Identify_Customer_Segments

July 27, 2019

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

        # magic word for producing visualizations in notebook
        %matplotlib inline
```

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).

        azdias.head()
        #feat_info.shape
```

```
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.shape

Out[4]: (891221, 85)

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]: feat_info.describe()

Out[6]:            attribute information_level     type missing_or_unknown
        count             85                85       85                 85
        unique            85                 9        5                  9
        top     FINANZ_SPARER           person  ordinal               [-1]
        freq               1                43       49                 26
```

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]: # Identify missing or unknown data values and convert them to NaNs.
        print('Number of the sum of missing values is :', azdias.isnull().sum().sum())

Number of the sum of missing values is : 4896838
```

```
In [8]: azdias_null = azdias.isnull().sum()[azdias.isnull().sum() != 0]
        azdias_null_dict = {'Count': azdias_null.values}
        azdias_null = pd.DataFrame(azdias_null_dict, index = azdias_null.index)
        azdias_null.sort_values(by='Count', ascending=False, inplace=True)
        azdias_null.head()

Out[8]:                  Count
        KK_KUNDENTYP    584612
        KBA05_BAUMAX    133324
        KBA05_ANTG4     133324
        KBA05_ANTG1     133324
        KBA05_ANTG2     133324
```

```
In [9]: azdias_null.shape

Out[9]: (53, 1)
```

There 53 columns which contains missing values.

```
In [10]: for i in range(len(feat_info)):
             missing = feat_info.iloc[i]['missing_or_unknown']
             missing = missing.replace('[','')
             missing = missing.replace(']','')
             missing = missing.split(sep = ',')
             missing = [int(x) if (x != 'X' and x !='XX' and x !='') else x for x in missing]
             if missing != ['']:
                 azdias = azdias.replace({feat_info.iloc[i]['attribute']: missing}, np.nan)
```

```
In [11]: #for col in azdias.columns:
         #    azdias[col] = azdias[col].replace(missing.loc[col][0], np.nan)
```

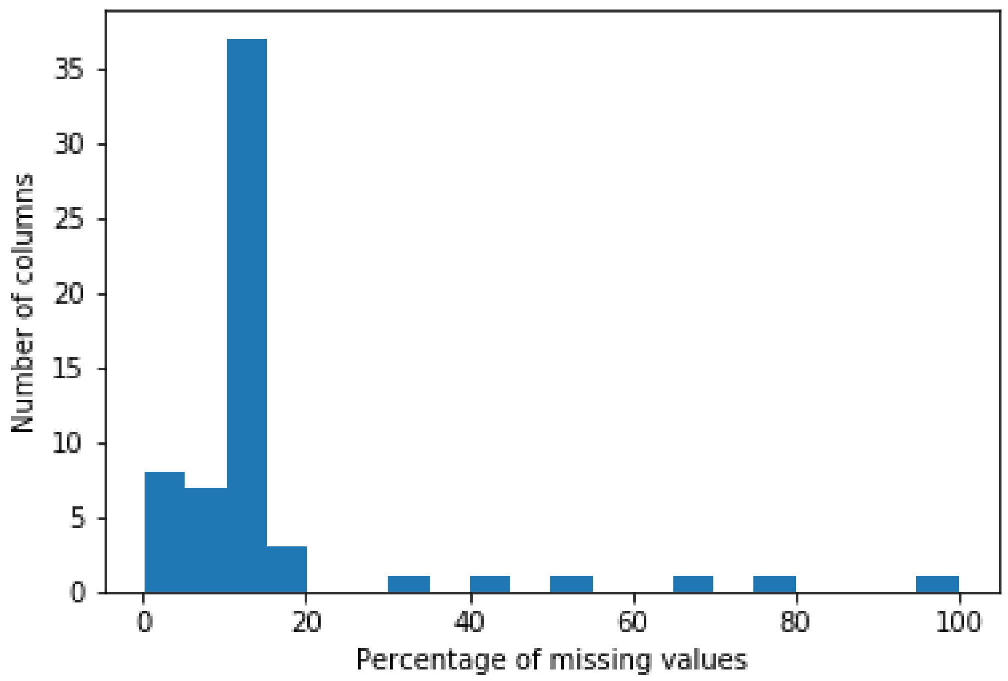
```
In [12]: print(azdias.isnull().sum().sum())
```

```
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 [13]: # Perform an assessment of how much missing data there is in each column of the
         # dataset.
         null = azdias.isnull().sum()[azdias.isnull().sum() != 0]
         null_dict_new = {'Count': null.values, '%of_null': np.round(null.values * 100 /891211,2
         null_dict = pd.DataFrame(null_dict_new, index = null.index)
         null_dict.sort_values('Count').head()

         plt.hist(null_dict['%of_null'], 20)
         plt.xlabel('Percentage of missing values')
         plt.ylabel('Number of columns')
         plt.show()
```



```
In [14]: null_dict.describe()

Out[14]:                 Count    %of_null
         count      61.000000  61.000000
         mean   137277.524590  15.403115
         std    157099.496443  17.628423
         min      2881.000000   0.320000
         25%     93148.000000  10.450000
         50%     99352.000000  11.150000
         75%    116515.000000  13.070000
         max    889061.000000  99.760000

In [15]: # Investigate patterns in the amount of missing data in each column.
         dict2 = {'count':azdias.isnull().sum().values,
                  '%of_null':np.round(azdias.isnull().sum().values *100/891221,2)}

         dict2 = pd.DataFrame(dict2, azdias.isnull().sum().index)
         dict2.head()

Out[15]:                       count  %of_null
         AGER_TYP             685843     76.96
         ALTERSKATEGORIE_GROB   2881      0.32
         ANREDE_KZ                 0      0.00
         CJT_GESAMTTYP          4854      0.54
         FINANZ_MINIMALIST         0      0.00
```

```
In [16]: # Remove the outlier columns from the dataset. (You'll perform other data
         # engineering tasks such as re-encoding and imputation later.)
         Miss_greater25 = dict2[dict2['%of_null'] > 25]
         Miss_greater25
```

```
Out[16]:                 count  %of_null
         AGER_TYP       685843     76.96
         GEBURTSJAHR    392318     44.02
         TITEL_KZ       889061     99.76
         ALTER_HH       310267     34.81
         KK_KUNDENTYP   584612     65.60
         KBA05_BAUMAX   476524     53.47
```

```
In [17]: drop_columns = ['AGER_TYP','GEBURTSJAHR','TITEL_KZ','ALTER_HH','KK_KUNDENTYP','KBA05_BA
         azdias = azdias.drop(drop_columns, axis = 1)
```

Discussion 1.1.2: Assess Missing Data in Each Column There are six columns with missing value over 30%. I drop these six columns because they may not be providing meaningful value to later analysis. The highest two percentage of missing values is for TITEL_KZ 99.76% and AGER_TYP which is 76.96%. There are 61 columns with missing valueas and the average missing value percentage for each column is 17.28%.

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

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

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

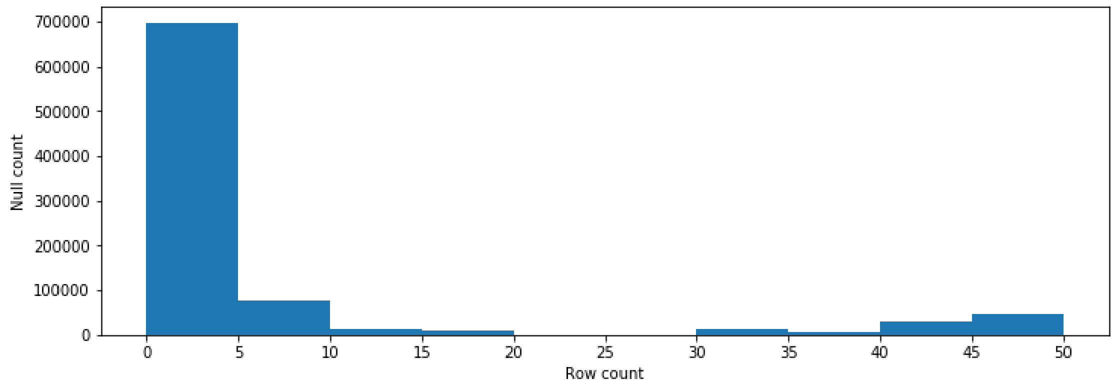
```
In [18]: # How much data is missing in each row of the dataset?
         row = azdias.isnull().sum(axis = 1)
         row.describe()
```

```
Out[18]: count    891221.000000
         mean          5.649894
```

```
std          13.234687
min           0.000000
25%           0.000000
50%           0.000000
75%           3.000000
max          49.000000
dtype: float64
```

```
In [19]: # Write code to divide the data into two subsets based on the number of missing
         # values in each row.
         plt.figure(figsize = (12,4))
         plt.hist(row, bins = np.arange(0,55,5))
         plt.ylabel('Null count')
         plt.xlabel('Row count')
         plt.xticks(np.arange(0,55,5))
```

```
Out[19]: ([<matplotlib.axis.XTick at 0x7fbdf4dc4278>,
           <matplotlib.axis.XTick at 0x7fbdf2da0128>,
           <matplotlib.axis.XTick at 0x7fbdf2d9b6d8>,
           <matplotlib.axis.XTick at 0x7fbdf2d7af98>,
           <matplotlib.axis.XTick at 0x7fbdf2d83a58>,
           <matplotlib.axis.XTick at 0x7fbdf2d838d0>,
           <matplotlib.axis.XTick at 0x7fbdf4ed9828>,
           <matplotlib.axis.XTick at 0x7fbdf4ed9978>,
           <matplotlib.axis.XTick at 0x7fbdf4e00e10>,
           <matplotlib.axis.XTick at 0x7fbdf4e00278>,
           <matplotlib.axis.XTick at 0x7fbdf4ed8eb8>],
          <a list of 11 Text xticklabel objects>)
```

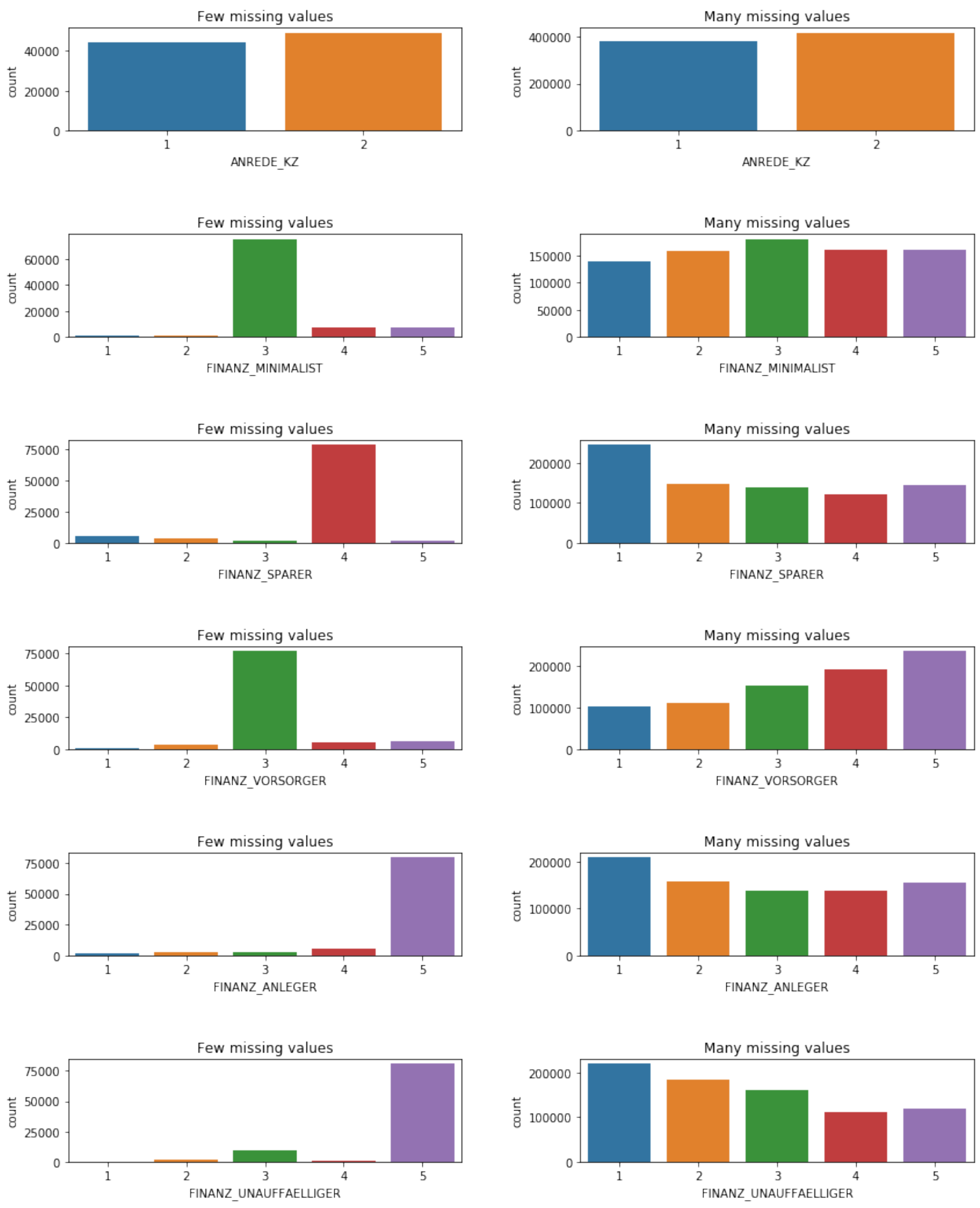


Compare the distribution of values for at least five columns where there are no or few missing values, between the two subsets.

```
In [20]: rows_greater30 = azdias[azdias.isnull().sum(axis=1) >30]
         rows_less30 = azdias[azdias.isnull().sum(axis=1) <= 30]
         print('There are', rows_greater30.shape[0],'and around', np.round(rows_greater30.shape[
There are 93154 and around 10.45 % of all observations.
```

```
In [21]: rows_less30.head()

Out[21]:    ALTERSKATEGORIE_GROB  ANREDE_KZ  CJT_GESAMTTYP  FINANZ_MINIMALIST  \
         1                   1.0          2            5.0                  1
         2                   3.0          2            3.0                  1
         3                   4.0          2            2.0                  4
         4                   3.0          1            5.0                  4
         5                   1.0          2            2.0                  3

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

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

            PLZ8_ANTG4  PLZ8_BAUMAX  PLZ8_HHZ  PLZ8_GBZ  ARBEIT  ORTSGR_KLS9  RELAT_AB
         1         1.0          1.0       5.0       4.0     3.0          5.0       4.0
         2         0.0          1.0       4.0       4.0     3.0          5.0       2.0
         3         0.0          1.0       3.0       4.0     2.0          3.0       3.0
         4         1.0          2.0       3.0       3.0     4.0          6.0       5.0
         5         1.0          1.0       5.0       5.0     2.0          3.0       3.0

         [5 rows x 79 columns]

In [22]: no_missing_col = dict2[dict2['count'] == 0].index.tolist()
         no_missing_5 = no_missing_col[:6]
         no_missing_5

Out[22]: ['ANREDE_KZ',
          'FINANZ_MINIMALIST',
          'FINANZ_SPARER',
          'FINANZ_VORSORGER',
          'FINANZ_ANLEGER',
          'FINANZ_UNAUFFAELLIGER']

In [23]: figure, axs = plt.subplots(len(no_missing_5), ncols=2, figsize = (14,18))
         figure.subplots_adjust(hspace = 1, wspace=.3)
         for i in range(len(no_missing_5)):
```

```
sns.countplot(rows_greater30[no_missing_5[i]], ax=axs[i][0])
axs[i][0].set_title('Few missing values')
sns.countplot(rows_less30[no_missing_5[i]], ax=axs[i][1])
axs[i][1].set_title('Many missing values')
```



Discussion 1.1.3: Assess Missing Data in Each Row There are 93186 observations that are missing more than 30 columns out of 83 columns. The bar chart compare the difference between rows with many missing values and no missing values among no missing values in columns. It shows that the distribution between these two datasets are similar for gender and quite difference for other features.

1.1.2 Step 1.2: Select and Re-Encode Features

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

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

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

```
In [24]: # How many features are there of each data type?
         feat_info.type.value_counts()
```

```
Out[24]: ordinal        49
         categorical    21
         mixed           7
         numeric         7
         interval        1
         Name: type, dtype: int64
```

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

```
In [25]: cate_col = feat_info.loc[feat_info['type'] == 'categorical', 'attribute'].values
         bi_col = []
         multi_col = []
```

```
        for col in cate_col:
            try:
                azdias[col]
            except:
                continue
            if azdias[col].nunique() > 2:
                multi_col.append(col)
            else:
                bi_col.append(col)
```

```
In [26]: # Re-encode categorical variable(s) to be kept in the analysis.
         for c in bi_col:
             print(azdias[c].value_counts())
```

```
2    465305
1    425916
Name: ANREDE_KZ, dtype: int64
0    715996
1    175225
Name: GREEN_AVANTGARDE, dtype: int64
0.0    810834
1.0      6888
Name: SOHO_KZ, dtype: int64
2.0    398722
1.0    381303
Name: VERS_TYP, dtype: int64
W    629528
O    168545
Name: OST_WEST_KZ, dtype: int64
```

```
In [27]: azdias['ANREDE_KZ'].replace([2,1], [1,0], inplace=True)
         azdias['VERS_TYP'].replace([2.0,1.0], [1,0], inplace=True)
         azdias['OST_WEST_KZ'].replace(['W','O'], [1,0], inplace=True)
```

```
In [28]: azdias['OST_WEST_KZ'].value_counts()
```

```
Out[28]: 1.0    629528
         0.0    168545
         Name: OST_WEST_KZ, dtype: int64
```

```
In [29]: multi_col
```

```
Out[29]: ['CJT_GESAMTTYP',
          'FINANZTYP',
          'GFK_URLAUBERTYP',
          'LP_FAMILIE_FEIN',
          'LP_FAMILIE_GROB',
          'LP_STATUS_FEIN',
           'LP_STATUS_GROB',
           'NATIONALITAET_KZ',
           'SHOPPER_TYP',
           'ZABEOTYP',
           'GEBAEUDETYP',
           'CAMEO_DEUG_2015',
           'CAMEO_DEU_2015']
```

```
In [30]: azdias = pd.get_dummies(azdias, columns=multi_col)
```

```
In [31]: azdias.shape
```

```
Out[31]: (891221, 194)
```

Discussion 1.2.1: Re-Encode Categorical Features For binary data, I encoded them into 0 and 1. For multi level categorical values data i use pd.get dumies to encode them.

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 [32]: # Investigate "PRAEGENDE_JUGENDJAHRE" and engineer two new variables.
         azdias[['PRAEGENDE_JUGENDJAHRE']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891221 entries, 0 to 891220
Data columns (total 1 columns):
PRAEGENDE_JUGENDJAHRE    783057 non-null float64
dtypes: float64(1)
memory usage: 6.8 MB
```

```
In [33]: dict2 = {0: [1, 2], 1: [3, 4], 2: [5, 6, 7], 3: [8, 9], 4: [10, 11, 12, 13], 5:[14, 15]

         def decades(x):
             try:
                 for key, value in dict2.items():
                     if x in value:
```

```
                    return key
            except ValueError:
                return np.nan

        mainstream = [1, 3, 5, 8, 10, 12, 14]

        def movement(x):
            try:
                if x in mainstream:
                    return 0
                else:
                    return 1
            except ValueError:
                return np.nan
```

```
In [34]: azdias['PRAEGENDE_JUGENDJAHRE_Generation'] = azdias['PRAEGENDE_JUGENDJAHRE'].apply(deca
         azdias['PRAEGENDE_JUGENDJAHRE_movement'] = azdias['PRAEGENDE_JUGENDJAHRE'].apply(moveme
```

```
In [35]: azdias[['PRAEGENDE_JUGENDJAHRE_Generation']].head()
```

```
Out[35]:    PRAEGENDE_JUGENDJAHRE_Generation
         0                               NaN
         1                               5.0
         2                               5.0
         3                               3.0
         4                               3.0
```

```
In [36]: azdias[['PRAEGENDE_JUGENDJAHRE_movement']].head()
```

```
Out[36]:    PRAEGENDE_JUGENDJAHRE_movement
         0                               1
         1                               0
         2                               1
         3                               0
         4                               0
```

```
In [37]: # Investigate "CAMEO_INTL_2015" and engineer two new variables.
         azdias[['CAMEO_INTL_2015']].head()
```

```
Out[37]:   CAMEO_INTL_2015
         0             NaN
         1              51
         2              24
         3              12
         4              43
```

```
In [38]: def map_wealth(x):
             if pd.isnull(x):
                 return np.nan
```

```
            else:
                return int(str(x)[0])
        def map_lifestage(x):
            if pd.isnull(x):
                return np.nan
            else:
                return int(str(x)[1])
```

```
In [39]: azdias['CAMEO_INTL_2015_wealth'] = azdias['CAMEO_INTL_2015'].apply(map_wealth)
         azdias['CAMEO_INTL_2015_lifestage'] = azdias['CAMEO_INTL_2015'].apply(map_lifestage)
```

```
In [40]: azdias['CAMEO_INTL_2015_lifestage'].isnull().sum()
```

```
Out[40]: 99352
```

Discussion 1.2.2: Engineer Mixed-Type Features I kept both PRAEGENDE_JUGENDJAHRE and CAMEO_INTL_2015 columns. For each column I created two features column through apply functions to map their values.

For PRAEGENDE_JUGENDJAHRE I created two new columns which are for decades and movement. For CAMEO_INTL_2015 I created two columns, which are for lifestage and wealth.

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 [41]: # 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.)

         azdias.drop(['PRAEGENDE_JUGENDJAHRE', 'CAMEO_INTL_2015'], axis = 1, inplace = True)
```

```
In [42]: # Do whatever you need to in order to ensure that the dataframe only contains
         # the columns that should be passed to the algorithm functions.

         np.unique(azdias.dtypes.values)
```

```
Out[42]: array([dtype('uint8'), dtype('int64'), dtype('float64')], dtype=object)
```

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 [43]: def clean_data(df):
             """
             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 = feat_info.iloc[i]['missing_or_unknown']
                 missing = missing.replace('[','')
                 missing = missing.replace(']','')
                 missing = missing.split(sep = ',')
                 missing = [int(x) if (x != 'X' and x !='XX' and x !='') else x for x in missing
                 if missing != ['']:
                     df = df.replace({feat_info.iloc[i]['attribute']: missing}, np.nan)

             # remove selected columns and rows, ...
             drop_columns = ['AGER_TYP','GEBURTSJAHR','TITEL_KZ','ALTER_HH','KK_KUNDENTYP','KBAO
             df = df.drop(drop_columns, axis = 1)
             df = df[df.isnull().sum(axis=1) <= 25]
             # select, re-encode, and engineer column values.
             cate_col = feat_info.loc[feat_info['type'] == 'categorical', 'attribute'].values
             bi_col = []
             multi_col = []
             for col in cate_col:
                 try:
                     df[col]
                 except:
                     continue
                 if df[col].nunique() > 2:
                     multi_col.append(col)
                 else:
                     bi_col.append(col)
             df['ANREDE_KZ'].replace([2,1], [1,0], inplace=True)
             df['VERS_TYP'].replace([2.0,1.0], [1,0], inplace=True)
             df['OST_WEST_KZ'].replace(['W','O'], [1,0], inplace=True)

             df = pd.get_dummies(df, columns=multi_col)
```

```
dict2 = {0: [1, 2], 1: [3, 4], 2: [5, 6, 7], 3: [8, 9], 4: [10, 11, 12, 13], 5:[14,

def decades(x):
    try:
        for key, value in dict2.items():
            if x in value:
                return key
    except ValueError:
        return np.nan

mainstream = [1, 3, 5, 8, 10, 12, 14]

def movement(x):
    try:
        if x in mainstream:
            return 0
        else:
            return 1
    except ValueError:
        return np.nan
df['PRAEGENDE_JUGENDJAHRE_Generation'] = df['PRAEGENDE_JUGENDJAHRE'].apply(decades)
df['PRAEGENDE_JUGENDJAHRE_movement'] = df['PRAEGENDE_JUGENDJAHRE'].apply(movement)

def map_wealth(x):
    if pd.isnull(x):
        return np.nan
    else:
        return int(str(x)[0])

def map_lifestage(x):
    if pd.isnull(x):
        return np.nan
    else:
        return int(str(x)[1])

df['CAMEO_INTL_2015_wealth'] = df['CAMEO_INTL_2015'].apply(map_wealth)
df['CAMEO_INTL_2015_lifestage'] = df['CAMEO_INTL_2015'].apply(map_lifestage)
df.drop(['PRAEGENDE_JUGENDJAHRE', 'CAMEO_INTL_2015'], axis = 1, inplace = True)
# Return the cleaned dataframe.
return df
```

1.2 Step 2: Feature Transformation

1.2.1 Step 2.1: Apply Feature Scaling

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

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

```
In [44]: # If you've not yet cleaned the dataset of all NaN values, then investigate and
         # do that now.
         clean = azdias.copy()
         clean.isna().sum()
```

```
Out[44]: ALTERSKATEGORIE_GROB                2881
         ANREDE_KZ                              0
         FINANZ_MINIMALIST                      0
         FINANZ_SPARER                          0
         FINANZ_VORSORGER                       0
         FINANZ_ANLEGER                         0
         FINANZ_UNAUFFAELLIGER                  0
         FINANZ_HAUSBAUER                       0
         GREEN_AVANTGARDE                       0
         HEALTH_TYP                        111196
         LP_LEBENSPHASE_FEIN                97632
         LP_LEBENSPHASE_GROB                94572
         RETOURTYP_BK_S                      4854
         SEMIO_SOZ                              0
         SEMIO_FAM                              0
         SEMIO_REL                              0
         SEMIO_MAT                              0
         SEMIO_VERT                             0
```

```
SEMIO_LUST                                0
SEMIO_ERL                                 0
SEMIO_KULT                                0
SEMIO_RAT                                 0
SEMIO_KRIT                                0
SEMIO_DOM                                 0
SEMIO_KAEM                                0
SEMIO_PFLICHT                             0
SEMIO_TRADV                               0
SOHO_KZ                               73499
VERS_TYP                             111196
ANZ_PERSONEN                          73499
                                        ...
CAMEO_DEU_2015_5A                         0
CAMEO_DEU_2015_5B                         0
CAMEO_DEU_2015_5C                         0
CAMEO_DEU_2015_5D                         0
CAMEO_DEU_2015_5E                         0
CAMEO_DEU_2015_5F                         0
CAMEO_DEU_2015_6A                         0
CAMEO_DEU_2015_6B                         0
CAMEO_DEU_2015_6C                         0
CAMEO_DEU_2015_6D                         0
CAMEO_DEU_2015_6E                         0
CAMEO_DEU_2015_6F                         0
CAMEO_DEU_2015_7A                         0
CAMEO_DEU_2015_7B                         0
CAMEO_DEU_2015_7C                         0
CAMEO_DEU_2015_7D                         0
CAMEO_DEU_2015_7E                         0
CAMEO_DEU_2015_8A                         0
CAMEO_DEU_2015_8B                         0
CAMEO_DEU_2015_8C                         0
CAMEO_DEU_2015_8D                         0
CAMEO_DEU_2015_9A                         0
CAMEO_DEU_2015_9B                         0
CAMEO_DEU_2015_9C                         0
CAMEO_DEU_2015_9D                         0
CAMEO_DEU_2015_9E                         0
PRAEGENDE_JUGENDJAHRE_Generation     108164
PRAEGENDE_JUGENDJAHRE_movement            0
CAMEO_INTL_2015_wealth                99352
CAMEO_INTL_2015_lifestage             99352
Length: 196, dtype: int64
```

```
In [45]: from sklearn.preprocessing import Imputer, StandardScaler
```

```
In [46]: # Apply feature scaling to the general population demographics data.
```

```
         fill_na = Imputer(strategy='most_frequent')
         clean_fill = pd.DataFrame(fill_na.fit_transform(clean))

In [47]: clean_fill.columns =clean.columns
         clean_fill.index = clean.index

In [48]: clean_fill.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891221 entries, 0 to 891220
Columns: 196 entries, ALTERSKATEGORIE_GROB to CAMEO_INTL_2015_lifestage
dtypes: float64(196)
memory usage: 1.3 GB


In [49]: scaler = StandardScaler()
         clean_scaled = scaler.fit_transform(clean_fill)

In [50]: clean_scaled = pd.DataFrame(clean_scaled)
         clean_scaled.head()

Out[50]:         0         1         2         3         4         5         6    \
         0 -0.751680 -1.045218 -0.056416  0.804890 -0.303378  1.285741  1.429871
         1 -1.743339  0.956738 -1.570358  1.487601 -1.059731  1.285741  0.757254
         2  0.239980  0.956738 -1.570358  0.804890 -1.816084 -0.675554  0.084637
         3  1.231640  0.956738  0.700556 -0.560532  1.209329 -0.675554 -1.260597
         4  0.239980 -1.045218  0.700556  0.122179  0.452976 -1.329319  0.084637

                 7         8         9      ...          186     187       188  \
         0 -0.055511 -0.494701  0.938197    ...    -0.141838 -0.1536 -0.179023
         1  1.422415 -0.494701  0.938197    ...    -0.141838 -0.1536 -0.179023
         2  1.422415  2.021423  0.938197    ...    -0.141838 -0.1536 -0.179023
         3 -0.794475 -0.494701 -0.385397    ...    -0.141838 -0.1536 -0.179023
         4 -0.794475 -0.494701  0.938197    ...    -0.141838 -0.1536 -0.179023

                189       190       191       192       193       194       195
         0 -0.16984 -0.182061 -0.084907  0.994991  1.464537  1.039938 -1.095866
         1 -0.16984 -0.182061 -0.084907  0.994991 -0.682810  1.039938 -1.095866
         2 -0.16984 -0.182061 -0.084907  0.994991  1.464537 -0.980453  0.879581
         3 -0.16984 -0.182061 -0.084907 -0.363493 -0.682810 -1.653917 -0.437384
         4 -0.16984 -0.182061 -0.084907 -0.363493 -0.682810  0.366474  0.221098

         [5 rows x 196 columns]

In [51]: clean_scaled.columns = list(clean_fill)
         clean_scaled.head()

Out[51]:    ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
         0             -0.751680  -1.045218          -0.056416       0.804890
```

```
1            -1.743339   0.956738          -1.570358       1.487601
2             0.239980   0.956738          -1.570358       0.804890
3             1.231640   0.956738           0.700556      -0.560532
4             0.239980  -1.045218           0.700556       0.122179

   FINANZ_VORSORGER  FINANZ_ANLEGER  FINANZ_UNAUFFAELLIGER  FINANZ_HAUSBAUER  \
0         -0.303378        1.285741               1.429871         -0.055511
1         -1.059731        1.285741               0.757254          1.422415
2         -1.816084       -0.675554               0.084637          1.422415
3          1.209329       -0.675554              -1.260597         -0.794475
4          0.452976       -1.329319               0.084637         -0.794475

   GREEN_AVANTGARDE  HEALTH_TYP            ...              CAMEO_DEU_2015_8D  \
0         -0.494701    0.938197            ...                      -0.141838
1         -0.494701    0.938197            ...                      -0.141838
2          2.021423    0.938197            ...                      -0.141838
3         -0.494701   -0.385397            ...                      -0.141838
4         -0.494701    0.938197            ...                      -0.141838

   CAMEO_DEU_2015_9A  CAMEO_DEU_2015_9B  CAMEO_DEU_2015_9C  CAMEO_DEU_2015_9D  \
0            -0.1536          -0.179023           -0.16984          -0.182061
1            -0.1536          -0.179023           -0.16984          -0.182061
2            -0.1536          -0.179023           -0.16984          -0.182061
3            -0.1536          -0.179023           -0.16984          -0.182061
4            -0.1536          -0.179023           -0.16984          -0.182061

   CAMEO_DEU_2015_9E  PRAEGENDE_JUGENDJAHRE_Generation  \
0          -0.084907                          0.994991
1          -0.084907                          0.994991
2          -0.084907                          0.994991
3          -0.084907                         -0.363493
4          -0.084907                         -0.363493

   PRAEGENDE_JUGENDJAHRE_movement  CAMEO_INTL_2015_wealth  \
0                        1.464537                1.039938
1                       -0.682810                1.039938
2                        1.464537               -0.980453
3                       -0.682810               -1.653917
4                       -0.682810                0.366474

   CAMEO_INTL_2015_lifestage
0                  -1.095866
1                  -1.095866
2                   0.879581
3                  -0.437384
4                   0.221098

[5 rows x 196 columns]
```

1.2.2 Discussion 2.1: Apply Feature Scaling

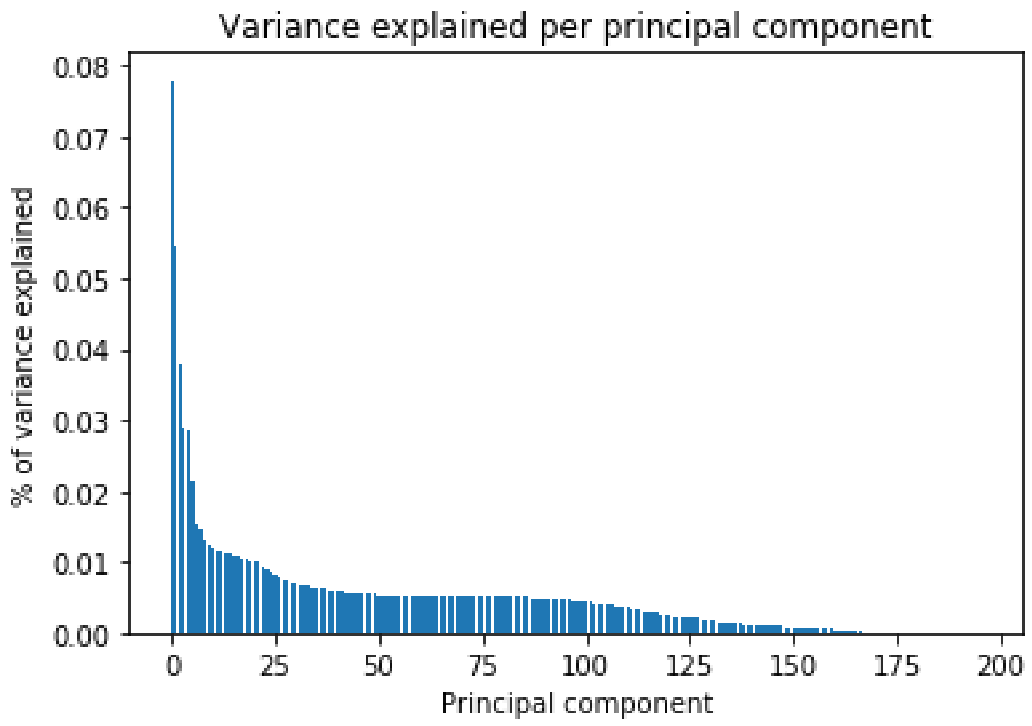
I replaced the missing value with the most frequent values and all features are scaled according to the standardscaler.

1.2.3 Step 2.2: Perform Dimensionality Reduction

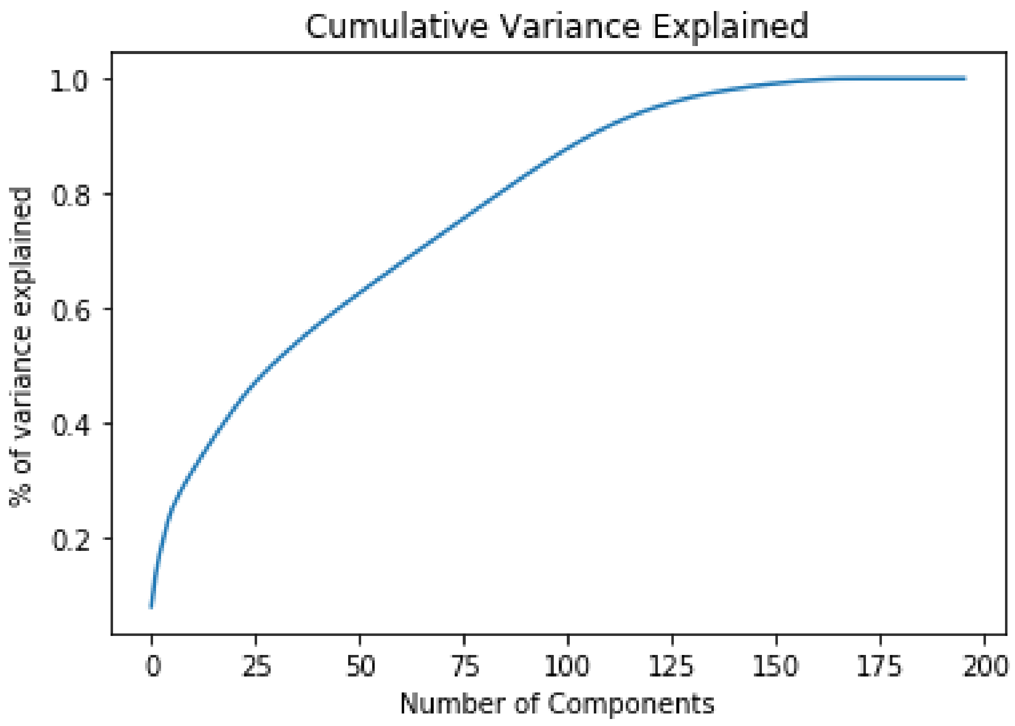
On your scaled data, you are now ready to apply dimensionality reduction techniques.

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

```
In [52]: # Apply PCA to the data.
         from sklearn.decomposition import PCA
         pca = PCA()
         pca.fit(clean_scaled)

Out[52]: PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
           svd_solver='auto', tol=0.0, whiten=False)

In [53]: # Investigate the variance accounted for by each principal component.
         plt.bar(range(len(pca.explained_variance_ratio_)), pca.explained_variance_ratio_)
         plt.title("Variance explained per principal component")
         plt.xlabel("Principal component")
         plt.ylabel("% of variance explained")
         plt.show()
```



```
In [54]: plt.plot(range(len(pca.explained_variance_ratio_)),np.cumsum(pca.explained_variance_rat
         plt.title("Cumulative Variance Explained")
         plt.xlabel("Number of Components")
         plt.ylabel("% of variance explained")
         plt.show()
```



```
In [55]: pca_40 = PCA(n_components=40)
         azdias_pca = pca_40.fit_transform(clean_scaled)
```

1.2.4 Discussion 2.2: Perform Dimensionality Reduction

PCA was applied to a set of components that was half the number of initial features. After looking at the variance explained by each component,I chose to keep 40 principal components, because it explained around 60% of the variance from the dataset.

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 [56]: # 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_w(pca, i):
             df = pd.DataFrame(pca.components_)
             df.columns=list(clean_scaled.columns)
             w = df.iloc[i].sort_values(ascending=False)
             return w
```

```
In [57]: # Map weights for the second principal component to corresponding feature names
         # and then print the linked values, sorted by weight.
         pca_w_0 = pca_w(pca_40, 0)
         print (pca_w_0)
```

```
PLZ8_ANTG3                          0.182685
CAMEO_INTL_2015_wealth              0.180481
LP_STATUS_GROB_1.0                  0.177280
PLZ8_BAUMAX                         0.157360
EWDICHTE                            0.156604
PLZ8_ANTG4                          0.153060
ORTSGR_KLS9                         0.152182
HH_EINKOMMEN_SCORE                  0.143627
FINANZ_HAUSBAUER                    0.141251
PLZ8_ANTG2                          0.126162
FINANZ_SPARER                       0.119656
KBA05_ANTG4                         0.118081
ARBEIT                              0.117924
LP_STATUS_FEIN_1.0                  0.114471
LP_STATUS_FEIN_2.0                  0.108161
ANZ_HAUSHALTE_AKTIV                 0.105053
RELAT_AB                            0.104897
CAMEO_DEUG_2015_9                   0.104183
KBA05_ANTG3                         0.103452
FINANZTYP_1                         0.102667
SEMIO_REL                           0.087085
SEMIO_PFLICHT                       0.086810
LP_FAMILIE_FEIN_1.0                 0.086668
LP_FAMILIE_GROB_1.0                 0.086668
```

```
PRAEGENDE_JUGENDJAHRE_Generation    0.085680
CAMEO_DEUG_2015_8                   0.082675
ZABEOTYP_5                          0.071650
SEMIO_RAT                           0.070250
FINANZ_UNAUFFAELLIGER               0.067499
FINANZ_ANLEGER                      0.066629
                                      ...
CAMEO_DEUG_2015_3                  -0.068035
WOHNLAGE                           -0.069692
CAMEO_DEUG_2015_4                  -0.075764
NATIONALITAET_KZ_1.0               -0.077327
FINANZTYP_2                        -0.078624
BALLRAUM                           -0.081350
LP_FAMILIE_GROB_5.0                -0.081406
ALTERSKATEGORIE_GROB               -0.084392
FINANZ_VORSORGER                   -0.086588
CAMEO_DEUG_2015_2                  -0.088975
GEBAEUDETYP_RASTER                 -0.091995
GEBAEUDETYP_1.0                    -0.095940
ZABEOTYP_1                         -0.099203
ANZ_PERSONEN                       -0.099809
GREEN_AVANTGARDE                   -0.102840
LP_STATUS_FEIN_10.0                -0.111184
LP_STATUS_GROB_5.0                 -0.111184
LP_STATUS_FEIN_9.0                 -0.112425
LP_STATUS_GROB_4.0                 -0.113779
CAMEO_INTL_2015_lifestage          -0.113969
INNENSTADT                         -0.123016
PLZ8_GBZ                           -0.132921
LP_LEBENSPHASE_GROB                -0.133416
KONSUMNAEHE                        -0.140552
LP_LEBENSPHASE_FEIN                -0.145992
KBA05_GBZ                          -0.176844
KBA05_ANTG1                        -0.178364
PLZ8_ANTG1                         -0.182247
MOBI_REGIO                         -0.183851
FINANZ_MINIMALIST                  -0.186181
Name: 0, Length: 196, dtype: float64
```

```
In [58]: # Map weights for the third principal component to corresponding feature names
         # and then print the linked values, sorted by weight.
         pca_w_1 = pca_w(pca_40, 1)
         print (pca_w_1)
```

```
ALTERSKATEGORIE_GROB                0.226923
FINANZ_VORSORGER                    0.218186
SEMIO_ERL                           0.175144
```

```
RETOURTYP_BK_S                        0.155847
SEMIO_LUST                            0.147613
ZABEOTYP_3                            0.144720
LP_STATUS_FEIN_1.0                    0.111962
FINANZTYP_5                           0.111591
CJT_GESAMTTYP_2.0                     0.107990
NATIONALITAET_KZ_1.0                  0.102568
W_KEIT_KIND_HH                        0.096624
PLZ8_ANTG4                            0.094841
FINANZTYP_2                           0.094713
FINANZ_HAUSBAUER                      0.090998
PLZ8_BAUMAX                           0.086161
FINANZ_MINIMALIST                     0.084523
FINANZTYP_6                           0.082626
LP_FAMILIE_GROB_1.0                   0.080525
LP_FAMILIE_FEIN_1.0                   0.080525
SHOPPER_TYP_3.0                       0.078670
HH_EINKOMMEN_SCORE                    0.076203
ORTSGR_KLS9                           0.075496
CJT_GESAMTTYP_1.0                     0.075058
KBA05_ANTG4                           0.070157
ANZ_HAUSHALTE_AKTIV                   0.065485
GFK_URLAUBERTYP_4.0                   0.064221
PLZ8_ANTG3                            0.063425
LP_STATUS_FEIN_3.0                    0.054356
CAMEO_DEUG_2015_8                     0.053222
RELAT_AB                              0.052856
                                        ...
PLZ8_GBZ                             -0.050425
ANZ_PERSONEN                         -0.053757
INNENSTADT                           -0.064561
LP_FAMILIE_GROB_4.0                  -0.065379
KBA05_GBZ                            -0.066187
BALLRAUM                             -0.066917
GFK_URLAUBERTYP_9.0                  -0.067515
PLZ8_ANTG1                           -0.067515
FINANZTYP_3                          -0.070021
KBA13_ANZAHL_PKW                     -0.072381
CJT_GESAMTTYP_6.0                    -0.076324
ZABEOTYP_5                           -0.077563
HEALTH_TYP                           -0.078003
LP_STATUS_FEIN_2.0                   -0.081396
ZABEOTYP_4                           -0.086481
LP_STATUS_FEIN_5.0                   -0.107014
FINANZTYP_1                          -0.113051
FINANZTYP_4                          -0.119972
SEMIO_KULT                           -0.125201
SEMIO_FAM                            -0.131172
```

```
SEMIO_MAT                              -0.131542
ONLINE_AFFINITAET                      -0.150890
SEMIO_RAT                              -0.172710
SEMIO_TRADV                            -0.188190
SEMIO_PFLICHT                          -0.207816
SEMIO_REL                              -0.215830
FINANZ_ANLEGER                         -0.222355
FINANZ_UNAUFFAELLIGER                  -0.227434
FINANZ_SPARER                          -0.233402
PRAEGENDE_JUGENDJAHRE_Generation       -0.239039
Name: 1, Length: 196, dtype: float64
```

```
In [59]: pca_w_2 = pca_w(pca_40, 2)
         print (pca_w_2)
```

```
SEMIO_VERT                            0.282017
SEMIO_KULT                            0.255112
SEMIO_SOZ                             0.251176
SEMIO_FAM                             0.133401
NATIONALITAET_KZ_1.0                  0.126548
SHOPPER_TYP_0.0                       0.122538
PLZ8_ANTG4                            0.116668
HH_EINKOMMEN_SCORE                    0.113267
PLZ8_BAUMAX                           0.101802
SEMIO_TRADV                           0.099453
ZABEOTYP_1                            0.097185
SHOPPER_TYP_1.0                       0.085095
ANZ_PERSONEN                          0.080650
LP_LEBENSPHASE_FEIN                   0.080040
LP_STATUS_GROB_1.0                    0.078095
ONLINE_AFFINITAET                     0.070377
FINANZTYP_5                           0.067650
LP_LEBENSPHASE_GROB                   0.066266
KBA05_ANTG4                           0.065441
ANZ_HAUSHALTE_AKTIV                   0.064955
ZABEOTYP_4                            0.064206
CJT_GESAMTTYP_4.0                     0.064127
KBA05_ANTG3                           0.063824
KBA05_ANTG2                           0.063675
CAMEO_DEUG_2015_9                     0.063525
LP_STATUS_FEIN_1.0                    0.063329
KONSUMNAEHE                           0.061547
GEBAEUDETYP_3.0                       0.061525
SEMIO_MAT                             0.059507
NATIONALITAET_KZ_2.0                  0.058315
                                        ...
CJT_GESAMTTYP_2.0                    -0.036357
```

```
FINANZTYP_2                        -0.039486
ALTERSKATEGORIE_GROB               -0.045855
CAMEO_INTL_2015_wealth             -0.048470
EWDICHTE                           -0.050644
GEBAEUDETYP_RASTER                 -0.052138
INNENSTADT                         -0.053770
WOHNDAUER_2008                     -0.055566
REGIOTYP                           -0.056337
OST_WEST_KZ                        -0.064092
HEALTH_TYP                         -0.069725
SEMIO_LUST                         -0.077750
VERS_TYP                           -0.080093
FINANZ_UNAUFFAELLIGER              -0.084726
BALLRAUM                           -0.089059
W_KEIT_KIND_HH                     -0.092634
FINANZ_ANLEGER                     -0.108929
CJT_GESAMTTYP_6.0                  -0.121714
SEMIO_ERL                          -0.124921
PRAEGENDE_JUGENDJAHRE_movement     -0.125463
LP_STATUS_GROB_2.0                 -0.128956
GFK_URLAUBERTYP_5.0                -0.143777
KBA13_ANZAHL_PKW                   -0.158235
FINANZTYP_4                        -0.167764
LP_STATUS_FEIN_5.0                 -0.176455
ZABEOTYP_3                         -0.182959
ANREDE_KZ                          -0.216789
SEMIO_DOM                          -0.232563
SEMIO_KAEM                         -0.260238
SEMIO_KRIT                         -0.273175
Name: 2, Length: 196, dtype: float64
```

1.2.6 Discussion 2.3: Interpret Principal Components

For the first principal component has a positive association with:
PLZ8_ANTG3 0.182685 Number of 6-10 family houses in the PLZ8 region
CAMEO_INTL_2015_wealth 0.180481 Wealth Households
LP_STATUS_GROB_1.0 0.177280 Social status, typical low-income earners
and a negative association with: PLZ8_ANTG1 -0.182247 Number of 1-2 family houses in the PLZ8 region MOBI_REGIO -0.183851 Movement patterns FINANZ_MINIMALIST -0.186181 Financial topology, low financial interest The first component is related to the financial, social status, movement, and share of 1-2 family homes.

The second principal component has a positive association with:
ALTERSKATEGORIE_GROB 0.226923 Estimated age
FINANZ_VORSORGER 0.218186 Financial typology, be prepared
SEMIO_ERL 0.175144 Personality typology,event-oriented
and a negative association with:
FINANZ_UNAUFFAELLIGER -0.227434 Financial typology, inconspicuous

FINANZ_SPARER -0.233402 Financial typology, money-saver
PRAEGENDE_JUGENDJAHRE_Generation -0.239039 Dominating movement of person's youth
It seems like the second component is linked to age, movement, and financial status.

The third principal component has a positive association with:
SEMIO_VERT 0.282017 Personality typology, dreamful
SEMIO_KULT 0.255112 Personality typology, cultural-minded
SEMIO_SOZ 0.251176 Personality typology, socially-minded
and a negative association with:
SEMIO_DOM -0.232563 Personality typology, dominant-minded
SEMIO_KAEM -0.260238 Personality typology, combative attitude
SEMIO_KRIT -0.273175 Personality typology, critical-minded
The third principal component is related to personality typology.

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 [65]: from sklearn.cluster import KMeans
```

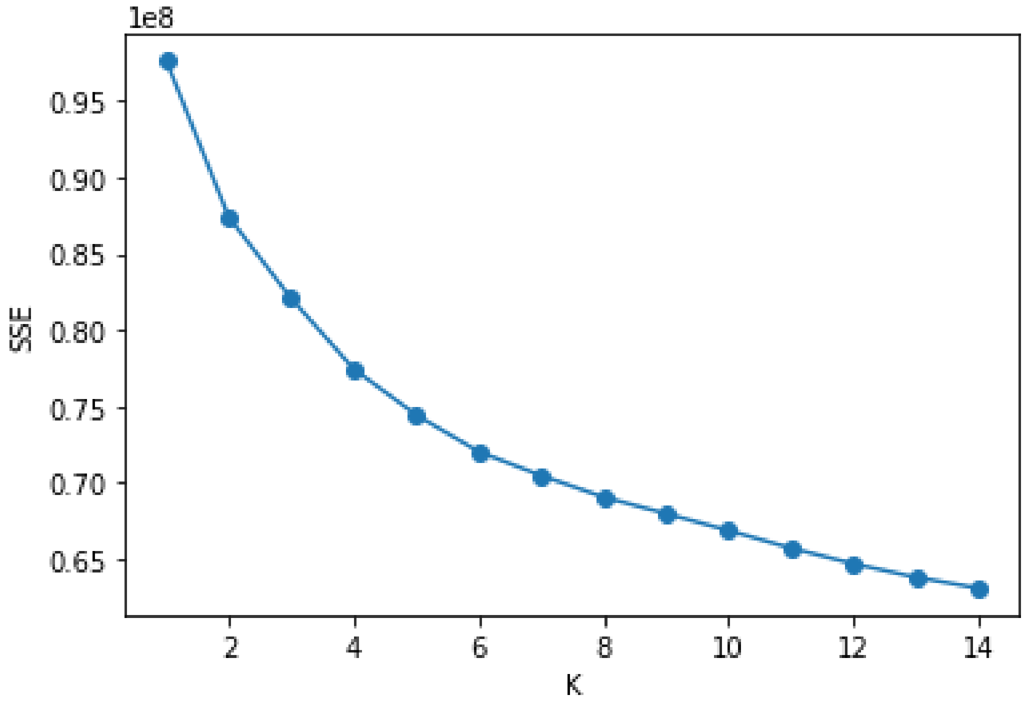
```
In [66]: def k_mean_score(data, n):
             kmeans = KMeans(n_clusters = n)
             model = kmeans.fit(data)
             score = np.abs(model.score(data))
             return score
```

```
In [67]: # Over a number of different cluster counts...
         score_list = []
         kscore = list(range(1,15))
```

```
In [68]: for k in kscore:
             score_list.append(k_mean_score(azdias_pca, k))
             # run k-means clustering on the data and.
```

```
In [69]: # Investigate the change in within-cluster distance across number of clusters.
         # HINT: Use matplotlib's plot function to visualize this relationship.
         plt.plot(kscore, score_list, linestyle='-', marker='o')
         plt.xlabel('K')
         plt.ylabel('SSE')
```

```
Out[69]: Text(0,0.5,'SSE')
```



```
In [105]: # 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 = 6)
          model_6 = kmeans.fit(azdias_pca)
          azdias_pred = model_6.predict(azdias_pca)
```

1.3.2 Discussion 3.1: Apply Clustering to General Population

I chose 4 clusters, as there is a elbow.

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

In [85]: # Apply preprocessing, feature transformation, and clustering from the general
         # demographics onto the customer data, obtaining cluster predictions for the
         # customer demographics data.
         customers_clean = clean_data(customers)

In [86]: customers_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 141713 entries, 0 to 191651
Columns: 195 entries, ALTERSKATEGORIE_GROB to CAMEO_INTL_2015_lifestage
dtypes: float64(44), int64(24), uint8(127)
memory usage: 91.8 MB
```

```
In [87]: customers_clean.head()
```

```
Out[87]:    ALTERSKATEGORIE_GROB  ANREDE_KZ  FINANZ_MINIMALIST  FINANZ_SPARER  \
         0                   4.0          0                  5              1
         2                   4.0          1                  5              1
         3                   4.0          0                  5              1
         4                   3.0          0                  3              1
         5                   3.0          0                  5              1

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

            GREEN_AVANTGARDE  HEALTH_TYP            ...              CAMEO_DEU_2015_8D  \
0                 1         1.0            ...                               0
2                 1         2.0            ...                               0
3                 0         2.0            ...                               0
4                 0         3.0            ...                               0
5                 1         3.0            ...                               0

   CAMEO_DEU_2015_9A  CAMEO_DEU_2015_9B  CAMEO_DEU_2015_9C  CAMEO_DEU_2015_9D  \
0                  0                  0                  0                  0
2                  0                  0                  0                  0
3                  0                  0                  0                  0
4                  0                  0                  0                  0
5                  0                  0                  0                  0

   CAMEO_DEU_2015_9E  PRAEGENDE_JUGENDJAHRE_Generation  \
0                  0                               1.0
2                  0                               1.0
3                  0                               0.0
4                  0                               3.0
5                  0                               1.0

   PRAEGENDE_JUGENDJAHRE_movement  CAMEO_INTL_2015_wealth  \
0                               1                     1.0
2                               1                     3.0
3                               0                     2.0
4                               0                     4.0
5                               1                     3.0

   CAMEO_INTL_2015_lifestage
0                        3.0
2                        4.0
3                        4.0
4                        1.0
5                        4.0

[5 rows x 195 columns]
```

```
In [88]: cust_clean_imp = pd.DataFrame(fill_na.fit_transform(customers_clean))
```

```
In [89]: cust_clean_imp.columns = customers_clean.columns
         cust_clean_imp.index = customers_clean.index
```

```
In [90]: # Apply scaler
         cust_clean_scaled = scaler.fit_transform(cust_clean_imp)
```

```
In [91]: cust_clean_scaled = pd.DataFrame(cust_clean_scaled)
```

```
In [92]: cust_clean_scaled.columns =list(cust_clean_imp)
```

```
In [93]: # PCA transformation
         customers_pca = pca_40.fit_transform(cust_clean_scaled)
```

```
In [106]: # Predict using Kmeans model_12
          customers_pred = model_6.predict(customers_pca)
```

1.3.4 Step 3.3: Compare Customer Data to Demographics Data

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

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

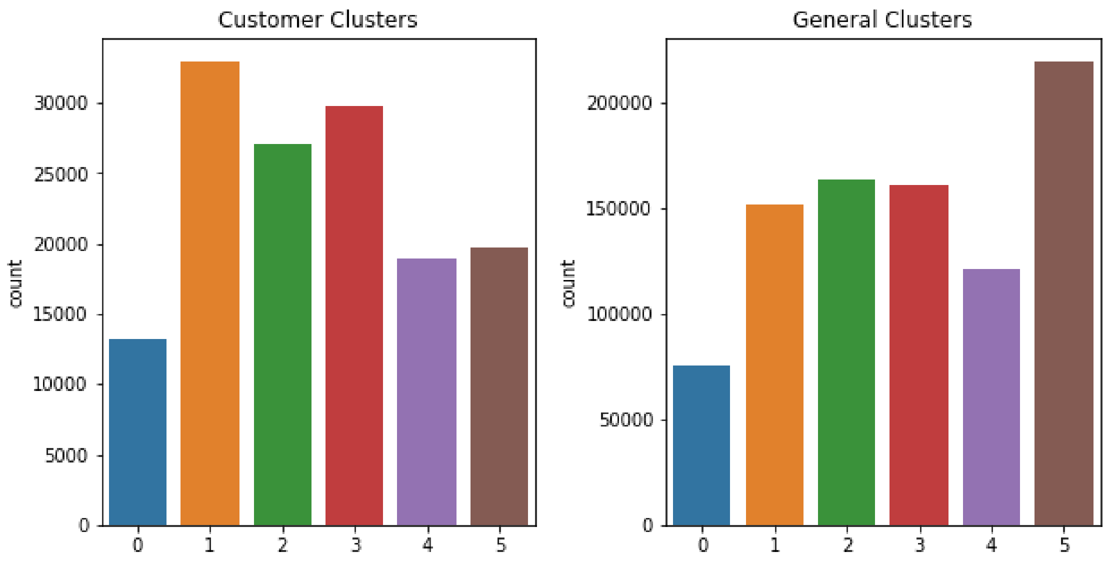
Take a look at the following points in this step:

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

```
In [107]: # Compare the proportion of data in each cluster for the customer data to the
          # proportion of data in each cluster for the general population.
          figure, axs = plt.subplots(nrows=1, ncols=2, figsize = (10,5))
          figure.subplots_adjust(hspace = 1, wspace=.3)
          sns.countplot(customers_pred, ax=axs[0])
          axs[0].set_title('Customer Clusters')
```

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

```
Out[107]: Text(0.5,1,'General Clusters')
```



```
In [109]: # What kinds of people are part of a cluster that is overrepresented in the
          # customer data compared to the general population?
          cluster_1 = scaler.inverse_transform(pca_40.inverse_transform(model_6.cluster_centers_
```

```
In [110]: overrepresented = pd.Series(data = cluster_1, index = customers_clean.columns)
          overrepresented
```

```
Out[110]: ALTERSKATEGORIE_GROB                 3.874270
          ANREDE_KZ                           -0.087455
          FINANZ_MINIMALIST                    5.047010
          FINANZ_SPARER                        0.637423
          FINANZ_VORSORGER                     5.337155
          FINANZ_ANLEGER                       0.518205
          FINANZ_UNAUFFAELLIGER                1.296374
          FINANZ_HAUSBAUER                     2.650494
          GREEN_AVANTGARDE                     0.800302
          HEALTH_TYP                           1.754824
          LP_LEBENSPHASE_FEIN                 25.363299
          LP_LEBENSPHASE_GROB                  7.354715
          RETOURTYP_BK_S                       4.639121
          SEMIO_SOZ                            5.161808
          SEMIO_FAM                            4.681332
          SEMIO_REL                            3.471374
          SEMIO_MAT                            3.816338
```

```
SEMIO_VERT                          6.663739
SEMIO_LUST                          6.295289
SEMIO_ERL                           4.007930
SEMIO_KULT                          4.904678
SEMIO_RAT                           1.960686
SEMIO_KRIT                          2.613467
SEMIO_DOM                           2.830080
SEMIO_KAEM                          1.999540
SEMIO_PFLICHT                       2.371891
SEMIO_TRADV                         2.644247
SOHO_KZ                             0.008810
VERS_TYP                            0.421729
ANZ_PERSONEN                        2.191105
                                      ...
CAMEO_DEU_2015_5A                   0.002311
CAMEO_DEU_2015_5B                   0.003728
CAMEO_DEU_2015_5C                   0.001512
CAMEO_DEU_2015_5D                   0.019068
CAMEO_DEU_2015_5E                   0.007691
CAMEO_DEU_2015_5F                   0.007628
CAMEO_DEU_2015_6A                   0.005406
CAMEO_DEU_2015_6B                   0.066729
CAMEO_DEU_2015_6C                   0.023220
CAMEO_DEU_2015_6D                   0.006789
CAMEO_DEU_2015_6E                   0.015433
CAMEO_DEU_2015_6F                   0.007965
CAMEO_DEU_2015_7A                   0.006566
CAMEO_DEU_2015_7B                   0.013292
CAMEO_DEU_2015_7C                   0.006866
CAMEO_DEU_2015_7D                   0.005125
CAMEO_DEU_2015_7E                   0.012072
CAMEO_DEU_2015_8A                   0.036688
CAMEO_DEU_2015_8B                   0.020618
CAMEO_DEU_2015_8C                   0.016776
CAMEO_DEU_2015_8D                   0.019690
CAMEO_DEU_2015_9A                   0.005042
CAMEO_DEU_2015_9B                   0.002082
CAMEO_DEU_2015_9C                   0.001899
CAMEO_DEU_2015_9D                   0.007661
CAMEO_DEU_2015_9E                   0.007659
PRAEGENDE_JUGENDJAHRE_Generation    0.655066
PRAEGENDE_JUGENDJAHRE_movement      0.800637
CAMEO_INTL_2015_wealth              2.219974
CAMEO_INTL_2015_lifestage           3.638926
Length: 195, dtype: float64
```

```
In [111]: # What kinds of people are part of a cluster that is underrepresented in the
          # customer data compared to the general population?`a
```

```
        cluster5 = scaler.inverse_transform(pca_40.inverse_transform(model_6.cluster_centers_[
```

```
In [112]: underrepresented = pd.Series(data = cluster5, index = customers_clean.columns)
          underrepresented
```

```
Out[112]: ALTERSKATEGORIE_GROB                  2.872039
          ANREDE_KZ                             0.562648
          FINANZ_MINIMALIST                     3.355831
          FINANZ_SPARER                         2.209898
          FINANZ_VORSORGER                      3.846880
          FINANZ_ANLEGER                        2.500590
          FINANZ_UNAUFFAELLIGER                 2.375765
          FINANZ_HAUSBAUER                      3.008812
          GREEN_AVANTGARDE                      0.128685
          HEALTH_TYP                            2.359024
          LP_LEBENSPHASE_FEIN                  22.564873
          LP_LEBENSPHASE_GROB                   7.030138
          RETOURTYP_BK_S                        3.563825
          SEMIO_SOZ                             3.929800
          SEMIO_FAM                             3.789651
          SEMIO_REL                             3.589582
          SEMIO_MAT                             3.477693
          SEMIO_VERT                            4.135164
          SEMIO_LUST                            4.383255
          SEMIO_ERL                             5.135069
          SEMIO_KULT                            3.750129
          SEMIO_RAT                             3.970403
          SEMIO_KRIT                            4.743049
          SEMIO_DOM                             4.882841
          SEMIO_KAEM                            4.705652
          SEMIO_PFLICHT                         4.010989
          SEMIO_TRADV                           3.510628
          SOHO_KZ                               0.012767
          VERS_TYP                              0.626545
          ANZ_PERSONEN                          2.215778
                                                  ...
          CAMEO_DEU_2015_5A                     0.023504
          CAMEO_DEU_2015_5B                     0.013886
          CAMEO_DEU_2015_5C                     0.015614
          CAMEO_DEU_2015_5D                     0.021947
          CAMEO_DEU_2015_5E                     0.006213
          CAMEO_DEU_2015_5F                     0.003740
          CAMEO_DEU_2015_6A                     0.006574
          CAMEO_DEU_2015_6B                     0.040569
          CAMEO_DEU_2015_6C                     0.010800
          CAMEO_DEU_2015_6D                     0.002754
          CAMEO_DEU_2015_6E                     0.014009
          CAMEO_DEU_2015_6F                     0.001727
```

```
CAMEO_DEU_2015_7A                    0.051623
CAMEO_DEU_2015_7B                    0.045247
CAMEO_DEU_2015_7C                    0.010751
CAMEO_DEU_2015_7D                    0.007581
CAMEO_DEU_2015_7E                    0.005210
CAMEO_DEU_2015_8A                    0.066241
CAMEO_DEU_2015_8B                    0.056769
CAMEO_DEU_2015_8C                    0.032940
CAMEO_DEU_2015_8D                    0.022941
CAMEO_DEU_2015_9A                    0.014256
CAMEO_DEU_2015_9B                    0.016344
CAMEO_DEU_2015_9C                    0.017839
CAMEO_DEU_2015_9D                    0.029052
CAMEO_DEU_2015_9E                    0.014658
PRAEGENDE_JUGENDJAHRE_Generation     2.965214
PRAEGENDE_JUGENDJAHRE_movement       0.148908
CAMEO_INTL_2015_wealth               3.110039
CAMEO_INTL_2015_lifestage            2.675339
Length: 195, dtype: float64
```

1.3.5 Discussion 3.3: Compare Customer Data to Demographics Data

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 [ ]:
```