Clustering Crypto

In [10]:
```
# Initial imports
import requests
import json
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
import hvplot.pandas
import plotly.express as px
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from urllib.request import Request, urlopen
from pandas import json_normalize
```

In [2]:
```
# !pip install plotly
```

In [3]:
```
# !pip install hvplot
```

In [4]:
```
# !pip install fsspec
```

Fetching Cryptocurrency Data

In [5]:
```
# Use the following endpoint to fetch json data
url = "https://min-api.cryptocompare.com/data/all/coinlist"
```

In [9]:
```
# Create a DataFrame
json_request = Request(url)
json_response = urlopen(json_request)
data = json_response.read()
```

In [11]:
```
json_data = json.loads(data)
```

In [13]:
```
# data must be transposed
df = pd.DataFrame(json_data['Data']).T
df.head()
```

Out[13]:

	Id	Url	ImageUrl	ContentCreatedOn	Name	Symbol
42	4321	/coins/42/overview	/media/35650717/42.jpg	1427211129	42	42
300	749869	/coins/300/overview	/media/27010595/300.png	1517935016	300	300
365	33639	/coins/365/overview	/media/352070/365.png	1480032918	365	365
404	21227	/coins/404/overview	/media/35650851/404-300x300.jpg	1466100361	404	404
433	926547	/coins/433/overview	/media/34836095/433.png	1541597321	433	433

5 rows × 36 columns

In [14]:

```
# Alternatively, use the provided csv file:
file_path = Path("Resources/crypto_data.csv")

df_cd = pd.read_csv(file_path)
df_cd.head()
```

Out[14]:

	Unnamed: 0	CoinName	Algorithm	IsTrading	ProofType	TotalCoinsMined	TotalCoinSupply
0	42	42 Coin	Scrypt	True	PoW/PoS	4.199995e+01	42
1	365	365Coin	X11	True	PoW/PoS	NaN	2300000000
2	404	404Coin	Scrypt	True	PoW/PoS	1.055185e+09	532000000
3	611	SixEleven	SHA-256	True	PoW	NaN	611000
4	808	808	SHA-256	True	PoW/PoS	0.000000e+00	0

In [ ]:

Data Preprocessing

In [15]:
```
df.columns
```

Out[15]:
```
Index(['Id', 'Url', 'ImageUrl', 'ContentCreatedOn', 'Name', 'Symbol',
       'CoinName', 'FullName', 'Description', 'AssetTokenStatus', 'Algorithm',
       'ProofType', 'SortOrder', 'Sponsored', 'Taxonomy', 'Rating',
       'IsTrading', 'TotalCoinsMined', 'CirculatingSupply', 'BlockNumber',
       'NetHashesPerSecond', 'BlockReward', 'BlockTime', 'AssetLaunchDate',
       'AssetWhitepaperUrl', 'AssetWebsiteUrl', 'MaxSupply', 'MktCapPenalty',
       'IsUsedInDefi', 'IsUsedInNft', 'PlatformType', 'BuiltOn',
       'SmartContractAddress', 'DecimalPoints', 'Difficulty', 'AlgorithmType']
,
      dtype='object')
```

In [16]:
```
df.shape
```

Out[16]:
```
(7338, 36)
```

In [17]:
```
# Keep only necessary columns:
# 'CoinName','Algorithm','IsTrading','ProofType','TotalCoinsMined','TotalCoin
df_nec = df[['CoinName','Algorithm','IsTrading','ProofType','TotalCoinsMined'
```

In [18]:
```
df_nec.shape
```

Out[18]:
```
(7338, 6)
```

In [19]:
```
# Keep only cryptocurrencies that are trading
df_nec = df_nec.loc[df_nec['IsTrading'] == True]
```

In [20]:
```
# Keep only cryptocurrencies with a working algorithm
df_nec = df_nec.dropna(subset=['Algorithm'], how='all')
```

In [21]:
```
df_nec.sample()
```

Out[21]:

	CoinName	Algorithm	IsTrading	ProofType	TotalCoinsMined	MaxSupply
USE	Usechain Token	N/A	True	N/A	NaN	NaN

In [22]:
```
df_nec.shape
```

Out[22]:
```
(5874, 6)
```

In [23]:
```
# Remove the "IsTrading" column
df_nec = df_nec.drop(columns='IsTrading')
```

In [24]:
```
df_nec
```

Out[24]:

	CoinName	Algorithm	ProofType	TotalCoinsMined	MaxSupply
42	42 Coin	Scrypt	PoW/PoS	0	0
300	300 token	N/A	N/A	300	300
365	365Coin	X11	PoW/PoS	0	0
404	404Coin	Scrypt	PoW/PoS	0	0
611	SixEleven	SHA-256	PoW	0	0
...	...	...	...	...	...
AQUAC	Aquachain	Argon2	PoW	NaN	NaN
AQUA	Planet Finance	N/A	N/A	97701.434187	-1
NEETCOIN	Neetcoin	Scrypt	PoW/PoS	NaN	NaN
CHESSCOIN	ChessCoin	Scrypt	PoW/PoS	NaN	NaN
CHESS	Tranchess	N/A	N/A	300000000	-1

5874 rows × 5 columns

In [25]:
```
# Remove rows with at least 1 null value
df_nec = df_nec.dropna(how='any')
```

In [26]:
```
df_nec.shape
```

Out[26]:
```
(2626, 5)
```

In [27]:
```
# Remove rows with cryptocurrencies having no coins mined
df_nec = df_nec[df_nec.TotalCoinsMined != 0]
```

In [28]:
```
df_nec.shape
```

Out[28]:
```
(1999, 5)
```

In [29]:
```
# Drop rows where there are 'N/A' text values
df_nec = df_nec[(df_nec.CoinName != 'N/A') & (df_nec.Algorithm != 'N/A') & (d
```

In [30]:
```
df_nec.shape
```

Out[30]:
```
(129, 5)
```

In [31]:
```
df_nec
```

Out[31]:

	CoinName	Algorithm	ProofType	TotalCoinsMined	MaxSupply
NSR	NuShares	PoS	PoS	6168460313.8311	0
TRI	Triangles Coin	X13	PoW/PoS	190079.31648	0
CMTC	CometCoin	Scrypt	PoW	872830	0
CHAT	OpenChat	Scrypt	PoW/PoS	1000000000	-1
QRL	Quantum Resistant Ledger	RandomX	PoW	75320310.142578	105000000
...	...	...	...	...	...
MEC	MegaCoin	Scrypt	PoW	39738139.0556	42000000
ZEC	ZCash	Equihash	PoW	11803347.2008	21000000
OXYC	Oxycoin	DPoS	DPoS	1122382283.37	-1
PIVX	Private Instant Verified Transaction	Quark	PoW/PoS	67563118.059209	-1
POA	Poa Network	Proof-of-Authority	PoA	293804488.623202	-1

129 rows × 5 columns

In [33]:
```
# Store the 'CoinName'column in its own DataFrame prior to dropping it from c
df_coinname = df_nec.CoinName
```

In [36]:
```
type(df_coinname)
```

Out[36]:
```
pandas.core.series.Series
```

In [37]:
```
# Drop the 'CoinName' column since it's not going to be used on the clusterin
df_nec = df_nec.drop(columns='CoinName')
```

In [38]:
```
# Create dummy variables for text features
dummy_df = pd.get_dummies(df_nec, columns=['Algorithm', 'ProofType'])
```

In [39]:
```
dummy_df.shape
```

Out[39]:
```
(129, 79)
```

In [40]:
```
dummy_df.head()
```

Out[40]:

	TotalCoinsMined	MaxSupply	Algorithm_Autolykos	Algorithm_BEP-2	Algorithm_BEP-20 Token	A
NSR	6168460313.8311	0	0	0	0	
TRI	190079.31648	0	0	0	0	
CMTC	872830	0	0	0	0	
CHAT	1000000000	-1	0	0	0	
QRL	75320310.142578	105000000	0	0	0	

5 rows × 79 columns

In [41]:
```
# Standardize data
X = StandardScaler().fit_transform(dummy_df)
X[:5]
```

Out[41]:
```
array([[-0.09722672, -0.09465805, -0.08838835, -0.08838835, -0.08838835,
        -0.12549116, -0.08838835, -0.08838835, -0.12549116, -0.12549116,
        -0.15430335, -0.08838835, -0.08838835, -0.23953507, -0.12549116,
        -0.08838835, -0.08838835, -0.08838835, -0.30532006, -0.08838835,
        -0.08838835, -0.23953507, -0.08838835, -0.08838835, -0.12549116,
        -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
        -0.15430335, -0.08838835, -0.08838835, -0.12549116,  4.97995984,
        -0.08838835, -0.08838835, -0.15430335, -0.12549116, -0.30532006,
        -0.12549116, -0.08838835, -0.08838835, -0.08838835, -0.44095855,
        -0.08838835, -0.08838835, -0.08838835, -0.17888544, -0.08838835,
        -0.20080483, -0.12549116, -0.08838835, -0.08838835, -0.08838835,
        -0.08838835, -0.08838835, -0.25712974, -0.08838835, -0.08838835,
        -0.12549116, -0.08838835, -0.08838835,  3.27525155, -0.08838835,
        -0.08838835, -0.08838835, -0.96196317, -0.49029034, -0.08838835,
        -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
        -0.08838835, -0.08838835, -0.08838835, -0.08838835],
       [-0.10094212, -0.09465805, -0.08838835, -0.08838835, -0.08838835,
        -0.12549116, -0.08838835, -0.08838835, -0.12549116, -0.12549116,
        -0.15430335, -0.08838835, -0.08838835, -0.23953507, -0.12549116,
```

```
 -0.08838835, -0.08838835, -0.08838835, -0.30532006, -0.08838835,
 -0.08838835, -0.23953507, -0.08838835, -0.08838835, -0.12549116,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
 -0.15430335, -0.08838835, -0.08838835, -0.12549116, -0.20080483,
 -0.08838835, -0.08838835, -0.15430335, -0.12549116, -0.30532006,
 -0.12549116, -0.08838835, -0.08838835, -0.08838835, -0.44095855,
 -0.08838835, -0.08838835, -0.08838835, -0.17888544, -0.08838835,
  4.97995984, -0.12549116, -0.08838835, -0.08838835, -0.08838835,
 -0.08838835, -0.08838835, -0.25712974, -0.08838835, -0.08838835,
 -0.12549116, -0.08838835, -0.08838835, -0.30532006, -0.08838835,
 -0.08838835, -0.08838835, -0.96196317,  2.03960781, -0.08838835,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835],
[-0.10094171, -0.09465805, -0.08838835, -0.08838835, -0.08838835,
 -0.12549116, -0.08838835, -0.08838835, -0.12549116, -0.12549116,
 -0.15430335, -0.08838835, -0.08838835, -0.23953507, -0.12549116,
 -0.08838835, -0.08838835, -0.08838835, -0.30532006, -0.08838835,
 -0.08838835, -0.23953507, -0.08838835, -0.08838835, -0.12549116,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
 -0.15430335, -0.08838835, -0.08838835, -0.12549116, -0.20080483,
 -0.08838835, -0.08838835, -0.15430335, -0.12549116, -0.30532006,
 -0.12549116, -0.08838835, -0.08838835, -0.08838835,  2.26778684,
 -0.08838835, -0.08838835, -0.08838835, -0.17888544, -0.08838835,
 -0.20080483, -0.12549116, -0.08838835, -0.08838835, -0.08838835,
 -0.08838835, -0.08838835, -0.25712974, -0.08838835, -0.08838835,
 -0.12549116, -0.08838835, -0.08838835, -0.30532006, -0.08838835,
 -0.08838835, -0.08838835,  1.03954084, -0.49029034, -0.08838835,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835],
[-0.10033989, -0.09465805, -0.08838835, -0.08838835, -0.08838835,
 -0.12549116, -0.08838835, -0.08838835, -0.12549116, -0.12549116,
 -0.15430335, -0.08838835, -0.08838835, -0.23953507, -0.12549116,
 -0.08838835, -0.08838835, -0.08838835, -0.30532006, -0.08838835,
 -0.08838835, -0.23953507, -0.08838835, -0.08838835, -0.12549116,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
 -0.15430335, -0.08838835, -0.08838835, -0.12549116, -0.20080483,
 -0.08838835, -0.08838835, -0.15430335, -0.12549116, -0.30532006,
 -0.12549116, -0.08838835, -0.08838835, -0.08838835,  2.26778684,
 -0.08838835, -0.08838835, -0.08838835, -0.17888544, -0.08838835,
 -0.20080483, -0.12549116, -0.08838835, -0.08838835, -0.08838835,
 -0.08838835, -0.08838835, -0.25712974, -0.08838835, -0.08838835,
 -0.12549116, -0.08838835, -0.08838835, -0.30532006, -0.08838835,
 -0.08838835, -0.08838835, -0.96196317,  2.03960781, -0.08838835,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835],
[-0.10089687, -0.09460107, -0.08838835, -0.08838835, -0.08838835,
 -0.12549116, -0.08838835, -0.08838835, -0.12549116, -0.12549116,
 -0.15430335, -0.08838835, -0.08838835, -0.23953507, -0.12549116,
 -0.08838835, -0.08838835, -0.08838835, -0.30532006, -0.08838835,
 -0.08838835, -0.23953507, -0.08838835, -0.08838835, -0.12549116,
 -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
 -0.15430335, -0.08838835, -0.08838835, -0.12549116, -0.20080483,
 -0.08838835, -0.08838835, -0.15430335,  7.96868873, -0.30532006,
```

```
       -0.12549116, -0.08838835, -0.08838835, -0.08838835, -0.44095855,
       -0.08838835, -0.08838835, -0.08838835, -0.17888544, -0.08838835,
       -0.20080483, -0.12549116, -0.08838835, -0.08838835, -0.08838835,
       -0.08838835, -0.08838835, -0.25712974, -0.08838835, -0.08838835,
       -0.12549116, -0.08838835, -0.08838835, -0.30532006, -0.08838835,
       -0.08838835, -0.08838835,  1.03954084, -0.49029034, -0.08838835,
       -0.08838835, -0.08838835, -0.08838835, -0.08838835, -0.08838835,
       -0.08838835, -0.08838835, -0.08838835, -0.08838835]])
```

Reducing Dimensions Using PCA

In [42]:

```
# Use PCA to reduce dimensions to 3 principal components
pca = PCA(n_components=3)

# Get two principal components for the data.
X_pca = pca.fit_transform(X)
```

In [48]:

```
# Create a DataFrame with the principal components data
df_pca = pd.DataFrame(
    data=X_pca, columns=['prin component 1', 'prin comp 2', 'prin comp 3'], i
)
df_pca.head()
```

Out[48]:

	prin component 1	prin comp 2	prin comp 3
NSR	-1.335591	0.685818	-0.697175
TRI	-1.526655	-0.878860	-0.469809
CMTC	0.706005	-0.797232	-0.255953
CHAT	-0.929454	-1.021911	-0.405687
QRL	1.276871	-0.609129	-0.123259

Clustering Crytocurrencies Using K-Means

Find the Best Value for k Using the Elbow Curve

In [44]:

```
inertia = []
k = list(range(1, 11))

# Calculate the inertia for the range of k values
for i in k:
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(df_pca)
    inertia.append(km.inertia_)

# Create the Elbow Curve using hvPlot
elbow_data = {"k": k, "inertia": inertia}
df_elbow = pd.DataFrame(elbow_data)
df_elbow.hvplot.line(x="k", y="inertia", xticks=k, title="Elbow Curve")
```

Out[44]:

Running K-Means with `k=<your best value for k here>`

In [51]:

```
# Initialize the K-Means model

model = KMeans(n_clusters=4, random_state=0)

# Fit the model
model.fit(df_pca)

# Predict clusters
predictions = model.predict(df_pca)

# Create return DataFrame with predicted clusters
df_clust = pd.concat([df_nec, df_pca], axis=1, sort=False)
df_clust["CoinName"] = df_coinname
df_clust["class"] = model.labels_
df_clust.head()
```

Out[51]:

	Algorithm	ProofType	TotalCoinsMined	MaxSupply	prin component 1	prin comp 2	prin comp 3
NSR	PoS	PoS	6168460313.8311	0	-1.335591	0.685818	-0.697175
TRI	X13	PoW/PoS	190079.31648	0	-1.526655	-0.878860	-0.469809
CMTC	Scrypt	PoW	872830	0	0.706005	-0.797232	-0.255953
CHAT	Scrypt	PoW/PoS	1000000000	-1	-0.929454	-1.021911	-0.405687
QRL	RandomX	PoW	75320310.142578	105000000	1.276871	-0.609129	-0.123259

Visualizing Results

3D-Scatter with Clusters

In [56]:

```
# Create a 3D-Scatter with the PCA data and the clusters
fig = px.scatter_3d(
    df_clust,
    x = 'prin component 1',
    y = 'prin comp 2',
    z = 'prin comp 3',
    color = 'class',
    symbol = 'class',
    hover_name = 'CoinName',
    hover_data = ['Algorithm'],
    width=800,
)
fig.update_layout(legend=dict(x=0, y=1))
fig.show()
```

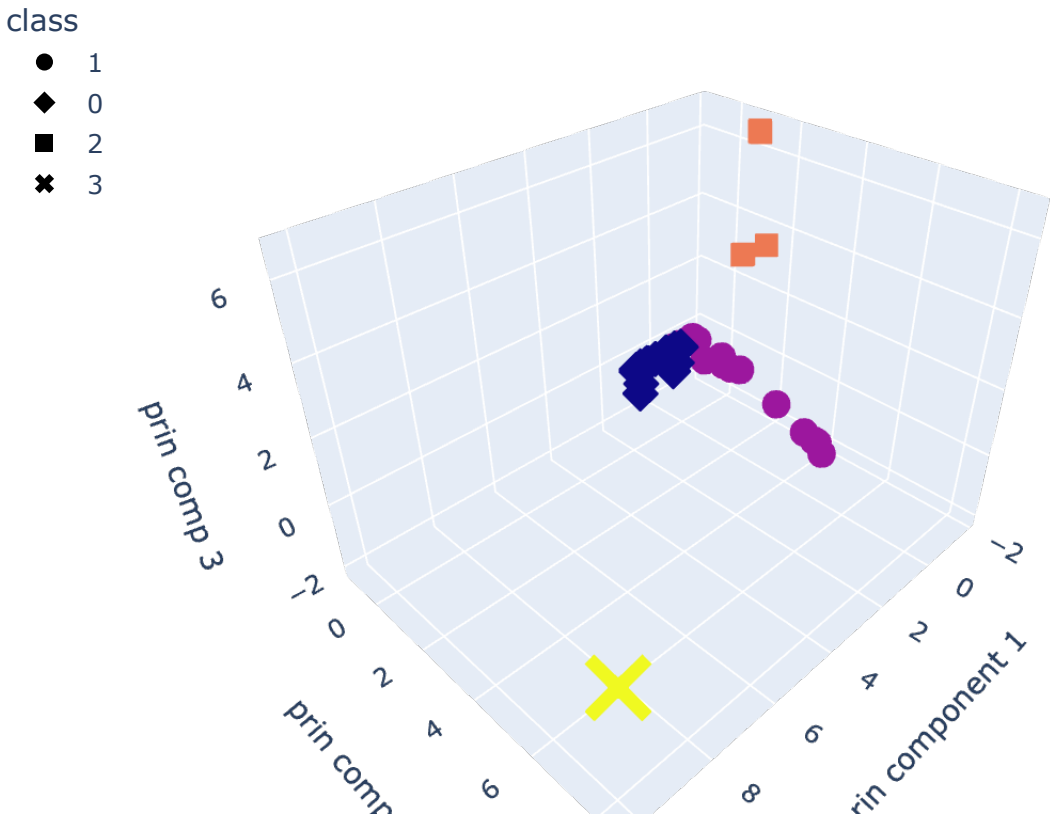

Table of Tradable Cryptocurrencies

In [57]:

```
# Table with tradable cryptos
df_clust[
    [
        "CoinName",
        "Algorithm",
        "ProofType",
        "MaxSupply",
        "TotalCoinsMined",
        "class",
    ]
].hvplot.table()
```

Out[57]:

In [58]:
```
# Print the total number of tradable cryptocurrencies
print(f'There are {df_clust.shape[0]} tradeable cryptocurrencies.')
```

```
There are 129 tradeable cryptocurrencies.
```

Scatter Plot with Tradable Cryptocurrencies

In [60]:
```
# Scale data to create the scatter plot
scaler = MinMaxScaler()

plot_data = scaler.fit_transform(
    df_clust[['MaxSupply', 'TotalCoinsMined']]
)
plot_df = pd.DataFrame(
    plot_data, columns=['MaxSupply', 'TotalCoinsMined'], index=df_clust.index
)
plot_df["CoinName"] = df_clust["CoinName"]
plot_df["class"] = df_clust["class"]
plot_df.head()
```

Out[60]:

	MaxSupply	TotalCoinsMined	CoinName	class
NSR	4.761905e-14	3.264108e-04	NuShares	1
TRI	4.761905e-14	9.920193e-09	Triangles Coin	1
CMTC	4.761905e-14	4.604871e-08	CometCoin	0
CHAT	0.000000e+00	5.291598e-05	OpenChat	1
QRL	5.000000e-06	3.985520e-06	Quantum Resistant Ledger	0

In [61]:
```
# Plot the scatter with x="TotalCoinsMined" and y="MaxSupply"
plot_df.hvplot.scatter(
    x="TotalCoinsMined", y="MaxSupply", hover_cols=["CoinName"], by="class"
)
```

Out[61]:

In [ ]:
```

```