

As a next step I will be imputing and scaling remaining missing values as a preparation for the next step of the project. The data should be imputed so the clustering techniques are able to process the data and scaled as the standard deviation requires so due to great difference between them.



```
In [4]: azdias.describe()
```

Out[4]:

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_
count	8.912210e+05	891221.000000	817722.000000	817722.000000	81058.000000	29499.000000	6170.000000	1205.000000	628274.000000	
mean	6.372630e+05	-0.358435	4.421928	10.864126	11.745392	13.402658	14.476013	15.089627	13.700717	
std	2.572735e+05	1.198724	3.638805	7.639683	4.097660	3.243300	2.712427	2.452932	5.079849	
min	1.916530e+05	-1.000000	1.000000	0.000000	2.000000	2.000000	4.000000	7.000000	0.000000	
25%	4.144580e+05	-1.000000	1.000000	0.000000	8.000000	11.000000	13.000000	14.000000	11.000000	
50%	6.372630e+05	-1.000000	3.000000	13.000000	12.000000	14.000000	15.000000	15.000000	14.000000	
75%	8.600680e+05	-1.000000	9.000000	17.000000	15.000000	16.000000	17.000000	17.000000	17.000000	
max	1.082873e+06	3.000000	9.000000	21.000000	18.000000	18.000000	18.000000	18.000000	25.000000	

8 rows × 360 columns

I will then be applying PCA to the data, a method previously described.

Then I will be applying k-means clustering on the PCA transformed data as to be able to have a idea of which part of the population would be a potential customer to the Mail company.

In the second half of the project I will be building a prediction model to decide whether or not it will be worth to invest in X person in the campaign.

The train and test dataset has been provided so that this step is possible. The train dataset is labeled so I will need to use supervised learning models for the prediction.

Logistic Regression: It is one of the most popular algorithms for binary classification problems. The input values are combined linearly using weights or co-efficient values to predict an output value. This output value is a probability which can be converted to binary value (0 or 1) based on a step function.

A function that can be used to express feature values and log odds is the Logit function,

$$\mathrm{logit}(P(y=1|x)) = W_0X_0 + \ldots + W_mW_m = \mathrm{Sum}(W_iX_i) = W^TX$$

P(y=1|x) is the conditional probability that a particular sample belongs to class 1 given its features x.

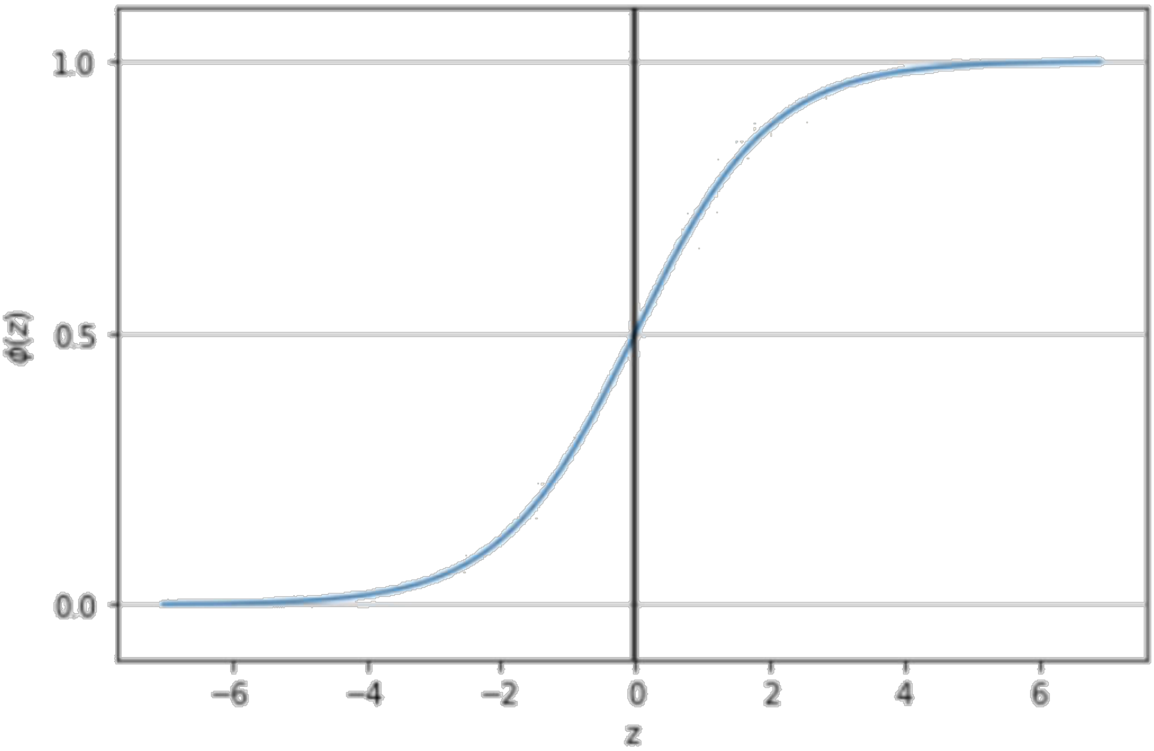
To be able to predict the probability that a sample belongs to a certain class, the algorithm utilizes a sigmoid function. The formula for sigmoid function is as follows:

$$\phi(z) = \frac{1}{1+e^{-z}}$$

where *z* can be calculated as follows:

z = wTx = w0 + w1x1 + …… + wmxm

The graphical representation of this is as follows:



The sigmoid function takes the values and converts them to 0 or depending on z and if it is tending to infinity or minus infinity as we can see above.

Now that we have this predicted value, it can be transformed into a a binary value using a step function such as:

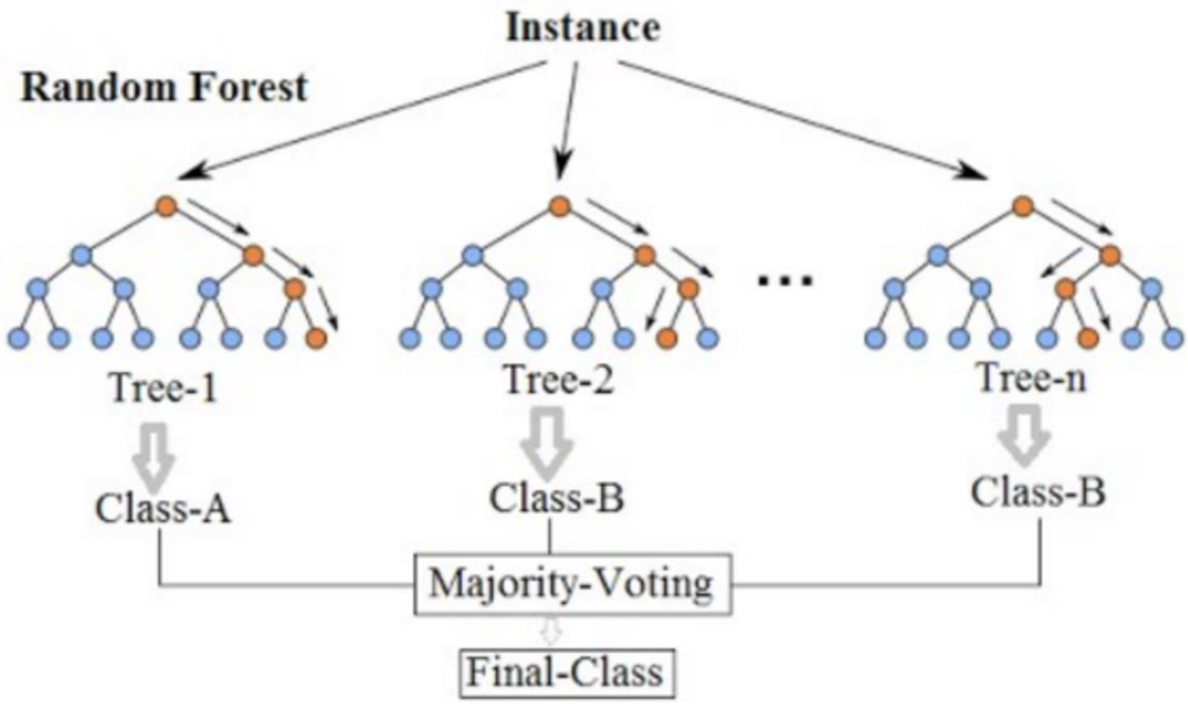
$$\hat{y} = \begin{cases} 1 & if\,\phi(z) \geq 0.5 \\ 0 & otherwise \end{cases}$$

Random Forest: Random Forest is a supervised learning algorithm use din regression and classification problems. It is a decision tree algorithm. The way random forest works is by selecting random columns, batching them and making decision trees for each group. Normally in datasets with large number of features, using decision tree algorithms leads to overfitting. Random Forest does not produce this due to the batching or grouping it does to the features.

This is how Random Forest works:

1. Select random samples from a given dataset.

2. Construct a decision tree for each sample and get a prediction from each decision tree(weak learners)
3. Perform a vote for each predicted result.
4. Select the prediction with the most votes as the final result.



Gradient Boosting: Gradient Boosting combines a number of weak learners to build a strong learner which is then used to make predictions. It depends on a loss function which should be differentiable.

It is divided in two components: weak learner and an additive component. Decision Trees are used as weak learners. The trees are then added to the model over time without manipulating or interfering with existing trees being this the Additive component.

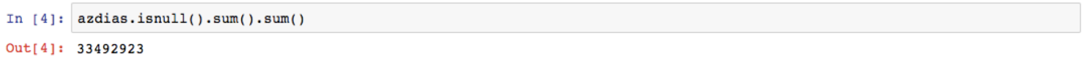
To minimize errors between the parameters, gradient descent is applied to the calculated loss reducing significantly that loss. Then the parameters of the trees are modified to reduce the residual loss.

I will use GridSearch to establish the best model and then perform hyper-parameter tuning to optimize the model for best performance.

3. METHODOLOGY

3.1 DATA PREP

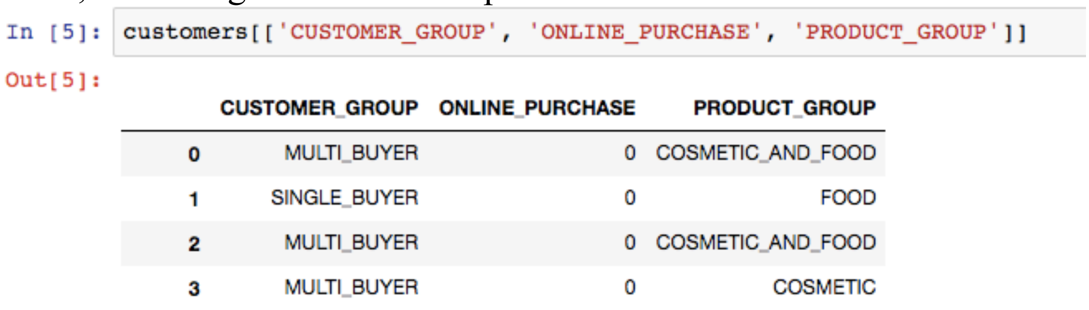
The first part of data to process is the Udacity_AZDIAS file, which contains Demographics data for the general population. At first glance we can see that there are 33492923 missing values in AZDIAS.



```
In [4]: azdias.isnull().sum().sum()
Out[4]: 33492923
```

It will be necessary to clean the data and then use impute and Scaler to complete missing data after cleaned

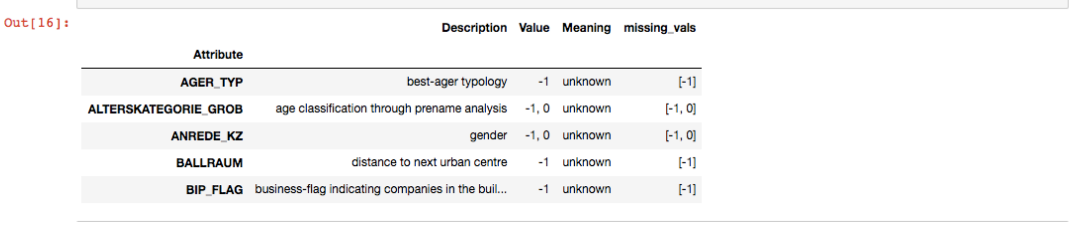
Analyzing the 3 extra columns in CUSTOMERS, there are no demographic data, so I will go ahead and drop them.



```
In [5]: customers[['CUSTOMER_GROUP', 'ONLINE_PURCHASE', 'PRODUCT_GROUP']]
Out[5]:
```

	CUSTOMER_GROUP	ONLINE_PURCHASE	PRODUCT_GROUP
0	MULTI_BUYER	0	COSMETIC_AND_FOOD
1	SINGLE_BUYER	0	FOOD
2	MULTI_BUYER	0	COSMETIC_AND_FOOD
3	MULTI_BUYER	0	COSMETIC

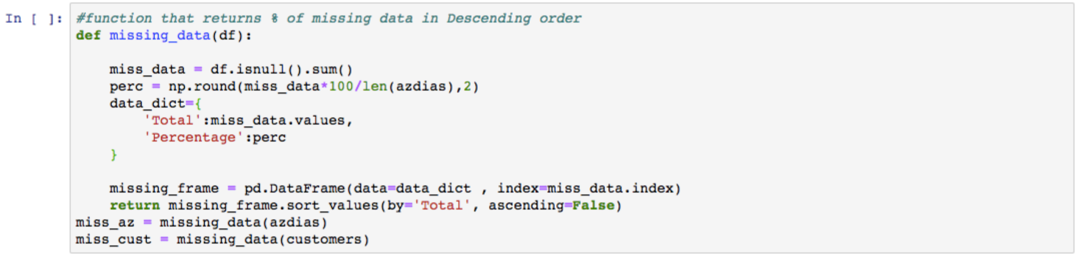
As previously discussed I will now create a DF with missing and unknown values from the excel file DIAS Attributes – Values 2017.xlsx



```
Out[16]:
```

