Capstone Project
Customer Segmentation Report for Arvato Financial Services

Project Overview

This is a capstone project for the Data Scientist Nanodegree provided in collaboration with Bertelsmann Arvato Analytics.

The project includes 4 data files and 2 other files that provide supporting information on these 4 files. The data files are:

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

The 2 Supporting files are:

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

Please note that the 2 supporting files were converted into xls format, as the pandas version installed on this environment is a bit old and doesn't support reading xlsx files.

Problem Statement

This project is trying to answer this business question: How can the client (mail-order company) acquire more clients more efficiently? The project is divided into 3 main parts:

- Unsupervised Learning: to Segment Customers
- Supervised Learning: to train a model to predict data for the mailout campaign.
- Kaggle Competition: to submit predictions of the mailout_test file using the model from the supervised learning part.

Metrics

The metric used for this project in the 3rd part (supervised learning) is the **Area Under the Curve** (AUC) for the **Receiver Operating Characteristic** curve (ROC). The ROC summarizes the trade-off between true positive rate and false positive rate for a predictive model using different probability points/scenarios.

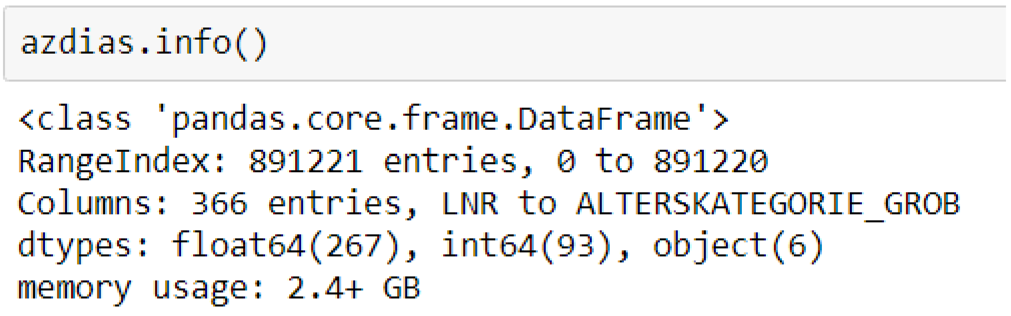
$$True\ Positive\ Rate = Sensitivity = \frac{\#True\ Positives}{\#True\ Positives + \#False\ Negatives}$$

$$False\ Positive\ Rate = 1 - Specifity = \frac{\#False\ Positive}{\#True\ Negatives + \#False\ Positives}$$

The roc_auc_score from thee sklearn.metrics package is used to evaluate this metric. The main reason for preferring this metric over the straightforward accuracy metric is because the labels for the mailout campaign are heavily skewed towards one class (No Response is 90% of data). Therefore, AUC is objectively better here. Another option was to use Precision-Recall curve. However, since the 3rd part (Kaggle Competition) requires ROC-AUC, it was used for part 2 as well.

Data Exploration

The first thing to notice about the azdias data is the warning when reading the file about columns 18 and 19 being of mixed types (shown below). Which was investigated later. However, let us start by just looking at the basic structure of the data (azdias).



```
azdias.info()
```

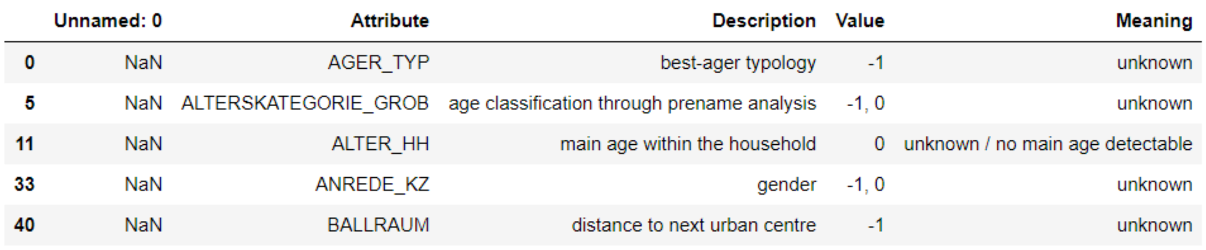
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891221 entries, 0 to 891220
Columns: 366 entries, LNR to ALTERSKATEGORIE_GROB
dtypes: float64(267), int64(93), object(6)
memory usage: 2.4+ GB
```

As can be seen from the image above, the data is fairly large with about 2.5 Gigabytes of storage. There are 366 columns, out of which, 267 are floats, 93 are integers, and 6 are strings (but could be a mixture of features as well).

In order to proceed to the Unsupervised Learning part, we need to clean that data first. The first step of that is to identify missing values

Identifying Missing Values

For this task, we need to read the codes that correspond to missing values for each column. These codes are found in the DIAS Attributes - Values 2017.xls file. The following is a sample of the rows from that file:



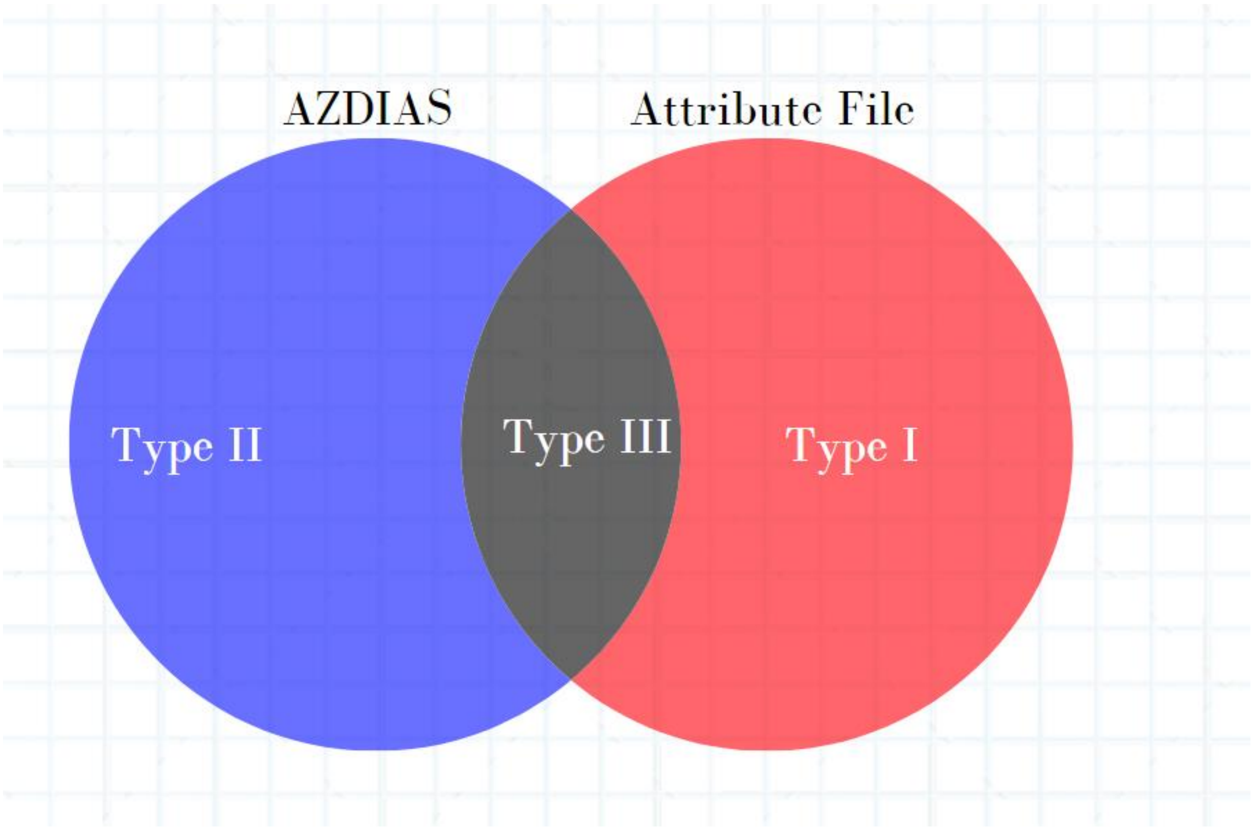
	Unnamed: 0	Attribute	Description	Value	Meaning
0	NaN	AGER_TYP	best-ager typology	-1	unknown
5	NaN	ALTERSKATEGORIE_GROB	age classification through prename analysis	-1, 0	unknown
11	NaN	ALTER_HH	main age within the household	0	unknown / no main age detectable
33	NaN	ANREDE_KZ	gender	-1, 0	unknown
40	NaN	BALLRAUM	distance to next urban centre	-1	unknown

It turns out that the following phrases under the **Meaning** column, where sufficient to indicate the entries that describe missing values:

- unknown
- unknown / no main age detectable
- no transaction
- no transactions known

Out of the attributes mentioned in this file, there are 289 different attributes that needed to be addressed in terms of identifying missing values. And there are 25 columns that don't have missing values indicators.

However, upon further inspection, it turns out that there are differences between the columns specified in the Attributes sheet and the columns found in the AZDIAS dataframe. For example, there are 42 columns that we can identify nans for but are not available in the AZDIAS dataframe. On the other hand, there are 93 columns which we don't have information to identify nans with, since they are not mentioned in the Attribute sheet but available in the AZDIAS dataframe. Fortunately, there are 273 columns that overlap and we can identify nans for. In case this is confusing, here is a Venn Diagram to illustrate what we found.



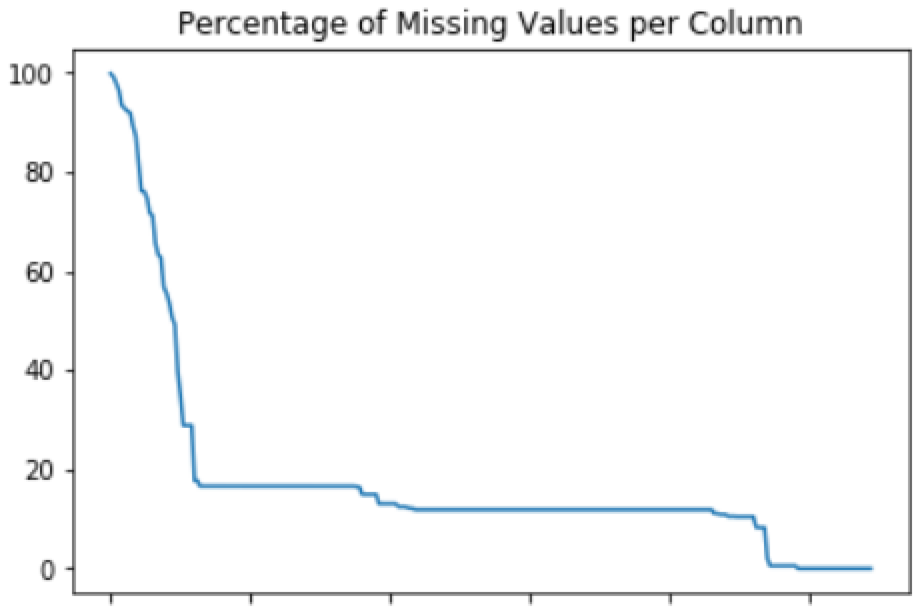
We have 42 columns that are Type I.(irrelevant since we don't have data on it), 93 columns of Type II (most like are going to be dropped as discussed below), and, 273 columns that are of Type III(what we are looking for).

On the good side, the majority of the columns can be processed easily, since they are overlapping between the 2 sheets. However, there is a large number of columns that are remaining. Those remaining columns are going to be addressed individually if there turned out to be a need to include them for further analysis. However, most likely, these columns are going to be dropped, because cleanliness of the data that is produced at this step is of top priority. Another reason is that learning models don't perform well (or not at all) with missing values.

The next step is to decide what to do with these missing values

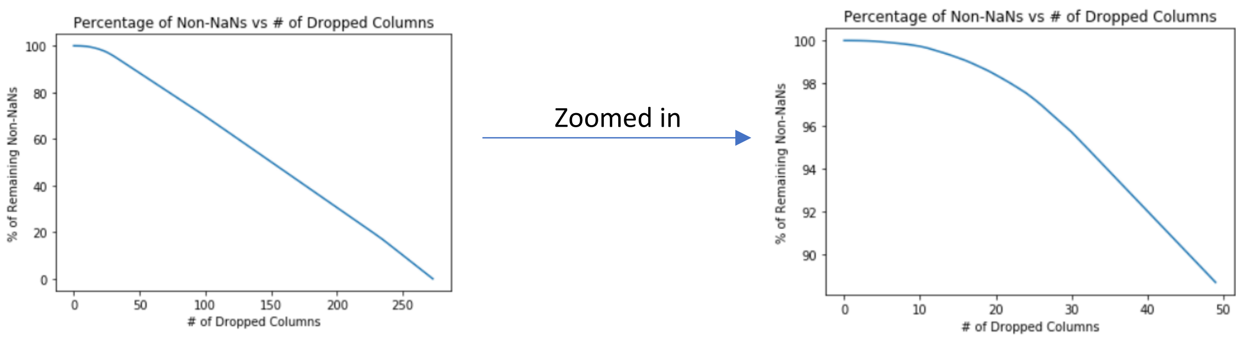
Handling Missing Values

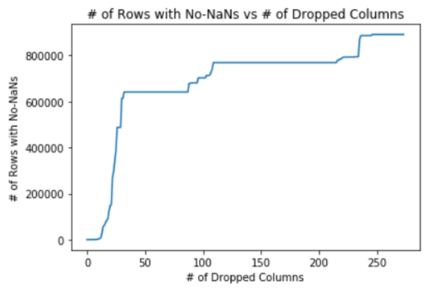
Upon plotting the percentages for missing values in each column, we get this graph:



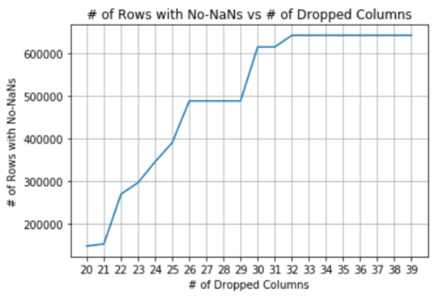
There are several columns with over 90% of missing values, while most columns are at about 20% of missing values. These columns with the highest percentages of missing values were dropped. In order to avoid over-dropping columns unnecessarily, some objective metric needed to be used.

The first metric was the percentage of non-nan values after dropping columns relative to the percentage before dropping them. The second metric was the number of rows that don't contain any missing values after dropping columns.





Zoomed in



It turns out that our magic number is 30. (Dropping the top most 30 columns with nans)

All that remains is to clean/pre-process the columns based on their types. There are 4 basic types:

- Numeric: such as integers, floats, ranges, etc.
- Ordinal data: Although, they could be treated as categorical or at least with some further processing, for the sake of simplicity, we are going to use them as is. It is not an unusual practice to treat ordinals as numeric values
- Categorical: this type is similar to ordinal, but doesn't have the property of following some kind of a trend/sequence. Those are going to be treated individually and would be converted into on-hot-encoded format using dummies
- Mixed: These contain multiple dimensions combined in one. This is, also, going to be treated individually and would be separtated into multiple columns accordingly.

That being said, the categorical and mixed columns are going to be very time consuming to process. So, the following analysis would try to do as many of them as possible within the allowed timeframe (without going overboard).

The first step is to determine, for each column, which type it belongs too. This will be done manually by reading the Attribute file. The main task is to determine the mixed type as well as the categorical type. The rest will be dealt with as numeric/ordinal, which we are going to leave untouched as mentioned above.

The following summarize the types of columns:

```
Type
Categorical     22
Mixed            3
Numeric          7
Ordinal        207
dtype: int64
```

Handling Different Types of Attributes

String Columns

There are 3 columns of type string, namely:

- CAMEO_DEU_2015
- CAMEO_DEUG_2015

- OST_WEST_KZ

All 3 would be considered as categorical and were converted into dummies (1-hot-encoded).

Categorical Columns

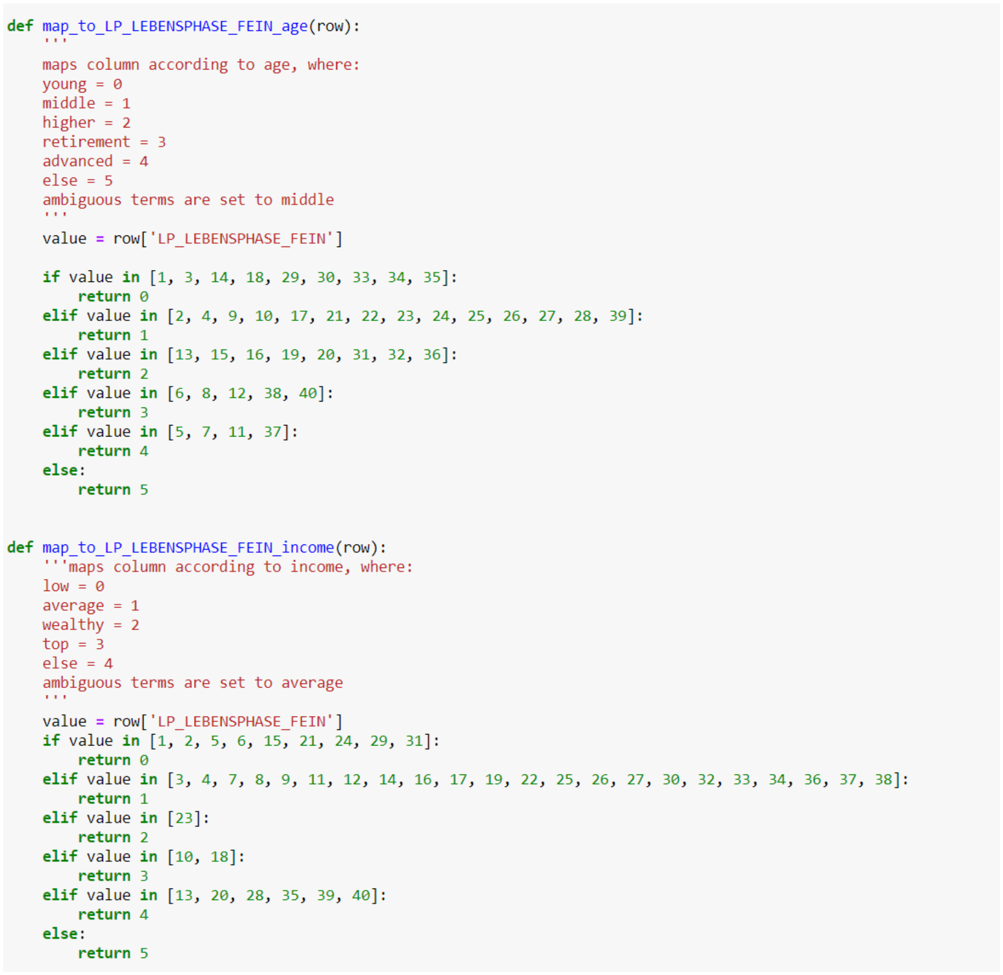
Since there are a lot of columns (22 to be specific) under this category, I am not going to list them. However, suffice it to say that they are all going to be converted into dummies, just like the string columns.

Mixed Columns

There are 3 columns to consider here:

- LP_LEBENSPHASE_FEIN
- PRAEGENDE_JUGENDJAHRE
- LP_LEBENSPHASE_GROB

As, LP_LEBENSPHASE_GROB is a coarser scale of LP_LEBENSPHASE_FEIN, it was ignored (dropped without conversion), in order to avoid confusion. The following was used to split LP_LEBENSPHASE_FEIN into 2 different dimensions: age & income



```python
def map_to_LP_LEBENSPHASE_FEIN_age(row):
    '''
    maps column according to age, where:
    young = 0
    middle = 1
    higher = 2
    retirement = 3
    advanced = 4
    else = 5
    ambiguous terms are set to middle
    '''
    value = row['LP_LEBENSPHASE_FEIN']

    if value in [1, 3, 14, 18, 29, 30, 33, 34, 35]:
        return 0
    elif value in [2, 4, 9, 10, 17, 21, 22, 23, 24, 25, 26, 27, 28, 39]:
        return 1
    elif value in [13, 15, 16, 19, 20, 31, 32, 36]:
        return 2
    elif value in [6, 8, 12, 38, 40]:
        return 3
    elif value in [5, 7, 11, 37]:
        return 4
    else:
        return 5


def map_to_LP_LEBENSPHASE_FEIN_income(row):
    '''maps column according to income, where:
    low = 0
    average = 1
    wealthy = 2
    top = 3
    else = 4
    ambiguous terms are set to average
    '''
    value = row['LP_LEBENSPHASE_FEIN']
    if value in [1, 2, 5, 6, 15, 21, 24, 29, 31]:
        return 0
    elif value in [3, 4, 7, 8, 9, 11, 12, 14, 16, 17, 19, 22, 25, 26, 27, 30, 32, 33, 34, 36, 37, 38]:
        return 1
    elif value in [23]:
        return 2
    elif value in [10, 18]:
        return 3
    elif value in [13, 20, 28, 35, 39, 40]:
        return 4
    else:
        return 5
```

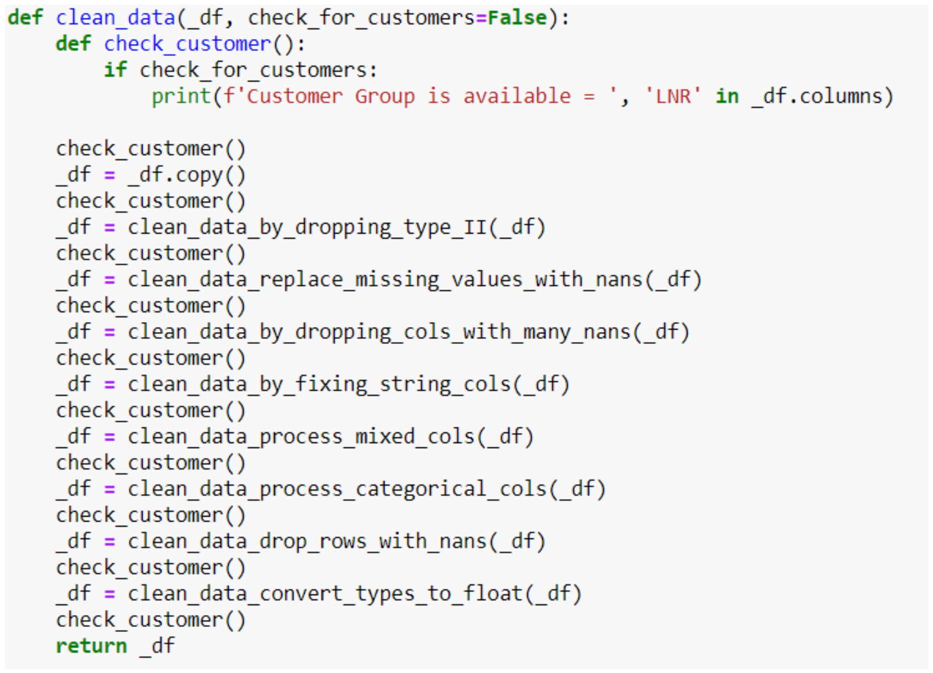
The following was used to convert the PRAEGENDE_JUGENDJAHRE column

```
def map_to_generation(row):
    value = row['PRAEGENDE_JUGENDJAHRE']
    if value == 1 or value == 2:
        return 40
    elif value == 3 or value == 4:
        return 50
    elif value == 5 or value == 6 or value == 7:
        return 60
    elif value == 8 or value == 9:
        return 70
    elif 10 <= value <= 13:
        return 80
    elif value >= 14:
        return 90
    else:
        return value


def map_to_movement(row):
    value = row['PRAEGENDE_JUGENDJAHRE']
    if value in [1, 3, 5, 8, 10, 12, 14]:
        return 0
    elif value in [2, 4, 6, 7, 9, 11, 13, 15]:
        return 1
    else:
        return value
```

Final Touches Before Moving to Next Part

The last thing that was needed to be done were removing the rows that contained missing values, then converting all values into float.

The following summarizes all steps done to clean the data



```python
def clean_data(_df, check_for_customers=False):
    def check_customer():
        if check_for_customers:
            print(f'Customer Group is available = ', 'LNR' in _df.columns)

    check_customer()
    _df = _df.copy()
    check_customer()
    _df = clean_data_by_dropping_type_II(_df)
    check_customer()
    _df = clean_data_replace_missing_values_with_nans(_df)
    check_customer()
    _df = clean_data_by_dropping_cols_with_many_nans(_df)
    check_customer()
    _df = clean_data_by_fixing_string_cols(_df)
    check_customer()
    _df = clean_data_process_mixed_cols(_df)
    check_customer()
    _df = clean_data_process_categorical_cols(_df)
    check_customer()
    _df = clean_data_drop_rows_with_nans(_df)
    check_customer()
    _df = clean_data_convert_types_to_float(_df)
    check_customer()
    return _df
```

Part 1: Customer Segmentation (Unsupervised Learning)

This part was about utilizing the data cleaning pipeline from the previous part, then applying basic feature transformations before, finally, applying clustering techniques.

Feature Transformation

In order to do the segmentation (clustering), we need to do some feature transformation first. That will be followed by a dimensionality reduction process. Then, the data will be ready for clustering. To be specific, the feature transformation has already started when we dropped the missing data points. Alternatively, we could have used and Imputer the fill in the missing data points.

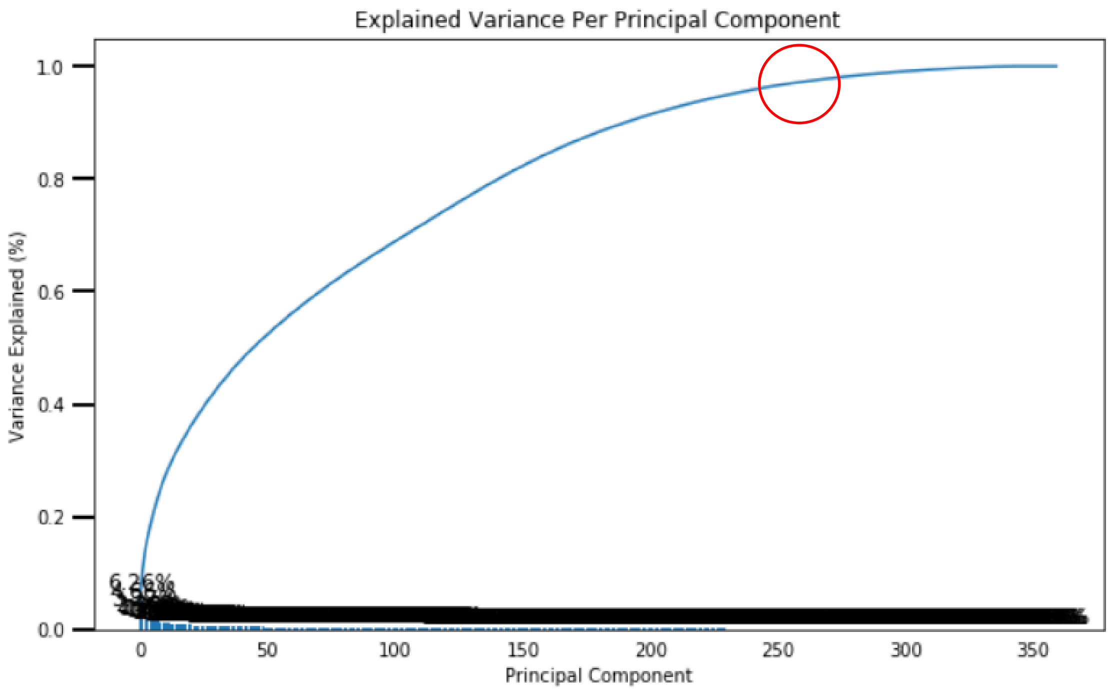
A) Feature Scaling

Feature scaling simply means to scale all columns to have a mean of 0 and a standard deviation of 1. Scaling is done in this phase to make learning easier. This helps accelerate the model training to converge. Scaling is pretty straightforward, and so, no important decisions were required during this step.

B) Dimensionality Reduction

As for the dimensionality reduction phase, the Principal Component Analysis is going to be used. The point of dimensionality reduction is to avoid the curse of dimensionality. It will, also, help in the learning phase.

The most important question to address here is the number of components to reduce the features to. The elbow technique was used by plotting the percentage of explained variance versus the number of principal components, as shown below.



There wasn't a clear elbow here. However, the point at **250 components** looked interesting because it reduced the number of features by about 33% while maintaining over 90% of the explained variance.

Before moving on to the clustering phase, it is worthwhile to investigate some of the meanings behind these components.

C) Interpreting Principal Components

Now that we have our transformed principal components, it's a nice idea to check out the weight of each variable on the first few components to see if they can be interpreted in some fashion.

Each principal component is a unit vector that points in the direction of highest variance (after accounting for the variance captured by earlier principal components). The absolute value of the component represents its weight. While the sign corresponds to its trend (positive components increase together and decrease together. Same goes for negative components.

In order to investigate features, they are going to be sorted by their weights and the associated feature names are going to be displayed. The first 3 components are going to be investigated below. The following functions were used to extract this information from the components.

```
In [70]: weights = pd.DataFrame(pca.components_[:3].T,
                                 index=df.columns,
                                 columns=range(3))
```

```
In [71]: def get_largest_factors(s,head_len=5):
             return s[s.abs().sort_values(ascending=False).index].head(head_len)
```

First Component

The first component represents the following:

- **HH_EINKOMMEN_SCORE** (estimated household net income) high number means low income
- **MOBI_REGIO** (moving patterns) high number means low mobility
- **KBA05_ANTG1** (number of 1-2 family houses in the cell)
- **PLZ8_ANTG1** (Number of 1-2 family houses in the PLZ8 region)
- **PLZ8_ANTG3** (Number of 6-10 family houses in the PLZ8 region)

In other words, this component is related to movement patterns and number of families per home in certain regions. (Some sort of population density distribution)

Second Component

The second component is:

KBA13_HERST_BMW_BENZ (share of BMW & Mercedes Benz within the PLZ8)

EWDICHTE (density of inhabitants per square kilometer)

KBA13_ALTERHALTER_60 (share of car owners between 46 and 60 within the PLZ8)

ORTSGR_KLS9 (size of the community)

KBA13_SEG_OBEREMITTELKLASSE (share of upper middle-class cars and upper class cars (BMW5er, BMW7er etc.)

In summary, this component is somehow related to density of inhabitants in PLZ8 and the share of BMW owners.

Third Component

The third component is:

PRAEGENDE_JUGENDJAHRE_GENERATION (generation)

ALTERSKATEGORIE_GROB (Estimated age based on given name analysis)

FINANZ_SPARER (money saver financial typology)

SEMIO_PFLICHT (affinity indicating in what way the person is dutiful traditional minded) high number corresponds to low affinity

SEMIO_REL (affinity indicating in what way the person is religious) high number corresponds to low affinity

In summary, this component is somehow related to people's age, financial saving typology, and their personal affinities (religion, traditionalism, etc.).

Clustering

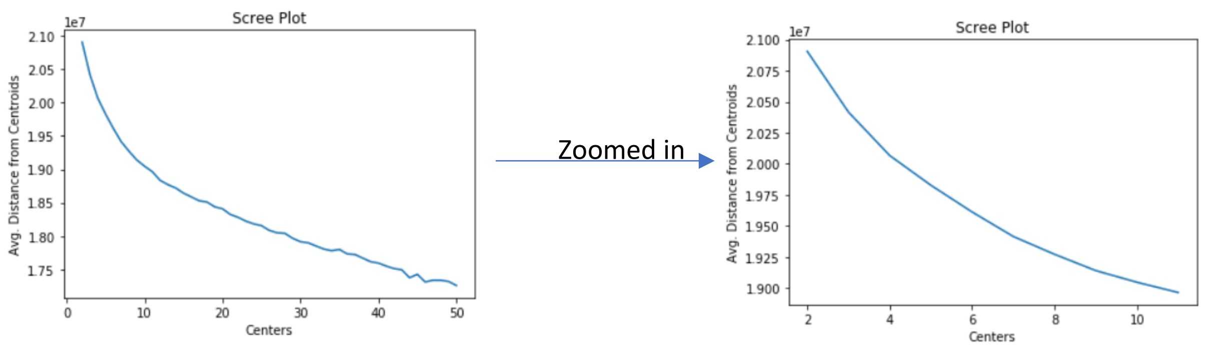
This section is split into 2 main steps. The first step is to apply clustering on the general population, while the second is to compare clustering results between the general population and the customer datapoints.

A) Applying Clustering to General Population

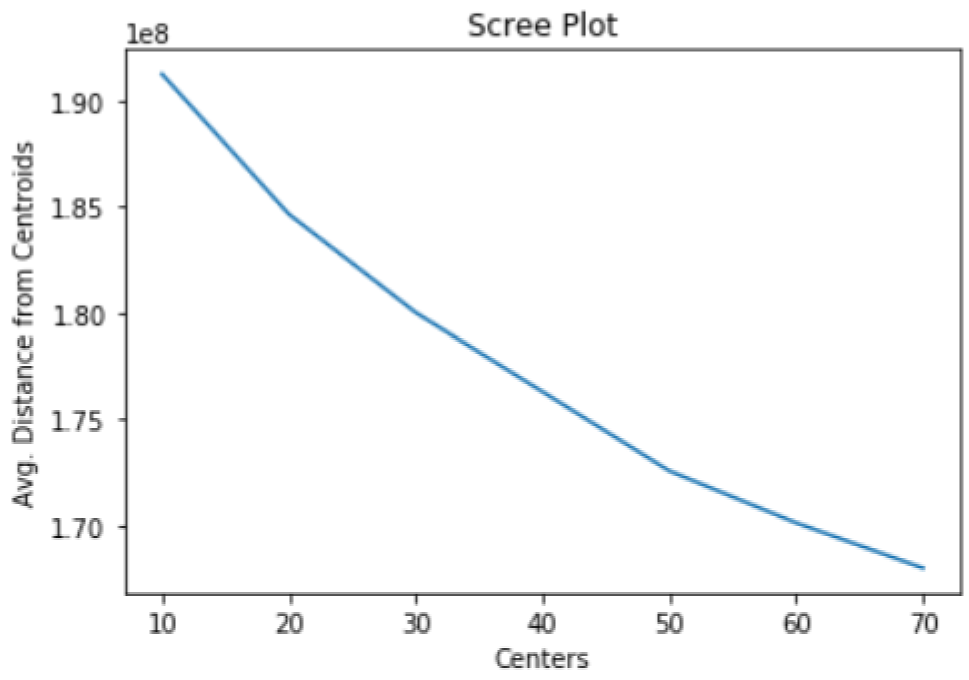
Now, it's finally the time to see how the data clusters in the principal components space. We will apply k-means clustering (KMeans class) to the dataset and use the average within-cluster distances from each point to their assigned cluster's centroid to decide on the number of clusters to keep. The k-means clustering is going to be performed on several values of k (between 10 and 30), and the best suited one is going to be chosen. As will be shown, that the average distance of clusters is going to decrease as we increase k. However, the overall value of clustering is going to decrease as we increase k. Therefore, a balance needs to be observed between these 2 metrics. The elbow method is going to be used here as well.

Since clustering over the whole population takes a very long time to compute, a k-means model was applied on a sample of the whole population over a wide range of k values. Then, another scan was used using the whole population. Finally, pick a value of k based on the points of interest (elbows) from the plots.

This is the plot for the population sample:



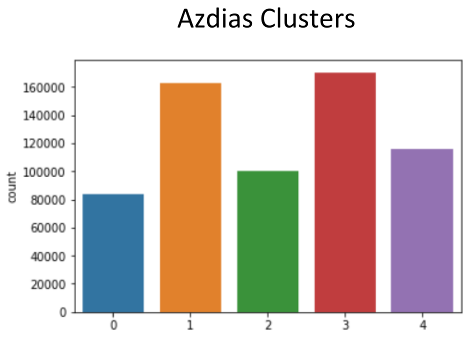
These is the plot for the whole of Azdias data's population:

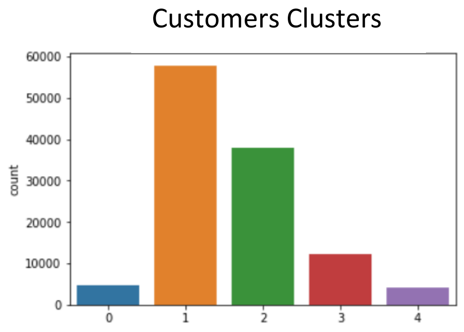


There are only 7 data points in the graph above, so it is relatively coarse. However, it seems that there are potentially 2 elbows (at 20 & 50). However, given the large gaps between the data points and that the elbows are not that sharp, I would stick with the finding from the elbow of the sample population (5 clusters). If a need arises, I might investigate the points from this plot (20 & 50).

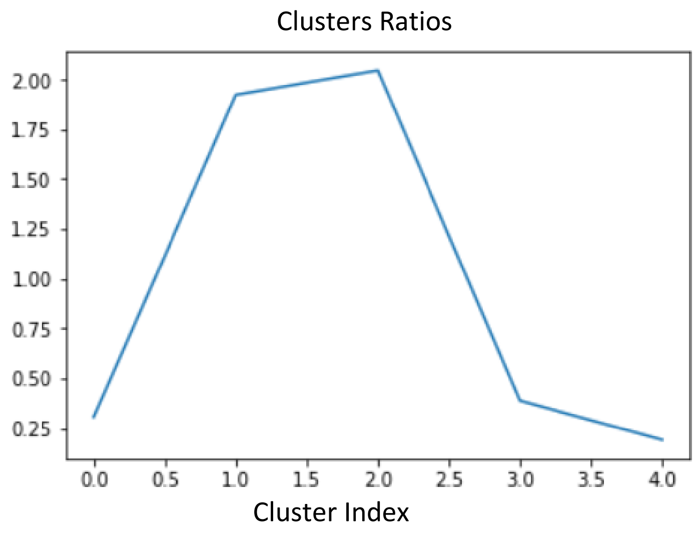
B) Comparing Clusters with Customers

After applying the 5-Means transformation based on the previous result, we can come up with a few conclusions. If we compare the proportions of the clusters from the Azdias dataset with the clusters from the Customers dataset, we would be able to tell which market segment(s) the customers are or aren't. By looking at the following plots, we see that the customers are concentrated around clusters 1 & 2 (over-represented) while the clusters 0, 3, & 4 are quite the opposite (under-represented)





To mathematically quantify which of these clusters is over-represented versus under-represented, the ratio between the clusters was taken. If the ratio for a given cluster is above 1, then it is over-represented. If it is less than 1, then that cluster is under-represented. The following plot shows these ratios:



Therefore, targeting clusters 1 and 2 with ads is going to help in increasing the overall efficiency of the marketing process. Another idea would be to, if an increase in the market segment is a goal, is to figure out if a new product (or way of marketing) is going to appeal to the clusters 0,3, and 4.

In order to do so, we need to understand what top columns each of these clusters are referring to in order to get a sense of what that cluster represents.

The top 5 columns of each cluster is mentioned below along with a brief description that summarizes all 5 columns: (For more information, please refer back to the attributes and information files)

Cluster 0

OST_WEST_KZ_1.0
KBA13_SITZE_5
KBA13_SITZE_4
KBA13_HERST_BMW_BENZ

KBA13_MERCEDES
This cluster seems to address those from **west** Germany who own small (5 or less seats) cars (especially BMW and Mercedes)

Cluster 1

FINANZ_MINIMALIST

MOBI_REGIO

KBA05_ANTG1

PLZ8_ANTG3

PLZ8_ANTG1

Looks like this cluster is representing poor and small families who have a relatively low moving pattern

Cluster 2

KBA13_HERST_BMW_BENZ

KBA13_SEG_SPORTWAGEN

KBA13_KMH_211

KBA13_KW_121

KBA05_KW3

Looks like this cluster is about people who own fast cars with powerful engines (especially BMW and Benz) in the PLZ8 area.

Cluster 3

MOBI_REGIO
PLZ8_ANTG3
KBA05_GBZ
PLZ8_ANTG1
KBA05_AUTOQUOT
This cluster represents people with large sized houses who are less likely to move out and have a relatively low number of cars per household.

Cluster 4

FINANZ_UNAUFFAELLIGER
PRAEGENDE_JUGENDJAHRE_GENERATION
FINANZ_SPARER

SEMIO_PFLICHT
FINANZ_VORSORGER
This cluster describes relatively old people with traditional mindset who are either money savers or at least prepared financially.

Supervised Learning Model

In this part, a small dataset is provided that contains some information from a previous mailout campaign as well as whether the customers responded or not to the mail. The task is to provide a model that can predict whether a given customer would respond to the mail or not. This part is pretty straightforward since it mainly depends on the cleaning process of the data. Unfortunately, the prediction results were not good with the current cleaning methodology, with an average of 50% AUC for the ROC. It was obvious that the available columns were not enough to predict the mailout response column.

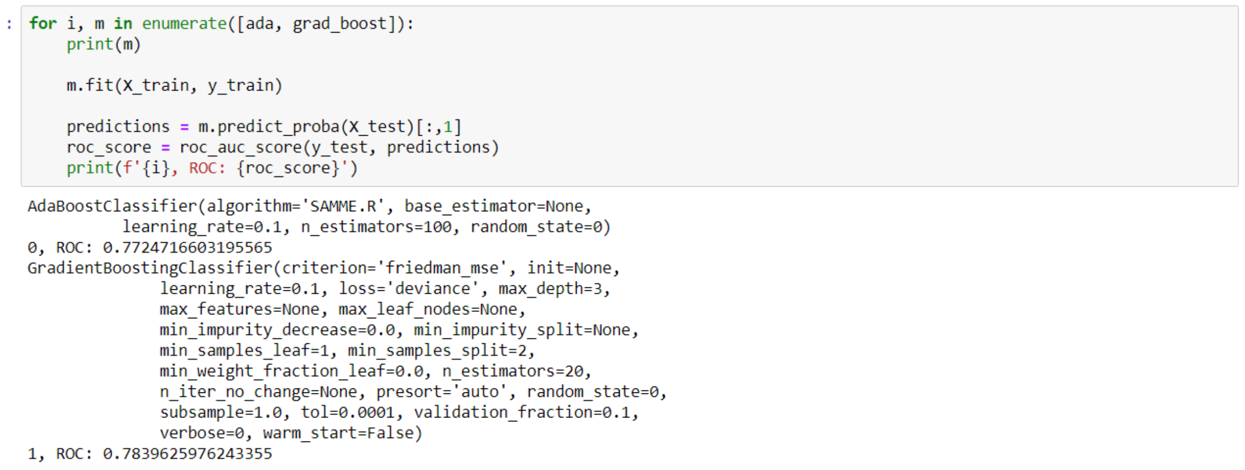
Upon further investigation, it turned out that the type II columns that were deleted in the cleaning process due to a lack of description (and hence lack of information about missing values) were essential to the prediction model's success. Another adjustment to the cleaning process was imputing the missing data based on the most frequently found entries in the Azdias data instead of just deleting rows with missing values. After making these adjustments, the prediction results were significantly better with a best score of 78%.

In order to determine the prediction model to be used, a quick initial run was done using the following models with default parameters. The following are the models along with their ROC results:

- Gradient Boosting Classifier
- AdaBoost Classifier
- Logistic Regression
- Decision Tree Classifier
- XGBoost Classifier

Model	Score
Gradient Boosting Classifier	0.77
AdaBoost Classifier	0.71
Logistic Regression	0.62
Decision Tree Classifier	0.50
XGBoost Classifier	0.66

Based on the initial results, a finer grid search was performed on the most promising models which were the Gradient Boosting Classifier and the AdaBoost. The grid search found even a better version of each of these 2 models which performed at 78% and 77% respectively. So, the Gradient Boosting Classifier was selected to be used in the Kaggle competition.

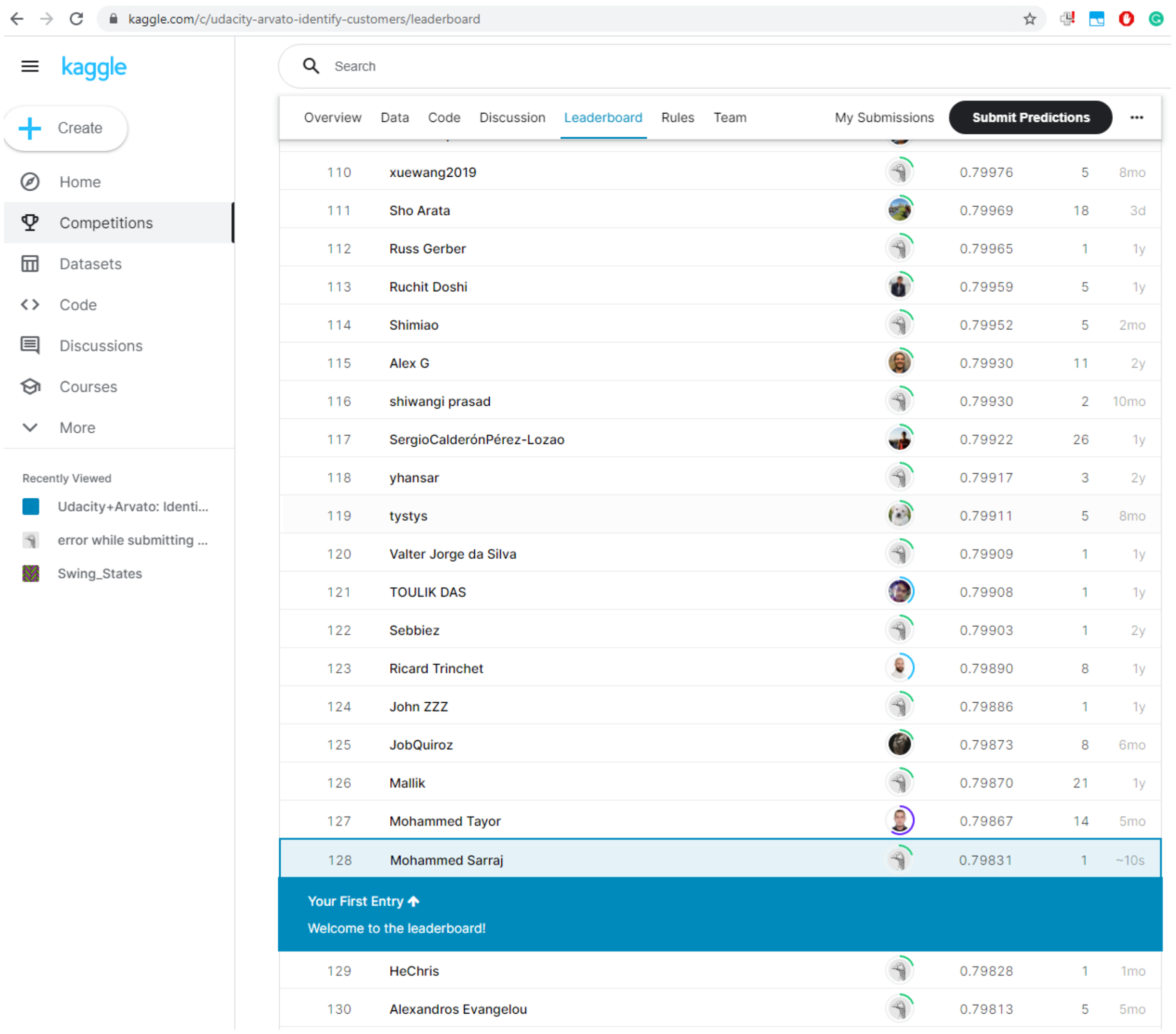


```
: for i, m in enumerate([ada, grad_boost]):
      print(m)

      m.fit(X_train, y_train)

      predictions = m.predict_proba(X_test)[:,1]
      roc_score = roc_auc_score(y_test, predictions)
      print(f'{i}, ROC: {roc_score}')
```

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=0.1, n_estimators=100, random_state=0)
0, ROC: 0.7724716603195565
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=20,
              n_iter_no_change=None, presort='auto', random_state=0,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
1, ROC: 0.7839625976243355
```

Kaggle Competition

For this part, a similar dataset to the one that was provided in the previous part was given here except that it was missing the customer response column. The task was to use the model from the previous part in order to predict the response column for this dataset. As mentioned above, the Gradient Boosting Classifier was used here and the result was, as shown below, 0.798 (ROC).



110	xuewang2019	0.79976	5	8mo
111	Sho Arata	0.79969	18	3d
112	Russ Gerber	0.79965	1	1y
113	Ruchit Doshi	0.79959	5	1y
114	Shimiao	0.79952	5	2mo
115	Alex G	0.79930	11	2y
116	shiwangi prasad	0.79930	2	10mo
117	SergioCalderónPérez-Lozao	0.79922	26	1y
118	yhansar	0.79917	3	2y
119	tystys	0.79911	5	8mo
120	Valter Jorge da Silva	0.79909	1	1y
121	TOULIK DAS	0.79908	1	1y
122	Sebbiez	0.79903	1	2y
123	Ricard Trinchet	0.79890	8	1y
124	John ZZZ	0.79886	1	1y
125	JobQuiroz	0.79873	8	6mo
126	Mallik	0.79870	21	1y
127	Mohammed Tayor	0.79867	14	5mo
128	Mohammed Sarraj	0.79831	1	~10s
129	HeChris	0.79828	1	1mo
130	Alexandros Evangelou	0.79813	5	5mo

Your First Entry ↑
Welcome to the leaderboard!

Given that our goal was to get at least 70% ROC, I believe that our objective is clearly met.