Scenario 4 - DATA 6310

- C2.S4.Py01 Digging deeper in the normalized results
- C2.S4.Py02 Creating a dendrogram
- C2.S4.Py03 Hierarchical cluster analysis
- C2.S4.Py04 Hierarchical cluster analysis with Ward linkage and comparisons to other clusters
- C2.S4.Py05 Hierarchical cluster analysis with larger sample
- C2.S4.Py06 Normalize the data and run hierarchical cluster analysis with larger sample
- C2.S4.Py07 K-Means clustering with the original dataset
- C2.S4.Py08 Hierarchical clustering with the original dataset
- C2.S4.Py09 Comparing the cluster results

BUSINESS UNDERSTANDING

Business Objective

- Can we find specific groups within the data to get a better idea of the data?
- Do we know the number of clusters?
  - If **NO** then use hiearchal clustering
  - If **YES** then use K-Means clustering
- This is an unsupervised learning technique

Technical Objective

- Create a dataset that can be used for clustering
- Normalize the data to see if it provides better results
- Determine if you would like to use a set number of groups
- Use hierarchal clustering to get a good idea of the number of clusters
- Use K-means clustering if you know the number of clusters expected (many times a business decision)

- https://www.geeksforgeeks.org/implementing-agglomerative-clustering-using-sklearn/

Digging deeper in the normalized results

In [1]:
```
#Code Block 1
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

pd.set_option('display.max_columns',500) #allows for up to 500 columns to be displayed when viewing a dat



#if you want graphs to automatically without plt.show
plt.style.use('seaborn-colorblind') #a style that can be used for plots - see style reference above

%matplotlib inline
```

In [2]:
```
%%time
```

```
#Code Block 2
url = 'https://data6300.file.core.windows.net/data6300/Scenario4.csv?st=2020-09-30T22%3A22%3A07Z&se=2022-1
url2 = 'https://data6300.file.core.windows.net/data6300/Scenario4_melt.csv?st=2020-09-30T22%3A22%3A26Z&se=

df_ap = pd.read_csv(url, index_col=0, header=0)
df_ap_melt = pd.read_csv(url2, index_col=0, header=0)
df_ap.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8717 entries, 0 to 8716
Data columns (total 32 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   APID                            8717 non-null   object
 1   Watch_Cooking_Shows             8717 non-null   float64
 2   Watch_Documentaries             8717 non-null   float64
 3   Watch_Drama_Shows               8717 non-null   float64
 4   Watch_Game_Shows                8717 non-null   float64
 5   Watch_Home_Improvement_Shows    8717 non-null   float64
 6   Watch_Music_Videos              8717 non-null   float64
 7   Watch_Auto_Racing               8717 non-null   float64
 8   Watch_News_Shows                8717 non-null   float64
 9   Watch_Police_or_Detective_Shows 8717 non-null   float64
 10  Watch_Reality_Television_Shows  8717 non-null   float64
 11  Watch_Religious_Shows           8717 non-null   float64
 12  Watch_Situational_Comedies      8717 non-null   float64
 13  Watch_Soap_Operas               8717 non-null   float64
 14  Watch_Sports_Shows              8717 non-null   float64
 15  Watch_Talk_Shows                8717 non-null   float64
 16  Watch_Wrestling_Shows           8717 non-null   float64
 17  Income_Dollars                  8717 non-null   float64
 18  HomeOwner_Renter                8717 non-null   object
 19  Marital_Status                  8717 non-null   object
 20  Age                             8717 non-null   int64
 21  Adults_in_Household             8717 non-null   int64
 22  Household_Size                  8717 non-null   int64
 23  Discretionary_Spending_Dollars  8717 non-null   float64
 24  PolicyNumber                    2417 non-null   object
 25  DriverNumber                    2417 non-null   float64
 26  DriverCount                     2417 non-null   float64
 27  Policy                          8717 non-null   int64
 28  Predict                         8717 non-null   int64
 29  Predict_n                       8717 non-null   int64
 30  Predict_4                       8717 non-null   int64
 31  Predict_3n                      8717 non-null   int64
dtypes: float64(20), int64(8), object(4)
memory usage: 2.2+ MB
Wall time: 2.74 s
```

In [3]:
```
#Code Block 3
round(df_ap.groupby('Predict_4').mean().T, 2)
```

Out[3]:

Predict_4	0	1	2	3
Watch_Cooking_Shows	15.61	6.46	8.15	11.39
Watch_Documentaries	11.85	7.62	11.76	10.98
Watch_Drama_Shows	16.15	7.65	10.56	9.24
Watch_Game_Shows	11.66	6.37	11.58	8.87
Watch_Home_Improvement_Shows	12.21	13.17	11.18	7.86
Watch_Music_Videos	7.64	4.23	7.24	13.56
Watch_Auto_Racing	8.69	15.48	8.10	6.57
Watch_News_Shows	14.79	14.63	14.54	5.33
Watch_Police_or_Detective_Shows	15.01	14.03	14.30	4.77
Watch_Reality_Television_Shows	10.53	4.25	4.76	14.01
Watch_Religious_Shows	11.19	7.03	11.12	8.14
Watch_Situational_Comedies	11.80	9.62	12.08	11.34

Predict_4	0	1	2	3
Watch_Soap_Operas	14.03	6.00	6.50	8.84
Watch_Sports_Shows	8.28	14.79	16.46	6.55
Watch_Talk_Shows	11.71	5.44	13.11	12.55
Watch_Wrestling_Shows	9.16	5.53	8.63	11.19
Income_Dollars	61282.15	49337.95	56095.26	60556.51
Age	42.58	40.02	42.63	60.51
Adults_in_Household	1.88	1.56	1.54	2.73
Householod_Size	2.40	1.95	2.01	3.50
Discretionary_Spending_Dollars	9515.99	7665.28	8779.89	7519.07
DriverNumber	1.03	1.05	1.04	1.02
DriverCount	2.20	2.15	2.13	2.27
Policy	0.29	0.26	0.28	0.27
Predict	1.46	1.57	2.97	1.90
Predict_n	1.73	2.41	2.17	0.57
Predict_3n	1.33	1.32	1.35	1.17

In [4]:
```
#Code Block 4
df_ap['Predict_4'].value_counts()
```

Out[4]:
```
0    2605
1    2240
2    2005
3    1867
Name: Predict_4, dtype: int64
```

In [5]:
```
#Code Block 5
round(df_ap.groupby('Predict_n').mean().T, 2)
```

Out[5]:

Predict_n	0	1	2	3	4
Watch_Cooking_Shows	10.59	13.65	10.54	11.58	9.49
Watch_Documentaries	10.73	11.63	10.33	10.89	9.99
Watch_Drama_Shows	10.63	12.71	12.06	12.28	10.84
Watch_Game_Shows	10.19	10.27	9.27	9.52	9.03
Watch_Home_Improvement_Shows	9.99	9.50	13.39	12.07	12.61
Watch_Music_Videos	9.39	12.40	5.09	8.77	5.57
Watch_Auto_Racing	9.68	5.68	10.77	9.33	11.32
Watch_News_Shows	11.16	7.82	15.80	11.74	15.15
Watch_Police_or_Detective_Shows	10.85	7.91	15.61	11.85	14.72
Watch_Reality_Television_Shows	9.19	13.74	6.52	10.64	5.77
Watch_Religious_Shows	9.80	9.20	9.18	9.31	9.25
Watch_Situational_Comedies	10.20	13.07	11.87	13.26	11.25
Watch_Soap_Operas	9.67	10.33	8.71	9.05	8.16
Watch_Sports_Shows	9.64	7.85	14.08	11.82	13.97
Watch_Talk_Shows	10.96	13.87	9.76	11.16	9.19
Watch_Wrestling_Shows	10.41	10.46	5.89	7.12	6.76

