EDA & Clustering

March 6, 2025

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
import warnings
import random
random.seed(40)
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns',None)
plt.style.use('bmh')
sns.set_style('ticks')

from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler, OneHotEncoder,␣
↪OrdinalEncoder, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import RidgeCV, Ridge, LassoCV, Lasso
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
from sklearn.cluster import SpectralClustering
```

K Means Visualization

[2]:
```
def kmeans_score(data,k):
    random.seed(40)
    wss = []
    sill = []

    for x in range(2,k):
        kmeans = KMeans(n_clusters=x,random_state=0, n_init = 'auto')
        kmeans.fit(data,x)
        labels = kmeans.fit_predict(data)
```

```
        wss.append(kmeans.inertia_)
        score = silhouette_score(data, labels)
        sill.append(score)

    plt.figure(figsize=(15,3))

    plt.subplot(1,2,1)
    plt.title('Elbow Plot')
    plt.plot(range(2,k), wss, marker = "o", linestyle ="--", color = 'teal')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Within Sum of Squares')
    plt.grid()
    sns.despine()

    plt.subplot(1,2,2)
    plt.title('Silhouette Scores')
    plt.plot(range(2,k), sill, marker = "o", linestyle="--", color = 'orange')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Shilhouette Score')
    plt.grid()
    sns.despine()


    plt.show()
```

GMM Visualization

[3]:
```
def gmm_score(data,k):
    random.seed(40)
    bic = []
    shil = []

    for x in range(2,k):
        gmm = GaussianMixture(n_components = x, random_state = 0).fit(data)
        bic.append(gmm.bic(data))
        labels = gmm.fit_predict(data)
        shill = silhouette_score(data,labels)
        shil.append(shill)

    plt.figure(figsize=(15,3))

    plt.subplot(1,2,1)
    plt.title('BIC Cluster Selection')
    plt.ylabel('BIC Scores')
    plt.xlabel('Number of Clusters')
    plt.plot(range(2,k), bic, marker = 'o', linestyle = '--', color = 'teal')
```

```
    plt.grid()
    sns.despine()

    plt.subplot(1,2,2)
    plt.title('Shilhouette Scores')
    plt.ylabel('Shilhouette Scores')
    plt.xlabel('Number of Clusters')
    plt.plot(range(2,k), shil, marker ='o', linestyle ='--', color = 'orange')
    plt.grid()
    sns.despine()

    plt.show()
```

PCA Visual

[4]:
```
def pca_visual(data,explained_variance):

    plt.figure(figsize=(15, 4))

    plt.subplot(1,2,1)
    plt.plot(range(1, len(explained_variance) + 1), explained_variance,␣
 ↪marker='o', linestyle='--', color='teal')
    plt.xlabel('Principal Component')
    plt.ylabel('Variance Explained')
    plt.title('Scree Plot')
    plt.xticks(range(1, len(explained_variance) + 1))
    plt.grid()
    sns.despine()

    plt.subplot(1,2,2)
    plt.scatter('PC1', 'PC2', data=data, alpha = 0.6, color = 'orange')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title('Pricipal Component Plot')
    sns.despine()

    plt.show()
```

[5]:
```
def cluster_visual(data, one, two):



    plt.figure(figsize=(15,4))

    plt.subplot(1,2,1)
    plt.title('Cluster Visualization Plot')
```

```
    sns.scatterplot(x = data.iloc[:,0], y = data.iloc[:,1], data=data, hue=one,␣
↪s=100, palette='viridis')
    sns.despine()

    plt.subplot(1,2,2)
    plt.title('Cluster Visualization Plot')
    sns.scatterplot(x = data.iloc[:,0], y = data.iloc[:,1], data=data, hue=two,␣
↪s=100, palette='viridis')
    sns.despine()
```

1 Data

[6]:
```
Actual = pd.read_csv('treated_data.csv')

data = Actual[Actual['Onboard year'].isin([2015, 2016, 2017, 2018, 2019, 2020,␣
↪2021, 2022])].reset_index(drop=True)

data['Last Date of purchase summer'] = pd.to_datetime(data['Last Date of␣
↪purchase summer'])
data['Last Date of purchase winter'] = pd.to_datetime(data['Last Date of␣
↪purchase winter'])
data['Last Date of purchase all seasons'] = pd.to_datetime(data['Last Date of␣
↪purchase all seasons'])
data['Customer onboard date'] = pd.to_datetime(data['Customer onboard date'])
```

[7]:
```
index = data[(data['Number of vehicles in household'] == 0)].index

data.loc[index,'Number of vehicles in household'] = 1
```

Converting last purchase into number of days elapssed

[8]:
```
def max_date(data):

    if(data['Last Date of purchase summer'] > data['Last Date of purchase␣
↪winter']):

        if(data['Last Date of purchase summer'] > data['Last Date of purchase␣
↪all seasons']):

            data['Time'] = dt.datetime.now() - data['Last Date of purchase␣
↪summer']
            data['Time'] = data['Time'].days

        else:
```

```
            data['Time'] = dt.datetime.now() - data['Last Date of purchase all␣
↪seasons']
            data['Time'] = data['Time'].days

    elif(data['Last Date of purchase winter'] > data['Last Date of purchase all␣
↪seasons']):

        data['Time'] = dt.datetime.now() - data['Last Date of purchase winter']
        data['Time'] = data['Time'].days
    else:
        data['Time'] = dt.datetime.now() - data['Last Date of purchase all␣
↪seasons']
        data['Time'] = data['Time'].days


    return data
```

[9]:
```
data = data.apply(max_date, axis=1)
```

[10]:
```
df = data.copy()
```

Dropping all unecessary features

[11]:
```
data = data.drop(columns=['Acct. #','Customer.ID','Cust. Year_Birth','Last Date␣
↪of purchase summer',
                          'Last Date of purchase winter','Last Date of purchase␣
↪all seasons','Onboard month',
                          'Onboard day','Onboard year','Customer onboard␣
↪date','Summer Tire','Winter Tire Base',
                          'Winter Tire Pre','All Season'], axis=1)
```

[12]:
```
numeric = [features for features in data.columns if (data[features].dtype !=␣
↪'O') &
                                                    (data[features].dtype !=␣
↪'<M8[ns]') &
                                                    (features not in ['Acct.␣
↪#','Customer.ID',
                                                                      'Onboard␣
↪month','Onboard day','Onboard year',
                                                                      'Cust.␣
↪Year_Birth'])]

categorical = [features for features in data.columns if (data[features].dtype␣
↪== 'O') &
                                                        (data[features].dtype !
↪= '<M8[ns]') &
```

```
                                                    (features not in ['Cust.
 ↪ Education','Household income',
                                                                        ␣
 ↪'Highest quality of rim purchased in the last 5 years'])]

ordinal = ['Cust. Education','Household income','Highest quality of rim␣
 ↪purchased in the last 5 years']

categories = [

              ['Basic','HS dipl.','College','Graduate','Bachelors'],
              ['< 32000','32000 - 55000','75000 - 99000','99000 -␣
 ↪140000','55000 - 75000','140000 - 200000','> 200000'],
              ['Not Mentioned','Aluminum','Steel','Specialised']

              ]
```

Feature Transfomration

[13]:
```
preprocessor = ColumnTransformer(
    transformers = [
        ('num',StandardScaler(),numeric),
        ('ord',OrdinalEncoder(categories = categories),ordinal),
        ('nom',OneHotEncoder(drop='first'),categorical)
        ])
```

[14]:
```
preprocessed = preprocessor.fit_transform(data)
preprocessed_df = pd.DataFrame(preprocessed, columns=preprocessor.
 ↪get_feature_names_out())
```

[15]:
```
preprocessed_df.shape
```

[15]: (1672, 56)

2 Feature Selection

Variance Threshold

[16]:
```
selector = VarianceThreshold(threshold=0.01)
selected = selector.fit_transform(preprocessed_df)
selected = pd.DataFrame(selected, columns=selector.get_feature_names_out())
```

[17]:
```
selected.shape
```

[17]: (1672, 53)

Lasso CV Selection

[18]:
```
X = selected.drop(columns = 'num__Total Tires', axis=1)
Y = selected['num__Total Tires']
```

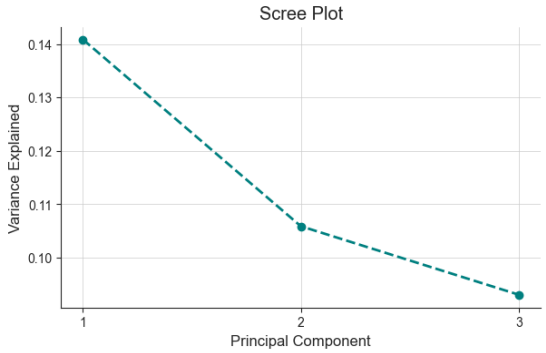
[19]:
```
random.seed(40)
alpha = np.logspace(-9, 9, 100)
lasso = LassoCV(alphas=alpha)
lasso.fit(X,Y)
l_alpha = lasso.alpha_
```

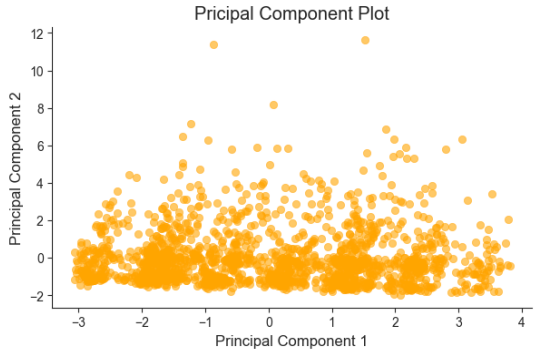
[20]:
```
lasso = Lasso(alpha=l_alpha)
lasso.fit(X,Y)
lasso_features = X.columns[lasso.coef_ != 0]
```

PCA on Lasso Features

[21]:
```
pca_data = selected[lasso_features]

pca = PCA(n_components=3)
pca = pca.fit(pca_data)
pca_data = pca.fit_transform(pca_data)

pca_data = pd.DataFrame(pca_data, columns=['PC1','PC2','PC3'])
lasso_loading = pd.DataFrame(pca.components_.T, columns=['PC1','PC2','PC3'],␣
 ↪index = lasso_features)
l_variance = pca.explained_variance_ratio_
```

[22]:
```
pca_visual(pca_data, l_variance)
```





Ridge CV Selection

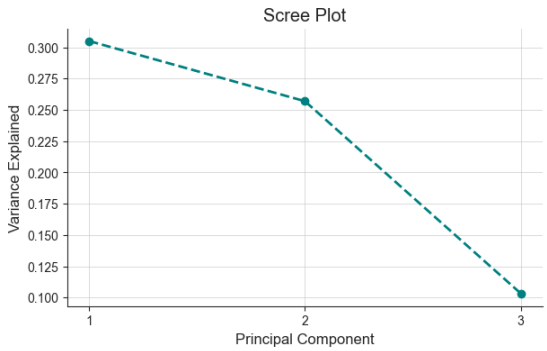
[23]:
```
random.seed(40)
alpha = np.logspace(-9, 9, 100)
ridge = RidgeCV(alphas=alpha)
ridge.fit(X,Y)
r_alpha = ridge.alpha_
```

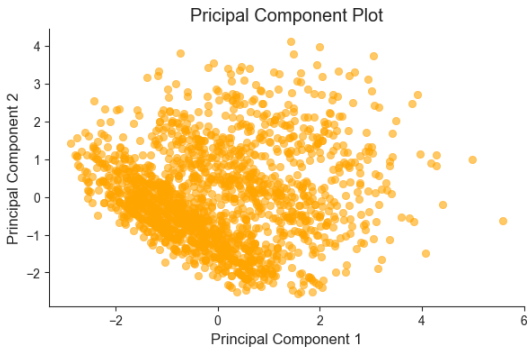
[24]:
```
ridge = Ridge(alpha = r_alpha)
ridge.fit(X,Y)

feature_importance = np.abs(ridge.coef_)
ridge_features = X.columns[feature_importance > np.
 ↪percentile(feature_importance, 50)]
```

PCA on Ridge Features

[25]:
```
pca = PCA(n_components=3)

pca_data1 = selected[ridge_features]

pca_data = pca.fit_transform(pca_data1)
pca_data = pd.DataFrame(pca_data, columns=['PC1','PC2','PC3'])

load = pca.fit(pca_data1)
ridge_loading = pd.DataFrame(load.components_.T, columns=['PC1','PC2','PC3'],␣
 ↪index=pca_data1.columns)
r_variance = pca.explained_variance_ratio_
```

[26]:
```
pca_visual(pca_data,r_variance)
```





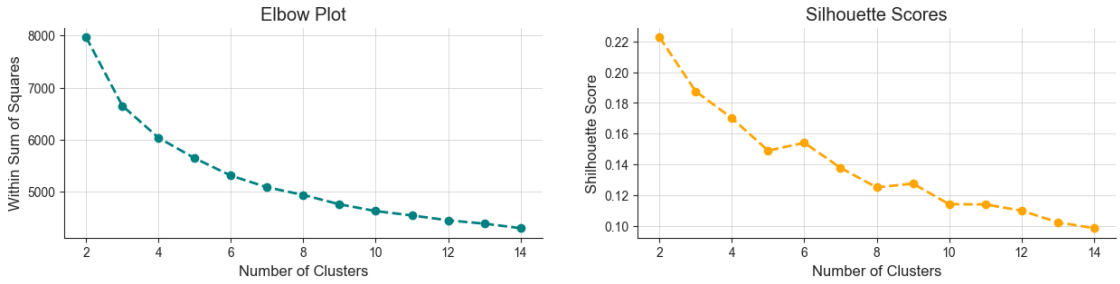
3 Clustering

[27]:
```
new_data = selected[ridge_features]
new_data.shape
```

[27]:
```
(1672, 26)
```

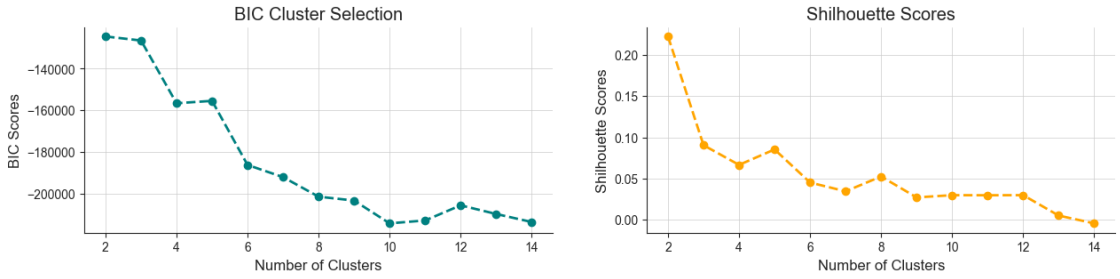
K Means Scores

[28]:
```
kmeans_score(new_data,15)
```



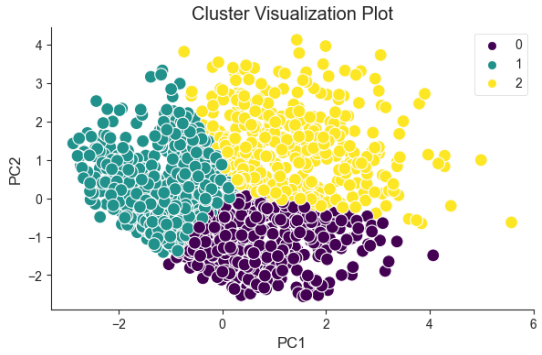
GMM Scores

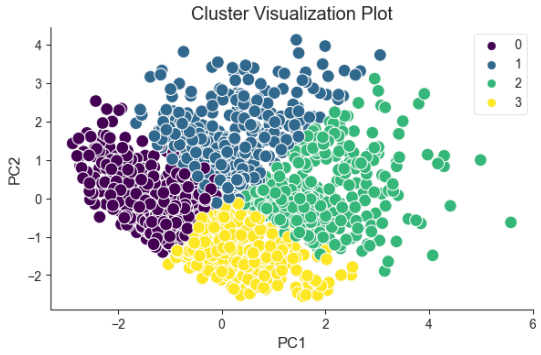
[29]:
```
gmm_score(new_data,15)
```



K Means Cluster Visualization

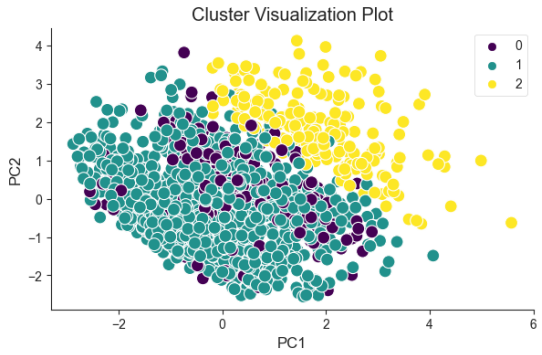
[30]:
```
k_labels3 = KMeans(3,random_state=0,n_init='auto').fit_predict(new_data)
k_labels4 = KMeans(4,random_state=0,n_init='auto').fit_predict(new_data)

cluster_visual(pca_data,k_labels3,k_labels4)
```

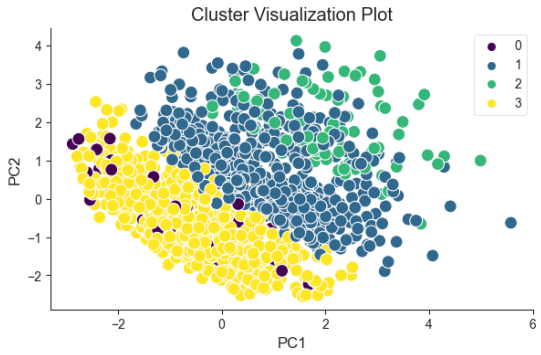




GMM Cluster Visualization

[31]:
```
g_labels3 = GaussianMixture(3,random_state=0).fit_predict(new_data)
g_labels4 = GaussianMixture(4,random_state=0).fit_predict(new_data)

cluster_visual(pca_data,g_labels3,g_labels4)
```

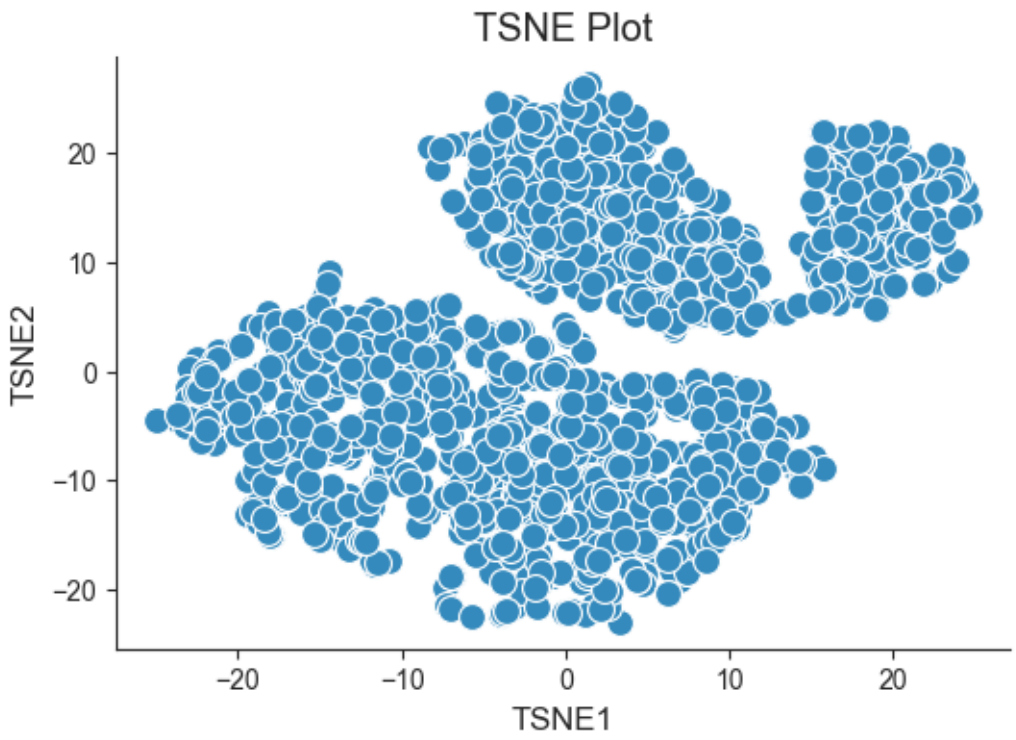




4 t-SNE

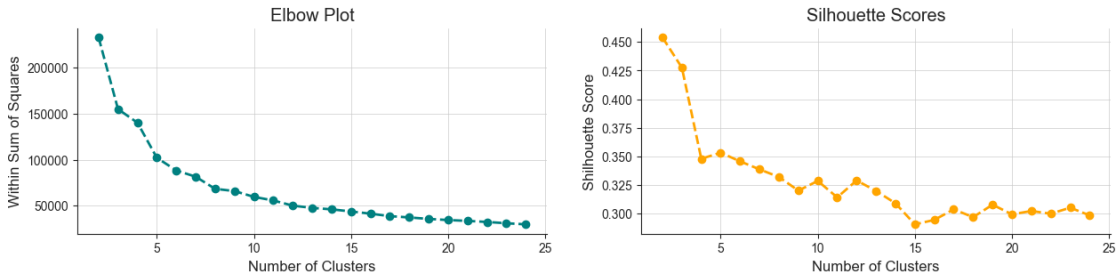
[32]:
```
from sklearn.manifold import TSNE
random.seed(40)
tsne = TSNE(n_components=3, random_state=0)
tsne_data = tsne.fit_transform(new_data)
tsne_data = pd.DataFrame(tsne_data,columns=['TSNE1','TSNE2','TSNE3'])
```

[33]:
```
plt.figure(figsize=(6,4))
plt.title('TSNE Plot')
sns.scatterplot(x='TSNE1',y='TSNE2',data=tsne_data,s=100)
sns.despine()
```



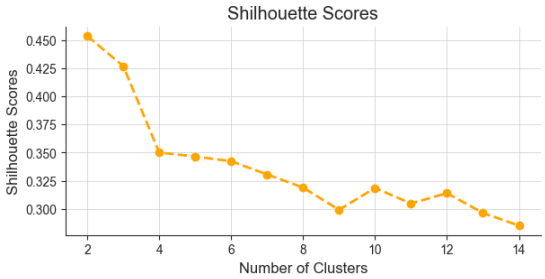
K Means Scores

[34]:
```
kmeans_score(tsne_data,25)
```



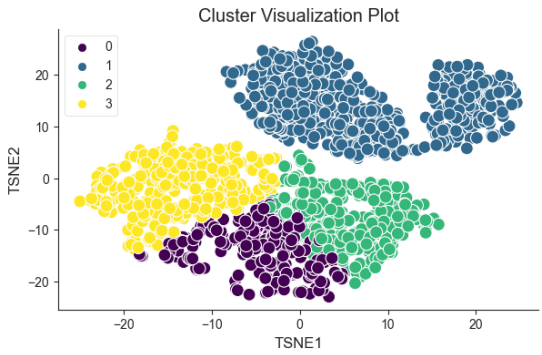
GMM Scores

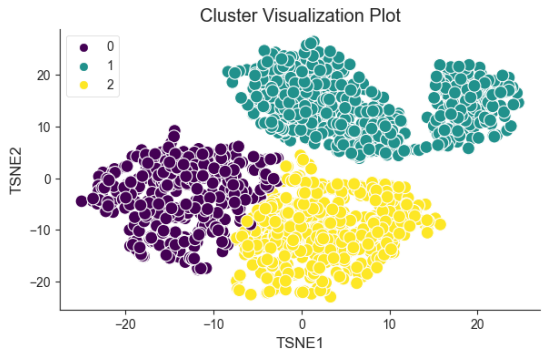
[35]:
```
gmm_score(tsne_data,15)
```



K Means Cluster Visualization

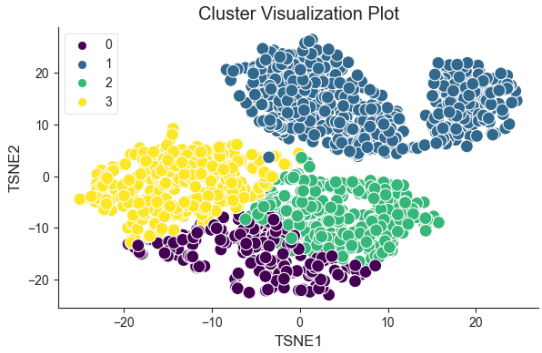
[36]:
```
k_labels3 = KMeans(3,n_init='auto',random_state=0).fit_predict(tsne_data)
k_labels4 = KMeans(4,n_init='auto',random_state=0).fit_predict(tsne_data)

cluster_visual(tsne_data,k_labels4,k_labels3)
```

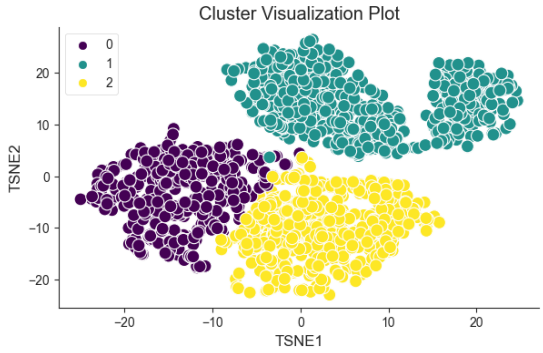




GMM Cluster Visualization

[37]:
```
g_labels4 = GaussianMixture(4,random_state=0).fit_predict(tsne_data)
g_labels3 = GaussianMixture(3,random_state=0).fit_predict(tsne_data)

cluster_visual(tsne_data,g_labels4,g_labels3)
```

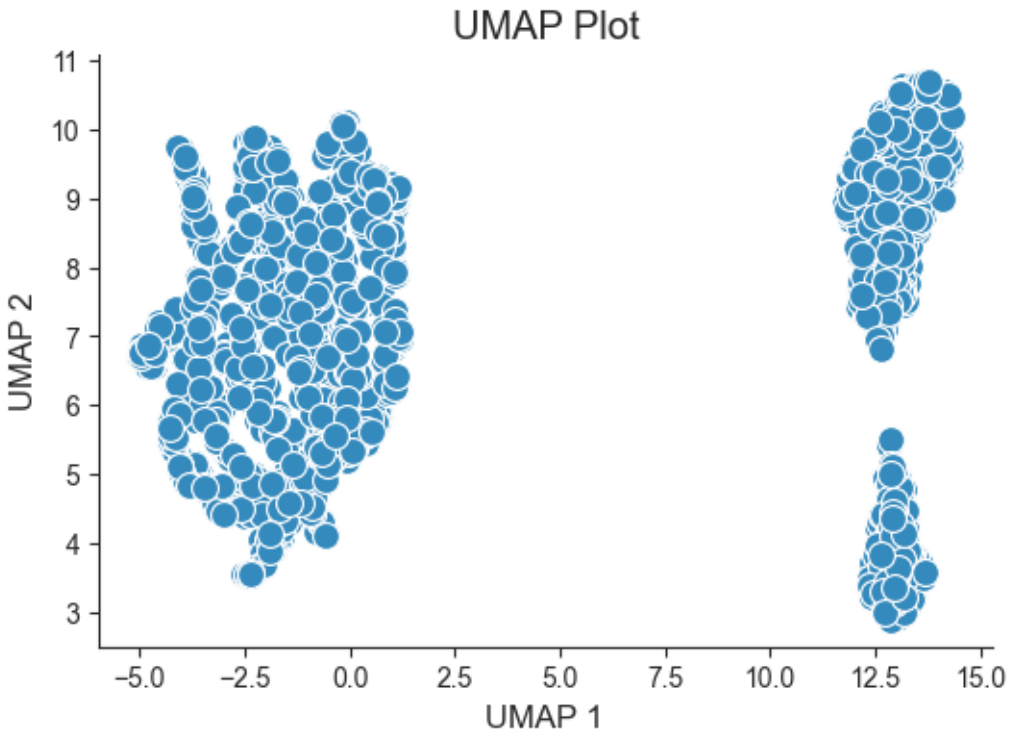




5 UMAP

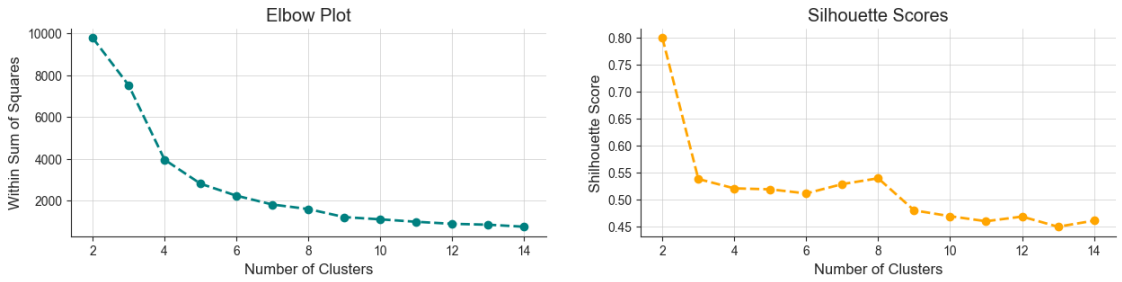
[38]:
```
import umap.umap_ as umap
random.seed(40)
standard_embedding = umap.UMAP(random_state=42).fit_transform(new_data)
standard_embedding = pd.DataFrame(standard_embedding, columns = ['UMAP 1','UMAP␣
↪2'])
```

[39]:
```
plt.figure(figsize=(6,4))
plt.title('UMAP Plot')
plt.xlabel('UMAP 1')
plt.ylabel('UMAP 2')
sns.scatterplot(x = 'UMAP 1', y = 'UMAP 2', data = standard_embedding, s=100)
sns.despine()
```



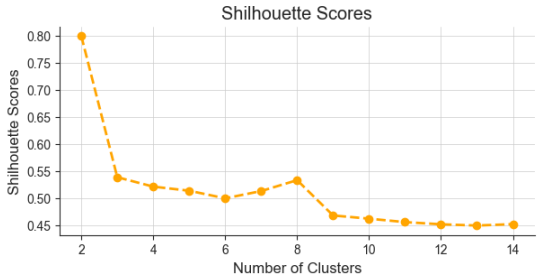
K Means Scores

[40]:
```
kmeans_score(standard_embedding,15)
```



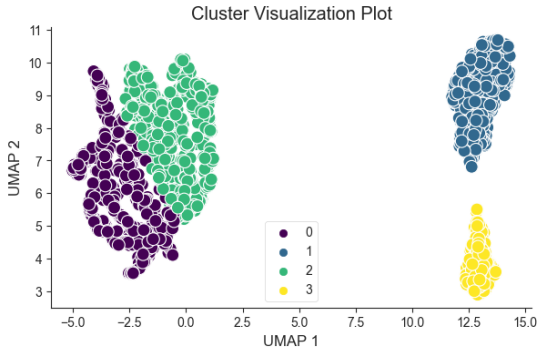
GMM Scores

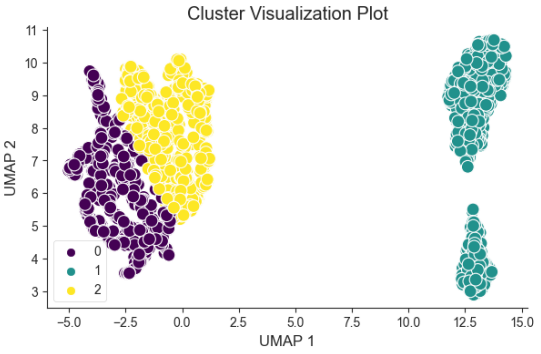
[41]:
```
gmm_score(standard_embedding, 15)
```



K Means Cluster Visualization

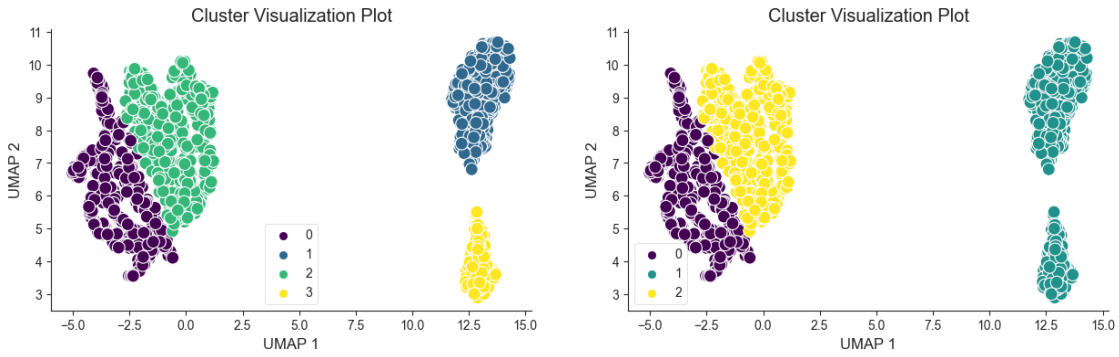
[42]:
```
k_labels3 = KMeans(3, random_state = 0, n_init='auto').
  ↪fit_predict(standard_embedding)
k_labels4 = KMeans(4, random_state = 0, n_init='auto').
  ↪fit_predict(standard_embedding)

cluster_visual(standard_embedding,k_labels4,k_labels3)
```





GMM Cluster Visualization

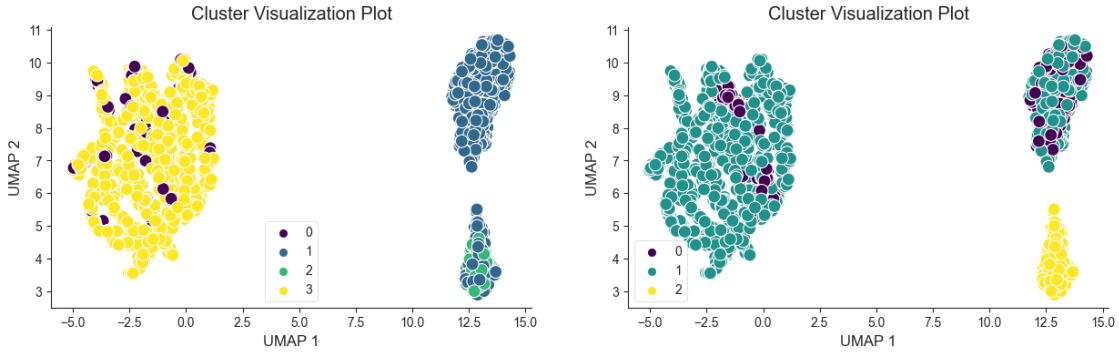
[43]:
```
g_labels3 = GaussianMixture(3, random_state = 0).fit_predict(standard_embedding)
g_labels4 = GaussianMixture(4, random_state = 0).fit_predict(standard_embedding)

cluster_visual(standard_embedding,g_labels4,g_labels3)
```



5.0.1 Out of all the results, Gaussian Mixture Model with 3 classes performed the best on PCA with Ridge Regression. It is also evident from the other decomposition methods that there are three clusters in the data.

5.0.2 PCA clusters visualized on UMAP decomposition as the row information is preserved in both the operations

[44]:
```
cluster_visual(standard_embedding,GaussianMixture(4,random_state=0).
↪fit_predict(new_data),GaussianMixture(3,random_state=0).
↪fit_predict(new_data))
```



5.0.3 This plot on the right side represents PCA clusters on UMAP decomposition. The clusters look well defined with a little bit of overlap. I decided to go with the GMM clusters on PCA

Deducing important features

[45]:
```
np.abs(ridge_loading['PC1']).sort_values(ascending=False)[0:6]
```

[45]:
```
num__Number of purchases made in the store    0.672982
num__Age                                      0.481700
num__Number of vehicles in household          0.479057
nom__2nd Vehicle_No                           0.211438
num__Number of online purchases               0.135612
nom__3rd Vehicle_No                           0.117394
Name: PC1, dtype: float64
```

[46]:
```
np.abs(ridge_loading['PC2']).sort_values(ascending=False)[0:5]
```

[46]:
```
num__Number of online purchases          0.619819
num__Number of vehicles in household     0.539182
num__Age                                 0.490559
nom__2nd Vehicle_No                      0.234986
nom__3rd Vehicle_No                      0.135541
Name: PC2, dtype: float64
```

[47]:
```
np.abs(ridge_loading['PC3']).sort_values(ascending=False)[0:5]
```

[47]:
```
num__Number of online purchases               0.764403
num__Age                                      0.400105
num__Number of vehicles in household          0.391367
num__Number of purchases made in the store    0.229024
nom__2nd Vehicle_No                           0.180130
Name: PC3, dtype: float64
```

5.1 Important Features are

Feature	Type
Number of purchases made in the store	Numeric
Number of vehicles in household	Numeric
Age	Numeric
Second Vehicle_No	Categorical
Number of online purchases	Numeric
Third Vehicle_No	Categorical

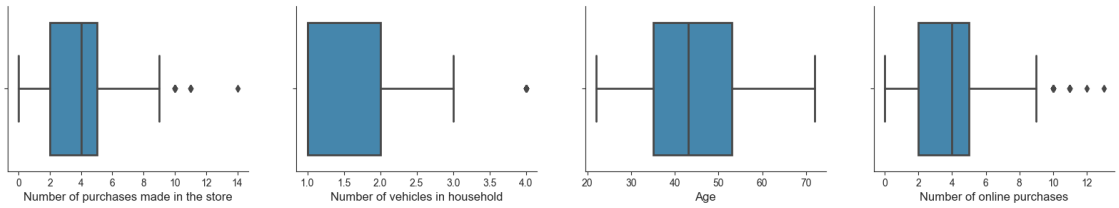
6 Analysing These Important Features

[48]:
```
analysis = df[['Customer.ID','Number of purchases made in the store','Number of␣
↪vehicles in household',
               'Age','2nd Vehicle','Number of online purchases','3rd␣
↪Vehicle']]
```

```
num = ['Number of purchases made in the store','Number of vehicles in␣
 ↪household','Age','Number of online purchases']
cat = ['2nd Vehicle','3rd Vehicle']
```

6.1 Outlier Detection

[49]:
```
plt.figure(figsize=(20,3))

plt.subplot(1,4,1)
sns.boxplot(x = analysis['Number of purchases made in the store'])
plt.subplot(1,4,2)
sns.boxplot(x = analysis['Number of vehicles in household'])
plt.subplot(1,4,3)
sns.boxplot(x = analysis['Age'])
plt.subplot(1,4,4)
sns.boxplot(x = analysis['Number of online purchases'])

sns.despine()
```



[50]:
```
pre = ColumnTransformer (
    transformers = [('num', StandardScaler(), num),
                    ('hot',OneHotEncoder(drop='first'),cat)]
)

analyse_transformed = pre.fit_transform(analysis)
analyse_transformed = pd.DataFrame(analyse_transformed, columns = pre.
 ↪get_feature_names_out())
```

[51]:
```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(analyse_transformed)

outliers = np.where(dbscan.labels_ == -1)[0]

analysis['Outliers'] = 0
analysis.loc[outliers,'Outliers'] = -1
```

[52]:
```
analysis.head()

outlier_data = analysis.loc[outliers].reset_index(drop='True')
non_outlier_data = analysis.drop(index=outliers).reset_index(drop='True')
```

[53]:
```
non_outlier_data['3rd Vehicle'].value_counts()
```

[53]:
```
3rd Vehicle
No    1077
Name: count, dtype: int64
```

[54]:
```
non_outlier_transformed = pre.fit_transform(non_outlier_data)
non_outlier_transformed = pd.DataFrame(non_outlier_transformed, columns = pre.
 ↪get_feature_names_out())
```

[55]:
```
outlier_transformed = pre.fit_transform(outlier_data)
outlier_transformed = pd.DataFrame(outlier_transformed, columns = pre.
 ↪get_feature_names_out())
```

6.2 PCA without outliers

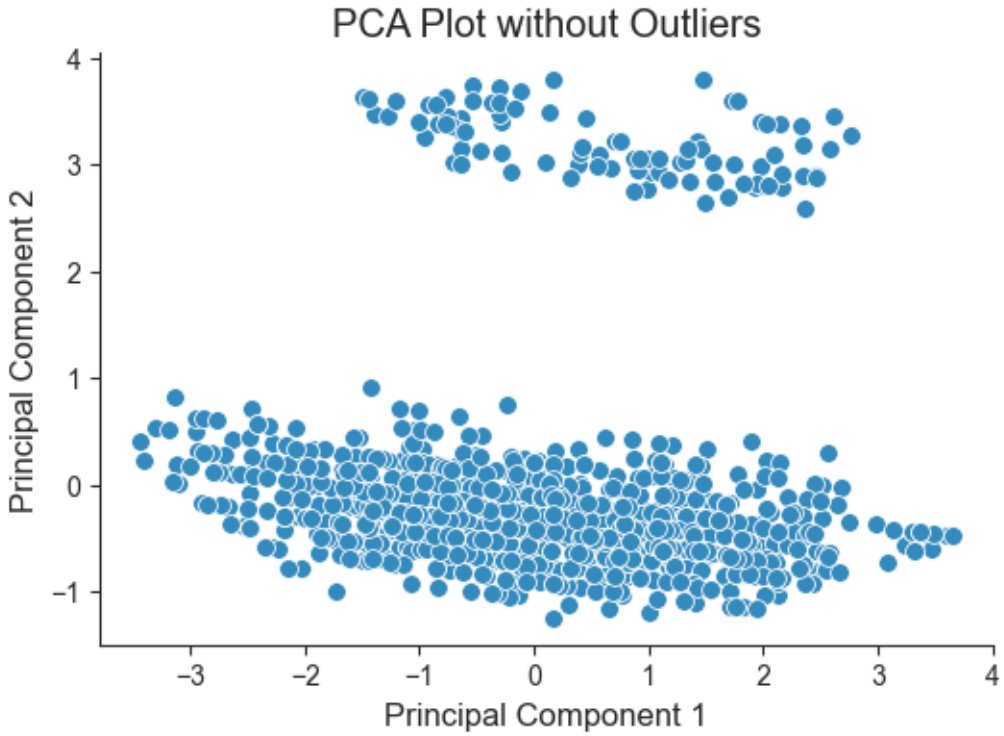
[56]:
```
random.seed(40)
pca = PCA(n_components = 3)

non_outlier_pca = pca.fit(non_outlier_transformed)
non_outlier_pca = pca.fit_transform(non_outlier_transformed)

non_outlier_pca = pd.DataFrame(non_outlier_pca, columns = ['PC1','PC2','PC3'])

non_outlier_loading = pd.DataFrame(pca.components_.T,␣
 ↪index=non_outlier_transformed.columns)
```

[57]:
```
non_outlier_loading[2]
```

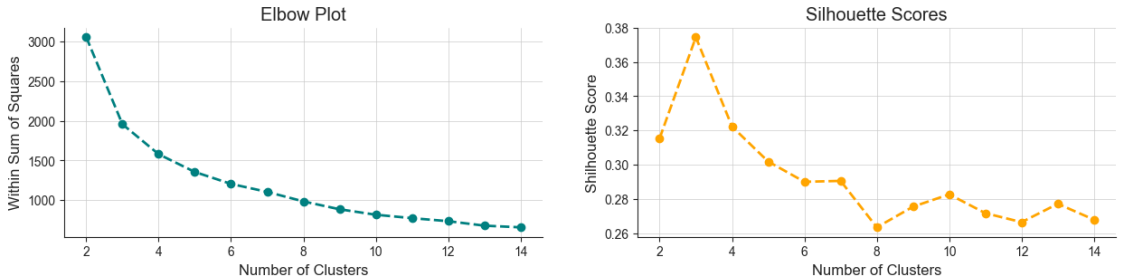
[57]:
```
num__Number of purchases made in the store    0.442625
num__Number of vehicles in household         -0.357597
num__Age                                      0.203038
num__Number of online purchases               0.789189
hot__2nd Vehicle_Mini Van                    -0.011336
hot__2nd Vehicle_No                           0.102845
hot__2nd Vehicle_SUV                         -0.013941
hot__2nd Vehicle_Sedan                       -0.035142
hot__2nd Vehicle_Truck                       -0.005356
Name: 2, dtype: float64
```

6.3 Clustering

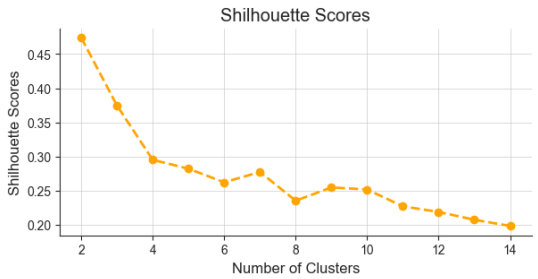
[58]:
```
plt.figure(figsize=(6, 4))
sns.scatterplot(x='PC1', y='PC2', data=non_outlier_pca, palette='Blues', s=50)
plt.title('PCA Plot without Outliers')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
sns.despine()
plt.show()
```



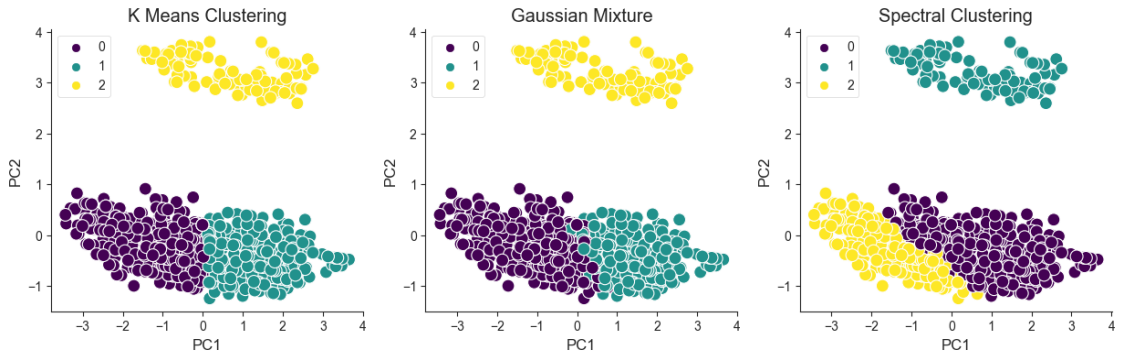
[59]:
```
kmeans_score(non_outlier_transformed,15)
```



[60]:
```
gmm_score(non_outlier_transformed, 15)
```

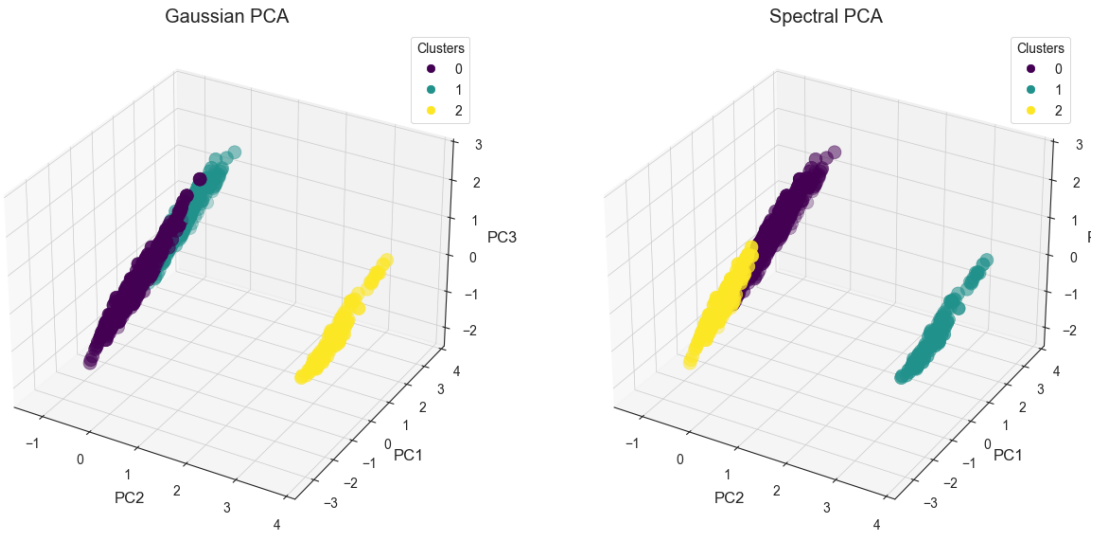


[61]:
```
plt.figure(figsize=(15,4))


plt.subplot(1,3,1)
plt.title('K Means Clustering')
sns.scatterplot(x='PC1',y='PC2',data=non_outlier_pca,s=100,
                hue=KMeans(n_clusters = 3, random_state=0, n_init='auto').
 ↪fit_predict(non_outlier_transformed),palette='viridis')


plt.subplot(1,3,2)
plt.title('Gaussian Mixture')
sns.scatterplot(x='PC1',y='PC2',data=non_outlier_pca,s=100,
                hue=GaussianMixture(n_components = 3, random_state=0).
 ↪fit_predict(non_outlier_transformed),palette='viridis')

plt.subplot(1,3,3)
plt.title('Spectral Clustering')
sns.scatterplot(x='PC1',y='PC2',data=non_outlier_pca,s=100,
                hue=SpectralClustering(n_clusters = 3,␣
 ↪affinity='nearest_neighbors', random_state=0).
 ↪fit_predict(non_outlier_transformed),palette='viridis')

sns.despine()
```



[62]:
```
fig = plt.figure(figsize=(15, 7))

ax1 = fig.add_subplot(121, projection='3d')
scatter = ax1.scatter(non_outlier_pca['PC2'], non_outlier_pca['PC1'],␣
 ↪non_outlier_pca['PC3'],
                      c = GaussianMixture(n_components = 3, random_state=0).
 ↪fit_predict(non_outlier_transformed),
                      cmap='viridis', s=100)
ax1.set_xlabel('PC2')
ax1.set_ylabel('PC1')
ax1.set_zlabel('PC3')
ax1.set_title('Gaussian PCA')
legend1 = ax1.legend(*scatter.legend_elements(), title="Clusters")
ax1.add_artist(legend1)



ax2 = fig.add_subplot(122, projection='3d')
scatter = ax2.scatter(non_outlier_pca['PC2'], non_outlier_pca['PC1'],␣
 ↪non_outlier_pca['PC3'],
                      c = SpectralClustering(n_clusters=3,␣
 ↪affinity='nearest_neighbors', random_state=0).
 ↪fit_predict(non_outlier_transformed),
                      cmap='viridis', s=100)
ax2.set_xlabel('PC2')
ax2.set_ylabel('PC1')
ax2.set_zlabel('PC3')
ax2.set_title('Spectral PCA')
legend2 = ax2.legend(*scatter.legend_elements(), title="Clusters")
ax2.add_artist(legend2)
```

[62]:
```
<matplotlib.legend.Legend at 0x17699385650>
```



[63]:
```
non_outlier_data['Clusters'] = SpectralClustering(n_clusters=3,␣
↪affinity='nearest_neighbors', random_state=0).
↪fit_predict(non_outlier_transformed)
non_outlier_data.head()
```

[63]:

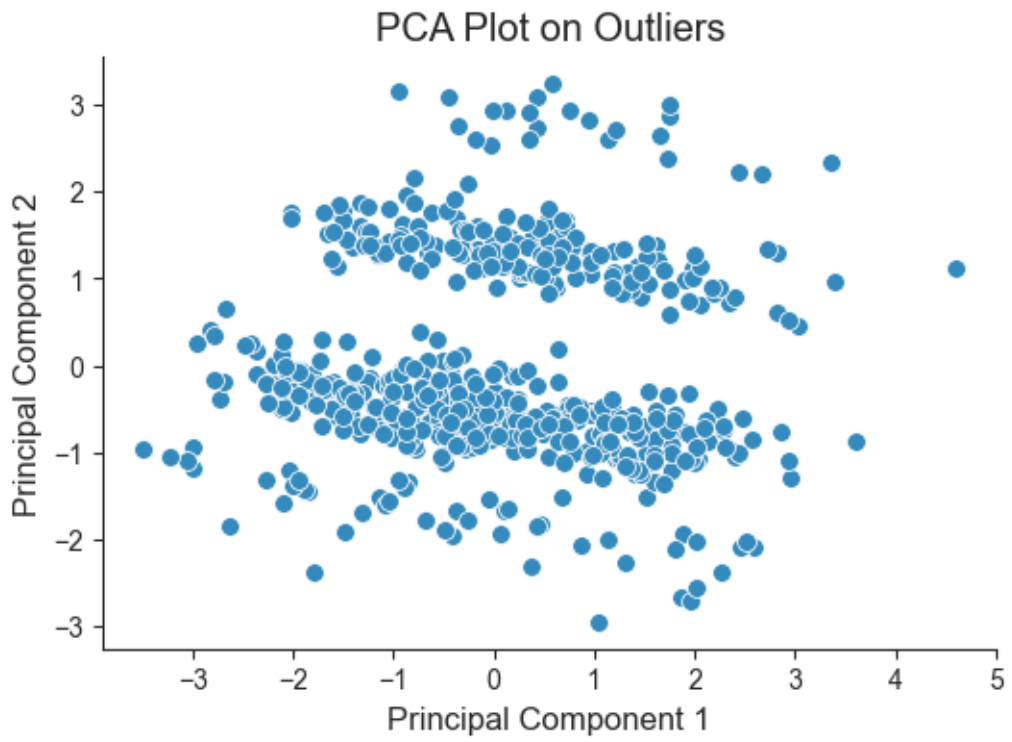
	Customer.ID	Number of purchases made in the store	Number of vehicles in household	Age	2nd Vehicle	Number of online purchases	3rd Vehicle	Outliers	Clusters
0	6870747	4	1	61.0	No	1	No	0	0
1	5394751	2	1	28.0	No	4	No	0	2
2	6735244	3	1	51.0	No	5	No	0	0
3	2224378	2	1	36.0	No	2	No	0	2
4	2683179	2	1	38.0	No	5	No	0	2

6.4 PCA With Outliers

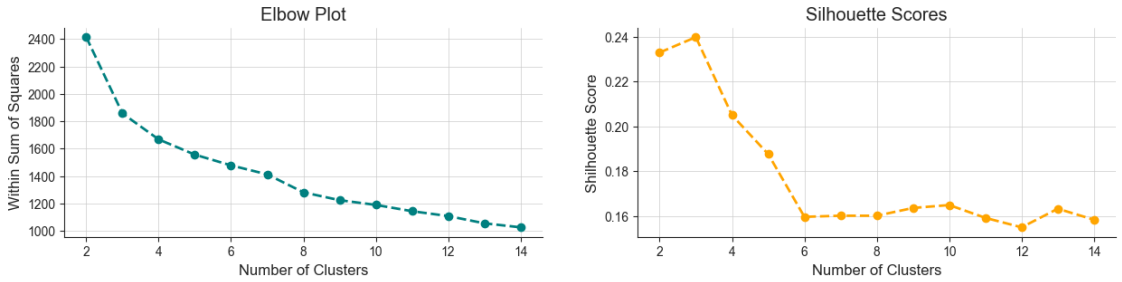
[64]:
```
pre = ColumnTransformer (
    transformers = [('num', StandardScaler(), num),
                    ('hot',OneHotEncoder(drop='first'),cat)]
)


outlier_transformed = pre.fit_transform(outlier_data)
outlier_transformed = pd.DataFrame(outlier_transformed, columns = pre.
 ↪get_feature_names_out())
```

[65]:
```
random.seed(40)
pca = PCA(n_components = 3)

outlier_pca = pca.fit(outlier_transformed)
outlier_pca = pca.fit_transform(outlier_transformed)

outlier_pca = pd.DataFrame(outlier_pca, columns = ['PC1','PC2','PC3'])

outlier_loading = pd.DataFrame(pca.components_.T, index=outlier_transformed.
 ↪columns)
```

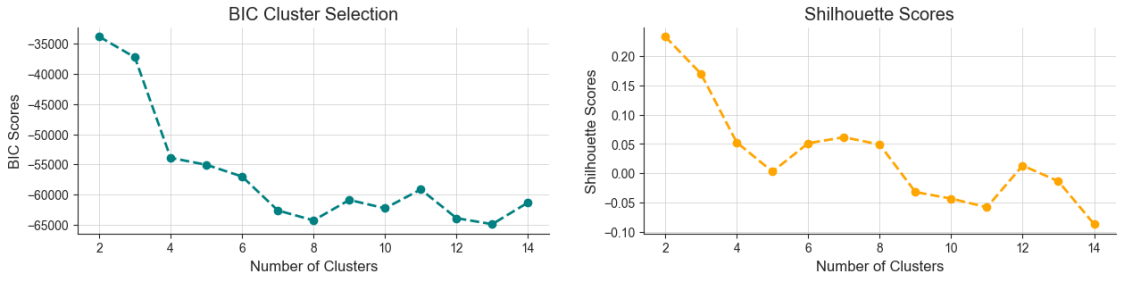
[66]:
```
plt.figure(figsize=(6, 4))
sns.scatterplot(x='PC1', y='PC2', data=outlier_pca, palette='Blues', s=50)
plt.title('PCA Plot on Outliers')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
sns.despine()
plt.show()
```



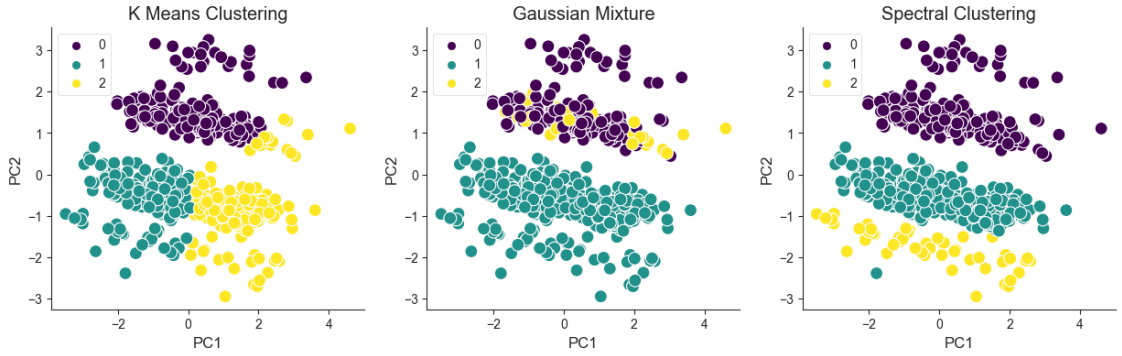
[67]:
```
kmeans_score(outlier_transformed, 15)
```



[68]:
```
gmm_score(outlier_transformed, 15)
```



[69]:
```
plt.figure(figsize=(15,4))


plt.subplot(1,3,1)
plt.title('K Means Clustering')
sns.scatterplot(x='PC1',y='PC2',data=outlier_pca,s=100,
                hue=KMeans(n_clusters = 3, random_state=0, n_init='auto').
 ↪fit_predict(outlier_transformed),palette='viridis')


plt.subplot(1,3,2)
plt.title('Gaussian Mixture')
sns.scatterplot(x='PC1',y='PC2',data=outlier_pca,s=100,
                hue=GaussianMixture(n_components = 3, random_state=0).
 ↪fit_predict(outlier_transformed),palette='viridis')

plt.subplot(1,3,3)
plt.title('Spectral Clustering')
sns.scatterplot(x='PC1',y='PC2',data=outlier_pca,s=100,
                hue=SpectralClustering(n_clusters=3,␣
 ↪affinity='nearest_neighbors', random_state=0).
 ↪fit_predict(outlier_transformed),palette='viridis')

sns.despine()
```

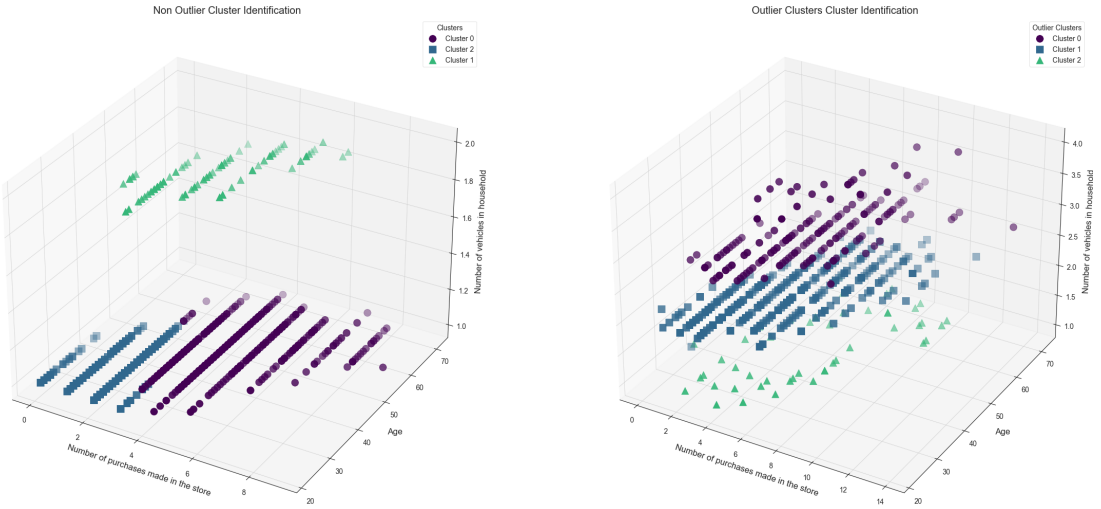


[91]:
```
outlier_data['Clusters'] = SpectralClustering(n_clusters=3,␣
 ↪affinity='nearest_neighbors', random_state=0).
 ↪fit_predict(outlier_transformed)
```

6.5 Visualising the clusters in terms of the data

[92]:
```
markers = ['o', 's', '^', 'D']

fig = plt.figure(figsize=(28,15))
ax = fig.add_subplot(121, projection='3d')


unique_clusters = non_outlier_data['Clusters'].unique()
cmap = plt.get_cmap('viridis')


for i, cluster in enumerate(unique_clusters):
    cluster_data = non_outlier_data[non_outlier_data['Clusters'] == cluster]
    ax.scatter(cluster_data['Number of purchases made in the store'],
               cluster_data['Age'],
               cluster_data['Number of vehicles in household'],
               c=cmap(i / len(unique_clusters)),
               s=100,
               marker=markers[i % len(markers)],
               label=f'Cluster {cluster}')

ax.set_xlabel('Number of purchases made in the store')
ax.set_ylabel('Age')
ax.set_zlabel('Number of vehicles in household')
ax.set_title('Non Outlier Cluster Identification')


ax.legend(title="Clusters")




markers = ['o', 's', '^', 'D']
ax = fig.add_subplot(122, projection='3d')


unique_clusters = outlier_data['Clusters'].unique()
cmap = plt.get_cmap('viridis')


for i, cluster in enumerate(unique_clusters):
    cluster_data = outlier_data[outlier_data['Clusters'] == cluster]
    ax.scatter(cluster_data['Number of purchases made in the store'],
               cluster_data['Age'],
               cluster_data['Number of vehicles in household'],
               c=cmap(i / len(unique_clusters)),
```

```
                s=100,
                marker=markers[i % len(markers)],
                label=f'Cluster {cluster}')

ax.set_xlabel('Number of purchases made in the store')
ax.set_ylabel('Age')
ax.set_zlabel('Number of vehicles in household')
ax.set_title('Outlier Clusters Cluster Identification')


ax.legend(title="Outlier Clusters")

plt.show()
```



[93]:
```
clusters = {
    1 : -1,
    0 : -3,
    2 : -2
}

outlier_data['Clusters'] = outlier_data['Clusters'].map(clusters)
```

7 Merging Outliers & Non-Outlier Dataset

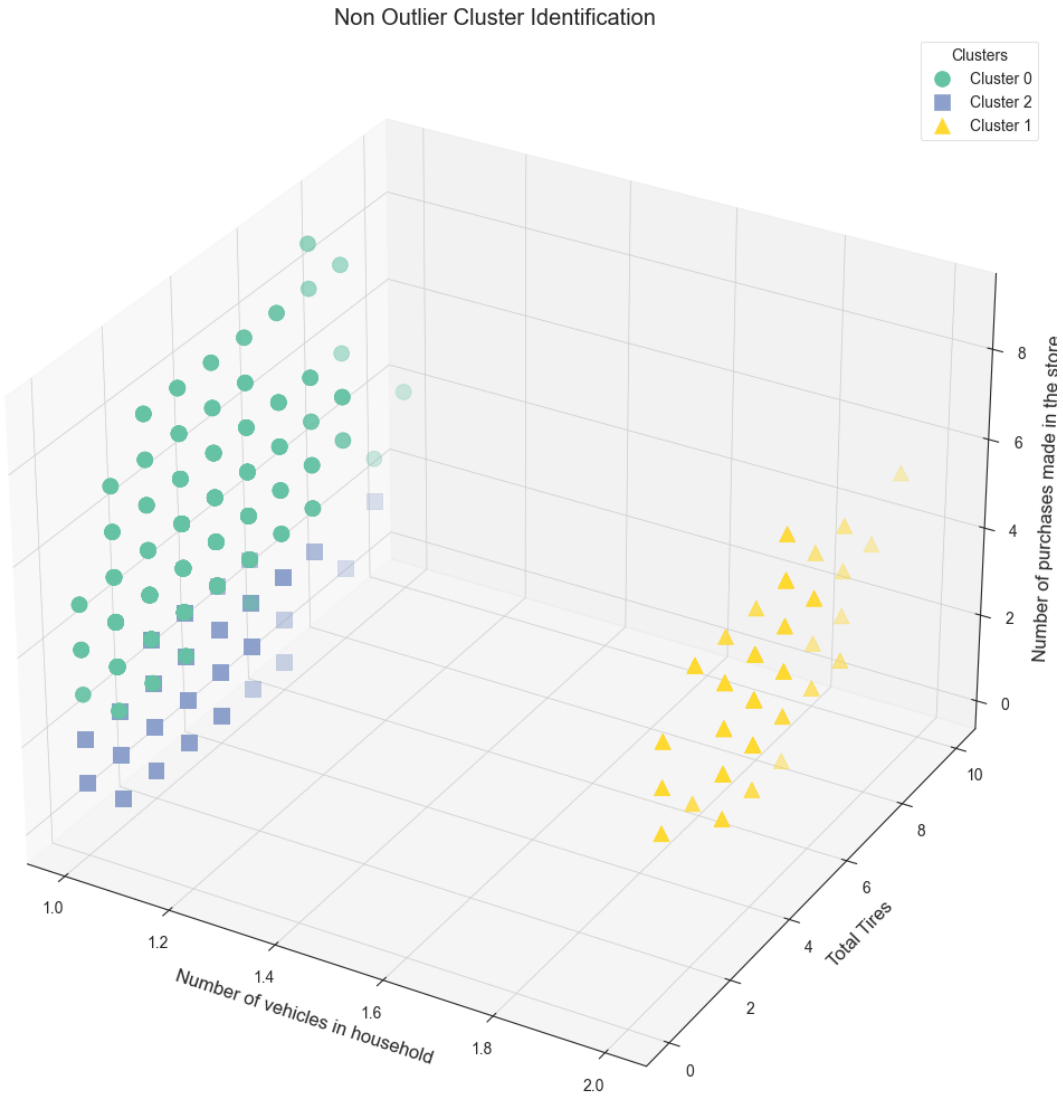
[94]:
```
clustered = pd.concat([outlier_data,non_outlier_data], axis=0).
 ↪reset_index(drop=True)
clustered = clustered.drop(['Outliers'], axis =  1)
```

[95]:
```
final = df.merge(clustered[['Customer.ID','Clusters']], on = 'Customer.ID',␣
↪how='left')
```

[96]:
```
non_outlier = final[final['Clusters']>=0]
outlier = final[final['Clusters']<0]
```

[97]:
```
markers = ['o', 's', '^', 'D']

fig = plt.figure(figsize=(28,15))
ax = fig.add_subplot(121, projection='3d')


unique_clusters = non_outlier['Clusters'].unique()
cmap = plt.get_cmap('Set2')


for i, cluster in enumerate(unique_clusters):
    cluster_data = non_outlier[non_outlier['Clusters'] == cluster]
    ax.scatter(cluster_data['Number of vehicles in household'],
               cluster_data['Total Tires'],
               cluster_data['Number of purchases made in the store'],
               c=cmap(i / len(unique_clusters)),
               s=100,
               marker=markers[i % len(markers)],
               label=f'Cluster {cluster}')

ax.set_xlabel('Number of vehicles in household')
ax.set_ylabel('Total Tires')
ax.set_zlabel('Number of purchases made in the store')
ax.set_title('Non Outlier Cluster Identification')


ax.legend(title="Clusters")
```

[97]:
```
<matplotlib.legend.Legend at 0x1769a48a010>
```



7.0.1 1 = People with 2 Vehicles and low to medium instore purchase and high tire purchase

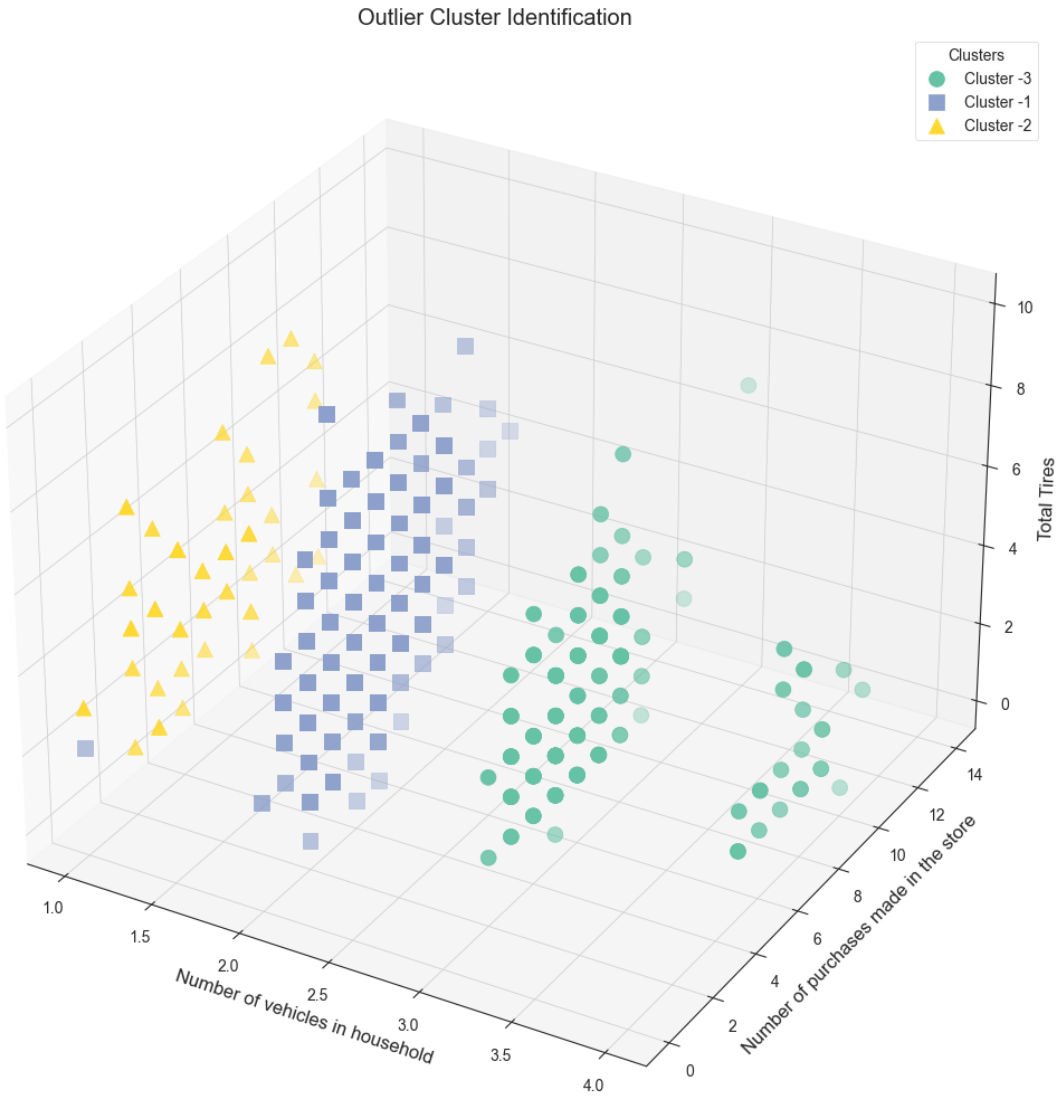
7.0.2 0 = People with 1 Vehicle and medium instore purchase low to mid tire purchase

7.0.3 2 = People with 1 Vehicle and low in-store purchase and low to mid tire purchase

[98]:
```
markers = ['o', 's', '^', 'D']

fig = plt.figure(figsize=(28,15))
ax = fig.add_subplot(121, projection='3d')
```

```
unique_clusters = outlier['Clusters'].unique()
cmap = plt.get_cmap('Set2')


for i, cluster in enumerate(unique_clusters):
    cluster_data = outlier[outlier['Clusters'] == cluster]
    ax.scatter(cluster_data['Number of vehicles in household'],
               cluster_data['Number of purchases made in the store'],
               cluster_data['Total Tires'],
               c=cmap(i / len(unique_clusters)),
               s=100,
               marker=markers[i % len(markers)],
               label=f'Cluster {cluster}')

ax.set_xlabel('Number of vehicles in household')
ax.set_ylabel('Number of purchases made in the store')
ax.set_zlabel('Total Tires')
ax.set_title('Outlier Cluster Identification')


ax.legend(title="Clusters")
```

[98]: <matplotlib.legend.Legend at 0x1769a482050>



7.0.4 -3 = 3 or 4 Vehicles in the household and a very high number of purchases in store but are not buying too many tires

7.0.5 -2 = 2 Vehicles in the household medium number of purchases in store and decent number of set of tires

7.0.6 -1 = 1 Vehicle in household with low number of instore purchases but 2 to 4 set of tires sets

[99]:
```
names = {
    1 : 'Serv Push',
    2 : 'Serv Push',
    0 : 'Retain',
```

```
    -3 : 'Tire Push',
    -1 : 'Serv Push',
    -2 : 'Retain'
}
```

[100]:
```
final['Category'] = final['Clusters'].map(names)
```

[101]:
```
final['Category'].value_counts()
```

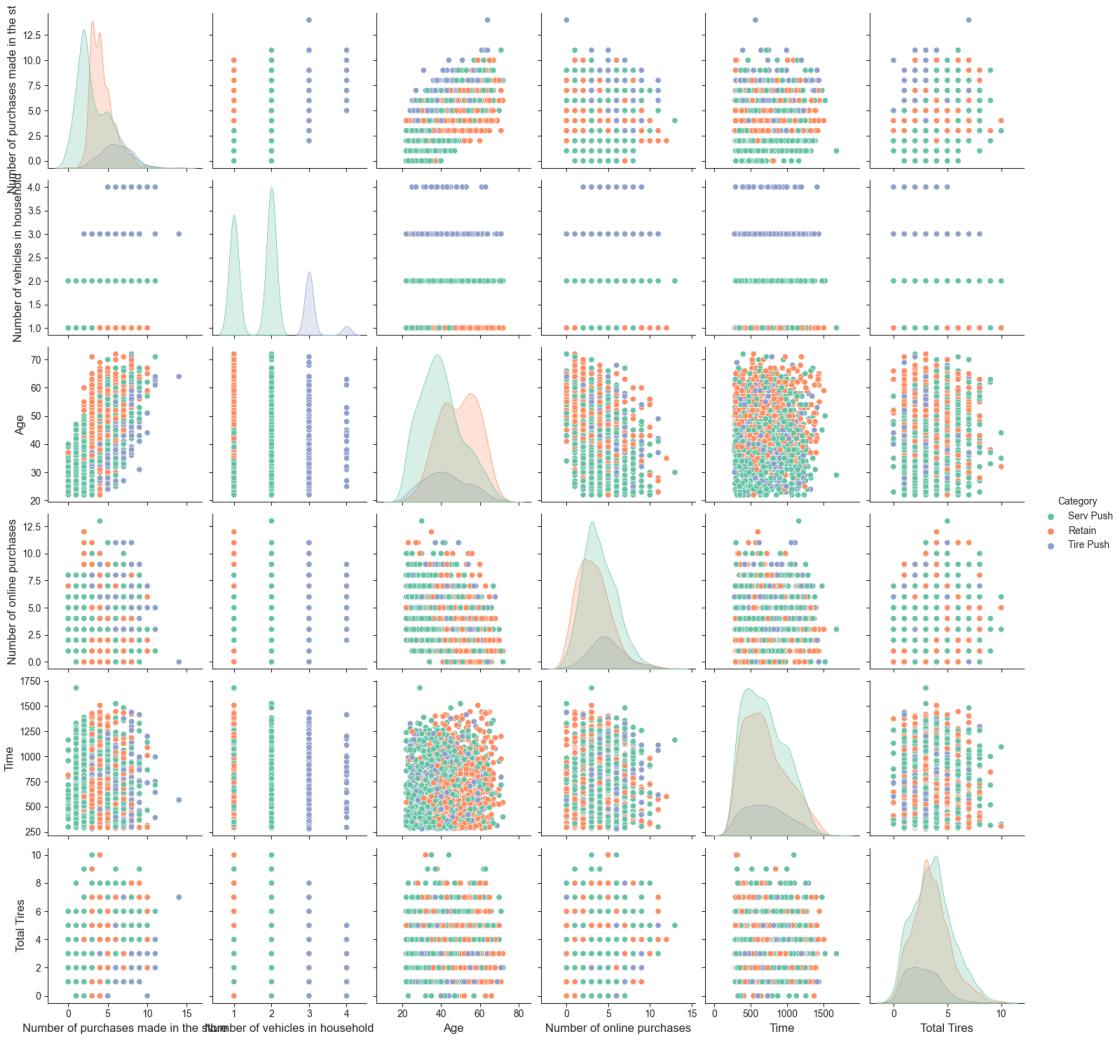
[101]:
```
Category
Serv Push    816
Retain       664
Tire Push    192
Name: count, dtype: int64
```

[102]:
```
final = final.sort_values(ascending=True,by='Customer onboard date')
```

[103]:
```
visual = final[['Number of purchases made in the store','Number of vehicles in␣
↪household',
                'Age','2nd Vehicle','Number of online purchases','3rd␣
↪Vehicle','Category','Time','Total Tires']].reset_index(drop=True)
```

[104]:
```
sns.pairplot(visual, hue='Category',palette='Set2')
```

[104]:
```
<seaborn.axisgrid.PairGrid at 0x1769aaa9110>
```



[105]:
```
#final.to_csv('Final.csv',index=False)
```

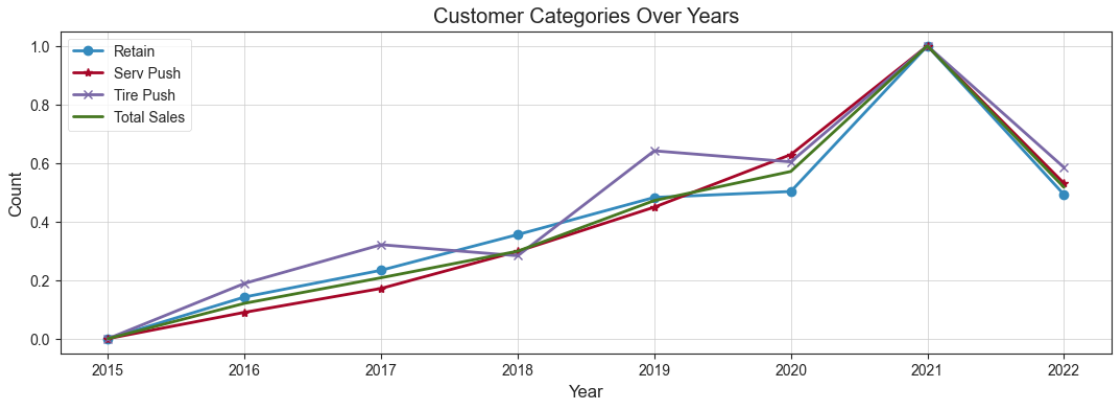
[106]:
```
final['Category'].unique()
```

[106]:
```
array(['Serv Push', 'Retain', 'Tire Push'], dtype=object)
```

[107]:
```
Retain = final[final['Category'] == 'Retain'].groupby('Onboard year').size().
 ↪values
Serv = final[final['Category'] == 'Serv Push'].groupby('Onboard year').size().
 ↪values
Tire = final[final['Category'] == 'Tire Push'].groupby('Onboard year').size().
 ↪values
Tire = np.insert(Tire,0,0)
Sales = final.groupby('Onboard year')['Total Tires'].sum().values
```

[108]:
```
Scaler = MinMaxScaler()

Retain = Scaler.fit_transform(Retain.reshape(-1, 1))
Serv = Scaler.fit_transform(Serv.reshape(-1, 1))
Tire = Scaler.fit_transform(Tire.reshape(-1, 1))
Sales = Scaler.fit_transform(Sales.reshape(-1,1))
```

[109]:
```
years = np.array([2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022])

plt.figure(figsize=(13, 4))
plt.plot(years, Retain, label='Retain', marker='o')
plt.plot(years, Serv, label='Serv Push', marker='*')
plt.plot(years, Tire, label='Tire Push', marker='x')
plt.plot(years, Sales, label='Total Sales', marker=' ')

plt.xlabel('Year')
plt.ylabel('Count')
plt.title('Customer Categories Over Years')
plt.legend()
plt.grid(True)
plt.show()
```



[110]:
```
final.groupby('Category')['Time'].max()
```

[110]:
```
Category
Retain       1508
Serv Push    1681
Tire Push    1439
Name: Time, dtype: int64
```

[ ]:
```

```