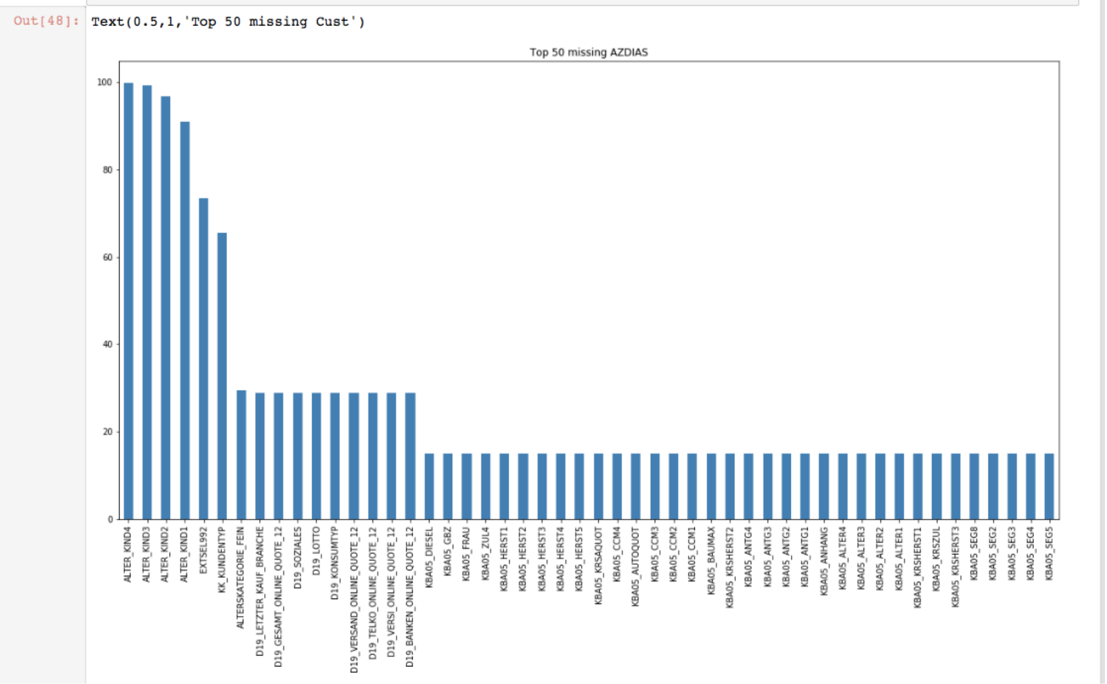
	Description	Value	Meaning	missing_vals
Attribute				
AGER_TYP	best-ager typology	-1	unknown	[-1]
ALTERSKATEGORIE_GROB	age classification through prename analysis	-1, 0	unknown	[-1, 0]
ANREDE_KZ	gender	-1, 0	unknown	[-1, 0]
BALLRAUM	distance to next urban centre	-1	unknown	[-1]
BIP_FLAG	business-flag indicating companies in the buil...	-1	unknown	[-1]

I will now evaluate by Column the missing data and sort it in descending order. This will help in visualizing which data to drop, making it clean but losing as little as possible as to maintain the highest possible proximity to the true data.



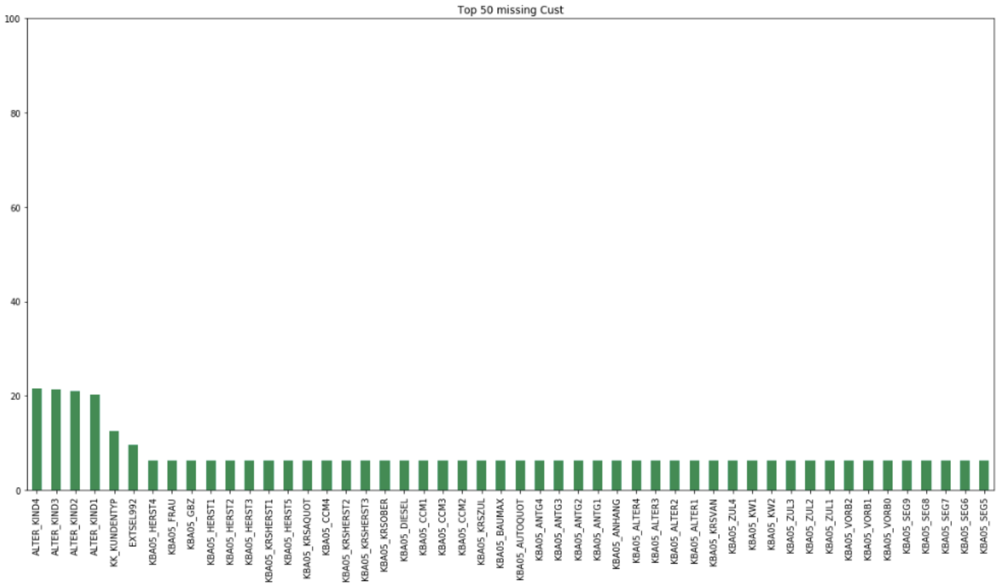
```
In [ ]: #function that returns % of missing data in Descending order
        def missing_data(df):

            miss_data = df.isnull().sum()
            perc = np.round(miss_data*100/len(azdias),2)
            data_dict={
                'Total':miss_data.values,
                'Percentage':perc
            }

            missing_frame = pd.DataFrame(data=data_dict , index=miss_data.index)
            return missing_frame.sort_values(by='Total', ascending=False)
        miss_az = missing_data(azdias)
        miss_cust = missing_data(customers)
```

Upon first inspection we see that there are 16 columns in AZDIAS that have more than 20% of their values missing.



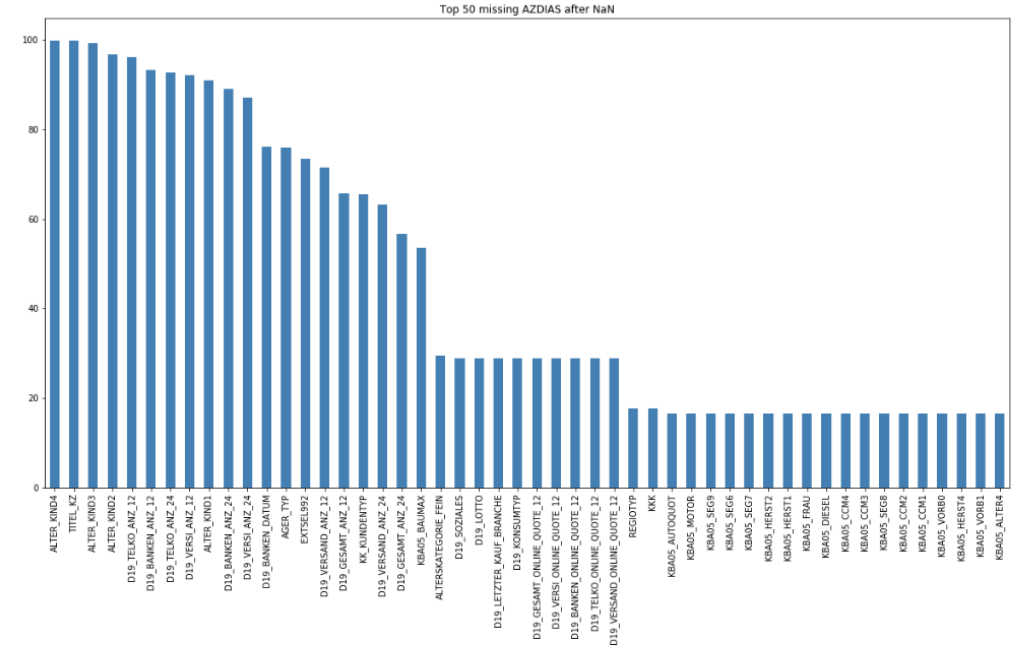
```
Out[48]: Text(0.5,1,'Top 50 missing Cust')
```