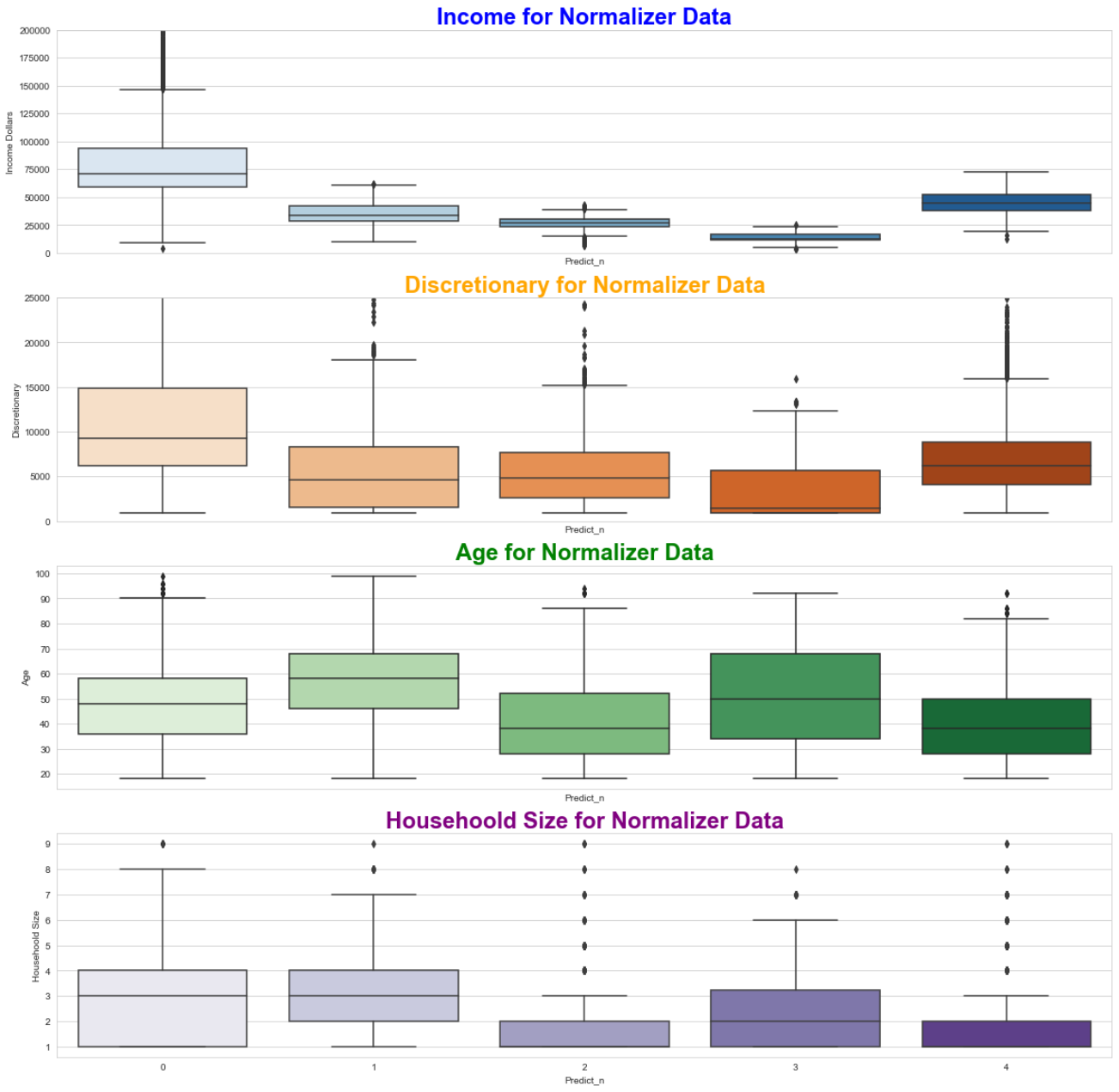
Predict_n	0	1	2	3	4
Income_Dollars	86855.26	35133.64	26721.57	14070.83	45095.82
Age	47.42	57.20	41.34	51.27	40.72
Adults_in_Household	2.13	2.37	1.56	2.18	1.56
Housheold_Size	2.79	3.04	1.93	2.64	1.94
Discretionary_Spending_Dollars	11692.52	5739.29	5571.24	3547.80	7158.13
DriverNumber	1.04	1.02	1.03	1.03	1.03
DriverCount	2.24	2.09	2.22	2.12	2.17
Policy	0.26	0.30	0.29	0.32	0.28
Predict	1.92	1.36	2.06	1.79	2.11
Predict_4	1.56	1.88	0.97	1.44	1.07
Predict_3n	1.00	1.69	1.84	0.00	1.39

In [6]:
```
#Code Block 6

df_ap['Predict_n'].value_counts()
```

Out[6]:
```
0    3420
4    2776
2    1196
1    1085
3     240
Name: Predict_n, dtype: int64
```

In [7]:
```
#Code Block 7

sns.set_style("whitegrid")
plt.figure(figsize=(20,20))

plt.subplot(411)
plt.title('Income for Normalizer Data', fontweight='bold', color = 'blue', fontsize='24', horizontalalignı
sns.boxplot(y='Income_Dollars', x='Predict_n', data=df_ap, palette='Blues')
plt.xticks([])
plt.ylabel('Income Dollars')
plt.ylim(0, 200000)


plt.subplot(412)
plt.title('Discretionary for Normalizer Data', fontweight='bold', color = 'orange', fontsize='24', horizoı
sns.boxplot(y='Discretionary_Spending_Dollars', x='Predict_n', data=df_ap, palette='Oranges')
plt.xticks([])
plt.ylabel('Discretionary')
plt.ylim(0, 25000)

plt.subplot(413)
plt.title('Age for Normalizer Data', fontweight='bold', color = 'green', fontsize='24', horizontalalignmeı
sns.boxplot(y='Age', x='Predict_n', data=df_ap, palette='Greens')
plt.xticks([])
plt.ylabel('Age')
#plt.ylim(0, 25000)

plt.subplot(414)
plt.title('Housheold Size for Normalizer Data', fontweight='bold', color = 'purple', fontsize='24', horiı
sns.boxplot(y='Housheold_Size', x='Predict_n', data=df_ap, palette='Purples')
#plt.xticks([])
plt.ylabel('Housheold Size')
```

Out[7]:
```
Text(0, 0.5, 'Housheold Size')
```



Compare income dollars to TV Show rank

In [8]:
```
#Code Block 8

df_ap_chart = df_ap.sample(1000, random_state=42)
```

In [9]:
```
#Code Block 9

df_ap_cols = df_ap_chart.columns
df_ap_cols = df_ap_cols.drop(['Income_Dollars', 'HomeOwner_Renter','Marital_Status', 'Age',
                              'Adults_in_Household', 'Household_Size',
                              'Discretionary_Spending_Dollars', 'PolicyNumber', 'DriverNumber',
                              'DriverCount', 'APID', 'Policy', 'Predict', 'Predict_4',
                              'Predict_3n', 'Predict_n','APID'])
df_ap_melt =pd.melt(df_ap_chart, id_vars=['APID'], value_vars=df_ap_cols)
df_ap_melt=df_ap_melt.rename(columns = {'variable':'TV_Show', \
                        'value':'Rank'})
df_ap_melt
```

Out[9]:

	APID	TV_Show	Rank
0	04ZJUS01JYTDGEY2	Watch_Cooking_Shows	20.0

	APID	TV_Show	Rank
1	04ZJUS01LRC2XFKY	Watch_Cooking_Shows	1.0
2	04ZJUS02Q3KK4WPD	Watch_Cooking_Shows	12.0
3	04ZJUS021DNF9FD2	Watch_Cooking_Shows	8.0
4	04ZJUS0236Q8T2B2	Watch_Cooking_Shows	6.0
...	...	...	...
15995	04ZJUS01FDG3RCQN	Watch_Wrestling_Shows	7.0
15996	04ZJUS02T7G2NKJ9	Watch_Wrestling_Shows	11.0
15997	04ZJUS02HX64M8X6	Watch_Wrestling_Shows	4.0
15998	04ZJUS01RDPBYQTT	Watch_Wrestling_Shows	13.0
15999	04ZJUS01Z71Z4SVF	Watch_Wrestling_Shows	1.0

16000 rows × 3 columns

In [10]:
```
#Code Block 10

df_ap_melt = pd.merge(df_ap_melt, df_ap_chart[['APID', 'Income_Dollars', 'Age' ,'Householod_Size',
                                               'Discretionary_Spending_Dollars' ,
                                               'Predict_4', 'Predict_n']], how='left', on='APID')
display(df_ap_melt.info())
df_ap_melt.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16000 entries, 0 to 15999
Data columns (total 9 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   APID                            16000 non-null  object
 1   TV_Show                         16000 non-null  object
 2   Rank                            16000 non-null  float64
 3   Income_Dollars                  16000 non-null  float64
 4   Age                             16000 non-null  int64
 5   Householod_Size                 16000 non-null  int64
 6   Discretionary_Spending_Dollars  16000 non-null  float64
 7   Predict_4                       16000 non-null  int64
 8   Predict_n                       16000 non-null  int64
dtypes: float64(3), int64(4), object(2)
memory usage: 1.2+ MB
None
```

Out[10]:

	APID	TV_Show	Rank	Income_Dollars	Age	Householod_Size	Discretionary_Spending_Dollars	P
0	04ZJUS01JYTDGEY2	Watch_Cooking_Shows	20.0	22000.0	38	4	8987.0	
1	04ZJUS01LRC2XFKY	Watch_Cooking_Shows	1.0	70000.0	58	2	10919.0	
2	04ZJUS02Q3KK4WPD	Watch_Cooking_Shows	12.0	64000.0	34	1	18072.0	
3	04ZJUS021DNF9FD2	Watch_Cooking_Shows	8.0	46000.0	58	4	6035.0	
4	04ZJUS0236Q8T2B2	Watch_Cooking_Shows	6.0	17000.0	70	6	888.0	

In [11]:
```
#Code Block 11

df_ap_melt.info()
```

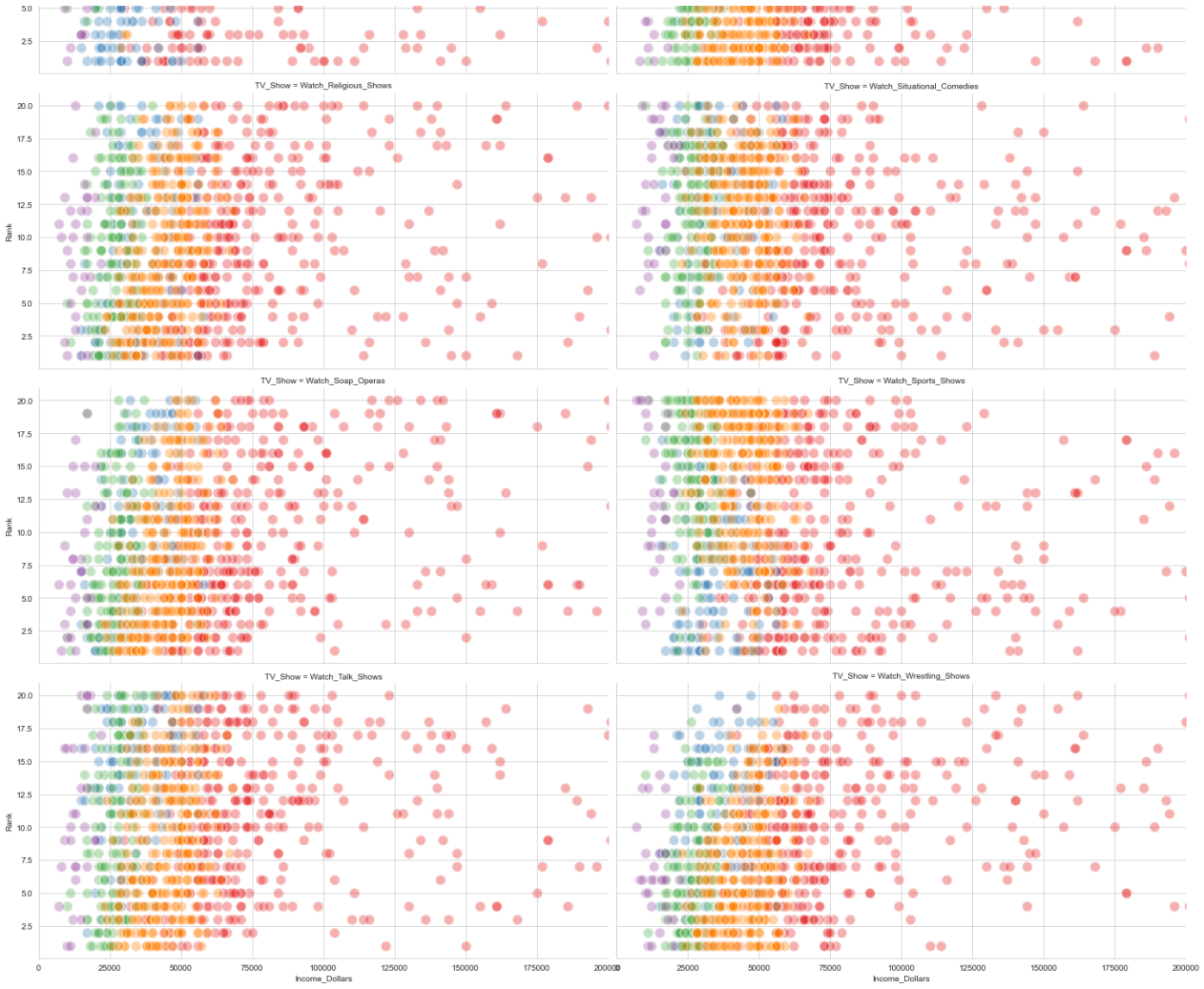
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16000 entries, 0 to 15999
Data columns (total 9 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   APID                            16000 non-null  object
 1   TV_Show                         16000 non-null  object
 2   Rank                            16000 non-null  float64
```

```
 3   Income_Dollars                 16000 non-null  float64
 4   Age                            16000 non-null  int64
 5   Household_Size                 16000 non-null  int64
 6   Discretionary_Spending_Dollars  16000 non-null  float64
 7   Predict_4                      16000 non-null  int64
 8   Predict_n                      16000 non-null  int64
dtypes: float64(3), int64(4), object(2)
memory usage: 1.2+ MB
```

In [12]:
```
#Code Block 12

plt.figure(figsize=(20,10)) #changes area of scatterplot
sns.lmplot(y='Rank', x='Income_Dollars', hue='Predict_n', fit_reg = False, col="TV_Show", col_wrap=2, data
           aspect = 2, scatter_kws={"alpha":0.35,"s":150,"linewidth":2,"edgecolor":"white"}, line_kws={'c
plt.xlim(0, 200000)
```

Out[12]:
```
(0.0, 200000.0)
```

```
<Figure size 1440x720 with 0 Axes>
```



Creating the Dendrogram

Approach to using Hierarchical clustering

- Create a dendrogram to visually depict the clusters

Dendrogram 1

In [13]:
```
#Code Block 13

df_ap_100_demo = df_ap.sample(100, random_state=42)
df_ap_100_demo = df_ap_100_demo.reset_index()
df_ap_100_demo = df_ap_100_demo.drop('index', axis=1)
df_ap_100 = df_ap_100_demo.iloc[:, 1:17]
df_ap_100.info()
```

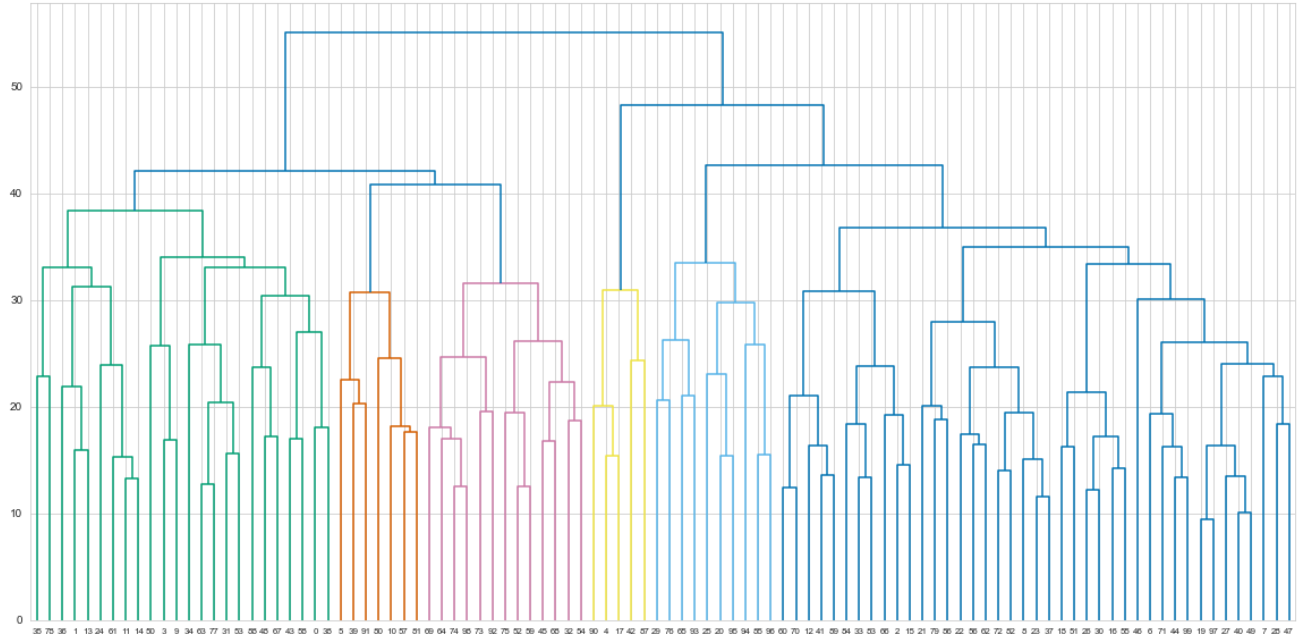
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Watch_Cooking_Shows             100 non-null    float64
 1   Watch_Documentaries             100 non-null    float64
 2   Watch_Drama_Shows               100 non-null    float64
 3   Watch_Game_Shows                100 non-null    float64
 4   Watch_Home_Improvement_Shows    100 non-null    float64
 5   Watch_Music_Videos              100 non-null    float64
 6   Watch_Auto_Racing               100 non-null    float64
 7   Watch_News_Shows                100 non-null    float64
 8   Watch_Police_or_Detective_Shows 100 non-null    float64
 9   Watch_Reality_Television_Shows  100 non-null    float64
```

```
 10  Watch_Religious_Shows           100 non-null    float64
 11  Watch_Situational_Comedies      100 non-null    float64
 12  Watch_Soap_Operas               100 non-null    float64
 13  Watch_Sports_Shows              100 non-null    float64
 14  Watch_Talk_Shows                100 non-null    float64
 15  Watch_Wrestling_Shows           100 non-null    float64
dtypes: float64(16)
memory usage: 12.6 KB
```

In [14]:
```
#Code Block 14

from scipy.cluster.hierarchy import linkage, dendrogram
```

In [15]:
```
#Code Block 15

mergings = linkage(df_ap_100, method='complete')
```

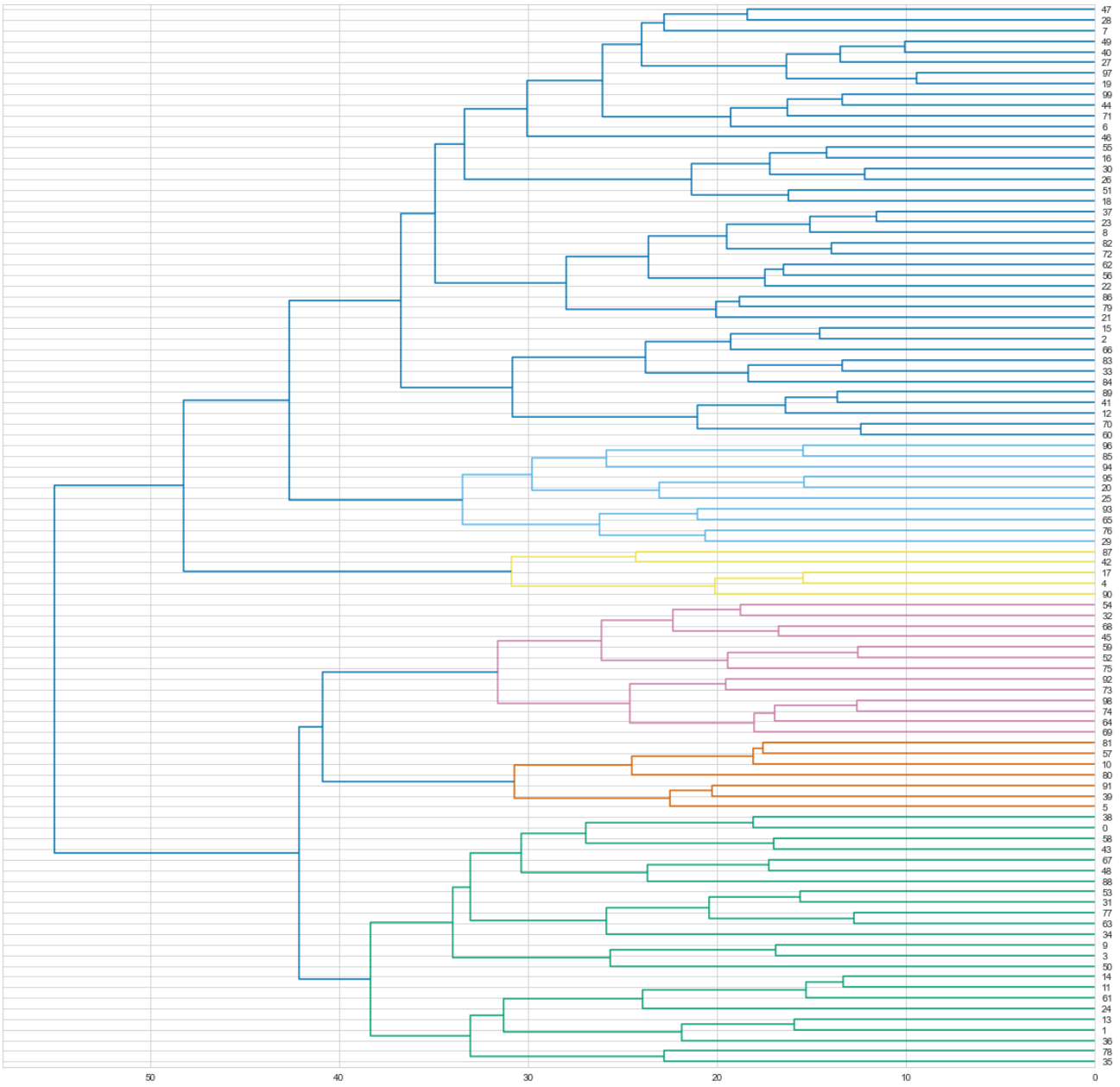
In [16]:
```
#Code Block 16

plt.figure(figsize=(20,10))
dendrogram(mergings, leaf_rotation=0, leaf_font_size=8)
plt.show()
```



Dendrogram 2

Orient to the left

In [17]:
```
#Code Block 17

plt.figure(figsize=(20,20))
dendrogram(mergings, leaf_rotation=0, leaf_font_size=10, orientation='left')
plt.show()
```

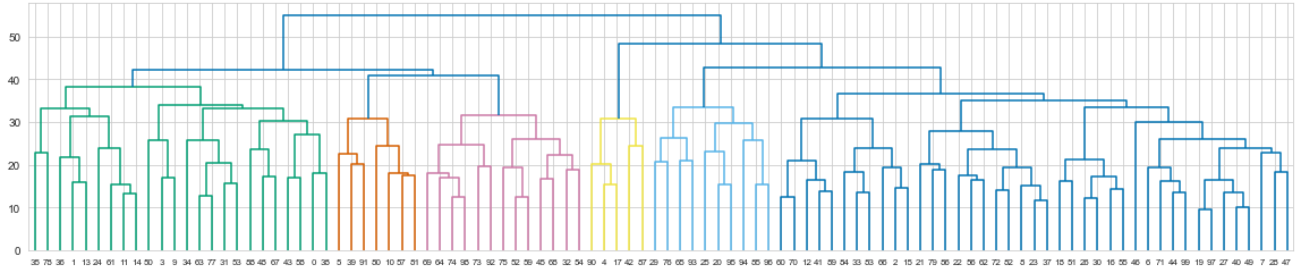


Dendrogram No. 3

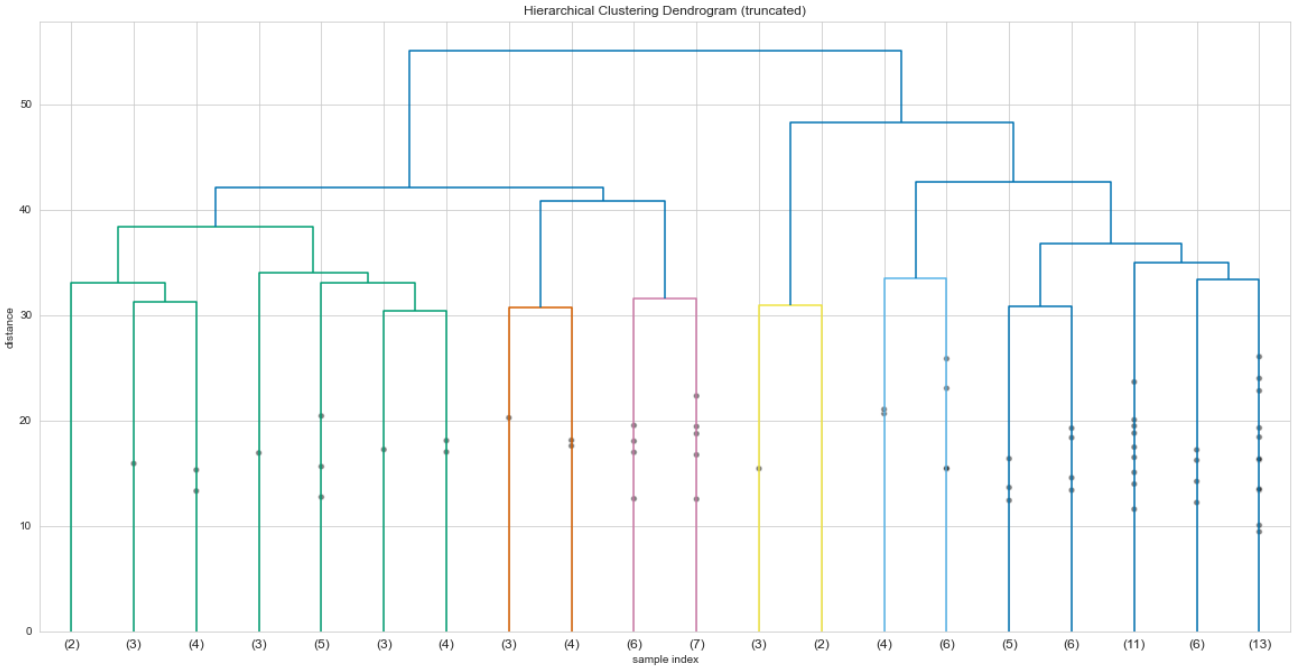
Dendrogram with more features

By setting truncate and p, it gives you an easier view of your dendrogram (p = 20)

In [18]:
```
#Code Block 18

plt.figure(figsize=(20,4))
dendrogram(mergings, leaf_rotation=0, leaf_font_size=8)
plt.show()
```



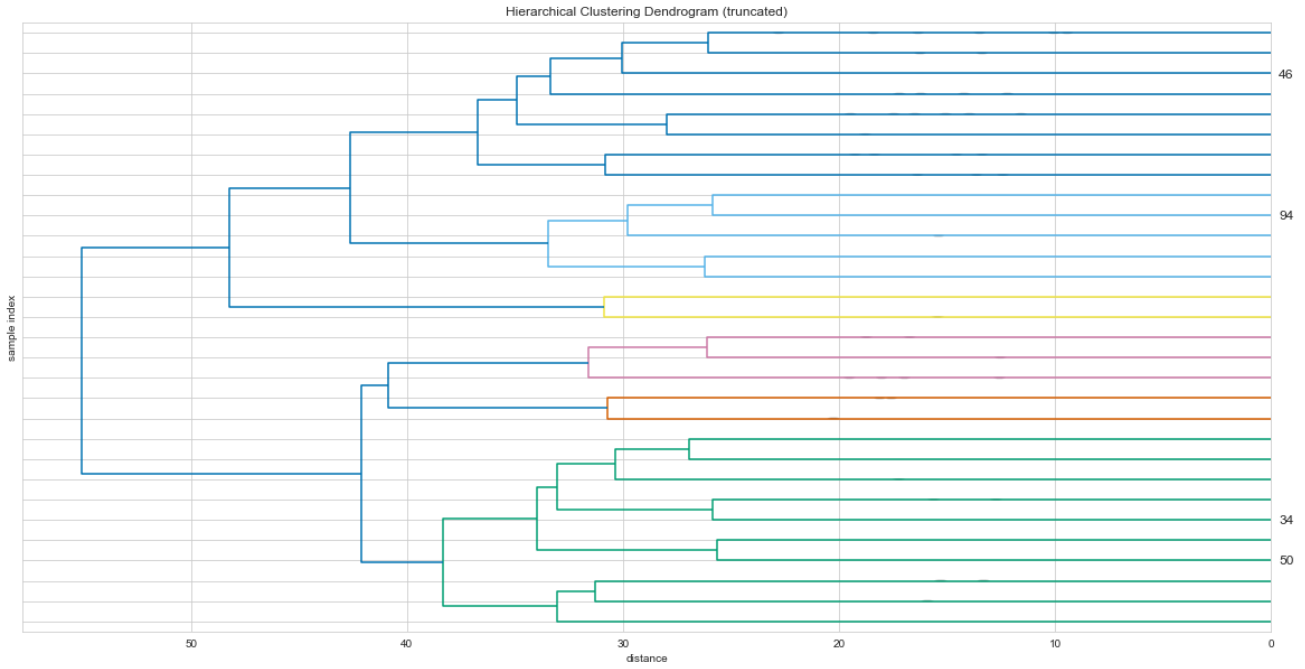
In [19]:
```
#Code Block 19

plt.figure(figsize=(20,10))
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    mergings,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=20,  # show only the last p merged clusters
    show_leaf_counts=True,  # otherwise numbers in brackets are counts
    leaf_rotation=0.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated branches
    orientation='top' #sets orientation to horizontal instead of vertical
)
plt.show()
```



Dendrogram No. 4

By setting truncate and p, it gives you an easier view of your dendrogram (p = 30)

In [20]:
```
#Code Block 20

plt.figure(figsize=(20,10))
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.ylabel('sample index')
plt.xlabel('distance')
dendrogram(
    mergings,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=30,  # show only the last p merged clusters
    show_leaf_counts=False,  # otherwise numbers in brackets are counts
    leaf_rotation=0.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated branches
    orientation='left' #sets orientation to horizontal instead of vertical
)
plt.show()
```



Hierarchical clustering

In [21]:
```
#Code Block 20

from scipy.cluster.hierarchy import fcluster
```

In [22]:
```
#Code Block 21

df_ap_pred = fcluster(mergings, 45, criterion='distance')
df_ap_pred = pd.DataFrame(df_ap_pred)
df_ap_pred.columns = ['Pred_45']
df_ap_pred.head()
```

Out[22]:

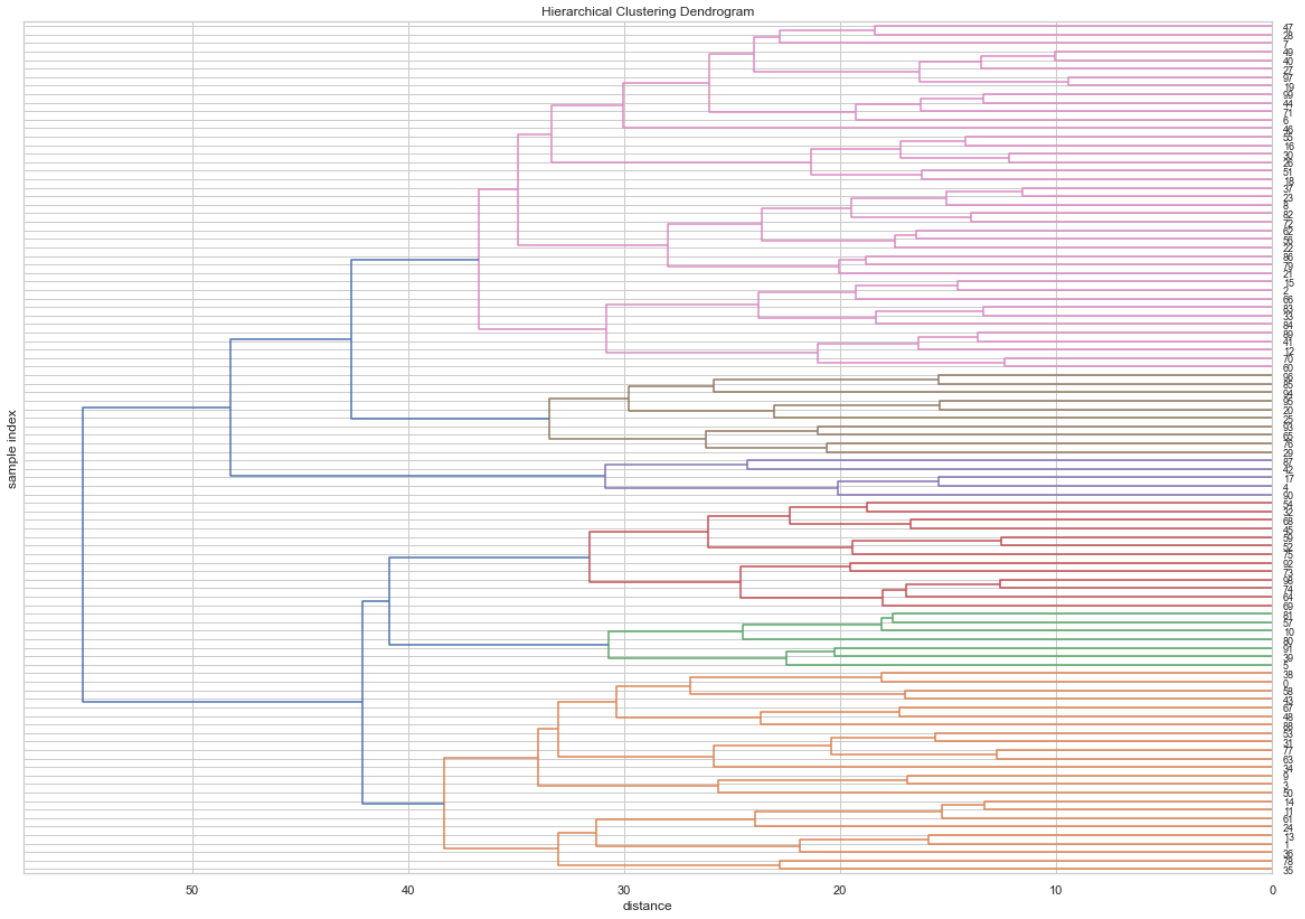
	Pred_45
0	1
1	1
2	3
3	1
4	2

In [23]:
```
#Code Block 22

df_ap_pred['Pred_45'].value_counts()
```

Out[23]:
```
3    51
1    44
2     5
Name: Pred_45, dtype: int64
```

In [24]:
```
#Code Block 23

sns.set(style='whitegrid')

plt.figure(figsize=(20,14))
plt.title('Hierarchical Clustering Dendrogram')
plt.ylabel('sample index')
plt.xlabel('distance')
dendrogram(
    mergings,
```

```
    show_leaf_counts=False,  # otherwise numbers in brackets are counts
    leaf_rotation=0.,
    leaf_font_size=9.,
    show_contracted=True,  # to get a distribution impression in truncated branches
    orientation='left' #sets orientation to horizontal instead of vertical
)
plt.show()
```



In [25]:
```
#Code Block 24

df_ap_pred_42 = fcluster(mergings, 42, criterion='distance')
df_ap_pred_42 = pd.DataFrame(df_ap_pred_42)
df_ap_pred_42.columns = ['Predict_h_42']
df_ap_pred_42.head()
```

Out[25]:

	Predict_h_42
0	1
1	1
2	5
3	1
4	3

In [26]:
```
#Code Block 25

df_ap_pred_42['Predict_h_42'].value_counts()
```

Out[26]:
```
5    41
1    24
2    20
4    10
3     5
Name: Predict_h_42, dtype: int64
```

In [27]:
```
#Code Block 26

df_ap_100_demo = pd.concat([df_ap_100_demo, df_ap_pred_42], axis=1)
round(df_ap_100_demo.groupby('Predict_h_42').mean().T,2)
```

Out[27]:

Predict_h_42	1	2	3	4	5
Watch_Cooking_Shows	12.75	14.10	6.8	13.1	7.78
Watch_Documentaries	12.04	10.90	7.8	14.8	9.07
Watch_Drama_Shows	12.00	13.00	3.6	12.2	10.17
Watch_Game_Shows	9.12	15.75	4.8	14.0	7.00
Watch_Home_Improvement_Shows	7.00	8.35	10.6	17.3	14.80
Watch_Music_Videos	14.79	10.80	10.0	8.0	4.76
Watch_Auto_Racing	5.67	7.25	16.8	11.9	13.20
Watch_News_Shows	5.54	11.05	2.4	16.5	15.61
Watch_Police_or_Detective_Shows	5.75	11.35	1.4	16.7	14.71
Watch_Reality_Television_Shows	16.17	8.80	12.8	9.0	4.17
Watch_Religious_Shows	10.08	11.85	1.4	15.3	7.63
Watch_Situational_Comedies	12.50	12.00	11.2	11.4	11.34
Watch_Soap_Operas	10.92	12.65	3.8	12.1	6.93
Watch_Sports_Shows	6.46	9.70	3.2	10.3	14.71
Watch_Talk_Shows	14.21	12.30	1.6	9.3	7.88
Watch_Wrestling_Shows	12.54	12.55	7.4	7.4	6.12
Income_Dollars	72208.33	59050.00	50200.0	47800.0	56121.95
Age	64.58	47.90	65.6	39.0	38.34
Adults_in_Household	2.83	1.65	3.4	1.8	1.46
Household_Size	3.71	2.05	3.8	2.3	1.83
Discretionary_Spending_Dollars	9387.58	7812.80	3972.2	7342.5	8223.80
DriverNumber	1.00	1.00	NaN	1.0	1.00
DriverCount	1.60	3.50	NaN	3.5	1.86
Policy	0.21	0.50	0.0	0.2	0.17
Predict	1.71	1.50	2.2	2.2	2.29
Predict_n	0.46	1.60	1.6	2.6	2.51
Predict_4	2.71	1.15	2.6	0.7	1.29
Predict_3n	1.33	1.20	1.0	1.6	1.29

In [28]:
```
#Code Block 27

round(df_ap_100_demo[df_ap_100_demo['Predict_h_42']==2][['Age', 'Income_Dollars', 'Adults_in_Household',
```

Out[28]:

	count	mean	std	min	25%	50%	75%	max
Age	20.0	47.90	10.31	24.0	42.0	48.0	56.0	70.0
Income_Dollars	20.0	59050.00	27390.88	13000.0	45000.0	53000.0	65500.0	140000.0
Adults_in_Household	20.0	1.65	0.93	1.0	1.0	1.5	2.0	5.0
Household_Size	20.0	2.05	1.05	1.0	1.0	2.0	3.0	5.0
Discretionary_Spending_Dollars	20.0	7812.80	7157.88	888.0	2958.0	6049.5	8782.5	29860.0

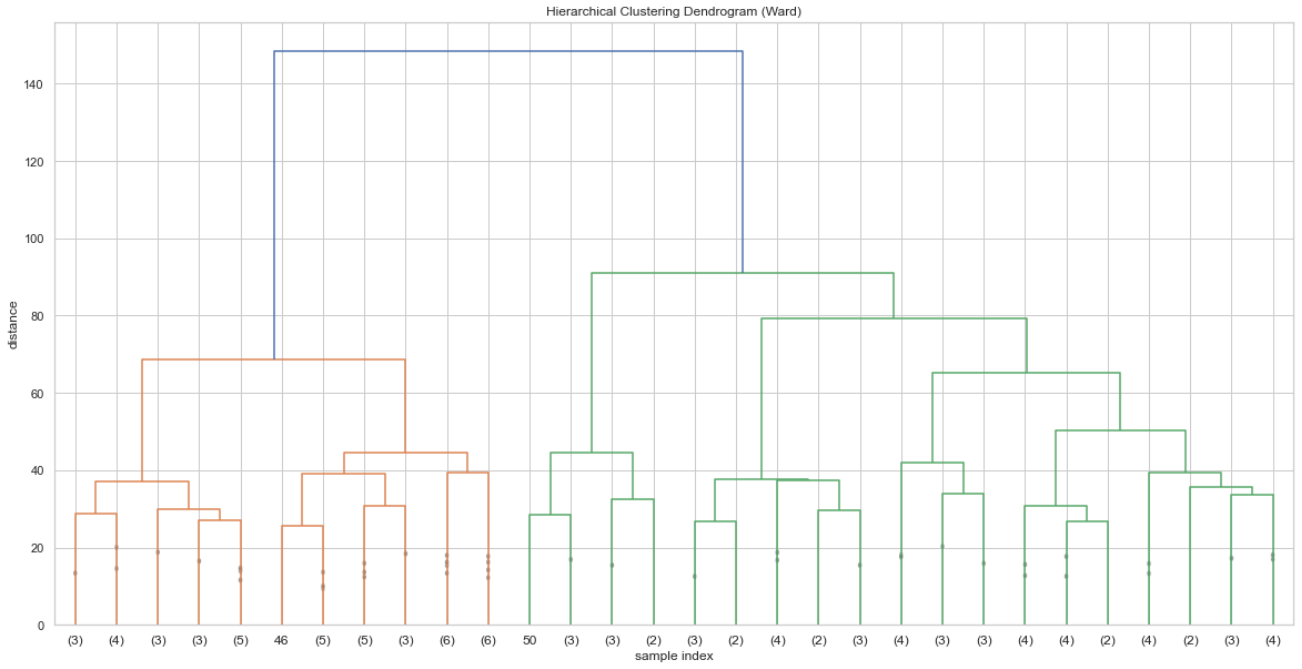
	count	mean	std	min	25%	50%	75%	max
Policy	20.0	0.50	0.51	0.0	0.0	0.5	1.0	1.0

Hierarchical cluster analysis with Ward and comparisons to other clusters

- **Complete** linkage clustering: It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the largest value (i.e., maximum value) of these dissimilarities as the distance between the two clusters. It tends to produce more compact clusters.
- **Single** linkage clustering: It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the smallest of these dissimilarities as a linkage criterion. It tends to produce long, "loose" clusters.
- **Average** linkage clustering: It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the average of these dissimilarities as the distance between the two clusters.
- **Centroid** linkage clustering: It computes the dissimilarity between the centroid for cluster 1 (a mean vector of length p variables) and the centroid for cluster 2.
- **Ward's** minimum variance method: It minimizes the total within-cluster variance. At each step the pair of clusters with minimum between-cluster distance are merged. *This is most similar to K-Means.*

In [29]:
```
#Code Block 28

mergings_w = linkage(df_ap_100, method='ward')
```

In [30]:
```
#Code Block 29

plt.figure(figsize=(20,10))
plt.title('Hierarchical Clustering Dendrogram (Ward)')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    mergings_w,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=30,  # show only the last p merged clusters
    show_leaf_counts=True,  # otherwise numbers in brackets are counts
    leaf_rotation=0.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated branches
    orientation='top' #sets orientation to horizontal instead of vertical
)
plt.show()
```



In [31]:
```
#Code Block 30

df_ap_pred_w = fcluster(mergings_w, 78, criterion='distance')
df_ap_pred_w = pd.DataFrame(df_ap_pred_w)
df_ap_pred_w.columns = ['Predict_ward']
df_ap_pred_w.head()
```

Out[31]:

	Predict_ward
0	4
1	4
2	1
3	2
4	2

In [32]:
```
#Code Block 31

df_ap_pred_w['Predict_ward'].value_counts()
```

Out[32]:
```
1    44
4    33
3    14
2     9
Name: Predict_ward, dtype: int64
```

Crosstab analysis of groups

In [33]:
```
#Code Block 32

df_ap_100_demo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 33 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   APID                            100 non-null    object
 1   Watch_Cooking_Shows             100 non-null    float64
 2   Watch_Documentaries             100 non-null    float64
 3   Watch_Drama_Shows               100 non-null    float64
 4   Watch_Game_Shows                100 non-null    float64
 5   Watch_Home_Improvement_Shows    100 non-null    float64
```

```
 6   Watch_Music_Videos               100 non-null    float64
 7   Watch_Auto_Racing                100 non-null    float64
 8   Watch_News_Shows                 100 non-null    float64
 9   Watch_Police_or_Detective_Shows  100 non-null    float64
 10  Watch_Reality_Television_Shows   100 non-null    float64
 11  Watch_Religious_Shows            100 non-null    float64
 12  Watch_Situational_Comedies       100 non-null    float64
 13  Watch_Soap_Operas                100 non-null    float64
 14  Watch_Sports_Shows               100 non-null    float64
 15  Watch_Talk_Shows                 100 non-null    float64
 16  Watch_Wrestling_Shows            100 non-null    float64
 17  Income_Dollars                   100 non-null    float64
 18  HomeOwner_Renter                 100 non-null    object
 19  Marital_Status                   100 non-null    object
 20  Age                              100 non-null    int64
 21  Adults_in_Household              100 non-null    int64
 22  Househould_Size                  100 non-null    int64
 23  Discretionary_Spending_Dollars   100 non-null    float64
 24  PolicyNumber                     24 non-null     object
 25  DriverNumber                     24 non-null     float64
 26  DriverCount                      24 non-null     float64
 27  Policy                           100 non-null    int64
 28  Predict                          100 non-null    int64
 29  Predict_n                        100 non-null    int64
 30  Predict_4                        100 non-null    int64
 31  Predict_3n                       100 non-null    int64
 32  Predict_h_42                     100 non-null    int32
dtypes: float64(20), int32(1), int64(8), object(4)
memory usage: 25.5+ KB
```

In [34]:
```
#Code Block 33

df_ap_100_demo = pd.concat([df_ap_100_demo, df_ap_pred_w], axis=1)
```

In [35]:
```
#Code Block 34

pd.crosstab(df_ap_100_demo['Predict_4'], df_ap_100_demo['Predict_h_42'])
```

Out[35]:

Predict_h_42	1	2	3	4	5
Predict_4					
0	2	11	0	6	3
1	0	0	1	1	23
2	1	4	0	3	15
3	21	5	4	0	0

In [36]:
```
#Code Block 35

pd.crosstab(df_ap_100_demo['Predict_4'], df_ap_100_demo['Predict_ward'])
```

Out[36]:

Predict_ward	1	2	3	4
Predict_4				
0	3	0	13	6
1	24	1	0	0
2	17	0	1	5
3	0	8	0	22

In [37]:
```
#Code Block 36

pd.crosstab(df_ap_100_demo['Predict'], df_ap_100_demo['Predict_n'])
```

Out[37]:

Predict_n	0	1	2	3	4
Predict					

Predict_n	0	1	2	3	4
Predict					
0	5	7	1	0	4
1	8	0	2	1	10
2	13	10	0	2	0
3	6	0	3	1	11
4	4	1	3	0	8

Hierarchical cluster analysis with larger sample

In [38]:
```
#Code Block 37

df_ap.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8717 entries, 0 to 8716
Data columns (total 32 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   APID                             8717 non-null   object
 1   Watch_Cooking_Shows              8717 non-null   float64
 2   Watch_Documentaries              8717 non-null   float64
 3   Watch_Drama_Shows                8717 non-null   float64
 4   Watch_Game_Shows                 8717 non-null   float64
 5   Watch_Home_Improvement_Shows     8717 non-null   float64
 6   Watch_Music_Videos               8717 non-null   float64
 7   Watch_Auto_Racing                8717 non-null   float64
 8   Watch_News_Shows                 8717 non-null   float64
 9   Watch_Police_or_Detective_Shows  8717 non-null   float64
 10  Watch_Reality_Television_Shows   8717 non-null   float64
 11  Watch_Religious_Shows            8717 non-null   float64
 12  Watch_Situational_Comedies       8717 non-null   float64
 13  Watch_Soap_Operas                8717 non-null   float64
 14  Watch_Sports_Shows               8717 non-null   float64
 15  Watch_Talk_Shows                 8717 non-null   float64
 16  Watch_Wrestling_Shows            8717 non-null   float64
 17  Income_Dollars                   8717 non-null   float64
 18  HomeOwner_Renter                 8717 non-null   object
 19  Marital_Status                   8717 non-null   object
 20  Age                              8717 non-null   int64
 21  Adults_in_Household              8717 non-null   int64
 22  Househoold_Size                  8717 non-null   int64
 23  Discretionary_Spending_Dollars   8717 non-null   float64
 24  PolicyNumber                     2417 non-null   object
 25  DriverNumber                     2417 non-null   float64
 26  DriverCount                      2417 non-null   float64
 27  Policy                           8717 non-null   int64
 28  Predict                          8717 non-null   int64
 29  Predict_n                        8717 non-null   int64
 30  Predict_4                        8717 non-null   int64
 31  Predict_3n                       8717 non-null   int64
dtypes: float64(20), int64(8), object(4)
memory usage: 2.5+ MB
```

In [39]:
```
#Code Block 38

df_ap_cluster = df_ap.iloc[:, 1:17]
df_ap_cluster.info()
```

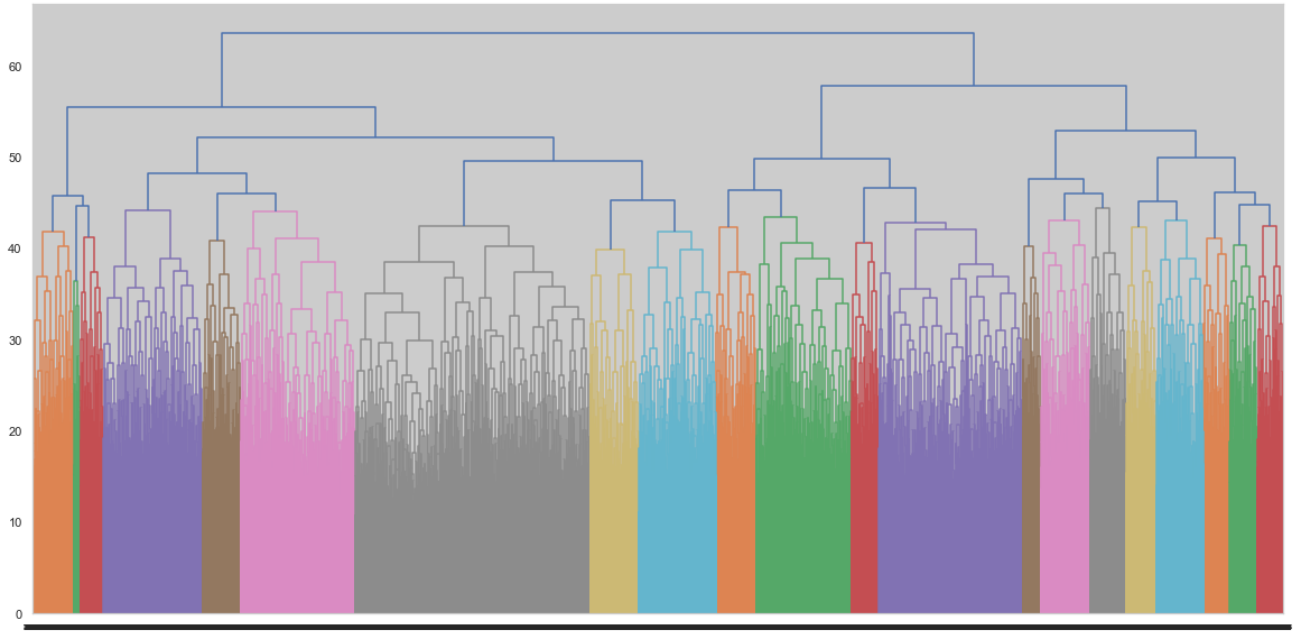
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8717 entries, 0 to 8716
Data columns (total 16 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Watch_Cooking_Shows              8717 non-null   float64
 1   Watch_Documentaries              8717 non-null   float64
 2   Watch_Drama_Shows                8717 non-null   float64
 3   Watch_Game_Shows                 8717 non-null   float64
```

```
 4   Watch_Home_Improvement_Shows     8717 non-null   float64
 5   Watch_Music_Videos               8717 non-null   float64
 6   Watch_Auto_Racing                8717 non-null   float64
 7   Watch_News_Shows                 8717 non-null   float64
 8   Watch_Police_or_Detective_Shows  8717 non-null   float64
 9   Watch_Reality_Television_Shows   8717 non-null   float64
 10  Watch_Religious_Shows            8717 non-null   float64
 11  Watch_Situational_Comedies       8717 non-null   float64
 12  Watch_Soap_Operas                8717 non-null   float64
 13  Watch_Sports_Shows               8717 non-null   float64
 14  Watch_Talk_Shows                 8717 non-null   float64
 15  Watch_Wrestling_Shows            8717 non-null   float64
dtypes: float64(16)
memory usage: 1.4 MB
```

In [40]:
```
%%time

#Code Block 39

mergings_all = linkage(df_ap_cluster, method='complete')
```

```
Wall time: 2.54 s
```

In [41]:
```
%%time

#Code Block 40

plt.figure(figsize=(20,10))
dendrogram(mergings_all, leaf_rotation=0, leaf_font_size=8)
plt.show()
```




```
Wall time: 3min 58s
```

In [42]:
```
%%time

#Code Block 41

df_ap_pred_all = fcluster(mergings_all, 55, criterion='distance')
df_ap_pred_all = pd.DataFrame(df_ap_pred_all)
df_ap_pred_all.columns = ['Pred_h_55']
df_ap_pred_all.head()
```

```
Wall time: 18.9 ms
```

Out[42]:

	Pred_h_55
0	2
1	2
2	4

	Pred_h_55
3	1
4	4

In [43]:
```
#Code Block 42

df_ap_pred_all['Pred_h_55'].value_counts()
```

Out[43]:
```
2    4271
3    2135
4    1817
1     494
Name: Pred_h_55, dtype: int64
```

Normalize the data and conduct hierarchical clustering

In [44]:
```
#Code Block 43

from sklearn import preprocessing

n_scaler = preprocessing.Normalizer()
```

In [45]:
```
#Code Block 44

df_ap_n = n_scaler.fit_transform(df_ap_cluster)
df_ap_n = pd.DataFrame(df_ap_n, columns=(df_ap_cols))
df_ap_n.head()
```

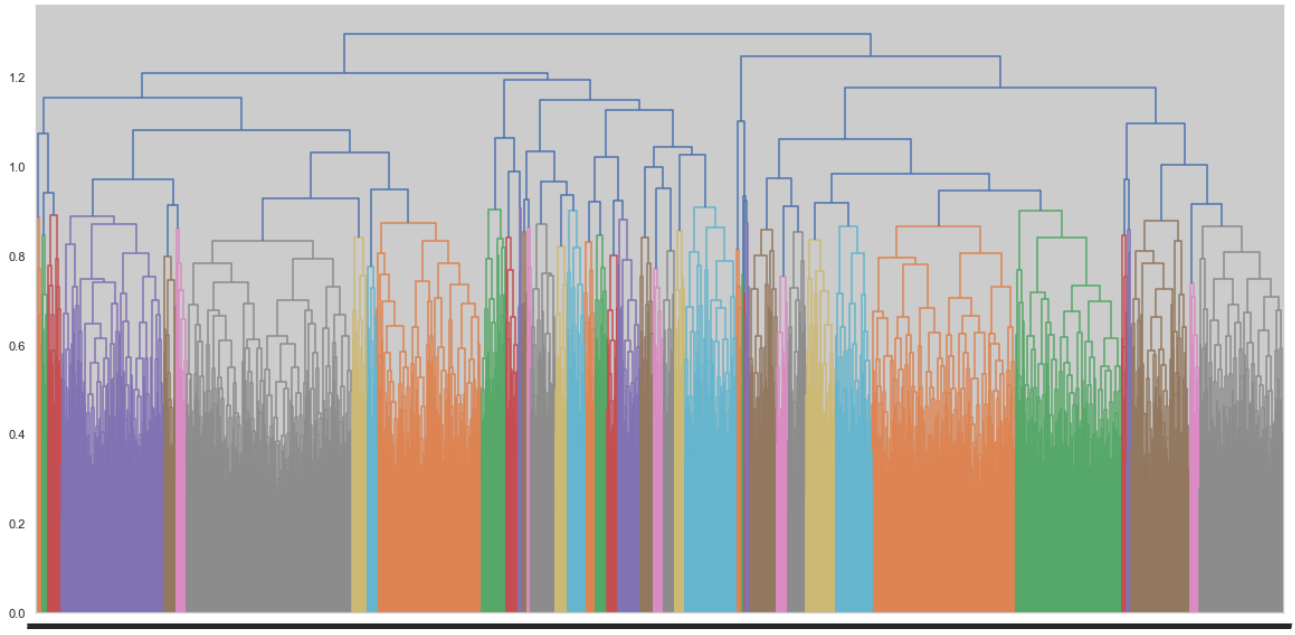
Out[45]:

	Watch_Cooking_Shows	Watch_Documentaries	Watch_Drama_Shows	Watch_Game_Shows	Watch_Home_Improvement_Shows
0	0.046714	0.140143	0.303642	0.046714	0.280285
1	0.093942	0.187885	0.164399	0.446226	0.046971
2	0.025000	0.125000	0.050000	0.300000	0.325000
3	0.136133	0.435626	0.027227	0.054453	0.381173
4	0.156760	0.089577	0.089577	0.156760	0.425492

In [46]:
```
%%time

#Code Block 45

mergings_n = linkage(df_ap_n, method='complete')
```

```
Wall time: 3.29 s
```

In [47]:
```
%%time

#Code Block 46

plt.figure(figsize=(20,10))
dendrogram(mergings_n, leaf_rotation=0, leaf_font_size=8)
plt.show()
```




```
Wall time: 3min 5s
```

In [48]:
```
%%time

#Code Block 47

df_ap_pred_n = fcluster(mergings_n, 1.2, criterion='distance')
df_ap_pred_n = pd.DataFrame(df_ap_pred_n)
df_ap_pred_n.columns = ['Pred_h_n']
display(df_ap_pred_n.head())
df_ap_pred_n['Pred_h_n'].value_counts()
```

	Pred_h_n
0	4
1	4
2	4
3	2
4	2

```
Wall time: 52.9 ms
```

Out[48]:
```
4    3728
1    3113
2    1779
3      97
Name: Pred_h_n, dtype: int64
```

In [49]:
```
#Code Block 48

df_ap = pd.concat([df_ap, df_ap_pred_n, df_ap_pred_all], axis=1)
df_ap.head()
```

Out[49]:

	APID	Watch_Cooking_Shows	Watch_Documentaries	Watch_Drama_Shows	Watch_Game_Shows	Watch_Home
0	04ZJUS01QFCGYRSZ	2.0	6.0	13.0	2.0	
1	04ZJUS020XX4JE5Y	4.0	8.0	7.0	19.0	
2	04ZJUS02NPKCSZM0	1.0	5.0	2.0	12.0	
3	04ZJUS02PJ80PDYQ	5.0	16.0	1.0	2.0	
4	04ZJUS01NKVLKN10	7.0	4.0	4.0	7.0	

In [50]:
```
#Code Block 49

round(df_ap.groupby('Pred_h_n').mean().T, 2)
```

Out[50]:

Pred_h_n	1	2	3	4
Watch_Cooking_Shows	14.61	10.83	4.13	7.40
Watch_Documentaries	11.06	11.21	7.92	9.89
Watch_Drama_Shows	14.93	9.04	4.87	9.28
Watch_Game_Shows	10.83	9.57	6.03	8.88
Watch_Home_Improvement_Shows	11.93	8.28	8.35	12.25
Watch_Music_Videos	7.70	13.58	9.42	5.40
Watch_Auto_Racing	8.85	6.78	16.63	11.97
Watch_News_Shows	13.63	5.96	6.48	15.22
Watch_Police_or_Detective_Shows	13.64	5.72	6.39	14.71
Watch_Reality_Television_Shows	10.27	13.14	8.72	4.42
Watch_Religious_Shows	10.13	8.96	4.57	9.25
Watch_Situational_Comedies	11.39	11.87	7.63	10.83
Watch_Soap_Operas	12.55	9.05	4.04	6.43
Watch_Sports_Shows	8.36	8.17	9.62	15.68
Watch_Talk_Shows	11.32	13.02	6.11	8.96
Watch_Wrestling_Shows	8.82	11.71	6.78	6.84
Income_Dollars	58888.21	62418.21	41670.10	52919.53
Age	44.04	59.56	55.84	40.38
Adults_in_Household	1.96	2.59	2.37	1.52
Household_Size	2.52	3.34	3.12	1.90
Discretionary_Spending_Dollars	9056.86	7843.25	6446.00	8269.52
DriverNumber	1.03	1.03	1.00	1.04
DriverCount	2.20	2.22	2.57	2.16
Policy	0.28	0.28	0.22	0.28
Predict	1.66	1.95	1.41	2.17
Predict_n	1.64	0.73	1.28	2.35
Predict_4	0.57	2.61	1.84	1.41
Predict_3n	1.32	1.18	1.15	1.34
Pred_h_55	2.75	3.38	1.59	2.14

In [51]:
```
#Code Block 50

df_ap.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8717 entries, 0 to 8716
Data columns (total 34 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   APID                            8717 non-null   object
 1   Watch_Cooking_Shows             8717 non-null   float64
 2   Watch_Documentaries             8717 non-null   float64
 3   Watch_Drama_Shows               8717 non-null   float64
 4   Watch_Game_Shows                8717 non-null   float64
```

```
 5   Watch_Home_Improvement_Shows     8717 non-null   float64
 6   Watch_Music_Videos               8717 non-null   float64
 7   Watch_Auto_Racing                8717 non-null   float64
 8   Watch_News_Shows                 8717 non-null   float64
 9   Watch_Police_or_Detective_Shows  8717 non-null   float64
 10  Watch_Reality_Television_Shows   8717 non-null   float64
 11  Watch_Religious_Shows            8717 non-null   float64
 12  Watch_Situational_Comedies       8717 non-null   float64
 13  Watch_Soap_Operas                8717 non-null   float64
 14  Watch_Sports_Shows               8717 non-null   float64
 15  Watch_Talk_Shows                 8717 non-null   float64
 16  Watch_Wrestling_Shows            8717 non-null   float64
 17  Income_Dollars                   8717 non-null   float64
 18  HomeOwner_Renter                 8717 non-null   object
 19  Marital_Status                   8717 non-null   object
 20  Age                              8717 non-null   int64
 21  Adults_in_Household              8717 non-null   int64
 22  Household_Size                   8717 non-null   int64
 23  Discretionary_Spending_Dollars   8717 non-null   float64
 24  PolicyNumber                     2417 non-null   object
 25  DriverNumber                     2417 non-null   float64
 26  DriverCount                      2417 non-null   float64
 27  Policy                           8717 non-null   int64
 28  Predict                          8717 non-null   int64
 29  Predict_n                        8717 non-null   int64
 30  Predict_4                        8717 non-null   int64
 31  Predict_3n                       8717 non-null   int64
 32  Pred_h_n                         8717 non-null   int32
 33  Pred_h_55                        8717 non-null   int32
dtypes: float64(20), int32(2), int64(8), object(4)
memory usage: 2.6+ MB
```

In [52]:
```
#Code Block 51

pd.crosstab(df_ap['Pred_h_n'], df_ap['Pred_h_55'])
```

Out[52]:

Pred_h_55	1	2	3	4
Pred_h_n				
1	94	958	1688	373
2	163	199	211	1206
3	62	24	0	11
4	175	3090	236	227

In [53]:
```
#Code Block 52

pd.crosstab(df_ap['Predict_4'], df_ap['Pred_h_55'])
```

Out[53]:

Pred_h_55	1	2	3	4
Predict_4				
0	3	673	1754	175
1	218	1901	87	34
2	50	1604	92	259
3	223	93	202	1349

In [54]:
```
#Code Block 53

pd.crosstab(df_ap['Household_Size'], df_ap['Pred_h_55'])
```

Out[54]:

Pred_h_55	1	2	3	4
Household_Size				
1	81	2134	820	190

Pred_h_55	1	2	3	4
Hoseholld_Size				
2	121	1087	437	401
3	140	567	350	526
4	77	257	232	331
5	34	132	157	191
6	27	59	76	109
7	11	21	44	51
8	2	10	13	15
9	1	4	6	3

In [ ]: