Identify_Customer_Segments

November 5, 2023

1 Project: Identify Customer Segments

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

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

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

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

```
In [1]: # import libraries here; add more as necessary
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import operator
        import re
        from sklearn.preprocessing import Imputer
        from sklearn.cluster import KMeans
        from sklearn.preprocessing import StandardScaler
        from sklearn.decomposition import PCA
```

```
from sklearn.preprocessing import LabelEncoder

# magic word for producing visualizations in notebook
%matplotlib inline

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

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

1.0.1 Step 0: Load the Data

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

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

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

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

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

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

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

In [3]:
```
# Check the structure of the data after it's loaded (e.g. print the number of
# rows and columns, print the first few rows).
num_rows, num_cols  = azdias.shape
print('# of columns: {}'.format(num_cols))
print('# of rows: {}'.format(num_rows))
azdias.head(5)
```

```
# of columns: 85
# of rows: 891221
```

Out[3]:

	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 [4]:
```
azdias.describe()
```

Out[4]:

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

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

	FINANZ_UNAUFFAELLIGER	FINANZ_HAUSBAUER	...	PLZ8_ANTG1
count	891221.000000	891221.000000	...	774706.000000
mean	2.874167	3.075121	...	2.253330
std	1.486731	1.353248	...	0.972008
min	1.000000	1.000000	...	0.000000
25%	2.000000	2.000000	...	1.000000
50%	3.000000	3.000000	...	2.000000
75%	4.000000	4.000000	...	3.000000
max	5.000000	5.000000	...	4.000000

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

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

```
max         5.000000       5.000000       9.000000       9.000000

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

[8 rows x 81 columns]
```

In [5]:
```
azdias.dtypes
```

Out[5]:
```
AGER_TYP                  int64
ALTERSKATEGORIE_GROB      int64
ANREDE_KZ                 int64
CJT_GESAMTTYP           float64
FINANZ_MINIMALIST         int64
FINANZ_SPARER             int64
FINANZ_VORSORGER          int64
FINANZ_ANLEGER            int64
FINANZ_UNAUFFAELLIGER     int64
FINANZ_HAUSBAUER          int64
FINANZTYP                 int64
GEBURTSJAHR               int64
GFK_URLAUBERTYP         float64
GREEN_AVANTGARDE          int64
HEALTH_TYP                int64
LP_LEBENSPHASE_FEIN     float64
LP_LEBENSPHASE_GROB     float64
LP_FAMILIE_FEIN         float64
LP_FAMILIE_GROB         float64
LP_STATUS_FEIN          float64
LP_STATUS_GROB          float64
NATIONALITAET_KZ          int64
PRAEGENDE_JUGENDJAHRE     int64
RETOURTYP_BK_S          float64
SEMIO_SOZ                 int64
SEMIO_FAM                 int64
SEMIO_REL                 int64
SEMIO_MAT                 int64
SEMIO_VERT                int64
SEMIO_LUST                int64
                          ...
OST_WEST_KZ              object
```

```
WOHNLAGE                float64
CAMEO_DEUG_2015          object
CAMEO_DEU_2015           object
CAMEO_INTL_2015          object
KBA05_ANTG1             float64
KBA05_ANTG2             float64
KBA05_ANTG3             float64
KBA05_ANTG4             float64
KBA05_BAUMAX            float64
KBA05_GBZ               float64
BALLRAUM                float64
EWDICHTE                float64
INNENSTADT              float64
GEBAEUDETYP_RASTER      float64
KKK                     float64
MOBI_REGIO              float64
ONLINE_AFFINITAET       float64
REGIOTYP                float64
KBA13_ANZAHL_PKW        float64
PLZ8_ANTG1              float64
PLZ8_ANTG2              float64
PLZ8_ANTG3              float64
PLZ8_ANTG4              float64
PLZ8_BAUMAX             float64
PLZ8_HHZ                float64
PLZ8_GBZ                float64
ARBEIT                  float64
ORTSGR_KLS9             float64
RELAT_AB                float64
Length: 85, dtype: object
```

```
In [6]: 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
```

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

1.1 Step 1: Preprocessing

1.1.1 Step 1.1: Assess Missing Data

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

Step 1.1.1: Convert Missing Value Codes to NaNs The fourth column of the feature attributes summary (loaded in above as `feat_info`) documents the codes from the data dictionary that indicate missing or unknown data. While the file encodes this as a list (e.g. `[-1,0]`), this will get

read in as a string object. You'll need to do a little bit of parsing to make use of it to identify and clean the data. Convert data that matches a 'missing' or 'unknown' value code into a numpy NaN value. You might want to see how much data takes on a 'missing' or 'unknown' code, and how much data is naturally missing, as a point of interest.

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

```
In [7]: print('Number of missing values is {}'.format(azdias.isnull().sum().sum()))
```

```
Number of missing values is 4896838
```

```
In [8]: def missing_splited(missing_or_unknown):
            create_new_list=list()
            for values in missing_or_unknown:

                create_new_list.append(values[1:-1].split(","))

            return create_new_list

        values_missing=missing_splited(feat_info["missing_or_unknown"])
```

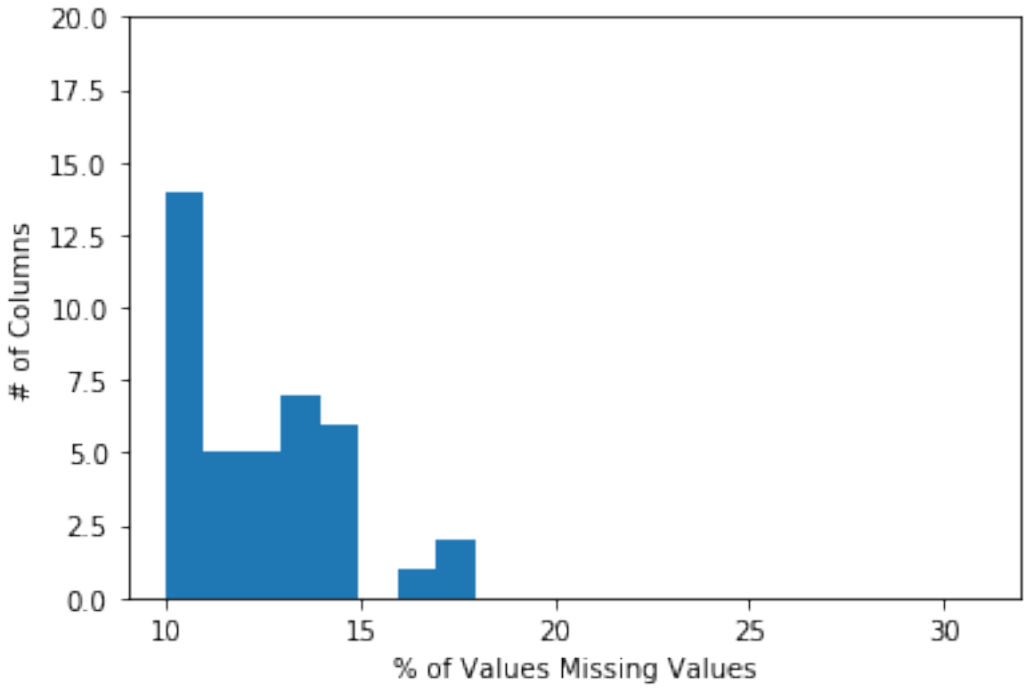
```
In [9]: # Identifying missing and unknown data values to convert them to NaNs.
        for attribute,values_missing_list in zip(feat_info["attribute"],values_missing):
            if values_missing_list[0] != "":
                for values_missing in values_missing_list:
                    #Checking to see if the missing value a positive or negative number
                    if values_missing.isnumeric() or values_missing.lstrip('-').isnumeric():
                        values_missing = int(values_missing)

                    azdias.loc[azdias[attribute] == values_missing, attribute] = np.nan
```

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

```
Total values missing after conversion is 8373929
```

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

```
In [12]: # Investigate patterns in the amount of missing data in each column.
         plt.hist(percent_values_missing_per_column, bins=100)
         plt.ylabel('# of Columns')
         plt.xlabel('% of Values Missing Values')
         x = np.arange(9,32,0.1)
         plt.xlim(9,32)
         y = np.sin(x)
         plt.ylim(0,20)
         plt.show()
```




```
In [13]: # Remove the outlier columns from the dataset. (You'll perform other data
         # engineering tasks such as re-encoding and imputation later.)
         percent_values_missing_per_column_20 = percent_values_missing_per_column[percent_values
```

```
In [14]: drop_columns = percent_values_missing_per_column_20.index.tolist()
         print('The columns with more than 20 percent missing data will be dropped:{}'.format(dr
```

```
The columns with more than 20 percent missing data will be dropped:['AGER_TYP', 'GEBURTSJAHR', '
```

```
In [15]: azdias = azdias.drop(drop_columns, axis=1)
```

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?

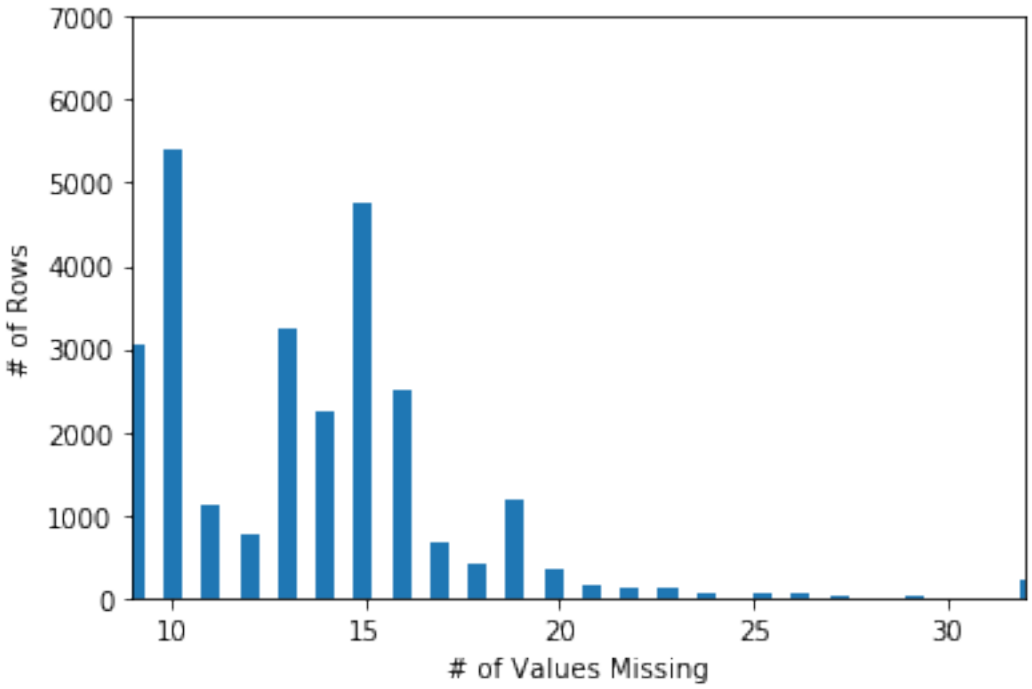
Discussion 1.1.2: Assess Missing Data in Each Column The majority of the columns are missing less than 20%. With that being the case the columns that are missing more will be considered outliers and will be dropped.

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.**

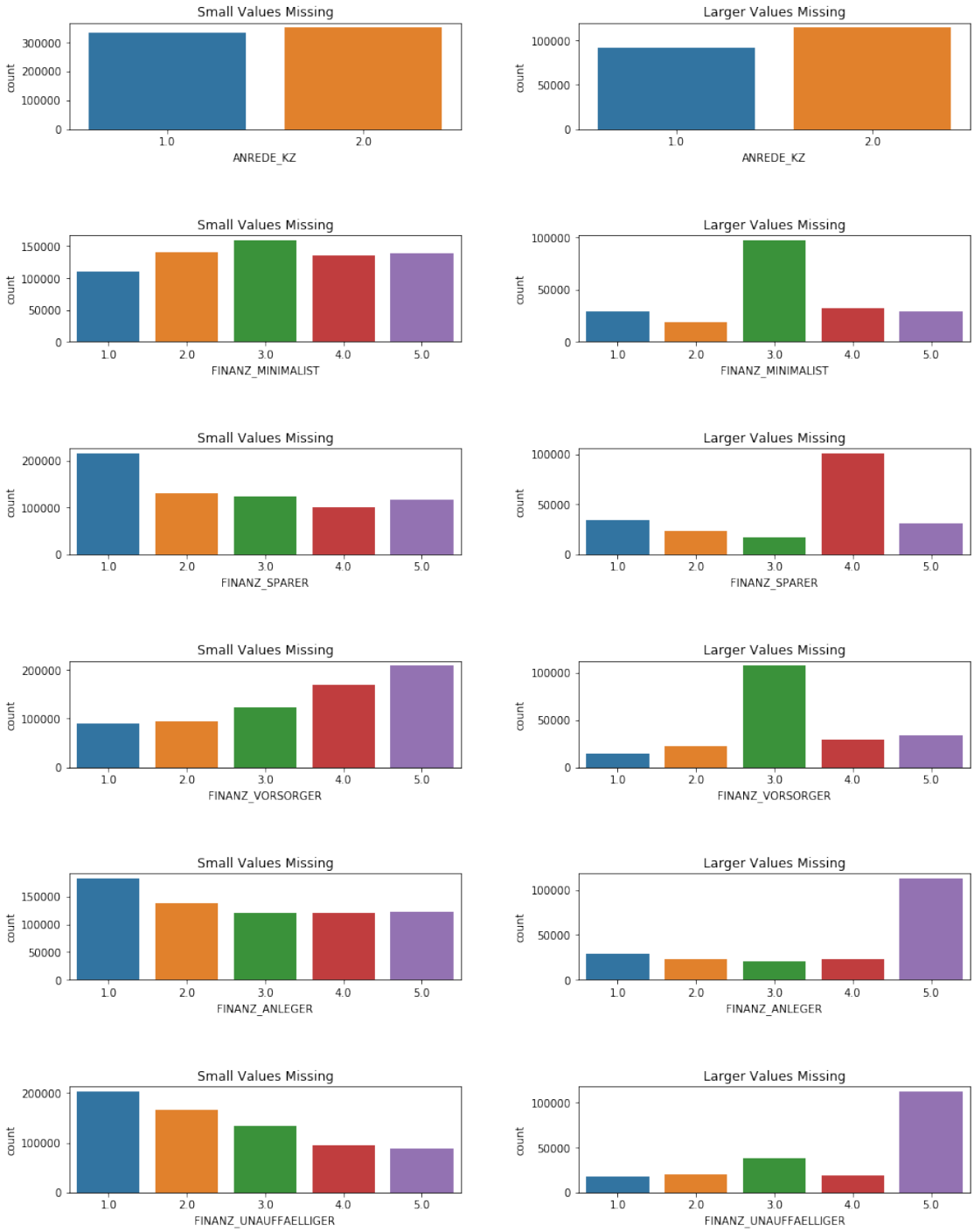
```
In [16]: # How much data is missing in each row of the dataset?
         values_missing_per_row =azdias.isnull().sum(axis=1)
         plt.hist(values_missing_per_row, bins=100)
         plt.ylabel('# of Rows')
         plt.xlabel('# of Values Missing')
         x = np.arange(9,32,0.1)
         plt.xlim(9,32)
         y = np.sin(x)
         plt.ylim(0,7000)
         plt.show()
```




```
In [17]: # Write code to divide the data into two subsets based on the number of missing
         # values in each row.
         azdias_lt_3 = azdias[azdias.isnull().sum(axis=1) <= 3]
         azdias_mt_3 = azdias[azdias.isnull().sum(axis=1) > 3]
```

```
In [18]: # Compare the distribution of values for at least five columns where there are
         # no or few missing values, between the two subsets.
         zero_values_missing_coloumns = percent_values_missing_per_column[percent_values_missing
         compare_columns = zero_values_missing_coloumns[:6]
```

```
In [19]: 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_lt_3[compare_columns[i]], ax=axs[i][0])
             axs[i][0].set_title('Small Values Missing')
             sns.countplot(azdias_mt_3[compare_columns[i]], ax=axs[i][1])
             axs[i][1].set_title('Larger Values Missing')
```



Discussion 1.1.3: Assess Missing Data in Each Row We can see from the charts that there seems to be a correlation between the Gender Feature, however looking at features such as Financial we seem to get a different view.

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 [20]: # How many features are there of each data type?
         features = list(azdias_lt_3.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 [21]: # Assess categorical variables: which are binary, which are multi-level, and
         # which one needs to be re-encoded?
         category_features = feat_info_clean[feat_info_clean["type"]=="categorical"]["attribute"
```

```
In [22]: binary_feature = []
         multi_feature=[]
         for feature in category_features:
```

```
            if (len(azdias_lt_3[feature].unique())==2):
                binary_feature.append(feature)
            elif (len(azdias_lt_3[feature].unique())>2):
                multi_feature.append(feature)
```

```
In [23]: print('Binary features: {}'.format(binary_feature))
         print('Multi features: {}'.format(multi_feature))
```

```
Binary features: ['ANREDE_KZ', 'GREEN_AVANTGARDE', 'SOHO_KZ', 'VERS_TYP', 'OST_WEST_KZ']
Multi features: ['CJT_GESAMTTYP', 'FINANZTYP', 'GFK_URLAUBERTYP', 'LP_FAMILIE_FEIN', 'LP_FAMILIE
```

```
In [24]: for feature in binary_feature:
             print('Unique values for {} are {}'.format(feature, azdias_lt_3[feature].unique()))
```

```
Unique values for ANREDE_KZ are [ 2.  1.]
Unique values for GREEN_AVANTGARDE are [0 1]
Unique values for SOHO_KZ are [ 1.  0.]
Unique values for VERS_TYP are [ 2.  1.]
Unique values for OST_WEST_KZ are ['W' 'O']
```

Discussion 1.2.1: Re-Encode Categorical Features In order to achieve the required results I started by. * Dropping the multi_level features. This allows me to trim down the dataset. * Re-encoding values in 'OST_West_KZ' to numbers. This will allow this feature to stay in the analysis. * Maintaining all binary 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.

```
In [25]: # Re-encode categorical variable(s) to be kept in the analysis.
         new_values = {'W': 0, 'O': 1}
         azdias_clean = azdias_lt_3.replace({'OST_WEST_KZ':new_values})
```

```
In [26]: # Drop multi-level features
         for feature in multi_feature:
             azdias_clean=azdias_clean.drop(feature, axis=1)
```

```
In [27]: # Investigate "PRAEGENDE_JUGENDJAHRE" and engineer two new variables.
         azdias_clean['DECADE'] = azdias_clean['PRAEGENDE_JUGENDJAHRE']
         azdias_clean['MOVEMENT'] = azdias_clean['PRAEGENDE_JUGENDJAHRE']

In [28]: 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 [29]: azdias_clean['DECADE'].replace(decade_dict, inplace=True)
         azdias_clean['MOVEMENT'].replace(movement_dict, inplace=True)

In [30]: # Investigate "CAMEO_INTL_2015" and engineer two new variables.
         azdias_clean['WEALTH'] = azdias_clean['CAMEO_INTL_2015']
         azdias_clean['LIFE_STAGE'] = azdias_clean['CAMEO_INTL_2015']

In [31]: 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 [32]: azdias_clean['WEALTH'].replace(wealth_dict, inplace=True)
         azdias_clean['LIFE_STAGE'].replace(life_stage_dict, inplace=True)
```

Discussion 1.2.2: Engineer Mixed-Type Features In order to keep PRAEGENDE_JUGENDJAHRE and CAMEO_INTL_2015 I decided create 2 new columns and copied the values to a new column. In the process of creating a new dictionary to map the original values in the feature column I to use the replace operation using dictionaries on the new column.

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

In [34]: # 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 [35]: azdias_clean.head()
```

Out[35]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	FINANZ_MINIMALIST	FINANZ_SPARER	\
1	1.0	2.0	1.0	5.0	
2	3.0	2.0	1.0	4.0	
4	3.0	1.0	4.0	3.0	
5	1.0	2.0	3.0	1.0	
6	2.0	2.0	1.0	5.0	

	FINANZ_VORSORGER	FINANZ_ANLEGER	FINANZ_UNAUFFAELLIGER	FINANZ_HAUSBAUER	\
1	2.0	5.0	4.0	5.0	
2	1.0	2.0	3.0	5.0	
4	4.0	1.0	3.0	2.0	
5	5.0	2.0	2.0	5.0	
6	1.0	5.0	4.0	3.0	

	GREEN_AVANTGARDE	HEALTH_TYP	...	PLZ8_ANTG4	PLZ8_HHZ	PLZ8_GBZ	\
1	0	3.0	...	1.0	5.0	4.0	
2	1	3.0	...	0.0	4.0	4.0	
4	0	3.0	...	1.0	3.0	3.0	
5	0	3.0	...	1.0	5.0	5.0	
6	0	2.0	...	0.0	5.0	5.0	

	ARBEIT	ORTSGR_KLS9	RELAT_AB	DECADE	MOVEMENT	WEALTH	LIFE_STAGE
1	3.0	5.0	4.0	6.0	1.0	5.0	1.0
2	3.0	5.0	2.0	6.0	0.0	2.0	4.0
4	4.0	6.0	5.0	4.0	1.0	4.0	3.0
5	2.0	3.0	3.0	2.0	1.0	5.0	4.0
6	4.0	6.0	3.0	5.0	1.0	2.0	2.0

[5 rows x 64 columns]

1.1.3 Step 1.3: Create a Cleaning Function

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

```
In [36]: def azidas_clean(azidas):

             """
             Perform feature trimming, re-encoding, and engineering for demographics
             data

             INPUT: Demographics DataFrame
```

```
    OUTPUT: Trimmed and cleaned demographics DataFrame
    """

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



    # remove selected columns and rows, ...
    azidas = azidas.drop(drop_columns, axis=1)
    azidas = azidas[azidas.isnull().sum(axis=1) < 40]




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

    # 2 drop the multi categorical features
    azidas = azidas.drop(multi_feature, axis = 1)

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

    azidas['WEALTH'] = azidas['CAMEO_INTL_2015']
    azidas['WEALTH'].replace('WEALTH', inplace = True)
    azidas['LIFE_STAGE'] = azidas['CAMEO_INTL_2015']
    azidas['LIFE_STAGE'].replace('LIFE_STAGE', inplace = True)

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




    # Return the cleaned dataframe.
    return azidas
```

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 [37]: # If you've not yet cleaned the dataset of all NaN values, then investigate and
         # do that now.
         fill_values_missing = Imputer(strategy='most_frequent')
         azdias_clean_imputed = pd.DataFrame(fill_values_missing.fit_transform(azdias_clean))
```

```
In [38]: azdias_clean_imputed.columns = azdias_clean.columns
         azdias_clean_imputed.index = azdias_clean.index
```

1.2.2 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 [39]:
```
# Apply feature scaling to the general population demographics data.
scaler = StandardScaler()
azdias_clean_scaled = scaler.fit_transform(azdias_clean_imputed)
```

In [40]:
```
azdias_clean_scaled = pd.DataFrame(azdias_clean_scaled, columns=list(azdias_clean_imput
```

In [41]:
```
azdias_clean_scaled.head()
```

Out[41]:

	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	PLZ8_HHZ	PLZ8_GBZ	\
0	-0.542999	1.038860	...	0.409122	1.432172	0.564740	
1	1.841624	1.038860	...	-0.963869	0.402503	0.564740	
2	-0.542999	1.038860	...	0.409122	-0.627167	-0.334972	
3	-0.542999	1.038860	...	0.409122	1.432172	1.464451	
4	-0.542999	-0.285764	...	-0.963869	1.432172	1.464451	

	ARBEIT	ORTSGR_KLS9	RELAT_AB	DECADE	MOVEMENT	WEALTH	LIFE_STAGE
0	-0.187976	-0.133875	0.678924	1.144730	0.542999	1.169744	-1.255608
1	-0.187976	-0.133875	-0.799090	1.144730	-1.841624	-0.873113	0.753418
2	0.816965	0.301888	1.417930	-0.232759	0.542999	0.488792	0.083743
3	-1.192917	-1.005401	-0.060083	-1.610248	0.542999	1.169744	0.753418
4	0.816965	0.301888	-0.060083	0.455986	0.542999	-0.873113	-0.585933

[5 rows x 64 columns]

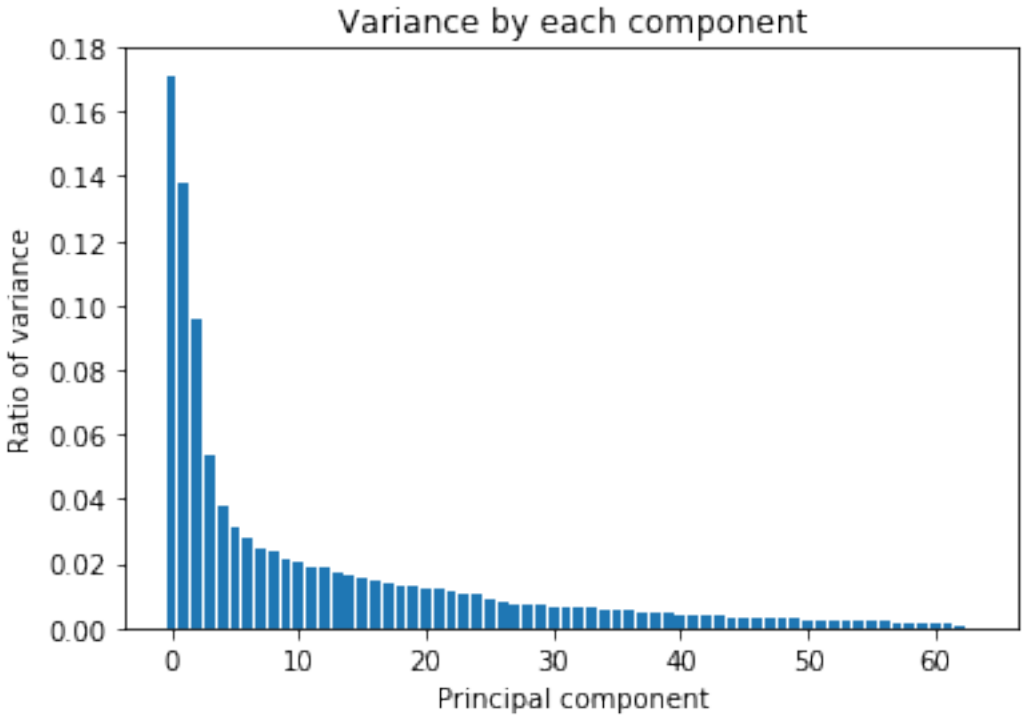
1.2.3 Discussion 2.1: Apply Feature Scaling

By replacing the missing values with the frequent values in the targeted column using the imputer method. All of the features have been scaled using the StandardScaler.

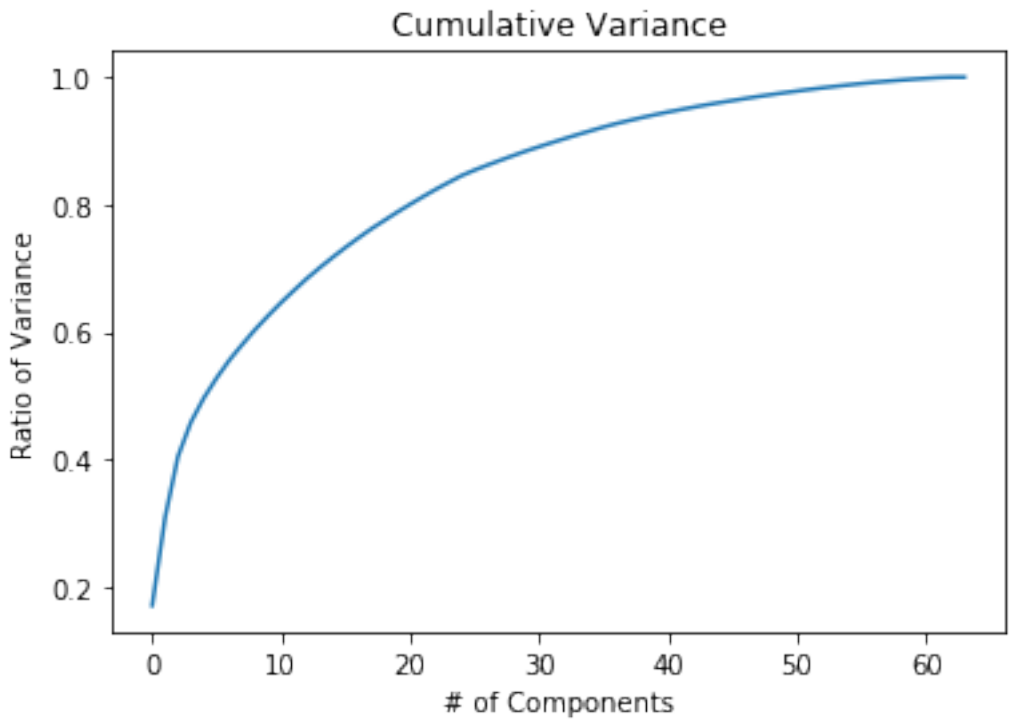
In [42]:
```
# Apply PCA to the data.
pca = PCA()
pca.fit(azdias_clean_scaled)
```

```
Out[42]: PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
           svd_solver='auto', tol=0.0, whiten=False)
```

```
In [43]: # Investigate the variance accounted for by each principal component.
         plt.bar(range(len(pca.explained_variance_ratio_)), pca.explained_variance_ratio_)
         plt.title("Variance by each component")
         plt.xlabel("Principal component")
         plt.ylabel("Ratio of variance")
         plt.show()
```




```
In [44]: plt.plot(range(len(pca.explained_variance_ratio_)),np.cumsum(pca.explained_variance_rat
         plt.title("Cumulative Variance")
         plt.xlabel("# of Components")
         plt.ylabel("Ratio of Variance")
         plt.show()
```




```
In [45]: # Re-apply PCA to the data while selecting for number of components to retain.
         pca_40 = PCA(n_components=30)
         data_PCA = pca_40.fit_transform(azdias_clean_scaled)
```

1.2.4 Discussion 2.2: Perform Dimensionality Reduction

By keeping 30 principal components, we can conlcude that more than 88% of the variances can be explained while reducing the number of by 50% or more.

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.

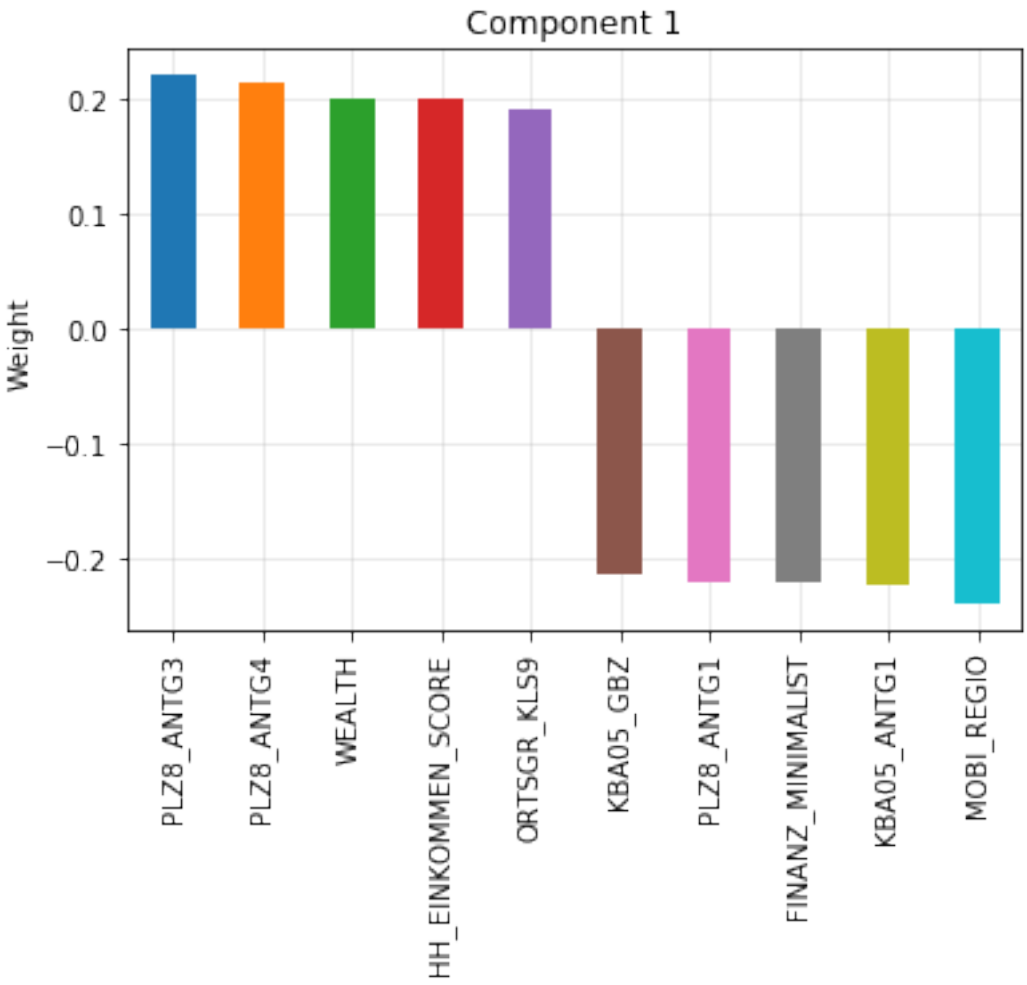
- To investigate the features, you should map each weight to their corresponding feature name, then sort the features according to weight. The most interesting features for each principal component, then, will be those at the beginning and end of the sorted list. Use the

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.

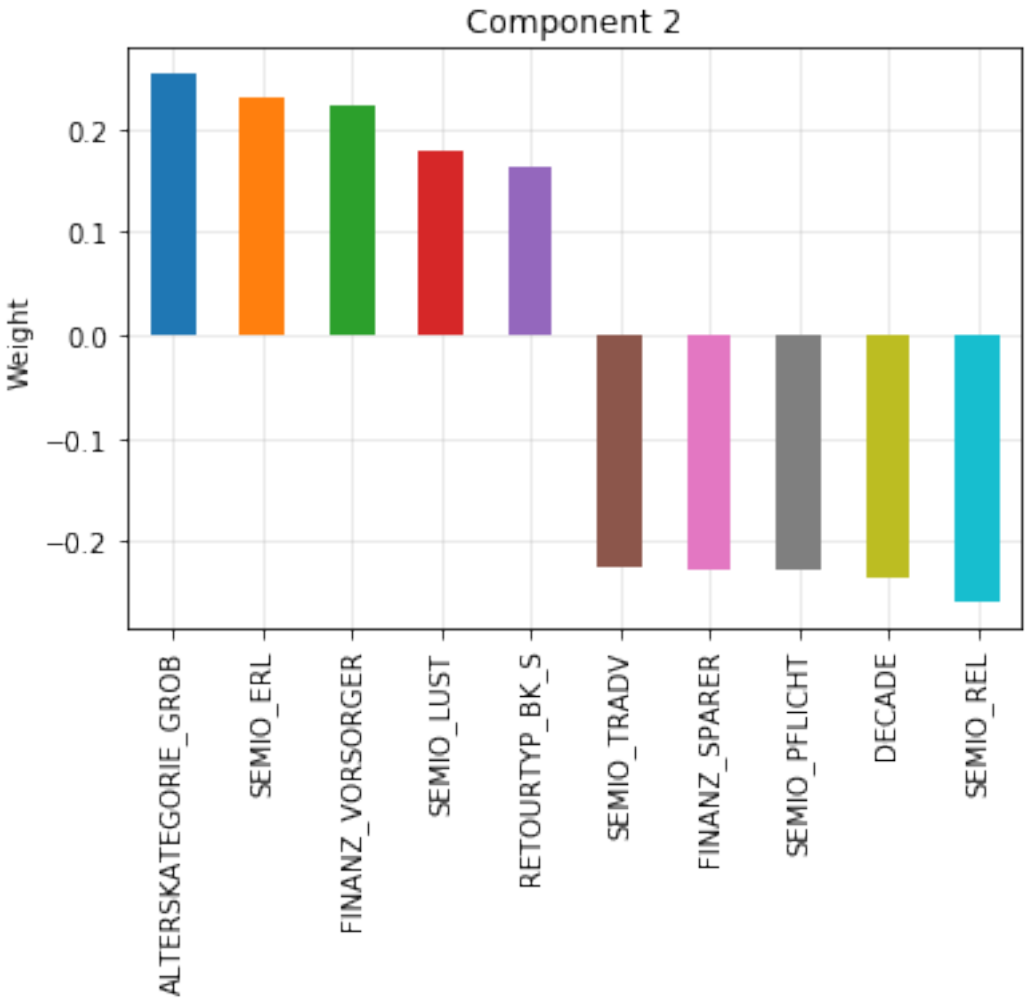
In [46]:
```
# Map weights for the first principal component to corresponding feature names
# and then print the linked values, sorted by weight.
# HINT: Try defining a function here or in a new cell that you can reuse in the
# other cells.
def pca_weights(pca, i):
    df = pd.DataFrame(pca.components_, columns=list(azdias_clean_scaled.columns))
    weights = df.iloc[i].sort_values(ascending=False)
    return weights
```

In [47]:
```
def plot_weights(df, pca, component, feature_plot):
    weights = pca_weights(pca, component)
    weights = pd.concat([weights.head(feature_plot), weights.tail(feature_plot)])

    weights.plot(kind='bar')
    plt.title('Component {}'.format(component+1))
    plt.ylabel('Weight')
    ax = plt.gca()
    ax.grid(linewidth='0.5', alpha=0.5)
    ax.set_axisbelow(True)
    plt.show()

    return weights
```

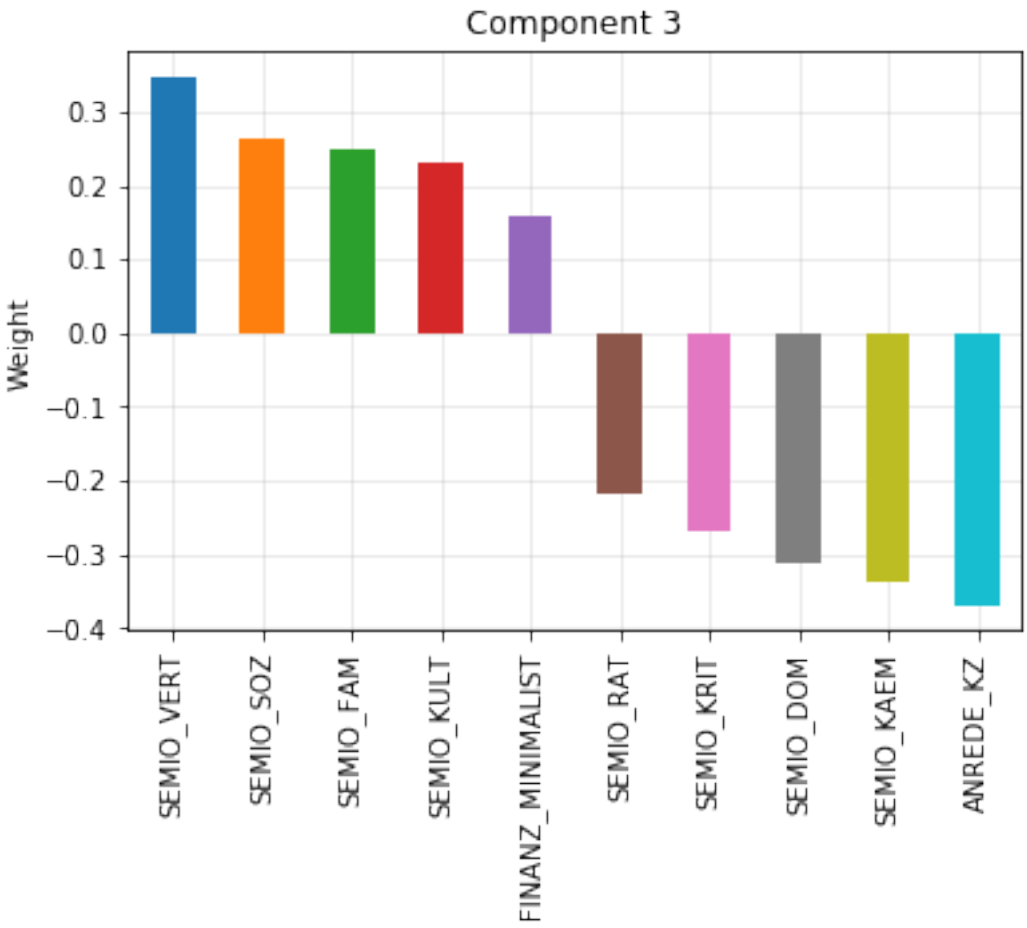
In [48]:
```
# Map weights for the first principal component to corresponding feature names
# and then print the linked values, sorted by weight.
weights = plot_weights(azdias_clean_scaled, pca, 0,5)
```



In [49]:
```
# Map weights for the second principal component to corresponding feature names
# and then print the linked values, sorted by weight.
weights = plot_weights(azdias_clean_scaled, pca, 1,5)
```



In [50]:
```
# Map weights for the third principal component to corresponding feature names
# and then print the linked values, sorted by weight.
weights = plot_weights(azdias_clean_scaled, pca, 2,5)
```



1.3 Discussion 2.3: Interpret Principal Components

1.3.1 First Principal Component

Positive weights

- PLZ8_ANTG3 (Number of 6-10 family houses in the PLZ8 region) and PLZ8_ANTG4 (Number of 10+ family houses in the PLZ8 region) seems to be positively correlated.

Negative weights

- FINANZ_MINIMALIST (Financial typology: low financial interest) and PLZ8_ANTG1 (Number of 1-2 family houses in the PLZ8 region) seems to be negatively correlated

1.3.2 Second Principal Component

Positive weights

- FINANZ_VORSORGER (Financial typology: be prepared) and SEMIO_ERL (Personality typology: event oritented) seems to be positively correlated.

Negative weights

- FINANZ_SPARER (Financial typology: money-saver) and SEMIO_PLFUCHT (dutiful) seems to have a negative correlation.

1.3.3 Third Principal Component

Positive weights

- SEMIO_FAM (Personality typology: family-minded) and SEMIO_SOZ (Personality typology: socially-minded) seems to be positively correlated.

Negative weights

- SEMIO_KAEM (Personality typology: combative attitude) and SEMIO_DOM (Personality typology: dominant-minded) seems to be negatively correlated

The first component is about the number of people inside the household. The second component seems to be about financial. And for the third component, it seems to be more about personality.

1.4 Step 3: Clustering

1.4.1 Step 3.1: Apply Clustering to General Population

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

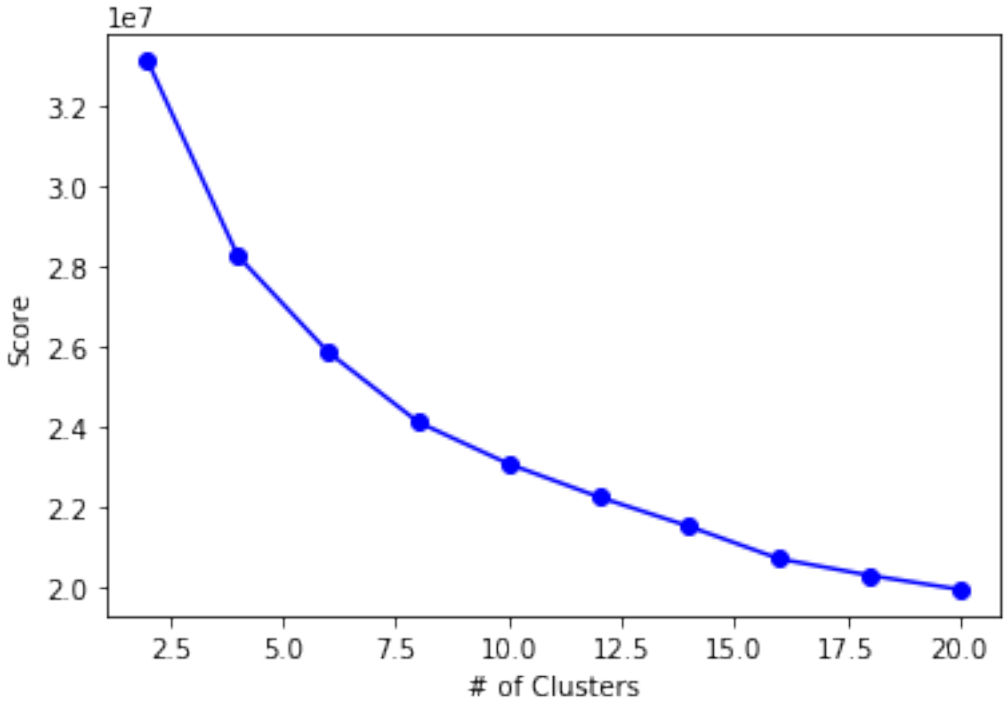
- Use sklearn's KMeans class to perform k-means clustering on the PCA-transformed data.
- Then, compute the average difference from each point to its assigned cluster's center. **Hint**: The KMeans object's `.score()` method might be useful here, but note that in sklearn, scores tend to be defined so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

```
In [51]: # Over a number of different cluster counts run k-means clustering on the data and
         # compute the average within-cluster distances.
         clusters = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
         scores = []
```

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

In [ ]:

In [52]:
```
# Investigate the change in within-cluster distance across number of clusters.
# HINT: Use matplotlib's plot function to visualize this relationship.
plt.plot(clusters, scores, marker = 'o', color = 'blue')
plt.xlabel('# of Clusters')
plt.ylabel('Score')
plt.show()
```



In [53]:
```
# 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=kmeans.fit(data_PCA)
data_pred=model.predict(data_PCA)
```

1.4.2 Discussion 3.1: Apply Clustering to General Population

Looking at the plot above we can see on the X axis that 14 seems to be the elbow and therefore we will use 14 clusters.

1.4.3 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 [54]:
```
# Load in the customer demographics data.
customers = pd.read_csv('Udacity_CUSTOMERS_Subset.csv', sep=';')
```

In [55]:
```
customers.head()
```

Out[55]:

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

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

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

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

```
3         1.0         0.0          1.0       3.0       4.0     1.0
4         2.0         1.0          2.0       3.0       3.0     3.0

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

[5 rows x 85 columns]
```

```
In [56]: # Apply preprocessing, feature transformation, and clustering from the general
         # demographics onto the customer data, obtaining cluster predictions for the
         # customer demographics data.
         # Get clean data
         customers_clean = azidas_clean(customers)
```

```
In [57]: # Replace NaN
         customers_clean_imputed = pd.DataFrame(fill_values_missing.transform(customers_clean))
         customers_clean_imputed.columns = customers_clean.columns
         customers_clean_imputed.index = customers_clean.index
```

```
In [58]: # Apply scaler
         customers_clean_scaled = scaler.transform(customers_clean_imputed)
         customers_clean_scaled = pd.DataFrame(customers_clean_scaled, columns=list(customers_cl
         # PCA transformation
         c_pca = pca_40.transform(customers_clean_scaled)
```

```
In [59]: # Predict using Kmeans model
         c_prediction = model.predict(c_pca)
```

1.4.4 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.

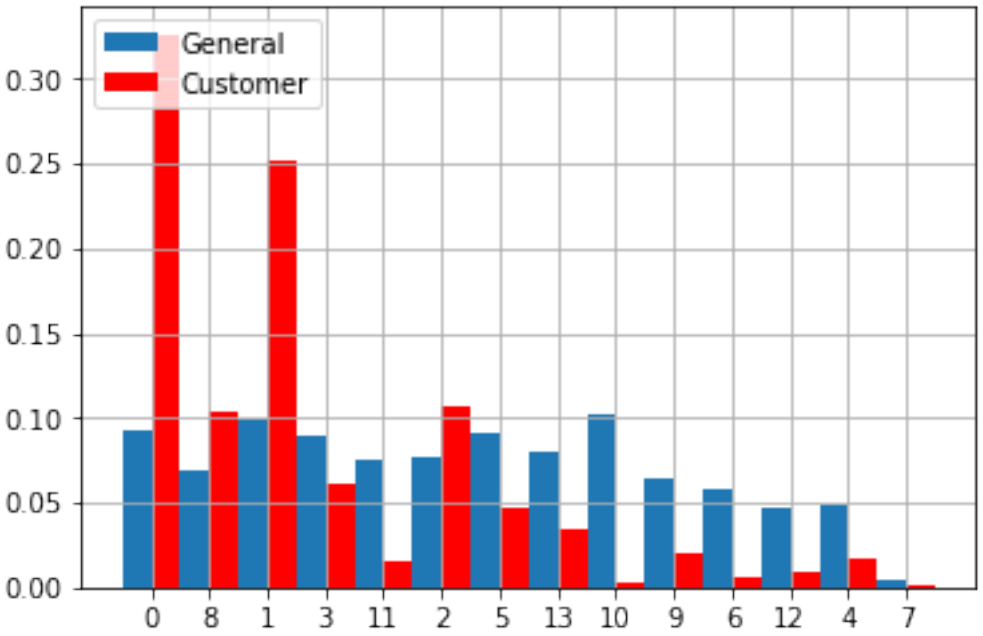
Consider the proportion of persons in each cluster for the general population, and the proportions for the customers. If we think the company's customer base to be universal, then the cluster assignment proportions should be fairly similar between the two. If there are only particular segments of the 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 [60]:
```
# Compare the proportion of data in each cluster for the customer data to the
# proportion of data in each cluster for the general population.
from collections import Counter
# generating proportion for the customer clusters.
labels, values = zip(*Counter(c_prediction).items())
v=list(values)
v[:] = [x/len(c_prediction) for x in v]
indexes = np.arange(len(labels))
# generating proportion for the azdias clusters.
labels1, values1 = zip(*Counter(data_pred).items())
v1=list(values1)
v1[:] = [x/len(data_pred) for x in v1]
indexes1 = np.arange(len(labels))
# configuring the graph
width = .5
plt.bar(indexes1, v1, width ,label='General')
plt.bar(indexes+width, v, width,  color='r' , label='Customer')
plt.xticks(indexes + width * 0.5, labels)
plt.legend(loc='upper left')
plt.grid()
plt.show()
```




```
In [61]: # What kinds of people are part of a cluster that is overrepresented in the
         # customer data compared to the general population?
         centroid_4 = scaler.inverse_transform(pca_40.inverse_transform(model.cluster_centers_[4

In [62]: overlyrepresented_c = pd.Series(data = centroid_4, index = customers_clean.columns)

In [63]: overlyrepresented_c

Out[63]: ALTERSKATEGORIE_GROB       1.469467
         ANREDE_KZ                  2.022148
         FINANZ_MINIMALIST          1.193696
         FINANZ_SPARER              4.232447
         FINANZ_VORSORGER           2.187280
         FINANZ_ANLEGER             4.115137
         FINANZ_UNAUFFAELLIGER      3.617001
         FINANZ_HAUSBAUER           4.164524
         GREEN_AVANTGARDE           0.034964
         HEALTH_TYP                 2.171886
         RETOURTYP_BK_S             2.409440
         SEMIO_SOZ                  3.187626
         SEMIO_FAM                  3.941149
         SEMIO_REL                  5.428611
         SEMIO_MAT                  5.241666
         SEMIO_VERT                 2.231456
         SEMIO_LUST                 2.496571
         SEMIO_ERL                  3.996831
```

```
SEMIO_KULT                  4.273675
SEMIO_RAT                   6.478831
SEMIO_KRIT                  5.501604
SEMIO_DOM                   6.319838
SEMIO_KAEM                  6.203085
SEMIO_PFLICHT               6.433973
SEMIO_TRADV                 6.113337
SOHO_KZ                     0.007963
VERS_TYP                    1.455227
ANZ_PERSONEN                1.512969
ANZ_TITEL                   0.000037
HH_EINKOMMEN_SCORE          5.586963
                            ...
KONSUMNAEHE                 2.105327
MIN_GEBAEUDEJAHR         1992.384225
OST_WEST_KZ                 0.280664
KBA05_ANTG1                 0.382120
KBA05_ANTG2                 1.297031
KBA05_ANTG3                 1.211225
KBA05_ANTG4                 0.600596
KBA05_GBZ                   2.150111
BALLRAUM                    3.208500
EWDICHTE                    5.159695
INNENSTADT                  3.398873
GEBAEUDETYP_RASTER          3.419209
KKK                         2.880837
MOBI_REGIO                  1.769944
ONLINE_AFFINITAET           3.138031
REGIOTYP                    5.020411
KBA13_ANZAHL_PKW          522.166276
PLZ8_ANTG1                  1.472583
PLZ8_ANTG2                  3.312014
PLZ8_ANTG3                  2.415951
PLZ8_ANTG4                  1.278428
PLZ8_HHZ                    3.759659
PLZ8_GBZ                    2.716037
ARBEIT                      3.683266
ORTSGR_KLS9                 6.957629
RELAT_AB                    3.720600
DECADE                      5.660252
MOVEMENT                    0.965036
WEALTH                      4.322140
LIFE_STAGE                  2.074654
Length: 64, dtype: float64
```

In [64]:
```
# What kinds of people are part of a cluster that is underrepresented in the
# customer data compared to the general population?
centroid_14 = scaler.inverse_transform(pca_40.inverse_transform(model.cluster_centers_[
```

```
In [65]: underlyrepresented_c = pd.Series(data = centroid_14, index = customers_clean.columns)

In [66]: underlyrepresented_c

Out[66]: ALTERSKATEGORIE_GROB        3.350870
         ANREDE_KZ                   1.035424
         FINANZ_MINIMALIST           3.434535
         FINANZ_SPARER               1.983314
         FINANZ_VORSORGER            4.151951
         FINANZ_ANLEGER              1.682503
         FINANZ_UNAUFFAELLIGER       1.798860
         FINANZ_HAUSBAUER            3.418189
         GREEN_AVANTGARDE            0.085211
         HEALTH_TYP                  2.426246
         RETOURTYP_BK_S              4.381781
         SEMIO_SOZ                   5.046072
         SEMIO_FAM                   5.131093
         SEMIO_REL                   3.713322
         SEMIO_MAT                   4.206604
         SEMIO_VERT                  5.829041
         SEMIO_LUST                  5.048131
         SEMIO_ERL                   4.250871
         SEMIO_KULT                  5.030558
         SEMIO_RAT                   2.608367
         SEMIO_KRIT                  3.659741
         SEMIO_DOM                   3.428769
         SEMIO_KAEM                  2.862970
         SEMIO_PFLICHT               3.249338
         SEMIO_TRADV                 2.891323
         SOHO_KZ                     0.007372
         VERS_TYP                    1.666698
         ANZ_PERSONEN                1.489832
         ANZ_TITEL                  -0.000137
         HH_EINKOMMEN_SCORE          5.297074
                                     ...
         KONSUMNAEHE                 2.386343
         MIN_GEBAEUDEJAHR         1992.422005
         OST_WEST_KZ                 0.231721
         KBA05_ANTG1                 0.596881
         KBA05_ANTG2                 1.986006
         KBA05_ANTG3                 1.453209
         KBA05_ANTG4                 0.123379
         KBA05_GBZ                   2.666531
         BALLRAUM                    3.726130
         EWDICHTE                    4.697453
         INNENSTADT                  3.895458
         GEBAEUDETYP_RASTER          3.620138
         KKK                         2.879324
```

```
MOBI_REGIO                 2.230249
ONLINE_AFFINITAET          1.831538
REGIOTYP                   4.878536
KBA13_ANZAHL_PKW         573.470755
PLZ8_ANTG1                 1.689667
PLZ8_ANTG2                 3.424114
PLZ8_ANTG3                 2.209526
PLZ8_ANTG4                 0.962307
PLZ8_HHZ                   3.728130
PLZ8_GBZ                   3.081168
ARBEIT                     3.614310
ORTSGR_KLS9                6.217915
RELAT_AB                   3.674767
DECADE                     3.554226
MOVEMENT                   0.914789
WEALTH                     4.141319
LIFE_STAGE                 2.467482
Length: 64, dtype: float64
```

1.4.5 Discussion 3.3: Compare Customer Data to Demographics Data

The analysis shows

Cluster four is overely represented in the Customer Dataset in comparison to the General Population Dataset. We can see that some characteristics of the group population are relatively popular with mail-order company in life stage of Older Families, Mature Couples, and Families With School Age

Cluster fourteen is underly represented in the Customer Dataset. We can see that some characterstics of the pieces of the population that are relatively unpopular with the company are in area where the in life stage of Pre_Family Couples, Young Couples, and Singles is = 1.98

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

In [ ]:

In [ ]: