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
from typing import List


# magic word for producing visualizations in notebook
%matplotlib inline

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

Out[1]:
```
'\nImport note: The classroom currently uses sklearn version 0.19.\nIf you n
eed to use an imputer, it is available in sklearn.preprocessing.Imputer,\nin
stead of sklearn.impute as in newer versions of sklearn.\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()` call to read in the data properly. Also, considering the size of the main dataset, it may take some time for it to load completely.

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

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

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

In [3]:
```
# Check the structure of the data after it's loaded (e.g. print the number c
# rows and columns, print the first few rows).

azdias.head(20)
```

Out[3]:

	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	3	1	2	2.0	
6	-1	2	2	5.0	
7	-1	1	1	3.0	
8	-1	3	1	3.0	
9	-1	3	2	4.0	
10	0	3	2	1.0	
11	-1	2	1	6.0	
12	-1	3	1	6.0	
13	-1	1	2	5.0	
14	-1	3	1	6.0	
15	1	4	2	4.0	
16	-1	1	2	1.0	
17	-1	2	1	6.0	
18	-1	2	2	6.0	
19	-1	3	1	3.0	

20 rows × 85 columns

In [4]:
```
# How many rows and columns in 'azdias'
azdias.shape
```

Out[4]:
```
(891221, 85)
```

In [5]:
```
# What type of data and missing values?
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 [6]:

```
# view the first few rows of 'feat_info'
feat_info.head(20)
```

Out[6]:

	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]

In [7]:

```
# How many rows and columns in 'feat_info'?
feat_info.shape
```

Out[7]:

```
(85, 4)
```

In [8]:

```
# What type of data in feat_info? Are there missing values?
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
```

> **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 [9]:
```
azdias.isnull().sum().sum()
```

Out[9]:
```
4896838
```

In [10]:
```
# Identify missing or unknown data values and convert them to NaNs.

# Create a copy of the dataset
azdias_clean = azdias.copy()

# Define missing and unknown values
missing_unknown_values = ['0', '-1', 'XX', 'X']

# Iterate through each column in 'feat_info'
for index, row in feat_info.iterrows():
    attribute = row['attribute']
    vals = row['missing_or_unknown']

    # Convert the string representation of the list to a list of integers an
    vals = [int(val) if val.lstrip('-').isdigit() else val for val in vals.s
    vals.extend(missing_unknown_values)

    # Replace the missing or unknown values with NaN in the DataFrame
    azdias_clean[attribute].replace(vals, np.nan, inplace=True)
```

In [11]:
```
azdias_clean.sample(25)
```

Out[11]:

	AGER_TYP	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIM
40965	2.0	3.0	1	5.0	
641034	NaN	3.0	2	4.0	
867785	NaN	3.0	1	2.0	
262625	2.0	3.0	1	1.0	
792975	NaN	2.0	1	4.0	
203841	NaN	3.0	1	6.0	
781782	1.0	4.0	1	3.0	
681787	NaN	3.0	2	4.0	
644919	NaN	2.0	1	1.0	
9251	1.0	4.0	1	3.0	
121841	1.0	4.0	1	6.0	
329418	NaN	3.0	2	6.0	
587049	NaN	4.0	1	2.0	
314056	2.0	4.0	2	2.0	
714071	NaN	1.0	2	6.0	
545298	NaN	1.0	1	4.0	
202475	NaN	3.0	1	2.0	
687099	NaN	2.0	2	6.0	
236081	2.0	4.0	1	1.0	
433450	1.0	4.0	2	2.0	
686382	NaN	4.0	1	4.0	
799912	1.0	4.0	2	2.0	
344854	NaN	4.0	2	1.0	
615916	NaN	3.0	1	5.0	
224609	NaN	3.0	1	4.0	

25 rows × 85 columns

In [12]:

```
azdias_clean.isnull().sum().sum()
```

Out[12]:

```
8373929
```

Step 1.1.2: Assess Missing Data in Each Column

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

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

In [13]:
```
# Perform an assessment of how much missing data there is in each column of
# dataset.

# Calculate the proportion of missing values for each column
missing_data_proportion = azdias_clean.isnull().mean()

# Calculate the count of missing values for each column
missing_data_count = azdias_clean.isnull().sum()

# Combine the results into a DataFrame for better representation
missing_data_info = pd.DataFrame({
    'Proportion of Missing Values': missing_data_proportion,
    'Count of Missing Values': missing_data_count
})

# Display the information about missing data for each column
print(missing_data_info)
```

```
                      Proportion of Missing Values  Count of Missing Values
AGER_TYP                                  0.769554                   685843
ALTERSKATEGORIE_GROB                      0.003233                     2881
ANREDE_KZ                                 0.000000                        0
CJT_GESAMTTYP                             0.005446                     4854
FINANZ_MINIMALIST                         0.000000                        0
...                                            ...                      ...
PLZ8_HHZ                                  0.130736                   116515
PLZ8_GBZ                                  0.130736                   116515
ARBEIT                                    0.109260                    97375
ORTSGR_KLS9                               0.109147                    97274
RELAT_AB                                  0.109260                    97375

[85 rows x 2 columns]
```

In [14]:
```
missing_data_info.sort_values(by='Proportion of Missing Values', ascending=
```

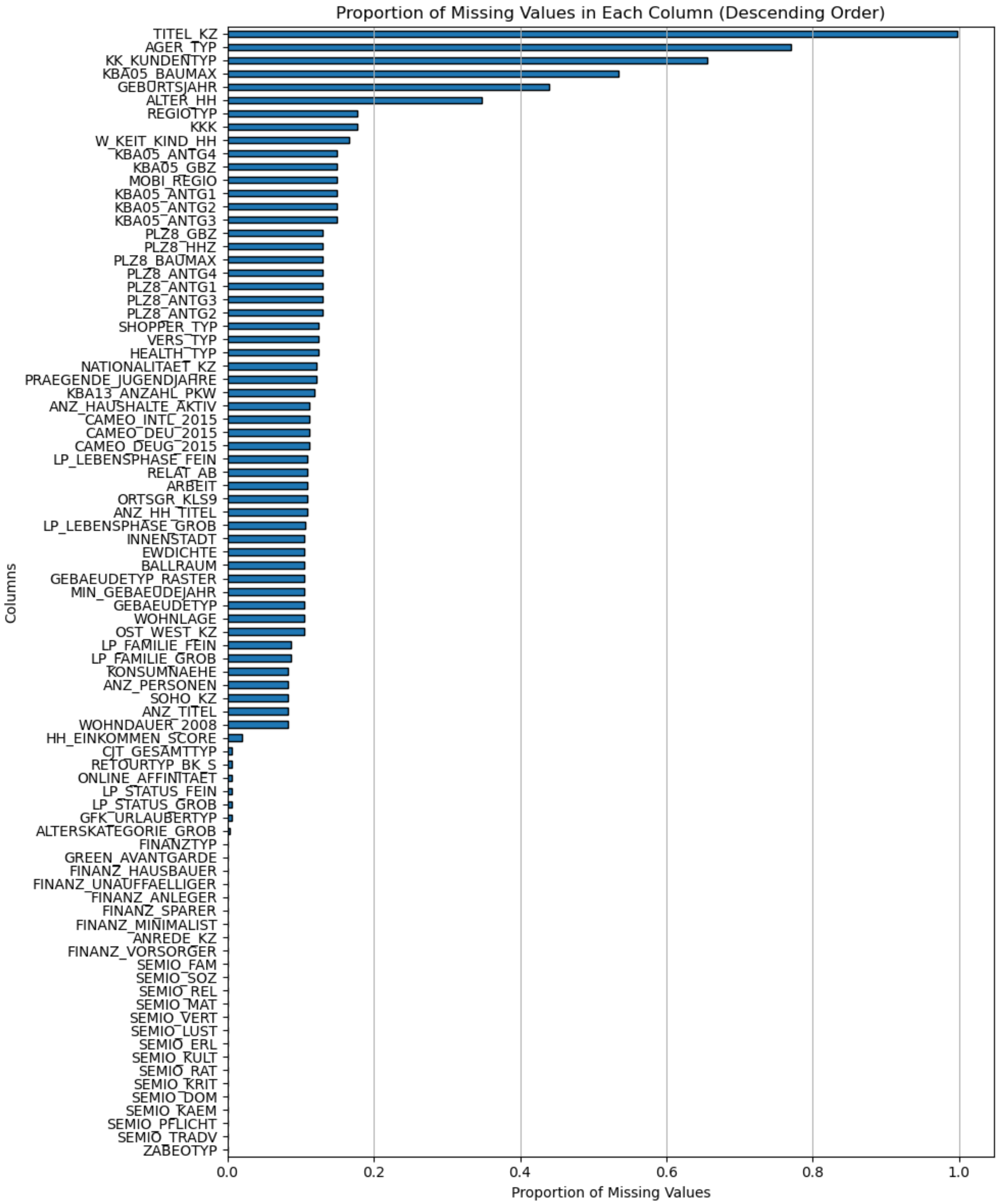
Out[14]:

	Proportion of Missing Values	Count of Missing Values
TITEL_KZ	0.997576	889061
AGER_TYP	0.769554	685843
KK_KUNDENTYP	0.655967	584612
KBA05_BAUMAX	0.534687	476524
GEBURTSJAHR	0.440203	392318
...	...	...
SEMIO_RAT	0.000000	0
SEMIO_KRIT	0.000000	0
SEMIO_DOM	0.000000	0
SEMIO_TRADV	0.000000	0
ZABEOTYP	0.000000	0

85 rows × 2 columns

In [15]:

```
# Plot a bar chart for the proportion of missing values in each column, in c

# Sort the missing_data_proportion in descending order
sorted_missing_data_proportion = missing_data_proportion.sort_values()

# Set a wider figure size
plt.figure(figsize=(10, 12))  # Adjust the figure size to your preference

# Plot a horizontal bar chart for the proportion of missing values in each c
sorted_missing_data_proportion.plot(kind='barh', edgecolor='black')
plt.xlabel('Proportion of Missing Values')
plt.ylabel('Columns')
plt.title('Proportion of Missing Values in Each Column (Descending Order)')
plt.grid(axis='x')  # Display grid along x-axis
plt.tight_layout()
plt.show()
```



In [16]:
```
# Remove the outlier columns from the dataset. (You'll perform other data
# engineering tasks such as re-encoding and imputation later.)
columns_to_drop = ['TITEL_KZ', 'AGER_TYP', 'KK_KUNDENTYP', 'KBA05_BAUMAX', '
azdias_clean.drop(labels = columns_to_drop, axis = 1, inplace = True)
```

Discussion 1.1.2: Assess Missing Data in Each Column

Observations:

>

> There were many examples of data that was missing, although it was not a typical 'null' value. After removing the '-1', '0', 'XX' and 'X' values (by converting them to typical 'NaN' values), which were a representation of missing or unknown values, the amount of null values grew from 4,896,838 to 8,373,929.
>
> In the visualization above, it is clear to see that the top six columns are missing a sizeable amount of data. Beyond the top six columns, the amount of missing data makes a significant drop, falling to less than 20% missing data. I will now remove these six columns, as I would consider that too much missing data compared to the rest of the columns. The columns removed were 'TITEL_KZ', 'AGER_TYP', 'KK_KUNDENTYP', 'KBA05_BAUMAX', 'GEBURTSJAHR', and 'ALTER_HH'.

Step 1.1.3: Assess Missing Data in Each Row

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

In order to know what to do with the outlier rows, we should see if the distribution of data values on columns that are not missing data (or are missing very little data) are similar or different between the two groups. Select at least five of these columns and compare the distribution of values.

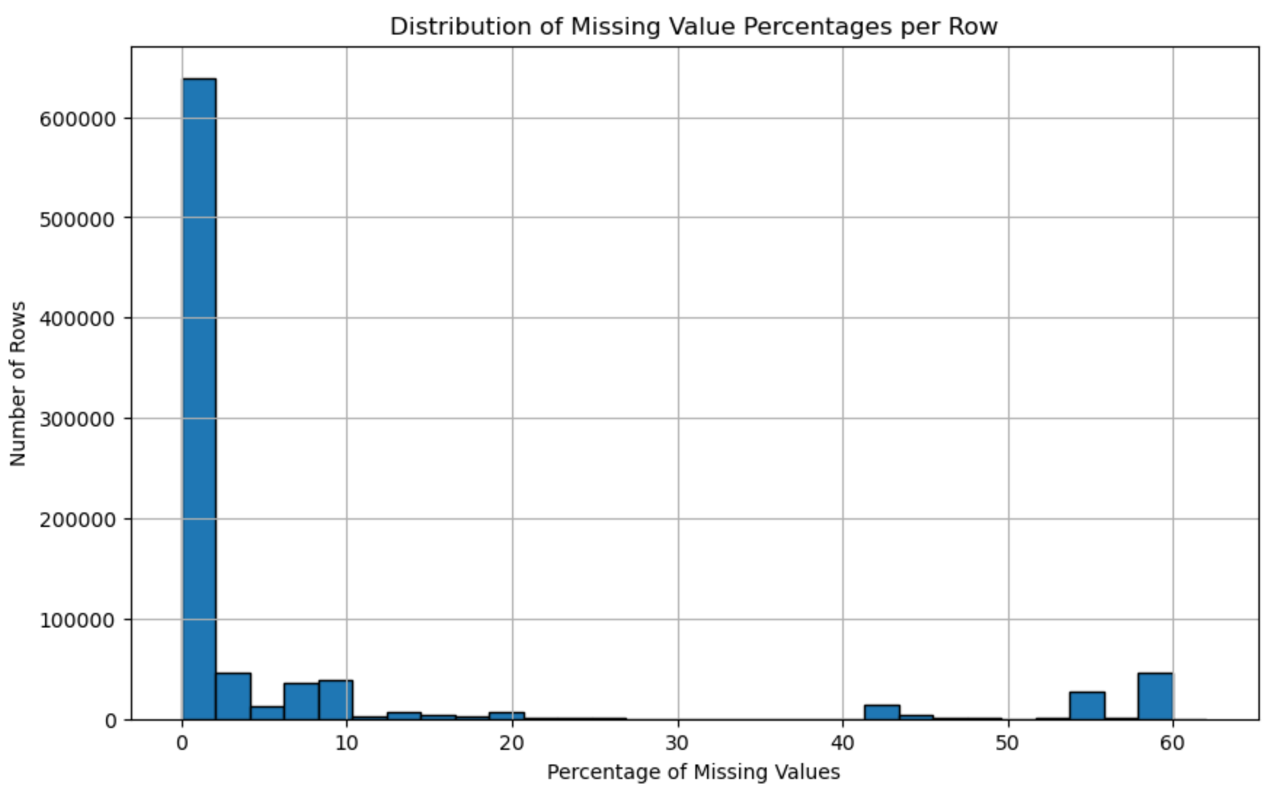
- You can use seaborn's `countplot()` function to create a bar chart of code frequencies and matplotlib's `subplot()` function to put bar charts for the two subplots side by side.
- To reduce repeated code, you might want to write a function that can perform this comparison, taking as one of its arguments a column to be compared.

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

In [17]:
```
# How much data is missing in each row of the dataset?

# Lets display the percentage of missing data from each row.b

# Create a binary matrix indicating missing values (1 if missing, 0 if not)
missing_values_indicator = azdias_clean.isnull().astype(int)

# Calculate the percentage of missing values for each row
percentage_missing_per_row = missing_values_indicator.mean(axis=1) * 100

# Print the resulting Series with the percentage of missing values for each
print(percentage_missing_per_row)
```

```
0          54.430380
1           0.000000
2           0.000000
3           8.860759
4           0.000000
             ...
891216      3.797468
891217      5.063291
891218      6.329114
891219      0.000000
891220      0.000000
Length: 891221, dtype: float64
```

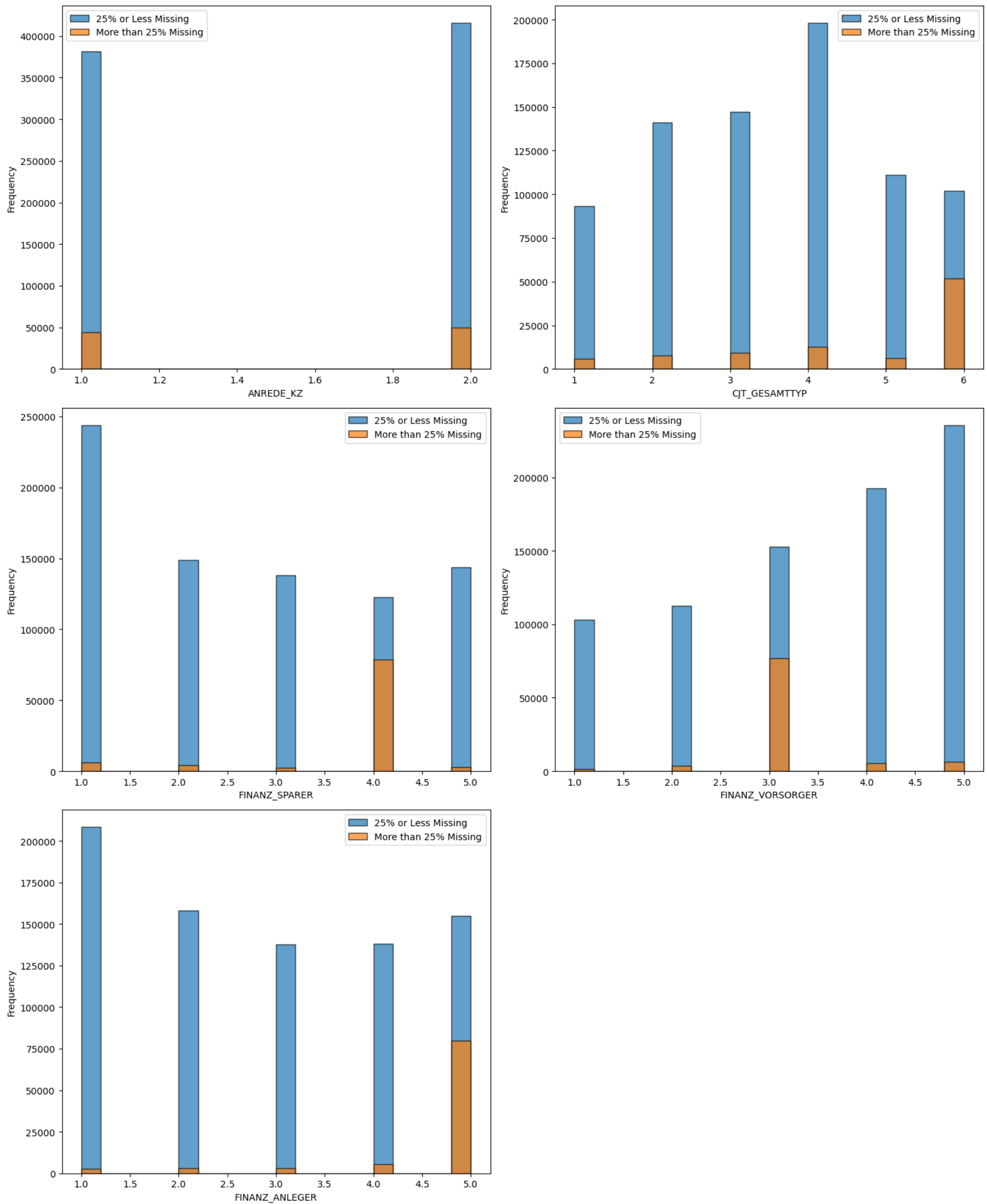
In [18]:
```
# Plot a histogram for the distribution of missing value percentages per row

plt.figure(figsize=(10, 6))
plt.hist(percentage_missing_per_row, bins=30, edgecolor='black')
plt.xlabel('Percentage of Missing Values')
plt.ylabel('Number of Rows')
plt.title('Distribution of Missing Value Percentages per Row')
plt.grid(True)
plt.show()
```



In [19]:
```
# Write code to divide the data into two subsets based on the number of miss
# values in each row.

# Define the threshold (25%) for splitting the data
threshold = 25

# Subset with rows having 25% or less missing values
less_than_25_missing = azdias_clean[percentage_missing_per_row <= threshold]

# Subset with rows having more than 25% missing values
more_than_25_missing = azdias_clean[percentage_missing_per_row > threshold]

# Display the shape of the subsets to show the number of rows in each subset
print('Subset with 25% or less missing values:', less_than_25_missing.shape)
print('Subset with more than 25% missing values:', more_than_25_missing.shap
```

```
Subset with 25% or less missing values: (797077, 79)
Subset with more than 25% missing values: (94144, 79)
```

In [20]:
```
# Compare the distribution of values for at least five columns where there a
# no or few missing values, between the two subsets.

# Define the threshold (5%) for columns with no or few missing values
threshold_missing_values = 5

# Identify columns with no or few missing values
columns_with_low_missing_indices = np.where(percentage_missing_per_row < thr

# Ensure we have at least five columns, if available
num_columns_for_comparison = min(5, len(columns_with_low_missing_indices))

# Visualize distributions for the selected columns in both subsets
plt.figure(figsize=(15, 30))

for i in range(num_columns_for_comparison):
    col_index = columns_with_low_missing_indices[i]
    col_name = azdias_clean.columns[col_index]
    plt.subplot(5, 2, i+1)
    plt.hist(less_than_25_missing[col_name].dropna(), bins=20, edgecolor='bl
    plt.hist(more_than_25_missing[col_name].dropna(), bins=20, edgecolor='bl
    plt.xlabel(col_name)
    plt.ylabel('Frequency')
    plt.legend()

plt.tight_layout()
plt.show()
```


25% or Less Missing
More than 25% Missing
Frequency
ANREDE_KZ
CJT_GESAMTTYP
FINANZ_SPARER
FINANZ_VORSORGER
FINANZ_ANLEGER

Discussion 1.1.3: Assess Missing Data in Each Row

>

> As seen in the randomly selected columns presented earlier, the distribution of values in the dataset containing few missing values is significantly different than the dataset containing a higher number of missing values.

Step 1.2: Select and Re-Encode Features

Checking for missing data isn't the only way in which you can prepare a dataset for analysis. Since the unsupervised learning techniques to be used will only work on data that is encoded numerically, you need to make a few encoding changes or additional assumptions to be able to make progress. In addition, while almost all of the values in the dataset are encoded using numbers, not all of them represent numeric values. Check the third column of the feature summary (`feat_info`) for a summary of types of measurement.

- For numeric and interval data, these features can be kept without changes.
- Most of the variables in the dataset are ordinal in nature. While ordinal values may technically be non-linear in spacing, make the simplifying assumption that the ordinal variables can be treated as being interval in nature (that is, kept without any changes).
- Special handling may be necessary for the remaining two variable types: categorical, and 'mixed'.

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

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

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

Out [21]:

	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 [22]:

```
# How many features are there of each data type?
feat_info.type.value_counts()
```

Out[22]:

```
type
ordinal        49
categorical    21
numeric         7
mixed           7
interval        1
Name: count, dtype: int64
```

Step 1.2.1: Re-Encode Categorical Features

For categorical data, you would ordinarily need to encode the levels as dummy variables. Depending on the number of categories, perform one of the following:

- For binary (two-level) categoricals that take numeric values, you can keep them without needing to do anything.
- There is one binary variable that takes on non-numeric values. For this one, you need to re-encode the values as numbers or create a dummy variable.
- For multi-level categoricals (three or more values), you can choose to encode the values using multiple dummy variables (e.g. via OneHotEncoder), or (to keep things straightforward) just drop them from the analysis. As always, document your choices in the Discussion section.

In [23]:

```
# Assess categorical variables: which are binary, which are multi-level, anc
# which one needs to be re-encoded?

feat_info.query('type == "categorical"').attribute.values
```

Out[23]:

```
array(['AGER_TYP', 'ANREDE_KZ', 'CJT_GESAMTTYP', 'FINANZTYP',
       'GFK_URLAUBERTYP', 'GREEN_AVANTGARDE', 'LP_FAMILIE_FEIN',
       'LP_FAMILIE_GROB', 'LP_STATUS_FEIN', 'LP_STATUS_GROB',
       'NATIONALITAET_KZ', 'SHOPPER_TYP', 'SOHO_KZ', 'TITEL_KZ',
       'VERS_TYP', 'ZABEOTYP', 'KK_KUNDENTYP', 'GEBAEUDETYP',
       'OST_WEST_KZ', 'CAMEO_DEUG_2015', 'CAMEO_DEU_2015'], dtype=object)
```

Note:

> The cell above displays columns that contain categorical features. I noticed 'AGER_TYP' was dropped from the 'azdias' dataframe earlier. I have decided to check and remove the other values/columns that were dropped from 'azdias', and remove them from 'feat_info'.

In [24]:
```
# Drop the rows where 'attribute' matches the items in 'columns_to_drop'
feat_info_clean = feat_info[~feat_info['attribute'].isin(columns_to_drop)]

# Print the DataFrame to verify the rows have been dropped
feat_info_clean.shape
```

Out[24]:
```
(79, 4)
```

In [25]:
```
# Create an array of the columns with categorical features.
categorical_features = feat_info_clean.query('type == "categorical"').attrib
categorical_features
```

Out[25]:
```
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 [26]:
```
# Create a copy of the df with less than 25% missing values to be used with
azdias_clean = less_than_25_missing.copy()
```

In [27]:
```
# Create a function to iterate over the columns and determine if they contai
# mulit-level, or mixed (non-numeric) data.

def identify_categorical_features(dataframe, categorical_features):
    """
    Identifies binary, multi-level, and non-numeric categorical features bas

    Parameters:
    dataframe (pd.DataFrame): The DataFrame containing the data.
    categorical_features (list): List of categorical feature names.

