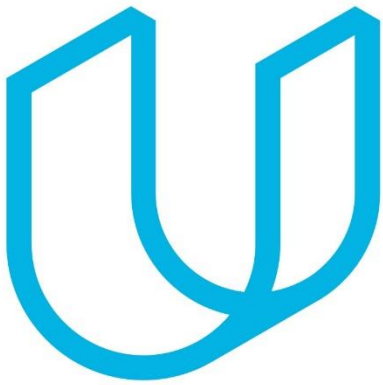


Capstone Report

UDACITY

Customer Segmentation & Optimizing the Customer Acquisition Process with Arvato Financial Solutions

Author: Hassan Essa Alghanim

Date of Report: 1st January 2022

Udacity - Machine Learning Engineer Nanodegree

Customer Segmentation – Arvato Financial Solutions

I. Definition

Project Overview

In this project, our objectives are two-fold.

First, I analyze the demographics data of the current customers of a mail-order German company, as well as the demographics data of the general population of the country (a sample of the entire German population) and compare those two populations/datasets to find similarities and discrepancies between the two.

Second, and most importantly, our end-goal is to become sufficiently confident in predicting, based on given demographics data, who (from the general population) would be highly probable to become a new customer of the mail-order company: these individuals would then be targeted by the mail-order company through their marketing campaign.
This, in particular, would optimize their marketing action and, ultimately, their customer acquisition process, through informed and sensible customer campaign.

This project is included as a requirement for the completion of the 'Machine Learning Engineer' Nanodegree at Udacity (School of AI), as the Capstone of the programme and the synthesis of my learning. It was made possible by the combined efforts of Udacity and Arvato Financial Services, who kindly gave us (restricted) access to their data – all datasets used in this project are their sole private property (refer to Terms & Conditions) and are inclusively used for the development of this project alone (single-use only).

Problem Statement

The problem I will be working on in this project is the following:

"How can the German mail-order company acquire new customers more efficiently, given the access to German demographics data?"

Arvato Financial Services wants, in a first phase, to enable their client (the mail-order company) to gain insight into their current customer base and compare their demographics data to a sample of the general German population (compare their attributes, using clustering). In a second phase, the goal is to confidently predict who is more likely to become a customer of the mail-order company, so that this company can more efficiently (and in a smarter way) target the 'right' type of clients in their marketing campaign (using classification).

The problem can be quantified in the following terms: number of current/established customer and general population clusters (customer segmentation unsupervised problem) and probability of being a new customer to the company (supervised problem).

Datasets

The project makes use of four datasets:

- **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 mailorder 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).

II. Analysis

Data Exploration

Exploring the AZDIAS and CUSTOMERS datasets is an important step of the project development, enabling us to gain insight into the data (features/columns present, value ranges…). Each row of both demographics datasets refers to one single person.

The MAILOUT train and test datasets are similar to the datasets above; train and test datasets have the same feature/columns, with the test dataset having one column less than the train dataset (dropped RESPONSE feature, as this is the probability of becoming a new customer we will predict in the final Part of the project – Part 3).

In data exploration, looking at the datasets at hand along with the 2 .xlsx informative files is also very helpful, in particular to better understand the meaning between features/columns, the range of unknown and missing values (they will need to be considered as NaN values), the range of values that the column can take up…

To load in the datasets from csv files to Pandas DataFrames:

```
# load in the data
azdias = pd.read_csv('Udacity_AZDIAS_052018.csv', sep=';')
customers = pd.read_csv('Udacity_CUSTOMERS_052018.csv', sep=';')
```

```
mailout_train = pd.read_csv('Udacity_MAILOUT_052018_TRAIN.csv', sep=';')
```

```
mailout_test = pd.read_csv('Udacity_MAILOUT_052018_TEST.csv', sep=';')
```

- AZDIAS dataset (original)

The AZDIAS dataset (Udacity_AZDIAS_052018.csv) contains the data of 891 211 persons (rows) and 366 features (columns): its shape, therefore, is (891 211, 366).

This dataset represents the demographics data for the **general** population of Germany.

Below, we can get a first look into the dataset by calling the head() method to `azdias` Pandas DataFrame.

```
azdias.head()
```

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

5 rows × 366 columns

- CUSTOMERS dataset (original)

The CUSTOMERS dataset (Udacity_CUSTOMERS_052018.csv) contains the data of 191 652 persons (rows) and 369 features (columns): its shape, therefore, is (191 652, 369).

In comparison to the AZDIAS dataset, the CUSTOMERS dataset contains 3 additional columns, which are: 'CUSTOMER_GROUP', 'ONLINE_PURCHASE', and 'PRODUCT_GROUP'.

This dataset represents the demographics data for the **customers** of a mail-order company of Germany: this mail-order company is the client of Arvato Financial Services, requesting their services to better target individuals in their campaign (e.g. marketing strategy targeted to individuals most likely to become customers, when you look at their demographics data).

Below, we can get a first look into the dataset by calling the *head*() method to *customers* Pandas DataFrame.

```
customers.head()
```

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

5 rows × 369 columns

Data Observation

Going a little deeper, we can now investigate the peculiarities of the DataFrames, e.g. with *azdias*. Similar observations hold true for the other DataFrames, so we will here take the *azdias* one to describe our data observation step; later on, similar data preprocessing steps will be done on all 4 DataFrames.

- Features not described

We previously mentioned the existence of 2 .xlsx files with information regarding the features/columns of the datasets. However, early on, we notice that some features in *azdias* are not described in these files: it is wise to consider these columns either not to bear much value to our data or not to be appropriate to be kept in our analysis.

This observation is made possible thanks to the manual creation of a .csv file (‘*attributes.csv’*), created from the 2 .xlsx. files, that contains 3 columns: ‘Attribute’, ‘Type’, and ‘Unknown_Values’.

```
attributes_info = pd.read_csv('attributes.csv', sep=';')
```

```
attributes_info.head()
```

	Attribute	Type	Unknown_Values
0	AGER_TYP	categorical	[-1,0]
1	ALTERSKATEGORIE_FEIN	ordinal	[-1,0.9]
2	ALTERSKATEGORIE_GROB	ordinal	[-1,0,9]
3	ALTER_HH	ordinal	[0]
4	ANREDE_KZ	categorical	[-1,0]

From that *attributes_info* DataFrame, we can create a *features_in_attributes_info* list, containing all the features/columns in *attributes_info*: these features/columns are the DESCRIBED columns in the 2 .xlsx files.

```
features_in_attributes_info = attributes_info['Attribute'].tolist()
```

```
features_in_attributes_info
```

```
['AGER_TYP',
 'ALTERSKATEGORIE_FEIN',
 'ALTERSKATEGORIE_GROB',
 'ALTER_HH',
 'ANREDE_KZ',
 'ANZ_HAUSHALTE_AKTIV',
 'ANZ_HH_TITEL',
 'ANZ_KINDER',
 'ANZ_PERSONEN',
 'ANZ_STATISTISCHE_HAUSHALTE',
 'ANZ_TITEL',
 'ARBEIT',
 'BALLRAUM',
 'CAMEO_DEUG_2015',
 'CAMEO_DEU_2015',
 'CAMEO_INTL_2015',
```

With a simple subtraction, we get to the non-described columns (i.e. columns in *azdias*, but not in the *features_in_attributes_info* list):

```
azdias_columns_not_described = list(set(azdias) - set(features_in_attributes_info))
```

```
# columns in AZDIAS dataset but not described in attributes_info: to drop
azdias_columns_not_described
```

```
['UMFELD_ALT',
 'ALTER_KIND2',
 'CJT_TYP_5',
 'CJT_TYP_6',
 'AKT_DAT_KL',
 'ALTER_KIND4',
 'D19_LETZTER_KAUF_BRANCHE',
 'STRUKTURTYP',
 'D19_KONSUMTYP_MAX',
 'CJT_TYP_2',
 'VK_DHT4A',
 'CJT_KATALOGNUTZER',
 'RT_UEBERGROESSE',
 'UMFELD_JUNG',
 'LNR',
 'EXTSEL992',
 'FIRMENDICHTE',
 'VHN',
 'EINGEFUEGT_AM',
 'GEMEINDETYP',
 'CJT_TYP_4',
 'CJT_TYP_3',
 'RT_KEIN_ANREIZ',
 'VHA',
 'CJT_TYP_1',
 'VK_ZG11',
 'KOMBIALTER',
 'VK_DISTANZ',
 'VERDICHTUNGSRAUM',
 'RT_SCHNAEPPCHEN',
 'ALTER_KIND3',
 'ALTER_KIND1']
```

- Unknown and missing values

The ‘Unknown Values’ column of *attributes_info* contains the ‘unknown values’ for each described feature/column (thus, including the columns of azdias and customers); we create a Pandas Series holding these values (*unknown_series*).

```
unknown_series = pd.Series(attributes_info['Unknown_Values'].values, index=attributes_info['Attribute'])
```

```
unknown_series
```

```
Attribute
AGER_TYP                  [-1,0]
ALTERSKATEGORIE_FEIN    [-1,0.9]
ALTERSKATEGORIE_GROB    [-1,0,9]
ALTER_HH                     [0]
ANREDE_KZ                 [-1,0]
                          ...
VERS_TYP                    [-1]
WOHNDAUER_2008            [-1,0]
WOHNLAGE                  [-1,0]
W_KEIT_KIND_HH            [-1,0]
ZABEOTYP                  [-1,9]
Length: 334, dtype: object
```

Some columns also contain missing values, denoted by ‘X’ and ‘XX’ for example.

Preprocessing of these unknown and missing values will be needed prior to employing machine learning algorithms in the next parts.

- NaN values in columns

Prior to this observation, we have converting unknown and missing values to NaN values (details of this preprocessing step are available in III. Methodology/Data Preprocessing).

Now, we can investigate the quantity of NaN values per column: it will be good practice to drop columns with high percentages/proportions of NaN values: this decision will be made on the basis of a set threshold, above which we will drop columns.

We can identify the proportion (out of 1) of NaN values for each column of the *azdias* DataFrame:

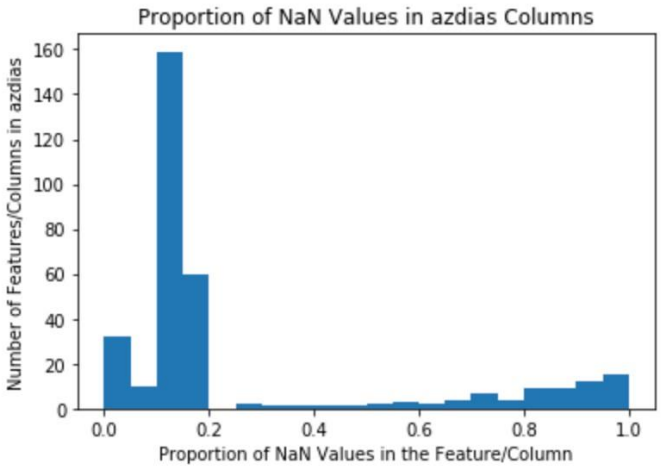
```
missing_perct_column = azdias.isnull().mean(axis=0)
```

```
missing_perct_column
```

```
AGER_TYP                0.769554
ALTER_HH                0.348137
ALTERSKATEGORIE_FEIN    0.295041
ANZ_HAUSHALTE_AKTIV     0.104517
ANZ_HH_TITEL            0.108848
                          ...
WOHNDAUER_2008          0.082470
WOHNLAGE                0.112316
ZABEOTYP                0.000000
ANREDE_KZ               0.000000
ALTERSKATEGORIE_GROB    0.003233
Length: 334, dtype: float64
```

```
plt.hist(missing_perct_column, bins=20);

plt.xlabel('Proportion of NaN Values in the Feature/Column')
plt.ylabel('Number of Features/Columns in azdias')
plt.title('Proportion of NaN Values in azdias Columns')
```

```
Text(0.5, 1.0, 'Proportion of NaN Values in azdias Columns')
```



- NaN values in rows

In a similar fashion, we wish to investigate the proportion of NaN values per row, so that it will become possible to drop rows surpassing a defined threshold (see Data Preprocessing section for more details).

We can identify the proportion (out of 1) of NaN values for each row of the *azdias* DataFrame:

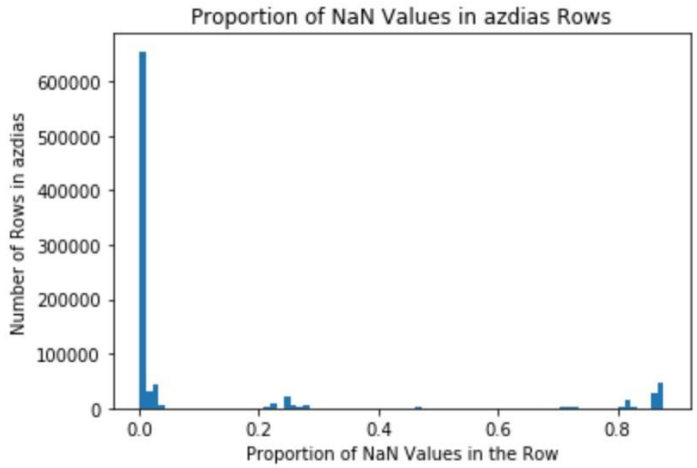
```
missing_perct_row = azdias.isnull().mean(axis=1)
```

```
missing_perct_row
```

```
0          0.862069
1          0.000000
2          0.000000
3          0.030651
4          0.000000
             ...
891216     0.011494
891217     0.015326
891218     0.022989
891219     0.000000
891220     0.000000
Length: 891221, dtype: float64
```

```
plt.hist(missing_perct_row, bins=80);

plt.xlabel('Proportion of NaN Values in the Row')
plt.ylabel('Number of Rows in azdias')
plt.title('Proportion of NaN Values in azdias Rows')
```

```
Text(0.5, 1.0, 'Proportion of NaN Values in azdias Rows')
```



- Categorical Features

Having a look at the columns of azdias, and at the *attributes_info* DataFrame we created, we can pinpoint columns whose type is categorical.

For subsequent machine learning algorithms, we will need to turn these features into numerical features, with strategies of re-encoding (from cat to num) depending on the ‘characteristic of the categorical feature (more details in the Data Preprocessing section)

```
# categorical features in azdias: we will encode these categorical features
cat_feat_azdias
```

```
['ANREDE_KZ',
 'CAMEO_DEUG_2015',
 'CAMEO_DEU_2015',
 'CJT_GESAMTTYP',
 'DSL_FLAG',
 'FINANZTYP',
 'GEBAEUDETYP',
 'GFK_URLAUBERTYP',
 'GREEN_AVANTGARDE',
 'HH_DELTA_FLAG',
 'KONSUMZELLE',
 'LP_FAMILIE_FEIN',
 'LP_FAMILIE_GROB',
 'LP_STATUS_FEIN',
 'LP_STATUS_GROB',
 'NATIONALITAET_KZ',
 'OST_WEST_KZ',
 'SHOPPER_TYP',
 'SOHO_KZ',
 'UNGLEICHENN_FLAG',
 'VERS_TYP',
 'ZABEOTYP']
```

- Mixed Features

Similarly, the azdias DataFrame possesses mixed features, which will also need to be taken care of (details in the Data Preprocessing section).

```
# mixed features in azdias: we will encode these mixed features
mixed_feat_azdias
```

```
['CAMEO_INTL_2015',
 'LP_LEBENSPHASE_FEIN',
 'LP_LEBENSPHASE_GROB',
 'PRAEGENDE_JUGENDJAHRE']
```

Algorithms and Techniques: Presentation

The precise implementation of these algorithms will be detailed in the Implementation section of this Report; here, we only introduce the algorithms and techniques used in the Project Notebook.

Introducing the algorithms and techniques used in this Project Notebook:

- PCA

In Part 1, we wish, using unsupervised machine learning algorithms, to create customer segments and compare demographics data of the general population (*azdias*) and of the customer base (*customers*), using clustering.

Prior to employing clustering with K-Means, it is best practice for the data to undergo dimensionality reducing, for which we used PCA, or Principal Component Analysis.

> In the first step of Part 1 (unsupervised learning), we will employ Principal Component Analysis, with the goal to reduce the number of features within a dataset while retaining the "principal components", defined as weighted, linear combinations of existing features.
>
> These principal components are meant to be linearly independent (solving the problem of highly correlated features) and account for the largest possible variability in the data. We will soon decide on the amount of variability we think is sufficient to then reliably employ for our (next) clustering task with k-Means clustering.

To get a refresher on PCA and better understand it, have a look at this insightful resource *[4]*.

- K-means

Once PCA is done on *azdias* and *customers* and both DataFrames have reduced dimensions (with more *meaningful* features, or principal components), we will use k-means clustering to actually create the clusters for customer segmentation, based on demographics data.

More generally, clustering refers to a bigger set of techniques employed to partition data into clusters (i.e. meaningful and useful groups). Those clusters are meaningful and useful, in that they are groups of data objects, which objects are more similar to other objects in their cluster than to others in other clusters.

Check out this resource *[5]*, to get insight into clustering in general and k-means clustering in python, step-by-step.

There are 3 types of clustering (partitional, hierarchical, density-based); k-means is one algorithm used for partitional clustering.

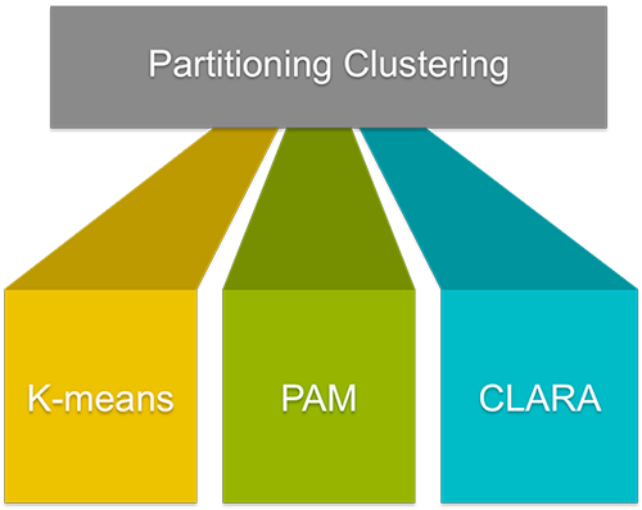


Image Source: Datanovia [6]

- Some Supervised Learning Algorithms

In Part 2 of this project, we used supervised machine learning tools to build a prediction classifier model, with the goal of predicting, based on new demographics data (MAILOUT data), the probability for each individual to become a new customer of the mail-order company.

We explored various supervised learning options, those being:

- RandomForestClassifer
- GradientBoostingClassifier
- C-Support Vector Classification (SVC) **
- AdaBoostClassifier

The main resources I have used in deepening my understanding of these algorithms are the following:

[7] and [8] for RandomForestClassifier

[9] and [10] for GradientBoostingClassifier

[11] for SVC

[12] and [13] for AdaBoostClassifier

****Warning**: a spelling error has been made in the Project Notebook file, where SVC has been incorrectly named 'Support Vector Machine' (SVM).

Benchmark

- Benchmark Model for Part 2: Logistic Regressor

As mentioned in the Project Proposal, we chose a basic Logistic Regression Model as a benchmark model, against which we will compare the final trained model in Part 2.

The main resources used in implanting this regressor were: [14] and [15].

III. Methodology

Data Preprocessing

In this section, we will guide the reader through the data preprocessing steps undertaken and informed by observations from the data.

We document here the preprocessing steps progressively done on the *azdias* DataFrame; refer to the project notebook to check out the cleaning function put together to then process, in a similar fashion, customers, and the train and test sets of the mailout DataFrame.*

*Note that we slightly tweaked the cleaning function (used for *azdias* and *customers*) to preprocess the train and test sets of mailout.

- Dropping features not described

As explained in the Data Observation section, some features of the DataFrames are not described, i.e. no information can be found in the 2 xlsx files given to us at the beginning of the project (and thus not found in the manually created *attributes.csv* file and *attributes_info* DataFrame).

We decided to drop such columns, whose added values cannot reliably drawn without any background information on their meaning and range of values taken.

```
azdias.drop(labels=azdias_columns_not_described, axis=1, inplace=True)
```

```
azdias.shape
```

```
(891221, 334)
```

> We went from 360 columns/features in azdias to 334 columns/features.

- Converting unknown values to NaN values

To deal with unknown and missing values, we convert them to NaN values.

First, we create a Pandas Series, unknown_series, from attributes_info DataFrame.

```
unknown_series = pd.Series(attributes_info['Unknown_Values'].values, index=attributes_info['Attribute'])
```

```
unknown_series
```

```
Attribute
AGER_TYP                  [-1,0]
ALTERSKATEGORIE_FEIN    [-1,0.9]
ALTERSKATEGORIE_GROB    [-1,0,9]
ALTER_HH                     [0]
ANREDE_KZ                 [-1,0]
                          ...
VERS_TYP                    [-1]
WOHNDAUER_2008            [-1,0]
WOHNLAGE                  [-1,0]
W_KEIT_KIND_HH            [-1,0]
ZABEOTYP                  [-1,9]
Length: 334, dtype: object
```

Then we map the unknown and missing values from *unknown_series* to NaN values in appropriate columns of *azdias*.

```
for column in azdias.columns:

    isin = ast.literal_eval(unknown_series[column])

    azdias[column] = azdias[column].mask(azdias[column].isin(isin), other=np.nan)
```

```
azdias.head()
```

	AGER_TYP	ALTER_HH	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV	ANZ_HH_TITEL	ANZ_KI
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	21.0	11.0	0.0	0.0
2	NaN	17.0	17.0	10.0	0.0	0.0
3	2.0	13.0	13.0	1.0	0.0	0.0
4	NaN	20.0	14.0	3.0	0.0	0.0

5 rows × 334 columns

- Dropping column-wise

Once we have NaN values, we wish to drop columns with 'too much' of these values that would impact the machine learning algorithms' performance later on.

In our case, we chose to set 0.2 as a threshold: columns with a NaN column-wise proportion greater than 0.2 will be dropped out of the DataFrame.

```
columns_to_drop = missing_perct_column[missing_perct_column > 0.2].index
azdias.drop(labels=columns_to_drop, axis=1, inplace=True)
```

```
# checking AZDIAS dataset shape
azdias.shape
```

```
(891221, 261)
```

> We went from 334 columns/features in azdias (from 2.) to 261 columns/features after 3.a)

- Dropping row-wise

With the same way of thinking as above, we decide to drop rows in the DataFrame with NaN row-wise proportion greater than 0.2.

```
azdias = azdias[missing_perct_row < 0.2]
```

```
azdias.shape
```

```
(737288, 261)
```

> We went from 891221 rows to 737288 rows in azdias after 3.b), with the NaN threshold set at 0.2

- Re-encoding Categorical Features

We previously explained that we must convert categorical features to numerical features in order for subsequent machine learning algorithms to be trained.

We also mentioned that re-encoding categorical features can be done in various ways. Indeed, the 3 ways we went about this re-encoding step are summarised in the code snippet below.

Essentially, we wish to create 3 lists, each containing a subtype of categorical features:
○ Binary categorical features (numeric) ○ Binary categorical features (non-numeric) ○ Multi-Level categorical features (>2 classes)

1. Categorical Features to Numerical Features

-> For binary categorical variables that take numeric values, no change is needed.

-> For binary categorical variables that take on non-numeric values, we need to re-encode the values as numbers (numerical type).

-> For multi-level categorical variables (>2 values), we need to one-hot encode the values.

```
binary_num_attributes = []
binary_non_num_attributes = []
multi_level_attributes = []

for attribute in cat_feat_azdias:
    dtype = azdias[attribute].dtype
    count = len(azdias[attribute].value_counts())

    # if multi-level categorical feature
    if count > 2:
        multi_level_attributes.append(attribute)
    else:
        if dtype == 'object':
            binary_non_num_attributes.append(attribute)
        else:
            binary_num_attributes.append(attribute)
```

No change has to be done for the features in the *binary_num_attributes* list.

The only feature in the *binary_non_num_attributes* list can be re-encoded as seen below:

Encoding binary string categorical attribute/feature: `OST_WEST_KZ`

```
# Re-encode OST_WEST_KZ binary non-numerical feature as dummy variable
# look at DIAS Attributes - Values 2017 (1) file to see what the original values ar
e for OST_EST_KZ
# it is: 'O' for East(GDR) and 'W' for West (FRG)

azdias['OST_WEST_KZ'] = azdias['OST_WEST_KZ'].map({'W': 1, 'O': 2})
```

Encoding multi-level categorical features/columns into dummy variables (one-hot encoding): See [1] reference

```
list_columns_to_add = []

for column in multi_level_attributes:

    # get rid of multi-level categorical columns with more than 10 levels
    if len(azdias[column].value_counts()) < 10:
        list_columns_to_add.append(pd.get_dummies(azdias[column], prefix=column))


# drop the original multi-level categorical features
azdias.drop(multi_level_attributes, axis=1, inplace=True)

list_columns_to_add.append(azdias)

# add the re-encoded multi-level categorical features to azdias dataframe
azdias = pd.concat(list_columns_to_add, axis=1)
```

- Re-encoding Mixed Features

There are 4 mixed features/columns to take care of.

'CAMEO_INTL_2015'

The CAMEO_INTL_2015 feature actually encodes 2 variables, one being the level of wealth ('wealthy', 'prosperous', 'comfortable', 'less affluent', 'poorer') and the other being status ('Pre-Family Couples & Singles', 'Young Couples With Children', 'Families With School Age Children', 'Older Families & Mature Couples', 'Elders In Retirement').

These values are encoded as the following:

--FOR STATUS--

> if the unit digit is 1 => Pre-Family Couples & Singles
>
> if the unit digit is 2 => Young Couples With Children
>
> if the unit digit is 3 => Families With School Age Children
>
> if the unit digit is 4 => Older Families & Mature Couples
>
> if the unit digit is 5 => Elders In Retirement

--FOR WEALTH LEVEL--

> if the ten digit is 1 => wealthy
>
> if the ten digit is 2 => prosperous
>
> if the ten digit is 3 => comfortable
>
> if the ten digit is 4 => less affluent
>
> if the ten digit is 5 => poorer

These patterns enable us to create 2 news features from the original CAMEO_INTL_2015, with the following code cell:

```
azdias['WEALTH_LEVEL'] = azdias['CAMEO_INTL_2015'].apply(lambda x: np.floor(pd.to_n
umeric(x)/10))

azdias['STATUS'] = azdias['CAMEO_INTL_2015'].apply(lambda x: pd.to_numeric(x)%10)
```

Don't forget to drop CAMEO_INTL_2015, that will no longer be used.

```
# drop CAMEO_INTL_2015
azdias.drop('CAMEO_INTL_2015', axis=1, inplace=True)
```

'LP_LEBENSPHASE_FEIN' and 'LP_LEBENSPHASE_GROB'

As these 2 features are quite complex (relative to the rest of features), and given the number of other features at hand, we make the decision to simply drop them.

```
# dropping the features, as decided above
azdias.drop(['LP_LEBENSPHASE_FEIN', 'LP_LEBENSPHASE_GROB'], axis=1, inplace=True)
```

'PRAEGENDE_JUGENDJAHRE'

```
# pj = short for PRAEGENDE_JUGENDJAHRE

def encode_pj(value):

    re_encoded_value = 0

    # re-encoding for Mainstream movement: 0
    if value in [1, 3, 5, 8, 10, 12, 14]:
        re_encoded_value = 0

    # re-encoding for Avantgarde movement: 1
    elif value in [2, 4, 6, 7, 9, 11, 13, 15]:
        re_encoded_value = 1

    return re_encoded_value
```

```
azdias['PRAEGENDE_JUGENDJAHRE'] = azdias['PRAEGENDE_JUGENDJAHRE'].apply(lambda x: e
ncode_pj(x))
```

'PRAEGENDE_JUGENDJAHRE' now can only take on the following values: 0 or 1.

- Imputing Remaning NaN Values (*Interesting Resource: [2]*)

Imputing the NaN Values with the median method

```
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
```

```
azdias_imputed = pd.DataFrame(imputer.fit_transform(azdias))
```

```
azdias_imputed.shape
```

```
(737288, 289)
```

```
customers_imputed = pd.DataFrame(imputer.fit_transform(customers))
```

```
customers_imputed.shape
```

```
(134246, 288)
```

- Feature Scaling (*Interesting Resource: [3]*)

We will now scale the remaining features: this step is critical prior to applying PCA, as principal component vectors would otherwise be influenced by the differences in scale of our data features.

```
scaler = StandardScaler()
```

```
azdias_scaled = pd.DataFrame(scaler.fit_transform(azdias_imputed))
```

```
customers_scaled = pd.DataFrame(scaler.fit_transform(customers_imputed))
```

Implementation

In this section, we will add the implementation details of the machine learning (unsupervised and supervised) algorithms we introduced in an earlier section.

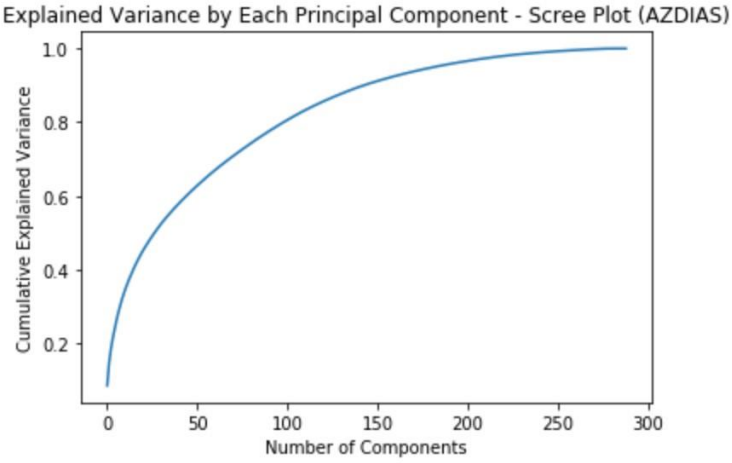
- Dimensionality Reduction with PCA

To decide the number of principal components we want to have (and thus, the final number of features we want to keep in our *azdias* and *customers* DataFrames), we used scree plot visualisations.

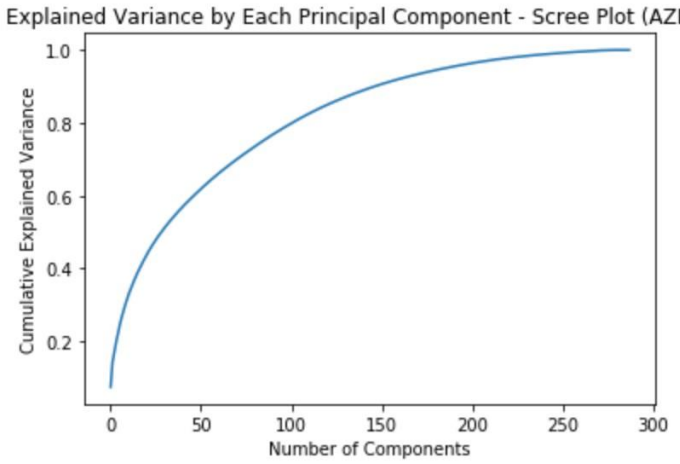
A scree plot displays the variance explained by each principal component. The goal is to only keep the number of principal components that will explain 90% of the variance in the data (a percentage deemed high enough that we don't lose so much information in the data). *[16]*

Employing this visualisation technique on both *azdias* and *customers* DataFrames (scaled), we decide that *150* is a good choice for the number of principal components we want to keep.

```
#Explained variance (AZDIAS)

pca = PCA()
azdias_pca = pca.fit_transform(azdias_scaled)

plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.title('Explained Variance by Each Principal Component - Scree Plot (AZDIAS)')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')

plt.show()
```




```
#Explained variance (CUSTOMERS)

pca = PCA()
customers_pca = pca.fit_transform(customers_scaled)

plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.title('Explained Variance by Each Principal Component - Scree Plot (AZDIAS)')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')

plt.show()
```



We then create a *reduce_dimension()* function to call PCA on *azdias* and *customers* DataFrames, with n_components=150.

```
def reduce_dimension(df, n=150):

    pca = PCA(n_components=n)

    reduced_df = pca.fit_transform(df)
    reduced_df = pd.DataFrame(reduced_df)

    print('The variance in the data explained by the principal components after em
ploying PCA is equal to ' + str(pca.explained_variance_ratio_.sum()))

    return reduced_df
```

```
reduced_azdias = reduce_dimension(azdias_scaled, n=150)
```

```
The variance in the data explained by the principal components after employing PCA
is equal to 0.9093084707136746
```

```
reduced_customers = reduce_dimension(customers_scaled, n=150)
```

```
The variance in the data explained by the principal components after employing PCA
is equal to 0.9037886613509647
```

Let's check the newly created `reduced_azdias` and `reduced_customers` dataframes.

```
reduced_azdias.shape
```

```
(737288, 150)
```

```
reduced_customers.shape
```

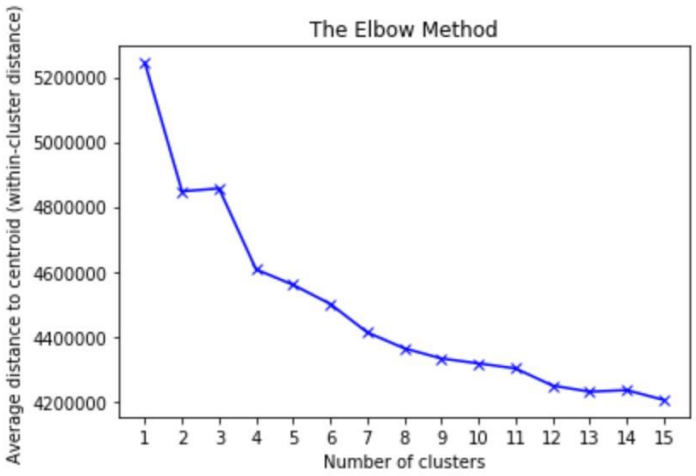
```
(134246, 150)
```

- Clustering with K-Means

When clustering with K-Means, the number K of centroids and clusters is a hyperparameter of the algorithm. To find the 'optimal' number K, we looped through a range of cluster numbers (1-15) and used the Elbow Method to make the final decision, applying k-means for each iteration of the loop. [17]

The Elbow Method gives insight into the sum of squared distance (SSE) between each data point in a cluster to the cluster's centroid, for each given number of clusters K.

```
within_cluster_distances = []
clusters = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

for cluster_num in clusters:
    within_cluster_distances.append(apply_kmeans(reduced_azdias.sample(20000), clu
ster_num))



plt.plot(clusters, distances, linestyle='-', marker='x', color='blue')

plt.xticks(ticks=clusters)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance to centroid (within-cluster distance)')
plt.title('The Elbow Method')

plt.show()
```



> The distance decreases as the number of clusters (k) we specify for k-means clustering increases.

From the graph above, we chose 12 as a good choice for K: K=12

```
n_clusters = 12

kmeans = KMeans(n_clusters=n_clusters)
```

For Azdias (General Population)

```
# AZDIAS (the general population)

azdias_preds = kmeans.fit_predict(reduced_azdias)
```

```
azdias_preds
```

```
array([11,  0,  1, ...,  5,  3,  0], dtype=int32)
```

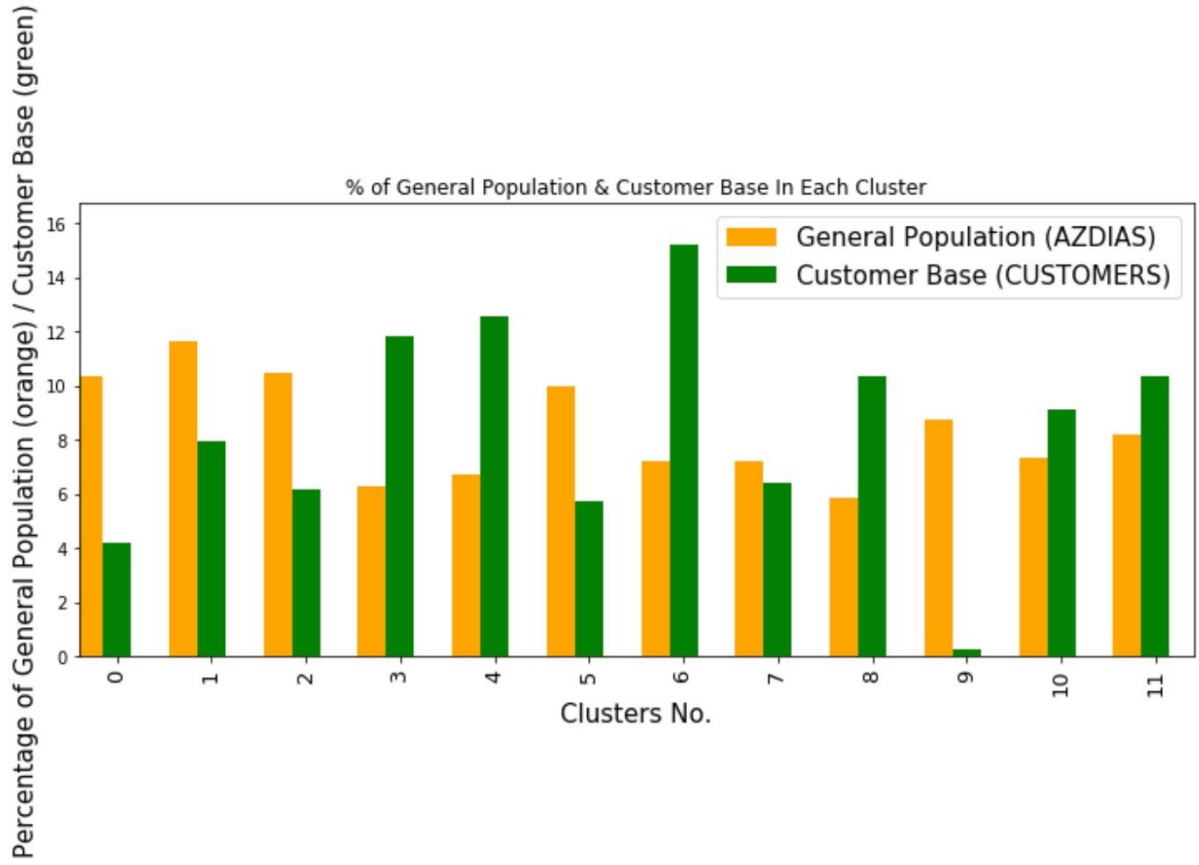
For Customers (Customer Base)

```
# CUSTOMERS (the customer base)

customers_preds = kmeans.fit_predict(reduced_customers)
```

```
customers_preds
```

```
array([ 4,  8,  4, ...,  3, 10,  4], dtype=int32)
```

Next, we compared the general (*azdias*) and customer (*customers*) populations for every cluster, which comparison can be summarised by the graph below:



We can notice that the customers population is significantly more present in the clusters No. 6, 4 and 3, relative to the general population.

The general population is significantly more present in the clusters No. 9, 0 and 5, relative to the customers population.

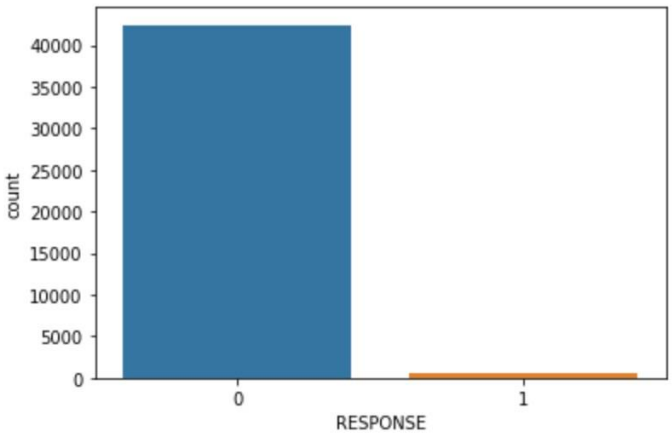
- Predictive Classifier Model (various supervised ML techniques)

After loading the training dataset (*mailout_train*), we noticed a salient class imbalance problem, where most responses were 0 (i.e. did not become customer). We needed to find a strategy for this issue, otherwise our supervised classifier would have been very inclined towards predicting near-0 values to every individual.

Dealing with Class Imbalance

```
sns.countplot('RESPONSE', data=mailout_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a342a4ed0>
```



> From the graph above, we can clearly see the class imbalance in the **mailout_train** dataset, with nearly all responses being 0 (i.e. not becoming a customer).

The strategy employed was 'resampling' (an up-sampling method), used to randomly duplicate data observations from the minority class with replacement (here, the minority class is RESPONSE = 1). This will lead to the same number of records/rows/observations in both classes (0 and 1) after resampling.

```
Yes_Customer = mailout_train[mailout_train['RESPONSE']==1]

No_Customer = mailout_train[mailout_train['RESPONSE']==0]
```

```
Yes_Customer_balanced = resample(Yes_Customer, replace=True, n_samples=42430, rand
om_state=1)
```

```
# Combine Yes_Customer_balanced and No_Customer data set

mailout_train_balanced = pd.concat([No_Customer, Yes_Customer_balanced])
```

```
mailout_train_balanced['RESPONSE']
```

```
0         0
1         0
2         0
3         0
4         0
         ..
38176     1
2506      1
21762     1
9517      1
35633     1
Name: RESPONSE, Length: 84860, dtype: int64
```

After cleaning and preprocessing the training dataset (*train*), we created a function, *classifier_training*(), callable with the different supervised learning classifiers we will use.

```
def classifier_training(classifier, param_grid, X=train, y=y_label):
    """
    Fit a classifier using GridSearchCV, compute ROC AUC metric

    INPUT:
    - classifier (classifier): classifier to fit
    - param_grid (dict): parameters of the classifier used with GridSearchCV
    - X (DataFrame): features of the training dataframe
    - y (DataFrame): labels of the training dataframe

    OUTPUT:
    - classifier: fitted classifier
    - prints elapsed time and ROX AUC
    """

    # using StratifiedKFold

    start = time.time()

    grid = GridSearchCV(estimator=classifier, param_grid=param_grid, scoring='roc_
auc', cv=5)
    grid.fit(X, y)

    end = time.time()
    print('Time Taken:' + str(end-start))

    print(grid.best_score_)

    return grid.best_estimator_
```

LogisticRegression

```
logistic_reg = LogisticRegression(random_state=12)
```

```
print(classifier_training(logistic_reg, {}))
```

```
17.693526029586792
0.8079572604618249
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=12, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

> **Time taken: 17.69s / AUC of ROC curve = 0.81**

RandomForestClassifer

```
random_forest_clf = RandomForestClassifier(random_state=12)
```

```
print(classifier_training(random_forest_clf, {}))
```

```
157.43871474266052
0.9933765830443498
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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_jobs=None, oob_score=False, random_state=12, verbose=0,
                       warm_start=False)
```

> **Time taken: 157.44s / AUC of ROC curve = 0.99**

GradientBoostingClassifier

```
gbc = GradientBoostingClassifier(random_state=12)
```

```
print(classifier_training(gbc, {}))
```

```
Time Taken:395.87442922592163
0.8886032804776567
GradientBoostingClassifier(ccp_alpha=0.0, 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,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=12, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

> **Time taken: 395.87s / AUC of ROC curve = 0.88**

C-Support Classification – not completed: see Reflection Section

```
svc = SVC(random_state=12)
```

```
print(classifier_training(svc, {}))
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-163-dcfd5e9bb10c> in <module>
```

AdaBoostClassifier

```
adaboost = AdaBoostClassifier()
```

```
print(classifier_training(adaboost, {}))
```

```
Time Taken:99.71391987800598
0.7859658744471563
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=None)
```

> **Time taken: 99s / AUC of ROC curve = 0.79**

```
kaggle.head()
```

	LNR	RESPONSE
0	1754	0.699479
1	1770	0.724889
2	1465	0.201267
3	1470	0.073405
4	1478	0.283444

```
len(kaggle[kaggle['RESPONSE']>0.5])
```

```
8883
```

Out of 42,833 individuals (rows in the test dataset), 8,883 individuals seems likely to become a customer (i.e. their individual prediction>0.5), which approximately equals to 20% of the total test individuals (42,833).

Justification

- Comparison of the final model and the benchmark model

The Logistic Regressor achieved an AUC of the ROC curve of 0.81 on the training dataset (*mailout_train*), while the GradientBoostingClassifier had scored 0.89.

This shows that our chosen model (GradientBoostingClassifier) has surpassed the benchmark model on the train dataset (which is good news), which made it a potentially great candidate for the final supervised model for Part 3.

We did not proceed further with the RandomForestClassifier model, due to concerns around overfitting the training dataset and a lack of time to look deeper into the potential issue.

Also, we interrupted the SVC model's training, as it took a very long time (several hours) and subsequent research showed us the incompatibility of this model with our training set (too big for this model to be computationally affordable, and very time-costly).

We nonetheless kept the code cell, to demonstrate the subsequent decision change.

IV. Conclusion

Reflection

- What do I think about the whole process and the project?

Overall, I have thoroughly enjoyed working on this project for a range of reasons.

Prior to starting it, in the decision process, I got excited when I saw this project recommended to us, as I had never really practiced/worked on such big and complex datasets; another reason I

really appreciated the datasets is that they were specifically handed to us for this one and only project (private property of Arvato Financial Services), which gave me insight into what type of datasets data scientists work on/with, in real-life.
I must say that I did not except to struggle as much as I did during the project; however, it definitely helped me gain confidence and skills, relevant to the projects developed in real-life settings. Adding this kind of elaborate project on top of the learning I have been 'subjecting' myself to over the past months (both in this nanodegree and with other resources) enabled me to consolidate some concepts I had learned throughout and 'best practices' I had probably forgotten about.

- What was the most interesting aspect of the project (personal)?

I was not so familiar with unsupervised machine learning, at least relatively to supervised machine learning (and I had nearly no practice with unsupervised ML algorithms), and therefore, to me, the most interesting aspect of the project has to be Part 1, with Dimensionality Reduction (using PCA) and Clustering (using k-Means).

This helped me open my eyes to unsupervised learning algorithms, which I wish to know more about, at both the conceptual and practical levels.

- What was the most difficult aspect of the project?

Not surprisingly (at least for a non-expert individual like me), the most difficult aspect of the project was the very first Part (Part 0), where I did the data preprocessing and engineering steps.

Generally speaking, when seeking to learn about Machine Learning, individuals tend to look into Machine Learning concepts, algorithms and other theoretical information, but little is said about data exploration, cleaning, preprocessing and engineering. Self-learners like me are, most of the time, given readily cleaned and preprocessed datasets, which make us rather incompetent when practicing with raw and complex datasets; this is a critical problem in the Machine Learning 'curriculum' for individual with non-technical/non-data science backgrounds, who then cannot do much with 'real' datasets: much of the data out there is 'raw', huge and difficult to interpret sometimes.

I am glad I encountered this difficulty, as this helped me realise the necessity, for me, to target further resources on data cleaning/preprocessing and feature engineering.
As of today, I am grateful to have realised the saliency of learning about those topics, which I will definitely look into in my own machine learning 'learning path'.

For now, I do not think that the score I received is sufficiently high to enable any real-life company to confidently predict future customers based on their demographics data.
For this to happen, the model needs to be further improved; another option is training another supervised machine learning estimator, and this is definitely one of the most critical option to work on, as my current machine learning knowledge is far from being advanced enough to enable me to use all the appropriate tools (algorithms, model optimization, hyperparameter tuning tricks…) the field has to offer.

- Reflection on RandomForestClassifier estimator

In Part 2, I have trained an estimator with RandomForestClassifier, which returned a AUC of the ROC curve seemingly too high (approx. 0.99) to not to be kind of 'erroneous' or overfitting the training data. This suggests something must have gone wrong during the training of the estimator, which will need further post-project investigation. Ultimately, my personal goal is to find out what triggered this score to be so high; maybe, once 'fixed', RandomForestClassifier would lead to a higher (this time, a 'plausible' *high*) AUC of the ROC curve than GradientBoostingClassifier on the train dataset.

- Reflection on SVC:

I recently heard about C-Support Vector Classification (SVC), and, after reflection, I maybe shouldn't have started using and training it before doing more research into the theory behind it and its most suitable applications.

Lots of relevant websites tackled this issue (SVC taking too long to train), which I had a look into. [18] [19]

Ultimately, there is a wide range of solutions/strategies to try to decrease the running time of SVC (e.g. using LinearSVC over SVC in this case), however, as has been often said, the best decision might be to just drop the idea of using SVC in the first place, at least in this use case and in training this dataset (which, moreover, has been resampled, and so contains even more observations/rows).

Considerations for Future Improvement

- Get more and deeper insights into the features present, before starting preprocessing and
- cleaning: maybe the preprocessing could have been better.
- More research into feature engineering: did I really make the best choices?
- Train more supervised models on the mailout_train dataset (I certainly left out many more suitable models, I wish to fill in the gaps in my knowledge!)
- Investigate more into the clusters in the customer segmentation part to gain more insights into the clusters.
- Better research on hyperparameters tuning for the supervised model in Part 2.
- Generally, document yourself more on the algorithms before employing them and potentially wasting time (e.g. with the SVC, for example)

V. References

[1] pandas.get_dummies. In *Pandas Documentation*. Retrieved from: https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

[2] Impute Missing Values with SciKit's Imputer – Python. In *Medium*. Retrieved from: https://medium.com/technofunnel/handling-missing-data-in-python-using-scikit-imputer7607c8957740

[3] StandardScaler. In *Scikit-Learn Documentation.* Retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[4] Understanding Principal Component Analysis. In *Medium*. Retrieved from: https://medium.com/@aptrishu/understanding-principle-component-analysis-e32be0253ef0

[5] K-Means Clustering in Python: A Practical Guide. In *Real Python*. Retrieved from: https://realpython.com/k-means-clustering-python/

[6] Classification: ROC Curve and AUC. In *Google Developers*. Retrieved from: https://www.datanovia.com/en/wp-content/uploads/dn-tutorials/002-partitionalclustering/images/partitioning-clustering.png

[7] RandomForestClassifier. In *Scikit-Learn Documentation*. Retrieved from: https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[8] Understanding Random Forests Classifiers in Python. In *Datacamp Community*. Retrieved from: https://www.datacamp.com/community/tutorials/random-forests-classifier-python

[9] GradientBoostingClassifier. In *Scikit-Learn Documentation*. Retrieved from: https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.ht ml

[10] Gradient Boosting with Scikit-Learn, XGBoost, LightGBM, and CatBoot. In *Machine Learning Mastery*. Retrieved from: https://machinelearningmastery.com/gradient-boostingwith-scikit-learn-xgboost-lightgbm-and-catboost/

[11] SVC. In *Scikit-Learn Documentation*. Retrieved from: https://scikitlearn.org/stable/modules/generated/sklearn.svm.SVC.html **[12]** AdaBoostClassifier. In *Scikit-Learn Documentation*. Retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html **[13]** AdaBoost Classifier in Python. In *Datacamp Community*. Retrieved from: https://www.datacamp.com/community/tutorials/adaboost-classifier-python

[14] LogisticRegression. In *Scikit-Learn Documentation*. Retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[15] Logistic Regression using Python (scikit-learn). In *Towards Data Science*. Retrieved from: https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnisthandwriting-recognition-matplotlib-a6b31e2b166a

[16] Scree Plot. In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Scree_plot

[17] Elbow Method for Optimal Value k in K-Means. In *GeeksforGeeks*. Retrieved from: https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/

[18] SVC Classifier taking too much time for training. In *StackOverFlow*. Retrieved from: https://stackoverflow.com/questions/53940258/svc-classifier-taking-too-much-time-fortraining/54004026

[19] LinearSVC. In *Scikit-Learn Documentation*. Retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

…and:

I have benefitted from many more resources of many different sources and formats throughout my project development, all of which I cannot cite here.

I'd nonetheless like to express my uttermost gratitude to the amazing community out there, to the forums, for providing invaluable help to others and non-expert enthusiast and curious individuals like me: the questions and responses you post online, the practical guides you create, and all of the free help and content you provide are beyond exceptional and extremely helpful.

Thank you all so much; and thank you to the mentor who reviewed my project proposal, to the mentor who will review my project repository, to the mentors answering my questions, and to everyone at Udacity providing these awesome nanodegrees.

Thank you very much, Udacity

Hassan Essa Alghanim