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

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

Out[1]:
```
'\nImport note: The classroom currently uses sklearn version 0.19.\nI
f you need to use an imputer, it is available in sklearn.preprocessin
g.Imputer,\ninstead of sklearn.impute as in newer versions of sklear
n.\n'
```

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()` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html) call to read in the data properly. Also, considering the size of the main dataset, it may take some time for it to load completely.

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

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

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

In [24]:
```
# Check the structure of the data after it's loaded (e.g. print the n
umber of
# rows and columns, print the first few rows).
print(azdias.shape)
azdias.head()
```

```
(891221, 85)
```

Out[24]:

	AGER_TYP	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST
0	-1	2	1	2.0	3
1	-1	1	2	5.0	1
2	-1	3	2	3.0	1
3	2	4	2	2.0	4
4	-1	3	1	5.0	4

5 rows × 85 columns

In [25]:
```
print(feat_info.shape[0])
feat_info.head(62)
```

85

Out[25]:

	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]
5	FINANZ_SPARER	person	ordinal	[-1]
6	FINANZ_VORSORGER	person	ordinal	[-1]
7	FINANZ_ANLEGER	person	ordinal	[-1]
8	FINANZ_UNAUFFAELLIGER	person	ordinal	[-1]
9	FINANZ_HAUSBAUER	person	ordinal	[-1]
10	FINANZTYP	person	categorical	[-1]
11	GEBURTSJAHR	person	numeric	[0]
12	GFK_URLAUBERTYP	person	categorical	[]
13	GREEN_AVANTGARDE	person	categorical	[]
14	HEALTH_TYP	person	ordinal	[-1,0]
15	LP_LEBENSPHASE_FEIN	person	mixed	[0]
16	LP_LEBENSPHASE_GROB	person	mixed	[0]
17	LP_FAMILIE_FEIN	person	categorical	[0]
18	LP_FAMILIE_GROB	person	categorical	[0]
19	LP_STATUS_FEIN	person	categorical	[0]
20	LP_STATUS_GROB	person	categorical	[0]
21	NATIONALITAET_KZ	person	categorical	[-1,0]
22	PRAEGENDE_JUGENDJAHRE	person	mixed	[-1,0]
23	RETOURTYP_BK_S	person	ordinal	[0]
24	SEMIO_SOZ	person	ordinal	[-1,9]
25	SEMIO_FAM	person	ordinal	[-1,9]
26	SEMIO_REL	person	ordinal	[-1,9]
27	SEMIO_MAT	person	ordinal	[-1,9]
28	SEMIO_VERT	person	ordinal	[-1,9]
29	SEMIO_LUST	person	ordinal	[-1,9]
...	...	...	...	...
32	SEMIO_RAT	person	ordinal	[-1,9]
33	SEMIO_KRIT	person	ordinal	[-1,9]
34	SEMIO_DOM	person	ordinal	[-1,9]
35	SEMIO_KAEM	person	ordinal	[-1,9]

	attribute	information_level	type	missing_or_unknown
36	SEMIO_PFLICHT	person	ordinal	[-1,9]
37	SEMIO_TRADV	person	ordinal	[-1,9]
38	SHOPPER_TYP	person	categorical	[-1]
39	SOHO_KZ	person	categorical	[-1]
40	TITEL_KZ	person	categorical	[-1,0]
41	VERS_TYP	person	categorical	[-1]
42	ZABEOTYP	person	categorical	[-1,9]
43	ALTER_HH	household	interval	[0]
44	ANZ_PERSONEN	household	numeric	[]
45	ANZ_TITEL	household	numeric	[]
46	HH_EINKOMMEN_SCORE	household	ordinal	[-1,0]
47	KK_KUNDENTYP	household	categorical	[-1]
48	W_KEIT_KIND_HH	household	ordinal	[-1,0]
49	WOHNDAUER_2008	household	ordinal	[-1,0]
50	ANZ_HAUSHALTE_AKTIV	building	numeric	[0]
51	ANZ_HH_TITEL	building	numeric	[]
52	GEBAEUDETYP	building	categorical	[-1,0]
53	KONSUMNAEHE	building	ordinal	[]
54	MIN_GEBAEUDEJAHR	building	numeric	[0]
55	OST_WEST_KZ	building	categorical	[-1]
56	WOHNLAGE	building	mixed	[-1]
57	CAMEO_DEUG_2015	microcell_rr4	categorical	[-1,X]
58	CAMEO_DEU_2015	microcell_rr4	categorical	[XX]
59	CAMEO_INTL_2015	microcell_rr4	mixed	[-1,XX]
60	KBA05_ANTG1	microcell_rr3	ordinal	[-1]
61	KBA05_ANTG2	microcell_rr3	ordinal	[-1]

62 rows × 4 columns

In [26]:
```
feat_info['missing_or_unknown'].unique()
```

Out[26]:
```
array(['[-1,0]', '[-1,0,9]', '[0]', '[-1]', '[]', '[-1,9]', '[-1,X]',
       '[XX]', '[-1,XX]'], dtype=object)
```

- Pre-dealing with missings and unknowns!

In [27]:
```
missings=feat_info['missing_or_unknown'][4].split('[')[1].split(']')[
0].split(',')
```

In [28]:
```
misslist=[]
for i in range(len(missings)):
    misslist.append(int(missings[i]))
```

In [29]:
```
misslist
```

Out[29]:
```
[-1]
```

In [30]:
```
#azdias[listcols[4]].replace([1,0],np.nan)
```

In [31]:
```
missings
```

Out[31]:
```
['-1']
```

In [32]:
```
listcols=list(azdias.columns)
```

In [33]:
```
azdias[listcols[57]].unique()
```

Out[33]:
```
array([nan, '8', '4', '2', '6', '1', '9', '5', '7', '3', 'X'], dtype=
object)
```

In [34]:
```
azdias.dtypes
```

Out[34]:
```
AGER_TYP                  int64
ALTERSKATEGORIE_GROB      int64
ANREDE_KZ                 int64
CJT_GESAMTTYP           float64
FINANZ_MINIMALIST         int64
FINANZ_SPARER             int64
FINANZ_VORSORGER          int64
FINANZ_ANLEGER            int64
FINANZ_UNAUFFAELLIGER     int64
FINANZ_HAUSBAUER          int64
FINANZTYP                 int64
GEBURTSJAHR               int64
GFK_URLAUBERTYP         float64
GREEN_AVANTGARDE          int64
HEALTH_TYP                int64
LP_LEBENSPHASE_FEIN     float64
LP_LEBENSPHASE_GROB     float64
LP_FAMILIE_FEIN         float64
LP_FAMILIE_GROB         float64
LP_STATUS_FEIN          float64
LP_STATUS_GROB          float64
NATIONALITAET_KZ          int64
PRAEGENDE_JUGENDJAHRE     int64
RETOURTYP_BK_S          float64
SEMIO_SOZ                 int64
SEMIO_FAM                 int64
SEMIO_REL                 int64
SEMIO_MAT                 int64
SEMIO_VERT                int64
SEMIO_LUST                int64
                          ...
OST_WEST_KZ              object
WOHNLAGE                float64
CAMEO_DEUG_2015          object
CAMEO_DEU_2015           object
CAMEO_INTL_2015          object
KBA05_ANTG1             float64
KBA05_ANTG2             float64
KBA05_ANTG3             float64
KBA05_ANTG4             float64
KBA05_BAUMAX            float64
KBA05_GBZ               float64
BALLRAUM                float64
EWDICHTE                float64
INNENSTADT              float64
GEBAEUDETYP_RASTER      float64
KKK                     float64
MOBI_REGIO              float64
ONLINE_AFFINITAET       float64
REGIOTYP                float64
KBA13_ANZAHL_PKW        float64
PLZ8_ANTG1              float64
PLZ8_ANTG2              float64
PLZ8_ANTG3              float64
PLZ8_ANTG4              float64
PLZ8_BAUMAX             float64
PLZ8_HHZ                float64
```

```
PLZ8_GBZ                float64
ARBEIT                  float64
ORTSGR_KLS9             float64
RELAT_AB                float64
Length: 85, dtype: object
```

In [35]:
```
azdias[listcols[0]][0] in [-1,0]
```

Out[35]:
```
True
```

In [36]:
```
azdias_clean=azdias.copy()
for feat in range(feat_info.shape[0]):
#for featttt in range(1):
#    feat=57
    missings=feat_info['missing_or_unknown'][feat].split('[')[1].spli
t(']')[0].split(',')
    misslist=[]
    misslist2=[]
#    print(missings,len(missings))
    for i in range(len(missings)):
#        if( isinstance(int(missings[i]),int)):
        if( missings[i]!='' and missings[i]!='X' and missings[i]!='X
X'):
            misslist.append(int(missings[i]))
        else:
            misslist.append(missings[i])
#    print(misslist)
#            continue
#    if(misslist!=[]):
    colname=listcols[feat]
#    print(colname)
    azdias_clean[colname].replace(misslist,np.nan,inplace=True)
#    print(str(misslist2[0]))
#    azdias_clean[colname].replace(misslist2,np.nan, inplace=True)
#    else:
#        continue
    """
    for j in range(azdias.shape[0]):
        colname=listcols[feat]
        if(azdias[colname][j] in misslist):
            azdias_clean[colname][j]=np.nan
        else:
            pass
    """
```

In [37]:
```
azdias_clean[listcols[57]].unique()
```

Out[37]:
```
array([nan, '8', '4', '2', '6', '1', '9', '5', '7', '3'], dtype=objec
t)
```

In [38]:
```
azdias['AGER_TYP'].unique()
```

Out[38]:
```
array([-1,  2,  3,  0,  1])
```

> **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`.

Step 1: Preprocessing

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 [18]:
```
# Identify missing or unknown data values and convert them to NaNs.
```

Step 1.1.2: Assess Missing Data in Each Column

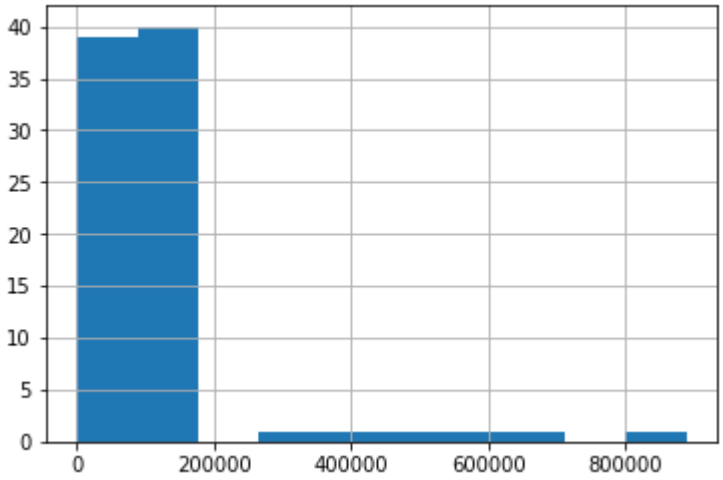
How much missing data is present in each column? There are a few columns that are outliers in terms of the proportion of values that are missing. You will want to use matplotlib's `hist()` (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html) function to visualize the distribution of missing 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 [19]:
```
# Perform an assessment of how much missing data there is in each col
umn of the
# dataset.
nulls=azdias_clean.isnull().sum()
```

In [20]:
```
# Investigate patterns in the amount of missing data in each column.
nulls.hist()
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa841ef2b0>
```



I'd say that columns with more than 200000 missing data are outliers

In [21]:
```
# Remove the outlier columns from the dataset. (You'll perform other
 data
# engineering tasks such as re-encoding and imputation later.)
remcols=[]
for i in range(len(nulls)):
    if(nulls[i]>=200000):
        remcols.append(i)
    else:
        pass
```

In [22]:
```
remcols
```

Out[22]:
```
[0, 11, 40, 43, 47, 64]
```

In [23]:
```
#listcols
```

In [24]:
```
azdias_clean2=azdias_clean.copy()
for i in range(len(remcols)):
#    print(listcols[remcols[i]])
    col_drop=listcols[remcols[i]]
    azdias_clean2.drop(col_drop,axis=1,inplace=True)
```

In [25]:
```
azdias_clean2
```

Out[25]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINA
0	2.0	1	2.0	3	
1	1.0	2	5.0	1	
2	3.0	2	3.0	1	
3	4.0	2	2.0	4	
4	3.0	1	5.0	4	
5	1.0	2	2.0	3	
6	2.0	2	5.0	1	
7	1.0	1	3.0	3	
8	3.0	1	3.0	4	
9	3.0	2	4.0	2	
10	3.0	2	1.0	2	
11	2.0	1	6.0	3	
12	3.0	1	6.0	5	
13	1.0	2	5.0	1	
14	3.0	1	6.0	3	
15	4.0	2	4.0	4	
16	1.0	2	1.0	4	
17	2.0	1	6.0	3	
18	2.0	2	6.0	2	
19	3.0	1	3.0	5	
20	2.0	2	4.0	4	
21	2.0	1	3.0	3	
22	1.0	1	4.0	1	
23	3.0	1	3.0	5	
24	3.0	2	6.0	3	
25	1.0	1	3.0	3	
26	3.0	1	3.0	5	
27	3.0	1	4.0	3	
28	3.0	1	2.0	3	
29	4.0	2	1.0	5	
...	...	...	...	...	
891191	4.0	2	1.0	4	
891192	1.0	2	3.0	1	
891193	4.0	1	3.0	4	
891194	3.0	1	4.0	4	

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINA
891195	4.0	2	6.0	3	
891196	2.0	2	6.0	1	
891197	3.0	2	1.0	3	
891198	3.0	1	5.0	2	
891199	2.0	1	3.0	2	
891200	1.0	2	3.0	1	
891201	3.0	1	3.0	4	
891202	2.0	2	5.0	1	
891203	4.0	2	1.0	4	
891204	3.0	1	5.0	4	
891205	4.0	1	2.0	4	
891206	1.0	2	4.0	3	
891207	3.0	2	1.0	5	
891208	4.0	1	2.0	5	
891209	1.0	2	5.0	1	
891210	3.0	1	5.0	3	
891211	3.0	1	2.0	3	
891212	4.0	1	1.0	3	
891213	4.0	2	5.0	3	
891214	1.0	2	4.0	1	
891215	2.0	2	6.0	1	
891216	3.0	2	5.0	1	
891217	2.0	1	4.0	3	
891218	2.0	2	4.0	2	
891219	1.0	1	3.0	1	
891220	4.0	1	1.0	4	

891221 rows × 79 columns

In [26]:
```
feat_info2=feat_info.copy()

feat_info2.drop(feat_info2.index[remcols],inplace=True)
```

In [27]:
```
feat_info2=feat_info2.reset_index()
feat_info2.drop('index',axis=1,inplace=True)
```

In [28]:
```
feat_info2
```

Out[28]:

	attribute	information_level	type	missing_or_unknown
0	ALTERSKATEGORIE_GROB	person	ordinal	[-1,0,9]
1	ANREDE_KZ	person	categorical	[-1,0]
2	CJT_GESAMTTYP	person	categorical	[0]
3	FINANZ_MINIMALIST	person	ordinal	[-1]
4	FINANZ_SPARER	person	ordinal	[-1]
5	FINANZ_VORSORGER	person	ordinal	[-1]
6	FINANZ_ANLEGER	person	ordinal	[-1]
7	FINANZ_UNAUFFAELLIGER	person	ordinal	[-1]
8	FINANZ_HAUSBAUER	person	ordinal	[-1]
9	FINANZTYP	person	categorical	[-1]
10	GFK_URLAUBERTYP	person	categorical	[]
11	GREEN_AVANTGARDE	person	categorical	[]
12	HEALTH_TYP	person	ordinal	[-1,0]
13	LP_LEBENSPHASE_FEIN	person	mixed	[0]
14	LP_LEBENSPHASE_GROB	person	mixed	[0]
15	LP_FAMILIE_FEIN	person	categorical	[0]
16	LP_FAMILIE_GROB	person	categorical	[0]
17	LP_STATUS_FEIN	person	categorical	[0]
18	LP_STATUS_GROB	person	categorical	[0]
19	NATIONALITAET_KZ	person	categorical	[-1,0]
20	PRAEGENDE_JUGENDJAHRE	person	mixed	[-1,0]
21	RETOURTYP_BK_S	person	ordinal	[0]
22	SEMIO_SOZ	person	ordinal	[-1,9]
23	SEMIO_FAM	person	ordinal	[-1,9]
24	SEMIO_REL	person	ordinal	[-1,9]
25	SEMIO_MAT	person	ordinal	[-1,9]
26	SEMIO_VERT	person	ordinal	[-1,9]
27	SEMIO_LUST	person	ordinal	[-1,9]
28	SEMIO_ERL	person	ordinal	[-1,9]
29	SEMIO_KULT	person	ordinal	[-1,9]
...	...	...	...	...
49	MIN_GEBAEUDEJAHR	building	numeric	[0]
50	OST_WEST_KZ	building	categorical	[-1]
51	WOHNLAGE	building	mixed	[-1]
52	CAMEO_DEUG_2015	microcell_rr4	categorical	[-1,X]

	attribute	information_level	type	missing_or_unknown
53	CAMEO_DEU_2015	microcell_rr4	categorical	[XX]
54	CAMEO_INTL_2015	microcell_rr4	mixed	[-1,XX]
55	KBA05_ANTG1	microcell_rr3	ordinal	[-1]
56	KBA05_ANTG2	microcell_rr3	ordinal	[-1]
57	KBA05_ANTG3	microcell_rr3	ordinal	[-1]
58	KBA05_ANTG4	microcell_rr3	ordinal	[-1]
59	KBA05_GBZ	microcell_rr3	ordinal	[-1,0]
60	BALLRAUM	postcode	ordinal	[-1]
61	EWDICHTE	postcode	ordinal	[-1]
62	INNENSTADT	postcode	ordinal	[-1]
63	GEBAEUDETYP_RASTER	region_rr1	ordinal	[]
64	KKK	region_rr1	ordinal	[-1,0]
65	MOBI_REGIO	region_rr1	ordinal	[]
66	ONLINE_AFFINITAET	region_rr1	ordinal	[]
67	REGIOTYP	region_rr1	ordinal	[-1,0]
68	KBA13_ANZAHL_PKW	macrocell_plz8	numeric	[]
69	PLZ8_ANTG1	macrocell_plz8	ordinal	[-1]
70	PLZ8_ANTG2	macrocell_plz8	ordinal	[-1]
71	PLZ8_ANTG3	macrocell_plz8	ordinal	[-1]
72	PLZ8_ANTG4	macrocell_plz8	ordinal	[-1]
73	PLZ8_BAUMAX	macrocell_plz8	mixed	[-1,0]
74	PLZ8_HHZ	macrocell_plz8	ordinal	[-1]
75	PLZ8_GBZ	macrocell_plz8	ordinal	[-1]
76	ARBEIT	community	ordinal	[-1,9]
77	ORTSGR_KLS9	community	ordinal	[-1,0]
78	RELAT_AB	community	ordinal	[-1,9]

79 rows × 4 columns

In [ ]:

In [ ]:

In [ ]:

Discussion 1.1.2: Assess Missing Data in Each Column

(Double click this cell and replace this text with your own text, reporting your observations regarding the amount of missing data in each column. Are there any patterns in missing values? Which columns were removed from the dataset?)

It seems, from the histogram, that the NaN's above a threshold of 200000 are uniformly distributed and they seem to be kind of outliers, therefore they have been removed.

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()` (https://seaborn.pydata.org/generated/seaborn.countplot.html) function to create a bar chart of code frequencies and matplotlib's `subplot()` (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html) 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 [29]:
```
# How much data is missing in each row of the dataset?
nulls_sep=((azdias_clean2.isnull().sum()/azdias_clean2.shape[0])>=0.0
8)

mymask=[]

for i in range(len(nulls_sep)):
    mymask.append(nulls_sep[i])
```

In [30]:
```
#mymask
```

In [31]:
```
# Write code to divide the data into two subsets based on the number
 of missing
# values in each row.

below=azdias_clean2.copy()
above=azdias_clean2.copy()

listcols2=azdias_clean2.columns

for i in range(len(mymask)):
    if(mymask[i]==True):
        below.drop(listcols2[i],axis=1,inplace=True)
    else:
        above.drop(listcols2[i],axis=1,inplace=True)
```

In [32]:
```
above.head(5)
```

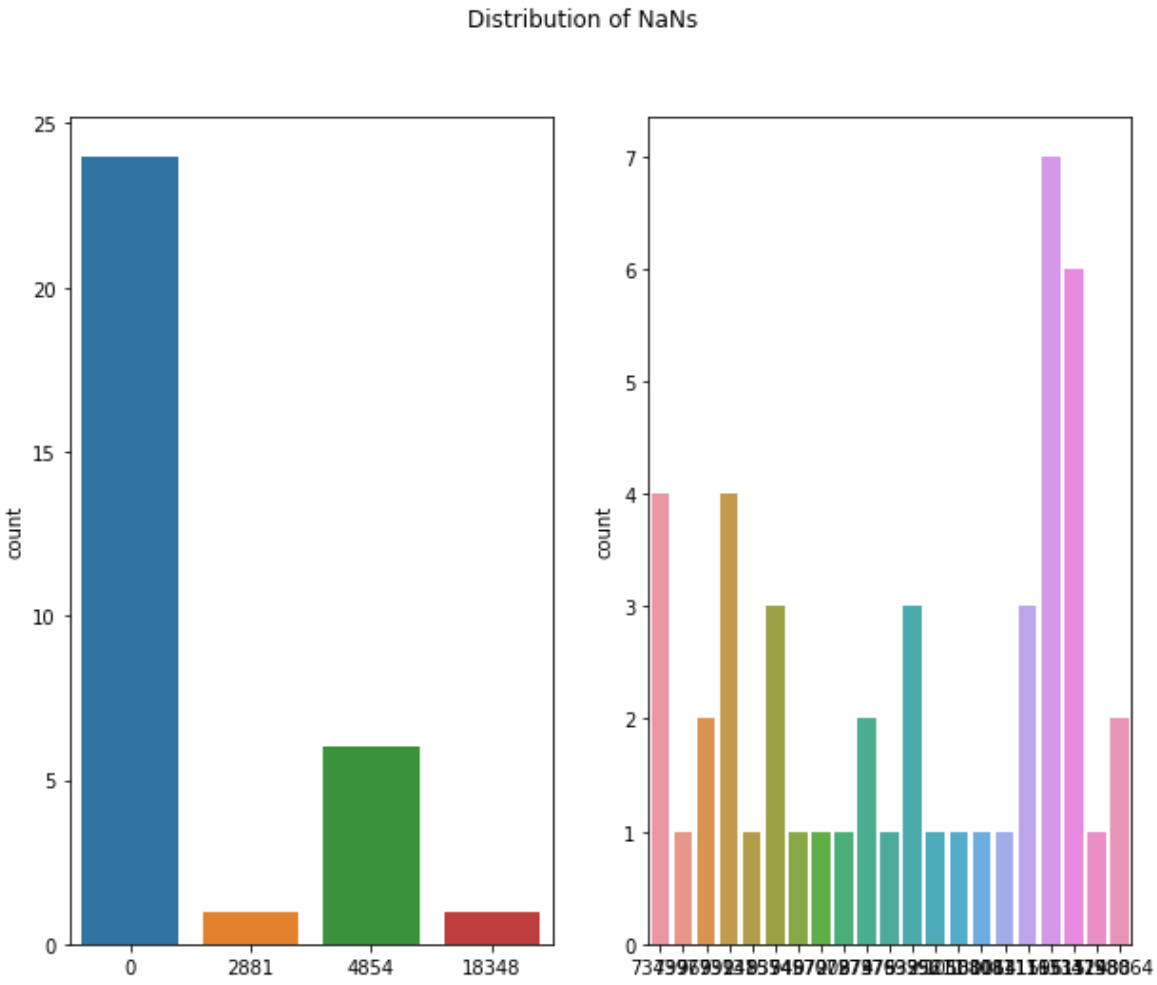
Out[32]:

	HEALTH_TYP	LP_LEBENSPHASE_FEIN	LP_LEBENSPHASE_GROB	LP_FAMILIE_FEIN	LP_FA
0	NaN	15.0	4.0	2.0	
1	3.0	21.0	6.0	5.0	
2	3.0	3.0	1.0	1.0	
3	2.0	NaN	NaN	NaN	
4	3.0	32.0	10.0	10.0	

5 rows × 47 columns

In [ ]:
```

```

In [33]:
```
# Compare the distribution of values for at least five columns where
 there are
# no or few missing values, between the two subsets.

from seaborn import countplot
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=[10,8])
#plt.figure(figsize=[10,8])
fig.suptitle('Distribution of NaNs')
countplot(below.isnull().sum(),ax=ax1)
countplot(above.isnull().sum(),ax=ax2)
plt.show()
```



Discussion 1.1.3: Assess Missing Data in Each Row

(Double-click this cell and replace this text with your own text, reporting your observations regarding missing data in rows. Are the data with lots of missing values are qualitatively different from data with few or no missing values?)

the above plot shows the splitted into two parts: below and above a thresold of 8% of NaN's. There seems to be a lot of columns above 8% of NaN's. The left graph has many columns with no NaN's, while the right graph has non-uniform distribution of NaN's across the columns, some of them having a lot of NaN's

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 [34]:
```
feat_info2
```

Out[34]:

	attribute	information_level	type	missing_or_unknown
0	ALTERSKATEGORIE_GROB	person	ordinal	[-1,0,9]
1	ANREDE_KZ	person	categorical	[-1,0]
2	CJT_GESAMTTYP	person	categorical	[0]
3	FINANZ_MINIMALIST	person	ordinal	[-1]
4	FINANZ_SPARER	person	ordinal	[-1]
5	FINANZ_VORSORGER	person	ordinal	[-1]
6	FINANZ_ANLEGER	person	ordinal	[-1]
7	FINANZ_UNAUFFAELLIGER	person	ordinal	[-1]
8	FINANZ_HAUSBAUER	person	ordinal	[-1]
9	FINANZTYP	person	categorical	[-1]
10	GFK_URLAUBERTYP	person	categorical	[]
11	GREEN_AVANTGARDE	person	categorical	[]
12	HEALTH_TYP	person	ordinal	[-1,0]
13	LP_LEBENSPHASE_FEIN	person	mixed	[0]
14	LP_LEBENSPHASE_GROB	person	mixed	[0]
15	LP_FAMILIE_FEIN	person	categorical	[0]
16	LP_FAMILIE_GROB	person	categorical	[0]
17	LP_STATUS_FEIN	person	categorical	[0]
18	LP_STATUS_GROB	person	categorical	[0]
19	NATIONALITAET_KZ	person	categorical	[-1,0]
20	PRAEGENDE_JUGENDJAHRE	person	mixed	[-1,0]
21	RETOURTYP_BK_S	person	ordinal	[0]
22	SEMIO_SOZ	person	ordinal	[-1,9]
23	SEMIO_FAM	person	ordinal	[-1,9]
24	SEMIO_REL	person	ordinal	[-1,9]
25	SEMIO_MAT	person	ordinal	[-1,9]
26	SEMIO_VERT	person	ordinal	[-1,9]
27	SEMIO_LUST	person	ordinal	[-1,9]
28	SEMIO_ERL	person	ordinal	[-1,9]
29	SEMIO_KULT	person	ordinal	[-1,9]
...	...	...	...	...
49	MIN_GEBAEUDEJAHR	building	numeric	[0]
50	OST_WEST_KZ	building	categorical	[-1]
51	WOHNLAGE	building	mixed	[-1]
52	CAMEO_DEUG_2015	microcell_rr4	categorical	[-1,X]

	attribute	information_level	type	missing_or_unknown
53	CAMEO_DEU_2015	microcell_rr4	categorical	[XX]
54	CAMEO_INTL_2015	microcell_rr4	mixed	[-1,XX]
55	KBA05_ANTG1	microcell_rr3	ordinal	[-1]
56	KBA05_ANTG2	microcell_rr3	ordinal	[-1]
57	KBA05_ANTG3	microcell_rr3	ordinal	[-1]
58	KBA05_ANTG4	microcell_rr3	ordinal	[-1]
59	KBA05_GBZ	microcell_rr3	ordinal	[-1,0]
60	BALLRAUM	postcode	ordinal	[-1]
61	EWDICHTE	postcode	ordinal	[-1]
62	INNENSTADT	postcode	ordinal	[-1]
63	GEBAEUDETYP_RASTER	region_rr1	ordinal	[]
64	KKK	region_rr1	ordinal	[-1,0]
65	MOBI_REGIO	region_rr1	ordinal	[]
66	ONLINE_AFFINITAET	region_rr1	ordinal	[]
67	REGIOTYP	region_rr1	ordinal	[-1,0]
68	KBA13_ANZAHL_PKW	macrocell_plz8	numeric	[]
69	PLZ8_ANTG1	macrocell_plz8	ordinal	[-1]
70	PLZ8_ANTG2	macrocell_plz8	ordinal	[-1]
71	PLZ8_ANTG3	macrocell_plz8	ordinal	[-1]
72	PLZ8_ANTG4	macrocell_plz8	ordinal	[-1]
73	PLZ8_BAUMAX	macrocell_plz8	mixed	[-1,0]
74	PLZ8_HHZ	macrocell_plz8	ordinal	[-1]
75	PLZ8_GBZ	macrocell_plz8	ordinal	[-1]
76	ARBEIT	community	ordinal	[-1,9]
77	ORTSGR_KLS9	community	ordinal	[-1,0]
78	RELAT_AB	community	ordinal	[-1,9]

79 rows × 4 columns

In [35]:
```
# How many features are there of each data type?
feat_info2['type'].unique()
```

Out[35]:
```
array(['ordinal', 'categorical', 'mixed', 'numeric'], dtype=object)
```

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 (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)), or (to keep things straightforward) just drop them from the analysis. As always, document your choices in the Discussion section.

In [36]:
```
feat_info2
```

Out[36]:

	attribute	information_level	type	missing_or_unknown
0	ALTERSKATEGORIE_GROB	person	ordinal	[-1,0,9]
1	ANREDE_KZ	person	categorical	[-1,0]
2	CJT_GESAMTTYP	person	categorical	[0]
3	FINANZ_MINIMALIST	person	ordinal	[-1]
4	FINANZ_SPARER	person	ordinal	[-1]
5	FINANZ_VORSORGER	person	ordinal	[-1]
6	FINANZ_ANLEGER	person	ordinal	[-1]
7	FINANZ_UNAUFFAELLIGER	person	ordinal	[-1]
8	FINANZ_HAUSBAUER	person	ordinal	[-1]
9	FINANZTYP	person	categorical	[-1]
10	GFK_URLAUBERTYP	person	categorical	[]
11	GREEN_AVANTGARDE	person	categorical	[]
12	HEALTH_TYP	person	ordinal	[-1,0]
13	LP_LEBENSPHASE_FEIN	person	mixed	[0]
14	LP_LEBENSPHASE_GROB	person	mixed	[0]
15	LP_FAMILIE_FEIN	person	categorical	[0]
16	LP_FAMILIE_GROB	person	categorical	[0]
17	LP_STATUS_FEIN	person	categorical	[0]
18	LP_STATUS_GROB	person	categorical	[0]
19	NATIONALITAET_KZ	person	categorical	[-1,0]
20	PRAEGENDE_JUGENDJAHRE	person	mixed	[-1,0]
21	RETOURTYP_BK_S	person	ordinal	[0]
22	SEMIO_SOZ	person	ordinal	[-1,9]
23	SEMIO_FAM	person	ordinal	[-1,9]
24	SEMIO_REL	person	ordinal	[-1,9]
25	SEMIO_MAT	person	ordinal	[-1,9]
26	SEMIO_VERT	person	ordinal	[-1,9]
27	SEMIO_LUST	person	ordinal	[-1,9]
28	SEMIO_ERL	person	ordinal	[-1,9]
29	SEMIO_KULT	person	ordinal	[-1,9]
...	...	...	...	...
49	MIN_GEBAEUDEJAHR	building	numeric	[0]
50	OST_WEST_KZ	building	categorical	[-1]
51	WOHNLAGE	building	mixed	[-1]
52	CAMEO_DEUG_2015	microcell_rr4	categorical	[-1,X]

	attribute	information_level	type	missing_or_unknown
53	CAMEO_DEU_2015	microcell_rr4	categorical	[XX]
54	CAMEO_INTL_2015	microcell_rr4	mixed	[-1,XX]
55	KBA05_ANTG1	microcell_rr3	ordinal	[-1]
56	KBA05_ANTG2	microcell_rr3	ordinal	[-1]
57	KBA05_ANTG3	microcell_rr3	ordinal	[-1]
58	KBA05_ANTG4	microcell_rr3	ordinal	[-1]
59	KBA05_GBZ	microcell_rr3	ordinal	[-1,0]
60	BALLRAUM	postcode	ordinal	[-1]
61	EWDICHTE	postcode	ordinal	[-1]
62	INNENSTADT	postcode	ordinal	[-1]
63	GEBAEUDETYP_RASTER	region_rr1	ordinal	[]
64	KKK	region_rr1	ordinal	[-1,0]
65	MOBI_REGIO	region_rr1	ordinal	[]
66	ONLINE_AFFINITAET	region_rr1	ordinal	[]
67	REGIOTYP	region_rr1	ordinal	[-1,0]
68	KBA13_ANZAHL_PKW	macrocell_plz8	numeric	[]
69	PLZ8_ANTG1	macrocell_plz8	ordinal	[-1]
70	PLZ8_ANTG2	macrocell_plz8	ordinal	[-1]
71	PLZ8_ANTG3	macrocell_plz8	ordinal	[-1]
72	PLZ8_ANTG4	macrocell_plz8	ordinal	[-1]
73	PLZ8_BAUMAX	macrocell_plz8	mixed	[-1,0]
74	PLZ8_HHZ	macrocell_plz8	ordinal	[-1]
75	PLZ8_GBZ	macrocell_plz8	ordinal	[-1]
76	ARBEIT	community	ordinal	[-1,9]
77	ORTSGR_KLS9	community	ordinal	[-1,0]
78	RELAT_AB	community	ordinal	[-1,9]

79 rows × 4 columns

In [37]:
```
# Assess categorical variables: which are binary, which are multi-lev
el, and
# which one needs to be re-encoded?

cats=(feat_info2['type']=='categorical')
```

In [38]:
```
mymask=[]
for i in range(len(cats)):
    mymask.append(cats[i])
```

In [39]:
```
len(mymask)
```

Out[39]:
```
79
```

In [ ]:
```

```

In [40]:
```
typp=azdias_clean2.dtypes
typ=[]
for i in range(len(typp)):
    typ.append(typp[i])
```

In [41]:
```
typ
```

Out[41]:
```
[dtype('float64'),
 dtype('int64'),
 dtype('float64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('float64'),
 dtype('int64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('int64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('int64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('float64'),
 dtype('O'),
 dtype('float64'),
 dtype('O'),
 dtype('O'),
 dtype('O'),
 dtype('float64'),
 dtype('float64'),
```

```
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64'),
dtype('float64')]
```

In [244]:
```
isinstance(azdias_clean2[listcols2[13]].unique()[1],str)
isinstance('i',str)
```

Out[244]: True

In [ ]:
```

```

In [246]:
```
#mymask

for feat in range(feat_info2.shape[0]):
#feat=13

    if(cats[feat]==True):
        if(len(azdias_clean2[listcols2[feat]].unique())==2):
            pass
        else:
            if(len(azdias_clean2[listcols2[feat]].unique())>2):
                print(azdias_clean2[listcols2[feat]].unique())
#                 if(isinstance(len(azdias_clean2[listcols2[feat]].uni
que()),str)):
#                 pd.getdummies()


#if(azdias_clean2[listcols2[feat]])
```

```
[  2.   5.   3.   4.   1.   6.  nan]
[4 1 6 5 2 3]
[ 10.   1.   5.  12.   9.   3.   8.  11.   4.   2.   7.   6.  nan]
[  2.   5.   1.  nan  10.   7.  11.   3.   8.   4.   6.   9.]
[  2.   3.   1.  nan   5.   4.]
[  1.   2.   3.   9.   4.  10.   5.   8.   6.   7.  nan]
[  1.   2.   4.   5.   3.  nan]
[ nan   1.   3.   2.]
[ nan   3.   2.   1.   0.]
[ nan   1.   0.]
[ nan   2.   1.]
[3 5 4 1 6 2]
[ nan   8.   1.   3.   2.   6.   4.   5.]
[nan 'W' 'O']
[nan '8' '4' '2' '6' '1' '9' '5' '7' '3']
[nan '8A' '4C' '2A' '6B' '8C' '4A' '2D' '1A' '1E' '9D' '5C' '8B' '7A'
'5D'
 '9E' '9B' '1B' '3D' '4E' '4B' '3C' '5A' '7B' '9A' '6D' '6E' '2C' '7
C' '9C'
 '7D' '5E' '1D' '8D' '6C' '6A' '5B' '4D' '3A' '2B' '7E' '3B' '6F' '5
F' '1C']
```

In [204]:
```
azdias_clean2[listcols2[73]]
```

Out[204]:

0	NaN
1	1.0
2	1.0
3	1.0
4	2.0
5	1.0
6	1.0
7	1.0
8	1.0
9	1.0
10	2.0
11	NaN
12	1.0
13	1.0
14	NaN
15	NaN
16	1.0
17	NaN
18	1.0
19	2.0
20	NaN
21	4.0
22	2.0
23	NaN
24	NaN
25	1.0
26	NaN
27	4.0
28	1.0
29	1.0
...	
891191	1.0
891192	1.0
891193	2.0
891194	1.0
891195	1.0
891196	1.0
891197	1.0
891198	1.0
891199	5.0
891200	2.0
891201	3.0
891202	5.0
891203	1.0
891204	1.0
891205	3.0
891206	1.0
891207	4.0
891208	1.0
891209	5.0
891210	1.0
891211	2.0
891212	5.0
891213	2.0
891214	5.0
891215	2.0
891216	1.0

```
891217    4.0
891218    1.0
891219    5.0
891220    1.0
Name: PLZ8_BAUMAX, Length: 891221, dtype: float64
```

In [ ]:

In [ ]:

In [ ]:

In [42]:
```
listdumm=list(listcols2[mymask])
```

In [43]:
```
listdumm
```

Out[43]:
```
['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',
 'VERS_TYP',
 'ZABEOTYP',
 'GEBAEUDETYP',
 'OST_WEST_KZ',
 'CAMEO_DEUG_2015',
 'CAMEO_DEU_2015']
```

In [ ]:

In [44]:
```
azdias_clean2=pd.get_dummies(azdias_clean2,columns=listdumm)
```

In [45]:
```
azdias.head(5)
```

Out[45]:

	AGER_TYP	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST
0	-1	2	1	2.0	3
1	-1	1	2	5.0	1
2	-1	3	2	3.0	1
3	2	4	2	2.0	4
4	-1	3	1	5.0	4

5 rows × 85 columns

In [46]:
```
azdias_clean2.dtypes
```

Out[46]:

ALTERSKATEGORIE_GROB	float64
FINANZ_MINIMALIST	int64
FINANZ_SPARER	int64
FINANZ_VORSORGER	int64
FINANZ_ANLEGER	int64
FINANZ_UNAUFFAELLIGER	int64
FINANZ_HAUSBAUER	int64
HEALTH_TYP	float64
LP_LEBENSPHASE_FEIN	float64
LP_LEBENSPHASE_GROB	float64
PRAEGENDE_JUGENDJAHRE	float64
RETOURTYP_BK_S	float64
SEMIO_SOZ	int64
SEMIO_FAM	int64
SEMIO_REL	int64
SEMIO_MAT	int64
SEMIO_VERT	int64
SEMIO_LUST	int64
SEMIO_ERL	int64
SEMIO_KULT	int64
SEMIO_RAT	int64
SEMIO_KRIT	int64
SEMIO_DOM	int64
SEMIO_KAEM	int64
SEMIO_PFLICHT	int64
SEMIO_TRADV	int64
ANZ_PERSONEN	float64
ANZ_TITEL	float64
HH_EINKOMMEN_SCORE	float64
W_KEIT_KIND_HH	float64
...	
CAMEO_DEU_2015_4B	uint8
CAMEO_DEU_2015_4C	uint8
CAMEO_DEU_2015_4D	uint8
CAMEO_DEU_2015_4E	uint8
CAMEO_DEU_2015_5A	uint8
CAMEO_DEU_2015_5B	uint8
CAMEO_DEU_2015_5C	uint8
CAMEO_DEU_2015_5D	uint8
CAMEO_DEU_2015_5E	uint8
CAMEO_DEU_2015_5F	uint8
CAMEO_DEU_2015_6A	uint8
CAMEO_DEU_2015_6B	uint8
CAMEO_DEU_2015_6C	uint8
CAMEO_DEU_2015_6D	uint8
CAMEO_DEU_2015_6E	uint8
CAMEO_DEU_2015_6F	uint8
CAMEO_DEU_2015_7A	uint8
CAMEO_DEU_2015_7B	uint8
CAMEO_DEU_2015_7C	uint8
CAMEO_DEU_2015_7D	uint8
CAMEO_DEU_2015_7E	uint8
CAMEO_DEU_2015_8A	uint8
CAMEO_DEU_2015_8B	uint8
CAMEO_DEU_2015_8C	uint8
CAMEO_DEU_2015_8D	uint8
CAMEO_DEU_2015_9A	uint8

```
CAMEO_DEU_2015_9B          uint8
CAMEO_DEU_2015_9C          uint8
CAMEO_DEU_2015_9D          uint8
CAMEO_DEU_2015_9E          uint8
Length: 199, dtype: object
```

In [ ]:
```
```

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

Discussion 1.2.1: Re-Encode Categorical Features

(Double-click this cell and replace this text with your own text, reporting your findings and decisions regarding categorical features. Which ones did you keep, which did you drop, and what engineering steps did you perform?)

I used pd.get_dummies () because it is a simple and straightforward technique to handle categorical features.

Step 1.2.2: Engineer Mixed-Type Features

There are a handful of features that are marked as "mixed" in the feature summary that require special treatment in order to be included in the analysis. There are two in particular that deserve attention; the handling of the rest are up to your own choices:

- "PRAEGENDE_JUGENDJAHRE" combines information on three dimensions: generation by decade, movement (mainstream vs. avantgarde), and nation (east vs. west). While there aren't enough levels to disentangle east from west, you should create two new variables to capture the other two dimensions: an interval-type variable for decade, and a binary variable for movement.
- "CAMEO_INTL_2015" combines information on two axes: wealth and life stage. Break up the two-digit codes by their 'tens'-place and 'ones'-place digits into two new ordinal variables (which, for the purposes of this project, is equivalent to just treating them as their raw numeric values).
- If you decide to keep or engineer new features around the other mixed-type features, make sure you note your steps in the Discussion section.

Be sure to check `Data_Dictionary.md` for the details needed to finish these tasks.

In [69]:
```
# Investigate "PRAEGENDE_JUGENDJAHRE" and engineer two new variables.
azdias_clean2['PRAEGENDE_JUGENDJAHRE'].unique()
```

Out[69]:
```
array([ nan,  14.,  15.,   8.,   3.,  10.,  11.,   5.,   9.,   6.,
4.,
         2.,   1.,  12.,  13.,   7.])
```

In [53]:
```
dictjugend={1:'40-60s',2:'40-60s',3:'40-60s',4:'40-60s',5:'40-60s',6:
'40-60s',7:'40-60s',8:'70-90s',9:'70-90s',10:'70-90s',11:'70-90s',12:
'70-90s',13:'70-90s',14:'70-90s',15:'70-90s'}
```

In [55]:
```
azdias_clean2.replace({'PRAEGENDE_JUGENDJAHRE':dictjugend})['PRAEGENDE_JUGENDJAHRE'].unique()
```

Out[55]:
```
array([nan, '70-90s', '40-60s'], dtype=object)
```

In [110]:
```
azdias_clean2['PRAEGENDE_new1']=azdias_clean2.replace({'PRAEGENDE_JUGENDJAHRE':dictjugend})['PRAEGENDE_JUGENDJAHRE']
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [58]:
```
dictjug_move={1:'main',2:'avant',3:'main',4:'avant',5:'main',6:'avant',7:'avant',8:'main',9:'avant',10:'main',11:'avant',12:'main',13:'avant',14:'main',15:'avant'}
```

In [59]:
```
azdias_clean2.replace({'PRAEGENDE_JUGENDJAHRE':dictjug_move})['PRAEGENDE_JUGENDJAHRE'].unique()
```

Out[59]:
```
array([nan, 'main', 'avant'], dtype=object)
```

In [111]:
```
azdias_clean2['PRAEGENDE_new2']=azdias_clean2.replace({'PRAEGENDE_JUGENDJAHRE':dictjug_move})['PRAEGENDE_JUGENDJAHRE']
```

In [113]:
```
azdias_clean2.drop(['PRAEGENDE_JUGENDJAHRE'],axis=1,inplace=True)
```

In [139]:
```
azdias_clean2=pd.get_dummies(azdias_clean2,columns=['PRAEGENDE_new1', 'PRAEGENDE_new2'])
```

In [140]:
```
azdias_clean2.shape
```

Out[140]:
```
(891221, 203)
```

In [85]:
```
import math
math.isnan(azdias_clean2['CAMEO_INTL_2015'][0])
```

Out[85]:
```
True
```

In [98]:
```
azdias_clean2.shape[0]
```

Out[98]:
```
891221
```

In [103]:
```
# Investigate "CAMEO_INTL_2015" and engineer two new variables.
feat1=[]
feat2=[]

for i in range(azdias_clean2.shape[0]):
#for i in range(100000):
    if(math.isnan(float(azdias_clean2['CAMEO_INTL_2015'][i]))):
        feat1.append(np.nan)
        feat2.append(np.nan)
    else:
        div10=int(np.round(int(azdias_clean2['CAMEO_INTL_2015'][i])/10))

        feat1.append(div10)
        feat2.append(int(int(azdias_clean2['CAMEO_INTL_2015'][i])-int(div10*10)))
```

In [104]:
```
feat1,feat2
```

Out[104]: ([nan,
5,
2,
1,
4,
5,
2,
1,
1,
2,
5,
nan,
4,
3,
nan,
4,
4,
nan,
2,
3,
2,
6,
5,
4,
nan,
3,
nan,
5,
1,
1,
nan,
5,
1,
4,
2,
nan,
4,
2,
2,
nan,
nan,
2,
2,
2,
2,
4,
nan,
2,
nan,
4,
4,
3,
4,
nan,
nan,
5,
2,

```
4,
4,
4,
5,
nan,
nan,
4,
5,
4,
5,
2,
2,
nan,
2,
4,
4,
4,
4,
nan,
nan,
4,
2,
4,
2,
nan,
4,
nan,
4,
1,
3,
2,
1,
4,
nan,
3,
2,
5,
4,
2,
5,
nan,
1,
nan,
1,
2,
2,
nan,
5,
2,
nan,
4,
nan,
nan,
5,
1,
nan,
5,
```

```
2,
2,
2,
3,
4,
5,
1,
2,
2,
1,
2,
3,
6,
5,
4,
4,
4,
3,
1,
nan,
5,
4,
3,
nan,
nan,
4,
5,
4,
nan,
nan,
nan,
4,
5,
2,
2,
2,
5,
5,
4,
nan,
nan,
3,
2,
2,
4,
4,
3,
2,
2,
nan,
5,
nan,
5,
2,
nan,
2,
5,
```

```
2,
1,
nan,
1,
nan,
nan,
5,
nan,
nan,
2,
5,
6,
2,
3,
2,
5,
5,
2,
5,
4,
4,
4,
5,
4,
5,
3,
1,
2,
nan,
5,
4,
5,
4,
5,
nan,
2,
2,
2,
nan,
nan,
6,
2,
4,
6,
2,
4,
5,
5,
4,
4,
5,
5,
5,
4,
4,
5,
nan,
```

```
nan,
nan,
nan,
1,
5,
nan,
5,
2,
nan,
nan,
5,
nan,
4,
1,
1,
2,
nan,
3,
4,
5,
nan,
5,
nan,
5,
4,
4,
4,
2,
2,
3,
2,
6,
1,
6,
2,
2,
4,
2,
2,
1,
2,
1,
1,
4,
1,
4,
3,
2,
5,
4,
2,
2,
5,
1,
1,
1,
2,
```

```
2,
nan,
2,
nan,
5,
2,
5,
4,
2,
5,
5,
2,
5,
5,
2,
2,
5,
2,
2,
2,
2,
1,
6,
4,
1,
2,
4,
3,
4,
1,
2,
3,
4,
4,
4,
2,
6,
3,
4,
nan,
2,
6,
5,
5,
4,
1,
2,
5,
4,
2,
4,
nan,
3,
3,
nan,
2,
2,
```

```
nan,
6,
4,
2,
3,
nan,
1,
4,
5,
nan,
3,
1,
4,
2,
5,
3,
1,
1,
2,
5,
2,
4,
4,
1,
2,
4,
2,
4,
5,
1,
1,
2,
1,
1,
2,
2,
nan,
5,
2,
2,
2,
2,
2,
2,
5,
2,
nan,
nan,
4,
2,
4,
4,
2,
2,
nan,
2,
2,
```

```
nan,
4,
2,
2,
2,
4,
2,
5,
5,
4,
nan,
nan,
nan,
5,
2,
nan,
1,
1,
2,
1,
2,
4,
1,
2,
4,
1,
4,
5,
4,
4,
1,
nan,
3,
4,
4,
6,
3,
5,
nan,
4,
nan,
2,
4,
2,
5,
5,
4,
4,
5,
4,
5,
1,
5,
5,
6,
5,
5,
```

```
4,
2,
5,
4,
4,
6,
5,
2,
2,
1,
2,
1,
1,
2,
2,
1,
1,
5,
4,
2,
nan,
4,
5,
2,
5,
2,
5,
1,
nan,
3,
5,
1,
5,
5,
5,
3,
4,
2,
5,
2,
4,
nan,
1,
5,
1,
5,
1,
2,
5,
4,
4,
2,
4,
nan,
2,
2,
4,
```

```
5,
2,
3,
2,
4,
1,
5,
5,
5,
4,
nan,
nan,
5,
1,
nan,
5,
nan,
1,
4,
nan,
nan,
5,
5,
2,
2,
5,
5,
3,
1,
nan,
1,
4,
5,
4,
2,
nan,
2,
4,
5,
5,
5,
5,
5,
2,
nan,
4,
6,
1,
2,
2,
5,
nan,
nan,
5,
nan,
3,
nan,
```

```
2,
2,
3,
nan,
1,
1,
5,
2,
nan,
nan,
4,
nan,
nan,
nan,
nan,
nan,
5,
4,
nan,
nan,
nan,
nan,
5,
2,
2,
nan,
2,
5,
nan,
nan,
4,
2,
nan,
2,
nan,
5,
nan,
1,
2,
5,
4,
2,
2,
nan,
2,
nan,
nan,
2,
nan,
4,
nan,
nan,
4,
5,
nan,
5,
nan,
```

```
4,
5,
5,
2,
4,
nan,
2,
4,
2,
2,
5,
4,
nan,
2,
2,
2,
5,
4,
3,
1,
5,
5,
nan,
2,
1,
5,
5,
2,
5,
2,
4,
nan,
4,
4,
4,
5,
5,
4,
3,
2,
4,
nan,
2,
2,
2,
2,
5,
1,
2,
nan,
5,
4,
5,
4,
nan,
5,
4,
```

```
4,
2,
2,
2,
5,
4,
2,
2,
4,
nan,
2,
5,
4,
6,
6,
4,
5,
5,
3,
2,
4,
4,
4,
3,
5,
5,
5,
4,
1,
3,
2,
2,
1,
3,
5,
2,
2,
nan,
4,
2,
4,
3,
5,
1,
4,
6,
1,
5,
2,
2,
5,
2,
2,
2,
4,
2,
3,
```

```
nan,
4,
5,
nan,
nan,
nan,
4,
2,
2,
1,
4,
2,
5,
nan,
2,
2,
2,
5,
nan,
2,
2,
4,
1,
1,
3,
2,
4,
2,
nan,
5,
4,
4,
5,
4,
2,
2,
4,
4,
2,
2,
nan,
2,
nan,
nan,
2,
nan,
5,
5,
2,
5,
nan,
nan,
5,
nan,
5,
nan,
4,
```

```
4,
2,
nan,
4,
nan,
4,
6,
nan,
nan,
nan,
5,
1,
1,
4,
nan,
nan,
nan,
2,
nan,
3,
4,
4,
5,
5,
5,
6,
4,
2,
2,
4,
2,
5,
4,
2,
nan,
4,
1,
3,
5,
4,
5,
5,
2,
2,
1,
4,
2,
3,
2,
2,
2,
2,
2,
1,
1,
4,
2,
```

```
4,
2,
3,
4,
2,
4,
2,
2,
5,
2,
4,
4,
4,
2,
2,
4,
1,
5,
nan,
nan,
5,
4,
nan,
1,
nan,
1,
5,
4,
4,
nan,
2,
3,
5,
nan,
5,
4,
1,
2,
4,
1,
5,
4,
1,
1,
4,
2,
2,
nan,
nan,
4,
4,
1,
4,
2,
5,
4,
4,
```

```
4,
5,
5,
4,
4,
5,
5,
4,
nan,
5,
4,
1,
3,
5,
5,
2,
nan,
1,
4,
5,
2,
nan,
2,
4,
4,
2,
4,
4,
4,
2,
1,
2,
1,
3,
2,
4,
2,
1,
2,
1,
5,
1,
2,
5,
5,
4,
2,
4,
4,
3,
4,
1,
4,
2,
3,
5,
4,
```

```
 3,
 4,
 4,
 6,
 2,
 3,
 3,
 5,
 5,
 5,
 4,
 5,
 4,
 3,
 5,
 2,
 3,
 4,
 5,
 4,
 2,
 4,
 nan,
 6,
 2,
 6,
 2,
 6,
 4,
 5,
 nan,
 ...],
[nan,
 1,
 4,
 2,
 3,
 4,
 2,
 4,
 3,
 -5,
 1,
 nan,
 3,
 3,
 nan,
 1,
 1,
 nan,
 4,
 4,
 4,
 -5,
 1,
 3,
 nan,
```

```
3,
nan,
1,
3,
2,
nan,
1,
4,
1,
5,
nan,
1,
4,
5,
nan,
nan,
5,
3,
5,
4,
1,
nan,
5,
nan,
3,
3,
1,
1,
nan,
nan,
1,
2,
1,
3,
1,
1,
nan,
nan,
1,
2,
3,
2,
5,
5,
nan,
2,
1,
-5,
1,
1,
nan,
nan,
1,
5,
1,
5,
nan,
```

```
5,
nan,
-5,
4,
3,
-5,
4,
1,
nan,
3,
4,
4,
3,
4,
4,
nan,
3,
nan,
4,
4,
5,
nan,
1,
4,
nan,
1,
nan,
nan,
1,
3,
nan,
4,
5,
4,
5,
4,
1,
1,
2,
2,
2,
4,
-5,
4,
-5,
1,
1,
4,
1,
3,
4,
nan,
1,
1,
1,
nan,
nan,
```

```
1,
1,
3,
nan,
nan,
nan,
1,
1,
5,
4,
4,
4,
4,
1,
nan,
nan,
2,
-5,
4,
3,
4,
3,
4,
4,
nan,
4,
nan,
4,
4,
nan,
3,
1,
2,
3,
nan,
3,
nan,
nan,
1,
nan,
nan,
4,
1,
-5,
4,
4,
4,
1,
1,
5,
1,
3,
1,
5,
1,
1,
1,
```

```
1,
3,
5,
nan,
4,
3,
1,
5,
1,
nan,
-5,
-5,
5,
nan,
nan,
-5,
5,
4,
-5,
5,
4,
1,
1,
3,
1,
1,
4,
1,
1,
4,
1,
nan,
nan,
nan,
nan,
3,
1,
nan,
1,
2,
nan,
nan,
4,
nan,
4,
4,
3,
5,
nan,
4,
3,
1,
nan,
1,
nan,
4,
3,
```

```
3,
1,
4,
-5,
3,
4,
-5,
4,
-5,
3,
-5,
3,
4,
5,
2,
4,
3,
3,
1,
2,
1,
1,
4,
1,
1,
4,
2,
1,
3,
2,
3,
4,
2,
nan,
4,
nan,
4,
4,
4,
3,
4,
4,
4,
3,
1,
2,
4,
3,
1,
4,
4,
4,
4,
3,
-5,
3,
3,
```

```
4,
3,
2,
1,
4,
2,
4,
1,
5,
1,
5,
-5,
4,
5,
nan,
3,
-5,
1,
1,
1,
4,
4,
1,
1,
4,
1,
nan,
4,
2,
nan,
4,
-5,
nan,
-5,
1,
2,
4,
nan,
4,
-5,
2,
nan,
4,
2,
1,
5,
1,
3,
4,
4,
-5,
4,
-5,
1,
1,
4,
5,
```

```
4,
2,
1,
4,
2,
3,
2,
2,
3,
2,
3,
nan,
1,
3,
4,
4,
2,
3,
4,
2,
2,
nan,
nan,
3,
2,
4,
3,
4,
2,
nan,
4,
3,
nan,
-5,
5,
4,
5,
3,
3,
1,
1,
1,
nan,
nan,
nan,
1,
4,
nan,
4,
4,
4,
4,
4,
1,
4,
5,
1,
```

```
4,
3,
1,
3,
3,
4,
nan,
1,
1,
1,
-5,
1,
1,
nan,
3,
nan,
4,
1,
3,
4,
1,
3,
4,
1,
3,
2,
4,
1,
1,
-5,
1,
1,
3,
2,
4,
1,
-5,
-5,
1,
3,
-5,
4,
4,
4,
4,
-5,
4,
2,
3,
1,
1,
5,
nan,
1,
1,
5,
1,
```

```
4,
1,
4,
nan,
3,
1,
4,
1,
1,
1,
1,
1,
4,
1,
2,
1,
nan,
4,
1,
4,
1,
4,
2,
1,
3,
3,
4,
1,
nan,
5,
2,
4,
1,
3,
2,
4,
4,
4,
4,
4,
1,
1,
nan,
nan,
1,
4,
nan,
1,
nan,
3,
3,
nan,
nan,
1,
1,
4,
5,
```

```
1,
4,
1,
3,
nan,
4,
3,
1,
-5,
5,
nan,
2,
1,
1,
4,
1,
1,
2,
5,
nan,
1,
-5,
4,
4,
3,
1,
nan,
nan,
1,
nan,
3,
nan,
4,
4,
4,
nan,
4,
2,
4,
2,
nan,
nan,
5,
nan,
nan,
nan,
nan,
nan,
4,
1,
nan,
nan,
nan,
nan,
1,
2,
5,
```

```
nan,
3,
1,
nan,
nan,
1,
3,
nan,
4,
nan,
1,
nan,
3,
4,
2,
-5,
2,
2,
nan,
4,
nan,
nan,
2,
nan,
3,
nan,
nan,
-5,
1,
nan,
1,
nan,
1,
4,
2,
4,
1,
nan,
5,
3,
3,
4,
1,
1,
nan,
4,
4,
3,
1,
1,
1,
4,
1,
4,
nan,
4,
3,
```

```
1,
1,
4,
1,
2,
3,
nan,
4,
1,
3,
1,
1,
1,
4,
4,
4,
nan,
2,
2,
-5,
2,
4,
3,
5,
nan,
4,
1,
1,
3,
nan,
1,
-5,
1,
5,
2,
4,
4,
1,
5,
4,
3,
nan,
5,
2,
4,
-5,
-5,
1,
1,
1,
1,
3,
-5,
4,
1,
3,
2,
```

```
1,
1,
1,
4,
4,
5,
3,
4,
3,
4,
5,
5,
nan,
1,
4,
1,
4,
1,
4,
3,
-5,
4,
4,
5,
4,
1,
2,
4,
4,
-5,
5,
1,
nan,
1,
4,
nan,
nan,
nan,
3,
4,
4,
2,
1,
4,
1,
nan,
4,
4,
4,
1,
nan,
5,
4,
5,
4,
3,
2,
```

```
5,
3,
2,
nan,
1,
1,
3,
1,
-5,
4,
4,
1,
1,
5,
4,
nan,
5,
nan,
nan,
5,
nan,
1,
1,
3,
1,
nan,
nan,
2,
nan,
1,
nan,
1,
3,
-5,
nan,
1,
nan,
1,
-5,
nan,
nan,
nan,
1,
4,
2,
-5,
nan,
nan,
nan,
5,
nan,
3,
1,
-5,
1,
2,
1,
```

```
-5,
3,
4,
2,
1,
4,
2,
5,
4,
nan,
3,
2,
2,
4,
3,
1,
1,
4,
3,
4,
1,
4,
4,
4,
4,
4,
4,
2,
4,
4,
3,
4,
1,
-5,
4,
1,
4,
1,
4,
2,
1,
4,
1,
5,
1,
4,
4,
1,
3,
1,
nan,
nan,
4,
3,
nan,
4,
nan,
```

```
4,
1,
1,
3,
nan,
2,
1,
1,
nan,
4,
3,
4,
-5,
3,
3,
4,
1,
4,
4,
5,
4,
2,
nan,
nan,
3,
1,
3,
1,
5,
1,
5,
5,
1,
4,
1,
1,
1,
4,
1,
1,
nan,
1,
1,
3,
4,
1,
4,
5,
nan,
4,
-5,
2,
2,
nan,
5,
5,
3,
```

```
-5,
1,
3,
3,
2,
4,
4,
4,
2,
2,
1,
4,
4,
4,
4,
1,
2,
2,
1,
1,
1,
5,
3,
3,
3,
3,
3,
1,
3,
4,
2,
4,
4,
-5,
3,
-5,
4,
3,
1,
1,
1,
1,
1,
2,
1,
1,
2,
3,
1,
1,
1,
3,
2,
3,
nan,
-5,
-5,
```

```
-5,
4,
-5,
1,
1,
nan,
...])
```

In [107]:
```
newfeat=pd.DataFrame(list(zip(feat1,feat2)),columns=['CAMEO_D1','CAME
O_D2'])
```

In [116]:
```
newfeat.head()
```

Out[116]:

	CAMEO_D1	CAMEO_D2
0	NaN	NaN
1	5.0	1.0
2	2.0	4.0
3	1.0	2.0
4	4.0	3.0

In [115]:
```
#azdias_clean2
```

In [125]:
```
#result = azdias_clean2.append(newfeat, ignore_index=False, sort=Fals
e)

azdias_clean2['CAMEO_D1']=feat1
```

In [126]:
```
azdias_clean2['CAMEO_D2']=feat2
```

In [127]:
```
azdias_clean2.head(5)
```

Out[127]:

	ALTERSKATEGORIE_GROB	FINANZ_MINIMALIST	FINANZ_SPARER	FINANZ_VORSORGER	FI
0	2.0	3	4	3	
1	1.0	1	5	2	
2	3.0	1	4	1	
3	4.0	4	2	5	
4	3.0	4	3	4	

5 rows × 202 columns

In [128]:
```
azdias_clean2.shape
```

Out[128]:
```
(891221, 202)
```

In [131]:
```
azdias_clean2.drop(['CAMEO_INTL_2015'],axis=1,inplace=True)
```

In [132]:
```
azdias_clean2.shape
```

Out[132]:
```
(891221, 201)
```

In [ ]:

In [ ]:

In [ ]:

Discussion 1.2.2: Engineer Mixed-Type Features

(Double-click this cell and replace this text with your own text, reporting your findings and decisions regarding mixed-value features. Which ones did you keep, which did you drop, and what engineering steps did you perform?)

I engineered the features as proposed in the text, by keeping in mind the dictionary .md given for consulting, and dropped the original columns.

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 [141]:
```
# If there are other re-engineering tasks you need to perform, make s
ure you
# take care of them here. (Dealing with missing data will come in ste
p 2.1.)
azdias_clean2.dtypes.unique()
```

Out[141]:
```
array([dtype('float64'), dtype('int64'), dtype('uint8')], dtype=objec
t)
```

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

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 [2]:
```
import math

def clean_data(df):
    """
    Perform feature trimming, re-encoding, and engineering for demogr
aphics
    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, ...

    df_clean=df.copy()
    listcols=df_clean.columns
    for feat in range(feat_info.shape[0]):
    #for featttt in range(1):
    #    feat=57
        missings=feat_info['missing_or_unknown'][feat].split('[')[1].
split(']')[0].split(',')
        misslist=[]
        misslist2=[]
    #    print(missings,len(missings))
        for i in range(len(missings)):
    #        if( isinstance(int(missings[i]),int)):
            if( missings[i]!='' and missings[i]!='X' and missings[i]!
='XX'):
                misslist.append(int(missings[i]))
            else:
                misslist.append(missings[i])
    #    print(misslist)
    #            continue
    #    if(misslist!=[]):
        colname=listcols[feat]
    #    print(colname)
        df_clean[colname].replace(misslist,np.nan,inplace=True)


    # remove selected columns and rows, ...
    nulls=df_clean.isnull().sum()
    remcols=[]
    for i in range(len(nulls)):
        if(nulls[i]>=200000):
            remcols.append(i)
        else:
            pass
    df_clean2=df_clean.copy()
    listcols=df_clean2.columns
    for i in range(len(remcols)):
    #    print(listcols[remcols[i]])
        col_drop=listcols[remcols[i]]
        df_clean2.drop(col_drop,axis=1,inplace=True)

    feat_info2=feat_info.copy()
```

```
    feat_info2.drop(feat_info2.index[remcols],inplace=True)

    feat_info2=feat_info2.reset_index()
    feat_info2.drop('index',axis=1,inplace=True)

    # select, re-encode, and engineer column values.
    cats=(feat_info2['type']=='categorical')
    mymask=[]
    for i in range(len(cats)):
        mymask.append(cats[i])
    typp=df_clean2.dtypes
    typ=[]
    listcols2=df_clean2.columns

    for i in range(len(typp)):
        typ.append(typp[i])
    listdumm=list(listcols2[mymask])
    df_clean2=pd.get_dummies(df_clean2,columns=listdumm)

    dictjugend={1:'40-60s',2:'40-60s',3:'40-60s',4:'40-60s',5:'40-60
s',6:'40-60s',7:'40-60s',8:'70-90s',9:'70-90s',10:'70-90s',11:'70-90
s',12:'70-90s',13:'70-90s',14:'70-90s',15:'70-90s'}
    df_clean2['PRAEGENDE_new1']=df_clean2.replace({'PRAEGENDE_JUGENDJ
AHRE':dictjugend})['PRAEGENDE_JUGENDJAHRE']
    dictjug_move={1:'main',2:'avant',3:'main',4:'avant',5:'main',6:'a
vant',7:'avant',8:'main',9:'avant',10:'main',11:'avant',12:'main',13:
'avant',14:'main',15:'avant'}
    df_clean2['PRAEGENDE_new2']=df_clean2.replace({'PRAEGENDE_JUGENDJ
AHRE':dictjug_move})['PRAEGENDE_JUGENDJAHRE']
    df_clean2.drop(['PRAEGENDE_JUGENDJAHRE'],axis=1,inplace=True)
    df_clean2=pd.get_dummies(df_clean2,columns=['PRAEGENDE_new1','PRA
EGENDE_new2'])

    feat1=[]
    feat2=[]

    for i in range(df_clean2.shape[0]):
    #for i in range(100000):
        if(math.isnan(float(df_clean2['CAMEO_INTL_2015'][i]))):
            feat1.append(np.nan)
            feat2.append(np.nan)
        else:
            div10=int(np.round(int(df_clean2['CAMEO_INTL_2015'][i])/1
0))

            feat1.append(div10)
            feat2.append(int(int(df_clean2['CAMEO_INTL_2015'][i])-int
(div10*10)))

    df_clean2['CAMEO_D1']=feat1
    df_clean2['CAMEO_D2']=feat2
    df_clean2.drop(['CAMEO_INTL_2015'],axis=1,inplace=True)
    # Return the cleaned dataframe.

    return df_clean2
```

In [3]:
```
# Load in the general demographics data.
azdias = pd.read_csv('Udacity_AZDIAS_Subset.csv',delimiter=';')

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

In [4]:
```
newdf=clean_data(azdias)
```

In [ ]:

In [ ]:

In [5]:
```
newdf.dtypes.unique()
```

Out[5]:
```
array([dtype('float64'), dtype('int64'), dtype('uint8')], dtype=objec
t)
```

In [ ]:

Step 2: Feature Transformation

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 (http://scikit-learn.org/stable/modules/classes.html) 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 (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Imputer.html) 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 (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html) 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 [6]:
```
# If you've not yet cleaned the dataset of all NaN values, then inves
tigate and
# do that now.
#newdf2=newdf.fillna(newdf.mean())
newdf2=newdf.dropna(axis='columns')
```

In [7]:
```
newdf2.shape
```

Out[7]: (891221, 162)

In [ ]:

In [8]:
```
newdf2.isnull().sum().sum()
```

Out[8]: 0

In [8]:

In [9]:

In [10]:
```
#demodf.head(5)
```

Out[10]:

	ALTERSKATEGORIE_GROB	FINANZ_MINIMALIST	FINANZ_SPARER	FINANZ_VORSORGER	FI
0	4.0	5	1	5	
1	4.0	5	1	5	
2	4.0	5	1	5	
3	4.0	5	1	5	
4	3.0	3	1	4	

5 rows × 218 columns

In [24]:
```
#demodf2=demodf.fillna(demodf.mean())
demo = pd.read_csv('Udacity_CUSTOMERS_Subset.csv',delimiter=';')
demodf=clean_data(demo)
demodf2=demodf.dropna(axis='columns')
```

In [25]:
```
demodf2.isnull().sum().sum()
```

Out[25]: 0

In [27]:
```
demodf2.shape
```

Out[27]: (191652, 174)

In [9]:
```
# Apply feature scaling to the general population demographics data.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
newdf2_scaled=scaler.fit_transform(newdf2)
```

In [10]:
```
#demodf2_scaled=scaler.fit_transform(demodf2)
```

In [11]:
```
newdf2_scaled
```

Out[11]:
```
array([[-0.05641562,  0.80489015, -0.30337778, ..., -1.30834687,
        -0.49470094, -1.46453679],
       [-1.57035806,  1.48760145, -1.0597311 , ...,  0.7643233 ,
        -0.49470094,  0.68280975],
       [-1.57035806,  0.80489015, -1.81608443, ...,  0.7643233 ,
         2.02142328, -1.46453679],
       ...,
       [-0.81338684,  0.80489015, -1.0597311 , ...,  0.7643233 ,
        -0.49470094,  0.68280975],
       [-1.57035806,  1.48760145, -0.30337778, ...,  0.7643233 ,
        -0.49470094,  0.68280975],
       [ 0.7005556 , -0.56053245,  1.20932887, ..., -1.30834687,
        -0.49470094,  0.68280975]])
```

Discussion 2.1: Apply Feature Scaling

(Double-click this cell and replace this text with your own text, reporting your decisions regarding feature scaling.)

I applied feature scaling in the data because otherwise the subsequent unsupervised learning would take too long, and probably yield wrong results. Feature Scaling is necessary because some columns vary significantly concerning their minimum and maximum values. Standardization is a good choice, since it is not so sensitive to outliers, as MinMaxScaler().

Step 2.2: Perform Dimensionality Reduction

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

- Use sklearn's PCA (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html) 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()` (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html) 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 [12]:
```
# Apply PCA to the data.
from sklearn.decomposition import PCA
pca_azdias = PCA()
pca_azdias.fit(newdf2_scaled)
```

Out[12]:
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state
=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

Let us choose all the components that accumulated retain 80% of the information

In [13]:
```
# Investigate the variance accounted for by each principal component.
#print(pca_azdias.explained_variance_ratio_)
pca_var=pca_azdias.explained_variance_ratio_

sum_var=0.
for i in range(len(pca_var)):
    sum_var+=pca_var[i]
    if(sum_var>=0.8):
        print(i)
        break
```

```
74
```

In [ ]:
```

```

In [12]:
```
# Re-apply PCA to the data while selecting for number of components t
o retain.

from sklearn.decomposition import PCA
pca_azdias_new = PCA(n_components=75)
pca_azdias_new.fit(newdf2_scaled)
```

Out[12]:
```
PCA(copy=True, iterated_power='auto', n_components=75, random_state=N
one,
  svd_solver='auto', tol=0.0, whiten=False)
```

In [13]:
```
newdf2_scaled_pca = pca_azdias_new.transform(newdf2_scaled)
```

In [14]:
```
newdf2_scaled_pca.shape
```

Out[14]:
```
(891221, 75)
```

In [15]:
```
len(pca_azdias.components_)
```

```
---------------------------------------------------------------------
------
NameError                                 Traceback (most recent call
last)
<ipython-input-15-7a227caebee9> in <module>()
----> 1 len(pca_azdias.components_)

NameError: name 'pca_azdias' is not defined
```

In [ ]:
```

```

Discussion 2.2: Perform Dimensionality Reduction

(Double-click this cell and replace this text with your own text, reporting your findings and decisions regarding dimensionality reduction. How many principal components / transformed features are you retaining for the next step of the analysis?)

I am aiming to retain 80% of the total information by using PCA. Therefore, as the above analysis has shown, I am using 75 components, which is quite a significant dimensionality reduction!

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 [ ]:
```
# Map weights for the first principal component to corresponding feat
ure names
# and then print the linked values, sorted by weight.
# HINT: Try defining a function here or in a new cell that you can re
use in the
# other cells.
```

In [16]:
```
len(pca_azdias_new.components_)
```

Out[16]: 75

In [25]:
```
pca_azdias_new.components_
```

Out[25]:
```
array([[ 0.22357424, -0.27164022,  0.23528212, ..., -0.12763652,
         0.12189683, -0.04516001],
       [ 0.103803  ,  0.03466574, -0.06272222, ...,  0.12044665,
         0.12141165, -0.02808014],
       [ 0.0993706 ,  0.06231506, -0.0580409 , ..., -0.0410165 ,
         0.19353698, -0.28241041],
       ...,
       [-0.00487659,  0.01842813,  0.03308872, ...,  0.02786805,
         0.00046612, -0.00860925],
       [ 0.00497581, -0.00467635, -0.02902589, ...,  0.00830241,
         0.01141076,  0.00656678],
       [-0.02378926,  0.02153713, -0.04271507, ...,  0.02694344,
         0.04423559, -0.02549927]])
```

In [17]:
```
n_pcs=pca_azdias_new.components_.shape[0]
```

In [18]:
```
n_pcs
```

Out[18]:
```
75
```

In [22]:
```
# Let us do the sort of the first three components
```

In [39]:
```
#first_comp_pca=[np.abs(pca_azdias_new.components_[i]).argsort() for
 i in range(n_pcs)]
arr1=np.abs(pca_azdias_new.components_[0]).argsort()
first_comp_pca1=pca_azdias_new.components_[0][arr1[::-1]]
feat_names1=np.array(newdf2.columns)[arr1[::-1]]
```

In [44]:
```
np.array(feat_names1)[:3],first_comp_pca1[:3]
```

Out[44]:
```
(array(['FINANZ_SPARER', 'FINANZ_VORSORGER', 'FINANZ_ANLEGER'], dtype
=object),
 array([-0.27164022,  0.23528212, -0.23166812]))
```

The three main components are "FINANZ_SPARER", "FINANZ_VORSOGER" AND "FINANZ_ANLEGER". The first one has the same trend as the third one (same sign), and different trend than the second one (opposite sign). The magnitude of them are similar

Creating a Function

In [48]:
```
def main_pca_comps(pca_df,df_old,ii):
    arr1=np.abs(pca_df.components_[ii]).argsort()
    first_comp_pca1=pca_df.components_[ii][arr1[::-1]]
    feat_names1=np.array(df_old.columns)[arr1[::-1]]
    print(np.array(feat_names1)[:3],first_comp_pca1[:3])
```

For the second component

In [49]:
```
main_pca_comps(pca_azdias_new,newdf2,1)
```

```
['SEMIO_KAEM' 'SEMIO_KULT' 'SEMIO_VERT'] [-0.28288862  0.27533711  0.
27283231]
```

The analysis is very similar to the first component. The first feature is "SEMIO_KAEM", the second "SEMIO_KULT" and the third "SEMIO_VERT". The first one has a different trend than others (opposite signs). The second and the third are quite similar (same sign). The magnitude of the components are quite similar

In [ ]:

In [50]:
```
main_pca_comps(pca_azdias_new,newdf2,2)
```

```
['PRAEGENDE_new2_main' 'LP_STATUS_GROB_1.0' 'LP_STATUS_FEIN_1.0'] [-
0.28241041 -0.22992586 -0.20702658]
```

Again, not so different than the others. The first one is 'PRAEGENDE_new2_main', the second one is 'LP_STATUS_GROB_1.0' and the third one is 'LP_STATUS_FEIN_1.0'. All of them have the same direction, and similar magnitudes, however the first one is significantly stronger than the others.

In [ ]:

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

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

Discussion 2.3: Interpret Principal Components

(Double-click this cell and replace this text with your own text, reporting your observations from detailed investigation of the first few principal components generated. Can we interpret positive and negative values from them in a meaningful way?)

The short discussions have been presented above.

Step 3: Clustering

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.

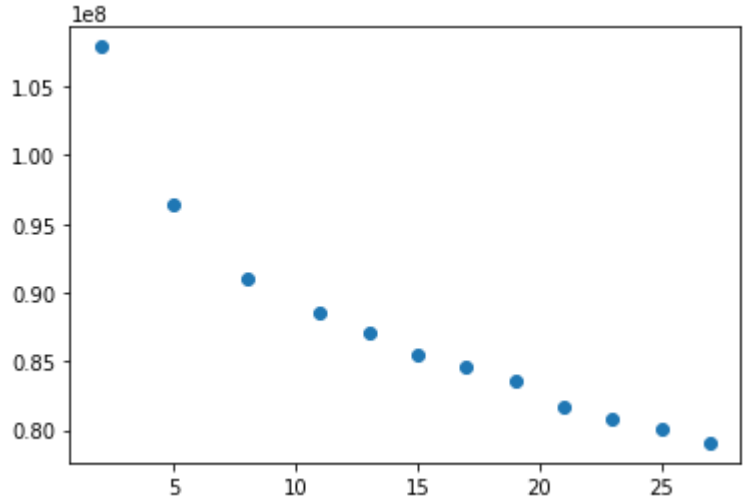
- Use sklearn's KMeans (http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans) class to perform k-means clustering on the PCA-transformed data.
- Then, compute the average difference from each point to its assigned cluster's center. **Hint**: The KMeans object's `.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 [24]:
```
# Over a number of different cluster counts...


    # run k-means clustering on the data and...


    # compute the average within-cluster distances.

from sklearn.cluster import KMeans
#from sklearn.metrics import silhouette_samples, silhouette_score
#cluster_labels = KMeans(n_clusters=20, random_state=0).fit_predict(n
ewdf2_scaled_pca)
#silhouette_avg = silhouette_score(newdf2_scaled_pca, cluster_labels,
sample_size=10000)
scores_km=[]
list_km=[2,5,8,11,13,15,17,19,21,23,25,27]
for i in range(len(list_km)):
    km=KMeans(n_clusters=list_km[i], random_state=0)
    model=km.fit(newdf2_scaled_pca)
    score=np.abs(model.score(newdf2_scaled_pca))
    scores_km.append(score)
```

In [25]:
```
#silhouette_avg = silhouette_score(newdf2_scaled_pca, cluster_labels,
sample_size=10000)
#score
```

In [26]:
```
#silhouette_avg
```

In [27]:
```
# Investigate the change in within-cluster distance across number of
 clusters.
# HINT: Use matplotlib's plot function to visualize this relationshi
p.
import matplotlib.pyplot as plt
plt.scatter(list_km,scores_km)

plt.show()
```



In [ ]:

In [ ]:

In [ ]:

21 seems to be the optimum number of clusters

In [ ]:

In [ ]:

In [ ]:

In [53]:
```
# Re-fit the k-means model with the selected number of clusters and o
btain
# cluster predictions for the general population demographics data.

from sklearn.cluster import KMeans

km_opt=KMeans(n_clusters=21, random_state=0)
model=km_opt.fit(newdf2_scaled_pca)
```

In [54]:
```
pred_opt=km_opt.predict(newdf2_scaled_pca)
```

In [55]:
```
pred_opt
```

Out[55]:
```
array([ 3, 13, 11, ..., 11,  8,  5], dtype=int32)
```

Discussion 3.1: Apply Clustering to General Population

(Double-click this cell and replace this text with your own text, reporting your findings and decisions regarding clustering. Into how many clusters have you decided to segment the population?)

By using the Elbow method of KMeans, I have found an (nearly) optimal number of clusters as to be 21.

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 [56]:
```
# Load in the customer demographics data.
customers = pd.read_csv('Udacity_CUSTOMERS_Subset.csv',delimiter=';')
custdf=clean_data(customers)
custdf2=custdf.dropna(axis='columns')
custdf2=custdf2.iloc[:,:162]
custdf2_scaled=scaler.fit_transform(custdf2)
```

In [ ]:
```

```

In [57]:
```
custdf2_scaled_pca = pca_azdias_new.transform(custdf2_scaled)
```

In [58]:
```
# Apply preprocessing, feature transformation, and clustering from th
e general
# demographics onto the customer data, obtaining cluster predictions
 for the
# customer demographics data.

from sklearn.cluster import KMeans

pred_opt_cust=km_opt.predict(custdf2_scaled_pca)
```

In [59]:
```
(pd.Series(pred_opt_cust)).value_counts()
```

Out[59]:
```
11    53393
19    43548
2     23241
17    12580
10    10755
5      9039
20     6742
4      5669
8      5163
12     3853
15     3362
18     2575
6      2201
1      1885
7      1802
9      1701
16     1649
0      1385
3       955
14      137
13       17
dtype: int64
```

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.

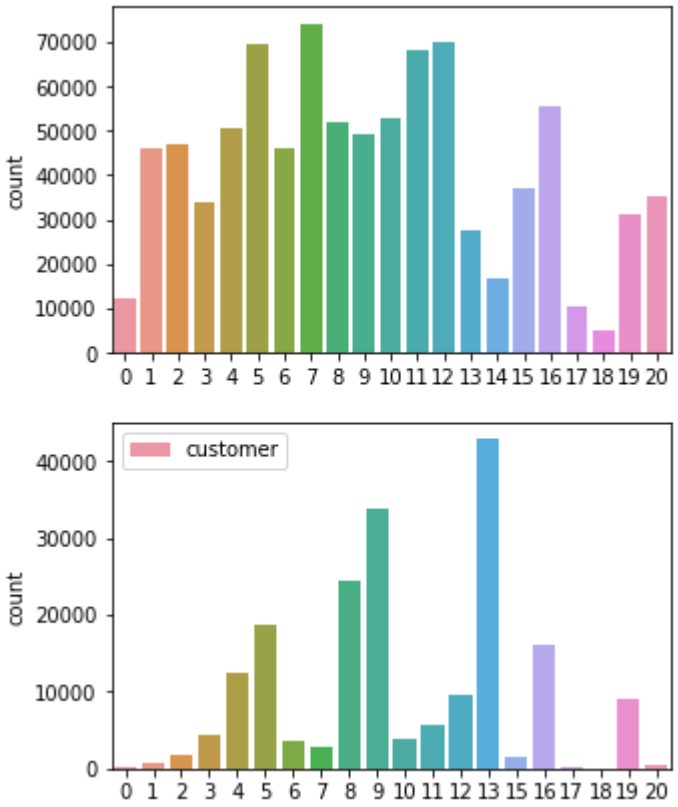
Take a look at the following points in this step:

- 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()` (https://seaborn.pydata.org/generated/seaborn.countplot.html) or `barplot()` (https://seaborn.pydata.org/generated/seaborn.barplot.html) 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 [79]:
```
pred_opt
```

Out[79]:
```
array([13, 14,  4, ...,  4, 11,  9], dtype=int32)
```

In [87]:
```
# Compare the proportion of data in each cluster for the customer dat
a to the
# proportion of data in each cluster for the general population.
import seaborn as sns
fig,ax=plt.subplots(2,1,figsize=[5,7])
sns.countplot(pred_opt,ax=ax[0],label='population')
#plt.legend()

sns.countplot(pred_opt_cust,label='customer',ax=ax[1])

plt.legend()

plt.show()
```



In [ ]:

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

It seems that 13 and 10 are over-represented

In [66]:
```
main_pca_comps(pca_azdias_new,custdf2,13)
```

```
['KK_KUNDENTYP_5.0' 'GEBAEUDETYP_3.0' 'KK_KUNDENTYP_4.0'] [ 0.4385435
6 -0.39246878 -0.28034239]
```

Kundentyp-> customer pattern over the past 12 months

Gebaudetyp-> Type of building (residential vs. commercial)

In [63]:
```
main_pca_comps(pca_azdias_new,custdf2,10)
```

```
['LP_FAMILIE_GROB_1.0' 'SHOPPER_TYP_3.0' 'SOHO_KZ_0.0'] [ 0.31357066
-0.27943721  0.2786269 ]
```

familie_grob-> Family type

shopper_typ-> Shopper typology

SOHO_KZ -> Small office / home office flag

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

6 and 7 are under-represented

In [64]:
```
main_pca_comps(pca_azdias_new,custdf2,6)
```

```
['LP_STATUS_FEIN_9.0' 'LP_FAMILIE_GROB_5.0' 'LP_STATUS_FEIN_10.0'] [
0.33578698  0.26050046  0.25252545]
```

LP_STATUS_FEIN -> Social status, fine scale

LP_STATUS_GROB -> Social status, rough scale

LP_FAMILIE_GROB -> Family type, rough scale

In [65]:
```
main_pca_comps(pca_azdias_new,custdf2,7)
```

```
['LP_STATUS_FEIN_10.0' 'LP_STATUS_FEIN_3.0' 'ZABEOTYP_2'] [ 0.3987417
3  0.36669405  0.26625791]
```

LP_STATUS_FEIN -> Social status, fine scale

ZABEOTYP -> Energy consumption typology

Discussion 3.3: Compare Customer Data to Demographics Data

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

yes. Popular (mainly): pattern of clients,type of building and family type. Unpopular (mainly): social status, and energy comsuption typology

> 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 [ ]: