In [31]:
```
# Import required libraries and dependencies
import pandas as pd
import hvplot.pandas
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Load the data into a Pandas DataFrame
df_market_data = pd.read_csv(
    "Resources/crypto_market_data.csv",
    index_col="coin_id")

# Display sample data
df_market_data.head(10)
```

Out[3]:

coin_id	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14
bitcoin	1.08388	7.60278	6.575
ethereum	0.22392	10.38134	4.808
tether	-0.21173	0.04935	0.006
ripple	-0.37819	-0.60926	2.249
bitcoin-cash	2.90585	17.09717	14.753
binancecoin	2.10423	12.85511	6.806
chainlink	-0.23935	20.69459	9.300
cardano	0.00322	13.99302	5.554
litecoin	-0.06341	6.60221	7.289
bitcoin-cash-sv	0.92530	3.29641	-1.866

In [4]:
```
# Generate summary statistics
df_market_data.describe()
```

Out[4]:

	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d	pr
count	41.000000	41.000000	41.000000	
mean	-0.269686	4.497147	0.185787	
std	2.694793	6.375218	8.376939	
min	-13.527860	-6.094560	-18.158900	
25%	-0.608970	0.047260	-5.026620	
50%	-0.063410	3.296410	0.109740	
75%	0.612090	7.602780	5.510740	
max	4.840330	20.694590	24.239190	

In [5]:
```
# Plot your data to see what's in your DataFrame
df_market_data.hvplot.line(
    width=800,
    height=400,
    rot=90
)
```

Out[5]:

Prepare the Data

In [6]:
```
column_names = df_market_data.columns.tolist()
data_values = df_market_data.values
```

In [7]:
```
# Use the `StandardScaler()` module from scikit-learn to normalize the data fro
scaler = StandardScaler()
scaled_df_market_data = scaler.fit_transform(data_values)
```

In [8]:
```
# Create a DataFrame with the scaled data
scaled_df = pd.DataFrame(scaled_df_market_data, columns=column_names)

# Copy the crypto names from the original data
crypto_coin_ids = df_market_data.index

# Set the coinid column as index
scaled_df.set_index(crypto_coin_ids, inplace=True)

# Display sample data
scaled_df.head(6)
```

Out[8]:

coin_id	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14
bitcoin	0.508529	0.493193	0.7722(
ethereum	0.185446	0.934445	0.5586(
tether	0.021774	-0.706337	-0.0216(
ripple	-0.040764	-0.810928	0.2494(
bitcoin-cash	1.193036	2.000959	1.7606
binancecoin	0.891871	1.327295	0.8002

Find the Best Value for k Using the Original Data.

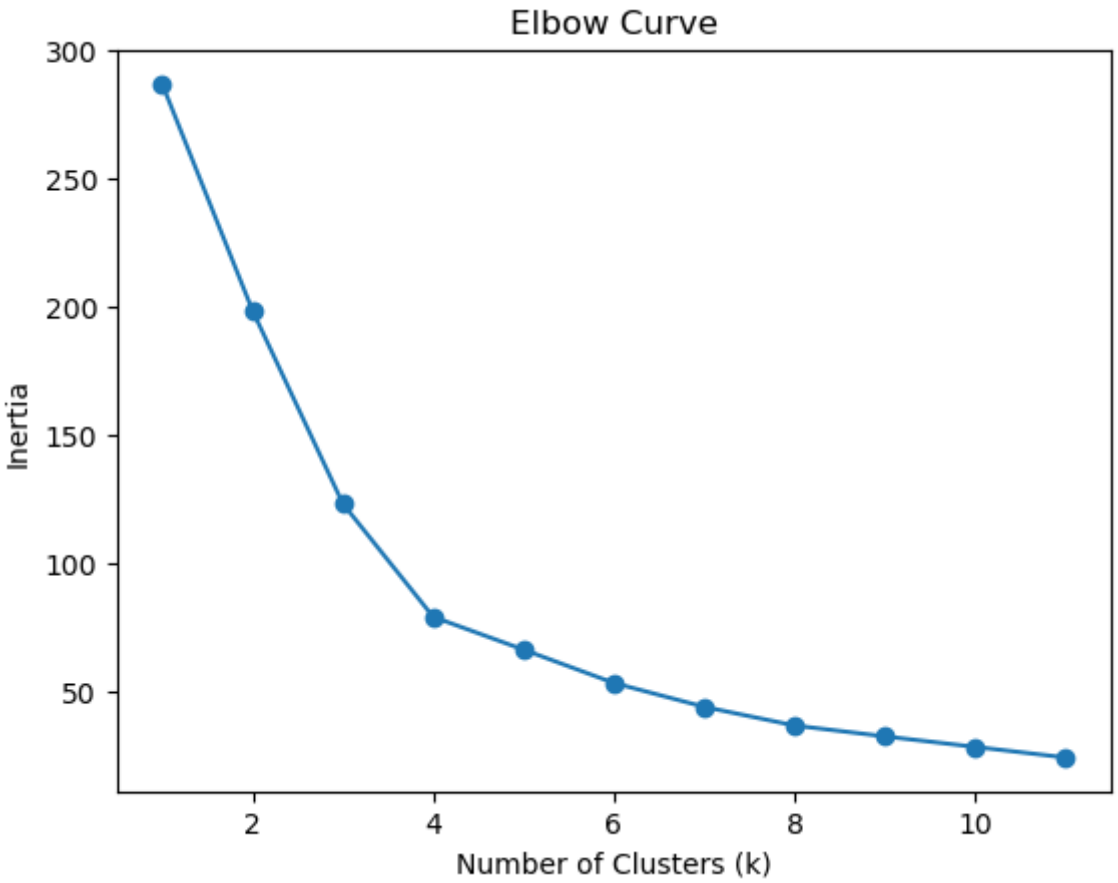
In [25]:
```
# Create a list with the number of k-values from 1 to 11
k_values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
k_values
```

Out[25]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

In [26]:
```
# Create an empty list to store the inertia values
inertia_values = list()
# Create a for loop to compute the inertia with each possible value of k
# Inside the loop:
# 1. Create a KMeans model using the loop counter for the n_clusters
# 2. Fit the model to the data using `df_market_data_scaled`
# 3. Append the model.inertia_ to the inertia list
for k in range(1, 12):
    kmeans_model = KMeans(n_clusters=k)
    kmeans_model.fit(scaled_df)
    inertia_values.append(kmeans_model.inertia_)
```

In [28]:
```
# Create a dictionary with the data to plot the Elbow curve
elbow_data = {
    'k': k_values,
    'inertia': inertia_values
}

# Create a DataFrame with the data to plot the Elbow curve
elbow_df = pd.DataFrame(elbow_data)
```

In [32]:
```
# Plot a line chart with all the inertia values computed with
# the different values of k to visually identify the optimal value for k.
plt.plot(elbow_df['k'], elbow_df['inertia'], marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Curve')
plt.show()
```



Answer the following question:

Question: What is the best value for `k`?

Answer: As given from the elbow curve, as the elbow is formed at 4, therefore, the best value of `k` in this case is **4**

Cluster Cryptocurrencies with K-means Using the Original Data

In [33]:
```
# Initialize the K-Means model using the best value for k
kmeans_optimal = KMeans(n_clusters=4)
```

In [34]:
```
# Fit the K-Means model using the scaled data
kmeans_optimal.fit(scaled_df)
```

Out[34]:
```
KMeans
KMeans(n_clusters=
```

In [36]:
```
# Predict the clusters to group the cryptocurrencies using the scaled data
cluster_labels = kmeans_optimal.fit_predict(scaled_df)

# Print the resulting array of cluster values.
print(cluster_labels)
```

```
[3 3 0 0 3 3 3 3 3 0 0 0 0 3 0 3 0 0 3 0 0 3 0 0 0 0 0 0 3 0 0 0 1 3 0 0 2
 0 0 0 0 0]
```

In [37]:
```
# Create a copy of the DataFrame
scaled_df_copy = scaled_df.copy()
```

In [39]:
```
# Add a new column to the DataFrame with the predicted clusters
scaled_df_copy['predicted_cluster'] = cluster_labels

# Display sample data
scaled_df_copy.head(5)
```

Out[39]:

coin_id	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d
bitcoin	0.508529	0.493193	0.772200
ethereum	0.185446	0.934445	0.558692
tether	0.021774	-0.706337	-0.021680
ripple	-0.040764	-0.810928	0.249458
bitcoin-cash	1.193036	2.000959	1.760610

In [44]:
```
# Create a scatter plot using hvPlot by setting
# `x="price_change_percentage_24h"` and `y="price_change_percentage_7d"`.
# Color the graph points with the labels found using K-Means and
# add the crypto name in the `hover_cols` parameter to identify
# the cryptocurrency represented by each data point.
scatter_plot = scaled_df_copy.hvplot.scatter(
    x="price_change_percentage_24h",
    y="price_change_percentage_7d",
    c="predicted_cluster",
    cmap="viridis",
    hover_cols=["crypto_name"]
)

scatter_plot
```

Out[44]:

Optimize Clusters with Principal Component Analysis.

In [46]:
```
# Create a PCA model instance and set `n_components=3`.
pca = PCA(n_components=3)
pca
```

Out[46]:
```
PCA
PCA(n_components=
```

In [50]:
```
# Use the PCA model with `fit_transform` to reduce to
# three principal components.
scaled_df_copy_pca = pca.fit_transform(scaled_df_copy)
scaled_df_copy_pca_df = pd.DataFrame(scaled_df_copy_pca, columns=['PC1', 'PC2', 'P

# View the first five rows of the DataFrame.
scaled_df_copy_pca_df.head(5)
```

Out[50]:

	PC1	PC2	PC3
0	2.059139	-0.536684	-0.438238
1	1.956401	-0.410821	-1.016966
2	-0.990142	-0.454228	0.690021
3	-0.973189	-0.494650	0.551234
4	3.330367	-1.022311	-0.501341

In [53]:

```
# Retrieve the explained variance to determine how much information
# can be attributed to each principal component.
explained_variance = pca.explained_variance_ratio_
explained_variance_df = pd.DataFrame(explained_variance, columns=['Explained Varia
explained_variance_df
```

Out[53]:

	Explained Variance
0	0.389392
1	0.291660
2	0.208101

In [55]:

```
total_explained_variance = sum(explained_variance)
print("Total Explained Variance: %.6f" % (total_explained_variance))
```

```
Total Explained Variance: 0.889153
```

Answer the following question:

Question: What is the total explained variance of the three principal components?

Answer: The total explained variance of 3 principal components is 0.889153

In [61]:

```
# Create a new DataFrame with the PCA data.

# Creating a DataFrame with the PCA data
df_market_data_pca = scaled_df_copy_pca_df

# Copy the crypto names from the original data
crypto_names = scaled_df_copy.index

# Set the coinid column as index
df_market_data_pca.set_index(crypto_names, inplace=True)

# Display sample data
df_market_data_pca.head(5)
```

Out[61]:

coin_id	PC1	PC2	PC3
bitcoin	2.059139	-0.536684	-0.438238
ethereum	1.956401	-0.410821	-1.016966
tether	-0.990142	-0.454228	0.690021
ripple	-0.973189	-0.494650	0.551234
bitcoin-cash	3.330367	-1.022311	-0.501341

Find the Best Value for k Using the PCA Data

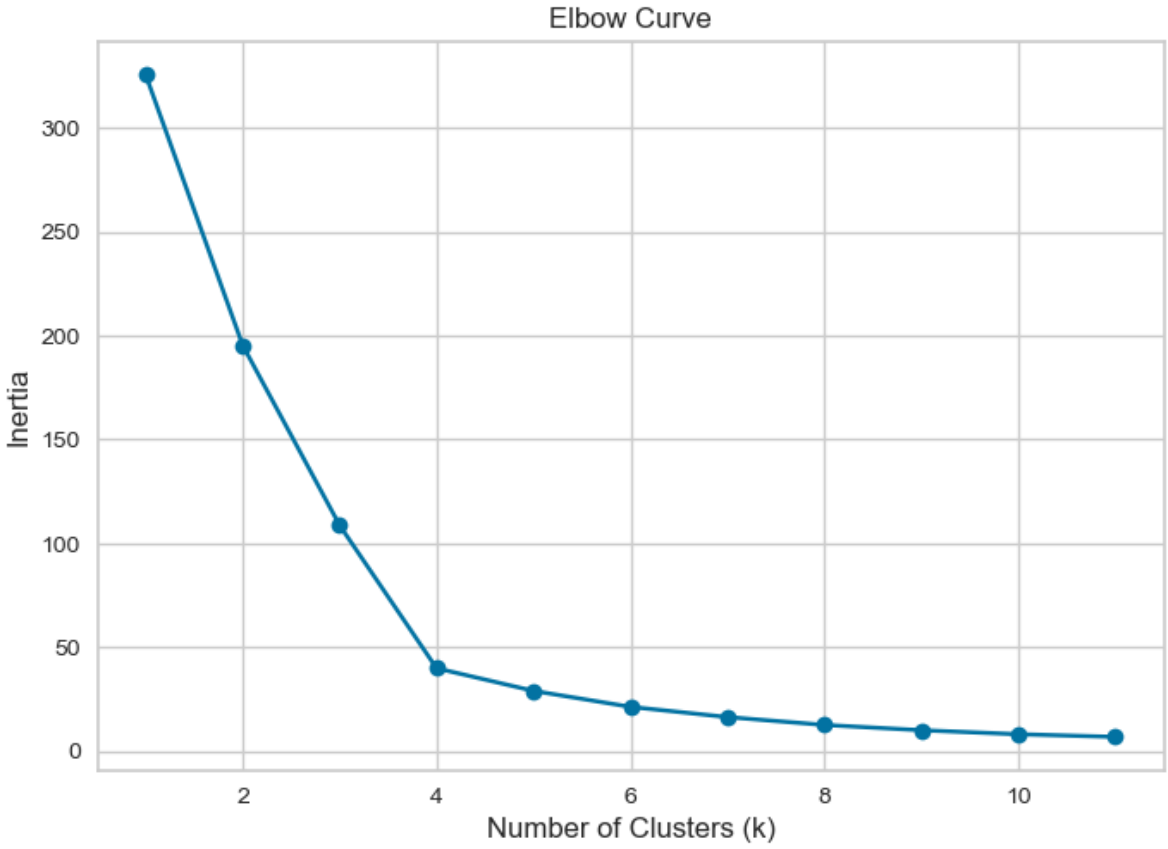
In [60]:
```
# Create a list with the number of k-values from 1 to 11
k_values = list(range(1, 12))
k_values
```

Out[60]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

In [62]:
```
# Create an empty list to store the inertia values
inertia_values = list()

# Create a for loop to compute the inertia with each possible value of k
# Inside the loop:
# 1. Create a KMeans model using the loop counter for the n_clusters
# 2. Fit the model to the data using `df_market_data_pca`
# 3. Append the model.inertia_ to the inertia list
for k in range(1, 12):
    kmeans_model = KMeans(n_clusters=k)
    kmeans_model.fit(df_market_data_pca)
    inertia_values.append(kmeans_model.inertia_)
```

In [74]:
```
# Create a dictionary with the data to plot the Elbow curve
elbow_data_pca = {
    'k': k_values,
    'inertia': inertia_values
}

# Create a DataFrame with the data to plot the Elbow curve
elbow_df_pca = pd.DataFrame(elbow_data_pca)
```

In [75]:
```
# Plot a line chart with all the inertia values computed with
# the different values of k to visually identify the optimal value for k.
plt.plot(elbow_df_pca['k'], elbow_df_pca['inertia'], marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Curve')
plt.show()
```



Answer the following questions:

- **Question:** What is the best value for `k` when using the PCA data?
  - **Answer:** The best value of `k` when using the PCA data is **4**
- **Question:** Does it differ from the best k value found using the original data?
  - **Answer:** Nom the value of `k` didn't differ using the original data

Cluster Cryptocurrencies with K-means Using the PCA Data

In [77]:
```
# Initialize the K-Means model using the best value for k
kmeans_best = KMeans(n_clusters=4)
```

In [78]:
```
# Fit the K-Means model using the PCA data
kmeans_best.fit(df_market_data_pca)
```

Out[78]:
```
KMeans
KMeans(n_clusters=
```

In [79]:
```
# Predict the clusters to group the cryptocurrencies using the PCA data
kmeans_best.fit_predict(df_market_data_pca)

# Print the resulting array of cluster values.
cluster_labels_data_pca = kmeans_optimal.fit_predict(df_market_data_pca)
print(cluster_labels_data_pca)
```

```
[1 1 0 0 1 1 1 1 1 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 2 1 0 0 3
 0 0 0 0]
```

In [80]:
```
# Create a copy of the DataFrame with the PCA data
df_market_data_pca_copy = df_market_data_pca.copy()

# Add a new column to the DataFrame with the predicted clusters
df_market_data_pca_copy['predicted_cluster'] = cluster_labels_data_pca

# Display sample data
df_market_data_pca_copy.head(5)
```

Out[80]:

coin_id	PC1	PC2	PC3	predicted_cluster
bitcoin	2.059139	-0.536684	-0.438238	1
ethereum	1.956401	-0.410821	-1.016966	1
tether	-0.990142	-0.454228	0.690021	0
ripple	-0.973189	-0.494650	0.551234	0
bitcoin-cash	3.330367	-1.022311	-0.501341	1

In [81]:
```
# Create a scatter plot using hvPlot by setting
# `x="PC1"` and `y="PC2"`.
# Color the graph points with the labels found using K-Means and
# add the crypto name in the `hover_cols` parameter to identify
# the cryptocurrency represented by each data point.
scatter_plot = df_market_data_pca_copy.hvplot.scatter(
    x="PC1",
    y="PC2",
    c="predicted_cluster",
    cmap="viridis",
    hover_cols=["crypto_name"]
)

scatter_plot
```

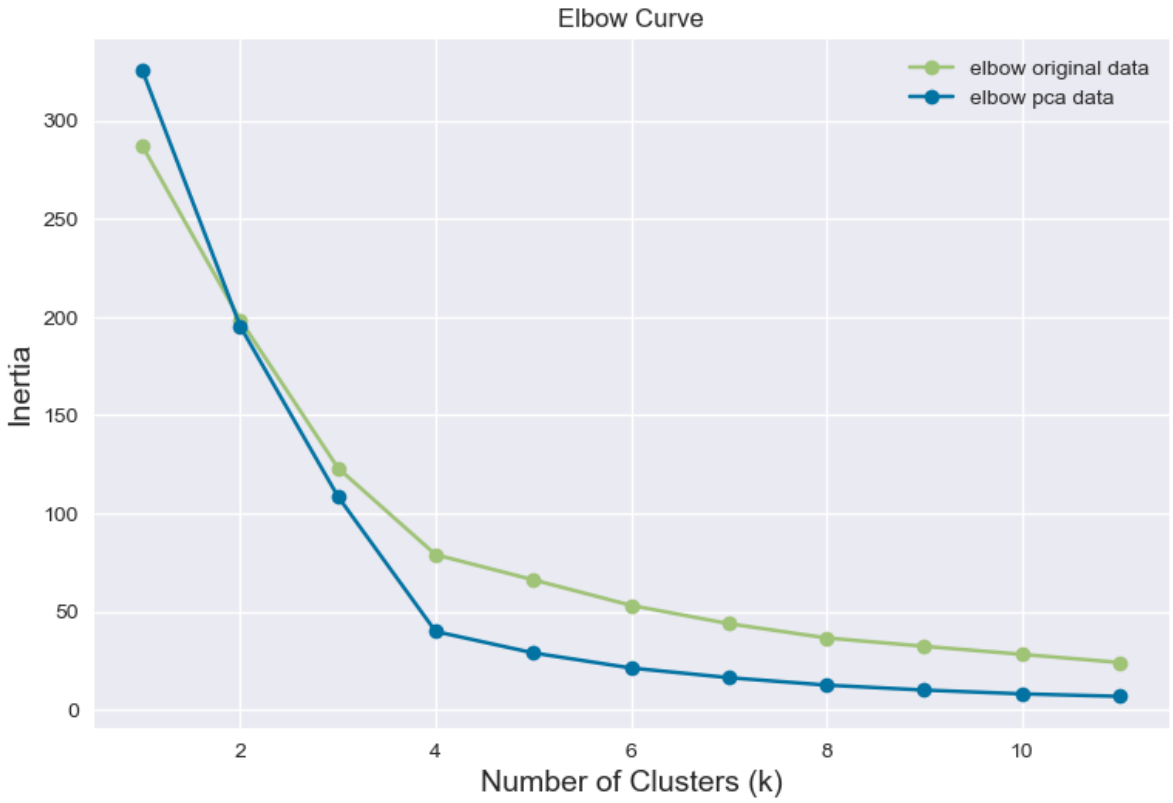
Out[81]:

Visualize and Compare the Results

In this section, you will visually analyze the cluster analysis results by contrasting the outcome with and without using the optimization techniques.

In [132…
```
# Composite plot to contrast the Elbow curves
plt.style.use('seaborn')
k_values_original = elbow_df['k']
inertia_values_original = elbow_df['inertia']

k_values_pca = elbow_df_pca['k']
inertia_values_pca = elbow_df_pca['inertia']

fig, ax = plt.subplots()
ax.plot(k_values_original, inertia_values_original, marker='o', color='g',
        label='elbow original data')
ax.plot(k_values_pca, inertia_values_pca, marker='o', color='b',
        label='elbow pca data')
ax.set_xlabel('Number of Clusters (k)')
ax.set_ylabel('Inertia')
ax.set_title('Elbow Curve')
```

```
ax.legend()
plt.show()
```



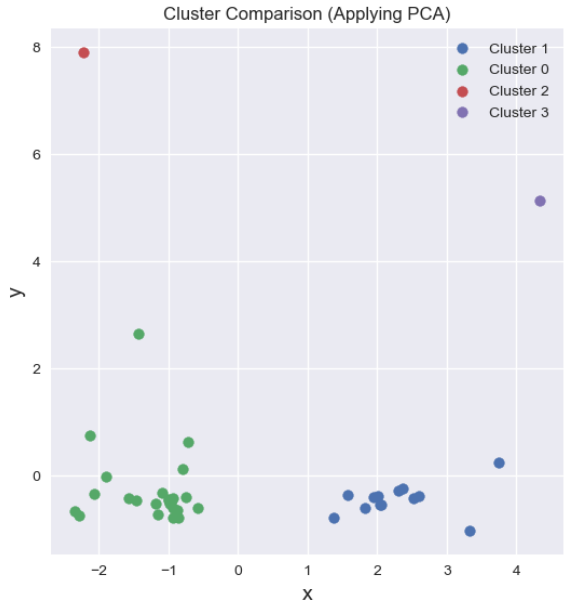
In [126…

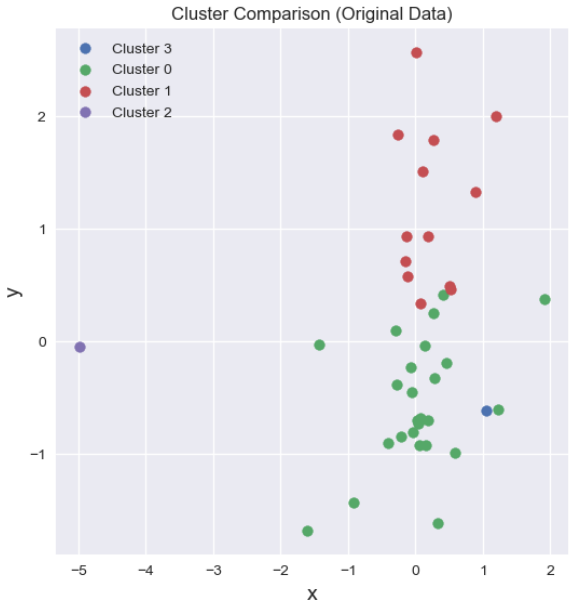
```
df_cluster_labels_pca = pd.DataFrame({'Label':cluster_labels_data_pca})
df_clsuter_labels = pd.DataFrame({'Label':cluster_labels})

labels = df_market_data_pca_copy.predicted_cluster
labels_original = scaled_df_copy.predicted_cluster
```

In [138…

```
# Composite plot to contrast the clusters
# YOUR CODE HERE!

fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax1 = ax[0]
for cluster_label in df_cluster_labels_pca.Label.unique():
    cluster_data = df_market_data_pca_copy[labels == cluster_label]

    ax1.scatter(cluster_data['PC1'],
                cluster_data['PC2'],
                label=f'Cluster {cluster_label}')

ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title('Cluster Comparison (Applying PCA)')
ax1.legend()

ax2 = ax[1]
for cluster_label in df_clsuter_labels.Label.unique():
    cluster_data = scaled_df_copy[labels == cluster_label]

    ax2.scatter(cluster_data['price_change_percentage_24h'],
                cluster_data['price_change_percentage_7d'],
                label=f'Cluster {cluster_label}')

ax2.set_xlabel('x')
ax2.set_ylabel('y')
```

```
ax2.set_title('Cluster Comparison (Original Data)')

ax2.legend()
plt.show()
```





Answer the following question:

- **Question:** After visually analyzing the cluster analysis results, what is the impact of using fewer features to cluster the data using K-Means?
- **Answer:** After visualizing the cluster analysis results, it can be seen that clsuters tend to be more precise and to the point using PCA. PCA redueces the dimensionality of original data and only includes those features that matter the most.

In [ ]: