Analyzing customer data for targeted marketing campaigns

Author: Jay Xiong
July 11, 2022

Table of Contents

PART 1 - PROJECT DEFINITION 2

PART 2 - ANALYSIS 2

2.1 EXPLORE THE AZDIAS DATASET 2
2.2 EXPLORE THE CUSTOMERS DATASET 4
2.3 EXPLORE THE ATTRIBUTES VALUES AND INFORMATION LEVELS DATASET 5

PART 3 - METHODOLOGY 7

3.1 DATA PRE-PROCESSING 7
3.2 DATA CLEANING IMPLEMENTATION 9
3.3 METHODOLOGIES USED AT CUSTOMER SEGMENTATION REPORT 10
3.4 METHODOLOGIES USED FOR SUPERVISED LEARNING MODEL 14
3.5 REFINEMENT 17

PART 4 - RESULTS 18

4.1 MODEL EVALUATION AND VALIDATION 19
4.2 JUSTIFICATION 19

PART 5 - CONCLUSION 21

5.1 REFLECTION 21
5.2 IMPROVEMENT 21

Part 1 - Project Definition

Overview
This is the Capstone project of the Data Scientist Nanodegree program at Udacity. I choose to focus on the Batelsmann project, which is using data provided by AZ Direct GmbH to create a customer segmentation report and a supervised learning model.

Problem statement
Through this project, we would like to find a solution that helps a mail-order company to acquire new clients more efficiently. We have the attributes and demographic information of the existing clients. Through analyzing the attributes of the existing clients, we want to match them with a bigger dataset that includes attributes for people in Germany and figure out which people in Germany at most likely new customers for our client, the mail-order company selling organic products.

Before the solution is adopted, our client makes decisions based on experience and intuition. Now with the solution, our client will be able to make decisions based on data insights.

Metrics
How efficient can the client target their customers can technically be evaluated by the accuracy score of the model chosen and the roc_auc score. At a scale of 0 to 1, the higher the accuracy score and the roc_auc score the better the results.

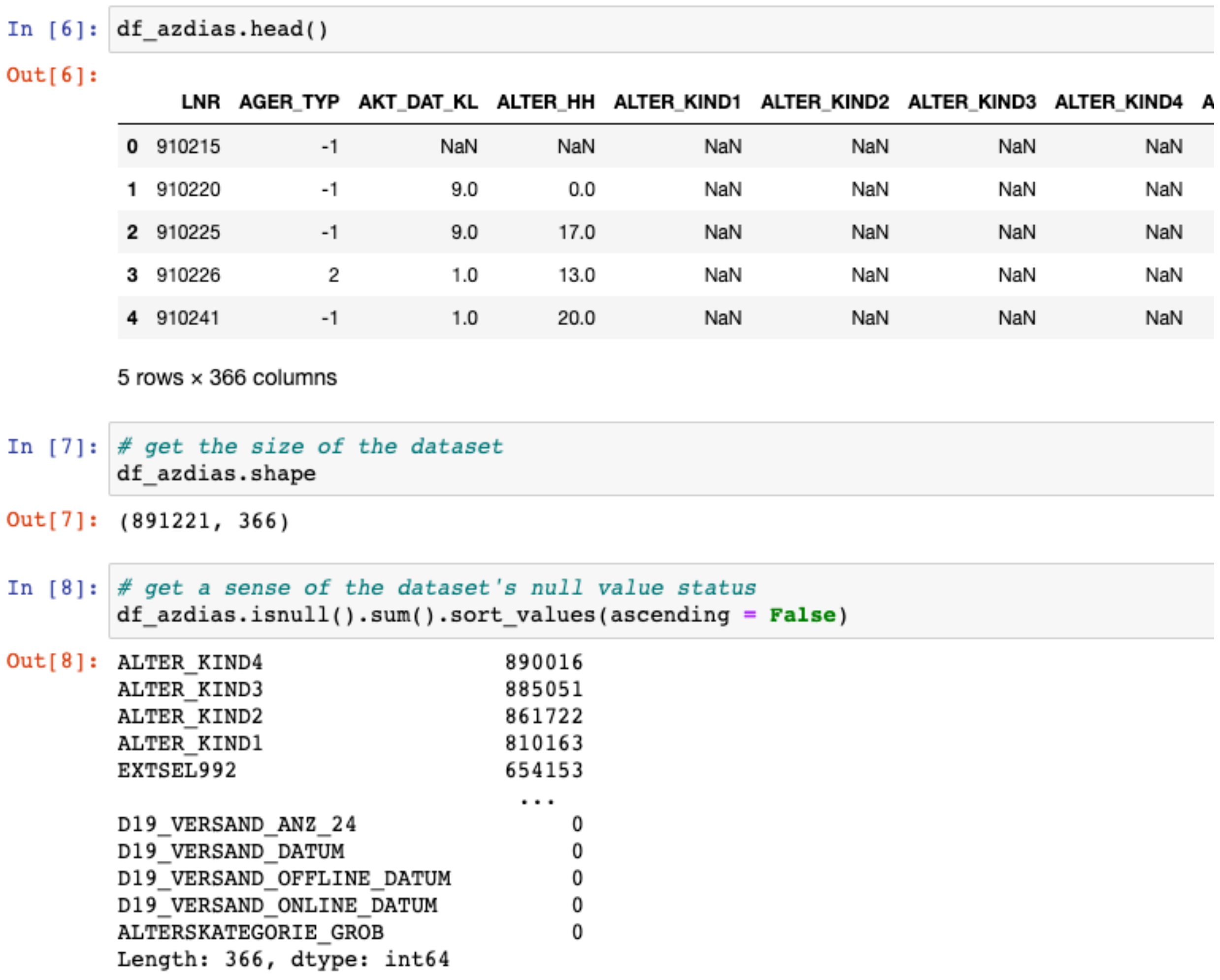
Part 2 - Analysis

We first start with understanding the data, which includes data exploration and data visualization combined exercise.

2.1 explore the AZDIAS dataset

To do this, I created two copies of the same dataset,
- **df_azdias:** a copy of the azdias dataset, used for data exploration and analysis
- **df_azdias_prep**: a copy of the azdias dataset, used for the actual implementation of data cleaning

I find this is a very big dataset with 891,221 rows and 366 columns, and it has lots of null values.



In [6]:
```
df_azdias.head()
```

Out[6]:

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	A
0	910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	
1	910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	
2	910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	
3	910226	2	1.0	13.0	NaN	NaN	NaN	NaN	
4	910241	-1	1.0	20.0	NaN	NaN	NaN	NaN	

5 rows × 366 columns

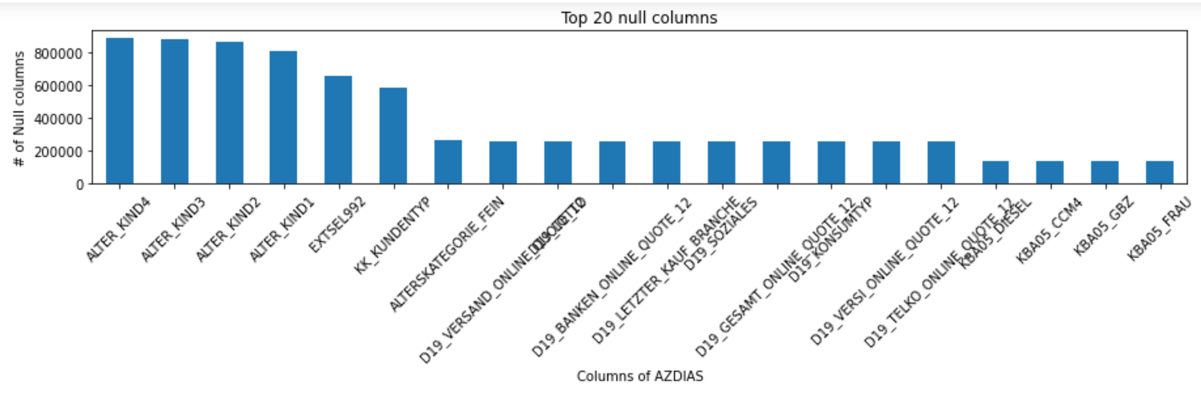
In [7]:
```
# get the size of the dataset
df_azdias.shape
```

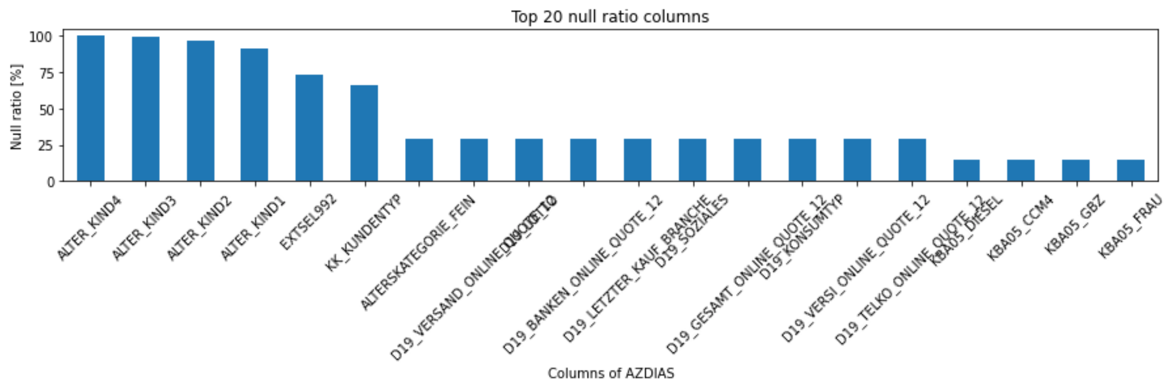
Out[7]:
```
(891221, 366)
```

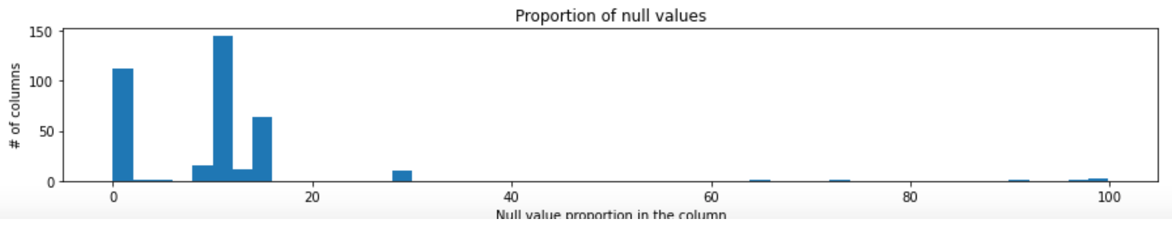
In [8]:
```
# get a sense of the dataset's null value status
df_azdias.isnull().sum().sort_values(ascending = False)
```

Out[8]:
```
ALTER_KIND4                    890016
ALTER_KIND3                    885051
ALTER_KIND2                    861722
ALTER_KIND1                    810163
EXTSEL992                      654153
                                ...
D19_VERSAND_ANZ_24                  0
D19_VERSAND_DATUM                   0
D19_VERSAND_OFFLINE_DATUM           0
D19_VERSAND_ONLINE_DATUM            0
ALTERSKATEGORIE_GROB                0
Length: 366, dtype: int64
```

It is easier to visualize the null value status







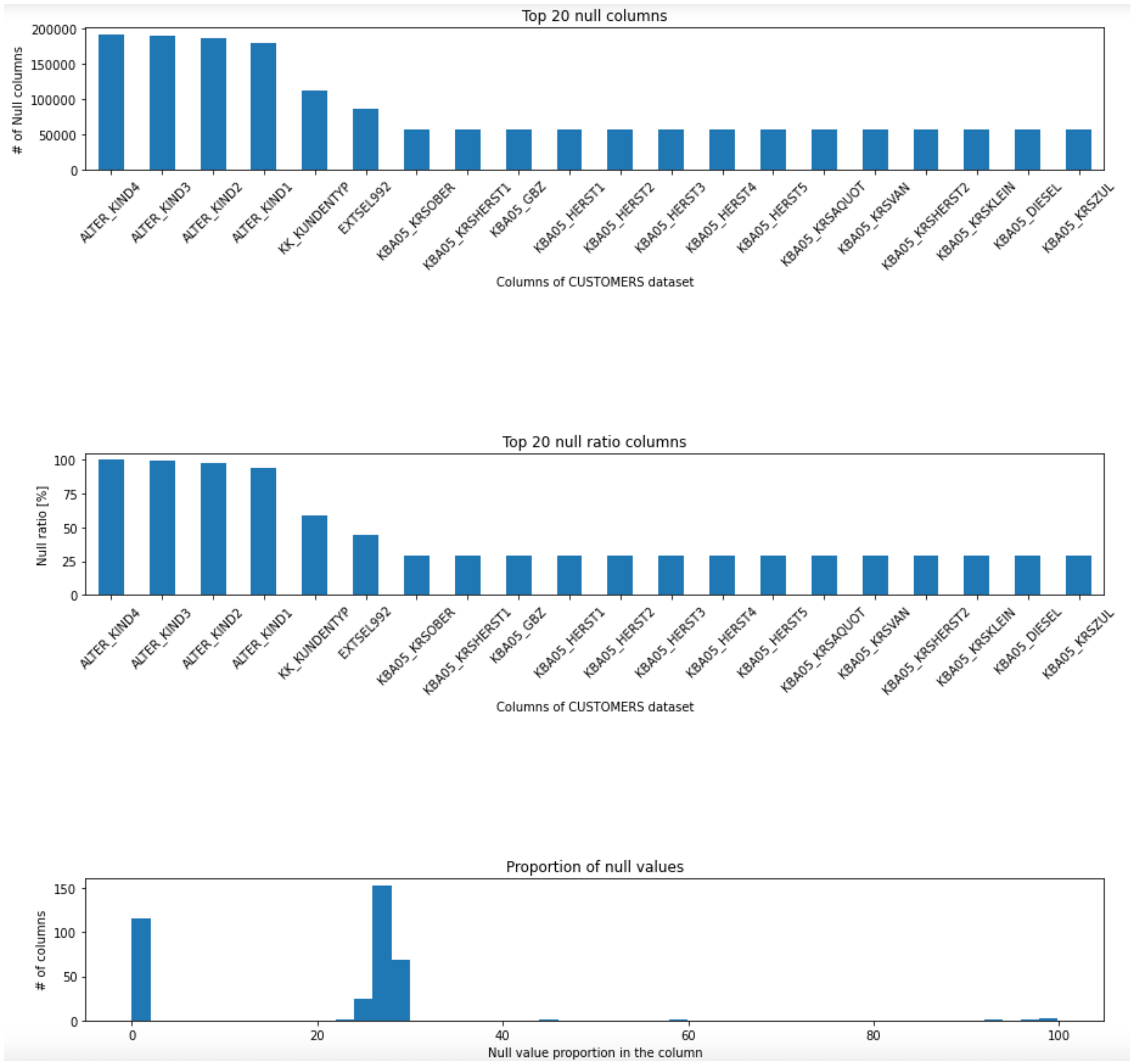
2.2 Explore the CUSTOMERS dataset

I took a similar approach as described in 2.1, created two copies of the CUSTOMERS dataset

- **df_customers:** a copy of the customers dataset, used for data exploration and analysis
- **df_customers_prep**: a copy of the customers dataset, used for the actual implementation of data cleaning

This is a dataset with 191,652 rows and 369 columns. It is smaller than the azdias dataset, but it has 3 more columns.

The null value status of the customers dataset is similar to that of azdias.



For both datasets, we can see right away they have 4 common columns that have 90%+ null values. These columns shall be dropped during data cleaning. But that should not be all, we will analyze more to find out what other columns need to be dropped.

2.3 Explore the Attributes values and information levels dataset

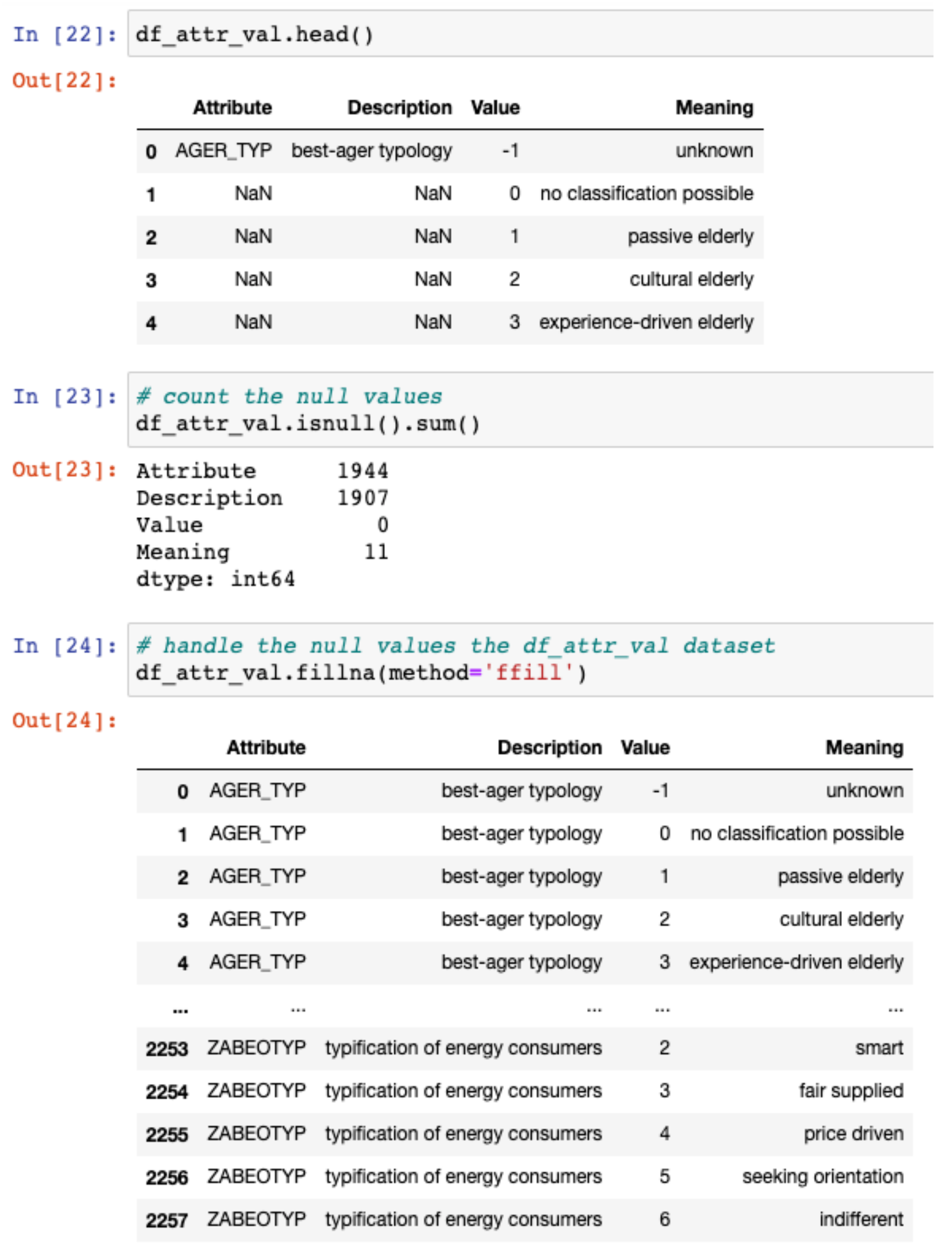
For this, I created two dataset copies,

- **df_attr_val:** a copy of the attribute values dataset
- **df_attr_info**: a copy of the attribute information levels dataset

These two datasets help us understand better what the attributes (in azdias and customers) datasets mean.

For example, in both azdias and customers dataset, a column head (attribute) named “D19_GESAMT_ANZ_24” means nothing to us, until we cross check the “DIAS Attributes - Values 2017.xlsx” dataset and look up its Description and Meaning columns. The attributes values dataset is especially useful here.

Unfortunately, the attributes values dataset has lots of null values too.



```
In [22]: df_attr_val.head()
```

Out[22]:

	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

```
In [23]: # count the null values
         df_attr_val.isnull().sum()
```

```
Out[23]: Attribute      1944
         Description    1907
         Value             0
         Meaning          11
         dtype: int64
```

```
In [24]: # handle the null values the df_attr_val dataset
         df_attr_val.fillna(method='ffill')
```

Out[24]:

	Attribute	Description	Value	Meaning
0	AGER_TYP	best-ager typology	-1	unknown
1	AGER_TYP	best-ager typology	0	no classification possible
2	AGER_TYP	best-ager typology	1	passive elderly
3	AGER_TYP	best-ager typology	2	cultural elderly
4	AGER_TYP	best-ager typology	3	experience-driven elderly
...	...	...	...	...
2253	ZABEOTYP	typification of energy consumers	2	smart
2254	ZABEOTYP	typification of energy consumers	3	fair supplied
2255	ZABEOTYP	typification of energy consumers	4	price driven
2256	ZABEOTYP	typification of energy consumers	5	seeking orientation
2257	ZABEOTYP	typification of energy consumers	6	indifferent

I used forward filling to fill the null values there. There are other strategies you could use too, such as backward filling, or simply dropping them. Dropping these records doesn't look like a good strategy, because for example, 1907 out of 2258 description fields have null values. Dropping them will make this whole dataset useless.

Part 3 - Methodology

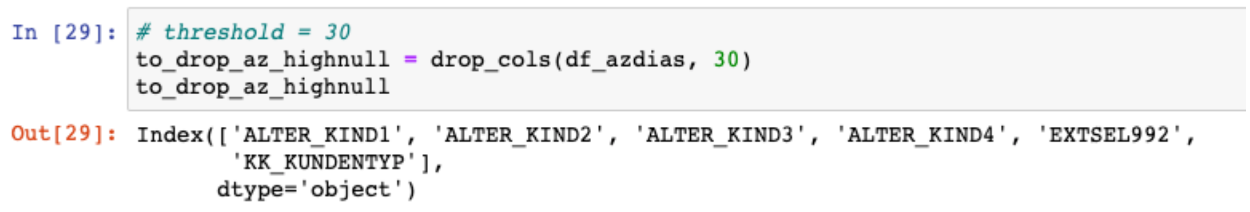
3.1 Data pre-processing

In this part, my strategy is dig deeper into the datasets that gives me clue what kind of data cleaning needs to be implemented later. Hence, in most of the pre-processing, I am not cleaning the data at the same time.

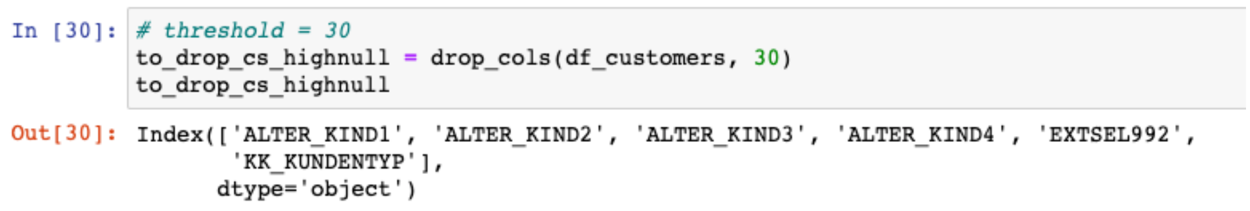
I tried to answer a few key questions

3.1.1 What is a good missing value threshold to drop columns?

I found or both AZDIAS and CUSTOMERS datasets, there are 98% of columns that have less than 30 percent of missing values. I decide to drop columns that have 30% or more missing values.



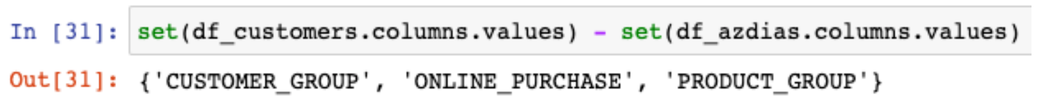
```
In [29]: # threshold = 30
         to_drop_az_highnull = drop_cols(df_azdias, 30)
         to_drop_az_highnull

Out[29]: Index(['ALTER_KIND1', 'ALTER_KIND2', 'ALTER_KIND3', 'ALTER_KIND4', 'EXTSEL992',
                'KK_KUNDENTYP'],
               dtype='object')
```



```
In [30]: # threshold = 30
         to_drop_cs_highnull = drop_cols(df_customers, 30)
         to_drop_cs_highnull

Out[30]: Index(['ALTER_KIND1', 'ALTER_KIND2', 'ALTER_KIND3', 'ALTER_KIND4', 'EXTSEL992',
                'KK_KUNDENTYP'],
               dtype='object')
```

Both azdias and customers datasets have 6 common columns to drop

3.1.2 What are the 3 extra columns that are in the customers dataset but are not in the azdias dataset?

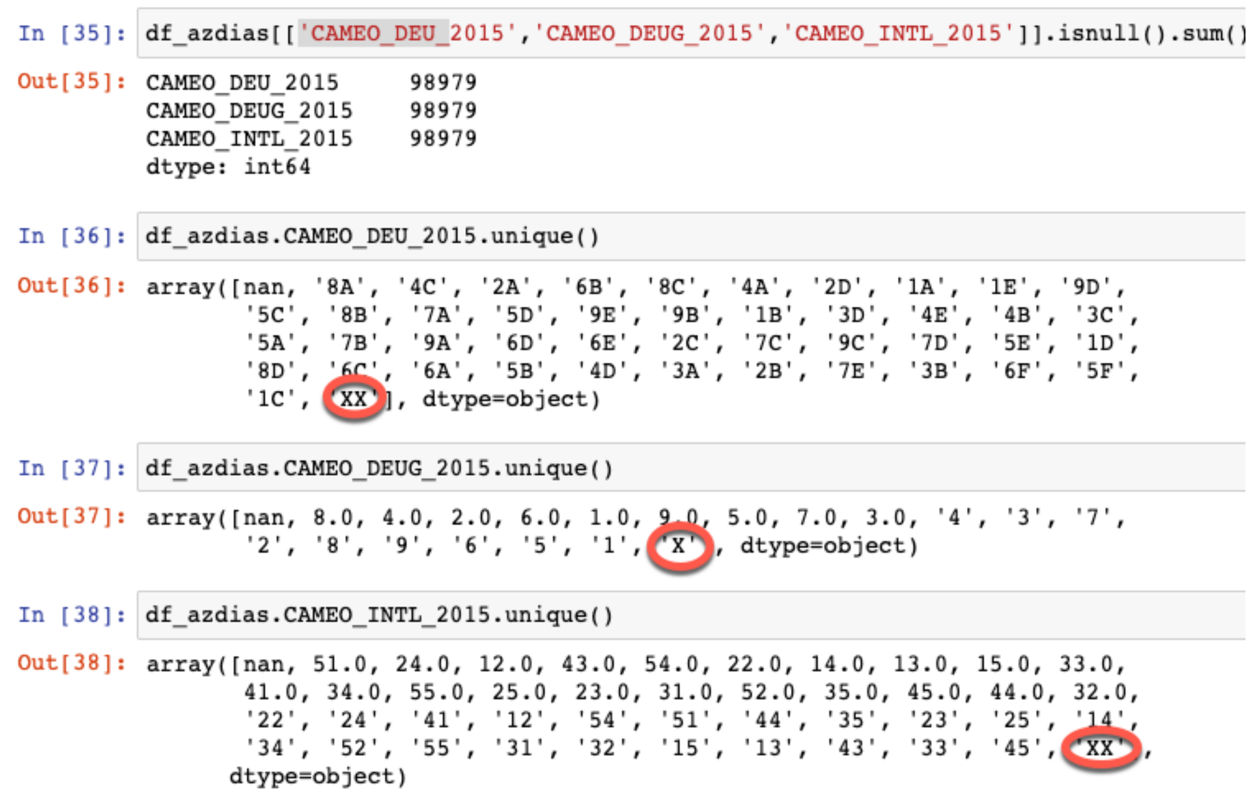


```
In [31]: set(df_customers.columns.values) - set(df_azdias.columns.values)

Out[31]: {'CUSTOMER_GROUP', 'ONLINE_PURCHASE', 'PRODUCT_GROUP'}
```

3.1.3 How about object data types in the dataset?

I found

- There are 6 columns (attributes) that are of object data type
- The D19_LETZTER_KAUF_BRANCHE and EINGEFUEGT_AM columns have too many unique values that doesn't help with prediction
- The columns CAMEO_DEU_2015','CAMEO_DEUG_2015'and 'CAMEO_INTL_2015' have lots of null values, we want to fill the null values in the data cleaning implementation stage. They also have mixed value types that we want to replace X and XX in the data cleaning implementation stage



```
In [35]: df_azdias[['CAMEO_DEU_2015','CAMEO_DEUG_2015','CAMEO_INTL_2015']].isnull().sum()

Out[35]: CAMEO_DEU_2015     98979
         CAMEO_DEUG_2015    98979
         CAMEO_INTL_2015    98979
         dtype: int64

In [36]: df_azdias.CAMEO_DEU_2015.unique()

Out[36]: array([nan, '8A', '4C', '2A', '6B', '8C', '4A', '2D', '1A', '1E', '9D',
                '5C', '8B', '7A', '5D', '9E', '9B', '1B', '3D', '4E', '4B', '3C',
                '5A', '7B', '9A', '6D', '6E', '2C', '7C', '9C', '7D', '5E', '1D',
                '8D', '6C', '6A', '5B', '4D', '3A', '2B', '7E', '3B', '6F', '5F',
                '1C', 'XX'], dtype=object)

In [37]: df_azdias.CAMEO_DEUG_2015.unique()

Out[37]: array([nan, 8.0, 4.0, 2.0, 6.0, 1.0, 9.0, 5.0, 7.0, 3.0, '4', '3', '7',
                '2', '8', '9', '6', '5', '1', 'X'], dtype=object)

In [38]: df_azdias.CAMEO_INTL_2015.unique()

Out[38]: array([nan, 51.0, 24.0, 12.0, 43.0, 54.0, 22.0, 14.0, 13.0, 15.0, 33.0,
                41.0, 34.0, 55.0, 25.0, 23.0, 31.0, 52.0, 35.0, 45.0, 44.0, 32.0,
                '22', '24', '41', '12', '54', '51', '44', '35', '23', '25', '14',
                '34', '52', '55', '31', '32', '15', '13', '43', '33', '45', 'XX'],
               dtype=object)
```

3.1.4 *How about high correlation of the attributes?*

In datasets, highly correlated attributes (features) mean they are somewhat linearly dependent with other features. They contribute very less in predicting the output but increases the computational cost, so we want to find out these attributes and drop them later

Interestingly, how high is high could be debatable. I'd rather take an experimental approach. In this exercise, I take a correlation threshold of 0.8. You could go for .7 or .95 and try the difference. The higher the threshold, the less number of attributes you are going to drop.

3.1.5 *How about the unknowns?*

In the attributes values dataset, we have a Meaning column. Some attribute's meaning is marked as unknown when the attribute's value is -1, 0 or 9. These unknowns are not helpful for us to understand the attributes. I'd like reencode the unknowns by NaNs.

3.1.6 Now how to handle missing data/null values?

We haven't actually processed null (missing) values as of this step. In the data cleaning implementation stage we do need to handle them.

There are a few different ways to handle missing data. For example, deleting missing data or imputing missing data.

Deleting only makes sense when the amount of missing data is really small, and when they are missing at random (MAR) or missing completely at random (MCAR) types. Otherwise you could end up deleting useful data.

Even about imputation itself, there are different ways. Some popular ways are replacing with Mean, Mode or Meidan, replacing with the previous value (ffill) or replacing with the next value (bfill), and replacing with the most frequent value, etc.

The topic itself is huge. In the meantime, it is also very experimental by nature. We should understand how each method works. I picked replacing with Mode which you will see in the implementation.

3.2 Data Cleaning Implementation

With the previous pre-processing, I have developed a step-by-step strategy on how I should clean the data.

I put them all together into one function to do this job, which includes,

- Drop 3 extra unique columns in the customers dataset
- Drop columns with more than 30% missing data
- Drop columns with high correlation
- Encode features with object data type (replace 'X', 'XX' by -1, fillna with -1, map 'W' to 1, map 'O' to 2, etc.)
- Handle 'unknowns' – replace missing value with -1, 0 or 9 respectively for corresponding fields that have -1, 0, 9 marked as unknown in the Meaning field

In addition,

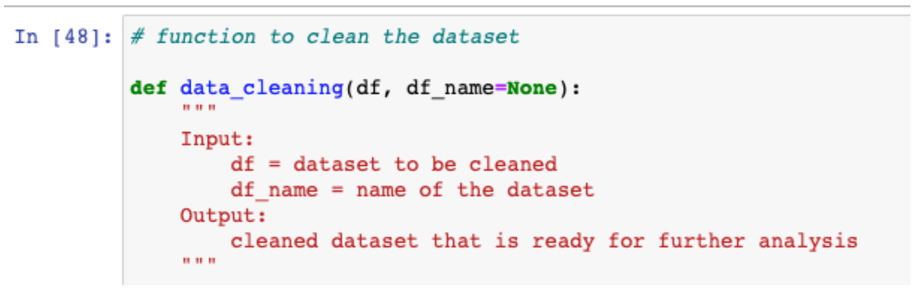
- Convert categorical variables with one-hot encoding
- Impute missing values with Mode

Lastly, scale and standardize the dataset

- Variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bias

- When variables have different ranges, we change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values

You can see in my code that I defined a data_cleaning function to put them all together.



```
In [48]: # function to clean the dataset

         def data_cleaning(df, df_name=None):
             """
             Input:
                 df = dataset to be cleaned
                 df_name = name of the dataset
             Output:
                 cleaned dataset that is ready for further analysis
             """
```

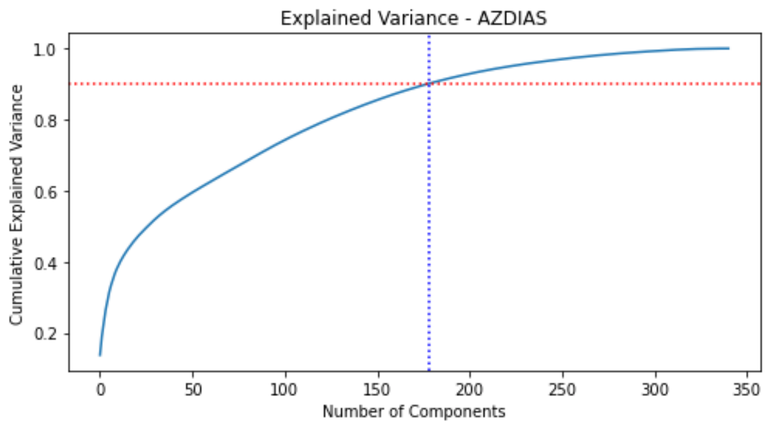
3.3 Methodologies used at Customer Segmentation Report

3.3.1 Principal Component Analysis (PCA)

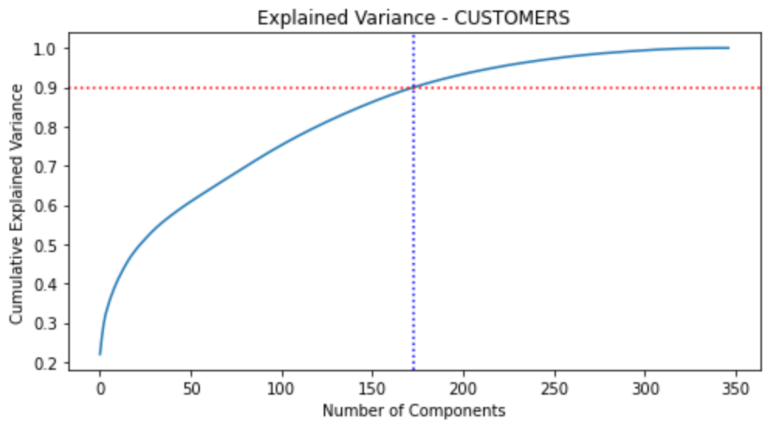
The datasets we use have lots of features. It could take impossibly long model training time to complete the analysis. We want to transform high-dimensions data to lower-dimensions while retaining as much information as possible. Therefore, let's use the Principle Component Analysis (PCA) method.

I wrote a function to plot the PCA analysis

```
In [56]: # let's see what the visualization look like if we want to preserve 90% of variance
         plot_min_pcom(df_azdias_clean, 0.9, 'AZDIAS')
```



```
In [57]: # let's see what the visualization look like if we want to preserve 90% of variance
         plot_min_pcom(df_customers_clean, 0.9, 'CUSTOMERS')
```



As you can see, we can preserve **90%** of the variance by reducing the number of components from more than 300 to less than 200. So I decide to keep **180** components for further analysis.

With n_components = 180 chosen, I wrote a function to reduce dimensions of the dataset. For both azdias and customers datasets, I am able to preserve ~90% of the variance

```
In [59]: # reduce the azdias dataset and preint the preserved variance ratio score
         reduced_azdias = reduce_dimensions(df_azdias_clean, 180)

         Percentage of variance explained with reduced components 0.9015406775429244:

In [60]: # reduce the customers dataset and print the preserved variance ratio score
         reduced_customers = reduce_dimensions(df_customers_clean, 180)

         Percentage of variance explained with reduced components 0.9075717213155874:
```

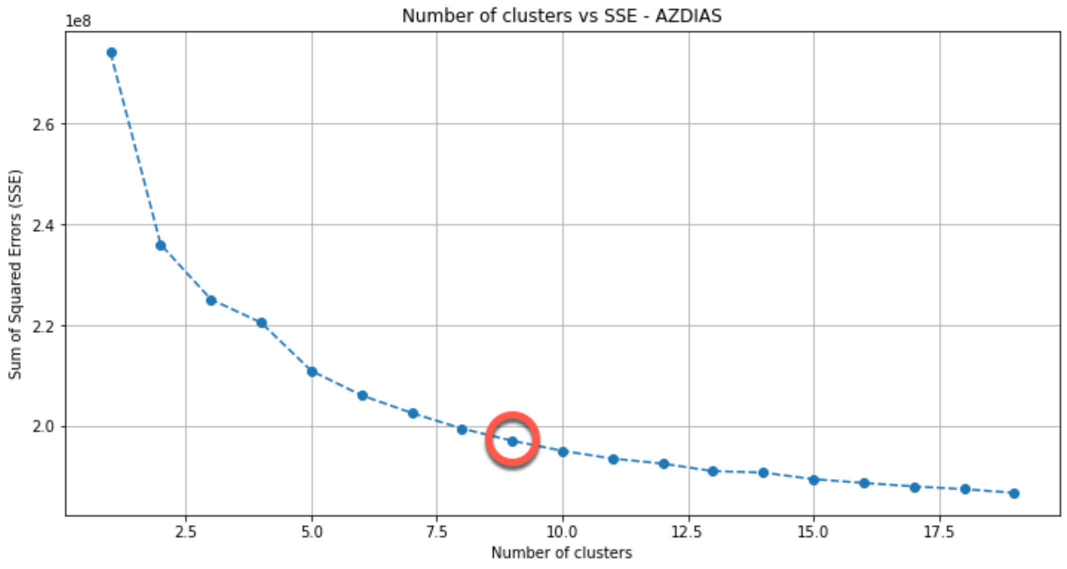
3.3.3 KMeans Clustering

With the dataset dimensions largely reduced, now I can go ahead creating the clusters as this is an unsupervised learning.

I need to get the optimal number of clusters to do KMeans clustering. The method I chose is called elbow method. The idea is to run k-means clustering on a given dataset for a range of values of k, and for each value of k, calculate sum of squared errors (SSE). After that, we plot a line graph of the SSE for each value of k. Here, we want to minimize SSE. SSE tends to decrease toward 0 as we increase k. My goal is to choose a small value of k that still has a low SSE, and the elbow usually represents where we start to have diminishing returns by increasing k.

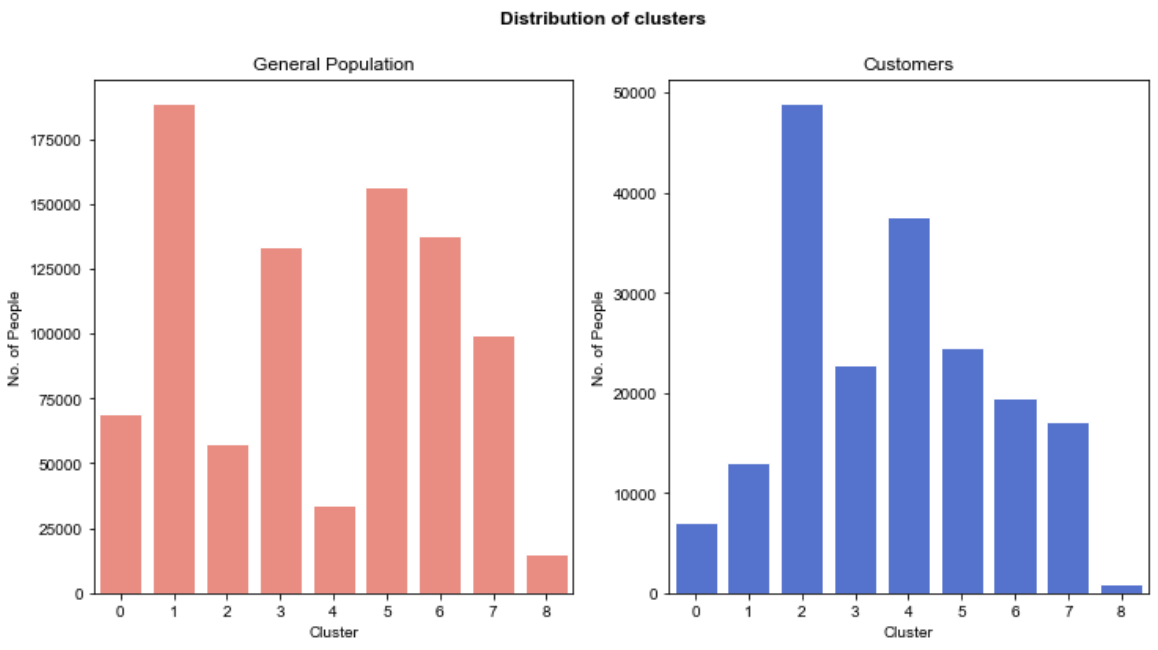
I wrote a function to plot the curve of clusters.

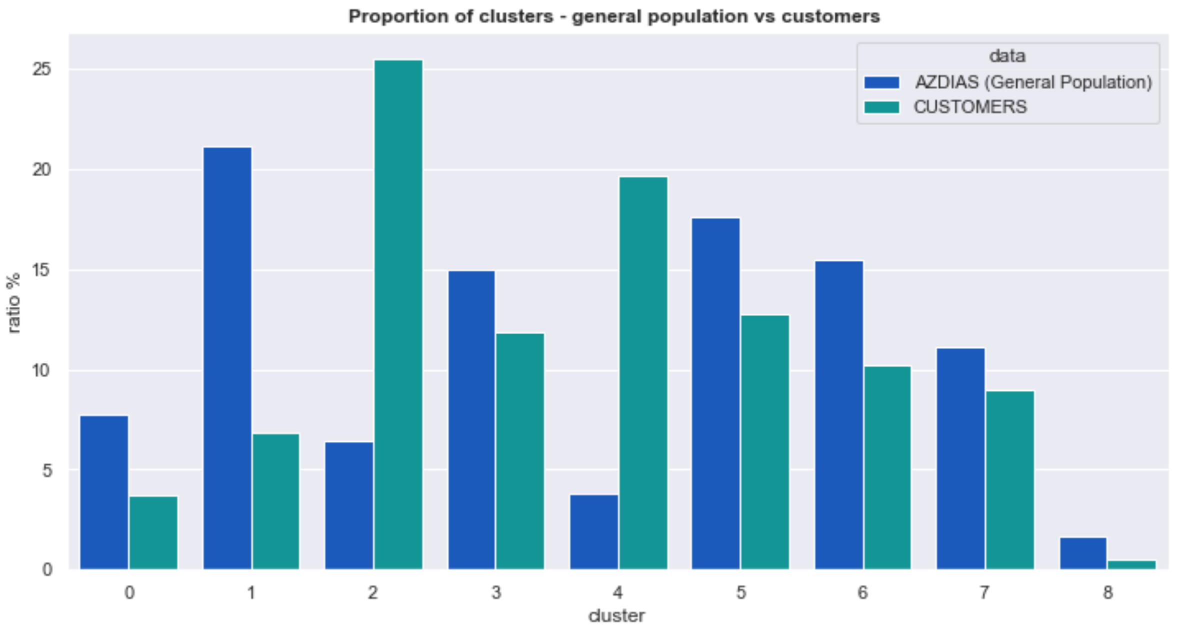
```
In [62]: plot_optimal_cluster(reduced_azdias, 'AZDIAS')
```



The SSE slope decreases sharply until it reaches 9 clusters, then the rest of the curve looks a mot more smooth. So I choose k =9 as the number of clusters.

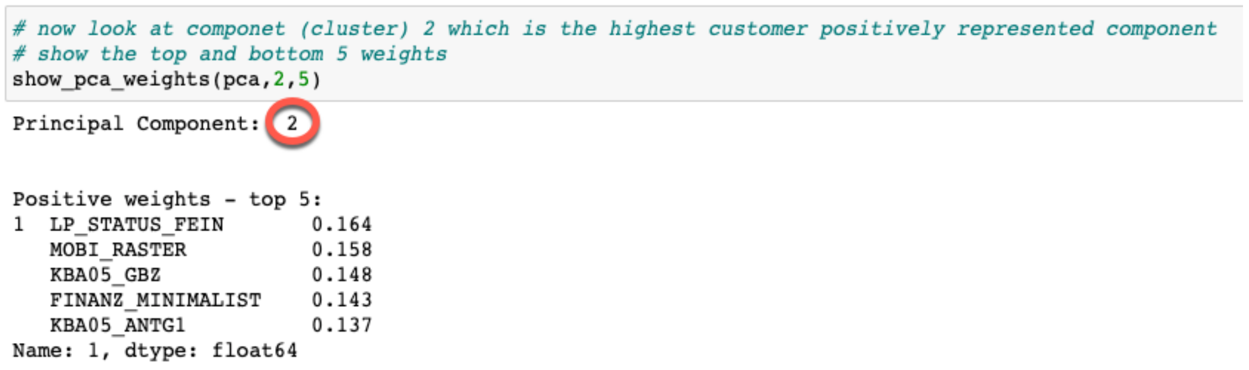
Then I am able to KMeans clustering algorithm to get the culsters and the distribution of number of people in each cluster for both the azdias (general population in Germany) dataset and the customers dataset.





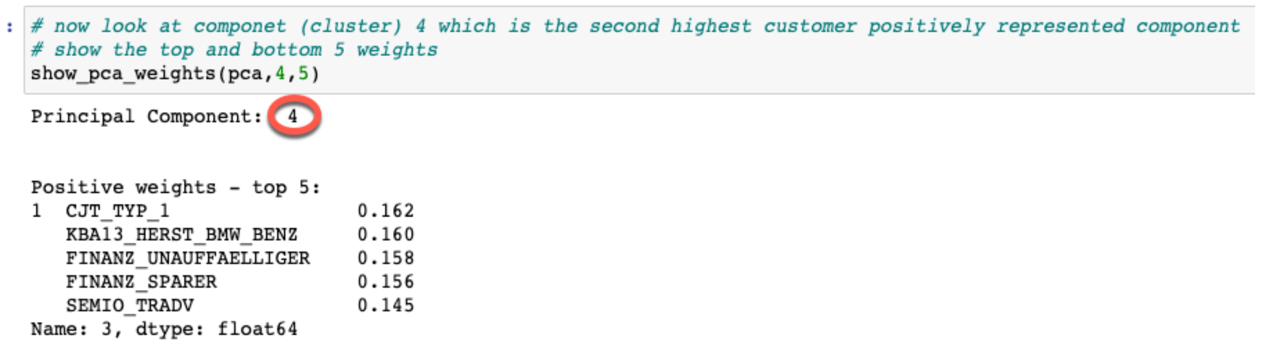
The above visualizations show that,

- General population is distributed across all clusters
- Customers are mostly from clusters 2 and 4 (results could be different in a new run)
- At the same scale, the ratio of customer proportion is higher than that of general population in clusters 2 and 4.
- This tells us these 2 clusters contain more customers, and they can be target for future customers. (results could be different in a new run)



```
# now look at componet (cluster) 2 which is the highest customer positively represented component
# show the top and bottom 5 weights
show_pca_weights(pca,2,5)
```

```
Principal Component: 2


Positive weights - top 5:
1  LP_STATUS_FEIN        0.164
   MOBI_RASTER           0.158
   KBA05_GBZ             0.148
   FINANZ_MINIMALIST     0.143
   KBA05_ANTG1           0.137
Name: 1, dtype: float64
```



```
# now look at componet (cluster) 4 which is the second highest customer positively represented component
# show the top and bottom 5 weights
show_pca_weights(pca,4,5)
```

```
Principal Component: 4


Positive weights - top 5:
1  CJT_TYP_1               0.162
   KBA13_HERST_BMW_BENZ    0.160
   FINANZ_UNAUFFAELLIGER   0.158
   FINANZ_SPARER           0.156
   SEMIO_TRADV             0.145
Name: 3, dtype: float64
```

By identifying the 2 positively represented components and by crossing checking the description of their top weights,
we can conclude the target customers share a few common attributes,
- have low financial interest
- have number of 1-2 family houses
- money savers
- traditional minded
- share BMW & Mercedes Benz within the PLZ8

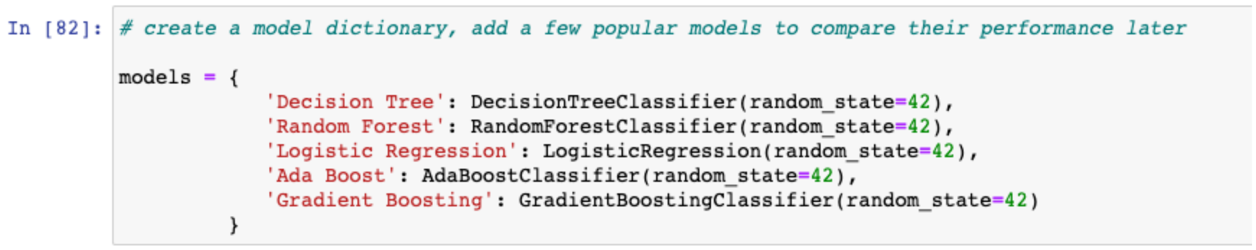
3.4 Methodologies used for Supervised Learning Model

3.4.1 Compare 5 popular classification algorithms

For the mail-order company, the mailout data has a "RESPONSE" column, that states whether or not a person became a customer of the company following the campaign.
The output data of the predication is of categorical type, a Yes vs No answer. Therefore, we should use classification algorithms to make the prediction.

Instead of using one algorithm, I chose to use 5 algorithms to compare their performance.

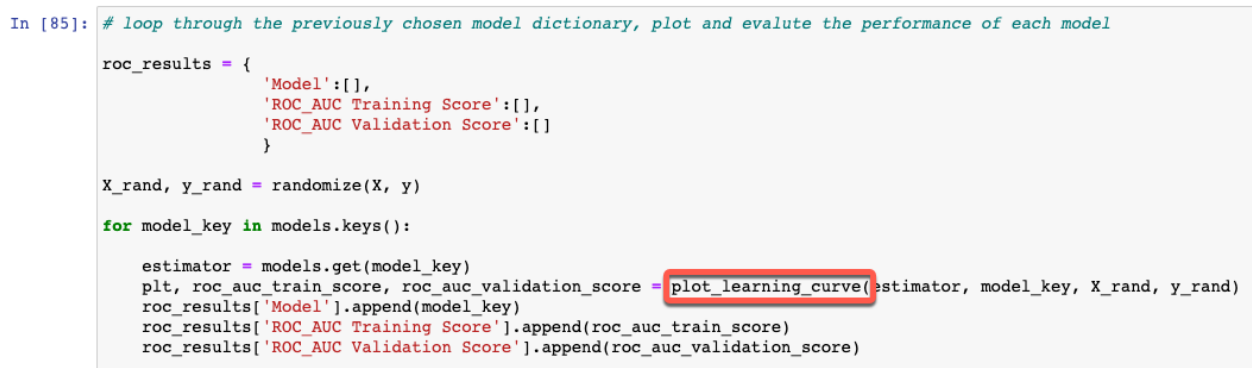
I created a dictionary of models using these 5 algorithms. Then I ran through predictions from all 5 models in a for loop.



```
In [82]: # create a model dictionary, add a few popular models to compare their performance later

         models = {
                       'Decision Tree': DecisionTreeClassifier(random_state=42),
                       'Random Forest': RandomForestClassifier(random_state=42),
                       'Logistic Regression': LogisticRegression(random_state=42),
                       'Ada Boost': AdaBoostClassifier(random_state=42),
                       'Gradient Boosting': GradientBoostingClassifier(random_state=42)
                  }
```

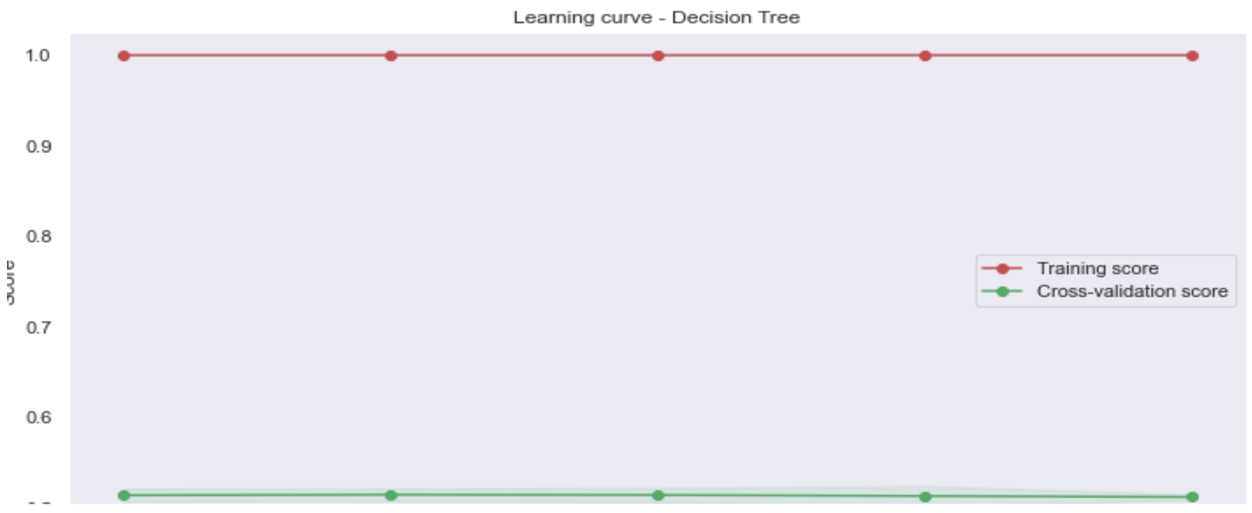
3.4.2 Plot learning curves

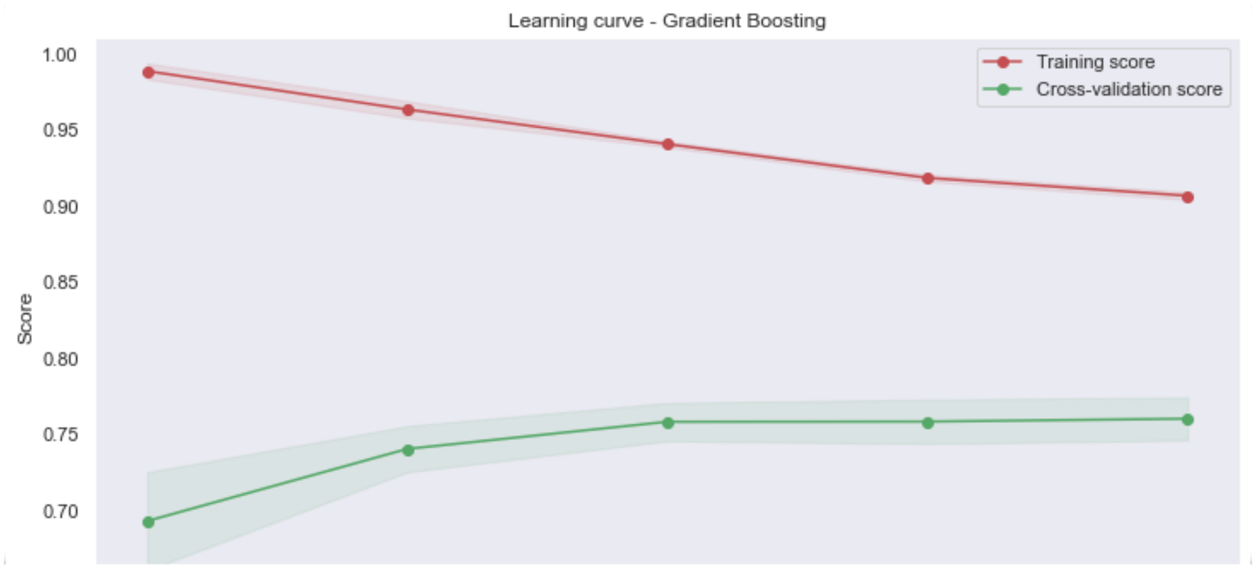
For observing the model performance in a visual way, I wrote a function to plot the learning curve. The plot learning curve method basically plots the model training score and n-fold cross validation score, shows the trending of these scores as more data is provided.



```
In [85]: # loop through the previously chosen model dictionary, plot and evalute the performance of each model

         roc_results = {
                         'Model':[],
                         'ROC_AUC Training Score':[],
                         'ROC_AUC Validation Score':[]
                         }

         X_rand, y_rand = randomize(X, y)

         for model_key in models.keys():

             estimator = models.get(model_key)
             plt, roc_auc_train_score, roc_auc_validation_score = plot_learning_curve(estimator, model_key, X_rand, y_rand)
             roc_results['Model'].append(model_key)
             roc_results['ROC_AUC Training Score'].append(roc_auc_train_score)
             roc_results['ROC_AUC Validation Score'].append(roc_auc_validation_score)
```

Combined with the plot learning curve method, I also used roc_auc score to calculate performance of each model.

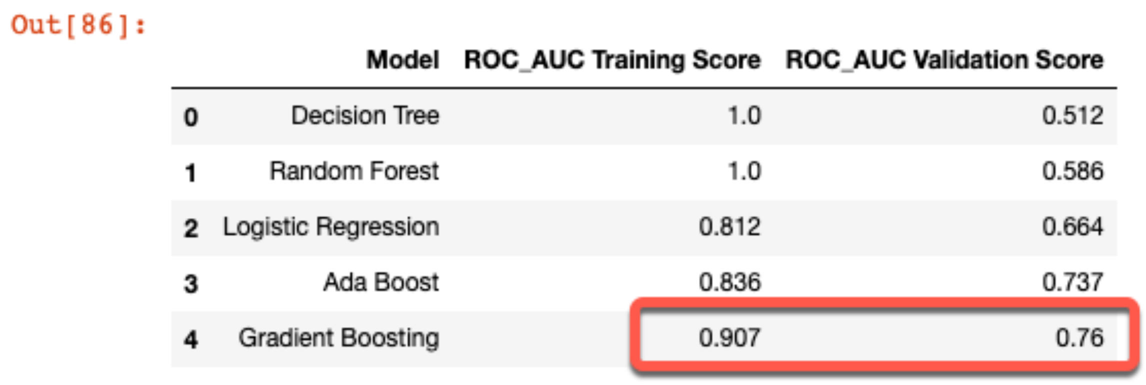




Let’s compare the learning curves of the Random Forest classifier and the Gradient Boosting classifier.   You can see the training score of the Random Forest classifier is pretty high (close to 1.0), but the cross validation score is very low (slightly above 0.5), and the two curves don’t converge, which means the model is overfitting.

On the contrary, although the training score of the Gradient Boosting classifer is a little bit lower (still .90+), the cross validation score is much higher, and the two curves are show a converging trend. Given more data, I have reason to believe the curves will converge.

At the end, by look at the roc_auc scores, I am able to see that the Gradient Boostinng Classifier is the winner among the 5 chosen models.



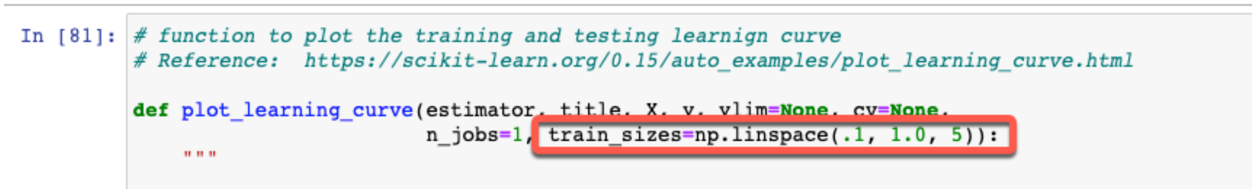
Out[86]:

	Model	ROC_AUC Training Score	ROC_AUC Validation Score
0	Decision Tree	1.0	0.512
1	Random Forest	1.0	0.586
2	Logistic Regression	0.812	0.664
3	Ada Boost	0.836	0.737
4	Gradient Boosting	0.907	0.76

3.5 Refinement

3.5.1 cross-validation

In fact, I have mentioned one refinement technique in chapter 3.4.2. When plotting the learning curves, I used 5-fold cross-validation. So that the cross-validation scores show a more real output. In the screenshot below, 5 is the number of samples to generate.



```
In [81]: # function to plot the training and testing learnign curve
         # Reference:  https://scikit-learn.org/0.15/auto_examples/plot_learning_curve.html

         def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                                 n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
             """
```

3.5.2 GridSearchCV

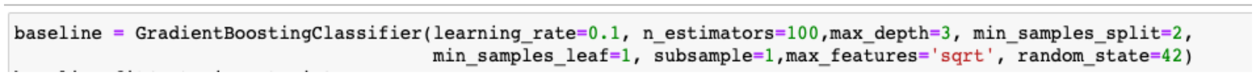
For model performance refinement, I focused on the GradientBoostingClassifier which is the best performing model and practiced refining its hyper-parameters with GridSearchCV.

Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes.

GridSearchCV implements a “fit” and a “score” method and it conducts exhaustive search over specified parameter values for an estimator.

My approach to tune the hyper-parameters is,

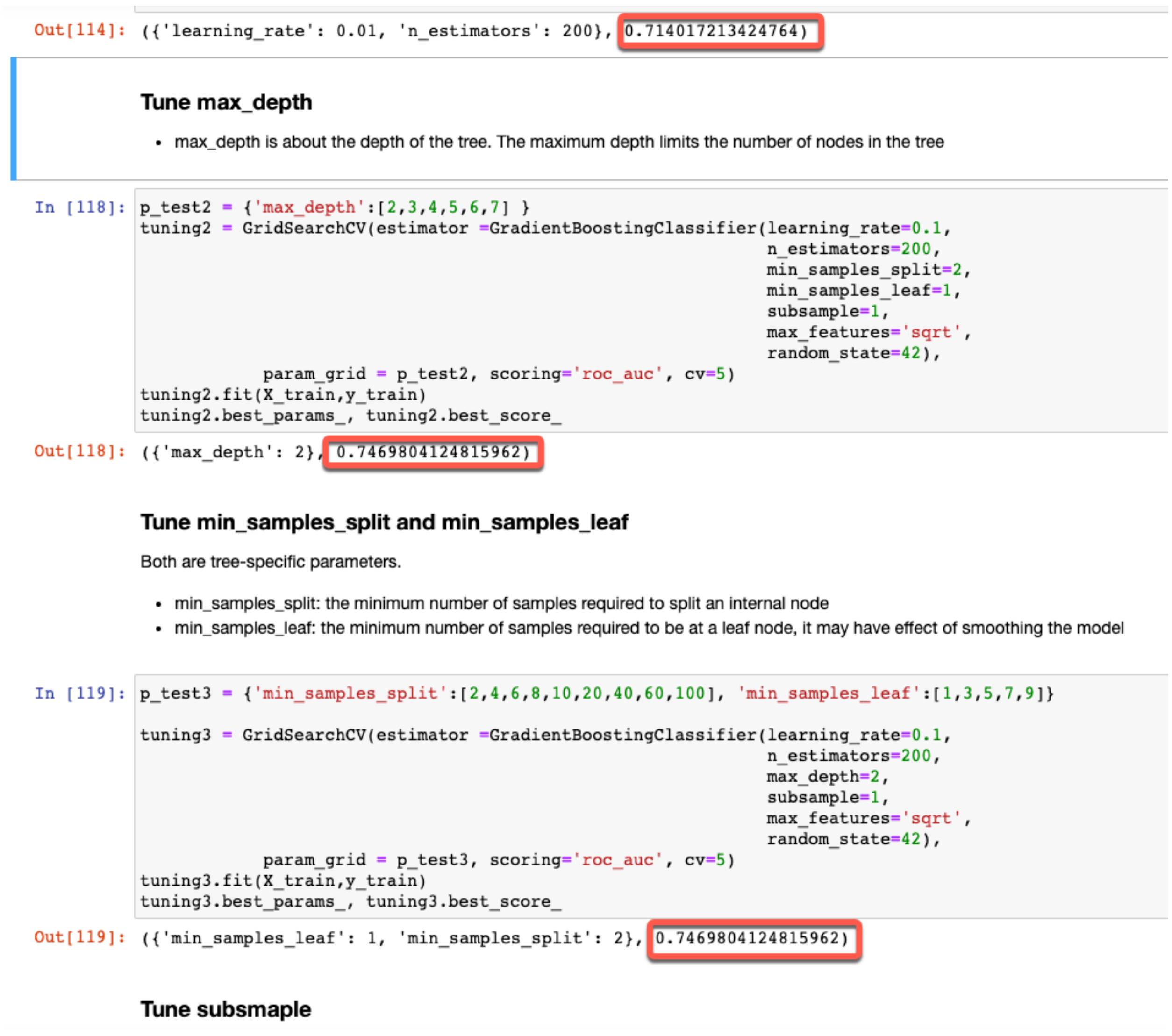
1) Create a baseline model and get its accuracy report



```
baseline = GradientBoostingClassifier(learning_rate=0.1, n_estimators=100,max_depth=3, min_samples_split=2,
                                      min_samples_leaf=1, subsample=1,max_features='sqrt', random_state=42)
```

2) Tune the key hyper-parameters step by step instead of tuning them all in one step, monitor the roc_auc score at each step

   2-1). Tune n_estimators and Learning rate
   2-2). Tune max_depth
   2-3). Tune min_samples_split and min_samples_leaf
   2-4). Tune subsample


```
Out[114]: ({'learning_rate': 0.01, 'n_estimators': 200}, 0.714017213424764)
```

Tune max_depth

- max_depth is about the depth of the tree. The maximum depth limits the number of nodes in the tree

```
In [118]: p_test2 = {'max_depth':[2,3,4,5,6,7] }
          tuning2 = GridSearchCV(estimator =GradientBoostingClassifier(learning_rate=0.1,
                                                                        n_estimators=200,
                                                                        min_samples_split=2,
                                                                        min_samples_leaf=1,
                                                                        subsample=1,
                                                                        max_features='sqrt',
                                                                        random_state=42),
                      param_grid = p_test2, scoring='roc_auc', cv=5)
          tuning2.fit(X_train,y_train)
          tuning2.best_params_, tuning2.best_score_
Out[118]: ({'max_depth': 2}, 0.7469804124815962)
```

Tune min_samples_split and min_samples_leaf

Both are tree-specific parameters.

- min_samples_split: the minimum number of samples required to split an internal node
- min_samples_leaf: the minimum number of samples required to be at a leaf node, it may have effect of smoothing the model

```
In [119]: p_test3 = {'min_samples_split':[2,4,6,8,10,20,40,60,100], 'min_samples_leaf':[1,3,5,7,9]}

          tuning3 = GridSearchCV(estimator =GradientBoostingClassifier(learning_rate=0.1,
                                                                        n_estimators=200,
                                                                        max_depth=2,
                                                                        subsample=1,
                                                                        max_features='sqrt',
                                                                        random_state=42),
                      param_grid = p_test3, scoring='roc_auc', cv=5)
          tuning3.fit(X_train,y_train)
          tuning3.best_params_, tuning3.best_score_
Out[119]: ({'min_samples_leaf': 1, 'min_samples_split': 2}, 0.7469804124815962)
```

Tune subsmaple

At each step, I get the best parameters (values, e.g. max_depth = 2) and I am able to see the roc_auc score gets higher.

3) Use findings from step 2 to get the best parameter values, then create a final model and get its accuracy report
4) Compare the roc_auc scores of the baseline and final models

Part 4 - Results

4.1 Model evaluation and validation

This has been discussed to some degree in chapter 3.3 and 3.4. Let me rephrase them in a summarized way.

4.1.1 KMeans clustering

I used the elbow method to find the optimal number of clusters k, where k =9 is the key parameter for the KMeans model.

With that, I created 9 clusters for the general population (azdias) and the customers datasets.

For the top 2 clusters that positively represent customers, I found the top weight features that describe the attributes of the customers our client should target in the marketing campaign.

4.1.2 Supervised learning

- The supervised learning part is to solve a classification problem with categorical results (yes vs no) as the prediction output
- Therefore, I chose to use 5 popular algorithms and built 5 models, namely
    - Decision Tree
    - Random Forest
    - Logistic Regression
    - Ada Boost
    - Gradient Boosting
- For all models, I gave a random state of 42 as the key parameter
- For each model, I calculated the train score, 5-fold cross validation score and roc_auc scores

4.2 Justification

This has also been discussed to some degree in chapter 3.3 and 3.4. Let me summarize them here.

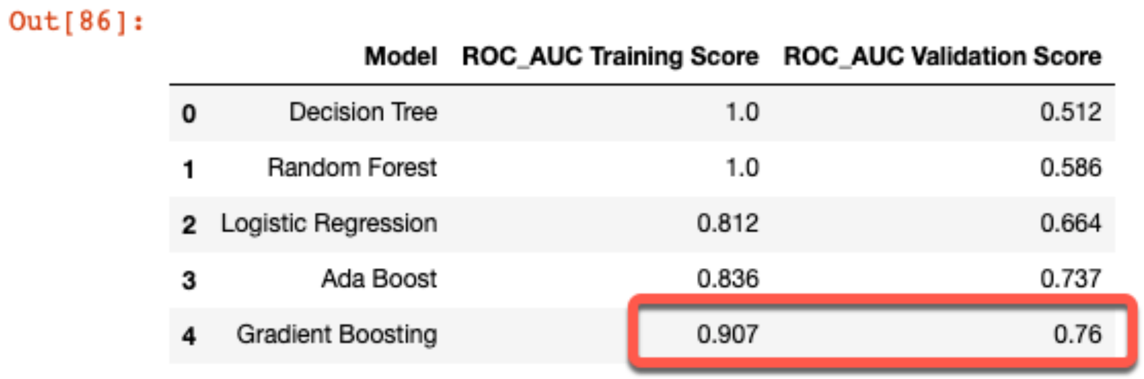
4.2.1 KMeans clustering

The key parameter here is what is the optimal number of clusters. I find a good approach is to visualize the SSE slope. As the number of clusters increase, the slope changes from dropping sharply to dropping very smoothly at a certain elbow point.

Then, to find the target customers, the key is to find out which clusters have over (positively) represented customers when compared to the general population. The technique here is to compare the proportion of clusters of the customers and the general population datasets. For example, cluster 2 and 4 have higher customer proportion.

4.2.2 Supervised Learning

- The GradientBoostingClassifier model turns out to the best performing model out of the 5



```
Out[86]:
```

	Model	ROC_AUC Training Score	ROC_AUC Validation Score
0	Decision Tree	1.0	0.512
1	Random Forest	1.0	0.586
2	Logistic Regression	0.812	0.664
3	Ada Boost	0.836	0.737
4	Gradient Boosting	0.907	0.76

It's roc_auc training score is the second highest, and its roc_auc validation score is the highest. From the learning curve, I can see the training and validation curves have a clear trend of converging. It means the model doesn't have an overfitting problem. Other models, such as the DecisionTreeClassifier and RandomForestClassifier show an overfitting problem.

4.2.3 Tuning parameters

For the GradientBoostingClassifier model, the first key approach here is to tune tree-specific parameters and boosting specific parameters.

Tree-specific:

- n_estimators
- max_depth
- min_samples_split
- min_samples_leaf
- subsample

Boosting-specific:

- learning-rate

The second key approach is to track the model's roc_auc score after each parameter tuning. For example, in the baseline model n_estimators = 100. In tuning n_estimators I found the best value for n_estimators is 200, I used it in the next step to tune max_depth, and I found the roc_auc score improved from 0.71 to 0.74.

Part 5 - Conclusion

5.1 Reflection

The project consists of 3 parts.
Part 1 is getting to know the data. This is where I load the data, explore and analyze the data and finally clean the data so it is ready for prediction.

Part 2 is Customer Segmentation Report, where I used PCA analysis, KMeans clustering to find the optimal number of clusters, create the clusters and identify which clusters represent the customers our client should target. My findings tell me the target customers have low financial interest and have 1-2 family houses. They are money savers who are traditional minded.

Part 3 is the Supervised Learning Model, where I chose 5 classification models, feed the mailout data to them and evaluate the model performance. I found the GradientBoostingClassifier is the best performing model among the 5.

Here are some additional findings,

1) Data preparation is super important. You can't do high accuracy prediction without high quality data. However, there is no written rules that you have to do X first, then Y, and then Z. I went through a few rounds of test and trial to figure out some best practices. For example,
    a. it is better to drop unnecessary columns for various cases before you do other cleaning
    b. Handle missing values itself is a gigantic topic, there are several strategies to choose, the best way to learn is to try and see what the results look like.
    c. I prefer to put all data cleaning steps into one relatively big function, so it is easier to have a full picture of what steps are taken. Plus, the function is reusable in later part of the project

2) Try a few different models for the same objective, then select the best performing model to get optimized results.

   If I only used Decision Tree or Random Forest, I could end up overlooking problems like overfitting.

5.2 Improvement

Here are a few ideas for future improvement

- Try some additional data cleaning methods, do more test and learn. For example, try different imputation strategy, try different encoding strategy, explore outliers more
- Running KMeans clustering takes a lot of time, I might try MiniBatchKMeans next time to improve the performance
- Try different threshold for determining what columns to drop, for deciding what is a high feature correlation, for selecting number of components to keep in PCA while preserving a fairly high percentage of variance