In [1]:
```
# Import required libraries and dependencies
import pandas as pd
import hvplot.pandas
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
# 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_percentage_14d	price_change_percentage_30d	price_change_percentage_60d	p
bitcoin	1.08388	7.60278	6.57509	7.67258	-3.25185	
ethereum	0.22392	10.38134	4.80849	0.13169	-12.88890	
tether	-0.21173	0.04935	0.00640	-0.04237	0.28037	
ripple	-0.37819	-0.60926	2.24984	0.23455	-17.55245	
bitcoin-cash	2.90585	17.09717	14.75334	15.74903	-13.71793	
binancecoin	2.10423	12.85511	6.80688	0.05865	36.33486	
chainlink	-0.23935	20.69459	9.30098	-11.21747	-43.69522	
cardano	0.00322	13.99302	5.55476	10.10553	-22.84776	
litecoin	-0.06341	6.60221	7.28931	1.21662	-17.23960	
bitcoin-cash-sv	0.92530	3.29641	-1.86656	2.88926	-24.87434	

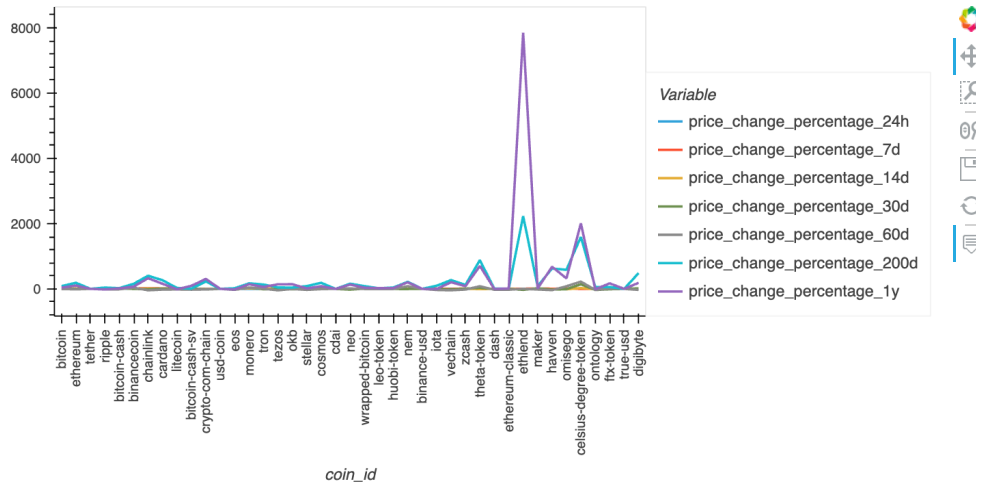
In [3]:
```
# Generate summary statistics
df_market_data.describe()
```

Out[3]:

	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d	price_change_percentage_30d	price_change_percentage_60d	price_c
count	41.000000	41.000000	41.000000	41.000000	41.000000	
mean	-0.269686	4.497147	0.185787	1.545693	-0.094119	
std	2.694793	6.375218	8.376939	26.344218	47.365803	
min	-13.527860	-6.094560	-18.158900	-34.705480	-44.822480	
25%	-0.608970	0.047260	-5.026620	-10.438470	-25.907990	
50%	-0.063410	3.296410	0.109740	-0.042370	-7.544550	
75%	0.612090	7.602780	5.510740	4.578130	0.657260	
max	4.840330	20.694590	24.239190	140.795700	223.064370	

In [4]:
```
# Plot your data to see what's in your DataFrame
df_market_data.hvplot.line(
    width=800,
    height=400,
    rot=90
)
```

Out[4]:



Prepare the Data

In [5]:
```
# Use the `StandardScaler()` module from scikit-learn to normalize the data from the CSV file
df_market_data_scaled = StandardScaler().fit_transform(df_market_data)
```

In [6]:
```
# Create a DataFrame with the scaled data
df_market_data_transformed = pd.DataFrame(df_market_data_scaled, columns=df_market_data.columns)

# Copy the crypto names from the original data (add a column called coin_id and copy the index fromt the original df)
df_market_data_transformed["coin_id"] = df_market_data.index

