Identify_Customer_Segments

November 7, 2020

1 Project: Identify Customer Segments

In this project, you will apply unsupervised learning techniques to identify segments of the population that form the core customer base for a mail-order sales company in Germany. These segments can then be used to direct marketing campaigns towards audiences that will have the highest expected rate of returns. The data that you will use has been provided by our partners at Bertelsmann Arvato Analytics, and represents a real-life data science task.

This notebook will help you complete this task by providing a framework within which you will perform your analysis steps. In each step of the project, you will see some text describing the subtask that you will perform, followed by one or more code cells for you to complete your work. **Feel free to add additional code and markdown cells as you go along so that you can explore everything in precise chunks.** The code cells provided in the base template will outline only the major tasks, and will usually not be enough to cover all of the minor tasks that comprise it.

It should be noted that while there will be precise guidelines on how you should handle certain tasks in the project, there will also be places where an exact specification is not provided. **There will be times in the project where you will need to make and justify your own decisions on how to treat the data.** These are places where there may not be only one way to handle the data. In real-life tasks, there may be many valid ways to approach an analysis task. One of the most important things you can do is clearly document your approach so that other scientists can understand the decisions you've made.

At the end of most sections, there will be a Markdown cell labeled **Discussion**. In these cells, you will report your findings for the completed section, as well as document the decisions that you made in your approach to each subtask. **Your project will be evaluated not just on the code used to complete the tasks outlined, but also your communication about your observations and conclusions at each stage.**

In [1]:
```
# import libraries here; add more as necessary
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import numbers
import operator

from sklearn.preprocessing import Imputer, StandardScaler
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

# magic word for producing visualizations in notebook
%matplotlib inline

print("Libraries successfully imported!")

'''
Import note: The classroom currently uses sklearn version 0.19.
If you need to use an imputer, it is available in sklearn.preprocessing.Imputer,
instead of sklearn.impute as in newer versions of sklearn.
'''
```

```
Libraries successfully imported!
```

```
Out[1]: '\nImport note: The classroom currently uses sklearn version 0.19.\nIf you need to use a
```

1.0.1 Step 0: Load the Data

There are four files associated with this project (not including this one):

- `Udacity_AZDIAS_Subset.csv`: Demographics data for the general population of Germany; 891211 persons (rows) x 85 features (columns).
- `Udacity_CUSTOMERS_Subset.csv`: Demographics data for customers of a mail-order company; 191652 persons (rows) x 85 features (columns).
- `Data_Dictionary.md`: Detailed information file about the features in the provided datasets.
- `AZDIAS_Feature_Summary.csv`: Summary of feature attributes for demographics data; 85 features (rows) x 4 columns

Each row of the demographics files represents a single person, but also includes information outside of individuals, including information about their household, building, and neighborhood. You will use this information to cluster the general population into groups with similar demographic properties. Then, you will see how the people in the customers dataset fit into those created clusters. The hope here is that certain clusters are over-represented in the customers data, as compared to the general population; those over-represented clusters will be assumed to be part of the core userbase. This information can then be used for further applications, such as targeting for a marketing campaign.

To start off with, load in the demographics data for the general population into a pandas DataFrame, and do the same for the feature attributes summary. Note for all of the `.csv` data files in this project: they're semicolon (`;`) delimited, so you'll need an additional argument in your `read_csv()` call to read in the data properly. Also, considering the size of the main dataset, it may take some time for it to load completely.

Once the dataset is loaded, it's recommended that you take a little bit of time just browsing the general structure of the dataset and feature summary file. You'll be getting deep into the innards of the cleaning in the first major step of the project, so gaining some general familiarity can help you get your bearings.

In [2]:
```
# Load in the general demographics data.
azdias = pd.read_csv('Udacity_AZDIAS_Subset.csv', sep = ';')

# Load in the feature summary file.
feat_info = pd.read_csv('AZDIAS_Feature_Summary.csv', sep = ';')
```

In [3]:
```
# Check the structure of the data after it's loaded (e.g. print the number of
# rows and columns, print the first few rows).
print('Number of rows in AZDIAS dataset: ', azdias.shape[0])
print('Number of columns in AZDIAS dataset: ', azdias.shape[1])
print('*'*50)
print('Number of rows in Feature Summary file: ', feat_info.shape[0])
print('Number of columns in Feature Summary file: ', feat_info.shape[1])
```

```
Number of rows in AZDIAS dataset:  891221
Number of columns in AZDIAS dataset:  85
**************************************************
Number of rows in Feature Summary file:  85
Number of columns in Feature Summary file:  4
```

In [4]:
```
print('First few rows of AZDIAS dataset:')
azdias.head(5)
```

```
First few rows of AZDIAS dataset:
```

Out[4]:
```
   AGER_TYP  ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  \
0        -1                     2          1            2.0
1        -1                     1          2            5.0
2        -1                     3          2            3.0
3         2                     4          2            2.0
4        -1                     3          1            5.0

   FINANZ_MINIMALIST  FINANZ_SPARER  FINANZ_VORSORGER  FINANZ_ANLEGER  \
0                  3              4                 3               5
1                  1              5                 2               5
2                  1              4                 1               2
3                  4              2                 5               2
4                  4              3                 4               1

   FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER    ...     PLZ8_ANTG1  PLZ8_ANTG2  \
0                      5                 3    ...            NaN         NaN
1                      4                 5    ...            2.0         3.0
2                      3                 5    ...            3.0         3.0
3                      1                 2    ...            2.0         2.0
4                      3                 2    ...            2.0         4.0

   PLZ8_ANTG3  PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  \
```

```
0         NaN        NaN         NaN       NaN       NaN      NaN
1         2.0        1.0         1.0       5.0       4.0      3.0
2         1.0        0.0         1.0       4.0       4.0      3.0
3         2.0        0.0         1.0       3.0       4.0      2.0
4         2.0        1.0         2.0       3.0       3.0      4.0

   ORTSGR_KLS9  RELAT_AB
0          NaN       NaN
1          5.0       4.0
2          5.0       2.0
3          3.0       3.0
4          6.0       5.0

[5 rows x 85 columns]
```

In [5]:
```python
print('Summary statistics for AZDIAS dataset:')
azdias.describe()
```

```
Summary statistics for AZDIAS dataset:
```

Out[5]:
```
             AGER_TYP  ALTERSKATEGORIE_GROB      ANREDE_KZ  CJT_GESAMTTYP  \
count   891221.000000         891221.000000  891221.000000  886367.000000
mean        -0.358435              2.777398       1.522098       3.632838
std          1.198724              1.068775       0.499512       1.595021
min         -1.000000              1.000000       1.000000       1.000000
25%         -1.000000              2.000000       1.000000       2.000000
50%         -1.000000              3.000000       2.000000       4.000000
75%         -1.000000              4.000000       2.000000       5.000000
max          3.000000              9.000000       2.000000       6.000000

       FINANZ_MINIMALIST  FINANZ_SPARER  FINANZ_VORSORGER  FINANZ_ANLEGER  \
count      891221.000000  891221.000000     891221.000000   891221.000000
mean            3.074528       2.821039          3.401106        3.033328
std             1.321055       1.464749          1.322134        1.529603
min             1.000000       1.000000          1.000000        1.000000
25%             2.000000       1.000000          3.000000        2.000000
50%             3.000000       3.000000          3.000000        3.000000
75%             4.000000       4.000000          5.000000        5.000000
max             5.000000       5.000000          5.000000        5.000000

       FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER      ...           PLZ8_ANTG1  \
count          891221.000000     891221.000000      ...        774706.000000
mean                2.874167          3.075121      ...             2.253330
std                 1.486731          1.353248      ...             0.972008
min                 1.000000          1.000000      ...             0.000000
25%                 2.000000          2.000000      ...             1.000000
50%                 3.000000          3.000000      ...             2.000000
```

```
75%                  4.000000            4.000000      ...            3.000000
max                  5.000000            5.000000      ...            4.000000

          PLZ8_ANTG2     PLZ8_ANTG3     PLZ8_ANTG4    PLZ8_BAUMAX  \
count  774706.000000  774706.000000  774706.000000  774706.000000
mean        2.801858       1.595426       0.699166       1.943913
std         0.920309       0.986736       0.727137       1.459654
min         0.000000       0.000000       0.000000       1.000000
25%         2.000000       1.000000       0.000000       1.000000
50%         3.000000       2.000000       1.000000       1.000000
75%         3.000000       2.000000       1.000000       3.000000
max         4.000000       3.000000       2.000000       5.000000

            PLZ8_HHZ       PLZ8_GBZ         ARBEIT    ORTSGR_KLS9  \
count  774706.000000  774706.000000  794005.000000  794005.000000
mean        3.612821       3.381087       3.167854       5.293002
std         0.973967       1.111598       1.002376       2.303739
min         1.000000       1.000000       1.000000       0.000000
25%         3.000000       3.000000       3.000000       4.000000
50%         4.000000       3.000000       3.000000       5.000000
75%         4.000000       4.000000       4.000000       7.000000
max         5.000000       5.000000       9.000000       9.000000

           RELAT_AB
count  794005.00000
mean        3.07222
std         1.36298
min         1.00000
25%         2.00000
50%         3.00000
75%         4.00000
max         9.00000

[8 rows x 81 columns]
```

In [6]:
```python
print('First few rows of Feature Summary file:')
feat_info.head(5)
```

```
First few rows of Feature Summary file:
```

Out[6]:

	attribute	information_level	type	missing_or_unknown
0	AGER_TYP	person	categorical	[-1,0]
1	ALTERSKATEGORIE_GROB	person	ordinal	[-1,0,9]
2	ANREDE_KZ	person	categorical	[-1,0]
3	CJT_GESAMTTYP	person	categorical	[0]
4	FINANZ_MINIMALIST	person	ordinal	[-1]

In [7]:
```python
print('Entire Feature Summary file:')
feat_info
```

Entire Feature Summary file:

Out[7]:

	attribute	information_level	type	missing_or_unknown
0	AGER_TYP	person	categorical	[-1,0]
1	ALTERSKATEGORIE_GROB	person	ordinal	[-1,0,9]
2	ANREDE_KZ	person	categorical	[-1,0]
3	CJT_GESAMTTYP	person	categorical	[0]
4	FINANZ_MINIMALIST	person	ordinal	[-1]
5	FINANZ_SPARER	person	ordinal	[-1]
6	FINANZ_VORSORGER	person	ordinal	[-1]
7	FINANZ_ANLEGER	person	ordinal	[-1]
8	FINANZ_UNAUFFAELLIGER	person	ordinal	[-1]
9	FINANZ_HAUSBAUER	person	ordinal	[-1]
10	FINANZTYP	person	categorical	[-1]
11	GEBURTSJAHR	person	numeric	[0]
12	GFK_URLAUBERTYP	person	categorical	[]
13	GREEN_AVANTGARDE	person	categorical	[]
14	HEALTH_TYP	person	ordinal	[-1,0]
15	LP_LEBENSPHASE_FEIN	person	mixed	[0]
16	LP_LEBENSPHASE_GROB	person	mixed	[0]
17	LP_FAMILIE_FEIN	person	categorical	[0]
18	LP_FAMILIE_GROB	person	categorical	[0]
19	LP_STATUS_FEIN	person	categorical	[0]
20	LP_STATUS_GROB	person	categorical	[0]
21	NATIONALITAET_KZ	person	categorical	[-1,0]
22	PRAEGENDE_JUGENDJAHRE	person	mixed	[-1,0]
23	RETOURTYP_BK_S	person	ordinal	[0]
24	SEMIO_SOZ	person	ordinal	[-1,9]
25	SEMIO_FAM	person	ordinal	[-1,9]
26	SEMIO_REL	person	ordinal	[-1,9]
27	SEMIO_MAT	person	ordinal	[-1,9]
28	SEMIO_VERT	person	ordinal	[-1,9]
29	SEMIO_LUST	person	ordinal	[-1,9]
..	...	...	...	...
55	OST_WEST_KZ	building	categorical	[-1]
56	WOHNLAGE	building	mixed	[-1]
57	CAMEO_DEUG_2015	microcell_rr4	categorical	[-1,X]
58	CAMEO_DEU_2015	microcell_rr4	categorical	[XX]
59	CAMEO_INTL_2015	microcell_rr4	mixed	[-1,XX]
60	KBA05_ANTG1	microcell_rr3	ordinal	[-1]
61	KBA05_ANTG2	microcell_rr3	ordinal	[-1]
62	KBA05_ANTG3	microcell_rr3	ordinal	[-1]
63	KBA05_ANTG4	microcell_rr3	ordinal	[-1]
64	KBA05_BAUMAX	microcell_rr3	mixed	[-1,0]
65	KBA05_GBZ	microcell_rr3	ordinal	[-1,0]
66	BALLRAUM	postcode	ordinal	[-1]
67	EWDICHTE	postcode	ordinal	[-1]

```
68          INNENSTADT         postcode   ordinal       [-1]
69   GEBAEUDETYP_RASTER      region_rr1   ordinal         []
70                  KKK      region_rr1   ordinal     [-1,0]
71           MOBI_REGIO      region_rr1   ordinal         []
72    ONLINE_AFFINITAET      region_rr1   ordinal         []
73            REGIOTYP       region_rr1   ordinal     [-1,0]
74     KBA13_ANZAHL_PKW  macrocell_plz8   numeric         []
75           PLZ8_ANTG1  macrocell_plz8   ordinal       [-1]
76           PLZ8_ANTG2  macrocell_plz8   ordinal       [-1]
77           PLZ8_ANTG3  macrocell_plz8   ordinal       [-1]
78           PLZ8_ANTG4  macrocell_plz8   ordinal       [-1]
79          PLZ8_BAUMAX  macrocell_plz8     mixed     [-1,0]
80             PLZ8_HHZ  macrocell_plz8   ordinal       [-1]
81             PLZ8_GBZ  macrocell_plz8   ordinal       [-1]
82               ARBEIT       community   ordinal     [-1,9]
83          ORTSGR_KLS9       community   ordinal     [-1,0]
84             RELAT_AB       community   ordinal     [-1,9]

[85 rows x 4 columns]
```

Tip: Add additional cells to keep everything in reasonably-sized chunks! Keyboard shortcut `esc --> a` (press escape to enter command mode, then press the 'A' key) adds a new cell before the active cell, and `esc --> b` adds a new cell after the active cell. If you need to convert an active cell to a markdown cell, use `esc --> m` and to convert to a code cell, use `esc --> y`.

1.1 Step 1: Preprocessing

1.1.1 Step 1.1: Assess Missing Data

The feature summary file contains a summary of properties for each demographics data column. You will use this file to help you make cleaning decisions during this stage of the project. First of all, you should assess the demographics data in terms of missing data. Pay attention to the following points as you perform your analysis, and take notes on what you observe. Make sure that you fill in the **Discussion** cell with your findings and decisions at the end of each step that has one!

Step 1.1.1: Convert Missing Value Codes to NaNs The fourth column of the feature attributes summary (loaded in above as `feat_info`) documents the codes from the data dictionary that indicate missing or unknown data. While the file encodes this as a list (e.g. `[-1,0]`), this will get read in as a string object. You'll need to do a little bit of parsing to make use of it to identify and clean the data. Convert data that matches a 'missing' or 'unknown' value code into a numpy NaN value. You might want to see how much data takes on a 'missing' or 'unknown' code, and how much data is naturally missing, as a point of interest.

As one more reminder, you are encouraged to add additional cells to break up your analysis into manageable chunks.

In [8]:
```
# Identify originally missing values per column in AZDIAS dataset.
originally_missing_columns = azdias.isnull().sum()
```

```python
print('Number of naturally missing values per AZDIAS dataset column:')
originally_missing_columns
```

```
Number of naturally missing values per AZDIAS dataset column:
```

Out[8]:

```
AGER_TYP                      0
ALTERSKATEGORIE_GROB          0
ANREDE_KZ                     0
CJT_GESAMTTYP              4854
FINANZ_MINIMALIST             0
FINANZ_SPARER                 0
FINANZ_VORSORGER              0
FINANZ_ANLEGER                0
FINANZ_UNAUFFAELLIGER         0
FINANZ_HAUSBAUER              0
FINANZTYP                     0
GEBURTSJAHR                   0
GFK_URLAUBERTYP            4854
GREEN_AVANTGARDE              0
HEALTH_TYP                    0
LP_LEBENSPHASE_FEIN        4854
LP_LEBENSPHASE_GROB        4854
LP_FAMILIE_FEIN            4854
LP_FAMILIE_GROB            4854
LP_STATUS_FEIN             4854
LP_STATUS_GROB             4854
NATIONALITAET_KZ              0
PRAEGENDE_JUGENDJAHRE         0
RETOURTYP_BK_S             4854
SEMIO_SOZ                     0
SEMIO_FAM                     0
SEMIO_REL                     0
SEMIO_MAT                     0
SEMIO_VERT                    0
SEMIO_LUST                    0
                          ...
OST_WEST_KZ               93148
WOHNLAGE                  93148
CAMEO_DEUG_2015           98979
CAMEO_DEU_2015            98979
CAMEO_INTL_2015           98979
KBA05_ANTG1              133324
KBA05_ANTG2              133324
KBA05_ANTG3              133324
KBA05_ANTG4              133324
KBA05_BAUMAX             133324
```

```
KBA05_GBZ              133324
BALLRAUM                93740
EWDICHTE                93740
INNENSTADT              93740
GEBAEUDETYP_RASTER      93155
KKK                    121196
MOBI_REGIO             133324
ONLINE_AFFINITAET        4854
REGIOTYP               121196
KBA13_ANZAHL_PKW       105800
PLZ8_ANTG1             116515
PLZ8_ANTG2             116515
PLZ8_ANTG3             116515
PLZ8_ANTG4             116515
PLZ8_BAUMAX            116515
PLZ8_HHZ               116515
PLZ8_GBZ               116515
ARBEIT                  97216
ORTSGR_KLS9             97216
RELAT_AB                97216
Length: 85, dtype: int64
```

In [9]:
```
# Identify missing or unknown data values and convert them to NaNs.
for attribute, missing_values_list in zip(feat_info['attribute'], feat_info['missing_or_
    # Strip string values of formatting elements and split values.
    missing_values_list = missing_values_list.strip('[]').split(',')

    for value in missing_values_list:
        # If value in missing values list is numeric, convert into integer type.
        try:
            value = int(value)
        except:
            continue
        # Convert respective AZDIAS value to NaN.
        azdias.loc[azdias[attribute] == value, attribute] = np.nan
```

In [10]:
```
# Print first few lines of AZDIAS dataset to verify that missing or unknown values are
azdias.head(5)
```

Out[10]:
```
   AGER_TYP  ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  \
0       NaN                   2.0        1.0            2.0
1       NaN                   1.0        2.0            5.0
2       NaN                   3.0        2.0            3.0
3       2.0                   4.0        2.0            2.0
4       NaN                   3.0        1.0            5.0

   FINANZ_MINIMALIST  FINANZ_SPARER  FINANZ_VORSORGER  FINANZ_ANLEGER  \
0                3.0            4.0               3.0             5.0
```

```
1                 1.0            5.0                2.0             5.0
2                 1.0            4.0                1.0             2.0
3                 4.0            2.0                5.0             2.0
4                 4.0            3.0                4.0             1.0

   FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER    ...     PLZ8_ANTG1  PLZ8_ANTG2  \
0                    5.0               3.0    ...            NaN         NaN
1                    4.0               5.0    ...            2.0         3.0
2                    3.0               5.0    ...            3.0         3.0
3                    1.0               2.0    ...            2.0         2.0
4                    3.0               2.0    ...            2.0         4.0

   PLZ8_ANTG3  PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  \
0         NaN         NaN          NaN       NaN       NaN     NaN
1         2.0         1.0          1.0       5.0       4.0     3.0
2         1.0         0.0          1.0       4.0       4.0     3.0
3         2.0         0.0          1.0       3.0       4.0     2.0
4         2.0         1.0          2.0       3.0       3.0     4.0

   ORTSGR_KLS9  RELAT_AB
0          NaN       NaN
1          5.0       4.0
2          5.0       2.0
3          3.0       3.0
4          6.0       5.0

[5 rows x 85 columns]
```

In [11]:
```python
# Print the total number of missing or unknown values.
print('Total number of missing or unknown values in AZDIAS dataset: ', azdias.isnull().
```

```
Total number of missing or unknown values in AZDIAS dataset:  8372810
```

Step 1.1.2: Assess Missing Data in Each Column How much missing data is present in each column? There are a few columns that are outliers in terms of the proportion of values that are missing. You will want to use matplotlib's `hist()` function to visualize the distribution of missing value counts to find these columns. Identify and document these columns. While some of these columns might have justifications for keeping or re-encoding the data, for this project you should just remove them from the dataframe. (Feel free to make remarks about these outlier columns in the discussion, however!)

For the remaining features, are there any patterns in which columns have, or share, missing data?

In [12]:
```python
# Perform an assessment of how much missing data there is in each column of the
# dataset.

#Compare number of naturally missing values in AZDIAS dataset to number of missing valu
```

```
updated_missing_columns = azdias.isnull().sum()

before = pd.Series(originally_missing_columns, name = 'original nr. of missing or unkno
after = pd.Series(updated_missing_columns, name = 'updated nr. of missing or unknown va

pd.concat([before, after], axis=1)
```

Out[12]:

```
                          original nr. of missing or unknown values  \
AGER_TYP                                                          0
ALTERSKATEGORIE_GROB                                              0
ANREDE_KZ                                                         0
CJT_GESAMTTYP                                                  4854
FINANZ_MINIMALIST                                                 0
FINANZ_SPARER                                                     0
FINANZ_VORSORGER                                                  0
FINANZ_ANLEGER                                                    0
FINANZ_UNAUFFAELLIGER                                             0
FINANZ_HAUSBAUER                                                  0
FINANZTYP                                                         0
GEBURTSJAHR                                                       0
GFK_URLAUBERTYP                                                4854
GREEN_AVANTGARDE                                                  0
HEALTH_TYP                                                        0
LP_LEBENSPHASE_FEIN                                            4854
LP_LEBENSPHASE_GROB                                            4854
LP_FAMILIE_FEIN                                                4854
LP_FAMILIE_GROB                                                4854
LP_STATUS_FEIN                                                 4854
LP_STATUS_GROB                                                 4854
NATIONALITAET_KZ                                                  0
PRAEGENDE_JUGENDJAHRE                                             0
RETOURTYP_BK_S                                                 4854
SEMIO_SOZ                                                         0
SEMIO_FAM                                                         0
SEMIO_REL                                                         0
SEMIO_MAT                                                         0
SEMIO_VERT                                                        0
SEMIO_LUST                                                        0
...                                                             ...
OST_WEST_KZ                                                   93148
WOHNLAGE                                                      93148
CAMEO_DEUG_2015                                               98979
CAMEO_DEU_2015                                                98979
CAMEO_INTL_2015                                               98979
KBA05_ANTG1                                                  133324
KBA05_ANTG2                                                  133324
KBA05_ANTG3                                                  133324
KBA05_ANTG4                                                  133324
```

KBA05_BAUMAX	133324
KBA05_GBZ	133324
BALLRAUM	93740
EWDICHTE	93740
INNENSTADT	93740
GEBAEUDETYP_RASTER	93155
KKK	121196
MOBI_REGIO	133324
ONLINE_AFFINITAET	4854
REGIOTYP	121196
KBA13_ANZAHL_PKW	105800
PLZ8_ANTG1	116515
PLZ8_ANTG2	116515
PLZ8_ANTG3	116515
PLZ8_ANTG4	116515
PLZ8_BAUMAX	116515
PLZ8_HHZ	116515
PLZ8_GBZ	116515
ARBEIT	97216
ORTSGR_KLS9	97216
RELAT_AB	97216

	updated nr. of missing or unknown values
AGER_TYP	685843
ALTERSKATEGORIE_GROB	2881
ANREDE_KZ	0
CJT_GESAMTTYP	4854
FINANZ_MINIMALIST	0
FINANZ_SPARER	0
FINANZ_VORSORGER	0
FINANZ_ANLEGER	0
FINANZ_UNAUFFAELLIGER	0
FINANZ_HAUSBAUER	0
FINANZTYP	0
GEBURTSJAHR	392318
GFK_URLAUBERTYP	4854
GREEN_AVANTGARDE	0
HEALTH_TYP	111196
LP_LEBENSPHASE_FEIN	97632
LP_LEBENSPHASE_GROB	94572
LP_FAMILIE_FEIN	77792
LP_FAMILIE_GROB	77792
LP_STATUS_FEIN	4854
LP_STATUS_GROB	4854
NATIONALITAET_KZ	108315
PRAEGENDE_JUGENDJAHRE	108164
RETOURTYP_BK_S	4854
SEMIO_SOZ	0

```
SEMIO_FAM                                 0
SEMIO_REL                                 0
SEMIO_MAT                                 0
SEMIO_VERT                                0
SEMIO_LUST                                0
...                                     ...
OST_WEST_KZ                           93148
WOHNLAGE                              93148
CAMEO_DEUG_2015                       98979
CAMEO_DEU_2015                        98979
CAMEO_INTL_2015                       98979
KBA05_ANTG1                          133324
KBA05_ANTG2                          133324
KBA05_ANTG3                          133324
KBA05_ANTG4                          133324
KBA05_BAUMAX                         476524
KBA05_GBZ                            133324
BALLRAUM                              93740
EWDICHTE                              93740
INNENSTADT                            93740
GEBAEUDETYP_RASTER                    93155
KKK                                  158064
MOBI_REGIO                           133324
ONLINE_AFFINITAET                      4854
REGIOTYP                             158064
KBA13_ANZAHL_PKW                     105800
PLZ8_ANTG1                           116515
PLZ8_ANTG2                           116515
PLZ8_ANTG3                           116515
PLZ8_ANTG4                           116515
PLZ8_BAUMAX                          116515
PLZ8_HHZ                             116515
PLZ8_GBZ                             116515
ARBEIT                                97375
ORTSGR_KLS9                           97274
RELAT_AB                              97375

[85 rows x 2 columns]
```

In [13]:
```
# Investigate patterns in the amount of missing data in each column.

# Calculate percentage of missing values per column
n_rows = azdias.shape[0] # total number of rows in dataset
percentage_missing_columns = (updated_missing_columns / n_rows) * 100
percentage_missing_columns = pd.Series(percentage_missing_columns, name = '%_values_mis

print('Updated nr. of missing or unknown values in %')
percentage_missing_columns
```

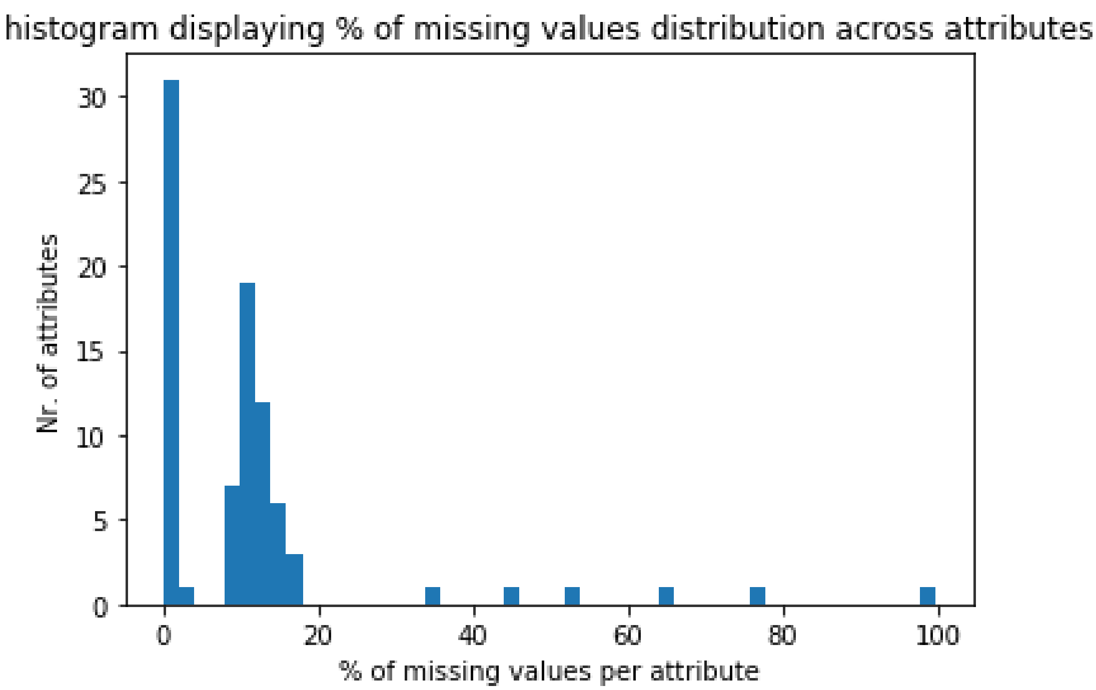
Updated nr. of missing or unknown values in %

Out[13]:

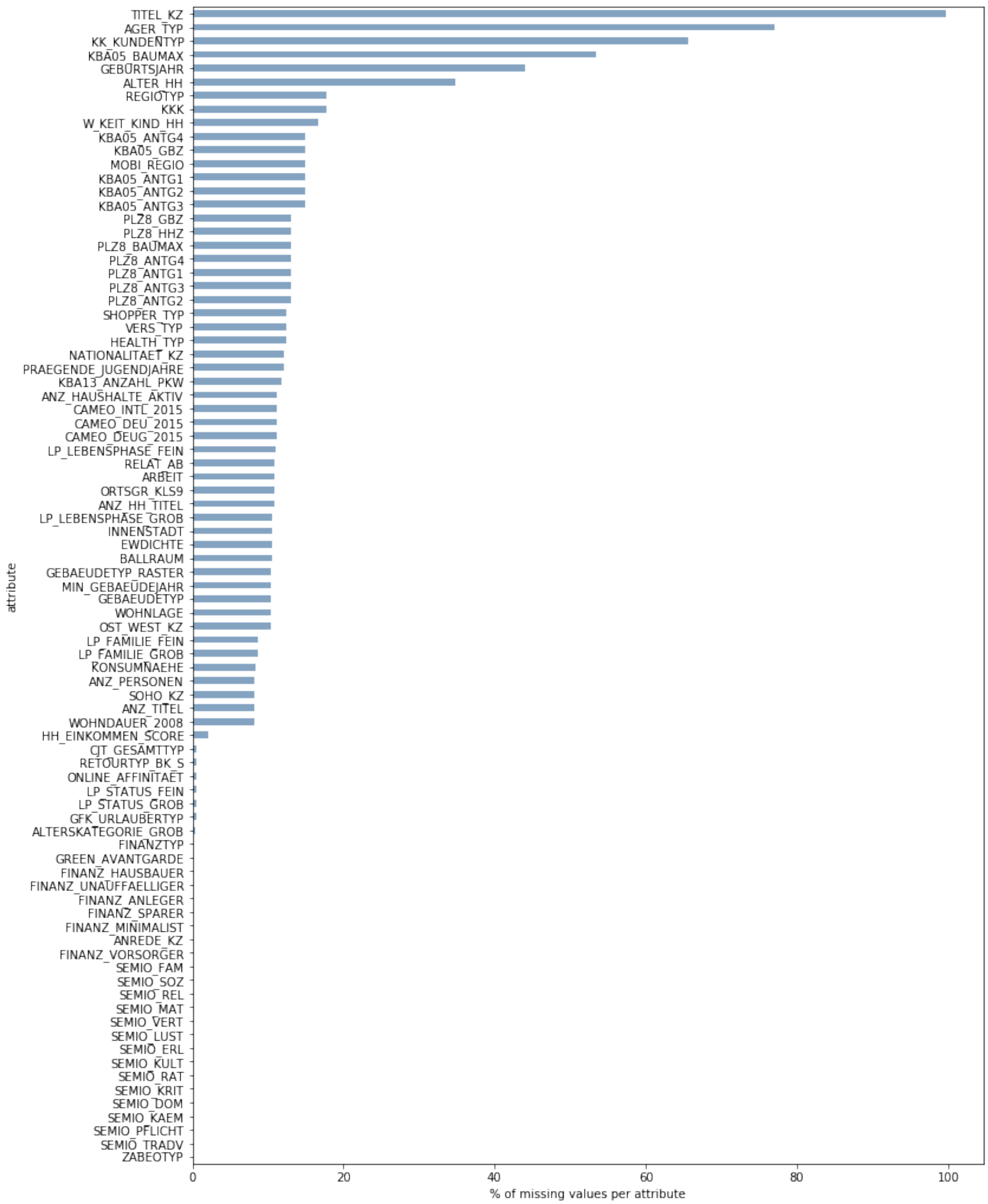
```
AGER_TYP                 76.955435
ALTERSKATEGORIE_GROB      0.323264
ANREDE_KZ                 0.000000
CJT_GESAMTTYP             0.544646
FINANZ_MINIMALIST         0.000000
FINANZ_SPARER             0.000000
FINANZ_VORSORGER          0.000000
FINANZ_ANLEGER            0.000000
FINANZ_UNAUFFAELLIGER     0.000000
FINANZ_HAUSBAUER          0.000000
FINANZTYP                 0.000000
GEBURTSJAHR              44.020282
GFK_URLAUBERTYP           0.544646
GREEN_AVANTGARDE          0.000000
HEALTH_TYP               12.476816
LP_LEBENSPHASE_FEIN      10.954859
LP_LEBENSPHASE_GROB      10.611509
LP_FAMILIE_FEIN           8.728699
LP_FAMILIE_GROB           8.728699
LP_STATUS_FEIN            0.544646
LP_STATUS_GROB            0.544646
NATIONALITAET_KZ         12.153551
PRAEGENDE_JUGENDJAHRE    12.136608
RETOURTYP_BK_S            0.544646
SEMIO_SOZ                 0.000000
SEMIO_FAM                 0.000000
SEMIO_REL                 0.000000
SEMIO_MAT                 0.000000
SEMIO_VERT                0.000000
SEMIO_LUST                0.000000
                            ...
OST_WEST_KZ              10.451729
WOHNLAGE                 10.451729
CAMEO_DEUG_2015          11.106000
CAMEO_DEU_2015           11.106000
CAMEO_INTL_2015          11.106000
KBA05_ANTG1              14.959701
KBA05_ANTG2              14.959701
KBA05_ANTG3              14.959701
KBA05_ANTG4              14.959701
KBA05_BAUMAX             53.468668
KBA05_GBZ                14.959701
BALLRAUM                 10.518154
EWDICHTE                 10.518154
INNENSTADT               10.518154
```

```
GEBAEUDETYP_RASTER        10.452514
KKK                       17.735668
MOBI_REGIO                14.959701
ONLINE_AFFINITAET          0.544646
REGIOTYP                  17.735668
KBA13_ANZAHL_PKW          11.871354
PLZ8_ANTG1                13.073637
PLZ8_ANTG2                13.073637
PLZ8_ANTG3                13.073637
PLZ8_ANTG4                13.073637
PLZ8_BAUMAX               13.073637
PLZ8_HHZ                  13.073637
PLZ8_GBZ                  13.073637
ARBEIT                    10.926022
ORTSGR_KLS9               10.914689
RELAT_AB                  10.926022
Name: %_values_missing, Length: 85, dtype: float64
```

In [14]:
```
# Visualize the distribution of missing value counts per column via a Histogram
plt.hist(percentage_missing_columns, bins = 50)
plt.xlabel('% of missing values per attribute')
plt.ylabel('Nr. of attributes')
plt.title('histogram displaying % of missing values distribution across attributes')
plt.show()
```



In [15]:
```
# Display data in order of descending percentage of missing values.
percentage_missing_columns.sort_values(ascending = True, inplace = True)
percentage_missing_columns.plot.barh(figsize=(12,18), color=(0.2, 0.4, 0.6, 0.6))
plt.xlabel('% of missing values per attribute')
plt.ylabel('attribute')
plt.show()
```



```
In [16]: # Remove the outlier columns from the dataset. (You'll perform other data
         # engineering tasks such as re-encoding and imputation later.)

         # Out of 85 attributes only 6 attributes have a % of missing values significantly highe
         #(i.e. more than 200000 missing values). Therefore, the following removes these 6 attri
         updated_missing_columns = pd.DataFrame(updated_missing_columns, columns = ['values_miss

         outliers = updated_missing_columns[updated_missing_columns['values_missing_columns'] >
         # print('Outlier attributes to be removed with corresponding % of missing values: \n',
         print('*' * 70)

         azdias_clean = azdias.drop(outliers.index, axis = 1)

         print("Outliers have been successfully removed from AZDIAS dataset.")
```

```
**********************************************************************
Outliers have been successfully removed from AZDIAS dataset.
```

```
In [17]: # Also remove the outlier features from the feat_info dataset.
         feat_info_clean = feat_info.set_index("attribute")
         for i, j in outliers.iterrows():
             feat_info_clean.drop(i, axis=0, inplace = True)
```

Discussion 1.1.2: Assess Missing Data in Each Column As can be seen from the bar chart, the majority of attributes in the AZDIAS dataset have either no or few missing values. Only 6 of the 85 attributes in the dataset have a percentage of missing values significantly higher than 18% of the total values. Therefore, I decided to consider these 6 attributes as outliers in terms of their number of missing values, and removed these from the dataset.

Step 1.1.3: Assess Missing Data in Each Row Now, you'll perform a similar assessment for the rows of the dataset. How much data is missing in each row? As with the columns, you should see some groups of points that have a very different numbers of missing values. Divide the data into two subsets: one for data points that are above some threshold for missing values, and a second subset for points below that threshold.

In order to know what to do with the outlier rows, we should see if the distribution of data values on columns that are not missing data (or are missing very little data) are similar or different between the two groups. Select at least five of these columns and compare the distribution of values. - You can use seaborn's `countplot()` function to create a bar chart of code frequencies and matplotlib's `subplot()` function to put bar charts for the two subplots side by side. - To reduce repeated code, you might want to write a function that can perform this comparison, taking as one of its arguments a column to be compared.

Depending on what you observe in your comparison, this will have implications on how you approach your conclusions later in the analysis. If the distributions of non-missing features look similar between the data with many missing values and the data with few or no missing values, then we could argue that simply dropping those points from the analysis won't present a major issue. On the other hand, if the data with many missing values looks very different from the data with few or no missing values, then we should make a note on those data as special. We'll revisit these data later on. **Either way, you should continue your analysis for now using just the subset of the data with few or no missing values.**

In [18]:
```
# How much data is missing in each row of the dataset?
missing_rows = pd.DataFrame(azdias_clean.isnull().sum(axis = 1), columns = ['values_mis
print('Number of values missing in first 20 rows of AZDIAS dataset:')
missing_rows.head(20)
```

```
Number of values missing in first 20 rows of AZDIAS dataset:
```

Out[18]:

	values_missing_rows
0	43
1	0
2	0
3	7
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	47
12	6
13	8
14	47
15	8
16	6
17	47
18	3
19	0

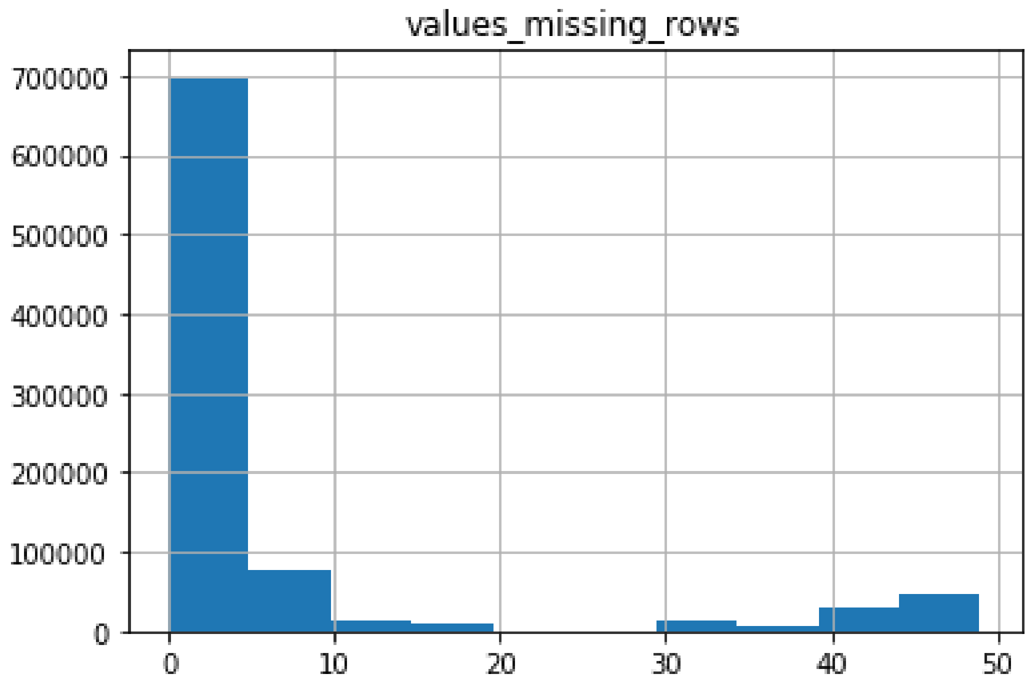
In [19]:
```
# Display summary statistics on number of missing values per row
missing_rows.describe()
```

Out[19]:

	values_missing_rows
count	891221.000000
mean	5.648638
std	13.234726
min	0.000000
25%	0.000000
50%	0.000000
75%	3.000000
max	49.000000

In [20]:
```
# Visualize the distribution of missing value counts per row via a Histogram.
missing_rows.hist()
```

Out[20]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f465ba832b0>]], dtype=objec
```



In [21]:
```
# Add number of missing values per row to AZDIAS dataset
azdias_joined = azdias_clean.join(missing_rows)

# Print first 5 rows in AZDIAS dataset to verify successful merger:
azdias_joined.head(5)
```

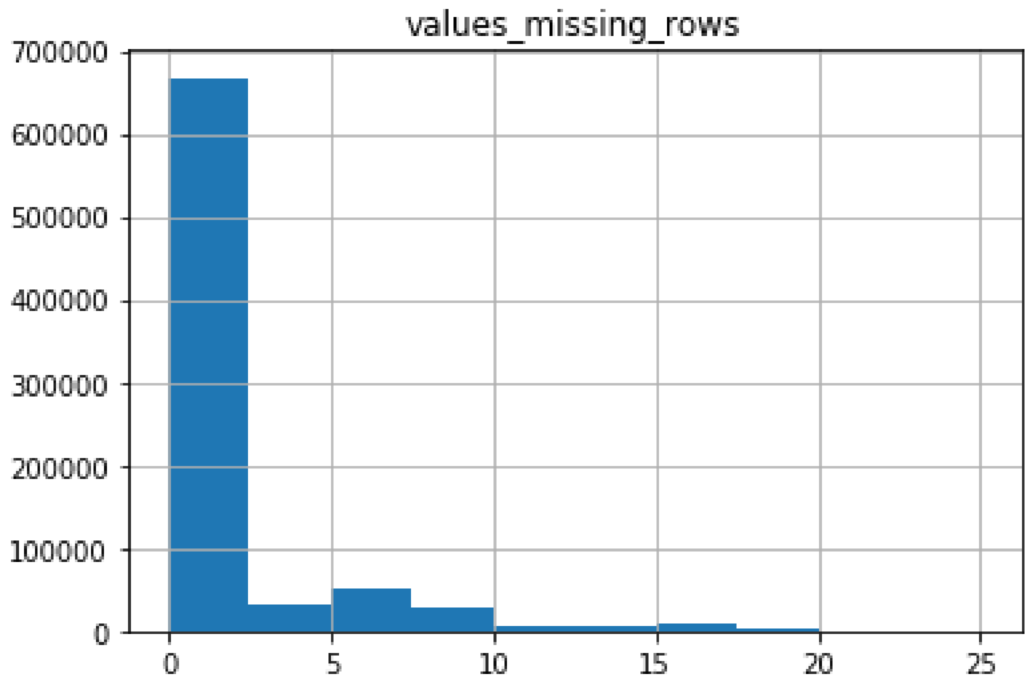
Out[21]:
```
   ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  FINANZ_MINIMALIST  \
0                   2.0        1.0            2.0                3.0
1                   1.0        2.0            5.0                1.0
2                   3.0        2.0            3.0                1.0
3                   4.0        2.0            2.0                4.0
4                   3.0        1.0            5.0                4.0

   FINANZ_SPARER  FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  \
0            4.0               3.0             5.0                    5.0
1            5.0               2.0             5.0                    4.0
2            4.0               1.0             2.0                    3.0
3            2.0               5.0             2.0                    1.0
4            3.0               4.0             1.0                    3.0

   FINANZ_HAUSBAUER  FINANZTYP       ...        PLZ8_ANTG2  PLZ8_ANTG3  \
0               3.0        4.0       ...               NaN         NaN
1               5.0        1.0       ...               3.0         2.0
2               5.0        1.0       ...               3.0         1.0
3               2.0        6.0       ...               2.0         2.0
```

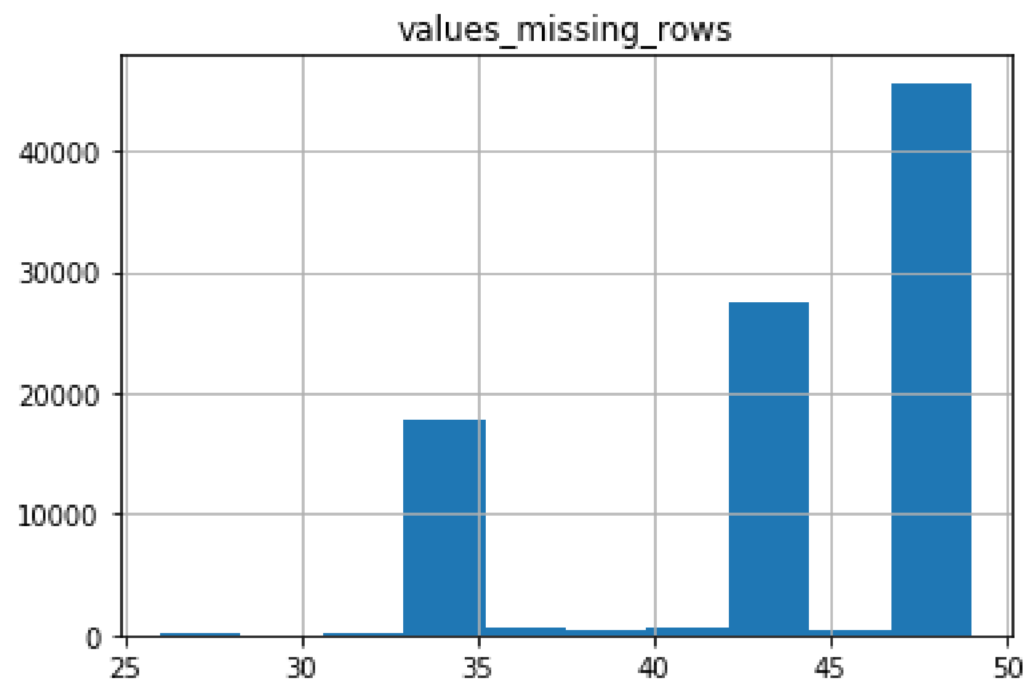
```
4              2.0        5.0         ...                 4.0         2.0

   PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  ORTSGR_KLS9  RELAT_AB  \
0         NaN          NaN       NaN       NaN     NaN          NaN       NaN
1         1.0          1.0       5.0       4.0     3.0          5.0       4.0
2         0.0          1.0       4.0       4.0     3.0          5.0       2.0
3         0.0          1.0       3.0       4.0     2.0          3.0       3.0
4         1.0          2.0       3.0       3.0     4.0          6.0       5.0

   values_missing_rows
0                   43
1                    0
2                    0
3                    7
4                    0

[5 rows x 80 columns]
```

In [22]:
```python
# Write code to divide the data into two subsets based on the number of missing
# values in each row.
azdias_fewer_missing = azdias_joined[azdias_joined['values_missing_rows'] <= 25]
azdias_more_missing = azdias_joined[azdias_joined['values_missing_rows'] > 25]
```

In [23]:
```python
# Verify successful split by plotting histograms of both datasets.
missing_rows_1 = pd.DataFrame(azdias_fewer_missing.isnull().sum(axis = 1), columns = ['
missing_rows_2 = pd.DataFrame(azdias_more_missing.isnull().sum(axis = 1), columns = ['v
missing_rows_1.hist()
missing_rows_2.hist()
```

Out[23]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f4659348048>]], dtype=objec
```





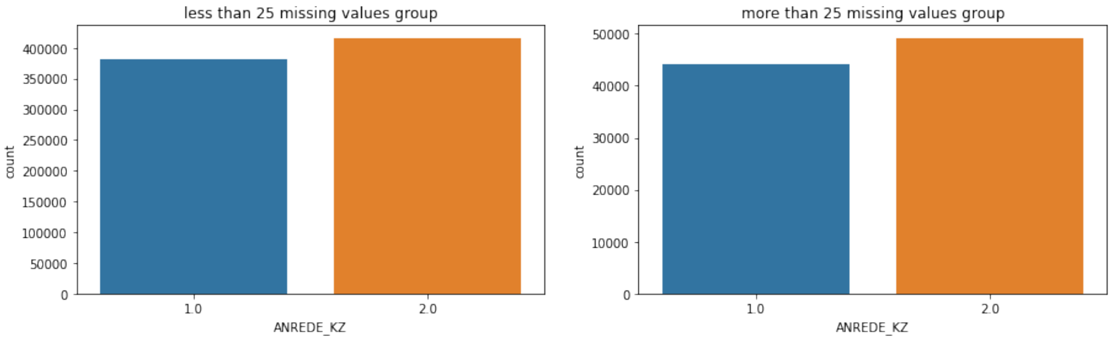

```
In [24]: # Compare the distribution of values for at least five columns where there are
         # no or few missing values, between the two subsets.
```

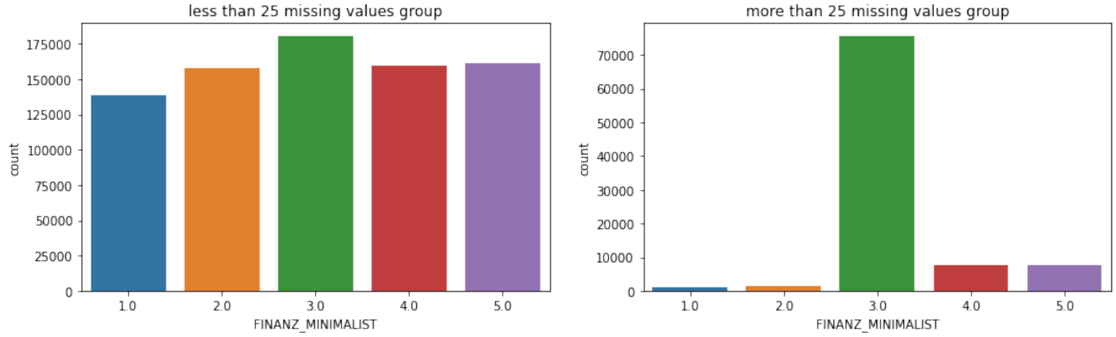
```
# Identify 5 columns with no missing values
zero_missing = updated_missing_columns[updated_missing_columns['values_missing_columns'
comparison_columns = zero_missing[:5]
print('5 Columns with no missing values for comparison: \n', comparison_columns)
```

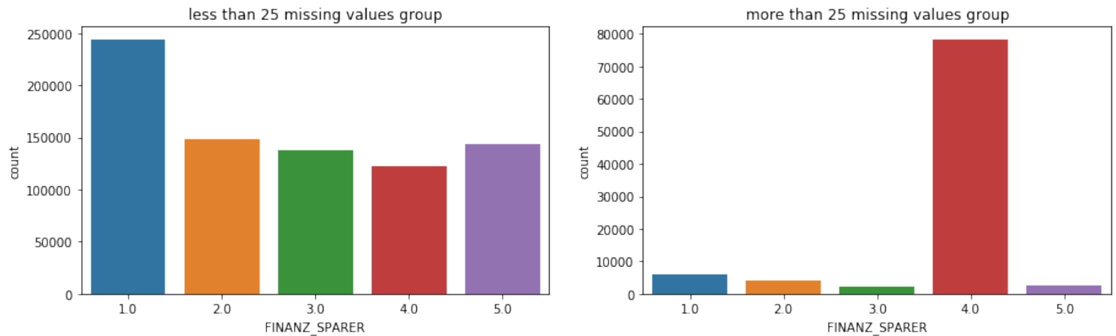
```
5 Columns with no missing values for comparison:
                     values_missing_columns
ANREDE_KZ                                0
FINANZ_MINIMALIST                        0
FINANZ_SPARER                            0
FINANZ_VORSORGER                         0
FINANZ_ANLEGER                           0
```

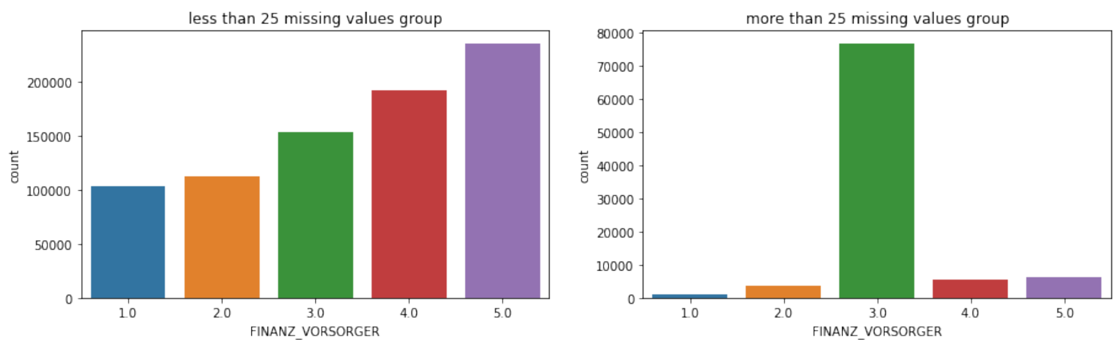
In [25]:
```
# Function that creates comparison subplots of both datasets per specified column.
def compare_values(set_1, set_2, column):
    fig, ax = plt.subplots(1, 2)
    fig.set_figwidth(15)
    ax[0].set_title('less than 25 missing values group')
    sns.countplot(set_1[column], ax=ax[0])
    ax[1].set_title('more than 25 missing values group')
    sns.countplot(set_2[column], ax=ax[1])
    plt.show()
```

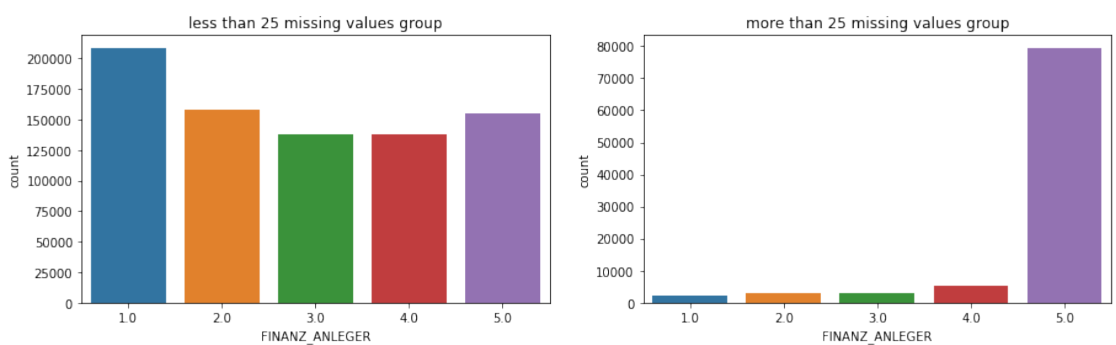
In [26]:
```
# Iterate through the dataframe containing the 5 columns with no missing values and plo
for i, j in comparison_columns.iterrows():
    compare_values(azdias_fewer_missing, azdias_more_missing, i)
```




less than 25 missing values group
count
175000
150000
125000
100000
75000
50000
25000
0
1.0
2.0
3.0
4.0
5.0
FINANZ_MINIMALIST
more than 25 missing values group
count
70000
60000
50000
40000
30000
20000
10000
0
1.0
2.0
3.0
4.0
5.0
FINANZ_MINIMALIST


less than 25 missing values group
count
250000
200000
150000
100000
50000
0
1.0
2.0
3.0
4.0
5.0
FINANZ_SPARER
more than 25 missing values group
count
80000
70000
60000
50000
40000
30000
20000
10000
0
1.0
2.0
3.0
4.0
5.0
FINANZ_SPARER


less than 25 missing values group
count
200000
150000
100000
50000
0
1.0
2.0
3.0
4.0
5.0
FINANZ_VORSORGER
more than 25 missing values group
count
80000
70000
60000
50000
40000
30000
20000
10000
0
1.0
2.0
3.0
4.0
5.0
FINANZ_VORSORGER


less than 25 missing values group
count
200000
175000
150000
125000
100000
75000
50000
25000
0
1.0
2.0
3.0
4.0
5.0
FINANZ_ANLEGER
more than 25 missing values group
count
80000
70000
60000
50000
40000
30000
20000
10000
0
1.0
2.0
3.0
4.0
5.0
FINANZ_ANLEGER

Discussion 1.1.3: Assess Missing Data in Each Row As can be discerned from the Histogram plot of the number of missing values per entitity, the majority of values has less than 6 missing values. Visually, two groups of entitites can be distinguished: entities with less than 25 missing values and entities with 25-50 missing values. Therefore, I decided to split the data accordingly.

From the comparison between the two groups along 5 columns with no missing values, it can be discerned that there are strong, sustematic differences between both subsets of the AZDIAS dataset in terms of missing values. It might therefore be wise to only utilize the dataset with entities containing less than 25 missing values.

```
In [27]: # Remove rows with more than 25 missing values from new AZDIAS dataset.
         azdias_new = azdias_fewer_missing
```

1.1.2 Step 1.2: Select and Re-Encode Features

Checking for missing data isn't the only way in which you can prepare a dataset for analysis. Since the unsupervised learning techniques to be used will only work on data that is encoded numerically, you need to make a few encoding changes or additional assumptions to be able to make progress. In addition, while almost all of the values in the dataset are encoded using numbers, not all of them represent numeric values. Check the third column of the feature summary (`feat_info`) for a summary of types of measurement. - For numeric and interval data, these features can be kept without changes. - Most of the variables in the dataset are ordinal in nature. While ordinal values may technically be non-linear in spacing, make the simplifying assumption that the ordinal variables can be treated as being interval in nature (that is, kept without any changes). - Special handling may be necessary for the remaining two variable types: categorical, and 'mixed'.

In the first two parts of this sub-step, you will perform an investigation of the categorical and mixed-type features and make a decision on each of them, whether you will keep, drop, or re-encode each. Then, in the last part, you will create a new data frame with only the selected and engineered columns.

Data wrangling is often the trickiest part of the data analysis process, and there's a lot of it to be done here. But stick with it: once you're done with this step, you'll be ready to get to the machine learning parts of the project!

```
In [28]: # How many features are there of each data type?
         features_type_count = feat_info_clean.type.value_counts()

         print('Number of features of each data type:')
         features_type_count
```

```
Number of features of each data type:
```

```
Out[28]: ordinal        49
         categorical    18
         mixed           6
         numeric         6
         Name: type, dtype: int64
```

Step 1.2.1: Re-Encode Categorical Features For categorical data, you would ordinarily need to encode the levels as dummy variables. Depending on the number of categories, perform one of the following: - For binary (two-level) categoricals that take numeric values, you can keep them without needing to do anything. - There is one binary variable that takes on non-numeric values. For this one, you need to re-encode the values as numbers or create a dummy variable. - For multi-level categoricals (three or more values), you can choose to encode the values using multiple dummy variables (e.g. via OneHotEncoder), or (to keep things straightforward) just drop them from the analysis. As always, document your choices in the Discussion section.

In [29]:
```
# Assess categorical variables: which are binary, which are multi-level, and
# which one needs to be re-encoded?

# Display categorical variables in features summary dataset.
categorical_variables = feat_info_clean[feat_info_clean.type == 'categorical']
categorical_variables
```

Out[29]:
```
                  information_level         type missing_or_unknown
attribute
ANREDE_KZ                    person  categorical             [-1,0]
CJT_GESAMTTYP                person  categorical                [0]
FINANZTYP                    person  categorical               [-1]
GFK_URLAUBERTYP              person  categorical                 []
GREEN_AVANTGARDE             person  categorical                 []
LP_FAMILIE_FEIN              person  categorical                [0]
LP_FAMILIE_GROB              person  categorical                [0]
LP_STATUS_FEIN               person  categorical                [0]
LP_STATUS_GROB               person  categorical                [0]
NATIONALITAET_KZ             person  categorical             [-1,0]
SHOPPER_TYP                  person  categorical               [-1]
SOHO_KZ                      person  categorical               [-1]
VERS_TYP                     person  categorical               [-1]
ZABEOTYP                     person  categorical             [-1,9]
GEBAEUDETYP                building  categorical             [-1,0]
OST_WEST_KZ                building  categorical               [-1]
CAMEO_DEUG_2015        microcell_rr4  categorical             [-1,X]
CAMEO_DEU_2015         microcell_rr4  categorical               [XX]
```

In [30]:
```
# Iterate through categorical variables and classify whether binary or multi-level
binary_categoricals = []
multi_level_categoricals = []

for attribute in categorical_variables.index:
    if azdias_new[attribute].nunique() == 2:
        binary_categoricals.append(attribute)
    else:
        multi_level_categoricals.append(attribute)
```

In [31]:
```
# Display Categories.
print('Binary Categoricals: ', binary_categoricals)
```

```
    print('*' * 125)
    print('Multi-Level Categoricals: ', multi_level_categoricals)
```

```
Binary Categoricals:  ['ANREDE_KZ', 'GREEN_AVANTGARDE', 'SOHO_KZ', 'VERS_TYP', 'OST_WEST_KZ']
*********************************************************************************************
Multi-Level Categoricals:  ['CJT_GESAMTTYP', 'FINANZTYP', 'GFK_URLAUBERTYP', 'LP_FAMILIE_FEIN',
```

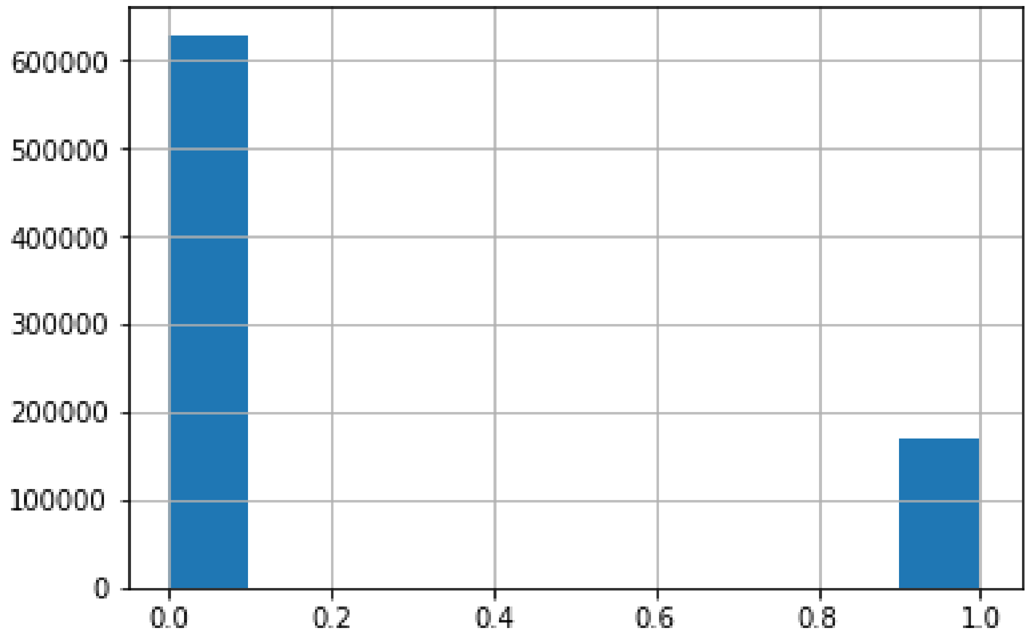
In [32]:
```
# Iterate through categorical variables list and check whether numeric or non-numeric
binary_categoricals_non_numeric = []

for attribute in binary_categoricals:
    values = azdias_new[attribute].unique()

    for value in values:
        if not np.isreal(value):
            binary_categoricals_non_numeric.append(attribute)
            break

print('Binary Non-Numeric Categoricals: ', binary_categoricals_non_numeric)
```

```
Binary Non-Numeric Categoricals:  ['OST_WEST_KZ']
```

In [33]:
```
# Convert binary non-numeric variable into binary numeric variable.
var_old = azdias_new[binary_categoricals_non_numeric[0]].unique()
print('The two non-numeric variables are: ', var_old)

# Replace non-numeric with binary variables in current AZDIAS dataset.
azdias_new["OST_WEST_KZ"].replace({'W' : 0.0,'O' : 1.0}, inplace = True)
# Set type to integer.
azdias_new["OST_WEST_KZ"].astype('int', inplace=True)

# Verify success of conversion.
var_new = azdias_new[binary_categoricals_non_numeric[0]].unique()
print('The two converted variables are: ', var_new)
azdias_new["OST_WEST_KZ"].hist()
```

```
The two non-numeric variables are:  ['W' 'O']
The two converted variables are:  [ 0.  1.]


/opt/conda/lib/python3.6/site-packages/pandas/core/generic.py:5890: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  self._update_inplace(new_data)
```

Out[33]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f465ba50438>`




```
In [34]: # One-hot encode multi-level cotegoricals.
         azdias_reencoded = pd.get_dummies(data = azdias_new, columns = multi_level_categoricals
```

Discussion 1.2.1: Re-Encode Categorical Features My analysis identified numeric binary, non-numeric binary and multi-level categorical variables. Moving forward, the numeric binary variables do not need reencoding. I identified one non-numeric binary variable that I converted into a numeric binary variable. Finally, I decided to one-hot encode the multi-level cattegorial values in the AZDIAS dataset.

Step 1.2.2: Engineer Mixed-Type Features There are a handful of features that are marked as "mixed" in the feature summary that require special treatment in order to be included in the analysis. There are two in particular that deserve attention; the handling of the rest are up to your own choices: - "PRAEGENDE_JUGENDJAHRE" combines information on three dimensions: generation by decade, movement (mainstream vs. avantgarde), and nation (east vs. west). While there aren't enough levels to disentangle east from west, you should create two new variables to capture the other two dimensions: an interval-type variable for decade, and a binary variable for movement. - "CAMEO_INTL_2015" combines information on two axes: wealth and life stage. Break up the two-digit codes by their 'tens'-place and 'ones'-place digits into two new ordinal variables (which, for the purposes of this project, is equivalent to just treating them as their raw numeric values). - If you decide to keep or engineer new features around the other mixed-type features, make sure you note your steps in the Discussion section.

Be sure to check `Data_Dictionary.md` for the details needed to finish these tasks.

```
In [35]: # Get overview of all mixed features in features summary dataset.
         mixed_features = feat_info_clean[feat_info_clean.type == 'mixed']
```

```
mixed_features
```

```
Out[35]:                        information_level   type missing_or_unknown
         attribute
         LP_LEBENSPHASE_FEIN               person  mixed                [0]
         LP_LEBENSPHASE_GROB               person  mixed                [0]
         PRAEGENDE_JUGENDJAHRE             person  mixed             [-1,0]
         WOHNLAGE                        building  mixed               [-1]
         CAMEO_INTL_2015           microcell_rr4  mixed            [-1,XX]
         PLZ8_BAUMAX              macrocell_plz8  mixed             [-1,0]
```

```
In [36]: # Investigate "PRAEGENDE_JUGENDJAHRE" and engineer two new variables.

         ''' INFORMATION FROM Data_Dictionary.md
         ### 1.18. PRAEGENDE_JUGENDJAHRE
         Dominating movement of person's youth (avantgarde vs. mainstream; east vs. west)
         - -1: unknown
         -  0: unknown
         -  1: 40s - war years (Mainstream, E+W)
         -  2: 40s - reconstruction years (Avantgarde, E+W)
         -  3: 50s - economic miracle (Mainstream, E+W)
         -  4: 50s - milk bar / Individualisation (Avantgarde, E+W)
         -  5: 60s - economic miracle (Mainstream, E+W)
         -  6: 60s - generation 68 / student protestors (Avantgarde, W)
         -  7: 60s - opponents to the building of the Wall (Avantgarde, E)
         -  8: 70s - family orientation (Mainstream, E+W)
         -  9: 70s - peace movement (Avantgarde, E+W)
         - 10: 80s - Generation Golf (Mainstream, W)
         - 11: 80s - ecological awareness (Avantgarde, W)
         - 12: 80s - FDJ / communist party youth organisation (Mainstream, E)
         - 13: 80s - Swords into ploughshares (Avantgarde, E)
         - 14: 90s - digital media kids (Mainstream, E+W)
         - 15: 90s - ecological awareness (Avantgarde, E+W)
         '''

         # Display first 10 results for PRAEGENDE_JUGENDJAHRE in current AZDIAS dataset.
         azdias_reencoded.PRAEGENDE_JUGENDJAHRE.head(10)
```

```
Out[36]: 1    14.0
         2    15.0
         3     8.0
         4     8.0
         5     3.0
         6    10.0
         7     8.0
         8    11.0
         9    15.0
```

```
10     3.0
Name: PRAEGENDE_JUGENDJAHRE, dtype: float64
```

In [37]:
```
# PRAEGENDE_JUGENDJAHRE: Engineer two new variables.

# Create dictionaries for the variables generation by decade and movement (mainstream v
# contain the original variables (1-15) as keys, and the respective decate (40-90) / a
# "Aventgarde" respectively.

generation_dict =   {
                    1: 40.0,
                    2: 40.0,
                    3: 50.0,
                    4: 50.0,
                    5: 60.0,
                    6: 60.0,
                    7: 60.0,
                    8: 70.0,
                    9: 70.0,
                    10: 80.0,
                    11: 80.0,
                    12: 80.0,
                    13: 80.0,
                    14: 90.0,
                    15: 90.0
                    }

movement_dict =     {
                    1: 0.0,
                    2: 1.0,
                    3: 0.0,
                    4: 1.0,
                    5: 0.0,
                    6: 1.0,
                    7: 1.0,
                    8: 0.0,
                    9: 1.0,
                    10: 0.0,
                    11: 1.0,
                    12: 0.0,
                    13: 1.0,
                    14: 0.0,
                    15: 1.0,
                    }
```

In [38]:
```
# Replace the original PRAEGENDE_JUGENDJAHRE from the current AZDIAS dataset with the t

# Create two new columns as copies of original column.
```

```
azdias_reencoded["DECADE"] = azdias_reencoded.PRAEGENDE_JUGENDJAHRE
azdias_reencoded["MOVEMENT"] = azdias_reencoded.PRAEGENDE_JUGENDJAHRE

# Drop the original column.
azdias_reencoded.drop("PRAEGENDE_JUGENDJAHRE", axis=1, inplace=True)

# Replace values in two new columns with dictionary values.
azdias_reencoded["DECADE"].replace(generation_dict, inplace=True)
azdias_reencoded["MOVEMENT"].replace(movement_dict, inplace=True)
```

In [39]:
```
# Verify successful replacement.
print("First 10 rows in DECADE column: ")
display(azdias_reencoded["DECADE"].head(10))

print("First 10 rows in MOVEMENT column: ")
display(azdias_reencoded["MOVEMENT"].head(10))
```

```
First 10 rows in DECADE column:


1     90.0
2     90.0
3     70.0
4     70.0
5     50.0
6     80.0
7     70.0
8     80.0
9     90.0
10    50.0
Name: DECADE, dtype: float64


First 10 rows in MOVEMENT column:


1     0.0
2     1.0
3     0.0
4     0.0
5     0.0
6     0.0
7     0.0
8     1.0
9     1.0
10    0.0
Name: MOVEMENT, dtype: float64
```

In [40]:
```
# Investigate "CAMEO_INTL_2015" and engineer two new variables.

''' INFORMATION FROM Data_Dictionary.md
### 4.3. CAMEO_INTL_2015
German CAMEO: Wealth / Life Stage Typology, mapped to international code
- -1: unknown
- 11: Wealthy Households - Pre-Family Couples & Singles
- 12: Wealthy Households - Young Couples With Children
- 13: Wealthy Households - Families With School Age Children
- 14: Wealthy Households - Older Families &  Mature Couples
- 15: Wealthy Households - Elders In Retirement
- 21: Prosperous Households - Pre-Family Couples & Singles
- 22: Prosperous Households - Young Couples With Children
- 23: Prosperous Households - Families With School Age Children
- 24: Prosperous Households - Older Families & Mature Couples
- 25: Prosperous Households - Elders In Retirement
- 31: Comfortable Households - Pre-Family Couples & Singles
- 32: Comfortable Households - Young Couples With Children
- 33: Comfortable Households - Families With School Age Children
- 34: Comfortable Households - Older Families & Mature Couples
- 35: Comfortable Households - Elders In Retirement
- 41: Less Affluent Households - Pre-Family Couples & Singles
- 42: Less Affluent Households - Young Couples With Children
- 43: Less Affluent Households - Families With School Age Children
- 44: Less Affluent Households - Older Families & Mature Couples
- 45: Less Affluent Households - Elders In Retirement
- 51: Poorer Households - Pre-Family Couples & Singles
- 52: Poorer Households - Young Couples With Children
- 53: Poorer Households - Families With School Age Children
- 54: Poorer Households - Older Families & Mature Couples
- 55: Poorer Households - Elders In Retirement
- XX: unknown
'''
# Display first 10 results for CAMEO_INTL_2015 in current AZDIAS dataset.
azdias_reencoded.CAMEO_INTL_2015.head(10)
```

Out[40]:
```
1      51
2      24
3      12
4      43
5      54
6      22
7      14
8      13
9      15
10     51
Name: CAMEO_INTL_2015, dtype: object
```

In [41]:
```
# CAMEO_INTL_2015: Engineer two new variables.
```

```python
# Create dictionaries for the variables wealth and lifestage. Both dictionaries
# contain the original numerical variables as keys, and the respective numerical value
# statuses / life stages respectively. Unknown values listed in Data_Dictionary.md can
# listed as NaNs)

wealth_dict =       {
                    '11': 1.0,
                    '12': 1.0,
                    '13': 1.0,
                    '14': 1.0,
                    '15': 1.0,
                    '21': 2.0,
                    '22': 2.0,
                    '23': 2.0,
                    '24': 2.0,
                    '25': 2.0,
                    '31': 3.0,
                    '32': 3.0,
                    '33': 3.0,
                    '34': 3.0,
                    '35': 3.0,
                    '41': 4.0,
                    '42': 4.0,
                    '43': 4.0,
                    '44': 4.0,
                    '45': 4.0,
                    '51': 5.0,
                    '52': 5.0,
                    '53': 5.0,
                    '54': 5.0,
                    '55': 5.0
                    }

lifestage_dict =    {
                    '11': 1.0,
                    '12': 2.0,
                    '13': 3.0,
                    '14': 4.0,
                    '15': 5.0,
                    '21': 1.0,
                    '22': 2.0,
                    '23': 3.0,
                    '24': 4.0,
                    '25': 5.0,
                    '31': 1.0,
                    '32': 2.0,
                    '33': 3.0,
```

```
                                '34': 4.0,
                                '35': 5.0,
                                '41': 1.0,
                                '42': 2.0,
                                '43': 3.0,
                                '44': 4.0,
                                '45': 5.0,
                                '51': 1.0,
                                '52': 2.0,
                                '53': 3.0,
                                '54': 4.0,
                                '55': 5.0
                                }
```

```
In [42]: # Create two new columns as copies of original column.
         azdias_reencoded["WEALTH"] = azdias_reencoded.CAMEO_INTL_2015
         azdias_reencoded["LIFESTAGE"] = azdias_reencoded.CAMEO_INTL_2015

         # Drop the original column.
         azdias_reencoded.drop("CAMEO_INTL_2015", axis = 1, inplace = True)

         # Replace values in two new columns with dictionary values.
         azdias_reencoded["WEALTH"].replace(wealth_dict, inplace = True)
         azdias_reencoded["LIFESTAGE"].replace(lifestage_dict, inplace = True)
```

```
In [43]: # Verify successful replacement.
         print("First 10 rows in WEALTH column: ")
         display(azdias_reencoded["WEALTH"].head(10))

         print("First 10 rows in LIFESTAGE column: ")
         display(azdias_reencoded["LIFESTAGE"].head(10))
```

```
First 10 rows in WEALTH column:


1     5
2     2
3     1
4     4
5     5
6     2
7     1
8     1
9     1
10    5
Name: WEALTH, dtype: object


First 10 rows in LIFESTAGE column:
```

```
1     1
2     4
3     2
4     3
5     4
6     2
7     4
8     3
9     5
10    1
Name: LIFESTAGE, dtype: object
```

```
In [44]: # Review information on four remaining mixed-type features.
         '''INFORMATION FROM Data_Dictionary.md
         ### 1.11. LP_LEBENSPHASE_FEIN
         Life stage, fine scale
         -  1: single low-income earners of younger age
         -  2: single low-income earners of middle age
         -  3: single average earners of younger age
         -  4: single average earners of middle age
         -  5: single low-income earners of advanced age
         -  6: single low-income earners at retirement age
         -  7: single average earners of advanced age
         -  8: single average earners at retirement age
         -  9: single independent persons
         - 10: wealthy single homeowners
         - 11: single homeowners of advanced age
         - 12: single homeowners at retirement age
         - 13: single top earners of higher age
         - 14: low-income and average earner couples of younger age
         - 15: low-income earner couples of higher age
         - 16: average earner couples of higher age
         - 17: independent couples
         - 18: wealthy homeowner couples of younger age
         - 19: homeowner couples of higher age
         - 20: top earner couples of higher age
         - 21: single parent low-income earners
         - 22: single parent average earners
         - 23: single parent high-income earners
         - 24: low-income earner families
         - 25: average earner families
         - 26: independent families
         - 27: homeowner families
         - 28: top earner families
         - 29: low-income earners of younger age from multiperson households
         - 30: average earners of younger age from multiperson households
         - 31: low-income earners of higher age from multiperson households
```

```
- 32: average earners of higher age from multiperson households
- 33: independent persons of younger age from multiperson households
- 34: homeowners of younger age from multiperson households
- 35: top earners of younger age from multiperson households
- 36: independent persons of higher age from multiperson households
- 37: homeowners of advanced age from multiperson households
- 38: homeowners at retirement age from multiperson households
- 39: top earners of middle age from multiperson households
- 40: top earners at retirement age from multiperson households
'''

'''INFORMATION FROM Data_Dictionary.md
### 1.12. LP_LEBENSPHASE_GROB
Life stage, rough scale
-  1: single low-income and average earners of younger age
-  2: single low-income and average earners of higher age
-  3: single high-income earners
-  4: single low-income and average-earner couples
-  5: single high-income earner couples
-  6: single parents
-  7: single low-income and average earner families
-  8: high-income earner families
-  9: average earners of younger age from multiperson households
- 10: low-income and average earners of higher age from multiperson households
- 11: high-income earners of younger age from multiperson households
- 12: high-income earners of higher age from multiperson households
'''

'''INFORMATION FROM Data_Dictionary.md
### 3.7. WOHNLAGE
Neighborhood quality (or rural flag)
- -1: unknown
-  0: no score calculated
-  1: very good neighborhood
-  2: good neighborhood
-  3: average neighborhood
-  4: poor neighborhood
-  5: very poor neighborhood
-  7: rural neighborhood
-  8: new building in rural neighborhood
'''

'''INFORMATION FROM Data_Dictionary.md
### 8.6. PLZ8_BAUMAX
Most common building type within the PLZ8 region
- -1: unknown
-  0: unknown
-  1: mainly 1-2 family homes
```

```
    - 2: mainly 3-5 family homes
    - 3: mainly 6-10 family homes
    - 4: mainly 10+ family homes
    - 5: mainly business buildings
    '''
```

```
Out[44]: 'INFORMATION FROM Data_Dictionary.md\n### 8.6. PLZ8_BAUMAX\nMost common building type w
```

```
In [45]: # Drop LP_LEBENSPHASE_FEIN and LP_LEBENSPHASE_GROB from dataset.
         azdias_reencoded.drop(["LP_LEBENSPHASE_FEIN", "LP_LEBENSPHASE_GROB"], axis = 1, inplace
```

Discussion 1.2.2: Engineer Mixed-Type Features As specified in the instructions, for both PRAEGENDE_JUGENDJAHRE and CAMEO_INTL_2015 I created two new columns respectively.

For PRAEGENDE_JUGENDJAHRE, I ignored the nation (east vs. west) as there were insufficient levels. However, I created new columns for the other two multi-level variables: DECADE and MOVEMENT. For decade, I implemented the corresponding decade values (40-90) representing the decades 1940ies to 1990ies. For the movement variables (Aventgarde vs. Mainstream) i used the binary variabled 0.0 and 1.0.

For CAMEO_INTL_2015, I created new columns for WEALTH and LIFESTAGE by representing each of their five categorical sub-variables through numerical values (1.0-5.0).

These were my engineering steps: 1. Look up column information in Data_Dictionary.md 2. Display first few rows of multi-variable column 3. Using the information from (1), create dictionaries with the original column values as key, and each corresponding relevant sub-variable (or numeric representation thereof) as value.
4. Create two new columns as copies of original column respectively. 5. Drop the original column. 6. Replace values in the new columns with the corresponding dictionary values. 7. Print the first 10 rows of each column to verify successful conversion.

Besides the above two mixed-type features, I identified four additional mixed-type featured through the cleaned feat_info datatset: LP_LEBENSPHASE_FEIN, LP_LEBENSPHASE_GROB, WOHNLAGE, and PLZ8_BAUMAX.

LP_LEBENSPHASE_FEIN: Even though this column contains potentially useful multi-level information for our alaysis, from the structure of the variables (some contain features such as age lifestage, age, household status) whereas others contain only a sub-set of this and other information) it is not possible to convert the information into discrete columns, without incorporating many unknown values. Furthermore, the included information (age, income, household type, lifestage) are already included in other more discrete features in this dataset. Therefore, I decided to drop this column.

LP_LEBENSPHASE_GROB: This mixed-type feature contains aggregate information contained in LP_LEBENSPHASE_FEIN. Because this feature also included mixed combinations of some of the variables age, lifestage, income and numer of people in household, it was also not reasonable to convert the information into distinct columns. As with LP_LEBENSPHASE_FEIN, I decided to also drop this column as it replicated information already included in other more discrete features in this dataset.

WOHNLAGE: Even though I am not certain how much value this feature will contribute to our analysis, I decided to keep the information within the dataset. A review of the mixed_type values revealed that it would not be practical to convert these into separate columns.

PLZ8_BAUMAX: Same as for WOHNLAGE.

Step 1.2.3: Complete Feature Selection In order to finish this step up, you need to make sure that your data frame now only has the columns that you want to keep. To summarize, the dataframe should consist of the following: - All numeric, interval, and ordinal type columns from the original dataset. - Binary categorical features (all numerically-encoded). - Engineered features from other multi-level categorical features and mixed features.

Make sure that for any new columns that you have engineered, that you've excluded the original columns from the final dataset. Otherwise, their values will interfere with the analysis later on the project. For example, you should not keep "PRAEGENDE_JUGENDJAHRE", since its values won't be useful for the algorithm: only the values derived from it in the engineered features you created should be retained. As a reminder, your data should only be from **the subset with few or no missing values**.

```
In [46]: # Ensure that the dataframe only contains the columns that should be passed to the algo
         removed_features = ["PRAEGENDE_JUGENDJAHRE", "CAMEO_INTL_2015", "LP_LEBENSPHASE_FEIN",

         for feature in removed_features:
             if feature in azdias_reencoded:
                 print("Error! this feature should be removed: ", feature)
             else:
                 print("Success! this feature is removed: ", feature)
```

```
Success! this feature is removed:  PRAEGENDE_JUGENDJAHRE
Success! this feature is removed:  CAMEO_INTL_2015
Success! this feature is removed:  LP_LEBENSPHASE_FEIN
Success! this feature is removed:  LP_LEBENSPHASE_GROB
```

1.1.3 Step 1.3: Create a Cleaning Function

Even though you've finished cleaning up the general population demographics data, it's important to look ahead to the future and realize that you'll need to perform the same cleaning steps on the customer demographics data. In this substep, complete the function below to execute the main feature selection, encoding, and re-engineering steps you performed above. Then, when it comes to looking at the customer data in Step 3, you can just run this function on that DataFrame to get the trimmed dataset in a single step.

```
In [47]: def clean_data(df, feat_info):
             """
             Perform feature trimming, re-encoding, and engineering for demographics
             data

             INPUT: Demographics DataFrame
             OUTPUT: Trimmed and cleaned demographics DataFrame
             """
             ### Convert missing value codes into NaNs. ######################################

             # Identify missing or unknown data values and convert them to NaNs.
             for attribute, missing_values_list in zip(feat_info['attribute'], feat_info['missin
                 # Strip string values of formatting elements and split values.
```

```
    missing_values_list = missing_values_list.strip('[]').split(',')

    for value in missing_values_list:
        # If value in missing values list is numeric, convert into integer type.
        try:
            value = int(value)
        except:
            continue
        # Convert respective AZDIAS value to NaN.
        df.loc[azdias[attribute] == value, attribute] = np.nan

# Remove "X", "XX" values
for col in df.columns:
    df = df.replace({col: ['XX', 'X']}, np.nan)

### Remove selected columns and rows. #############################################

# Remove columns with outlier features.
outliers = ["AGER_TYP", "GEBURTSJAHR", "TITEL_KZ", "ALTER_HH", "KK_KUNDENTYP", "KBA
df_cleaned = df.drop(outliers, axis = 1, inplace = True)

# Get number of missing values per row.
missing_rows = df_cleaned[df_cleaned.isnull().sum(axis=1)]

# Only use subset with few missing rows.
df_new = df_cleaned[df_cleaned[missing_rows <= 25]]
df_new = df_new.fillna(df_new.mode().iloc[0])

### Select, re-encode, and engineer column values. ################################

# Assess categorical variables in features summary dataset.
categorical_variables = feat_info_clean[feat_info_clean.type == 'categorical']

# Iterate through categorical variables and classify whether binary or multi-level
binary_categoricals = []
multi_level_categoricals = []

for attribute in categorical_variables.index:
    if df_new[attribute].nunique() == 2:
        binary_categoricals.append(attribute)
    else:
        multi_level_categoricals.append(attribute)

# Iterate through categorical variables list and check whether numeric or non-numer
binary_categoricals_non_numeric = []

for attribute in binary_categoricals:
    values = df_new[attribute].unique()
```

```
    for value in values:
        if not np.isreal(value):
            binary_categoricals_non_numeric.append(attribute)
            break

# Replace non-numeric with binary variables in current AZDIAS dataset.
df_new["OST_WEST_KZ"].replace({'W' : 0.0,'O' : 1.0}, inplace = True)
# Set type to integer.
df_new["OST_WEST_KZ"].astype('int', inplace=True)

# One-hot encode multi-level cotegoricals.
df_reencoded = pd.get_dummies(data = df_new, columns = multi_level_categoricals)

# PRAEGENDE_JUGENDJAHRE: Engineer two new variables.
# Create dictionaries for the variables generation by decade and movement (mainstre
# contain the original variables (1-15) as keys, and the respective decate (40-90)
# "Aventgarde" respectively.

generation_dict =   {
                    1: 40.0,
                    2: 40.0,
                    3: 50.0,
                    4: 50.0,
                    5: 60.0,
                    6: 60.0,
                    7: 60.0,
                    8: 70.0,
                    9: 70.0,
                    10: 80.0,
                    11: 80.0,
                    12: 80.0,
                    13: 80.0,
                    14: 90.0,
                    15: 90.0
                    }

movement_dict =     {
                    1: 0.0,
                    2: 1.0,
                    3: 0.0,
                    4: 1.0,
                    5: 0.0,
                    6: 1.0,
                    7: 1.0,
                    8: 0.0,
                    9: 1.0,
                    10: 0.0,
```

```
                    11: 1.0,
                    12: 0.0,
                    13: 1.0,
                    14: 0.0,
                    15: 1.0,
                    }

# Replace the original PRAEGENDE_JUGENDJAHRE from the current AZDIAS dataset with t
# Create two new columns as copies of original column.
df_reencoded["DECADE"] = df_reencoded.PRAEGENDE_JUGENDJAHRE
df_reencoded["MOVEMENT"] = df_reencoded.PRAEGENDE_JUGENDJAHRE

# Drop the original column.
df_reencoded.drop("PRAEGENDE_JUGENDJAHRE", axis = 1, inplace = True)

# Replace values in two new columns with dictionary values.
df_reencoded["DECADE"].replace(generation_dict, inplace = True)
df_reencoded["MOVEMENT"].replace(movement_dict, inplace = True)

# CAMEO_INTL_2015: Engineer two new variables.
# Create dictionaries for the variables wealth and lifestage. Both dictionaries
# contain the original numerical variables as keys, and the respective numerical va
# statuses / life stages respectively. Unknown values listed in Data_Dictionary.md
# listed as NaNs)

wealth_dict =       {
                    '11': 1.0,
                    '12': 1.0,
                    '13': 1.0,
                    '14': 1.0,
                    '15': 1.0,
                    '21': 2.0,
                    '22': 2.0,
                    '23': 2.0,
                    '24': 2.0,
                    '25': 2.0,
                    '31': 3.0,
                    '32': 3.0,
                    '33': 3.0,
                    '34': 3.0,
                    '35': 3.0,
                    '41': 4.0,
                    '42': 4.0,
                    '43': 4.0,
                    '44': 4.0,
                    '45': 4.0,
                    '51': 5.0,
                    '52': 5.0,
```

```
                    '53': 5.0,
                    '54': 5.0,
                    '55': 5.0
                    }

lifestage_dict =    {
                    '11': 1.0,
                    '12': 2.0,
                    '13': 3.0,
                    '14': 4.0,
                    '15': 5.0,
                    '21': 1.0,
                    '22': 2.0,
                    '23': 3.0,
                    '24': 4.0,
                    '25': 5.0,
                    '31': 1.0,
                    '32': 2.0,
                    '33': 3.0,
                    '34': 4.0,
                    '35': 5.0,
                    '41': 1.0,
                    '42': 2.0,
                    '43': 3.0,
                    '44': 4.0,
                    '45': 5.0,
                    '51': 1.0,
                    '52': 2.0,
                    '53': 3.0,
                    '54': 4.0,
                    '55': 5.0
                    }
# Create two new columns as copies of original column.
df_reencoded["WEALTH"] = df_reencoded.CAMEO_INTL_2015
df_reencoded["LIFESTAGE"] = df_reencoded.CAMEO_INTL_2015

# Drop the original column.
df_reencoded.drop("CAMEO_INTL_2015", axis = 1, inplace = True)

# Replace values in two new columns with dictionary values.
df_reencoded["WEALTH"].replace(wealth_dict, inplace = True)
df_reencoded["LIFESTAGE"].replace(lifestage_dict, inplace = True)

# Drop LP_LEBENSPHASE_FEIN and LP_LEBENSPHASE_GROB from dataset.
df_reencoded.drop(["LP_LEBENSPHASE_FEIN", "LP_LEBENSPHASE_GROB"], axis = 1, inplace

### Return the cleaned dataframe. ##################################################
return df_reencoded
```

1.2 Step 2: Feature Transformation

1.2.1 Step 2.1: Apply Feature Scaling

Before we apply dimensionality reduction techniques to the data, we need to perform feature scaling so that the principal component vectors are not influenced by the natural differences in scale for features. Starting from this part of the project, you'll want to keep an eye on the API reference page for sklearn to help you navigate to all of the classes and functions that you'll need. In this substep, you'll need to check the following:

- sklearn requires that data not have missing values in order for its estimators to work properly. So, before applying the scaler to your data, make sure that you've cleaned the DataFrame of the remaining missing values. This can be as simple as just removing all data points with missing data, or applying an Imputer to replace all missing values. You might also try a more complicated procedure where you temporarily remove missing values in order to compute the scaling parameters before re-introducing those missing values and applying imputation. Think about how much missing data you have and what possible effects each approach might have on your analysis, and justify your decision in the discussion section below.
- For the actual scaling function, a StandardScaler instance is suggested, scaling each feature to mean 0 and standard deviation 1.
- For these classes, you can make use of the .fit_transform() method to both fit a procedure to the data as well as apply the transformation to the data at the same time. Don't forget to keep the fit sklearn objects handy, since you'll be applying them to the customer demographics data towards the end of the project.

```
In [48]: # Check the dataset for NaN values.
         print("Number of NaN values in current AZDIAS dataset: ", azdias_reencoded.isnull().sum
```

```
Number of NaN values in current AZDIAS dataset:  783801
```

```
In [49]: # Replace NaN values.
         azdias_reencoded = azdias_reencoded.fillna(azdias_reencoded.mode().iloc[0])
```

```
In [50]: # Check if "X" or "XX" values remaining in dataset.
         print("'XX' values remaining: ", "XX" in azdias_reencoded.values)
         print("'X' values remaining: ", "X" in azdias_reencoded.values)
```

```
'XX' values remaining:  True
'X' values remaining:  False
```

```
In [51]: # Remove rows with "XX" values from dataset.
         index_XX = azdias_reencoded[azdias_reencoded.values == 'XX'].index
         azdias_reencoded.drop(index_XX, inplace = True)
```

```
In [52]: # Verify successful replacement.
         print("Number of NaN values in current AZDIAS dataset after replacement: ", azdias_reen
         print("'XX' values remaining: ", "XX" in azdias_reencoded.values)
```

```
Number of NaN values in current AZDIAS dataset after replacement:  0
'XX' values remaining:  False
```

In [53]:
```
# Apply feature scaling to the general population demographics data.
scaler = StandardScaler()
azdias_scaled = pd.DataFrame(scaler.fit_transform(azdias_reencoded), columns = azdias_r
```

In [54]:
```
# Verify successful Feature Scaling.
azdias_scaled.head(10)
```

Out[54]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	FINANZ_MINIMALIST	FINANZ_SPARER	\
0	-1.766703	0.957969	-1.494352	1.537947	
1	0.200460	0.957969	-1.494352	0.864596	
2	1.184042	0.957969	0.683483	-0.482104	
3	0.200460	-1.043875	0.683483	0.191246	
4	-1.766703	0.957969	-0.042462	-1.155455	
5	-0.783122	0.957969	-1.494352	1.537947	
6	-1.766703	-1.043875	-0.042462	0.191246	
7	0.200460	-1.043875	0.683483	0.864596	
8	0.200460	0.957969	-0.768407	0.864596	
9	0.200460	0.957969	-0.768407	-0.482104	

	FINANZ_VORSORGER	FINANZ_ANLEGER	FINANZ_UNAUFFAELLIGER	FINANZ_HAUSBAUER	\
0	-1.040838	1.466047	0.958994	1.338929	
1	-1.767140	-0.570960	0.244408	1.338929	
2	1.138069	-0.570960	-1.184764	-0.791753	
3	0.411767	-1.249963	0.244408	-0.791753	
4	1.138069	-0.570960	-0.470178	1.338929	
5	-1.767140	1.466047	0.958994	-0.081526	
6	0.411767	-1.249963	0.244408	-0.791753	
7	-1.040838	0.787045	-0.470178	-0.791753	
8	-1.040838	0.108042	1.673581	0.628702	
9	1.138069	0.108042	-1.184764	1.338929	

	GREEN_AVANTGARDE	HEALTH_TYP	...	CAMEO_DEU_2015_9A	\
0	-0.530280	1.010162	...	-0.162591	
1	1.885797	1.010162	...	-0.162591	
2	-0.530280	-0.311822	...	-0.162591	
3	-0.530280	1.010162	...	-0.162591	
4	-0.530280	1.010162	...	-0.162591	
5	-0.530280	-0.311822	...	-0.162591	
6	-0.530280	-1.633805	...	-0.162591	
7	1.885797	1.010162	...	-0.162591	
8	1.885797	-0.311822	...	-0.162591	
9	-0.530280	-0.311822	...	-0.162591	

	CAMEO_DEU_2015_9B	CAMEO_DEU_2015_9C	CAMEO_DEU_2015_9D	CAMEO_DEU_2015_9E	\

```
0          -0.189596          -0.179837          -0.192827          -0.089783
1          -0.189596          -0.179837          -0.192827          -0.089783
2          -0.189596          -0.179837          -0.192827          -0.089783
3          -0.189596          -0.179837          -0.192827          -0.089783
4          -0.189596          -0.179837          -0.192827          -0.089783
5          -0.189596          -0.179837          -0.192827          -0.089783
6          -0.189596          -0.179837          -0.192827          -0.089783
7          -0.189596          -0.179837          -0.192827          -0.089783
8          -0.189596          -0.179837          -0.192827          -0.089783
9          -0.189596          -0.179837           5.186003          -0.089783

   CAMEO_DEU_2015_XX    DECADE  MOVEMENT    WEALTH  LIFESTAGE
0                0.0  1.098185 -0.530280  1.176424  -1.249781
1                0.0  1.098185  1.885797 -0.869133   0.766518
2                0.0 -0.267616 -0.530280 -1.550985  -0.577681
3                0.0 -0.267616 -0.530280  0.494572   0.094419
4                0.0 -1.633417 -0.530280  1.176424   0.766518
5                0.0  0.415285 -0.530280 -0.869133  -0.577681
6                0.0 -0.267616 -0.530280 -1.550985   0.766518
7                0.0  0.415285  1.885797 -1.550985   0.094419
8                0.0  1.098185  1.885797 -1.550985   1.438618
9                0.0 -1.633417 -0.530280  1.176424  -1.249781

[10 rows x 197 columns]
```

1.2.2 Discussion 2.1: Apply Feature Scaling

Before starting the Feature Scaling process, I first had to replace the remaining 783801 NaN values in the dataset. Similarly, I had to remove rows containing "X" or "XX" values. After this, feature scaling could be easily applied.

1.2.3 Step 2.2: Perform Dimensionality Reduction

On your scaled data, you are now ready to apply dimensionality reduction techniques.

- Use sklearn's PCA class to apply principal component analysis on the data, thus finding the vectors of maximal variance in the data. To start, you should not set any parameters (so all components are computed) or set a number of components that is at least half the number of features (so there's enough features to see the general trend in variability).
- Check out the ratio of variance explained by each principal component as well as the cumulative variance explained. Try plotting the cumulative or sequential values using matplotlib's `plot()` function. Based on what you find, select a value for the number of transformed features you'll retain for the clustering part of the project.
- Once you've made a choice for the number of components to keep, make sure you re-fit a PCA instance to perform the decided-on transformation.

```
In [55]: # Apply PCA to the data.
         pca_1 = PCA()
         pca_1.fit_transform(azdias_scaled)
```

```
Out[55]: array([[ 4.66554941e+00, -3.94513471e+00, -3.12461900e+00, ...,
                 -2.00583785e-17,  5.58381264e-18,  3.12116467e-17],
               [ -5.17830272e-01, -5.78149033e-01, -3.15998401e+00, ...,
                 -7.19626059e-17,  7.98714993e-18,  1.47199986e-17],
               [ -4.90933422e+00,  1.64843977e+00, -1.48434187e+00, ...,
                 -3.86231951e-17,  1.13540667e-18,  1.06030980e-17],
               ...,
               [ -1.10731033e+00, -4.00180485e+00, -3.49878183e+00, ...,
                 -9.36139520e-19,  4.59248490e-18, -9.23447819e-19],
               [  6.47415931e+00, -4.40115562e+00,  2.84175712e+00, ...,
                  3.74126165e-18,  1.65870842e-18,  3.72455662e-19],
               [  4.94287894e-01,  2.53050540e+00,  1.85694365e+00, ...,
                  4.23037932e-18, -1.00112218e-18, -1.58511602e-18]])
```

```
In [56]: # Investigate the variance accounted for by each principal component.
         # Define functions that visualize PCA results
         def pca_results(full_dataset, pca):
             '''
             Create a DataFrame of the PCA results
             Includes dimension feature weights and explained variance
             Visualizes the PCA results
             '''

             # Dimension indexing
             dimensions = dimensions = ['Dimension {}'.format(i) for i in range(1,len(pca.compon

             # PCA components
             components = pd.DataFrame(np.round(pca.components_, 4), columns = full_dataset.keys
             components.index = dimensions

             # PCA explained variance
             ratios = pca.explained_variance_ratio_.reshape(len(pca.components_), 1)
             variance_ratios = pd.DataFrame(np.round(ratios, 4), columns = ['Explained Variance'
             variance_ratios.index = dimensions

             # Create a bar plot visualization
             fig, ax = plt.subplots(figsize = (14,8))

             # Plot the feature weights as a function of the components
             components.plot(ax = ax, kind = 'bar');
             ax.set_ylabel("Feature Weights")
             ax.set_xticklabels(dimensions, rotation=0)


             # Display the explained variance ratios
             for i, ev in enumerate(pca.explained_variance_ratio_):
                 ax.text(i-0.40, ax.get_ylim()[1] + 0.05, "Explained Variance\n          %.4f"%(
```

```python
        # Return a concatenated DataFrame
        return pd.concat([variance_ratios, components], axis = 1)
```

In [57]:
```python
pca_results(azdias_scaled, pca_1)
```

Out[57]:

	Explained Variance	ALTERSKATEGORIE_GROB	ANREDE_KZ	\
Dimension 1	0.0797	-0.0801	0.0142	
Dimension 2	0.0569	0.2341	0.0376	
Dimension 3	0.0352	0.0118	-0.3403	
Dimension 4	0.0281	-0.0299	0.1022	
Dimension 5	0.0205	0.0488	-0.0219	
Dimension 6	0.0164	0.0142	-0.0010	
Dimension 7	0.0153	0.0272	0.0105	
Dimension 8	0.0137	0.0322	-0.0220	
Dimension 9	0.0130	-0.0416	0.0164	
Dimension 10	0.0126	-0.0210	0.0230	
Dimension 11	0.0120	-0.0697	0.0246	
Dimension 12	0.0119	-0.0181	0.0087	
Dimension 13	0.0116	0.0125	-0.0422	
Dimension 14	0.0115	0.0153	-0.0056	
Dimension 15	0.0112	0.0086	-0.0075	
Dimension 16	0.0110	0.0493	0.0034	
Dimension 17	0.0110	-0.0441	0.0050	
Dimension 18	0.0107	0.0325	-0.0258	
Dimension 19	0.0105	-0.0505	0.0177	
Dimension 20	0.0103	-0.0158	-0.0134	
Dimension 21	0.0102	0.0056	-0.0128	
Dimension 22	0.0099	0.0393	-0.0211	
Dimension 23	0.0092	-0.0161	-0.0416	
Dimension 24	0.0092	-0.0655	0.0393	
Dimension 25	0.0089	-0.0411	0.0809	
Dimension 26	0.0085	-0.0106	0.0041	
Dimension 27	0.0083	-0.0836	0.0296	
Dimension 28	0.0080	0.0106	-0.0254	
Dimension 29	0.0079	0.0799	-0.0121	
Dimension 30	0.0073	-0.0449	0.0187	
...	...	...	...	
Dimension 168	0.0000	-0.0140	-0.0134	
Dimension 169	0.0000	0.0000	-0.0000	
Dimension 170	0.0000	0.0000	0.0000	
Dimension 171	0.0000	0.0000	-0.0000	
Dimension 172	0.0000	0.0000	-0.0000	
Dimension 173	0.0000	-0.0000	-0.0000	
Dimension 174	0.0000	0.0000	0.0000	
Dimension 175	0.0000	0.0000	0.0000	
Dimension 176	0.0000	-0.0000	-0.0000	
Dimension 177	0.0000	0.0000	-0.0000	
Dimension 178	0.0000	0.0000	-0.0000	

Dimension 179	0.0000	0.0000	-0.0000
Dimension 180	0.0000	-0.0000	0.0000
Dimension 181	0.0000	0.0000	0.0000
Dimension 182	0.0000	-0.0000	-0.0000
Dimension 183	0.0000	0.0000	-0.0000
Dimension 184	0.0000	-0.0000	0.0000
Dimension 185	0.0000	-0.0000	-0.0000
Dimension 186	0.0000	-0.0000	0.0000
Dimension 187	0.0000	0.0000	0.0000
Dimension 188	0.0000	-0.0000	-0.0000
Dimension 189	0.0000	-0.0000	0.0000
Dimension 190	0.0000	-0.0000	0.0000
Dimension 191	0.0000	0.0000	0.0000
Dimension 192	0.0000	-0.0000	0.0000
Dimension 193	0.0000	-0.0000	0.0000
Dimension 194	0.0000	-0.0000	-0.0000
Dimension 195	0.0000	-0.0000	0.0000
Dimension 196	0.0000	0.0000	0.0000
Dimension 197	0.0000	-0.0000	-0.0000

	FINANZ_MINIMALIST	FINANZ_SPARER	FINANZ_VORSORGER	\
Dimension 1	-0.1871	0.1107	-0.0805	
Dimension 2	0.0925	-0.2298	0.2204	
Dimension 3	0.1306	-0.0539	0.0438	
Dimension 4	-0.0057	0.0150	-0.0207	
Dimension 5	0.0524	-0.0285	0.0276	
Dimension 6	0.1267	-0.0145	-0.0355	
Dimension 7	0.0042	-0.0118	-0.0203	
Dimension 8	-0.0257	-0.0235	0.0368	
Dimension 9	0.0523	-0.0141	-0.0989	
Dimension 10	-0.0818	0.0277	-0.0041	
Dimension 11	0.0754	-0.0325	0.0327	
Dimension 12	0.0070	0.0031	0.0069	
Dimension 13	-0.0012	-0.0053	0.0447	
Dimension 14	-0.0274	0.0202	-0.0017	
Dimension 15	0.0121	-0.0019	-0.0265	
Dimension 16	-0.0301	0.0445	-0.0502	
Dimension 17	0.0315	-0.0494	0.0547	
Dimension 18	-0.0060	0.0094	0.0122	
Dimension 19	0.0249	-0.0567	0.0021	
Dimension 20	-0.0384	0.0094	0.0244	
Dimension 21	-0.0439	0.0233	0.0296	
Dimension 22	-0.0089	-0.0021	0.0368	
Dimension 23	-0.0408	0.0047	-0.0049	
Dimension 24	0.0109	-0.0859	0.1508	
Dimension 25	-0.0014	-0.0689	0.0426	
Dimension 26	0.0497	0.0072	-0.0118	
Dimension 27	0.0522	-0.0207	0.0213	

Dimension 28	-0.0176	0.0341	0.0178
Dimension 29	0.0140	0.0290	-0.1474
Dimension 30	0.0342	-0.0723	0.0579
...	...	...	...
Dimension 168	0.0019	-0.0031	0.0035
Dimension 169	-0.0000	-0.0000	-0.0000
Dimension 170	0.0000	0.0000	0.0000
Dimension 171	0.0000	0.0000	-0.0000
Dimension 172	-0.0000	-0.0000	-0.0000
Dimension 173	0.0000	0.0000	0.0000
Dimension 174	-0.0000	-0.0000	-0.0000
Dimension 175	0.0000	-0.0000	-0.0000
Dimension 176	-0.0000	-0.0000	-0.0000
Dimension 177	-0.0000	-0.0000	0.0000
Dimension 178	-0.0000	-0.0000	-0.0000
Dimension 179	-0.0000	0.0000	0.0000
Dimension 180	-0.0000	0.0000	0.0000
Dimension 181	0.0000	0.0000	-0.0000
Dimension 182	-0.0000	-0.0000	0.0000
Dimension 183	-0.0000	-0.0000	-0.0000
Dimension 184	-0.0000	-0.0000	0.0000
Dimension 185	0.0000	0.0000	0.0000
Dimension 186	-0.0000	-0.0000	0.0000
Dimension 187	-0.0000	-0.0000	0.0000
Dimension 188	0.0000	0.0000	-0.0000
Dimension 189	0.0000	0.0000	0.0000
Dimension 190	0.0000	-0.0000	-0.0000
Dimension 191	0.0000	-0.0000	0.0000
Dimension 192	0.0000	0.0000	-0.0000
Dimension 193	0.0000	-0.0000	0.0000
Dimension 194	-0.0000	-0.0000	0.0000
Dimension 195	0.0000	0.0000	-0.0000
Dimension 196	-0.0000	0.0000	0.0000
Dimension 197	0.0000	0.0000	0.0000

	FINANZ_ANLEGER	FINANZ_UNAUFFAELLIGER	FINANZ_HAUSBAUER	\
Dimension 1	0.0506	0.0489	0.1474	
Dimension 2	-0.2069	-0.2169	0.0833	
Dimension 3	-0.1535	-0.0428	-0.0648	
Dimension 4	-0.0491	0.0613	-0.0261	
Dimension 5	0.0233	-0.0943	-0.0452	
Dimension 6	-0.0457	0.0384	-0.2190	
Dimension 7	-0.0134	0.0155	-0.0246	
Dimension 8	-0.0647	0.0715	0.0125	
Dimension 9	0.0346	-0.0858	-0.0195	
Dimension 10	0.0550	0.0291	0.1369	
Dimension 11	-0.0581	-0.0215	-0.0644	
Dimension 12	-0.0285	0.0487	-0.0344	

Dimension 13	-0.0087	-0.0651	-0.0115
Dimension 14	0.0198	0.0312	0.0048
Dimension 15	0.0351	-0.0648	-0.0244
Dimension 16	0.0474	0.0962	0.0182
Dimension 17	-0.0580	-0.0654	-0.0253
Dimension 18	0.0032	0.0039	0.0179
Dimension 19	-0.0487	-0.0889	-0.0213
Dimension 20	-0.0267	0.0573	0.0513
Dimension 21	0.0387	-0.0004	0.0642
Dimension 22	0.0119	0.0463	0.0218
Dimension 23	0.0357	-0.0589	0.0042
Dimension 24	-0.0652	-0.0603	0.0498
Dimension 25	-0.0596	-0.0438	0.1008
Dimension 26	0.0823	-0.0795	0.0476
Dimension 27	0.0447	-0.0116	-0.0514
Dimension 28	-0.0147	0.0191	-0.0285
Dimension 29	-0.0310	0.1110	-0.0515
Dimension 30	-0.0763	-0.0607	0.0320
...	...	...	...
Dimension 168	-0.0033	0.0004	-0.0015
Dimension 169	-0.0000	0.0000	0.0000
Dimension 170	-0.0000	-0.0000	-0.0000
Dimension 171	0.0000	-0.0000	-0.0000
Dimension 172	0.0000	0.0000	-0.0000
Dimension 173	-0.0000	-0.0000	0.0000
Dimension 174	-0.0000	-0.0000	-0.0000
Dimension 175	-0.0000	0.0000	-0.0000
Dimension 176	0.0000	-0.0000	-0.0000
Dimension 177	-0.0000	0.0000	-0.0000
Dimension 178	-0.0000	-0.0000	-0.0000
Dimension 179	-0.0000	-0.0000	-0.0000
Dimension 180	0.0000	-0.0000	-0.0000
Dimension 181	0.0000	-0.0000	0.0000
Dimension 182	0.0000	0.0000	0.0000
Dimension 183	-0.0000	-0.0000	0.0000
Dimension 184	-0.0000	-0.0000	-0.0000
Dimension 185	-0.0000	0.0000	-0.0000
Dimension 186	-0.0000	-0.0000	-0.0000
Dimension 187	-0.0000	-0.0000	-0.0000
Dimension 188	0.0000	0.0000	0.0000
Dimension 189	-0.0000	-0.0000	-0.0000
Dimension 190	-0.0000	-0.0000	-0.0000
Dimension 191	-0.0000	-0.0000	-0.0000
Dimension 192	-0.0000	-0.0000	-0.0000
Dimension 193	0.0000	0.0000	-0.0000
Dimension 194	0.0000	-0.0000	-0.0000
Dimension 195	-0.0000	-0.0000	0.0000
Dimension 196	0.0000	-0.0000	0.0000

```
Dimension 197        -0.0000               -0.0000          -0.0000

              GREEN_AVANTGARDE    ...     CAMEO_DEU_2015_9A  \
Dimension 1            -0.1005    ...                0.0379
Dimension 2            -0.0019    ...               -0.0067
Dimension 3             0.0895    ...               -0.0049
Dimension 4             0.2859    ...               -0.0077
Dimension 5            -0.1160    ...                0.0597
Dimension 6             0.0029    ...                0.0089
Dimension 7            -0.0425    ...               -0.0279
Dimension 8            -0.0304    ...               -0.0613
Dimension 9             0.0236    ...               -0.0782
Dimension 10           -0.0177    ...                0.0394
Dimension 11           -0.0669    ...               -0.0433
Dimension 12           -0.0649    ...               -0.0037
Dimension 13           -0.0054    ...                0.0670
Dimension 14           -0.0055    ...               -0.1127
Dimension 15            0.0257    ...               -0.0223
Dimension 16            0.0429    ...               -0.0331
Dimension 17           -0.0672    ...                0.0174
Dimension 18           -0.0147    ...               -0.0045
Dimension 19           -0.0698    ...               -0.0120
Dimension 20           -0.1072    ...               -0.1042
Dimension 21           -0.0783    ...                0.0219
Dimension 22           -0.0168    ...                0.0853
Dimension 23           -0.0106    ...                0.0257
Dimension 24            0.0469    ...                0.0082
Dimension 25           -0.0463    ...                0.0270
Dimension 26            0.0512    ...                0.0307
Dimension 27            0.0606    ...               -0.0593
Dimension 28            0.0314    ...                0.0306
Dimension 29            0.0132    ...                0.0378
Dimension 30            0.0214    ...                0.1293
...                        ...    ...                   ...
Dimension 168          -0.0003    ...               -0.0018
Dimension 169           0.0972    ...               -0.0694
Dimension 170          -0.0273    ...                0.1074
Dimension 171          -0.2150    ...               -0.1241
Dimension 172          -0.0699    ...                0.1256
Dimension 173          -0.3310    ...                0.0441
Dimension 174           0.1149    ...               -0.0614
Dimension 175          -0.1715    ...                0.0334
Dimension 176          -0.1293    ...               -0.0029
Dimension 177           0.0414    ...                0.0874
Dimension 178          -0.0036    ...                0.0116
Dimension 179           0.0357    ...                0.1262
Dimension 180          -0.0333    ...                0.0244
Dimension 181          -0.0257    ...               -0.0253
```

Dimension 182	0.0528	...	-0.0399
Dimension 183	-0.0173	...	0.0145
Dimension 184	0.0410	...	0.0183
Dimension 185	0.1322	...	0.0613
Dimension 186	-0.0828	...	0.0075
Dimension 187	-0.1572	...	-0.0505
Dimension 188	-0.1319	...	0.0356
Dimension 189	-0.2041	...	0.0665
Dimension 190	0.0064	...	-0.0887
Dimension 191	-0.0521	...	-0.0085
Dimension 192	-0.3855	...	-0.0422
Dimension 193	0.0284	...	-0.0292
Dimension 194	0.0049	...	-0.0005
Dimension 195	0.0000	...	-0.0000
Dimension 196	0.0000	...	0.0000
Dimension 197	-0.0004	...	-0.0000

	CAMEO_DEU_2015_9B	CAMEO_DEU_2015_9C	CAMEO_DEU_2015_9D \
Dimension 1	0.0572	0.0584	0.0574
Dimension 2	0.0022	0.0064	0.0118
Dimension 3	0.0116	0.0244	0.0104
Dimension 4	-0.0112	-0.0017	-0.0078
Dimension 5	0.0385	0.0205	0.0211
Dimension 6	0.0447	0.0503	0.0526
Dimension 7	-0.0034	0.0192	-0.0142
Dimension 8	-0.1186	-0.1094	-0.0890
Dimension 9	-0.0160	-0.0557	-0.0993
Dimension 10	0.0305	0.0238	0.0224
Dimension 11	-0.0580	-0.0304	-0.0018
Dimension 12	-0.0468	-0.0159	0.0013
Dimension 13	0.0482	0.0481	0.0181
Dimension 14	-0.1429	-0.1355	-0.1406
Dimension 15	-0.0121	-0.0306	-0.0416
Dimension 16	-0.0149	-0.0306	-0.0192
Dimension 17	0.0028	0.0183	0.0022
Dimension 18	-0.0127	-0.0256	-0.0351
Dimension 19	-0.0072	-0.0377	-0.0774
Dimension 20	-0.1357	-0.0802	-0.1121
Dimension 21	-0.0072	-0.0177	-0.0151
Dimension 22	0.0690	0.0700	0.1586
Dimension 23	-0.0035	-0.0661	-0.0404
Dimension 24	-0.0992	-0.0688	-0.0395
Dimension 25	0.0596	0.0304	0.0969
Dimension 26	0.0848	0.0115	-0.0819
Dimension 27	-0.0815	-0.0115	0.0425
Dimension 28	0.0568	0.0190	-0.0653
Dimension 29	-0.0313	-0.0346	0.0599
Dimension 30	0.0855	0.0140	-0.0743

...	...	...	...
Dimension 168	-0.0022	-0.0029	-0.0028
Dimension 169	-0.0667	-0.0635	-0.0677
Dimension 170	0.1056	0.1005	0.1073
Dimension 171	-0.1222	-0.1163	-0.1241
Dimension 172	0.1263	0.1203	0.1283
Dimension 173	0.0531	0.0506	0.0540
Dimension 174	-0.0667	-0.0635	-0.0678
Dimension 175	0.0534	0.0508	0.0542
Dimension 176	-0.0119	-0.0113	-0.0121
Dimension 177	0.0965	0.0918	0.0980
Dimension 178	0.0035	0.0034	0.0036
Dimension 179	0.1671	0.1590	0.1697
Dimension 180	0.0360	0.0343	0.0366
Dimension 181	-0.0348	-0.0332	-0.0354
Dimension 182	-0.0588	-0.0560	-0.0597
Dimension 183	0.0037	0.0035	0.0038
Dimension 184	0.0360	0.0343	0.0366
Dimension 185	0.0411	0.0391	0.0417
Dimension 186	0.0218	0.0207	0.0221
Dimension 187	-0.0625	-0.0595	-0.0635
Dimension 188	0.0053	0.0050	0.0053
Dimension 189	0.1056	0.1005	0.1072
Dimension 190	-0.1025	-0.0976	-0.1041
Dimension 191	-0.0357	-0.0340	-0.0363
Dimension 192	-0.0651	-0.0620	-0.0662
Dimension 193	-0.0233	-0.0222	-0.0237
Dimension 194	-0.0001	-0.0001	-0.0001
Dimension 195	-0.0000	-0.0000	-0.0000
Dimension 196	-0.0000	-0.0000	-0.0000
Dimension 197	-0.0000	-0.0000	-0.0000

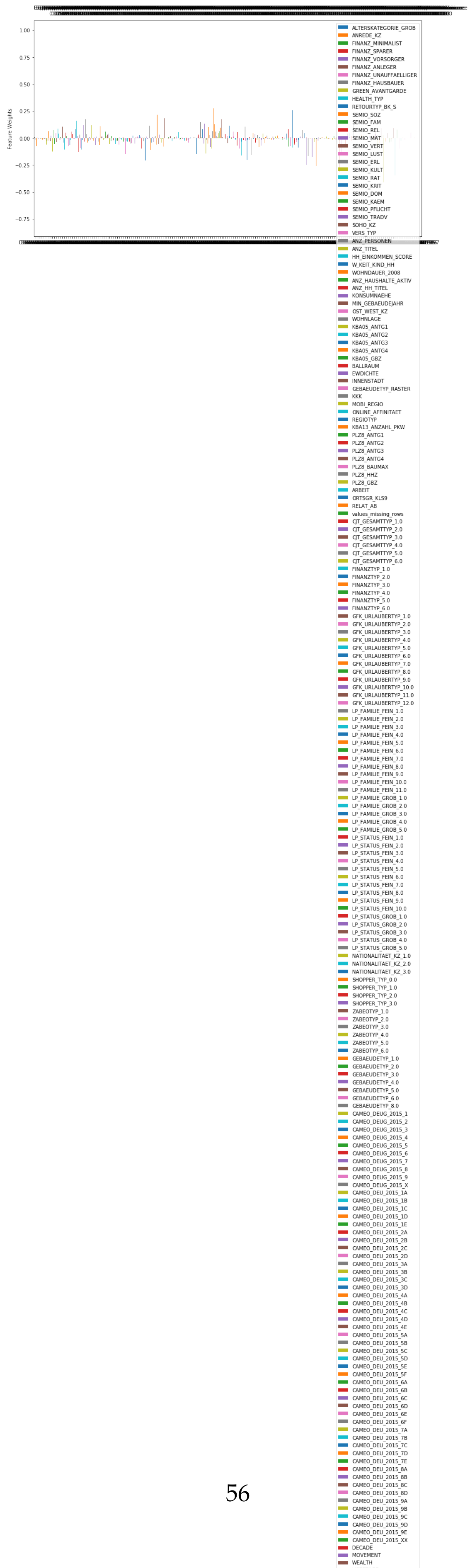
	CAMEO_DEU_2015_9E	CAMEO_DEU_2015_XX	DECADE	MOVEMENT	WEALTH	\
Dimension 1	0.0170	-0.0	0.0742	-0.1005	0.1826	
Dimension 2	0.0379	-0.0	-0.2324	-0.0019	0.0545	
Dimension 3	-0.0031	-0.0	-0.0523	0.0895	0.0117	
Dimension 4	-0.0015	-0.0	0.0324	0.2859	-0.0743	
Dimension 5	-0.0057	-0.0	-0.0170	-0.1160	0.1005	
Dimension 6	0.0130	0.0	0.0315	0.0029	-0.0430	
Dimension 7	0.0734	0.0	0.0206	-0.0425	-0.0516	
Dimension 8	-0.0235	-0.0	-0.0074	-0.0304	0.0433	
Dimension 9	-0.0414	-0.0	0.0097	0.0236	-0.0197	
Dimension 10	0.0929	0.0	-0.0160	-0.0177	-0.0852	
Dimension 11	-0.0974	-0.0	-0.0365	-0.0669	0.0266	
Dimension 12	0.0354	0.0	-0.0004	-0.0649	-0.0082	
Dimension 13	0.0561	0.0	-0.0080	-0.0054	-0.0205	
Dimension 14	-0.0673	-0.0	0.0078	-0.0055	-0.0081	
Dimension 15	-0.0357	0.0	0.0082	0.0257	0.0801	

Dimension 16	0.0028	-0.0	0.0278	0.0429	0.0733
Dimension 17	0.0002	0.0	-0.0333	-0.0672	-0.0659
Dimension 18	-0.0250	0.0	-0.0051	-0.0147	0.0054
Dimension 19	0.0356	0.0	-0.0338	-0.0698	-0.0572
Dimension 20	0.0413	-0.0	0.0018	-0.1072	-0.1763
Dimension 21	-0.0080	0.0	0.0007	-0.0783	-0.0391
Dimension 22	-0.0208	-0.0	-0.0073	-0.0168	-0.0501
Dimension 23	0.0124	0.0	0.0097	-0.0106	-0.0962
Dimension 24	0.0488	-0.0	-0.1080	0.0469	0.0608
Dimension 25	0.0302	0.0	-0.0814	-0.0463	0.0034
Dimension 26	0.0025	0.0	-0.0201	0.0512	0.0508
Dimension 27	0.1358	-0.0	-0.0023	0.0606	0.1270
Dimension 28	0.0270	-0.0	0.0395	0.0314	0.0364
Dimension 29	0.1356	-0.0	0.0316	0.0132	0.0717
Dimension 30	0.0997	-0.0	-0.0761	0.0214	0.0705
...	...	...	...	...	...
Dimension 168	-0.0001	0.0	0.0073	-0.0003	0.0055
Dimension 169	-0.0587	0.0	-0.0000	-0.0972	0.2319
Dimension 170	0.0873	-0.0	0.0000	0.0273	0.0569
Dimension 171	-0.1008	0.0	-0.0000	0.2150	0.2684
Dimension 172	0.0981	-0.0	0.0000	0.0699	-0.1990
Dimension 173	0.0217	0.0	0.0000	0.3310	0.0610
Dimension 174	-0.0406	0.0	0.0000	-0.1149	0.1860
Dimension 175	-0.0029	-0.0	0.0000	0.1715	0.2250
Dimension 176	0.0107	0.0	-0.0000	0.1293	-0.0847
Dimension 177	0.0559	0.0	0.0000	-0.0414	0.2120
Dimension 178	0.0209	0.0	0.0000	0.0036	0.0680
Dimension 179	0.0399	0.0	0.0000	-0.0357	0.1417
Dimension 180	0.0022	-0.0	-0.0000	0.0333	-0.0232
Dimension 181	-0.0061	-0.0	0.0000	0.0257	0.1066
Dimension 182	-0.0039	0.0	-0.0000	-0.0528	0.0016
Dimension 183	0.0273	-0.0	-0.0000	0.0173	0.0919
Dimension 184	-0.0115	-0.0	0.0000	-0.0410	0.1027
Dimension 185	0.0778	-0.0	-0.0000	-0.1322	0.0053
Dimension 186	-0.0150	-0.0	0.0000	0.0828	0.1825
Dimension 187	-0.0223	0.0	0.0000	0.1572	-0.3292
Dimension 188	0.0725	0.0	0.0000	0.1319	-0.1164
Dimension 189	-0.0045	-0.0	-0.0000	0.2041	0.0281
Dimension 190	-0.0498	0.0	0.0000	-0.0064	-0.0597
Dimension 191	0.0331	0.0	-0.0000	0.0521	0.1158
Dimension 192	0.0001	-0.0	-0.0000	0.3855	0.0222
Dimension 193	-0.0316	-0.0	-0.0000	-0.0284	0.1053
Dimension 194	-0.0009	-0.0	0.0000	-0.0049	-0.0025
Dimension 195	-0.0000	0.0	-0.0000	-0.0000	-0.0000
Dimension 196	-0.0000	1.0	-0.0000	-0.0000	0.0000
Dimension 197	-0.0000	-0.0	-0.0000	0.0004	0.0000

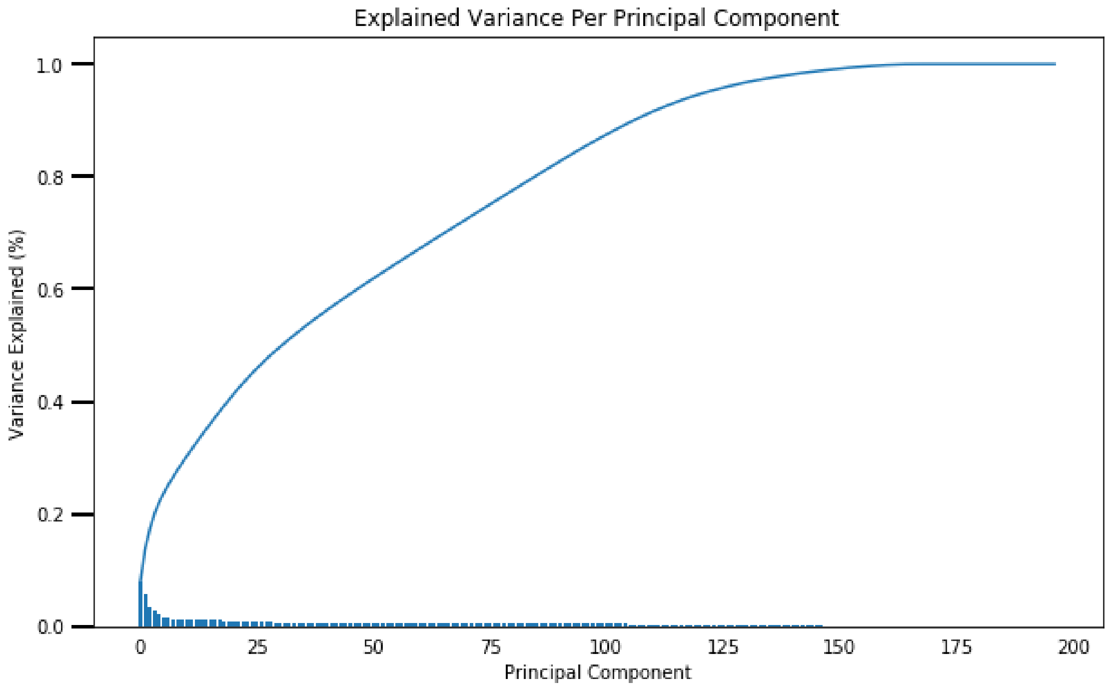
LIFESTAGE

Dimension 1	-0.1089
Dimension 2	0.0133
Dimension 3	-0.0118
Dimension 4	0.0396
Dimension 5	-0.0627
Dimension 6	-0.0679
Dimension 7	-0.0370
Dimension 8	0.0773
Dimension 9	0.2126
Dimension 10	0.1821
Dimension 11	-0.0717
Dimension 12	0.2367
Dimension 13	0.1296
Dimension 14	0.0191
Dimension 15	-0.0437
Dimension 16	-0.0388
Dimension 17	0.0031
Dimension 18	0.0016
Dimension 19	0.0521
Dimension 20	0.1764
Dimension 21	-0.0104
Dimension 22	-0.0919
Dimension 23	0.2317
Dimension 24	0.0592
Dimension 25	0.0138
Dimension 26	0.0396
Dimension 27	-0.0720
Dimension 28	-0.1263
Dimension 29	0.0941
Dimension 30	-0.0535
...	...
Dimension 168	-0.0051
Dimension 169	0.1095
Dimension 170	-0.1497
Dimension 171	0.1728
Dimension 172	-0.1529
Dimension 173	0.0176
Dimension 174	0.0341
Dimension 175	0.1205
Dimension 176	-0.0691
Dimension 177	-0.0372
Dimension 178	-0.0801
Dimension 179	0.1729
Dimension 180	0.0640
Dimension 181	-0.0455
Dimension 182	-0.1034
Dimension 183	-0.1064
Dimension 184	0.1211

```
Dimension 185    -0.2414
Dimension 186     0.1068
Dimension 187    -0.0340
Dimension 188    -0.2919
Dimension 189     0.2331
Dimension 190    -0.0006
Dimension 191    -0.2108
Dimension 192    -0.1328
Dimension 193     0.0845
Dimension 194     0.0033
Dimension 195     0.0000
Dimension 196     0.0000
Dimension 197     0.0000

[197 rows x 198 columns]
```

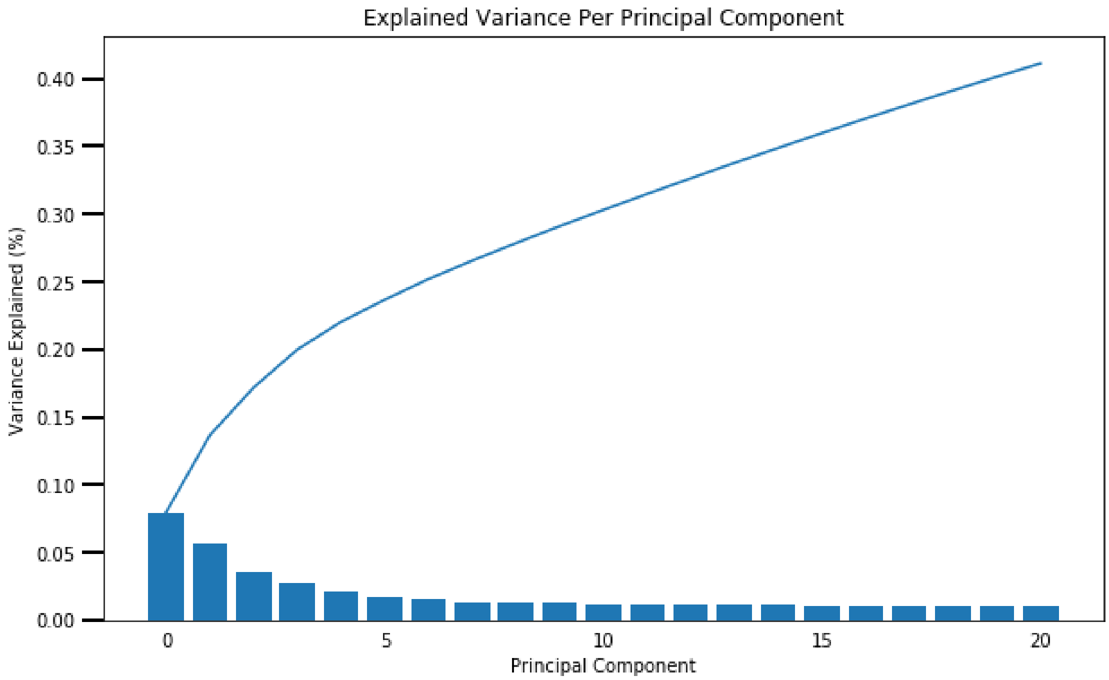

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Feature Weights
ALTERSKATEGORIE_GROB
ANREDE_KZ
FINANZ_MINIMALIST
FINANZ_SPARER
FINANZ_VORSORGER
FINANZ_ANLEGER
FINANZ_UNAUFFAELLIGER
FINANZ_HAUSBAUER
GREEN_AVANTGARDE
HEALTH_TYP
RETOURTYP_BK_S
SEMIO_SOZ
SEMIO_FAM
SEMIO_REL
SEMIO_MAT
SEMIO_VERT
SEMIO_LUST
SEMIO_ERL
SEMIO_KULT
SEMIO_RAT
SEMIO_KRIT
SEMIO_DOM
SEMIO_KAEM
SEMIO_PFLICHT
SEMIO_TRADV
SOHO_KZ
VERS_TYP
ANZ_PERSONEN
ANZ_TITEL
HH_EINKOMMEN_SCORE
W_KEIT_KIND_HH
WOHNDAUER_2008
ANZ_HAUSHALTE_AKTIV
ANZ_HH_TITEL
KONSUMNAEHE
MIN_GEBAEUDEJAHR
OST_WEST_KZ
WOHNLAGE
KBA05_ANTG1
KBA05_ANTG2
KBA05_ANTG3
KBA05_ANTG4
KBA05_GBZ
BALLRAUM
EWDICHTE
INNENSTADT
GEBAEUDETYP_RASTER
KKK
MOBI_REGIO
ONLINE_AFFINITAET
REGIOTYP
KBA13_ANZAHL_PKW
PLZ8_ANTG1
PLZ8_ANTG2
PLZ8_ANTG3
PLZ8_ANTG4
PLZ8_BAUMAX
PLZ8_HHZ
PLZ8_GBZ
ARBEIT
ORTSGR_KLS9
RELAT_AB
values_missing_rows
CJT_GESAMTTYP_1.0
CJT_GESAMTTYP_2.0
CJT_GESAMTTYP_3.0
CJT_GESAMTTYP_4.0
CJT_GESAMTTYP_5.0
CJT_GESAMTTYP_6.0
FINANZTYP_1.0
FINANZTYP_2.0
FINANZTYP_3.0
FINANZTYP_4.0
FINANZTYP_5.0
FINANZTYP_6.0
GFK_URLAUBERTYP_1.0
GFK_URLAUBERTYP_2.0
GFK_URLAUBERTYP_3.0
GFK_URLAUBERTYP_4.0
GFK_URLAUBERTYP_5.0
GFK_URLAUBERTYP_6.0
GFK_URLAUBERTYP_7.0
GFK_URLAUBERTYP_8.0
GFK_URLAUBERTYP_9.0
GFK_URLAUBERTYP_10.0
GFK_URLAUBERTYP_11.0
GFK_URLAUBERTYP_12.0
LP_FAMILIE_FEIN_1.0
LP_FAMILIE_FEIN_2.0
LP_FAMILIE_FEIN_3.0
LP_FAMILIE_FEIN_4.0
LP_FAMILIE_FEIN_5.0
LP_FAMILIE_FEIN_6.0
LP_FAMILIE_FEIN_7.0
LP_FAMILIE_FEIN_8.0
LP_FAMILIE_FEIN_9.0
LP_FAMILIE_FEIN_10.0
LP_FAMILIE_FEIN_11.0
LP_FAMILIE_GROB_1.0
LP_FAMILIE_GROB_2.0
LP_FAMILIE_GROB_3.0
LP_FAMILIE_GROB_4.0
LP_FAMILIE_GROB_5.0
LP_STATUS_FEIN_1.0
LP_STATUS_FEIN_2.0
LP_STATUS_FEIN_3.0
LP_STATUS_FEIN_4.0
LP_STATUS_FEIN_5.0
LP_STATUS_FEIN_6.0
LP_STATUS_FEIN_7.0
LP_STATUS_FEIN_8.0
LP_STATUS_FEIN_9.0
LP_STATUS_FEIN_10.0
LP_STATUS_GROB_1.0
LP_STATUS_GROB_2.0
LP_STATUS_GROB_3.0
LP_STATUS_GROB_4.0
LP_STATUS_GROB_5.0
NATIONALITAET_KZ_1.0
NATIONALITAET_KZ_2.0
NATIONALITAET_KZ_3.0
SHOPPER_TYP_0.0
SHOPPER_TYP_1.0
SHOPPER_TYP_2.0
SHOPPER_TYP_3.0
ZABEOTYP_1.0
ZABEOTYP_2.0
ZABEOTYP_3.0
ZABEOTYP_4.0
ZABEOTYP_5.0
ZABEOTYP_6.0
GEBAEUDETYP_1.0
GEBAEUDETYP_2.0
GEBAEUDETYP_3.0
GEBAEUDETYP_4.0
GEBAEUDETYP_5.0
GEBAEUDETYP_6.0
GEBAEUDETYP_8.0
CAMEO_DEUG_2015_1
CAMEO_DEUG_2015_2
CAMEO_DEUG_2015_3
CAMEO_DEUG_2015_4
CAMEO_DEUG_2015_5
CAMEO_DEUG_2015_6
CAMEO_DEUG_2015_7
CAMEO_DEUG_2015_8
CAMEO_DEUG_2015_9
CAMEO_DEUG_2015_X
CAMEO_DEU_2015_1A
CAMEO_DEU_2015_1B
CAMEO_DEU_2015_1C
CAMEO_DEU_2015_1D
CAMEO_DEU_2015_1E
CAMEO_DEU_2015_2A
CAMEO_DEU_2015_2B
CAMEO_DEU_2015_2C
CAMEO_DEU_2015_2D
CAMEO_DEU_2015_3A
CAMEO_DEU_2015_3B
CAMEO_DEU_2015_3C
CAMEO_DEU_2015_3D
CAMEO_DEU_2015_4A
CAMEO_DEU_2015_4B
CAMEO_DEU_2015_4C
CAMEO_DEU_2015_4D
CAMEO_DEU_2015_4E
CAMEO_DEU_2015_5A
CAMEO_DEU_2015_5B
CAMEO_DEU_2015_5C
CAMEO_DEU_2015_5D
CAMEO_DEU_2015_5E
CAMEO_DEU_2015_5F
CAMEO_DEU_2015_6A
CAMEO_DEU_2015_6B
CAMEO_DEU_2015_6C
CAMEO_DEU_2015_6D
CAMEO_DEU_2015_6E
CAMEO_DEU_2015_6F
CAMEO_DEU_2015_7A
CAMEO_DEU_2015_7B
CAMEO_DEU_2015_7C
CAMEO_DEU_2015_7D
CAMEO_DEU_2015_7E
CAMEO_DEU_2015_8A
CAMEO_DEU_2015_8B
CAMEO_DEU_2015_8C
CAMEO_DEU_2015_8D
CAMEO_DEU_2015_9A
CAMEO_DEU_2015_9B
CAMEO_DEU_2015_9C
CAMEO_DEU_2015_9D
CAMEO_DEU_2015_9E
CAMEO_DEU_2015_XX
DECADE
MOVEMENT
WEALTH
LIFESTAGE

```
In [58]: def scree_plot(pca):
             '''
              Creates a scree plot associated with the principal components

              INPUT: pca - the result of instantian of PCA in scikit learn

              OUTPUT:
                      None
             '''
             num_components=len(pca.explained_variance_ratio_)
             ind = np.arange(num_components)
             vals = pca.explained_variance_ratio_

             plt.figure(figsize=(10, 6))
             ax = plt.subplot(111)
             cumvals = np.cumsum(vals)
             ax.bar(ind, vals)
             ax.plot(ind, cumvals)

             ax.xaxis.set_tick_params(width=0)
             ax.yaxis.set_tick_params(width=2, length=12)

             ax.set_xlabel("Principal Component")
             ax.set_ylabel("Variance Explained (%)")
             plt.title('Explained Variance Per Principal Component')

In [59]: scree_plot(pca_1)
```




```
In [60]: # Re-apply PCA to the data while selecting for number of components to retain.
         pca_2 = PCA(21)
         azdias_pca = pca_2.fit_transform(azdias_scaled)
```

```
In [61]: scree_plot(pca_2)
```



2 Discussion 2.2: Perform Dimensionality Reduction

After applying PCA and visualizing the results in a scree plot, I decided to reduce the number of dimensions used within the second PCA to 21. The reason for this is that around 21 dimensions, the cumulative variance reaches around 0.4% and each addidional components adds around 0% to the cumulative variance.

2.0.1 Step 2.3: Interpret Principal Components

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

As a reminder, each principal component is a unit vector that points in the direction of highest variance (after accounting for the variance captured by earlier principal components). The further a weight is from zero, the more the principal component is in the direction of the corresponding feature. If two features have large weights of the same sign (both positive or both negative), then increases in one tend expect to be associated with increases in the other. To contrast, features with different signs can be expected to show a negative correlation: increases in one variable should result in a decrease in the other.

- To investigate the features, you should map each weight to their corresponding feature name, then sort the features according to weight. The most interesting features for each principal component, then, will be those at the beginning and end of the sorted list. Use the data dictionary document to help you understand these most prominent features, their relationships, and what a positive or negative value on the principal component might indicate.
- You should investigate and interpret feature associations from the first three principal components in this substep. To help facilitate this, you should write a function that you can call at any time to print the sorted list of feature weights, for the *i*-th principal component. This might come in handy in the next step of the project, when you interpret the tendencies of the discovered clusters.

In [62]:
```
# Map weights for the first principal component to corresponding feature names
# and then print the linked values, sorted by weight.
# HINT: Try defining a function here or in a new cell that you can reuse in the
# other cells.

def map_weights(dataset, pca, n_element):
    weights_dataset = pd.DataFrame(pca.components_, columns=list(dataset.columns)).iloc
    weights_dataset.sort_values(ascending = False, inplace = True)

    return weights_dataset
```

In [63]:
```
map_weights(dataset = azdias_scaled, pca = pca_2, n_element = 1)
```

Out[63]:
```
LP_STATUS_GROB_1.0      0.193787
PLZ8_ANTG3              0.183503
```

WEALTH	0.182637
HH_EINKOMMEN_SCORE	0.181637
PLZ8_ANTG4	0.178236
PLZ8_BAUMAX	0.174903
ORTSGR_KLS9	0.158082
EWDICHTE	0.156088
FINANZ_HAUSBAUER	0.147351
KBA05_ANTG4	0.129581
LP_STATUS_FEIN_1.0	0.126889
PLZ8_ANTG2	0.125775
KBA05_ANTG3	0.116148
ANZ_HAUSHALTE_AKTIV	0.115919
ARBEIT	0.114540
CAMEO_DEUG_2015_9	0.113467
LP_STATUS_FEIN_2.0	0.113415
FINANZ_SPARER	0.110694
FINANZTYP_1.0	0.109746
RELAT_AB	0.108525
CAMEO_DEUG_2015_8	0.092197
LP_FAMILIE_FEIN_1.0	0.085370
LP_FAMILIE_GROB_1.0	0.085370
SEMIO_PFLICHT	0.080072
SEMIO_REL	0.075978
DECADE	0.074225
ZABEOTYP_5.0	0.073671
GEBAEUDETYP_3.0	0.068125
SEMIO_RAT	0.066279
CAMEO_DEU_2015_8A	0.063201
	...
KBA13_ANZAHL_PKW	-0.060072
LP_FAMILIE_GROB_5.0	-0.063943
WOHNLAGE	-0.064003
NATIONALITAET_KZ_1.0	-0.065444
CAMEO_DEUG_2015_3	-0.065580
CAMEO_DEUG_2015_4	-0.072842
FINANZTYP_2.0	-0.076577
ALTERSKATEGORIE_GROB	-0.080052
FINANZ_VORSORGER	-0.080470
ANZ_PERSONEN	-0.081983
CAMEO_DEUG_2015_2	-0.087438
GEBAEUDETYP_1.0	-0.088912
ZABEOTYP_1.0	-0.094620
BALLRAUM	-0.098530
GEBAEUDETYP_RASTER	-0.098660
GREEN_AVANTGARDE	-0.100500
MOVEMENT	-0.100500
LIFESTAGE	-0.108890
LP_STATUS_FEIN_9.0	-0.109279

```
LP_STATUS_GROB_5.0      -0.110794
LP_STATUS_FEIN_10.0     -0.110794
LP_STATUS_GROB_4.0      -0.111372
INNENSTADT              -0.129926
PLZ8_GBZ                -0.135045
KONSUMNAEHE             -0.138089
KBA05_ANTG1             -0.178430
KBA05_GBZ               -0.180082
PLZ8_ANTG1              -0.184600
MOBI_REGIO              -0.186226
FINANZ_MINIMALIST       -0.187084
Name: 0, Length: 197, dtype: float64
```

In [64]:
```
# Map weights for the second principal component to corresponding feature names
# and then print the linked values, sorted by weight.
map_weights(dataset = azdias_scaled, pca = pca_2, n_element = 2)
```

Out[64]:
```
ALTERSKATEGORIE_GROB     0.234053
FINANZ_VORSORGER         0.220420
ZABEOTYP_3.0             0.198965
SEMIO_ERL                0.179673
SEMIO_LUST               0.163919
RETOURTYP_BK_S           0.154773
W_KEIT_KIND_HH           0.119489
CJT_GESAMTTYP_2.0        0.108703
FINANZTYP_5.0            0.099634
LP_STATUS_FEIN_1.0       0.094932
FINANZ_MINIMALIST        0.092495
FINANZTYP_2.0            0.092494
FINANZ_HAUSBAUER         0.083268
SHOPPER_TYP_3.0          0.075240
CJT_GESAMTTYP_1.0        0.070803
SEMIO_KRIT               0.070769
NATIONALITAET_KZ_1.0     0.069576
FINANZTYP_6.0            0.068713
EWDICHTE                 0.064863
ORTSGR_KLS9              0.064030
PLZ8_ANTG3               0.063347
PLZ8_ANTG4               0.062798
GFK_URLAUBERTYP_4.0      0.059091
PLZ8_BAUMAX              0.058812
WOHNDAUER_2008           0.056382
SEMIO_KAEM               0.056228
LP_FAMILIE_FEIN_1.0      0.056196
LP_FAMILIE_GROB_1.0      0.056196
WEALTH                   0.054480
LP_STATUS_FEIN_3.0       0.051716
```

...

```
KONSUMNAEHE            -0.050427
INNENSTADT             -0.051140
CJT_GESAMTTYP_4.0      -0.054761
HEALTH_TYP             -0.055656
ZABEOTYP_1.0           -0.056769
KBA05_GBZ              -0.057670
SEMIO_SOZ              -0.060749
ANZ_PERSONEN           -0.061824
PLZ8_ANTG1             -0.062213
LP_FAMILIE_GROB_4.0    -0.068638
GFK_URLAUBERTYP_9.0    -0.073111
FINANZTYP_3.0          -0.074932
LP_STATUS_FEIN_5.0     -0.081195
FINANZTYP_4.0          -0.088093
ZABEOTYP_5.0           -0.091385
LP_STATUS_FEIN_2.0     -0.096085
ZABEOTYP_4.0           -0.106977
SEMIO_MAT              -0.131288
SEMIO_FAM              -0.134437
FINANZTYP_1.0          -0.135409
ONLINE_AFFINITAET      -0.158992
SEMIO_KULT             -0.164497
SEMIO_RAT              -0.170158
FINANZ_ANLEGER         -0.206889
SEMIO_PFLICHT          -0.207549
SEMIO_TRADV            -0.208223
SEMIO_REL              -0.216287
FINANZ_UNAUFFAELLIGER  -0.216902
FINANZ_SPARER          -0.229799
DECADE                 -0.232353
Name: 1, Length: 197, dtype: float64
```

In [65]:
```
# Map weights for the third principal component to corresponding feature names
# and then print the linked values, sorted by weight.
map_weights(dataset = azdias_scaled, pca = pca_2, n_element = 3)
```

Out[65]:
```
SEMIO_VERT              0.315676
SEMIO_FAM               0.257303
SEMIO_SOZ               0.255503
SEMIO_KULT              0.249469
FINANZTYP_5.0           0.136210
FINANZ_MINIMALIST       0.130590
SHOPPER_TYP_0.0         0.122154
ZABEOTYP_1.0            0.121696
SEMIO_REL               0.112652
MOVEMENT                0.089511
GREEN_AVANTGARDE        0.089511
SEMIO_MAT               0.088993
```

```
RETOURTYP_BK_S            0.070763
ORTSGR_KLS9               0.070418
EWDICHTE                  0.069769
LP_STATUS_FEIN_10.0       0.064747
LP_STATUS_GROB_5.0        0.064747
SHOPPER_TYP_1.0           0.053329
PLZ8_BAUMAX               0.053226
W_KEIT_KIND_HH            0.052685
PLZ8_ANTG4                0.051098
PLZ8_ANTG3                0.049975
LP_STATUS_FEIN_3.0        0.047292
ZABEOTYP_6.0              0.046785
FINANZ_VORSORGER          0.043807
LP_STATUS_FEIN_1.0        0.038607
RELAT_AB                  0.037113
PLZ8_ANTG2                0.036877
ARBEIT                    0.035044
LP_STATUS_GROB_3.0        0.034956
                            ...
LP_STATUS_GROB_2.0       -0.032101
PLZ8_GBZ                 -0.033380
LP_FAMILIE_FEIN_4.0      -0.038235
GEBAEUDETYP_RASTER       -0.041188
ZABEOTYP_3.0             -0.042004
FINANZ_UNAUFFAELLIGER    -0.042784
KKK                      -0.045913
PLZ8_ANTG1               -0.046185
HH_EINKOMMEN_SCORE       -0.046269
WOHNLAGE                 -0.048622
SHOPPER_TYP_3.0          -0.050411
KONSUMNAEHE              -0.050466
DECADE                   -0.052290
LP_FAMILIE_GROB_3.0      -0.053440
FINANZ_SPARER            -0.053856
BALLRAUM                 -0.055244
INNENSTADT               -0.061362
ZABEOTYP_4.0             -0.062400
FINANZ_HAUSBAUER         -0.064850
LP_STATUS_FEIN_4.0       -0.077207
LP_STATUS_FEIN_2.0       -0.079216
SHOPPER_TYP_2.0          -0.091977
FINANZTYP_1.0            -0.099685
SEMIO_RAT                -0.149934
FINANZ_ANLEGER           -0.153512
SEMIO_ERL                -0.208165
SEMIO_KRIT               -0.265913
SEMIO_DOM                -0.276561
SEMIO_KAEM               -0.309801
```

```
ANREDE_KZ              -0.340336
Name: 2, Length: 197, dtype: float64
```

2.0.2 Discussion 2.3: Interpret Principal Components

First Principal Component: > Strongest Positive Correlation: - LP_STATUS_GROB_1.0 0.193787 Social status: low-income earners - PLZ8_ANTG3 0.183502 Number of 6-10 family houses in the PLZ8 region (none -> very high) - WEALTH 0.182637 Household wealth - HH_EINKOMMEN_SCORE 0.181638 Estimated household net income (highest -> very low)

> Strongest Negative Correlation: - KBA05_GBZ -0.180082 Number of buildings in the microcell (none -> very high) - PLZ8_ANTG1 -0.184599 Number of 1-2 family houses in the PLZ8 region (none -> very high) - MOBI_REGIO -0.186226 Movement patterns (very high -> very low movement) - FINANZ_MINIMALIST -0.187084 Low financial interest (very high -> very low)

Second Principal Component: > Strongest Positive Correlation: - ALTERSKATEGORIE_GROB 0.234056 Estimated age based on given name analysis (young -> older) - FINANZ_VORSORGER 0.220420 Financially prepared (very high -> very low) - ZABEOTYP_3.0 0.198964 Energy consumption typology (green -> indifferent)

> Strongest Negative Correlation: - FINANZ_ANLEGER -0.206889 Financial Investor (very high -> very low) - SEMIO_PFLICHT -0.207550 Dutyful personality (high -> lowest) - SEMIO_TRADV -0.208227 Tradional-minded personality (high -> lowest) - SEMIO_REL -0.216288 Religious personality (high -> lowest)
> - FINANZ_UNAUFFAELLIGER -0.216900 Inconspicuous financial interest (very high -> very low) - FINANZ_SPARER -0.229798 Money-saver (very high -> very low)

Third Principal Component: > Strongest Positive Correlation: - SEMIO_VERT 0.315717 Dreamful personality (highest -> lowest) - SEMIO_FAM 0.257311 Family-minded personality (highest -> lowest) - SEMIO_SOZ 0.255512 Socially-minded personality (highest -> lowest) - SEMIO_KULT 0.249493 Cultural-minded personality (highest -> lowest)

> Strongest Negative Correlation: - SEMIO_ERL -0.208151 Event-oriented personality (highest -> lowest) - SEMIO_KRIT -0.265917 Critical-minded personality (highest -> lowest) - SEMIO_DOM -0.276579 Dominant-minded personality (highest -> lowest) - SEMIO_KAEM -0.309794 Combative attitude personality (highest -> lowest) - ANREDE_KZ -0.340355 Gender (male -> female)

Conclusion: ##

The first component has the highest positive correlation with features such as lower household income, wealth and number of multi-family houses in the region. The first component has a negative correlation with a high number of 1-2 family houses, low movement patterns and a greater financial interest.

The second component has the highest positive correlation with features such as older estimated age, lower financial preparedness, and an indifferent energy consumption. It has a strong negative correlation with the features of a less dutiful, less traditional, less religious personality.

The third component has the highest positive correlation with features such as a less dreamful, less family-minded, less socially- and culturally-minded personality. It has a stronger negative correlation with a less event-oriented, less critial-minded, less dominant-mined, less combative personality and female gender.

2.1 Step 3: Clustering

2.1.1 Step 3.1: Apply Clustering to General Population

You've assessed and cleaned the demographics data, then scaled and transformed them. Now, it's time to see how the data clusters in the principal components space. In this substep, you will apply k-means clustering to the dataset and use the average within-cluster distances from each point to their assigned cluster's centroid to decide on a number of clusters to keep.

- Use sklearn's KMeans class to perform k-means clustering on the PCA-transformed data.
- Then, compute the average difference from each point to its assigned cluster's center. **Hint**: The KMeans object's `.score()` method might be useful here, but note that in sklearn, scores tend to be defined so that larger is better. Try applying it to a small, toy dataset, or use an internet search to help your understanding.
- Perform the above two steps for a number of different cluster counts. You can then see how the average distance decreases with an increasing number of clusters. However, each additional cluster provides a smaller net benefit. Use this fact to select a final number of clusters in which to group the data. **Warning**: because of the large size of the dataset, it can take a long time for the algorithm to resolve. The more clusters to fit, the longer the algorithm will take. You should test for cluster counts through at least 10 clusters to get the full picture, but you shouldn't need to test for a number of clusters above about 30.
- Once you've selected a final number of clusters to use, re-fit a KMeans instance to perform the clustering operation. Make sure that you also obtain the cluster assignments for the general demographics data, since you'll be using them in the final Step 3.3.

```
In [66]: #defining function that was given during the course.
         def kmeans_score(data, center):
             '''
             returns the kmeans score regarding SSE for points to centers
             INPUT:
                 data - the dataset you want to fit kmeans to
                 center - the number of centers you want (the k value)
             OUTPUT:
                 score - the SSE score for the kmeans model fit to the data
             '''
             #instantiate kmeans
             kmeans = KMeans(center)

             # Then fit the model to your data using the fit method
             model = kmeans.fit(data)

             # Obtain a score related to the model fit
             score = np.abs(model.score(data))

             return score
```

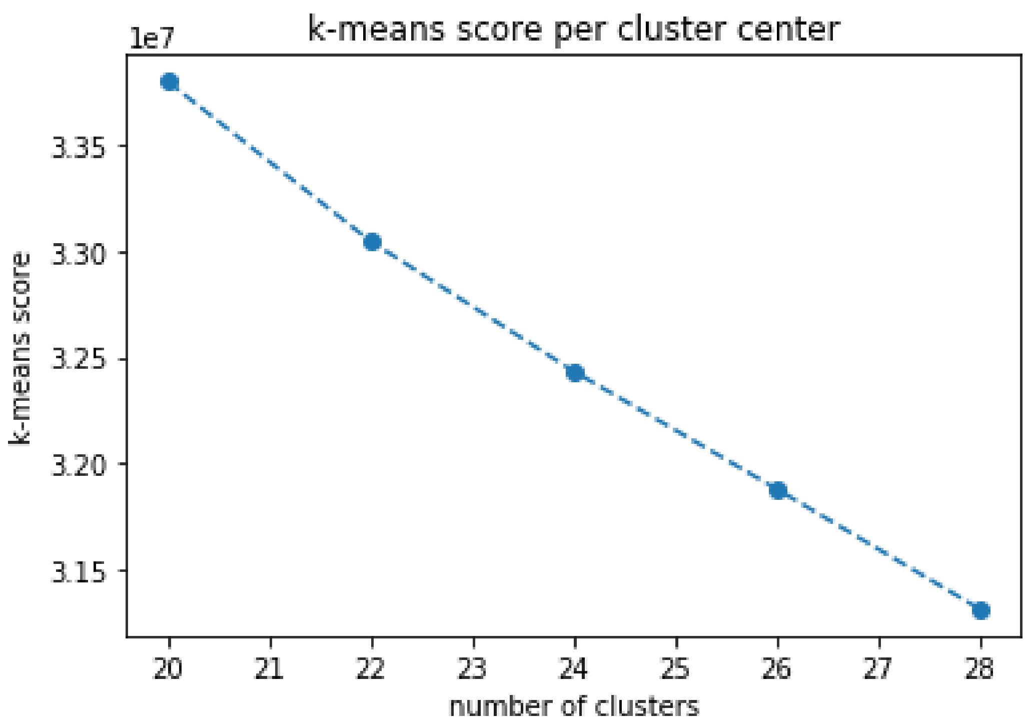
```
In [67]: # Over a number of different cluster counts run k-means clustering on the data and comp
         centers = np.arange(20, 30, 2)
         scores = []
```

```
for center in centers:
    scores.append(kmeans_score(azdias_pca, center))
    print("cluster ", center , " calculated")
```

```
cluster  20  calculated
cluster  22  calculated
cluster  24  calculated
cluster  26  calculated
cluster  28  calculated
```

In [68]:
```
# Investigate the change in within-cluster distance across number of clusters.
# HINT: Use matplotlib's plot function to visualize this relationship.
plt.plot(centers, scores, linestyle = '--', marker = 'o')
plt.xlabel("number of clusters")
plt.ylabel("k-means score")
plt.title("k-means score per cluster center")
```

Out[68]:
```
Text(0.5,1,'k-means score per cluster center')
```



In [69]:
```
# Re-fit the k-means model with the selected number of clusters and obtain
# cluster predictions for the general population demographics data.
azdias_kmeans = KMeans(22)
```

```
azdias_model = azdias_kmeans.fit(azdias_pca)
azdias_score = np.abs(azdias_model.score(azdias_pca))

azdias_cluster = azdias_model.predict(azdias_pca)
```

2.1.2 Discussion 3.1: Apply Clustering to General Population

(Double-click this cell and replace this text with your own text, reporting your findings and decisions regarding clustering. Into how many clusters have you decided to segment the population?)

Because there appears a slight "elbow" in the k-means score at 22 clusters, I decided to use 22 clusters moving forward.

2.1.3 Step 3.2: Apply All Steps to the Customer Data

Now that you have clusters and cluster centers for the general population, it's time to see how the customer data maps on to those clusters. Take care to not confuse this for re-fitting all of the models to the customer data. Instead, you're going to use the fits from the general population to clean, transform, and cluster the customer data. In the last step of the project, you will interpret how the general population fits apply to the customer data.

- Don't forget when loading in the customers data, that it is semicolon (`;`) delimited.
- Apply the same feature wrangling, selection, and engineering steps to the customer demographics using the `clean_data()` function you created earlier. (You can assume that the customer demographics data has similar meaning behind missing data patterns as the general demographics data.)
- Use the sklearn objects from the general demographics data, and apply their transformations to the customers data. That is, you should not be using a `.fit()` or `.fit_transform()` method to re-fit the old objects, nor should you be creating new sklearn objects! Carry the data through the feature scaling, PCA, and clustering steps, obtaining cluster assignments for all of the data in the customer demographics data.

```
In [70]: # Load in the customer demographics data.
         customers = pd.read_csv('Udacity_CUSTOMERS_Subset.csv', delimiter = ';')
```

```
In [71]: # Apply preprocessing, feature transformation, and clustering from the general
         # demographics onto the customer data, obtaining cluster predictions for the
         # customer demographics data.

         # Clean the customer data.
         customers_clean = clean_data(customers, feat_info)
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-71-1890898be3c8> in <module>()
  4
  5 # Clean the customer data.
```

```
----> 6 customers_clean = clean_data(customers, feat_info)


    <ipython-input-47-95cedb845189> in clean_data(df, feat_info)
     34
     35     # Get number of missing values per row.
---> 36     missing_rows = df_cleaned[df_cleaned.isnull().sum(axis=1)]
     37
     38     # Only use subset with few missing rows.


    AttributeError: 'NoneType' object has no attribute 'isnull'
```

```
In [ ]: # Replace NaN values in customer data.
        customers_reencoded = customers_clean.fillna(customers_clean.mode().iloc[0])
```

```
In [ ]: # Apply Feature Scaling to customer data.
        # Apply feature scaling to the general population demographics data.
        customers_scaled = pd.DataFrame(scaler.fit_transform(customers_reencoded), columns = cus
```

```
In [ ]: # Verify successful Feature Scaling.
        customers_scaled.head(10)
```

```
In [ ]: # Apply PCA transformations.
        customers_pca = pca_2.fit_transform(customers_scaled)
```

```
In [ ]: # Re-fit the k-means model with the selected number of clusters and obtain cluster predi
        customers_score, customers_model = kmeans_score(customers_pca, 25)
        customers_cluster = customers_model.predict(customers_pca)
```

2.1.4 Step 3.3: Compare Customer Data to Demographics Data

At this point, you have clustered data based on demographics of the general population of Germany, and seen how the customer data for a mail-order sales company maps onto those demographic clusters. In this final substep, you will compare the two cluster distributions to see where the strongest customer base for the company is.

Consider the proportion of persons in each cluster for the general population, and the proportions for the customers. If we think the company's customer base to be universal, then the cluster assignment proportions should be fairly similar between the two. If there are only particular segments of the population that are interested in the company's products, then we should see a mismatch from one to the other. If there is a higher proportion of persons in a cluster for the customer data compared to the general population (e.g. 5% of persons are assigned to a cluster for the general population, but 15% of the customer data is closest to that cluster's centroid) then that suggests the people in that cluster to be a target audience for the company. On the other hand, the proportion of the data in a cluster being larger in the general population than the customer data (e.g. only 2% of customers closest to a population centroid that captures 6% of the data) suggests that group of persons to be outside of the target demographics.

Take a look at the following points in this step:

- Compute the proportion of data points in each cluster for the general population and the customer data. Visualizations will be useful here: both for the individual dataset proportions, but also to visualize the ratios in cluster representation between groups. Seaborn's `countplot()` or `barplot()` function could be handy.
- Recall the analysis you performed in step 1.1.3 of the project, where you separated out certain data points from the dataset if they had more than a specified threshold of missing values. If you found that this group was qualitatively different from the main bulk of the data, you should treat this as an additional data cluster in this analysis. Make sure that you account for the number of data points in this subset, for both the general population and customer datasets, when making your computations!
- Which cluster or clusters are overrepresented in the customer dataset compared to the general population? Select at least one such cluster and infer what kind of people might be represented by that cluster. Use the principal component interpretations from step 2.3 or look at additional components to help you make this inference. Alternatively, you can use the `.inverse_transform()` method of the PCA and StandardScaler objects to transform centroids back to the original data space and interpret the retrieved values directly.
- Perform a similar investigation for the underrepresented clusters. Which cluster or clusters are underrepresented in the customer dataset compared to the general population, and what kinds of people are typified by these clusters?

In [ ]:
```
# Compare the proportion of data in each cluster for the customer data to the
# proportion of data in each cluster for the general population.
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_figwidth(18)

ax1.set_title('General Demographics Clusters')
sns.countplot(azdias_cluster, ax = ax1)

ax2.set_title('Customers Demographics Clusters')
sns.countplot(customers_cluster, ax = ax2)

plt.show()
```

In [ ]:
```
# What kinds of people are part of a cluster that is overrepresented in the
# customer data compared to the general population?
```

In [ ]:
```
# What kinds of people are part of a cluster that is underrepresented in the
# customer data compared to the general population?
```

2.1.5 Discussion 3.3: Compare Customer Data to Demographics Data

(Double-click this cell and replace this text with your own text, reporting findings and conclusions from the clustering analysis. Can we describe segments of the population that are relatively popular with the mail-order company, or relatively unpopular with the company?)

> Congratulations on making it this far in the project! Before you finish, make sure to check through the entire notebook from top to bottom to make sure that your analysis follows a logical flow and all of your findings are documented in **Discussion** cells. Once you've checked over all of your work, you should export the notebook as an

HTML document to submit for evaluation. You can do this from the menu, navigating to **File -> Download as -> HTML (.html)**. You will submit both that document and this notebook for your project submission.

```
In [ ]:
```