Identify_Customer_Segments

March 22, 2022

1 Project: Identify Customer Segments

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

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

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

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

```
In [1]: # import libraries here; add more as necessary
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.preprocessing import Imputer, 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', sep=';')

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

In [3]:
```
# Output shape of files to see number of rows and columns
azdias.shape, feat_info.shape
```

Out[3]: ((891221, 85), (85, 4))

In [4]:
```
# Output the first few lines to get a feel for the data
azdias.head()
```

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

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

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

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

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

[5 rows x 85 columns]
```

In [5]:
```
# Output the first few lines to get a feel for the data
feat_info.head()
```

Out[5]:
```
              attribute information_level         type missing_or_unknown
0              AGER_TYP            person  categorical             [-1,0]
1  ALTERSKATEGORIE_GROB            person      ordinal           [-1,0,9]
2             ANREDE_KZ            person  categorical             [-1,0]
3         CJT_GESAMTTYP            person  categorical                [0]
4     FINANZ_MINIMALIST            person      ordinal               [-1]
```

In [6]: # Outputting dataframe info to see datatypes and you can also see many rows have missing
```
        azdias.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891221 entries, 0 to 891220
Data columns (total 85 columns):
AGER_TYP                 891221 non-null int64
ALTERSKATEGORIE_GROB     891221 non-null int64
ANREDE_KZ                891221 non-null int64
CJT_GESAMTTYP            886367 non-null float64
FINANZ_MINIMALIST        891221 non-null int64
FINANZ_SPARER            891221 non-null int64
FINANZ_VORSORGER         891221 non-null int64
FINANZ_ANLEGER           891221 non-null int64
FINANZ_UNAUFFAELLIGER    891221 non-null int64
FINANZ_HAUSBAUER         891221 non-null int64
FINANZTYP                891221 non-null int64
GEBURTSJAHR              891221 non-null int64
GFK_URLAUBERTYP          886367 non-null float64
GREEN_AVANTGARDE         891221 non-null int64
HEALTH_TYP               891221 non-null int64
LP_LEBENSPHASE_FEIN      886367 non-null float64
LP_LEBENSPHASE_GROB      886367 non-null float64
LP_FAMILIE_FEIN          886367 non-null float64
LP_FAMILIE_GROB          886367 non-null float64
LP_STATUS_FEIN           886367 non-null float64
LP_STATUS_GROB           886367 non-null float64
NATIONALITAET_KZ         891221 non-null int64
PRAEGENDE_JUGENDJAHRE    891221 non-null int64
RETOURTYP_BK_S           886367 non-null float64
SEMIO_SOZ                891221 non-null int64
SEMIO_FAM                891221 non-null int64
SEMIO_REL                891221 non-null int64
SEMIO_MAT                891221 non-null int64
SEMIO_VERT               891221 non-null int64
SEMIO_LUST               891221 non-null int64
SEMIO_ERL                891221 non-null int64
SEMIO_KULT               891221 non-null int64
SEMIO_RAT                891221 non-null int64
SEMIO_KRIT               891221 non-null int64
SEMIO_DOM                891221 non-null int64
SEMIO_KAEM               891221 non-null int64
SEMIO_PFLICHT            891221 non-null int64
SEMIO_TRADV              891221 non-null int64
SHOPPER_TYP              891221 non-null int64
SOHO_KZ                  817722 non-null float64
TITEL_KZ                 817722 non-null float64
VERS_TYP                 891221 non-null int64
```

```
ZABEOTYP                 891221 non-null int64
ALTER_HH                 817722 non-null float64
ANZ_PERSONEN             817722 non-null float64
ANZ_TITEL                817722 non-null float64
HH_EINKOMMEN_SCORE       872873 non-null float64
KK_KUNDENTYP             306609 non-null float64
W_KEIT_KIND_HH           783619 non-null float64
WOHNDAUER_2008           817722 non-null float64
ANZ_HAUSHALTE_AKTIV      798073 non-null float64
ANZ_HH_TITEL             794213 non-null float64
GEBAEUDETYP              798073 non-null float64
KONSUMNAEHE              817252 non-null float64
MIN_GEBAEUDEJAHR         798073 non-null float64
OST_WEST_KZ              798073 non-null object
WOHNLAGE                 798073 non-null float64
CAMEO_DEUG_2015          792242 non-null object
CAMEO_DEU_2015           792242 non-null object
CAMEO_INTL_2015          792242 non-null object
KBA05_ANTG1              757897 non-null float64
KBA05_ANTG2              757897 non-null float64
KBA05_ANTG3              757897 non-null float64
KBA05_ANTG4              757897 non-null float64
KBA05_BAUMAX             757897 non-null float64
KBA05_GBZ                757897 non-null float64
BALLRAUM                 797481 non-null float64
EWDICHTE                 797481 non-null float64
INNENSTADT               797481 non-null float64
GEBAEUDETYP_RASTER       798066 non-null float64
KKK                      770025 non-null float64
MOBI_REGIO               757897 non-null float64
ONLINE_AFFINITAET        886367 non-null float64
REGIOTYP                 770025 non-null float64
KBA13_ANZAHL_PKW         785421 non-null float64
PLZ8_ANTG1               774706 non-null float64
PLZ8_ANTG2               774706 non-null float64
PLZ8_ANTG3               774706 non-null float64
PLZ8_ANTG4               774706 non-null float64
PLZ8_BAUMAX              774706 non-null float64
PLZ8_HHZ                 774706 non-null float64
PLZ8_GBZ                 774706 non-null float64
ARBEIT                   794005 non-null float64
ORTSGR_KLS9              794005 non-null float64
RELAT_AB                 794005 non-null float64
dtypes: float64(49), int64(32), object(4)
memory usage: 578.0+ MB
```

```
In [7]: # Outputting dataframe info. No missing data in this one which is expected
```

```
feat_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85 entries, 0 to 84
Data columns (total 4 columns):
attribute             85 non-null object
information_level     85 non-null object
type                  85 non-null object
missing_or_unknown    85 non-null object
dtypes: object(4)
memory usage: 2.7+ KB
```

1.1 Step 1: Preprocessing

1.1.1 Step 1.1: Assess Missing Data

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

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

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

```
In [8]: # Identify missing or unknown data values and convert them to NaNs.
        print('The sum total of naturally missing values is {}'.format(azdias.isnull().sum().sum
```

```
The sum total of naturally missing values is 4896838
```

```
In [9]: # Convert missing values into NaNs
        for i in range(len(feat_info)):
            missing_or_unknown = feat_info.iloc[i]['missing_or_unknown']
            missing_or_unknown = missing_or_unknown.strip('[')
            missing_or_unknown = missing_or_unknown.strip(']')
            missing_or_unknown = missing_or_unknown.split(sep=',')
            missing_or_unknown = [int(value) if (value!='X' and value!='XX' and value!='') else
            if missing_or_unknown != ['']:
                azdias = azdias.replace({feat_info.iloc[i]['attribute']: missing_or_unknown}, np
```

```
In [10]: print('The sum total of missing values after conversion is {}'.format(azdias.isnull().s
```

```
The sum total of missing values after conversion 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!)

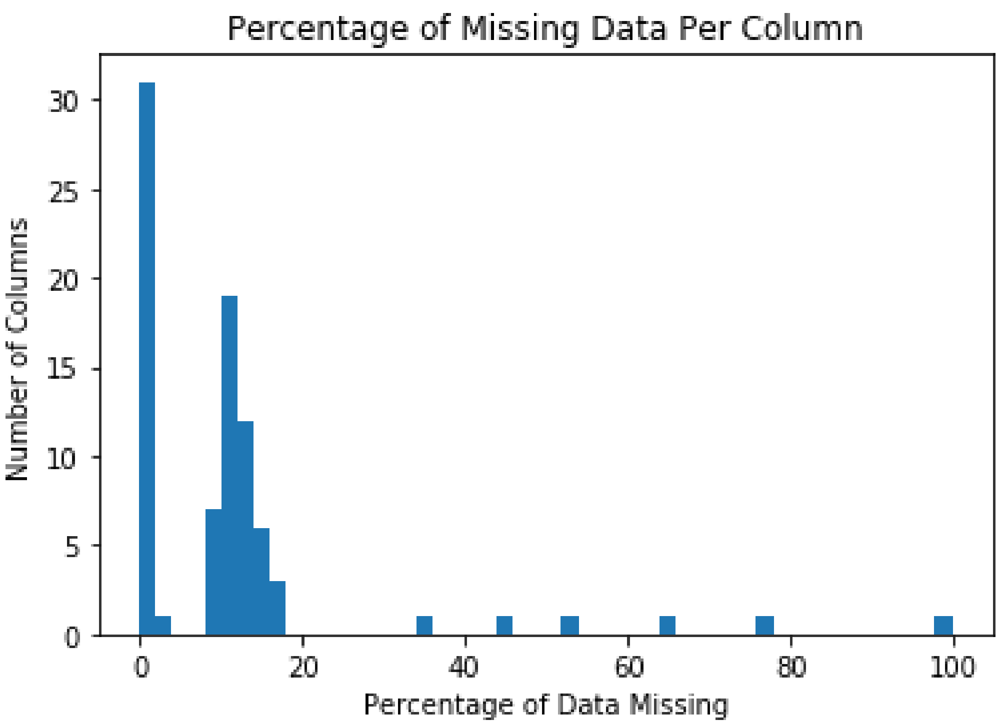
For the remaining features, are there any patterns in which columns have, or share, missing data?

```
In [11]: # Perform an assessment of how much missing data there is in each column of the dataset
         missing_data_per_column = (azdias.isnull().sum()/len(azdias))*100
```

```
In [12]: print('The sum total of naturally missing values after conversion is {}'.format(azdias.
```

```
The sum total of naturally missing values after conversion is 8373929
```

```
In [13]: # Investigate patterns in the amount of missing data in each column by plotting a histo
         plt.hist(missing_data_per_column, bins=50)
         plt.title("Percentage of Missing Data Per Column")
         plt.ylabel('Number of Columns')
         plt.xlabel('Percentage of Data Missing')
         plt.show()
```



```
In [14]: # Get list of columns with 20% or more data missing
         temp = missing_data_per_column[missing_data_per_column>20]
         drop_columns = temp.index.tolist()

In [15]: # Print list of columns with 20% or more data missing
         print('These columns have 20 percent, or more, missing data, and will be dropped:{}'.fo
```

```
These columns have 20 percent, or more, missing data, and will be dropped:['AGER_TYP', 'GEBURTSJ
```

```
In [16]: # Drop list of columns from original dataframe
         azdias = azdias.drop(drop_columns, axis=1)

In [17]: # Verify 6 columns were dropped
         azdias.shape

Out[17]: (891221, 79)

In [18]: # Verify 6 columns were dropped
         azdias.head()

Out[18]:    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 There were 6 columns in this dataset with 20% or more missing data. Their amount of missing data were 99%, 77%, 66%, 54%, 44%, and 35%. This wide array of values suggests there is no pattern discernable to the missing data. While I chose to remove columns that were missing 20% or more data, these columns are missing much larger chunks of data that 20% which makes them good targets to be dropped.

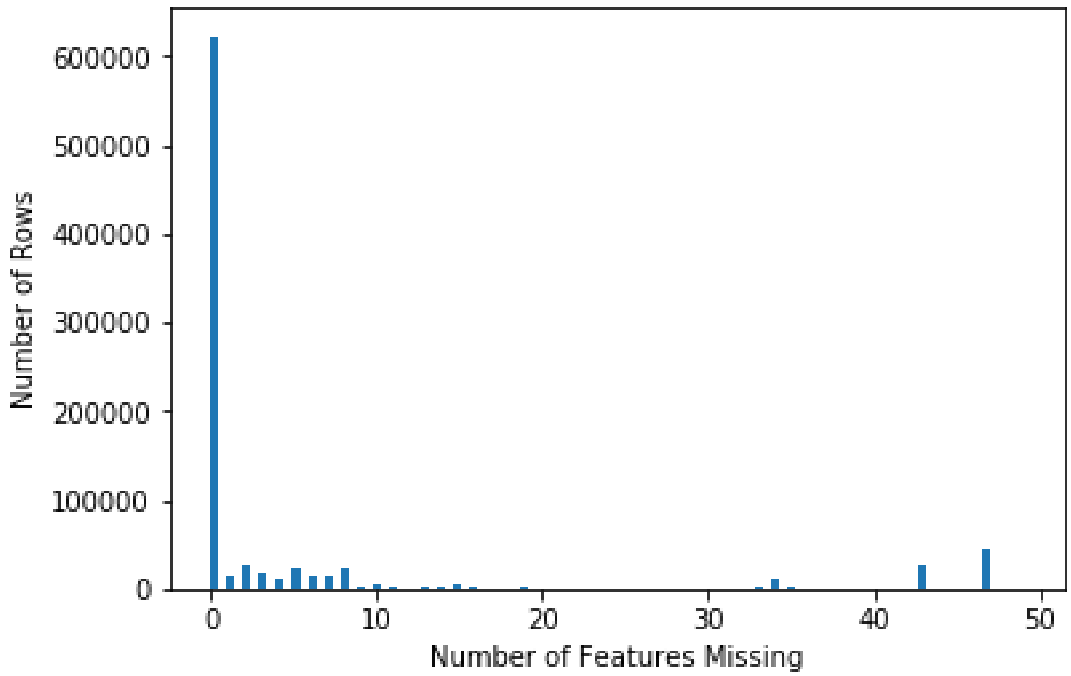
Step 1.1.3: Assess Missing Data in Each Row Now, you'll perform a similar assessment for the rows of the dataset. How much data is missing in each row? As with the columns, you should see some groups of points that have a very different numbers of missing values. Divide the data into two subsets: one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

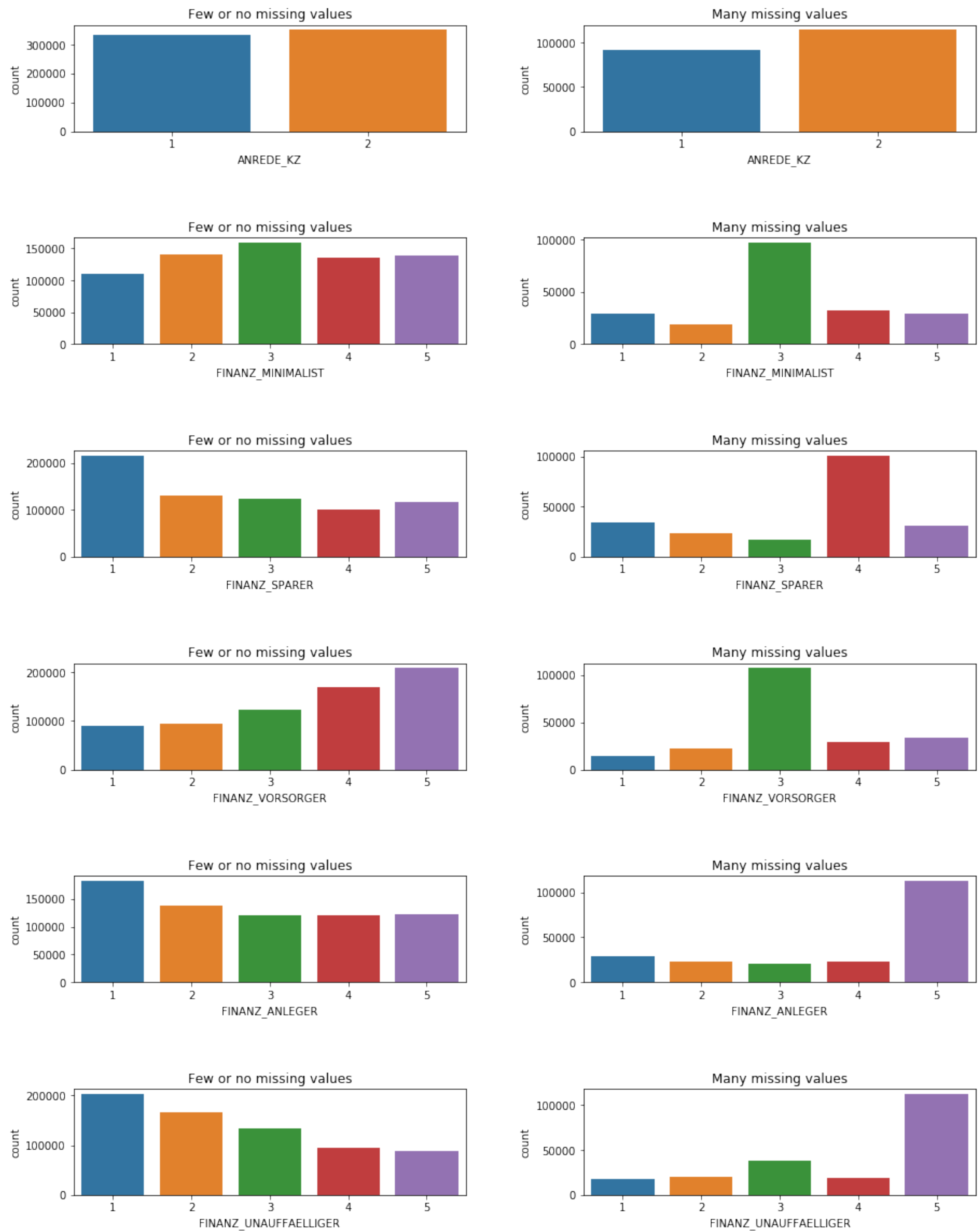
Depending on what you observe in your comparison, this will have implications on how you approach your conclusions later in the analysis. If the distributions of non-missing features look similar between the data with many missing values and the data with few or no missing values, then we could argue 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 [19]:
```
# How much data is missing in each row of the dataset?
missing_data_per_row = (azdias.isnull().sum(axis=1))
```

In [20]:
```
# Plot of above variable
plt.hist(missing_data_per_row, bins=100)
plt.ylabel('Number of Rows')
plt.xlabel('Number of Features Missing')
plt.show()
```




```
In [21]: # Separating data into 2 sets, those missing more than 10 features and those missing 10
         azdias_high = azdias[azdias.isnull().sum(axis=1) > 3]
         azdias_low = azdias[azdias.isnull().sum(axis=1) <= 3]

In [22]: # Compare the distribution of values for at least five columns where there are
         # no or few missing values, between the two subsets.
         full_columns = missing_data_per_column[missing_data_per_column==0].index.tolist()
         compare_columns = full_columns[:6]

In [23]: # Plot comparing distribution of data
         figure, axs = plt.subplots(nrows=len(compare_columns), ncols=2, figsize = (15,20))
         figure.subplots_adjust(hspace = 1, wspace=.3)
         for i in range(len(compare_columns)):
             sns.countplot(azdias_low[compare_columns[i]], ax=axs[i][0])
             axs[i][0].set_title('Few or no missing values')
             sns.countplot(azdias_high[compare_columns[i]], ax=axs[i][1])
             axs[i][1].set_title('Many missing values')
```



Discussion 1.1.3: Assess Missing Data in Each Row From the above plots we can see that the financial columns. which use a 5 value system ranging from very low to very high. there is a significant difference in the distribution of the data from those that are missing few values to those that are missing many. For the ANREDE_KZ the distribution is similar between the 2 sets. This is to be excepted as this column represents gender and only has 2 values of male or female.

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 [24]: # How many features are there of each data type?
         features = list(azdias_low.columns)
         feat_info_clean = feat_info[feat_info['attribute'].isin(features)]
         data_type_count = feat_info_clean['type'].value_counts()
         for i in range(len(data_type_count)):
             print('There are {} {} features.'.format(data_type_count[i], data_type_count.index[
```

```
There are 49 ordinal features.
There are 18 categorical features.
There are 6 numeric features.
There are 6 mixed features.
```

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 [25]: # Assess categorical variables: which are binary, which are multi-level, and
         # which one needs to be re-encoded?
         cat_feat = feat_info_clean[feat_info_clean["type"]=="categorical"]["attribute"]
```

```
In [26]: # Separating features in binary and multi-level categorical lists
         binary_feat = []
         multi_feat = []
```

```
        for cat in cat_feat:
            if (len(azdias_low[cat].unique())==2):
                binary_feat.append(cat)
            elif (len(azdias_low[cat].unique())>2):
                multi_feat.append(cat)
```

```
In [27]: # Print lists
         print(f"Binary features:\n{binary_feat}\n\nMulti level features:\n{multi_feat}")
```

```
Binary features:
['ANREDE_KZ', 'GREEN_AVANTGARDE', 'SOHO_KZ', 'VERS_TYP', 'OST_WEST_KZ']

Multi level features:
['CJT_GESAMTTYP', 'FINANZTYP', 'GFK_URLAUBERTYP', 'LP_FAMILIE_FEIN', 'LP_FAMILIE_GROB', 'LP_STAT
```

```
In [28]: # Re-encode categorical variable(s) to be kept in the analysis.
         for feat in binary_feat:
             print(f'{feat}: {azdias_low[feat].unique()}')
```

```
ANREDE_KZ: [2 1]
GREEN_AVANTGARDE: [0 1]
SOHO_KZ: [ 1.  0.]
VERS_TYP: [ 2.  1.]
OST_WEST_KZ: ['W' 'O']
```

```
In [29]: # Replace char W and O with numerical 0 and 1
         value_map = {'W':0, 'O':1}
         azdias_clean = azdias_low.replace({'OST_WEST_KZ':value_map})
```

```
In [30]: # Verify change
         print(f"OST_WEST_KZ: {azdias_clean['OST_WEST_KZ'].unique()}")
```

```
OST_WEST_KZ: [0 1]
```

```
In [31]: # Getting number of columns before dropping columns
         azdias_clean.shape[1]
```

```
Out[31]: 79
```

```
In [32]: azdias_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 684502 entries, 1 to 891220
Data columns (total 79 columns):
ALTERSKATEGORIE_GROB     684502 non-null float64
ANREDE_KZ                684502 non-null int64
```

```
CJT_GESAMTTYP           684502 non-null float64
FINANZ_MINIMALIST       684502 non-null int64
FINANZ_SPARER           684502 non-null int64
FINANZ_VORSORGER        684502 non-null int64
FINANZ_ANLEGER          684502 non-null int64
FINANZ_UNAUFFAELLIGER   684502 non-null int64
FINANZ_HAUSBAUER        684502 non-null int64
FINANZTYP               684502 non-null int64
GFK_URLAUBERTYP         684502 non-null float64
GREEN_AVANTGARDE        684502 non-null int64
HEALTH_TYP              684502 non-null float64
LP_LEBENSPHASE_FEIN     684500 non-null float64
LP_LEBENSPHASE_GROB     684501 non-null float64
LP_FAMILIE_FEIN         684502 non-null float64
LP_FAMILIE_GROB         684502 non-null float64
LP_STATUS_FEIN          684502 non-null float64
LP_STATUS_GROB          684502 non-null float64
NATIONALITAET_KZ        684502 non-null float64
PRAEGENDE_JUGENDJAHRE   677706 non-null float64
RETOURTYP_BK_S          684502 non-null float64
SEMIO_SOZ               684502 non-null int64
SEMIO_FAM               684502 non-null int64
SEMIO_REL               684502 non-null int64
SEMIO_MAT               684502 non-null int64
SEMIO_VERT              684502 non-null int64
SEMIO_LUST              684502 non-null int64
SEMIO_ERL               684502 non-null int64
SEMIO_KULT              684502 non-null int64
SEMIO_RAT               684502 non-null int64
SEMIO_KRIT              684502 non-null int64
SEMIO_DOM               684502 non-null int64
SEMIO_KAEM              684502 non-null int64
SEMIO_PFLICHT           684502 non-null int64
SEMIO_TRADV             684502 non-null int64
SHOPPER_TYP             684502 non-null float64
SOHO_KZ                 684502 non-null float64
VERS_TYP                684502 non-null float64
ZABEOTYP                684502 non-null int64
ANZ_PERSONEN            684502 non-null float64
ANZ_TITEL               684502 non-null float64
HH_EINKOMMEN_SCORE      684502 non-null float64
W_KEIT_KIND_HH          666425 non-null float64
WOHNDAUER_2008          684502 non-null float64
ANZ_HAUSHALTE_AKTIV     682041 non-null float64
ANZ_HH_TITEL            683945 non-null float64
GEBAEUDETYP             684502 non-null float64
KONSUMNAEHE             684466 non-null float64
MIN_GEBAEUDEJAHR        684502 non-null float64
```

```
OST_WEST_KZ             684502 non-null int64
WOHNLAGE                684502 non-null float64
CAMEO_DEUG_2015         681863 non-null object
CAMEO_DEU_2015          681863 non-null object
CAMEO_INTL_2015         681863 non-null object
KBA05_ANTG1             684502 non-null float64
KBA05_ANTG2             684502 non-null float64
KBA05_ANTG3             684502 non-null float64
KBA05_ANTG4             684502 non-null float64
KBA05_GBZ               684502 non-null float64
BALLRAUM                684099 non-null float64
EWDICHTE                684099 non-null float64
INNENSTADT              684099 non-null float64
GEBAEUDETYP_RASTER      684500 non-null float64
KKK                     646176 non-null float64
MOBI_REGIO              684502 non-null float64
ONLINE_AFFINITAET       684502 non-null float64
REGIOTYP                646176 non-null float64
KBA13_ANZAHL_PKW        683949 non-null float64
PLZ8_ANTG1              684502 non-null float64
PLZ8_ANTG2              684502 non-null float64
PLZ8_ANTG3              684502 non-null float64
PLZ8_ANTG4              684502 non-null float64
PLZ8_BAUMAX             684502 non-null float64
PLZ8_HHZ                684502 non-null float64
PLZ8_GBZ                684502 non-null float64
ARBEIT                  681074 non-null float64
ORTSGR_KLS9             681144 non-null float64
RELAT_AB                681074 non-null float64
dtypes: float64(51), int64(25), object(3)
memory usage: 417.8+ MB
```

```
In [33]: for feat in multi_feat:
             print(f'{feat}: {azdias_low[feat].unique()}')
```

```
CJT_GESAMTTYP: [ 5.  3.  2.  4.  1.  6.]
FINANZTYP: [1 5 2 4 6 3]
GFK_URLAUBERTYP: [ 10.   5.   1.  12.   9.   3.   8.  11.   4.   7.   6.   2.]
LP_FAMILIE_FEIN: [  5.   1.  10.   2.   7.  11.   8.   4.   6.   9.   3.]
LP_FAMILIE_GROB: [ 3.  1.  5.  2.  4.]
LP_STATUS_FEIN: [  2.   3.   4.   1.  10.   8.   9.   5.   6.   7.]
LP_STATUS_GROB: [ 1.  2.  5.  4.  3.]
NATIONALITAET_KZ: [ 1.  2.  3.]
SHOPPER_TYP: [ 3.  2.  0.  1.]
ZABEOTYP: [5 4 1 6 3 2]
GEBAEUDETYP: [ 8.  1.  3.  6.  2.  4.  5.]
CAMEO_DEUG_2015: ['8' '4' '6' '2' '1' '9' '5' '7' '3' nan]
```

```
CAMEO_DEU_2015: ['8A' '4C' '6B' '8C' '4A' '2D' '1A' '1E' '9D' '5D' '9E' '9B' '2A' '1B' '8B'
 '7A' '3D' '4E' '4B' '3C' '5A' '7B' '9A' '6D' '6E' '2C' '5C' '9C' '7D' '5E'
 '1D' '8D' '6C' '5B' '7C' '4D' '3A' '2B' '7E' '3B' '6F' nan '5F' '1C' '6A']
```

```
In [34]: # Drop multi-level features
         for feat in multi_feat:
             azdias_clean=azdias_clean.drop(feat, axis=1)
```

```
In [35]: # Getting count of columns after dropping columns
         azdias_clean.shape[1]
```

```
Out[35]: 66
```

```
In [36]: azdias_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 684502 entries, 1 to 891220
Data columns (total 66 columns):
ALTERSKATEGORIE_GROB     684502 non-null float64
ANREDE_KZ                684502 non-null int64
FINANZ_MINIMALIST        684502 non-null int64
FINANZ_SPARER            684502 non-null int64
FINANZ_VORSORGER         684502 non-null int64
FINANZ_ANLEGER           684502 non-null int64
FINANZ_UNAUFFAELLIGER    684502 non-null int64
FINANZ_HAUSBAUER         684502 non-null int64
GREEN_AVANTGARDE         684502 non-null int64
HEALTH_TYP               684502 non-null float64
LP_LEBENSPHASE_FEIN      684500 non-null float64
LP_LEBENSPHASE_GROB      684501 non-null float64
PRAEGENDE_JUGENDJAHRE    677706 non-null float64
RETOURTYP_BK_S           684502 non-null float64
SEMIO_SOZ                684502 non-null int64
SEMIO_FAM                684502 non-null int64
SEMIO_REL                684502 non-null int64
SEMIO_MAT                684502 non-null int64
SEMIO_VERT               684502 non-null int64
SEMIO_LUST               684502 non-null int64
SEMIO_ERL                684502 non-null int64
SEMIO_KULT               684502 non-null int64
SEMIO_RAT                684502 non-null int64
SEMIO_KRIT               684502 non-null int64
SEMIO_DOM                684502 non-null int64
SEMIO_KAEM               684502 non-null int64
SEMIO_PFLICHT            684502 non-null int64
SEMIO_TRADV              684502 non-null int64
SOHO_KZ                  684502 non-null float64
VERS_TYP                 684502 non-null float64
```

```
ANZ_PERSONEN             684502 non-null float64
ANZ_TITEL                684502 non-null float64
HH_EINKOMMEN_SCORE       684502 non-null float64
W_KEIT_KIND_HH           666425 non-null float64
WOHNDAUER_2008           684502 non-null float64
ANZ_HAUSHALTE_AKTIV      682041 non-null float64
ANZ_HH_TITEL             683945 non-null float64
KONSUMNAEHE              684466 non-null float64
MIN_GEBAEUDEJAHR         684502 non-null float64
OST_WEST_KZ              684502 non-null int64
WOHNLAGE                 684502 non-null float64
CAMEO_INTL_2015          681863 non-null object
KBA05_ANTG1              684502 non-null float64
KBA05_ANTG2              684502 non-null float64
KBA05_ANTG3              684502 non-null float64
KBA05_ANTG4              684502 non-null float64
KBA05_GBZ                684502 non-null float64
BALLRAUM                 684099 non-null float64
EWDICHTE                 684099 non-null float64
INNENSTADT               684099 non-null float64
GEBAEUDETYP_RASTER       684500 non-null float64
KKK                      646176 non-null float64
MOBI_REGIO               684502 non-null float64
ONLINE_AFFINITAET        684502 non-null float64
REGIOTYP                 646176 non-null float64
KBA13_ANZAHL_PKW         683949 non-null float64
PLZ8_ANTG1               684502 non-null float64
PLZ8_ANTG2               684502 non-null float64
PLZ8_ANTG3               684502 non-null float64
PLZ8_ANTG4               684502 non-null float64
PLZ8_BAUMAX              684502 non-null float64
PLZ8_HHZ                 684502 non-null float64
PLZ8_GBZ                 684502 non-null float64
ARBEIT                   681074 non-null float64
ORTSGR_KLS9              681144 non-null float64
RELAT_AB                 681074 non-null float64
dtypes: float64(42), int64(23), object(1)
memory usage: 349.9+ MB
```

Discussion 1.2.1: Re-Encode Categorical Features I re-encoded OST_WEST_KZ from 'W' and 'O' to 0 and 1 respectively. All other binary featureswere kept in tact. Finally I dropped all multi-level features.

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.

Checking data dictonary to understand feature breakdown

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

```
In [37]: # Investigate "PRAEGENDE_JUGENDJAHRE" and engineer two new variables.
         azdias_clean['PRAEGENDE_JUGENDJAHRE_DECADE'] = azdias_clean['PRAEGENDE_JUGENDJAHRE']
         azdias_clean['PRAEGENDE_JUGENDJAHRE_MOVEMENT'] = azdias_clean['PRAEGENDE_JUGENDJAHRE']
```

```
In [38]: decade_dict = {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:
         movement_dict = {1:1, 2:0, 3:1, 4:0, 5:1, 6:0, 7:0, 8:1, 9:0, 10:1, 11:0, 12:1, 13:0, 1
```

```
In [39]: azdias_clean['PRAEGENDE_JUGENDJAHRE_DECADE'].replace(decade_dict, inplace=True)
         azdias_clean['PRAEGENDE_JUGENDJAHRE_MOVEMENT'].replace(movement_dict, inplace=True)
```

Checking data dictonary to understand feature breakdown

German CAMEO: Wealth / Life Stage Typology, mapped to international code - -1: unknown - 11: Wealthy Households - Pre-Family Couples & Singles - 12: Wealthy Households - Young Couples With Children - 13: Wealthy Households - Families With School Age Children - 14: Wealthy Households - Older Families & Mature Couples - 15: Wealthy Households - Elders In Retirement - 21: Prosperous Households - Pre-Family Couples & Singles - 22: Prosperous Households - Young Couples With Children - 23: Prosperous Households - Families With School Age Children - 24: Prosperous Households - Older Families & Mature Couples - 25: Prosperous Households - Elders In Retirement - 31: Comfortable Households - Pre-Family Couples & Singles - 32: Comfortable Households - Young Couples With Children - 33: Comfortable Households - Families With School Age Children - 34: Comfortable Households - Older Families & Mature Couples - 35: Comfortable Households - Elders In Retirement - 41: Less Affluent Households - Pre-Family Couples & Singles - 42: Less Affluent Households - Young Couples With Children - 43: Less Affluent Households - Families With School Age Children - 44: Less Affluent Households - Older Families & Mature Couples - 45: Less Affluent Households - Elders In Retirement - 51: Poorer Households - Pre-Family Couples & Singles - 52: Poorer Households - Young Couples With Children - 53: Poorer

Households - Families With School Age Children - 54: Poorer Households - Older Families & Mature Couples - 55: Poorer Households - Elders In Retirement - XX: unknown

```
In [40]: # Investigate "CAMEO_INTL_2015" and engineer two new variables.
         azdias_clean['CAMEO_INTL_2015_WEALTH'] = azdias_clean['CAMEO_INTL_2015']
         azdias_clean['CAMEO_INTL_2015_LIFE_STAGE'] = azdias_clean['CAMEO_INTL_2015']
```

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

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

```
In [42]: azdias_clean['CAMEO_INTL_2015_WEALTH'].replace(wealth_dict, inplace=True)
         azdias_clean['CAMEO_INTL_2015_LIFE_STAGE'].replace(life_stage_dict, inplace=True)
```

```
In [43]: # Verifying changes
         azdias_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 684502 entries, 1 to 891220
Data columns (total 70 columns):
ALTERSKATEGORIE_GROB                684502 non-null float64
ANREDE_KZ                           684502 non-null int64
FINANZ_MINIMALIST                   684502 non-null int64
FINANZ_SPARER                       684502 non-null int64
FINANZ_VORSORGER                    684502 non-null int64
FINANZ_ANLEGER                      684502 non-null int64
FINANZ_UNAUFFAELLIGER               684502 non-null int64
FINANZ_HAUSBAUER                    684502 non-null int64
GREEN_AVANTGARDE                    684502 non-null int64
HEALTH_TYP                          684502 non-null float64
LP_LEBENSPHASE_FEIN                 684500 non-null float64
LP_LEBENSPHASE_GROB                 684501 non-null float64
PRAEGENDE_JUGENDJAHRE               677706 non-null float64
RETOURTYP_BK_S                      684502 non-null float64
SEMIO_SOZ                           684502 non-null int64
SEMIO_FAM                           684502 non-null int64
SEMIO_REL                           684502 non-null int64
SEMIO_MAT                           684502 non-null int64
SEMIO_VERT                          684502 non-null int64
SEMIO_LUST                          684502 non-null int64
SEMIO_ERL                           684502 non-null int64
SEMIO_KULT                          684502 non-null int64
SEMIO_RAT                           684502 non-null int64
SEMIO_KRIT                          684502 non-null int64
SEMIO_DOM                           684502 non-null int64
```

```
SEMIO_KAEM                      684502 non-null int64
SEMIO_PFLICHT                   684502 non-null int64
SEMIO_TRADV                     684502 non-null int64
SOHO_KZ                         684502 non-null float64
VERS_TYP                        684502 non-null float64
ANZ_PERSONEN                    684502 non-null float64
ANZ_TITEL                       684502 non-null float64
HH_EINKOMMEN_SCORE              684502 non-null float64
W_KEIT_KIND_HH                  666425 non-null float64
WOHNDAUER_2008                  684502 non-null float64
ANZ_HAUSHALTE_AKTIV             682041 non-null float64
ANZ_HH_TITEL                    683945 non-null float64
KONSUMNAEHE                     684466 non-null float64
MIN_GEBAEUDEJAHR                684502 non-null float64
OST_WEST_KZ                     684502 non-null int64
WOHNLAGE                        684502 non-null float64
CAMEO_INTL_2015                 681863 non-null object
KBA05_ANTG1                     684502 non-null float64
KBA05_ANTG2                     684502 non-null float64
KBA05_ANTG3                     684502 non-null float64
KBA05_ANTG4                     684502 non-null float64
KBA05_GBZ                       684502 non-null float64
BALLRAUM                        684099 non-null float64
EWDICHTE                        684099 non-null float64
INNENSTADT                      684099 non-null float64
GEBAEUDETYP_RASTER              684500 non-null float64
KKK                             646176 non-null float64
MOBI_REGIO                      684502 non-null float64
ONLINE_AFFINITAET               684502 non-null float64
REGIOTYP                        646176 non-null float64
KBA13_ANZAHL_PKW                683949 non-null float64
PLZ8_ANTG1                      684502 non-null float64
PLZ8_ANTG2                      684502 non-null float64
PLZ8_ANTG3                      684502 non-null float64
PLZ8_ANTG4                      684502 non-null float64
PLZ8_BAUMAX                     684502 non-null float64
PLZ8_HHZ                        684502 non-null float64
PLZ8_GBZ                        684502 non-null float64
ARBEIT                          681074 non-null float64
ORTSGR_KLS9                     681144 non-null float64
RELAT_AB                        681074 non-null float64
PRAEGENDE_JUGENDJAHRE_DECADE    677706 non-null float64
PRAEGENDE_JUGENDJAHRE_MOVEMENT  677706 non-null float64
CAMEO_INTL_2015_WEALTH          681863 non-null float64
CAMEO_INTL_2015_LIFE_STAGE      681863 non-null float64
dtypes: float64(46), int64(23), object(1)
memory usage: 370.8+ MB
```

Discussion 1.2.2: Engineer Mixed-Type Features For both PRAEGENDE_JUGENDJAHRE and CAMEO_INTL_2015 features I created 2 new features. PRAEGENDE_JUGENDJAHRE was broken out into PRAEGENDE_JUGENDJAHRE_MOVEMENT and PRAEGENDE_JUGENDJAHRE_DECADE. CAMEO_INTL_2015 was broken out into CAMEO_INTL_2015_WEALTh and CAMEO_INTL_2015_LIFE_STAGE. Original features were then dropped form the data set.

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 [44]: # Do whatever you need to in order to ensure that the dataframe only contains
         # the columns that should be passed to the algorithm functions.
         mixed_features = feat_info_clean[feat_info_clean["type"]=="mixed"]["attribute"]
         for feature in mixed_features:
             azdias_clean.drop(feature, axis=1, inplace=True)
```

```
In [45]: azdias_clean.head()
```

```
Out[45]:    ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
         1                   1.0          2                  1              5
         2                   3.0          2                  1              4
         4                   3.0          1                  4              3
         5                   1.0          2                  3              1
         6                   2.0          2                  1              5

            FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER  \
         1                 2               5                      4                 5
         2                 1               2                      3                 5
         4                 4               1                      3                 2
         5                 5               2                      2                 5
         6                 1               5                      4                 3

            GREEN_AVANTGARDE  HEALTH_TYP           ...             PLZ8_ANTG4  \
         1                 0         3.0           ...                    1.0
         2                 1         3.0           ...                    0.0
         4                 0         3.0           ...                    1.0
         5                 0         3.0           ...                    1.0
         6                 0         2.0           ...                    0.0
```

```
   PLZ8_HHZ  PLZ8_GBZ  ARBEIT  ORTSGR_KLS9  RELAT_AB  \
1       5.0       4.0     3.0          5.0       4.0
2       4.0       4.0     3.0          5.0       2.0
4       3.0       3.0     4.0          6.0       5.0
5       5.0       5.0     2.0          3.0       3.0
6       5.0       5.0     4.0          6.0       3.0

   PRAEGENDE_JUGENDJAHRE_DECADE  PRAEGENDE_JUGENDJAHRE_MOVEMENT  \
1                           6.0                             1.0
2                           6.0                             0.0
4                           4.0                             1.0
5                           2.0                             1.0
6                           5.0                             1.0

   CAMEO_INTL_2015_WEALTH  CAMEO_INTL_2015_LIFE_STAGE
1                     5.0                         1.0
2                     2.0                         4.0
4                     4.0                         3.0
5                     5.0                         4.0
6                     2.0                         2.0

[5 rows x 64 columns]
```

In [46]:
```
azdias_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 684502 entries, 1 to 891220
Data columns (total 64 columns):
ALTERSKATEGORIE_GROB                  684502 non-null float64
ANREDE_KZ                             684502 non-null int64
FINANZ_MINIMALIST                     684502 non-null int64
FINANZ_SPARER                         684502 non-null int64
FINANZ_VORSORGER                      684502 non-null int64
FINANZ_ANLEGER                        684502 non-null int64
FINANZ_UNAUFFAELLIGER                 684502 non-null int64
FINANZ_HAUSBAUER                      684502 non-null int64
GREEN_AVANTGARDE                      684502 non-null int64
HEALTH_TYP                            684502 non-null float64
RETOURTYP_BK_S                        684502 non-null float64
SEMIO_SOZ                             684502 non-null int64
SEMIO_FAM                             684502 non-null int64
SEMIO_REL                             684502 non-null int64
SEMIO_MAT                             684502 non-null int64
SEMIO_VERT                            684502 non-null int64
SEMIO_LUST                            684502 non-null int64
SEMIO_ERL                             684502 non-null int64
SEMIO_KULT                            684502 non-null int64
SEMIO_RAT                             684502 non-null int64
```

```
SEMIO_KRIT                        684502 non-null int64
SEMIO_DOM                         684502 non-null int64
SEMIO_KAEM                        684502 non-null int64
SEMIO_PFLICHT                     684502 non-null int64
SEMIO_TRADV                       684502 non-null int64
SOHO_KZ                           684502 non-null float64
VERS_TYP                          684502 non-null float64
ANZ_PERSONEN                      684502 non-null float64
ANZ_TITEL                         684502 non-null float64
HH_EINKOMMEN_SCORE                684502 non-null float64
W_KEIT_KIND_HH                    666425 non-null float64
WOHNDAUER_2008                    684502 non-null float64
ANZ_HAUSHALTE_AKTIV               682041 non-null float64
ANZ_HH_TITEL                      683945 non-null float64
KONSUMNAEHE                       684466 non-null float64
MIN_GEBAEUDEJAHR                  684502 non-null float64
OST_WEST_KZ                       684502 non-null int64
KBA05_ANTG1                       684502 non-null float64
KBA05_ANTG2                       684502 non-null float64
KBA05_ANTG3                       684502 non-null float64
KBA05_ANTG4                       684502 non-null float64
KBA05_GBZ                         684502 non-null float64
BALLRAUM                          684099 non-null float64
EWDICHTE                          684099 non-null float64
INNENSTADT                        684099 non-null float64
GEBAEUDETYP_RASTER                684500 non-null float64
KKK                               646176 non-null float64
MOBI_REGIO                        684502 non-null float64
ONLINE_AFFINITAET                 684502 non-null float64
REGIOTYP                          646176 non-null float64
KBA13_ANZAHL_PKW                  683949 non-null float64
PLZ8_ANTG1                        684502 non-null float64
PLZ8_ANTG2                        684502 non-null float64
PLZ8_ANTG3                        684502 non-null float64
PLZ8_ANTG4                        684502 non-null float64
PLZ8_HHZ                          684502 non-null float64
PLZ8_GBZ                          684502 non-null float64
ARBEIT                            681074 non-null float64
ORTSGR_KLS9                       681144 non-null float64
RELAT_AB                          681074 non-null float64
PRAEGENDE_JUGENDJAHRE_DECADE      677706 non-null float64
PRAEGENDE_JUGENDJAHRE_MOVEMENT    677706 non-null float64
CAMEO_INTL_2015_WEALTH            681863 non-null float64
CAMEO_INTL_2015_LIFE_STAGE        681863 non-null float64
dtypes: float64(41), int64(23)
memory usage: 339.5 MB
```

1.1.3 Step 1.3: Create a Cleaning Function

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

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

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

             # convert missing value codes into NaNs, ...
             values = ['-1','0','1','2','3','4','5','6','7','8','9']
             for i in range(len(feat_info)):
                 missing_or_unknown = feat_info.iloc[i]['missing_or_unknown']
                 missing_or_unknown = missing_or_unknown.strip('[')
                 missing_or_unknown = missing_or_unknown.strip(']')
                 missing_or_unknown = missing_or_unknown.split(sep=',')
                 missing_or_unknown = [int(value) if (value!='X' and value!='XX' and value!='')
                 if missing_or_unknown != ['']:
                     df_clean = df.replace({feat_info.iloc[i]['attribute']: missing_or_unknown},


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


             # remove selected columns and rows, ...
             # drop columns with more than 20% missing values
             columns_miss_20 = ['AGER_TYP', 'GEBURTSJAHR', 'TITEL_KZ', 'ALTER_HH', 'KK_KUNDENTYP
             df_clean = df_clean.drop(columns_miss_20, axis=1)


             # drop rows with more than 3 missing values
             df_clean = df_clean[df_clean.isnull().sum(axis=1) <= 3]


             # Re-encoding char, numeric and non-numeric binary features
             value_map = {'W':0, 'O':1}
             df_clean = df_clean.replace({'OST_WEST_KZ':value_map})
```

```
    # drop multi-leve features
    cat_features = feat_info_clean[feat_info_clean["type"]=="categorical"]["attribute"]
    multi_level_feature=[]
    for feature in cat_features:
        if (len(df_clean[feature].unique())>2):
            multi_level_feature.append(feature)
    for feature in multi_level_feature:
        df_clean=df_clean.drop(feature, axis=1)

    # engineer mixed features
    df_clean['PRAEGENDE_JUGENDJAHRE_DECADE'] = df_clean['PRAEGENDE_JUGENDJAHRE']
    df_clean['PRAEGENDE_JUGENDJAHRE_MOVEMENT'] = df_clean['PRAEGENDE_JUGENDJAHRE']

    decade_dict = {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,
    movement_dict = {1:1, 2:0, 3:1, 4:0, 5:1, 6:0, 7:0, 8:1, 9:0, 10:1, 11:0, 12:1, 13:

    df_clean['PRAEGENDE_JUGENDJAHRE_DECADE'].replace(decade_dict, inplace=True)
    df_clean['PRAEGENDE_JUGENDJAHRE_MOVEMENT'].replace(movement_dict, inplace=True)

    df_clean['CAMEO_INTL_2015_WEALTH'] = df_clean['CAMEO_INTL_2015']
    df_clean['CAMEO_INTL_2015_LIFE_STAGE'] = df_clean['CAMEO_INTL_2015']

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

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

    df_clean['CAMEO_INTL_2015_WEALTH'].replace(wealth_dict, inplace=True)
    df_clean['CAMEO_INTL_2015_LIFE_STAGE'].replace(life_stage_dict, inplace=True)

    mixed_features = feat_info_clean[feat_info_clean["type"]=="mixed"]["attribute"]
    for feature in mixed_features:
        df_clean.drop(feature, axis=1, inplace=True)


    # Return the cleaned dataframe.
    return df_clean
```

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 [77]:
```
# Fill NaNs with mode
fill_missing = Imputer(strategy='most_frequent')
azdias_clean_imputed = pd.DataFrame(imputer.fit_transform(azdias_clean))
print('NaN values in the dataset:', azdias_clean_imputed.isna().sum().sum())
```

```
NaN values in the dataset: 0
```

In [49]:
```
azdias_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 684502 entries, 1 to 891220
Data columns (total 64 columns):
ALTERSKATEGORIE_GROB              684502 non-null float64
ANREDE_KZ                         684502 non-null int64
FINANZ_MINIMALIST                 684502 non-null int64
FINANZ_SPARER                     684502 non-null int64
FINANZ_VORSORGER                  684502 non-null int64
FINANZ_ANLEGER                    684502 non-null int64
FINANZ_UNAUFFAELLIGER             684502 non-null int64
FINANZ_HAUSBAUER                  684502 non-null int64
GREEN_AVANTGARDE                  684502 non-null int64
HEALTH_TYP                        684502 non-null float64
RETOURTYP_BK_S                    684502 non-null float64
SEMIO_SOZ                         684502 non-null int64
SEMIO_FAM                         684502 non-null int64
SEMIO_REL                         684502 non-null int64
SEMIO_MAT                         684502 non-null int64
SEMIO_VERT                        684502 non-null int64
SEMIO_LUST                        684502 non-null int64
```

```
SEMIO_ERL                       684502 non-null int64
SEMIO_KULT                      684502 non-null int64
SEMIO_RAT                       684502 non-null int64
SEMIO_KRIT                      684502 non-null int64
SEMIO_DOM                       684502 non-null int64
SEMIO_KAEM                      684502 non-null int64
SEMIO_PFLICHT                   684502 non-null int64
SEMIO_TRADV                     684502 non-null int64
SOHO_KZ                         684502 non-null float64
VERS_TYP                        684502 non-null float64
ANZ_PERSONEN                    684502 non-null float64
ANZ_TITEL                       684502 non-null float64
HH_EINKOMMEN_SCORE              684502 non-null float64
W_KEIT_KIND_HH                  666425 non-null float64
WOHNDAUER_2008                  684502 non-null float64
ANZ_HAUSHALTE_AKTIV             682041 non-null float64
ANZ_HH_TITEL                    683945 non-null float64
KONSUMNAEHE                     684466 non-null float64
MIN_GEBAEUDEJAHR                684502 non-null float64
OST_WEST_KZ                     684502 non-null int64
KBA05_ANTG1                     684502 non-null float64
KBA05_ANTG2                     684502 non-null float64
KBA05_ANTG3                     684502 non-null float64
KBA05_ANTG4                     684502 non-null float64
KBA05_GBZ                       684502 non-null float64
BALLRAUM                        684099 non-null float64
EWDICHTE                        684099 non-null float64
INNENSTADT                      684099 non-null float64
GEBAEUDETYP_RASTER              684500 non-null float64
KKK                             646176 non-null float64
MOBI_REGIO                      684502 non-null float64
ONLINE_AFFINITAET               684502 non-null float64
REGIOTYP                        646176 non-null float64
KBA13_ANZAHL_PKW                683949 non-null float64
PLZ8_ANTG1                      684502 non-null float64
PLZ8_ANTG2                      684502 non-null float64
PLZ8_ANTG3                      684502 non-null float64
PLZ8_ANTG4                      684502 non-null float64
PLZ8_HHZ                        684502 non-null float64
PLZ8_GBZ                        684502 non-null float64
ARBEIT                          681074 non-null float64
ORTSGR_KLS9                     681144 non-null float64
RELAT_AB                        681074 non-null float64
PRAEGENDE_JUGENDJAHRE_DECADE    677706 non-null float64
PRAEGENDE_JUGENDJAHRE_MOVEMENT  677706 non-null float64
CAMEO_INTL_2015_WEALTH          681863 non-null float64
CAMEO_INTL_2015_LIFE_STAGE      681863 non-null float64
dtypes: float64(41), int64(23)
```

```
memory usage: 339.5 MB
```

In [50]:
```python
# Preparing for data reformat after scaling
azdias_clean_imputed.columns = azdias_clean.columns
azdias_clean_imputed.index = azdias_clean.index
```

In [51]:
```python
# Apply feature scaling to the general population demographics data.
scaler = StandardScaler()
azdias_clean_scaled = scaler.fit_transform(azdias_clean_imputed)
```

In [52]:
```python
# Converting array to dataframe
azdias_clean_scaled = pd.DataFrame(azdias_clean_scaled, columns=list(azdias_clean_imput
```

In [53]:
```python
azdias_clean_scaled.head()
```

Out[53]:
```
   ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
0             -1.747634   0.975423          -1.523655       1.588878
1              0.193497   0.975423          -1.523655       0.908468
2              0.193497  -1.025197           0.677626       0.228057
3             -1.747634   0.975423          -0.056134      -1.132765
4             -0.777068   0.975423          -1.523655       1.588878

   FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER  \
0         -1.050212        1.513292               1.048651          1.341142
1         -1.771419       -0.548762               0.320698          1.341142
2          0.392200       -1.236113               0.320698         -0.834925
3          1.113406       -0.548762              -0.407255          1.341142
4         -1.771419        1.513292               1.048651         -0.109569

   GREEN_AVANTGARDE  HEALTH_TYP            ...               PLZ8_ANTG4  \
0         -0.542999    1.038860            ...                 0.409122
1          1.841624    1.038860            ...                -0.963869
2         -0.542999    1.038860            ...                 0.409122
3         -0.542999    1.038860            ...                 0.409122
4         -0.542999   -0.285764            ...                -0.963869

   PLZ8_HHZ  PLZ8_GBZ    ARBEIT  ORTSGR_KLS9  RELAT_AB  \
0  1.432172  0.564740 -0.187976    -0.133875  0.678924
1  0.402503  0.564740 -0.187976    -0.133875 -0.799090
2 -0.627167 -0.334972  0.816965     0.301888  1.417930
3  1.432172  1.464451 -1.192917    -1.005401 -0.060083
4  1.432172  1.464451  0.816965     0.301888 -0.060083

   PRAEGENDE_JUGENDJAHRE_DECADE  PRAEGENDE_JUGENDJAHRE_MOVEMENT  \
0                      1.144730                        0.542999
1                      1.144730                       -1.841624
2                     -0.232759                        0.542999
3                     -1.610248                        0.542999
```

```
4                      0.455986                          0.542999

   CAMEO_INTL_2015_WEALTH  CAMEO_INTL_2015_LIFE_STAGE
0                1.169744                   -1.255608
1               -0.873113                    0.753418
2                0.488792                    0.083743
3                1.169744                    0.753418
4               -0.873113                   -0.585933

[5 rows x 64 columns]
```

In [54]:
```
azdias_clean_scaled.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 684502 entries, 0 to 684501
Data columns (total 64 columns):
ALTERSKATEGORIE_GROB                 684502 non-null float64
ANREDE_KZ                            684502 non-null float64
FINANZ_MINIMALIST                    684502 non-null float64
FINANZ_SPARER                        684502 non-null float64
FINANZ_VORSORGER                     684502 non-null float64
FINANZ_ANLEGER                       684502 non-null float64
FINANZ_UNAUFFAELLIGER                684502 non-null float64
FINANZ_HAUSBAUER                     684502 non-null float64
GREEN_AVANTGARDE                     684502 non-null float64
HEALTH_TYP                           684502 non-null float64
RETOURTYP_BK_S                       684502 non-null float64
SEMIO_SOZ                            684502 non-null float64
SEMIO_FAM                            684502 non-null float64
SEMIO_REL                            684502 non-null float64
SEMIO_MAT                            684502 non-null float64
SEMIO_VERT                           684502 non-null float64
SEMIO_LUST                           684502 non-null float64
SEMIO_ERL                            684502 non-null float64
SEMIO_KULT                           684502 non-null float64
SEMIO_RAT                            684502 non-null float64
SEMIO_KRIT                           684502 non-null float64
SEMIO_DOM                            684502 non-null float64
SEMIO_KAEM                           684502 non-null float64
SEMIO_PFLICHT                        684502 non-null float64
SEMIO_TRADV                          684502 non-null float64
SOHO_KZ                              684502 non-null float64
VERS_TYP                             684502 non-null float64
ANZ_PERSONEN                         684502 non-null float64
ANZ_TITEL                            684502 non-null float64
HH_EINKOMMEN_SCORE                   684502 non-null float64
W_KEIT_KIND_HH                       684502 non-null float64
WOHNDAUER_2008                       684502 non-null float64
```

```
ANZ_HAUSHALTE_AKTIV              684502 non-null float64
ANZ_HH_TITEL                     684502 non-null float64
KONSUMNAEHE                      684502 non-null float64
MIN_GEBAEUDEJAHR                 684502 non-null float64
OST_WEST_KZ                      684502 non-null float64
KBA05_ANTG1                      684502 non-null float64
KBA05_ANTG2                      684502 non-null float64
KBA05_ANTG3                      684502 non-null float64
KBA05_ANTG4                      684502 non-null float64
KBA05_GBZ                        684502 non-null float64
BALLRAUM                         684502 non-null float64
EWDICHTE                         684502 non-null float64
INNENSTADT                       684502 non-null float64
GEBAEUDETYP_RASTER               684502 non-null float64
KKK                              684502 non-null float64
MOBI_REGIO                       684502 non-null float64
ONLINE_AFFINITAET                684502 non-null float64
REGIOTYP                         684502 non-null float64
KBA13_ANZAHL_PKW                 684502 non-null float64
PLZ8_ANTG1                       684502 non-null float64
PLZ8_ANTG2                       684502 non-null float64
PLZ8_ANTG3                       684502 non-null float64
PLZ8_ANTG4                       684502 non-null float64
PLZ8_HHZ                         684502 non-null float64
PLZ8_GBZ                         684502 non-null float64
ARBEIT                           684502 non-null float64
ORTSGR_KLS9                      684502 non-null float64
RELAT_AB                         684502 non-null float64
PRAEGENDE_JUGENDJAHRE_DECADE     684502 non-null float64
PRAEGENDE_JUGENDJAHRE_MOVEMENT   684502 non-null float64
CAMEO_INTL_2015_WEALTH           684502 non-null float64
CAMEO_INTL_2015_LIFE_STAGE       684502 non-null float64
dtypes: float64(64)
memory usage: 334.2 MB
```

1.2.2 Discussion 2.1: Apply Feature Scaling

I imputed the dataset to transform all Nan's to the mode value. I then did a check to ensure there were no longer any NaNs in the dataset. I thne used sklearns standard scaler to scale all the values in the dataset. Lastly I use the head command to view the dataframe to ensure everything was successful

1.2.3 Step 2.2: Perform Dimensionality Reduction

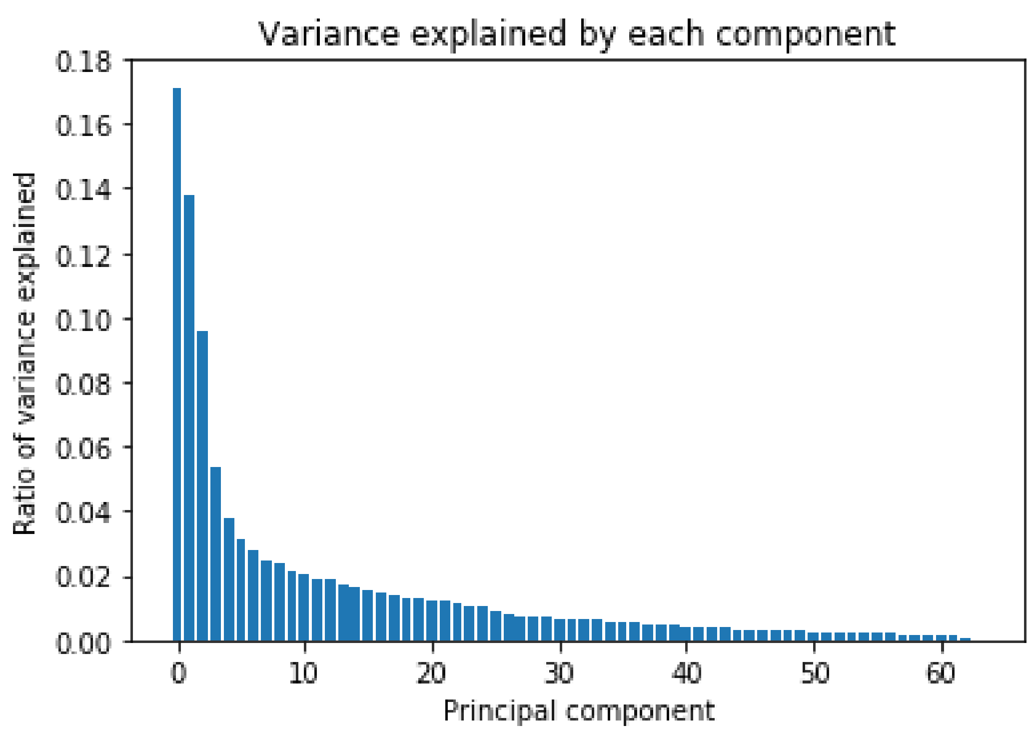
On your scaled data, you are now ready to apply dimensionality reduction techniques.

- Use sklearn's PCA class to apply principal component analysis on the data, thus finding the vectors of maximal variance in the data. To start, you should not set any parameters (so all components are computed) or set a number of components that is at least half the number of features (so there's enough features to see the general trend in variability).

- Check out the ratio of variance explained by each principal component as well as the cumulative variance explained. Try plotting the cumulative or sequential values using matplotlib's `plot()` function. Based on what you find, select a value for the number of transformed features you'll retain for the clustering part of the project.
- Once you've made a choice for the number of components to keep, make sure you re-fit a PCA instance to perform the decided-on transformation.

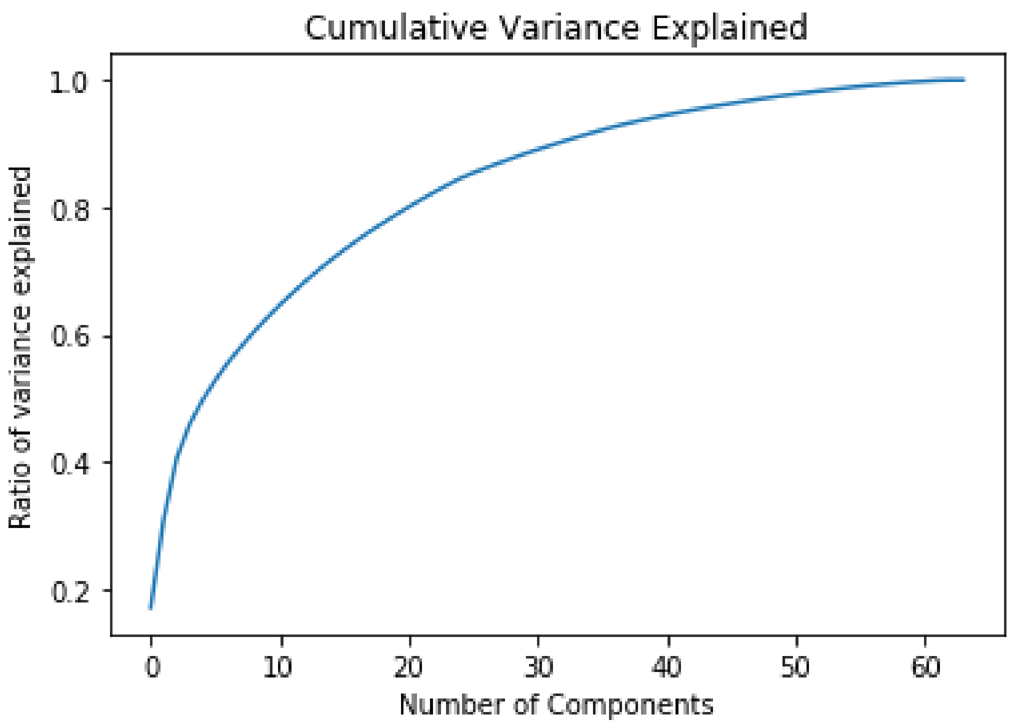
```
In [55]: # Apply PCA to the data.
         pca = PCA()
         pca.fit(azdias_clean_scaled)

Out[55]: PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
           svd_solver='auto', tol=0.0, whiten=False)
```

```
In [56]: # Investigate the variance accounted for by each principal component.
         plt.bar(range(len(pca.explained_variance_ratio_)), pca.explained_variance_ratio_)
         plt.title("Variance explained by each component")
         plt.xlabel("Principal component")
         plt.ylabel("Ratio of variance explained")
         plt.show()
```




```
In [57]: plt.plot(range(len(pca.explained_variance_ratio_)),np.cumsum(pca.explained_variance_rat
         plt.title("Cumulative Variance Explained")
```

```
plt.xlabel("Number of Components")
plt.ylabel("Ratio of variance explained")
plt.show()
```




```
In [58]: # Re-apply PCA to the data while selecting for number of components to retain.
         pca_30 = PCA(n_components=30)
         azdias_pca = pca_30.fit_transform(azdias_clean_scaled)
```

1.2.4 Discussion 2.2: Perform Dimensionality Reduction

The above plots show that 30 is a good number of components to keep. It is less than half of the total number of components while explaining over 80% of the variance.

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 [59]:
```
def pca_weights(pca, i):
    df = pd.DataFrame(pca.components_, columns=list(azdias_clean_scaled.columns))
    weights = df.iloc[i].sort_values(ascending=False)
    return weights
```

In [60]:
```
# Map weights for the first principal component to corresponding feature names
# and then print the linked values, sorted by weight.
pca_weight_0 = pca_weights(pca_30, 0)
print (pca_weight_0)
```

```
PLZ8_ANTG3                    0.220711
PLZ8_ANTG4                    0.214165
CAMEO_INTL_2015_WEALTH        0.201747
HH_EINKOMMEN_SCORE            0.199923
ORTSGR_KLS9                   0.190927
EWDICHTE                      0.188370
FINANZ_SPARER                 0.154826
FINANZ_HAUSBAUER              0.153379
KBA05_ANTG4                   0.150497
PLZ8_ANTG2                    0.148882
ARBEIT                        0.139579
KBA05_ANTG3                   0.135383
ANZ_HAUSHALTE_AKTIV           0.134509
RELAT_AB                      0.129916
SEMIO_PFLICHT                 0.122400
SEMIO_REL                     0.120072
PRAEGENDE_JUGENDJAHRE_DECADE  0.115818
PRAEGENDE_JUGENDJAHRE_MOVEMENT 0.111461
SEMIO_TRADV                   0.106040
SEMIO_RAT                     0.101987
SEMIO_MAT                     0.089735
SEMIO_FAM                     0.086258
FINANZ_UNAUFFAELLIGER         0.085244
SEMIO_KULT                    0.082415
FINANZ_ANLEGER                0.073740
REGIOTYP                      0.061374
OST_WEST_KZ                   0.056272
```

```
SEMIO_SOZ                       0.049426
PLZ8_HHZ                        0.040628
KKK                             0.040502
                                  ...
VERS_TYP                        0.021459
SEMIO_DOM                       0.017313
KBA05_ANTG2                     0.010936
ANREDE_KZ                      -0.001228
SEMIO_KRIT                     -0.001638
SOHO_KZ                        -0.001933
ANZ_TITEL                      -0.005200
SEMIO_VERT                     -0.028651
RETOURTYP_BK_S                 -0.029776
ONLINE_AFFINITAET              -0.033356
MIN_GEBAEUDEJAHR               -0.051081
WOHNDAUER_2008                 -0.067494
KBA13_ANZAHL_PKW               -0.076114
ANZ_PERSONEN                   -0.080256
SEMIO_LUST                     -0.094487
SEMIO_ERL                      -0.095340
GREEN_AVANTGARDE               -0.111461
GEBAEUDETYP_RASTER             -0.113546
BALLRAUM                       -0.122375
CAMEO_INTL_2015_LIFE_STAGE     -0.123520
FINANZ_VORSORGER               -0.124331
ALTERSKATEGORIE_GROB           -0.135227
INNENSTADT                     -0.159287
KONSUMNAEHE                    -0.162624
PLZ8_GBZ                       -0.163857
KBA05_GBZ                      -0.214052
PLZ8_ANTG1                     -0.221102
FINANZ_MINIMALIST              -0.221245
KBA05_ANTG1                    -0.222708
MOBI_REGIO                     -0.238902
Name: 0, Length: 64, dtype: float64
```

In [61]:
```
# Map weights for the second principal component to corresponding feature names
# and then print the linked values, sorted by weight.
pca_weight_1 = pca_weights(pca_30, 1)
print (pca_weight_1)
```

```
ALTERSKATEGORIE_GROB            0.252939
SEMIO_ERL                       0.228755
FINANZ_VORSORGER                0.221934
SEMIO_LUST                      0.177210
RETOURTYP_BK_S                  0.162699
FINANZ_HAUSBAUER                0.124830
```

SEMIO_KRIT	0.121044
W_KEIT_KIND_HH	0.115419
SEMIO_KAEM	0.113679
PLZ8_ANTG3	0.105631
PLZ8_ANTG4	0.101174
EWDICHTE	0.100779
ORTSGR_KLS9	0.099729
ANREDE_KZ	0.091042
CAMEO_INTL_2015_WEALTH	0.083804
KBA05_ANTG4	0.078008
ARBEIT	0.073660
PLZ8_ANTG2	0.071626
RELAT_AB	0.070655
SEMIO_DOM	0.068838
ANZ_HAUSHALTE_AKTIV	0.068454
HH_EINKOMMEN_SCORE	0.063433
WOHNDAUER_2008	0.054839
KBA05_ANTG3	0.053660
FINANZ_MINIMALIST	0.052308
ANZ_HH_TITEL	0.032066
VERS_TYP	0.031368
OST_WEST_KZ	0.030104
PRAEGENDE_JUGENDJAHRE_MOVEMENT	0.023189
REGIOTYP	0.018014
	...
CAMEO_INTL_2015_LIFE_STAGE	-0.006989
KBA05_ANTG2	-0.007224
GREEN_AVANTGARDE	-0.023189
KBA13_ANZAHL_PKW	-0.039042
MIN_GEBAEUDEJAHR	-0.042984
GEBAEUDETYP_RASTER	-0.048364
HEALTH_TYP	-0.054563
BALLRAUM	-0.066186
ANZ_PERSONEN	-0.066397
SEMIO_VERT	-0.072376
KONSUMNAEHE	-0.076633
PLZ8_GBZ	-0.080638
INNENSTADT	-0.082353
KBA05_ANTG1	-0.093779
KBA05_GBZ	-0.102256
SEMIO_SOZ	-0.102863
PLZ8_ANTG1	-0.103476
MOBI_REGIO	-0.104431
SEMIO_MAT	-0.153998
ONLINE_AFFINITAET	-0.164028
SEMIO_RAT	-0.166629
SEMIO_FAM	-0.177330
FINANZ_ANLEGER	-0.200528

```
SEMIO_KULT                      -0.217151
FINANZ_UNAUFFAELLIGER           -0.220778
SEMIO_TRADV                     -0.224854
FINANZ_SPARER                   -0.226785
SEMIO_PFLICHT                   -0.227848
PRAEGENDE_JUGENDJAHRE_DECADE    -0.234656
SEMIO_REL                       -0.258086
Name: 1, Length: 64, dtype: float64
```

In [62]:
```python
# Map weights for the third principal component to corresponding feature names
# and then print the linked values, sorted by weight.
pca_weight_2 = pca_weights(pca_30, 2)
print (pca_weight_2)
```

```
SEMIO_VERT                       0.347771
SEMIO_SOZ                        0.263247
SEMIO_FAM                        0.250498
SEMIO_KULT                       0.231953
FINANZ_MINIMALIST                0.157938
RETOURTYP_BK_S                   0.116842
FINANZ_VORSORGER                 0.100528
W_KEIT_KIND_HH                   0.090749
ALTERSKATEGORIE_GROB             0.084443
SEMIO_REL                        0.078693
SEMIO_LUST                       0.075136
SEMIO_MAT                        0.051723
GREEN_AVANTGARDE                 0.049770
EWDICHTE                         0.046941
ORTSGR_KLS9                      0.046833
PLZ8_ANTG4                       0.044875
PLZ8_ANTG3                       0.044330
WOHNDAUER_2008                   0.035593
ARBEIT                           0.031726
RELAT_AB                         0.030602
PLZ8_ANTG2                       0.029972
KBA05_ANTG4                      0.027934
ANZ_HAUSHALTE_AKTIV              0.025824
CAMEO_INTL_2015_WEALTH           0.024789
VERS_TYP                         0.021824
ANZ_HH_TITEL                     0.014213
OST_WEST_KZ                      0.011511
ANZ_TITEL                        0.010495
KBA05_ANTG3                      0.006882
PLZ8_HHZ                         0.006031
                                   ...
KBA05_ANTG2                     -0.010749
HEALTH_TYP                      -0.013749
```

```
MIN_GEBAEUDEJAHR                 -0.014119
KKK                              -0.017450
KBA13_ANZAHL_PKW                 -0.021148
KBA05_ANTG1                      -0.022671
HH_EINKOMMEN_SCORE               -0.023407
KBA05_GBZ                        -0.026829
MOBI_REGIO                       -0.029614
GEBAEUDETYP_RASTER               -0.030395
BALLRAUM                         -0.035567
PLZ8_GBZ                         -0.036451
KONSUMNAEHE                      -0.038344
INNENSTADT                       -0.043077
PLZ8_ANTG1                       -0.044711
FINANZ_HAUSBAUER                 -0.045769
PRAEGENDE_JUGENDJAHRE_MOVEMENT   -0.049770
ONLINE_AFFINITAET                -0.056798
SEMIO_PFLICHT                    -0.077715
SEMIO_TRADV                      -0.088932
FINANZ_UNAUFFAELLIGER            -0.094608
FINANZ_SPARER                    -0.103610
PRAEGENDE_JUGENDJAHRE_DECADE     -0.108221
SEMIO_ERL                        -0.169458
FINANZ_ANLEGER                   -0.189672
SEMIO_RAT                        -0.217304
SEMIO_KRIT                       -0.268380
SEMIO_DOM                        -0.312446
SEMIO_KAEM                       -0.336687
ANREDE_KZ                        -0.368244
Name: 2, Length: 64, dtype: float64
```

1.2.6 Discussion 2.3: Interpret Principal Components

By weighting the first principal compent we can read some correlations amongst the features. PLZ8_ANTG3(0.218150) and PLZ8_ANTG4(0.212194) have a positive correlation. These features correspond to the number of 6-10 family houses and 10+ family houses. The positive correlation shows that these features tend to increase similaryly.

PLZ8_ANTG3(0.218150) and HH_EINKOMMEN_SCORE(0.204670), the 2nd of which relates to household net income, shows that when the share of 6-10 family homes increases it tends to result in lower household net incomes.

PLZ8_ANTG3(0.218150) and MOBI_REGIO(0.238674), the 2nd of which relates to movement patterns, have a negative correlation. This suggest that when the share of 6-10 family homes increases it tends to have higher movements.

1.3 Step 3: Clustering

1.3.1 Step 3.1: Apply Clustering to General Population

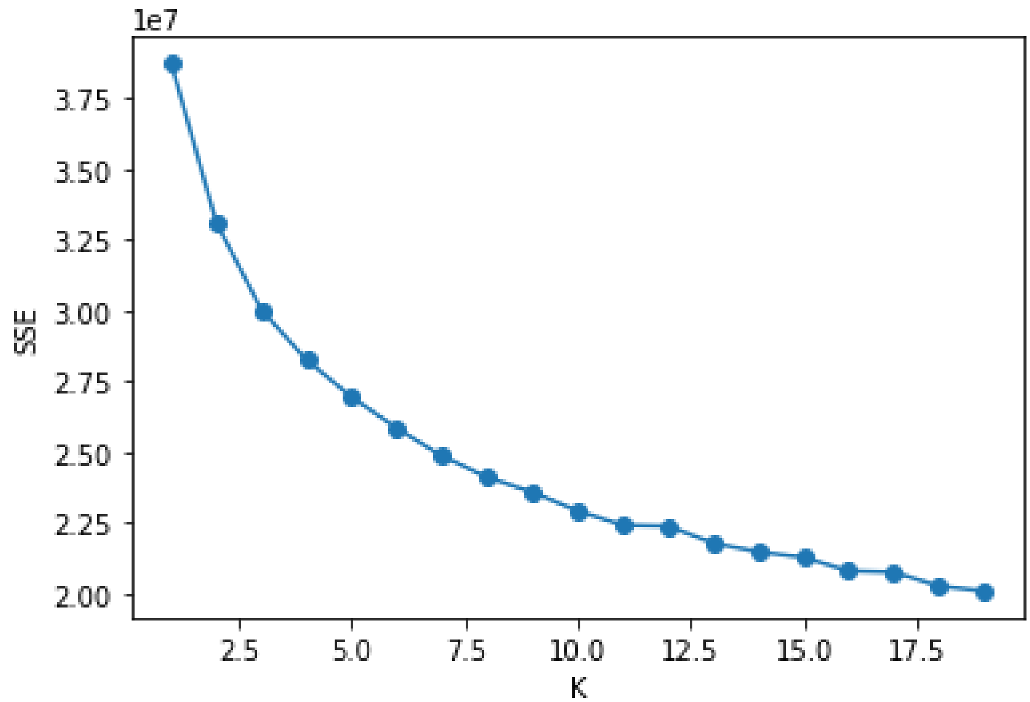
You've assessed and cleaned the demographics data, then scaled and transformed them. Now, it's time to see how the data clusters in the principal components space. In this substep, you will apply k-means clustering to the dataset and use the average within-cluster distances from each point to their assigned 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 [63]: def kmeans_score(data, n_cluster):
             kmeans = KMeans(n_clusters = n_cluster)
             model = kmeans.fit(data)
             score = np.abs(model.score(data))
             return score
```

```
In [64]: # Over a number of different cluster counts...
         # run k-means clustering on the data and...
         # compute the average within-cluster distances.
         scores = []
         ks = list(range(1,20))
         for i in ks:
             scores.append(kmeans_score(azdias_pca, i))
```

```
In [65]: # Investigate the change in within-cluster distance across number of clusters.
         plt.plot(ks, scores, linestyle='-', marker='o');
         plt.xlabel('K')
         plt.ylabel('SSE')

Out[65]: Text(0,0.5,'SSE')
```




```
In [66]: # 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 = 14)
         model_14 = kmeans.fit(azdias_pca)
         azdias_pred = model_14.predict(azdias_pca)
```

1.3.2 Discussion 3.1: Apply Clustering to General Population

Based on the above plot I can see a clear elbow at around 14, therefore I chose 14 clusters to sement the data.

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 [67]:
```
# Load in the customer demographics data.
customers = pd.read_csv('Udacity_CUSTOMERS_Subset.csv', sep=';')
```

In [68]:
```
# Apply preprocessing, feature transformation, and clustering from the general
# demographics onto the customer data, obtaining cluster predictions for the
# customer demographics data.
customers_clean = clean_data(customers)
```

In [72]:
```
azdias_clean.drop(columns = 'VERS_TYP', inplace = True)
```

In [80]:
```
# Replace NaN
customers_clean_imputed = pd.DataFrame(fill_missing.fit_transform(customers_clean))
customers_clean_imputed.columns = customers_clean.columns
customers_clean_imputed.index = customers_clean.index
```

In [82]:
```
# Use scaler to transform data
customers_clean_scaled = scaler.transform(customers_clean_imputed)
customers_clean_scaled = pd.DataFrame(customers_clean_scaled, columns=list(customers_cl
```

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

ValueError                                Traceback (most recent call last)

<ipython-input-82-781e0290dbd4> in <module>()
  1 # Use scaler to transform data
----> 2 customers_clean_scaled = scaler.transform(customers_clean_imputed)
  3 customers_clean_scaled = pd.DataFrame(customers_clean_scaled, columns=list(customers


/opt/conda/lib/python3.6/site-packages/sklearn/preprocessing/data.py in transform(self,
690         else:
691             if self.with_mean:
--> 692                 X -= self.mean_
693             if self.with_std:
694                 X /= self.scale_


ValueError: operands could not be broadcast together with shapes (131582,63) (64,) (1315
```

In [84]:
```
customers_clean_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 131582 entries, 0 to 191651
Data columns (total 63 columns):
ALTERSKATEGORIE_GROB            131582 non-null float64
ANREDE_KZ                       131582 non-null float64
FINANZ_MINIMALIST               131582 non-null float64
FINANZ_SPARER                   131582 non-null float64
FINANZ_VORSORGER                131582 non-null float64
FINANZ_ANLEGER                  131582 non-null float64
FINANZ_UNAUFFAELLIGER           131582 non-null float64
FINANZ_HAUSBAUER                131582 non-null float64
GREEN_AVANTGARDE                131582 non-null float64
HEALTH_TYP                      131582 non-null float64
RETOURTYP_BK_S                  131582 non-null float64
SEMIO_SOZ                       131582 non-null float64
SEMIO_FAM                       131582 non-null float64
SEMIO_REL                       131582 non-null float64
SEMIO_MAT                       131582 non-null float64
SEMIO_VERT                      131582 non-null float64
SEMIO_LUST                      131582 non-null float64
SEMIO_ERL                       131582 non-null float64
SEMIO_KULT                      131582 non-null float64
SEMIO_RAT                       131582 non-null float64
SEMIO_KRIT                      131582 non-null float64
SEMIO_DOM                       131582 non-null float64
SEMIO_KAEM                      131582 non-null float64
SEMIO_PFLICHT                   131582 non-null float64
SEMIO_TRADV                     131582 non-null float64
SOHO_KZ                         131582 non-null float64
ANZ_PERSONEN                    131582 non-null float64
ANZ_TITEL                       131582 non-null float64
HH_EINKOMMEN_SCORE              131582 non-null float64
W_KEIT_KIND_HH                  131582 non-null float64
WOHNDAUER_2008                  131582 non-null float64
ANZ_HAUSHALTE_AKTIV             131582 non-null float64
ANZ_HH_TITEL                    131582 non-null float64
KONSUMNAEHE                     131582 non-null float64
MIN_GEBAEUDEJAHR                131582 non-null float64
OST_WEST_KZ                     131582 non-null float64
KBA05_ANTG1                     131582 non-null float64
KBA05_ANTG2                     131582 non-null float64
KBA05_ANTG3                     131582 non-null float64
KBA05_ANTG4                     131582 non-null float64
KBA05_GBZ                       131582 non-null float64
BALLRAUM                        131582 non-null float64
EWDICHTE                        131582 non-null float64
INNENSTADT                      131582 non-null float64
GEBAEUDETYP_RASTER              131582 non-null float64
```

```
KKK                              131582 non-null float64
MOBI_REGIO                       131582 non-null float64
ONLINE_AFFINITAET                131582 non-null float64
REGIOTYP                         131582 non-null float64
KBA13_ANZAHL_PKW                 131582 non-null float64
PLZ8_ANTG1                       131582 non-null float64
PLZ8_ANTG2                       131582 non-null float64
PLZ8_ANTG3                       131582 non-null float64
PLZ8_ANTG4                       131582 non-null float64
PLZ8_HHZ                         131582 non-null float64
PLZ8_GBZ                         131582 non-null float64
ARBEIT                           131582 non-null float64
ORTSGR_KLS9                      131582 non-null float64
RELAT_AB                         131582 non-null float64
PRAEGENDE_JUGENDJAHRE_DECADE     131582 non-null float64
PRAEGENDE_JUGENDJAHRE_MOVEMENT   131582 non-null float64
CAMEO_INTL_2015_WEALTH           131582 non-null float64
CAMEO_INTL_2015_LIFE_STAGE       131582 non-null float64
dtypes: float64(63)
memory usage: 64.2 MB
```

In [ ]:

In [ ]:
```
# PCA transformation
customers_pca = pca_30.transform(customers_clean_scaled)
```

In [ ]:
```
# Predict using Kmeans model_14
customers_pred = model_14.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 [ ]:
```
# Compare the proportion of data in each cluster for the customer data to the
# proportion of data in each cluster for the general population.
figure, axs = plt.subplots(nrows=1, ncols=2, figsize = (10,5));
figure.subplots_adjust(hspace = 1, wspace=.3);

sns.countplot(customers_pred, ax=axs[0])
axs[0].set_title('Customer Clusters')
sns.countplot(azdias_pred, ax=axs[1])
axs[1].set_title('General Clusters')
```

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

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

1.3.5 Discussion 3.3: Compare Customer Data to Demographics Data

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

> Congratulations on making it this far in the project! Before you finish, make sure to check through the entire notebook from top to bottom to make sure that your analysis follows a logical flow and all of your findings are documented in **Discussion** cells. Once you've checked over all of your work, you should export the notebook as an HTML document to submit for evaluation. You can do this from the menu, navigating to **File -> Download as -> HTML (.html)**. You will submit both that document and this notebook for your project submission.