# Set the coinid column as index
df_market_data_transformed = df_market_data_transformed.set_index("coin_id")

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

Out[6]:

coin_id	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d	price_change_percentage_30d	price_change_percentage_60d	p
bitcoin	0.508529	0.493193	0.772200	0.235460	-0.067495	
ethereum	0.185446	0.934445	0.558692	-0.054341	-0.273483	
tether	0.021774	-0.706337	-0.021680	-0.061030	0.008005	
ripple	-0.040764	-0.810928	0.249458	-0.050388	-0.373164	
bitcoin-cash	1.193036	2.000959	1.760610	0.545842	-0.291203	
binancecoin	0.891871	1.327295	0.800214	-0.057148	0.778653	
chainlink	0.011397	2.572251	1.101647	-0.490495	-0.931954	
cardano	0.102530	1.508001	0.648885	0.328959	-0.486349	
litecoin	0.077497	0.334297	0.858520	-0.012646	-0.366477	
bitcoin-cash-sv	0.448952	-0.190684	-0.248043	0.051634	-0.529666	

Find the Best Value for k Using the Original Data.

In [7]:
```
# Create a list with the number of k-values from 1 to 11
k = list(range(1,12))
k
```

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

In [8]:
```
# Create an empty list to store the inertia values
inertia = []

# 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 i in k:
    k_model = KMeans(n_clusters=i, random_state=1)
    k_model.fit(df_market_data_transformed)
    inertia.append(k_model.inertia_)

inertia
```

Out[8]:
```
[287.0,
 195.82021818036043,
 123.19048183836958,
 79.02243535120977,
 65.40592346140596,
 52.933558921015006,
 47.983124098110004,
 37.28818726271726,
 33.06168486647883,
 28.779752431429223,
 25.248499378382775]
```

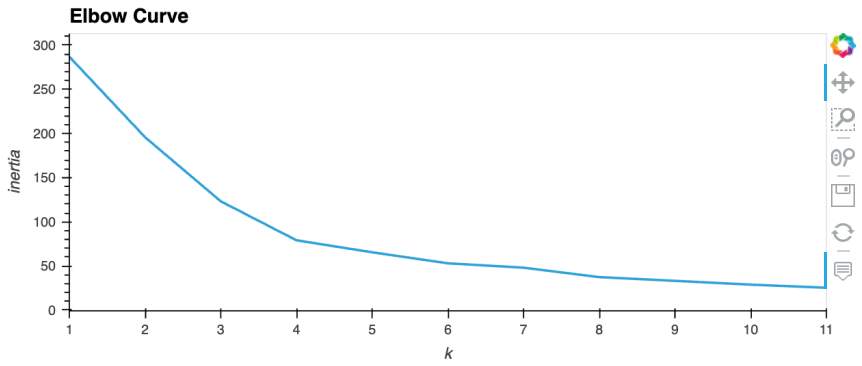
In [9]:
```
# Create a dictionary with the data to plot the Elbow curve
elbow_data = {"k": k, "inertia": inertia}

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

Out[9]:

	k	inertia
0	1	287.000000
1	2	195.820218
2	3	123.190482
3	4	79.022435
4	5	65.405923
5	6	52.933559
6	7	47.983124
7	8	37.288187
8	9	33.061685
9	10	28.779752
10	11	25.248499

In [10]:
```
# Plot a line chart with all the inertia values computed with
# the different values of k to visually identify the optimal value for k.
df_elbow.hvplot.line(
    x = "k",
    y = "inertia",
    title = "Elbow Curve",
    xticks = k)
```

Out[10]:



Answer the following question:

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

Answer: Four (4)

Cluster Cryptocurrencies with K-means Using the Original Data

In [11]:
```
# Initialize the K-Means model using the best value for k
model = KMeans(n_clusters=4, random_state=1)
```

In [12]:
```
# Fit the K-Means model using the scaled data
model.fit(df_market_data_transformed)
```

Out[12]:
```
KMeans(n_clusters=4, random_state=1)
```

In [13]:
```
# Predict the clusters to group the cryptocurrencies using the scaled data
k_4 = model.predict(df_market_data_transformed)

# Print the resulting array of cluster values.
k_4
```

Out[13]:
```
array([3, 3, 1, 1, 3, 3, 3, 3, 3, 1, 1, 1, 1, 3, 1, 3, 1, 1, 3, 1, 1, 3,
       1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 0, 3, 1, 1, 2, 1, 1, 1, 1],
      dtype=int32)
```

In [14]:
```
# Create a copy of the DataFrame
df_market_data_predictions = df_market_data_transformed.copy()
```

In [15]:
```
# Add a new column to the DataFrame with the predicted clusters
df_market_data_predictions["cluster_segment"] = k_4

# Display sample data
df_market_data_predictions
```

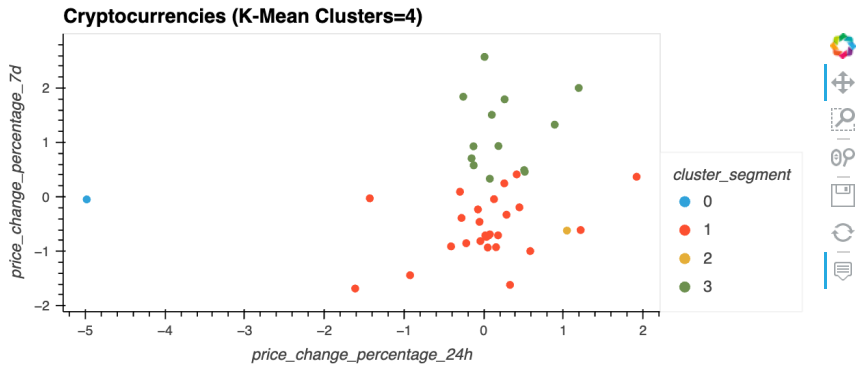
Out[15]:

coin_id	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d	price_change_percentage_30d	price_change_percentage_60d
bitcoin	0.508529	0.493193	0.772200	0.235460	-0.067495
ethereum	0.185446	0.934445	0.558692	-0.054341	-0.273483
tether	0.021774	-0.706337	-0.021680	-0.061030	0.008005
ripple	-0.040764	-0.810928	0.249458	-0.050388	-0.373164
bitcoin-cash	1.193036	2.000959	1.760610	0.545842	-0.291203
binancecoin	0.891871	1.327295	0.800214	-0.057148	0.778653
chainlink	0.011397	2.572251	1.101647	-0.490495	-0.931954
cardano	0.102530	1.508001	0.648885	0.328959	-0.486349
litecoin	0.077497	0.334297	0.858520	-0.012646	-0.366477
bitcoin-cash-sv	0.448952	-0.190684	-0.248043	0.051634	-0.529666
crypto-com-chain	0.331280	-1.614844	-1.054521	-0.729931	-0.350155
usd-coin	0.034352	-0.733026	-0.023140	-0.065775	0.002925
eos	0.155710	-0.922491	0.115024	-0.237488	-0.642837
monero	0.262723	1.792602	2.202665	1.437842	0.893865
tron	0.130050	-0.041018	0.147155	-0.543776	0.120116
tezos	-0.151583	0.708196	0.258012	-0.602296	-0.956049
okb	-0.923203	-1.437359	-0.629963	-0.460558	-0.058504
stellar	-0.277543	-0.385209	-0.153243	-0.371816	-0.656403
cosmos	-0.255978	1.840274	0.643565	0.116538	-0.151913
cdai	0.180851	-0.704931	-0.001816	-0.143237	0.016060
neo	0.286546	-0.326301	-1.212670	-0.903134	0.290970
wrapped-bitcoin	0.515453	0.461843	0.769975	0.224045	-0.074674
leo-token	0.051758	-0.928381	-0.871918	0.058782	-0.159250
huobi-token	-0.052032	-0.457229	0.032522	-0.184489	-0.070809
nem	-0.217984	-0.849381	0.297632	-0.199820	1.773127
binance-usd	0.061339	-0.706669	-0.015321	-0.058694	0.004017
iota	0.259097	0.249508	-0.478953	-0.218997	-0.735815
vechain	0.585089	-0.994231	-2.217108	-0.603898	-0.930423
zcash	-0.127467	0.929119	0.677532	0.223834	-0.437068
theta-token	-1.612188	-1.682027	-0.816921	1.148607	1.712641
dash	-0.296940	0.094763	0.040040	-0.358830	-0.558527
ethereum-classic	-0.071312	-0.229484	-0.175544	0.051882	-0.551760
ethlend	-4.981042	-0.045178	-1.206956	-1.212126	0.047736
maker	-0.125168	0.580730	-0.202356	0.582911	-0.395923
havven	-1.428574	-0.025510	-1.628859	-0.860354	-0.840714
omisego	1.919812	0.370447	-1.619761	-0.409716	1.696480
celsius-degree-token	1.045530	-0.618328	2.907054	5.351455	4.769913
ontology	-0.409044	-0.906963	-1.298986	-1.393153	-0.696937
ftx-token	0.414711	0.414044	-0.047386	-0.465380	0.128185
true-usd	0.078038	-0.687745	-0.009191	-0.058214	0.007388
digibyte	1.217453	-0.607714	-0.907066	0.449939	-0.662530

In [16]:

```
# 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.

df_market_data_predictions.hvplot.scatter(
    x = "price_change_percentage_24h",
    y = "price_change_percentage_7d",
    title = "Cryptocurrencies (K-Mean Clusters=4)",
    by = "cluster_segment",
    hover_cols = 'coin_id'
)
```

Out[16]:



Optimize Clusters with Principal Component Analysis.

In [17]:
```
# Create a PCA model instance and set `n_components=3`.
pca = PCA(n_components=3)
```

In [18]:
```
# Use the PCA model with `fit_transform` to reduce to
# three principal components.
market_data_pca = pca.fit_transform(df_market_data_transformed)

# View the first five rows of the DataFrame.
market_data_pca[:5]
```

Out[18]:
```
array([[-0.60066733,  0.84276006,  0.46159457],
       [-0.45826071,  0.45846566,  0.95287678],
       [-0.43306981, -0.16812638, -0.64175193],
       [-0.47183495, -0.22266008, -0.47905316],
       [-1.15779997,  2.04120919,  1.85971527]])
```

In [19]:
```
# Retrieve the explained variance to determine how much information
# can be attributed to each principal component.
pca.explained_variance_ratio_
```

Out[19]:
```
array([0.3719856 , 0.34700813, 0.17603793])
```

Answer the following question:

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

Answer: The highest fraction of explained variance among these variables is 37%, and the lowest one is 18%. We can also compute these fractions for subsets of variables. For instance, variables 1 and 2 together explain 72% of the total variance, and variables 1 and 3 explain 55%. Combined, all three components explain 90% of the total variance.

The concept of Explained variance is useful in assessing how important each component is. In general, the larger the variance explained by a principal component, the more important that component is. PCA is a technique used to reduce the dimensionality of data. It does this by finding the directions of maximum variance in the data and projecting the data onto those directions. The amount of variance explained by each direction is called the "explained variance." Explained variance can be used to choose the number of dimensions to keep in a reduced dataset. It can also be used to assess the quality of a machine learning model. In general, a model with high explained variance will have good predictive power, while a model with low explained variance may not be as accurate.

When the overall sum of the two variance ratios is extremely low and the plot doesn't give us any insight, using a 3rd Principal Component is highly beneficial.

In [20]:
```
# Create a new DataFrame with the PCA data.
df_market_data_pca = pd.DataFrame(
    market_data_pca,
    columns=["PCA1", "PCA2", "PCA3"]
)

# Copy the crypto names from the original data
df_market_data_pca['coin_id'] = df_market_data.index

# Set the coinid column as index
df_market_data_pca = df_market_data_pca.set_index('coin_id')

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

Out[20]:

coin_id	PCA1	PCA2	PCA3
bitcoin	-0.600667	0.842760	0.461595
ethereum	-0.458261	0.458466	0.952877
tether	-0.433070	-0.168126	-0.641752
ripple	-0.471835	-0.222660	-0.479053
bitcoin-cash	-1.157800	2.041209	1.859715
binancecoin	-0.516534	1.388377	0.804071
chainlink	-0.450711	0.517699	2.846143
cardano	-0.345600	0.729439	1.478013
litecoin	-0.649468	0.432165	0.600303
bitcoin-cash-sv	-0.759014	-0.201200	-0.217653

Find the Best Value for k Using the PCA Data

In [21]:
```
# Create a list with the number of k-values from 1 to 11
k_pca = list(range(1,12))
k_pca
```

Out[21]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

In [22]:
```
# Create an empy list to store the inertia values
inertia_pca = []

# 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 i in k_pca:
    k_model_pca = KMeans(n_clusters=i, random_state=1)
    k_model_pca.fit(df_market_data_pca)
    inertia_pca.append(k_model_pca.inertia_)

inertia_pca
```

Out[22]:
```
[256.8740855678925,
 165.90199402036015,
 93.77462568057307,
 49.6654966517974,
 38.35225121638686,
 27.618971787957456,
 21.134056037473627,
 17.437664237020417,
 13.742791960480574,
 10.484890485976932,
 8.114985790821049]
```

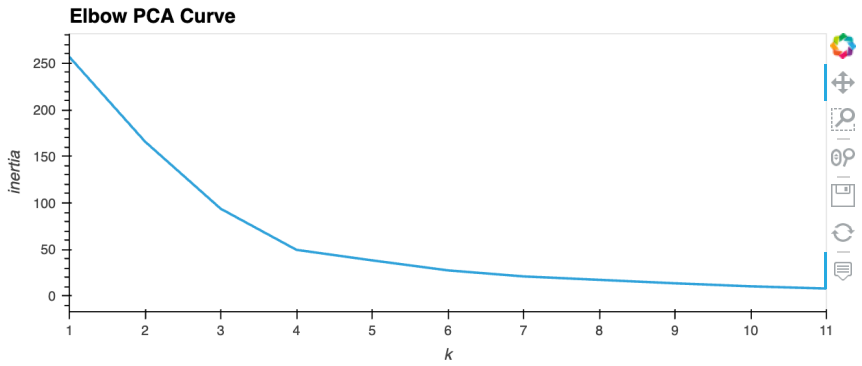
In [23]:
```
# Create a dictionary with the data to plot the Elbow curve
elbow_data_pca = {"k": k_pca, "inertia": inertia_pca}

# Create a DataFrame with the data to plot the Elbow curve
df_elbow_pca = pd.DataFrame(elbow_data_pca)
df_elbow_pca.head(10)
```

Out[23]:

	k	inertia
0	1	256.874086
1	2	165.901994
2	3	93.774626
3	4	49.665497
4	5	38.352251
5	6	27.618972
6	7	21.134056
7	8	17.437664
8	9	13.742792
9	10	10.484890

In [24]:
```
# Plot a line chart with all the inertia values computed with
# the different values of k to visually identify the optimal value for k.
df_elbow_pca.hvplot.line(
    x = "k",
    y = "inertia",
    title = "Elbow PCA Curve",
    xticks = k)
```

Out[24]:



Answer the following questions:

- **Question:** What is the best value for `k` when using the PCA data?
- **Question:** Does it differ from the best k value found using the original data?

Answer: Based on the Elbow curve, the best value for the number of clusters k is 4. Ultimately, this does not differ from the original data, however, the Inertia itself varies at that point from the original data. With the original data, the inertia is 79 with k=4, and only 50 using 3 PCA. This is significant because it shows that the strength of 'k' has improved with 3 PCA. Even at k=6 inertia was 53% inertia with the original data.

By choosing k=4 with 3 PCA, we get a significant inertia reduction from 94 in k=3 to 50 in k=4. After that, reductions do not improve in more than 11 inertia units. Graphically, the elbow on k=4 is more pronounced, with subsequent points tapering off.

Cluster Cryptocurrencies with K-means Using the PCA Data

In [25]:
```
# Initialize the K-Means model using the best value for k
model_pca = KMeans(n_clusters=4, random_state=1)
```

In [26]:
```
# Fit the K-Means model using the PCA data
model_pca.fit(df_market_data_pca)
```

Out[26]:
```
KMeans(n_clusters=4, random_state=1)
```

In [27]:
```
# Predict the clusters to group the cryptocurrencies using the PCA data
k_4_pca = model_pca.predict(df_market_data_pca)

# Print the resulting array of cluster values.
k_4_pca
```

Out[27]:
```
array([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],
      dtype=int32)
```

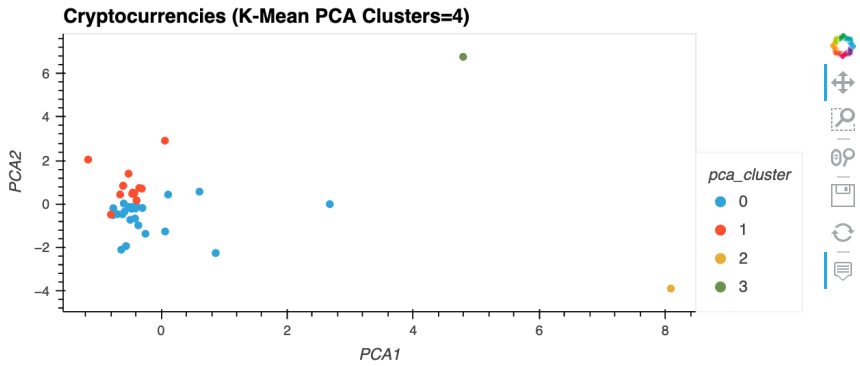
In [28]:
```
# Create a copy of the DataFrame with the PCA data
df_market_data_predictions_pca = df_market_data_pca.copy()

# Add a new column to the DataFrame with the predicted clusters
df_market_data_predictions_pca["pca_cluster"] = k_4_pca

# Display sample data
df_market_data_predictions_pca
```

Out[28]:

coin_id	PCA1	PCA2	PCA3	pca_cluster
bitcoin	-0.600667	0.842760	0.461595	1
ethereum	-0.458261	0.458466	0.952877	1
tether	-0.433070	-0.168126	-0.641752	0
ripple	-0.471835	-0.222660	-0.479053	0
bitcoin-cash	-1.157800	2.041209	1.859715	1
binancecoin	-0.516534	1.388377	0.804071	1
chainlink	-0.450711	0.517699	2.846143	1
cardano	-0.345600	0.729439	1.478013	1
litecoin	-0.649468	0.432165	0.600303	1
bitcoin-cash-sv	-0.759014	-0.201200	-0.217653	0
crypto-com-chain	-0.248198	-1.376252	-1.462026	0
usd-coin	-0.438408	-0.175337	-0.663388	0
eos	-0.693425	-0.473815	-0.527597	0
monero	0.060499	2.909404	1.498571	1
tron	-0.393352	-0.108192	-0.012756	0
tezos	-0.796176	-0.494409	1.082812	1
okb	0.064075	-1.269825	-1.098829	0
stellar	-0.489015	-0.732719	-0.062543	0
cosmos	-0.306272	0.703415	1.714224	1
cdai	-0.513528	-0.142802	-0.656566	0
neo	-0.362120	-0.986914	-0.728752	0
wrapped-bitcoin	-0.604265	0.827398	0.439316	1
leo-token	-0.413296	-0.674115	-1.076628	0
huobi-token	-0.407483	-0.212507	-0.351426	0
nem	0.608974	0.563532	-1.148742	0
binance-usd	-0.450211	-0.151019	-0.647401	0
iota	-0.764665	-0.517886	0.204990	0
vechain	-0.556315	-1.938209	-1.261776	0
zcash	-0.425147	0.492976	1.058048	1
theta-token	2.676868	-0.013954	-1.965207	0
dash	-0.613923	-0.479337	0.339565	0
ethereum-classic	-0.579924	-0.356334	-0.114942	0
ethlend	8.089018	-3.896891	2.301382	2
maker	-0.389045	0.165041	0.379414	1
havven	0.865762	-2.261882	0.275583	0
omisego	0.111675	0.428316	-1.205398	0
celsius-degree-token	4.792395	6.767679	-1.986985	3
ontology	-0.632355	-2.108117	-0.652227	0
ftx-token	-0.593142	0.021485	0.209911	0
true-usd	-0.458131	-0.135734	-0.635284	0
digibyte	-0.297910	-0.191126	-0.909602	0

In [29]:
```
# 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.

df_market_data_predictions_pca.hvplot.scatter(
    x = "PCA1",
    y = "PCA2",
    title = "Cryptocurrencies (K-Mean PCA Clusters=4)",
    by = "pca_cluster",
    hover_cols = 'coin_id'
)
```

Out[29]:

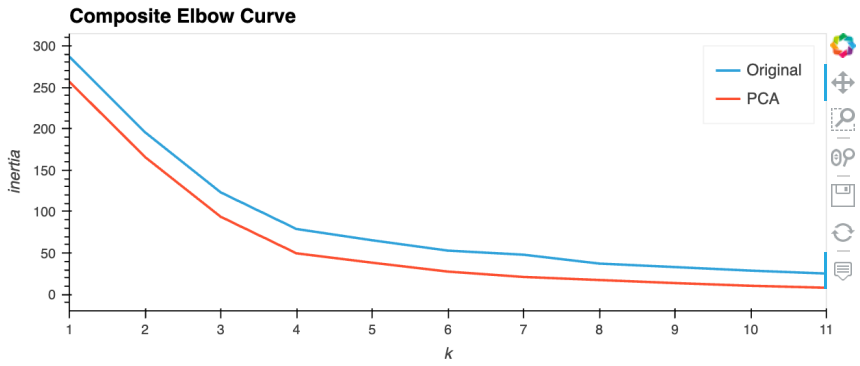


Visualize and Compare the Results

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

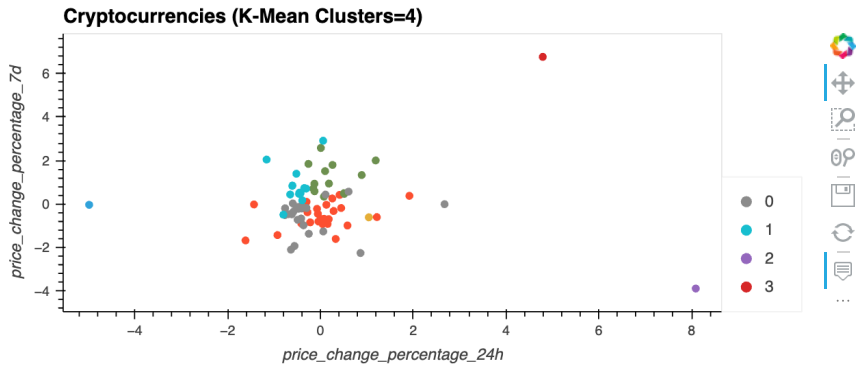
In [30]:
```
# Composite plot to contrast the Elbow curves
# https://hvplot.holoviz.org/user_guide/Plotting.html

df_elbow.hvplot.line(
    x = "k",
    y = "inertia",
    label="Original",
    title="Composite Elbow Curve",
    xticks=k
) * df_elbow_pca.hvplot.line(
    x = "k",
    y = "inertia",
    label="PCA"
)
```

Out[30]:



In [31]:
```
# Composite plot to contrast the clusters
df_market_data_predictions.hvplot.scatter(
    x = "price_change_percentage_24h",
    y = "price_change_percentage_7d",
    title = "Cryptocurrencies (K-Mean Clusters=4)",
    by = "cluster_segment",
    hover_cols = 'coin_id',
) * df_market_data_predictions_pca.hvplot.scatter(
    x = "PCA1",
    y = "PCA2",
    title = "Cryptocurrencies (K-Mean PCA Clusters=4)",
    by = "pca_cluster",
    hover_cols = 'coin_id',
)
```

Out[31]:



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:** Impacts of using fewer features to cluster the data using K-Means First, at the level of inertia Using less amount of features reduces the amount of inertia. This happens because the reduction of dimensionality implies a reduction in the variance of the clustered data. In the analyzed case, we reduced the variance in 10% by using three components, which implied a reduction in the inertia in a similar amount (11% reduction from 287 to 256). This kind of reduction could have involved a reduction in optimal number of clusters. However in this case, it didn't. PCA achieves a higher reduction rate of the initial inertia. That means, the components are more efficient in setting up the data for clustering. In the case of PCA, the reduction was from 256 to 50 units of inertia, which is a reduction in 80% of the initial value; whereas the reduction with standarized data was from 287 to 79 units, wich is only a 72% reduction rate. Second, in terms of the clusters The original clusters and the pca clusters exactly match. That supports the use of dimentionality reduction. We dramatically reduced dimentions from seven to three, and still we got the same results. An impact of using fewer features is the benefit in reducing the resources needed to manage large amounts of data, without compromising optimal results.

Another notizable benefit in this particular case, is about the interpretation of the principal components. It is not easy at the beggining, but after making sense of the representation involved in the components, it allows us to visualize the data, and understand the clusters with less graphs. For example, in this case, we have realized that:

a. principal component one reflects mainly returns of longer terms, such as 1Y and 200 days returns;

b. principal component two reflects mostly middle term returns (60 days and 30 days returns); and

c. principal component three describes mostly short term returns such as 7 and 14 days.

Then, we can identify the cluster representation (here by returns I mean standarized returns): red represent cryptocurrencies with the worse performance of the set. Negative or small return in all periods (short, middle and long term). For example, vechain, ontology. orange performs a little better than the red. Those are cryptos with moderate positive or slighly negative in the middle and short term returns (ie. bitcoin, chainlink, bitcoin-cash) blue performs really well in the long term, but may have large drops in shorter terms (ie. Ethlend) green performs well in the middle term side, without the large drops that a blue crypto could have. (ie. celcius-degree-token) Disclaimer: I referred to 1 year as long term because it is the longest in the data. usually 1 year is short term. In this case, I have use short term for 2 weeks or less, middle term for 30-60 days, and long term for 200-365 days.

In [ ]: