In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import math
import matplotlib.pyplot as plt
%matplotlib inline

from kneed import KneeLocator
from sklearn.cluster import KMeans, Birch
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn import metrics
from sklearn.metrics import silhouette_score, mean_squared_error
from sklearn.ensemble import RandomForestClassifier, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from yellowbrick.cluster import KElbowVisualizer
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, KFold
```

In [3]:
```
data = pd.read_csv('CensusCanada2016Training.csv')
```

In [4]:
```
data
```

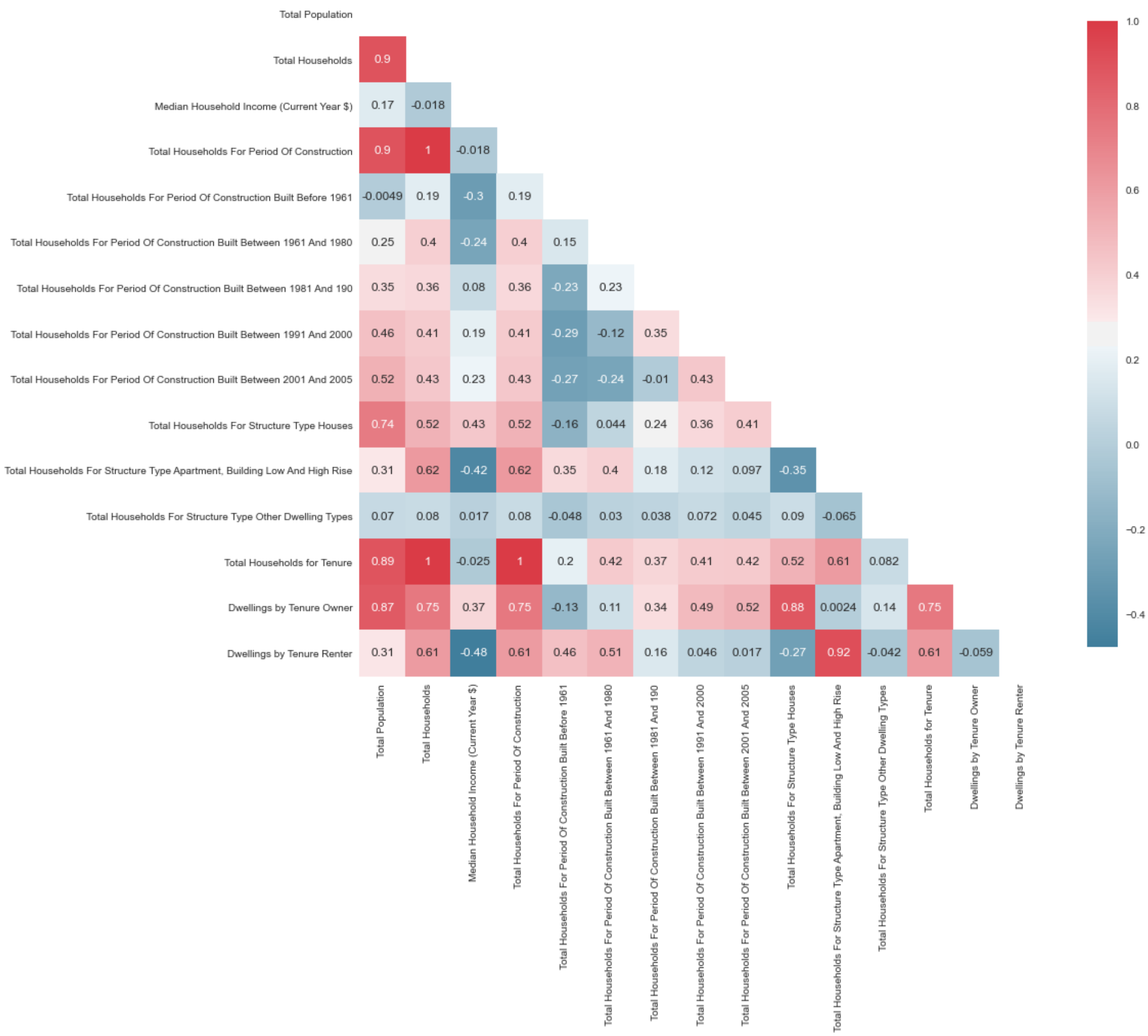
Out[4]:

	Total Population	Total Households	Median Household Income (Current Year $)	Total Households For Period Of Construction	Total Households For Period Of Construction Built Before 1961	Total Households For Period Of Construction Built Between 1961 And 1980	Total Households For Period Of Construction Built Between 1981 And 190	Total Households For Period Of Construction Built Between 1991 And 2000	Total Households For Period Of Construction Built Between 2001 And 2005	Tot Househol Fo Structu Typ House
0	4051	1441	68242.12	1441	323	199	53	182	526	91
1	2329	1026	88172.37	1026	927	70	15	3	0	79
2	5276	2071	103853.38	2071	3	607	567	651	106	141
3	5967	2203	82796.63	2203	133	1695	248	79	0	139
4	4236	1419	91648.22	1419	0	7	127	938	143	91
...	...	...	...	...	...	...	...	...	...	
4995	2588	953	108823.38	953	0	3	31	501	276	92
4996	9036	3859	68735.64	3859	678	986	386	359	448	238
4997	4689	1895	71370.58	1895	164	485	511	523	29	67
4998	3673	1038	58258.26	1038	544	185	40	95	13	79
4999	6010	1830	111457.48	1830	11	59	671	514	402	175

5000 rows × 15 columns

In [5]:
```python
# Correlation matrix
corr = data.corr()

plt.figure(figsize=(15,15))
sns.heatmap(corr,
            mask=np.triu(np.ones_like(corr, dtype=bool)),
            cmap=sns.diverging_palette(230, 10, as_cmap=True),
            square=True,
            cbar_kws={"shrink": 0.75},
            annot=True,
            annot_kws={'size':12})
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe9bed39a60>



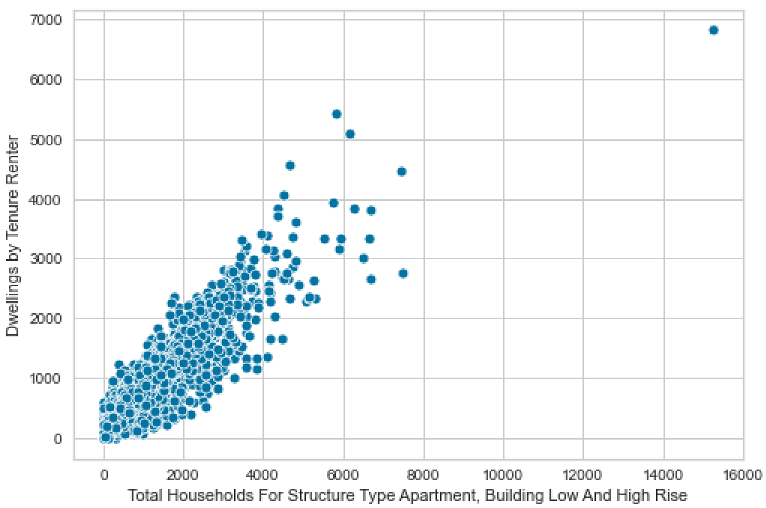
In [4]:
```
data.columns
```

Out[4]:
```
Index(['Total Population', 'Total Households',
       'Median Household Income (Current Year $)',
       'Total Households For Period Of Construction',
       'Total Households For Period Of Construction Built Before 1961',
       'Total Households For Period Of Construction Built Between 1961 And 1980',
       'Total Households For Period Of Construction Built Between 1981 And 190',
       'Total Households For Period Of Construction Built Between 1991 And 2000',
       'Total Households For Period Of Construction Built Between 2001 And 2005',
       'Total Households For Structure Type Houses',
       'Total Households For Structure Type Apartment, Building Low And High Rise',
       'Total Households For Structure Type Other Dwelling Types',
       'Total Households for Tenure', 'Dwellings by Tenure Owner',
       'Dwellings by Tenure Renter'],
      dtype='object')
```

EDA

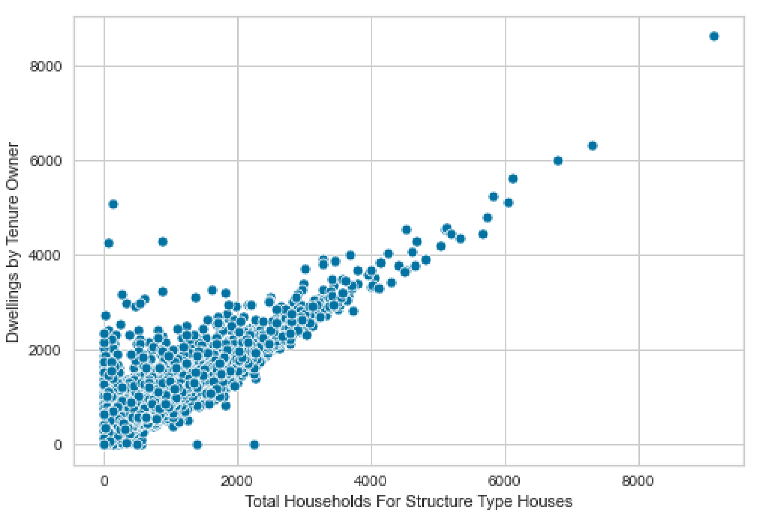
In [5]:
```
sns.scatterplot(data=data,
                x="Total Households For Structure Type Apartment, Building Low And High Rise",
                y="Dwellings by Tenure Renter")
```

Out[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffece9ea190>
```



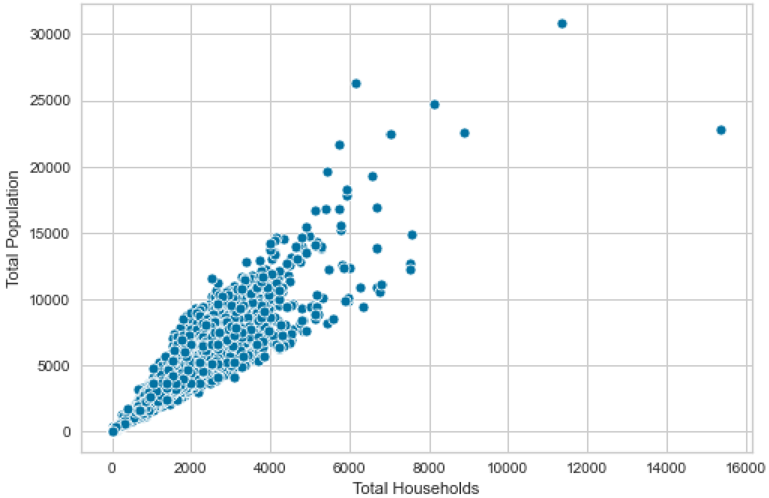
In [6]:
```
sns.scatterplot(data=data, x="Total Households For Structure Type Houses", y="Dwellings by Tenure Owne
r")
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffeceb09fa0>
```



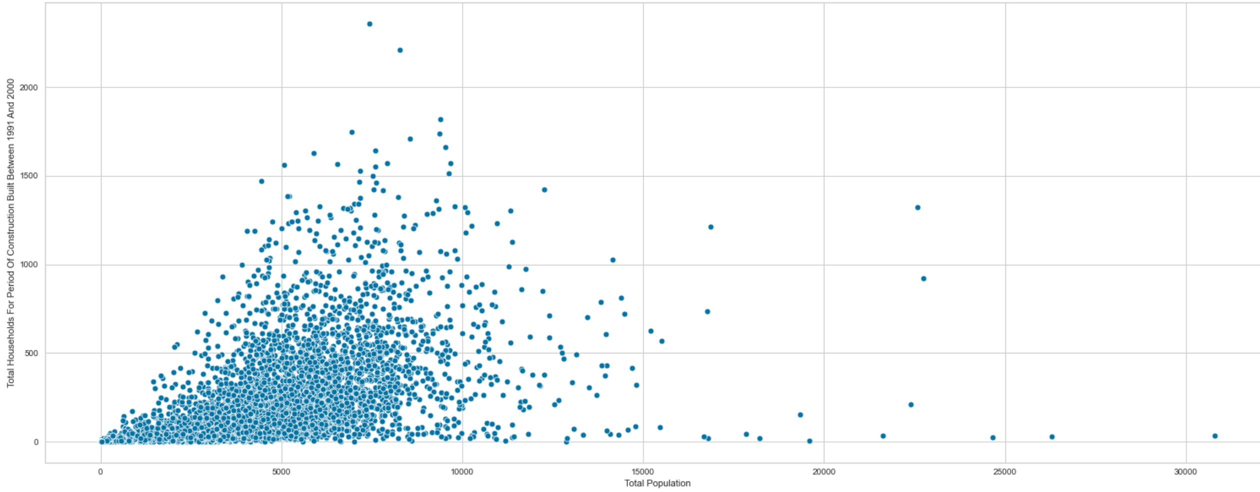
In [7]:
```
sns.scatterplot(data=data, x="Total Households", y="Total Population")
```

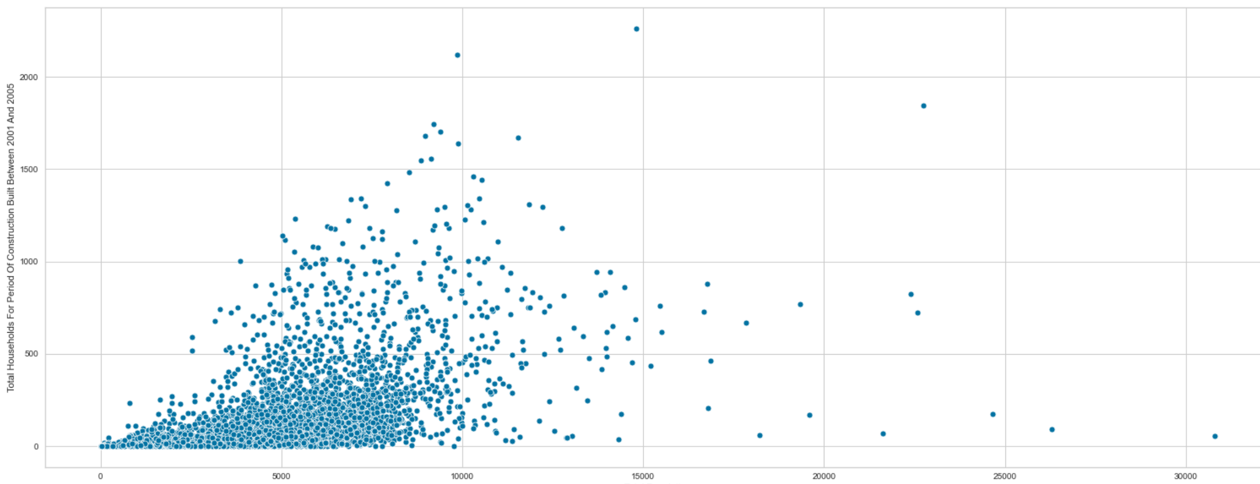
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffecace3d00>

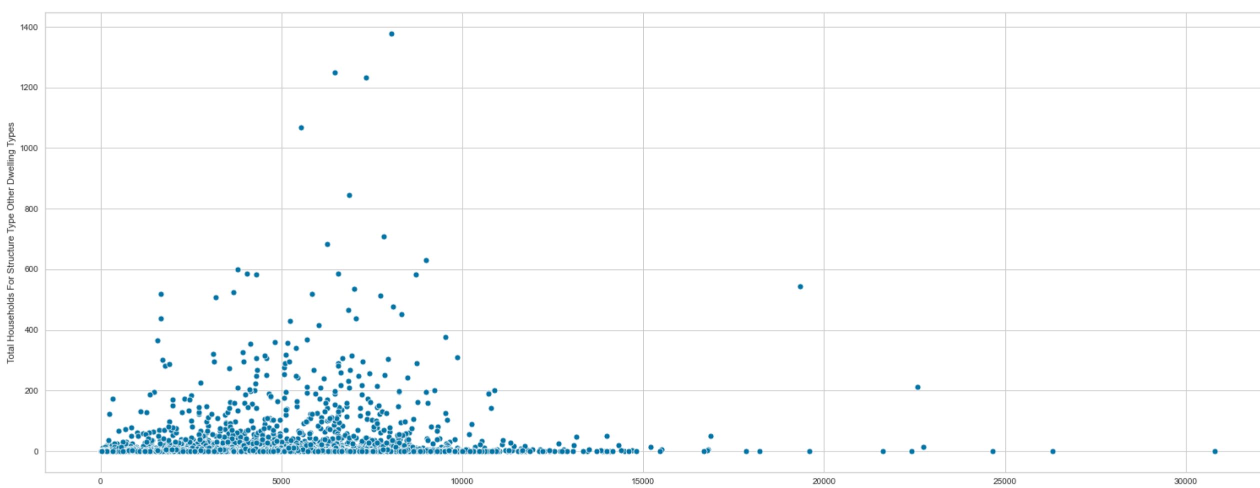


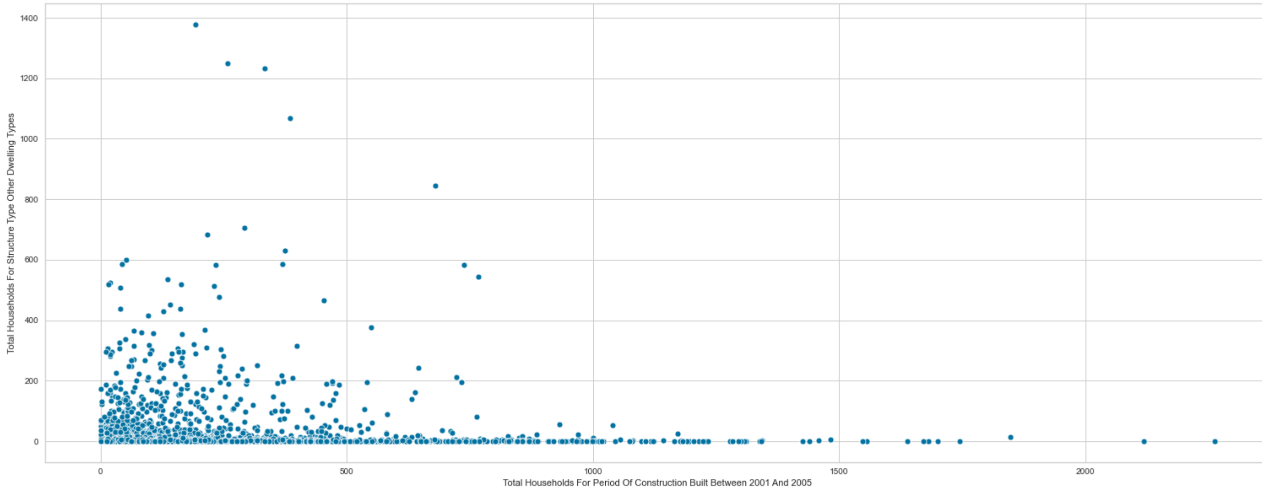
In [8]:
```
# visualizing highly correlated variables based on the correlation matrix
plt.figure(figsize=(26, 10))
sns.scatterplot(x = data.columns[0], data = data, y = data.columns[7])

plt.figure(figsize=(26, 10))
sns.scatterplot(x = data.columns[0], data = data, y = data.columns[8])

plt.figure(figsize=(26, 10))
sns.scatterplot(x = data.columns[0], data = data, y = data.columns[11])

plt.figure(figsize=(26, 10))
sns.scatterplot(x = data.columns[8], data = data, y = data.columns[11])

plt.figure(figsize=(26, 10))
sns.scatterplot(x = data.columns[9], data = data, y = data.columns[12])

plt.figure(figsize=(26, 10))
sns.scatterplot(x = data.columns[11], data = data, y = data.columns[14])
```

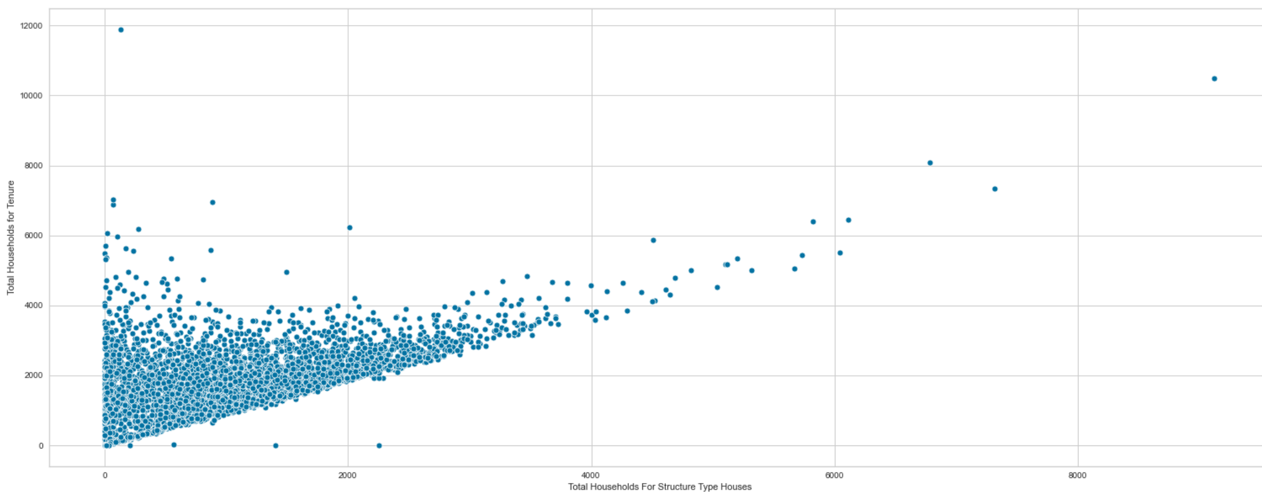
```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffecef358e0>
```

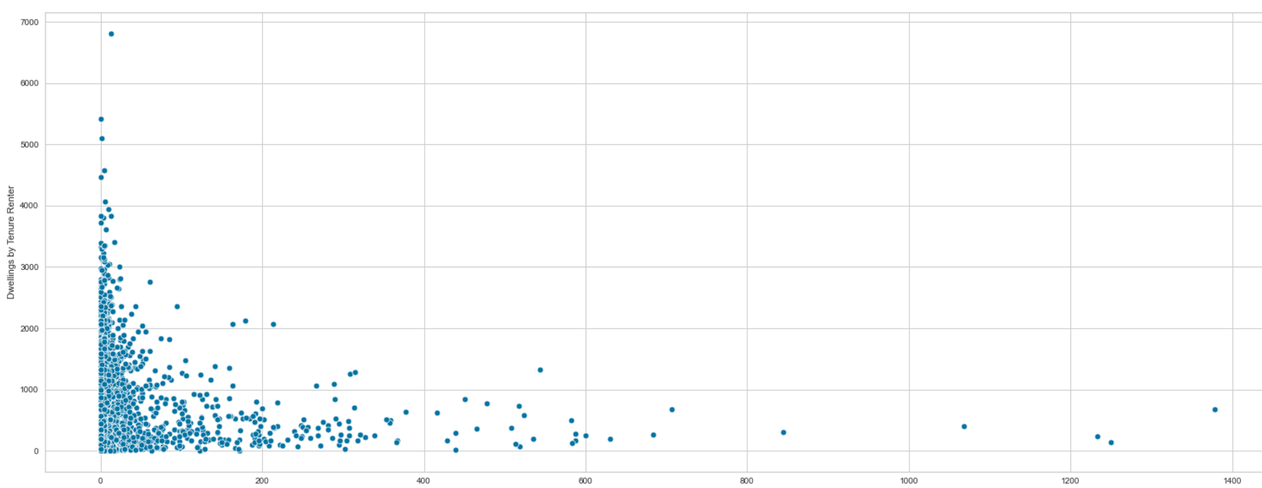






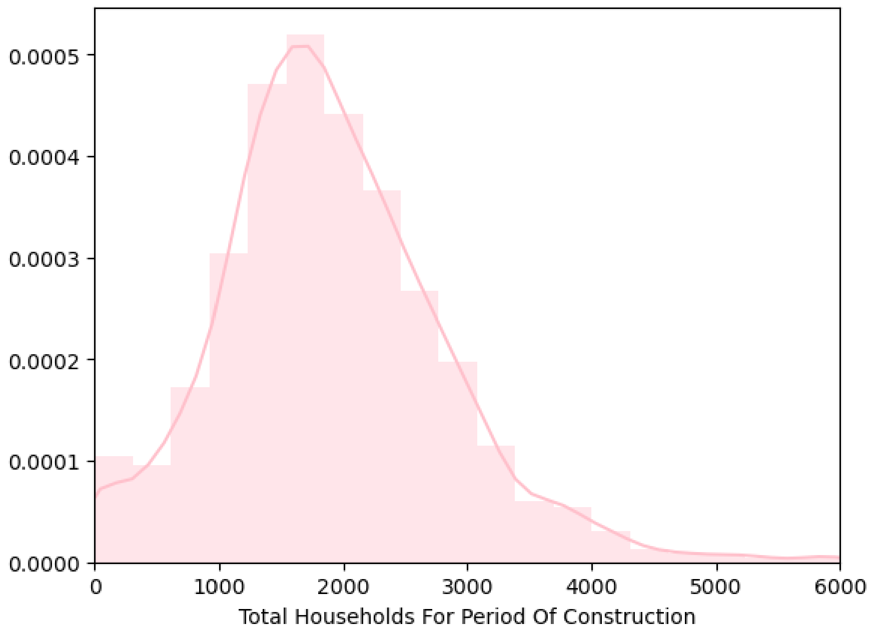

Total Households For Structure Type Other Dwelling Types
Total Households For Period Of Construction Built Between 2001 And 2005


Total Households for Tenure
Total Households For Structure Type Houses


Dwellings by Tenure Renter
Total Households For Structure Type Other Dwelling Types

In [9]:
```
plt.style.use("default")
sns.distplot(data['Total Households For Period Of Construction'], color="pink")
plt.xlim(0,6000)
```

Out[9]: (0.0, 6000.0)



Data Pre-processing

In [10]:
```
# drop equal value columns
df = data.drop(columns=['Total Households For Period Of Construction'])

# remove 20 rows with 0 income
df = df[df['Median Household Income (Current Year $)'] > 0]
df = df.reset_index(drop=True)

# rename columns
df.columns = ['population', 'households', 'income',
              'hh_before_1961','hh_1961_1980','hh_1981_1990','hh_1991_2000','hh_2001_2005',
              'hh_type_house', 'hh_type_app', 'hh_type_other',
              'hh_tenure', 'hh_tenure_owner', 'hh_tenure_renter']

# derive additional columns
df['hh_tenure_other'] = df['hh_tenure'] - df['hh_tenure_owner'] - df['hh_tenure_renter']
df['hh_after_2005'] = df['households'] - df['hh_before_1961'] - df['hh_1961_1980'] \
                      - df['hh_1981_1990'] - df['hh_1991_2000'] - df['hh_2001_2005']
df["persons_per_hh"] = df['population']/df['households']

# drop columns
df = df.drop(columns=['population'])
df = df.drop(columns=['hh_tenure'])
df = df.drop(columns=['households'])
```

In [11]:
```
df
```

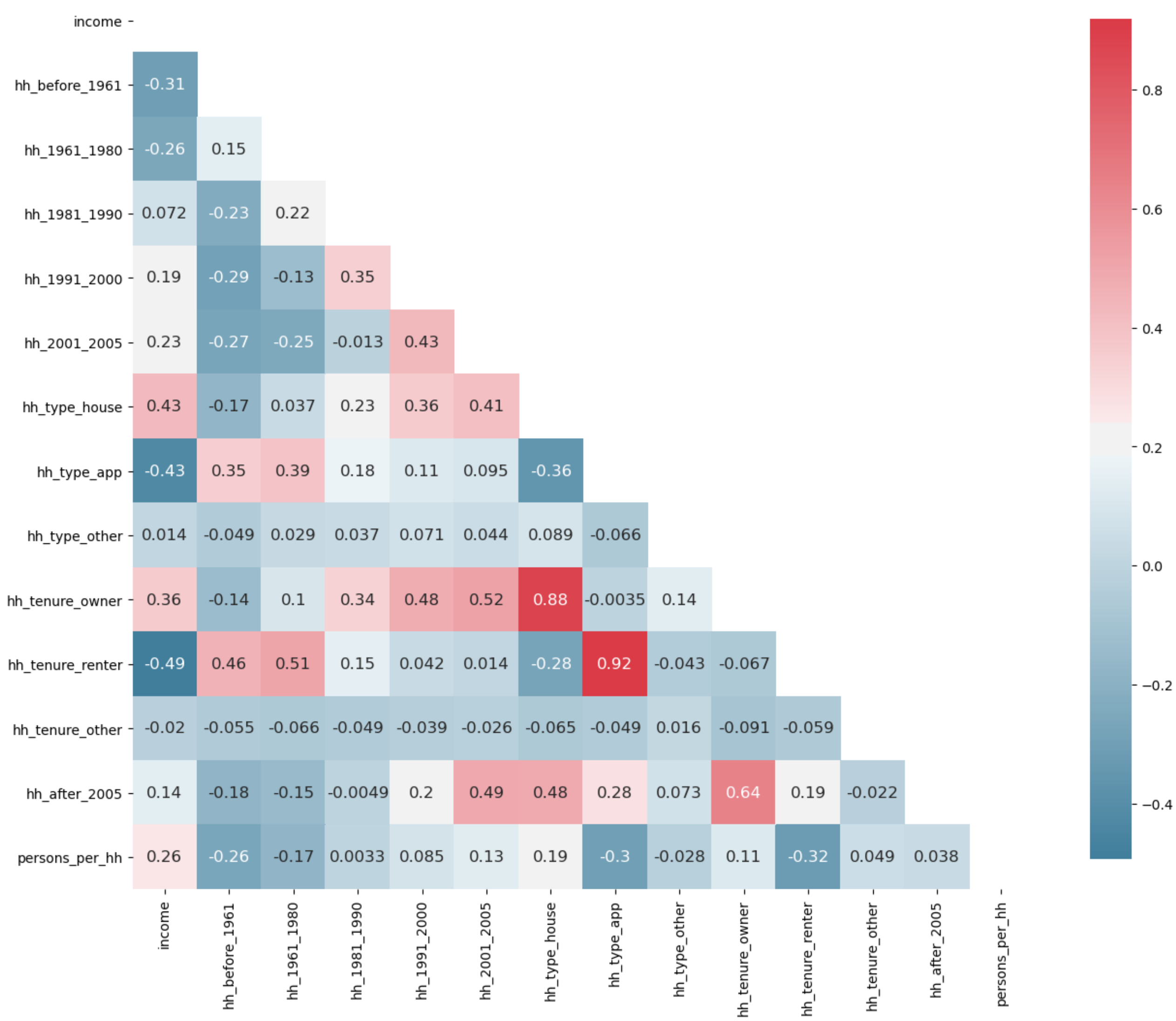
Out[11]:

	income	hh_before_1961	hh_1961_1980	hh_1981_1990	hh_1991_2000	hh_2001_2005	hh_type_house	hh_type_app	hh_type_other
0	68242.12	323	199	53	182	526	911	525	5
1	88172.37	927	70	15	3	0	792	230	4
2	103853.38	3	607	567	651	106	1418	652	1
3	82796.63	133	1695	248	79	0	1397	806	0
4	91648.22	0	7	127	938	143	914	505	0
...	...	...	...	...	...	...	...	...	...
4975	108823.38	0	3	31	501	276	926	27	0
4976	68735.64	678	986	386	359	448	2388	1436	35
4977	71370.58	164	485	511	523	29	677	1038	180
4978	58258.26	544	185	40	95	13	796	242	0
4979	111457.48	11	59	671	514	402	1751	79	0

4980 rows × 14 columns

In [12]:
```
# Correlation matrix
corr = df.corr()

plt.figure(figsize=(15,15))
sns.heatmap(corr,
            mask=np.triu(np.ones_like(corr, dtype=bool)),
            cmap=sns.diverging_palette(230, 10, as_cmap=True),
            square=True,
            cbar_kws={"shrink": 0.75},
            annot=True,
            annot_kws={'size':12})
```

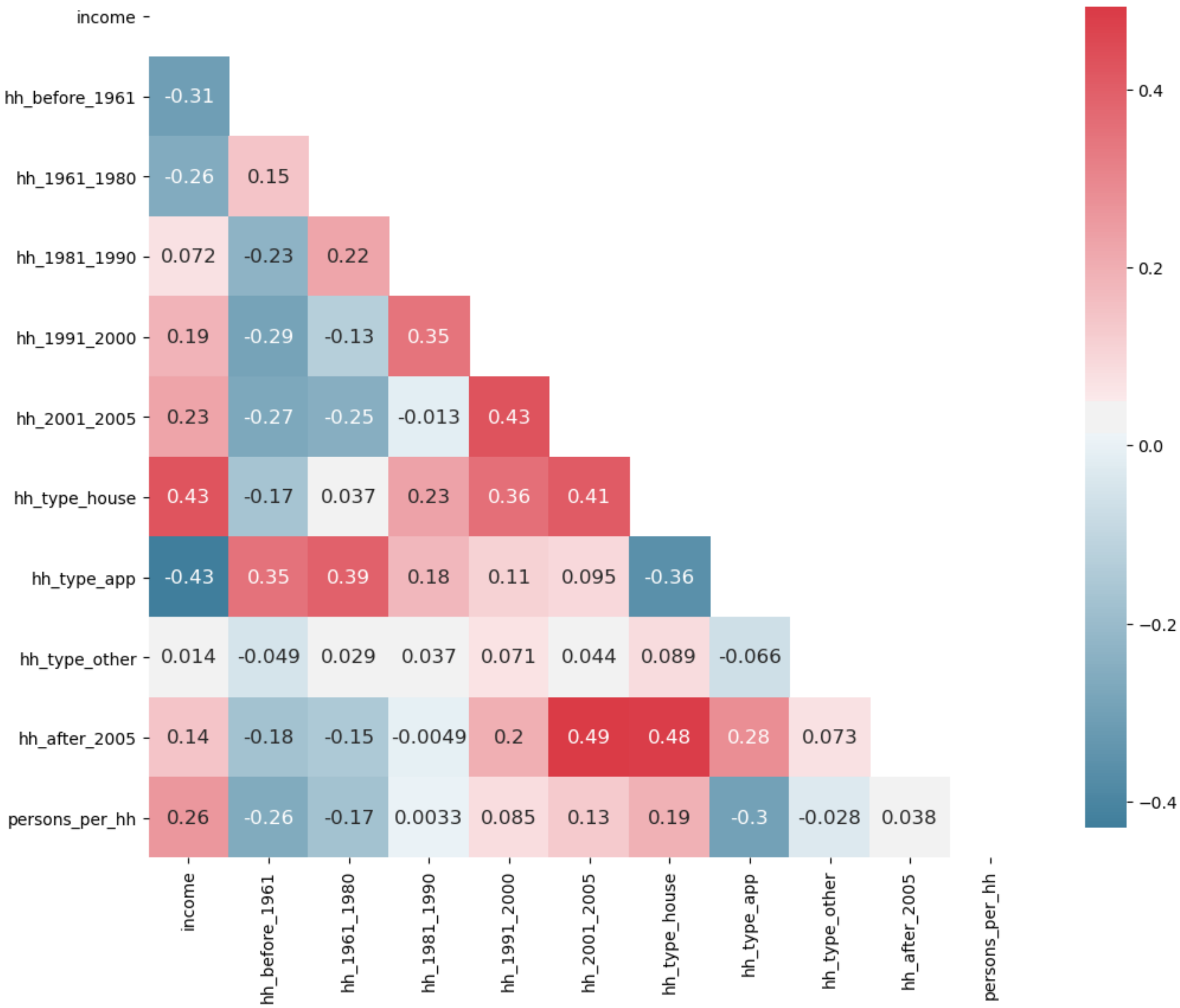
Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffecebee4f0>
```



In [13]:
```
# drop hightly correlated columns
df = df.drop(columns=['hh_tenure_owner'])
df = df.drop(columns=['hh_tenure_renter'])
df = df.drop(columns=['hh_tenure_other'])
```

In [14]:
```python
# Correlation matrix
corr = df.corr()

plt.figure(figsize=(12,12))
sns.heatmap(corr,
            mask=np.triu(np.ones_like(corr, dtype=bool)),
            cmap=sns.diverging_palette(230, 10, as_cmap=True),
            square=True,
            cbar_kws={"shrink": 0.75},
            annot=True,
            annot_kws={'size':12})
```

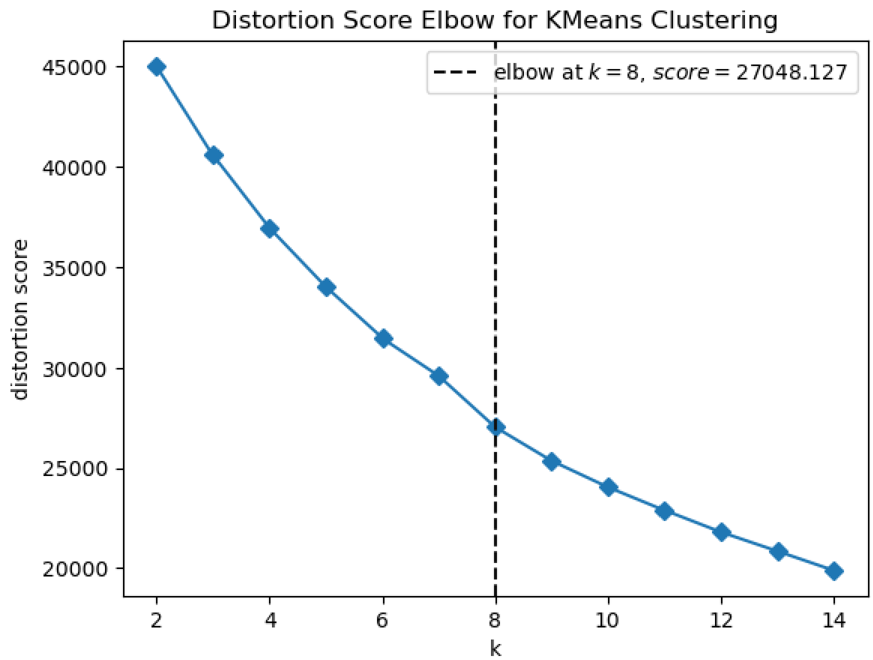
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffecf834460>



k-Means Clustering

In [15]:
```python
# scale variables
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)
```

In [16]:
```
# choose a suitable k  - instantiate the clustering model and visualizer
model = KMeans(random_state=123)
visualizer = KElbowVisualizer(model, k=(2,15), timings=False)
visualizer.fit(scaled_df)
visualizer.show()
```

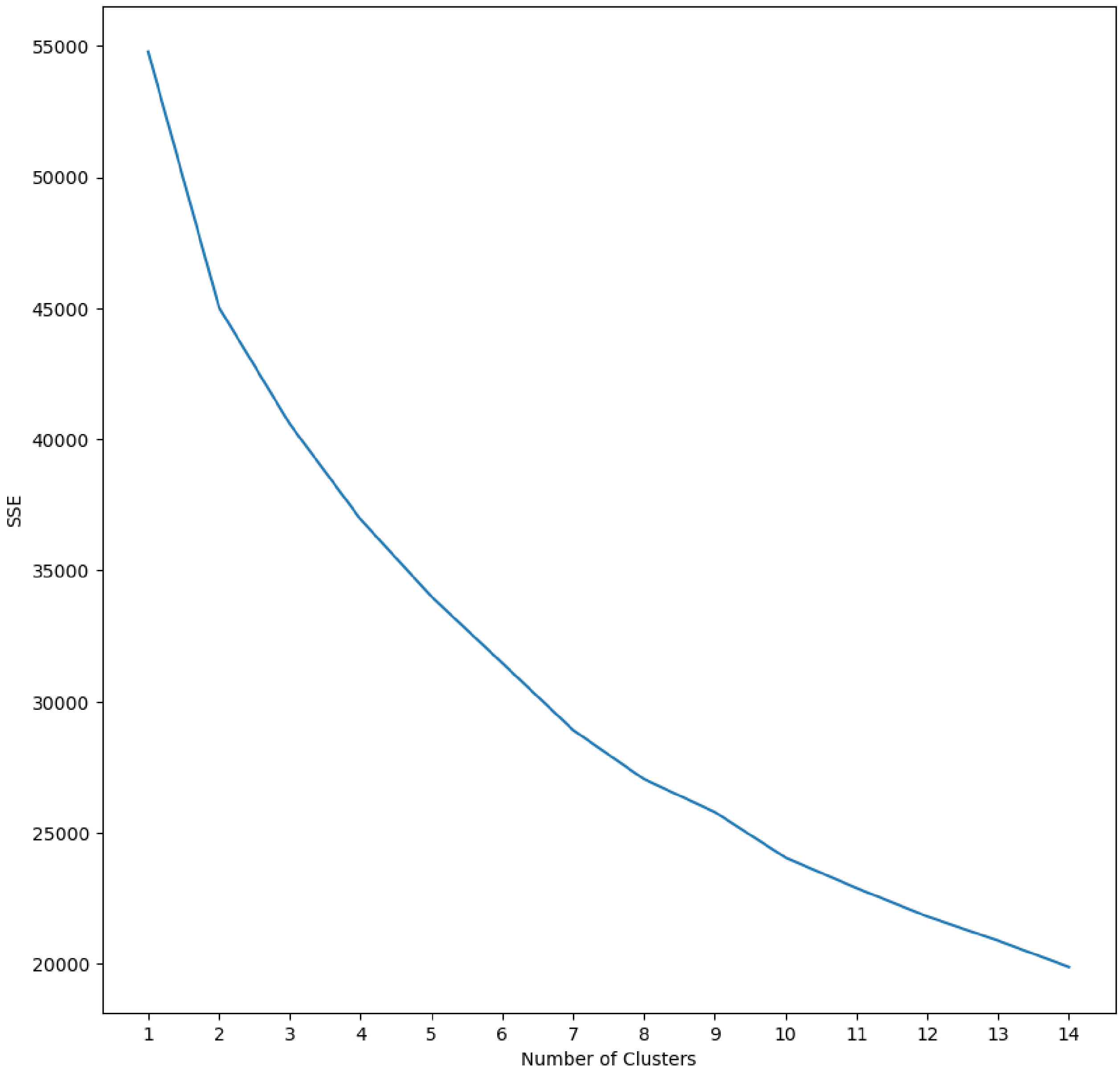


Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffecf9cfc70>

In [17]:
```
# choose a suitable k - elbow method
kmeans_kwargs = {
    "random_state": 1111,
}

sse = []
for k in range(1, 15):
    model = KMeans(n_clusters = k, **kmeans_kwargs)
    model.fit(scaled_df)
    sse.append(model.inertia_)
```

In [19]:
```
plt.figure(figsize = (10,10))
plt.plot(range(1, 15), sse)
plt.xticks(range(1, 15))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
```

Out[19]: Text(0, 0.5, 'SSE')



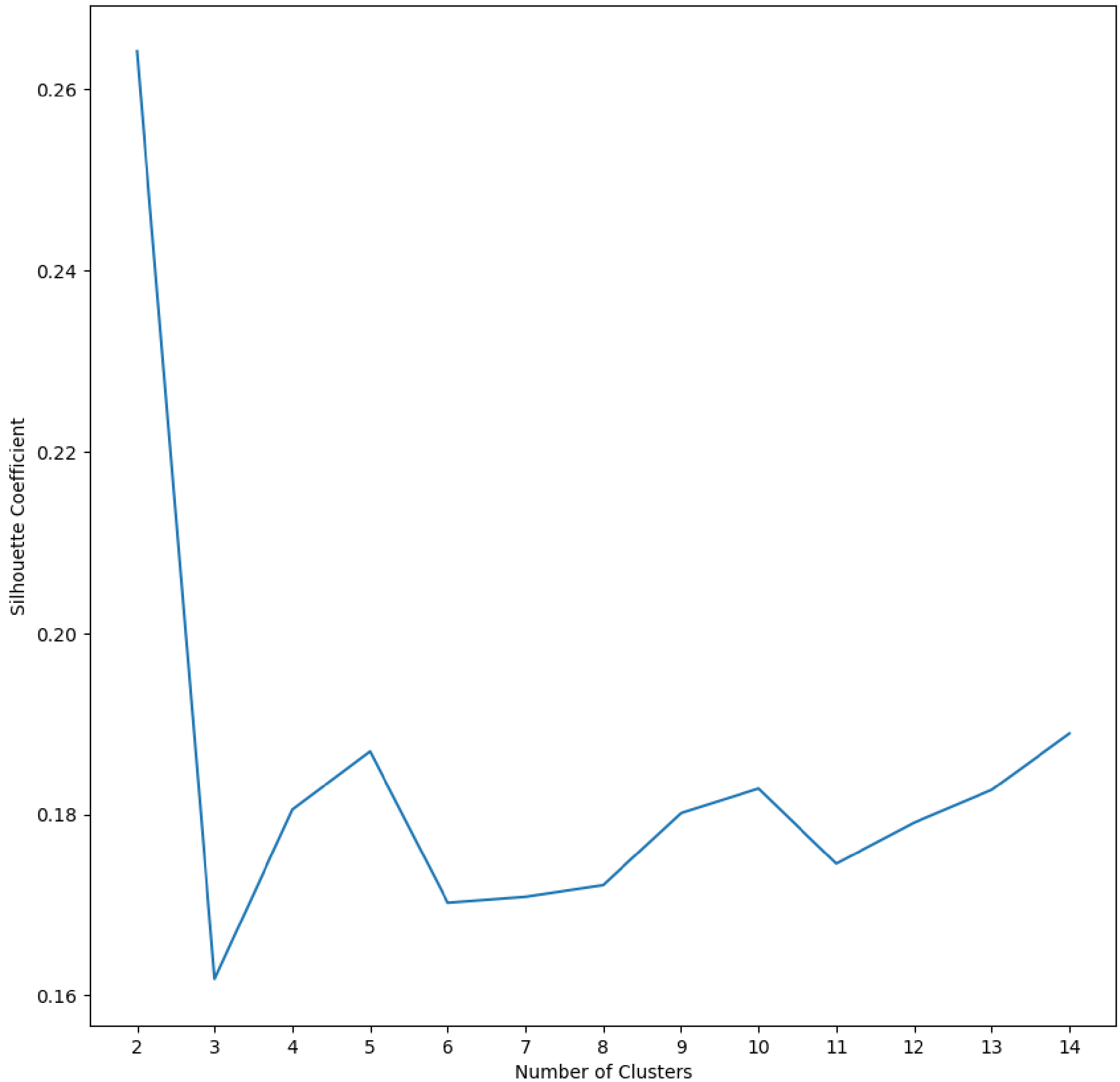
In [20]:
```
kl = KneeLocator(range(1, 15), sse, curve = "convex", direction = "decreasing")
kl.elbow
```

Out[20]: 5

In [21]:
```
# choose a suitable k - silhouette coefficient
silhouette_coefficients = []

for k in range(2, 15):
    model = KMeans(n_clusters = k, **kmeans_kwargs)
    model.fit(scaled_df)
    score = silhouette_score(scaled_df, model.labels_)
    silhouette_coefficients.append(score)
```

In [22]:
```
plt.figure(figsize = (10,10))
plt.plot(range(2, 15), silhouette_coefficients)
plt.xticks(range(2, 15))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
```

Out[22]: Text(0, 0.5, 'Silhouette Coefficient')



In [23]:
```
# final model
model = KMeans(n_clusters = 5, **kmeans_kwargs)
model.fit(scaled_df)
```

Out[23]: KMeans(n_clusters=5, random_state=1111)

In [24]:
```
labeled_data = data[data['Median Household Income (Current Year $)'] > 0]
labeled_data = labeled_data.reset_index(drop=True)
```

In [25]:
```
# add labels back to original data
labels = pd.DataFrame(model.labels_)
df['labels'] = labels
labeled_data['labels'] = labels
labeled_data
```

Out[25]:

	Total Population	Total Households	Median Household Income (Current Year $)	Total Households For Period Of Construction	Total Households For Period Of Construction Built Before 1961	Total Households For Period Of Construction Built Between 1961 And 1980	Total Households For Period Of Construction Built Between 1981 And 190	Total Households For Period Of Construction Built Between 1991 And 2000	Total Households For Period Of Construction Built Between 2001 And 2005	Tot Househol Fo Structur Typ House
0	4051	1441	68242.12	1441	323	199	53	182	526	91
1	2329	1026	88172.37	1026	927	70	15	3	0	79
2	5276	2071	103853.38	2071	3	607	567	651	106	141
3	5967	2203	82796.63	2203	133	1695	248	79	0	139
4	4236	1419	91648.22	1419	0	7	127	938	143	91
...	...	...	...	...	...	...	...	...	...	.
4975	2588	953	108823.38	953	0	3	31	501	276	92
4976	9036	3859	68735.64	3859	678	986	386	359	448	238
4977	4689	1895	71370.58	1895	164	485	511	523	29	67
4978	3673	1038	58258.26	1038	544	185	40	95	13	79
4979	6010	1830	111457.48	1830	11	59	671	514	402	175

4980 rows × 16 columns

In [26]:
```
labeled_data.groupby('labels').mean()
```

Out[26]:

labels	Total Population	Total Households	Median Household Income (Current Year $)	Total Households For Period Of Construction	Total Households For Period Of Construction Built Before 1961	Total Households For Period Of Construction Built Between 1961 And 1980	Total Households For Period Of Construction Built Between 1981 And 190	Total Households For Period Of Construction Built Between 1991 And 2000	Total Households For Period Of Construction Built Between 2001 And 2005	Ho
0	3180.061137	1317.436019	69222.866512	1317.436019	524.484360	436.758768	106.481043	80.595735	37.010427	76
1	5833.310231	2729.161716	51427.130495	2729.161716	624.809681	1057.130913	349.074807	249.870187	104.680968	67
2	5860.950000	2318.412500	82865.744000	2318.412500	173.762500	667.862500	359.250000	364.237500	188.650000	159
3	5551.864250	1991.425018	98657.546034	1991.425018	151.530206	550.063255	472.690121	390.146411	139.840085	160
4	8526.987342	2934.335443	101175.993523	2934.335443	65.052743	142.042194	140.533755	452.272152	649.544304	221

In [27]:
```
labeled_data['labels'].value_counts()
```

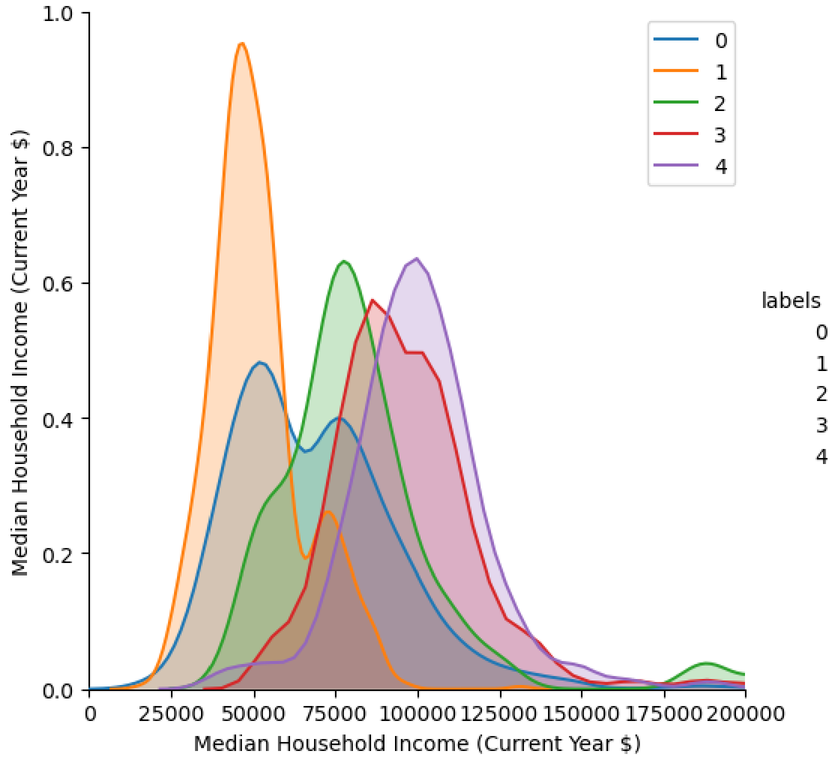
Out[27]:
```
0    2110
3    1407
1     909
4     474
2      80
Name: labels, dtype: int64
```

In [28]:
```
plt.figure(figsize=(12,8))
sns.pairplot(labeled_data[["Median Household Income (Current Year $)", "labels"]], hue="labels", size=5)
plt.xlim(0,200000)
plt.legend()
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/axisgrid.py:2071: UserWarning: The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

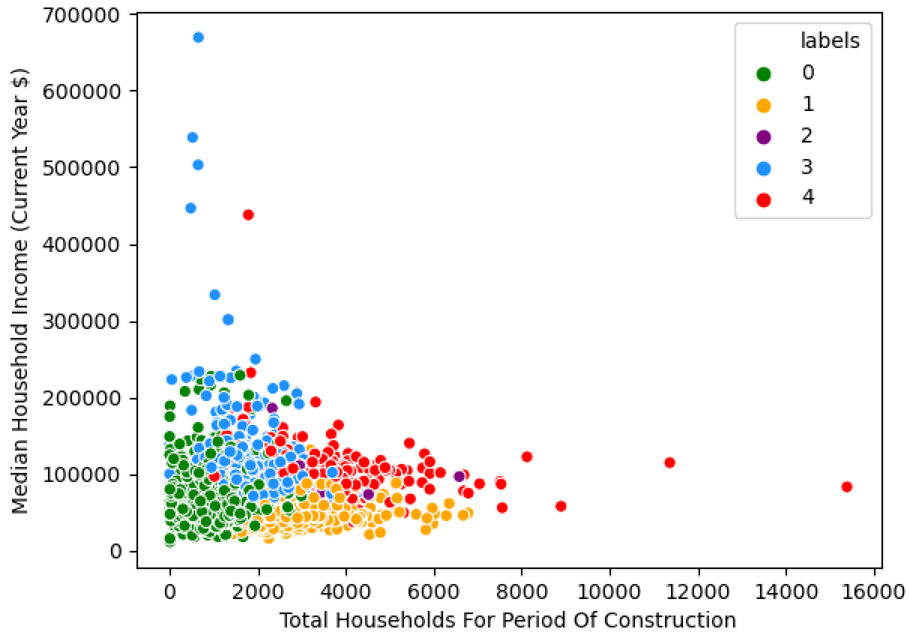
Out[28]: `<matplotlib.legend.Legend at 0x7ffeced0ee50>`

`<Figure size 1200x800 with 0 Axes>`



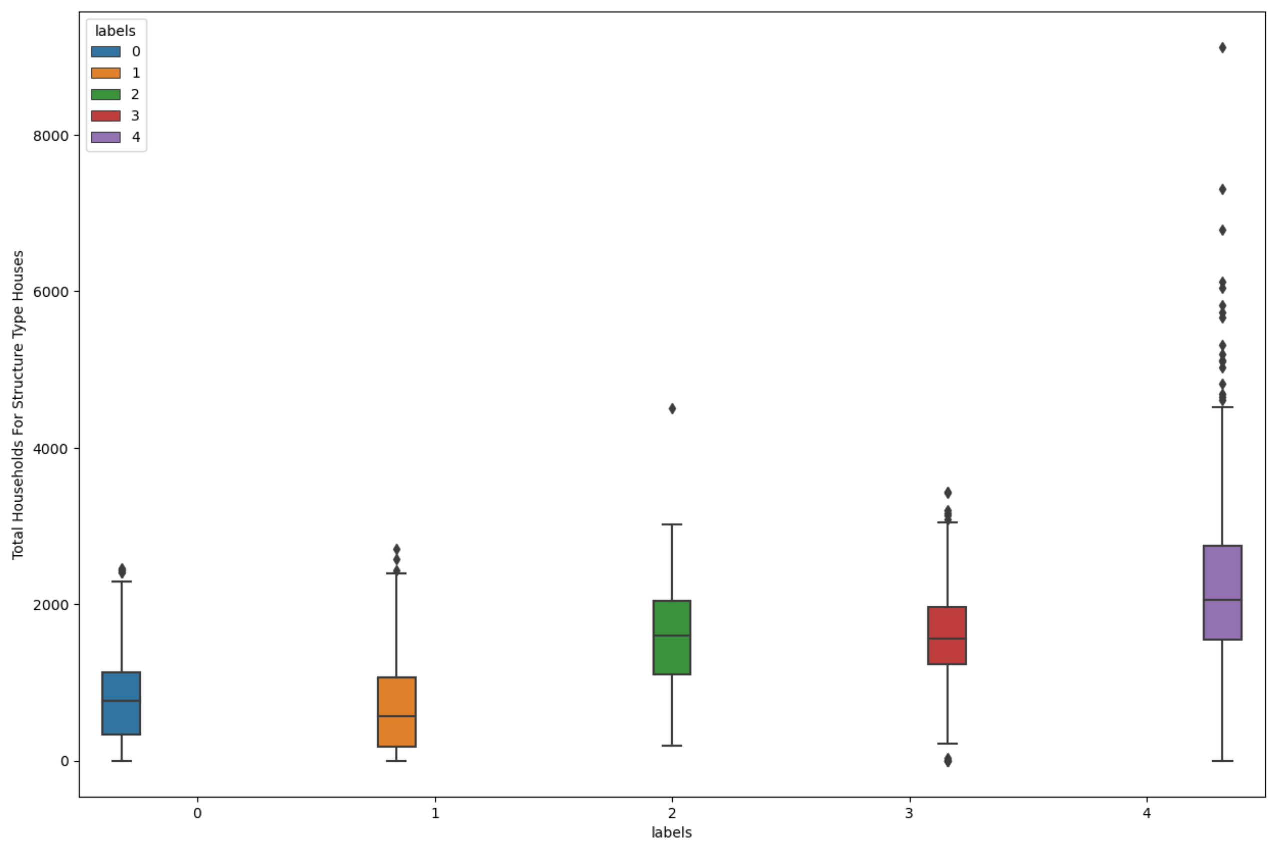
In [68]:
```
sns.scatterplot(x=labeled_data["Total Households For Period Of Construction"],
                y=labeled_data["Median Household Income (Current Year $)"],
                hue=labeled_data["labels"],
                palette=['green','orange','purple','dodgerblue','red'])
```

Out[68]: `<matplotlib.axes._subplots.AxesSubplot at 0x7ffebada1a90>`



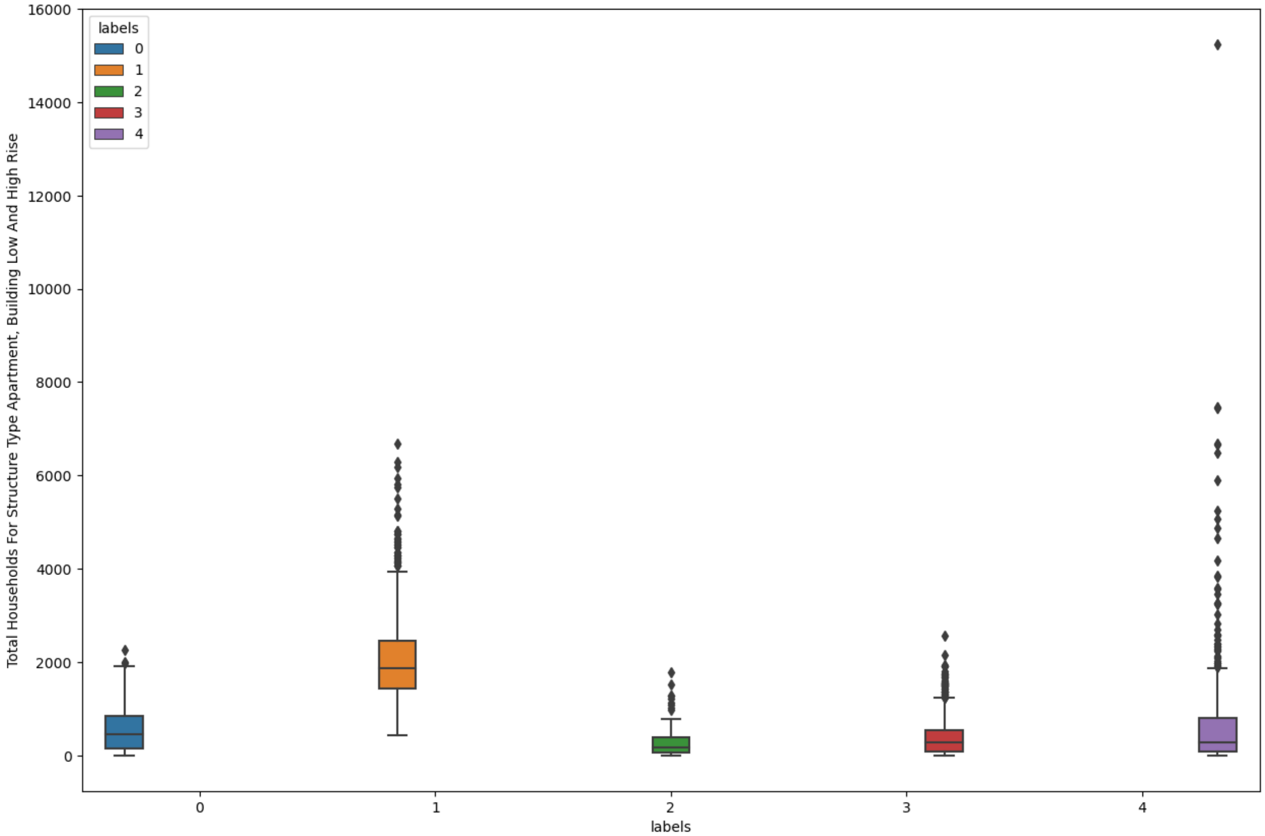
In [30]:
```
plt.figure(figsize=(15,10))
sns.boxplot(x="labels", y="Total Households For Structure Type Houses",
            hue="labels",
            data=labeled_data)
```

Out[30]: `<matplotlib.axes._subplots.AxesSubplot at 0x7ffecfe5bd30>`



In [31]:
```
plt.figure(figsize=(15,10))
sns.boxplot(x="labels", y="Total Households For Structure Type Apartment, Building Low And High Rise",
            hue="labels",
            data=labeled_data)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffeb56ddf70>



BIRCH Clustering

In [32]:
```
# model
brc = Birch(n_clusters = 5)
brc.fit(scaled_df)
```

Out[32]: Birch(n_clusters=5)

In [33]:
```
# add labels back to original data
labels = pd.DataFrame(brc.labels_)
df['brc_labels'] = labels
labeled_data['brc_labels'] = labels
labeled_data
```

Out[33]:

	Total Population	Total Households	Median Household Income (Current Year $)	Total Households For Period Of Construction	Total Households For Period Of Construction Built Before 1961	Total Households For Period Of Construction Built Between 1961 And 1980	Total Households For Period Of Construction Built Between 1981 And 190	Total Households For Period Of Construction Built Between 1991 And 2000	Total Households For Period Of Construction Built Between 2001 And 2005	Tot Househol F Structur Typ House
0	4051	1441	68242.12	1441	323	199	53	182	526	91
1	2329	1026	88172.37	1026	927	70	15	3	0	79
2	5276	2071	103853.38	2071	3	607	567	651	106	141
3	5967	2203	82796.63	2203	133	1695	248	79	0	139
4	4236	1419	91648.22	1419	0	7	127	938	143	91
...	...	...	...	...	...	...	...	...	...	
4975	2588	953	108823.38	953	0	3	31	501	276	92
4976	9036	3859	68735.64	3859	678	986	386	359	448	238
4977	4689	1895	71370.58	1895	164	485	511	523	29	67
4978	3673	1038	58258.26	1038	544	185	40	95	13	79
4979	6010	1830	111457.48	1830	11	59	671	514	402	175

4980 rows × 17 columns

In [34]:
```
labeled_data.groupby('brc_labels').mean()
```

Out[34]:

brc_labels	Total Population	Total Households	Median Household Income (Current Year $)	Total Households For Period Of Construction	Total Households For Period Of Construction Built Before 1961	Total Households For Period Of Construction Built Between 1961 And 1980	Total Households For Period Of Construction Built Between 1981 And 190	Total Households For Period Of Construction Built Between 1991 And 2000	Total Households For Period Of Construction Built Between 2001 And 2005
0	7485.835145	2478.092391	114831.495091	2478.092391	78.324275	101.079710	130.923913	507.072464	519.771739
1	4256.498260	1612.153793	81703.681886	1612.153793	251.660056	556.416145	287.885177	202.614823	88.605776
2	5091.650847	2343.185763	55256.221871	2343.185763	781.395254	724.776271	254.214237	203.937627	93.126780
3	5879.653846	2326.115385	82680.299872	2326.115385	172.884615	682.141026	356.333333	341.205128	180.115385
4	394.000000	8.000000	50000.000000	8.000000	4.000000	1.000000	0.000000	0.000000	0.000000

In [35]:
```
labeled_data['brc_labels'].value_counts()
```

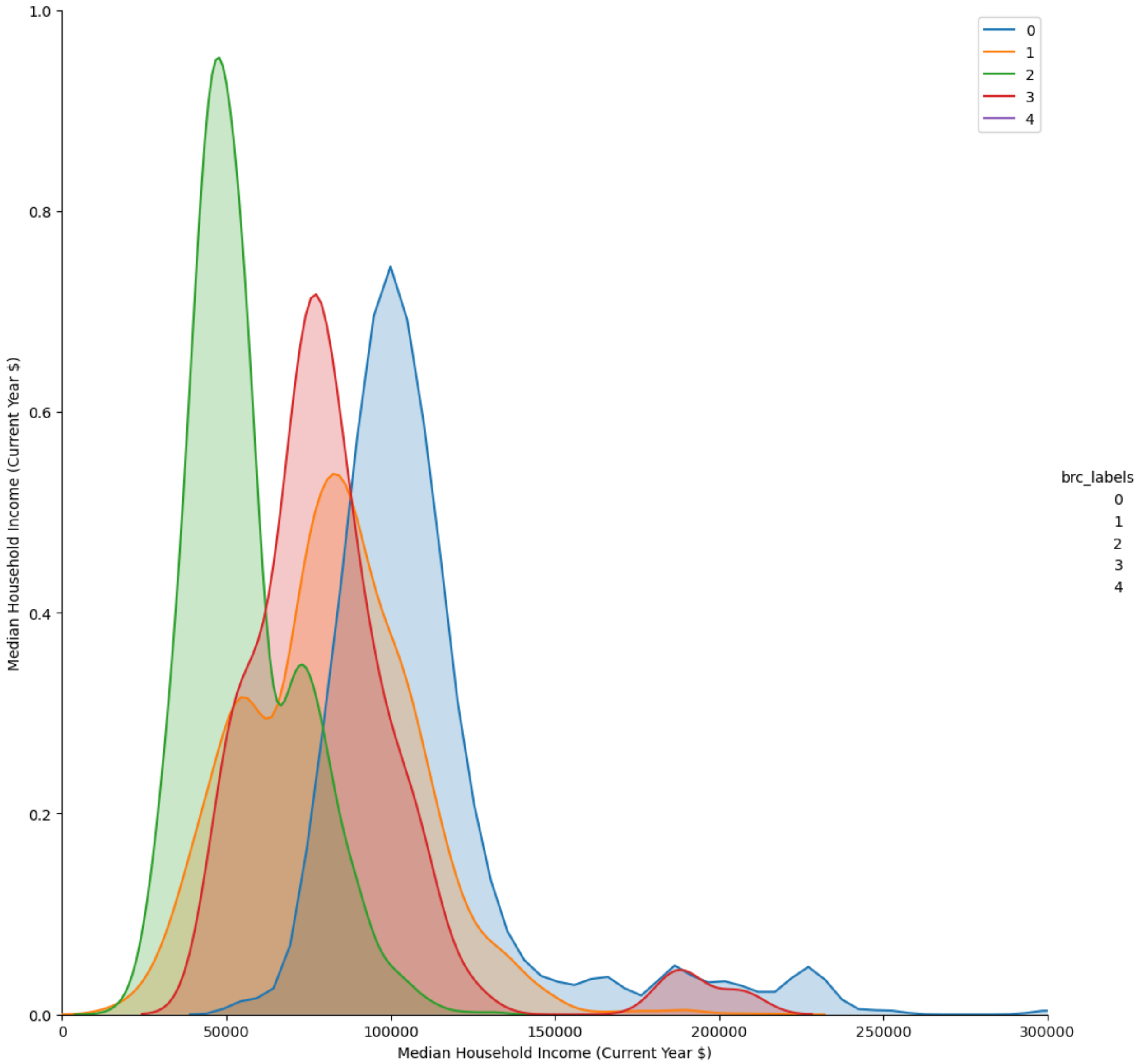
Out[35]:
```
1    2874
2    1475
0     552
3      78
4       1
Name: brc_labels, dtype: int64
```

In [51]:
```
plt.figure(figsize=(10,10))
sns.pairplot(labeled_data[["Median Household Income (Current Year $)", "brc_labels"]], hue="brc_label
s", size=10)
plt.legend()
plt.xlim(0,300000)
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/axisgrid.py:2071: UserWarning: The `size` paramet
er has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
/opt/anaconda3/lib/python3.8/site-packages/distributions.py:283: UserWarning: Data must have
variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
```

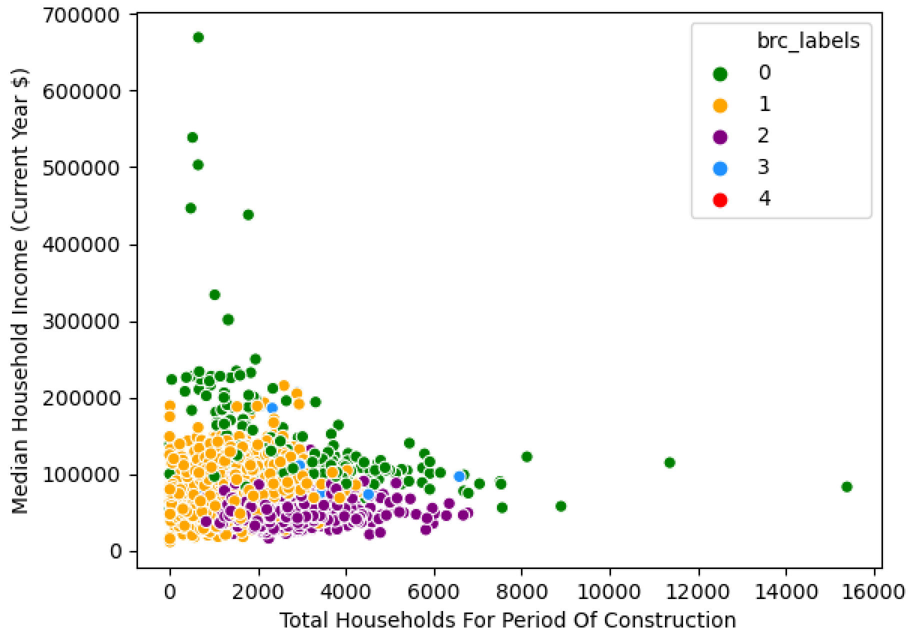
Out[51]:
```
(0.0, 300000.0)
```

```
<Figure size 1000x1000 with 0 Axes>
```



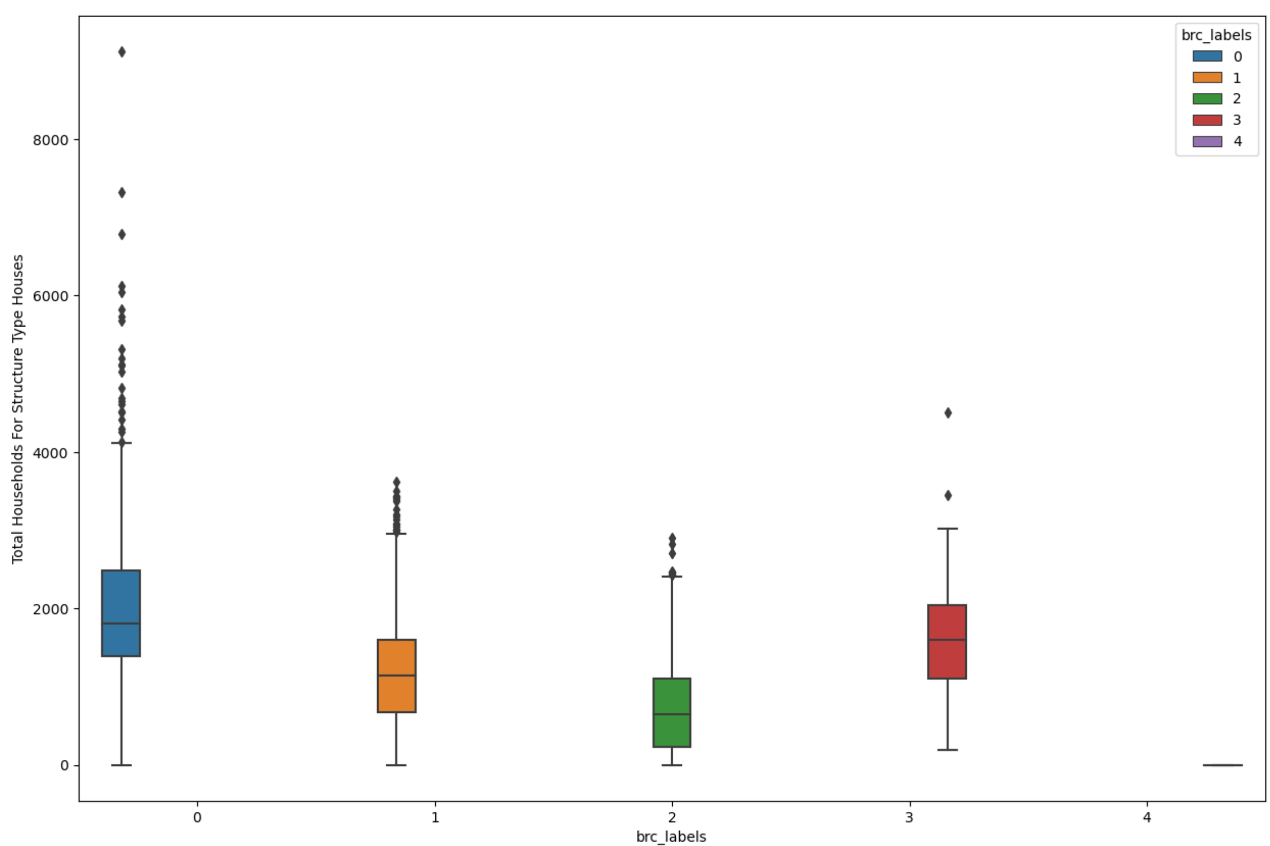
In [69]:
```
sns.scatterplot(x=labeled_data["Total Households For Period Of Construction"],
                y=labeled_data["Median Household Income (Current Year $)"],
                hue=labeled_data["brc_labels"],
                palette=['green','orange','purple','dodgerblue','red'])
```

Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffeb9df87f0>



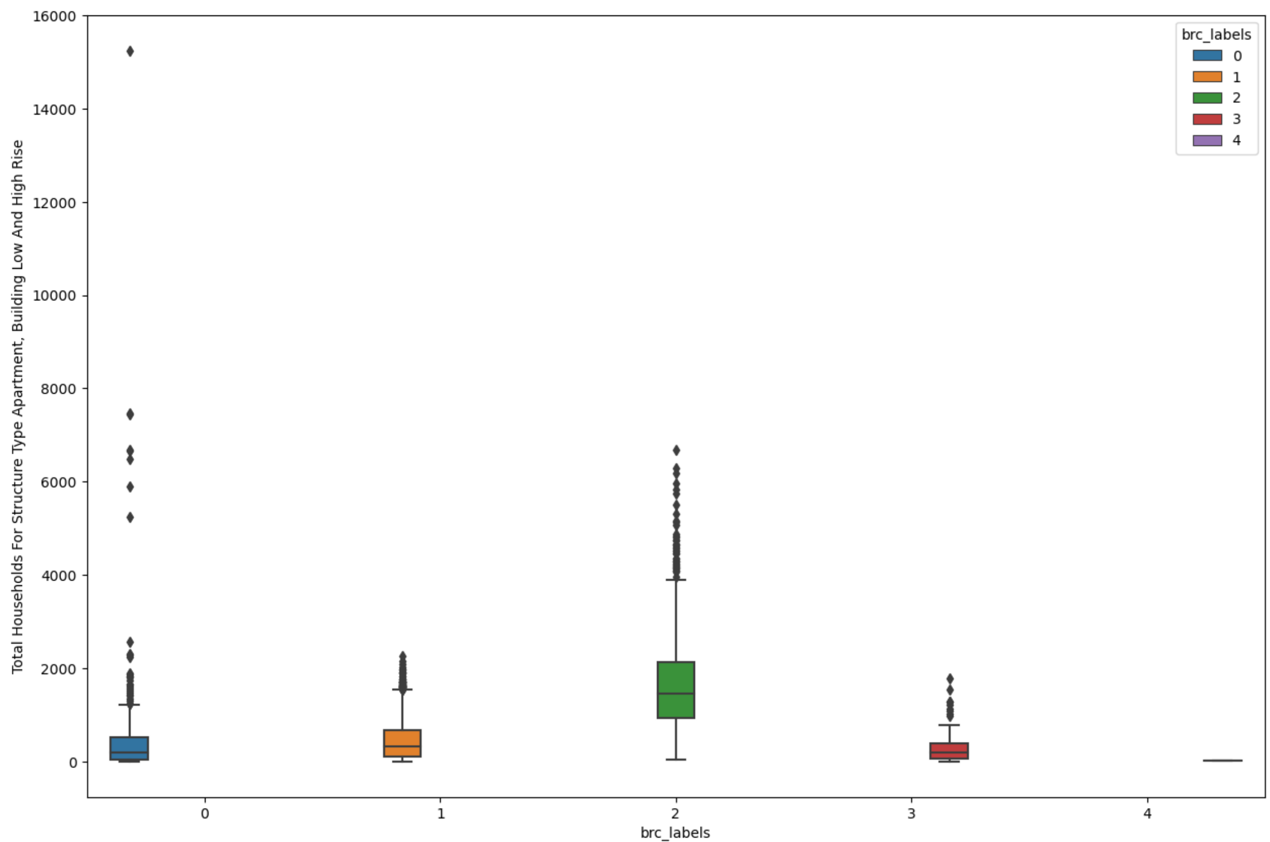
In [42]:
```
plt.figure(figsize=(15,10))
sns.boxplot(x="brc_labels", y="Total Households For Structure Type Houses",
            hue="brc_labels",
            data=labeled_data)
```

Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffeb5f37af0>



In [43]:
```
plt.figure(figsize=(15,10))
sns.boxplot(x="brc_labels", y="Total Households For Structure Type Apartment, Building Low And High Ri
se",
            hue="brc_labels",
            data=labeled_data)
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffeb781f940>



Part Two

In [44]:
```
# final model
model2 = KMeans(n_clusters = 5, **kmeans_kwargs)
X = df.drop(columns=['income','labels','brc_labels'])
scaler = StandardScaler()
scaled_df2 = scaler.fit_transform(X)
model2.fit(scaled_df2)
```

Out[44]: KMeans(n_clusters=5, random_state=1111)

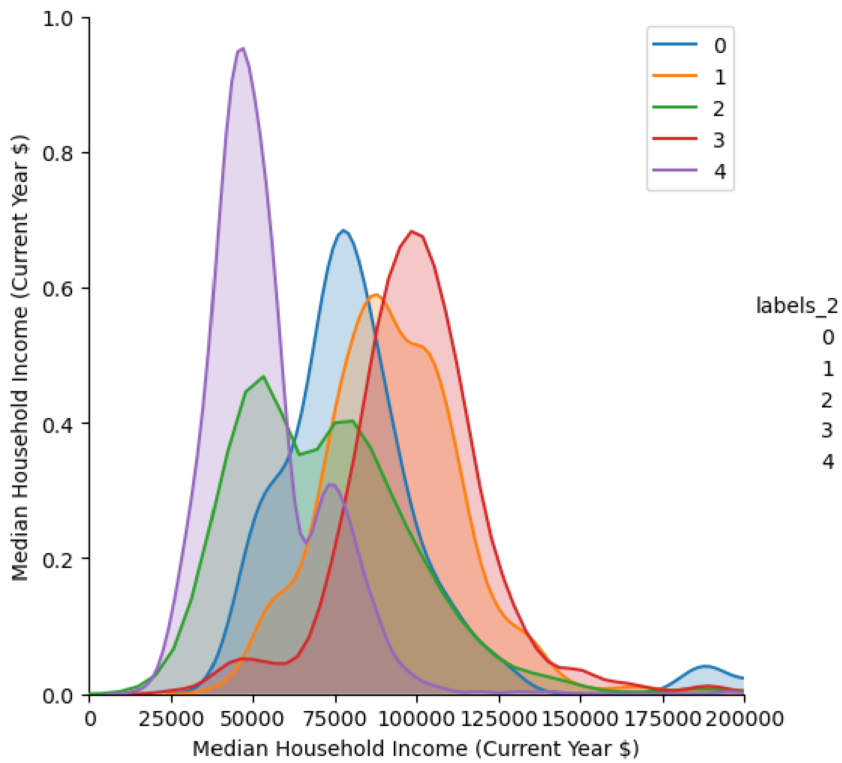
In [45]:
```
# add labels back to original data
labels = pd.DataFrame(model2.labels_)
df['labels_2'] = labels
labeled_data['labels_2'] = labels
```

In [46]:
```
plt.figure(figsize=(12,8))
sns.pairplot(labeled_data[["Median Household Income (Current Year $)", "labels_2"]], hue="labels_2", s
ize=5)
plt.xlim(0,200000)
plt.legend()
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/axisgrid.py:2071: UserWarning: The `size` paramet
er has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

Out[46]: <matplotlib.legend.Legend at 0x7ffeb6d42d00>

<Figure size 1200x800 with 0 Axes>



In [47]:
```
labeled_data.groupby('labels_2').mean()
```

Out[47]:

labels_2	Total Population	Total Households	Median Household Income (Current Year $)	Total Households For Period Of Construction	Total Households For Period Of Construction Built Before 1961	Total Households For Period Of Construction Built Between 1961 And 1980	Total Households For Period Of Construction Built Between 1981 And 190	Total Households For Period Of Construction Built Between 1991 And 2000	Total Households For Period Of Construction Built Between 2001 And 2005	I
0	5860.950000	2318.412500	82865.744000	2318.412500	173.762500	667.862500	359.250000	364.237500	188.650000	1
1	5832.127483	2106.517384	93496.181026	2106.517384	141.566225	525.937914	529.585265	448.505795	153.903146	1
2	3246.736207	1316.897845	73802.120409	1316.897845	470.934483	456.445259	117.064655	87.718103	41.961638	
3	8537.023656	2943.507527	100609.399785	2943.507527	63.718280	142.131183	138.092473	441.873118	657.913978	2
4	5863.637266	2739.582139	53614.576880	2739.582139	689.573319	1062.164278	328.699008	231.234840	97.783903	

In [48]:
```
labeled_data.groupby('labels').mean()
```

Out[48]:

labels	Total Population	Total Households	Median Household Income (Current Year $)	Total Households For Period Of Construction	Total Households For Period Of Construction Built Before 1961	Total Households For Period Of Construction Built Between 1961 And 1980	Total Households For Period Of Construction Built Between 1981 And 190	Total Households For Period Of Construction Built Between 1991 And 2000	Total Households For Period Of Construction Built Between 2001 And 2005	Ho
0	3180.061137	1317.436019	69222.866512	1317.436019	524.484360	436.758768	106.481043	80.595735	37.010427	76
1	5833.310231	2729.161716	51427.130495	2729.161716	624.809681	1057.130913	349.074807	249.870187	104.680968	67
2	5860.950000	2318.412500	82865.744000	2318.412500	173.762500	667.862500	359.250000	364.237500	188.650000	159
3	5551.864250	1991.425018	98657.546034	1991.425018	151.530206	550.063255	472.690121	390.146411	139.840085	160
4	8526.987342	2934.335443	101175.993523	2934.335443	65.052743	142.042194	140.533755	452.272152	649.544304	221

In [49]:
```
# separate clusters
cluster_1 = df.loc[df['labels_2'] == 0]
cluster_1 = cluster_1.reset_index(drop=True)
cluster_1_X = cluster_1.drop(columns=['labels','income','brc_labels','labels_2'])
cluster_1_Y = cluster_1[['income']]
X_train1, X_test1, y_train1, y_test1 = train_test_split(cluster_1_X, cluster_1_Y, test_size=0.4, rando
m_state=42)


cluster_2 = df.loc[df['labels_2'] == 1]
cluster_2 = cluster_2.reset_index(drop=True)
cluster_2_X = cluster_2.drop(columns=['labels','income','brc_labels','labels_2'])
cluster_2_Y = cluster_2[['income']]
X_train2, X_test2, y_train2, y_test2 = train_test_split(cluster_2_X, cluster_2_Y, test_size=0.4, rando
m_state=42)


cluster_3 = df.loc[df['labels_2'] == 2]
cluster_3 = cluster_3.reset_index(drop=True)
cluster_3_X = cluster_3.drop(columns=['labels','income','brc_labels','labels_2'])
cluster_3_Y = cluster_3[['income']]
X_train3, X_test3, y_train3, y_test3 = train_test_split(cluster_3_X, cluster_3_Y, test_size=0.4, rando
m_state=42)

cluster_4 = df.loc[df['labels_2'] == 3]
cluster_4 = cluster_4.reset_index(drop=True)
cluster_4_X = cluster_4.drop(columns=['labels','income','brc_labels','labels_2'])
cluster_4_Y = cluster_4[['income']]
X_train4, X_test4, y_train4, y_test4 = train_test_split(cluster_4_X, cluster_4_Y, test_size=0.4, rando
m_state=42)

cluster_5 = df.loc[df['labels_2'] == 4]
cluster_5 = cluster_5.reset_index(drop=True)
cluster_5_X = cluster_5.drop(columns=['labels','income','brc_labels','labels_2'])
cluster_5_Y = cluster_5[['income']]
X_train5, X_test5, y_train5, y_test5 = train_test_split(cluster_5_X, cluster_5_Y, test_size=0.4, rando
m_state=42)
```

In [523]:
```
# Construct some pipelines
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('clf', LinearRegression())])

pipe_knn = Pipeline([('scl', StandardScaler()),
                     ('clf', KNeighborsRegressor(n_neighbors=5))])

pipe_dt = Pipeline([('scl', StandardScaler()),
                    ('clf', DecisionTreeRegressor(random_state=0))])

pipe_svr = Pipeline([('scl', StandardScaler()),
                     ('clf', SVR())])

pipe_gbr = Pipeline([('scl', StandardScaler()),
                     ('clf', GradientBoostingRegressor(random_state=0))])
```

In [565]:
```
param_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
param_range_fl = [1.0, 0.5, 0.1]
param_gbr_range= [100,200,300]

grid_params_lr = {'clf__fit_intercept':[True,False], 'clf__copy_X':[True, False]}

grid_params_dt = [{'clf__criterion': ['mse']}]

grid_params_knn = [{'clf__n_neighbors': [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]}]

grid_params_svr = [{'clf__kernel': ['linear', 'rbf','poly'], 'clf__C': param_range}]

grid_params_gbr =[{'clf__n_estimators': param_gbr_range,
                   'clf__learning_rate': param_range_fl,
                   'clf__loss': ['ls','lad']}]
```

In [566]:
```
gs_lr = GridSearchCV(estimator=pipe_lr,
                     param_grid=grid_params_lr,
                     scoring='neg_root_mean_squared_error',
                     cv=10)

gs_svr = GridSearchCV(estimator=pipe_svr,
                      param_grid=grid_params_svr,
                      scoring='neg_root_mean_squared_error',
                      cv=10)

gs_knn = GridSearchCV(estimator=pipe_knn,
                      param_grid=grid_params_knn,
                      scoring='neg_root_mean_squared_error',
                      cv=10)

gs_dt = GridSearchCV(estimator=pipe_dt,
                     param_grid=grid_params_dt,
                     scoring='neg_root_mean_squared_error',
                     cv=10)

gs_gbr = GridSearchCV(estimator=pipe_gbr,
                      param_grid=grid_params_gbr,
                      scoring='neg_root_mean_squared_error',
                      cv=10)
```

In [567]:
```
grids = [gs_lr, gs_dt, gs_knn, gs_svr, gs_gbr]
grid_dict = {0: 'Linear Regression', 1: 'Decision Tree', 2: 'KNN', 3: 'SVR', 4:'Gradient Boosting Regr
ession'}
```

In [568]:
```
xtrain_dict = [X_train1, X_train2, X_train3, X_train4, X_train5]
ytrain_dict = [y_train1, y_train2, y_train3, y_train4, y_train5]

xtest_dict = [X_test1, X_test2, X_test3, X_test4, X_test5]
ytest_dict = [y_test1, y_test2, y_test3, y_test4, y_test5]
```

In [569]:
```
print('Performing model optimizations...')
best_rmse = 10000000000000000000000000000000000
best_clf = 0
best_gs = ''
for i in range(0,5):
    n=  int(i+1)
    print("Cluster: %i" % (n))
    for idx, gs in enumerate(grids):
        print('\nEstimator: %s' % grid_dict[idx])
        # Fit grid search
        gs.fit(xtrain_dict[i], ytrain_dict[i].values.ravel())
        # Best params
        print('Best params: %s' % gs.best_params_)
        # Predict on test data with best params
        y_pred = gs.predict(xtest_dict[i])
        rmse = math.sqrt(mean_squared_error(ytest_dict[i], y_pred))
        print('Test set RMSE score for best params: %.3f ' % (rmse))
        # Track best (highest test accuracy) model
        if rmse < best_rmse:
            best_rmse = rmse
            best_gs = gs
            best_clf = idx
    print('\nRegressor with best test set accuracy: %s' % grid_dict[best_clf])
    print('\n')
    best_rmse=10000000000000000000000000000000000
```

```
Performing model optimizations...
Cluster: 1

Estimator: Linear Regression
Best params: {'clf__copy_X': True, 'clf__fit_intercept': True}
Test set RMSE score for best params: 17404.741

Estimator: Decision Tree
Best params: {'clf__criterion': 'mse'}
Test set RMSE score for best params: 19171.941

Estimator: KNN
Best params: {'clf__n_neighbors': 17}
Test set RMSE score for best params: 17514.557

Estimator: SVR
Best params: {'clf__C': 10, 'clf__kernel': 'linear'}
Test set RMSE score for best params: 19054.725

Estimator: Gradient Boosting Regression
Best params: {'clf__learning_rate': 0.1, 'clf__loss': 'lad', 'clf__n_estimators': 300}
Test set RMSE score for best params: 16628.483

Regressor with best test set accuracy: Gradient Boosting Regression


Cluster: 2

Estimator: Linear Regression
Best params: {'clf__copy_X': True, 'clf__fit_intercept': True}
Test set RMSE score for best params: 19737.766

Estimator: Decision Tree
Best params: {'clf__criterion': 'mse'}
Test set RMSE score for best params: 34574.710

Estimator: KNN
Best params: {'clf__n_neighbors': 20}
Test set RMSE score for best params: 20954.298

Estimator: SVR
Best params: {'clf__C': 10, 'clf__kernel': 'linear'}
Test set RMSE score for best params: 22674.469

Estimator: Gradient Boosting Regression
Best params: {'clf__learning_rate': 0.1, 'clf__loss': 'lad', 'clf__n_estimators': 200}
Test set RMSE score for best params: 19351.322

Regressor with best test set accuracy: Gradient Boosting Regression


Cluster: 3

Estimator: Linear Regression
Best params: {'clf__copy_X': True, 'clf__fit_intercept': True}
Test set RMSE score for best params: 18919.312

Estimator: Decision Tree
Best params: {'clf__criterion': 'mse'}
Test set RMSE score for best params: 23474.508

Estimator: KNN
Best params: {'clf__n_neighbors': 10}
Test set RMSE score for best params: 19275.042

Estimator: SVR
Best params: {'clf__C': 10, 'clf__kernel': 'linear'}
Test set RMSE score for best params: 21909.535

Estimator: Gradient Boosting Regression
Best params: {'clf__learning_rate': 0.1, 'clf__loss': 'lad', 'clf__n_estimators': 200}
Test set RMSE score for best params: 17686.090

Regressor with best test set accuracy: Gradient Boosting Regression
```

```
Cluster: 4

Estimator: Linear Regression
Best params: {'clf__copy_X': True, 'clf__fit_intercept': True}
Test set RMSE score for best params: 20162.986

Estimator: Decision Tree
Best params: {'clf__criterion': 'mse'}
Test set RMSE score for best params: 27330.917

Estimator: KNN
Best params: {'clf__n_neighbors': 4}
Test set RMSE score for best params: 19953.755

Estimator: SVR
Best params: {'clf__C': 10, 'clf__kernel': 'linear'}
Test set RMSE score for best params: 24324.974

Estimator: Gradient Boosting Regression
Best params: {'clf__learning_rate': 0.1, 'clf__loss': 'lad', 'clf__n_estimators': 300}
Test set RMSE score for best params: 23330.710

Regressor with best test set accuracy: KNN


Cluster: 5

Estimator: Linear Regression
Best params: {'clf__copy_X': True, 'clf__fit_intercept': True}
Test set RMSE score for best params: 31145.849

Estimator: Decision Tree
Best params: {'clf__criterion': 'mse'}
Test set RMSE score for best params: 39710.209

Estimator: KNN
Best params: {'clf__n_neighbors': 20}
Test set RMSE score for best params: 29233.750

Estimator: SVR
Best params: {'clf__C': 10, 'clf__kernel': 'linear'}
Test set RMSE score for best params: 33017.427

Estimator: Gradient Boosting Regression
Best params: {'clf__learning_rate': 0.1, 'clf__loss': 'lad', 'clf__n_estimators': 200}
Test set RMSE score for best params: 30056.512

Regressor with best test set accuracy: KNN
```

Test Set Data

In [1076]:
```
df_test = pd.read_csv('CensusCanada2016Test.csv')
```

In [1077]:
```
# drop equal value columns
df_test = df_test.drop(columns=['Total Households For Period Of Construction'])

# rename columns
df_test.columns=['population', 'households',
             'hh_before_1961','hh_1961_1980','hh_1981_1990','hh_1991_2000','hh_2001_2005',
             'hh_type_house', 'hh_type_app', 'hh_type_other',
             'hh_tenure', 'hh_tenure_owner', 'hh_tenure_renter']

# derive additional columns
df_test['hh_tenure_other'] = df_test['hh_tenure'] - df_test['hh_tenure_owner'] - df_test['hh_tenure_r
enter']
df_test['hh_after_2005'] = df_test['households'] - df_test['hh_before_1961'] - df_test['hh_1961_1980'
] \
                      - df_test['hh_1981_1990'] - df_test['hh_1991_2000'] - df_test['hh_2001_2005']

df_test["persons_per_hh"]=df_test['population']/df_test['households']

for i in range(0,721):
    if df_test['households'][i]>0:

        df_test["persons_per_hh"][i] = df_test['population'][i]/df_test['households'][i]
    else:
        df_test["persons_per_hh"][i]=0

# drop columns
df_test = df_test.drop(columns=['population'])
df_test = df_test.drop(columns=['hh_tenure'])
df_test = df_test.drop(columns=['households'])
```

```
C:\Users\buttb\Anaconda3\lib\site-packages\ipykernel_launcher.py:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
C:\Users\buttb\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
```

In [1078]:
```
df_test = df_test.drop(columns=['hh_tenure_owner'])
df_test = df_test.drop(columns=['hh_tenure_renter'])
df_test = df_test.drop(columns=['hh_tenure_other'])
df_test
```

Out[1078]:

	hh_before_1961	hh_1961_1980	hh_1981_1990	hh_1991_2000	hh_2001_2005	hh_type_house	hh_type_app	hh_type_other	hh_after_20
0	15	21	46	648	114	883	10	0	[illegible]
1	17	839	218	27	33	1025	486	0	3
2	767	615	223	435	166	1390	1378	0	5
3	1540	969	437	244	457	2102	2461	14	9
4	44	94	34	115	184	1069	98	0	6
...	...	...	...	...	...	...	...	...	
716	0	32	14	12	0	55	0	9	
717	799	575	186	114	25	152	1561	5	[illegible]
718	360	101	3	11	0	481	20	0	[illegible]
719	227	553	408	91	19	1143	236	0	[illegible]
720	50	1726	301	107	57	1881	518	5	1

721 rows × 10 columns

In [1079]:
```
# check for null values
df_test.isnull().sum()
```

Out[1079]:
```
hh_before_1961     0
hh_1961_1980       0
hh_1981_1990       0
hh_1991_2000       0
hh_2001_2005       0
hh_type_house      0
hh_type_app        0
hh_type_other      0
hh_after_2005      0
persons_per_hh     0
dtype: int64
```

In [1080]:
```
# final model
test_pred = model2.predict(df_test) #labels test data
```

In [1081]:
```
# append the cluster label to dataframe
labels = pd.DataFrame(model2.labels_)
df_test['cluster'] = labels
df_test
```

Out[1081]:

	hh_before_1961	hh_1961_1980	hh_1981_1990	hh_1991_2000	hh_2001_2005	hh_type_house	hh_type_app	hh_type_other	hh_after_20
0	15	21	46	648	114	883	10	0	[illegible]
1	17	839	218	27	33	1025	486	0	3
2	767	615	223	435	166	1390	1378	0	5
3	1540	969	437	244	457	2102	2461	14	9
4	44	94	34	115	184	1069	98	0	6
...	...	...	...	...	...	...	...	...	
716	0	32	14	12	0	55	0	9	
717	799	575	186	114	25	152	1561	5	[illegible]
718	360	101	3	11	0	481	20	0	[illegible]
719	227	553	408	91	19	1143	236	0	[illegible]
720	50	1726	301	107	57	1881	518	5	1

721 rows × 11 columns

In [1082]:
```
# create sliced dataframes for each cluster
cluster1_test = df_test[df_test["cluster"]==0].drop(columns=["cluster"])
cluster2_test = df_test[df_test["cluster"]==1].drop(columns=["cluster"])
cluster3_test = df_test[df_test["cluster"]==2].drop(columns=["cluster"])
cluster4_test = df_test[df_test["cluster"]==3].drop(columns=["cluster"])
cluster5_test = df_test[df_test["cluster"]==4].drop(columns=["cluster"])
```

In [995]:
```
# Cluster 1
cluster1bestparam = [{'clf__learning_rate': [0.1], 'clf__loss': ['lad'], 'clf__n_estimators': [300]}]

# Cluster 2
cluster2bestparam = [{'clf__learning_rate': [0.1], 'clf__loss':[ 'lad'], 'clf__n_estimators':[200]}]

# Cluster 3
cluster3bestparam = [{'clf__learning_rate': [0.1], 'clf__loss':[ 'lad'], 'clf__n_estimators':[200]}]

# Cluster 4
cluster4bestparam = [{'clf__n_neighbors': [4]}]

# Cluster 5
cluster5bestparam = [{'clf__n_neighbors': [20]}]
```

In [732]:
```
# Run Each Model Using Grid Search with Optimial Parameters
cluster1_model = GridSearchCV(estimator=pipe_gbr,
                              param_grid=cluster1bestparam,
                              scoring='neg_root_mean_squared_error',
                              cv=10)

cluster2_model = GridSearchCV(estimator=pipe_gbr,
                              param_grid=cluster2bestparam,
                              scoring='neg_root_mean_squared_error',
                              cv=10)

cluster3_model = GridSearchCV(estimator=pipe_gbr,
                              param_grid=cluster3bestparam,
                              scoring='neg_root_mean_squared_error',
                              cv=10)

cluster4_model = GridSearchCV(estimator=pipe_knn,
                              param_grid=cluster4bestparam,
                              scoring='neg_root_mean_squared_error',
                              cv=10)

cluster5_model = GridSearchCV(estimator=pipe_knn,
                              param_grid=cluster5bestparam,
                              scoring='neg_root_mean_squared_error',
                              cv=10)
```

In [733]:
```
cluster1_model.fit(xtrain_dict[0], ytrain_dict[0].values.ravel())
```

Out[733]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('scl', StandardScaler()),
                                       ('clf',
                                        GradientBoostingRegressor(random_state=0))]),
             param_grid=[{'clf__learning_rate': [0.1], 'clf__loss': ['lad'],
                          'clf__n_estimators': [300]}],
             scoring='neg_root_mean_squared_error')
```

In [734]:
```
cluster2_model.fit(xtrain_dict[1], ytrain_dict[1].values.ravel())
```

Out[734]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('scl', StandardScaler()),
                                       ('clf',
                                        GradientBoostingRegressor(random_state=0))]),
             param_grid=[{'clf__learning_rate': [0.1], 'clf__loss': ['lad'],
                          'clf__n_estimators': [200]}],
             scoring='neg_root_mean_squared_error')
```

In [735]:
```
cluster3_model.fit(xtrain_dict[2], ytrain_dict[2].values.ravel())
```

Out[735]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('scl', StandardScaler()),
                                       ('clf',
                                        GradientBoostingRegressor(random_state=0))]),
             param_grid=[{'clf__learning_rate': [0.1], 'clf__loss': ['lad'],
                          'clf__n_estimators': [200]}],
             scoring='neg_root_mean_squared_error')
```

In [736]:
```
cluster4_model.fit(xtrain_dict[3], ytrain_dict[3].values.ravel())
```

Out[736]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('scl', StandardScaler()),
                                       ('clf', KNeighborsRegressor())]),
             param_grid=[{'clf__n_neighbors': [4]}],
             scoring='neg_root_mean_squared_error')
```

In [737]:
```
cluster5_model.fit(xtrain_dict[4], ytrain_dict[4].values.ravel())
```

Out[737]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('scl', StandardScaler()),
                                       ('clf', KNeighborsRegressor())]),
             param_grid=[{'clf__n_neighbors': [20]}],
             scoring='neg_root_mean_squared_error')
```

In [996]:
```
# Predict Income from Test Data
test_pred1 = cluster1_model.predict(cluster1_test)
test_pred2 = cluster1_model.predict(cluster2_test)
test_pred3 = cluster1_model.predict(cluster3_test)
test_pred4 = cluster1_model.predict(cluster4_test)
test_pred5 = cluster1_model.predict(cluster5_test)
```

In [1083]:
```
# Create series based of predictions
cluster1_test["predictions"] = test_pred1
cluster2_test["predictions"] = test_pred2
cluster3_test["predictions"] = test_pred3
cluster4_test["predictions"] = test_pred4
cluster5_test["predictions"] = test_pred5
```

In [1084]:
```
# create temporary predictions variables
df_test["predictions1"] = cluster1_test["predictions"]
df_test["predictions2"] = cluster2_test["predictions"]
df_test["predictions3"] = cluster3_test["predictions"]
df_test["predictions4"] = cluster4_test["predictions"]
df_test["predictions5"] = cluster5_test["predictions"]
```

In [1085]:
```
# fill null values with 0
df_test.fillna(0, inplace=True)
```

In [1070]:
```
# Create final prediction column
df_test['income_pred'] = []
```

In [1087]:
```
# Add all the columns up
# should only ad itself to 0 as null values were based on index numbers
df_test['income_pred'] = df_test["predictions1"] + df_test["predictions2"] + df_test["predictions3"]
+ df_test["predictions4"] + df_test["predictions5"]
```

In [1113]:
```
df_test.columns
```

Out[1113]:
```
Index(['hh_before_1961', 'hh_1961_1980', 'hh_1981_1990', 'hh_1991_2000',
       'hh_2001_2005', 'hh_type_house', 'hh_type_app', 'hh_type_other',
       'hh_after_2005', 'persons_per_hh', 'cluster', 'predictions1',
       'predictions2', 'predictions3', 'predictions4', 'predictions5',
       'income_pred'],
      dtype='object')
```

Save Predictions As Array in Text File

In [1091]:
```
final_pred = df_test["income_pred"]
```

In [1093]:
```
with open('Team7Predictions.txt', 'wb') as f:
    np.save(f, final_pred, allow_pickle=False)
```

In [1094]:
```
with open('Team7Predictions.txt', 'rb') as f:
    test_set_predictions = np.load(f)
```

In [1095]:
```
test_set_predictions
```

```
Out[1095]: array([63847.23919571, 75715.91618319, 68044.03709325, 56622.44970382,
       72683.70995762, 49296.97776449, 79019.3465423 , 63457.42148888,
       81966.31353318, 85661.97093091, 61268.2480181 , 88758.30194351,
       69199.98081459, 82661.68496392, 69805.62239174, 74082.58238851,
       66460.2519266 , 82585.35173191, 84317.74166464, 56377.52310396,
       58711.51411911, 74225.74377752, 74554.58493001, 60925.27050366,
       72304.43658128, 57224.31959963, 68222.70118005, 57224.31959963,
       78888.13974899, 60884.3620511 , 76425.75244567, 45099.82024986,
       53424.70017379, 62829.64121959, 68040.97856012, 75136.55091743,
       76476.35937779, 71331.44616949, 85416.78183384, 62724.41504603,
       80499.51999803, 54859.21291356, 80120.18730668, 54254.92194037,
       58059.53896542, 71282.4908152 , 48287.55541326, 54928.54983194,
       40790.20813674, 49728.84007551, 69598.71561688, 79011.39233303,
       62258.52994211, 67160.60984415, 44392.95287391, 46975.91668106,
       83497.57566459, 53288.37032515, 72146.35033864, 59303.34113735,
       85136.20525634, 68607.25980484, 70655.74676476, 66926.45111718,
       68150.97297057, 85443.22634673, 67146.2082814 , 64827.39794095,
       65505.05414254, 76518.6558295 , 61864.95225417, 59957.63998619,
       85369.68751341, 70021.10398708, 80000.18032694, 58212.05204126,
       46100.79246368, 74685.82753556, 46917.12324998, 64866.12373149,
       70730.4822077 , 60438.00944967, 57224.31959963, 76710.80755383,
       42578.64095067, 72580.779922  , 60740.87066673, 80999.1612226 ,
       61293.00705421, 83956.35675028, 45600.75779536, 71957.15460176,
       55284.82293223, 69539.39410833, 59415.10882241, 87631.10974098,
       88298.92862565, 43785.85495322, 67521.64172743, 52886.29042825,
       76538.84285104, 84056.46217507, 57395.90345546, 76552.55565116,
       75691.38537579, 74425.62452055, 62650.18489964, 55984.29685803,
       44159.79528444, 61494.31954959, 68255.91707479, 48698.21482033,
       55513.14870891, 62714.57333449, 81370.77054533, 79769.38064341,
       60973.8594949 , 76313.30985266, 56106.68854527, 69806.48304463,
       58758.88193692, 49359.34577591, 59006.9228262 , 42686.47439718,
       50383.78577329, 50605.14070324, 60417.32805546, 62101.43356257,
       70807.13857575, 53184.68810328, 66496.57117144, 81997.19322438,
       65446.68921598, 69387.05164679, 60623.40600946, 83639.99452505,
       71353.51776914, 77460.43720825, 45906.88751888, 38229.38661346,
       73324.94722839, 58209.32284401, 59602.45370678, 62850.31257226,
       56548.56312113, 48389.73005135, 59417.23305332, 65393.07123293,
       64883.92583793, 58711.51411911, 46249.11044914, 75788.70510552,
       46740.77068443, 61942.58164925, 70415.97175457, 64320.69327217,
       74522.31938915, 52817.76567307, 76227.44860517, 80470.91225355,
       65158.14187478, 70020.15785367, 44291.48767641, 61082.37896256,
       56842.17970071, 54889.58606468, 75678.48214034, 77039.56271063,
       46238.2629707 , 49991.2913403 , 71597.38015117, 41669.82670681,
       76646.57147458, 85426.79286815, 79403.46574798, 77513.01867701,
       56039.05454711, 48165.49852406, 72362.14479123, 70082.41065805,
       47633.08656007, 84269.90724164, 57751.10460952, 44674.78753999,
       59494.91774355, 72158.47311183, 58262.97230797, 77333.39581319,
       72498.33631656, 67142.37153467, 75214.58701025, 62719.00757388,
       71392.30524402, 64667.22889817, 56695.47700798, 76742.76444589,
       53725.00832299, 65227.57129913, 57099.59516521, 56132.1126438 ,
       71545.90567325, 75059.52252417, 52388.58258653, 44405.2337351 ,
       58797.65929809, 62767.22336681, 90211.50933361, 68936.89740723,
       73359.72596827, 76298.67956018, 65373.83636754, 77134.86000736,
       49721.6887218 , 68362.33648062, 71985.93720377, 77859.81998098,
       49135.07579024, 65580.33779007, 60137.41023279, 55031.63056183,
       62894.59711858, 54575.30653535, 82339.56257101, 58938.58403168,
       72745.50073655, 79097.4138935 , 64615.60646596, 71285.07986042,
       47772.16590536, 57224.31959963, 49110.6411467 , 46145.08934953,
       82616.61148068, 59140.52893558, 63874.21640052, 66173.28331473,
       62796.12756651, 61993.88738249, 78603.52379556, 79864.29278933,
       74651.85425186, 70010.53226473, 79328.26582924, 84889.46800865,
       77964.93257614, 79939.44340925, 41339.33198132, 84391.18212368,
       73012.7666129 , 84822.98571036, 65802.18071826, 62330.61433053,
       67719.00144832, 50331.26670541, 76569.21032517, 78370.53138562,
       87416.40325694, 72560.49749095, 92888.49808448, 84318.80208211,
       51150.98138526, 72332.23585804, 50611.09468317, 80753.04265784,
       56657.41303634, 72188.81956084, 57025.24767752, 50429.14949844,
       87384.14846336, 71472.42475671, 64844.46476986, 72198.19381877,
       69502.20099332, 41644.94419916, 55235.3764789 , 55182.09266385,
       84398.50878016, 80882.28545846, 58444.3011447 , 80588.33813747,
       66699.67225593, 47096.62524774, 65069.08391215, 80205.14479195,
       72775.95377693, 56207.89192514, 80642.13528773, 64961.59196449,
       48589.92505517, 61788.69667829, 74652.05854861, 56225.8611763 ,
       51787.30758035, 61882.69307713, 58106.70041627, 54917.27885945,
       58080.78480397, 72880.40104288, 39126.94556245, 63292.38232261,
       78772.44390383, 82018.47551598, 68217.31546747, 81873.29599878,
```

```
68977.84475541, 78946.47264182, 78625.61004144, 52033.35847938,
67723.26889981, 76892.19088822, 78706.59375604, 57224.31959963,
82587.34704389, 69385.93572689, 69834.83592656, 45251.27914786,
69670.23047042, 56333.84693343, 64745.77404405, 72001.39367534,
76130.21535525, 74435.43523504, 84824.6537918 , 59048.28981388,
84057.01684189, 81125.75922982, 79564.46118953, 62824.841422  ,
47508.60063576, 51546.96213318, 75363.30326624, 78774.477243  ,
71702.52168897, 86174.61100431, 55074.23780416, 86552.83282002,
49051.03614859, 60861.8510538 , 59103.59790663, 59915.03980765,
66157.75562317, 69203.48925734, 75319.92169978, 69738.94864848,
60965.46913468, 57027.98757806, 46993.1318764 , 56765.97201027,
47306.40556442, 75653.24085446, 53032.87495313, 43183.26386141,
66933.51384549, 79473.35526397, 53038.53439839, 67015.88732846,
55508.14823588, 69018.57377966, 76030.4213551 , 66049.01850182,
64041.96397833, 42477.18567687, 71645.28932639, 45354.61890417,
73715.0417774 , 71686.37922417, 42920.43671523, 66070.09929062,
49666.09516434, 79349.6857122 , 86673.73204054, 69214.29148982,
78407.32743368, 76190.98756978, 70593.33820771, 54378.802454  ,
69662.2613571 , 80410.51817121, 51671.88270639, 63442.03437868,
78874.66896559, 83792.3876255 , 83905.68032486, 64908.68169842,
77597.03597554, 69496.18494581, 55137.18545706, 50068.71186529,
82873.35336823, 65685.81714894, 80184.64222707, 65261.92525887,
74132.02508463, 57818.25350494, 44012.38842077, 43006.03973123,
40785.04580311, 65574.77899992, 62464.33034827, 72565.28720286,
79711.80788618, 66712.16246142, 69676.09604032, 68562.78247306,
56001.73994799, 47183.85224173, 69441.49814071, 77695.20017346,
45582.55725418, 51111.79173729, 56419.75569831, 54533.10715395,
65988.81101484, 83115.12687831, 73424.49901622, 44815.40997972,
57185.92938166, 48938.74220681, 77108.31057689, 70980.51279249,
60513.44068554, 69899.83680493, 53211.78650905, 77738.08258757,
82714.60275649, 45914.17503259, 75088.98567621, 86051.60848497,
68473.93678799, 45461.70391022, 92303.68094744, 56200.63241582,
57701.30861923, 79182.20276312, 71904.83961745, 53570.43471951,
45589.53459476, 59291.68618625, 72243.6345604 , 87455.97018082,
59979.55042148, 55801.56481073, 65464.00586796, 47077.0060498 ,
52009.24659105, 54740.03711619, 80177.78666173, 61818.6106603 ,
77814.48859931, 70109.27066829, 55660.68785218, 46304.00760872,
71384.2987403 , 44447.21884709, 73354.65166734, 58396.52654901,
67303.59829121, 46448.01298154, 66613.70553598, 42006.87961552,
77939.27245124, 64587.45577267, 50687.9925602 , 76032.05464092,
74645.57363539, 45888.6178219 , 48909.9466829 , 69577.23623965,
76260.83661533, 59292.3059499 , 76846.25505989, 72970.35075922,
75072.77096248, 50234.44964372, 56465.82488546, 85618.06565951,
51712.68367739, 56426.0330719 , 66579.10213922, 65452.44019007,
48137.53810712, 44986.82840034, 65738.78439513, 57119.51092279,
69356.27909559, 53371.22818647, 76517.80602827, 80106.87465851,
41279.59755285, 77981.80580787, 62932.55678023, 59822.06101628,
80826.53910603, 67296.38286425, 60608.22939428, 72703.1338573 ,
62637.98076154, 66571.09494835, 65062.87133787, 59483.20707896,
42206.69211352, 43852.47149669, 69151.1807469 , 77876.5547242 ,
79704.30901405, 59745.7190599 , 56745.31738926, 72325.33198028,
86392.67441413, 49457.98169998, 61687.70692775, 75706.65057584,
77141.68263768, 86130.50640317, 79717.03041932, 66371.9691687 ,
54496.96191129, 65772.61928542, 61494.9473611 , 74659.47205136,
80215.12814978, 83949.52605908, 82764.08344199, 69639.26321761,
55375.57959816, 78224.97978503, 61318.18136412, 47050.89375485,
79308.94608677, 74681.70506823, 39388.16117145, 78530.89852848,
77321.72835061, 40751.83678306, 65892.32796835, 60181.51751471,
52169.05334022, 58198.24515938, 86213.16466869, 62079.06382922,
84461.28994156, 65524.66023529, 65934.28415804, 58790.31196112,
74180.82035362, 65542.90459768, 74532.93044945, 57350.8969211 ,
74414.40991301, 80206.48243254, 86683.57542699, 63449.43896   ,
79862.95744596, 56256.29957672, 47338.75910654, 82636.50720794,
58222.2584909 , 52991.1916119 , 42302.62270769, 50791.90828278,
89410.05405728, 61692.93639825, 45461.32793294, 77426.57479434,
77059.63305851, 88513.42646252, 79453.84345171, 59352.22648242,
56499.09875474, 61534.78761136, 85134.83587947, 65768.08249511,
51116.36923394, 60761.22444901, 56255.28005036, 59017.36193288,
78132.83964205, 53955.89241078, 63843.31584304, 64222.98970699,
74551.5850239 , 81245.94517484, 75201.30337703, 80588.5104785 ,
42768.88265062, 70839.77177431, 75901.9514994 , 61381.02787985,
75948.99984781, 71272.94859078, 61483.26562156, 45740.48301923,
49860.50523443, 67042.73019983, 63146.38916098, 93410.62496181,
58121.051455  , 39728.88587473, 41210.88428859, 67360.44710988,
82759.63067793, 69773.82099435, 46938.51710856, 67337.96454024,
77398.37888708, 73087.15230169, 74683.3804971 , 80549.19386034,
```

```
       57421.29288209, 47981.35278352, 62795.9560626 , 70565.74874306,
       45108.44167654, 76337.49333931, 50426.56471969, 60140.45054203,
       75710.35012131, 86787.18305184, 60038.50331179, 53411.30651188,
       77311.67479255, 63578.96451136, 53373.52511404, 77433.37870814,
       57902.07499151, 82798.15256076, 45604.53666196, 76753.41636205,
       63961.54040517, 77389.48136418, 72005.71460488, 53879.63377756,
       68491.15697969, 65631.2013569 , 51020.53211288, 61114.82113135,
       58348.97777721, 90334.36304444, 43861.95659709, 73097.39879466,
       71919.87883196, 58999.70583567, 62531.92987688, 68401.1943863 ,
       62089.62281698, 49317.71953919, 41900.86106614, 63746.6796675 ,
       43898.04202855, 65996.83371331, 71717.87941378, 61372.12048537,
       61846.44764544, 45650.1011014 , 60702.83165792, 57000.95770447,
       62606.67699931, 65236.26241945, 69353.08530515, 64814.49333581,
       69571.74664759, 76059.01247696, 41179.05717162, 56097.41990149,
       66339.62001848, 70626.04701286, 59245.49971212, 79750.25226856,
       63398.54828963, 54084.88040104, 67035.56040189, 66875.53557645,
       82098.33993487, 51560.6496913 , 64755.83419344, 52948.84127019,
       66563.53308865, 70335.85932521, 76133.00984431, 39522.61814245,
       57835.12925278, 48892.33165223, 83607.63663187, 44849.55597693,
       84626.5047868 , 56273.31236206, 64762.09871942, 58909.49119898,
       61965.78846284, 53784.84288666, 44657.32216226, 62412.70318176,
       45130.82742723, 79117.63809827, 59092.59652109, 85628.98300249,
       74974.39402534, 65146.88441998, 63787.62347158, 81695.12175338,
       61009.12403074, 46375.8164263 , 56570.86002403, 73947.67161615,
       44471.67520608, 45673.67918885, 83035.93004695, 42290.4300772 ,
       51701.511569  , 68043.73177214, 67199.86656306, 80940.68003664,
       41448.76522158, 74092.14155716, 66865.62495729, 49363.52139393,
       63029.57449668, 40349.80699306, 63454.55578989, 70824.78978473,
       74196.51475708])
```

```
In [ ]:
```