Identify_Customer_Segments

June 14, 2020

1 Project: Identify Customer Segments

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

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

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

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

In [1]:
```
# import libraries here; add more as necessary
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import ast
# 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]: # Check the structure of the data after it's loaded (e.g. print the number of
        # rows and columns, print the first few rows)
        print("Shape of Udacity_AZDIAS_Subset.csv :",azdias.shape)
        print("Shape of AZDIAS_Feature_Summary.csv :",feat_info.shape)
```

```
Shape of Udacity_AZDIAS_Subset.csv : (891221, 85)
Shape of AZDIAS_Feature_Summary.csv : (85, 4)
```

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

Out[6]:

	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]
```

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]: #Number of missing values
        azdias.isnull().sum().sum()

Out[7]: 4896838

In [8]: # Identify missing or unknown data values and convert them to NaNs.
        def unknown_to_nan(df,features):
            for name, uk in zip(features["attribute"], features["missing_or_unknown"]):
```

```
                try:
                    rep=ast.literal_eval(uk)
                except:
                    rep=uk.replace("[","").replace("]","").split(",")
                if rep:
                    df[name]=df[name].replace(rep,np.NaN)
            return df
        azdias=unknown_to_nan(azdias,feat_info)
```

```
In [9]: #Number of missing values after converting
        azdias.isnull().sum().sum()
```

```
Out[9]: 8373929
```

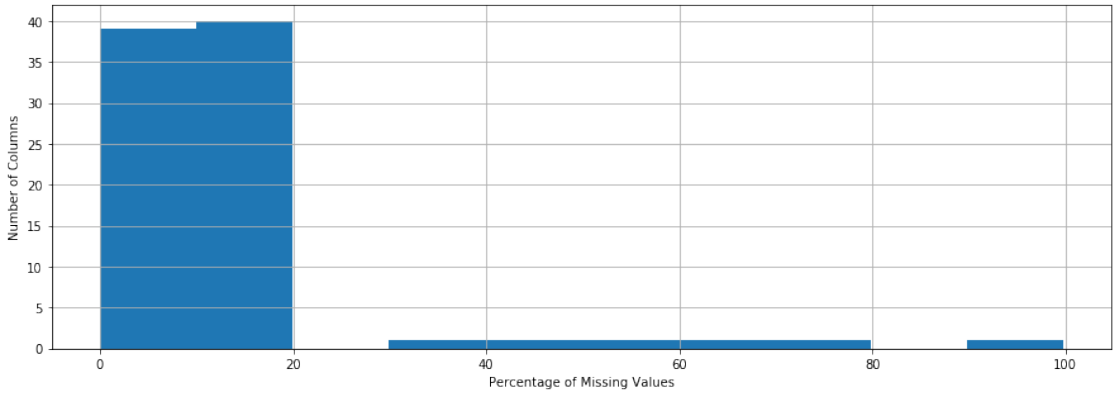
Step 1.1.2: Assess Missing Data in Each Column How much missing data is present in each column? There are a few columns that are outliers in terms of the proportion of values that are missing. You will want to use matplotlib's `hist()` function to visualize the distribution of missing value counts to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

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

```
In [10]: # Perform an assessment of how much missing data there is in each column of the
         # dataset.
         column_missing_data=(azdias.isnull().sum()/len(azdias)).sort_values(ascending=False)*10
         print(column_missing_data.head())
```

```
TITEL_KZ        99.757636
AGER_TYP        76.955435
KK_KUNDENTYP    65.596749
KBA05_BAUMAX    53.468668
GEBURTSJAHR     44.020282
dtype: float64
```

```
In [11]: # Investigate patterns in the amount of missing data in each column.
         fig, ax = plt.subplots(figsize =(15, 5))
         column_missing_data.hist()
         plt.ylabel('Number of Columns')
         plt.xlabel('Percentage of Missing Values')
         plt.show()
```




```
In [12]: # Remove the outlier columns from the dataset. (You'll perform other data
         # engineering tasks such as re-encoding and imputation later.)
         drop_columns=list(column_missing_data[column_missing_data>33].index)
         print("Dropped column:")
         print(drop_columns)
         azdias.drop(drop_columns,axis="columns",inplace=True)
```

```
Dropped column:
['TITEL_KZ', 'AGER_TYP', 'KK_KUNDENTYP', 'KBA05_BAUMAX', 'GEBURTSJAHR', 'ALTER_HH']
```

```
In [13]: # shape of dataframe after droping columns
         azdias.shape
```

```
Out[13]: (891221, 79)
```

```
In [14]: #Number of missing values after dropped columns
         azdias.isnull().sum().sum()
```

```
Out[14]: 5035304
```

Discussion 1.1.2: Assess Missing Data in Each Column Are there any patterns in missing values? Yes, Most of the columns have less then 34% data missing.
Which columns were removed from the dataset? TITEL_KZ, AGER_TYP, KK_KUNDENTYP, KBA05_BAUMAX, GEBURTSJAHR, ALTER_HH

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

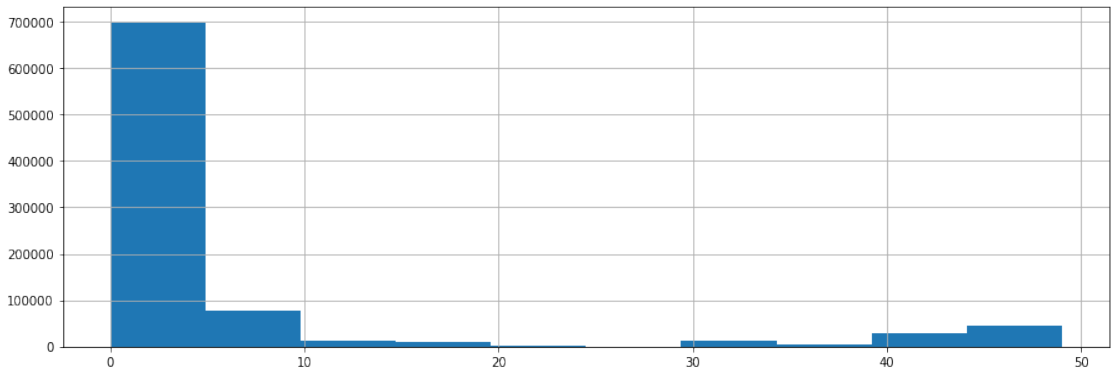
between the two groups. Select at least five of these columns and compare the distribution of values. - You can use seaborn's `countplot()` function to create a bar chart of code frequencies and matplotlib's `subplot()` function to put bar charts for the two subplots side by side. - To reduce repeated 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 [15]:
```
# How much data is missing in each row of the dataset?
row_missing_data=azdias.isnull().sum(axis=1).sort_values(ascending=False)
print(row_missing_data.head(10))
```

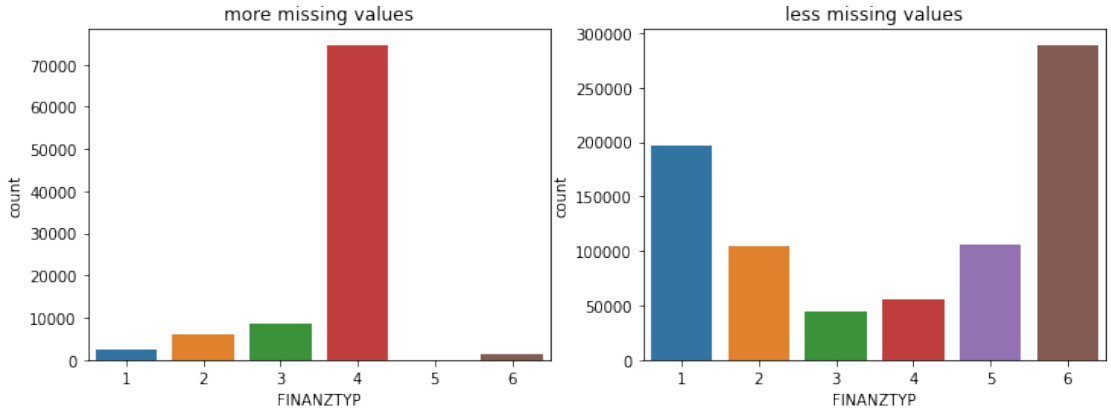
```
643174    49
732775    49
472919    48
183108    47
139316    47
691141    47
691142    47
691171    47
691183    47
139332    47
dtype: int64
```

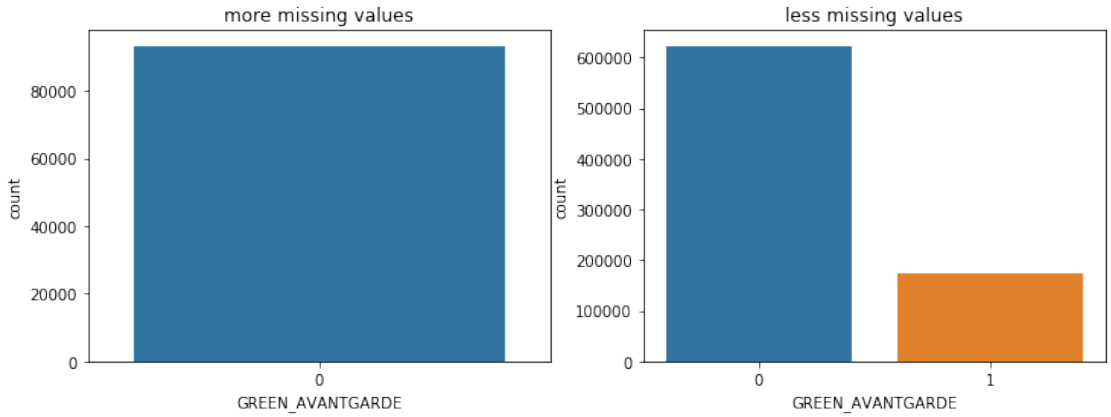
In [16]:
```
# Write code to divide the data into two subsets based on the number of missing
# values in each row.
fig, ax = plt.subplots(figsize =(15, 5))
row_missing_data.hist()
plt.show()
row_above_30=azdias[row_missing_data>=30]
row_below_30=azdias[row_missing_data<30]
```

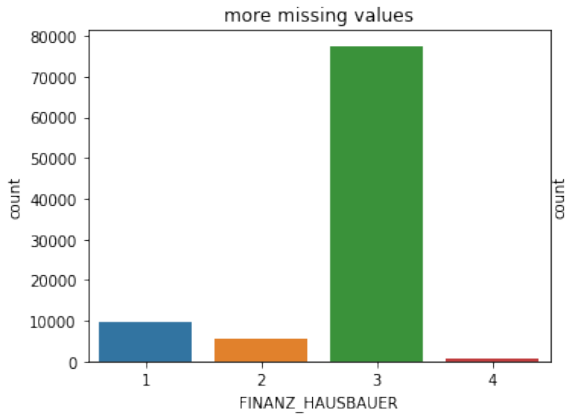


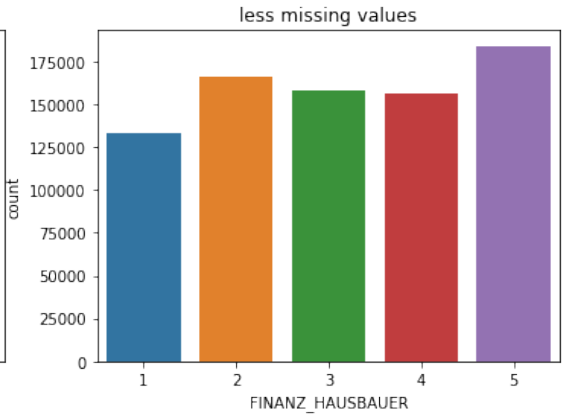
```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:6: UserWarning: Boolean Series key

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:7: UserWarning: Boolean Series key
  import sys
```

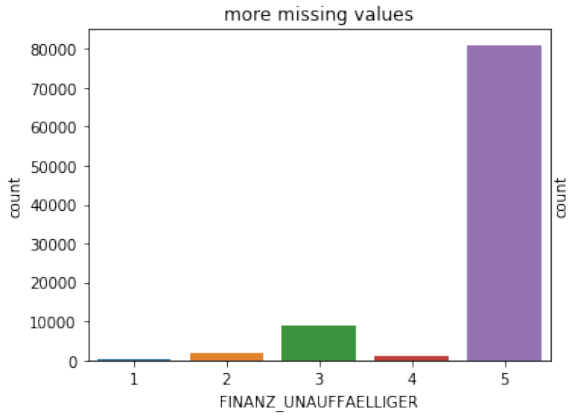
In [17]:
```
# Compare the distribution of values for at least five columns where there are
# no or few missing values, between the two subsets.
no_missing_columns=column_missing_data[column_missing_data==0].index.tolist()
for i in range(5):
    fig = plt.figure(10, figsize=(12,4))
    ax1 = fig.add_subplot(121)
    ax1.title.set_text("more missing values")
    sns.countplot(row_above_30[no_missing_columns[i]])
    ax2 = fig.add_subplot(122)
    ax2.title.set_text("less missing values")
    sns.countplot(row_below_30[no_missing_columns[i]])
    plt.show()
```

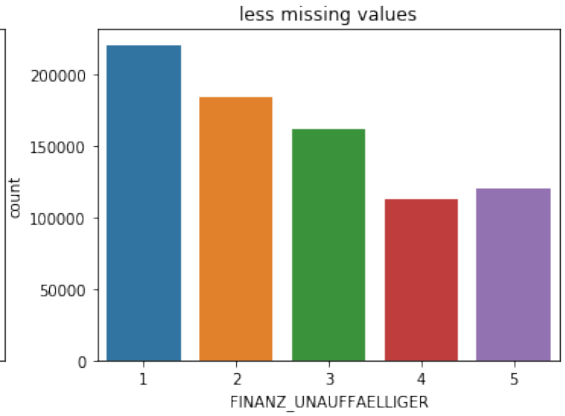


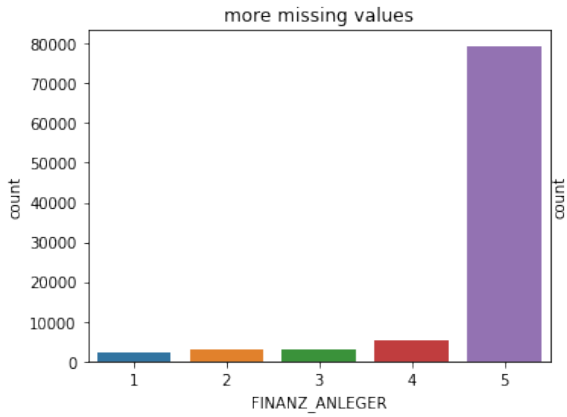



more missing values
count
FINANZ_HAUSBAUER


less missing values
count
FINANZ_HAUSBAUER


more missing values
count
FINANZ_UNAUFFAELLIGER


less missing values
count
FINANZ_UNAUFFAELLIGER


more missing values
count
FINANZ_ANLEGER


less missing values
count
FINANZ_ANLEGER

```
In [18]: filled_data=row_below_30.fillna(row_below_30.mode().iloc[0])

In [19]: filled_data.isnull().sum().sum()

Out[19]: 0

In [20]: filled_data.shape

Out[20]: (798061, 79)
```

Discussion 1.1.3: Assess Missing Data in Each Row In particular, data related to finance have the opposite values in both groups. NaN values were replaced with the most common value used in each category.

Are the data with lots of missing values are qualitatively different from data with few or no missing values? Yes, there is a difference between data distributions.

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 [21]: # How many features are there of each data type?
         feat_info["type"].value_counts()

Out[21]: ordinal        49
         categorical    21
         mixed           7
         numeric         7
         interval        1
         Name: type, dtype: int64

In [22]: feat_info.head()
```

Out[22]:

	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]

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

In [23]:
```
# Assess categorical variables: which are binary, which are multi-level, and
# which one needs to be re-encoded?
colegorical_column=feat_info["attribute"][feat_info["type"]=="categorical"].values
print(colegorical_column)
```

```
['AGER_TYP' 'ANREDE_KZ' 'CJT_GESAMTTYP' 'FINANZTYP' 'GFK_URLAUBERTYP'
 'GREEN_AVANTGARDE' 'LP_FAMILIE_FEIN' 'LP_FAMILIE_GROB' 'LP_STATUS_FEIN'
 'LP_STATUS_GROB' 'NATIONALITAET_KZ' 'SHOPPER_TYP' 'SOHO_KZ' 'TITEL_KZ'
 'VERS_TYP' 'ZABEOTYP' 'KK_KUNDENTYP' 'GEBAEUDETYP' 'OST_WEST_KZ'
 'CAMEO_DEUG_2015' 'CAMEO_DEU_2015']
```

In [24]:
```
two_level=[]
multi_level=[]
for i in colegorical_column:
    if i in filled_data.columns and filled_data[i].nunique()>2:
        multi_level.append(i)
    else:
        two_level.append(i)
print("Two level: ",two_level)
print()
print("Multi level: ",multi_level)
```

```
Two level:  ['AGER_TYP', 'ANREDE_KZ', 'GREEN_AVANTGARDE', 'SOHO_KZ', 'TITEL_KZ', 'VERS_TYP', 'KK

Multi level:  ['CJT_GESAMTTYP', 'FINANZTYP', 'GFK_URLAUBERTYP', 'LP_FAMILIE_FEIN', 'LP_FAMILIE_G
```

In [25]:
```
for i in two_level:
    if i in filled_data.columns:
        print(i,filled_data[i].unique())
```

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

```
In [26]: multi_level.append("OST_WEST_KZ")
         two_level.remove("OST_WEST_KZ")
```

```
In [27]: for i in two_level:
             if i in filled_data.columns:
                 print(i,filled_data[i].unique())
```

```
ANREDE_KZ [2 1]
GREEN_AVANTGARDE [0 1]
SOHO_KZ [ 1.  0.]
VERS_TYP [ 2.  1.]
```

```
In [28]: # Re-encode categorical variable(s) to be kept in the analysis.
         one_hot_encoded_data=pd.get_dummies(filled_data,columns=multi_level,drop_first=True)
         one_hot_encoded_data.shape
```

```
Out[28]: (798061, 181)
```

Discussion 1.2.1: Re-Encode Categorical Features I have keeped binary and multi-level features, and I have applied OneHotEncoder to multilevel features to ensure their values were 1 and 0. Thus, new features have increased to 181

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 [29]: mixed_column=feat_info["attribute"][feat_info["type"]=="mixed"].values
         print(mixed_column)
```

```
['LP_LEBENSPHASE_FEIN' 'LP_LEBENSPHASE_GROB' 'PRAEGENDE_JUGENDJAHRE'
 'WOHNLAGE' 'CAMEO_INTL_2015' 'KBA05_BAUMAX' 'PLZ8_BAUMAX']
```

1.1.3 PRAEGENDE_JUGENDJAHRE

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

In [30]:
```
# Investigate "PRAEGENDE_JUGENDJAHRE" and engineer two new variables.
def avantgarde_vs_mainstream(n):
    if n in (1,3,5,8,10,12,14):
        return 0
    else:
        return 1
def east_vs_west(n):
    if n in (1,2,3,4,5,8,9,14,15):
        return 0
    elif n in (6,10,11):
        return 1
    else:
        return 3
def decade(x):
    if x in (1,2):
        return 1
    if x in (3,4):
        return 2
    if x in (5,6,7):
        return 3
    if x in (8,9):
        return 4
    if x in (10,11,12,13):
        return 5
    if x in (14,15):
        return 6
```

In [31]:
```
one_hot_encoded_data["avantgarde_vs_mainstream"]=one_hot_encoded_data["PRAEGENDE_JUGEND
one_hot_encoded_data["east_vs_west"]=one_hot_encoded_data["PRAEGENDE_JUGENDJAHRE"].appl
one_hot_encoded_data["decade"]=one_hot_encoded_data["PRAEGENDE_JUGENDJAHRE"].apply(deca
one_hot_encoded_data.drop(columns="PRAEGENDE_JUGENDJAHRE",axis=1, inplace=True)
```

In [32]:
```
one_hot_encoded_data.columns[-3:]
```

Out[32]:
```
Index(['avantgarde_vs_mainstream', 'east_vs_west', 'decade'], dtype='object')
```

1.1.4 CAMEO_INTL_2015

German CAMEO: Wealth / Life Stage Typology, mapped to international code - -1: unknown - 11: Wealthy Households - Pre-Family Couples & Singles - 12: Wealthy Households - Young Couples With Children - 13: Wealthy Households - Families With School Age Children - 14: Wealthy Households - Older Families & Mature Couples - 15: Wealthy Households - Elders In Retirement - 21: Prosperous Households - Pre-Family Couples & Singles - 22: Prosperous Households - Young Couples With Children - 23: Prosperous Households - Families With School Age Children - 24: Prosperous Households - Older Families & Mature Couples - 25: Prosperous Households - Elders In Retirement - 31: Comfortable Households - Pre-Family Couples & Singles - 32: Comfortable Households - Young Couples With Children - 33: Comfortable Households - Families With School Age Children - 34: Comfortable Households - Older Families & Mature Couples - 35: Comfortable Households - Elders In Retirement - 41: Less Affluent Households - Pre-Family Couples & Singles - 42: Less Affluent Households - Young Couples With Children - 43: Less Affluent Households - Families With School Age Children - 44: Less Affluent Households - Older Families & Mature Couples - 45: Less Affluent Households - Elders In Retirement - 51: Poorer Households - Pre-Family Couples & Singles - 52: Poorer Households - Young Couples With Children - 53: Poorer Households - Families With School Age Children - 54: Poorer Households - Older Families & Mature Couples - 55: Poorer Households - Elders In Retirement - XX: unknown

In [33]:
```
# Investigate "CAMEO_INTL_2015" and engineer two new variables.
def wealth(x):
    if int(x) // 10 == 1:
        return 0
    elif int(x) // 10 == 2:
        return 1
    elif int(x) // 10 == 3:
        return 2
    elif int(x) // 10 == 4:
        return 3
    elif int(x) // 10 == 5:
        return 4

def life_stage(x):
    if int(x) % 10 == 1:
        return 0
    elif int(x) % 10 == 2:
        return 1
    elif int(x) % 10 == 3:
        return 2
    elif int(x) % 10 == 4:
        return 3
    elif int(x) % 10 == 5:
        return 4
```

In [34]:
```
one_hot_encoded_data["wealth"]=one_hot_encoded_data["CAMEO_INTL_2015"].apply(wealth)
one_hot_encoded_data["life_stage"]=one_hot_encoded_data["CAMEO_INTL_2015"].apply(life_s
one_hot_encoded_data.drop("CAMEO_INTL_2015", axis=1, inplace=True)
```

```
In [35]: one_hot_encoded_data.columns[-5:]
```

```
Out[35]: Index(['avantgarde_vs_mainstream', 'east_vs_west', 'decade', 'wealth',
                'life_stage'],
               dtype='object')
```

1.1.5 LP_LEBENSPHASE_FEIN

Life stage, fine scale - 1: single low-income earners of younger age - 2: single low-income earners of middle age - 3: single average earners of younger age - 4: single average earners of middle age - 5: single low-income earners of advanced age - 6: single low-income earners at retirement age - 7: single average earners of advanced age - 8: single average earners at retirement age - 9: single independent persons - 10: wealthy single homeowners - 11: single homeowners of advanced age - 12: single homeowners at retirement age - 13: single top earners of higher age - 14: low-income and average earner couples of younger age - 15: low-income earner couples of higher age - 16: average earner couples of higher age - 17: independent couples - 18: wealthy homeowner couples of younger age - 19: homeowner couples of higher age - 20: top earner couples of higher age - 21: single parent low-income earners - 22: single parent average earners - 23: single parent high-income earners - 24: low-income earner families - 25: average earner families - 26: independent families - 27: homeowner families - 28: top earner families - 29: low-income earners of younger age from multiperson households - 30: average earners of younger age from multiperson households - 31: low-income earners of higher age from multiperson households - 32: average earners of higher age from multiperson households - 33: independent persons of younger age from multiperson households - 34: homeowners of younger age from multiperson households - 35: top earners of younger age from multiperson households - 36: independent persons of higher age from multiperson households - 37: homeowners of advanced age from multiperson households - 38: homeowners at retirement age from multiperson households - 39: top earners of middle age from multiperson households - 40: top earners at retirement age from multiperson households

```
In [36]: def single_vs_family(n):
             if n in (1,2,3,4,5,6,7,8,9,10,11,12,13,21,22,23):
                 return 0
             elif n in (14,15,16,17,18,19,20,24,25,26,27,28):
                 return 1
             else:
                 return 2
         def fine_scale(n):
             if n in (40,39,35,28,23,20,18,10):
                 return 0
             elif n in (1,2,5,6,14,15,21,24,29,31):
                 return 1
             elif n in (3,4,7,8,16,22,25,30,32):
                 return 2
             elif n in (11,12,19,27,34,37,38):
                 return 3
             else:
                 return 4
```

```
In [37]: one_hot_encoded_data["single_vs_family"]=one_hot_encoded_data["LP_LEBENSPHASE_FEIN"].ap
         one_hot_encoded_data["fine_scale"]=one_hot_encoded_data["LP_LEBENSPHASE_FEIN"].apply(fi
         one_hot_encoded_data.drop("LP_LEBENSPHASE_FEIN", axis=1, inplace=True)

In [38]: one_hot_encoded_data.columns[-7:]

Out[38]: Index(['avantgarde_vs_mainstream', 'east_vs_west', 'decade', 'wealth',
                'life_stage', 'single_vs_family', 'fine_scale'],
               dtype='object')
```

Discussion 1.2.2: Engineer Mixed-Type Features Separation of existing columns into sub-columns was made here. Existing columns are also droped from the table.

The PRAEGENDE_JUGENDJAHRE column have been replaced by the three new columns avantgarde_vs_mainstream, east_vs_west, decade.

The CAMEO_INTL_2015 column have been replaced by the two new columns wealth and life_stage.

The LP_LEBENSPHASE_FEIN column have been replaced by the two new columns single_vs_family and fine_scale.

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 [39]: # 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.)
         one_hot_encoded_data.shape

Out[39]: (798061, 185)

In [40]: # Do whatever you need to in order to ensure that the dataframe only contains
         # the columns that should be passed to the algorithm functions.
         one_hot_encoded_data.describe()

Out[40]:        ALTERSKATEGORIE_GROB      ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
         count         798061.000000  798061.000000      798061.000000  798061.000000
         mean               2.796131       1.521485           3.058917       2.716047
         std                1.016691       0.499538           1.377577       1.485090
         min                1.000000       1.000000           1.000000       1.000000
         25%                2.000000       1.000000           2.000000       1.000000
         50%                3.000000       2.000000           3.000000       3.000000
```

```
75%               4.000000       2.000000           4.000000       4.000000
max               4.000000       2.000000           5.000000       5.000000

       FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  \
count     798061.000000   798061.000000          798061.000000
mean           3.432892        2.840955               2.658348
std            1.376866        1.472782               1.399530
min            1.000000        1.000000               1.000000
25%            2.000000        1.000000               1.000000
50%            4.000000        3.000000               2.000000
75%            5.000000        4.000000               4.000000
max            5.000000        5.000000               5.000000

       FINANZ_HAUSBAUER  GREEN_AVANTGARDE     HEALTH_TYP      ...        \
count     798061.000000     798061.000000  798061.000000      ...
mean           3.114102          0.219563       2.235870      ...
std            1.408109          0.413951       0.756442      ...
min            1.000000          0.000000       1.000000      ...
25%            2.000000          0.000000       2.000000      ...
50%            3.000000          0.000000       2.000000      ...
75%            4.000000          0.000000       3.000000      ...
max            5.000000          1.000000       3.000000      ...

       CAMEO_DEU_2015_9D  CAMEO_DEU_2015_9E  OST_WEST_KZ_W  \
count      798061.000000      798061.000000  798061.000000
mean            0.035828           0.007993       0.788812
std             0.185861           0.089046       0.408152
min             0.000000           0.000000       0.000000
25%             0.000000           0.000000       1.000000
50%             0.000000           0.000000       1.000000
75%             0.000000           0.000000       1.000000
max             1.000000           1.000000       1.000000

       avantgarde_vs_mainstream   east_vs_west         decade         wealth  \
count             798061.000000  798061.000000  798061.000000  798061.000000
mean                   0.219563       0.313099       4.392105       2.275604
std                    0.413951       0.687272       1.464292       1.466752
min                    0.000000       0.000000       1.000000       0.000000
25%                    0.000000       0.000000       3.000000       1.000000
50%                    0.000000       0.000000       5.000000       3.000000
75%                    0.000000       0.000000       6.000000       4.000000
max                    1.000000       3.000000       6.000000       4.000000

          life_stage  single_vs_family     fine_scale
count  798061.000000     798061.000000  798061.000000
mean        1.858483          0.645849       1.596278
std         1.488105          0.831643       1.133413
min         0.000000          0.000000       0.000000
```

```
25%         0.000000          0.000000       1.000000
50%         2.000000          0.000000       1.000000
75%         3.000000          1.000000       2.000000
max         4.000000          2.000000       4.000000

[8 rows x 185 columns]
```

```
In [41]: one_hot_encoded_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 798061 entries, 1 to 891220
Columns: 185 entries, ALTERSKATEGORIE_GROB to fine_scale
dtypes: float64(40), int64(29), uint8(116)
memory usage: 514.5 MB
```

1.1.6 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 [42]: def unknown_to_nan(df,features):
             for name, uk in zip(features["attribute"], features["missing_or_unknown"]):
                 try:
                     rep=ast.literal_eval(uk)
                 except:
                     rep=uk.replace("[","").replace("]","").split(",")
                 if rep:
                     df[name]=df[name].replace(rep,np.NaN)
             return df
         def level_est(cal_column,data):
             two_level=[]
             multi_level=[]
             for i in cal_column:
                 if i in data.columns and data[i].nunique()>2:
                     multi_level.append(i)
                 else:
                     two_level.append(i)
             multi_level.append("OST_WEST_KZ")
             two_level.remove("OST_WEST_KZ")
             return two_level, multi_level
         def avantgarde_vs_mainstream(n):
             if n in (1,3,5,8,10,12,14):
                 return 0
             else:
```

```
        return 1
def east_vs_west(n):
    if n in (1,2,3,4,5,8,9,14,15):
        return 0
    elif n in (6,10,11):
        return 1
    else:
        return 3
def decade(x):
    if x in (1,2):
        return 1
    if x in (3,4):
        return 2
    if x in (5,6,7):
        return 3
    if x in (8,9):
        return 4
    if x in (10,11,12,13):
        return 5
    if x in (14,15):
        return 6
def wealth(x):
    if int(x) // 10 == 1:
        return 0
    elif int(x) // 10 == 2:
        return 1
    elif int(x) // 10 == 3:
        return 2
    elif int(x) // 10 == 4:
        return 3
    elif int(x) // 10 == 5:
        return 4

def life_stage(x):
    if int(x) % 10 == 1:
        return 0
    elif int(x) % 10 == 2:
        return 1
    elif int(x) % 10 == 3:
        return 2
    elif int(x) % 10 == 4:
        return 3
    elif int(x) % 10 == 5:
        return 4
def single_vs_family(n):
    if n in (1,2,3,4,5,6,7,8,9,10,11,12,13,21,22,23):
        return 0
    elif n in (14,15,16,17,18,19,20,24,25,26,27,28):
```

```
            return 1
        else:
            return 2
    def fine_scale(n):
        if n in (40,39,35,28,23,20,18,10):
            return 0
        elif n in (1,2,5,6,14,15,21,24,29,31):
            return 1
        elif n in (3,4,7,8,16,22,25,30,32):
            return 2
        elif n in (11,12,19,27,34,37,38):
            return 3
        else:
            return 4
```

In [43]:
```
def clean_data(data):
    """
    Perform feature trimming, re-encoding, and engineering for demographics
    data
    INPUT: Demographics DataFrame
    OUTPUT: Trimmed and cleaned demographics DataFrame
    """
    features=pd.read_csv("AZDIAS_Feature_Summary.csv",sep=";")
    # Put in code here to execute all main cleaning steps:

    # convert missing value codes into NaNs, ...
    data=unknown_to_nan(data,features)

    # remove selected columns,
    column_missing_per=(data.isnull().sum()/len(data)).sort_values(ascending=False)*100
    drop_column_name=column_missing_per[column_missing_per>33].index
    df = data.drop(drop_column_name, axis=1)

    # remove selected rows
    row_missing_data=df.isnull().sum(axis=1).sort_values(ascending=False)
    new_df=df[row_missing_data<35]

    # filling NaN values
    new_df=new_df.fillna(data.mode().iloc[0])
    assert(new_df.isnull().sum().sum()==0)

    # select, re-encode, and engineer column values.
    cal_columns=features["attribute"][features["type"]=="categorical"].values
    two_level,multi_level=level_est(cal_columns, new_df)

    one_hot_encoded_data=pd.get_dummies(new_df, columns=multi_level, drop_first=True)

    one_hot_encoded_data["avantgarde_vs_mainstream"]=one_hot_encoded_data["PRAEGENDE_JU
```

```
    one_hot_encoded_data["east_vs_west"]=one_hot_encoded_data["PRAEGENDE_JUGENDJAHRE"].
    one_hot_encoded_data["decade"]=one_hot_encoded_data["PRAEGENDE_JUGENDJAHRE"].apply(
    one_hot_encoded_data.drop(columns="PRAEGENDE_JUGENDJAHRE",axis=1, inplace=True)

    one_hot_encoded_data["wealth"]=one_hot_encoded_data["CAMEO_INTL_2015"].apply(wealth
    one_hot_encoded_data["life_stage"]=one_hot_encoded_data["CAMEO_INTL_2015"].apply(li
    one_hot_encoded_data.drop("CAMEO_INTL_2015", axis=1, inplace=True)

    one_hot_encoded_data["single_vs_family"]=one_hot_encoded_data["LP_LEBENSPHASE_FEIN"
    one_hot_encoded_data["fine_scale"]=one_hot_encoded_data["LP_LEBENSPHASE_FEIN"].appl
    one_hot_encoded_data.drop("LP_LEBENSPHASE_FEIN", axis=1, inplace=True)

    multi_level_v2=list(one_hot_encoded_data.columns[-7:])
    multi_level_v2.remove("single_vs_family")
    multi_level_v2.remove("avantgarde_vs_mainstream")

    # re-encoding data
    final_df=pd.get_dummies(one_hot_encoded_data, columns=multi_level_v2, drop_first=Tr

    # Return the cleaned dataframe.
    return final_df
```

In [44]:
```
Azdias_data=pd.read_csv("Udacity_AZDIAS_Subset.csv",sep=";")
Customer_data=pd.read_csv("Udacity_CUSTOMERS_Subset.csv",sep=";")
clean_azdias=clean_data(Azdias_data)
clean_customer=clean_data(Customer_data)
print("[Info] Cleaning Done!")
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:21: UserWarning: Boolean Series key


[Info] Cleaning Done!
```

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 [45]: # If you've not yet cleaned the dataset of all NaN values, then investigate and
         # do that now.
         clean_azdias.isnull().sum().sum()
```

```
Out[45]: 0
```

```
In [46]: # Apply feature scaling to the general population demographics data.
         from sklearn.preprocessing import StandardScaler
         scaler=StandardScaler()
         X_scaled=scaler.fit_transform(clean_azdias)
         X_scaled.shape
```

```
Out[46]: (812077, 199)
```

1.2.2 Discussion 2.1: Apply Feature Scaling

Here StandardScaler was used to scale all numerical data to 0 and standard deviation to 1.

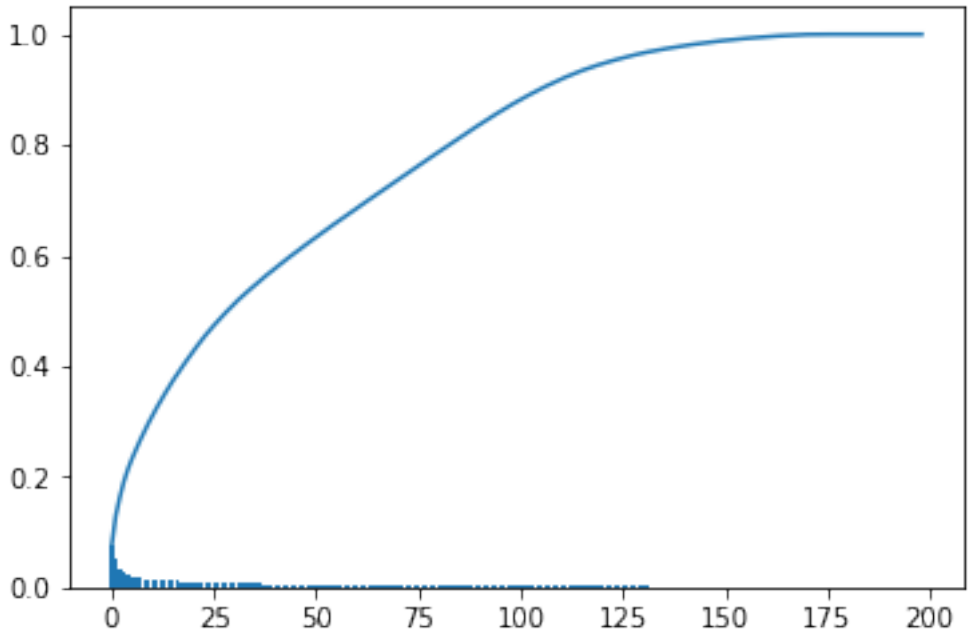
1.2.3 Step 2.2: Perform Dimensionality Reduction

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

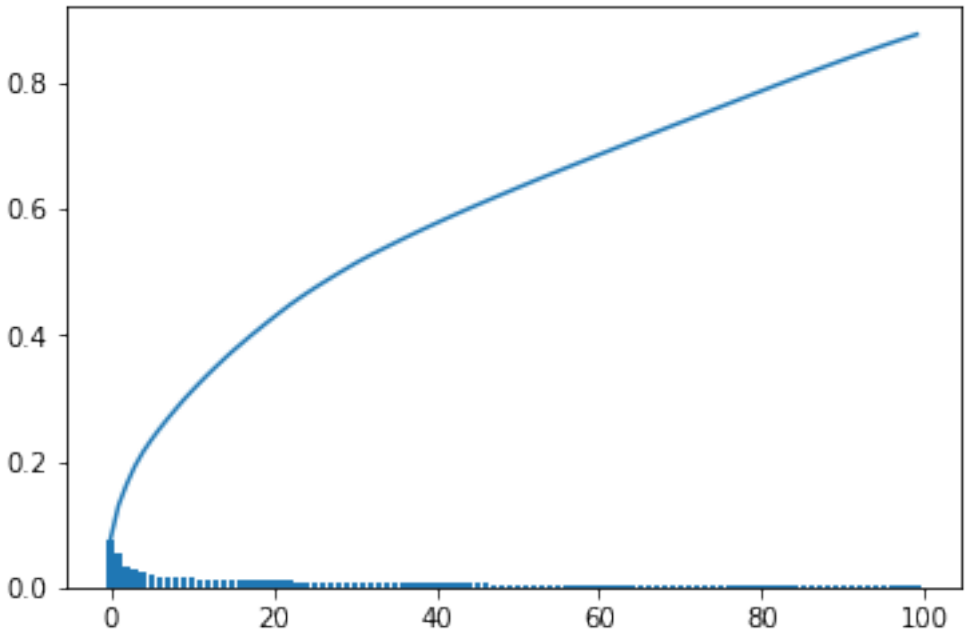
- Use sklearn's PCA class to apply principal component analysis on the data, thus finding the vectors of maximal variance in the data. To start, you should not set any parameters (so all components are computed) or set a number of components that is at least half the number of features (so there's enough 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 [47]: # Apply PCA to the data.
         from sklearn.decomposition import PCA
         pca=PCA()
         default_pca=pca.fit_transform(X_scaled)
```

In [48]:
```
# Investigate the variance accounted for by each principal component.
def screen_plot(pca):
    l=len(pca.explained_variance_ratio_)
    itr=np.arange(l)
    val=pca.explained_variance_ratio_
    plt.bar(itr, val)
    camvals=np.cumsum(val)
    plt.plot(itr, camvals)
    plt.show()
screen_plot(pca)
```



In [49]:
```
# Re-apply PCA to the data while selecting for number of components to retain.
pca=PCA(n_components=100)
data_pca=pca.fit_transform(X_scaled)
screen_plot(pca)
```



1.2.4 Discussion 2.2: Perform Dimensionality Reduction

How many principal components / transformed features are you retaining for the next step of the analysis?

By using 100 components, we capture almost 85% of the total variability in the features.

1.2.5 Step 2.3: Interpret Principal Components

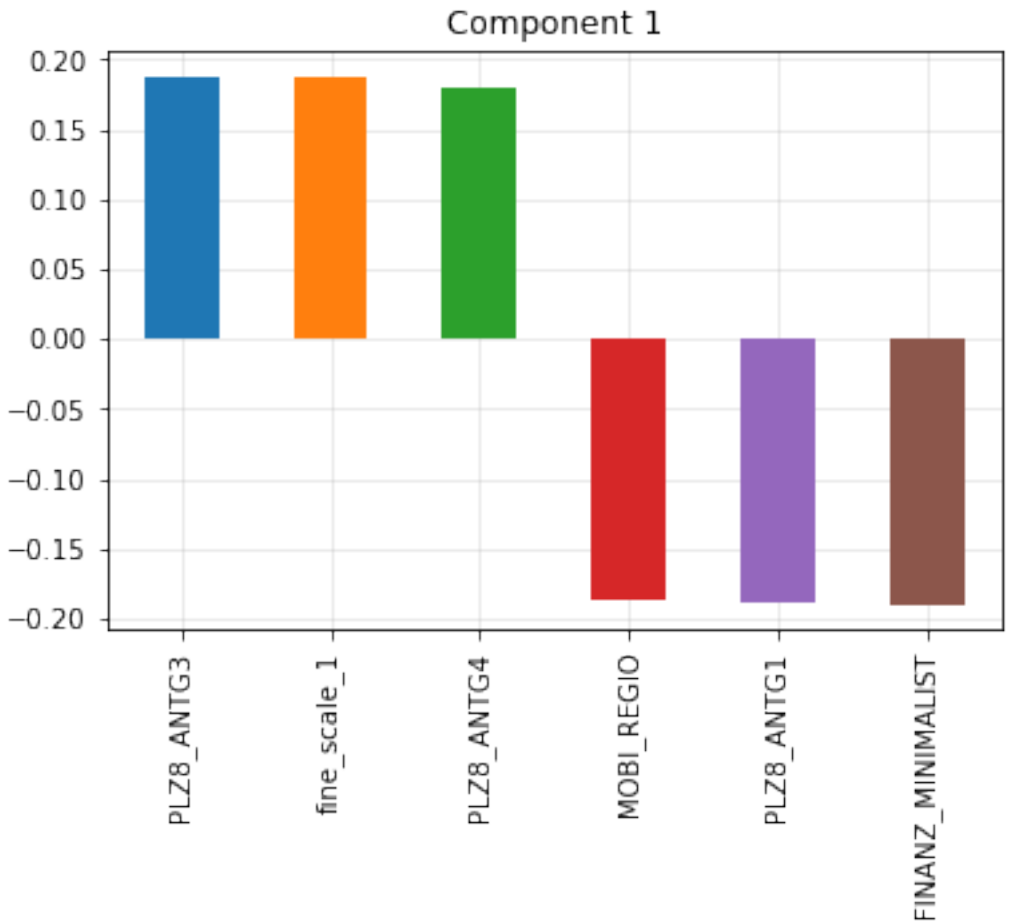
Now that we have our transformed principal components, it's a nice idea to check out the weight of each variable on the first few components to see if they can be interpreted in some fashion.

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.

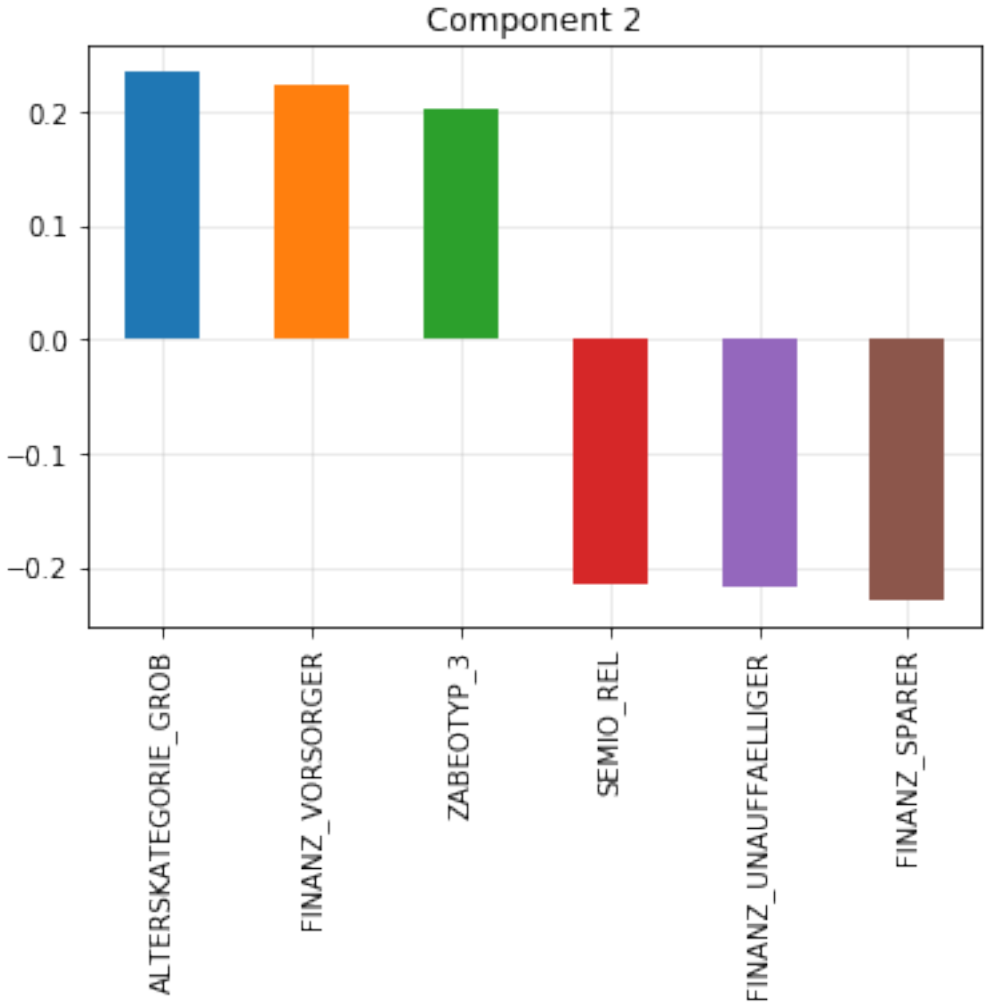
- To investigate the features, you should map each weight to their corresponding feature name, then sort the features according to weight. The most interesting features for each principal component, then, will be those at the beginning and end of the sorted list. Use the data dictionary document to help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

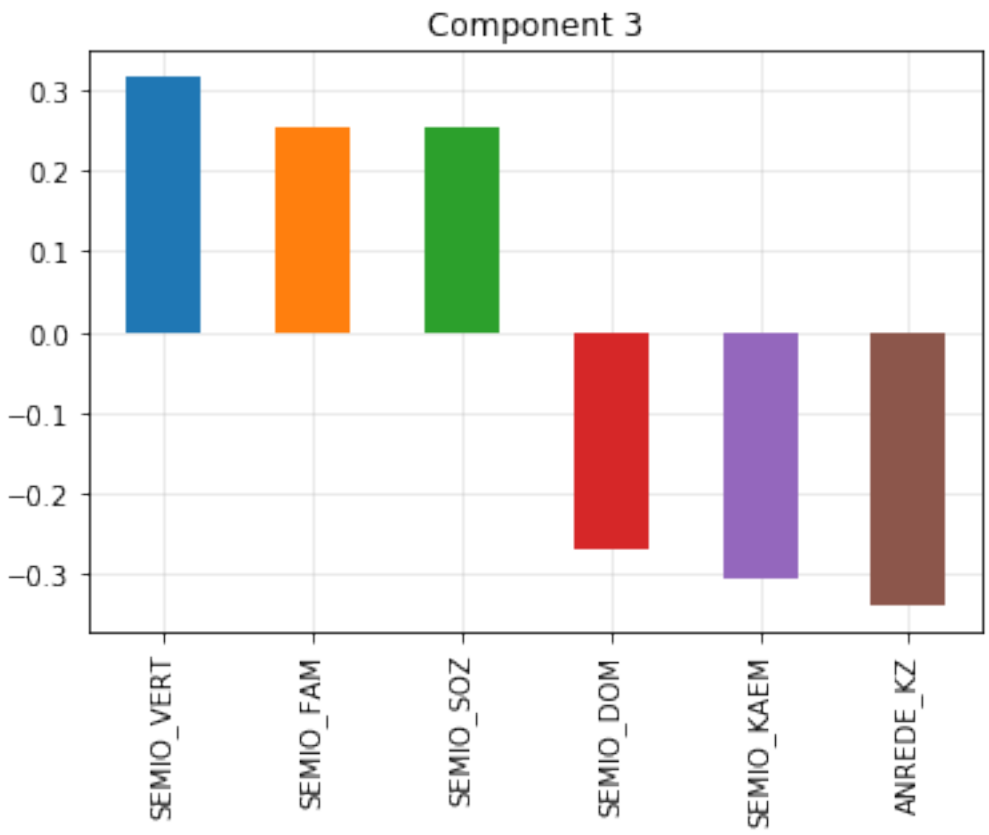
```
In [50]: # Map weights for the first principal component to corresponding feature names
         # and then print the linked values, sorted by weight.
         # HINT: Try defining a function here or in a new cell that you can reuse in the
         # other cells.
         def pca_plt(df, pca, component):
             c = pd.DataFrame(np.round(pca.components_, 4), columns = df.keys()).iloc[component-
             c.sort_values(ascending=False, inplace=True)
             c = pd.concat([c.head(3), c.tail(3)])
             c.plot(kind='bar', title='Component ' + str(component))

             ax = plt.gca()
             ax.grid(linewidth='0.5', alpha=0.5)
             ax.set_axisbelow(True)

             plt.show()
         pca_plt(clean_azdias,pca,1)
```



In [51]:
```
# Map weights for the second principal component to corresponding feature names
# and then print the linked values, sorted by weight.
pca_plt(clean_azdias,pca,2)
```



In [52]:
```
# Map weights for the third principal component to corresponding feature names
# and then print the linked values, sorted by weight.
pca_plt(clean_azdias,pca,3)
```



1.2.6 Discussion 2.3: Interpret Principal Components

Can we interpret positive and negative values from them in a meaningful way?

Yes, we can. For example, LP_STATUS_GROB_10 is positive weight but PLZ8_ANTIG1 is negative weight in the first principal component. That means if LP_STATUS_GROB_10 increases, PLZ8_ANTIG1 decreases. So, there is a positive correlation between them. The same is true for other components.

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 [53]:
```
# Over a number of different cluster counts...
from sklearn.cluster import KMeans
# run k-means clustering on the data and...

def kmean_score(data,num):
    kmean=KMeans(n_clusters=num)
    model=kmean.fit(data)
    s=np.abs(model.score(data))
    return s
```

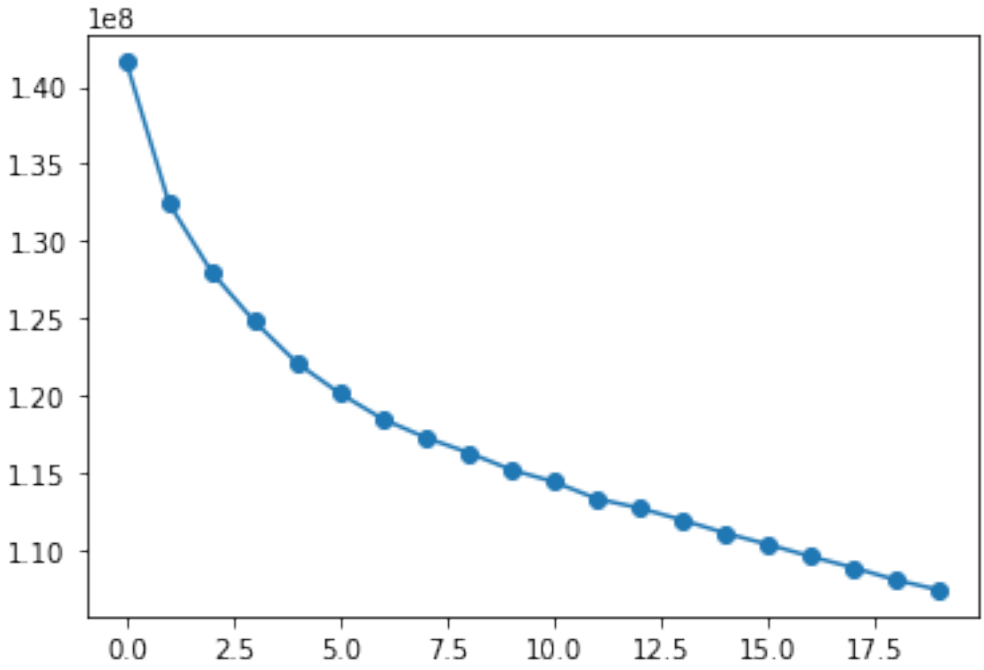
In [54]:
```
# Investigate the change in within-cluster distance across number of clusters.
# HINT: Use matplotlib's plot function to visualize this relationship.
score=[]
for i in range(1,21):
    x=kmean_score(data_pca,i)
    score.append(x)
    print(i,x)
```

```
1 141617228.187
2 132438105.243
3 127976007.525
4 124786094.378
5 122082007.595
6 120119459.973
7 118465909.063
8 117268098.057
9 116293976.875
10 115184372.609
11 114400943.025
12 113308208.94
13 112704774.26
14 111926791.441
15 111081297.288
16 110349180.315
17 109567858.054
18 108840528.706
19 108040762.005
```

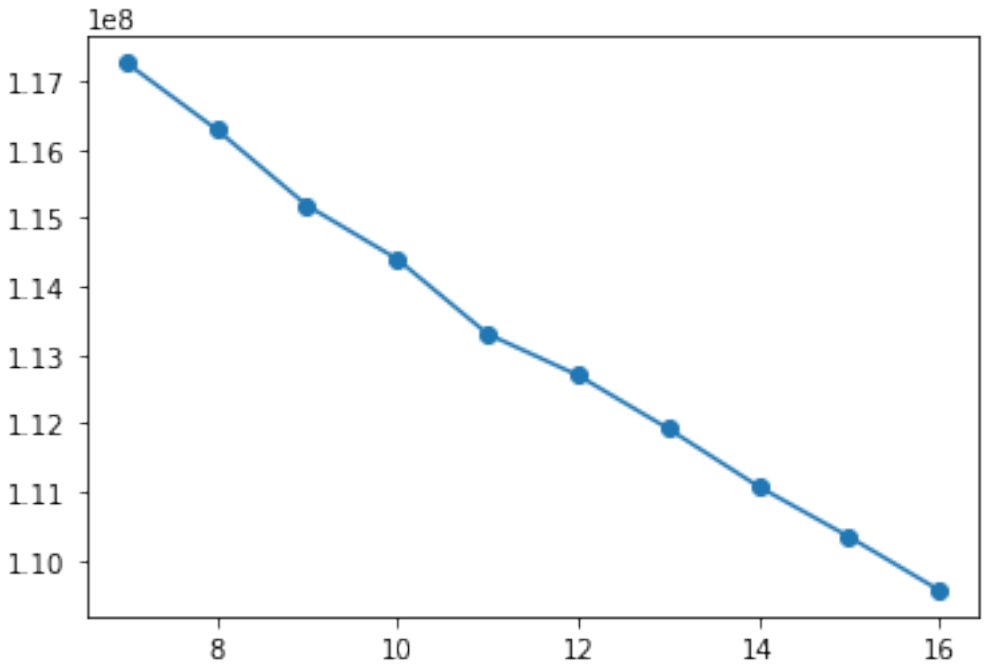
```
20 107398869.719
```

```
In [55]: plt.plot(range(20),score,marker='o')
         plt.xlabel('Number of clusters')
         plt.ylabel('Score')
         plt.show
```

```
Out[55]: <function matplotlib.pyplot.show(*args, **kw)>
```




```
In [59]: plt.plot(range(4,17),score[7:17],marker='o')
         plt.show()
```




```
In [77]: # 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 = 10)
         model_10 = kmeans.fit(data_pca)
         pred = model_10.predict(data_pca)
         print(pred)
```

```
[8 6 5 ..., 0 4 3]
```

1.3.2 Discussion 3.1: Apply Clustering to General Population

Into how many clusters have you decided to segment the population?

It looks like K=10 is the elbow from the above plot. Thus, I have decided to use 10 clusters for this problem.

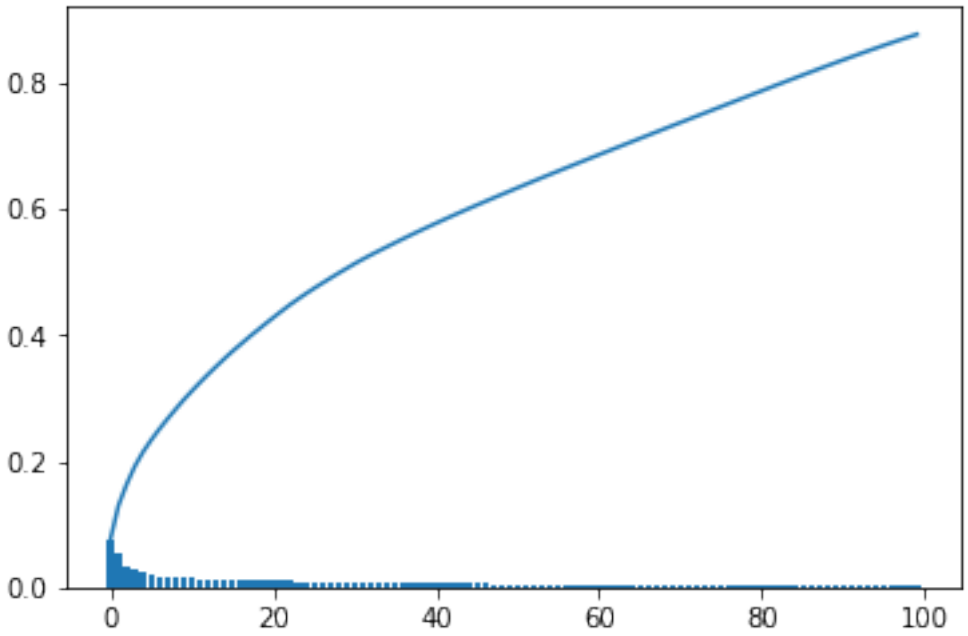
1.3.3 Step 3.2: Apply All Steps to the Customer Data

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

In [63]: # Apply preprocessing, feature transformation, and clustering from the general
         # demographics onto the customer data, obtaining cluster predictions for the
         # customer demographics data.
         customers_data=clean_data(customers)
         customers_data.shape

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:21: UserWarning: Boolean Series key


Out[63]: (144491, 198)

In [67]: list(set(clean_azdias.columns)-set(customers_data.columns))

Out[67]: ['GEBAEUDETYP_5.0']

In [68]: customers_data['GEBAEUDETYP_5.0']=0

In [69]: customers_data.shape

Out[69]: (144491, 199)

In [70]: customers_scaled=scaler.transform(customers_data)

In [71]: customers_pca=pca.transform(customers_scaled)

In [72]: screen_plot(pca)
```




```
In [80]: customers_pred = model_10.predict(customers_pca)
         print(customers_pred)
```

```
[7 2 5 ..., 7 6 5]
```

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.

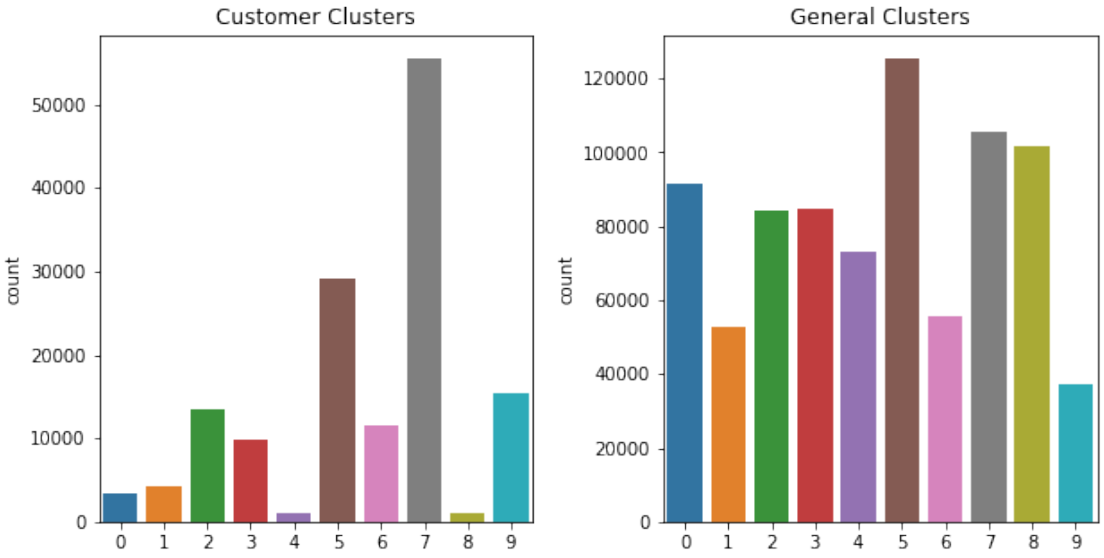
Consider the proportion of persons in each cluster for the general population, and the proportions for the customers. If we think the company's customer base to be universal, then the cluster assignment proportions should be fairly similar between the two. If there are only particular segments of the 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 [81]:
```
# 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(pred, ax=axs[1])
axs[1].set_title('General Clusters')
```

Out[81]:
```
Text(0.5,1,'General Clusters')
```



In [86]:
```
# What kinds of people are part of a cluster that is overrepresented in the
# customer data compared to the general population?
over_data = scaler.inverse_transform(pca.inverse_transform(customers_pca[np.where(custo
over = pd.DataFrame(data=over_data,index=np.array(range(0, over_data.shape[0])), column
over.head(10)
```

Out[86]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	FINANZ_MINIMALIST	FINANZ_SPARER	\
0	4.0	1.0	5.0	1.0	
1	3.0	1.0	5.0	1.0	
2	4.0	1.0	5.0	1.0	
3	3.0	1.0	5.0	1.0	
4	3.0	1.0	5.0	1.0	
5	3.0	1.0	5.0	1.0	
6	4.0	1.0	6.0	1.0	
7	4.0	1.0	5.0	3.0	
8	4.0	1.0	5.0	2.0	
9	2.0	2.0	5.0	2.0	

	FINANZ_VORSORGER	FINANZ_ANLEGER	FINANZ_UNAUFFAELLIGER	FINANZ_HAUSBAUER	\
0	4.0	1.0	2.0	3.0	
1	4.0	2.0	2.0	3.0	
2	4.0	1.0	2.0	3.0	
3	5.0	1.0	3.0	2.0	
4	4.0	1.0	2.0	3.0	
5	5.0	0.0	2.0	3.0	
6	4.0	1.0	2.0	2.0	
7	2.0	3.0	4.0	0.0	
8	4.0	2.0	3.0	0.0	
9	4.0	3.0	2.0	0.0	

	GREEN_AVANTGARDE	HEALTH_TYP	...	wealth_4	life_stage_1	\
0	1.0	1.0	...	0.0	0.0	
1	1.0	2.0	...	-0.0	0.0	
2	1.0	1.0	...	-0.0	0.0	
3	1.0	2.0	...	-0.0	0.0	
4	1.0	1.0	...	0.0	0.0	
5	1.0	1.0	...	-0.0	0.0	
6	1.0	1.0	...	-0.0	0.0	
7	1.0	2.0	...	0.0	0.0	
8	1.0	1.0	...	-0.0	0.0	
9	1.0	2.0	...	-0.0	1.0	

	life_stage_2	life_stage_3	life_stage_4	fine_scale_1	fine_scale_2	\
0	1.0	0.0	0.0	0.0	0.0	
1	1.0	1.0	0.0	0.0	-0.0	

```
2          -0.0          0.0          1.0          0.0          0.0
3          -0.0         -0.0          1.0          0.0         -0.0
4          -0.0          0.0          0.0          0.0         -0.0
5          -0.0          0.0          1.0          0.0         -0.0
6          -0.0         -0.0          1.0          0.0          0.0
7          -0.0          0.0          0.0          0.0         -0.0
8           1.0         -0.0          0.0          0.0         -0.0
9          -0.0         -0.0         -0.0          1.0         -0.0

   fine_scale_3  fine_scale_4  GEBAEUDETYP_5.0
0          -0.0           0.0              0.0
1          -0.0           0.0              0.0
2          -0.0          -0.0              0.0
3           0.0           0.0              0.0
4          -0.0           0.0              0.0
5           0.0           0.0              0.0
6          -0.0          -0.0              0.0
7           0.0          -0.0              0.0
8          -0.0           0.0              0.0
9           0.0           0.0              0.0

[10 rows x 199 columns]
```

In [87]:
```
# What kinds of people are part of a cluster that is underrepresented in the
# customer data compared to the general population?
under_data = scaler.inverse_transform(pca.inverse_transform(customers_pca[np.where(cust
under = pd.DataFrame(data=under_data,index=np.array(range(0, under_data.shape[0])), col
over.head(10)
```

Out[87]:
```
   ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
0                   4.0        1.0                5.0            1.0
1                   3.0        1.0                5.0            1.0
2                   4.0        1.0                5.0            1.0
3                   3.0        1.0                5.0            1.0
4                   3.0        1.0                5.0            1.0
5                   3.0        1.0                5.0            1.0
6                   4.0        1.0                6.0            1.0
7                   4.0        1.0                5.0            3.0
8                   4.0        1.0                5.0            2.0
9                   2.0        2.0                5.0            2.0

   FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER  \
0               4.0             1.0                    2.0               3.0
1               4.0             2.0                    2.0               3.0
2               4.0             1.0                    2.0               3.0
3               5.0             1.0                    3.0               2.0
4               4.0             1.0                    2.0               3.0
5               5.0             0.0                    2.0               3.0
```

```
6                4.0             1.0                   2.0              2.0
7                2.0             3.0                   4.0              0.0
8                4.0             2.0                   3.0              0.0
9                4.0             3.0                   2.0              0.0

   GREEN_AVANTGARDE  HEALTH_TYP       ...         wealth_4  life_stage_1  \
0               1.0         1.0       ...              0.0           0.0
1               1.0         2.0       ...             -0.0           0.0
2               1.0         1.0       ...             -0.0           0.0
3               1.0         2.0       ...             -0.0           0.0
4               1.0         1.0       ...              0.0           0.0
5               1.0         1.0       ...             -0.0           0.0
6               1.0         1.0       ...             -0.0           0.0
7               1.0         2.0       ...              0.0           0.0
8               1.0         1.0       ...             -0.0           0.0
9               1.0         2.0       ...             -0.0           1.0

   life_stage_2  life_stage_3  life_stage_4  fine_scale_1  fine_scale_2  \
0           1.0           0.0           0.0           0.0           0.0
1           1.0           1.0           0.0           0.0          -0.0
2          -0.0           0.0           1.0           0.0           0.0
3          -0.0          -0.0           1.0           0.0          -0.0
4          -0.0           0.0           0.0           0.0          -0.0
5          -0.0           0.0           1.0           0.0          -0.0
6          -0.0          -0.0           1.0           0.0           0.0
7          -0.0           0.0           0.0           0.0          -0.0
8           1.0          -0.0           0.0           0.0          -0.0
9          -0.0          -0.0          -0.0           1.0          -0.0

   fine_scale_3  fine_scale_4  GEBAEUDETYP_5.0
0          -0.0           0.0              0.0
1          -0.0           0.0              0.0
2          -0.0          -0.0              0.0
3           0.0           0.0              0.0
4          -0.0           0.0              0.0
5           0.0           0.0              0.0
6          -0.0          -0.0              0.0
7           0.0          -0.0              0.0
8          -0.0           0.0              0.0
9           0.0           0.0              0.0

[10 rows x 199 columns]
```

1.3.5 Discussion 3.3: Compare Customer Data to Demographics Data

Can we describe segments of the population that are relatively popular with the mail-order company, or relatively unpopular with the company?

Looking at the chart, we can easily find that sets 2, 5, 6, 7 and 9 are most likely customer

segments.

> 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 [ ]: