Definition:

Project Overview:

The project is divided into 2 parts:

1- Unsupervised Learning:
Cluster ‘Udacity_AZDIAS_052018.csv’, demographics data for the general population of Germany, then analyze the clusters that are more likely to be future customers by comparing them to ‘Udacity_CUSTOMERS_052018.csv, demographics data for Arvato Financial Solutions’ actual customers.

2- Supervised Learning:
Classify the demographics data for individuals who were targets of a marketing campaign, according to their responses (becoming customers or not). So, it is a binary classification problem. ‘Udacity_MAILOUT_052018_TRAIN.csv’ is the train dataset, while, ‘Udacity_MAILOUT_052018_TEST.csv’ is the test dataset.

Problem Statement:

Arvato Financial Solutions, a Bertelsmann subsidiary, is a mail-order sales company in Germany interested in identifying segments of the general population to target with their marketing in order to acquire new clients more efficiently.

Metrics:

In consideration of the second part (supervised learning), the metric is AUC in Kaggle competition: https://www.kaggle.com/competitions/udacity-arvato-identify-customers/rules However, this message is displayed, “This is a limited-participation competition. Only invited users may participate.”.

AUC (Area Under the Curve) for the ROC curve (Receiver Operating Characteristic curve), relative to the detection of customers from the mail campaign. A ROC, or receiver operating characteristic, is a graphic used to plot the true positive rate (TPR, proportion of actual customers that are labeled as so) against the false positive rate (FPR, proportion of non-customers labeled as customers).

The line plotted on these axes depicts the performance of an algorithm as we sweep across the entire output value range. We start by accepting no individuals as customers (thus giving a 0.0 TPR and FPR) then gradually increase the threshold for accepting customers until all individuals are accepted (thus giving a 1.0 TPR and FPR). The AUC, or area under the curve, summarizes the performance of the model. If a model does not discriminate between classes at all, its curve should be approximately a diagonal line from (0, 0) to (1, 1), earning a score of 0.5. A model that identifies most of the customers first, before starting to make errors, will see its curve start with a steep upward slope towards the upper-left corner before making a shallow slope towards the upper-right. The maximum score possible is 1.0, if all customers are perfectly captured by the model first. For visual explanation, see this video: https://www.youtube.com/watch?v=2Iw5TiGzJI4

Analysis

Data Exploration

The size of the data is too big; especially 'Udacity_AZDIAS_052018.csv', whose size on disk is more than 1GB. Therefore, it was necessary to use memory efficiently. This was handled by explicitly using the garbage collector after freeing up unnecessary variables. In addition, using suitable data types, such as 'float32' instead of 'float 64'. Furthermore, converting objects and strings to numerical categorical data.

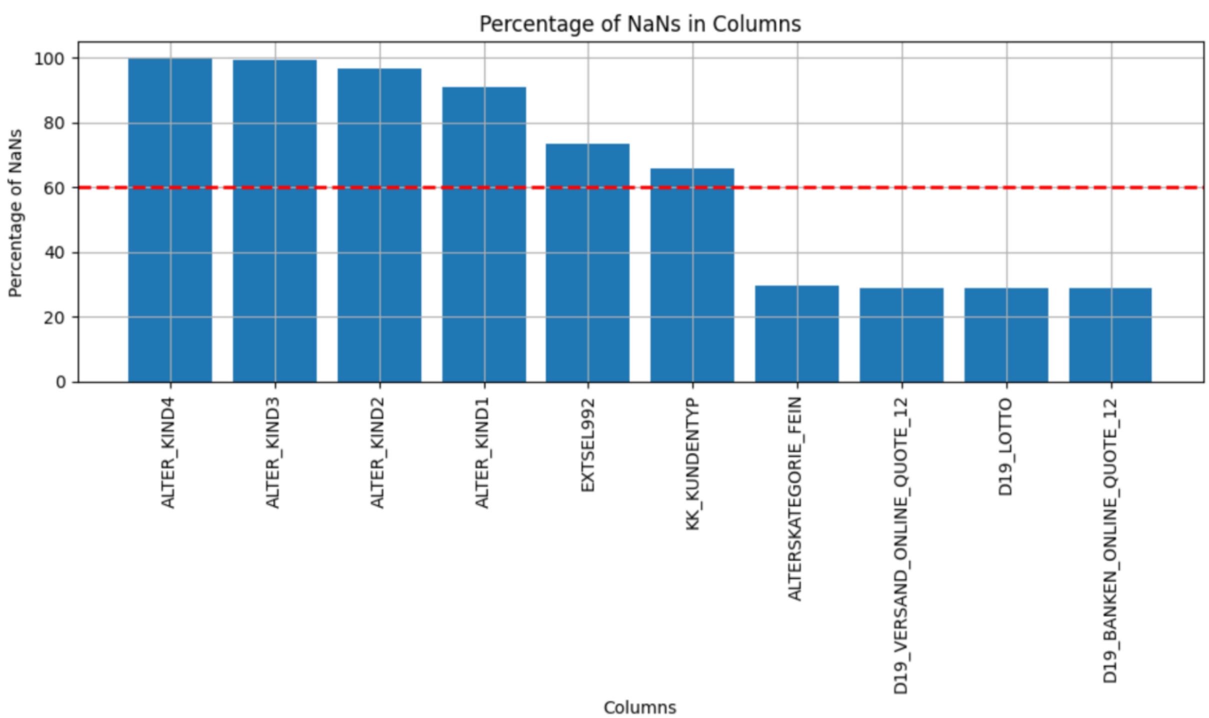
There is a lot of data inconsistency. The missing data is sometimes -1, 0, 9, np.nan, None, NAType, <NA>, 'X', 'XX'. Therefore, all these values were imputed.

There is a lot of missing data.

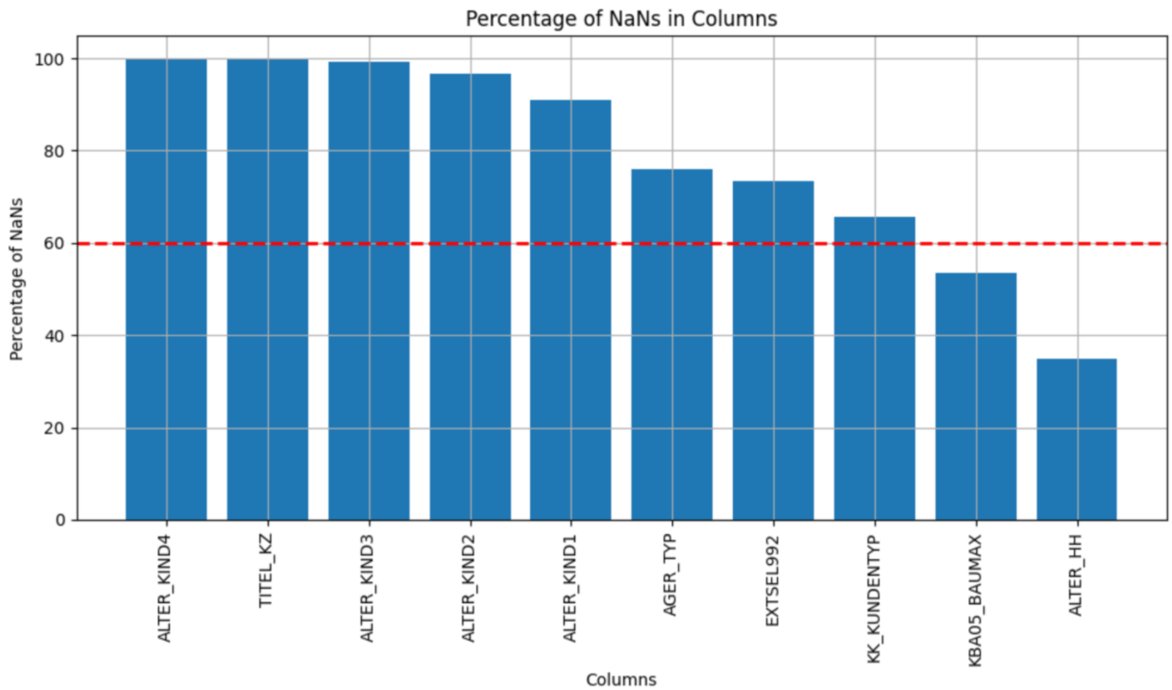
Most of the columns have limited values and can be converted to ordinal categorical data.

Exploratory Visualization

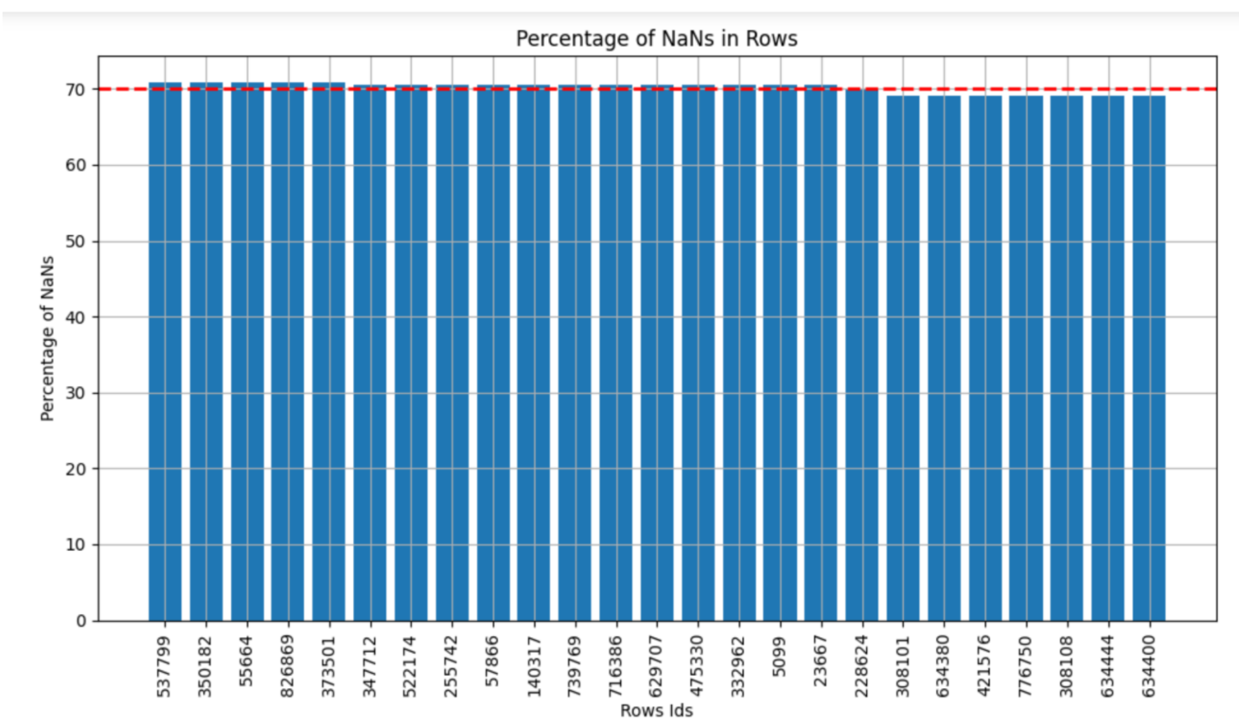
In the following figure, a representation of the percentage of missing data in columns BEFORE removing data inconsistencies (converting unknowns into NaNs):



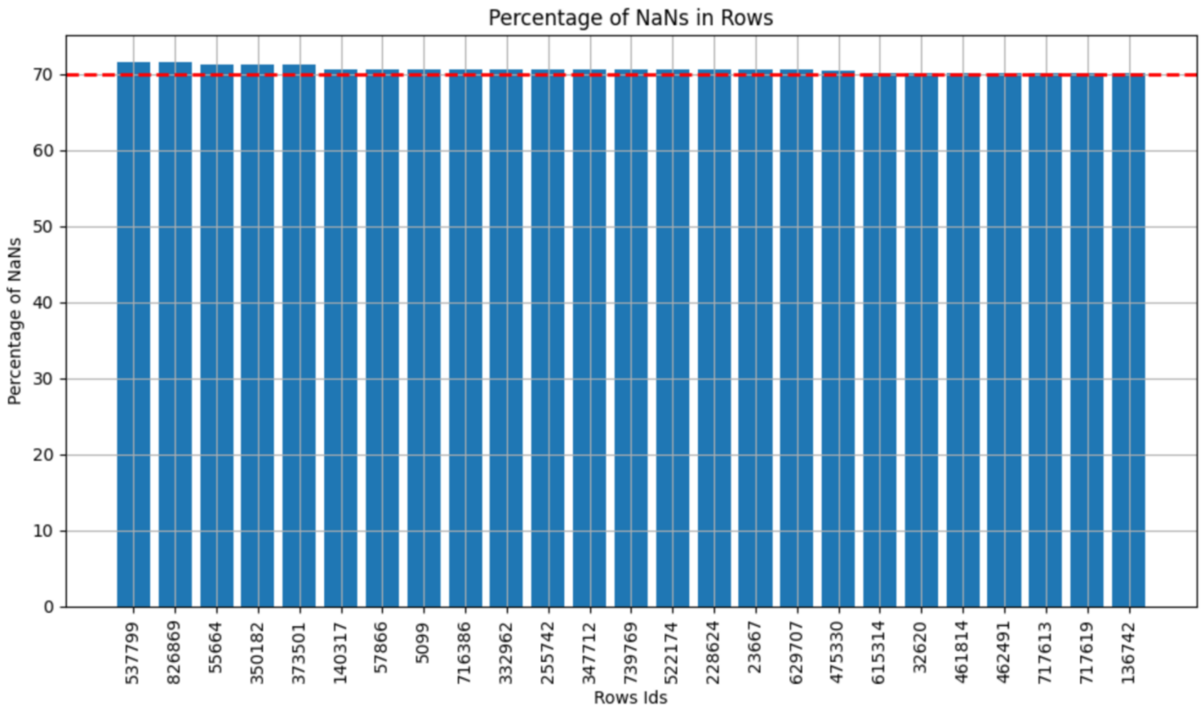
In the following figure, a representation of the percentage of missing data in columns AFTER removing data inconsistencies (converting unknowns into NaNs):



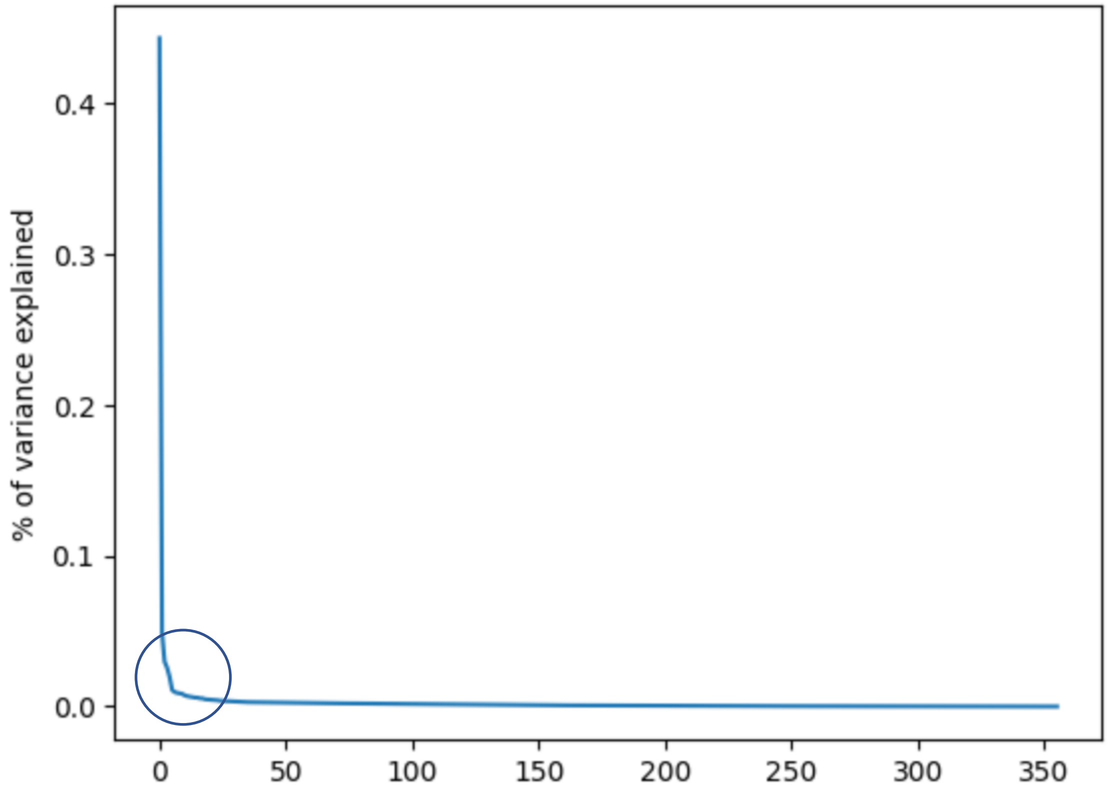
In the following figure, a representation of the percentage of missing data in rows BEFORE removing data inconsistencies (converting unknowns into NaNs):



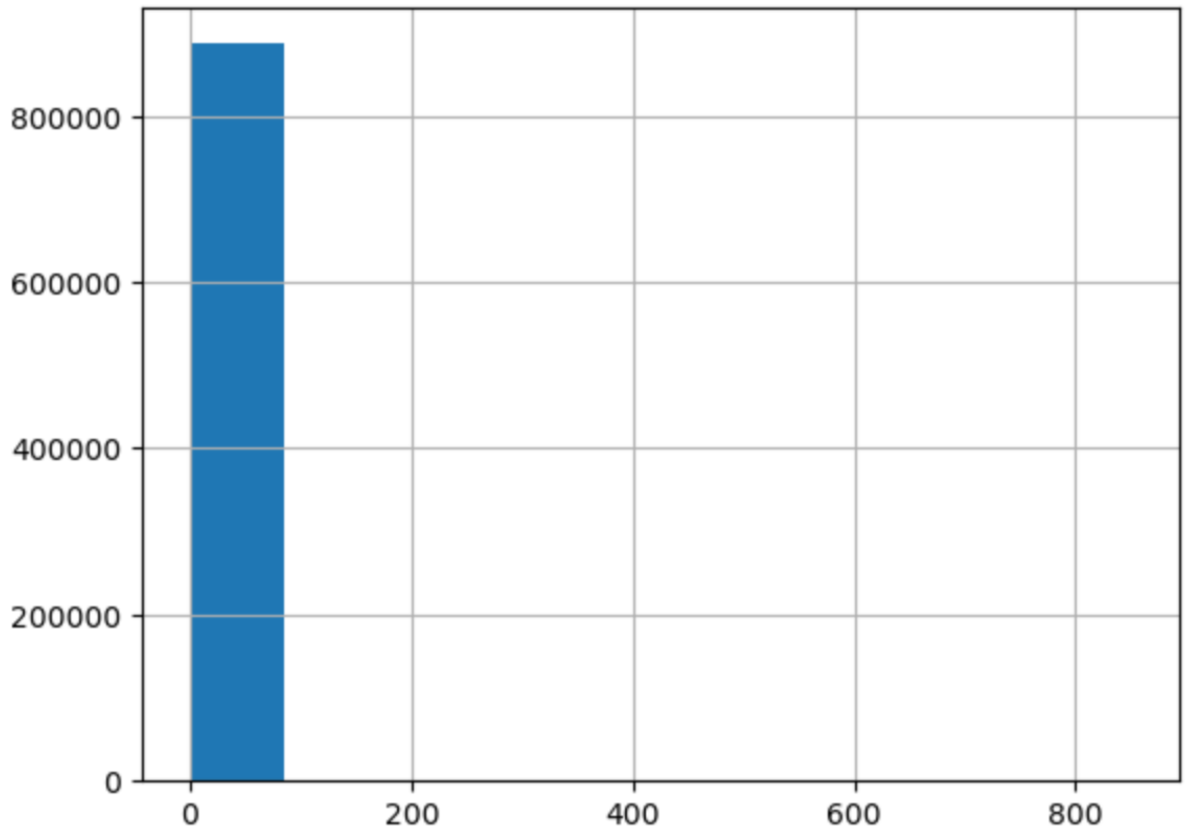
In the following figure, a representation of the percentage of missing data in rows AFTER removing data inconsistencies (converting unknowns into NaNs):



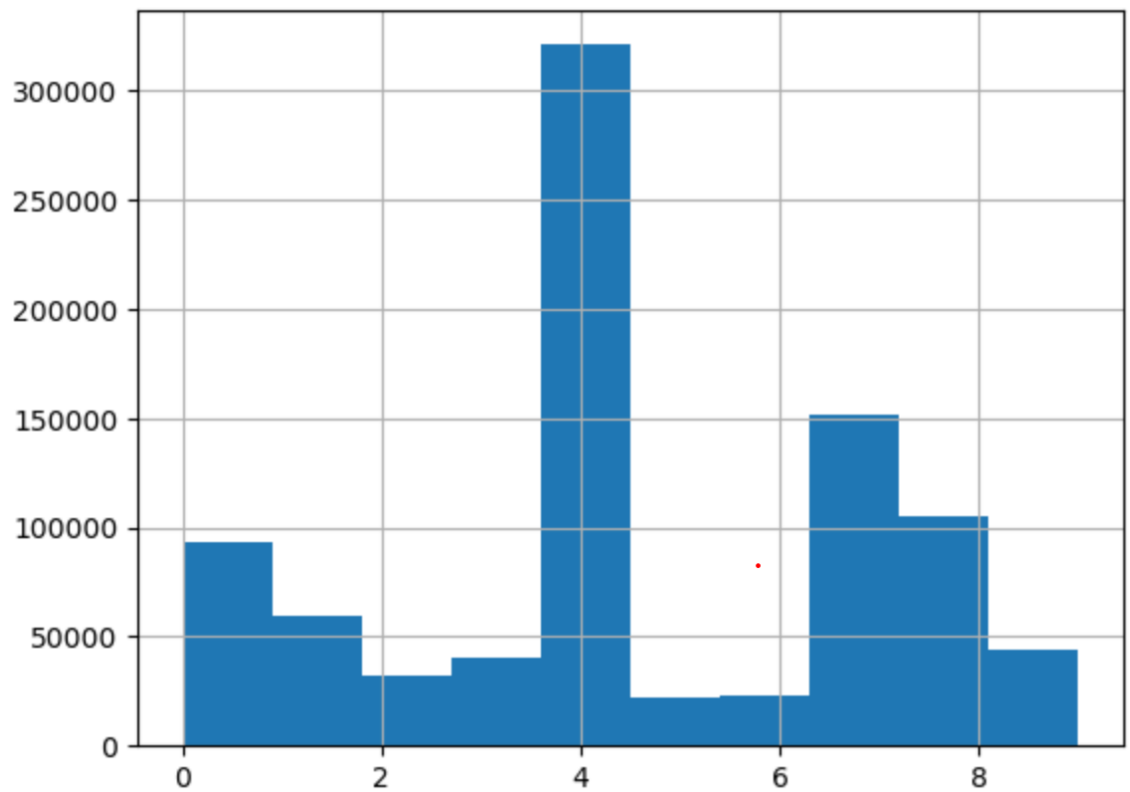
The following figure is a representation of how the variance of the fitted PCA decreases as the number of involved components (features) increases. The Elbow method is used to choose the best number of components that are not memory-consuming and at the same time gives good accuracy.



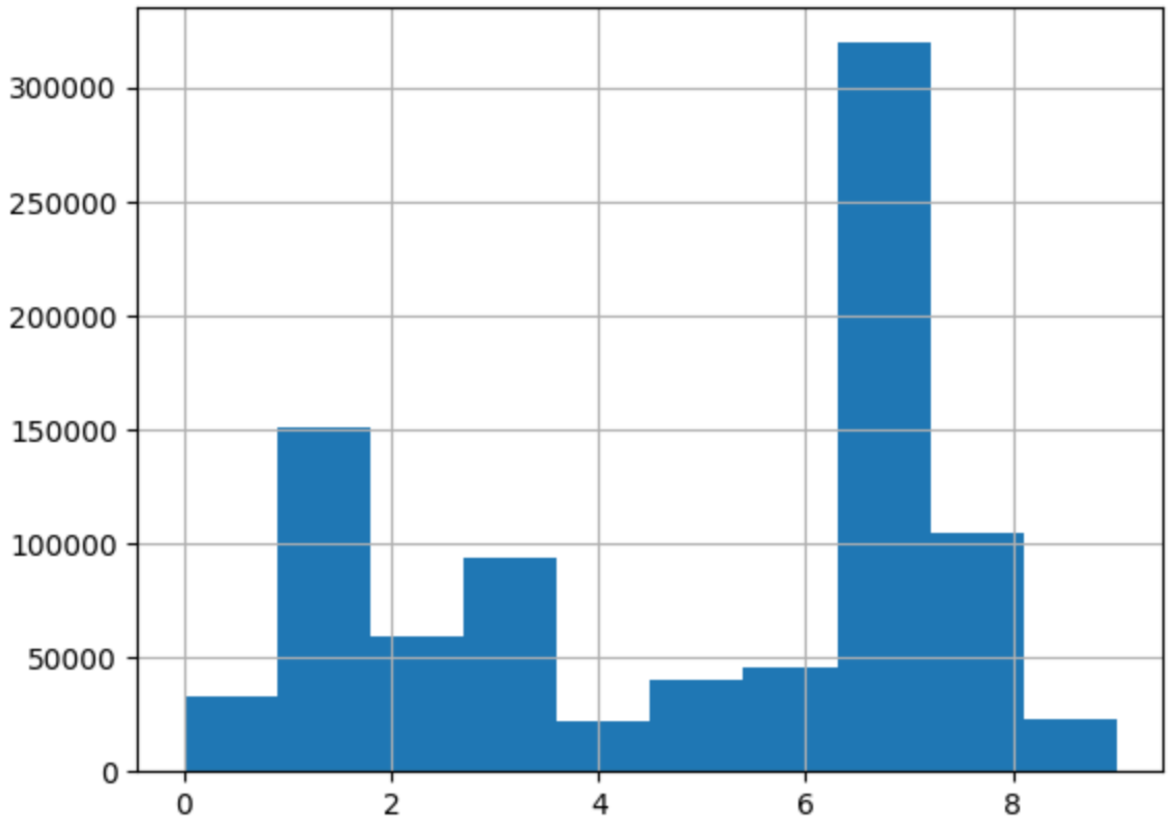
The following figure represents that all the individuals in the dataset were clustered as noise when DBSCAN was used:



Therefore, it was necessary to use change the parameters or/and algorithm. Using KMeans resulted in better clustering shown by the following figure:



When the same algorithm with the same parameters was run again, different clusters were fitted as follows:



Algorithms and Techniques

For the first part, the unsupervised learning part, PCA for dimensionality reduction, and KMeans for clustering.

For the second part, the supervised learning part, Autogluon has tried several algorithms and chose WeightedEnsemble_L2 every time.

Benchmark

Kaggle competition was a benchmark, but unfortunately, I discovered that I can't submit my results because I have no invitation. https://www.kaggle.com/competitions/udacity-arvato-identify-customers/rules

However, there were other implementations on github and blogs such as:
https://medium.com/@mt3915/customer-segmentation-for-arvato-bertelsmann-b0026efbb554
https://365datascience.com/tutorials/python-tutorials/pca-k-means/
https://medium.com/@tongxiaoling1022/create-a-customer-segmentation-report-for-arvato-financial-solutions-udacity-data-scientist-bb1194218e82
https://github.com/sallytxl/capstone
https://github.com/olgared/Capstone_Arvato_project_Term_2
https://github.com/sanjeevai/customer_segments_arvato/blob/master/README.md
https://github.com/sanjeevai/customer_segments_arvato/blob/master/Project_Rubric.pdf
https://github.com/patelatharva/Arvato_Customer_Segmentation

Methodology

Data Preprocessing

For the first part, the unsupervised learning part, done on both 'Udacity_AZDIAS_052018.csv', and 'Udacity_CUSTOMERS _052018.csv',:

1- Preparation: replacing all unknowns in the datasets with np.nan to remove inconsistency
Firstly, getting the attributes and their unknown values from 'DIAS Attributes - Values 2017.xlsx'

	A	B	C	D	E
1					
2		Attribute	Description	Value	Meaning
3		AGER_TYP	best-ager typology	-1	unknown
4				0	no classification possible
5				1	passive elderly
6				2	cultural elderly
7				3	experience-driven elderly
8		ALTERSKATEGORIE_GROB	age classification through prename analysis	-1, 0	unknown
9				1	< 30 years
10				2	30 - 45 years
11				3	46 - 60 years
12				4	> 60 years
13				9	uniformly distributed
14		ALTER_HH	main age within the household	0	unknown / no main age detectable
15				1	01.01.1895 bis 31.12.1899
16				2	01.01.1900 bis 31.12.1904
17				3	01.01.1905 bis 31.12.1909
18				4	01.01.1910 bis 31.12.1914
19				5	01.01.1915 bis 31.12.1919
20				6	01.01.1920 bis 31.12.1924
21				7	01.01.1925 bis 31.12.1929
22				8	01.01.1930 bis 31.12.1934

This was done by the following code

Clean Data

In [9]:
```
#find columns in the dataset that are not in Dias Attributes
original_df = pd.read_excel('DIAS Attributes - Values 2017.xlsx')
```

In [10]:
```
original_df.head()
```

Out[10]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	NaN	Attribute	Description	Value	Meaning
1	NaN	AGER_TYP	best-ager typology	-1	unknown
2	NaN	NaN	NaN	0	no classification possible
3	NaN	NaN	NaN	1	passive elderly
4	NaN	NaN	NaN	2	cultural elderly

In [11]:
```
attributes_in_file = original_df.iloc[1:, 1].copy()
```

In [12]:
```
attributes_in_file.head()
```

Out[12]:
```
1     AGER_TYP
2          NaN
3          NaN
4          NaN
5          NaN
Name: Unnamed: 1, dtype: object
```

In [13]:
```
attributes_in_file.dropna(inplace=True)
```

In [14]:
```
attributes_in_file.head()
```

Out[14]:
```
1                 AGER_TYP
6      ALTERSKATEGORIE_GROB
12                ALTER_HH
34               ANREDE_KZ
37       ANZ_HAUSHALTE_AKTIV
Name: Unnamed: 1, dtype: object
```

In [15]:
```
attributes_in_file.count()
```

Out[15]:
```
314
```

```
In [17]: #convert unknowns into Nans only--> Data consistency: sum unknowns are NaNs, some are -1, some are 0 and some are 9.
         unknowns = original_df.copy()#pd.read_excel('DIAS Attributes - Values 2017.xlsx')
```

```
In [18]: unknowns = unknowns.iloc[:,1:] #remove first column
         unknowns.set_axis(unknowns.iloc[0,:], axis='columns', inplace=True)#inplace worked on another environment #rename column names
         unknowns = unknowns.iloc[1:,:] #remove first row
         original_df = unknowns.copy() #after cleaning
         unknowns.head()
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_12048\2597164357.py:2: FutureWarning: DataFrame.set_axis 'inplace' keyword is deprecated a
  unknowns.set_axis(unknowns.iloc[0,:], axis='columns', inplace=True)#inplace worked on another environment #rename column names
```

Out[18]:

	Attribute	Description	Value	Meaning
1	AGER_TYP	best-ager typology	-1	unknown
2	NaN	NaN	0	no classification possible
3	NaN	NaN	1	passive elderly
4	NaN	NaN	2	cultural elderly
5	NaN	NaN	3	experience-driven elderly

```
In [22]: unknowns = unknowns[['Attribute', 'Value']].where(unknowns['Meaning'].str.contains('unknown')).dropna(how='all') #drop if all row values are all Nones
         #when I tried it without how='all' I got 231 rows only instead of 233
         unknowns
```

Out[22]:

	Attribute	Value
1	AGER_TYP	-1
6	ALTERSKATEGORIE_GROB	-1, 0
12	ALTER_HH	0
34	ANREDE_KZ	-1, 0
41	BALLRAUM	-1
...	...	...
2220	WOHNDAUER_2008	-1, 0
2230	WOHNLAGE	-1
2239	WACHSTUMSGEBIET_NB	-1, 0
2245	W_KEIT_KIND_HH	-1, 0
2252	ZABEOTYP	-1, 9

233 rows × 2 columns

But wait! Sometimes the unknown value is not on the same row as the attribute name like in this screenshot:

KBA05_AUTOQUOT	share of cars per household	1	very low car quote
		2	low car quote
		3	average car quote
		4	high car quote
		5	very high car quote
		-1, 9	unknown

Here we should return back by index a few steps until we find the name as shown in the function get_attribute_name:

```
In [2]: def get_attribute_name(original_df, index):
            up_index = index
            while type(original_df.iloc[up_index, 0]) == float: #equals NaN
                up_index = up_index - 1
            return up_index, original_df.iloc[up_index, 0] #type == str means not equal NaN
```

This can be done by already-made ffill(), but it is not preferable as it traverses the whole dataset. Then, convert all unknowns into dictionary for ease of access O(1)

```
In [27]: unknowns_dict = dict(zip(unknowns['Attribute'], unknowns['Value']))
         unknowns_dict
```

Then, traversing this dict, check if this attribute is already in the dataset or not, if yes, handle both cases, whether it has 2 values in ‘try’ block or only one value in ‘except’ block:

```
In [29]: for column_name, values in unknowns_dict.items():
             if column_name in azdias.columns:
                 try:
                     values_list = values.split(',')
                 except:
                     values_list = [int(values)]
                 azdias[column_name] = azdias[column_name].replace([int(value) for value in values_list], float('NaN'))
             else:
                 print(f"Column '{column_name}' not found in DataFrame.")
```

```
Column 'BIP_FLAG' not found in DataFrame.
Column 'CAMEO_DEUINTL_2015' not found in DataFrame.
Column 'D19_KK_KUNDENTYP' not found in DataFrame.
Column 'GEOSCORE_KLS7' not found in DataFrame.
Column 'HAUSHALTSSTRUKTUR' not found in DataFrame.
Column 'KBA13_CCM_1400_2500' not found in DataFrame.
Column 'SOHO_FLAG' not found in DataFrame.
Column 'WACHSTUMSGEBIET_NB' not found in DataFrame.
```

2- Drop the columns of more than 60% NaNs and rows of more than 70% NaNs

3- Converting all columns into numerical so that they can be fitted in the PCA and KMeans
For example, ‘CAMEO_DEU_2015’

```
In [40]: L1 = azdias['CAMEO_DEU_2015'].value_counts().index
         L1.sort_values()
         print(L1)
         print(len(L1))#45

         L2 = list(range(1,45))
         L2.extend([np.nan])

         print(L2)
         print(len(L2))#45

         azdias['CAMEO_DEU_2015'].replace(L1, L2, inplace=True)
```

```
Index(['6B', '8A', '4C', '2D', '3C', '7A', '3D', '8B', '4A', '8C', '9D', '9B',
       '9C', '7B', '9A', '2C', '8D', '6E', '2B', '5D', '6C', '2A', '5A', '1D',
       '1A', '3A', '5B', '5C', '7C', '4B', '4D', '3B', '6A', '9E', '6D', '6F',
       '7D', '4E', '1E', '7E', '1C', '5F', '1B', '5E', 'XX'],
      dtype='object')
45
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, nan]
45
```

4- After removing ‘X’ and ‘XX’, it was necessary to fill the new NaNs using forward fill, leaving first rows having NaNs, then using backward fill to remove the first rows NaNs, then dropping the columns with too many NaNs

```
In [59]: azdias.fillna(method='ffill', inplace=True)
         azdias.fillna(method='bfill', inplace=True)
         azdias.head()
```

Out[59]:

	LNR	AKT_DAT_KL	ALTER_HH	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV	ANZ_HH_TITEL	ANZ_KINDER	ANZ_PERSONEN	ANZ_STATISTISCHE_HAUSHALTE	ANZ_TITEL	...	VHN	VK_DHT4A	VK_DISTANZ	VK_ZG11	W_KEIT_KIND_HI
0	910215	9	17	21	11	0	0	2	12	0	...	4	8	11	10	
1	910220	9	17	21	11	0	0	2	12	0	...	4	8	11	10	
2	910225	9	17	17	10	0	0	1	7	0	...	2	9	9	6	
3	910226	1	13	13	1	0	0	0	2	0	...	0	7	10	11	
4	910241	1	20	14	3	0	0	4	3	0	...	2	3	5	4	

5 rows × 357 columns

```
In [60]: azdias.isnull().sum().sum()
```

```
Out[60]: 891203
```

```
In [61]: azdias.isnull().sum().sort_values(ascending=False)
```

```
Out[61]: CAMEO_INTL_2015             891203
         LNR                              0
         KBA13_KMH_180                    0
         KBA13_KRSHERST_FORD_OPEL         0
         KBA13_KRSHERST_BMW_BENZ          0
                                       ...
         KBA05_ANTG1                      0
         KBA05_ANHANG                     0
         KBA05_ALTER4                     0
         KBA05_ALTER3                     0
         ALTERSKATEGORIE_GROB             0
         Length: 357, dtype: int64
```

```
In [62]: azdias.drop(columns='CAMEO_INTL_2015', inplace=True)
```

5- Normalization

6- Standardization

```
In [7]:  from sklearn import preprocessing
         azdias_normalized = preprocessing.normalize(azdias_cleaned)
         scaler = preprocessing.StandardScaler()
         azdias_scaled = scaler.fit_transform(azdias_normalized)
```

For the supervised learning part:

```
In [46]: #cleaning function for mailout_train and mailout_test

         def clean_supervised(df):
             #remove missing values
             df['CAMEO_DEUG_2015'].replace('X', np.nan, inplace=True)
             df['CAMEO_INTL_2015'].replace('XX', np.nan, inplace=True)
             df.fillna(np.nan, inplace=True) #replace None by np.nan
             df = replace_unknowns(df)
             imputer = SimpleImputer(strategy='most_frequent', copy=False)
             x = imputer.fit_transform(df)
             df = pd.DataFrame(x, columns = df.columns)

             #change dtypes
             df = df.set_index('LNR')

             df = df.convert_dtypes()

             df['EINGEFUEGT_AM'] = pd.to_datetime(df['EINGEFUEGT_AM'])

             df['CAMEO_DEUG_2015'] = df['CAMEO_DEUG_2015'].astype('float')
             df['CAMEO_DEUG_2015'] = df['CAMEO_DEUG_2015'].astype('int32')#'Int64') #not int to allow NaNs
             df['CAMEO_DEUG_2015'] = df['CAMEO_DEUG_2015'].astype('category')

             df['CAMEO_INTL_2015'] = df['CAMEO_INTL_2015'].astype('float')
             df['CAMEO_INTL_2015'] = df['CAMEO_INTL_2015'].astype('Int64') #not int to allow NaNs
             df['CAMEO_INTL_2015'] = df['CAMEO_INTL_2015'].astype('category')

             df[['CAMEO_DEU_2015', 'D19_LETZTER_KAUF_BRANCHE', 'OST_WEST_KZ']] = df[['CAMEO_DEU_2015', 'D19_LETZTER_KAUF_BRANCHE', 'OST_WEST_KZ']].astype('category')

             df[df.select_dtypes(include=['Int64']).columns.drop('KBA13_ANZAHL_PKW')] = df[df.select_dtypes(include=['Int64']).columns.drop('KBA13_ANZAHL_PKW')].astype(pd.CategoricalDtype(ordered=True))#'categor

             df['KBA13_ANZAHL_PKW'] = df['KBA13_ANZAHL_PKW'].astype(int)
```

1- Removing inconsistency of missing values as explained before.
2- Replacing missing values by the mode or the most common values in each column using SimpleImputer.
   There are 4 imputers in sklearn: sklearn.impute.SimpleImputer, sklearn.impute.IterativeImputer, sklearn.impute.KNNImputer, and sklearn.impute.MissingIndicator. Since, the data is too big, SimpleImputer was best for efficiency.
3- Setting ‘LNR’ as index since it has unique values.
4- Converting 'EINGEFUEGT_AM' into datetime
5- Converting all columns except 'KBA13_ANZAHL_PKW' into ordinal categorical type since all columns have limited set of values except 'KBA13_ANZAHL_PKW' which was converted to integer.

Implementation and Refinement

There were several challenges:

1- The data is not explicitly available; it was only available in the workplace in Udacity and this was so time-consuming as it depends on the internet connectivity and the kernel dies and all the variables are lost and the cells need to be run again from the beginning. So, it was necessary to download the data on my local machine.

Download Data

Write these lines in the Udacity workspace

```
import pandas as pd

azidas = pd.read_csv('../../data/Term2/capstone/arvato_data/Udacity_AZDIAS_052018.csv', sep=';')
azidas.to_csv('./Udacity_AZDIAS_052018.csv', sep=';', index=False)

customers = pd.read_csv('../../data/Term2/capstone/arvato_data/Udacity_CUSTOMERS_052018.csv', sep=';')
customers.to_csv('./Udacity_CUSTOMERS_052018.csv', sep=';', index=False)

mailout_train = pd.read_csv('../../data/Term2/capstone/arvato_data/Udacity_MAILOUT_052018_TRAIN.csv', sep=';')
mailout_train.to_csv('./Udacity_MAILOUT_052018_TRAIN.csv', sep=';', index=False)

mailout_test = pd.read_csv('../../data/Term2/capstone/arvato_data/Udacity_MAILOUT_052018_TEST.csv', sep=';')
mailout_test.to_csv('./Udacity_MAILOUT_052018_TEST.csv', sep=';', index=False)
```

Then, open them and download them from the workspace to your computer manually: File --> Download

2- The data is too big:
The program produced MemoryError, but it was handled by calling the garbage collector and pickling important variables only to be loaded when they are needed only. Also, choosing 'category' dtype in the supervised learning part, converting strings to integers, converting float64 to float32.

```
In [13]: azdias_scaled = azdias_scaled.astype('float32') #convert from float64 to float32 to reduce memory
```

```
In [50]: azdias['D19_LETZTER_KAUF_BRANCHE'].value_counts().index
```

```
Out[50]: Index(['D19_UNBEKANNT', 'D19_VERSICHERUNGEN', 'D19_SONSTIGE',
       'D19_VOLLSORTIMENT', 'D19_SCHUHE', 'D19_BUCH_CD', 'D19_VERSAND_REST',
       'D19_DROGERIEARTIKEL', 'D19_BANKEN_DIREKT', 'D19_BEKLEIDUNG_REST',
       'D19_HAUS_DEKO', 'D19_TELKO_MOBILE', 'D19_ENERGIE', 'D19_TELKO_REST',
       'D19_BANKEN_GROSS', 'D19_BEKLEIDUNG_GEH', 'D19_KINDERARTIKEL',
       'D19_FREIZEIT', 'D19_TECHNIK', 'D19_LEBENSMITTEL', 'D19_BANKEN_REST',
       'D19_RATGEBER', 'D19_NAHRUNGSERGAENZUNG', 'D19_DIGIT_SERV',
       'D19_REISEN', 'D19_TIERARTIKEL', 'D19_SAMMELARTIKEL', 'D19_HANDWERK',
       'D19_WEIN_FEINKOST', 'D19_GARTEN', 'D19_BANKEN_LOKAL', 'D19_BIO_OEKO',
       'D19_BILDUNG', 'D19_LOTTO', 'D19_KOSMETIK'],
      dtype='object')
```

Seems to be categorical also

```
In [51]: L1 = ['D19_UNBEKANNT', 'D19_VERSICHERUNGEN', 'D19_SONSTIGE',
        'D19_VOLLSORTIMENT', 'D19_SCHUHE', 'D19_BUCH_CD', 'D19_VERSAND_REST',
        'D19_DROGERIEARTIKEL', 'D19_BANKEN_DIREKT', 'D19_BEKLEIDUNG_REST',
        'D19_HAUS_DEKO', 'D19_TELKO_MOBILE', 'D19_ENERGIE', 'D19_TELKO_REST',
        'D19_BANKEN_GROSS', 'D19_BEKLEIDUNG_GEH', 'D19_TECHNIK',
        'D19_KINDERARTIKEL', 'D19_FREIZEIT', 'D19_LEBENSMITTEL',
        'D19_BANKEN_REST', 'D19_RATGEBER', 'D19_NAHRUNGSERGAENZUNG',
        'D19_DIGIT_SERV', 'D19_REISEN', 'D19_SAMMELARTIKEL', 'D19_TIERARTIKEL',
        'D19_HANDWERK', 'D19_WEIN_FEINKOST', 'D19_GARTEN', 'D19_BANKEN_LOKAL',
        'D19_BIO_OEKO', 'D19_BILDUNG', 'D19_KOSMETIK', 'D19_LOTTO']
len(L1)
L2 = list(range(1, 36))
len(L2)
azdias['D19_LETZTER_KAUF_BRANCHE'].replace(L1, L2, inplace=True)
azdias['D19_LETZTER_KAUF_BRANCHE'] = azdias['D19_LETZTER_KAUF_BRANCHE'].astype('Int64')
```

3- The dependencies of some packages:
At the beginning of my trials I tried MCA and FAMD after converting most of the data to

‘category’ instead of PCA. This required me to install ‘prince’ package, which ruined the already installed packages. Then I decided to convert my data into numerical and to use PCA instead. This happened again when it came to the supervised learning when I installed ‘autogluon’. So, I decided to install autogluon on a new fresh virtual environment then install other packages, following these exact steps:

```
conda activate base
python --version (AutoGluon requires Python version 3.8, 3.9, or 3.10 and
is available on Linux, MacOS, and Windows.)
python -m venv myenv
myenv\Scripts\activate
pip install autogluon
myenv\Scripts\activate (again)
pip install ipykernel
python -m ipykernel install --user --name=myenv --display-name "Autogluon
Virtual Environment"
close jupyter notebook and open again
pip install pca
change kernel and choose Autogluon Virtual Environment again
pip install openpyxl
```

4- Finding the best algorithm and parameters:
Adding to trying categorical data with MCA and FAMD, I tried DBSCAN 2 times but it clustered all points as noise.
First time:

In [5]:
```
#from the above figure, and using the elbow method, the suitable n_components around 12
pca = decomposition.PCA(n_components= 12)
pca.fit(azdias_scaled)
azdias_pca = pd.DataFrame(pca.transform(azdias_scaled))
```

In [6]:
```
from sklearn import cluster
minPoints = 1000
epsilon = 1
dbscan = cluster.DBSCAN(eps=epsilon, min_samples=minPoints)
clustering_labels = dbscan.fit_predict(azdias_pca)
```

In [9]:
```
noise = list(clustering_labels).count(-1)
noise
```

Out[9]: 891203

Since, all points are considered noise, I'll try another epsilon and minPoints

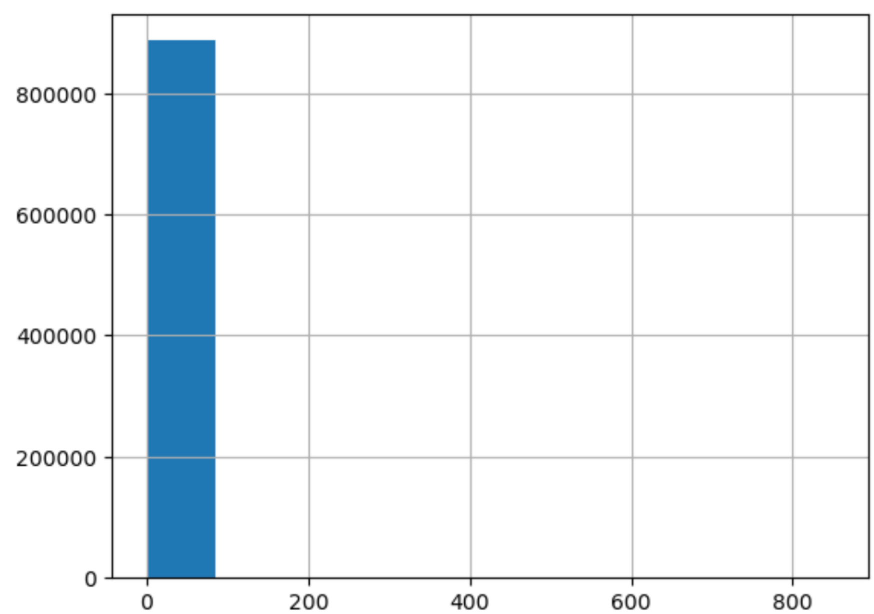
Second time:

```
In [32]: from sklearn import cluster
minPoints = 5
epsilon = 1
dbscan = cluster.DBSCAN(eps=epsilon, min_samples=minPoints)
clustering_labels_1 = dbscan.fit_predict(azdias_pca.iloc[:, 0:12])
```

```
In [33]: list(clustering_labels_1).count(-1)
Out[33]: 844450
```

```
In [37]: set(clustering_labels_1)
Out[37]: {-1,
 0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
```

```
In [36]: azdias_pca['cluster_dbscan']=clustering_labels_1
azdias_pca['cluster_dbscan'].hist()
Out[36]: <Axes: >
```



Since, most of points are considered noise, I'll try another algorithm

Results

Model Evaluation and Validation

For the unsupervised learning part, the model should be evaluated by finding the components that make a person in Germany a customer.

For the supervised learning part, the model is evaluated by the score given by Kaggle competition.

Justification

The results are not satisfying. Considering the unsupervised learning part, although, the demographic data of Germany was clustered, no customers were found in this data having the same id ('LNR'). Considering the supervised learning part, the score (AUC) of the training set was approximately 0.7788 which is very good. But, on the test set, no score was received due to the inability to submit to Kaggle. The results on the test data predicted 100% of people to NOT become customers in the future. I think that this is because the training data is unbalanced giving only 1.2% to be customers.

```
In [50]:  mailout_train['RESPONSE'].describe()

Out[50]:  count    42962.000000
          mean         0.012383
          std          0.110589
          min          0.000000
          25%          0.000000
          50%          0.000000
          75%          0.000000
          max          1.000000
          Name: RESPONSE, dtype: float64
```

Future Work

If I had more time before the deadline of the project, I would have tried the following:

1- Clustering using Ground Truth on aws. (I tried but I the data failed to be uploaded as it took too much time)
2- Fitting PCA on 2 components only to be able to plot them and hence be able to find out what makes a person in the community a customer for Arvato company.
3- Adjust the mailout_train data to make it balanced with around 50% response equals zero and 50% response equals one.