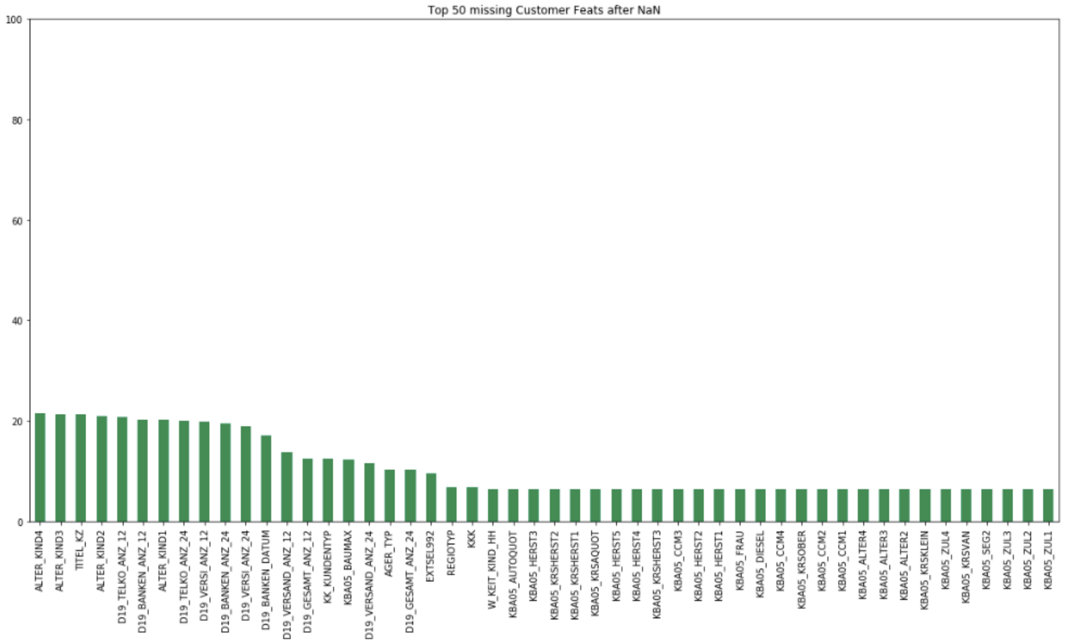
and in the CUSTOMERS we have only 4



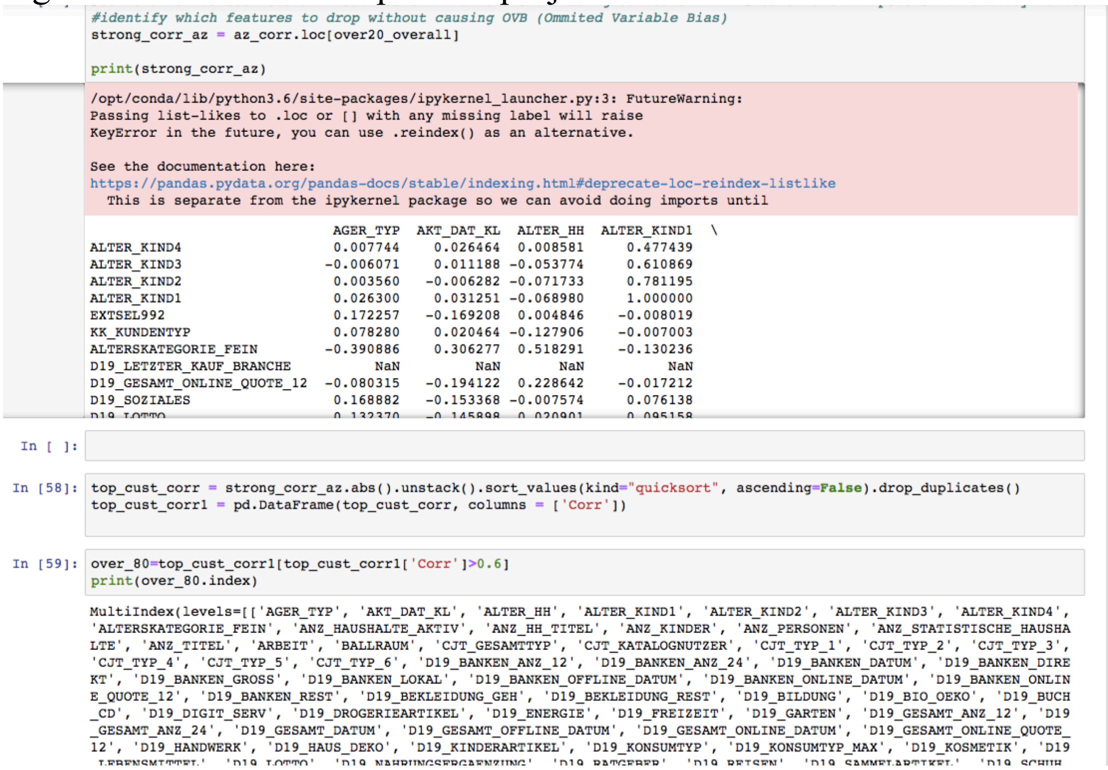
I created a function called missing_to_nan. The fucntions takes in 2 Dataframes, one with values to be changed and the other containing the code of the values that are supposed to be NaN.

After applying the missing_to_nan function, which takes values that are categorized as missing from the DataFrame previously created and transforms them to NaN, we can see a statistically significant increase in missing data in both AZDIAS and CUSTOMERS.





After data has been processed and missing values in both Data’s are transformed to NaN, I can now evaluate any correlation between the Datasets to establish which ones should be dropped. Leaving strongly correlated Data in the set can cause Omitted Variable Bias in case of regressions in the last step of the project.



```
#identify which features to drop without causing OVB (Ommited Variable Bias)
strong_corr_az = az_corr.loc[over20_overall]

print(strong_corr_az)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:3: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
                              AGER_TYP  AKT_DAT_KL  ALTER_HH  ALTER_KIND1  \
ALTER_KIND4                   0.007744    0.026464  0.008581     0.477439
ALTER_KIND3                  -0.006071    0.011188 -0.053774     0.610869
ALTER_KIND2                   0.003560   -0.006282 -0.071733     0.781195
ALTER_KIND1                   0.026300    0.031251 -0.068980     1.000000
EXTSEL992                     0.172257   -0.169208  0.004846    -0.008019
KK_KUNDENTYP                  0.078280    0.020464 -0.127906    -0.007003
ALTERSKATEGORIE_FEIN         -0.390886    0.306277  0.518291    -0.130236
D19_LETZTER_KAUF_BRANCHE           NaN         NaN       NaN          NaN
D19_GESAMT_ONLINE_QUOTE_12   -0.080315   -0.194122  0.228642    -0.017212
D19_SOZIALES                  0.168882   -0.153368 -0.007574     0.076138
D19_LOTTO                     0.132370   -0.145898  0.020901     0.095158
```

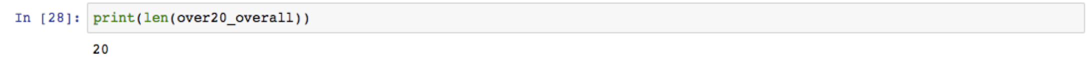
```
In [ ]:
```

```
In [58]: top_cust_corr = strong_corr_az.abs().unstack().sort_values(kind="quicksort", ascending=False).drop_duplicates()
         top_cust_corr1 = pd.DataFrame(top_cust_corr, columns = ['Corr'])
```

```
In [59]: over_80=top_cust_corr1[top_cust_corr1['Corr']>0.6]
         print(over_80.index)
```

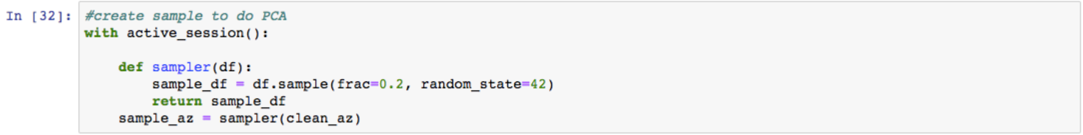
```
MultiIndex(levels=[['AGER_TYP', 'AKT_DAT_KL', 'ALTER_HH', 'ALTER_KIND1', 'ALTER_KIND2', 'ALTER_KIND3', 'ALTER_KIND4',
'ALTERSKATEGORIE_FEIN', 'ANZ_HAUSHALTE_AKTIV', 'ANZ_HH_TITEL', 'ANZ_KINDER', 'ANZ_PERSONEN', 'ANZ_STATISTISCHE_HAUSHA
LTE', 'ANZ_TITEL', 'ARBEIT', 'BALLRAUM', 'CJT_GESAMTTYP', 'CJT_KATALOGNUTZER', 'CJT_TYP_1', 'CJT_TYP_2', 'CJT_TYP_3',
'CJT_TYP_4', 'CJT_TYP_5', 'CJT_TYP_6', 'D19_BANKEN_ANZ_12', 'D19_BANKEN_ANZ_24', 'D19_BANKEN_DATUM', 'D19_BANKEN_DIRE
KT', 'D19_BANKEN_GROSS', 'D19_BANKEN_LOKAL', 'D19_BANKEN_OFFLINE_DATUM', 'D19_BANKEN_ONLINE_DATUM', 'D19_BANKEN_ONLIN
E_QUOTE_12', 'D19_BANKEN_REST', 'D19_BEKLEIDUNG_GEH', 'D19_BEKLEIDUNG_REST', 'D19_BILDUNG', 'D19_BIO_OEKO', 'D19_BUCH
_CD', 'D19_DIGIT_SERV', 'D19_DROGERIEARTIKEL', 'D19_ENERGIE', 'D19_FREIZEIT', 'D19_GARTEN', 'D19_GESAMT_ANZ_12', 'D19
_GESAMT_ANZ_24', 'D19_GESAMT_DATUM', 'D19_GESAMT_OFFLINE_DATUM', 'D19_GESAMT_ONLINE_DATUM', 'D19_GESAMT_ONLINE_QUOTE_
12', 'D19_HANDWERK', 'D19_HAUS_DEKO', 'D19_KINDERARTIKEL', 'D19_KONSUMTYP', 'D19_KONSUMTYP_MAX', 'D19_KOSMETIK', 'D19
_LEBENSMITTEL', 'D19_LOTTO', 'D19_NAHRUNGSERGAENZUNG', 'D19_RATGEBER', 'D19_REISEN', 'D19_SAMMELARTIKEL', 'D19_SCHUH
```

Correlation analysis indicates that any strong correlation of the features are mainly between the Strongly missing values Data Sets themselves, meaning I will choose to drop the features with over 20% missing values on both Data sets for optimal results. This totals to 20 features being dropped.



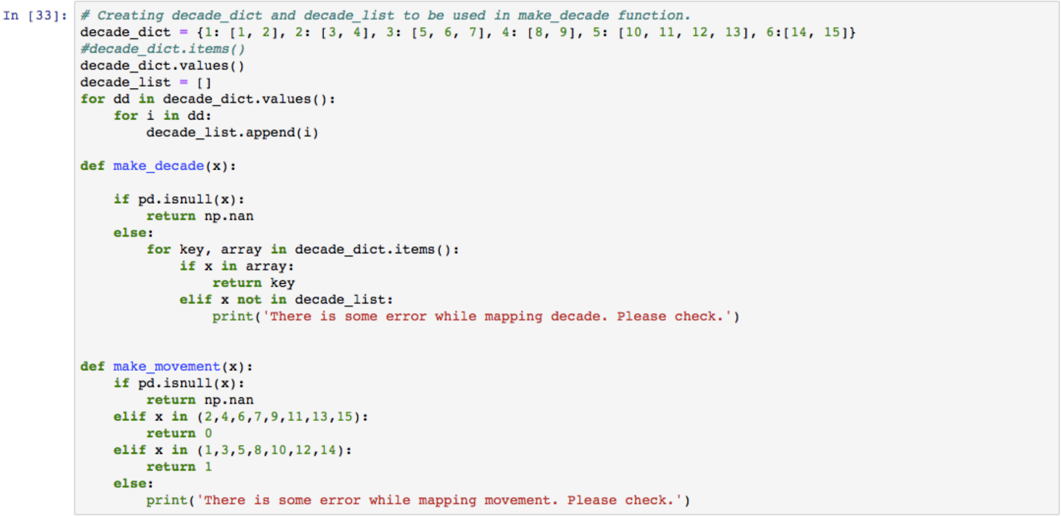
```
In [28]: print(len(over20_overall))

         20
```

I will create a random sample of the AZDIAS of 20% of the data as to be able to process PCA effectively and reduce processing time, while maintaining as much as possible the Originality of the data. As random samples are statistically comparable to original data, there should not be any divergence of results as compared to processing 100% of the data.



```
In [32]: #create sample to do PCA
         with active_session():

             def sampler(df):
                 sample_df = df.sample(frac=0.2, random_state=42)
                 return sample_df
             sample_az = sampler(clean_az)
```

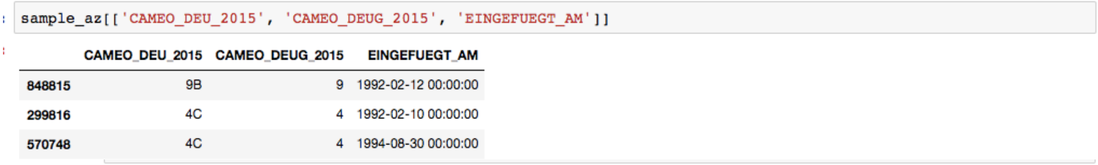
To deal with the mixed typed columns, I will be applying a function that creates two new columns of information out of the mixed type column. make_decade and make_movement will be applied to the PRAEGENDE_JUGENDJAHRE column



```
In [33]: # Creating decade_dict and decade_list to be used in make_decade function.
         decade_dict = {1: [1, 2], 2: [3, 4], 3: [5, 6, 7], 4: [8, 9], 5: [10, 11, 12, 13], 6:[14, 15]}
         #decade_dict.items()
         decade_dict.values()
         decade_list = []
         for dd in decade_dict.values():
             for i in dd:
                 decade_list.append(i)

         def make_decade(x):

             if pd.isnull(x):
                 return np.nan
             else:
                 for key, array in decade_dict.items():
                     if x in array:
                         return key
                     elif x not in decade_list:
                         print('There is some error while mapping decade. Please check.')


         def make_movement(x):
             if pd.isnull(x):
                 return np.nan
             elif x in (2,4,6,7,9,11,13,15):
                 return 0
             elif x in (1,3,5,8,10,12,14):
                 return 1
             else:
                 print('There is some error while mapping movement. Please check.')
```

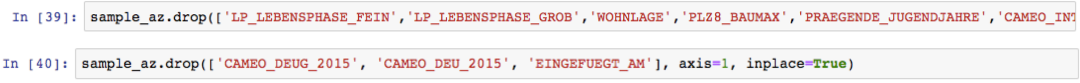
and make_wealth and make_life_stage will be applied to the CAMEO_INTL_2015 column.
Both original columns will be dropped, as their info is in the 4 newly created columns.

After transforming features, I will be dropping object features which are not relevant to the data set. Such as CAMEO_DEU_2015 and EINGENFUEGT_AM that are not described and are just dates, in the second case.



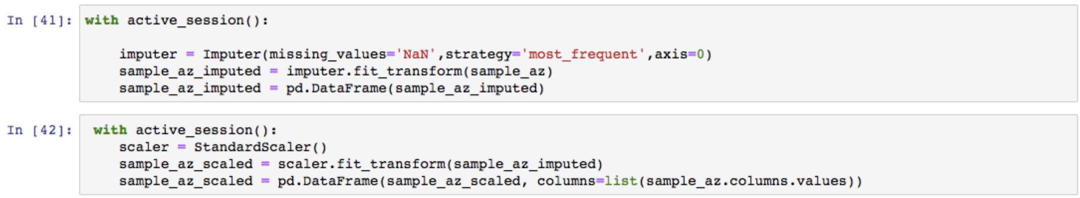
```
sample_az[['CAMEO_DEU_2015', 'CAMEO_DEUG_2015', 'EINGEFUEGT_AM']]
```

	CAMEO_DEU_2015	CAMEO_DEUG_2015	EINGEFUEGT_AM
848815	9B	9	1992-02-12 00:00:00
299816	4C	4	1992-02-10 00:00:00
570748	4C	4	1994-08-30 00:00:00



```
In [39]: sample_az.drop(['LP_LEBENSPHASE_FEIN','LP_LEBENSPHASE_GROB','WOHNLAGE','PLZ8_BAUMAX','PRAEGENDE_JUGENDJAHRE','CAMEO_IN

In [40]: sample_az.drop(['CAMEO_DEUG_2015', 'CAMEO_DEU_2015', 'EINGEFUEGT_AM'], axis=1, inplace=True)
```

Now that the data has been pre-processed and cleaned, I will Impute and Scale the data as to prepare it for PCA application on it.



```
In [41]: with active_session():

             imputer = Imputer(missing_values='NaN',strategy='most_frequent',axis=0)
             sample_az_imputed = imputer.fit_transform(sample_az)
             sample_az_imputed = pd.DataFrame(sample_az_imputed)
```

```
In [42]:  with active_session():
             scaler = StandardScaler()
             sample_az_scaled = scaler.fit_transform(sample_az_imputed)
             sample_az_scaled = pd.DataFrame(sample_az_scaled, columns=list(sample_az.columns.values))
```

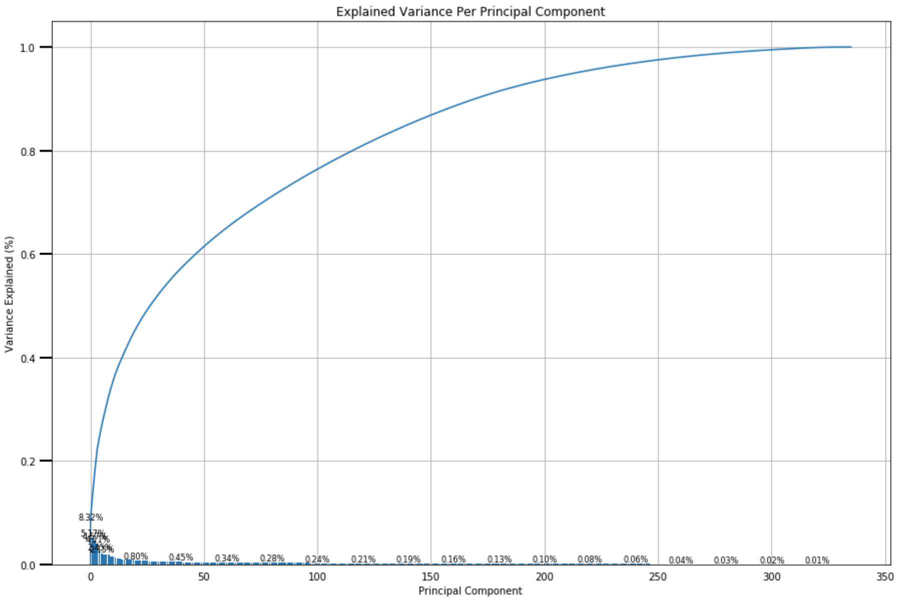
3.1.2 – BENCHMARK

Benchmark for this project is the Kaggle competition. After the model has been trained and tuned, I will need to submit the results in a .csv file which will be analyzed, tested and graded. Giving the overall “real” score of the model produced in this project. The best scoring model on Kaggle was 0.80936.

3.2. IMPLEMENTATION

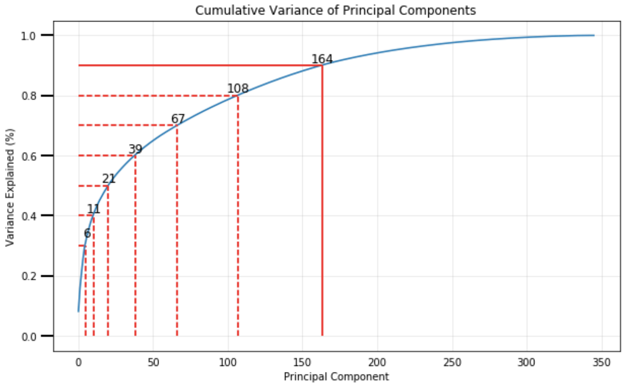
3.2.1 – UNSUPERVISED LEARNING

As a primary step, I will apply PCA to the sampled AZDIAS to have data on variance so I can be able to choose the number of dimension the data should be reduced to. To better visualize the explained variance I will use two functions given by Udacity. The scree plot and the Cumulative Variance explained by components. These two functions will plot the information returned by the first application of PCA to the data, aiding me in identifying the optimum number of dimensions.



From this graph we can see that the first 200 components explain about 90% of the variance of the dataset. To get a more accurate result, I will apply the Cumulative Variance.

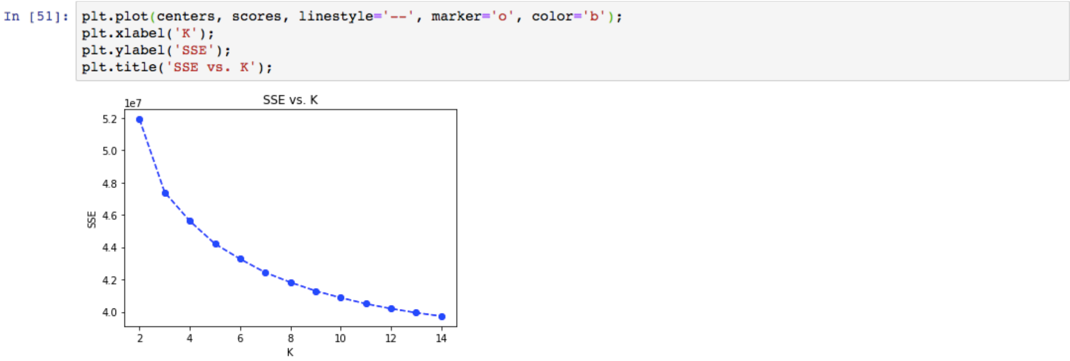
```
In [51]: plot_cum_var(pca)
```



Here we can see the the exact number of components that explain for 90% of the variance is 164.

I will apply PCA to the dataset preserving 164 components.

I will not analyze PCA deeply as I will apply Clustering to the data using KMeans, which is the data that will be fed to the model. This analysis will be available in results.



```
In [51]: plt.plot(centers, scores, linestyle='--', marker='o', color='b');
         plt.xlabel('K');
         plt.ylabel('SSE');
         plt.title('SSE vs. K');
```

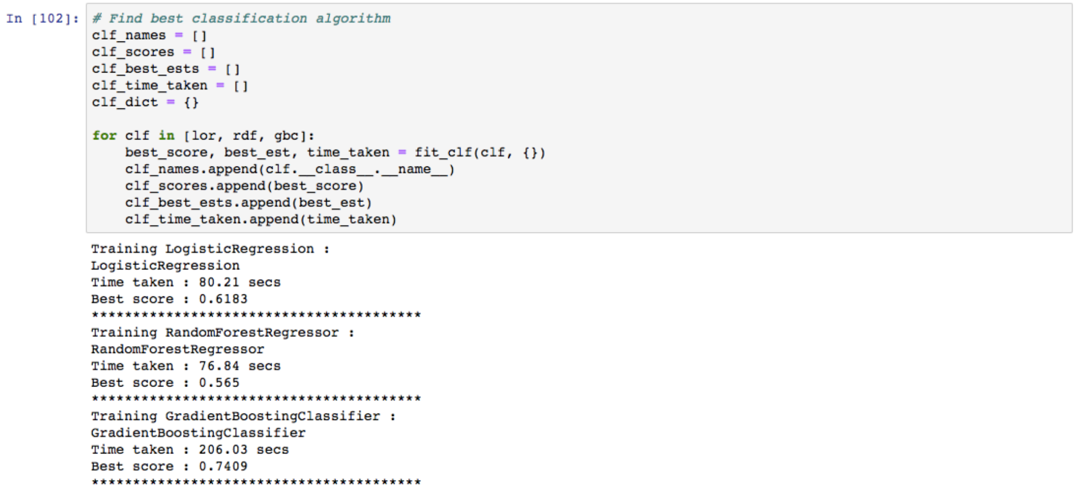
Using the Elbow method, there is not a 100% accurate visual of the point where the distortion starts decreasing, I will choose 6 clusters, as it is the best approximated guess. I have tried to install and use Yellowbrick module that returns the exact number of cluster to be used, but unfortunately, I was unable to install it in the jupyter notebook provided.

I will now apply all the steps used in AZDIAS to CUSTOMERS, preparing the data.
Comparing results form both Datasets we have the following table,

After the steps are applied, we will be able to produce the results from this part of the project, and have a overview of the data our model will be dealing with. This could help my interpretation and eventually see what could have gone wrong if the model does not produce minimum satisfactory results.

3.2.2 SUPERVISED LEARNING

I will now pass the data through 4 different Models: Logistic Regression, Random Forest Regressor, Gradient Boosting Regressor and Gradient Boosting Classifier. Using GridSearch to aid me in selecting the best models, I will afterwards tune the best model for optimum results.



```
In [102]: # Find best classification algorithm
          clf_names = []
          clf_scores = []
          clf_best_ests = []
          clf_time_taken = []
          clf_dict = {}

          for clf in [lor, rdf, gbc]:
              best_score, best_est, time_taken = fit_clf(clf, {})
              clf_names.append(clf.__class__.__name__)
              clf_scores.append(best_score)
              clf_best_ests.append(best_est)
              clf_time_taken.append(time_taken)
```

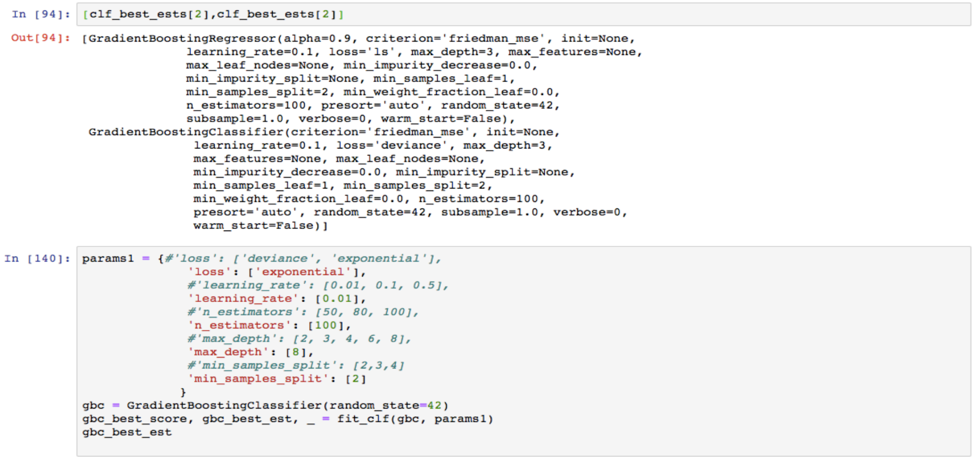
```
Training LogisticRegression :
LogisticRegression
Time taken : 80.21 secs
Best score : 0.6183
****************************************
Training RandomForestRegressor :
RandomForestRegressor
Time taken : 76.84 secs
Best score : 0.565
****************************************
Training GradientBoostingClassifier :
GradientBoostingClassifier
Time taken : 206.03 secs
Best score : 0.7409
****************************************
```

Gradient Boosting Regressor and Gradient Boosting Classifier came out with almost the exact result, so I chose to drop GBR and keep GBC. The score for GBC was 0.7482

3.3 - REFINEMENT

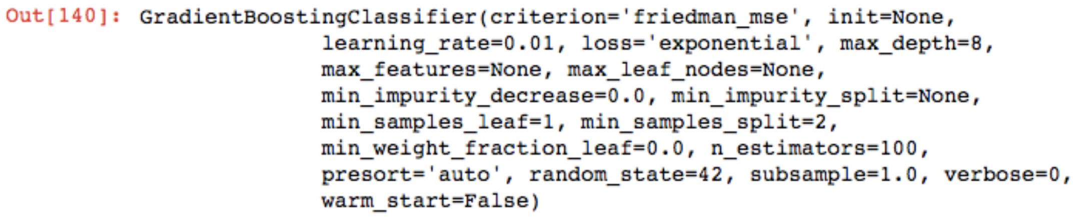
I will now tune the Hyper-parameters so the model can produce the best result possible. Tuning will be necessary as I have not chosen any configuration through GridSearch which initially uses default parameters. I will also reprocess the data as to make sure all theoretical steps of data prep were done correctly.

To tune the Hyper-parameters of the model I will use GridSearch with a selection of possibilities for it to test, then I will use those Hyper-parameters to produce the best model.



```
In [94]: [clf_best_ests[2],clf_best_ests[2]]

Out[94]: [GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                       learning_rate=0.1, loss='ls', max_depth=3, max_features=None,
                       max_leaf_nodes=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, presort='auto', random_state=42,
                       subsample=1.0, verbose=0, warm_start=False),
          GradientBoostingClassifier(criterion='friedman_mse', init=None,
                        learning_rate=0.1, loss='deviance', max_depth=3,
                        max_features=None, max_leaf_nodes=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,
                        presort='auto', random_state=42, subsample=1.0, verbose=0,
                        warm_start=False)]

In [140]: params1 = {#'loss': ['deviance', 'exponential'],
                      'loss': ['exponential'],
                      #'learning_rate': [0.01, 0.1, 0.5],
                      'learning_rate': [0.01],
                      #'n_estimators': [50, 80, 100],
                      'n_estimators': [100],
                      #'max_depth': [2, 3, 4, 6, 8],
                      'max_depth': [8],
                      #'min_samples_split': [2,3,4]
                      'min_samples_split': [2]
                     }
          gbc = GradientBoostingClassifier(random_state=42)
          gbc_best_score, gbc_best_est, _ = fit_clf(gbc, params1)
          gbc_best_est
```

The best performing Hyper-parameters were:



```
Out[140]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
                        learning_rate=0.01, loss='exponential', max_depth=8,
                        max_features=None, max_leaf_nodes=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,
                        presort='auto', random_state=42, subsample=1.0, verbose=0,
                        warm_start=False)
```

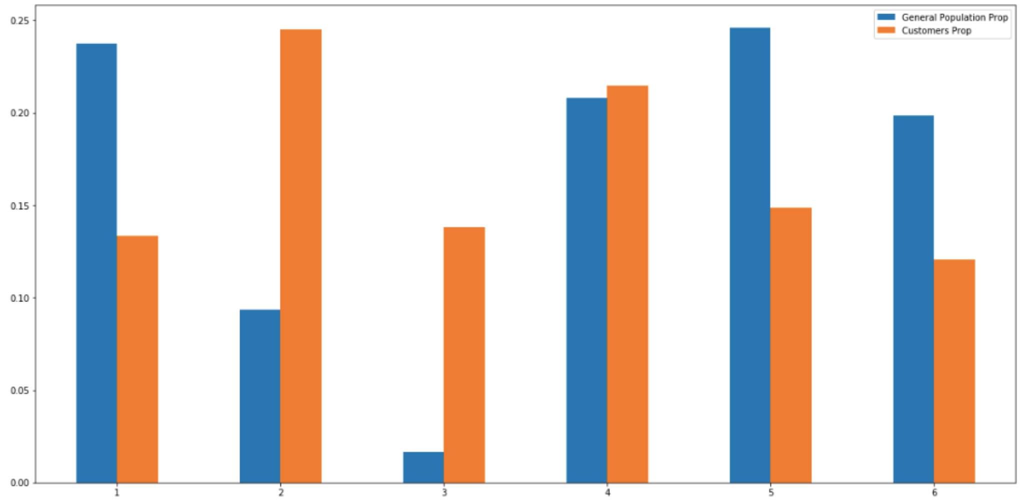
After tuning and reprocessing, I had a few different results being the best of them a score of 0.79. The last changes were not satisfactory and brought the model down to 0.758 again.

I will now utilize this model to predict the responses of MAILOUT_TEST, save it to a CSV and upload it to Kaggle as it will provide a Benchmark for the project. Before doing this, I will analyze MAILOUT_TEST and have a general overview of the data contained in it. I will also separate the data and prepare if for upload as a CSV.

4. RESULTS

4.1 –UNSUPERVISED LEARNING

Comparing results form both Datasets we have the following table,



We can see that there are 2 clusters that are Over-represented in CUSTOMERS in comparison to AZDIAS.
The characteristics of the first over-represented cluster, cluster number 2, is as follows:

High income, not financially averse, with new residences as their property and a high number of cars in the neighborhood. Senior Age members in this group, retired. This cluster is a good match for the Mail-Order company.

Cluster number 3, or the second over-represented cluster in CUSTOMERS:

Younger then previous cluster, own more properties especially vehicles. Good income, not financially averse. Age could be between 45 and 60 years. Appears to be a good match for the mail company awell.

4.2 – UNSUPERVISED RESULTS

Utilizing the GBC model to predict the responses of MAILOUT_TEST and then saving it to a CSV file so I could upload to the Kaggle competition, I could achieve a score of 0.80370, earning 19th place. Which is excellent in terms of model accuracy.

5. CONCLUSION

I tried different approaches to the data prep and what I find that returned better results was to not drop rows. Even though the rows have a lot of missing data, imputing and scaling return better results then dropping rows with more than 20% missing values.
Also, number of cluster in the Kmeans part of project, had a bigger influence in the results than I had expected.

The last changes gave worse results, but I kept my main model, the one reported and described here, as the optimum achievement.

6. REFERENCES

https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html

https://www.r-bloggers.com/how-to-prepare-and-apply-machine-learning-to-your-dataset/

https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

https://machinelearningmastery.com/quick-and-dirty-data-analysis-for-your-machine-learning-problem/

https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

https://towardsdatascience.com/understanding-logistic-regression-9b02c2aec102

https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

Artificial Intelligence: A Modern Approach, Global Edition Paperback – 2011. - by Stuart Russell;Peter Norvig (Author)