Project: Identify Customer Segments

In [1]:
```
# Imports libraries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings("ignore")

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

Step 0: Load the Data

In [2]:
```
# Loads in the general demographics data and create a copy.
azdias2 = pd.read_csv ('Udacity_AZDIAS_Subset.csv', sep = ';')
azdias = azdias2.copy()

# Loads in the feature summary file and create a copy.
feat_info2 = pd.read_csv ('AZDIAS_Feature_Summary.csv', sep = ';')
feat_info = feat_info2.copy()
```

In [3]:
```
# Display the number of rows and columns.
azdias.shape
```

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

In [4]:
```
# Displays the number of rows and columns.
feat_info.shape
```

Out[4]:
```
(85, 4)
```

In [5]:
```
# Displays general quantitized information.
azdias.describe()
```

Out[5]:

	AGER_TYP	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_M
count	891221.000000	891221.000000	891221.000000	886367.000000	89
mean	-0.358435	2.777398	1.522098	3.632838	
std	1.198724	1.068775	0.499512	1.595021	
min	-1.000000	1.000000	1.000000	1.000000	
25%	-1.000000	2.000000	1.000000	2.000000	
50%	-1.000000	3.000000	2.000000	4.000000	
75%	-1.000000	4.000000	2.000000	5.000000	
max	3.000000	9.000000	2.000000	6.000000	

8 rows × 81 columns

In [6]:
```
# Displays general quantitized information.
feat_info.describe()
```

Out[6]:

	attribute	information_level	type	missing_or_unknown
count	85	85	85	85
unique	85	9	5	9
top	AGER_TYP	person	ordinal	[-1]
freq	1	43	49	26

In [7]:
```
# Displays the number of rows and columns, and print the first 5 rows.
print('\nThe number of rows in the azdias dataset is: ', azdias.shape[0], '\n')

print('The number of columns in the azdias dataset is: ', azdias.shape[1], '\n')

azdias.head(5)
```

```
The number of rows in the azdias dataset is:  891221

The number of columns in the azdias dataset is:  85
```

Out[7]:

	AGER_TYP	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIS
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	

5 rows × 85 columns

In [8]:
```
# Displays the number of rows and columns, and print the first 5 rows.
print('\nThe number of rows in the feat_info dataset is: ', feat_info.shape[0], '\n

print('The number of columns in the feat_info dataset is: ', feat_info.shape[1], '\

feat_info.head(5)
```

```
The number of rows in the feat_info dataset is:  85

The number of columns in the feat_info dataset is:  4
```

Out[8]:

	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 [9]:
```
# Displays basic information of the dataset.
azdias.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891221 entries, 0 to 891220
Data columns (total 85 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   AGER_TYP               891221 non-null  int64  
 1   ALTERSKATEGORIE_GROB   891221 non-null  int64  
 2   ANREDE_KZ              891221 non-null  int64  
 3   CJT_GESAMTTYP          886367 non-null  float64
 4   FINANZ_MINIMALIST      891221 non-null  int64  
 5   FINANZ_SPARER          891221 non-null  int64  
 6   FINANZ_VORSORGER       891221 non-null  int64  
 7   FINANZ_ANLEGER         891221 non-null  int64  
 8   FINANZ_UNAUFFAELLIGER  891221 non-null  int64  
 9   FINANZ_HAUSBAUER       891221 non-null  int64  
 10  FINANZTYP              891221 non-null  int64  
 11  GEBURTSJAHR            891221 non-null  int64  
 12  GFK_URLAUBERTYP        886367 non-null  float64
 13  GREEN_AVANTGARDE       891221 non-null  int64  
 14  HEALTH_TYP             891221 non-null  int64  
 15  LP_LEBENSPHASE_FEIN    886367 non-null  float64
 16  LP_LEBENSPHASE_GROB    886367 non-null  float64
 17  LP_FAMILIE_FEIN        886367 non-null  float64
 18  LP_FAMILIE_GROB        886367 non-null  float64
 19  LP_STATUS_FEIN         886367 non-null  float64
 20  LP_STATUS_GROB         886367 non-null  float64
 21  NATIONALITAET_KZ       891221 non-null  int64  
 22  PRAEGENDE_JUGENDJAHRE  891221 non-null  int64  
 23  RETOURTYP_BK_S         886367 non-null  float64
 24  SEMIO_SOZ              891221 non-null  int64  
 25  SEMIO_FAM              891221 non-null  int64  
 26  SEMIO_REL              891221 non-null  int64  
 27  SEMIO_MAT              891221 non-null  int64  
 28  SEMIO_VERT             891221 non-null  int64  
 29  SEMIO_LUST             891221 non-null  int64  
 30  SEMIO_ERL              891221 non-null  int64  
 31  SEMIO_KULT             891221 non-null  int64  
 32  SEMIO_RAT              891221 non-null  int64  
 33  SEMIO_KRIT             891221 non-null  int64  
 34  SEMIO_DOM              891221 non-null  int64  
 35  SEMIO_KAEM             891221 non-null  int64  
 36  SEMIO_PFLICHT          891221 non-null  int64  
 37  SEMIO_TRADV            891221 non-null  int64  
 38  SHOPPER_TYP            891221 non-null  int64  
 39  SOHO_KZ                817722 non-null  float64
 40  TITEL_KZ               817722 non-null  float64
 41  VERS_TYP               891221 non-null  int64  
 42  ZABEOTYP               891221 non-null  int64  
 43  ALTER_HH               817722 non-null  float64
 44  ANZ_PERSONEN           817722 non-null  float64
 45  ANZ_TITEL              817722 non-null  float64
 46  HH_EINKOMMEN_SCORE     872873 non-null  float64
 47  KK_KUNDENTYP           306609 non-null  float64
 48  W_KEIT_KIND_HH         783619 non-null  float64
 49  WOHNDAUER_2008         817722 non-null  float64
 50  ANZ_HAUSHALTE_AKTIV    798073 non-null  float64
```

```
 51  ANZ_HH_TITEL           794213 non-null  float64
 52  GEBAEUDETYP            798073 non-null  float64
 53  KONSUMNAEHE            817252 non-null  float64
 54  MIN_GEBAEUDEJAHR       798073 non-null  float64
 55  OST_WEST_KZ            798073 non-null  object
 56  WOHNLAGE               798073 non-null  float64
 57  CAMEO_DEUG_2015        792242 non-null  object
 58  CAMEO_DEU_2015         792242 non-null  object
 59  CAMEO_INTL_2015        792242 non-null  object
 60  KBA05_ANTG1            757897 non-null  float64
 61  KBA05_ANTG2            757897 non-null  float64
 62  KBA05_ANTG3            757897 non-null  float64
 63  KBA05_ANTG4            757897 non-null  float64
 64  KBA05_BAUMAX           757897 non-null  float64
 65  KBA05_GBZ              757897 non-null  float64
 66  BALLRAUM               797481 non-null  float64
 67  EWDICHTE               797481 non-null  float64
 68  INNENSTADT             797481 non-null  float64
 69  GEBAEUDETYP_RASTER     798066 non-null  float64
 70  KKK                    770025 non-null  float64
 71  MOBI_REGIO             757897 non-null  float64
 72  ONLINE_AFFINITAET      886367 non-null  float64
 73  REGIOTYP               770025 non-null  float64
 74  KBA13_ANZAHL_PKW       785421 non-null  float64
 75  PLZ8_ANTG1             774706 non-null  float64
 76  PLZ8_ANTG2             774706 non-null  float64
 77  PLZ8_ANTG3             774706 non-null  float64
 78  PLZ8_ANTG4             774706 non-null  float64
 79  PLZ8_BAUMAX            774706 non-null  float64
 80  PLZ8_HHZ               774706 non-null  float64
 81  PLZ8_GBZ               774706 non-null  float64
 82  ARBEIT                 794005 non-null  float64
 83  ORTSGR_KLS9            794005 non-null  float64
 84  RELAT_AB               794005 non-null  float64
dtypes: float64(49), int64(32), object(4)
memory usage: 578.0+ MB
```

In [10]:
```
# Displays basic information of the dataset.
feat_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85 entries, 0 to 84
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   attribute           85 non-null     object
 1   information_level   85 non-null     object
 2   type                85 non-null     object
 3   missing_or_unknown  85 non-null     object
dtypes: object(4)
memory usage: 2.8+ KB
```

Step 1: Preprocessing

Step 1.1: Assess Missing Data

In [11]:
```
# Calculates and displays the amount of NaN values in the dataset.
print('\nThe number of missing or unknown values before converting to NaN is' , sum
```

```
The number of missing or unknown values before converting to NaN is 4896838
```

In [12]:
```
# Identifies missing or unknown data values and convert them to NaNs.
columns = feat_info.attribute.values

miss_unkn = ['-1', '0', '1', '9']

# Iterates through each features attributes missing_or_unknown values to compare wi
for value in columns:

    # Retrieves missing_or_unknown value to compare.
    missing_value = feat_info[feat_info.attribute == value].missing_or_unknown.valu

    for v in range(len(missing_value)):
        missing_value[v] = missing_value[v].strip('[').strip(']').split(',')

    for v in range(len(missing_value[0])):
        if missing_value[0][v] in miss_unkn:
            missing_value[0][v] = int(missing_value[0][v])

    # Replaces non-valid values with NaNs.
    azdias[value] = azdias[value].replace(missing_value[0], np.nan)

print('\nThe number of NaNs from missing or unknown values after converting is: ' ,
```

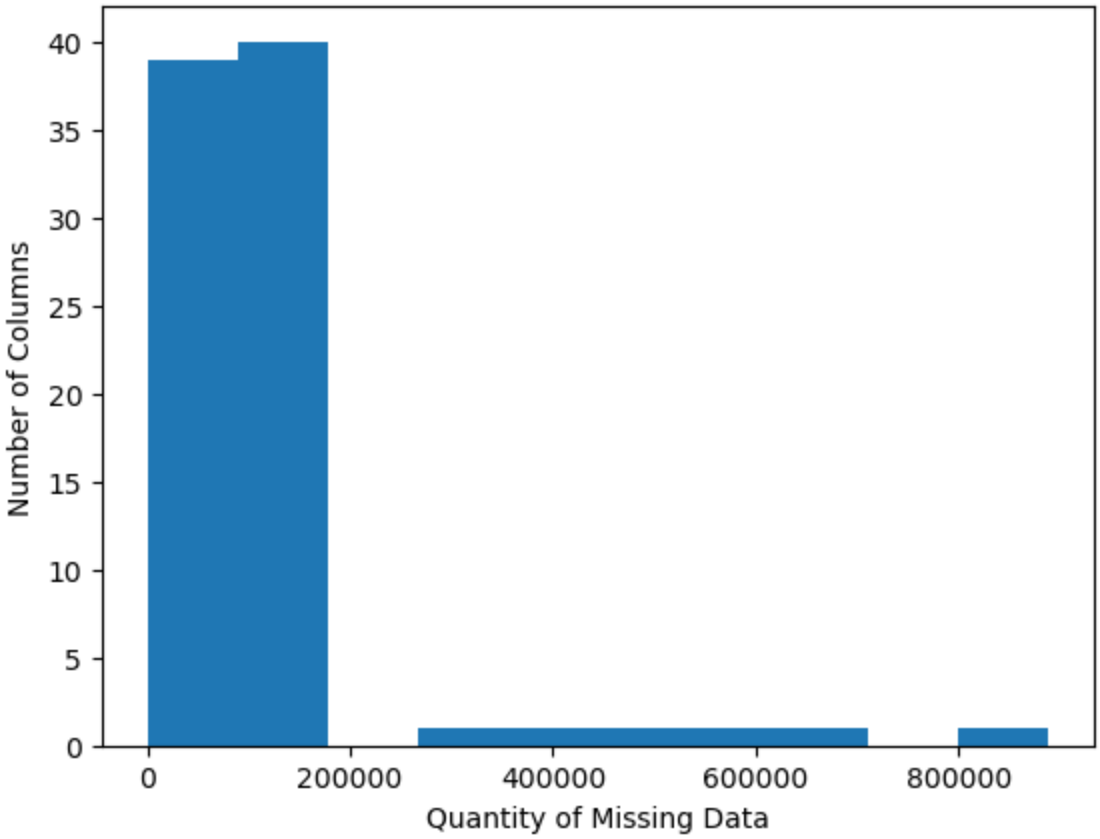
```
The number of NaNs from missing or unknown values after converting is:  8373929
```

Step 1.1.2: Assess Missing Data in Each Column

In [13]:
```
# Assesses how much missing data there is in each column of the dataset.
miss_col = azdias.isnull().sum().sort_values(ascending = False)

miss_col
```

Out[13]:
```
TITEL_KZ        889061
AGER_TYP        685843
KK_KUNDENTYP    584612
KBA05_BAUMAX    476524
GEBURTSJAHR     392318
                 ...
SEMIO_RAT            0
SEMIO_KRIT           0
SEMIO_DOM            0
SEMIO_TRADV          0
ZABEOTYP             0
Length: 85, dtype: int64
```

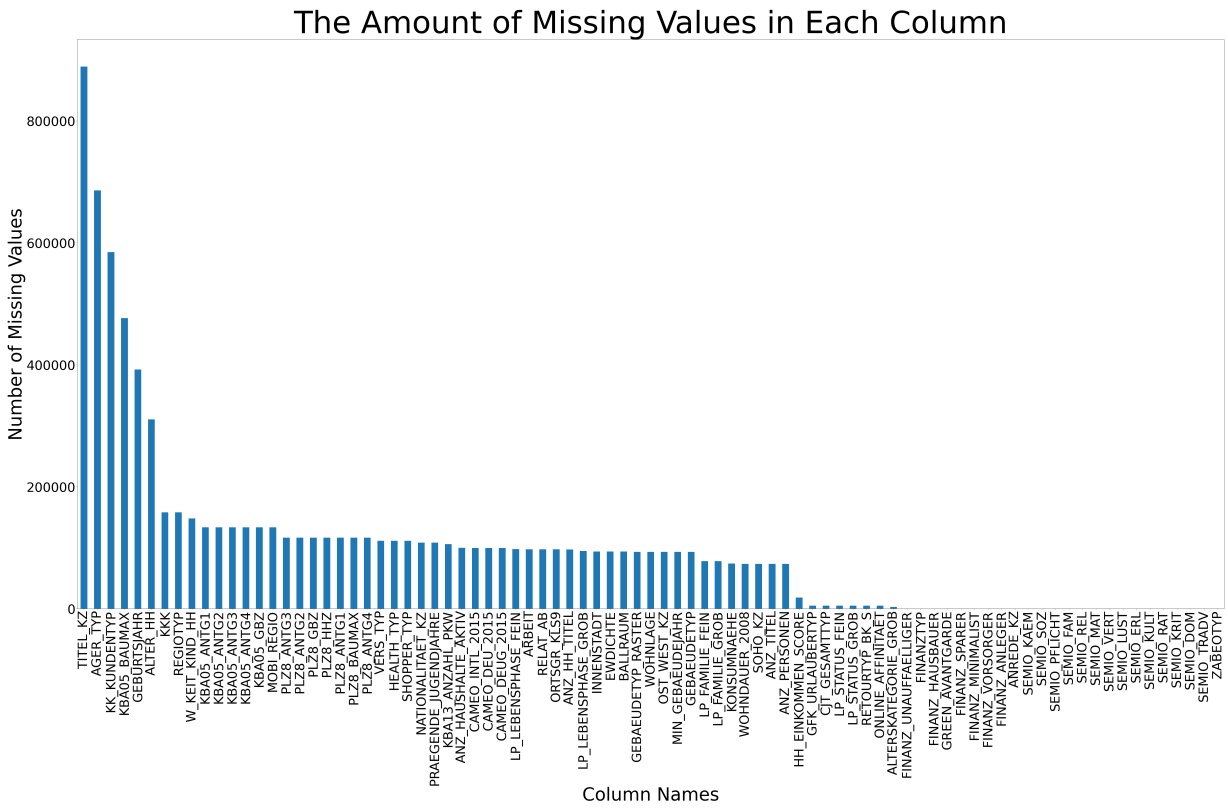
In [14]:
```
# Histogram to investigate patterns in the amount of missing data in each column.
tyu = plt.hist(miss_col)
tyu = plt.xlabel('Quantity of Missing Data')
tyu = plt.ylabel('Number of Columns')

tyu = plt.show()
```



79 columns are missing less than 190,000.

6 columns are missing more than 250,000.

In [15]:
```
# Bar graph to investigate patterns in the amount of missing data in each column.
mcp = miss_col.plot.bar(figsize = (80, 40))
mcp = plt.title('The Amount of Missing Values in Each Column', fontsize = 120)
mcp = plt.ylabel('Number of Missing Values', fontsize = 70)
mcp = plt.yticks(fontsize = 50)
mcp = plt.xlabel('Column Names', fontsize = 70)
mcp = plt.xticks(fontsize = 50)

mcp = plt.show()
```



In [16]:
```
# Calculate and display the number of columns with more than 250,000 missing data v
drop_cols = len(miss_col[miss_col > 250000])
print('\nThere are', drop_cols, 'outlier columns with more than 250,000 missing dat
```

```
There are 6 outlier columns with more than 250,000 missing data values.
```

In [17]:
```
# Calculates and creates a list of the outlier columns.
drop_outlier_cols = miss_col[miss_col > 250000].index.tolist()

drop_outlier_cols
```

Out[17]:
```
['TITEL_KZ',
 'AGER_TYP',
 'KK_KUNDENTYP',
 'KBA05_BAUMAX',
 'GEBURTSJAHR',
 'ALTER_HH']
```

In [18]:
```
# Removes the columns with more than 250,000 missing data values.
azdias = azdias.drop(drop_outlier_cols, axis = 1).copy()

azdias
```

Out[18]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FIN
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	
...	...	...	...	...	
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 [19]:
```
# Loop and condition to remove attribute column values (columns)
# with more than 250,000 missing data values.
for i in range(len(drop_outlier_cols)):

    feat_info = feat_info.drop(feat_info[feat_info.attribute == drop_outlier_cols[i

feat_info
```

Out[19]:

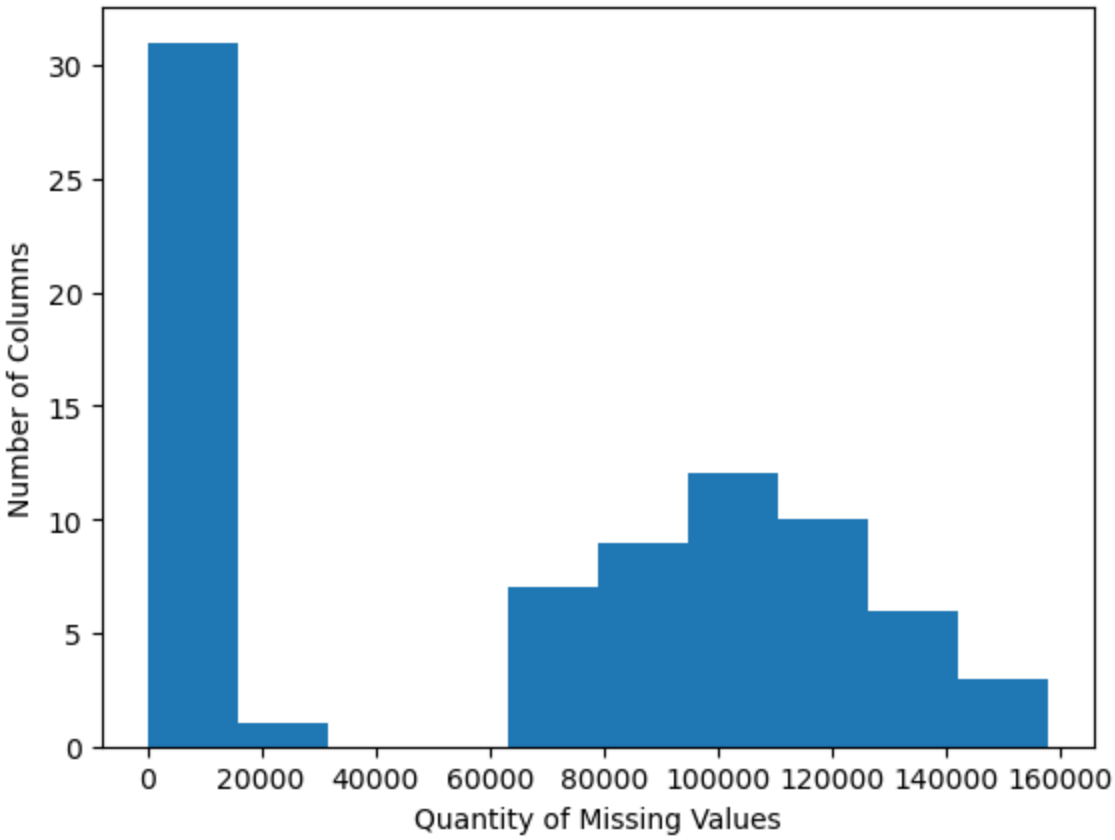
	attribute	information_level	type	missing_or_unknown
1	ALTERSKATEGORIE_GROB	person	ordinal	[-1,0,9]
2	ANREDE_KZ	person	categorical	[-1,0]
3	CJT_GESAMTTYP	person	categorical	[0]
4	FINANZ_MINIMALIST	person	ordinal	[-1]
5	FINANZ_SPARER	person	ordinal	[-1]
...	...	...	...	...
80	PLZ8_HHZ	macrocell_plz8	ordinal	[-1]
81	PLZ8_GBZ	macrocell_plz8	ordinal	[-1]
82	ARBEIT	community	ordinal	[-1,9]
83	ORTSGR_KLS9	community	ordinal	[-1,0]
84	RELAT_AB	community	ordinal	[-1,9]

79 rows × 4 columns

In [20]:
```
# Assesses how much missing data there is in each
# column of the dataset after removing outlier columns.
miss_col2 = azdias.isnull().sum().sort_values(ascending = False)

miss_col2
```

Out[20]:
```
KKK                158064
REGIOTYP           158064
W_KEIT_KIND_HH     147988
KBA05_ANTG4        133324
KBA05_ANTG1        133324
                    ...
SEMIO_KRIT              0
SEMIO_DOM               0
SEMIO_KAEM              0
SEMIO_PFLICHT           0
ZABEOTYP                0
Length: 79, dtype: int64
```

In [21]:
```
# Histogram to display the changes to the number of missim=ng values in columns.
dfg = plt.hist(miss_col2)
dfg = plt.xlabel('Quantity of Missing Values')
dfg = plt.ylabel('Number of Columns')

dfg = plt.show()
```



There are about 30 columns that have less than 20,000 missing data values.

The remainder is spread from approximately 60,000 to 160,000.

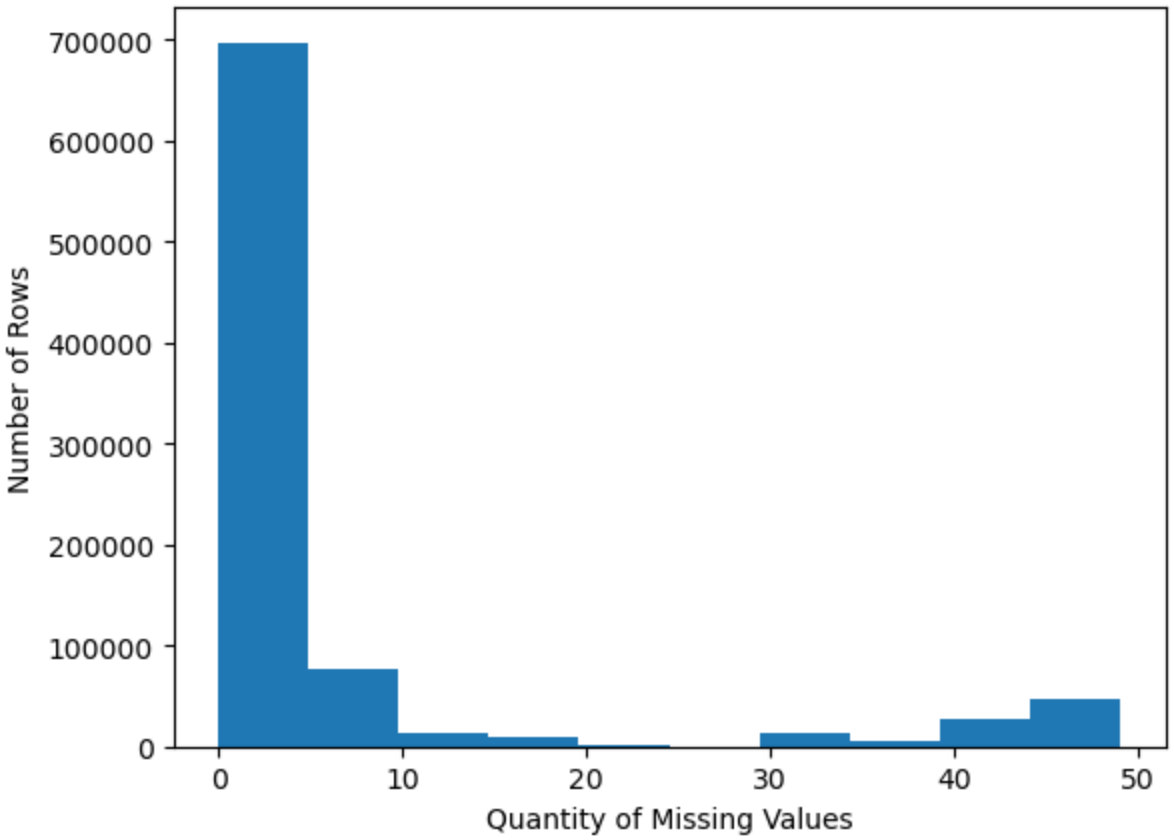
The median of the remaining columns is approximately 100,000 to 120,000.

Step 1.1.3: Assess Missing Data in Each Row

In [22]:
```
# Displays the amount of missing data in each row of the dataset.
miss_row = azdias.isnull().sum(axis = 1)

miss_row
```

Out[22]:
```
0         43
1          0
2          0
3          7
4          0
          ..
891216     3
891217     4
891218     5
891219     0
891220     0
Length: 891221, dtype: int64
```

In [23]:
```
# Histogram to investigate patterns in the amount of missing data in each row.
vbn = plt.hist(miss_row)
vbn = plt.ylabel('Number of Rows')
vbn = plt.xlabel('Quantity of Missing Values')

vbn = plt.show()
```



Approximately 700,000 rows have 5 missing values or less.

Approximately 80,000 rows have 5 - 10 missing values.

Approximately 100,000 rows have more than 25 (range of 29 - 49) missing data values.

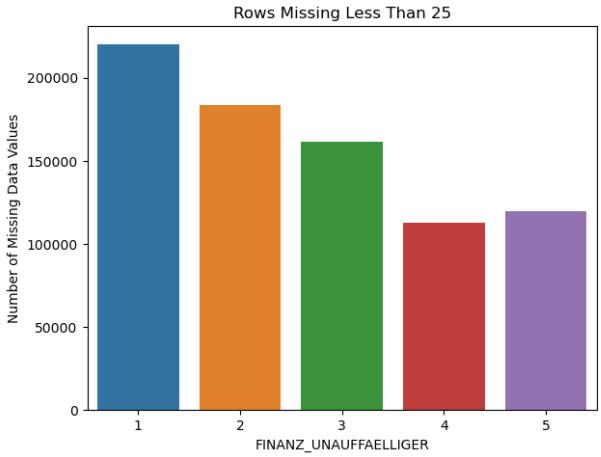
In [24]:
```
# Divides the data into two subsets based on the number of missing values in each r
miss_low = azdias[miss_row <= 25]
miss_high = azdias[miss_row >= 26]
```

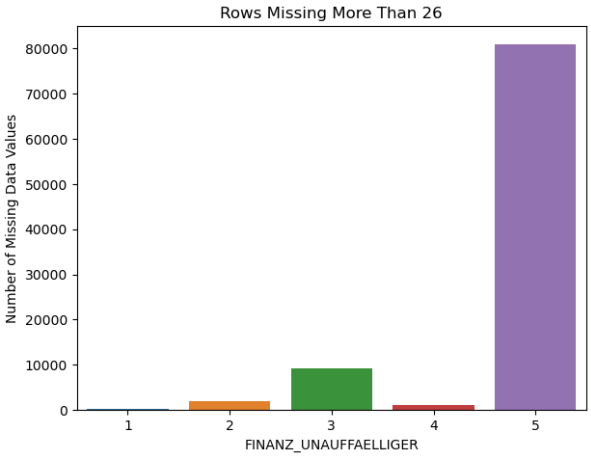
In [25]:
```
# Creates a list of 5 columns to compare that has no missing data.
comp_cols = miss_col.loc[(miss_col == 0)][:5].index.tolist()

comp_cols
```

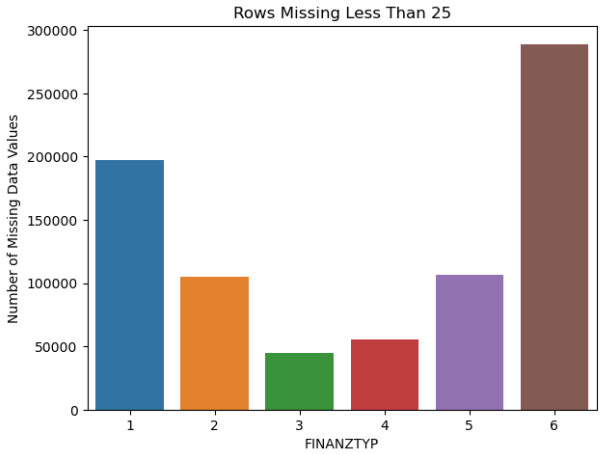
Out[25]:
```
['FINANZ_UNAUFFAELLIGER',
 'FINANZTYP',
 'FINANZ_HAUSBAUER',
 'GREEN_AVANTGARDE',
 'FINANZ_SPARER']
```

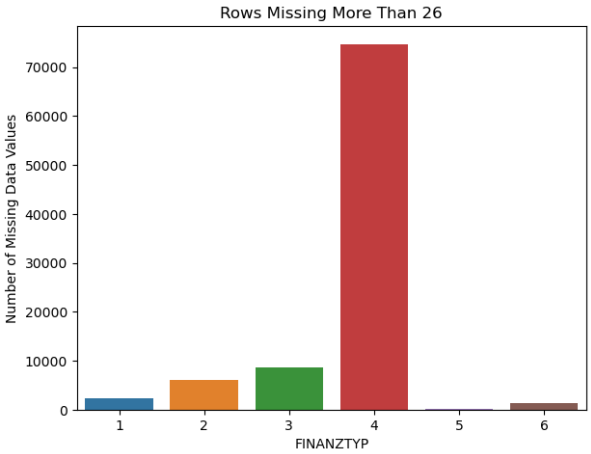
In [26]:
```
# Bar graphs to compare 5 feature columns that has no missing data.
figure1, axs10 = plt.subplots(nrows = 5, ncols = 2, figsize = (15, 30))

figure1.suptitle('\nFeatures With Few Missing Values vs. Many Missing Values', font

# Loop to iterate through the 5 feature columns with no missing data to compare.
for i in range(len(comp_cols)):
    b = sns.countplot(x = comp_cols[i], data = miss_high, ax = axs10[i][1])
    axs10[i][1].set_title('Rows Missing More Than 26')
    b.set(ylabel = 'Number of Missing Data Values')

    k = sns.countplot(x = comp_cols[i], data = miss_low, ax = axs10[i][0])
    axs10[i][0].set_title('Rows Missing Less Than 25')
    k.set(ylabel = 'Number of Missing Data Values')
```

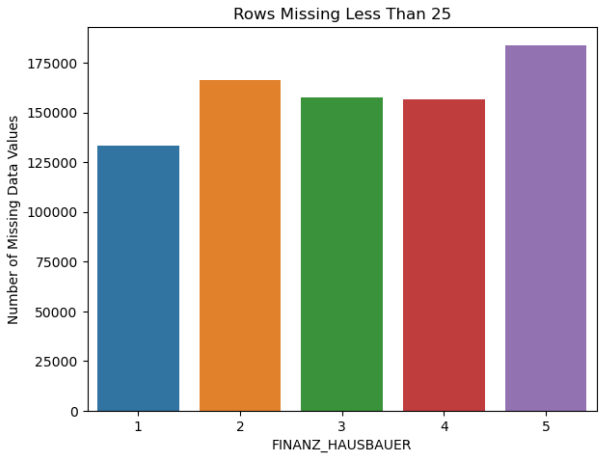
Features With Few Missing Values vs. Many Missing Values

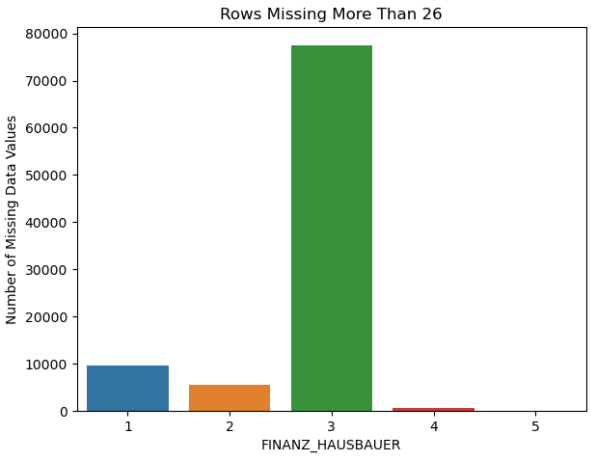


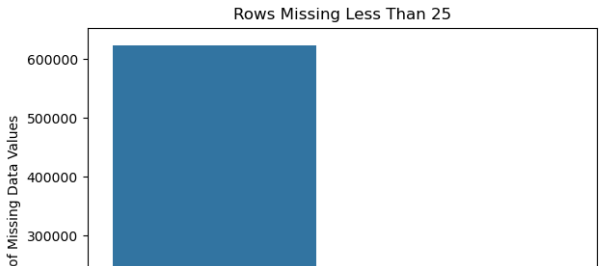


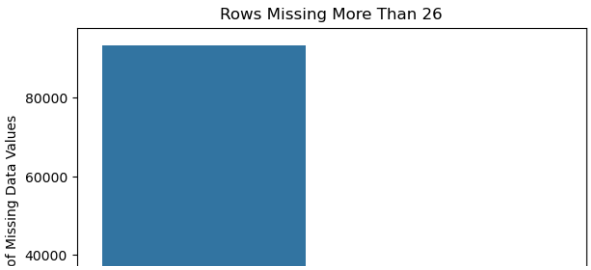


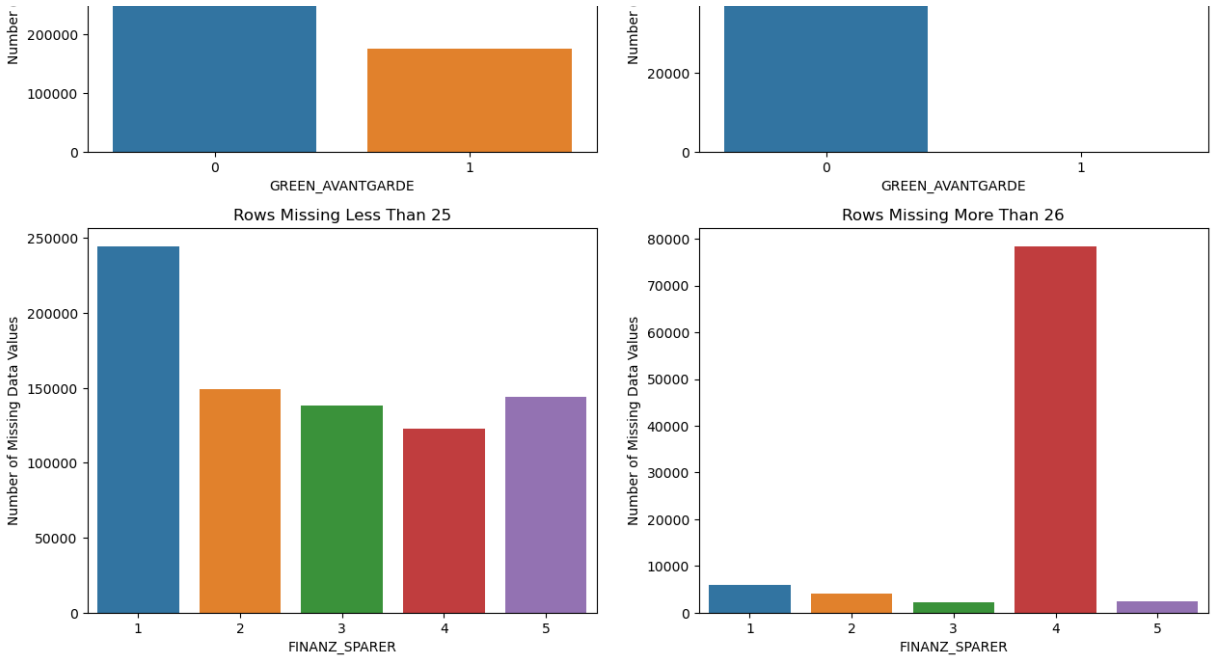












The rows missing less data values are generally more evenly distributed.

The rows missing more data values are practically grouped distinctly all together.

Step 1.2: Select and Re-Encode Features

In [27]:
```
# Displays the number of features of each data type.
feat_info.type.value_counts()
```

Out[27]:
```
ordinal        49
categorical    18
mixed           6
numeric         6
Name: type, dtype: int64
```

In [28]:
```
# Creates a list of the columns with categorical variables.
cat_feats = feat_info[feat_info['type'] == 'categorical'].attribute.values

cat_feats
```

Out[28]:
```
array(['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'], dtype=object)
```

In [29]:
```
# Displays the number of unique values for each categorical column.
cat_vals = miss_low[cat_feats].nunique()
```

```
cat_vals
```

Out[29]:
```
ANREDE_KZ            2
CJT_GESAMTTYP        6
FINANZTYP            6
GFK_URLAUBERTYP     12
GREEN_AVANTGARDE     2
LP_FAMILIE_FEIN     11
LP_FAMILIE_GROB      5
LP_STATUS_FEIN      10
LP_STATUS_GROB       5
NATIONALITAET_KZ     3
SHOPPER_TYP          4
SOHO_KZ              2
VERS_TYP             2
ZABEOTYP             6
GEBAEUDETYP          7
OST_WEST_KZ          2
CAMEO_DEUG_2015      9
CAMEO_DEU_2015      44
dtype: int64
```

In [30]:
```
# Checks the unique values in the column.
miss_low.CAMEO_DEU_2015.unique()
```

Out[30]:
```
array(['8A', '4C', '2A', '6B', '8C', '4A', '2D', '1A', '1E', '9D', '5C',
       '8B', '7A', '5D', '9E', nan, '9B', '1B', '3D', '4E', '4B', '3C',
       '5A', '7B', '9A', '6D', '6E', '2C', '7C', '9C', '7D', '5E', '1D',
       '8D', '6C', '6A', '5B', '4D', '3A', '2B', '7E', '3B', '6F', '5F',
       '1C'], dtype=object)
```

In [31]:
```
# Drops the problematic values from the dataset.
miss_low = miss_low.drop('CAMEO_DEU_2015', axis = 1).copy()
```

In [32]:
```
# Drops the problematic values from the dataset.
feat_info = feat_info.drop(feat_info[feat_info['attribute'] == 'CAMEO_DEU_2015'].in
```

In [33]:
```
# Divides the categorical features into binary and multi.
bin_cat = []
multi_cat = []

# Loop to iterate through the columns
# in list of categorical features.
for column in cat_feats:

    # Condition to check if binary, and append to bin_cat list.
    if azdias[column].nunique() == 2:
        bin_cat.append(column)

    # Appends to multi_cat list if not binary (multi).
    else:
        multi_cat.append(column)
```

In [34]:
```
# Displays the binary categorical features.
bin_cat
```

Out[34]:
```
['ANREDE_KZ', 'GREEN_AVANTGARDE', 'SOHO_KZ', 'VERS_TYP', 'OST_WEST_KZ']
```

In [35]:
```
# Displays the multi categorical features.
multi_cat
```

Out[35]:
```
['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 [36]:
```
# Loop to display the uinque categorical variables.
for i in range(len(bin_cat)):
    print(bin_cat[i])
    print(azdias[bin_cat[i]].unique(), '\n')
```

```
ANREDE_KZ
[1 2]

GREEN_AVANTGARDE
[0 1]

SOHO_KZ
[nan  1.  0.]

VERS_TYP
[nan  2.  1.]

OST_WEST_KZ
[nan 'W' 'O']
```

In [37]:
```
# Re-encodes categorical variable(s) to be kept in the analysis.
miss_low = miss_low.astype({'SOHO_KZ':'Int64'})
miss_low = miss_low.astype({'VERS_TYP':'Int64'})
```

In [38]:
```
# Re-encodes categorical variable(s) to be kept in the analysis.
miss_low["OST_WEST_KZ"].replace({'W':0, 'O':1}, inplace = True)
miss_low["ANREDE_KZ"].replace({2:0, 1:1}, inplace = True)
miss_low["VERS_TYP"].replace({2:0, 1:1}, inplace = True)
```

In [39]:
```
# Loop to display changes to the categorical variables.
for i in range(len(bin_cat)):
    print(miss_low[bin_cat[i]].value_counts(), '\n')
```

```
0    416117
1    381844
Name: ANREDE_KZ, dtype: int64

0    622741
1    175220
Name: GREEN_AVANTGARDE, dtype: int64

0    791245
1      6716
Name: SOHO_KZ, dtype: Int64

0    394229
1    367052
Name: VERS_TYP, dtype: Int64

0    629433
1    168528
Name: OST_WEST_KZ, dtype: int64
```

In [40]:
```
# Engineers two new variables for the PRAEGENDE_JUGENDJAHRE column using the data d
decade = {1:1, 2:1, 3:2, 4:2, 5:3, 6:3, 7:3, 8:4, 9:4, 10:5, 11:5, 12:5, 13:5, 14:6
movement = {1:0, 2:1, 3:0, 4:1, 5:0, 6:1, 7:1, 8:0, 9:1, 10:0, 11:1, 12:0, 13:1, 14

# Creates the 2 new columns and populate the data.
miss_low['DECADE'] = miss_low.PRAEGENDE_JUGENDJAHRE
miss_low.DECADE.replace(decade, inplace = True)

miss_low['MOVEMENT'] = miss_low.PRAEGENDE_JUGENDJAHRE
miss_low.MOVEMENT.replace(movement, inplace = True)
```

In [41]:
```
# Displays the new column with values after changes.
miss_low.DECADE.value_counts()
```

Out[41]:
```
6.0    225511
4.0    175182
5.0    151761
3.0    114349
2.0     74292
1.0     28157
Name: DECADE, dtype: int64
```

In [42]:
```
# Displays the new column with values after changes.
miss_low.MOVEMENT.value_counts()
```

Out[42]:
```
0.0    594032
1.0    175220
Name: MOVEMENT, dtype: int64
```

In [43]:
```
# Drops the column after the 2 new columns
# were created and populated with the data.
miss_low = miss_low.drop('PRAEGENDE_JUGENDJAHRE', axis = 1).copy()

miss_low.head(5)
```

Out[43]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
1	1.0	0	5.0	1	
2	3.0	0	3.0	1	
3	4.0	0	2.0	4	
4	3.0	1	5.0	4	
5	1.0	0	2.0	3	

5 rows × 79 columns

In [44]:
```
# Engineers two new variables for the CAMEO_INTL_2015 column using the data diction
wealth = {'11':1, '12':1, '13':1, '14':1, '15':1, '21':2, '22':2, '23':2, '24':2, '
        '31':3, '32':3, '33':3, '34':3, '35':3, '41':4, '42':4, '43':4, '44':4, '45
        '51':5, '52':5, '53':5, '54':5, '55':5}

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

# Creates the 2 new columns and populate the data.
miss_low['WEALTH'] = miss_low['CAMEO_INTL_2015']
miss_low.WEALTH.replace(wealth, inplace = True)

miss_low['LIFE_STAGE'] = miss_low['CAMEO_INTL_2015']
miss_low.LIFE_STAGE.replace(life_stage, inplace = True)
```

In [45]:
```
# Displays the new column with values after changes.
miss_low.WEALTH.value_counts()
```

Out[45]:
```
5.0    223579
2.0    190674
4.0    189960
1.0    119438
3.0     68189
Name: WEALTH, dtype: int64
```

In [46]:
```
# Displays the new column with values after changes.
miss_low.LIFE_STAGE.value_counts()
```

Out[46]:
```
1.0    245049
4.0    232768
3.0    119687
5.0    117040
2.0     77296
Name: LIFE_STAGE, dtype: int64
```

In [47]:
```
# Drops the column after the 2 new columns
# were created and populated with the data.
miss_low = miss_low.drop('CAMEO_INTL_2015', axis = 1).copy()
```

In [48]:
```
# Drops the CAMEO_INTL_2015 column from the dataset.
feat_info = feat_info.drop(feat_info[feat_info['attribute'] == 'CAMEO_INTL_2015'].i
```

In [49]:
```
# Drops the CAMEO_DEUG_2015 column from the dataset.
miss_low = miss_low.drop('CAMEO_DEUG_2015', axis = 1).copy()
```

In [50]:
```
# Drops the problematic column/attribute from the dataset.
feat_info = feat_info.drop(feat_info[feat_info['attribute'] == 'CAMEO_DEUG_2015'].i
```

In [51]:
```
# Creates a copy of the finalized dataframe.
final = miss_low.copy()

final.head(5)
```

Out[51]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
1	1.0	0	5.0	1	
2	3.0	0	3.0	1	
3	4.0	0	2.0	4	
4	3.0	1	5.0	4	
5	1.0	0	2.0	3	

5 rows × 79 columns

Categorical variables have been re-encoded, and several attributes have been dropped.

Step 1.3: Create a Cleaning Function

In [52]:
```
def clean_data(df):
    """
    Perform feature trimming, re-encoding, and engineering for demographics
    data

    INPUT: Demographics DataFrame
    OUTPUT: Trimmed and cleaned demographics DataFrame
    """
    # Loads in the general demographics data.
    azdias = pd.read_csv ('Udacity_AZDIAS_Subset.csv', sep = ';')

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

    # Creates empty dataframe.
    df = pd.DataFrame()
```

```
# Identifies missing or unknown data values and convert them to NaNs.
columns = feat_info.attribute.values

miss_unkn = ['-1', '0', '1', '9']

# Iterate through each features attributes missing_or_unknown values to compare
for value in columns:

    # Retrieves missing_or_unknown value to compare.
    missing_value = feat_info[feat_info.attribute == value].missing_or_unknown.

    for v in range(len(missing_value)):
        missing_value[v] = missing_value[v].strip('[').strip(']').split(',')

    for v in range(len(missing_value[0])):
        if missing_value[0][v] in miss_unkn:
            missing_value[0][v] = int(missing_value[0][v])

# Replaces non-valid values with NaNs.
df[value] = azdias[value].replace(missing_value[0], np.nan)

# Removes columns with exorbitant missing values.
df = azdias.drop(drop_outlier_cols, axis = 1)

# The amount of missing data in each row of the dataset.
miss_row = df.isnull().sum(axis = 1)

# Removes all rows with larger amounts pf missing data.
miss_low = df[miss_row <= 25]

# Creates a list of the columns with categorical variables.
cat_feats = feat_info[feat_info['type'] == 'categorical'].attribute.values

# Divides categorical features into binary and multi.
bin_cat = []
multi_cat = []

# Loop to iterate through the columns
# in list of categorical features.
for column in cat_feats:

    # Condition to check if binary, and append to bin_cat list.
    if azdias[column].nunique() == 2:
        bin_cat.append(column)

    # Appends to multi_cat list if not binary (multi).
    else:
        multi_cat.append(column)

# Re-encodes categorical variable(s) to be kept in the analysis.
miss_low = miss_low.astype({'SOHO_KZ':'Int64'})
miss_low = miss_low.astype({'VERS_TYP':'Int64'})
miss_low["OST_WEST_KZ"].replace({'W':0, 'O':1}, inplace = True)
miss_low["ANREDE_KZ"].replace({2:0, 1:1}, inplace = True)
```

```
    miss_low["VERS_TYP"].replace({2:0, 1:1}, inplace = True)

    # Engineers two new variables for the PRAEGENDE_JUGENDJAHRE column using the da
    decade = {1:1, 2:1, 3:2, 4:2, 5:3, 6:3, 7:3, 8:4, 9:4, 10:5, 11:5, 12:5, 13:5,
    movement = {1:0, 2:1, 3:0, 4:1, 5:0, 6:1, 7:1, 8:0, 9:1, 10:0, 11:1, 12:0, 13:1

    # Creates the 2 new columns and populate the data.
    miss_low['DECADE'] = miss_low.PRAEGENDE_JUGENDJAHRE
    miss_low.DECADE.replace(decade, inplace = True)

    miss_low['MOVEMENT'] = miss_low.PRAEGENDE_JUGENDJAHRE
    miss_low.MOVEMENT.replace(movement, inplace = True)

    # Engineers two new variables for the CAMEO_INTL_2015 column using the data dic
    wealth = {'11':1, '12':1, '13':1, '14':1, '15':1, '21':2, '22':2, '23':2, '24':
            '31':3, '32':3, '33':3, '34':3, '35':3, '41':4, '42':4, '43':4, '44':4,
            '51':5, '52':5, '53':5, '54':5, '55':5}

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

    # Creates the 2 new columns and populate the data.
    miss_low['WEALTH'] = miss_low.CAMEO_INTL_2015
    miss_low.WEALTH.replace(wealth, inplace = True)

    miss_low['LIFE_STAGE'] = miss_low.CAMEO_INTL_2015
    miss_low.LIFE_STAGE.replace(life_stage, inplace = True)

    # Drops the column after the 2 new columns
    # were created and populated with the data.
    miss_low = miss_low.drop('PRAEGENDE_JUGENDJAHRE', axis = 1).copy()
    miss_low = miss_low.drop('CAMEO_INTL_2015', axis = 1).copy()

    # Drops the problematic column/attribute from the datasets.
    miss_low = miss_low.drop('CAMEO_DEU_2015', axis = 1).copy()

    # Drops the CAMEO_DEUG_2015 column also becasue it Was
    # still causing unexpected problems with the dataset.
    miss_low = miss_low.drop('CAMEO_DEUG_2015', axis = 1).copy()

    miss_low.loc[miss_low["LIFE_STAGE"] == "XX", "LIFE_STAGE"] = 0
    miss_low.loc[miss_low["WEALTH"] == "XX", "WEALTH"] = 0

    # Returns the cleaned dataframe.
    return miss_low
```

Step 2: Feature Transformation

Step 2.1: Apply Feature Scaling

In [53]:
```
# Makes a copy of the final cleaned dataset.
final0 = final.copy()

final0.head(5)
```

Out[53]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
1	1.0	0	5.0	1	
2	3.0	0	3.0	1	
3	4.0	0	2.0	4	
4	3.0	1	5.0	4	
5	1.0	0	2.0	3	

5 rows × 79 columns

In [54]:
```
# Cleans the dataset of all NaN values using the imputer function.
imputer = SimpleImputer()
final_imp = pd.DataFrame(imputer.fit_transform(final0), columns = final0.columns).c

final_imp.head(5)
```

Out[54]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
0	1.0	0.0	5.0	1.0	
1	3.0	0.0	3.0	1.0	
2	4.0	0.0	2.0	4.0	
3	3.0	1.0	5.0	4.0	
4	1.0	0.0	2.0	3.0	

5 rows × 79 columns

In [55]:
```
# Applies feature scaling to the general population demographics data.
scaler = StandardScaler()
final_scale = pd.DataFrame(scaler.fit_transform(final_imp), columns = final0.column

final_scale.head(5)
```

Out[55]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
0	-1.766034	-0.957933	0.976621	-1.494605	1
1	0.201250	-0.957933	-0.327986	-1.494605	0
2	1.184892	-0.957933	-0.980289	0.683156	-0
3	0.201250	1.043914	0.976621	0.683156	0
4	-1.766034	-0.957933	-0.980289	-0.042764	-1

5 rows × 79 columns

Discussion 2.1: Apply Feature Scaling

All NaN values have been converted to the mean using the imputer function.

The scaler was utilized to quantify the invocations to a 1-dimensional value.

Step 2.2: Perform Dimensionality Reduction

In [56]:
```
# Applies PCA to the data.
pca1 = PCA()
final3_pca = pca1.fit_transform(final_scale)
final2_pca = pd.DataFrame(final3_pca, columns = final0.columns ).copy()

final2_pca.head(5)
```

Out[56]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
0	3.771763	-3.326633	-3.154797	0.481585	0
1	0.049799	0.370608	-3.298293	-0.887104	-3
2	-4.231395	1.909257	-0.609001	-1.633209	0
3	-0.199777	-0.141563	2.999128	3.685004	2
4	-0.409134	-1.132591	-0.835877	-3.813928	0

5 rows × 79 columns

In [57]:
```
# Displays the explained variance accounted for by each principal component.
pca1.explained_variance_
```

Out[57]:
```
array([1.35709976e+01, 9.17557650e+00, 6.30363917e+00, 4.63728884e+00,
       3.42375814e+00, 2.42390520e+00, 2.09534056e+00, 1.76051818e+00,
       1.57964628e+00, 1.57587275e+00, 1.50460476e+00, 1.39999243e+00,
       1.27927092e+00, 1.21032443e+00, 1.12024324e+00, 1.08026554e+00,
       1.02554944e+00, 1.00159152e+00, 9.98850887e-01, 9.39382332e-01,
       9.06037446e-01, 8.90417900e-01, 8.48744415e-01, 8.42093325e-01,
       8.08583249e-01, 7.82368461e-01, 7.70333841e-01, 7.58741073e-01,
       7.31313997e-01, 6.99891076e-01, 6.82513189e-01, 6.58276040e-01,
       6.26893869e-01, 5.48147849e-01, 5.15793570e-01, 4.84005978e-01,
       4.78637860e-01, 4.65169001e-01, 4.54265002e-01, 4.36753023e-01,
       4.18012206e-01, 3.97132337e-01, 3.82745809e-01, 3.73872526e-01,
       3.41312030e-01, 3.38733440e-01, 3.16771153e-01, 2.83659986e-01,
       2.75310841e-01, 2.69894683e-01, 2.62508308e-01, 2.54740628e-01,
       2.48189477e-01, 2.43220563e-01, 2.34756761e-01, 2.21062541e-01,
       2.13632356e-01, 2.01172434e-01, 1.97679155e-01, 1.92609245e-01,
       1.74785158e-01, 1.72823140e-01, 1.65179352e-01, 1.63378127e-01,
       1.44757281e-01, 1.40033113e-01, 1.36235993e-01, 1.22597697e-01,
       1.19503634e-01, 1.12856789e-01, 9.91717470e-02, 8.12219764e-02,
       7.89189286e-02, 5.33545827e-02, 4.79530093e-02, 1.08097379e-02,
       7.16639294e-03, 3.86814652e-03, 2.86888779e-03])
```

In [58]:
```
# Displays the explained variance ratio accounted for by each principal component.
pca1.explained_variance_ratio_
```

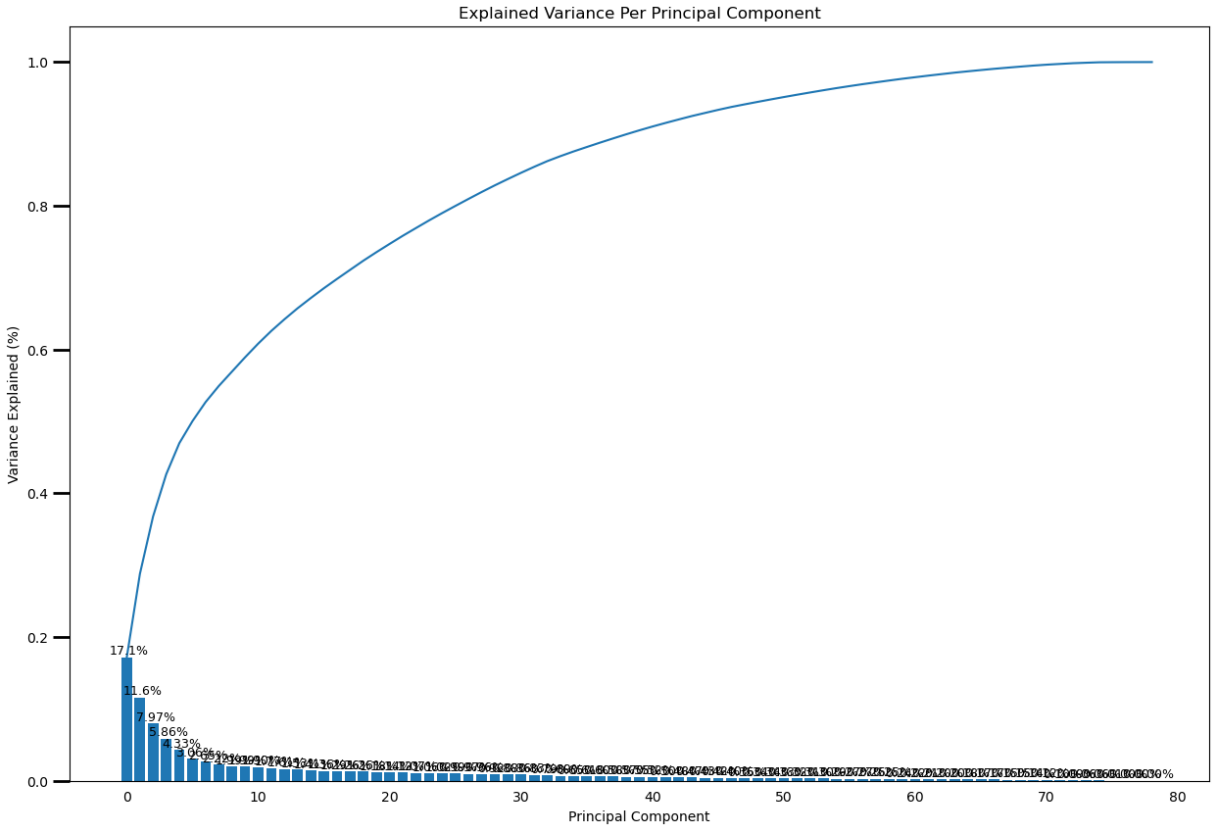
Out[58]:
```
array([1.71784564e-01, 1.16146392e-01, 7.97928008e-02, 5.86997852e-02,
       4.33386563e-02, 3.06823058e-02, 2.65232650e-02, 2.22850123e-02,
       1.99954975e-02, 1.99477313e-02, 1.90456060e-02, 1.77214010e-02,
       1.61932825e-02, 1.53205432e-02, 1.41802764e-02, 1.36742302e-02,
       1.29816222e-02, 1.26783577e-02, 1.26436663e-02, 1.18909007e-02,
       1.14688141e-02, 1.12710985e-02, 1.07435867e-02, 1.06593958e-02,
       1.02352182e-02, 9.90338583e-03, 9.75104905e-03, 9.60430535e-03,
       9.25712760e-03, 8.85936961e-03, 8.63939663e-03, 8.33259766e-03,
       7.93535549e-03, 6.93857168e-03, 6.52902436e-03, 6.12665027e-03,
       6.05869949e-03, 5.88820783e-03, 5.75018269e-03, 5.52851234e-03,
       5.29128711e-03, 5.02698530e-03, 4.84487758e-03, 4.73255769e-03,
       4.32040003e-03, 4.28775969e-03, 4.00975640e-03, 3.59062823e-03,
       3.48494298e-03, 3.41638411e-03, 3.32288581e-03, 3.22456087e-03,
       3.14163501e-03, 3.07873744e-03, 2.97160085e-03, 2.79825650e-03,
       2.70420365e-03, 2.54648332e-03, 2.50226465e-03, 2.43808865e-03,
       2.21246758e-03, 2.18763194e-03, 2.09087525e-03, 2.06807496e-03,
       1.83236836e-03, 1.77256883e-03, 1.72450408e-03, 1.55186764e-03,
       1.51270233e-03, 1.42856516e-03, 1.25533700e-03, 1.02812499e-03,
       9.98972528e-04, 6.75373619e-04, 6.06999358e-04, 1.36831953e-04,
       9.07137210e-05, 4.89638186e-05, 3.63149898e-05])
```

In [59]:
```
# Displays the cummulative explained variance ratio.
np.cumsum(pca1.explained_variance_ratio_)
```

Out[59]:
```
array([0.17178456, 0.28793096, 0.36772376, 0.42642354, 0.4697622 ,
       0.5004445 , 0.52696777, 0.54925278, 0.56924828, 0.58919601,
       0.60824162, 0.62596302, 0.6421563 , 0.65747684, 0.67165712,
       0.68533135, 0.69831297, 0.71099133, 0.723635  , 0.7355259 ,
       0.74699471, 0.75826581, 0.7690094 , 0.77966879, 0.78990401,
       0.7998074 , 0.80955845, 0.81916275, 0.82841988, 0.83727925,
       0.84591864, 0.85425124, 0.8621866 , 0.86912517, 0.87565419,
       0.88178084, 0.88783954, 0.89372775, 0.89947793, 0.90500645,
       0.91029773, 0.91532472, 0.9201696 , 0.92490215, 0.92922255,
       0.93351031, 0.93752007, 0.9411107 , 0.94459564, 0.94801203,
       0.95133491, 0.95455947, 0.95770111, 0.96077984, 0.96375145,
       0.9665497 , 0.96925391, 0.97180039, 0.97430265, 0.97674074,
       0.97895321, 0.98114084, 0.98323172, 0.98529979, 0.98713216,
       0.98890473, 0.99062923, 0.9921811 , 0.9936938 , 0.99512237,
       0.99637771, 0.99740583, 0.9984048 , 0.99908018, 0.99968718,
       0.99982401, 0.99991472, 0.99996369, 1.        ])
```

In [60]:
```
# Creates a function to plot the weights of each component.
def screen_plot(pca):
    '''
    Creates a screen plot associated with the principal components

    INPUT: pca - the result of instantian of PCA in scikit learn

    OUTPUT:
            None
    '''
    num_components = len(pca.explained_variance_ratio_)
    ind = np.arange(num_components)
    vals = pca.explained_variance_ratio_

    qwe = plt.figure(figsize = (15, 10))
    qwe = ax20 = plt.subplot(111)
    cumvals = np.cumsum(vals)
    qwe = ax20.bar(ind, vals)
    qwe = ax20.plot(ind, cumvals)
    for i in range(num_components):
        qwe = ax20.annotate(r"%s%%" % ((str(vals[i] * 100)[:4])), (ind[i] + 0.2, va
                    va = "bottom", ha = "center", fontsize = 9)

    qwe = ax20.xaxis.set_tick_params(width = 0)
    qwe = ax20.yaxis.set_tick_params(width = 2, length = 12)

    qwe = ax20.set_xlabel("Principal Component")
    qwe = ax20.set_ylabel("Variance Explained (%)")
    qwe = plt.title('Explained Variance Per Principal Component')
```

In [61]:
```
# Visualization of the explained_variance_ratio for each of the components.
screen_plot(pca1)
```



Reducing the principal components to 30 will retain approximately 80% of the data.

In [62]:
```
# Re-applies PCA to the data while selecting 30 components to retain.
pca_components = PCA(30)
final_pca = pca_components.fit_transform(final_scale)
final3 = pd.DataFrame(final_scale).copy()

final3.head(5)
```

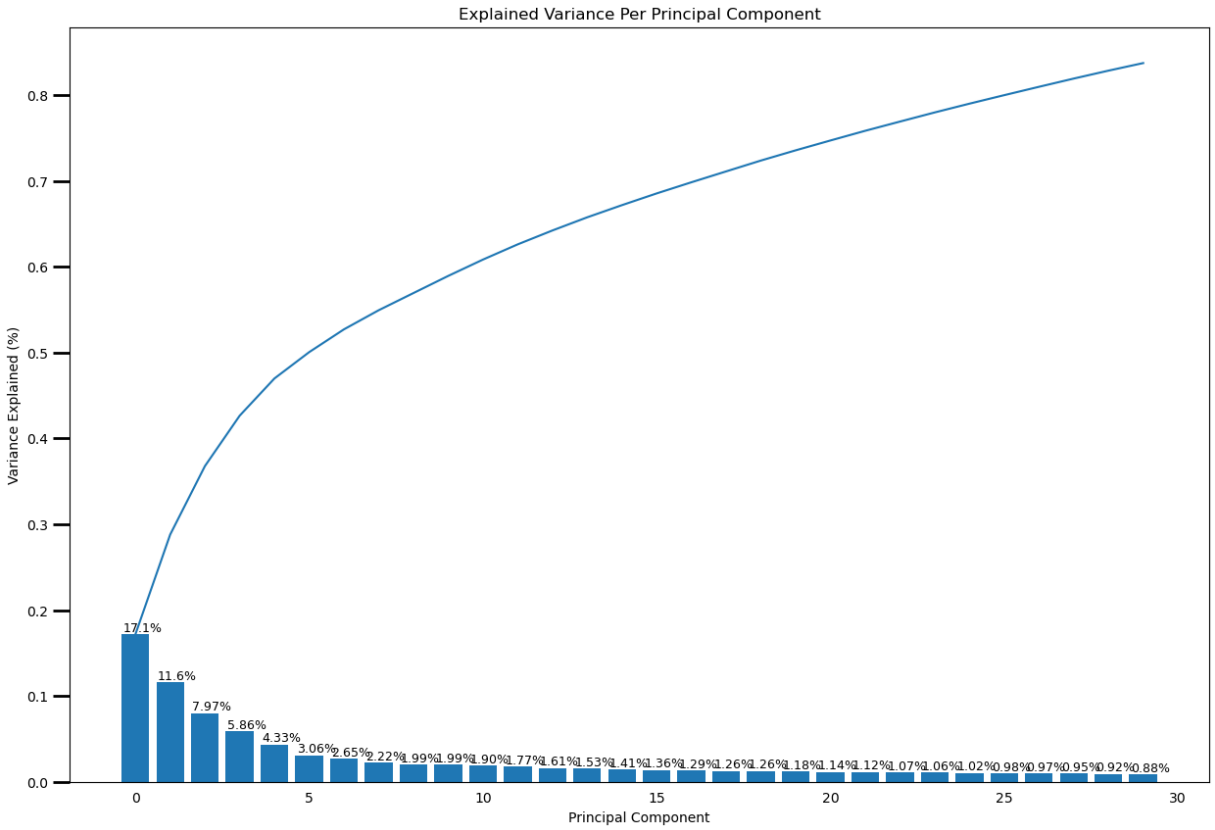
Out[62]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
0	-1.766034	-0.957933	0.976621	-1.494605	1
1	0.201250	-0.957933	-0.327986	-1.494605	0
2	1.184892	-0.957933	-0.980289	0.683156	-0
3	0.201250	1.043914	0.976621	0.683156	0
4	-1.766034	-0.957933	-0.980289	-0.042764	-1

5 rows × 79 columns

In [63]:
```
# Visualization of the explained_variance_ratio for each of the components.
```

```
screen_plot(pca_components)
```



Discussion 2.2: Perform Dimensionality Reduction

Approximately 50% of the data is within the first 5 components.

Step 2.3: Interpret Principal Components

In [64]:
```
# Creates a function to map the feature attribute weights.
def map_feat_weight(df, pca, i):
    '''
    Input:  - DataFrame (df)
            - PCA Value (pca)
            - Feature Index (i)

    Output: - Principal component weght for all features.S

    '''
    df = pd.DataFrame(pca.components_, columns = df.columns)
    return df.iloc[i].sort_values(ascending = False)
```

In [65]:
```
# Maps weights for the first principal component to corresponding feature names
# and then print the linked values, sorted by weight.
```

```
map_feat_weight(final_scale, pca_components , 0)
```

Out[65]:
```
PLZ8_ANTG3            0.199335
PLZ8_ANTG4            0.193703
HH_EINKOMMEN_SCORE    0.193105
PLZ8_BAUMAX           0.188985
WEALTH                0.185785
                         ...
PLZ8_ANTG1           -0.199288
KBA05_ANTG1          -0.204849
LP_STATUS_GROB       -0.216248
LP_STATUS_FEIN       -0.218529
MOBI_REGIO           -0.219084
Name: 0, Length: 79, dtype: float64
```

In [66]:
```
# Maps weights for the second principal component to corresponding feature names
# and then print the linked values, sorted by weight.
map_feat_weight(final_scale, pca_components , 1)
```

Out[66]:
```
ALTERSKATEGORIE_GROB    0.264746
FINANZ_VORSORGER        0.237760
SEMIO_ERL               0.222903
SEMIO_LUST              0.184266
RETOURTYP_BK_S          0.162578
                           ...
SEMIO_TRADV            -0.232203
SEMIO_PFLICHT          -0.233864
FINANZ_SPARER          -0.247597
SEMIO_REL              -0.254937
DECADE                 -0.256397
Name: 1, Length: 79, dtype: float64
```

In [67]:
```
# Maps weights for the third principal component to corresponding feature names
# and then print the linked values, sorted by weight.
map_feat_weight(final_scale, pca_components , 2)
```

Out[67]:
```
ANREDE_KZ     0.363551
SEMIO_VERT    0.336962
SEMIO_SOZ     0.260649
SEMIO_FAM     0.255286
SEMIO_KULT    0.243121
                 ...
SEMIO_ERL    -0.187894
SEMIO_RAT    -0.198943
SEMIO_KRIT   -0.272851
SEMIO_DOM    -0.307403
SEMIO_KAEM   -0.332700
Name: 2, Length: 79, dtype: float64
```

Discussion 2.3: Interpret Principal Components

The positive weights are associated with features that are above average.

The negative weights are associated with features that are below average.

Step 3: Clustering

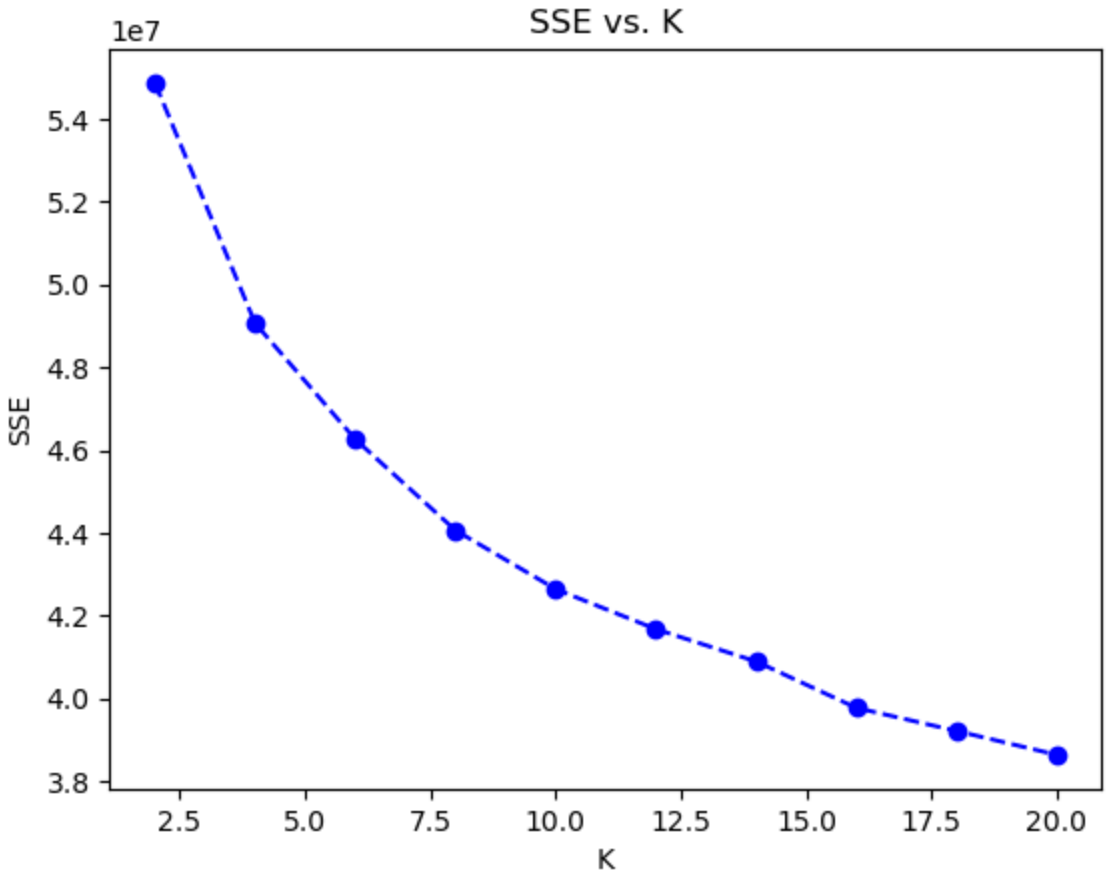
Step 3.1: Apply Clustering to General Population

In [68]:
```
# Creates a function to calculate the kmeans score.
def get_kmeans_score(data, center):
    '''
    returns the kmeans score regarding SSE for points to centers
    INPUT:
        data - the dataset you want to fit kmeans to
        center - the number of centers you want (the k value)
    OUTPUT:
        score - the SSE score for the kmeans model fit to the data
    '''
    # Instantiate kmeans model.
    kmeans = KMeans(n_clusters = center)

    # Then fit the model to your data using the fit method
    model = kmeans.fit(data)

    # Obtain a score related to the model fit.
    score = np.abs(model.score(data))

    return score
```

In [69]:
```
# Gets kmeans score for 11 clusters.
print(get_kmeans_score(final3, 11))
```

```
42145143.39419271
```

In [70]:
```
# Visualization of the change in within-cluster distance across number of clusters.
scores = []
centers = list(range(2, 22, 2))

for center in centers:
    scores.append(get_kmeans_score(final3, center))

abc = plt.plot(centers, scores, linestyle = '--', marker = 'o', color = 'b')
abc = plt.xlabel('K')
abc = plt.ylabel('SSE')
abc = plt.title('SSE vs. K')
```



In [71]:
```
# Re-fits the k-means model with 12 clusters and obtain cluster predictions.
# Instantiates a model with 12 centers.
kmeans = KMeans(n_clusters = 12)

# Fits the model using the fit method.
model = kmeans.fit(final_pca)

# Predicts the labels on the same data to show the category that point belongs to.
gen_predict = model.predict(final_pca)

gen_predict
```

Out[71]:
```
array([1, 7, 4, ..., 7, 9, 2])
```

In [72]:
```
# Gets kmeans score for 12 clusters.
print(get_kmeans_score(final3, 12))
```

```
41675699.94817487
```

Discussion 3.1: Apply Clustering to General Population

12 clusters are selected for the analysis by means of the elbow method.

Step 3.2: Apply All Steps to the Customer Data

In [73]:
```
# Loads in the customer demographics data and creates a copy.
customers0 = pd.read_csv("Udacity_CUSTOMERS_Subset.csv", sep = ';')

customers2 = customers0.copy()

customers2.head(5)
```

Out[73]:

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

5 rows × 85 columns

In [74]:
```
# Applies preprocessing, feature transformation, and clustering
# on the customer demographics data using the clean_data function.
customers = clean_data(customers2).copy()

customers.head(5)
```

Out[74]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
1	1	0	5.0	1	
2	3	0	3.0	1	
3	4	0	2.0	4	
4	3	1	5.0	4	
5	1	0	2.0	3	

5 rows × 79 columns

In [75]:
```
# Display the number of rows and columns for comparison.
final0.shape
```

Out[75]:
```
(797961, 79)
```

In [76]:
```
# Display the number of rows and columns for comparison.
customers.shape
```

Out[76]: (798039, 79)

In [77]:
```
# Calculate the difference in rows.
customers.shape[0] - final0.shape[0]
```

Out[77]: 78

In [78]:
```
# Create a list of the rows not matching indices.
drp_rws = customers.drop(final0.index).index.tolist()

drp_rws[:5]
```

Out[78]: [15959, 19021, 25985, 46811, 51197]

In [79]:
```
# Confirm the number of rows to be removed.
len(drp_rws)
```

Out[79]: 78

In [80]:
```
# Loop to remove the rows not matching index of other dataframe.
for i in range(len(drp_rws)):
    customers = customers.drop([drp_rws[i]])
```

In [81]:
```
# Display the number of rows and columns for comparison.
final0.shape
```

Out[81]: (797961, 79)

In [82]:
```
# Display the number of rows and columns for comparison.
customers.shape
```

Out[82]: (797961, 79)

In [83]:
```
# Cleans the dataset of all NaN values using the imputer function.
imputer2 = SimpleImputer()
customers_imp = pd.DataFrame(imputer2.fit_transform(customers),
                             columns = customers.columns).copy()
customers_imp.head(5)
```

Out[83]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
0	1.0	0.0	5.0	1.0	
1	3.0	0.0	3.0	1.0	
2	4.0	0.0	2.0	4.0	
3	3.0	1.0	5.0	4.0	
4	1.0	0.0	2.0	3.0	

5 rows × 79 columns

In [84]:
```
# Applies feature scaling to the general population demographics data.
scaler2 = StandardScaler()
customers_scale = pd.DataFrame(scaler2.fit_transform(customers_imp),
                               columns = customers_imp.columns).copy()

customers_scale.head(5)
```

Out[84]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST	FINANZ_
0	-1.681284	-0.957933	0.976621	-1.494605	1
1	0.169145	-0.957933	-0.327986	-1.494605	0
2	1.094360	-0.957933	-0.980289	0.683156	-0
3	0.169145	1.043914	0.976621	0.683156	0
4	-1.681284	-0.957933	-0.980289	-0.042764	-1

5 rows × 79 columns

In [85]:
```
# Applies PCA to the data with the selected 30 components.
pca_components2 = PCA(30)
cust_pca = pd.DataFrame(pca_components2.fit_transform(customers_scale)).copy()

cust_pca.head(5)
```

Out[85]:

	0	1	2	3	4	5	6	7	
0	3.598388	-3.371157	-3.197286	0.501612	1.150745	2.251718	2.787738	0.518178	0
1	-0.052340	0.468008	-3.262679	-0.795277	-3.011292	0.306102	0.605312	-0.886906	-0
2	-3.832932	2.240854	-0.524317	-3.021262	-1.786723	-1.332045	1.605438	1.014918	-1
3	-0.167358	-0.212718	3.099932	3.638417	2.047730	0.161778	-1.918317	-2.431411	-0
4	-0.430444	-1.120543	-1.059025	-3.632857	0.213887	2.302937	-0.504327	-1.064493	0

5 rows × 30 columns

In [86]:
```
# Maps weights for the first principal component to corresponding feature names
# and then print the linked values, sorted by weight.
map_feat_weight(customers_scale, pca_components2 , 0)
```

Out[86]:
```
PLZ8_ANTG3            0.200889
PLZ8_ANTG4            0.195041
HH_EINKOMMEN_SCORE    0.194533
PLZ8_BAUMAX           0.190309
WEALTH                0.187478
                         ...
PLZ8_ANTG1           -0.200706
KBA05_ANTG1          -0.205875
LP_STATUS_GROB       -0.216301
LP_STATUS_FEIN       -0.218955
MOBI_REGIO           -0.220527
Name: 0, Length: 79, dtype: float64
```

In [87]:
```
# Maps weights for the second principal component to corresponding feature names
# and then print the linked values, sorted by weight.
map_feat_weight(customers_scale, pca_components2 , 1)
```

Out[87]:
```
ALTERSKATEGORIE_GROB    0.256282
FINANZ_VORSORGER        0.241584
SEMIO_ERL               0.232150
SEMIO_LUST              0.188783
RETOURTYP_BK_S          0.164986
                            ...
SEMIO_PFLICHT          -0.237249
DECADE                 -0.238275
SEMIO_TRADV            -0.239095
FINANZ_SPARER          -0.250755
SEMIO_REL              -0.259652
Name: 1, Length: 79, dtype: float64
```

In [88]:
```
# Maps weights for the third principal component to corresponding feature names
# and then print the linked values, sorted by weight.
map_feat_weight(customers_scale, pca_components2 , 2)
```

Out[88]:
```
ANREDE_KZ         0.363707
SEMIO_VERT        0.337724
SEMIO_SOZ         0.259348
SEMIO_FAM         0.251146
SEMIO_KULT        0.238209
                     ...
FINANZ_ANLEGER   -0.180820
SEMIO_RAT        -0.206088
SEMIO_KRIT       -0.272010
SEMIO_DOM        -0.306995
SEMIO_KAEM       -0.331276
Name: 2, Length: 79, dtype: float64
```

In [89]:
```
# Fits the k-means model with 12 clusters and obtain
# cluster predictions for the customer population demographics data.
# Instantiates a model with 12 centers.
kmeans2 = KMeans(n_clusters = 12)

# Fits the model to your data using the fit method.
model2 = kmeans2.fit(cust_pca)

# Predicts the labels on the same data to show the category that point belongs to.
```

```
cust_predict = model2.predict(cust_pca)

cust_predict
```

Out[89]:
```
array([9, 2, 4, ..., 2, 5, 7])
```

Step 3.3: Compare Customer Data to Demographics Data

In [90]:
```
# Creates a series from the general population prediction model data.
gen_predict2 = pd.Series(gen_predict).value_counts().sort_index().copy()

gen_predict2
```

Out[90]:
```
0     63784
1     80677
2     82523
3     53330
4     59597
5     43721
6     59675
7     82724
8     64001
9     67304
10    56181
11    84444
dtype: int64
```

In [91]:
```
# Creates a series from the customers prediction model data.
cust_predict2 = pd.Series(cust_predict).value_counts().sort_index().copy()

cust_predict2
```

Out[91]:
```
0     44511
1     83976
2     73479
3     31639
4     72100
5     63619
6     59027
7     85571
8     88819
9     71283
10    60501
11    63436
dtype: int64
```

In [92]:
```
# Creates a dataframe of the cluster prediction
# models for the general and customers data.
gen_cust = pd.concat([gen_predict2, cust_predict2], axis = 1)
gen_cust.columns = ['General', 'Customers']
gen_cust['Cluster'] = (gen_cust.index + 1)
gen_cust = gen_cust.set_index('Cluster').copy().reset_index()
gen_cust.set_index(gen_cust.index).reset_index()
```

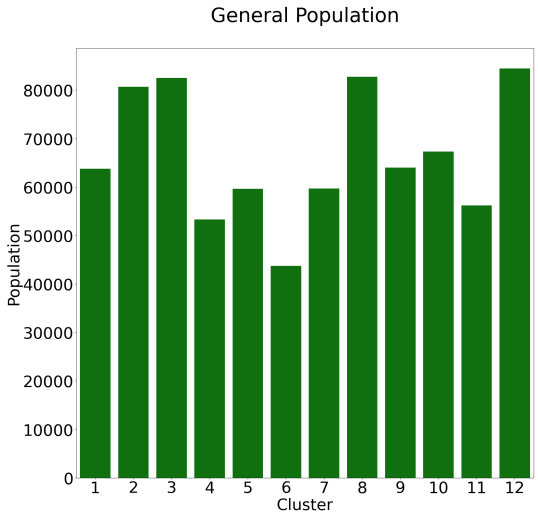
```
gen_cust
```

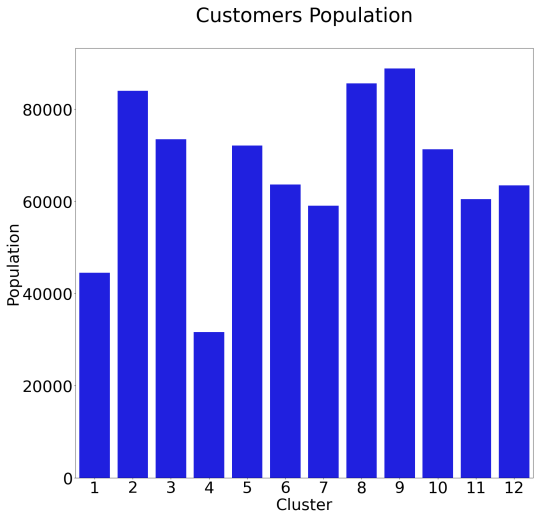
Out[92]:

	Cluster	General	Customers
0	1	63784	44511
1	2	80677	83976
2	3	82523	73479
3	4	53330	31639
4	5	59597	72100
5	6	43721	63619
6	7	59675	59027
7	8	82724	85571
8	9	64001	88819
9	10	67304	71283
10	11	56181	60501
11	12	84444	63436

In [93]:

```
# Visualization to compare the general population and customer dataset model predic
fig = plt.figure(figsize = (40,15))

ax1 = fig.add_subplot(121)
ax1 = sns.barplot(x = 'Cluster', y = 'General', data = gen_cust, color = 'g')
ax1.set_xlabel('Cluster', fontsize = 40)
ax1.set_ylabel('Population', fontsize = 40)
ax1.xaxis.set_tick_params(labelsize = 40)
ax1.yaxis.set_tick_params(labelsize = 40)
ax1 = plt.title("General Population\n", fontsize = 50)

ax2 = fig.add_subplot(122)
ax2 = sns.barplot(x = 'Cluster', y = 'Customers', data = gen_cust,color = 'b')
ax2.set_xlabel('Cluster', fontsize = 40)
ax2.set_ylabel('Population', fontsize = 40)
ax2.xaxis.set_tick_params(labelsize = 40)
ax2.yaxis.set_tick_params(labelsize = 40)
ax2 = plt.title("Customers Population\n", fontsize = 50)

fig = plt.subplots_adjust(left = 1, bottom = 1, right = 2, top = 2, wspace = 0.5)
```





In [94]:
```
# Loop to create lists of the proportion of values in each
# cluster for the customer and general population datasets.
gen_proport = []
cust_proport = []

# Loop to calculate proportional data for the clusters of
# the general and customer datasets and appends to the lists.
for i in range(len(gen_cust)):
    gen_proport.append((gen_predict2[i]).sum() / len(gen_predict2))
    cust_proport.append((cust_predict2[i]).sum() / len(cust_predict2))
```

In [95]:
```
# Creates a series from the general model prediction proportional data.
gen_proport = pd.Series(gen_proport).copy()

gen_proport
```

Out[95]:
```
0     5315.333333
1     6723.083333
2     6876.916667
3     4444.166667
4     4966.416667
5     3643.416667
6     4972.916667
7     6893.666667
8     5333.416667
9     5608.666667
10    4681.750000
11    7037.000000
dtype: float64
```

In [96]:
```
# Creates a series from the customers model prediction proportional data.
cust_proport = pd.Series(cust_proport).copy()

cust_proport
```

Out[96]:
```
0     3709.250000
1     6998.000000
2     6123.250000
3     2636.583333
4     6008.333333
5     5301.583333
6     4918.916667
7     7130.916667
8     7401.583333
9     5940.250000
10    5041.750000
11    5286.333333
dtype: float64
```

In [97]:
```
# Creates a dataframe of the cluster prediction
# models for the general and customers data.
gen_cust2 = pd.concat([gen_predict2, cust_predict2], axis = 1)
gen_cust2.columns = ['General', 'Customers']
gen_cust2['Cluster'] = (gen_cust2.index + 1)
gen_cust2 = gen_cust2.set_index('Cluster').copy().reset_index()
gen_cust2.set_index(gen_cust2.index).reset_index()

gen_cust2
```

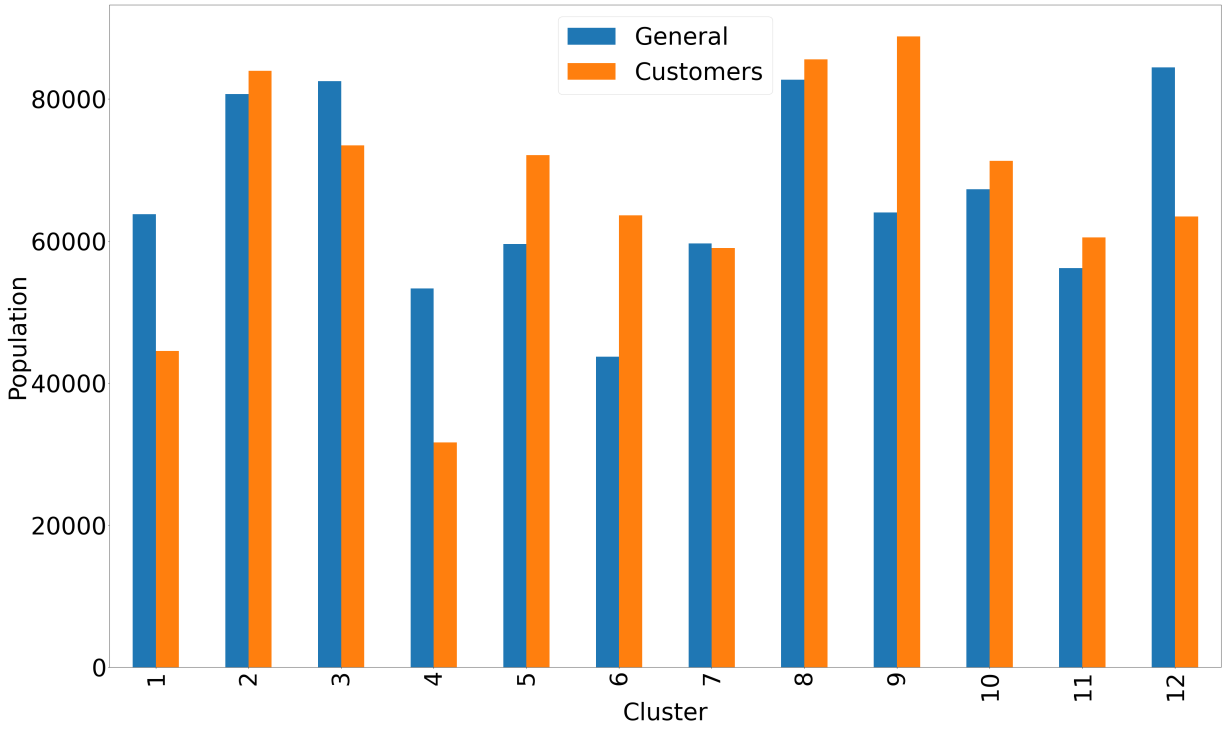
Out[97]:

	Cluster	General	Customers
0	1	63784	44511
1	2	80677	83976
2	3	82523	73479
3	4	53330	31639
4	5	59597	72100
5	6	43721	63619
6	7	59675	59027
7	8	82724	85571
8	9	64001	88819
9	10	67304	71283
10	11	56181	60501
11	12	84444	63436

In [98]:
```
# Visualization of the clusters prediction proportional data comparing the general
gen_cust_comp = gen_cust2.plot(x='Cluster', y = ['General', 'Customers'], kind = 'b
gen_cust_comp.set_xlabel('Cluster', fontsize = 60)
gen_cust_comp.set_ylabel('Population', fontsize = 60)
gen_cust_comp.xaxis.set_tick_params(labelsize = 60)
gen_cust_comp.yaxis.set_tick_params(labelsize = 60)
```

```
gen_cust_comp.legend(['General', 'Customers'], prop = {'size' : 60}, loc = 'upper c
```

Out[98]: <matplotlib.legend.Legend at 0x166e4ed70a0>



Further analyze the proportional differences between the 2 demographic datasets.

In [99]:
```
# Loop to calculate the differences in proportional values for the general and cust
gen_cust_diff_proport = []

for i in range(len(gen_cust2)):
    gen_cust_diff_proport.append((gen_cust2.General[i] - gen_cust2.Customers[i]))
```

In [100…
```
# Convert the list of differences in proportional values
# for the general and customers datasets to a series.
gen_cust_diff_proport = pd.Series(gen_cust_diff_proport)

gen_cust_diff_proport
```

Out[100]:
```
0     19273
1     -3299
2      9044
3     21691
4    -12503
5    -19898
6       648
7     -2847
8    -24818
9     -3979
10    -4320
11    21008
dtype: int64
```

In [101…
```
# Creates a dataframe of the clusters prediction proportional data for the
# general and customers models, including the calculated difference column.
gen_cust3 = gen_cust2.copy()
gen_cust3['Difference'] = gen_cust_diff_proport

gen_cust3
```

Out[101]:

	Cluster	General	Customers	Difference
0	1	63784	44511	19273
1	2	80677	83976	-3299
2	3	82523	73479	9044
3	4	53330	31639	21691
4	5	59597	72100	-12503
5	6	43721	63619	-19898
6	7	59675	59027	648
7	8	82724	85571	-2847
8	9	64001	88819	-24818
9	10	67304	71283	-3979
10	11	56181	60501	-4320
11	12	84444	63436	21008

In [102…
```
# Creates a dataframe of the clusters prediction proportional data for the
# general and customers models, including the calculated difference column.
gen_cust4 = gen_cust3.copy()

# Convert the values to absolute numbers for visualization.
gen_cust4.Difference = abs(gen_cust4.Difference).copy()

gen_cust4
```

Out[102]:

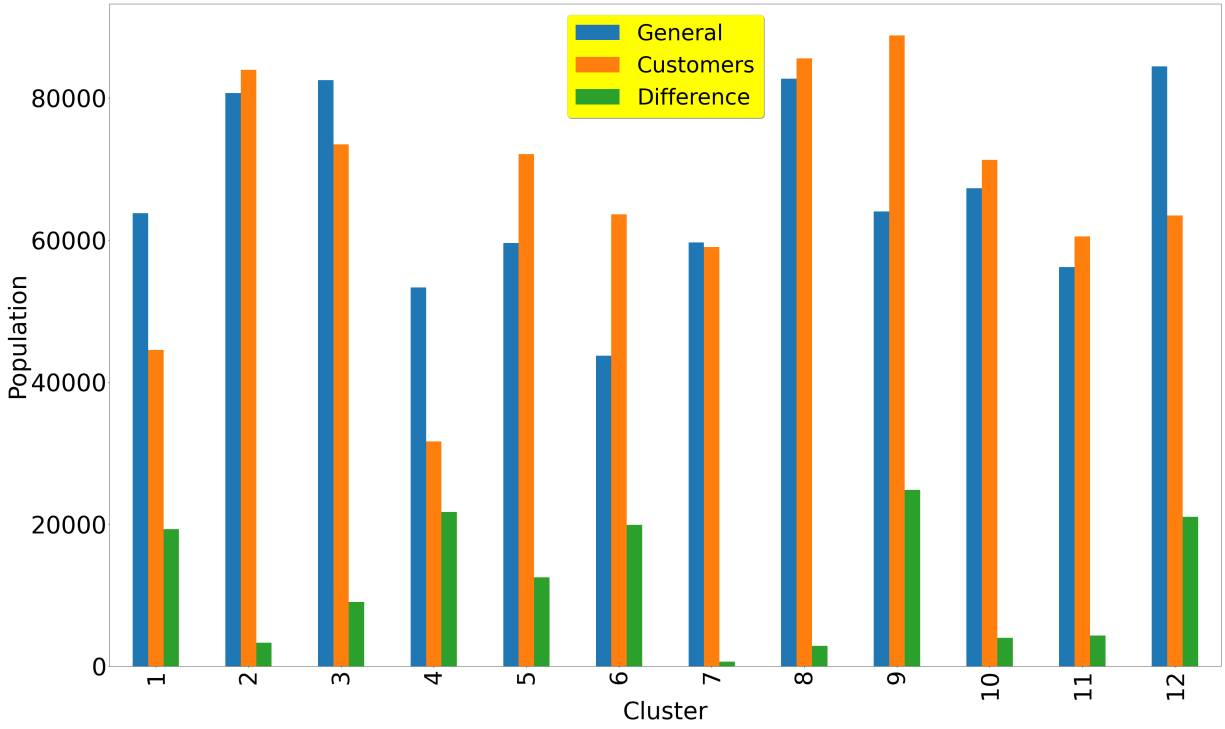
	Cluster	General	Customers	Difference
0	1	63784	44511	19273
1	2	80677	83976	3299
2	3	82523	73479	9044
3	4	53330	31639	21691
4	5	59597	72100	12503
5	6	43721	63619	19898
6	7	59675	59027	648
7	8	82724	85571	2847
8	9	64001	88819	24818
9	10	67304	71283	3979
10	11	56181	60501	4320
11	12	84444	63436	21008

In [103…

```
# Visualization of the clusters prediction proportional data comparing the
# general and customers models, including the calculated difference in values.
clust_comp = gen_cust4.plot(x = 'Cluster', y = ['General', 'Customers', 'Difference

clust_comp.set_xlabel('Cluster', fontsize = 60)
clust_comp.set_ylabel('Population', fontsize = 60)
clust_comp.xaxis.set_tick_params(labelsize = 60)
clust_comp.yaxis.set_tick_params(labelsize = 60)


clust_comp.legend(['General', 'Customers', 'Difference'], prop = {'size' : 55},
                  loc = 'upper center', shadow = True, facecolor = 'yellow')
```

Out[103]:

```
<matplotlib.legend.Legend at 0x16560f87a90>
```



In [104…
```
# Removes the General and Customers columns from the dataframe for further analysis
gen_cust5 = gen_cust3[['Cluster', 'Difference']].sort_values(by = 'Difference',
                                                             ascending = False).res

gen_cust5
```

Out[104]:

	Cluster	Difference
0	4	21691
1	12	21008
2	1	19273
3	3	9044
4	7	648
5	8	-2847
6	2	-3299
7	10	-3979
8	11	-4320
9	5	-12503
10	6	-19898
11	9	-24818

Dataframe displays the cluster number and the difference of proportional values from the the

customers demographic to the general population.

Discussion 3.3: Compare Customer Data to Demographics Data

In [105…
```
# Removes rows (1 - 11) from the dataframe, resulting in the for further analysis.
gen_cust6 = gen_cust5.drop(gen_cust5.index[1:11]).copy()

gen_cust6
```

Out[105]:

	Cluster	Difference
0	4	21691
11	9	-24818

In [106…
```
print('\n\n\t\t\tCluster', gen_cust6.Cluster.values[0], 'is the overrepresented clu
print('\n\n\t\t\tCluster', gen_cust6.Cluster.values[1], 'is the underrepresented cl
```

```
Cluster 4 is the overrepresented cluster.


Cluster 9 is the underrepresented cluster.
```

Overepresented Customer Cluster & Features Details.

In [107…
```
# The types of people are part of a cluster that is overrepresented
# in the customer data compared to the general population.
# Drop the row that is the underrepresented.
overrep = gen_cust6.drop(gen_cust6.index[1]).copy()

overrep
```

Out[107]:

	Cluster	Difference
0	4	21691

In [108…
```
# Retrieve the value from the 1st index of the Cluster column.
overrep_cen = overrep.Cluster.values
overrep_cen = overrep_cen[0]

print("\n\t\tCluster", overrep_cen, "is the overrepresented customer cluster.\n")
```

```
Cluster 4 is the overrepresented customer cluster.
```

In [109…

```
# Inverse transform the pca and the scaler as well as get
# data for the features form the overrepresented cluster.
over_clus = kmeans2.cluster_centers_[overrep_cen - 1]
over_clus1 = pca_components2.inverse_transform(over_clus)
over_clus1 = over_clus1.reshape(1,-1)
overrep_clus = scaler2.inverse_transform(over_clus1)
over_clus2 = overrep_clus[0]
over_clus2 = pd.Series(over_clus2).reset_index()
over_clus2.index = customers.columns
over_clus2 = over_clus2.sort_values(by = over_clus2.columns[0], ascending = True).r
over_clus2 = over_clus2.set_index("index").reset_index(drop = True)

over_clus2.columns.values[0] = "Feature"
over_clus2.columns.values[1] = "Value"
over_clus2.Value = over_clus2.Value.astype('int')

print('\n\tAnalyze 4 of the overrepresented cluster features.')

df100 = pd.DataFrame()

a = over_clus2.Feature[0]
b = over_clus2.Feature[22]
c = over_clus2.Feature[9]
d = over_clus2.Feature[10]

w =  over_clus2.Value[0]
x =  over_clus2.Value[22]
y =  over_clus2.Value[9]
z =  over_clus2.Value[10]

df100['Feature'] = [a, b, c, d]

df100['Value'] = [w, x, y, z]

print('\n\n', df100, '\n')
```

```
	Analyze 4 of the overrepresented cluster features.


                Feature  Value
0  ALTERSKATEGORIE_GROB      3
1            SEMIO_FAM      4
2            FINANZTYP      2
3       GFK_URLAUBERTYP      8
```

In [110…

```
# Displays the ALTERSKATEGORIE_GROB feature value data.
print('\n\t', over_clus2.Feature[0], '\n')
if over_clus2.Value.values[0] == 1:
    print(' is a person that is less than 30 years old.\n')

elif over_clus2.Value.values[0] == 2:
    print(' is a person that is 30 - 45 years old.\n')
```

```
elif over_clus2.Value.values[0] == 3:
    print(' is a person that is 46 - 60 years old\n.')

elif over_clus2.Value.values[0] == 4:
     print(' is a person that is greater than 60 years old.\n')

else:
    print('Age is unknown.\n')
```

```
        ALTERSKATEGORIE_GROB

 is a person that is 46 - 60 years old
.
```

In [111…

```
# Displays the SEMIO_FAM feature value data.
print('\n\t', over_clus2.Feature[22], '\n')
if over_clus2.Value.values[4] == 1:
    print(' is a family-minded individual with highest affinity.\n')

elif over_clus2.Value.values[22] == 2:
    print('is a family-minded individual with very high affinity.\n')

elif over_clus2.Value.values[22] == 3:
    print('is a family-minded individual with high affinity\n.')

elif over_clus2.Value.values[22] == 4:
     print('is a family-minded individual with average affinity.\n')

elif over_clus2.Value.values[22] == 5:
     print('is a family-minded individual with low affinity.\n')

elif over_clus2.Value.values[22] == 6:
     print('is a family-minded individual with very low affinity.\n')

elif over_clus2.Value.values[22] == 7:
     print('is a family-minded individual with lowest affinity.\n')

else:
    print('Affinity is unknown.\n')
```

```
        SEMIO_FAM

is a family-minded individual with average affinity.
```

In [112…

```
# Displays the FINANZTYP feature value data.
print('\n\t', over_clus2.Feature.values[9], '\n')
if over_clus2.Value.values[4] == 1:
    print('has a low financial interest.\n')

elif over_clus2.Value.values[22] == 2:
    print('is a money-savert.\n')

elif over_clus2.Value.values[22] == 3:
    print('has home ownership.\n')
```

```
elif over_clus2.Value.values[22] == 4:
     print('tends to be prepared.\n')

elif over_clus2.Value.values[22] == 5:
     print('is an investor.\n')

elif over_clus2.Value.values[22] == 6:
     print(' has an inconspicuous financial interest.\n')

else:
    print('Financial interest is unknown.\n')
```

```
        FINANZTYP

tends to be prepared.
```

In [113…

```
# Displays the GFK_URLAUBERTYP feature value data.
print('\n\t', over_clus2.Feature.values[10], '\n')
if over_clus2.Value.values[4] == 1:
    print('are event travelers.')

elif over_clus2.Value.values[22] == 2:
    print('are family-oriented vacationiststs.')

elif over_clus2.Value.values[22] == 3:
    print('are winter sportspeople.')

elif over_clus2.Value.values[22] == 4:
     print('are culture lovers.')

elif over_clus2.Value.values[22] == 5:
     print('are nature fans.')

elif over_clus2.Value.values[22] == 6:
     print('is a hiker.')

elif over_clus2.Value.values[22] == 7:
    print('is a golden ager.')

elif over_clus2.Value.values[22] == 8:
    print('is a homeland-connected vacationists.')

elif over_clus2.Value.values[22] == 9:
     print('is a package tour travelers', '')

elif over_clus2.Value.values[22] == 10:
     print('are connoisseurs.')

elif over_clus2.Value.values[22] == 11:
     print('are active families.')

elif over_clus2.Value.values[22] == 12:
     print('has a lifestyle without vacations.')
```

```
else:
    print('Lifestyle is unknown.')
```

```
        GFK_URLAUBERTYP
```

are culture lovers.

Underepresented Customer Cluster & Features Details.

In [114…

```
# The types of people that are part of a cluster that is underrepresented
# in the customer data compared to the general population.
# Drop the row that is the overrepresented.
underrep = gen_cust6.drop(gen_cust6.index[0]).copy()

underrep
```

Out[114]:

	Cluster	Difference
11	9	-24818

In [115…

```
# Retrieve the value from the 1st index of the Cluster column.
underrep_cen = underrep.Cluster.values
underrep_cen = underrep_cen[0]

print("\nCluster", underrep_cen, "is the underrepresented customer cluster.\n")
```

```
Cluster 9 is the underrepresented customer cluster.
```

In [116…

```
# Inverse transform the pca and the scaler as well as get
# data for the features form the underrepresented cluster.
under_clus = kmeans2.cluster_centers_[underrep_cen - 1]
under_clus1 = pca_components2.inverse_transform(under_clus)
under_clus1 = under_clus1.reshape(1,-1)
underrep_clus = scaler2.inverse_transform(under_clus1)
under_clus2 = underrep_clus[0]
under_clus2 = pd.Series(under_clus2).reset_index()
under_clus2.index = customers.columns
under_clus2 = under_clus2.sort_values(by = under_clus2.columns[0], ascending = True
under_clus2 = under_clus2.set_index("index").reset_index(drop = True)

under_clus2.columns.values[0] = "Feature"
under_clus2.columns.values[1] = "Value"
under_clus2.Value = under_clus2.Value.astype('int')

print('\n\tAnalyze 4 of the underrepresented cluster features.')

df100 = pd.DataFrame()

a = under_clus2.Feature[0]
```

```
b = under_clus2.Feature[22]
c = under_clus2.Feature[9]
d = under_clus2.Feature[10]

w =  under_clus2.Value[0]
x =  under_clus2.Value[22]
y =  under_clus2.Value[9]
z =  under_clus2.Value[10]

df100['Feature'] = [a, b, c, d]

df100['Value'] = [w, x, y, z]

print('\t\t\n', df100, '\n')
```

```
       Analyze 4 of the underrepresented cluster features.

                Feature  Value
0  ALTERSKATEGORIE_GROB      3
1             SEMIO_FAM      2
2             FINANZTYP      4
3       GFK_URLAUBERTYP      7
```

In [117…

```
# Displays the ALTERSKATEGORIE_GROB feature value data.
print('\n\t', over_clus2.Feature[0], '\n')
if under_clus2.Value.values[0] == 1:
    print(' is a person that is less than 30 years old.\n')

elif under_clus2.Value.values[0] == 2:
    print(' is a person that is 30 - 45 years old.\n')

elif under_clus2.Value.values[0] == 3:
    print(' is a person that is 46 - 60 years old\n.')

elif under_clus2.Value.values[0] == 4:
     print(' is a person that is greater than 60 years old.\n')

else:
    print('Age is unknown.\n')
```

```
        ALTERSKATEGORIE_GROB

 is a person that is 46 - 60 years old
.
```

In [118…

```
# Displays the SEMIO_FAM feature value data.
print('\n\t', under_clus2.Feature[22], '\n')
if under_clus2.Value.values[4] == 1:
    print(' is a family-minded individual with highest affinity.\n')

elif under_clus2.Value.values[22] == 2:
    print('is a family-minded individual with very high affinity.\n')

elif under_clus2.Value.values[22] == 3:
    print('is a family-minded individual with high affinity\n.')
```

```
elif under_clus2.Value.values[22] == 4:
     print('is a family-minded individual with average affinity.\n')

elif under_clus2.Value.values[22] == 5:
     print('is a family-minded individual with low affinity.\n')

elif under_clus2.Value.values[22] == 6:
     print('is a family-minded individual with very low affinity.\n')

elif under_clus2.Value.values[22] == 7:
     print('is a family-minded individual with lowest affinity.\n')

else:
    print('Affinity is unknown.\n')
```

```
        SEMIO_FAM

is a family-minded individual with highest affinity.
```

In [119…

```
# Displays the FINANZTYP feature value data.
print('\n\t', under_clus2.Feature.values[9], '\n')
if under_clus2.Value.values[4] == 1:
    print('has a low financial interest.\n')

elif under_clus2.Value.values[22] == 2:
    print('is a money-savert.\n')

elif under_clus2.Value.values[22] == 3:
    print('has home ownership.\n')

elif under_clus2.Value.values[22] == 4:
     print('tends to be prepared.\n')

elif under_clus2.Value.values[22] == 5:
     print('is an investor.\n')

elif under_clus2.Value.values[22] == 6:
     print(' has an inconspicuous financial interest.\n')

else:
    print('Financial interest is unknown.\n')
```

```
        FINANZTYP

has a low financial interest.
```

In [120…

```
# Displays the GFK_URLAUBERTYP feature value data.
print('\n\t', under_clus2.Feature.values[10], '\n')
if under_clus2.Value.values[4] == 1:
    print('are event travelers.\n')

elif under_clus2.Value.values[22] == 2:
    print('are family-oriented vacationistsis.\n')

elif under_clus2.Value.values[22] == 3:
```

```
    print('are winter sportspeople.\n')

elif under_clus2.Value.values[22] == 4:
     print('are culture lovers.\n')

elif under_clus2.Value.values[22] == 5:
     print('are nature fans.\n')

elif under_clus2.Value.values[22] == 6:
     print('is a hiker.\n')

elif under_clus2.Value.values[22] == 7:
    print('is a golden ager.\n')

elif under_clus2.Value.values[22] == 8:
    print('is a homeland-connected vacationists.\n')

elif under_clus2.Value.values[22] == 9:
     print('is a package tour travelers', '\n')

elif under_clus2.Value.values[22] == 10:
     print('are connoisseurs.\n')

elif under_clus2.Value.values[22] == 11:
     print('are active families.\n')

elif under_clus2.Value.values[22] == 12:
     print('has a lifestyle without vacations.\n')

else:
    print('Lifestyle is unknown.\n')
```

```
        GFK_URLAUBERTYP

are event travelers.
```