Identify_Customer_Segments-Copy1

October 1, 2023

1 Project: Identify Customer Segments

In this project, I 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 I will use has been provided by Bertelsmann Arvato Analytics, and represents a real-life data science task.

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

# additional libraries
import re
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# magic word for producing visualizations in notebook
%matplotlib inline

'''
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.
'''
```

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', delimiter = ';')

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

```
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).
        display(azdias.head())
        print("AZDIAS Shape: ", azdias.shape)
```

```
   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]


AZDIAS Shape:  (891221, 85)


In [4]: display(feat_info.head())
        display(feat_info.tail())
        print("Features Information Shape", feat_info.shape)

             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]


      attribute information_level     type missing_or_unknown
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]
```

```
Features Information Shape (85, 4)
```

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.

```
In [5]: # Identify missing or unknown data values and convert them to NaNs.

        num_missing = azdias.isnull().sum()
        display(num_missing.head(15))
        print("The total number of missing or unkown values is", num_missing.sum())
```

```
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
dtype: int64


The total number of missing or unkown values is 4896838
```

```
In [6]: display(feat_info['missing_or_unknown'].head(15))
```

```
0       [-1,0]
1     [-1,0,9]
2       [-1,0]
3          [0]
4         [-1]
5         [-1]
6         [-1]
7         [-1]
8         [-1]
9         [-1]
10        [-1]
11         [0]
12          []
13          []
14      [-1,0]
Name: missing_or_unknown, dtype: object
```

```
In [7]: import re
        import numpy as np

        for i in range(len(feat_info)):
            # using regex to extract the digits
            missing_data = re.sub('[\[|\]]', '',feat_info.iloc[i]['missing_or_unknown']).split('
            # return the digit values for integer strings only
            if missing_data != ['']:
                missing_data = [np.int64(data) if (data!='X' and data!='XX') else data for data
                azdias = azdias.replace({feat_info.iloc[i]['attribute']: missing_data}, np.nan)
```

```
In [8]: display(azdias.head())
```

```
   AGER_TYP  ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  \
0       NaN                   2.0          1            2.0
1       NaN                   1.0          2            5.0
2       NaN                   3.0          2            3.0
3       2.0                   4.0          2            2.0
4       NaN                   3.0          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 [9]: num_missing_after = azdias.isnull().sum()
        display(num_missing_after.head(15))
        print("The total number of missing or unkown values after converting is", num_missing_af
```

```
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
dtype: int64


The total number of missing or unkown values after converting is 8373929
```

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 [10]: # Perform an assessment of how much missing data there is in each column of the
         # dataset.
         num_missing_columns = azdias.isnull().sum()
         display(num_missing_columns.head())
         print(num_missing_columns.describe())
         print("The total number of misssing data", num_missing_columns.sum())
```

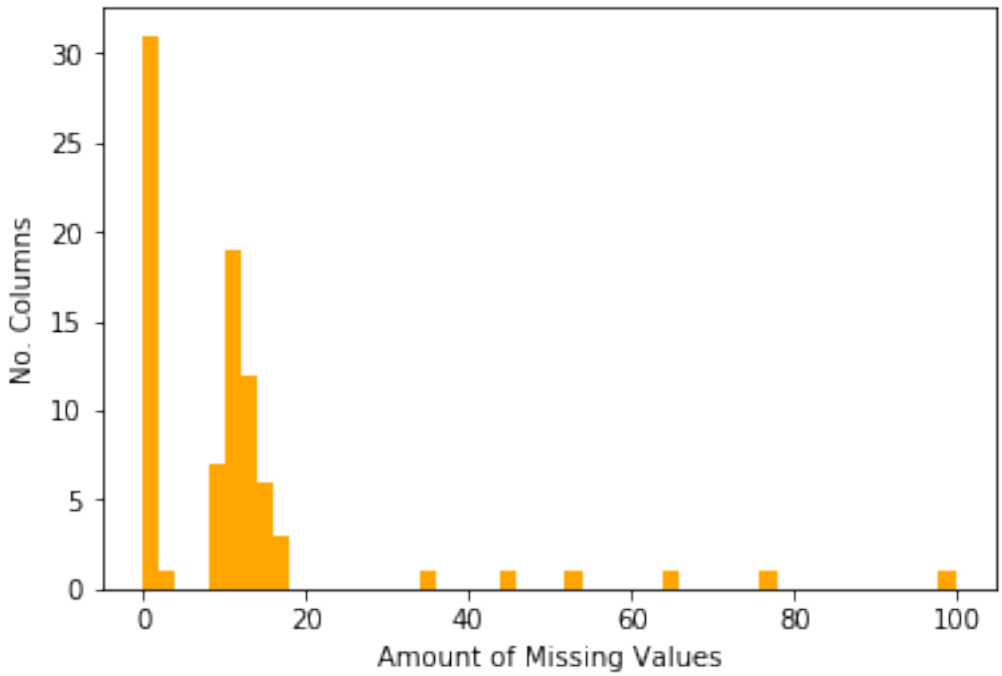
```
AGER_TYP                685843
ALTERSKATEGORIE_GROB      2881
ANREDE_KZ                    0
CJT_GESAMTTYP             4854
FINANZ_MINIMALIST            0
dtype: int64


count        85.000000
mean      98516.811765
std      146604.203317
min           0.000000
25%           0.000000
50%       93148.000000
75%      116515.000000
max      889061.000000
dtype: float64
The total number of misssing data 8373929
```

```
In [11]: percent_missing_columns = num_missing_columns/len(azdias)*100
         display(percent_missing_columns.head())
         print(percent_missing_columns.describe())
```

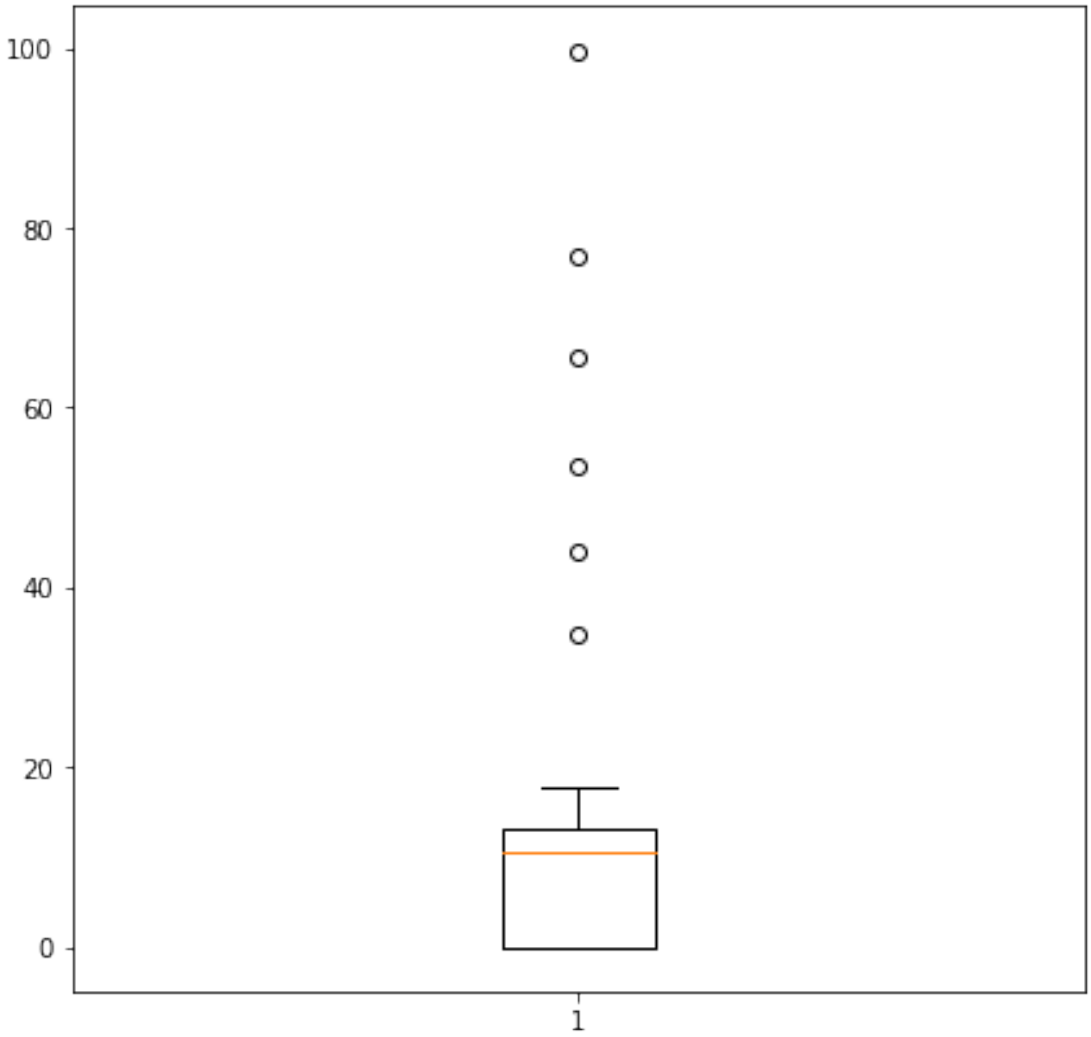
```
AGER_TYP                76.955435
ALTERSKATEGORIE_GROB     0.323264
ANREDE_KZ                0.000000
CJT_GESAMTTYP            0.544646
FINANZ_MINIMALIST        0.000000
dtype: float64


count    85.000000
mean     11.054139
```

```
std       16.449815
min        0.000000
25%        0.000000
50%       10.451729
75%       13.073637
max       99.757636
dtype: float64
```

In [12]:
```
# Investigate patterns in the amount of missing data in each column.
plt.hist(percent_missing_columns, bins = 50, color = 'orange')
plt.ylabel('No. Columns')
plt.xlabel('Amount of Missing Values')
plt.show()
```



In [13]:
```
# Remove the outlier columns from the dataset. (You'll perform other data
# engineering tasks such as re-encoding and imputation later.)

fig = plt.figure(figsize =(7, 7))
plt.boxplot(percent_missing_columns)
plt.show()
```




```
In [14]: columns = percent_missing_columns[percent_missing_columns > 20]
         columns_names = columns.keys().tolist()
         print(columns, columns_names)
         # display the dataset before removing outliers
         display(azdias.head())
```

```
AGER_TYP         76.955435
GEBURTSJAHR      44.020282
TITEL_KZ         99.757636
ALTER_HH         34.813699
KK_KUNDENTYP     65.596749
KBA05_BAUMAX     53.468668
dtype: float64 ['AGER_TYP', 'GEBURTSJAHR', 'TITEL_KZ', 'ALTER_HH', 'KK_KUNDENTYP', 'KBA05_BAUMAX
```

```
   AGER_TYP  ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  \
0       NaN                   2.0          1            2.0
1       NaN                   1.0          2            5.0
2       NaN                   3.0          2            3.0
3       2.0                   4.0          2            2.0
4       NaN                   3.0          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 [15]:
```
azdias = azdias.drop(columns_names, axis=1)
display(azdias.head())
```

```
   ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  FINANZ_MINIMALIST  \
0                   2.0          1            2.0                  3
1                   1.0          2            5.0                  1
2                   3.0          2            3.0                  1
3                   4.0          2            2.0                  4
4                   3.0          1            5.0                  4
```

```
   FINANZ_SPARER  FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  \
0              4                 3               5                      5
1              5                 2               5                      4
2              4                 1               2                      3
3              2                 5               2                      1
4              3                 4               1                      3

   FINANZ_HAUSBAUER  FINANZTYP    ...     PLZ8_ANTG1  PLZ8_ANTG2  PLZ8_ANTG3  \
0                 3          4    ...            NaN         NaN         NaN
1                 5          1    ...            2.0         3.0         2.0
2                 5          1    ...            3.0         3.0         1.0
3                 2          6    ...            2.0         2.0         2.0
4                 2          5    ...            2.0         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

[5 rows x 79 columns]
```

Discussion 1.1.2: Assess Missing Data in Each Column Upon generating a boxplot, it became evident that the outlier columns exhibited a prevalence of over 20% missing values. Specifically, there were six columns characterized by missing values exceeding 40%. This circumstance has the potential to introduce bias and inaccuracies into the model's predictions.

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.**

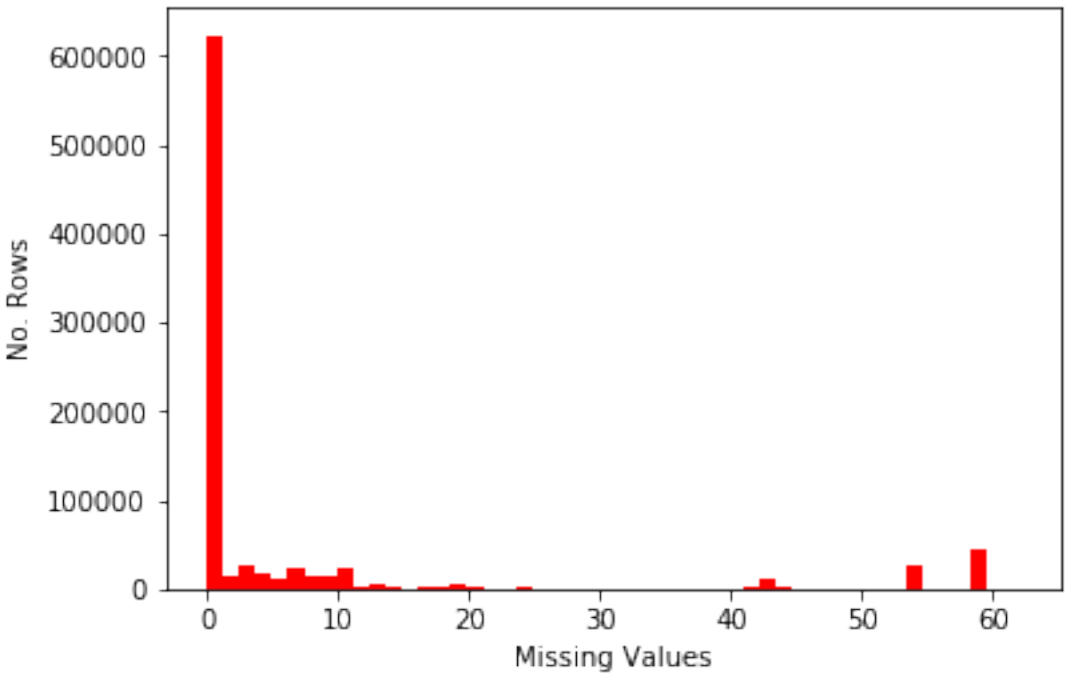
```
In [16]: # How much data is missing in each row of the dataset?

         num_missing_rows = azdias.isnull().sum(axis = 1)
         display(num_missing_rows.head())
         print("The total number of missing values", num_missing_rows.sum())
```

```
0    43
1     0
2     0
3     7
4     0
dtype: int64


The total number of missing values 5035304
```

```
In [17]: percent_missing_rows = num_missing_rows/azdias.shape[1]*100
         print(percent_missing_rows.describe())
```

```
count    891221.000000
mean          7.151764
std          16.752768
min           0.000000
25%           0.000000
50%           0.000000
75%           3.797468
max          62.025316
dtype: float64
```

```
In [18]: plt.hist(percent_missing_rows, bins=50, color = 'red')
         plt.ylabel('No. Rows')
         plt.xlabel('Missing Values')
         plt.show()
```



In [19]:
```
# Write code to divide the data into two subsets based on the number of missing
# values in each row.

low_missing_values_data = azdias[azdias.isnull().sum(axis=1) < 30]
high_missing_values_data = azdias[azdias.isnull().sum(axis=1) >= 30]

display(low_missing_values_data.head())
display(high_missing_values_data.head())
```

```
   ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  FINANZ_MINIMALIST  \
1                   1.0          2            5.0                  1
2                   3.0          2            3.0                  1
3                   4.0          2            2.0                  4
4                   3.0          1            5.0                  4
5                   1.0          2            2.0                  3

   FINANZ_SPARER  FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  \
1              5                 2               5                      4
2              4                 1               2                      3
3              2                 5               2                      1
4              3                 4               1                      3
5              1                 5               2                      2

   FINANZ_HAUSBAUER  FINANZTYP     ...      PLZ8_ANTG1  PLZ8_ANTG2  PLZ8_ANTG3  \
1                 5          1    ...             2.0         3.0         2.0
```

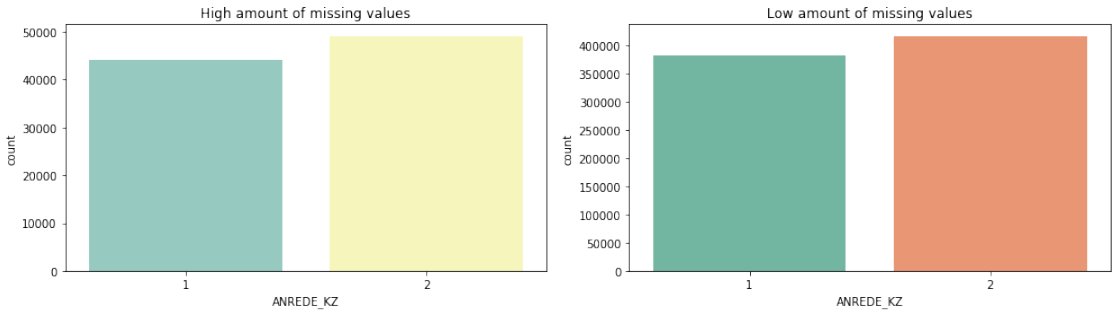
```
2                 5          1    ...            3.0         3.0         1.0
3                 2          6    ...            2.0         2.0         2.0
4                 2          5    ...            2.0         4.0         2.0
5                 5          2    ...            2.0         3.0         1.0

   PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  ORTSGR_KLS9  RELAT_AB
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
5         1.0          1.0       5.0       5.0     2.0          3.0       3.0

[5 rows x 79 columns]


    ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  FINANZ_MINIMALIST  \
0                    2.0          1            2.0                  3
11                   2.0          1            6.0                  3
14                   3.0          1            6.0                  3
17                   2.0          1            6.0                  3
24                   3.0          2            6.0                  3

    FINANZ_SPARER  FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  \
0               4                 3               5                      5
11              4                 3               5                      5
14              4                 3               5                      5
17              4                 3               5                      5
24              4                 3               5                      5

    FINANZ_HAUSBAUER  FINANZTYP    ...     PLZ8_ANTG1  PLZ8_ANTG2  PLZ8_ANTG3  \
0                  3          4    ...            NaN         NaN         NaN
11                 3          4    ...            NaN         NaN         NaN
14                 3          4    ...            NaN         NaN         NaN
17                 3          4    ...            NaN         NaN         NaN
24                 3          4    ...            NaN         NaN         NaN

    PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  ORTSGR_KLS9  RELAT_AB
0          NaN          NaN       NaN       NaN     NaN          NaN       NaN
11         NaN          NaN       NaN       NaN     NaN          NaN       NaN
14         NaN          NaN       NaN       NaN     NaN          NaN       NaN
17         NaN          NaN       NaN       NaN     NaN          NaN       NaN
24         NaN          NaN       NaN       NaN     NaN          NaN       NaN

[5 rows x 79 columns]
```

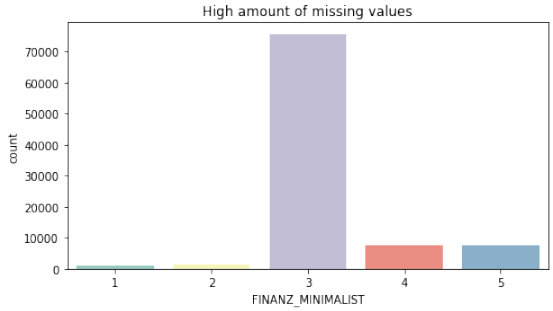
In [20]:
```
# Compare the distribution of values for at least five columns where there are
# no or few missing values, between the two subsets.
```

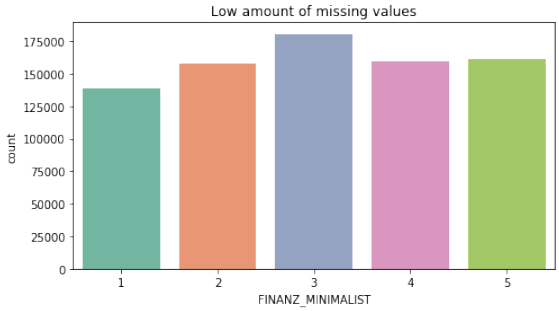
```
        complete_cols = percent_missing_columns[percent_missing_columns == 0]
        complete_cols = complete_cols[:8]
        complete_cols_names = complete_cols.keys().tolist()
        print(complete_cols, complete_cols_names)
```

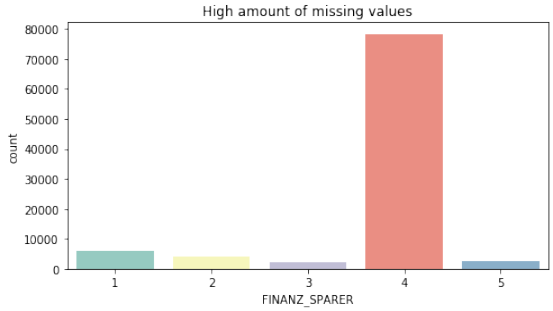
```
ANREDE_KZ               0.0
FINANZ_MINIMALIST       0.0
FINANZ_SPARER           0.0
FINANZ_VORSORGER        0.0
FINANZ_ANLEGER          0.0
FINANZ_UNAUFFAELLIGER   0.0
FINANZ_HAUSBAUER        0.0
FINANZTYP               0.0
dtype: float64 ['ANREDE_KZ', 'FINANZ_MINIMALIST', 'FINANZ_SPARER', 'FINANZ_VORSORGER', 'FINANZ_A
```

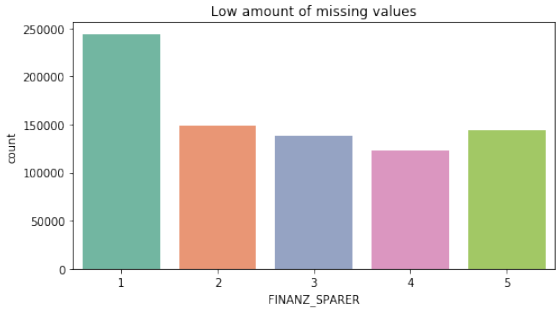
In [21]:
```
#function

        def compare_dist(col_name):
            fig = plt.figure(figsize=(14,4))
            ax1 = fig.add_subplot(121)
            ax1.title.set_text('High amount of missing values')
            sns.countplot(high_missing_values_data[col_name], palette = 'Set3')
            ax2 = fig.add_subplot(122)
            ax2.title.set_text('Low amount of missing values')
            sns.countplot(low_missing_values_data[col_name], palette = 'Set2')
            fig.tight_layout()
            plt.show()

        for col in complete_cols_names:
            compare_dist(col)
```

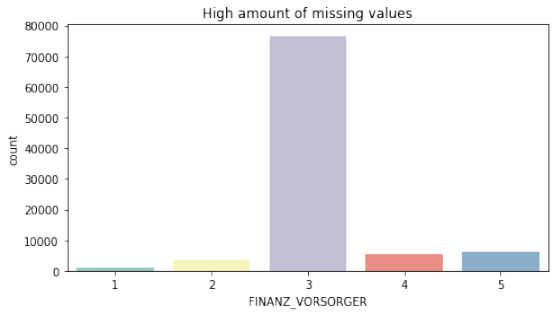


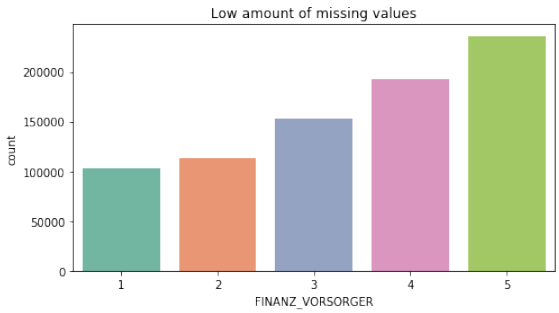

High amount of missing values
count
FINANZ_MINIMALIST

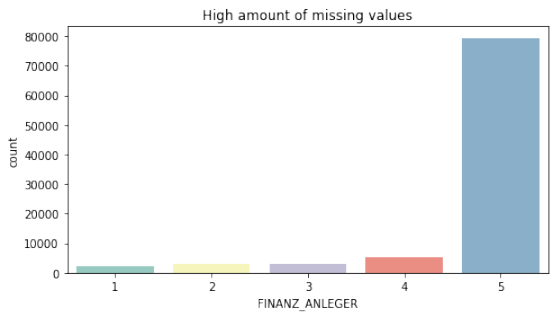

Low amount of missing values
count
FINANZ_MINIMALIST

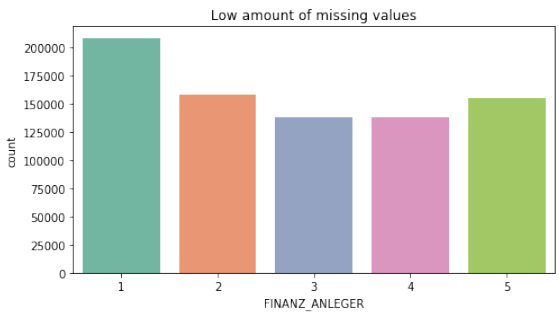

High amount of missing values
count
FINANZ_SPARER

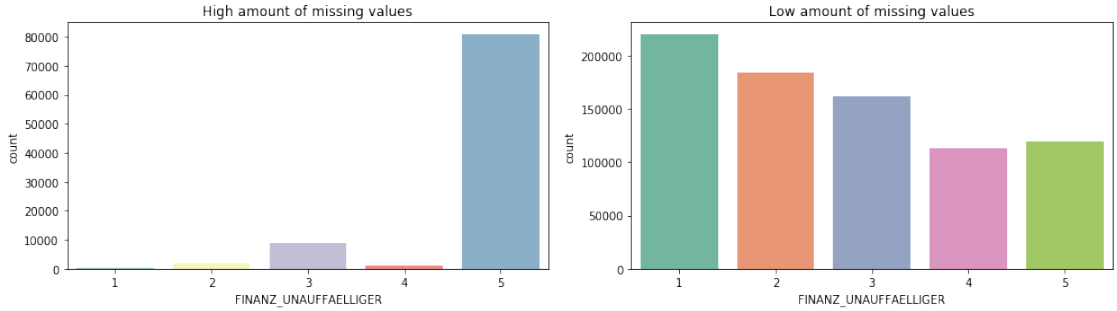

Low amount of missing values
count
FINANZ_SPARER

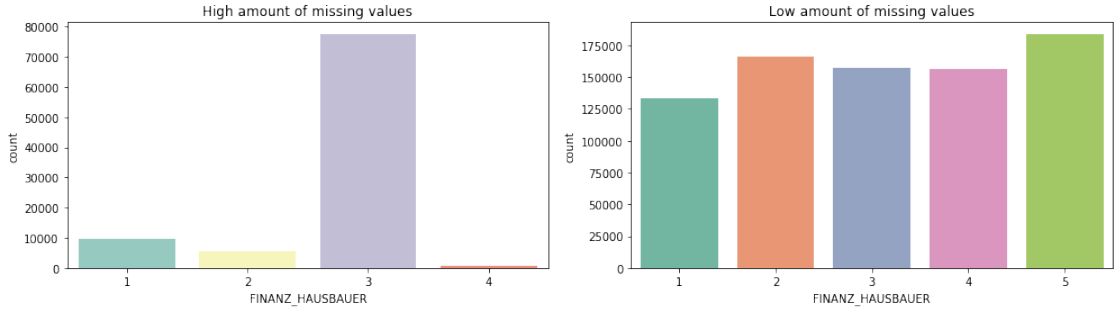

High amount of missing values
count
FINANZ_VORSORGER

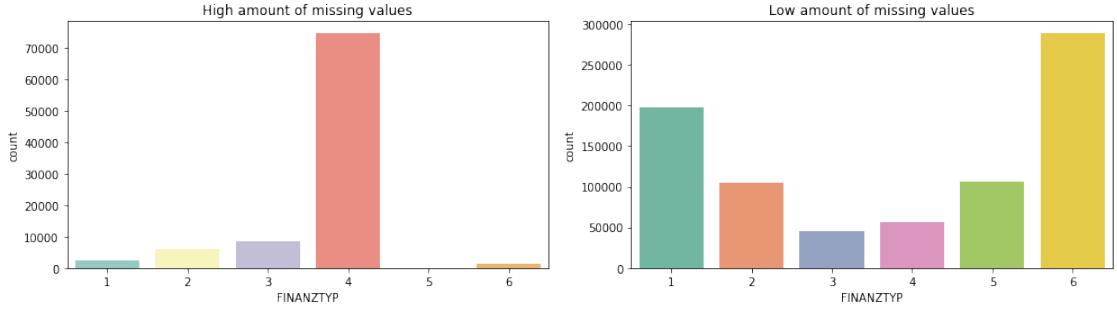

Low amount of missing values
count
FINANZ_VORSORGER


High amount of missing values
count
FINANZ_ANLEGER


Low amount of missing values
count
FINANZ_ANLEGER







Discussion 1.1.3: Assess Missing Data in Each Row Typically, datasets with a substantial proportion of missing values differ significantly from those without such gaps. This underscores the importance of favoring data containing minimal or no missing values, as utilizing the former can lead to erroneous predictions.

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 [22]: # How many features are there of each data type?

         features = azdias.keys().tolist()
         features = feat_info[feat_info['attribute'].isin(features)] #extracting the needed feat
         features['type'].value_counts()
```

```
Out[22]: 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 [23]: # Assess categorical variables: which are binary, which are multi-level, and
         # which one needs to be re-encoded?

         categorical_features = features[features['type'] == 'categorical']['attribute']

         display(features.head())
         display(categorical_features.head())
```

```
              attribute information_level         type missing_or_unknown
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]


2            ANREDE_KZ
3        CJT_GESAMTTYP
10           FINANZTYP
12     GFK_URLAUBERTYP
13    GREEN_AVANTGARDE
Name: attribute, dtype: object
```

```
In [24]: # Split the categorical features
         bi_features = []
         multi_features = []

         for feature in categorical_features:
             # check for multi categorical features
             if (low_missing_values_data[feature].nunique() > 2):
                 multi_features.append(feature)
             # check for binary categorical features
             else:
                 bi_features.append(feature)

         display(bi_features)
         display(multi_features)
```

```
['ANREDE_KZ', 'GREEN_AVANTGARDE', 'SOHO_KZ', 'VERS_TYP', 'OST_WEST_KZ']
```

```
['CJT_GESAMTTYP',
 'FINANZTYP',
 'GFK_URLAUBERTYP',
 'LP_FAMILIE_FEIN',
 'LP_FAMILIE_GROB',
 'LP_STATUS_FEIN',
 'LP_STATUS_GROB',
 'NATIONALITAET_KZ',
 'SHOPPER_TYP',
 'ZABEOTYP',
 'GEBAEUDETYP',
 'CAMEO_DEUG_2015',
 'CAMEO_DEU_2015']
```

```
In [25]: low_missing_values_data[bi_features].head()
```

```
Out[25]:    ANREDE_KZ  GREEN_AVANTGARDE  SOHO_KZ  VERS_TYP OST_WEST_KZ
         1          2                 0      1.0       2.0           W
```

```
2          2                1      0.0       1.0           W
3          2                0      0.0       1.0           W
4          1                0      0.0       2.0           W
5          2                0      0.0       2.0           W
```

In [26]:
```
low_missing_values_data['OST_WEST_KZ'] = low_missing_values_data['OST_WEST_KZ'].astype(


low_missing_values_data.loc[:, 'OST_WEST_KZ'].replace({'W': '0', '0': '1'}, inplace=Tru
```

In [27]:
```
low_missing_values_data[bi_features].head()
```

Out[27]:

	ANREDE_KZ	GREEN_AVANTGARDE	SOHO_KZ	VERS_TYP	OST_WEST_KZ
1	2	0	1.0	2.0	0
2	2	1	0.0	1.0	0
3	2	0	0.0	1.0	0
4	1	0	0.0	2.0	0
5	2	0	0.0	2.0	0

In [28]:
```
low_missing_values_data[multi_features].head()
```

Out[28]:

	CJT_GESAMTTYP	FINANZTYP	GFK_URLAUBERTYP	LP_FAMILIE_FEIN	\
1	5.0	1	10.0	5.0	
2	3.0	1	10.0	1.0	
3	2.0	6	1.0	NaN	
4	5.0	5	5.0	10.0	
5	2.0	2	1.0	1.0	

	LP_FAMILIE_GROB	LP_STATUS_FEIN	LP_STATUS_GROB	NATIONALITAET_KZ	\
1	3.0	2.0	1.0	1.0	
2	1.0	3.0	2.0	1.0	
3	NaN	9.0	4.0	1.0	
4	5.0	3.0	2.0	1.0	
5	1.0	4.0	2.0	1.0	

	SHOPPER_TYP	ZABEOTYP	GEBAEUDETYP	CAMEO_DEUG_2015	CAMEO_DEU_2015
1	3.0	5	8.0	8	8A
2	2.0	5	1.0	4	4C
3	1.0	3	1.0	2	2A
4	2.0	4	1.0	6	6B
5	0.0	4	1.0	8	8C

In [29]:
```
# Re-encode categorical variable(s) to be kept in the analysis.

low_missing_values_data = low_missing_values_data.drop(multi_features, axis = 1)
display(low_missing_values_data.head())
```

	ALTERSKATEGORIE_GROB	ANREDE_KZ	FINANZ_MINIMALIST	FINANZ_SPARER	\
1	1.0	2	1	5	

```
2                    3.0          2                    1              4
3                    4.0          2                    4              2
4                    3.0          1                    4              3
5                    1.0          2                    3              1

   FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER  \
1                 2               5                      4                 5
2                 1               2                      3                 5
3                 5               2                      1                 2
4                 4               1                      3                 2
5                 5               2                      2                 5

   GREEN_AVANTGARDE  HEALTH_TYP    ...     PLZ8_ANTG1  PLZ8_ANTG2  PLZ8_ANTG3  \
1                 0         3.0    ...            2.0         3.0         2.0
2                 1         3.0    ...            3.0         3.0         1.0
3                 0         2.0    ...            2.0         2.0         2.0
4                 0         3.0    ...            2.0         4.0         2.0
5                 0         3.0    ...            2.0         3.0         1.0

   PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  ORTSGR_KLS9  RELAT_AB
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
5         1.0          1.0       5.0       5.0     2.0          3.0       3.0

[5 rows x 66 columns]
```

Discussion 1.2.1: Re-Encode Categorical Features I retained all the numerical, ordinal, and mixed-type variables. However, for the categorical variables, I excluded those with multiple categories and retained only the binary ones. Additionally, I transformed one of the binary categorical features, OST_WEST_KZ, by mapping 'W' to 0 and 'O' to 1.

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 [30]: # Investigate "PRAEGENDE_JUGENDJAHRE" and engineer two new variables.

         mixed_features = features[features['type'] == 'mixed']['attribute']
         display(mixed_features.head())

15      LP_LEBENSPHASE_FEIN
16      LP_LEBENSPHASE_GROB
22    PRAEGENDE_JUGENDJAHRE
56                WOHNLAGE
59          CAMEO_INTL_2015
Name: attribute, dtype: object


In [31]: # Investigate "CAMEO_INTL_2015" and engineer two new variables.

         mixed_features = features[features['type'] == 'mixed']['attribute']
         display(mixed_features.head())

15      LP_LEBENSPHASE_FEIN
16      LP_LEBENSPHASE_GROB
22    PRAEGENDE_JUGENDJAHRE
56                WOHNLAGE
59          CAMEO_INTL_2015
Name: attribute, dtype: object


In [32]: display(low_missing_values_data['PRAEGENDE_JUGENDJAHRE'].head())

1    14.0
2    15.0
3     8.0
4     8.0
5     3.0
Name: PRAEGENDE_JUGENDJAHRE, dtype: float64


In [33]: decade = {1:1, 2:1, 3:2, 4:2, 5:3, 6:3, 7:3, 8:4, 9:4, 10:5, 11:5, 12:5, 13:5, 14:6, 15
         movement = {1:0, 2:1, 3:0, 4:1, 5:0, 6:1, 7:1, 8:0, 9:1, 10:0, 11:1, 12:0, 13:1, 14:0,


         low_missing_values_data['DECADE'] = low_missing_values_data['PRAEGENDE_JUGENDJAHRE']
         low_missing_values_data['DECADE'].replace(decade, inplace = True)

         low_missing_values_data['MOVEMENT'] = low_missing_values_data['PRAEGENDE_JUGENDJAHRE']
         low_missing_values_data['MOVEMENT'].replace(movement, inplace = True)

         display(low_missing_values_data['DECADE'].head())

         display(low_missing_values_data['MOVEMENT'].head())
```

```
        display(low_missing_values_data['CAMEO_INTL_2015'].head())
```

```
1    6.0
2    6.0
3    4.0
4    4.0
5    2.0
Name: DECADE, dtype: float64


1    0.0
2    1.0
3    0.0
4    0.0
5    0.0
Name: MOVEMENT, dtype: float64


1    51
2    24
3    12
4    43
5    54
Name: CAMEO_INTL_2015, dtype: object
```

```
In [34]: wealth = {
             '11':1, '12':1, '13':1, '14':1, '15':1,
             '21':2, '22':2, '23':2, '24':2, '25':2,
             '31':3, '32':3, '33':3, '34':3, '35':3,
             '41':4, '42':4, '43':4, '44':4, '45':4,
             '51':5, '52':5, '53':5, '54':5, '55':5,
         }

         life_stage ={
             '11':1, '12':2, '13':3, '14':4, '15':5,
             '21':1, '22':2, '23':3, '24':4, '25':5,
             '31':1, '32':2, '33':3, '34':4, '35':5,
             '41':1, '42':2, '43':3, '44':4, '45':5,
             '51':1, '52':2, '53':3, '54':4, '55':5,
         }


         low_missing_values_data['WEALTH'] = low_missing_values_data['CAMEO_INTL_2015']
         low_missing_values_data['WEALTH'].replace(wealth, inplace = True)

         low_missing_values_data['LIFE_STAGE'] = low_missing_values_data['CAMEO_INTL_2015']
         low_missing_values_data['LIFE_STAGE'].replace(life_stage, inplace = True)
```

```
In [35]: low_missing_values_data = low_missing_values_data.drop(mixed_features, axis = 1)

In [36]: display(low_missing_values_data.head())
```

	ALTERSKATEGORIE_GROB	ANREDE_KZ	FINANZ_MINIMALIST	FINANZ_SPARER	\
1	1.0	2	1	5	
2	3.0	2	1	4	
3	4.0	2	4	2	
4	3.0	1	4	3	
5	1.0	2	3	1	

	FINANZ_VORSORGER	FINANZ_ANLEGER	FINANZ_UNAUFFAELLIGER	FINANZ_HAUSBAUER	\
1	2	5	4	5	
2	1	2	3	5	
3	5	2	1	2	
4	4	1	3	2	
5	5	2	2	5	

	GREEN_AVANTGARDE	HEALTH_TYP	...	PLZ8_ANTG4	PLZ8_HHZ	PLZ8_GBZ	\
1	0	3.0	...	1.0	5.0	4.0	
2	1	3.0	...	0.0	4.0	4.0	
3	0	2.0	...	0.0	3.0	4.0	
4	0	3.0	...	1.0	3.0	3.0	
5	0	3.0	...	1.0	5.0	5.0	

	ARBEIT	ORTSGR_KLS9	RELAT_AB	DECADE	MOVEMENT	WEALTH	LIFE_STAGE
1	3.0	5.0	4.0	6.0	0.0	5.0	1.0
2	3.0	5.0	2.0	6.0	1.0	2.0	4.0
3	2.0	3.0	3.0	4.0	0.0	1.0	2.0
4	4.0	6.0	5.0	4.0	0.0	4.0	3.0
5	2.0	3.0	3.0	2.0	0.0	5.0	4.0

[5 rows x 64 columns]

Discussion 1.2.2: Engineer Mixed-Type Features I've introduced two additional columns with the necessary encodings for both CAMEO_INTL_2015 and PRAEGENDE_JUGENDJAHRE. Regarding the remaining mixed-type features, I've opted to remove them from the dataset altogether.

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 [37]: # If there are other re-engineering tasks you need to perform, make sure you
         # take care of them here. (Dealing with missing data will come in step 2.1.)

         print(low_missing_values_data.dtypes)
```

```
ALTERSKATEGORIE_GROB     float64
ANREDE_KZ                  int64
FINANZ_MINIMALIST          int64
FINANZ_SPARER              int64
FINANZ_VORSORGER           int64
FINANZ_ANLEGER             int64
FINANZ_UNAUFFAELLIGER      int64
FINANZ_HAUSBAUER           int64
GREEN_AVANTGARDE           int64
HEALTH_TYP               float64
RETOURTYP_BK_S           float64
SEMIO_SOZ                  int64
SEMIO_FAM                  int64
SEMIO_REL                  int64
SEMIO_MAT                  int64
SEMIO_VERT                 int64
SEMIO_LUST                 int64
SEMIO_ERL                  int64
SEMIO_KULT                 int64
SEMIO_RAT                  int64
SEMIO_KRIT                 int64
SEMIO_DOM                  int64
SEMIO_KAEM                 int64
SEMIO_PFLICHT              int64
SEMIO_TRADV                int64
SOHO_KZ                  float64
VERS_TYP                 float64
ANZ_PERSONEN             float64
ANZ_TITEL                float64
HH_EINKOMMEN_SCORE       float64
                          ...
KONSUMNAEHE              float64
MIN_GEBAEUDEJAHR         float64
OST_WEST_KZ               object
KBA05_ANTG1              float64
KBA05_ANTG2              float64
KBA05_ANTG3              float64
KBA05_ANTG4              float64
KBA05_GBZ                float64
BALLRAUM                 float64
```

```
EWDICHTE                float64
INNENSTADT              float64
GEBAEUDETYP_RASTER      float64
KKK                     float64
MOBI_REGIO              float64
ONLINE_AFFINITAET       float64
REGIOTYP                float64
KBA13_ANZAHL_PKW        float64
PLZ8_ANTG1              float64
PLZ8_ANTG2              float64
PLZ8_ANTG3              float64
PLZ8_ANTG4              float64
PLZ8_HHZ                float64
PLZ8_GBZ                float64
ARBEIT                  float64
ORTSGR_KLS9             float64
RELAT_AB                float64
DECADE                  float64
MOVEMENT                float64
WEALTH                  float64
LIFE_STAGE              float64
Length: 64, dtype: object
```

In [38]:
```
# Do whatever you need to in order to ensure that the dataframe only contains
# the columns that should be passed to the algorithm functions.

import numpy as np
import pandas as pd

# Get the column names
low_missing_columns = low_missing_values_data.columns.tolist()

# Function to fill missing values with the most frequent value
def fill_missing_most_frequent(column):
    most_frequent_value = column.value_counts().idxmax()
    return column.fillna(most_frequent_value)

# Apply the function to fill missing values in the DataFrame
low_missing_values_data = low_missing_values_data.apply(fill_missing_most_frequent)

# Convert the imputed data back to a DataFrame with original column names
low_missing_values_data = pd.DataFrame(low_missing_values_data, columns=low_missing_col

# Display the first 20 rows of the imputed DataFrame
display(low_missing_values_data.head(20))
```

```
   ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
```

1	1.0	2	1	5
2	3.0	2	1	4
3	4.0	2	4	2
4	3.0	1	4	3
5	1.0	2	3	1
6	2.0	2	1	5
7	1.0	1	3	3
8	3.0	1	4	4
9	3.0	2	2	4
10	3.0	2	2	2
12	3.0	1	5	3
13	1.0	2	1	4
15	4.0	2	4	1
16	1.0	2	4	3
18	2.0	2	2	4
19	3.0	1	5	2
20	2.0	2	4	3
21	2.0	1	3	4
22	1.0	1	1	5
23	3.0	1	5	3

	FINANZ_VORSORGER	FINANZ_ANLEGER	FINANZ_UNAUFFAELLIGER	FINANZ_HAUSBAUER	\
1	2	5	4	5	
2	1	2	3	5	
3	5	2	1	2	
4	4	1	3	2	
5	5	2	2	5	
6	1	5	4	3	
7	4	1	3	2	
8	2	4	2	2	
9	2	3	5	4	
10	5	3	1	5	
12	4	2	4	1	
13	3	5	5	2	
15	5	1	1	4	
16	1	4	5	1	
18	1	5	4	1	
19	3	1	3	1	
20	1	4	5	1	
21	1	2	5	1	
22	3	5	5	5	
23	3	2	2	1	

	GREEN_AVANTGARDE	HEALTH_TYP	...	PLZ8_ANTG4	PLZ8_HHZ	PLZ8_GBZ	\
1	0	3.0	...	1.0	5.0	4.0	
2	1	3.0	...	0.0	4.0	4.0	
3	0	2.0	...	0.0	3.0	4.0	
4	0	3.0	...	1.0	3.0	3.0	

5	0	3.0	...	1.0	5.0	5.0
6	0	2.0	...	0.0	5.0	5.0
7	0	1.0	...	0.0	4.0	4.0
8	1	3.0	...	1.0	3.0	3.0
9	1	2.0	...	1.0	3.0	3.0
10	0	2.0	...	0.0	3.0	3.0
12	0	1.0	...	0.0	5.0	5.0
13	1	3.0	...	1.0	3.0	3.0
15	0	2.0	...	0.0	3.0	3.0
16	0	3.0	...	0.0	3.0	4.0
18	0	2.0	...	1.0	3.0	3.0
19	1	3.0	...	1.0	5.0	4.0
20	1	3.0	...	0.0	3.0	3.0
21	0	2.0	...	2.0	4.0	3.0
22	0	2.0	...	1.0	4.0	3.0
23	1	3.0	...	0.0	3.0	3.0

	ARBEIT	ORTSGR_KLS9	RELAT_AB	DECADE	MOVEMENT	WEALTH	LIFE_STAGE
1	3.0	5.0	4.0	6.0	0.0	5.0	1.0
2	3.0	5.0	2.0	6.0	1.0	2.0	4.0
3	2.0	3.0	3.0	4.0	0.0	1.0	2.0
4	4.0	6.0	5.0	4.0	0.0	4.0	3.0
5	2.0	3.0	3.0	2.0	0.0	5.0	4.0
6	4.0	6.0	3.0	5.0	0.0	2.0	2.0
7	2.0	5.0	2.0	4.0	0.0	1.0	4.0
8	2.0	4.0	3.0	5.0	1.0	1.0	3.0
9	2.0	3.0	1.0	6.0	1.0	1.0	5.0
10	4.0	6.0	5.0	2.0	0.0	5.0	1.0
12	3.0	6.0	4.0	4.0	0.0	4.0	3.0
13	3.0	6.0	4.0	6.0	1.0	3.0	3.0
15	4.0	8.0	5.0	3.0	0.0	4.0	1.0
16	1.0	2.0	1.0	6.0	0.0	4.0	1.0
18	3.0	4.0	3.0	5.0	0.0	2.0	4.0
19	4.0	6.0	3.0	4.0	1.0	3.0	4.0
20	3.0	4.0	1.0	5.0	1.0	2.0	4.0
21	5.0	7.0	5.0	5.0	0.0	5.0	5.0
22	4.0	5.0	5.0	6.0	0.0	5.0	1.0
23	3.0	6.0	2.0	4.0	1.0	4.0	3.0

[20 rows x 64 columns]

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

              INPUT: Demographics DataFrame
              OUTPUT: Trimmed and cleaned demographics DataFrame
              """

              print("Raw Data:")
              display(df.head())

              # Put in code here to execute all main cleaning steps:
              # convert missing value codes into NaNs, ...
              for i in range(len(feat_info)):
                  missing_data = re.sub('[\[|\]]', '', feat_info.iloc[i]['missing_or_unknown']).s
                  if missing_data != ['']:
                      missing_data = [int(data) if (data != 'X' and data != 'XX') else data for d
                      df = df.replace({feat_info.iloc[i]['attribute']: missing_data}, np.nan)

              print("Data After Missing Values Conversions: ")
              display(df.head())

              # remove selected columns and rows, ...
              df = df.drop(columns_names, axis=1)
              df = df[df.isnull().sum(axis=1) < 30]

              print("Data After Removal of Selected Rows And Columns: ")
              display(df.head())

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

              # 1) convert the binary categorical feature to numerical values
              df.replace({'OST_WEST_KZ':{'W':0, 'O': 1}}, inplace = True)

                # 2) drop the multi categorical features
              df = df.drop(multi_features, axis = 1)

              # 3) re-encode the mixed values of PRAEGENDE_JUGENDJAHRE and CAMEO_INTL_2015
              df['DECADE'] = df['PRAEGENDE_JUGENDJAHRE']
              df['DECADE'].replace(decade, inplace = True)
              df['MOVEMENT'] = df['PRAEGENDE_JUGENDJAHRE']
              df['MOVEMENT'].replace(movement, inplace = True)

              df['WEALTH'] = df['CAMEO_INTL_2015']
```

```
    df['WEALTH'].replace(wealth, inplace = True)
    df['LIFE_STAGE'] = df['CAMEO_INTL_2015']
    df['LIFE_STAGE'].replace(life_stage, inplace = True)

    # 4) drop the rest of the mixed features
    df = df.drop(mixed_features, axis = 1)

    print("Data After Re-encoding And Feature Engineering: ")
    display(df.head())

#     # replacing the NaNs
#     df_cols = df.keys().tolist()
#     imp_mean = Imputer(missing_values = np.nan, strategy = 'most_frequent')
#     df = imp_mean.fit_transform(df)
#     df = pd.DataFrame(df, columns=df_cols)

    # Replace NaNs with the most frequent value in each column
    for col in df.columns:
        most_frequent_value = df[col].mode()[0]  # Get the most frequent value
        df[col].fillna(most_frequent_value, inplace=True)

    print("Data After Replacing NaN Values: ")
    display(df.head())
    # Return the cleaned dataframe.
    return df
```

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 [40]: # If you've not yet cleaned the dataset of all NaN values, then investigate and
         # do that now.

         azdias_general = pd.read_csv('Udacity_AZDIAS_Subset.csv', delimiter = ';')
         azdias_general = clean_data(azdias_general)
         azdias_general_columns = azdias_general.keys().tolist()
```

Raw Data:

```
   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]


Data After Missing Values Conversions:


   AGER_TYP  ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  \
0       NaN                   2.0          1            2.0
1       NaN                   1.0          2            5.0
2       NaN                   3.0          2            3.0
3       2.0                   4.0          2            2.0
4       NaN                   3.0          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]


Data After Removal of Selected Rows And Columns:


   ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  FINANZ_MINIMALIST  \
1                   1.0          2            5.0                  1
```

```
2                   3.0          2            3.0                   1
3                   4.0          2            2.0                   4
4                   3.0          1            5.0                   4
5                   1.0          2            2.0                   3

   FINANZ_SPARER  FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  \
1              5                 2               5                      4
2              4                 1               2                      3
3              2                 5               2                      1
4              3                 4               1                      3
5              1                 5               2                      2

   FINANZ_HAUSBAUER  FINANZTYP    ...     PLZ8_ANTG1  PLZ8_ANTG2  PLZ8_ANTG3  \
1                 5          1    ...            2.0         3.0         2.0
2                 5          1    ...            3.0         3.0         1.0
3                 2          6    ...            2.0         2.0         2.0
4                 2          5    ...            2.0         4.0         2.0
5                 5          2    ...            2.0         3.0         1.0

   PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  ORTSGR_KLS9  RELAT_AB
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
5         1.0         1.0       5.0       5.0     2.0          3.0       3.0

[5 rows x 79 columns]


Data After Re-encoding And Feature Engineering:


   ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
1                   1.0          2                  1              5
2                   3.0          2                  1              4
3                   4.0          2                  4              2
4                   3.0          1                  4              3
5                   1.0          2                  3              1

   FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER  \
1                 2               5                      4                 5
2                 1               2                      3                 5
3                 5               2                      1                 2
4                 4               1                      3                 2
5                 5               2                      2                 5

   GREEN_AVANTGARDE  HEALTH_TYP     ...     PLZ8_ANTG4  PLZ8_HHZ  PLZ8_GBZ  \
1                 0         3.0    ...            1.0       5.0       4.0
```

```
2                 1         3.0     ...             0.0       4.0       4.0
3                 0         2.0     ...             0.0       3.0       4.0
4                 0         3.0     ...             1.0       3.0       3.0
5                 0         3.0     ...             1.0       5.0       5.0

   ARBEIT  ORTSGR_KLS9  RELAT_AB  DECADE  MOVEMENT  WEALTH  LIFE_STAGE
1     3.0          5.0       4.0     6.0       0.0     5.0         1.0
2     3.0          5.0       2.0     6.0       1.0     2.0         4.0
3     2.0          3.0       3.0     4.0       0.0     1.0         2.0
4     4.0          6.0       5.0     4.0       0.0     4.0         3.0
5     2.0          3.0       3.0     2.0       0.0     5.0         4.0

[5 rows x 64 columns]


Data After Replacing NaN Values:


   ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
1                   1.0          2                  1              5
2                   3.0          2                  1              4
3                   4.0          2                  4              2
4                   3.0          1                  4              3
5                   1.0          2                  3              1

   FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER  \
1                 2               5                      4                 5
2                 1               2                      3                 5
3                 5               2                      1                 2
4                 4               1                      3                 2
5                 5               2                      2                 5

   GREEN_AVANTGARDE  HEALTH_TYP     ...      PLZ8_ANTG4  PLZ8_HHZ  PLZ8_GBZ  \
1                 0         3.0     ...             1.0       5.0       4.0
2                 1         3.0     ...             0.0       4.0       4.0
3                 0         2.0     ...             0.0       3.0       4.0
4                 0         3.0     ...             1.0       3.0       3.0
5                 0         3.0     ...             1.0       5.0       5.0

   ARBEIT  ORTSGR_KLS9  RELAT_AB  DECADE  MOVEMENT  WEALTH  LIFE_STAGE
1     3.0          5.0       4.0     6.0       0.0     5.0         1.0
2     3.0          5.0       2.0     6.0       1.0     2.0         4.0
3     2.0          3.0       3.0     4.0       0.0     1.0         2.0
4     4.0          6.0       5.0     4.0       0.0     4.0         3.0
5     2.0          3.0       3.0     2.0       0.0     5.0         4.0

[5 rows x 64 columns]
```

```
In [41]: fill_missing = Imputer(strategy='most_frequent')
         azdias_clean_imputed = pd.DataFrame(fill_missing.fit_transform(azdias_general))

In [42]: # Apply feature scaling to the general population demographics data.

         scaler = StandardScaler()
         azdias_general = scaler.fit_transform(azdias_general)
```

1.2.2 Discussion 2.1: Apply Feature Scaling

I loaded the azdias dataset and applied essential transformations using the clean_data function. Afterward, I conducted feature scaling on the azdias dataset using the StandardScaler.

1.2.3 Step 2.2: Perform Dimensionality Reduction

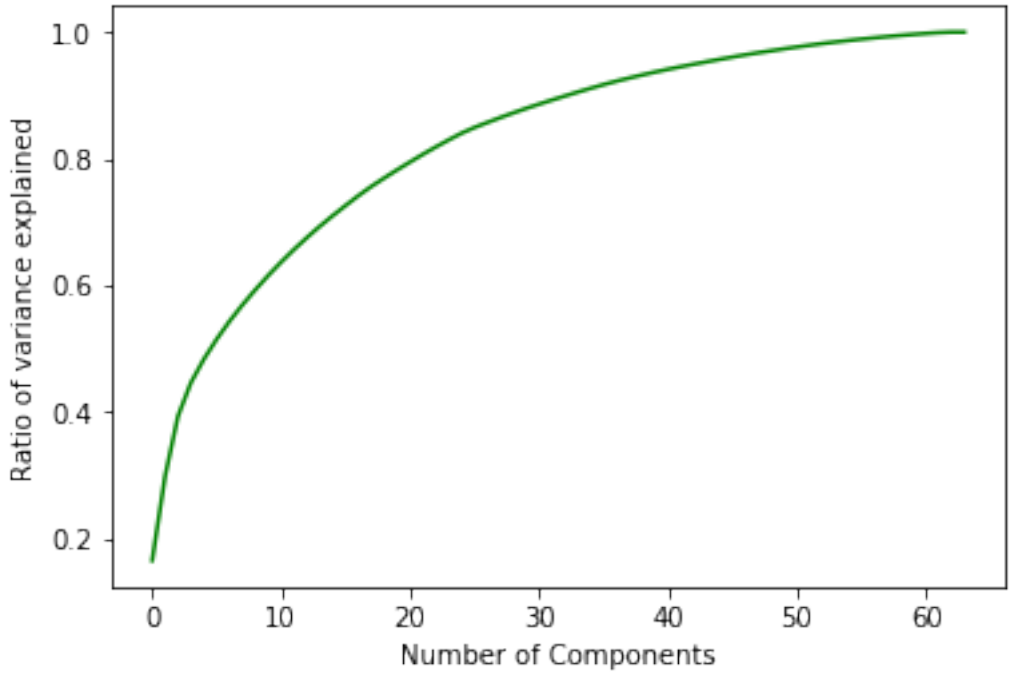
On your scaled data, you are now ready to apply dimensionality reduction techniques.

- 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 [43]: # Apply PCA to the data.

         pca = PCA()
         pca.fit(azdias_general)

Out[43]: PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
           svd_solver='auto', tol=0.0, whiten=False)

In [44]: # Investigate the variance accounted for by each principal component.

         from sklearn.decomposition import PCA

         # Apply PCA to the data
         pca_test = PCA()
         pca_test.fit(azdias_general)

         # Print the explained variance ratios
         print("Explained Variance Ratios:")
         print(pca_test.explained_variance_ratio_)
```

```
Explained Variance Ratios:
[  1.65510980e-01   1.33144261e-01   9.47137708e-02   5.28523205e-02
   3.69563856e-02   3.17986073e-02   2.84191333e-02   2.58211468e-02
   2.38201593e-02   2.25718558e-02   2.11125706e-02   1.99924688e-02
   1.87472878e-02   1.77909086e-02   1.67309604e-02   1.60705755e-02
   1.56198862e-02   1.46012830e-02   1.32771054e-02   1.28677578e-02
   1.23969879e-02   1.21999527e-02   1.15800487e-02   1.11276251e-02
   1.06557113e-02   8.97404907e-03   7.88584978e-03   7.70959885e-03
   7.38972486e-03   7.08564062e-03   6.72406413e-03   6.50164710e-03
   6.31738200e-03   6.27317394e-03   5.93578191e-03   5.81208819e-03
   5.47346217e-03   4.97806492e-03   4.84186464e-03   4.59646141e-03
   4.35045000e-03   4.08767872e-03   3.95077724e-03   3.83820339e-03
   3.79588860e-03   3.63823590e-03   3.55068184e-03   3.26232650e-03
   3.10592814e-03   3.08225706e-03   2.92499673e-03   2.75931648e-03
   2.71285627e-03   2.47586014e-03   2.26441706e-03   2.22435164e-03
   2.01321738e-03   1.93602584e-03   1.79966908e-03   1.65103823e-03
   1.55653509e-03   1.29283646e-03   8.47848670e-04   2.50042854e-29]
```

In [45]:
```
# Re-apply PCA to the data while selecting for number of components to retain.

plt.plot(range(pca_test.explained_variance_ratio_.shape[0]), np.cumsum(pca_test.explain
plt.xlabel("Number of Components")
plt.ylabel("Ratio of variance explained")
plt.show()
```



```
In [46]: pca = PCA(n_components = 30)
         azdias_data_pca = pca.fit_transform(azdias_general)

In [47]: print(azdias_data_pca.shape)

(798061, 30)
```

1.2.4 Discussion 2.2: Perform Dimensionality Reduction

I chose to retain 30 principal components since they account for over 88% of the variance while reducing the feature count by more than 50%.

1.2.5 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 [48]: # 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 weight_map(pca, i):
             data = pd.DataFrame(pca.components_, columns = azdias_general_columns)
             return data.iloc[i].sort_values(ascending = False)


         weight_map(pca , 0)
```

Out[48]:
```
PLZ8_ANTG3                0.225345
PLZ8_ANTG4                0.216873
WEALTH                    0.204628
HH_EINKOMMEN_SCORE        0.202187
ORTSGR_KLS9               0.196765
EWDICHTE                  0.194654
FINANZ_HAUSBAUER          0.159510
KBA05_ANTG4               0.153988
PLZ8_ANTG2                0.153685
FINANZ_SPARER             0.153109
ARBEIT                    0.142565
KBA05_ANTG3               0.136723
ANZ_HAUSHALTE_AKTIV       0.136056
RELAT_AB                  0.134948
SEMIO_PFLICHT             0.121171
SEMIO_REL                 0.118599
DECADE                    0.112256
SEMIO_RAT                 0.099580
SEMIO_TRADV               0.093724
SEMIO_MAT                 0.082442
FINANZ_UNAUFFAELLIGER     0.081093
SEMIO_FAM                 0.081012
SEMIO_KULT                0.075521
FINANZ_ANLEGER            0.075106
REGIOTYP                  0.060337
OST_WEST_KZ               0.053663
SEMIO_SOZ                 0.043295
PLZ8_HHZ                  0.042248
HEALTH_TYP                0.041221
KKK                       0.039587
                             ...
SEMIO_DOM                 0.023777
KBA05_ANTG2               0.013725
ANREDE_KZ                 0.007528
SEMIO_KRIT                0.003808
SOHO_KZ                  -0.001989
ANZ_TITEL                -0.004228
RETOURTYP_BK_S           -0.021723
SEMIO_VERT               -0.040621
ONLINE_AFFINITAET        -0.041278
MIN_GEBAEUDEJAHR         -0.042954
WOHNDAUER_2008           -0.061243
KBA13_ANZAHL_PKW         -0.073929
SEMIO_LUST               -0.076894
ANZ_PERSONEN             -0.078068
SEMIO_ERL                -0.080173
MOVEMENT                 -0.110213
GREEN_AVANTGARDE         -0.110213
```

```
GEBAEUDETYP_RASTER      -0.117185
FINANZ_VORSORGER        -0.120183
LIFE_STAGE              -0.125163
ALTERSKATEGORIE_GROB    -0.125613
BALLRAUM                -0.127346
INNENSTADT              -0.164639
PLZ8_GBZ                -0.166380
KONSUMNAEHE             -0.167395
KBA05_ANTG1             -0.214438
KBA05_GBZ               -0.216061
FINANZ_MINIMALIST       -0.223092
MOBI_REGIO              -0.224811
PLZ8_ANTG1              -0.225665
Name: 0, Length: 64, dtype: float64
```

In [49]:
```
# Map weights for the second principal component to corresponding feature names
# and then print the linked values, sorted by weight.

weight_map(pca , 1)
```

Out[49]:
```
ALTERSKATEGORIE_GROB     0.256265
SEMIO_ERL                0.229915
FINANZ_VORSORGER         0.229239
SEMIO_LUST               0.180110
RETOURTYP_BK_S           0.161878
FINANZ_HAUSBAUER         0.121830
SEMIO_KRIT               0.117850
SEMIO_KAEM               0.116437
W_KEIT_KIND_HH           0.114740
PLZ8_ANTG3               0.098404
EWDICHTE                 0.097909
ORTSGR_KLS9              0.096589
PLZ8_ANTG4               0.096257
ANREDE_KZ                0.093140
WEALTH                   0.079218
KBA05_ANTG4              0.075239
SEMIO_DOM                0.074492
ARBEIT                   0.071835
RELAT_AB                 0.069307
PLZ8_ANTG2               0.068172
ANZ_HAUSHALTE_AKTIV      0.066489
HH_EINKOMMEN_SCORE       0.061967
FINANZ_MINIMALIST        0.059375
WOHNDAUER_2008           0.058466
KBA05_ANTG3              0.050577
VERS_TYP                 0.032144
ANZ_HH_TITEL             0.032079
OST_WEST_KZ              0.027484
```

```
PLZ8_HHZ                 0.015914
REGIOTYP                 0.011859
                            ...
SOHO_KZ                 -0.001965
GREEN_AVANTGARDE        -0.018376
MOVEMENT                -0.018376
KBA13_ANZAHL_PKW        -0.038001
GEBAEUDETYP_RASTER      -0.047251
MIN_GEBAEUDEJAHR        -0.049020
HEALTH_TYP              -0.056973
ANZ_PERSONEN            -0.064181
BALLRAUM                -0.064699
SEMIO_VERT              -0.072776
KBA05_ANTG1             -0.074407
KONSUMNAEHE             -0.074798
PLZ8_GBZ                -0.075153
INNENSTADT              -0.079661
MOBI_REGIO              -0.080841
KBA05_GBZ               -0.092366
PLZ8_ANTG1              -0.096339
SEMIO_SOZ               -0.103209
SEMIO_MAT               -0.161720
ONLINE_AFFINITAET       -0.165413
SEMIO_RAT               -0.166974
SEMIO_FAM               -0.184303
FINANZ_ANLEGER          -0.202721
SEMIO_KULT              -0.219735
SEMIO_PFLICHT           -0.225445
FINANZ_UNAUFFAELLIGER   -0.225492
SEMIO_TRADV             -0.228273
FINANZ_SPARER           -0.231660
DECADE                  -0.238553
SEMIO_REL               -0.253635
Name: 1, Length: 64, dtype: float64
```

In [50]:
```
# Map weights for the third principal component to corresponding feature names
# and then print the linked values, sorted by weight.

weight_map(pca , 2)
```

Out[50]:
```
SEMIO_VERT              0.344664
SEMIO_SOZ               0.261932
SEMIO_FAM               0.248836
SEMIO_KULT              0.233936
FINANZ_MINIMALIST       0.154493
RETOURTYP_BK_S          0.108975
FINANZ_VORSORGER        0.101711
W_KEIT_KIND_HH          0.085079
```

ALTERSKATEGORIE_GROB	0.079618
SEMIO_REL	0.067673
SEMIO_LUST	0.064452
SEMIO_MAT	0.055671
ORTSGR_KLS9	0.050284
EWDICHTE	0.049493
PLZ8_ANTG4	0.049205
PLZ8_ANTG3	0.048410
GREEN_AVANTGARDE	0.047104
MOVEMENT	0.047104
ARBEIT	0.037364
RELAT_AB	0.034366
PLZ8_ANTG2	0.032674
WOHNDAUER_2008	0.032648
WEALTH	0.030183
KBA05_ANTG4	0.029997
ANZ_HAUSHALTE_AKTIV	0.026814
OST_WEST_KZ	0.016319
ANZ_HH_TITEL	0.013918
KBA05_ANTG3	0.011904
ANZ_TITEL	0.009635
PLZ8_HHZ	0.005637
	...
LIFE_STAGE	-0.010033
ANZ_PERSONEN	-0.010726
KKK	-0.015512
HH_EINKOMMEN_SCORE	-0.016394
MIN_GEBAEUDEJAHR	-0.018361
KBA05_ANTG1	-0.022127
KBA13_ANZAHL_PKW	-0.023908
MOBI_REGIO	-0.026795
KBA05_GBZ	-0.029017
GEBAEUDETYP_RASTER	-0.032111
HEALTH_TYP	-0.034073
BALLRAUM	-0.037239
FINANZ_HAUSBAUER	-0.039724
KONSUMNAEHE	-0.040451
PLZ8_GBZ	-0.040540
INNENSTADT	-0.045748
PLZ8_ANTG1	-0.049401
ONLINE_AFFINITAET	-0.055715
SEMIO_TRADV	-0.079278
SEMIO_PFLICHT	-0.080066
FINANZ_UNAUFFAELLIGER	-0.101115
FINANZ_SPARER	-0.107210
DECADE	-0.111703
SEMIO_ERL	-0.175285
FINANZ_ANLEGER	-0.190987

```
SEMIO_RAT               -0.217717
SEMIO_KRIT              -0.275200
SEMIO_DOM               -0.313070
SEMIO_KAEM              -0.335050
ANREDE_KZ               -0.367353
Name: 2, Length: 64, dtype: float64
```

1.2.6 Discussion 2.3: Interpret Principal Components

Principal Component 1: The variables PLZ8_ANTG3 (Number of 6-10 family houses in the PLZ8 region) and PLZ8_ANTG4 (Number of 10+ family houses in the PLZ8 region) exhibit a strong positive correlation, while SEMIO_VERT (dreamful personality topology) and ANREDE_KZ (Gender) display a negative correlation.

Principal Component 2: ALTERSKATEGORIE_GROB (Estimated age based on given name analysis) and SEMIO_ERL (event-oriented personality topology) demonstrate a significant positive correlation.

Principal Component 3: SEMIO_VERT and SEMIO_SOZ (socially-minded personality topology) exhibit a substantial positive correlation.

1.3 Step 3: Clustering

1.3.1 Step 3.1: Apply Clustering to General Population

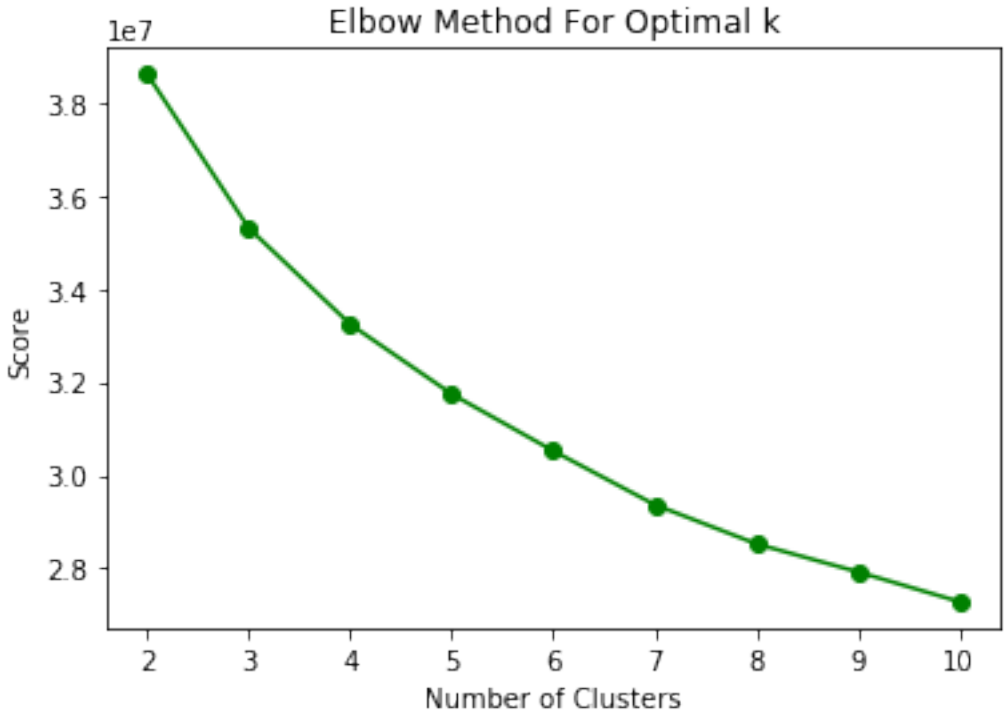
You've assessed and cleaned the demographics data, then scaled and transformed them. Now, it's time to see how the data clusters in the principal components space. In this substep, you will apply k-means clustering to the dataset and use the average within-cluster distances from each point to their assigned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

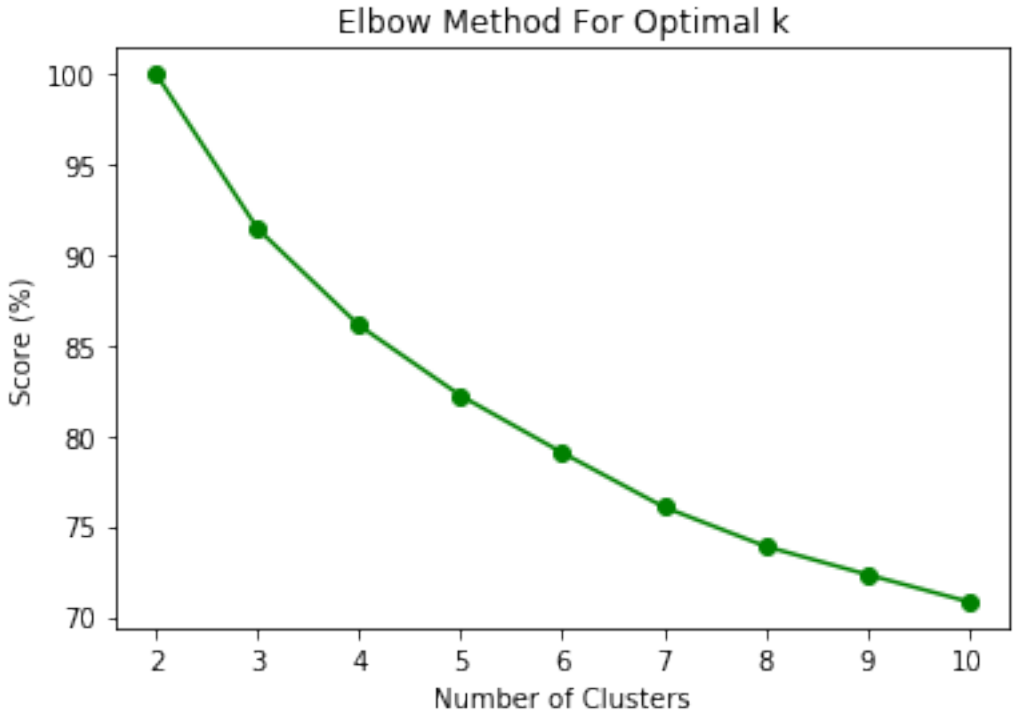
In [51]:
```
from sklearn.cluster import KMeans

clusters = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
scores = []
for cluster in clusters:
```

```
    kmeans = KMeans(n_clusters = cluster)
    model = kmeans.fit(azdias_data_pca)
    score = np.abs(model.score(azdias_data_pca))
    scores.append(score)
```

In [52]:
```
# Investigate the change in within-cluster distance across number of clusters.
# HINT: Use matplotlib's plot function to visualize this relationship.

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10]
scores = []

for cluster in clusters:
    kmeans = KMeans(n_clusters=cluster)
    model = kmeans.fit(azdias_data_pca)
    score = np.abs(model.score(azdias_data_pca))
    scores.append(score)

plt.plot(clusters, scores, marker='o', color='green')
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Method For Optimal k')
plt.show()
```



In [53]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np

clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10]
scores = []

for cluster in clusters:
    kmeans = KMeans(n_clusters=cluster)
    model = kmeans.fit(azdias_data_pca)
    score = np.abs(model.score(azdias_data_pca))
    scores.append(score)

scores_percent = [(score / scores[0]) * 100 for score in scores]  # Convert scores to p

plt.plot(clusters, scores_percent, marker='o', color='green')
plt.xlabel('Number of Clusters')
plt.ylabel('Score (%)')
plt.title('Elbow Method For Optimal k')
plt.show()
```



In [54]:
```
# Re-fit the k-means model with the selected number of clusters and obtain
# cluster predictions for the general population demographics data.
```

```
kmeans = KMeans(n_clusters = 15)
model = kmeans.fit(azdias_data_pca)
predections_azdias_data = model.predict(azdias_data_pca)
```

1.3.2 Discussion 3.1: Apply Clustering to General Population

After employing the elbow method and visualizing the relationship between the number of clusters and the corresponding scores, it was determined that the optimal number of clusters should be 15.

1.3.3 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 [55]: # Load in the customer demographics data.
         customers = pd.read_csv('Udacity_CUSTOMERS_Subset.csv', delimiter = ';')
         display(customers.head())
```

```
   AGER_TYP  ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  \
0         2                     4          1            5.0
1        -1                     4          1            NaN
2        -1                     4          2            2.0
3         1                     4          1            2.0
4        -1                     3          1            6.0

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

   FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER    ...     PLZ8_ANTG1  PLZ8_ANTG2  \
```

```
0                     2                 2    ...            3.0         3.0
1                     3                 2    ...            NaN         NaN
2                     4                 4    ...            2.0         3.0
3                     1                 2    ...            3.0         2.0
4                     5                 2    ...            2.0         4.0

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

   ORTSGR_KLS9  RELAT_AB
0          2.0       1.0
1          NaN       NaN
2          5.0       3.0
3          3.0       1.0
4          5.0       1.0

[5 rows x 85 columns]
```

In [56]:
```
# Apply preprocessing, feature transformation, and clustering from the general
# demographics onto the customer data, obtaining cluster predictions for the
# customer demographics data.
fill_missing = Imputer(strategy='most_frequent')
azdias_clean_imputed = pd.DataFrame(fill_missing.fit_transform(azdias_general))
customers = clean_data(customers)
```

```
Raw Data:


   AGER_TYP  ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  \
0         2                     4          1            5.0
1        -1                     4          1            NaN
2        -1                     4          2            2.0
3         1                     4          1            2.0
4        -1                     3          1            6.0

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

   FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER    ...     PLZ8_ANTG1  PLZ8_ANTG2  \
```

```
0                    2                2    ...            3.0         3.0
1                    3                2    ...            NaN         NaN
2                    4                4    ...            2.0         3.0
3                    1                2    ...            3.0         2.0
4                    5                2    ...            2.0         4.0

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

   ORTSGR_KLS9  RELAT_AB
0          2.0       1.0
1          NaN       NaN
2          5.0       3.0
3          3.0       1.0
4          5.0       1.0

[5 rows x 85 columns]


Data After Missing Values Conversions:


   AGER_TYP  ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  \
0       2.0                   4.0          1            5.0
1       NaN                   4.0          1            NaN
2       NaN                   4.0          2            2.0
3       1.0                   4.0          1            2.0
4       NaN                   3.0          1            6.0

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

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

   PLZ8_ANTG3  PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  \
```

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

   ORTSGR_KLS9  RELAT_AB
0          2.0       1.0
1          NaN       NaN
2          5.0       3.0
3          3.0       1.0
4          5.0       1.0

[5 rows x 85 columns]


Data After Removal of Selected Rows And Columns:


   ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  FINANZ_MINIMALIST  \
0                   4.0          1            5.0                  5
2                   4.0          2            2.0                  5
3                   4.0          1            2.0                  5
4                   3.0          1            6.0                  3
5                   3.0          1            4.0                  5

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

   FINANZ_HAUSBAUER  FINANZTYP    ...     PLZ8_ANTG1  PLZ8_ANTG2  PLZ8_ANTG3  \
0                 2          2    ...            3.0         3.0         1.0
2                 4          2    ...            2.0         3.0         3.0
3                 2          6    ...            3.0         2.0         1.0
4                 2          2    ...            2.0         4.0         2.0
5                 3          5    ...            2.0         3.0         2.0

   PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  ORTSGR_KLS9  RELAT_AB
0         0.0          1.0       5.0       5.0     1.0          2.0       1.0
2         1.0          3.0       3.0       2.0     3.0          5.0       3.0
3         0.0          1.0       3.0       4.0     1.0          3.0       1.0
4         1.0          2.0       3.0       3.0     3.0          5.0       1.0
5         1.0          1.0       5.0       5.0     3.0          7.0       5.0

[5 rows x 79 columns]
```

Data After Re-encoding And Feature Engineering:

```
   ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
0                   4.0          1                  5              1
2                   4.0          2                  5              1
3                   4.0          1                  5              1
4                   3.0          1                  3              1
5                   3.0          1                  5              1

   FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER  \
0                 5               1                      2                 2
2                 5               1                      4                 4
3                 5               2                      1                 2
4                 4               4                      5                 2
5                 5               1                      2                 3

   GREEN_AVANTGARDE  HEALTH_TYP     ...      PLZ8_ANTG4  PLZ8_HHZ  PLZ8_GBZ  \
0                 1         1.0     ...             0.0       5.0       5.0
2                 1         2.0     ...             1.0       3.0       2.0
3                 0         2.0     ...             0.0       3.0       4.0
4                 0         3.0     ...             1.0       3.0       3.0
5                 1         3.0     ...             1.0       5.0       5.0

   ARBEIT  ORTSGR_KLS9  RELAT_AB  DECADE  MOVEMENT  WEALTH  LIFE_STAGE
0     1.0          2.0       1.0     2.0       1.0     1.0         3.0
2     3.0          5.0       3.0     2.0       1.0     3.0         4.0
3     1.0          3.0       1.0     1.0       0.0     2.0         4.0
4     3.0          5.0       1.0     4.0       0.0     4.0         1.0
5     3.0          7.0       5.0     2.0       1.0     3.0         4.0

[5 rows x 64 columns]
```

Data After Replacing NaN Values:

```
   ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
0                   4.0          1                  5              1
2                   4.0          2                  5              1
3                   4.0          1                  5              1
4                   3.0          1                  3              1
5                   3.0          1                  5              1

   FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER  \
0                 5               1                      2                 2
2                 5               1                      4                 4
```

```
3                5              2                     1                2
4                4              4                     5                2
5                5              1                     2                3

   GREEN_AVANTGARDE  HEALTH_TYP     ...      PLZ8_ANTG4  PLZ8_HHZ  PLZ8_GBZ  \
0                 1         1.0     ...             0.0       5.0       5.0
2                 1         2.0     ...             1.0       3.0       2.0
3                 0         2.0     ...             0.0       3.0       4.0
4                 0         3.0     ...             1.0       3.0       3.0
5                 1         3.0     ...             1.0       5.0       5.0

   ARBEIT  ORTSGR_KLS9  RELAT_AB  DECADE  MOVEMENT  WEALTH  LIFE_STAGE
0     1.0          2.0       1.0     2.0       1.0     1.0         3.0
2     3.0          5.0       3.0     2.0       1.0     3.0         4.0
3     1.0          3.0       1.0     1.0       0.0     2.0         4.0
4     3.0          5.0       1.0     4.0       0.0     4.0         1.0
5     3.0          7.0       5.0     2.0       1.0     3.0         4.0

[5 rows x 64 columns]
```

```
In [57]: customers = scaler.transform(customers)
         customers_pca = pca.transform(customers)
         predections_customers = model.predict(customers_pca)
```

1.3.4 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.

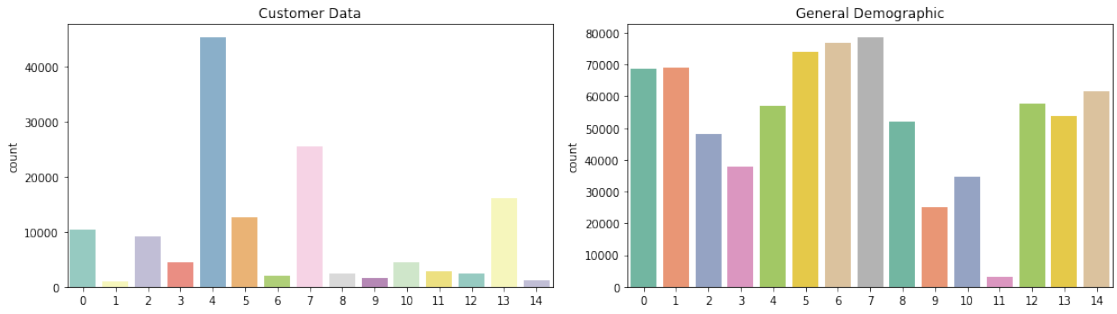
Consider the proportion of persons in each cluster for the general population, and the proportions for the customers. If we think the company's customer base to be universal, then the cluster assignment proportions should be fairly similar between the two. If there are only particular segments of the 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.

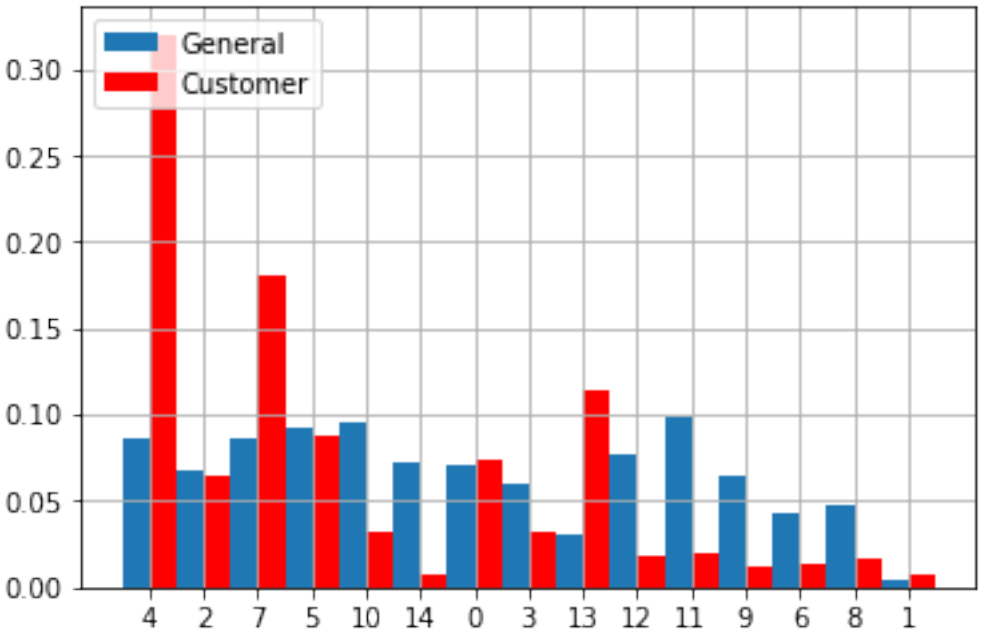
- Recall the analysis you performed in step 1.1.3 of the project, where you separated out certain data points from the dataset if they had more than a specified threshold of missing values. If you found that this group was qualitatively different from the main bulk of the data, you should treat this 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 [58]:
```
# 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 = plt.figure(figsize=(14,4))
ax1 = fig.add_subplot(121)
ax1.title.set_text('Customer Data')
sns.countplot(predections_customers, palette = 'Set3')

ax2 = fig.add_subplot(122)
ax2.title.set_text('General Demographic')
sns.countplot(predections_azdias_data, palette = 'Set2')
fig.tight_layout()
plt.show()
```



In [68]:
```
from collections import Counter

# generating proportion for the customer clusters.
labels, values = zip(*Counter(predections_customers).items())
v=list(values)
v[:] = [x/len(predections_customers) for x in v]
```

```
indexes = np.arange(len(labels))

# generating proportion for the azdias clusters.
labels1, values1 = zip(*Counter(predections_azdias_data).items())
v1=list(values1)
v1[:] = [x/len(predections_azdias_data) for x in v1]
indexes1 = np.arange(len(labels))

# configuring the graph
width = .5
plt.bar(indexes1, v1, width ,label='General')
plt.bar(indexes+width, v, width,  color='r' , label='Customer')
plt.xticks(indexes + width * 0.5, labels)
plt.legend(loc='upper left')
plt.grid()
plt.show()
```



In [62]:
```
# What kinds of people are part of a cluster that is overrepresented in the
# customer data compared to the general population?

cluster_14 = model.cluster_centers_[14]
pca_14 = pca.inverse_transform(cluster_14)
overrepresented = scaler.inverse_transform(pca_14)

overrepresented = pd.Series(data = overrepresented, index = azdias_general_columns)
display(overrepresented.head(45))
```

```
ALTERSKATEGORIE_GROB        1.822216
ANREDE_KZ                   0.967103
FINANZ_MINIMALIST           1.858698
FINANZ_SPARER               4.394156
FINANZ_VORSORGER            2.106516
FINANZ_ANLEGER              3.621081
FINANZ_UNAUFFAELLIGER       3.807727
FINANZ_HAUSBAUER            3.644642
GREEN_AVANTGARDE            0.123432
HEALTH_TYP                  2.365153
RETOURTYP_BK_S              2.775441
SEMIO_SOZ                   6.761393
SEMIO_FAM                   6.755196
SEMIO_REL                   6.325203
SEMIO_MAT                   5.115307
SEMIO_VERT                  6.048017
SEMIO_LUST                  2.902910
SEMIO_ERL                   2.168788
SEMIO_KULT                  6.881980
SEMIO_RAT                   4.118063
SEMIO_KRIT                  2.773405
SEMIO_DOM                   2.819522
SEMIO_KAEM                  2.439718
SEMIO_PFLICHT               5.684314
SEMIO_TRADV                 4.837207
SOHO_KZ                     0.008129
VERS_TYP                    1.516391
ANZ_PERSONEN                1.427544
ANZ_TITEL                   0.000126
HH_EINKOMMEN_SCORE          5.467693
W_KEIT_KIND_HH              4.554938
WOHNDAUER_2008              7.125523
ANZ_HAUSHALTE_AKTIV        13.425446
ANZ_HH_TITEL                0.042148
KONSUMNAEHE                 2.097448
MIN_GEBAEUDEJAHR         1992.629212
OST_WEST_KZ                 0.297656
KBA05_ANTG1                 0.373552
KBA05_ANTG2                 1.354055
KBA05_ANTG3                 1.227420
KBA05_ANTG4                 0.535710
KBA05_GBZ                   2.241230
BALLRAUM                    3.200063
EWDICHTE                    5.082708
INNENSTADT                  3.406594
dtype: float64
```

In [60]:
```
# What kinds of people are part of a cluster that is underrepresented in the
# customer data compared to the general population?

cluster_8 = model.cluster_centers_[8]
pca_8 = pca.inverse_transform(cluster_8)
underrepresented = scaler.inverse_transform(pca_8)

underrepresented = pd.Series(data = underrepresented, index = azdias_general_columns)
display(underrepresented.head(45))
```

```
ALTERSKATEGORIE_GROB        2.983413
ANREDE_KZ                   1.956229
FINANZ_MINIMALIST           2.022787
FINANZ_SPARER               3.287121
FINANZ_VORSORGER            3.062214
FINANZ_ANLEGER              3.427144
FINANZ_UNAUFFAELLIGER       2.983369
FINANZ_HAUSBAUER            3.710147
GREEN_AVANTGARDE            0.067832
HEALTH_TYP                  2.596432
RETOURTYP_BK_S              3.769945
SEMIO_SOZ                   2.926332
SEMIO_FAM                   2.727596
SEMIO_REL                   3.073697
SEMIO_MAT                   3.053143
SEMIO_VERT                  2.504274
SEMIO_LUST                  4.484378
SEMIO_ERL                   5.910096
SEMIO_KULT                  2.604278
SEMIO_RAT                   4.089178
SEMIO_KRIT                  5.916562
SEMIO_DOM                   6.134913
SEMIO_KAEM                  6.057826
SEMIO_PFLICHT               3.992110
SEMIO_TRADV                 3.099925
SOHO_KZ                     0.007346
VERS_TYP                    1.598096
ANZ_PERSONEN                1.580709
ANZ_TITEL                  -0.000494
HH_EINKOMMEN_SCORE          5.369289
W_KEIT_KIND_HH              3.984993
WOHNDAUER_2008              7.685368
ANZ_HAUSHALTE_AKTIV         7.511239
ANZ_HH_TITEL                0.032015
KONSUMNAEHE                 2.423156
MIN_GEBAEUDEJAHR         1992.490598
OST_WEST_KZ                 0.173927
KBA05_ANTG1                 0.717031
```

```
KBA05_ANTG2                1.918022
KBA05_ANTG3                1.273509
KBA05_ANTG4                0.180344
KBA05_GBZ                  2.685343
BALLRAUM                   3.680639
EWDICHTE                   4.682707
INNENSTADT                 3.868344
dtype: float64
```

1.3.5 Discussion 3.3: Compare Customer Data to Demographics Data

I conducted a comparison between popular and unpopular population clusters, focusing on attributes related to income, housing, age, gender, and movement patterns.

The popular cluster predominantly comprised individuals with high income and affluent households. They resided in areas with a significant number of 1-2 family houses within the microcell. These individuals were typically aged between 46 and 60, evenly distributed between genders, and exhibited low mobility patterns.

Conversely, the unpopular cluster primarily consisted of individuals with lower income and less privileged households. They tended to reside in areas with a lower concentration of 1-2 family houses within the microcell. The age group of this cluster was generally below 45, with a higher proportion of females, and showed a propensity for higher movement patterns.