Clustering Crypto

In [1]:
```
# Initial imports
import pandas as pd
import hvplot.pandas
from pathlib import Path
import plotly.express as px
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
```

Deliverable 1: Preprocessing the Data for PCA

In [2]:
```
file_path = "Resources/crypto_data.csv"
crypto_df = pd.read_csv(file_path, index_col=0)
# crypto_df=crypto_df.rename(columns={'Unnamed: 0':''})
# crypto_df=crypto_df.set_index("")
crypto_df
```

Out[2]:

	CoinName	Algorithm	IsTrading	ProofType	TotalCoinsMined	TotalCoinSupply
42	42 Coin	Scrypt	True	PoW/PoS	4.199995e+01	42
365	365Coin	X11	True	PoW/PoS	NaN	2300000000
404	404Coin	Scrypt	True	PoW/PoS	1.055185e+09	532000000
611	SixEleven	SHA-256	True	PoW	NaN	611000
808	808	SHA-256	True	PoW/PoS	0.000000e+00	0
...	...	...	...	...	...	...
XBC	BitcoinPlus	Scrypt	True	PoS	1.283270e+05	1000000
DVTC	DivotyCoin	Scrypt	False	PoW/PoS	2.149121e+07	100000000
GIOT	Giotto Coin	Scrypt	False	PoW/PoS	NaN	233100000
OPSC	OpenSourceCoin	SHA-256	False	PoW/PoS	NaN	21000000
PUNK	SteamPunk	PoS	False	PoS	NaN	40000000

1252 rows × 6 columns

In [3]:
```
crypto_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1252 entries, 42 to PUNK
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CoinName         1252 non-null   object 
 1   Algorithm        1252 non-null   object 
 2   IsTrading        1252 non-null   bool   
 3   ProofType        1252 non-null   object 
 4   TotalCoinsMined  744 non-null    float64
 5   TotalCoinSupply  1252 non-null   object 
dtypes: bool(1), float64(1), object(4)
memory usage: 59.9+ KB
```

In [4]:
```
# Keep all the cryptocurrencies that are being traded.
crypto_df=crypto_df[crypto_df['IsTrading']==True]
crypto_df
```

Out[4]:

	CoinName	Algorithm	IsTrading	ProofType	TotalCoinsMined	TotalCoinSupply
42	42 Coin	Scrypt	True	PoW/PoS	4.199995e+01	42
365	365Coin	X11	True	PoW/PoS	NaN	2300000000
404	404Coin	Scrypt	True	PoW/PoS	1.055185e+09	532000000
611	SixEleven	SHA-256	True	PoW	NaN	611000
808	808	SHA-256	True	PoW/PoS	0.000000e+00	0
...	...	...	...	...	...	...
SERO	Super Zero	Ethash	True	PoW	NaN	1000000000
UOS	UOS	SHA-256	True	DPoI	NaN	1000000000
BDX	Beldex	CryptoNight	True	PoW	9.802226e+08	1400222610
ZEN	Horizen	Equihash	True	PoW	7.296538e+06	21000000
XBC	BitcoinPlus	Scrypt	True	PoS	1.283270e+05	1000000

1144 rows × 6 columns

In [5]:
```
# Keep all the cryptocurrencies that have a working algorithm.
crypto_df=crypto_df[crypto_df['Algorithm'].notnull()==True]
crypto_df
```

Out[5]:

	CoinName	Algorithm	IsTrading	ProofType	TotalCoinsMined	TotalCoinSupply
42	42 Coin	Scrypt	True	PoW/PoS	4.199995e+01	42
365	365Coin	X11	True	PoW/PoS	NaN	2300000000
404	404Coin	Scrypt	True	PoW/PoS	1.055185e+09	532000000
611	SixEleven	SHA-256	True	PoW	NaN	611000
808	808	SHA-256	True	PoW/PoS	0.000000e+00	0
...	...	...	...	...	...	...
SERO	Super Zero	Ethash	True	PoW	NaN	1000000000
UOS	UOS	SHA-256	True	DPoI	NaN	1000000000
BDX	Beldex	CryptoNight	True	PoW	9.802226e+08	1400222610
ZEN	Horizen	Equihash	True	PoW	7.296538e+06	21000000
XBC	BitcoinPlus	Scrypt	True	PoS	1.283270e+05	1000000

1144 rows × 6 columns

In [6]:
```
# Remove the "IsTrading" column.
crypto_df=crypto_df.drop(columns='IsTrading')
crypto_df
```

Out[6]:

	CoinName	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply
42	42 Coin	Scrypt	PoW/PoS	4.199995e+01	42
365	365Coin	X11	PoW/PoS	NaN	2300000000
404	404Coin	Scrypt	PoW/PoS	1.055185e+09	532000000
611	SixEleven	SHA-256	PoW	NaN	611000
808	808	SHA-256	PoW/PoS	0.000000e+00	0
...	...	...	...	...	...
SERO	Super Zero	Ethash	PoW	NaN	1000000000
UOS	UOS	SHA-256	DPoI	NaN	1000000000
BDX	Beldex	CryptoNight	PoW	9.802226e+08	1400222610
ZEN	Horizen	Equihash	PoW	7.296538e+06	21000000
XBC	BitcoinPlus	Scrypt	PoS	1.283270e+05	1000000

1144 rows × 5 columns

In [7]:
```
print(crypto_df.shape)
```

```
(1144, 5)
```

In [8]:
```
# Remove rows that have at least 1 null value.
crypto_df=crypto_df.dropna()
print(crypto_df.shape)
crypto_df
```

```
(685, 5)
```

Out[8]:

	CoinName	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply
42	42 Coin	Scrypt	PoW/PoS	4.199995e+01	42
404	404Coin	Scrypt	PoW/PoS	1.055185e+09	532000000
808	808	SHA-256	PoW/PoS	0.000000e+00	0
1337	EliteCoin	X13	PoW/PoS	2.927942e+10	314159265359
BTC	Bitcoin	SHA-256	PoW	1.792718e+07	21000000
...	...	...	...	...	...
ZEPH	ZEPHYR	SHA-256	DPoS	2.000000e+09	2000000000
GAP	Gapcoin	Scrypt	PoW/PoS	1.493105e+07	250000000
BDX	Beldex	CryptoNight	PoW	9.802226e+08	1400222610
ZEN	Horizen	Equihash	PoW	7.296538e+06	21000000
XBC	BitcoinPlus	Scrypt	PoS	1.283270e+05	1000000

685 rows × 5 columns

In [9]:
```
# Keep the rows where coins are mined.
crypto_df=crypto_df[crypto_df['TotalCoinsMined']>0]
print(crypto_df.shape)
crypto_df.head(10)
```

```
(532, 5)
```

Out[9]:

	CoinName	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply
42	42 Coin	Scrypt	PoW/PoS	4.199995e+01	42
404	404Coin	Scrypt	PoW/PoS	1.055185e+09	532000000
1337	EliteCoin	X13	PoW/PoS	2.927942e+10	314159265359
BTC	Bitcoin	SHA-256	PoW	1.792718e+07	21000000
ETH	Ethereum	Ethash	PoW	1.076842e+08	0
LTC	Litecoin	Scrypt	PoW	6.303924e+07	84000000
DASH	Dash	X11	PoW/PoS	9.031294e+06	22000000
XMR	Monero	CryptoNight-V7	PoW	1.720114e+07	0
ETC	Ethereum Classic	Ethash	PoW	1.133597e+08	210000000
ZEC	ZCash	Equihash	PoW	7.383056e+06	21000000

In [10]:
```
# Create a new DataFrame that holds only the cryptocurrencies names.
crypto_name_df=crypto_df[['CoinName']]
print(crypto_name_df.shape)
crypto_name_df.head()
```

```
(532, 1)
```

Out[10]:

	CoinName
42	42 Coin
404	404Coin
1337	EliteCoin
BTC	Bitcoin
ETH	Ethereum

In [11]:
```
# Drop the 'CoinName' column since it's not going to be used on the cluster
crypto_df=crypto_df.drop(columns='CoinName')
print(crypto_df.shape)
crypto_df
```

```
(532, 4)
```

Out[11]:

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply
42	Scrypt	PoW/PoS	4.199995e+01	42
404	Scrypt	PoW/PoS	1.055185e+09	532000000
1337	X13	PoW/PoS	2.927942e+10	314159265359
BTC	SHA-256	PoW	1.792718e+07	21000000
ETH	Ethash	PoW	1.076842e+08	0
...	...	...	...	...
ZEPH	SHA-256	DPoS	2.000000e+09	2000000000
GAP	Scrypt	PoW/PoS	1.493105e+07	250000000
BDX	CryptoNight	PoW	9.802226e+08	1400222610
ZEN	Equihash	PoW	7.296538e+06	21000000
XBC	Scrypt	PoS	1.283270e+05	1000000

532 rows × 4 columns

In [12]:
```
crypto_df["TotalCoinSupply"] = crypto_df["TotalCoinSupply"].astype(dtype='f
crypto_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 532 entries, 42 to XBC
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Algorithm        532 non-null    object 
 1   ProofType        532 non-null    object 
 2   TotalCoinsMined  532 non-null    float64
 3   TotalCoinSupply  532 non-null    float64
dtypes: float64(2), object(2)
memory usage: 20.8+ KB
```

In [13]:
```
# Use get_dummies() to create variables for text features.
X=pd.get_dummies(crypto_df, columns=['Algorithm', 'ProofType'])
print(X.shape)
X.head(10)
```

```
(532, 98)
```

Out[13]:

	TotalCoinsMined	TotalCoinSupply	Algorithm_1GB AES Pattern Search	Algorithm_536	Algorithm_Argon2d	Algorit
42	4.199995e+01	4.200000e+01	0	0	0	
404	1.055185e+09	5.320000e+08	0	0	0	
1337	2.927942e+10	3.141593e+11	0	0	0	
BTC	1.792718e+07	2.100000e+07	0	0	0	
ETH	1.076842e+08	0.000000e+00	0	0	0	
LTC	6.303924e+07	8.400000e+07	0	0	0	
DASH	9.031294e+06	2.200000e+07	0	0	0	
XMR	1.720114e+07	0.000000e+00	0	0	0	
ETC	1.133597e+08	2.100000e+08	0	0	0	
ZEC	7.383056e+06	2.100000e+07	0	0	0	

10 rows × 98 columns

In [14]:
```
# Standardize the data with StandardScaler().
ss = StandardScaler()
X_scaled=ss.fit_transform(X)
X_scaled[0:5]
```

Out[14]:
```
array([[-0.11710817, -0.1528703 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.07530656, -0.0433963 , -0.06142951, -0.06142951,
        -0.0433963 , -0.0433963 , -0.19245009, -0.06142951, -0.09740465,
        -0.0433963 , -0.11547005, -0.07530656, -0.0433963 , -0.0433963 ,
        -0.15191091, -0.0433963 , -0.13118084, -0.0433963 , -0.0433963 ,
        -0.08703883, -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.0433963 , -0.08703883, -0.08703883, -0.08703883,
        -0.0433963 , -0.13118084, -0.13840913, -0.13840913, -0.0433963 ,
        -0.06142951, -0.0433963 , -0.07530656, -0.18168574, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.07530656, -0.15826614, -0.31491833,
        -0.0433963 , -0.08703883, -0.07530656, -0.06142951,  1.38675049,
        -0.0433963 , -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.39879994, -0.0433963 , -0.18168574, -0.0433963 ,
        -0.08703883, -0.08703883, -0.10680283, -0.0433963 , -0.13118084,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.07530656,
        -0.43911856, -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
        -0.89632016, -0.0433963 , -0.0433963 ,  1.42222617, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 ],
       [-0.09396955, -0.145009  , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.07530656, -0.0433963 , -0.06142951, -0.06142951,
        -0.0433963 , -0.0433963 , -0.19245009, -0.06142951, -0.09740465,
        -0.0433963 , -0.11547005, -0.07530656, -0.0433963 , -0.0433963 ,
        -0.15191091, -0.0433963 , -0.13118084, -0.0433963 , -0.0433963 ,
        -0.08703883, -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.0433963 , -0.08703883, -0.08703883, -0.08703883,
        -0.0433963 , -0.13118084, -0.13840913, -0.13840913, -0.0433963 ,
        -0.06142951, -0.0433963 , -0.07530656, -0.18168574, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.07530656, -0.15826614, -0.31491833,
        -0.0433963 , -0.08703883, -0.07530656, -0.06142951,  1.38675049,
        -0.0433963 , -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.39879994, -0.0433963 , -0.18168574, -0.0433963 ,
        -0.08703883, -0.08703883, -0.10680283, -0.0433963 , -0.13118084,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.07530656,
        -0.43911856, -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
        -0.89632016, -0.0433963 , -0.0433963 ,  1.42222617, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 ],
       [ 0.52494561,  4.48942416, -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.07530656, -0.0433963 , -0.06142951, -0.06142951,
        -0.0433963 , -0.0433963 , -0.19245009, -0.06142951, -0.09740465,
        -0.0433963 , -0.11547005, -0.07530656, -0.0433963 , -0.0433963 ,
        -0.15191091, -0.0433963 , -0.13118084, -0.0433963 , -0.0433963 ,
        -0.08703883, -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.0433963 , -0.08703883, -0.08703883, -0.08703883,
        -0.0433963 , -0.13118084, -0.13840913, -0.13840913, -0.0433963 ,
        -0.06142951, -0.0433963 , -0.07530656, -0.18168574, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.07530656, -0.15826614, -0.31491833,
        -0.0433963 , -0.08703883, -0.07530656, -0.06142951, -0.72111026,
        -0.0433963 , -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
```

```
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.39879994, -0.0433963 ,  5.50400923, -0.0433963 ,
        -0.08703883, -0.08703883, -0.10680283, -0.0433963 , -0.13118084,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.07530656,
        -0.43911856, -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
        -0.89632016, -0.0433963 , -0.0433963 ,  1.42222617, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 ],
       [-0.11671506, -0.15255998, -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.07530656, -0.0433963 , -0.06142951, -0.06142951,
        -0.0433963 , -0.0433963 , -0.19245009, -0.06142951, -0.09740465,
        -0.0433963 , -0.11547005, -0.07530656, -0.0433963 , -0.0433963 ,
        -0.15191091, -0.0433963 , -0.13118084, -0.0433963 , -0.0433963 ,
        -0.08703883, -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.0433963 , -0.08703883, -0.08703883, -0.08703883,
        -0.0433963 , -0.13118084, -0.13840913, -0.13840913, -0.0433963 ,
        -0.06142951, -0.0433963 , -0.07530656, -0.18168574, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.07530656, -0.15826614,  3.17542648,
        -0.0433963 , -0.08703883, -0.07530656, -0.06142951, -0.72111026,
        -0.0433963 , -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.39879994, -0.0433963 , -0.18168574, -0.0433963 ,
        -0.08703883, -0.08703883, -0.10680283, -0.0433963 , -0.13118084,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.07530656,
        -0.43911856, -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
         1.11567277, -0.0433963 , -0.0433963 , -0.70312305, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 ],
       [-0.11474682, -0.1528703 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.07530656, -0.0433963 , -0.06142951, -0.06142951,
        -0.0433963 , -0.0433963 , -0.19245009, -0.06142951, -0.09740465,
        -0.0433963 , -0.11547005, -0.07530656, -0.0433963 , -0.0433963 ,
        -0.15191091, -0.0433963 ,  7.62306442, -0.0433963 , -0.0433963 ,
        -0.08703883, -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.06142951, -0.0433963 , -0.08703883, -0.08703883, -0.08703883,
        -0.0433963 , -0.13118084, -0.13840913, -0.13840913, -0.0433963 ,
        -0.06142951, -0.0433963 , -0.07530656, -0.18168574, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.07530656, -0.15826614, -0.31491833,
        -0.0433963 , -0.08703883, -0.07530656, -0.06142951, -0.72111026,
        -0.0433963 , -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.39879994, -0.0433963 , -0.18168574, -0.0433963 ,
        -0.08703883, -0.08703883, -0.10680283, -0.0433963 , -0.13118084,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.07530656,
        -0.43911856, -0.0433963 , -0.06142951, -0.0433963 , -0.0433963 ,
         1.11567277, -0.0433963 , -0.0433963 , -0.70312305, -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 , -0.0433963 ,
        -0.0433963 , -0.0433963 , -0.0433963 ]])
```

Deliverable 2: Reducing Data Dimensions Using PCA

In [15]:
```
# Using PCA to reduce dimension to three principal components.
pca = PCA(n_components=3)
pca.fit(X_scaled)
print(pca.explained_variance_ratio_)
```

```
[0.02792896 0.02134723 0.02050469]
```

In [16]:
```
X_pca = pca.transform(X_scaled)
# x_pca = pca.fit_transform(x_scaled)
X_pca
```

Out[16]:
```
array([[-0.33665624,  1.01643122, -0.58495813],
       [-0.31997177,  1.01634871, -0.58525676],
       [ 2.31522196,  1.58532778, -0.674797  ],
       ...,
       [ 0.31839425, -2.23834341,  0.4358893 ],
       [-0.1442316 , -2.15543016,  0.45566662],
       [-0.29037736,  0.78326097, -0.26606354]])
```

In [17]:
```
# Create a DataFrame with the three principal components.
pca_df = pd.DataFrame(X_pca, columns=['PC 1','PC 2', 'PC 3'], index=crypto_
pca_df
```

Out[17]:

	PC 1	PC 2	PC 3
42	-0.336656	1.016431	-0.584958
404	-0.319972	1.016349	-0.585257
1337	2.315222	1.585328	-0.674797
BTC	-0.144116	-1.277804	0.205931
ETH	-0.157103	-1.971531	0.385568
...	...	...	...
ZEPH	2.464278	0.865244	0.018717
GAP	-0.334700	1.016296	-0.584976
BDX	0.318394	-2.238343	0.435889
ZEN	-0.144232	-2.155430	0.455667
XBC	-0.290377	0.783261	-0.266064

532 rows × 3 columns

Deliverable 3: Clustering Crytocurrencies Using K-Means

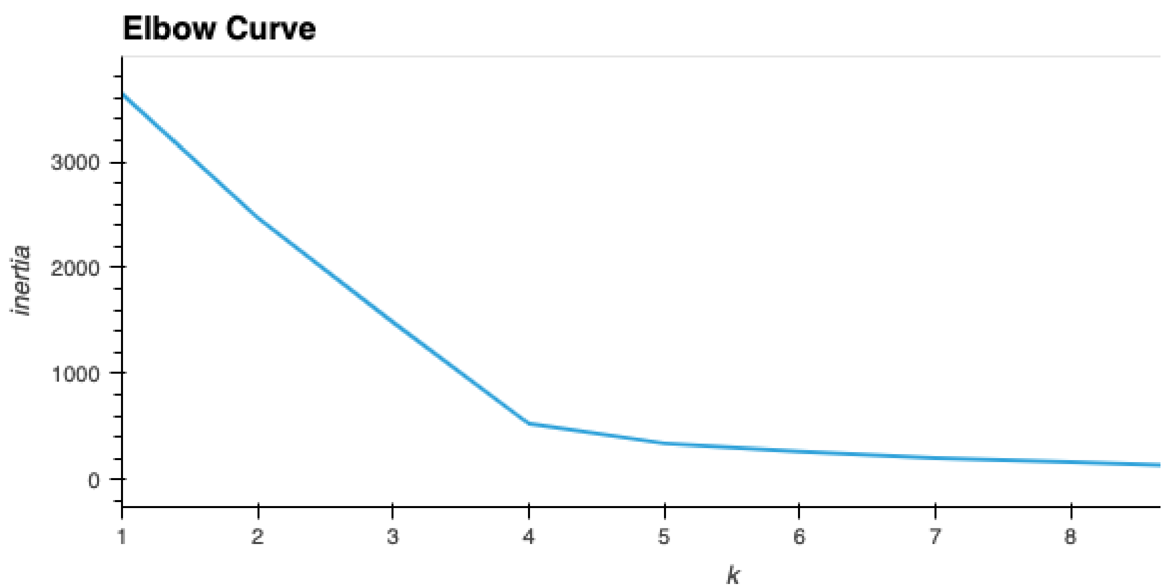
Finding the Best Value for `k` Using the Elbow Curve

In [18]:
```
# Create an elbow curve to find the best value for K.
# YOUR CODE HERE
inertia = []
k = list(range(1, 11))

for i in k:
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(pca_df)
    inertia.append(km.inertia_)

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

Out[18]:



Running K-Means with `k=4`

In [19]:
```
# Initialize the K-Means model.
model = KMeans(n_clusters=4, random_state=0)

# Fit the model
model.fit(pca_df)

# Predict clusters
predictions = model.predict(pca_df)
predictions
```

Out[19]:
```
array([0, 0, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 0, 0, 2, 0, 2, 2, 0, 0, 2, 2,
       2, 2, 2, 0, 2, 2, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 0, 0,
       2, 2, 2, 2, 2, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 0, 2, 0, 0, 0, 2,
       2, 2, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 2, 0, 2, 0, 0, 2, 2, 2, 2, 0,
       0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 2, 2, 0, 0, 2, 0, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 0, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 0, 2, 2, 2, 0,
       2, 0, 2, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 2, 2, 0, 0, 2, 0, 2, 0, 0,
       0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0,
       0, 2, 0, 2, 0, 0, 2, 0, 2, 0, 0, 2, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0,
       2, 0, 0, 0, 0, 0, 2, 2, 0, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0,
       0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0,
       0, 2, 0, 2, 2, 0, 2, 2, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0,
       0, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 2, 2, 2, 0, 2, 0, 0, 2,
       0, 2, 2, 2, 0, 2, 0, 2, 2, 2, 0, 2, 0, 2, 0, 0, 0, 2, 0, 2, 2, 2,
       2, 2, 0, 0, 2, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0,
       2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0, 2, 0, 2, 2, 2, 0, 0, 2, 2, 0,
       0, 2, 0, 2, 2, 2, 0, 2, 2, 0, 0, 0, 2, 2, 2, 0, 0, 0, 2, 2, 0, 2,
       2, 2, 2, 0, 3, 3, 2, 2, 2, 0, 3, 0, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0,
       2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 2, 2, 0, 2, 0, 2, 2, 2, 2,
       0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0,
       2, 2, 2, 2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 2, 3, 2, 0, 2, 2, 0, 0,
       2, 0, 2, 2, 2, 2, 2, 0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 2, 2, 2,
       0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 2, 0, 2, 0,
       0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 3, 0, 1, 0, 0,
       0, 2, 2, 0], dtype=int32)
```

In [20]:
```
# Create a new DataFrame including predicted clusters and cryptocurrencies
# Concatentate the crypto_df and pcs_df DataFrames on the same columns.

clustered_df=crypto_df.join(pca_df, how='left')
clustered_df
```

Out[20]:

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC 1	PC 2	PC 3
42	Scrypt	PoW/PoS	4.199995e+01	4.200000e+01	-0.336656	1.016431	-0.584958
404	Scrypt	PoW/PoS	1.055185e+09	5.320000e+08	-0.319972	1.016349	-0.585257
1337	X13	PoW/PoS	2.927942e+10	3.141593e+11	2.315222	1.585328	-0.674797
BTC	SHA-256	PoW	1.792718e+07	2.100000e+07	-0.144116	-1.277804	0.205931
ETH	Ethash	PoW	1.076842e+08	0.000000e+00	-0.157103	-1.971531	0.385568
...	...	...	...	...	...	...	...
ZEPH	SHA-256	DPoS	2.000000e+09	2.000000e+09	2.464278	0.865244	0.018717
GAP	Scrypt	PoW/PoS	1.493105e+07	2.500000e+08	-0.334700	1.016296	-0.584976
BDX	CryptoNight	PoW	9.802226e+08	1.400223e+09	0.318394	-2.238343	0.435889
ZEN	Equihash	PoW	7.296538e+06	2.100000e+07	-0.144232	-2.155430	0.455667
XBC	Scrypt	PoS	1.283270e+05	1.000000e+06	-0.290377	0.783261	-0.266064

532 rows × 7 columns

In [21]:
```
#  Add a new column, "CoinName" to the clustered_df DataFrame that holds th
clustered_df['CoinName']=crypto_name_df['CoinName']
clustered_df.head()
```

Out[21]:

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC 1	PC 2	PC 3	Coin
42	Scrypt	PoW/PoS	4.199995e+01	4.200000e+01	-0.336656	1.016431	-0.584958	4.
404	Scrypt	PoW/PoS	1.055185e+09	5.320000e+08	-0.319972	1.016349	-0.585257	40
1337	X13	PoW/PoS	2.927942e+10	3.141593e+11	2.315222	1.585328	-0.674797	Eli
BTC	SHA-256	PoW	1.792718e+07	2.100000e+07	-0.144116	-1.277804	0.205931	E
ETH	Ethash	PoW	1.076842e+08	0.000000e+00	-0.157103	-1.971531	0.385568	Eth

In [22]:
```
# Add a new column, "Class" to the clustered_df DataFrame that holds the pr
clustered_df['Class']=predictions

# Print the shape of the clustered_df
print(clustered_df.shape)
clustered_df.head(10)
```

(532, 9)

Out[22]:

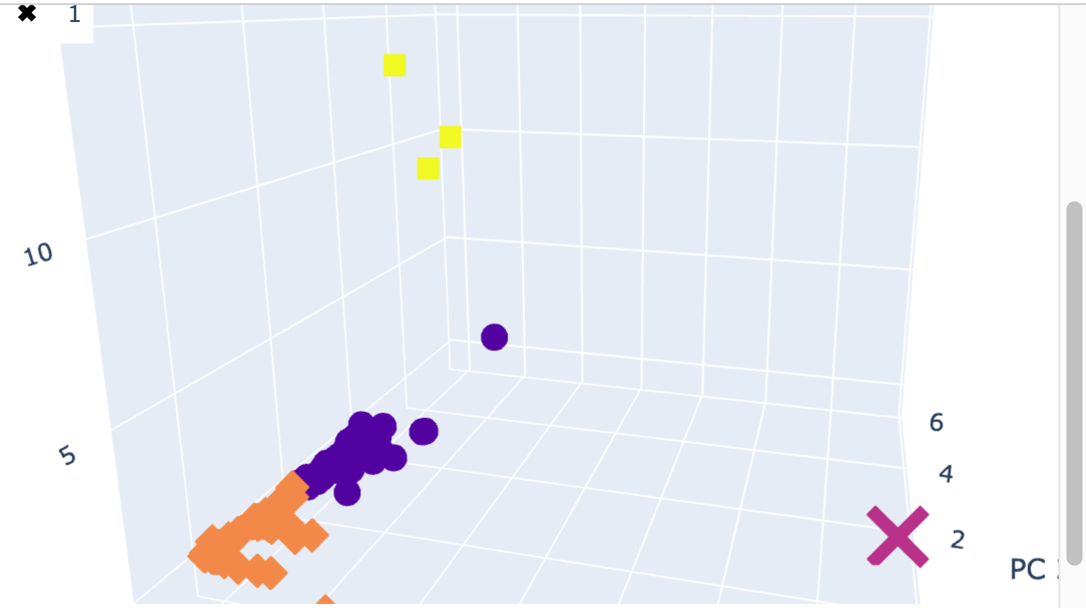
	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC 1	PC 2	PC 3
42	Scrypt	PoW/PoS	4.199995e+01	4.200000e+01	-0.336656	1.016431	-0.584958
404	Scrypt	PoW/PoS	1.055185e+09	5.320000e+08	-0.319972	1.016349	-0.585257
1337	X13	PoW/PoS	2.927942e+10	3.141593e+11	2.315222	1.585328	-0.674797
BTC	SHA-256	PoW	1.792718e+07	2.100000e+07	-0.144116	-1.277804	0.205931
ETH	Ethash	PoW	1.076842e+08	0.000000e+00	-0.157103	-1.971531	0.385568
LTC	Scrypt	PoW	6.303924e+07	8.400000e+07	-0.173957	-1.089301	0.001488
DASH	X11	PoW/PoS	9.031294e+06	2.200000e+07	-0.385430	1.146427	-0.500860
XMR	CryptoNight-V7	PoW	1.720114e+07	0.000000e+00	-0.155641	-2.183219	0.435679
ETC	Ethash	PoW	1.133597e+08	2.100000e+08	-0.155544	-1.971646	0.385554
ZEC	Equihash	PoW	7.383056e+06	2.100000e+07	-0.144231	-2.155430	0.455667

Deliverable 4: Visualizing Cryptocurrencies Results

3D-Scatter with Clusters

In [23]:
```
# Creating a 3D-Scatter with the PCA data and the clusters
fig = px.scatter_3d(
    clustered_df,
    x='PC 1',
    y='PC 2',
    z='PC 3',
    color='Class',
    symbol='Class',
    hover_name='CoinName',
    hover_data=['Algorithm'],
    width=800)

fig.update_layout(legend=dict(x=0,y=1))
fig.show()
```



In [24]:
```
# Create a table with tradable cryptocurrencies.
tradable_df=clustered_df.hvplot.table(columns=
['CoinName', 'Algorithm', 'ProofType', 'TotalCoinSupply','TotalCoinsMined',
tradable_df
```

Out[24]:

#	CoinName	Algorithm	ProofType	TotalCoinSupply	TotalCoinsMined
0	42 Coin	Scrypt	PoW/PoS	42.0	41.999954
1	404Coin	Scrypt	PoW/PoS	532,000,000.0	1,055,184,902.04
2	EliteCoin	X13	PoW/PoS	314,159,265,359.0	29,279,424,622.5027
3	Bitcoin	SHA-256	PoW	21,000,000.0	17,927,175.0
4	Ethereum	Ethash	PoW	0.0	107,684,222.6865
5	Litecoin	Scrypt	PoW	84,000,000.0	63,039,243.300005
6	Dash	X11	PoW/PoS	22,000,000.0	9,031,294.375634
7	Monero	CryptoNight-V7	PoW	0.0	17,201,143.144913
8	Ethereum Classic	Ethash	PoW	210,000,000.0	113,359,703.0
9	ZCash	Equihash	PoW	21,000,000.0	7,383,056.25
10	Bitshares	SHA-512	PoS	3,600,570,502.0	2,741,570,000.0

In [28]:
```
# Print the total number of tradable cryptocurrencies.
total_num=len(tradable_df['CoinName'])
print(f'There are {total_num} tradable cryptocurrencies.')
```

```
There are 532 tradable cryptocurrencies.
```

In [38]:
```
# Scaling data to create the scatter plot with tradable cryptocurrencies.
Mm=clustered_df[['TotalCoinSupply', 'TotalCoinsMined']]
Mm_scaled=MinMaxScaler().fit_transform(Mm)
Mm_scaled
```

Out[38]:
```
array([[4.20000000e-11, 0.00000000e+00],
       [5.32000000e-04, 1.06585544e-03],
       [3.14159265e-01, 2.95755135e-02],
       ...,
       [1.40022261e-03, 9.90135079e-04],
       [2.10000000e-05, 7.37028150e-06],
       [1.00000000e-06, 1.29582282e-07]])
```

In [43]:
```
# Create a new DataFrame that has the scaled data with the clustered_df Dat
plot_df = pd.DataFrame(Mm_scaled,columns=('TotalCoinSupply','TotalCoinsMine

# Add the "CoinName" column from the clustered_df DataFrame to the new Data
plot_df["CoinName"]= clustered_df["CoinName"]

# Add the "Class" column from the clustered_df DataFrame to the new DataFra
plot_df["Class"]= clustered_df["Class"]
plot_df.head(10)
```

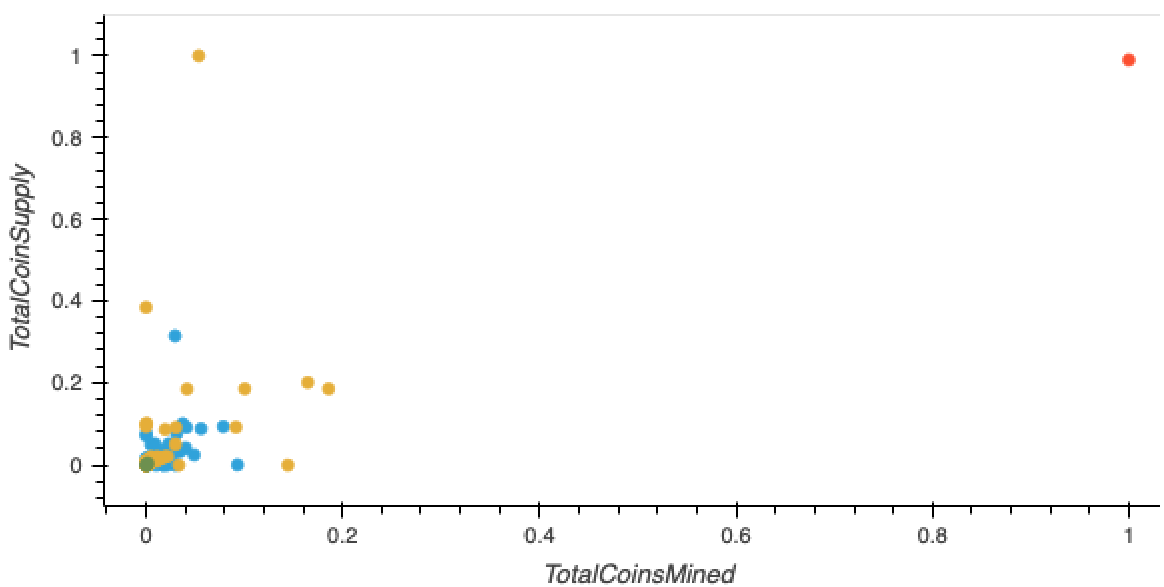
Out[43]:

	TotalCoinSupply	TotalCoinsMined	CoinName	Class
42	4.200000e-11	0.000000	42 Coin	0
404	5.320000e-04	0.001066	404Coin	0
1337	3.141593e-01	0.029576	EliteCoin	0
BTC	2.100000e-05	0.000018	Bitcoin	2
ETH	0.000000e+00	0.000109	Ethereum	2
LTC	8.400000e-05	0.000064	Litecoin	2
DASH	2.200000e-05	0.000009	Dash	0
XMR	0.000000e+00	0.000017	Monero	2
ETC	2.100000e-04	0.000115	Ethereum Classic	2
ZEC	2.100000e-05	0.000007	ZCash	2

In [44]:
```
# Create a hvplot.scatter plot using x="TotalCoinsMined" and y="TotalCoinSu

plot_df.hvplot.scatter(x="TotalCoinsMined", y="TotalCoinSupply", by="Class"
```

Out[44]:



In [ ]: