

Capstone Project: Create a Customer Segmentation Report for Arvato Financial Services

by Daniel Lee

Introduction

Arvato offers financial solutions offers a wide range of services from Id & Fraud Management to payment financial services. The company is looking to use its available dataset to analyze demographics data for customers of a mail-order sales company in Germany, comparing it against demographics information for the general population and predict which individuals are most likely to convert into becoming customers for the company.

The data represents a real-life data science task which includes the general population of Germany, Demographics data for customers of a mail-order company, Demographics data for individuals who were targets of a marketing campaign with response column, and Demographics data for individuals who were targets of a marketing campaign with no response column

Unsupervised and supervised learning techniques are used to analyze demographics data of customers to characterize customers segment of population, and to build a model that will be able to predict customers for Arvato Financial Solutions. We approach this project in 2 major steps:

1. Use Unsupervised Learning to perform customer segmentation and identify clusters/segments from general population who best match mail-order company's customer base.
2. Use Supervised Learning to identify targets for marketing campaign of the mail-order company who could possibly become their customers.

Problem Statement

The problem statement for this project is "How can company identify which individuals are most likely to convert into becoming customers for the company?"

We will use an unsupervised learning methods to analyze attributes of established customers and the general population in order to create customer segments, and then follow-up on the discovered customer segments with a supervised learning approach using a dataset with demographics information for the target customers and predict which individuals would be more likely to convert to company customers. The project is divided into four main parts:

1. Data Preprocessing

We have to do preprocess data for further analysis. Missing values, encoding features, dropping unnecessary rows and columns and transformation are used for this part.

2. Customer segmentation

In this part, we will use unsupervised learning techniques to perform customer segmentation and identify clusters/segments from general population who best match mail-order company's customer base. Principal component analysis (PCA) technique is used to reduce higher dimension data before applying KMeans algorithm which make segmentation from general population and

determine the description of target cluster for the company. I will use elbow method to choose a “good” k, which means data points in a single cluster are close together but there are enough clusters to effectively separate data.

3. Supervised learning model

In this part, we build machine learning model using Demographics data for individuals who were targets of a marketing campaign with response column to predict who could possibly become company customers. I have tried various machine learning algorithm and tuning hyperparameter to get optimum results.

4. Kaggle Competition

Once we’ve chosen a model, we’ll use it to make predictions on campaign data as part of Kaggle competition.

Metrics

In unsupervised learning part, explained variance ratio can be used when we are implementing PCA, as it accounts for the description of feature variance, allowing for the determination of more important features that stand out with more explained variance.

For the prediction part of the project (supervised learning), due to the nature of data that have high imbalance class, ROC curve can be used as a metric which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings. TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate.

Exploratory Data Analysis - Exploratory Visualization

There are four data files associated with this project:

- AZDIAS.csv: Demographics data for the general population of Germany
- CUSTOMERS.csv: Demographics data for customers of a mail-order company
- MAILOUT_TRAIN.csv: Demographics data for individuals who were targets of a marketing campaign
- MAILOUT_TEST.csv: Demographics data for individuals who were targets of a marketing campaign

And 2 metadata files associated with these datasets:

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

We’ll start our analysis with azdias and customers dataset which are two primary dataset for our project. DIAS Attributes is also analyzed to understand the dataset features. First have a quick look through both dataset:

azdias

```
azdias.shape
```

```
(891221, 366)
```

```
azdias.head()
```

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

5 rows × 366 columns

Customers

```
customers.shape
```

```
(191652, 369)
```

```
customers.head()
```

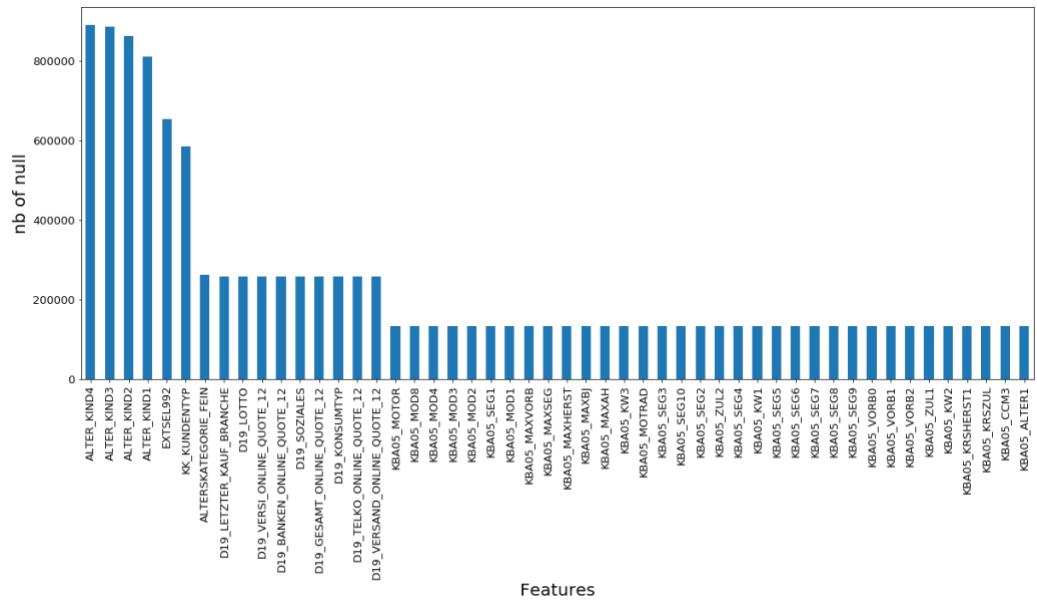
	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKAT
0	9626	2	1.0	10.0	NaN	NaN	NaN	NaN	10.0
1	9628	-1	9.0	11.0	NaN	NaN	NaN	NaN	NaN
2	143872	-1	1.0	6.0	NaN	NaN	NaN	NaN	0.0
3	143873	1	1.0	8.0	NaN	NaN	NaN	NaN	8.0
4	143874	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0

5 rows × 369 columns

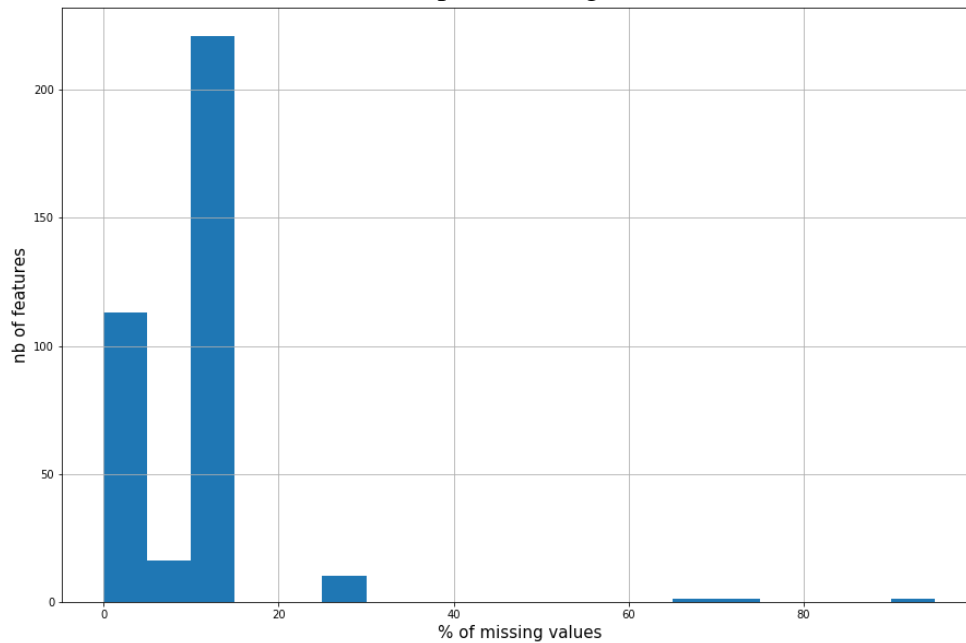
It is noticeable that customers dataset has 3 extra columns which are ('CUSTOMER_GROUP', 'ONLINE_PURCHASE', and 'PRODUCT_GROUP'. We'll drop these 3 columns that make features identical to azdias dataset before further analysis.

Analysis of columns and rows with missing values

Dealing with missing values should be one of the first step in data exploration which is important as many machine learning algorithms do not support data with missing values. Good handling missing data can affect significantly on ML model.



Features
Top 50 missing data

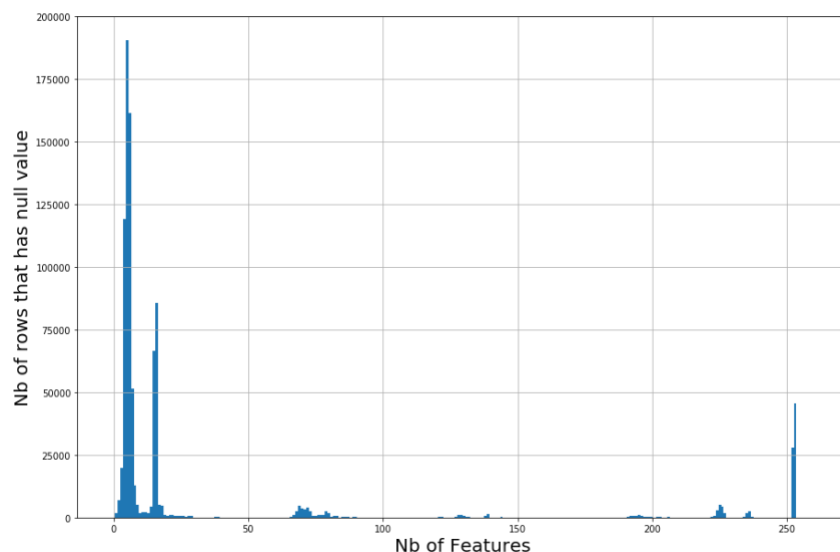


```
#We can see that % of null data in columns ranges from 0 to 15%
azdias_null_percentage.hist(bins = np.arange(0,100,5), figsize= (15,10))
plt.xlabel('% of missing values', fontsize = 15)
plt.ylabel('nb of features', fontsize = 15)
for i in range(100):
    if i%5 == 0:
        print('columns having missing values >{}%: {} features'.format(i,len(azdias_null_percentage[azdias_null_percentage>i])))

columns having missing values >0%: 273 features
columns having missing values >5%: 253 features
columns having missing values >10%: 237 features
columns having missing values >15%: 16 features
columns having missing values >20%: 16 features
columns having missing values >25%: 16 features
columns having missing values >30%: 6 features
columns having missing values >35%: 6 features
columns having missing values >40%: 6 features
columns having missing values >45%: 6 features
columns having missing values >50%: 6 features
columns having missing values >55%: 6 features
columns having missing values >60%: 6 features
columns having missing values >65%: 6 features
columns having missing values >70%: 5 features
columns having missing values >75%: 4 features
columns having missing values >80%: 4 features
columns having missing values >85%: 4 features
columns having missing values >90%: 4 features
columns having missing values >95%: 3 features
```

By looking missing values from many perspectives, we get better understanding at the data so we can deal with it properly. From above, we can see that about 6 features that missing values over 65% which were dropped in data preprocessing. Additionally, there are other columns that were dropped based on the following reasons:

- Features contain unnecessary items: D19_LEZTER_KAUF_BRANCHE, EINGEFUEFT_AM



For azdias dataset, there is slightly changes in number of rows missing values starting at 17 features. We will keep all the rows that missing less than or equal to 17 features. The rest will be dropped.

Summary:

- 6 features that missing values over 65%
- Features contain many different items
- Features that are non-numerical values needed to be handling properly as it is described in feature description.

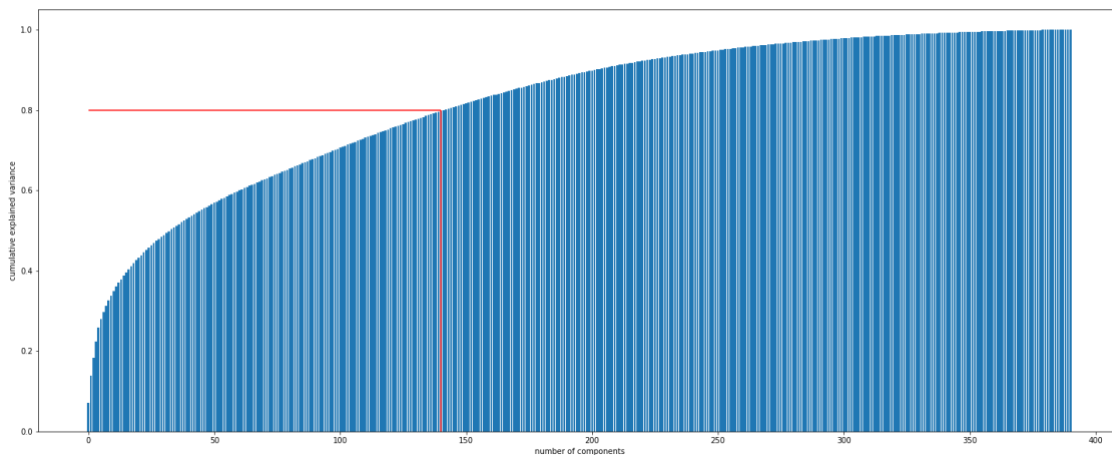
Data preprocessing summary

In summary, all dataset undergo cleaning procedure that:

1. Dropping columns and rows
2. Remove highly correlated features(only for unsupervised learning part)
3. Encode Categorical Features
4. Impute missing NaN values using SimpleImputer from sklearn.impute
5. Transform data using StandartScaler from sklearn.preprocessing

Dimensionality Reduction

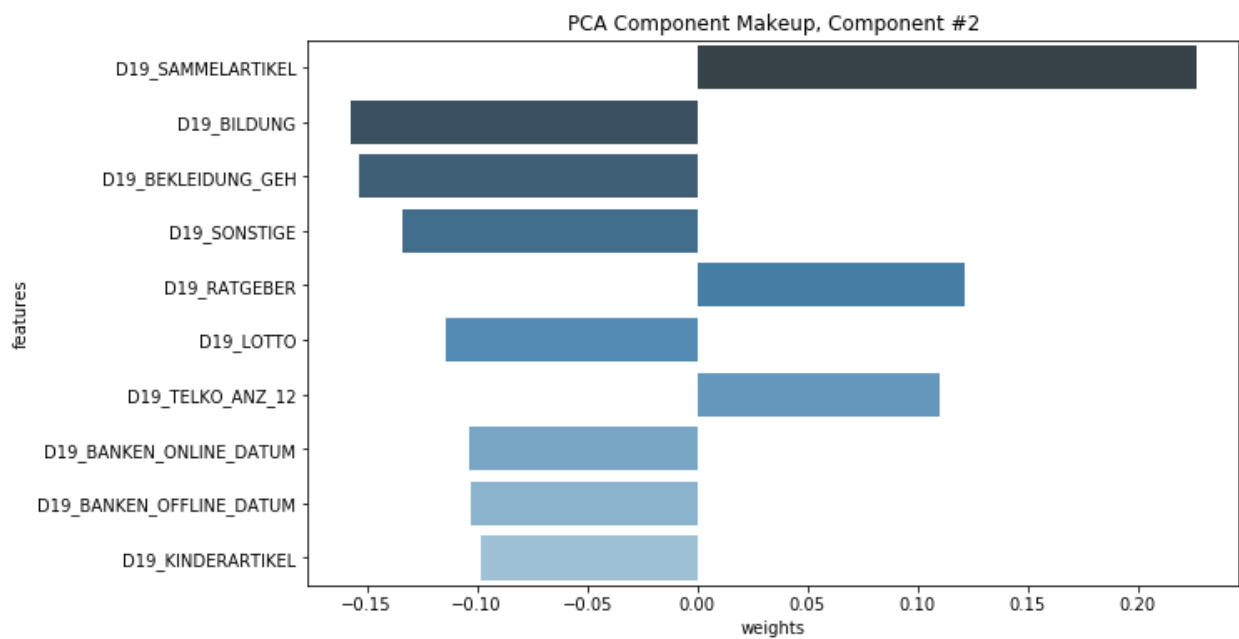
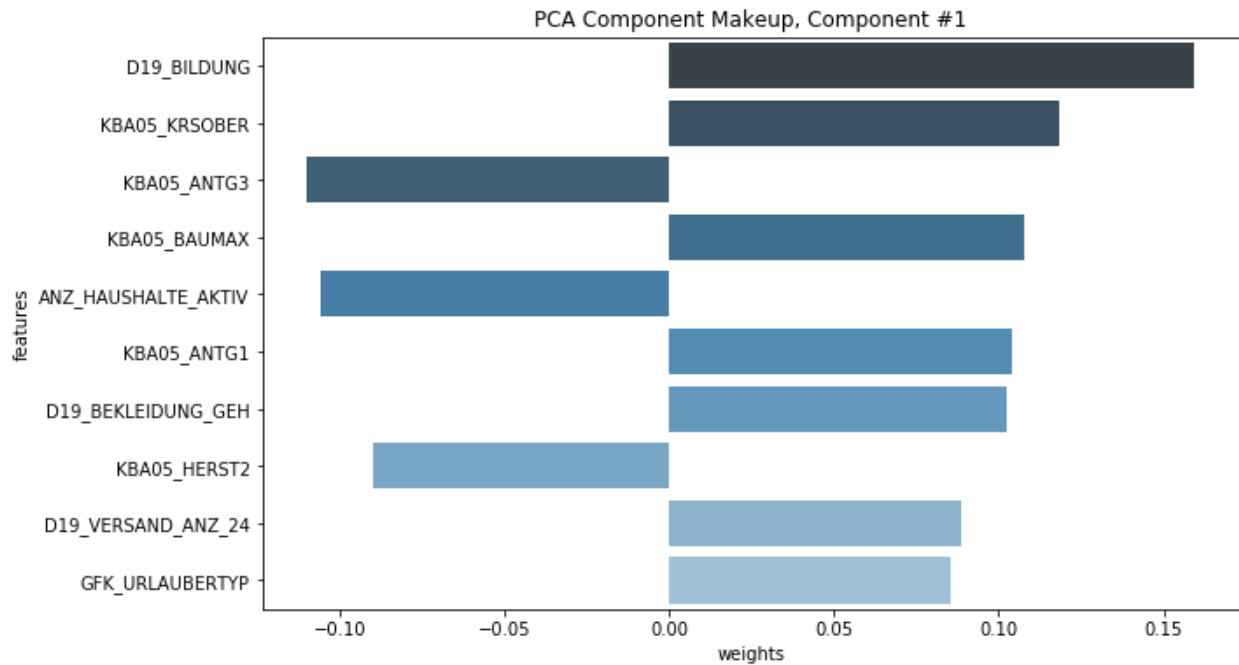
After data preprocessing, we find out that general population data still has high dimension (738283 rows, 391 columns) which is needed to be reduced before applying unsupervised learning method in next steps. Principal Component Analysis (PCA) is commonly used when we have data with many features.

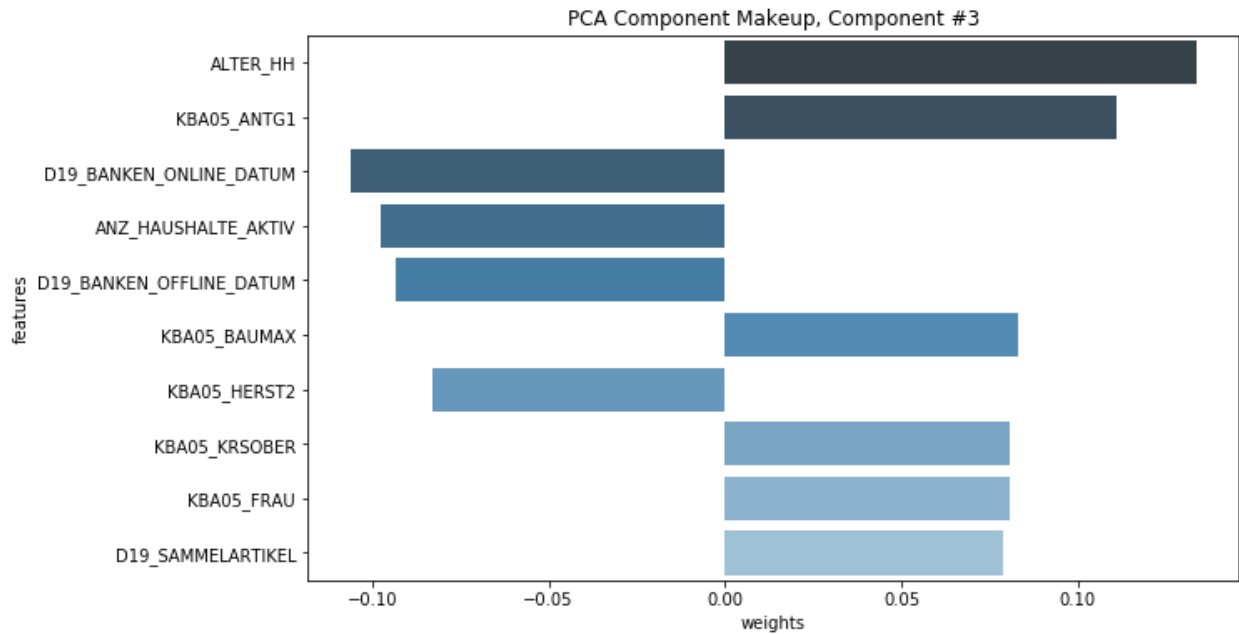


The explained variance tells you how much information (variance) can be contributed to each of the principal components. With PCA we want to make our data has high variance which means cumulative explained variance is high. When we select the top n components to use in a new data model, we'll typically want to include enough components to capture about 80-90% of the original data variance. In this case, we'll choose 140 top components that cumulative explained variance is 80%. The dimension of general population after reducing is 738283 rows and 140 columns

Component makeup

After reducing dimension, it is interesting to see what exactly the makeup of components is, which features define it. We can examine the makeup of each PCA component based on the weighting of the original features that are included in the components.





The first ten component make up 33.8% total variance of the dataset. Each component is a combination of 140 features with each feature given a weight relative to its importance for a particular principal component. The higher the weight either positive or negative, the more impact the feature has on the calculation of the principal component. What value of weight is important is subjective and depends on the context.

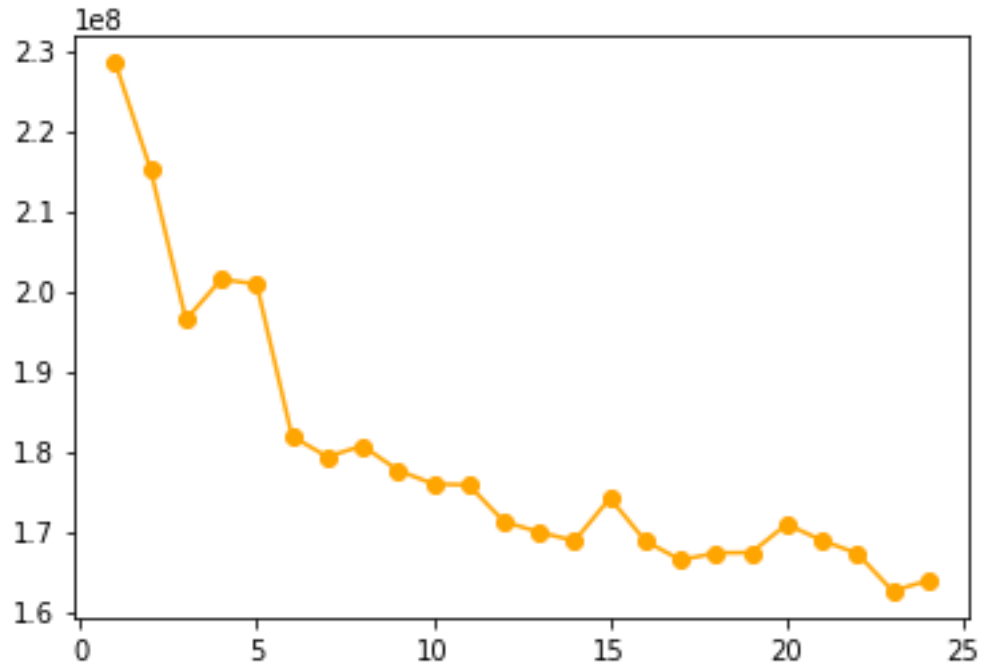
It's impossible to analyze 140 components. I will take one out of 140 to illustrate my point in this section. For component 1 Top 5 positive weights are:

- D19_BILDUNG: transactional activity based on the product group EDUCATION
 - KBA05_HERST4: share of European manufacturer (e.g. Fiat, Peugeot, Rover,...)
 - D19_BEKLEIDUNG_GEH: transactional activity based on the product group LUXURY CLOTHING
 - KBA05_ALTER2: share of car owners in between 31 and 45 years of age
 - KBA05_KRSVAN: share of vans (referred to the county average)
- Top 5 negatives are:
- KBA05_HERST2: share of Volkswagen-Cars (including Audi)
 - ANZ_HAUSHALTE_AKTIV: number of households in the building
 - HH_EINKOMMEN_SCORE: estimated household net income
 - D19_VERSAND_ANZ_24: transaction activity MAIL-ORDER in the last 24 months
 - D19_VERSICHERUNGEN: transactional activity based on the product group INSURANCES

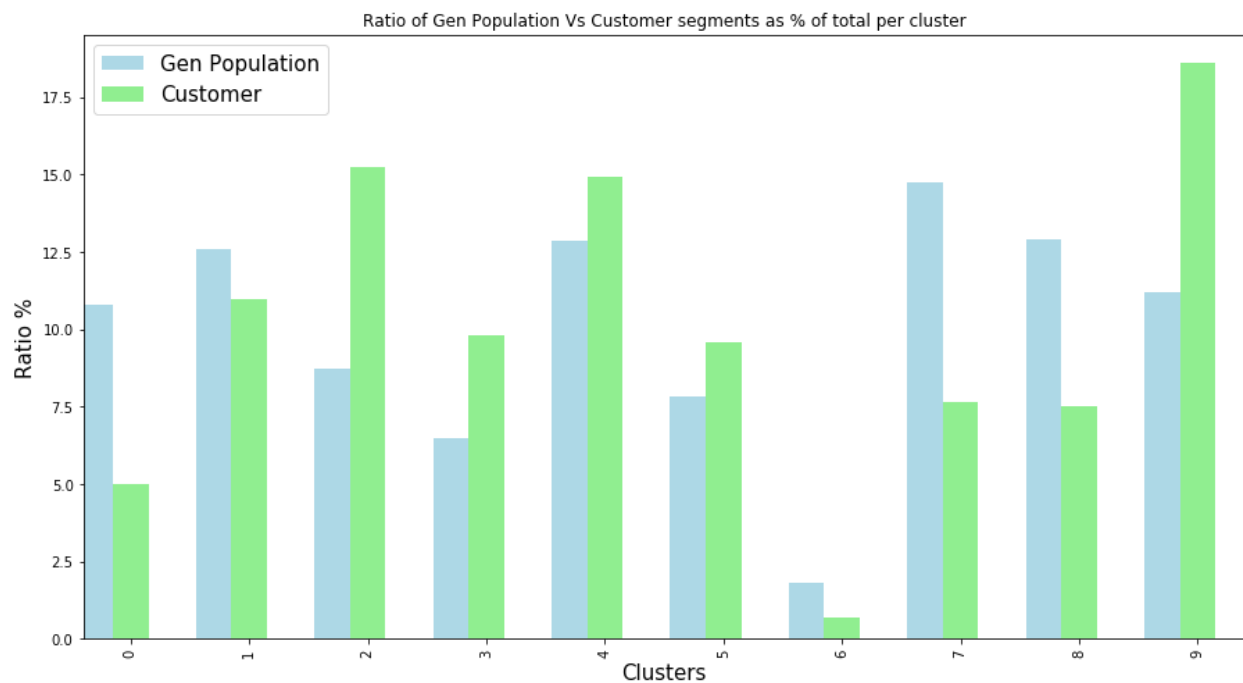
The first component seems to represent a combination of features that relate to car manufacturing and the associated wealth in that area

Clustering

Choosing a “good” number of cluster is important as it help effectively separate the data and data point in a single cluster are close together. Elbow method is commonly used to select a good K.



From above chart we can see that 10 cluster might be a good final number of clusters to construct our model.



Comparing the general and customer populations show that there are clusters that are overrepresented as well as underrepresented by the customer population.

We can observe that the three clusters with the largest amount of overrepresentation (customers) suggest that people in these clusters are more likely to be the target audience for the company. These clusters are:

- cluster 0
- cluster 1
- cluster 5
- cluster 6
- cluster 8

The three clusters that are most underrepresented are:

- cluster 2
- cluster 3
- cluster 4
- cluster 7
- cluster 9

Supervised Learning Model

In this part, we'll use MAILOUT dataset which each of the rows files represents an individual that was targeted for a mailout campaign. Ideally, we should be able to use the demographic information from each individual to decide whether or not it will be worth it to include that person in the campaign.

The "MAILOUT" data has been split into two approximately equal parts, each with almost 43 000 data rows. In this part, you can verify your model with the "TRAIN" partition, which includes a column, "RESPONSE", that states whether or not a person became a customer of the company following the campaign. Before fitting into a model, we need to preprocess data first.

These steps are:

1. Dropping unnecessary features as we find out in EDA section
2. Encode Categorical Features
3. Impute missing values
4. Transform dataset using StandardScaler from sklearn.preprocessing

After cleaning and transform MAILOUT dataset, the dimension of the data is 42962 rows and 400 columns.

```
df_train = clean_data_for_supervised_learning(X,
        unknown_columns_na_fill_0,
        unknown_columns_na_fill_9,
        unknown_columns_na_fill_neg1,
        transactions_columns_na_fill_0,
        transactions_columns_na_fill_10,
        online_transactions_columns_na_fill_0,
        1,
        'StandardScaler')
```

```
***** Step 1 - dropping columns *****
Number of features before dropping: 366
Number of features after dropping: 358

***** Step 2 - Encode Categorical Features *****
number of features before encoding categorical: 358
number of features after encoding categorical:401

***** Step 3 - Impute values *****
check NaN values in dataset: 0

***** Step 4 - Transform Dataset *****
Shape after clean data: (42962, 400)
FINISH CLEAN DATASET
```

The next step is splitting data into training and validation using `train_test_split` from `sklearn`.

```
# Split the dataset into Train/Validation/Test
X_train, X_test, y_train, y_test = train_test_split(df_train, y, test_size=0.2, random_state=42)
```

We'll try numbers of algorithm: `XGBRegressor`, `GradientBoostingRegressor`, `AdaBoostRegressor` and `LogisticRegression` to see which one produce the best results using ROC evaluation metrics. Although the best model score is `GradientBoostingRegressor`, when it comes to tune hyperparameter, the score is dropped significantly. So we will use `XGBRegressor` algorithm for this model.

After finding out which baseline model we are going to use, we do tuning hyperparameter to improve score

Benchmark model

The characteristic of the problem is classification problems (0 and 1), 0 which is a not customer and 1 is a customer of a company. Based on the characteristic of the problem, logistic regression is a good choice to test our data and see how they behave. Unlike linear regression, logistic regression can produce discrete binary outputs.

```
#Algorithm to use in the optimization problem.
#For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones.
benchmark = LogisticRegression(random_state=42,
                               solver= 'saga')
benchmark.fit(X_train,y_train)
predictions = benchmark.predict(X_test)
print(f'model score:{round(roc_auc_score(y_test, predictions), 5)} \n' )|

model score:0.5
```

The model score is quite reasonable for a simple linear model but we can improve model score by using more complex algorithm and see which one produce the best model score. Lets see if we'll use this simple model, how many scores we get on the Kaggle.

The Kaggle score is 0.5, which is an acceptable result for a simple model but which is quite far from the first place (>0.8). We need to choose others algorithm to improve model score.

Model selection

XGBRegressor, GradientBoostingRegressor, AdaBoostRegressor are commonly used for regression problems. It's interesting to see what and how these algorithm works under the hood.

- AdaBoostRegressor works by weighting the observations, putting more weight on difficult to classify instances and less on those already handled well. Predictions are made by majority vote of the weak learner's predictions, weighted by their individual accuracy.
- Gradient boosting involve three elements: a loss function to be optimized, a weak learner to make predictions, an additive model to add weak learners to minimize the loss function
- XGBoost: an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='binary:logistic',
             random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             seed=None, silent=None, subsample=1, verbosity=1)
model score:0.74226
```

```
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,
                          n_iter_no_change=None, presort='auto',
                          random_state=42, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
model score:0.74443
```

```
AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear',
                  n_estimators=50, random_state=42)
model score:0.74113
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning:
Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
FutureWarning)
```


```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                  intercept_scaling=1, l1_ratio=None, max_iter=100,
                  multi_class='warn', n_jobs=None, penalty='l2',
                  random_state=42, solver='warn', tol=0.0001, verbose=0,
                  warm_start=False)
model score:0.5
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             learning_rate=0.1, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
             nthread=None, objective='binary:logistic', random_state=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
model score:0.5
```

Kaggle Submission and Results

We'll use MAILOUT_TEST dataset to make inference and export and submit results to Kaggle. Cleaning and transform dataset is same as using for MAILOUT.

Our model score was able to reach >80% on the test set.

13	Dlee152		0.80570	9	24m
----	---------	--	---------	---	-----

Compared to benchmark model (using logistic regression), XGBRegressor produce higher score which is above 0.8 Although XGBRegressor is not always guaranteed to be better than a logistic regression in every setting, in this case it produce competitive results. These significal results is come from:

- Wide variety of tuning parameters : XGBoost internally has parameters for cross-validation, regularization, user-defined objective functions, missing values, tree parameters, scikit-learn compatible API.
- Consistently outperforms other algorithm methods : It has shown better performance on a variety of machine learning benchmark datasets.

XGBoost's hyperparameters

- learning_rate: step size shrinkage used to prevent overfitting. Range is [0,1]
- max_depth: determines how deeply each tree is allowed to grow during any boosting round.
- subsample: percentage of samples used per tree. Low value can lead to underfitting.
- colsample_bytree: percentage of features used per tree. High value can lead to overfitting.
- n_estimators: number of trees you want to build.
- objective: determines the loss function to be used like reg:linear for regression problems, reg:logistic for classification problems with only decision, binary:logistic for classification problems with probability.

XGBoost also supports regularization parameters to penalize models as they become more complex and reduce them to simple (parsimonious) models.

- gamma: controls whether a given node will split based on the expected reduction in loss after the split. A higher value leads to fewer splits. Supported only for tree-based learners.

Conclusion

From a real-life data science provided by Arvato Financials, we have been able to analyze demographics data, create segmentation of customers comparing it against demographics information for the general population and predict which individuals are most likely to convert into becoming customers for the company.

As this was my first time I have worked real life big dataset, it has been great learning experience on how to approach the problem in a methodical approach. The most challenging is preprocess data, data cleaning without losing too much key information.

For things that I could have been done differently/better, I think there are few areas like understanding features more to do feature engineering better, keep trying tuning model with few different hyperparameter to get better result. Doing all these things, I might be able to improve prediction performance further.

Finally, I would like to thank Udacity and Arvato Financials for creating a competitive environment to work on real life data science task which helps me have a better understanding how a real task can be solved.