CH 19

In [1]:

```python
# Import the required modules
import pandas as pd
pd.set_option('display.max_columns', None)
import numpy as np

# visualization
import matplotlib.pyplot as plt
import hvplot.pandas
import seaborn as sns

# Machine Learning
from sklearn.cluster import KMeans, AgglomerativeClustering, Birch
from sklearn.metrics import silhouette_score, calinski_harabasz_score
from sklearn.manifold import TSNE

# Preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# suppress warnings
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```python
# 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[2]:

coin_id	price_change_percentage_24h	price_change_percentage_7d	price_change_perce
bitcoin	1.08388	7.60278	
ethereum	0.22392	10.38134	
tether	-0.21173	0.04935	
ripple	-0.37819	-0.60926	
bitcoin-cash	2.90585	17.09717	
binancecoin	2.10423	12.85511	
chainlink	-0.23935	20.69459	
cardano	0.00322	13.99302	
litecoin	-0.06341	6.60221	
bitcoin-cash-sv	0.92530	3.29641	

In [3]:

```python
df_market_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 41 entries, bitcoin to digibyte
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   price_change_percentage_24h   41 non-null     float64
 1   price_change_percentage_7d    41 non-null     float64
 2   price_change_percentage_14d   41 non-null     float64
 3   price_change_percentage_30d   41 non-null     float64
 4   price_change_percentage_60d   41 non-null     float64
 5   price_change_percentage_200d  41 non-null     float64
 6   price_change_percentage_1y    41 non-null     float64
dtypes: float64(7)
memory usage: 2.6+ KB
```

In [4]:

```
# Generate summary statistics
df_market_data.describe()
```

Out[4]:

	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_
count	41.000000	41.000000	41.000
mean	-0.269686	4.497147	0.185
std	2.694793	6.375218	8.376
min	-13.527860	-6.094560	-18.158
25%	-0.608970	0.047260	-5.026
50%	-0.063410	3.296410	0.109
75%	0.612090	7.602780	5.510
max	4.840330	20.694590	24.239

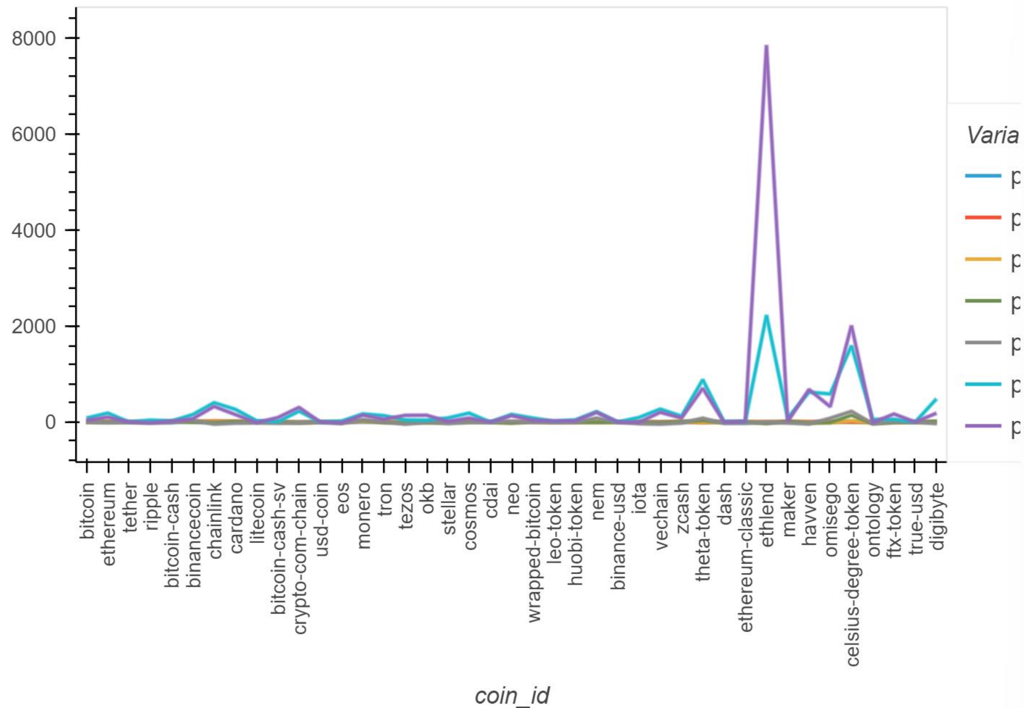
In [5]:

```
# we need a scaler due to variability in the max for each range
```

In [6]:

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

Out[6]:



Prepare the Data

In [7]:

```
# are we independent? or suffer from multi-collinearity?
corrs = df_market_data.corr()
corrs
```

Out[7]:

	price_change_percentage_24h	price_change_percentage_7d
price_change_percentage_24h	1.000000	0.169659
price_change_percentage_7d	0.169659	1.000000
price_change_percentage_14d	0.279682	0.538294
price_change_percentage_30d	0.292563	0.056899
price_change_percentage_60d	0.136974	-0.145099
price_change_percentage_200d	-0.541190	-0.052533
price_change_percentage_1y	-0.750630	-0.038424

In [8]:

```
sns.heatmap(corrs)
plt.show
```

Out[8]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

In [9]:

```
df_market_data.columns
```

Out[9]:

```
Index(['price_change_percentage_24h', 'price_change_percentage_7d',
       'price_change_percentage_14d', 'price_change_percentage_30d',
       'price_change_percentage_60d', 'price_change_percentage_200d',
       'price_change_percentage_1y'],
      dtype='object')
```

In [10]:

```
# Use the `StandardScaler()` module from scikit-learn to normalize the
num_features = [ 'price_change_percentage_24h', 'price_change_percenta
        'price_change_percentage_14d', 'price_change_percentage_30d',
        'price_change_percentage_60d', 'price_change_percentage_200d',
        'price_change_percentage_1y']
```

In [11]:

```
# subset
df_sub = df_market_data.loc[:, num_features]

# initialize
scaler = StandardScaler()

# fit
scaler.fit(df_sub)

# predict/transform
scaled_data = scaler.transform(df_sub)
df_scaled = pd.DataFrame(scaled_data, columns=num_features)


df_scaled.head()
```

Out[11]:

	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d
0	0.508529	0.493193	0.772200
1	0.185446	0.934445	0.558692
2	0.021774	-0.706337	-0.021680
3	-0.040764	-0.810928	0.249458
4	1.193036	2.000959	1.760610

In [12]:

```
df_scaled.describe()
```

Out[12]:

	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_
count	41.000000	4.100000e+01	4.100000e
mean	0.000000	1.895503e-16	2.707861
std	1.012423	1.012423e+00	1.012423e
min	-4.981042	-1.682027e+00	-2.217108e
25%	-0.127467	-7.066688e-01	-6.299628
50%	0.077497	-1.906843e-01	-9.190922
75%	0.331280	4.931931e-01	6.435649
max	1.919812	2.572251e+00	2.907054e

Find the Best Value for k Using the Original Data.

In [13]:

```
# Create a list with the number of k-values from 1 to 11
X = df_scaled.loc[:, num_features]
X.head()
```

Out[13]:

	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d
0	0.508529	0.493193	0.772200
1	0.185446	0.934445	0.558692
2	0.021774	-0.706337	-0.021680
3	-0.040764	-0.810928	0.249458
4	1.193036	2.000959	1.760610

In [14]:

```
 # Create a a list to store inertia values
inertia = []
silhouettes = []
cha_chas = []

# Create a a list to store the values of k
k = list(range(2, 11))

# Create a for-loop where each value of k is evaluated using the K-mec
# Fit the model using the spread_df DataFrame
# Append the value of the computed inertia from the `inertia_` attribu
for i in k:
    # initialize the model
    k_model = KMeans(n_clusters=i, random_state=1)

    # fit the model
    k_model.fit(X)

    # predict the model
    preds = k_model.predict(X)

    # evaluate the model (generate the metics)
    inertia.append(k_model.inertia_)
    score = silhouette_score(X, preds)
    silhouettes.append(score)

    cha_cha = calinski_harabasz_score(X, preds)
    cha_chas.append(cha_cha)

    print(f"Finished {i} out of {max(k)}")
```

In [15]:

```
# Define a DataFrame to hold the values for k and the corresponding i
elbow_data = {"k": k, "inertia": inertia, "silhouette_score": silhouet
df_elbow = pd.DataFrame(elbow_data)

df_elbow["acc"] = df_elbow.inertia.diff()

# Review the DataFrame
df_elbow.head(10)
```

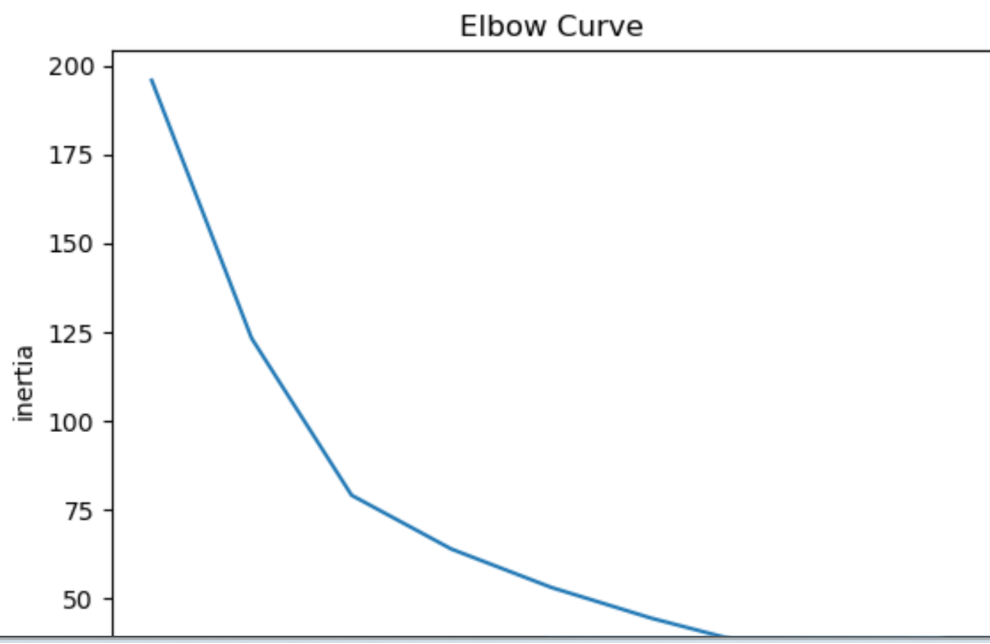
Out[15]:

	k	inertia	silhouette_score	cha_score	acc
0	2	195.820218	0.651576	18.159573	NaN
1	3	123.190482	0.702822	25.264783	-72.629736
2	4	79.022435	0.314482	32.459853	-44.168046
3	5	63.858668	0.329023	31.448698	-15.163768
4	6	53.057788	0.287883	30.864375	-10.800879
5	7	44.406791	0.290874	30.956861	-8.650998
6	8	37.078233	0.205692	31.776126	-7.328557
7	9	32.832187	0.258600	30.965687	-4.246046
8	10	28.165433	0.244422	31.653739	-4.666754

In [16]:

```python
# Plot a line chart with all the inertia values computed with
# the different values of k to visually identify the optimal value for

# Plot the DataFrame
plt.plot(df_elbow["k"], df_elbow["inertia"])
plt.title("Elbow Curve")
plt.xticks(df_elbow["k"])
plt.ylabel("inertia")
plt.xlabel("k")
plt.show()
```



Answer the following question:

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

Answer: 4 or 5

Cluster Cryptocurrencies with K-means Using the Original Data

In [17]:
```
# Initialize the K-Means model using the best value for k
model = KMeans(n_clusters=4, random_state=1)
```

In [18]:
```
# Fit the K-Means model using the scaled data
model.fit(X)
```

Out[18]:
```
KMeans
KMeans(n_clusters=4, random_state=1)
```

In [19]:
```
# Predict the clusters to group the cryptocurrencies using the scaled
preds = model.predict(X)

```

In [20]:
```
# Add a new column to the DataFrame with the predicted clusters
df2 = df_scaled.copy()
df2['clusters'] = preds


# Display sample data
df2.head()
```

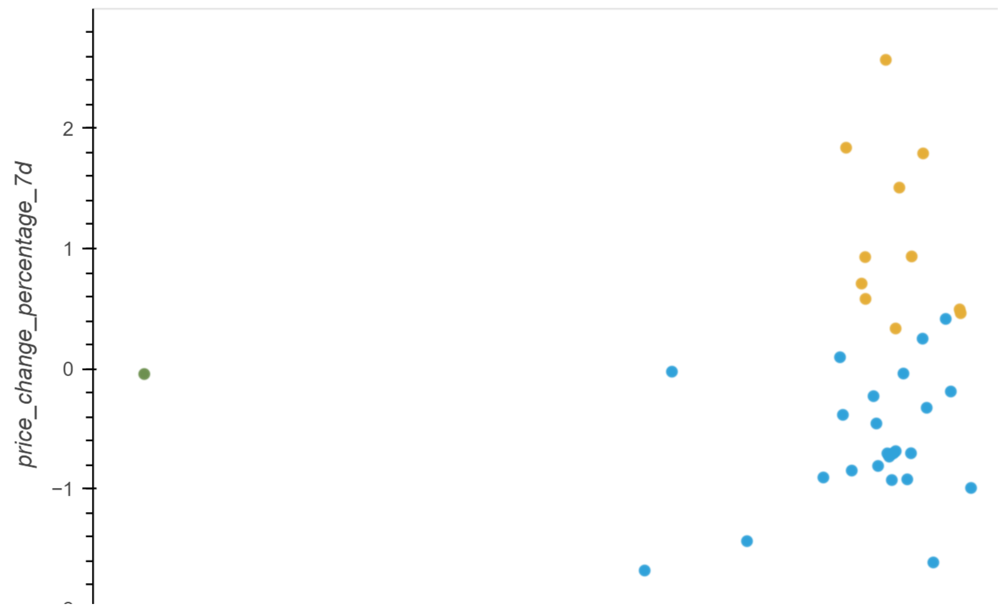
Out[20]:

	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d
0	0.508529	0.493193	0.772200
1	0.185446	0.934445	0.558692
2	0.021774	-0.706337	-0.021680

In [21]:

```python
# 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.
df2.hvplot.scatter(
    width=800,
    height=400,
    x="price_change_percentage_24h",
    y="price_change_percentage_7d",
    by="clusters",
    hover_cols="coin_id"
)
```

Out[21]:



In [22]:

```python
df2.clusters.value_counts()
```

Out[22]:

```
clusters
0    26
2    13
3     1
1     1
Name: count, dtype: int64
```

Optimize Clusters with Principal Component Analysis.

In [23]:

```
# Instantiate the PCA instance and declare the number of PCA variables
num_pca = len(num_features)
pca = PCA(n_components=num_pca)

# Fit the PCA model on the transformed credit card DataFrame
data_pca = pca.fit_transform(df_scaled.loc[:, num_features])

# Create the PCA DataFrame
df_pca = pd.DataFrame(
    data_pca,
    columns=[f"PCA{x+1}" for x in range(num_pca)]
)

df_pca.head()
```

Out[23]:

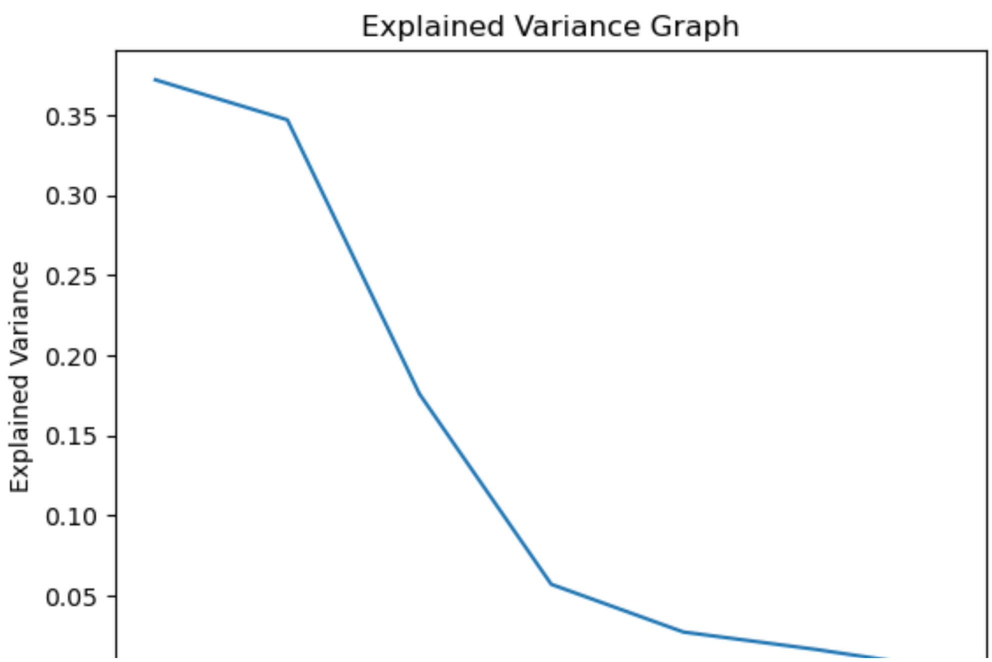
	PCA1	PCA2	PCA3	PCA4	PCA5	PCA6	PCA7
0	-0.600667	0.842760	0.461595	-0.109151	-0.033786	-0.225703	0.006595
1	-0.458261	0.458466	0.952877	0.095100	0.014588	0.034158	0.109593
2	-0.433070	-0.168126	-0.641752	-0.470282	0.115300	-0.127710	-0.086857
3	-0.471835	-0.222660	-0.479053	-0.737473	-0.148641	-0.273472	0.134870
4	-1.157800	2.041209	1.859715	0.236479	-0.191787	-0.411513	-0.070411

In [24]:

```python
# Can we reduce the dimensions?

# Calculate the PCA explained variance ratio
exp_var = pca.explained_variance_ratio_

plt.plot(range(1, num_pca + 1), exp_var)
plt.title("Explained Variance Graph")
plt.xlabel("PCA #")
plt.ylabel("Explained Variance")
plt.xticks(range(1, num_pca + 1))
plt.show()
```



In [25]:

```
# Use the PCA model with `fit_transform` to reduce to
# three principal components.

# View the first five rows of the DataFrame. (BOOOTH like 4 better)
df3 = df_pca.loc[:, ["PCA1", "PCA2", "PCA3", "PCA4"]]
df3.head()
```

Out[25]:

	PCA1	PCA2	PCA3	PCA4
0	-0.600667	0.842760	0.461595	-0.109151
1	-0.458261	0.458466	0.952877	0.095100
2	-0.433070	-0.168126	-0.641752	-0.470282
3	-0.471835	-0.222660	-0.479053	-0.737473
4	-1.157800	2.041209	1.859715	0.236479

In [26]:

```
print("Explained Variance")
for i in range(len(exp_var)):
    val = exp_var[i]
    print(f"PCA{i+1}:", round(val, 3))

print()
print("CUMULATIVE Explained Variance")

exp_var_cum = np.cumsum(exp_var)
for i in range(len(exp_var_cum)):
    val = exp_var_cum[i]
    print(f"PCA{i+1}:", round(val, 3))
```

```
Explained Variance
PCA1: 0.372
PCA2: 0.347
PCA3: 0.176
PCA4: 0.057
```

In [27]:
```
pca.explained_variance_ratio_
```

Out[27]:
```
array([0.3719856 , 0.34700813, 0.17603793, 0.05705673, 0.02729754,
       0.0164632 , 0.00415086])
```

In [28]:
```
pca.explained_variance_ratio_.sum()
```

Out[28]:
```
1.0
```

In [29]:
```
# Retrieve the explained variance to determine how much information
# can be attributed to each principal component.

```

Answer the following question:

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

Answer: 0.895

Find the Best Value for k Using the PCA Data

In [31]:
```
# Create a list with the number of k-values from 1 to 11
X = df3.loc[:, ["PCA1", "PCA2", "PCA3", "PCA4"]]
X.head()
```

Out[31]:

	PCA1	PCA2	PCA3	PCA4
0	-0.600667	0.842760	0.461595	-0.109151
1	-0.458261	0.458466	0.952877	0.095100
2	-0.433070	-0.168126	-0.641752	-0.470282
3	-0.471835	-0.222660	-0.479053	-0.737473
4	-1.157800	2.041209	1.859715	0.236479

In [32]:

```python
# Create a a list to store inertia values
inertia = []
silhouettes = []
cha_chas = []

# Create a a list to store the values of k
k = list(range(2, 11))

# Create a for-loop where each value of k is evaluated using the K-mea
# Fit the model using the spread_df DataFrame
# Append the value of the computed inertia from the `inertia_` attribu
for i in k:
    # initialize the model
    k_model = KMeans(n_clusters=i, random_state=1)

    # fit the model
    k_model.fit(X)

    # predict the model
    preds = k_model.predict(X)

    # evaluate the model (generate the metics)
    inertia.append(k_model.inertia_)
    score = silhouette_score(X, preds)
    silhouettes.append(score)

    cha_cha = calinski_harabasz_score(X, preds)
    cha_chas.append(cha_cha)

    print(f"Finished {i} out of {max(k)}")
```

```
Finished 2 out of 10
Finished 3 out of 10
Finished 4 out of 10
Finished 5 out of 10
```

In [33]:

```python
# Define a DataFrame to hold the values for k and the corresponding ir
elbow_data = {"k": k, "inertia": inertia, "silhouette_score": silhouet
df_elbow2 = pd.DataFrame(elbow_data)

df_elbow["acc"] = df_elbow.inertia.diff()

# Review the DataFrame
df_elbow.head(10)
```

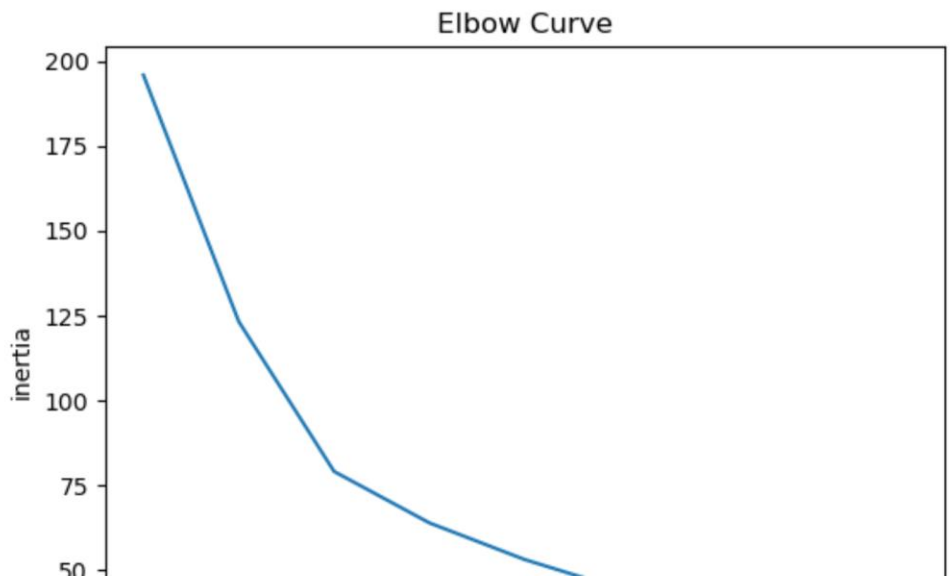
Out[33]:

	k	inertia	silhouette_score	cha_score	acc
0	2	195.820218	0.651576	18.159573	NaN
1	3	123.190482	0.702822	25.264783	-72.629736
2	4	79.022435	0.314482	32.459853	-44.168046
3	5	63.858668	0.329023	31.448698	-15.163768
4	6	53.057788	0.287883	30.864375	-10.800879
5	7	44.406791	0.290874	30.956861	-8.650998
6	8	37.078233	0.205692	31.776126	-7.328557
7	9	32.832187	0.258600	30.965687	-4.246046
8	10	28.165433	0.244422	31.653739	-4.666754

In [34]:

```
# Plot a line chart with all the inertia values computed with
# the different values of k to visually identify the optimal value for

# Plot the DataFrame
plt.plot(df_elbow2["k"], df_elbow["inertia"])
plt.title("Elbow Curve")
plt.xticks(df_elbow2["k"])
plt.ylabel("inertia")
plt.xlabel("k")
plt.show()
```



Answer the following questions:

- **Question:** What is the best value for `k` when using the PCA data?
    - **Answer:** 4
- **Question:** Does it differ from the best k value found using the original data?
    - **Answer:** No really, Just a bit clearer

Cluster Cryptocurrencies with K-means Using the PCA Data

In [35]:

```
# Initialize the K-Means model using the best value for k
model = KMeans(n_clusters=2, random_state=1)
```

In [36]:

```
# Fit the K-Means model using the PCA data
model.fit(X)
```

Out[36]:

```
KMeans
KMeans(n_clusters=2, random_state=1)
```

In [37]:

```
# Predict the clusters to group the cryptocurrencies using the PCA dat
k_lower = model.predict(X)
# Print the resulting array of cluster values.

```

In [38]:

```python
# Create a copy of the DataFrame with the PCA data
# Define the model with the higher value of k clusters
# Use a random_state of 1 to generate the model

# CHANGE THIS DEPENDING ON YOUR OPTIMAL k
model = KMeans(n_clusters=4, random_state=1)

# Fit the model
model.fit(X)

# Make predictions
preds = model.predict(X)

# Add a class column with the labels to the df DataFrame
df4 = df3.copy()
df4['clusters'] = preds

df4.head()

# Add a new column to the DataFrame with the predicted clusters


# Display sample data

```

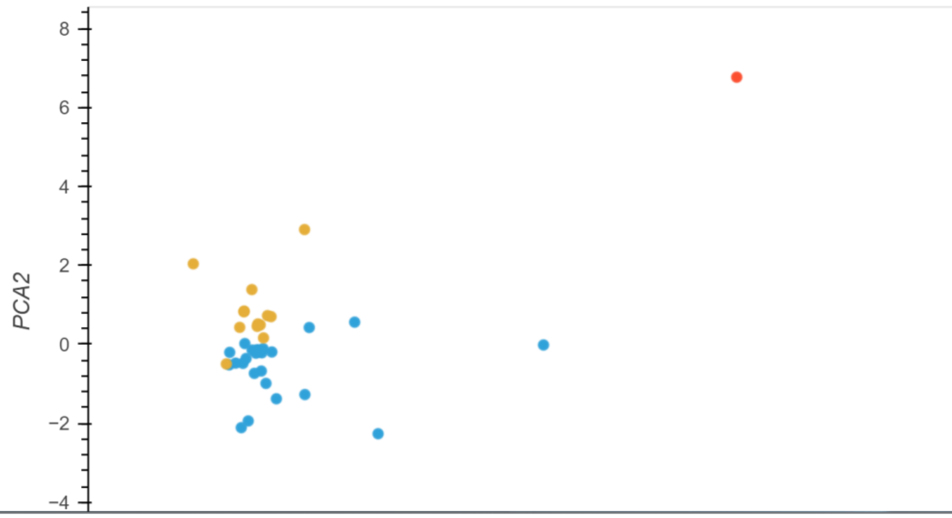
Out[38]:

	PCA1	PCA2	PCA3	PCA4	clusters
0	-0.600667	0.842760	0.461595	-0.109151	2
1	-0.458261	0.458466	0.952877	0.095100	2
2	-0.433070	-0.168126	-0.641752	-0.470282	0
3	-0.471835	-0.222660	-0.479053	-0.737473	0

In [39]:

```
# 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.
df4.hvplot.scatter(
    width=800,
    height=400,
    x="PCA1",
    y="PCA2",
    by="clusters",
    hover_cols="coin_id"
)
```

Out[39]:



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.

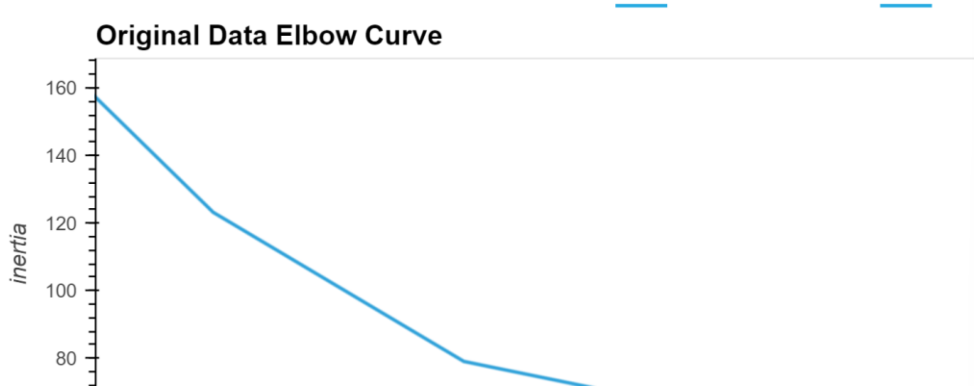
In [40]:

```
# Composite plot to contrast the Elbow curves
# Create the Elbow curve plots for the original and second Elbow curve
elbow_plot1 = df_elbow.hvplot.line(x='k', y='inertia', title='Original
elbow_plot2 = df_elbow2.hvplot.line(x='k', y='inertia', title='Second

# Create a composite plot to contrast the Elbow curves
composite_elbow_plot = elbow_plot1 + elbow_plot2

# Display the composite plot
composite_elbow_plot
```

Out[40]:



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:**DATA IS MORE CONSICE AND LESS DISPERSE