Signature Assignment: EDA for a Novel Data Set

Richard Martin Flores

School of Technology, Northcentral University

TIM-8501: Exploratory Data Analysis

Dr. Amir Schur

September 28th, 2022

EDA for a Novel Data Set

In this research assignment, we will conclude our study of the Exploratory Data Analysis lifecycle through study and practice of the Exploratory Analysis process. Exploratory data analysis involves the use of statistical and visualization techniques to help explore and summarize important aspects of data. We can use exploratory data analysis to gain insight into data before embarking on a data analysis project. In our conclusion to TIM-8501 Exploratory Data Analysis we will revisit each component of the EDA process including Measures of variability and central tendency, frequency, variance, and standard deviation, outlier detection and distribution modality, univariate analysis methods, and cluster analysis and data grouping. In this analysis we will use the NCU provided dataset *2019 CPS Food Security*. The goal of this research study is to understand the data better before making assumptions by examining variables through cluster analysis or segmentation.

Data Preparation

We begin our exploratory data analysis process with the initial step of data cleaning. NCU has afforded students with the options to munge data with R, Python, or Rapid Miner Studio. This analysis will use a combination of Python and the Tableau software which is like Rapid Miner Studio. Our research study begins with a display of the data cleaning process in Jupyter Notebook using Python and then we can begin the EDA process and move onto the next step of Exploratory Analysis. We will begin our analysis with data cleaning and exploration in Tableau. Then we will calculate and create visualizations for our cluster analysis using the Python language. The full Jupyter Notebook code and output can be found in the appendix.

Research Question

In this research study we will pose and analyze the following questions:

1. Is there a relationship between a recipient's living quarters and method of payment?
2. Is there a relationship between a recipient's income and housing type?
3. Is there a relationship between a recipient's income and level of education?

Hypothesis

We will make the following hypotheses about our research questions which will be answered after completing the cluster analysis.

1. We expect that a majority of recipients of assistance will fall into the occupied with payment of cash rent category and live in an apartment type of dwelling.
2. We can assume that most recipients will have a very low or no income and live in a mobile home or apartment.
3. We assume that as a recipient's level of education increases so will their level of income.

Dataset Dimensions

The data used in this research study was acquired from census.gov which is the official website for the United States Census Bureau, a repository for publicly accessible open data published by state agencies and the federal government. The dataset acquired for this research study is titled *2019 CPS Food Security* which was sourced from the Census CPS Team, Demographic Surveys Division, Bureau of the Census (Bureau U.S.C, 2021). The shape of the dataset contains 138,965 rows and 510 columns and each record represents a person or family subscribed to a government food assistance program.

Known Dimensions

The known dimensions of the dataset are derived from the raw data downloaded from the census.gov website. As previously noted, the dataset contains 138,965 rows and 510 columns and each record represents a person or family subscribed to a government food assistance program. As part of the data cleaning process, we will reduce the known dimensions of the dataset with the following changes:

- Remove “Filler” variable as it contains only null values.

Next, we will calculate Eigen vectors and Eigen values as part of the Principal Component Analysis (PCA) to determine our unknown dimensions.

Unknown Dimensions – Principal Component Analysis

The unknown dimensions are a result of the Principal Component Analysis process. Before using a clustering method on the data, principal component analysis (PCA) is conducted to reduce the dimensionality and "noise" in the dataset (Mukhiya et al., 2020). After completing a PCA analysis our dataset is reduced to only variables who contribute to meaningful insights. Thus, our final dimensions of our dataset are unknown until the PCA process is completed.

The PCA analysis is composed of four steps, data standardization, computing the covariance matrix, computing the eigenvectors and eigenvalues, and finally computing our feature vector. Principal component analysis for this research report was calculated using Python and the code can be seen in figure 1 below.

Figure 1

Python Code for calculating Eigenvalues and Eigenvectors.

```
# Standardize the data
data -= data.mean(axis=0)

# Calculate the covariance matrix
cov = np.cov(data, rowvar=False)

# Calculate eigenvalues and eigenvector of the covariance matrix
evals, evecs = la.eig(cov)

# Multiply the original data matrix with Eigenvector matrix.
# Sort the Eigen values and vector and select components
num_components=2
sorted_key = np.argsort(evals)[::-1][:num_components]
evals, evecs = evals[sorted_key], evecs[:, sorted_key]

print("Eigenvalues:", evals)
print("Eigenvector:", evecs)
print("Sorted and Selected Eigen Values:", evals)
print("Sorted and Selected Eigen Vector:", evecs)

# Multiply original data and Eigen vector
principal_components=np.dot(data,evecs)
print("Principal Components:", principal_components)

# Import principal component analysis
from sklearn.decomposition import PCA

# Create and fit_transformed PCA Model
pca_model = PCA(n_components=2)
components = pca_model.fit_transform(data)
components_df = pd.DataFrame(data = components,

columns = ['principal_component_1', 'principal_component_2'])
print(components_df)
```

The output from the PCA analysis can be found in the Jupyter Notebook in the appendix section of this report. The results from our PCA analysis have distilled the dataset into its most meaningful variables and to gain the greatest insights from the dataset we should focus our cluster analysis on the following variables, "*HETENURE*, *HEHOUSUT*, *HEFAMINC*, *PEEDUCA*".

Univariate Analysis

Univariate analysis contains a plethora of available methods for analyzing data and we will provide a brief description of Hypothesis Tests used in this analysis as described in *Python Data Analysis.*

Parametric Tests

- One-Sample T-Test - A t-test is a type of parametric test used to check if there is a significant difference between the means of the two participating groups. A single sample t-test is used to check if there is a significant difference between the sample and the estimated population mean.
- Two-Sample T-Test - The two-sample t-test is used to compare the significant difference between two independent groups, also known as the independent samples t-test.
- Paired Sample T-Test - A paired sample t-test is a paired t-test used to determine if the mean difference between two observations in the same group is zero.

Non-Parametric Tests

- Mann-Whitney U Test - The Mann-Whitney U-test is the non-parametric equivalent of the two-sample t-test. The difference between the samples is not considered a normal distribution.

It is unfortunate that our dataset does not lend itself well to univariate analysis as the variables have been converted from their true values into a data dictionary for ease of access. While completing the univariate analysis using various methods, we find that we have one accepted hypothesis using a One-Sample T-Test as show below in figure 2.

Figure 2

Hypothesis Testing.

One-Sample T-Test

```
from scipy.stats import ttest_1samp
```

```
t_test_value, p_value = ttest_1samp(data_HEFAMINC, 10)

print("P Value:",p_value)

print("t-test Value:",t_test_value)

# 0.05 or 5% is significance level or alpha.
if p_value < 0.05:

    print("Hypothesis Rejected")

else:

    print("Hypothesis Accepted")
```

```
P Value: 0.4144072283483081
t-test Value: 0.8161648993591065
Hypothesis Accepted
```

The One-Sample T-Test confirms our hypothesis that the median level of income across all families is in the 35,000 TO 39,999 range. The median household income in the USA is $67,000 and this shows the average level of income of food security recipients is below the national average (Amadeo, 2022).

In the appendix section of this research report, the Jupyter Notebook output shows our calculations of univariate analysis tests including a Two-Sample T-Test, Paired Sample T-Test, and Mann-Whitney U Test. However, these hypothesis tests were all rejected due to the dictionary data format of the dataset.

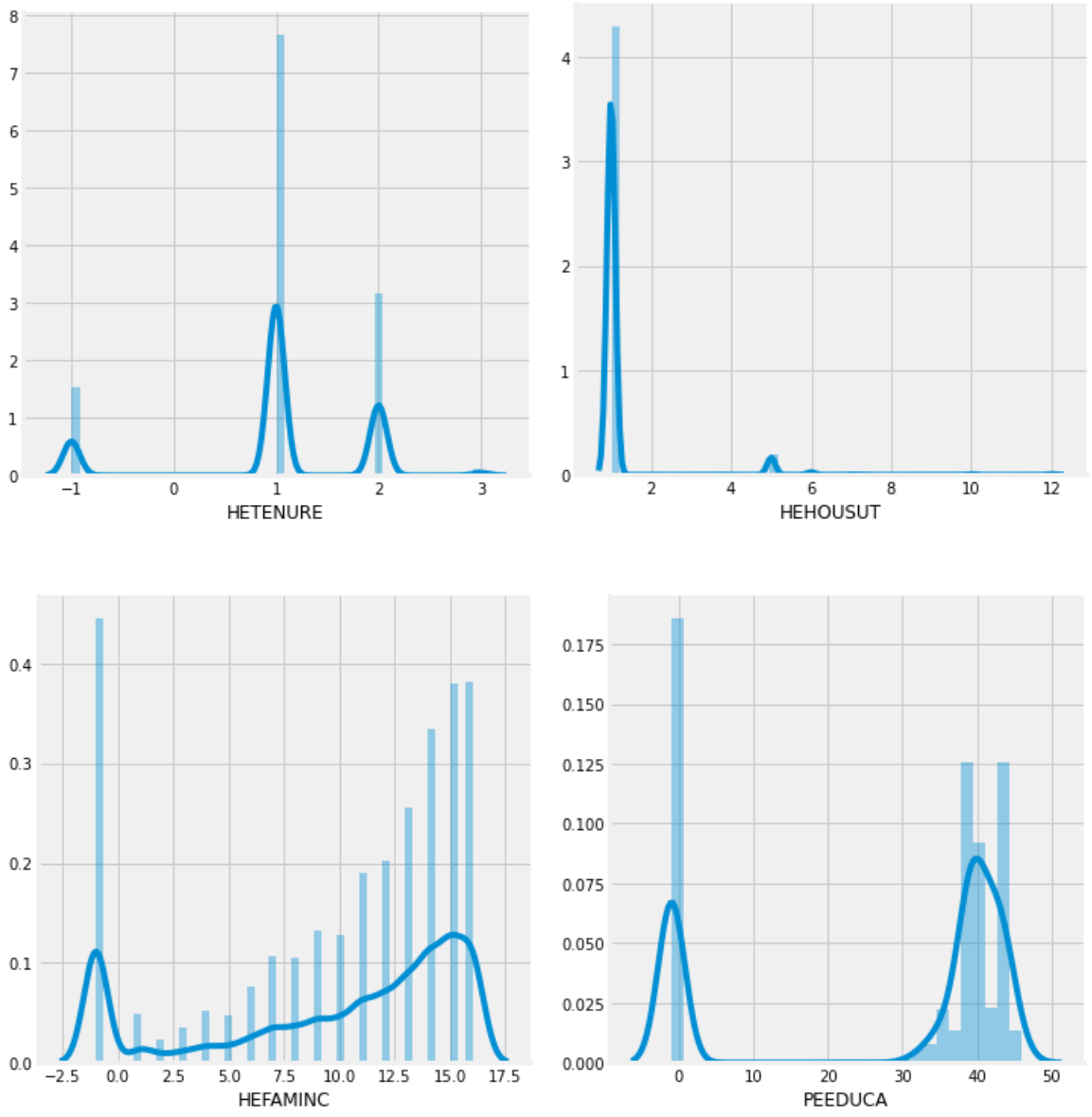
In the future we may desire to conduct further univariate analysis, and for our efforts to be successful we would need to request a full dataset from the US Census Bureau which contains the actual values reported by food security program subscribers so that we may properly ascertain values for our univariate analysis.

Univariate Distribution Plots

To assess the distribution modality of the variables being examined, we will use visual representations in the form of Distribution Plots. We can use the distribution plots to visually confirm distributions for attributes such as unimodal, bimodal, multimodal, symmetry, and skewness.

Figure 3

Distribution Plots for determining Modality.



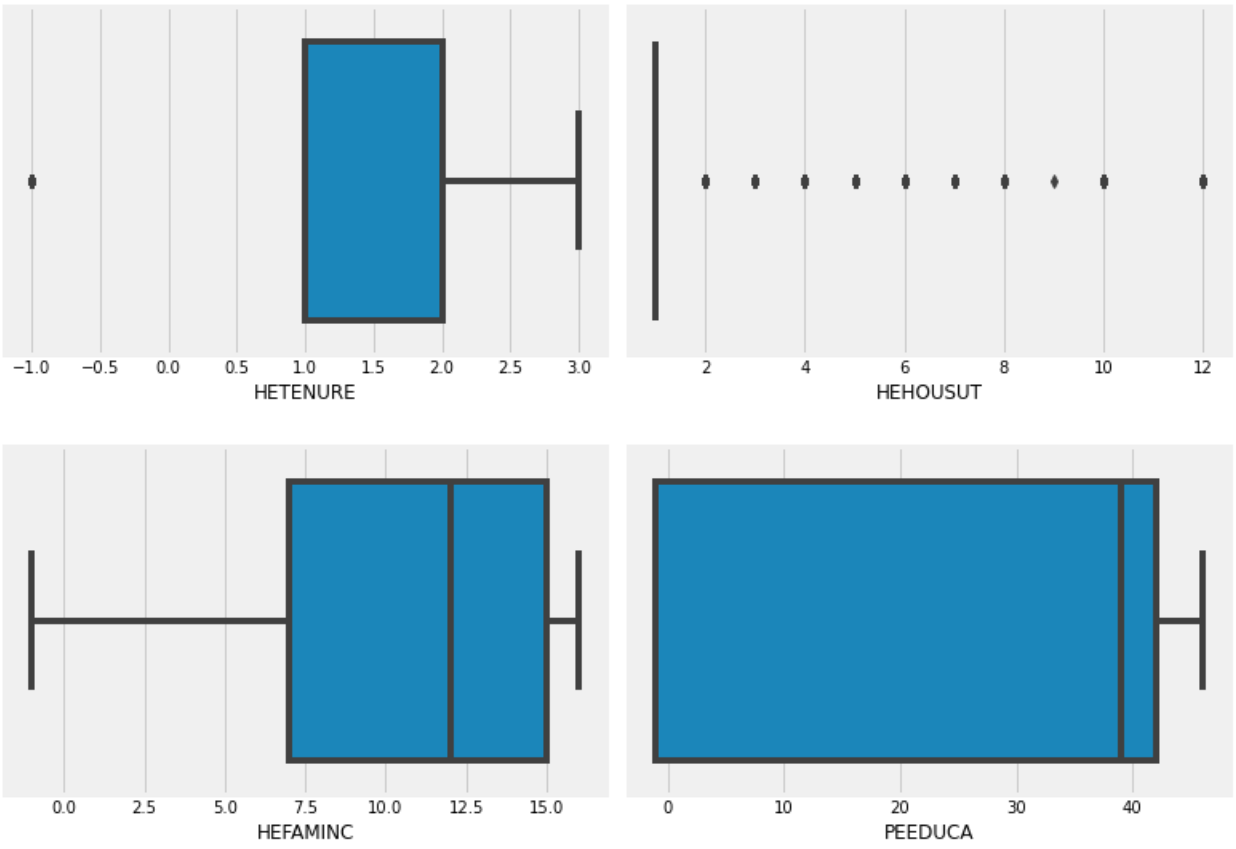
Examining our distribution plots, we see a wide range of attributes for each graph. In the case of modality, we see examples of unimodality such as in the variable HEHOUSUT, we see bimodality in PEEDUCA and HEFAMINC, and multimodality in HETENURE. We also see symmetry such as in HEHOUSUT and non-symmetry in HEFAMINC. We also see a left-skew attribute in HEFAMINC.

Outlier Detection

One time-tested method of detecting outliers is using Boxplots, also known as box and whisker plots. This is a great method for visually verifying the existence of outliers in a variable. We will create four boxplots to examine our variables being analyzed.

Figure 4

Boxplots for detecting existence of outliers.



In our assessment of outliers, it appears our variables are quite uniform; however, we do see what may appear to be outliers in the HETENURE and HEHOUSUT variables. In order to properly assess if these are indeed outliers or natural factors of the dataset, we must further examine the variables through the use of bins and their corresponding data dictionary values. It may be necessary to remove records containing the observed outliers or impute the records with either median or a set-value for analysis.

Cluster Analysis and Segmentation Methods

The process of completing a Cluster Analysis includes several preparation steps including choosing a similarity measure, choosing a grouping method, and selecting the number of clusters. In this report, we will elaborate on segmentation and cluster methods.

Segmentation Method

Two popular and effective segmentation methods are The Elbow Method and The Silhouette Method. In this analysis we will use the elbow method to determine the proper K-value or number of clusters to create for each pair of variables under analysis.

The elbow method is a popular method to find the best number of clusters. In this approach, we focus on the percentage variance of different numbers of clusters. The basic concept of this method is to choose the number of clusters that should not cause large fluctuations when adding another cluster. We can plot the number of squares in a cluster using the number of clusters to find the optimal value. The sum of squares is also called Within Cluster Sum of Squares (WCSS).

Clustering Method

Several clustering techniques have been used in academia and industry alike which produce reliable and insightful results. Clustering methods include K-Means, Hierarchical, DBSCAN, and Spectral Clustering.

K-means is one of the simplest and best-known clustering algorithms. It's kind of a method of grouping partitions (Fandango et al., 2021). Based on the specified number of clusters, the input is partitioned by specifying a random starting cluster center. In the next iteration, data items are assigned to the closest cluster center using the Euclidean distance. Thanks to this algorithm, the center of the first group can be selected manually or randomly.

Cluster Analysis and Segmentation Process

To complete the Cluster Analysis for this research report, preparatory and intermediate steps were necessary before the data could be processed. Initially it is imperative to reduce dimensionality and noise in the dataset through Principal Component Analysis. After defining our research questions and hypotheses, the next step was the choose a similarity measure. For this analysis we chose Euclidean distance to measures similarity as it is this most common similarity measure uses the shortest distance between two points in 2-dimensional space for continuous or numerical variables using a scale between 0 and 1 (Fandango et al., 2021). Our next step was the calculate and select the appropriate number of clusters for analysis. In this report we used the Elbow Method which is used to plot the data based on variation to determine how many clusters and at what point the variation becomes minimal. The "elbow of the curve" denotes the number of clusters that should be used (Fandango et al., 2021). Finally, we selected the Partitional grouping method which divides the data into a set number of groups by using the

measurement of the distance from the centroid to the data point for each cluster (Fandango et al., 2021). Once the distance from the centroid is measured for each cluster, the data point is placed into the cluster with the least distance.

In this analysis, we justify our selection of tools and techniques based on the requirements necessary for the cluster analysis algorithms to function. A core requirement of many tools used was the condition that variables in the dataset were of integer type. It was also necessary to impute null and NaN values as the statistical calculations will not run with these values in the dataset

Statistical Findings and Insights

Before answering our research questions, it is important to define certain variables of critical importance to the analysis.

HETENURE
LIVING QUARTERS
VALID ENTRIES
1 OWNED OR BEING BOUGHT BY A HH MEMBER
2 RENTED FOR CASH
3 OCCUPIED WITHOUT PAYMENT OF CASH RENT

HEHOUSUT
TYPE OF HOUSING
VALID ENTRIES
0 OTHER UNIT
1 HOUSE, APARTMENT, FLAT
2 HU IN NONTRANSIENT HOTEL, MOTEL, ETC.
3 HU PERMANENT IN TRANSIENT HOTEL, MOTEL
4 HU IN ROOMING HOUSE
5 MOBILE HOME OR TRAILER W/NO PERM. ROOM ADDED
6 MOBILE HOME OR TRAILER W/1 OR MORE PERM. ROOMS ADDED
7 HU NOT SPECIFIED ABOVE
8 QUARTERS NOT HU IN ROOMING OR BRDING HS
9 UNIT NOT PERM. IN TRANSIENT HOTL, MOTL
10 UNOCCUPIED TENT SITE OR TRLR SITE
11 STUDENT QUARTERS IN COLLEGE DORM
12 OTHER UNIT NOT SPECIFIED ABOVE

HEFAMINC
FAMILY INCOME
VALID ENTRIES
1 LESS THAN $5,000
2 5,000 TO 7,499
3 7,500 TO 9,999
4 10,000 TO 12,499
5 12,500 TO 14,999
6 15,000 TO 19,999
7 20,000 TO 24,999
8 25,000 TO 29,999
9 30,000 TO 34,999
10 35,000 TO 39,999
11 40,000 TO 49,999
12 50,000 TO 59,999
13 60,000 TO 74,999
14 75,000 TO 99,999
15 100,000 TO 149,999
16 150,000 OR MORE

PEEDUCA
HIGHEST LEVEL OF SCHOOL
COMPLETED OR DEGREE RECEIVED
VALID ENTRIES
31 LESS THAN 1ST GRADE
32 1ST, 2ND, 3RD OR 4TH GRADE
33 5TH OR 6TH GRADE
34 7TH OR 8TH GRADE
35 9TH GRADE
36 10TH GRADE
37 11TH GRADE
38 12TH GRADE NO DIPLOMA
39 HIGH SCHOOL GRAD-DIPLOMA OR EQUIV (GED)
40 SOME COLLEGE BUT NO DEGREE
41 ASSOCIATE DEGREE-OCCUPATIONAL/VOCATIONAL
42 ASSOCIATE DEGREE-ACADEMIC PROGRAM
43 BACHELOR'S DEGREE (EX: BA, AB, BS)
44 MASTER'S DEGREE (EX: MA, MS, MEng, MEd, MSW)
45 PROFESSIONAL SCHOOL DEG (EX: MD, DDS, DVM)
46 DOCTORATE DEGREE (EX: PhD, EdD)

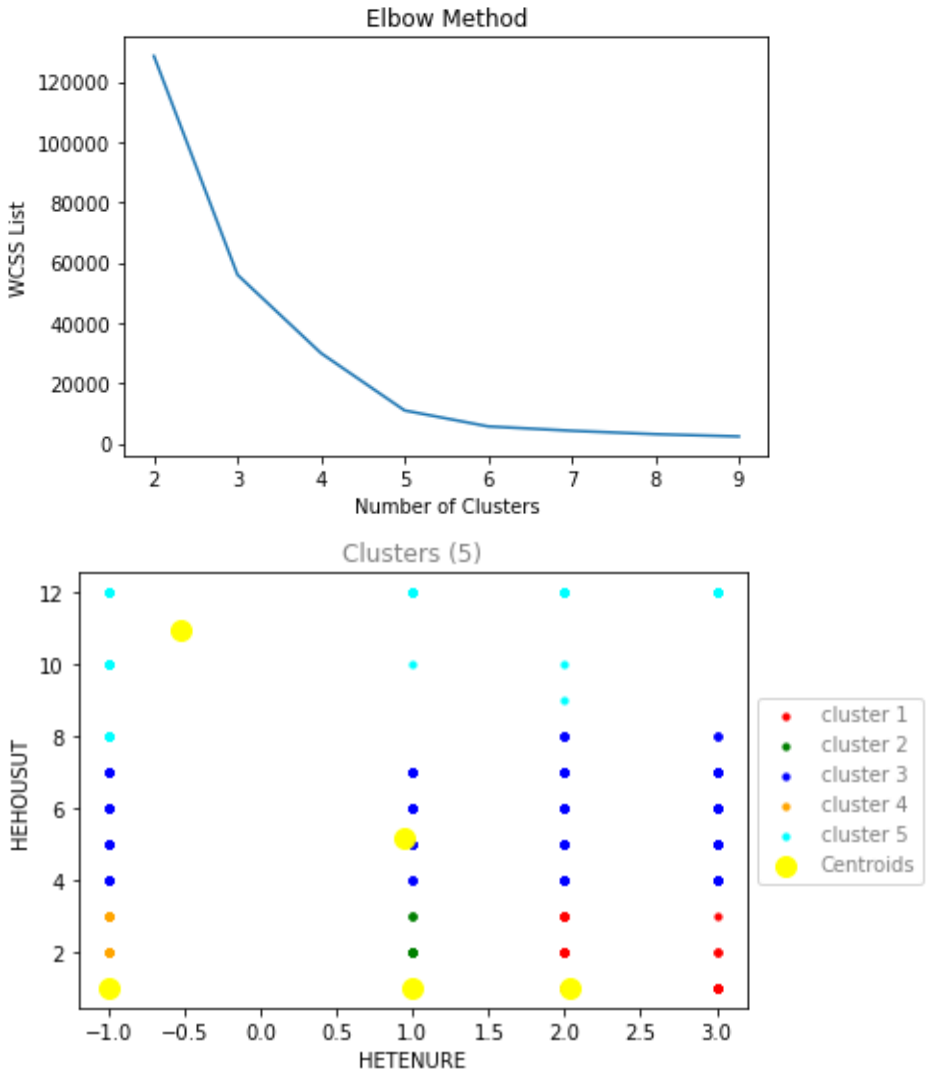
Findings

Research Question 1 - Is there a relationship between a recipient's living quarters and method of payment?

The Cluster Analysis results to our first research question are displayed in figure 5 below.

Figure 5

Cluster Analysis of First Research Question.



Using the elbow method segmentation process we determined that four is the optimal number of clusters for these variables. Our research question predicts that most recipients of

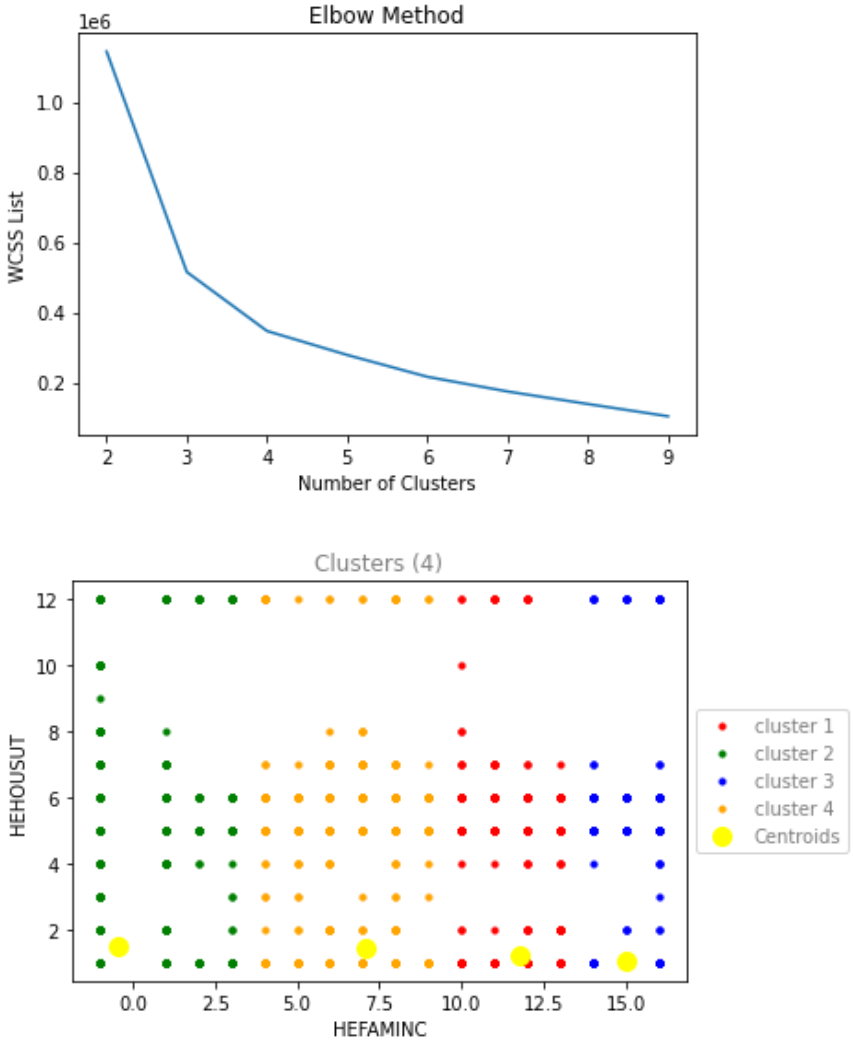
assistance will fall into the occupied without payment of cash rent category and live in an apartment type of dwelling. Our cluster analysis shows three centroids relevant to our question. Two centroids lie along the owned or being bought section of housing with the majority or recipients living in a house/apartment or mobile home. Our hypothesis for this research question is accepted.

Research Question 2 - Is there a relationship between a recipient's income and housing type?

The Cluster Analysis results to our second research question are displayed in figure 6 below.

Figure 6

Cluster Analysis of second Research Question.

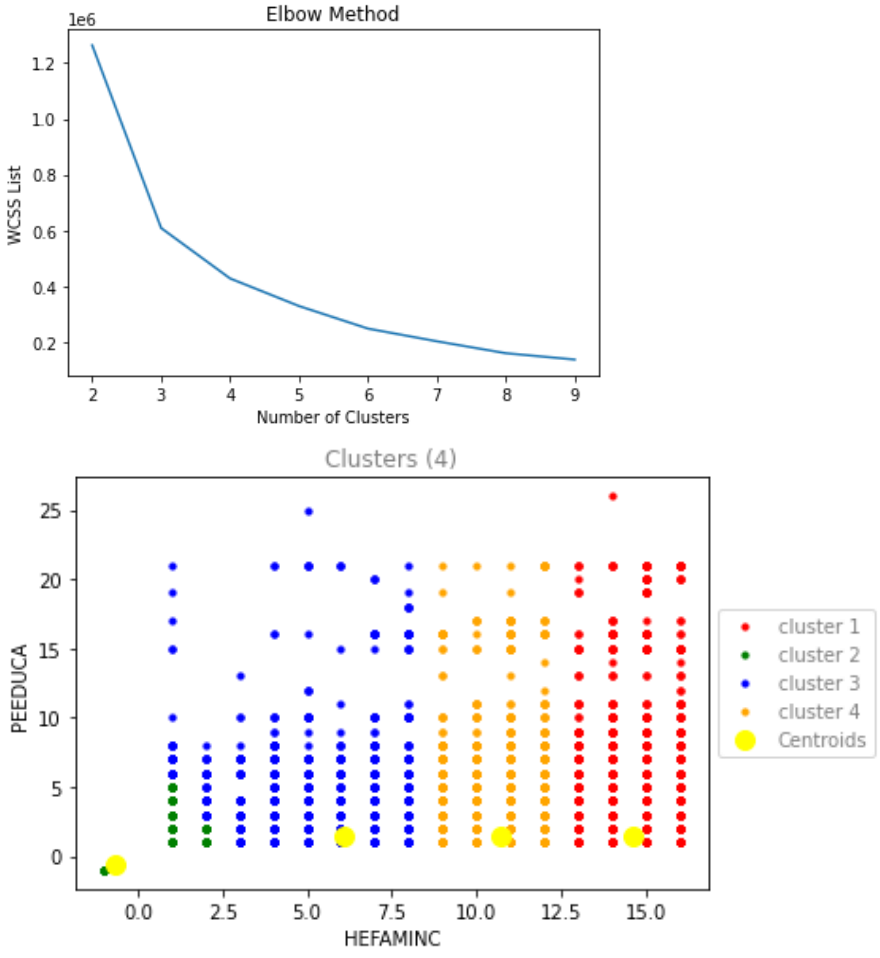


In our second cluster analysis, we see distinct groups across our four predetermined clusters. Across all four centroids we see that most recipients live in a house or apartment and are representative of four income groups. The first group is those with an income of less than $5000, the second group possesses an income of 15,000 TO 19,999, the third group has an income level between 40,000 TO 49,999, and our fourth group has an income level between 100,000 TO 149,999. We can accept our hypothesis that most recipients live in a house or apartment and possess a low or no level of income.

Research Question 3 - Is there a relationship between a recipient's income and level of education?

Figure 7

Cluster Analysis of third Research Question.



In our third cluster analysis, we hypothesized that as a recipient's level of education increases so will their level of income. In examining our cluster, we can observe four centroids with a common trend amongst all groups having a low level of education. There does not appear to be any correlation between level of education and income in this cluster. We can reject our hypothesis for our third research question.

Insights

In our research study of the 2019 CPS Food Security dataset through cluster analysis and segmentation we have gathered several key insights and recommendations. Before even starting our analysis, it would be logical to assume that families with low or no income would be the most dependent on a food security program. While we are correct to hypothesize that program recipients live in a house or apartment, it seems that those needing the most assistance are the families that own or are in the processing or purchasing their house and not families that are currently renting or leasing their housing.

We also see that in addition to residing in a house or apartment, a large group of recipients also live in mobile homes. A great insight discovered is that while low-income families find themselves in need of assistance, we also see a cluster of families in the 100,000 to 149,999 that qualify and receive food security services. Of course, many factors can be attributed to having a high level of income and needing assistance from a food security program such a dwelling location and the size of a family but it is interesting to see that medium to high level income families also subscribe to food security services.

Finally, we observed that the majority of families across all income levels possess a low level of education. There doesn't appear to be a correlation in this dataset between level of income and level of education.

Recommendations

From our exploratory data analysis and conclusions drawn from the univariate and cluster analysis we can recommend that the CPS Food Security program continue to aid and take proactive measures for certain demographics. Those that are in more dire need of aid may not be aware of assistance programs and usually include families with a low level of education and low income that are currently purchasing or have purchased their home. We recommend that families in this demographic receive service catered to families who may be struggling to find work or are limited to part-time employment.

We recommend a continued study of this dataset with an additional component of geophysical data so that we may observe trends in food security programs by city and region. Knowing which populations may be most in need of food security programs may help to target services such as education and employment opportunities. The more these targeted populations have access to education and the greater their income, the less reliant they will be on a food security program enabling resources to be further directed into improving the economy and health of our citizens.

References

365 Careers. (2018). *Statistics for data science and business analysis.* Packt Publishing.

Amadeo, K. (n.d.). What is the average income in the United States? The Balance. Retrieved September 28, 2022, from https://www.thebalancemoney.com/what-is-average-income-in-usa-family-household-history-3306189

Bureau, U.S.C. (2021, October 8). 2019 CPS Food Security. Census.gov. Retrieved September 28, 2022, from https://www.census.gov/data/datasets/2019/demo/cps/cps-food-security.html Fandango, A., Idris, I., & Navlani. A. (2021). *Python data analysis* (3rd ed.). Packt Publishing.

McKinney, W. (2017). *Python for data analysis* (2nd ed.). O'Reilly Media, Inc.

Mukhiya, S. K., & Ahmed, U. (2020). *Hands-on exploratory data analysis with Python.* Packt Publishing.

Week 8 Data Exploration Jupyter Notebook

September 28, 2022

1 TIM-8501 Exploratory Data Analysis - Week 08

1.0.1 Import Necessary Python Libraries

[1]:
```
# Pandas is necessary for array manipulation and calculation
import pandas
from pandas import DataFrame

# Matplotlib library to create visualizations
import matplotlib.pyplot as plt

# Missingno used for missing data visualization
import missingno as msno

# NumPy necessary for statistical calculations
import numpy as np
np.set_printoptions(threshold=np.inf)

# Change theme of graphs
plt.style.use('fivethirtyeight')

# Export images from Jupyter to PDF
%matplotlib inline

# Import Seaborn to graph distplots and boxplots
import seaborn as sns
```

1.0.2 Import Cleaned COVID dataset

[2]:
```
data = pandas.read_csv('dec19pub.csv', low_memory=False)
```

1.0.3 Verify Variable data types

[3]:
```
data.dtypes
```

[3]:
```
HRHHID      float64
HRMONTH       int64
```

```
HRYEAR4       int64
HURESPLI      int64
HUFINAL       int64
              …
HRFS30D6      int64
HRFS30D7      int64
HRFS30D8      int64
HRFS30D9      int64
HRFS30DE      int64
Length: 509, dtype: object
```

1.0.4 Check for null values in dataset

[4]:
```
data.isnull().sum()
```

[4]:
```
HRHHID      0
HRMONTH     0
HRYEAR4     0
HURESPLI    0
HUFINAL     0
            ..
HRFS30D6    0
HRFS30D7    0
HRFS30D8    0
HRFS30D9    0
HRFS30DE    0
Length: 509, dtype: int64
```

1.0.5 Visualize missing values in dataset

[5]:
```
msno.matrix(data)
```

[5]:
```
<AxesSubplot:>
```

1.0.6 Calculate five-number summary including interquartile range

[6]:
```
data.describe()
```

[6]:

	HRHHID	HRMONTH	HRYEAR4	HURESPLI	HUFINAL
count	1.389640e+05	138964.0	138964.0	138964.000000	138964.000000
mean	3.515847e+14	12.0	2019.0	1.003994	189.968085
std	2.926092e+14	0.0	0.0	0.964884	53.092961
min	4.792610e+09	12.0	2019.0	-1.000000	1.000000
25%	1.053120e+14	12.0	2019.0	1.000000	201.000000
50%	2.660010e+14	12.0	2019.0	1.000000	201.000000
75%	6.005100e+14	12.0	2019.0	1.000000	201.000000
max	9.997430e+14	12.0	2019.0	10.000000	259.000000

	HULANGCODE	HETENURE	HEHOUSUT	HETELHHD
count	138964.000000	138964.000000	138964.000000	138964.000000
mean	0.036448	1.027338	1.252958	0.763428
std	0.187403	0.884544	1.113010	0.713954
min	0.000000	-1.000000	1.000000	-3.000000
25%	0.000000	1.000000	1.000000	1.000000
50%	0.000000	1.000000	1.000000	1.000000
75%	0.000000	2.000000	1.000000	1.000000
max	1.000000	3.000000	12.000000	2.000000

	HETELAVL	…	HRFS30D1	HRFS30D2	HRFS30D3
count	138964.000000	…	138964.000000	138964.000000	138964.000000
mean	-0.855322	…	0.206053	0.264759	-0.205960
std	0.651628	…	1.104481	1.205541	1.346232
min	-3.000000	…	-9.000000	-9.000000	-9.000000
25%	-1.000000	…	-1.000000	-1.000000	-1.000000
50%	-1.000000	…	1.000000	1.000000	0.000000
75%	-1.000000	…	1.000000	1.000000	0.000000
max	2.000000	…	3.000000	4.000000	17.000000

	HRFS30D4	HRFS30D5	HRFS30D6	HRFS30D7
count	138964.000000	138964.000000	138964.000000	138964.000000
mean	20.294853	-0.454190	-0.702067	7.333648
std	113.857406	0.940244	0.620612	76.053052
min	-9.000000	-9.000000	-9.000000	-9.000000
25%	-6.000000	-1.000000	-1.000000	-6.000000
50%	-6.000000	-1.000000	-1.000000	-1.000000
75%	-1.000000	1.000000	0.000000	-1.000000
max	1216.000000	3.000000	7.000000	1150.000000

	HRFS30D8	HRFS30D9	HRFS30DE
count	138964.000000	138964.000000	138964.000000
mean	0.259017	-0.238659	20.461386

```
std         1.197984       1.193925     117.159890
min        -9.000000      -9.000000      -9.000000
25%        -1.000000      -1.000000      -6.000000
50%         1.000000       0.000000      -6.000000
75%         1.000000       0.000000      -1.000000
max         4.000000      10.000000    1105.000000

[8 rows x 509 columns]
```

1.0.7 Calculate Standard Deviation

[7]:
```
data.std()
```

[7]:
```
HRHHID      2.926092e+14
HRMONTH     0.000000e+00
HRYEAR4     0.000000e+00
HURESPLI    9.648843e-01
HUFINAL     5.309296e+01
               …
HRFS30D6    6.206125e-01
HRFS30D7    7.605305e+01
HRFS30D8    1.197984e+00
HRFS30D9    1.193925e+00
HRFS30DE    1.171599e+02
Length: 509, dtype: float64
```

1.0.8 Calculate Mode

[8]:
```
data.mode(axis=0, numeric_only=False, dropna=True)
```

[8]:
```
          HRHHID  HRMONTH  HRYEAR4  HURESPLI  HUFINAL  HULANGCODE  HETENURE  \
0  7.100030e+14       12     2019         1      201           0         1

   HEHOUSUT  HETELHHD  HETELAVL  …  HRFS30D1  HRFS30D2  HRFS30D3  HRFS30D4  \
0         1         1        -1  …         1         1         0        -6

   HRFS30D5  HRFS30D6  HRFS30D7  HRFS30D8  HRFS30D9  HRFS30DE
0        -1        -1        -1         1         0        -6

[1 rows x 509 columns]
```

1.0.9 Calculate Median

[9]:
```
median_array = np.median(data, axis=0)
print(median_array)
```

```
[ 2.66001000e+14  1.20000000e+01  2.01900000e+03  1.00000000e+00
  2.01000000e+02  0.00000000e+00  1.00000000e+00  1.00000000e+00
  1.00000000e+00 -1.00000000e+00  1.00000000e+00  1.20000000e+01
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00  2.55796150e+07
  1.00000000e+00  3.00000000e+00  1.00000000e+00  5.00000000e+00
  1.00000000e+00  0.00000000e+00  2.00000000e+00  9.11100000e+03
  1.00000000e+00  2.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00  3.00000000e+00  5.00000000e+00
  2.90000000e+01  2.47800000e+04  0.00000000e+00  2.00000000e+00
  1.00000000e+00  0.00000000e+00  4.00000000e+00  0.00000000e+00
  2.00000000e+00 -1.00000000e+00  3.40000000e+01  0.00000000e+00
  1.00000000e+00 -1.00000000e+00  1.00000000e+00  2.00000000e+00
  2.00000000e+00  3.90000000e+01  1.00000000e+00 -1.00000000e+00
  9.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
  1.00000000e+00  2.00000000e+00  1.00000000e+00  2.00000000e+00
  5.70000000e+01  5.70000000e+01  5.70000000e+01  1.00000000e+00
  0.00000000e+00  0.00000000e+00  1.00000000e+00  1.00000000e+00
  1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00  1.00000000e+00 -1.00000000e+00
  0.00000000e+00  1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00  0.00000000e+00  1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
```

-1.00000000e+00	-1.00000000e+00	0.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	0.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	0.00000000e+00
-1.00000000e+00	-1.00000000e+00	0.00000000e+00	-1.00000000e+00
-1.00000000e+00	0.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	2.54401095e+07	0.00000000e+00
0.00000000e+00	2.52974860e+07	1.51811215e+07	0.00000000e+00
0.00000000e+00	0.00000000e+00	-1.00000000e+00	-1.00000000e+00
0.00000000e+00	0.00000000e+00	0.00000000e+00	1.00000000e+00
0.00000000e+00	-1.00000000e+00	0.00000000e+00	5.00000000e+01
0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
1.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	0.00000000e+00
0.00000000e+00	0.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	3.46575000e+04	1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	1.00000000e+00
1.00000000e+00	1.00000000e+00	1.52019390e+07	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	0.00000000e+00	-1.00000000e+00	-1.00000000e+00
-1.00000000e+00	-1.00000000e+00	-1.00000000e+00	5.00000000e+01
5.00000000e+01	5.00000000e+01	5.00000000e+01	1.00000000e+00
2.00000000e+00	2.00000000e+00	2.00000000e+00	2.00000000e+00
2.00000000e+00	2.00000000e+00	2.00000000e+00	0.00000000e+00
0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
0.00000000e+00	0.00000000e+00	-1.00000000e+00	-1.00000000e+00

```
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00  2.00000000e+00
 -1.00000000e+00 -1.00000000e+00  2.00000000e+01  0.00000000e+00
  0.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
  2.00000000e+00  4.00000000e+01  0.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00  5.00000000e+01
  4.00000000e+01  1.00000000e+00 -1.00000000e+00 -1.00000000e+00
  1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
  0.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00  0.00000000e+00  1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
  1.05000540e+07  1.06731330e+07  1.00000000e+00  1.00000000e+00
  1.00000000e+00  1.00000000e+00  1.00000000e+00  0.00000000e+00
 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
  1.00000000e+00  0.00000000e+00 -1.00000000e+00  1.00000000e+00
  1.00000000e+00  0.00000000e+00 -6.00000000e+00 -1.00000000e+00
 -1.00000000e+00 -1.00000000e+00  1.00000000e+00  0.00000000e+00
 -6.00000000e+00]
```

1.0.10 Calculate Mean

[10]:
```
mean_array = np.mean(data, axis=0)
print(mean_array)
```

```
HRHHID      3.515847e+14
HRMONTH     1.200000e+01
HRYEAR4     2.019000e+03
HURESPLI    1.003994e+00
HUFINAL     1.899681e+02
                 …
HRFS30D6   -7.020667e-01
HRFS30D7    7.333648e+00
HRFS30D8    2.590167e-01
HRFS30D9   -2.386589e-01
```

```
HRFS30DE    2.046139e+01
Length: 509, dtype: float64
```

1.0.11 Calculate Skewness

[11]:
```
data.skew()
```

[11]:
```
HRHHID      0.587272
HRMONTH     0.000000
HRYEAR4     0.000000
HURESPLI    0.567900
HUFINAL    -3.180268
               …
HRFS30D6    0.864454
HRFS30D7    8.675021
HRFS30D8   -0.359773
HRFS30D9    3.460463
HRFS30DE    5.747845
Length: 509, dtype: float64
```

1.0.12 Calculate Kurtosis

[12]:
```
kurt = data.kurt(axis=0)
print(kurt)
```

```
HRHHID      -0.929609
HRMONTH      0.000000
HRYEAR4      0.000000
HURESPLI     7.500876
HUFINAL      8.514782
               …
HRFS30D6    40.945293
HRFS30D7    82.038154
HRFS30D8     4.280342
HRFS30D9    27.293055
HRFS30DE    35.975904
Length: 509, dtype: float64
```

1.0.13 Calculate Variance

[13]:
```
np.var(data, axis = 0)
```

[13]:
```
HRHHID      8.561955e+28
HRMONTH     0.000000e+00
HRYEAR4     0.000000e+00
HURESPLI    9.309950e-01
HUFINAL     2.818842e+03
                …
```

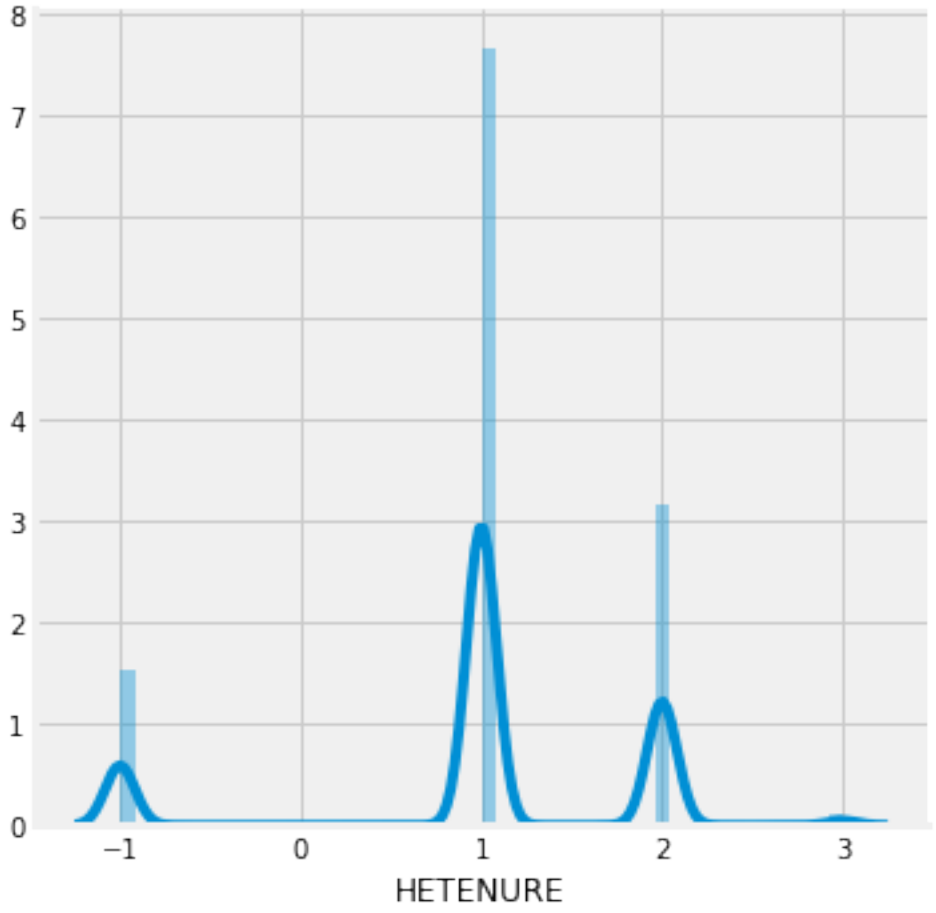
```
HRFS30D6    3.851571e-01
HRFS30D7    5.784025e+03
HRFS30D8    1.435156e+00
HRFS30D9    1.425447e+00
HRFS30DE    1.372634e+04
Length: 509, dtype: float64
```

1.0.14 Create Univariate Distribution Plots

[14]:
```
sns.FacetGrid(data,height=5).map(sns.distplot,"HETENURE").add_legend()
```

```
C:\Users\Richard\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

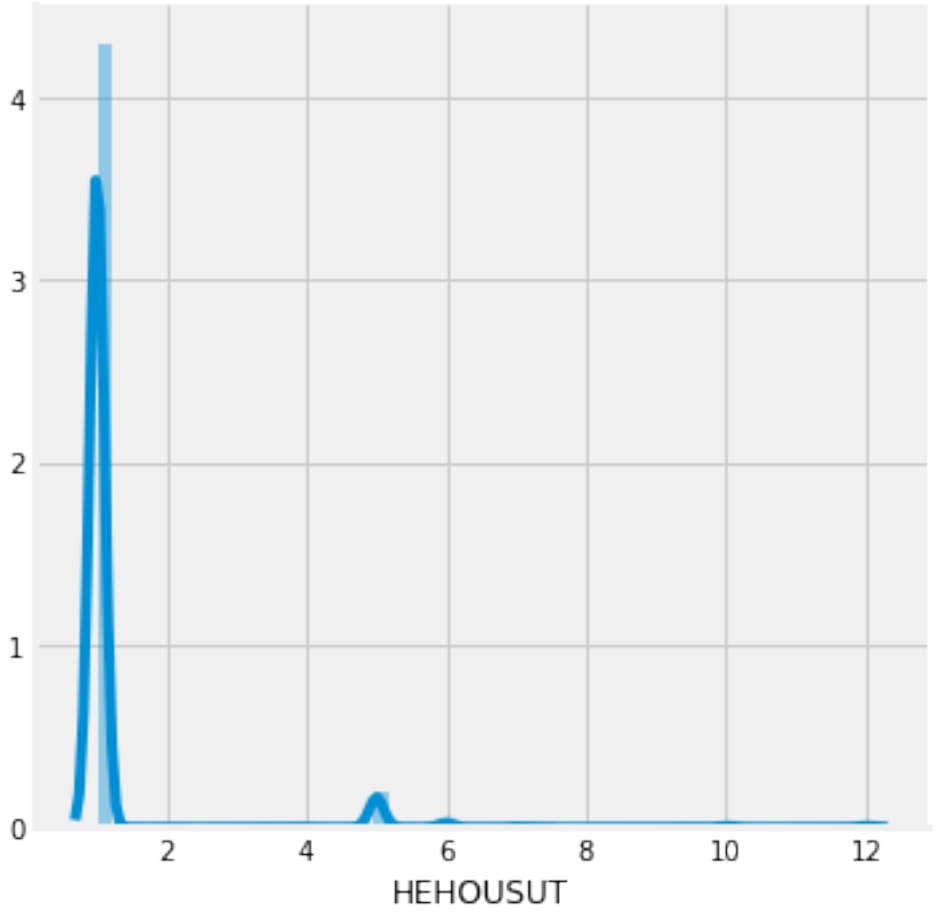
[14]:
```
<seaborn.axisgrid.FacetGrid at 0x24ba569c820>
```



[15]:
```
sns.FacetGrid(data,height=5).map(sns.distplot,"HEHOUSUT").add_legend()
```

```
C:\Users\Richard\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[15]:
```
<seaborn.axisgrid.FacetGrid at 0x24ba571c040>
```

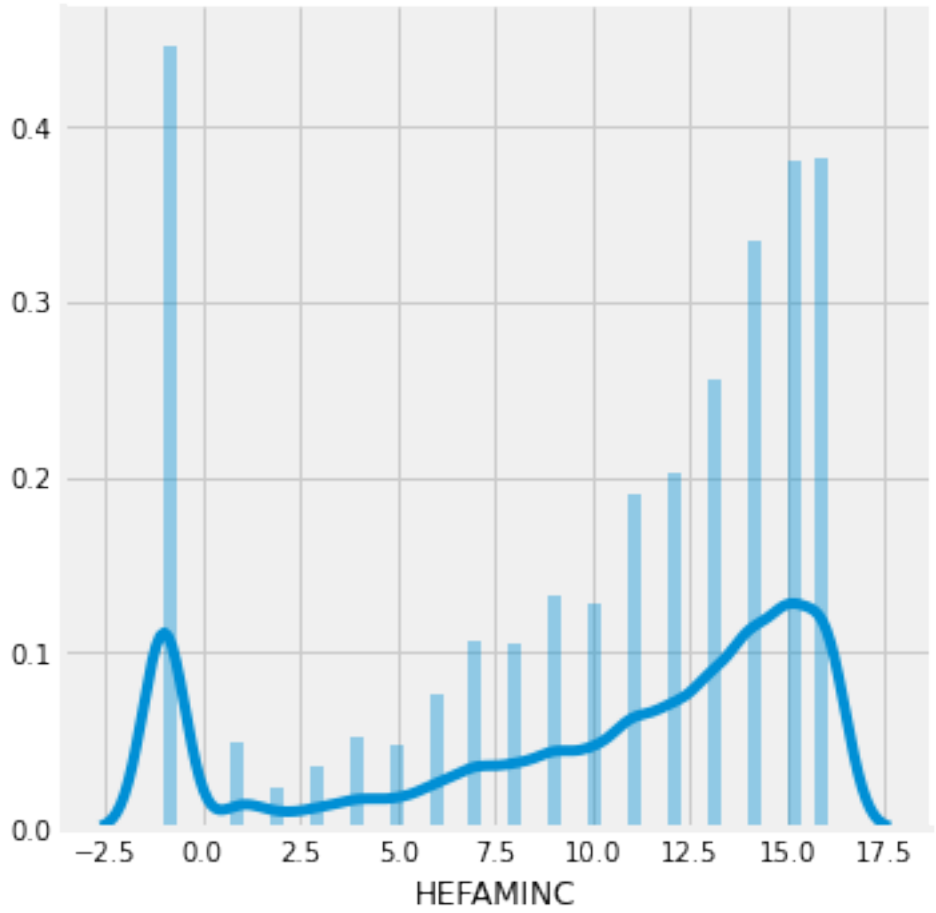


[16]:
```
sns.FacetGrid(data,height=5).map(sns.distplot,"HEFAMINC").add_legend()
```

```
C:\Users\Richard\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
```

```
histograms).
  warnings.warn(msg, FutureWarning)
```

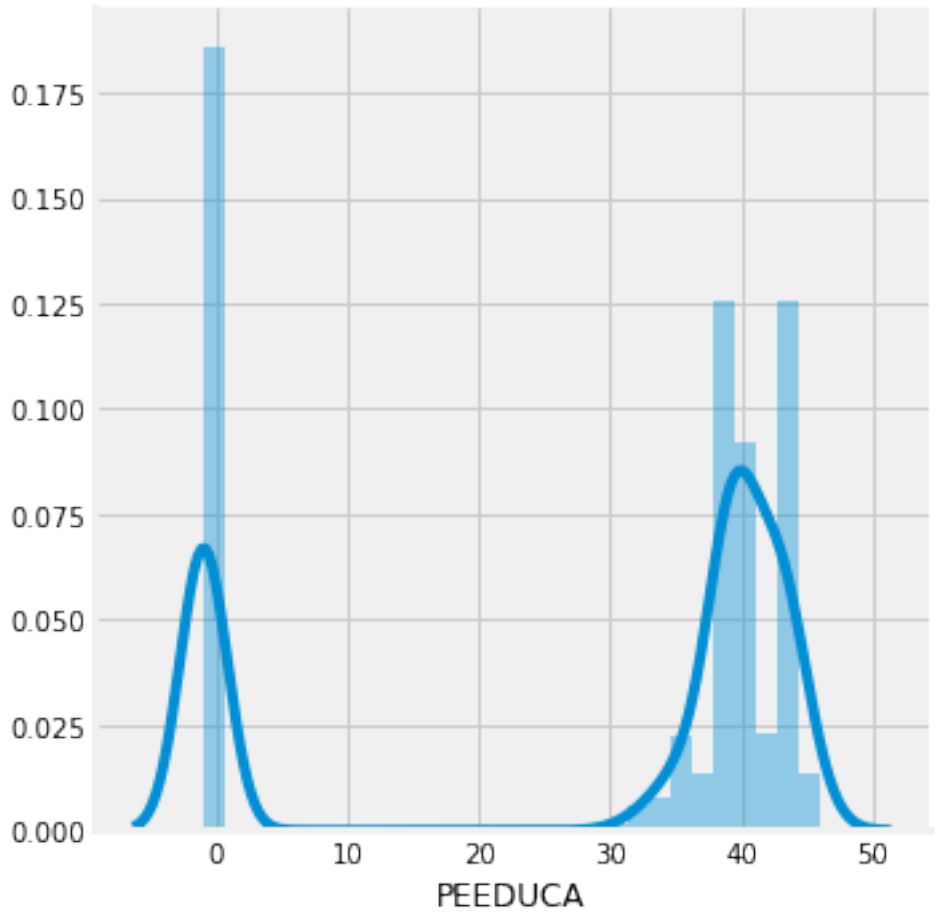
[16]: <seaborn.axisgrid.FacetGrid at 0x24ba57793d0>



[17]:
```
sns.FacetGrid(data,height=5).map(sns.distplot,"PEEDUCA").add_legend()
```

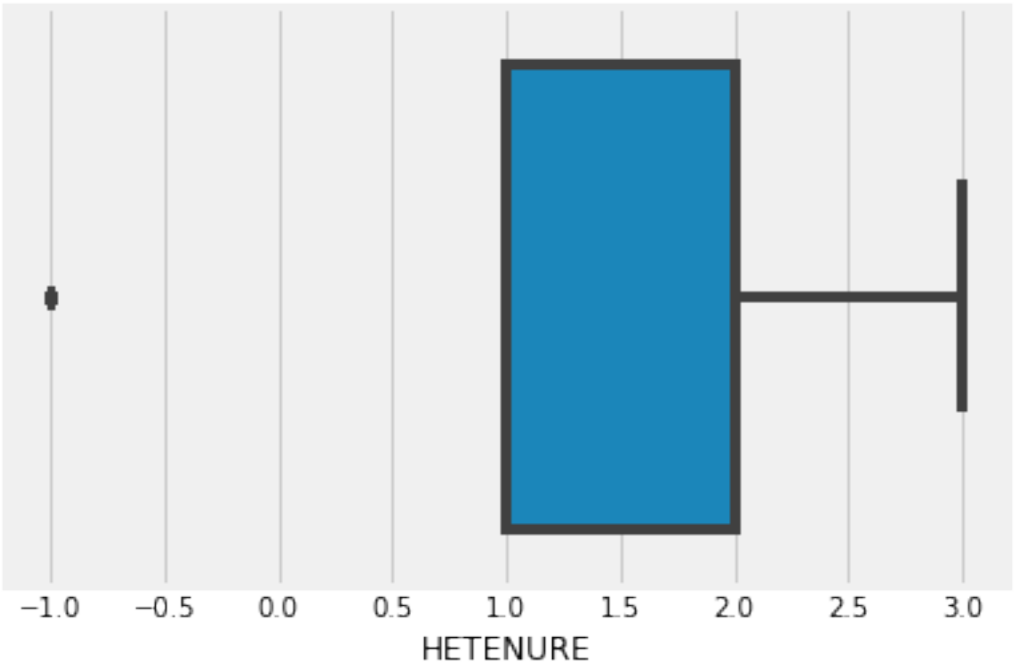
```
C:\Users\Richard\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[17]: <seaborn.axisgrid.FacetGrid at 0x24ba57a5940>

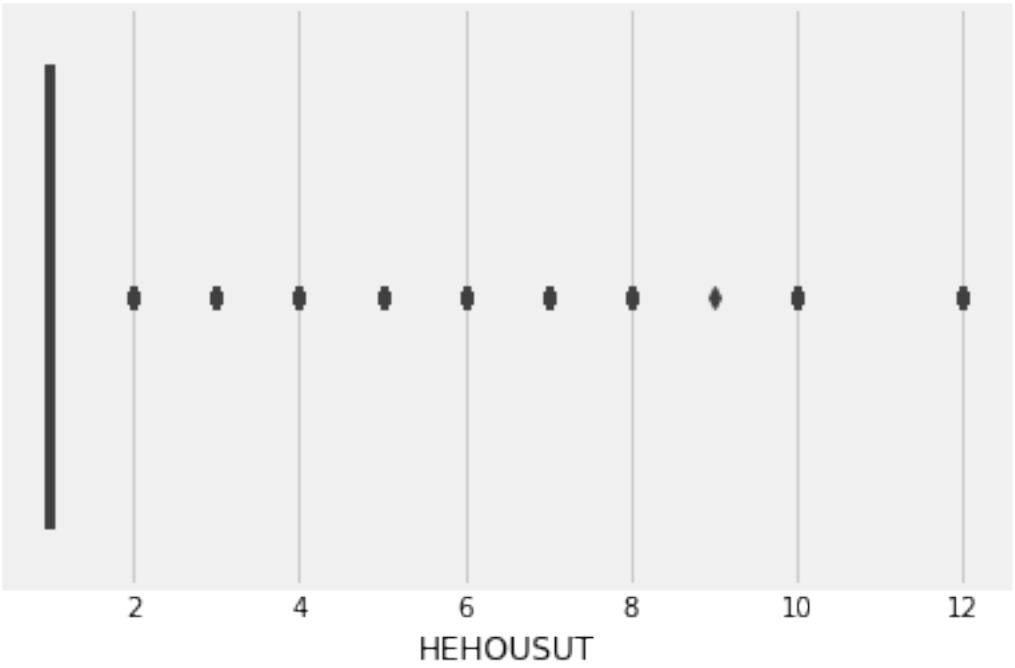


1.0.15 Boxplots

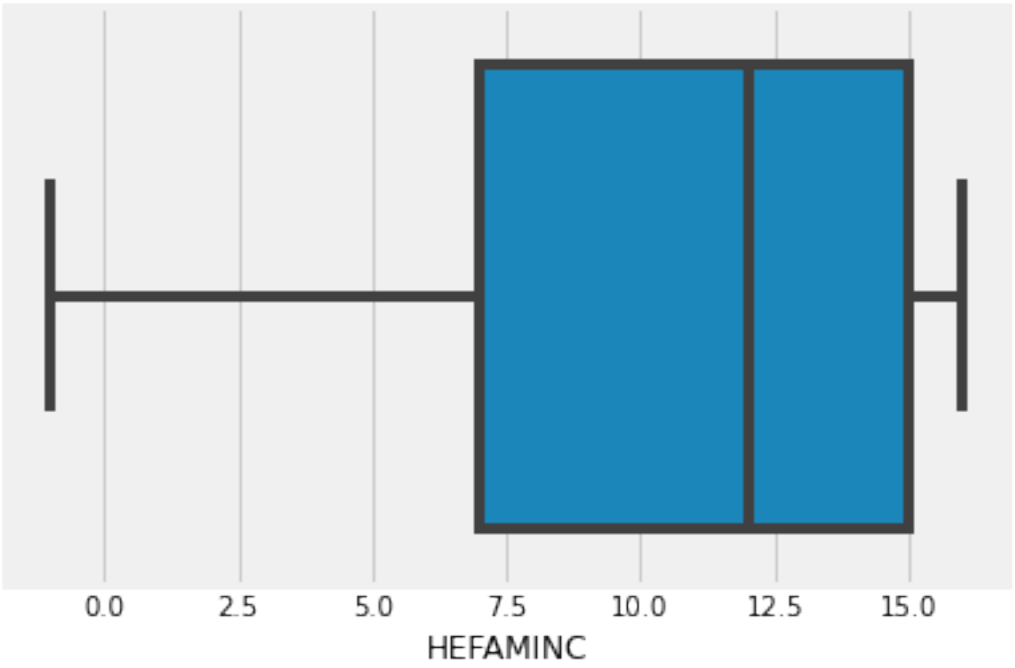
[18]:
```
sns.boxplot(x="HETENURE",data=data)
plt.show()
```



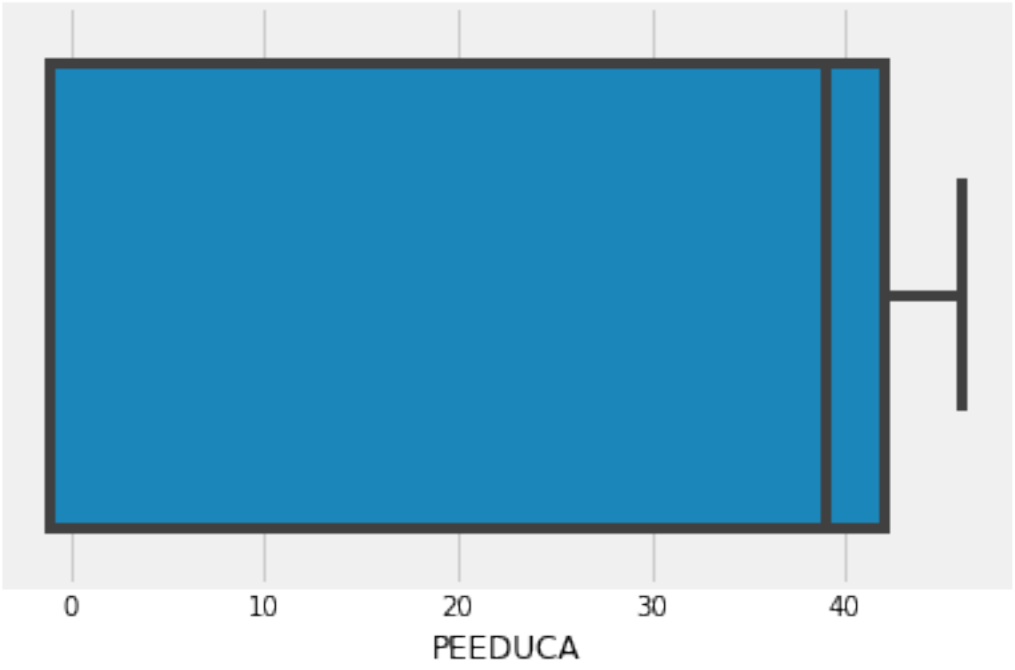
[19]:
```
sns.boxplot(x="HEHOUSUT",data=data)
plt.show()
```



[20]:
```
sns.boxplot(x="HEFAMINC",data=data)
plt.show()
```



[21]:
```
sns.boxplot(x="PEEDUCA",data=data)
plt.show()
```



1.1 Hypothesis Testing - Parametric Tests

1.1.1 Variables for Hypothesis Testing

[22]:
```
data_HETENURE = data["HETENURE"]
data_HEHOUSUT = data["HEHOUSUT"]
data_HEFAMINC = data["HEFAMINC"]
data_PEEDUCA = data["PEEDUCA"]
```

1.1.2 One-Sample T-Test

[23]:
```
from scipy.stats import ttest_1samp
```

[24]:
```
t_test_value, p_value = ttest_1samp(data_HEFAMINC, 10)

print("P Value:",p_value)

print("t-test Value:",t_test_value)

# 0.05 or 5% is significance level or alpha.
if p_value < 0.05:

    print("Hypothesis Rejected")

else:

    print("Hypothesis Accepted")
```

```
P Value: 2.3373812544089413e-120
t-test Value: 23.350303566529703
Hypothesis Rejected
```

1.1.3 Two-Sample T-Test

[25]:
```
from scipy.stats import ttest_ind
```

[26]:
```
# Compare samples

stat, p = ttest_ind(data_HETENURE, data_HEHOUSUT)

print("p-values:",p)

print("t-test:",stat)

# 0.05 or 5% is significance level or alpha.
```

```
if p < 0.05:

    print("Hypothesis Rejected")

else:

    print("Hypothesis Accepted")
```

```
p-values: 0.0
t-test: -59.1592155017553
Hypothesis Rejected
```

1.1.4 Paired Sample T-Test

[27]:
```
# paired test
from scipy.stats import ttest_rel
```

[28]:
```
# Compare weights

stat, p = ttest_rel(data_HEFAMINC, data_HEHOUSUT)

print("p-values:",p)

print("t-test:",stat)

# 0.05 or 5% is the significance level or alpha.

if p < 0.05:

    print("Hypothesis Rejected")

else:

    print("Hypothesis Accepted")
```

```
p-values: 0.0
t-test: 529.8501854709342
Hypothesis Rejected
```

1.2 Hypothesis Testing - Non-Parametric Tests

1.2.1 Mann-Whitney U Test

[29]:
```
from scipy.stats import mannwhitneyu
```

[30]:
```
# Apply Test

stat, p = mannwhitneyu(data_HEFAMINC, data_PEEDUCA)

print("p-values:",p)

# 0.01 or 1% is significance level or alpha.

if p < 0.01:

    print("Hypothesis Rejected")

else:
    print("Hypothesis Accepted")
```

```
p-values: 0.0
Hypothesis Rejected
```

Week 8 - Eigenvalues and Eigenvectors

September 28, 2022

1 TIM8501 Week 8 - Eigenvalues and Eigenvectors

1.0.1 Import Libraries

[1]:
```
# Import pandas as pd
import pandas as pd

# Import numpy
import numpy as np

# Import linear algebra module
from scipy import linalg as la
```

1.0.2 Import Dataset

[2]:
```
data = pd.read_csv("dec19pub.csv")
```

[4]:
```
data.dtypes
```

[4]:
```
HRHHID      float64
HRMONTH       int64
HRYEAR4       int64
HURESPLI      int64
HUFINAL       int64
              ...
HRFS30D6      int64
HRFS30D7      int64
HRFS30D8      int64
HRFS30D9      int64
HRFS30DE      int64
Length: 509, dtype: object
```

[5]:
```
# Calculate the covariance matrix
# Center your data

data -= data.mean(axis=0)
```

[6]:
```
cov = np.cov(data, rowvar=False)
```

[7]:
```
# Calculate eigenvalues and eigenvector of the covariance matrix
evals, evecs = la.eig(cov)
```

[8]:
```
# Multiply the original data matrix with Eigenvector matrix.

# Sort the Eigen values and vector and select components
num_components=2
sorted_key = np.argsort(evals)[::-1][:num_components]
evals, evecs = evals[sorted_key], evecs[:, sorted_key]

print("Eigenvalues:", evals)
print("Eigenvector:", evecs)
print("Sorted and Selected Eigen Values:", evals)
print("Sorted and Selected Eigen Vector:", evecs)

# Multiply original data and Eigen vector
principal_components=np.dot(data,evecs)
print("Principal Components:", principal_components)
```

```
Eigenvalues: [8.56201654e+28+0.j 3.06276760e+15+0.j]
Eigenvector: [[-1.00000000e+00  2.42301200e-09]
 [ 0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00]
 …
 [-4.63353306e-18 -6.99296831e-09]
 [ 4.65579345e-18 -4.19640478e-09]
 [ 3.10240206e-16 -1.10624942e-07]]
Sorted and Selected Eigen Values: [8.56201654e+28+0.j 3.06276760e+15+0.j]
Sorted and Selected Eigen Vector: [[-1.00000000e+00  2.42301200e-09]
 [ 0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00]
 …
 [-4.63353306e-18 -6.99296831e-09]
 [ 4.65579345e-18 -4.19640478e-09]
 [ 3.10240206e-16 -1.10624942e-07]]
Principal Components: [[-2.29540307e+14  4.76128973e+07]
 [-2.29540307e+14  4.81452846e+07]
 [-2.29540307e+14  4.64288926e+07]
 …
 [ 3.41575993e+14  5.56144222e+07]
 [-1.57726307e+14  5.68242376e+07]
 [-5.84243068e+13  5.65836277e+07]]
```

[9]:
```
# Import principal component analysis
from sklearn.decomposition import PCA
```

[10]:
```
# Create and fit_transformed PCA Model
pca_model = PCA(n_components=2)
components = pca_model.fit_transform(data)
components_df = pd.DataFrame(data = components,

columns = ['principal_component_1', 'principal_component_2'])
print(components_df)
```

```
        principal_component_1  principal_component_2
0                2.295403e+14          -4.761290e+07
1                2.295403e+14          -4.814528e+07
2                2.295403e+14          -4.642889e+07
3                2.295403e+14          -5.107007e+07
4                4.574173e+14           2.668156e+07
…                         …                     …
138959           3.145863e+14          -5.720431e+07
138960          -1.725197e+14          -5.602405e+07
138961          -3.415760e+14          -5.561442e+07
138962           1.577263e+14          -5.682424e+07
138963           5.842431e+13          -5.658363e+07

[138964 rows x 2 columns]
```

Week 8-Cluster Analysis

September 28, 2022

1 TIM8501 Week 6 - Cluster Analysis

1.0.1 Import Libraries

[1]:
```
# Import pandas as pd
import pandas as pd

# Import numpy
import numpy as np

# Import linear algebra module
from scipy import linalg as la
```

1.0.2 Import Dataset

[2]:
```
data = pd.read_csv("dec19pub.csv")
```

[3]:
```
data.dtypes
```

[3]:
```
HRHHID      float64
HRMONTH       int64
HRYEAR4       int64
HURESPLI      int64
HUFINAL       int64
              ...
HRFS30D6      int64
HRFS30D7      int64
HRFS30D8      int64
HRFS30D9      int64
HRFS30DE      int64
Length: 509, dtype: object
```

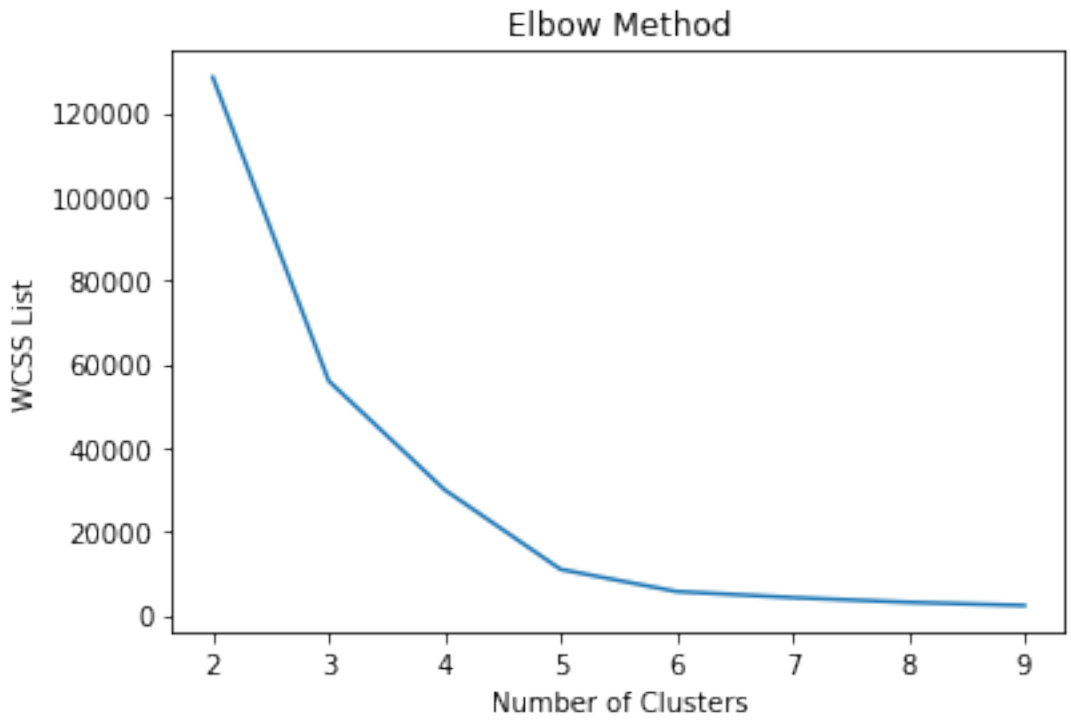
[4]:
```
# import matplotlib
import matplotlib.pyplot as plt

# import K-means
from sklearn.cluster import KMeans
```

[5]:
```
X = data.iloc[:, [6, 7]].values
```

[6]:
```
print(X)
```

```
[[ 1  1]
 [ 1  1]
 [ 1  1]
 …
 [ 2  1]
 [-1  1]
 [-1  1]]
```

[7]:
```
# Calculate optimal amount of clusters with the elbow method
# Instantiate a (WCSS) Within Cluster Sum of Squares list
wcss = []

# Utilize a for loop to append values to WCSS list by iterating through kmeans␣
 ↪datapoints
for i in range(2, 10):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Scree plot showing the optimal number of clusters
plt.plot(range(2, 10), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS List')
plt.show()
```



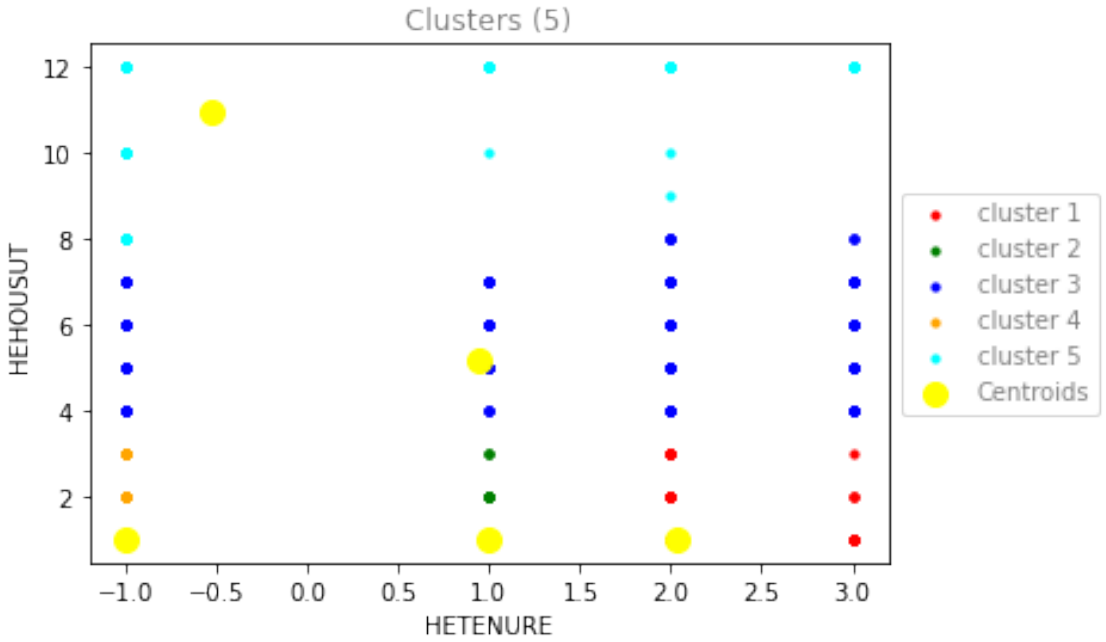
[8]:
```
# Train the K-means model
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)

# Dependent variable utilized to split customers into different clusters
y_kmeans = kmeans.fit_predict(X)
```

[9]:
```
# Create Scatter plot using 5 clusters for HETENURE v. HEHOUSUT
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 10, c = 'red', label␣
 ↪= 'cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 10, c = 'green',␣
 ↪label = 'cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 10, c = 'blue', label␣
 ↪= 'cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 10, c = 'orange',␣
 ↪label = 'cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 10, c = 'cyan', label␣
 ↪= 'cluster 5')

# Plot centroids for each cluster
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =␣
 ↪100, c = 'yellow', label = 'Centroids')

# Plot Labels
```

```
title_obj = plt.title('Clusters (5)')
plt.getp(title_obj)
plt.getp(title_obj, 'text')
plt.setp(title_obj, color='gray')

plt.xlabel('HETENURE')
plt.ylabel('HEHOUSUT')

legend = plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.setp(legend.get_texts(), color='gray')

# Display Plot
plt.show();
```

```
    agg_filter = None
    alpha = None
    animated = False
    bbox_patch = None
    children = []
    clip_box = None
    clip_on = True
    clip_path = None
    color or c = black
    contains = None
    figure = Figure(432x288)
    fontfamily or family = ['sans-serif']
    fontname or name = DejaVu Sans
    fontproperties or font or font_properties =
sans\-serif:style=normal:variant=normal:weight=nor…
    fontsize or size = 12.0
    fontstyle or style = normal
    fontvariant or variant = normal
    fontweight or weight = normal
    gid = None
    horizontalalignment or ha = center
    in_layout = True
    label =
    path_effects = []
    picker = None
    position = (0.5, 1.0)
    rasterized = None
    rotation = 0.0
    rotation_mode = None
    sketch_params = None
    snap = None
    stretch = normal
    text = Clusters (5)
```

```
transform = CompositeGenericTransform(     BboxTransformTo(    …
transformed_clip_path_and_affine = (None, None)
unitless_position = (0.5, 1.0)
url = None
usetex = False
verticalalignment or va = baseline
visible = True
wrap = False
zorder = 3
```



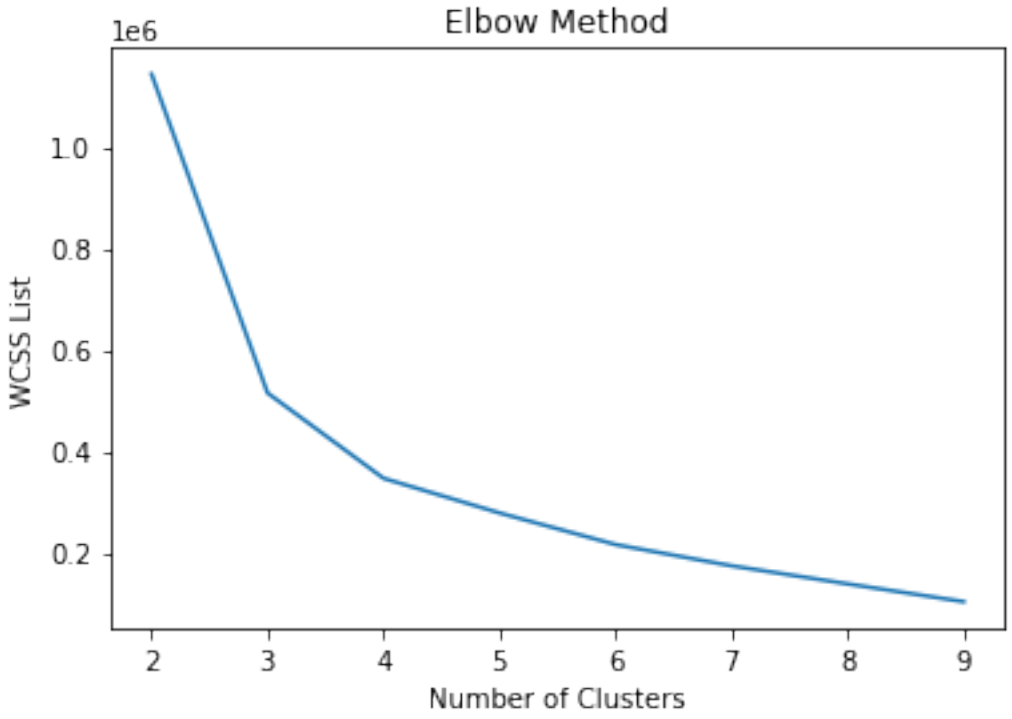
[10]:
```
X = data.iloc[:, [11, 7]].values
```

[11]:
```
print(X)
```

```
[[14  1]
 [14  1]
 [14  1]
 …
 [-1  1]
 [-1  1]
 [-1  1]]
```

[12]:
```
# Calculate optimal amount of clusters with the elbow method
# Instantiate a (WCSS) Within Cluster Sum of Squares list
wcss = []
```

```
# Utilize a for loop to append values to WCSS list by iterating through kmeans␣
 ↪datapoints
for i in range(2, 10):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Scree plot showing the optimal number of clusters
plt.plot(range(2, 10), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS List')
plt.show()
```



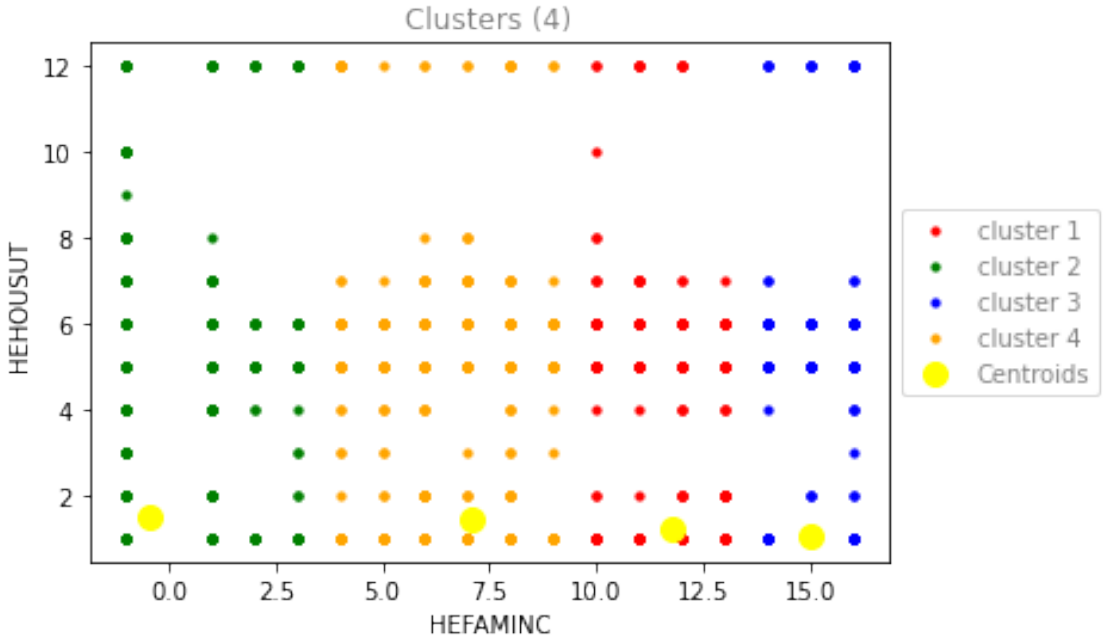
[13]:
```
# Train the K-means model
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=42)

# Dependent variable utilized to split customers into different clusters
y_kmeans = kmeans.fit_predict(X)
```

[14]:
```
# Create Scatter plot using 4 clusters for HEFAMINC v. HEHOUSUT
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 10, c = 'red', label␣
 ↪= 'cluster 1')
```

```
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 10, c = 'green',␣
↪label = 'cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 10, c = 'blue', label␣
↪= 'cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 10, c = 'orange',␣
↪label = 'cluster 4')

# Plot centroids for each cluster
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =␣
↪100, c = 'yellow', label = 'Centroids')

# Plot Labels
title_obj = plt.title('Clusters (4)')
plt.getp(title_obj)
plt.getp(title_obj, 'text')
plt.setp(title_obj, color='gray')

plt.xlabel('HEFAMINC')
plt.ylabel('HEHOUSUT')

legend = plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.setp(legend.get_texts(), color='gray')

# Display Plot
plt.show();
```

```
    agg_filter = None
    alpha = None
    animated = False
    bbox_patch = None
    children = []
    clip_box = None
    clip_on = True
    clip_path = None
    color or c = black
    contains = None
    figure = Figure(432x288)
    fontfamily or family = ['sans-serif']
    fontname or name = DejaVu Sans
    fontproperties or font or font_properties =
sans\-serif:style=normal:variant=normal:weight=nor…
    fontsize or size = 12.0
    fontstyle or style = normal
    fontvariant or variant = normal
    fontweight or weight = normal
    gid = None
    horizontalalignment or ha = center
```

```
in_layout = True
label =
path_effects = []
picker = None
position = (0.5, 1.0)
rasterized = None
rotation = 0.0
rotation_mode = None
sketch_params = None
snap = None
stretch = normal
text = Clusters (4)
transform = CompositeGenericTransform(     BboxTransformTo(    …
transformed_clip_path_and_affine = (None, None)
unitless_position = (0.5, 1.0)
url = None
usetex = False
verticalalignment or va = baseline
visible = True
wrap = False
zorder = 3
```

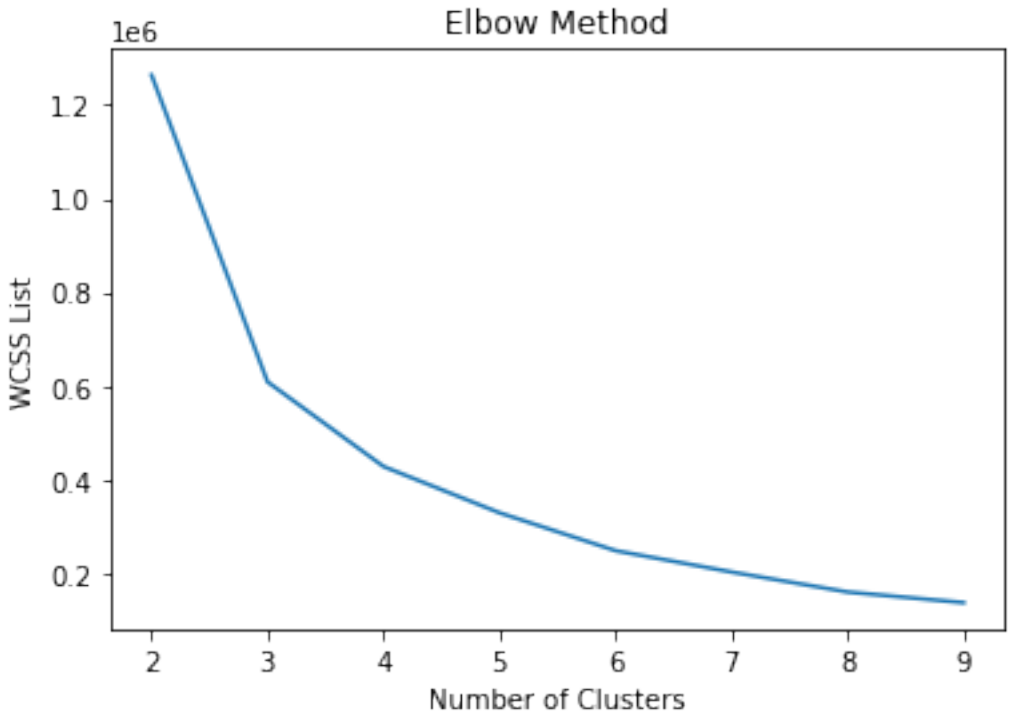


[15]:
```
X = data.iloc[:, [11, 50]].values
```

[16]:
```
print(X)
```

```
[[14  1]
 [14  1]
 [14  1]
 …
 [-1 -1]
 [-1 -1]
 [-1 -1]]
```

[17]:
```
# Calculate optimal amount of clusters with the elbow method
# Instantiate a (WCSS) Within Cluster Sum of Squares list
wcss = []

# Utilize a for loop to append values to WCSS list by iterating through kmeans␣
 ↪datapoints
for i in range(2, 10):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Scree plot showing the optimal number of clusters
plt.plot(range(2, 10), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS List')
plt.show()
```

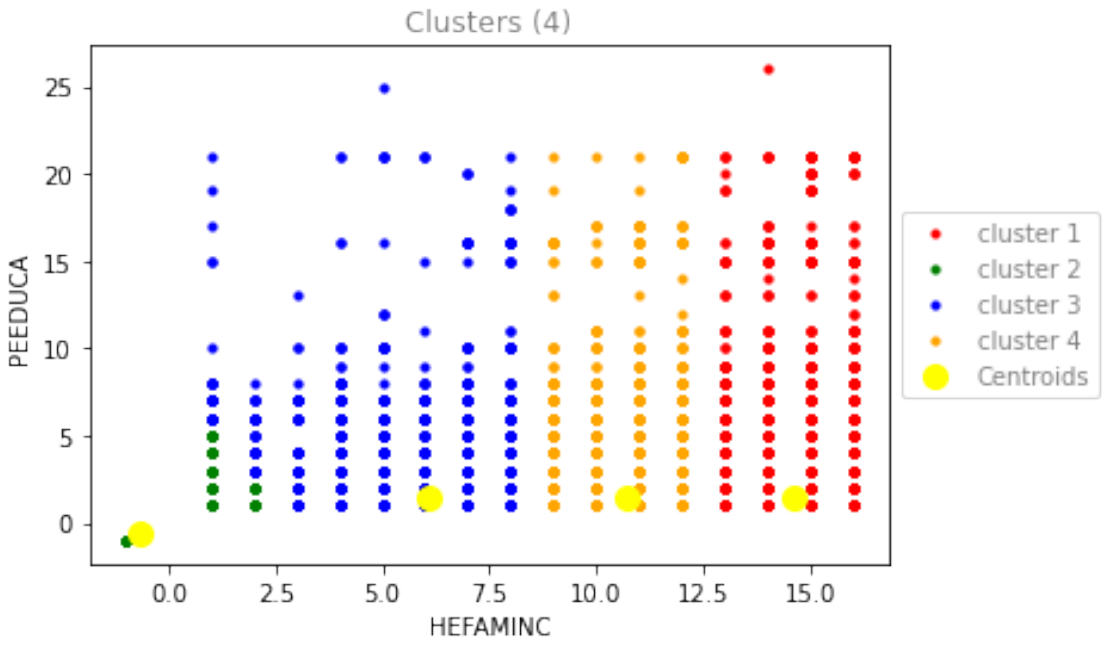


[18]:
```
# Train the K-means model
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=42)

# Dependent variable utilized to split customers into different clusters
y_kmeans = kmeans.fit_predict(X)
```

[19]:
```
# Create Scatter plot using 4 clusters for HEFAMINC v. PEEDUCA
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 10, c = 'red', label␣
↪= 'cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 10, c = 'green',␣
↪label = 'cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 10, c = 'blue', label␣
↪= 'cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 10, c = 'orange',␣
↪label = 'cluster 4')

# Plot centroids for each cluster
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =␣
↪100, c = 'yellow', label = 'Centroids')

# Plot Labels
title_obj = plt.title('Clusters (4)')
plt.getp(title_obj)
plt.getp(title_obj, 'text')
plt.setp(title_obj, color='gray')

plt.xlabel('HEFAMINC')
plt.ylabel('PEEDUCA')

legend = plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.setp(legend.get_texts(), color='gray')

# Display Plot
plt.show();
```

```
    agg_filter = None
    alpha = None
    animated = False
    bbox_patch = None
    children = []
    clip_box = None
    clip_on = True
    clip_path = None
    color or c = black
    contains = None
    figure = Figure(432x288)
```

```
    fontfamily or family = ['sans-serif']
    fontname or name = DejaVu Sans
    fontproperties or font or font_properties =
sans\-serif:style=normal:variant=normal:weight=nor…
    fontsize or size = 12.0
    fontstyle or style = normal
    fontvariant or variant = normal
    fontweight or weight = normal
    gid = None
    horizontalalignment or ha = center
    in_layout = True
    label =
    path_effects = []
    picker = None
    position = (0.5, 1.0)
    rasterized = None
    rotation = 0.0
    rotation_mode = None
    sketch_params = None
    snap = None
    stretch = normal
    text = Clusters (4)
    transform = CompositeGenericTransform(     BboxTransformTo(   …
    transformed_clip_path_and_affine = (None, None)
    unitless_position = (0.5, 1.0)
    url = None
    usetex = False
    verticalalignment or va = baseline
    visible = True
    wrap = False
    zorder = 3
```


Clusters (4)
PEEDUCA
HEFAMINC
0
5
10
15
20
25
0.0
2.5
5.0
7.5
10.0
12.5
15.0
cluster 1
cluster 2
cluster 3
cluster 4
Centroids