    Returns:
    binary_features (list): List of binary categorical feature names.
    multilevel_features (list): List of multi-level categorical feature name
    non_numeric_features (list): List of non-numeric categorical feature nam
    """
    binary_features = []
    multilevel_features = []
    non_numeric_features = []

    for feature in categorical_features:
        unique_values = dataframe[feature].nunique()
        if unique_values == 2:
            binary_features.append(feature)
        elif unique_values > 2:
            multilevel_features.append(feature)

        # Check if the feature is non-numeric (object dtype)
        if dataframe[feature].dtype == 'object':
            non_numeric_features.append(feature)

    return binary_features, multilevel_features, non_numeric_features


binary_features, multilevel_features, non_numeric_features = identify_catego

print("Binary Categorical Features:", binary_features)
print("Multi-level Categorical Features:", multilevel_features)
print("Non-numeric Features:", non_numeric_features)
```

```
Binary Categorical Features: ['ANREDE_KZ', 'GREEN_AVANTGARDE', 'SOHO_KZ', 'V
ERS_TYP', 'OST_WEST_KZ']
Multi-level Categorical Features: ['CJT_GESAMTTYP', 'FINANZTYP', 'GFK_URLAUB
ERTYP', 'LP_FAMILIE_FEIN', 'LP_FAMILIE_GROB', 'LP_STATUS_FEIN', 'LP_STATUS_G
ROB', 'NATIONALITAET_KZ', 'SHOPPER_TYP', 'ZABEOTYP', 'GEBAEUDETYP', 'CAMEO_D
EUG_2015', 'CAMEO_DEU_2015']
Non-numeric Features: ['OST_WEST_KZ', 'CAMEO_DEUG_2015', 'CAMEO_DEU_2015']
```

In [28]:
```
# Check for success, view the value counts, look for problems
for col in binary_features:
    print(azdias_clean[col].value_counts())
```

```
ANREDE_KZ
2    415578
1    381499
Name: count, dtype: int64
GREEN_AVANTGARDE
0    621942
1    175135
Name: count, dtype: int64
SOHO_KZ
0.0    790370
1.0      6707
Name: count, dtype: int64
VERS_TYP
2.0    394116
1.0    366623
Name: count, dtype: int64
OST_WEST_KZ
W    628695
O    168382
Name: count, dtype: int64
```

In [29]:

```
# Replace the 'W' and '0' values with '0' and '1'

azdias_clean['OST_WEST_KZ'].replace(['W', 'O'], [1, 0], inplace=True)
```

In [30]:

```
# Check for success
azdias_clean.OST_WEST_KZ.value_counts()
```

Out[30]:

```
OST_WEST_KZ
1    628695
0    168382
Name: count, dtype: int64
```

In [31]:

```
# Check for success, view the value counts, look for problems
for col in multilevel_features:
    print(azdias_clean[col].value_counts())
```

```
CJT_GESAMTTYP
4.0    198089
3.0    147068
2.0    141166
5.0    111032
6.0    101898
1.0     93192
Name: count, dtype: int64
FINANZTYP
6    289004
1    196805
5    106220
2    104577
4     55874
3     44597
Name: count, dtype: int64
```

```
GFK_URLAUBERTYP
12.0    129983
10.0    102748
8.0      82992
11.0     75051
5.0      70468
4.0      60413
9.0      57046
3.0      53094
1.0      50640
2.0      43647
7.0      40642
6.0      25721
Name: count, dtype: int64
LP_FAMILIE_FEIN
1.0     402248
10.0    128902
2.0      98491
11.0     48727
8.0      21777
7.0      19568
4.0      11573
5.0      11164
9.0      10451
6.0       8512
3.0       4682
Name: count, dtype: int64
LP_FAMILIE_GROB
1.0    402248
5.0    188080
2.0     98491
4.0     49857
3.0     27419
Name: count, dtype: int64
LP_STATUS_FEIN
1.0     206766
9.0     136229
10.0    111538
2.0     111016
4.0      73938
3.0      68893
6.0      28870
5.0      27472
8.0      18525
7.0       9198
Name: count, dtype: int64
LP_STATUS_GROB
1.0    317782
2.0    170303
4.0    154754
5.0    111538
3.0     38068
Name: count, dtype: int64
```

```
NATIONALITAET_KZ
1.0    667356
2.0     63619
3.0     32537
Name: count, dtype: int64
SHOPPER_TYP
1.0    247152
2.0    205874
3.0    180604
0.0    127109
Name: count, dtype: int64
ZABEOTYP
3    281772
4    207383
1    123270
5     80892
6     70817
2     32943
Name: count, dtype: int64
GEBAEUDETYP
1.0    459844
3.0    178507
8.0    152439
2.0      4789
4.0       885
6.0       612
5.0         1
Name: count, dtype: int64
CAMEO_DEUG_2015
8    134394
9    108138
6    105819
4    103814
3     86612
2     83149
7     77888
5     55216
1     36180
Name: count, dtype: int64
CAMEO_DEU_2015
6B    56642
8A    52427
4C    47765
2D    35047
3C    34740
7A    34384
3D    34275
8B    33424
4A    33128
8C    30978
9D    28591
9B    27661
9C    24986
```

```
7B    24489
9A    20537
2C    19408
8D    17565
6E    16104
2B    15468
5D    14934
6C    14815
2A    13226
5A    12153
1D    11908
1A    10837
3A    10454
5B    10345
5C     9926
7C     9059
4B     9038
4D     8565
3B     7143
6A     6799
9E     6363
6D     6068
6F     5391
7D     5329
4E     5318
1E     5057
7E     4627
1C     4310
5F     4281
1B     4068
5E     3577
Name: count, dtype: int64
```

In [32]:

```python
# Check for success, view the value counts, look for problems
for col in non_numeric_features:
    print(azdias_clean[col].value_counts())
```

```
OST_WEST_KZ
1    628695
0    168382
Name: count, dtype: int64
CAMEO_DEUG_2015
8    134394
9    108138
6    105819
4    103814
3     86612
2     83149
7     77888
5     55216
1     36180
Name: count, dtype: int64
CAMEO_DEU_2015
```

```
6B    56642
8A    52427
4C    47765
2D    35047
3C    34740
7A    34384
3D    34275
8B    33424
4A    33128
8C    30978
9D    28591
9B    27661
9C    24986
7B    24489
9A    20537
2C    19408
8D    17565
6E    16104
2B    15468
5D    14934
6C    14815
2A    13226
5A    12153
1D    11908
1A    10837
3A    10454
5B    10345
5C     9926
7C     9059
4B     9038
4D     8565
3B     7143
6A     6799
9E     6363
6D     6068
6F     5391
7D     5329
4E     5318
1E     5057
7E     4627
1C     4310
5F     4281
1B     4068
5E     3577
Name: count, dtype: int64
```

In [33]:
```
azdias_clean.isnull().sum().sum()
```

Out[33]:
```
992482
```

In [34]:
```
# Re-encode categorical variable(s) to be kept in the analysis.


# Use get_dummies() to perform one-hot encoding.
azdias_encoded = pd.get_dummies(azdias_clean, columns = multilevel_features)

azdias_encoded.shape
```

Out[34]:
```
(797077, 194)
```

In [35]:
```
azdias_encoded.head()
```

Out[35]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	FINANZ_MINIMALIST	FINANZ_SPARER	FINANZ_VO
1	1.0	2	1	5	
2	3.0	2	1	4	
3	4.0	2	4	2	
4	3.0	1	4	3	
5	1.0	2	3	1	

5 rows × 194 columns

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?)

> After sifting through the data held in all the columns, and comparing it to the information held in the Data_Dictionary.md file, I was able to make some decisions. First I noticed in the list of columns that were had binary features, there was one column that held non-numeric (yet still binary) data, OST_WEST_KZ. I then converted the 'W' and 'O' values to '1' and '0', so that they would be standardized. I then looked at the value counts of the columns in the multi-level list I made, and noticed all but one held numeric values, CAMEO_DEU_2015. I then decided to use create dummy values for all the columns in the multi-level list, thus making everything standardized. After performing these operations, the 'azdias_clean' dataframe now has 194 columns.

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.

From the Data_Dictionary.md file

1.18. PRAEGENDE_JUGENDJAHRE

Dominating movement of person's youth (avantgarde vs. mainstream; east vs. west)

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

In [36]:
```
# Investigate "PRAEGENDE_JUGENDJAHRE" and engineer two new variables.

azdias_encoded.PRAEGENDE_JUGENDJAHRE.value_counts()
```

Out[36]:
```
PRAEGENDE_JUGENDJAHRE
14.0    182833
8.0     141504
10.0     85746
5.0      84649
3.0      53811
15.0     42500
11.0     35729
9.0      33560
6.0      25649
12.0     24436
1.0      20639
4.0      20450
2.0       7479
13.0      5759
7.0       4009
Name: count, dtype: int64
```

In [37]:
```python
azdias_encoded[['PRAEGENDE_JUGENDJAHRE']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 797077 entries, 1 to 891220
Data columns (total 1 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   PRAEGENDE_JUGENDJAHRE  768753 non-null  float64
dtypes: float64(1)
memory usage: 12.2 MB
```

In [38]:
```python
azdias_encoded.shape
```

Out[38]:
```
(797077, 194)
```

In [39]:
```python
# From the information above, create a 'movement' column with Mainstream = 0

# Assign '0' or '1' for 'Mainstream' or 'Avantgarde'in a dictionary
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 : 0, 15 : 1}

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

In [40]:
```python
# Create a column for the the decades from the 'PRAEGENDE_JUGENDJAHRE' colum

# Assign a '1', '2', '3', '4', '5', or '6' depending on the decade in a dict
decades = {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, 15 : 6}

azdias_encoded['DECADE'] = azdias_encoded['PRAEGENDE_JUGENDJAHRE']
azdias_encoded['DECADE'].replace(decades, inplace=True)
```

In [41]:
```
# Investigate "CAMEO_INTL_2015" and engineer two new variables.

azdias_encoded.CAMEO_INTL_2015.value_counts()
```

Out[41]:
```
CAMEO_INTL_2015
51    133665
41     92297
24     91070
14     62833
43     56642
54     45366
25     39593
22     33128
23     26635
13     26305
45     26122
55     23928
52     20537
31     18952
34     18511
15     16965
44     14815
12     13226
35     10349
32     10345
33      9926
Name: count, dtype: int64
```

In [42]:
```
azdias_encoded[['CAMEO_INTL_2015']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 797077 entries, 1 to 891220
Data columns (total 1 columns):
 #   Column           Non-Null Count   Dtype 
---  ------           --------------   ----- 
 0   CAMEO_INTL_2015  791210 non-null  object
dtypes: object(1)
memory usage: 12.2+ MB
```

In [ ]:
```

```

In [43]:
```
# Create 'WEALTH' values in a dictionary
wealth = {'11':1, '12':1, '13':1, '14':1, '15':1, '21':2, '22':2, '23':2, '2
               '31':3, '32':3, '33':3, '34':3, '35':3, '41':4, '42':4, '43':
               '51':5, '52':5, '53':5, '54':5, '55':5}

# Create the 'WEALTH' feature
azdias_encoded['WEALTH'] = azdias_encoded['CAMEO_INTL_2015']
azdias_encoded['WEALTH'].replace(wealth, inplace = True)
```

In [44]:
```
# Create 'LIFE_STAGE' values in a dictionary
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, '
                   '51':1, '52':2, '53':3, '54':4, '55':5}

# Create the 'LIFE_STAGE' feature
azdias_encoded['LIFE_STAGE'] = azdias_encoded['CAMEO_INTL_2015']
azdias_encoded['LIFE_STAGE'].replace(life_stage, inplace = True)
```

In [45]:
```
# Drop the original 'PRAEGENDE_JUGENDJAHRE' and 'CAMEO_INTL_2015' columns

azdias_encoded.drop(['PRAEGENDE_JUGENDJAHRE', 'CAMEO_INTL_2015'], axis=1, in
```

In [46]:
```
# Check for remaining mixed features
feat_info[feat_info.type == "mixed"]
```

Out[46]:

	attribute	information_level	type	missing_or_unknown
15	LP_LEBENSPHASE_FEIN	person	mixed	[0]
16	LP_LEBENSPHASE_GROB	person	mixed	[0]
22	PRAEGENDE_JUGENDJAHRE	person	mixed	[-1,0]
56	WOHNLAGE	building	mixed	[-1]
59	CAMEO_INTL_2015	microcell_rr4	mixed	[-1,XX]
64	KBA05_BAUMAX	microcell_rr3	mixed	[-1,0]
79	PLZ8_BAUMAX	macrocell_plz8	mixed	[-1,0]

In [47]:
```
# Drop remaining columns with mixed values
azdias_encoded.drop(['LP_LEBENSPHASE_FEIN', 'LP_LEBENSPHASE_GROB', 'WOHNLAGE
```

In [48]:
```
# Check for success by iterating through each column and displaying value co
# also check for other possible issues
for column in azdias_encoded.columns:
    print(f"Value counts for column '{column}':")
    print(azdias_encoded[column].value_counts())
    print("\n")
```

```
Value counts for column 'ALTERSKATEGORIE_GROB':
ALTERSKATEGORIE_GROB
3.0    309965
4.0    222989
2.0    137019
1.0    124331
Name: count, dtype: int64


Value counts for column 'ANREDE_KZ':
```

```
ANREDE_KZ
2    415578
1    381499
Name: count, dtype: int64


Value counts for column 'FINANZ_MINIMALIST':
FINANZ_MINIMALIST
3    180406
5    161053
4    159416
2    157692
1    138510
Name: count, dtype: int64


Value counts for column 'FINANZ_SPARER':
FINANZ_SPARER
1    243945
2    148756
5    143758
3    138005
4    122613
Name: count, dtype: int64


Value counts for column 'FINANZ_VORSORGER':
FINANZ_VORSORGER
5    235727
4    192592
3    152920
2    112725
1    103113
Name: count, dtype: int64


Value counts for column 'FINANZ_ANLEGER':
FINANZ_ANLEGER
1    208297
2    158073
5    154809
4    138063
3    137835
Name: count, dtype: int64


Value counts for column 'FINANZ_UNAUFFAELLIGER':
FINANZ_UNAUFFAELLIGER
1    220322
2    183736
3    161242
5    119113
4    112664
```

```
Name: count, dtype: int64


Value counts for column 'FINANZ_HAUSBAUER':
FINANZ_HAUSBAUER
5    183897
2    166096
3    157505
4    156536
1    133043
Name: count, dtype: int64


Value counts for column 'GREEN_AVANTGARDE':
GREEN_AVANTGARDE
0    621942
1    175135
Name: count, dtype: int64


Value counts for column 'HEALTH_TYP':
HEALTH_TYP
3.0    307826
2.0    296727
1.0    156186
Name: count, dtype: int64


Value counts for column 'RETOURTYP_BK_S':
RETOURTYP_BK_S
5.0    281782
3.0    174073
4.0    123269
1.0    122711
2.0     90610
Name: count, dtype: int64


Value counts for column 'SEMIO_SOZ':
SEMIO_SOZ
2    162920
6    136199
5    121776
7    114381
3    112642
4     87071
1     62088
Name: count, dtype: int64


Value counts for column 'SEMIO_FAM':
SEMIO_FAM
2    139559
```

```
4    133717
5    131794
7    115482
6    106208
3     94802
1     75515
Name: count, dtype: int64


Value counts for column 'SEMIO_REL':
SEMIO_REL
4    200560
3    150800
7    135509
1    104720
5     75420
2     70544
6     59524
Name: count, dtype: int64


Value counts for column 'SEMIO_MAT':
SEMIO_MAT
4    157683
2    131149
3    123697
7    108464
1     97321
5     95440
6     83323
Name: count, dtype: int64


Value counts for column 'SEMIO_VERT':
SEMIO_VERT
2    204265
6    141713
5    135200
7    125210
4    114633
1     44262
3     31794
Name: count, dtype: int64


Value counts for column 'SEMIO_LUST':
SEMIO_LUST
6    158605
7    158146
2    107542
1    106046
4     94782
5     89883
```

```
3     82073
Name: count, dtype: int64


Value counts for column 'SEMIO_ERL':
SEMIO_ERL
4    190762
7    175739
6    135800
3    103866
2     77007
5     74189
1     39714
Name: count, dtype: int64


Value counts for column 'SEMIO_KULT':
SEMIO_KULT
5    172649
3    130905
1    122873
7    114381
6    101284
4     98510
2     56475
Name: count, dtype: int64


Value counts for column 'SEMIO_RAT':
SEMIO_RAT
4    251724
2    140412
3    131988
7     87003
5     84515
6     57367
1     44068
Name: count, dtype: int64


Value counts for column 'SEMIO_KRIT':
SEMIO_KRIT
5    153515
4    143764
7    135254
6    133049
3    126151
2     53895
1     51449
Name: count, dtype: int64


Value counts for column 'SEMIO_DOM':
```

```
SEMIO_DOM
5    177816
7    161472
4    124717
6     98987
2     94839
3     94513
1     44733
Name: count, dtype: int64


Value counts for column 'SEMIO_KAEM':
SEMIO_KAEM
3    177353
7    130221
5    128370
6    127048
2    109879
4     77030
1     47176
Name: count, dtype: int64


Value counts for column 'SEMIO_PFLICHT':
SEMIO_PFLICHT
4    149358
3    133952
5    122510
7    115458
6    109440
2     92205
1     74154
Name: count, dtype: int64


Value counts for column 'SEMIO_TRADV':
SEMIO_TRADV
4    170253
3    148761
2    132608
5    114381
1     91164
7     74189
6     65721
Name: count, dtype: int64


Value counts for column 'SOHO_KZ':
SOHO_KZ
0.0    790370
1.0      6707
Name: count, dtype: int64
```

```
Value counts for column 'VERS_TYP':
VERS_TYP
2.0    394116
1.0    366623
Name: count, dtype: int64


Value counts for column 'ANZ_PERSONEN':
ANZ_PERSONEN
1.0     412439
2.0     190695
3.0      92744
4.0      46063
0.0      32898
5.0      15163
6.0       4731
7.0       1496
8.0        513
9.0        176
10.0        65
11.0        38
12.0        16
13.0        11
21.0         4
14.0         4
20.0         3
15.0         3
23.0         2
22.0         2
38.0         2
37.0         2
31.0         1
45.0         1
18.0         1
35.0         1
17.0         1
40.0         1
16.0         1
Name: count, dtype: int64


Value counts for column 'ANZ_TITEL':
ANZ_TITEL
0.0    793981
1.0      2892
2.0       197
3.0         5
4.0         2
Name: count, dtype: int64


Value counts for column 'HH_EINKOMMEN_SCORE':
```

```
HH_EINKOMMEN_SCORE
6.0    252773
5.0    201427
4.0    139440
3.0     84021
2.0     66234
1.0     53182
Name: count, dtype: int64


Value counts for column 'W_KEIT_KIND_HH':
W_KEIT_KIND_HH
6.0    281801
4.0    128662
3.0     99424
2.0     81995
1.0     81770
5.0     64710
Name: count, dtype: int64


Value counts for column 'WOHNDAUER_2008':
WOHNDAUER_2008
9.0    537881
8.0     78038
4.0     49238
3.0     37669
6.0     34197
5.0     30102
7.0     23305
2.0      5998
1.0       649
Name: count, dtype: int64


Value counts for column 'ANZ_HAUSHALTE_AKTIV':
ANZ_HAUSHALTE_AKTIV
1.0      195582
2.0      120874
3.0       62522
4.0       43184
5.0       37773
          ...
213.0         2
366.0         1
220.0         1
536.0         1
232.0         1
Name: count, Length: 291, dtype: int64


Value counts for column 'ANZ_HH_TITEL':
ANZ_HH_TITEL
```

```
0.0     769440
1.0      20141
2.0       2457
3.0        585
4.0        232
5.0        117
6.0        106
8.0         68
7.0         65
9.0         34
13.0        29
12.0        22
11.0        22
14.0        16
10.0        16
17.0        13
20.0         9
15.0         7
18.0         6
16.0         3
23.0         3
Name: count, dtype: int64


Value counts for column 'KONSUMNAEHE':
KONSUMNAEHE
1.0    188293
3.0    166593
5.0    150711
2.0    131190
4.0    130138
6.0     25979
7.0      4111
Name: count, dtype: int64


Value counts for column 'MIN_GEBAEUDEJAHR':
MIN_GEBAEUDEJAHR
1992.0    568755
1994.0     78791
1993.0     25485
1995.0     25460
1996.0     16605
1997.0     14443
2000.0      7365
2001.0      5853
1991.0      5811
2005.0      5508
1990.0      4408
1999.0      4405
2002.0      4198
1998.0      4090
2003.0      3343
```

```
2004.0      2920
2008.0      2162
2007.0      2118
1989.0      2046
2009.0      1970
2006.0      1947
2011.0      1813
2012.0      1735
2010.0      1345
2013.0      1139
1988.0      1027
2014.0       909
2015.0       608
1987.0       470
1986.0       125
1985.0       116
2016.0       107
Name: count, dtype: int64


Value counts for column 'OST_WEST_KZ':
OST_WEST_KZ
1    628695
0    168382
Name: count, dtype: int64


Value counts for column 'KBA05_ANTG1':
KBA05_ANTG1
0.0    261048
1.0    161217
2.0    126720
3.0    117761
4.0     91137
Name: count, dtype: int64


Value counts for column 'KBA05_ANTG2':
KBA05_ANTG2
0.0    292535
1.0    163743
2.0    138271
3.0    134454
4.0     28880
Name: count, dtype: int64


Value counts for column 'KBA05_ANTG3':
KBA05_ANTG3
0.0    511532
1.0     92748
2.0     80233
3.0     73370
```

```
Name: count, dtype: int64


Value counts for column 'KBA05_ANTG4':
KBA05_ANTG4
0.0    600158
1.0     83590
2.0     74135
Name: count, dtype: int64


Value counts for column 'KBA05_GBZ':
KBA05_GBZ
3.0    197833
5.0    158960
4.0    155299
2.0    138528
1.0    107263
Name: count, dtype: int64


Value counts for column 'BALLRAUM':
BALLRAUM
6.0    254758
1.0    151624
2.0    104389
7.0     98921
3.0     73177
4.0     61287
5.0     52334
Name: count, dtype: int64


Value counts for column 'EWDICHTE':
EWDICHTE
6.0    200844
5.0    160997
2.0    138908
4.0    130549
1.0     83906
3.0     81286
Name: count, dtype: int64


Value counts for column 'INNENSTADT':
INNENSTADT
5.0    147428
4.0    133884
6.0    111532
2.0    108942
3.0     92710
8.0     82747
7.0     67376
```

```
1.0     51871
Name: count, dtype: int64


Value counts for column 'GEBAEUDETYP_RASTER':
GEBAEUDETYP_RASTER
4.0    359231
3.0    205094
5.0    158938
2.0     58900
1.0     14909
Name: count, dtype: int64


Value counts for column 'KKK':
KKK
3.0    272949
2.0    181445
4.0    178599
1.0     99929
Name: count, dtype: int64


Value counts for column 'MOBI_REGIO':
MOBI_REGIO
1.0    163992
3.0    150335
5.0    148707
4.0    148204
2.0    146305
6.0       340
Name: count, dtype: int64


Value counts for column 'ONLINE_AFFINITAET':
ONLINE_AFFINITAET
4.0    154812
3.0    153466
1.0    147979
2.0    143222
5.0    130320
0.0     62646
Name: count, dtype: int64


Value counts for column 'REGIOTYP':
REGIOTYP
6.0    195241
5.0    145312
3.0     93899
2.0     91621
7.0     83918
4.0     68154
```

```
1.0     54777
Name: count, dtype: int64


Value counts for column 'KBA13_ANZAHL_PKW':
KBA13_ANZAHL_PKW
1400.0    11712
1500.0     8282
1300.0     6421
1600.0     6133
1700.0     3793
          ...
8.0           6
3.0           6
2.0           6
6.0           5
7.0           5
Name: count, Length: 1261, dtype: int64


Value counts for column 'PLZ8_ANTG1':
PLZ8_ANTG1
2.0    270586
3.0    222351
1.0    189246
4.0     87042
0.0      5470
Name: count, dtype: int64


Value counts for column 'PLZ8_ANTG2':
PLZ8_ANTG2
3.0    307280
2.0    215762
4.0    191004
1.0     53211
0.0      7438
Name: count, dtype: int64


Value counts for column 'PLZ8_ANTG3':
PLZ8_ANTG3
2.0    252990
1.0    237875
3.0    164040
0.0    119790
Name: count, dtype: int64


Value counts for column 'PLZ8_ANTG4':
PLZ8_ANTG4
0.0    356382
1.0    294982
```

```
2.0    123331
Name: count, dtype: int64


Value counts for column 'PLZ8_HHZ':
PLZ8_HHZ
3.0    309141
4.0    211906
5.0    175812
2.0     66891
1.0     10945
Name: count, dtype: int64


Value counts for column 'PLZ8_GBZ':
PLZ8_GBZ
3.0    288381
4.0    180247
5.0    153880
2.0    111587
1.0     40600
Name: count, dtype: int64


Value counts for column 'ARBEIT':
ARBEIT
4.0    311053
3.0    254624
2.0    135448
1.0     56667
5.0     35067
Name: count, dtype: int64


Value counts for column 'ORTSGR_KLS9':
ORTSGR_KLS9
5.0    147903
4.0    114753
7.0    102771
9.0     91762
3.0     83421
6.0     75909
8.0     72653
2.0     63268
1.0     40519
Name: count, dtype: int64


Value counts for column 'RELAT_AB':
RELAT_AB
3.0    273688
5.0    174789
1.0    142682
```

```
2.0    104689
4.0     97011
Name: count, dtype: int64


Value counts for column 'CJT_GESAMTTYP_1.0':
CJT_GESAMTTYP_1.0
False    703885
True      93192
Name: count, dtype: int64


Value counts for column 'CJT_GESAMTTYP_2.0':
CJT_GESAMTTYP_2.0
False    655911
True     141166
Name: count, dtype: int64


Value counts for column 'CJT_GESAMTTYP_3.0':
CJT_GESAMTTYP_3.0
False    650009
True     147068
Name: count, dtype: int64


Value counts for column 'CJT_GESAMTTYP_4.0':
CJT_GESAMTTYP_4.0
False    598988
True     198089
Name: count, dtype: int64


Value counts for column 'CJT_GESAMTTYP_5.0':
CJT_GESAMTTYP_5.0
False    686045
True     111032
Name: count, dtype: int64


Value counts for column 'CJT_GESAMTTYP_6.0':
CJT_GESAMTTYP_6.0
False    695179
True     101898
Name: count, dtype: int64


Value counts for column 'FINANZTYP_1':
FINANZTYP_1
False    600272
True     196805
Name: count, dtype: int64
```

```
Value counts for column 'FINANZTYP_2':
FINANZTYP_2
False    692500
True     104577
Name: count, dtype: int64


Value counts for column 'FINANZTYP_3':
FINANZTYP_3
False    752480
True      44597
Name: count, dtype: int64


Value counts for column 'FINANZTYP_4':
FINANZTYP_4
False    741203
True      55874
Name: count, dtype: int64


Value counts for column 'FINANZTYP_5':
FINANZTYP_5
False    690857
True     106220
Name: count, dtype: int64


Value counts for column 'FINANZTYP_6':
FINANZTYP_6
False    508073
True     289004
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_1.0':
GFK_URLAUBERTYP_1.0
False    746437
True      50640
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_2.0':
GFK_URLAUBERTYP_2.0
False    753430
True      43647
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_3.0':
GFK_URLAUBERTYP_3.0
False    743983
```

```
True      53094
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_4.0':
GFK_URLAUBERTYP_4.0
False    736664
True      60413
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_5.0':
GFK_URLAUBERTYP_5.0
False    726609
True      70468
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_6.0':
GFK_URLAUBERTYP_6.0
False    771356
True      25721
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_7.0':
GFK_URLAUBERTYP_7.0
False    756435
True      40642
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_8.0':
GFK_URLAUBERTYP_8.0
False    714085
True      82992
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_9.0':
GFK_URLAUBERTYP_9.0
False    740031
True      57046
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_10.0':
GFK_URLAUBERTYP_10.0
False    694329
True     102748
Name: count, dtype: int64
```

```
Value counts for column 'GFK_URLAUBERTYP_11.0':
GFK_URLAUBERTYP_11.0
False    722026
True      75051
Name: count, dtype: int64


Value counts for column 'GFK_URLAUBERTYP_12.0':
GFK_URLAUBERTYP_12.0
False    667094
True     129983
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_1.0':
LP_FAMILIE_FEIN_1.0
True     402248
False    394829
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_2.0':
LP_FAMILIE_FEIN_2.0
False    698586
True      98491
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_3.0':
LP_FAMILIE_FEIN_3.0
False    792395
True       4682
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_4.0':
LP_FAMILIE_FEIN_4.0
False    785504
True      11573
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_5.0':
LP_FAMILIE_FEIN_5.0
False    785913
True      11164
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_6.0':
LP_FAMILIE_FEIN_6.0
False    788565
True       8512
```

```
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_7.0':
LP_FAMILIE_FEIN_7.0
False    777509
True      19568
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_8.0':
LP_FAMILIE_FEIN_8.0
False    775300
True      21777
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_9.0':
LP_FAMILIE_FEIN_9.0
False    786626
True      10451
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_10.0':
LP_FAMILIE_FEIN_10.0
False    668175
True     128902
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_FEIN_11.0':
LP_FAMILIE_FEIN_11.0
False    748350
True      48727
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_GROB_1.0':
LP_FAMILIE_GROB_1.0
True     402248
False    394829
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_GROB_2.0':
LP_FAMILIE_GROB_2.0
False    698586
True      98491
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_GROB_3.0':
```

```
LP_FAMILIE_GROB_3.0
False    769658
True      27419
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_GROB_4.0':
LP_FAMILIE_GROB_4.0
False    747220
True      49857
Name: count, dtype: int64


Value counts for column 'LP_FAMILIE_GROB_5.0':
LP_FAMILIE_GROB_5.0
False    608997
True     188080
Name: count, dtype: int64


Value counts for column 'LP_STATUS_FEIN_1.0':
LP_STATUS_FEIN_1.0
False    590311
True     206766
Name: count, dtype: int64


Value counts for column 'LP_STATUS_FEIN_2.0':
LP_STATUS_FEIN_2.0
False    686061
True     111016
Name: count, dtype: int64


Value counts for column 'LP_STATUS_FEIN_3.0':
LP_STATUS_FEIN_3.0
False    728184
True      68893
Name: count, dtype: int64


Value counts for column 'LP_STATUS_FEIN_4.0':
LP_STATUS_FEIN_4.0
False    723139
True      73938
Name: count, dtype: int64


Value counts for column 'LP_STATUS_FEIN_5.0':
LP_STATUS_FEIN_5.0
False    769605
True      27472
Name: count, dtype: int64
```

```
Value counts for column 'LP_STATUS_FEIN_6.0':
LP_STATUS_FEIN_6.0
False    768207
True      28870
Name: count, dtype: int64


Value counts for column 'LP_STATUS_FEIN_7.0':
LP_STATUS_FEIN_7.0
False    787879
True       9198
Name: count, dtype: int64


Value counts for column 'LP_STATUS_FEIN_8.0':
LP_STATUS_FEIN_8.0
False    778552
True      18525
Name: count, dtype: int64


Value counts for column 'LP_STATUS_FEIN_9.0':
LP_STATUS_FEIN_9.0
False    660848
True     136229
Name: count, dtype: int64


Value counts for column 'LP_STATUS_FEIN_10.0':
LP_STATUS_FEIN_10.0
False    685539
True     111538
Name: count, dtype: int64


Value counts for column 'LP_STATUS_GROB_1.0':
LP_STATUS_GROB_1.0
False    479295
True     317782
Name: count, dtype: int64


Value counts for column 'LP_STATUS_GROB_2.0':
LP_STATUS_GROB_2.0
False    626774
True     170303
Name: count, dtype: int64


Value counts for column 'LP_STATUS_GROB_3.0':
LP_STATUS_GROB_3.0
```

```
False    759009
True      38068
Name: count, dtype: int64


Value counts for column 'LP_STATUS_GROB_4.0':
LP_STATUS_GROB_4.0
False    642323
True     154754
Name: count, dtype: int64


Value counts for column 'LP_STATUS_GROB_5.0':
LP_STATUS_GROB_5.0
False    685539
True     111538
Name: count, dtype: int64


Value counts for column 'NATIONALITAET_KZ_1.0':
NATIONALITAET_KZ_1.0
True     667356
False    129721
Name: count, dtype: int64


Value counts for column 'NATIONALITAET_KZ_2.0':
NATIONALITAET_KZ_2.0
False    733458
True      63619
Name: count, dtype: int64


Value counts for column 'NATIONALITAET_KZ_3.0':
NATIONALITAET_KZ_3.0
False    764540
True      32537
Name: count, dtype: int64


Value counts for column 'SHOPPER_TYP_0.0':
SHOPPER_TYP_0.0
False    669968
True     127109
Name: count, dtype: int64


Value counts for column 'SHOPPER_TYP_1.0':
SHOPPER_TYP_1.0
False    549925
True     247152
Name: count, dtype: int64
```

```
Value counts for column 'SHOPPER_TYP_2.0':
SHOPPER_TYP_2.0
False    591203
True     205874
Name: count, dtype: int64


Value counts for column 'SHOPPER_TYP_3.0':
SHOPPER_TYP_3.0
False    616473
True     180604
Name: count, dtype: int64


Value counts for column 'ZABEOTYP_1':
ZABEOTYP_1
False    673807
True     123270
Name: count, dtype: int64


Value counts for column 'ZABEOTYP_2':
ZABEOTYP_2
False    764134
True      32943
Name: count, dtype: int64


Value counts for column 'ZABEOTYP_3':
ZABEOTYP_3
False    515305
True     281772
Name: count, dtype: int64


Value counts for column 'ZABEOTYP_4':
ZABEOTYP_4
False    589694
True     207383
Name: count, dtype: int64


Value counts for column 'ZABEOTYP_5':
ZABEOTYP_5
False    716185
True      80892
Name: count, dtype: int64


Value counts for column 'ZABEOTYP_6':
ZABEOTYP_6
False    726260
```

```
True      70817
Name: count, dtype: int64


Value counts for column 'GEBAEUDETYP_1.0':
GEBAEUDETYP_1.0
True     459844
False    337233
Name: count, dtype: int64


Value counts for column 'GEBAEUDETYP_2.0':
GEBAEUDETYP_2.0
False    792288
True       4789
Name: count, dtype: int64


Value counts for column 'GEBAEUDETYP_3.0':
GEBAEUDETYP_3.0
False    618570
True     178507
Name: count, dtype: int64


Value counts for column 'GEBAEUDETYP_4.0':
GEBAEUDETYP_4.0
False    796192
True        885
Name: count, dtype: int64


Value counts for column 'GEBAEUDETYP_5.0':
GEBAEUDETYP_5.0
False    797076
True          1
Name: count, dtype: int64


Value counts for column 'GEBAEUDETYP_6.0':
GEBAEUDETYP_6.0
False    796465
True        612
Name: count, dtype: int64


Value counts for column 'GEBAEUDETYP_8.0':
GEBAEUDETYP_8.0
False    644638
True     152439
Name: count, dtype: int64
```

```
Value counts for column 'CAMEO_DEUG_2015_1':
CAMEO_DEUG_2015_1
False    760897
True      36180
Name: count, dtype: int64


Value counts for column 'CAMEO_DEUG_2015_2':
CAMEO_DEUG_2015_2
False    713928
True      83149
Name: count, dtype: int64


Value counts for column 'CAMEO_DEUG_2015_3':
CAMEO_DEUG_2015_3
False    710465
True      86612
Name: count, dtype: int64


Value counts for column 'CAMEO_DEUG_2015_4':
CAMEO_DEUG_2015_4
False    693263
True     103814
Name: count, dtype: int64


Value counts for column 'CAMEO_DEUG_2015_5':
CAMEO_DEUG_2015_5
False    741861
True      55216
Name: count, dtype: int64


Value counts for column 'CAMEO_DEUG_2015_6':
CAMEO_DEUG_2015_6
False    691258
True     105819
Name: count, dtype: int64


Value counts for column 'CAMEO_DEUG_2015_7':
CAMEO_DEUG_2015_7
False    719189
True      77888
Name: count, dtype: int64


Value counts for column 'CAMEO_DEUG_2015_8':
CAMEO_DEUG_2015_8
False    662683
True     134394
```

```
Name: count, dtype: int64


Value counts for column 'CAMEO_DEUG_2015_9':
CAMEO_DEUG_2015_9
False    688939
True     108138
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_1A':
CAMEO_DEU_2015_1A
False    786240
True      10837
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_1B':
CAMEO_DEU_2015_1B
False    793009
True       4068
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_1C':
CAMEO_DEU_2015_1C
False    792767
True       4310
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_1D':
CAMEO_DEU_2015_1D
False    785169
True      11908
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_1E':
CAMEO_DEU_2015_1E
False    792020
True       5057
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_2A':
CAMEO_DEU_2015_2A
False    783851
True      13226
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_2B':
```

```
CAMEO_DEU_2015_2B
False    781609
True      15468
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_2C':
CAMEO_DEU_2015_2C
False    777669
True      19408
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_2D':
CAMEO_DEU_2015_2D
False    762030
True      35047
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_3A':
CAMEO_DEU_2015_3A
False    786623
True      10454
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_3B':
CAMEO_DEU_2015_3B
False    789934
True       7143
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_3C':
CAMEO_DEU_2015_3C
False    762337
True      34740
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_3D':
CAMEO_DEU_2015_3D
False    762802
True      34275
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_4A':
CAMEO_DEU_2015_4A
False    763949
True      33128
Name: count, dtype: int64
```

```
Value counts for column 'CAMEO_DEU_2015_4B':
CAMEO_DEU_2015_4B
False    788039
True       9038
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_4C':
CAMEO_DEU_2015_4C
False    749312
True      47765
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_4D':
CAMEO_DEU_2015_4D
False    788512
True       8565
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_4E':
CAMEO_DEU_2015_4E
False    791759
True       5318
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_5A':
CAMEO_DEU_2015_5A
False    784924
True      12153
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_5B':
CAMEO_DEU_2015_5B
False    786732
True      10345
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_5C':
CAMEO_DEU_2015_5C
False    787151
True       9926
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_5D':
CAMEO_DEU_2015_5D
```

```
False    782143
True      14934
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_5E':
CAMEO_DEU_2015_5E
False    793500
True       3577
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_5F':
CAMEO_DEU_2015_5F
False    792796
True       4281
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_6A':
CAMEO_DEU_2015_6A
False    790278
True       6799
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_6B':
CAMEO_DEU_2015_6B
False    740435
True      56642
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_6C':
CAMEO_DEU_2015_6C
False    782262
True      14815
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_6D':
CAMEO_DEU_2015_6D
False    791009
True       6068
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_6E':
CAMEO_DEU_2015_6E
False    780973
True      16104
Name: count, dtype: int64
```

```
Value counts for column 'CAMEO_DEU_2015_6F':
CAMEO_DEU_2015_6F
False    791686
True       5391
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_7A':
CAMEO_DEU_2015_7A
False    762693
True      34384
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_7B':
CAMEO_DEU_2015_7B
False    772588
True      24489
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_7C':
CAMEO_DEU_2015_7C
False    788018
True       9059
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_7D':
CAMEO_DEU_2015_7D
False    791748
True       5329
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_7E':
CAMEO_DEU_2015_7E
False    792450
True       4627
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_8A':
CAMEO_DEU_2015_8A
False    744650
True      52427
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_8B':
CAMEO_DEU_2015_8B
False    763653
```

```
True      33424
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_8C':
CAMEO_DEU_2015_8C
False    766099
True      30978
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_8D':
CAMEO_DEU_2015_8D
False    779512
True      17565
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_9A':
CAMEO_DEU_2015_9A
False    776540
True      20537
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_9B':
CAMEO_DEU_2015_9B
False    769416
True      27661
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_9C':
CAMEO_DEU_2015_9C
False    772091
True      24986
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_9D':
CAMEO_DEU_2015_9D
False    768486
True      28591
Name: count, dtype: int64


Value counts for column 'CAMEO_DEU_2015_9E':
CAMEO_DEU_2015_9E
False    790714
True       6363
Name: count, dtype: int64
```

```
Value counts for column 'MOVEMENT':
MOVEMENT
0.0    593618
1.0    175135
Name: count, dtype: int64


Value counts for column 'DECADE':
DECADE
6.0    225333
4.0    175064
5.0    151670
3.0    114307
2.0     74261
1.0     28118
Name: count, dtype: int64


Value counts for column 'WEALTH':
WEALTH
5.0    223496
2.0    190426
4.0    189876
1.0    119329
3.0     68083
Name: count, dtype: int64


Value counts for column 'LIFE_STAGE':
LIFE_STAGE
1.0    244914
4.0    232595
3.0    119508
5.0    116957
2.0     77236
Name: count, dtype: int64
```

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?)

Discussion:

>

> For the column "PRAEGENDE_JUGENDJAHRE," I created two dictionaries based on the data dictionary file, one for movement and another for decades. Using this information, I added two new columns named "movement" and "decade." I substituted the original data with values from the respective dictionaries. The 'PRAEGENDE_JUGENDJAHRE' column was removed from the dataset.
>
> For the columns 'CAMEO_INTL_2015' I used a similar approach, creating two dictionaries based on the information held in the data dictionary, one to create a column called 'WEALTH' and another called 'LIFE_STAGE'.
>
> I then checked in the 'feat_info' dataframe for other columns that may be mixed, and then removed them from the 'azdias_encoded' dataframe.

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 [49]:
```
# If there are other re-engineering tasks you need to perform, make sure you
# take care of them here. (Dealing with missing data will come in step 2.1.)
```

In [50]:
```
# Do whatever you need to in order to ensure that the dataframe only contain
# the columns that should be passed to the algorithm functions.
```

I think the steps in the two cells above are complete.

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 [51]:
```
def clean_data(df):
    """
    Perform feature trimming, re-encoding, and engineering for demographics

    INPUT: Demographics DataFrame
    OUTPUT: Trimmed and cleaned demographics DataFrame
    """
    # Load in the feature summary file.
    feat_info = pd.read_csv('AZDIAS_Feature_Summary.csv', sep=';')

    # Define missing and unknown values
    missing_unknown_values = ['0', '-1', 'XX', 'X']

    # Iterate through each column in 'feat_info'
    for index, row in feat_info.iterrows():
        attribute = row['attribute']
        vals = row['missing_or_unknown']

        # Convert the string representation of the list to a list of integer
        vals = [int(val) if val.lstrip('-').isdigit() else val for val in va
        vals.extend(missing_unknown_values)

        # Replace the missing or unknown values with NaN in the DataFrame
        df[attribute].replace(vals, np.nan, inplace=True)

    # Make a list of columns that were originally dropped from 'azdias' to k
    columns_to_drop = ['TITEL_KZ', 'AGER_TYP', 'KK_KUNDENTYP', 'KBA05_BAUMAX

    # Remove the columns
    df_clean = df.drop(labels = columns_to_drop, axis=1)

    # Calculate the proportion of missing values for each row
    missing_data_proportion = df_clean.isnull().mean(axis=1)

    # Filter rows with missing values less than 25%
    df_clean = df_clean[missing_data_proportion < 0.25]
```

```
# Replace the 'W' and 'O' values with '0' and '1'
df_clean['OST_WEST_KZ'].replace(['W', 'O'], [1, 0], inplace=True)

# Create a list of columns to encode (same as in 'azdias' dataframe')
multi_level_features = ['CJT_GESAMTTYP', 'FINANZTYP', 'GFK_URLAUBERTYP',
                        'LP_STATUS_FEIN', 'LP_STATUS_GROB', 'NATIONALIT
                        'ZABEOTYP', 'GEBAEUDETYP', 'CAMEO_DEUG_2015', '

# Encode the multilevel features (same as in 'azdias')
df_clean = pd.get_dummies(df_clean, columns = multi_level_features)



# Reengineer features from 'PRAEGENDE_JUGENDJAHRE'
# Assign '0' or '1' for 'Mainstream' or 'Avantgarde' in a dictionary
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 : 0, 15 : 1}

# Assign a '1', '2', '3', '4', '5', or '6' depending on the decade in a
decades = {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, 15 : 6}

# Create the 'MOVEMENT' feature
df_clean['MOVEMENT'] = df_clean['PRAEGENDE_JUGENDJAHRE']
df_clean['MOVEMENT'].replace(movement, inplace=True)

# Create the 'DECADE' feature
df_clean['DECADE'] = df_clean['PRAEGENDE_JUGENDJAHRE']
df_clean['DECADE'].replace(decades, inplace=True)


# Engineer two new variables from "CAMEO_INTL_2015"

# Create 'WEALTH' values in a dictionary
wealth = {'11':1, '12':1, '13':1, '14':1, '15':1, '21':2, '22':2, '23':2
          '31':3, '32':3, '33':3, '34':3, '35':3, '41':4, '42':4, '43':
          '51':5, '52':5, '53':5, '54':5, '55':5}

# Create 'LIFE_STAGE' values in a dictionary
life_stage = {'11':1, '12':2, '13':3, '14':4, '15':5, '21':1, '22':2, '2
              '31':1, '32':2, '33':3, '34':4, '35':5, '41':1, '42':2, '
              '51':1, '52':2, '53':3, '54':4, '55':5}

# Create the 'WEALTH' feature
df_clean['WEALTH'] = df_clean['CAMEO_INTL_2015']
df_clean['WEALTH'].replace(wealth, inplace = True)

# Create the 'LIFE_STAGE' feature
df_clean['LIFE_STAGE'] = df_clean['CAMEO_INTL_2015']
df_clean['LIFE_STAGE'].replace(life_stage, inplace = True)

# Drop the original 'PRAEGENDE_JUGENDJAHRE' and 'CAMEO_INTL_2015' column
```

```
    df_clean.drop(['PRAEGENDE_JUGENDJAHRE', 'CAMEO_INTL_2015'], axis=1, inpl

    # Drop remaining columns with mixed values
    df_clean.drop(['LP_LEBENSPHASE_FEIN', 'LP_LEBENSPHASE_GROB', 'WOHNLAGE',


    return df_clean
```

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 to help you navigate to all of the classes and functions that you'll need. In this substep, you'll need to check the following:

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

In [52]:

```
# If you've not yet cleaned the dataset of all NaN values, then investigate
# do that now.

# Import SimpleImputer
from sklearn.impute import SimpleImputer

# Impute the missing values to get rid of NaNs
imputer = SimpleImputer(strategy = 'most_frequent')
azdias_imputed = imputer.fit_transform(azdias_encoded)
```

In [53]:
```
# Check for missing values in the NumPy array
missing_values_count = np.isnan(azdias_imputed).sum()

# Sum the total number of missing values
total_missing_values = missing_values_count.sum()

print("Total missing values:", total_missing_values)
```

```
Total missing values: 0
```

In [54]:
```
# Apply feature scaling to the general population demographics data.

# Import the StandardScaler from scikit-learn
from sklearn.preprocessing import StandardScaler

# Create a StandardScaler instance
scaler = StandardScaler()

# Standardize the data in 'azdias_imputed' using the scaler
azdias_scaled = scaler.fit_transform(azdias_imputed)

# Create a new DataFrame to store the standardized data
azdias_scaled = pd.DataFrame(azdias_scaled, columns=azdias_encoded.columns)
```

In [55]:
```
# Look at the first five rows of 'azdias_scaled'
azdias_scaled.head()
```

Out[55]:

	ALTERSKATEGORIE_GROB	ANREDE_KZ	FINANZ_MINIMALIST	FINANZ_SPARER	FINANZ_VO
0	-1.766173	0.958121	-1.494463	1.538139	
1	0.200733	0.958121	-1.494463	0.864753	
2	1.184186	0.958121	0.683285	-0.482020	
3	0.200733	-1.043709	0.683285	0.191366	
4	-1.766173	0.958121	-0.042631	-1.155407	

5 rows × 192 columns

Discussion 2.1: Apply Feature Scaling

Before performing feature scaling, I used an Imputer (as suggested above) to address missing values (NaNs) in the dataset. Specifically, I utilized the SimpleImputer, opting for the "most_frequent" strategy to replace the null values. The reason for choosing "most_frequent" was to avoid using the mean strategy, especially since many features contained a wide range of values.

After successfully imputing all NaNs, I then to applied the StandardScaler function to fit and transform the encoded dataset. The choice of using StandardScaler was based on its appropriateness for this task. Standardization is the preferred method in this case because we're not attempting to normalize units with dissimilar scales but rather to conform the data to a standard statistical shape for improved handling.

Finally, I created a dataframe using the scaled data and retained the appropriate column names for clarity and further analysis.

Step 2.2: Perform Dimensionality Reduction

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

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

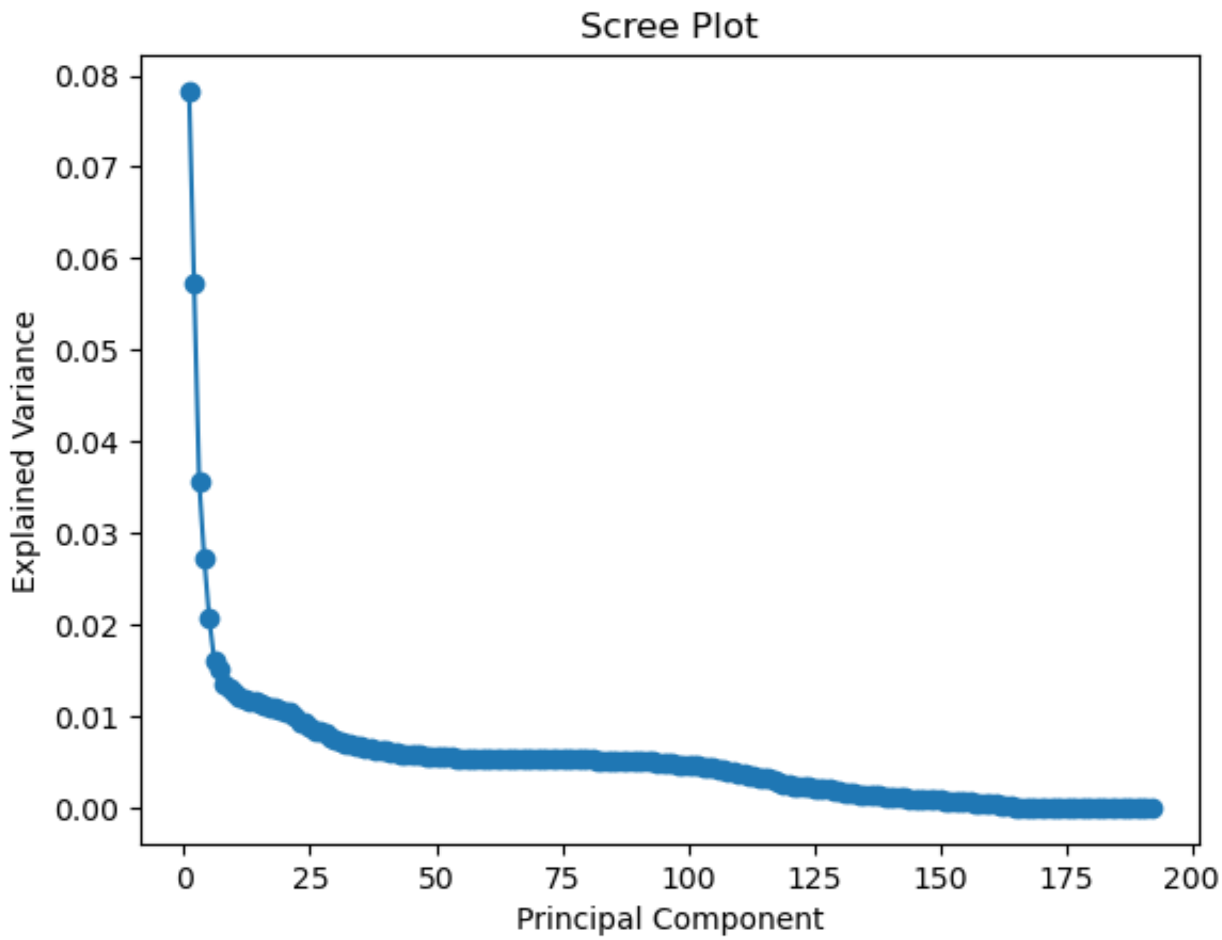
In [56]:
```
# Apply PCA to the data.

# Import the PCA (Principal Component Analysis) class from the scikit-learn
from sklearn.decomposition import PCA

# Create an instance of the PCA class
pca = PCA()

# Fit the PCA model to the 'azdias_scaled' dataframe
pca.fit(azdias_scaled)
```

Out[56]:
```
▾ PCA
PCA()
```

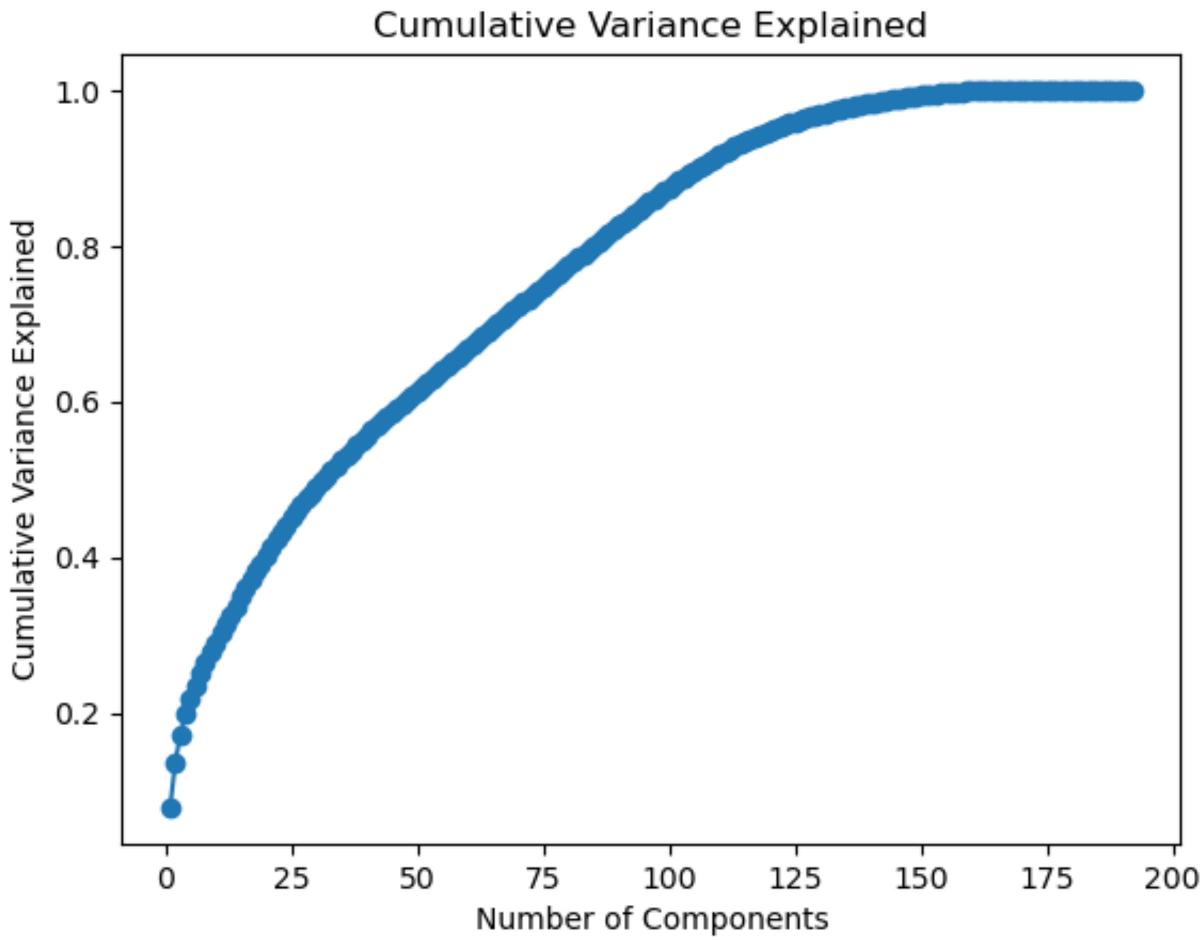
In [57]:
```
# Calculate the explained variance for each principal component

explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[7.82739515e-02 5.74024267e-02 3.56059706e-02 2.72620374e-02
 2.06530709e-02 1.60724064e-02 1.51880996e-02 1.35581863e-02
 1.30151872e-02 1.26628736e-02 1.21723415e-02 1.19306250e-02
 1.16815097e-02 1.15876952e-02 1.13443885e-02 1.11334987e-02
 1.09518804e-02 1.08206781e-02 1.06097980e-02 1.04122920e-02
 1.03518613e-02 9.89454356e-03 9.28529861e-03 9.20712969e-03
 8.79871092e-03 8.46174910e-03 8.37546113e-03 8.09891109e-03
 7.68381559e-03 7.32843789e-03 7.20712133e-03 7.03581718e-03
 6.88809920e-03 6.71945506e-03 6.67199691e-03 6.61316797e-03
 6.56829900e-03 6.32874141e-03 6.25380418e-03 6.18256925e-03
 6.14013221e-03 5.95306913e-03 5.88052138e-03 5.81469077e-03
 5.78304630e-03 5.72365797e-03 5.69715970e-03 5.63579562e-03
 5.59893689e-03 5.57477395e-03 5.55235642e-03 5.51614020e-03
 5.49313290e-03 5.45582177e-03 5.43732419e-03 5.41252133e-03
 5.39848051e-03 5.38344259e-03 5.38084045e-03 5.35776082e-03
 5.35228069e-03 5.33629131e-03 5.33555576e-03 5.32015548e-03
 5.30867558e-03 5.30672699e-03 5.29939781e-03 5.28966286e-03
 5.28209062e-03 5.27796551e-03 5.27501678e-03 5.26550085e-03
 5.26101216e-03 5.25787799e-03 5.25623007e-03 5.24882735e-03
 5.24602810e-03 5.24072569e-03 5.23952141e-03 5.23188032e-03
 5.22397836e-03 5.22139726e-03 5.21432012e-03 5.20690533e-03
 5.20191754e-03 5.19404749e-03 5.16961181e-03 5.15170253e-03
 5.14061720e-03 5.12012857e-03 5.11501665e-03 5.06989532e-03
 5.02085588e-03 4.92836409e-03 4.89783452e-03 4.82665694e-03
 4.76099848e-03 4.72470228e-03 4.67124813e-03 4.60318902e-03
 4.57315746e-03 4.53701776e-03 4.47358909e-03 4.34118945e-03
 4.30026025e-03 4.11509000e-03 4.08712634e-03 4.03499662e-03
 3.86934418e-03 3.73899981e-03 3.60905239e-03 3.52846740e-03
 3.43455489e-03 3.31621849e-03 3.18000799e-03 3.15199655e-03
 3.08978105e-03 2.83332952e-03 2.65972522e-03 2.60367511e-03
 2.37760030e-03 2.30776728e-03 2.28177420e-03 2.26659672e-03
 2.13680389e-03 2.09002182e-03 2.03488712e-03 1.98614958e-03
 1.86185972e-03 1.81586579e-03 1.68748617e-03 1.60967080e-03
 1.56223417e-03 1.48900921e-03 1.43246063e-03 1.34763605e-03
 1.32470622e-03 1.27909382e-03 1.22998160e-03 1.21317578e-03
 1.11859888e-03 1.09703039e-03 1.09269417e-03 1.01665876e-03
 1.01176914e-03 9.43354786e-04 9.16089262e-04 9.09186843e-04
 8.84962751e-04 8.30642090e-04 7.93922395e-04 7.37228361e-04
 6.93043594e-04 6.42530269e-04 6.29199895e-04 5.86918883e-04
 5.14364270e-04 4.63855189e-04 4.36371763e-04 3.99995280e-04
 3.55092196e-04 2.72843118e-04 1.40845182e-04 1.05368359e-04
 1.67504494e-05 6.61779200e-29 5.74832356e-30 4.89395050e-30
 3.75063055e-30 2.75712487e-30 2.48219453e-30 1.81396348e-30
 1.78042870e-30 1.76370648e-30 1.60343758e-30 1.43705429e-30
 1.30231603e-30 1.24798092e-30 9.74055546e-31 8.40181184e-31
 6.73915691e-31 5.72155034e-31 5.45019614e-31 5.39850925e-31
 5.00682486e-31 4.85830671e-31 3.05512256e-31 2.93321945e-31
 2.44946642e-31 1.51217373e-31 9.06716989e-33 2.34280549e-36]
```

In [58]:
```
# Investigate the variance accounted for by each principal component.


# Create a scree plot to visualize the explained variance for each principal

# Plot the explained variance for each component to identify an "elbow point
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='
plt.xlabel("Principal Component")
plt.ylabel("Explained Variance")
plt.title("Scree Plot")
plt.show()
```



In [59]:
```
# Calculate the cumulative explained variance for each number of components
cumulative_variance = np.cumsum(explained_variance)

# Plot the cumulative explained variance to determine how many components ar
# to reach a specific level of variance explainedb
plt.plot(range(1, len(explained_variance) + 1), cumulative_variance, marker=
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Variance Explained")
plt.title("Cumulative Variance Explained")
plt.show()
```



In [60]:
```
variances = [0.95, 0.9, 0.85, 0.8]

# Calculate the number of components for each target variance using list com
component_counts = [np.where(cumulative_variance > v)[0][0] + 1 for v in var

# Print the results
for v, n_components in zip(variances, component_counts):
    print(f'Number of components that explain {round(v * 100)}% variance: {n
```

```
Number of components that explain 95% variance: 121
Number of components that explain 90% variance: 106
Number of components that explain 85% variance: 95
Number of components that explain 80% variance: 85
```

In [61]:
```
# Re-apply PCA to the data while selecting for number of components to retai

# Create an instance of the PCA class
pca = PCA(n_components=95)

# Fit the PCA model to the 'azdias_scaledb' dataframe
pca.fit(azdias_scaled)

# Apply a PCA transformation to the dataset.
azdias_pca_reduced = pca.transform(azdias_scaled)
```

In [62]:
```
azdias_pca_reduced.shape
```

Out[62]:
```
(797077, 95)
```

Discussion 2.2: Perform Dimensionality Reduction

I chose to retain 95 components, as they account for 85% of the data's variance. This choice was made to preserve a significant amount of information while effectively reducing the data's dimensionality. Also, this can help with computational cost and visualization and interpretibility.

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 [63]:
```
# HINT: Try defining a function here or in a new cell that you can reuse in
# other cells.

def pca_weights(pca: PCA, component_num: int, feature_names: List[str]) -> p
    """
    Map the weights of the principal components to their corresponding featu
    and sort them by weight.

    Parameters:
    - pca: Fitted PCA object.
    - component_num: The component number for which the weights are required
    - feature_names: List of feature names.

    Returns:
    - sorted_weights_df: DataFrame containing features and their correspondi
    """
    component_weights = pca.components_[component_num]

    sorted_weights_df = pd.DataFrame(
        list(zip(feature_names, component_weights)),
        columns=['Feature', 'Weight']
    ).sort_values(by='Weight', ascending=False)

    return sorted_weights_df
```

In [64]:
```
# Map weights for the first principal component to corresponding feature nam
# and then print the linked values, sorted by weight.

first_component = pca_weights(pca, 0, azdias_scaled)
first_component
```

Out [64]:

	Feature	Weight
110	LP_STATUS_GROB_1.0	0.197340
29	HH_EINKOMMEN_SCORE	0.186400
190	WEALTH	0.184917
53	PLZ8_ANTG3	0.181302
54	PLZ8_ANTG4	0.175000
...	...	...
37	KBA05_ANTG1	-0.180855
41	KBA05_GBZ	-0.181257
51	PLZ8_ANTG1	-0.182537
47	MOBI_REGIO	-0.188279
2	FINANZ_MINIMALIST	-0.195234

192 rows × 2 columns

In [65]:

```
# Map weights for the second principal component to corresponding feature na
# and then print the linked values, sorted by weight.

second_component = pca_weights(pca, 1, azdias_scaled)
second_component
```

Out [65]:

	Feature	Weight
0	ALTERSKATEGORIE_GROB	0.231214
4	FINANZ_VORSORGER	0.217120
124	ZABEOTYP_3	0.200292
17	SEMIO_ERL	0.179682
16	SEMIO_LUST	0.161965
...	...	...
24	SEMIO_TRADV	-0.206639
13	SEMIO_REL	-0.213623
6	FINANZ_UNAUFFAELLIGER	-0.214807
3	FINANZ_SPARER	-0.224524
189	DECADE	-0.229177

192 rows × 2 columns

In [66]:
```
third_component = pca_weights(pca, 2, azdias_scaled)
third_component
```

Out[66]:

	Feature	Weight
15	SEMIO_VERT	0.318781
12	SEMIO_FAM	0.260622
11	SEMIO_SOZ	0.257082
18	SEMIO_KULT	0.251531
70	FINANZTYP_5	0.135489
...	...	...
17	SEMIO_ERL	-0.208149
20	SEMIO_KRIT	-0.266944
21	SEMIO_DOM	-0.283587
22	SEMIO_KAEM	-0.314740
1	ANREDE_KZ	-0.344708

192 rows × 2 columns

Discussion 2.3: Interpret Principal Components

Component 1:

- LP_STATUS_GROB_1.0 - social status (low income earners)
- HH_EINKOMMEN_SCORE - estimated household income
- WEALTH -
- PLZ8_ANTG3 - # of 6-10 family houses in PLZ8 REGION
- PLZ8_ANTG4 - # of 10+ family houses in PLZ8 REGION

VS

- KBAO5_ANTG1- # of 1-2 family houses in the microcell
- KBAO5_GBZ - # of buildings in the microcell
- PLZ28_ANTG1 - # of 1-2 family homes in the region
- MOBI_REGIO - movement patterns
- FINANZ_MIINIMALIST - population with low financial interest

Component 1 appears to be linked to financial status or well-being. We can see a negative correlation between multiple family homes vs single family homes (which tend to be more expensive). Also we can see correlations between low financial interest and movement patterns, which could suggest less financial stability.

Component 2:

- ALTERSKATEGORIE_GROB - estimated age
- FINANZ_VORSORGER - be prepared attitude (financially)
- ZABEOTYP_3- energy consumption
- SEMIO_ERL- event oriented (socially)
- SEMIO_LUST- sensual-minded (socially)

VS

- SEMIO_TRADV - traditionally-minded (socially)
- SEMIO_REL - religious
- FINANZ_UNAUFFAELLIGER - financially inconspicuous
- FINANZ_SPARER - money-saver
- DECADE - generation

The second component appears to be linked to social attitudes and financial traits. We can see strong positive correlation between people who are event-oriented and sensually minded. These traits show a negative correlation to those who are traditionally-minded, religious, and financially inconspicuous (these people are perhaps more reserved).

Component 3:

- SEMIO_VERT - dreamful
- SEMIO_FAM - family-minded
- SEMIO_SOZ - socially-minded
- SEMIO_KULT - cultural-minded
- FINANZTYP_5 - investor

VS

- SEMIO_ERL - event-oriented
- SEMIO_KRIT - critical-minded
- SEMIO_DOM - dominant-minded
- SEMIO_KAEM - combative attitude
- ANREDE_KZ - gender

Component 3 is linked to personality traits. This component shows a strong correlation between those who tend to be dreamful, family-oriented, socially-oriented, and culturally minded. We see a stong negative correlation to those who tend to be critical-minded, dominant, and exhibit a combative attitude.

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 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 [67]:
```
# Over a number of different cluster counts...

from sklearn.cluster import KMeans

# Define a range of cluster counts to test
cluster_counts_to_test = range(2, 31)

# Initialize lists to store the average distances for each cluster count
average_distances = []

# Loop through different cluster counts
for n_clusters in cluster_counts_to_test:
    # Create a KMeans instance with the current number of clusters
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)

    # Fit the KMeans model to the scaled data
    model = kmeans.fit(azdias_pca_reduced)

    # Calculate the average negative score (average within-cluster distance)
    avg_distance = -kmeans.score(azdias_pca_reduced)

    # Append the average distance to the list
    average_distances.append(avg_distance)
```

```
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
```

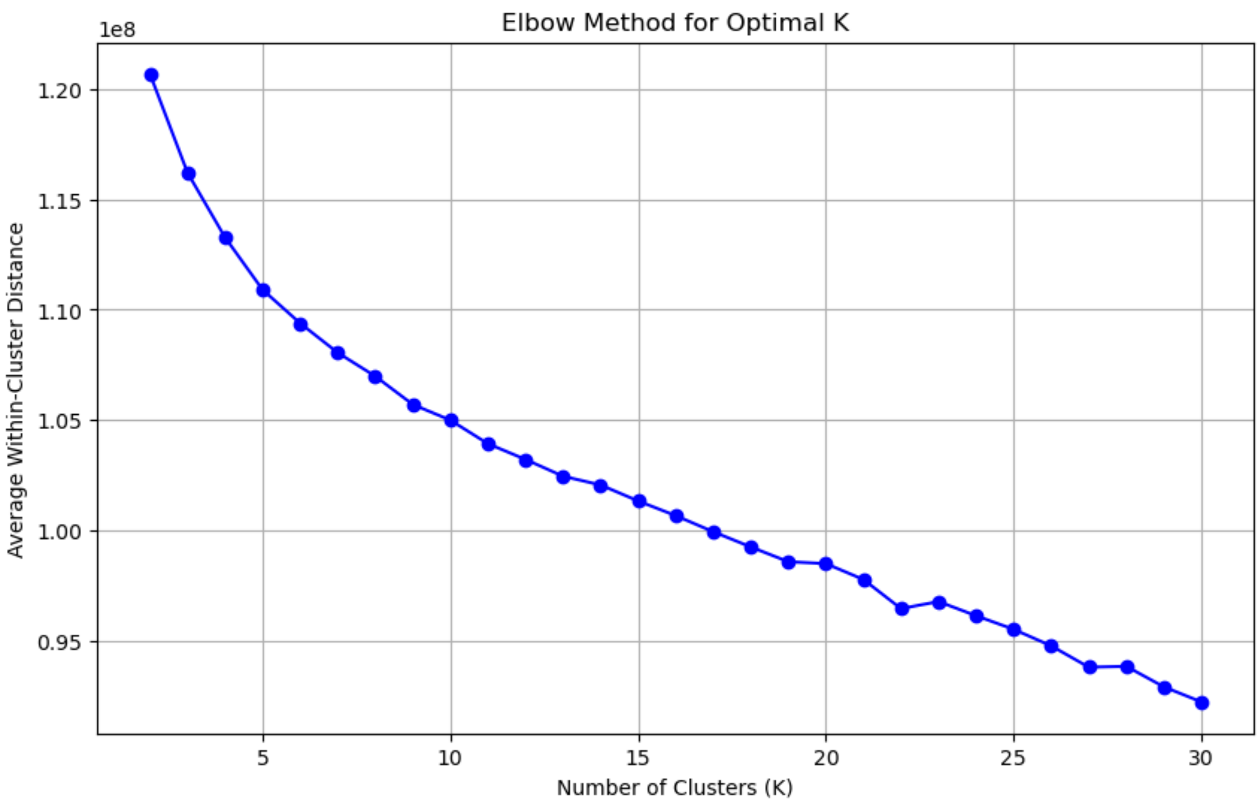
```
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
```

```
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
```

```
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
```

In [68]:

```
# Investigate the change in within-cluster distance across number of cluster
# HINT: Use matplotlib's plot function to visualize this relationship.

# Plot the average distances against the number of clusters
plt.figure(figsize=(10, 6))
plt.plot(cluster_counts_to_test, average_distances, marker='o', linestyle='-
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Average Within-Cluster Distance')
plt.grid(True)
plt.show()
```



In [69]:
```
# Re-fit the k-means model with the selected number of clusters and obtain
# cluster predictions for the general population demographics data.

# Optimal number of clusters
optimal_k = 16

# Create a KMeans instance with the selected number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=0)

# Fit the KMeans model to the scaled data (azdias_scaled) and obtain cluster
azdias_predictions = kmeans.fit_predict(azdias_pca_reduced)
```

```
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Discussion 3.1: Apply Clustering to General Population

I opted for 16 clusters because, as indicated by the visualization, this represents the final significant drop in the average within-cluster distance.

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 [70]:
```
# Load in the customer demographics data.
customers = pd.read_csv('Udacity_CUSTOMERS_Subset.csv', sep=';')

customers.head()
```

Out[70]:

	AGER_TYP	ALTERSKATEGORIE_GROB	ANREDE_KZ	CJT_GESAMTTYP	FINANZ_MINIMALIST
0	2	4	1	5.0	5
1	-1	4	1	NaN	5
2	-1	4	2	2.0	5
3	1	4	1	2.0	5
4	-1	3	1	6.0	3

5 rows × 85 columns

In [71]:
```
customers.shape
```

Out[71]:
```
(191652, 85)
```

In [72]:
```
# Apply preprocessing, feature transformation, and clustering from the gener
# demographics onto the customer data, obtaining cluster predictions for the
# customer demographics data.

customers_clean = clean_data(customers)

# *NOTE* I tried to run this code before, but had an error that this
# column was missing, so I am adding it to the dataframe.
customers_clean["GEBAEUDETYP_5.0"] = 0

# Get the column order from the model's original data (e.g., azdias_scaled)
original_column_order = azdias_scaled.columns

# Reorder columns in customers_clean to match the original order
customers_clean = customers_clean.reindex(columns=original_column_order)

# feature transformation,
imputer = SimpleImputer(strategy = 'most_frequent')
customers_imputed = imputer.fit_transform(customers_clean)

# Standardize the data in 'customers_imputed' using the scaler
customers_scaled = scaler.fit_transform(customers_imputed)

# Create a new DataFrame to store the standardized data
customers_scaled = pd.DataFrame(customers_scaled, columns=customers_clean.cc

# Apply PCA to customers data
customers_pca = pca.transform(customers_scaled)

#Clusterring  customers
kmeans_cust = KMeans(n_clusters = 15, random_state=0).fit(customers_pca)

customers_predictions = kmeans_cust.predict(customers_pca)
```

```
/Users/marcusthompson/anaconda3/lib/python3.11/site-packages/sklearn/cluster
/_kmeans.py:1412: FutureWarning: The default value of `n_init` will change f
rom 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
e warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Step 3.3: Compare Customer Data to Demographics Data

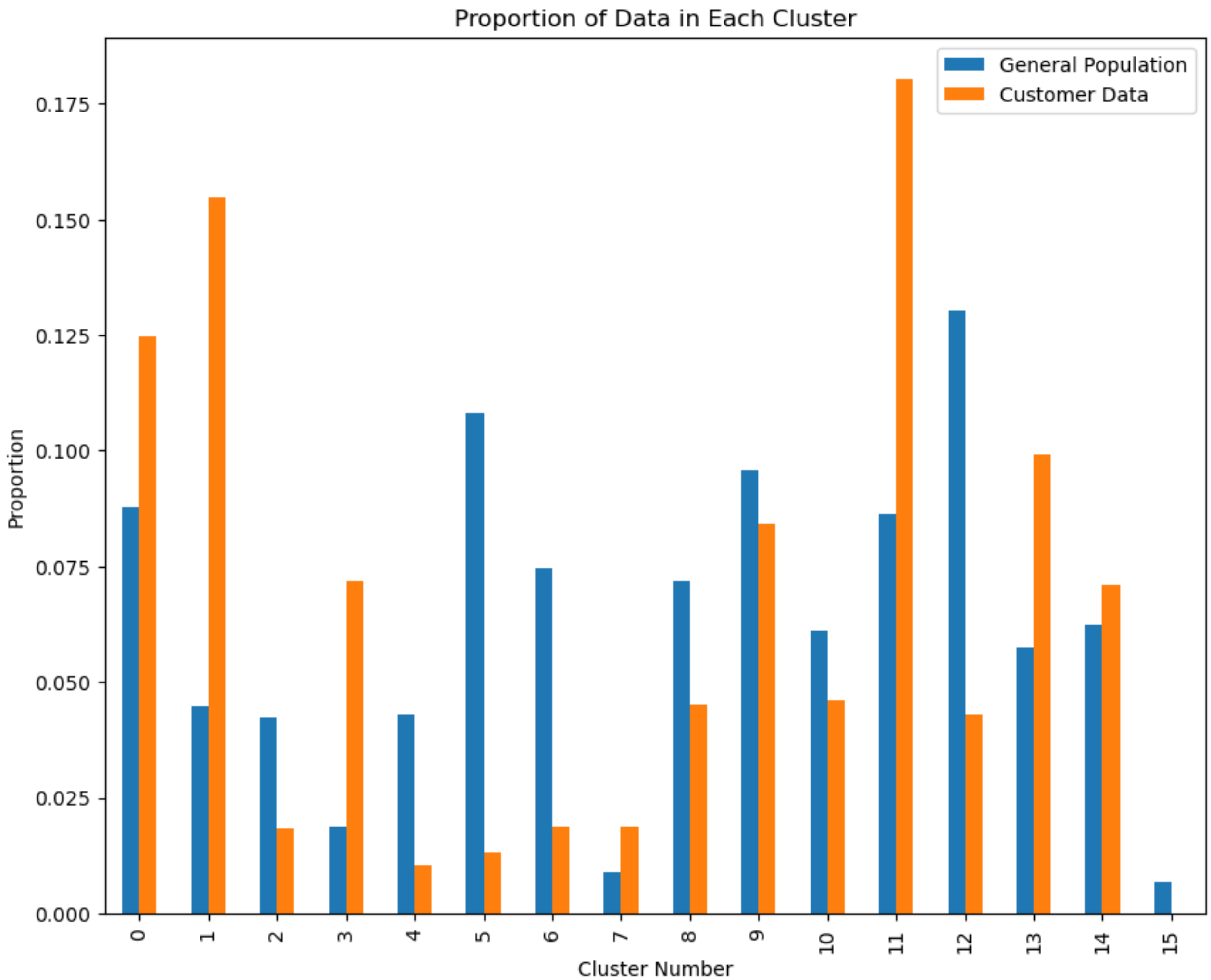
At this point, you have clustered data based on demographics of the general population of Germany, and seen how the customer data for a mail-order sales company maps onto those demographic clusters. In this final substep, you will compare the two cluster distributions to see where the strongest customer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

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

Take a look at the following points in this step:

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

In [73]:
```
# Compare the proportion of data in each cluster for the customer data to th
# proportion of data in each cluster for the general population.

# Compare the proportion of data in each cluster for the customer data to th
# proportion of data in each cluster for the general population.

# Get the proportion of data in each cluster for the customer data.
customers_preds_proportions = pd.Series(customers_predictions).value_counts(
azdias_preds_proportions = pd.Series(azdias_predictions).value_counts() / le

# Create a DataFrame to store the proportions for both datasets
cluster_proportions_df = pd.DataFrame({
    'General Population': azdias_preds_proportions,
    'Customer Data': customers_preds_proportions
})

# Create a bar plot to visualize the proportions
cluster_proportions_df.plot(kind='bar', figsize=(10, 8))
plt.xlabel('Cluster Number')
plt.ylabel('Proportion')
plt.title('Proportion of Data in Each Cluster')
plt.show()
```



In [79]:
```
# What kinds of people are part of a cluster that is overrepresented in the
# customer data compared to the general population?

# Reshape the cluster center into a 2D array with a single row
cluster_center_2d = model.cluster_centers_[3].reshape(1, -1)

# Inverse transform the reshaped cluster center
original_cluster_center = scaler.inverse_transform(pca.inverse_transform(clu

# Create a DataFrame to store the cluster center in the original data space
cluster_center_df = pd.DataFrame([pd.Series(original_cluster_center[0], inde
cluster_center_df.transpose().sort_values(0, ascending=False).head(15)
```

Out[79]:

	0
MIN_GEBAEUDEJAHR	1993.340723
KBA13_ANZAHL_PKW	690.404044
WOHNDAUER_2008	8.637339
INNENSTADT	5.211992
SEMIO_LUST	5.014138
SEMIO_VERT	4.848861
ORTSGR_KLS9	4.838761
SEMIO_ERL	4.757847
BALLRAUM	4.729578
ONLINE_AFFINITAET	4.549882
FINANZ_VORSORGER	4.499148
SEMIO_DOM	4.352952
SEMIO_SOZ	4.207129
REGIOTYP	4.154312
SEMIO_KRIT	4.055933

In [80]:

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

# Reshape the cluster center into a 2D array with a single row
cluster_center_2d = model.cluster_centers_[5].reshape(1, -1)

# Inverse transform the reshaped cluster center
original_cluster_center = scaler.inverse_transform(pca.inverse_transform(clu

# Create a DataFrame to store the cluster center in the original data space
cluster_center_df = pd.DataFrame([pd.Series(original_cluster_center[0], inde
cluster_center_df.transpose().sort_values(0, ascending=False).head(15)
```

Out[80]:

	0
MIN_GEBAEUDEJAHR	1992.792606
KBA13_ANZAHL_PKW	700.480228
ANZ_HAUSHALTE_AKTIV	66.153805
WOHNDAUER_2008	8.385264
SEMIO_LUST	6.719107
SEMIO_ERL	6.092859
ORTSGR_KLS9	6.068714
W_KEIT_KIND_HH	6.000566
FINANZ_VORSORGER	5.254080
LIFE_STAGE	4.855885
RETOURTYP_BK_S	4.603143
REGIOTYP	4.597505
HH_EINKOMMEN_SCORE	4.586551
EWDICHTE	4.574281
SEMIO_KRIT	4.565483

Discussion 3.3: Compare Customer Data to Demographics Data

Cluster 3

Over represented

- WOHNDAUER_2008 - 9 - length of residence (over ten years)
- INNENSTADT - 5 - distance to city center (10-20km)
- SEMIO_LUST - 5 - sensual minded (low affinity)
- SEMIO_VERT - 5 - dreamful (low affinity)
- ORTSGR_KLS9 - 5 - size of community (20,001-50,000 inhabitants)
- SEMIO_ERL - 5 - event-oriented (low affinityBALLRAUM - 5 - distance to nearest urban center (40-50km)
- ONLINE_AFFINITAET - 5 - online affinity (highest)
- FINANZ_VORSORGER - 5 - be prepared (very low)
- SEMIO_DOM - 4 - dominant-minded (average affinity)SEMIO_SOZ - 4 - socially-minded (average affinity)
- REGIO_TYP - 4 - neighborhood typology (middle class)

- SEMIO_KRIT - 4 - critial-minded (average affinity)

After checking our results above with the data dictionary, we can see the overrepresented customer population in cluster three shares some interesting qualities. Notably, they tend to have a stable household, having maintained the same residence for over ten years. They also live fairly close to the city center, in fairly small, middle class neighborhoods. This population tends to display moderate personality traits, such as an average affinity for being dominant-minded, socially-minded, and critical minded. Despite this seemingly stable lifestyle, this population exhibits a 'very low' affinity for financial preparedness.

Cluster 5

Under represented

- SEMIO_LUST - 7 - sensual minded (lowest affinity)
- SEMIO_ERL - 6 - event-oriented (very low affinity)
- ORTSGR_KLS9 - 6 - size of community (50,001-100,000 inhabitants)
- W_KEIT_KIND_HH - 6 - likelihood of children in household (very unlikely)
- FINANZ_VORSORGER - 5 - be prepared (very low)
- RETOURTYP_BK_S - 5 - shopping return habits (minimal returner)
- REGIOTYP - 5 - neighborhood typology (lower middle class)
- HH_EINKOMMEN_SCORE - 5 - estimated household income (lower income)
- EWDICHTE - 5 - density of households per square km (320-999 per square km)
- SEMIO_KRIT - 5 - critical minded (low affinity)

After checking our results above with the data dictionary, we can see the under represented customer population in cluster five shares some interesting, and somewhat similar qualities. They tend to live in densely populated, lower middle-class communities that are larger than those from cluster three (50,000-100,000 vs 20,000-50,000 inhabitants). They also are unlikely to have children in the household. Like the population from cluster three, they also exhibit low affinity for financial preparedness. They are less critical-minded, sensual-minded, and event-oriented, as well. An interesting trait that they display is that they are least likely to return a purchase, which is good for an online retailer.

> 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 [ ]:

```
```