Clustering Crypto

In [2]:
```
!pip install hvplot
```

```
Collecting hvplot
  Downloading hvplot-0.7.1-py2.py3-none-any.whl (3.1 MB)
Requirement already satisfied: numpy>=1.15 in c:\users\emman\anaconda3\li
b\site-packages (from hvplot) (1.19.2)
Requirement already satisfied: pandas in c:\users\emman\anaconda3\lib\sit
e-packages (from hvplot) (1.2.3)
Collecting holoviews>=1.11.0
  Downloading holoviews-1.14.2-py2.py3-none-any.whl (4.3 MB)
Collecting colorcet>=2
  Downloading colorcet-2.0.6-py2.py3-none-any.whl (1.6 MB)
Requirement already satisfied: bokeh>=1.0.0 in c:\users\emman\anaconda3\l
ib\site-packages (from hvplot) (2.3.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\emman\ana
conda3\lib\site-packages (from bokeh>=1.0.0->hvplot) (2.8.1)
Requirement already satisfied: pillow>=7.1.0 in c:\users\emman\anaconda3
\lib\site-packages (from bokeh>=1.0.0->hvplot) (8.1.1)
Requirement already satisfied: packaging>=16.8 in c:\users\emman\anaconda
3\lib\site-packages (from bokeh>=1.0.0->hvplot) (20.9)
Requirement already satisfied: tornado>=5.1 in c:\users\emman\anaconda3\l
```

In [4]:
```
!pip install plotly
```

```
Collecting plotly
  Downloading plotly-4.14.3-py2.py3-none-any.whl (13.2 MB)
Requirement already satisfied: six in c:\users\emman\anaconda3\lib\site-pac
kages (from plotly) (1.15.0)
Collecting retrying>=1.3.3
  Downloading retrying-1.3.3.tar.gz (10 kB)
Building wheels for collected packages: retrying
  Building wheel for retrying (setup.py): started
  Building wheel for retrying (setup.py): finished with status 'done'
  Created wheel for retrying: filename=retrying-1.3.3-py3-none-any.whl size
=11429 sha256=56863ea69897a397439f34f5ae30afbe1ceb4c5e12bc52948e095bcf879f7
4ec
  Stored in directory: c:\users\emman\appdata\local\pip\cache\wheels\c4\a7
\48\0a434133f6d56e878ca511c0e6c38326907c0792f67b476e56
Successfully built retrying
Installing collected packages: retrying, plotly
Successfully installed plotly-4.14.3 retrying-1.3.3
```

In [5]:
```
!pip install sklearn
```

```
Requirement already satisfied: sklearn in c:\users\emman\anaconda3\lib\site
-packages (0.0)
Requirement already satisfied: scikit-learn in c:\users\emman\anaconda3\lib
\site-packages (from sklearn) (0.24.1)
Requirement already satisfied: numpy>=1.13.3 in c:\users\emman\anaconda3\li
b\site-packages (from scikit-learn->sklearn) (1.19.2)
Requirement already satisfied: scipy>=0.19.1 in c:\users\emman\anaconda3\li
b\site-packages (from scikit-learn->sklearn) (1.6.1)
Requirement already satisfied: joblib>=0.11 in c:\users\emman\anaconda3\lib
\site-packages (from scikit-learn->sklearn) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\emman\anaco
nda3\lib\site-packages (from scikit-learn->sklearn) (2.1.0)
```

In [6]:
```
# Initial imports
import pandas as pd
import hvplot.pandas
from path 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 [7]:
```
# Load the crypto_data.csv dataset.
file_path = "./Resources/crypto_data.csv"
df = pd.read_csv(file_path, index_col=0)
df.head(10)
```

Out[7]:

	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
1337	EliteCoin	X13	True	PoW/PoS	2.927942e+10	314159265359
2015	2015 coin	X11	True	PoW/PoS	NaN	0
BTC	Bitcoin	SHA-256	True	PoW	1.792718e+07	21000000
ETH	Ethereum	Ethash	True	PoW	1.076842e+08	0
LTC	Litecoin	Scrypt	True	PoW	6.303924e+07	84000000

In [8]:

```python
# Keep all the cryptocurrencies that are being traded.
df1 = df.loc[df['IsTrading'] == True]
df1.head()
```

Out[8]:

	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

In [11]:

```python
# Keep all the cryptocurrencies that have a working algorithm.
df0 = df1.sort_values(by='Algorithm', ascending=False)
df0.tail()
```

Out[11]:

	CoinName	Algorithm	IsTrading	ProofType	TotalCoinsMined	TotalCoinSupply
AQUA	Aquachain	Argon2	True	PoW	0.000000e+00	42000000
OPES	Opes	Argon2	True	PoW	NaN	52000000
BOAT	Doubloon	536	True	PoW/PoS	NaN	500000000
ESP	Espers	536	True	PoW/PoS	2.280188e+10	50000000000
HODL	HOdlcoin	1GB AES Pattern Search	True	PoW	1.144895e+07	81962100

In [12]:

```python
# Keep all the cryptocurrencies that have a working algorithm: (Separated DF)
df2 = df1.dropna(axis=0, subset=['Algorithm'])
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1144 entries, 42 to XBC
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CoinName         1144 non-null   object
 1   Algorithm        1144 non-null   object
 2   IsTrading        1144 non-null   bool
 3   ProofType        1144 non-null   object
 4   TotalCoinsMined  685 non-null    float64
 5   TotalCoinSupply  1144 non-null   object
dtypes: bool(1), float64(1), object(4)
memory usage: 54.7+ KB
```

In [13]:

```python
# Remove the "IsTrading" column.
df3 = df2.drop(['IsTrading'], axis=1)
df3.head()
```

Out[13]:

	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

In [14]:

```python
# Remove rows that have at least 1 null value.
df4 = df3.dropna()
df4.head()
```

Out[14]:

	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

In [15]:

```python
# Keep the rows where coins are mined.
df5 = df4.loc[df4['TotalCoinsMined'] > 0]
df5.head()
```

Out[15]:

	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

In [20]:
```python
# Create a new DataFrame that holds only the cryptocurrencies names.
cc_names_df = df5[["CoinName"]]
cc_names_df.head()
```

Out[20]:

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

In [21]:
```python
# Drop the 'CoinName' column since it's not going to be used on the clusterin
crypto_df = df5.drop(['CoinName'], axis=1)
crypto_df.head()
```

Out[21]:

	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

In [24]:
```python
# Use get_dummies() to create variables for text features.
X = pd.get_dummies(crypto_df, columns=['Algorithm', 'ProofType'])
X.head()
```

Out[24]:

	TotalCoinsMined	TotalCoinSupply	Algorithm_1GB AES Pattern Search	Algorithm_536	Algorithm_Argon2d	A
42	4.199995e+01	42	0	0	0	
404	1.055185e+09	532000000	0	0	0	
1337	2.927942e+10	314159265359	0	0	0	
BTC	1.792718e+07	21000000	0	0	0	
ETH	1.076842e+08	0	0	0	0	

5 rows × 98 columns

In [25]:

```python
# Standardize the data with StandardScaler().
X_scaled = StandardScaler().fit_transform(X)
print(X_scaled[0:5])
```

```
[[-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 [29]:

```python
# Using PCA to reduce dimension to three principal components.
pca = PCA(n_components=3)
pca
```

Out[29]:

```
PCA(n_components=3)
```

In [32]:

```python
# Create a DataFrame with the three principal components.
X_pca = pca.fit_transform(X_scaled)
X_pca
```

Out[32]:

```
array([[-0.3220868 ,  1.03894094, -0.53677041],
       [-0.30544688,  1.03917226, -0.53698676],
       [ 2.27528664,  1.72867421, -0.61335136],
       ...,
       [ 0.31019403, -2.19750788,  0.42685255],
       [-0.17387158, -2.12812397,  0.49508571],
       [-0.26863594,  0.79024376, -0.24389572]])
```

In [34]:
```
index_values = (X.index.tolist())
index_values
```

Out[34]:
```
['42',
 '404',
 '1337',
 'BTC',
 'ETH',
 'LTC',
 'DASH',
 'XMR',
 'ETC',
 'ZEC',
 'BTS',
 'DGB',
 'BTCD',
 'XPY',
 'PRC',
 'KOBO',
 'SPR',
 'ARG',
 'AUR',
```

In [35]:
```
pcs_df = pd.DataFrame(data = X_pca, columns=["PC 1", "PC 2", "PC 3"], index =
pcs_df.head()
```

Out[35]:

	PC 1	PC 2	PC 3
42	-0.322087	1.038941	-0.536770
404	-0.305447	1.039172	-0.536987
1337	2.275287	1.728674	-0.613351
BTC	-0.144795	-1.269110	0.171645
ETH	-0.142958	-1.924491	0.301421

Deliverable 3: Clustering Crytocurrencies Using K-Means

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

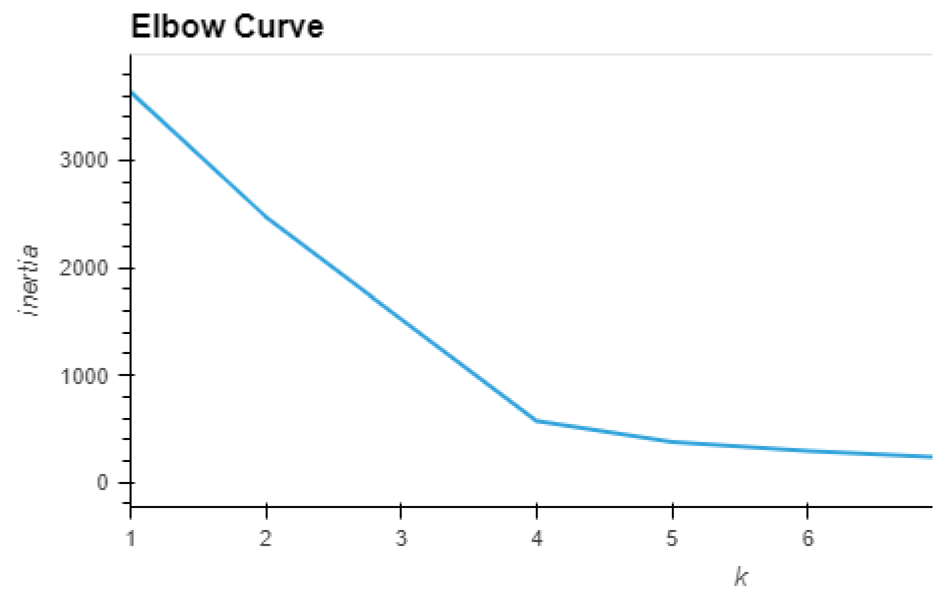
In [39]:
```
# Create an elbow curve to find the best value for K.
inertia = []
k = list(range(1, 11))
```

In [41]:
```
for i in k:
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(pcs_df)
    inertia.append(km.inertia_)
```

```
C:\Users\emman\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, whe
n there are less chunks than available threads. You can avoid it by setting
the environment variable OMP_NUM_THREADS=3.
  warnings.warn(
```

In [42]:
```
# Plot the elbow curve
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[42]:



Running K-Means with k=4

In [43]:

```python
# Initialize the K-Means model.
model = KMeans(n_clusters=4, random_state=0)

# Fit the model
model.fit(pcs_df)

# Predict clusters
predictions = model.predict(pcs_df)
print(predictions)
pcs_df["Class"] = model.labels_
```

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

In [49]:

```python
# Create a new DataFrame including predicted clusters and cryptocurrencies fe
# Concatentate the crypto_df and pcs_df DataFrames on the same columns.
clustered_df = crypto_df.join(pcs_df, how='inner')
clustered_df.head()
```

Out[49]:

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC 1	PC 2	PC 3
42	Scrypt	PoW/PoS	4.199995e+01	42	-0.322087	1.038941	-0.536770
404	Scrypt	PoW/PoS	1.055185e+09	532000000	-0.305447	1.039172	-0.536987
1337	X13	PoW/PoS	2.927942e+10	314159265359	2.275287	1.728674	-0.613351
BTC	SHA-256	PoW	1.792718e+07	21000000	-0.144795	-1.269110	0.171645
ETH	Ethash	PoW	1.076842e+08	0	-0.142958	-1.924491	0.301421

In [50]:

```python
#  Add a new column, "CoinName" to the clustered_df DataFrame that holds the
clustered_df = clustered_df.join(cc_names_df, how='inner')
clustered_df.head()
```

Out[50]:

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC 1	PC 2	PC 3
42	Scrypt	PoW/PoS	4.199995e+01	42	-0.322087	1.038941	-0.536770
404	Scrypt	PoW/PoS	1.055185e+09	532000000	-0.305447	1.039172	-0.536987
1337	X13	PoW/PoS	2.927942e+10	314159265359	2.275287	1.728674	-0.613351
BTC	SHA-256	PoW	1.792718e+07	21000000	-0.144795	-1.269110	0.171645
ETH	Ethash	PoW	1.076842e+08	0	-0.142958	-1.924491	0.301421

In [51]:

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

```
(532, 9)
```

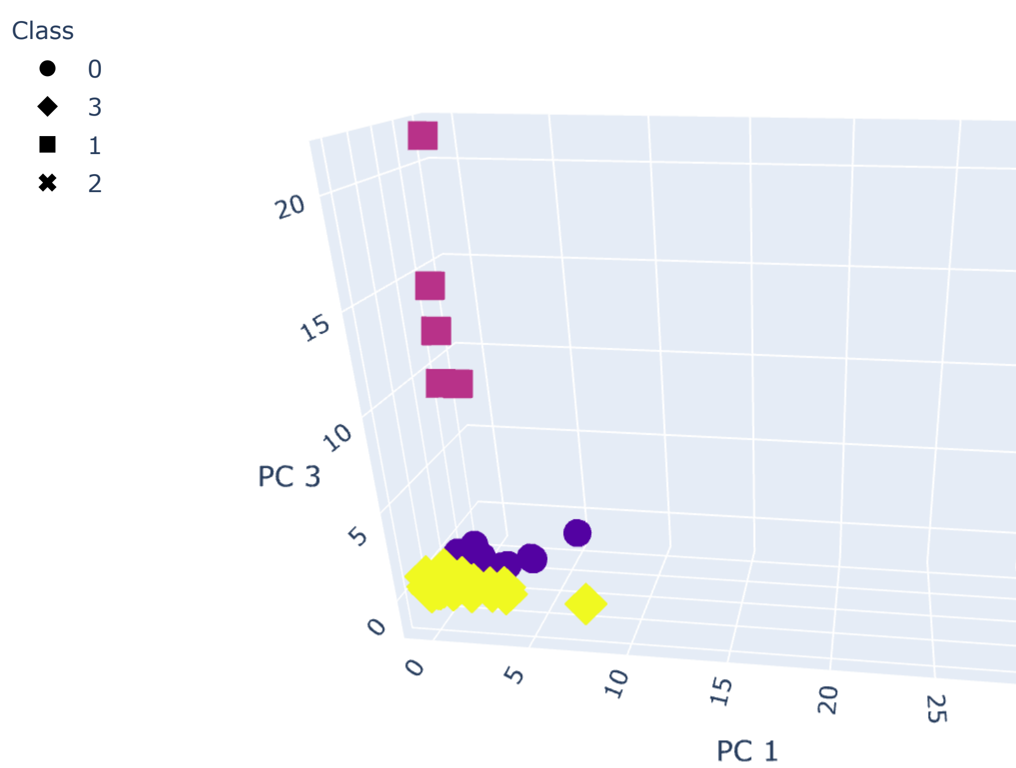
Out[51]:

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC 1	PC 2	P
42	Scrypt	PoW/PoS	4.199995e+01	42	-0.322087	1.038941	-0.5367
404	Scrypt	PoW/PoS	1.055185e+09	532000000	-0.305447	1.039172	-0.5369
1337	X13	PoW/PoS	2.927942e+10	314159265359	2.275287	1.728674	-0.613
BTC	SHA-256	PoW	1.792718e+07	21000000	-0.144795	-1.269110	0.171
ETH	Ethash	PoW	1.076842e+08	0	-0.142958	-1.924491	0.301
LTC	Scrypt	PoW	6.303924e+07	84000000	-0.145790	-1.085534	0.014
DASH	X11	PoW/PoS	9.031294e+06	22000000	-0.420950	1.164486	-0.552
XMR	CryptoNight-V7	PoW	1.720114e+07	0	-0.153835	-2.117314	0.387
ETC	Ethash	PoW	1.133597e+08	210000000	-0.141405	-1.924569	0.301
ZEC	Equihash	PoW	7.383056e+06	21000000	-0.173871	-2.128124	0.495

Deliverable 4: Visualizing Cryptocurrencies Results

3D-Scatter with Clusters

In [67]:
```python
# 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", "TotalCoinsMined", "TotalCoinSupply"])
fig.update_layout(legend=dict(x=0, y=1))
fig.show()
```



In [68]:

```python
# Create a table with tradable cryptocurrencies.
clustered_df.hvplot.table(columns=['CoinName', 'Algorithm', 'ProofType', 'Tot
```

Out[68]:

#	CoinName	Algorithm	ProofType	TotalCoinsMined
0	42 Coin	Scrypt	PoW/PoS	41.999954
1	404Coin	Scrypt	PoW/PoS	1,055,184,902.04
2	EliteCoin	X13	PoW/PoS	29,279,424,622.502
3	Bitcoin	SHA-256	PoW	17,927,175.0
4	Ethereum	Ethash	PoW	107,684,222.6865
5	Litecoin	Scrypt	PoW	63,039,243.300005
6	Dash	X11	PoW/PoS	9,031,294.375634
7	Monero	CryptoNight-V7	PoW	17,201,143.144913
8	Ethereum Classic	Ethash	PoW	113,359,703.0
9	ZCash	Equihash	PoW	7,383,056.25
10	Bitshares	SHA-512	PoS	2,741,570,000.0

In [74]:

```python
# Print the total number of tradable cryptocurrencies.
clustered_df['CoinName'].count()
```

Out[74]: 532

In [75]:

```python
# Scaling data to create the scatter plot with tradable cryptocurrencies.
cluster_df = clustered_df[['TotalCoinSupply', 'TotalCoinsMined']]
X_minmax = MinMaxScaler().fit_transform(cluster_df)
X_minmax
```

Out[75]:

```
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 [76]:

```python
# Create a new DataFrame that has the scaled data with the clustered_df DataF
index_values = (clustered_df.index.tolist())
plot_df = pd.DataFrame(
    data = X_minmax, columns=["TotalCoinSupply_scaled", "TotalCoinsMined_scal

# Add the "CoinName" column from the clustered_df DataFrame to the new DataFr
plot_df = plot_df.join(cc_names_df, how='inner')

# Add the "Class" column from the clustered_df DataFrame to the new DataFrame
class_df = clustered_df['Class']
plot_df = plot_df.join(class_df, how='inner')

plot_df.head(10)
```

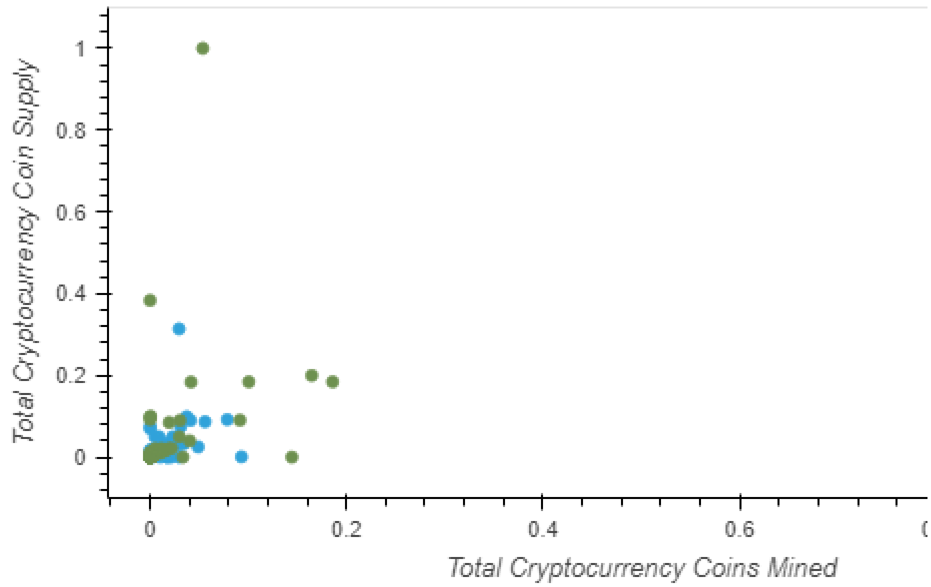
Out[76]:

	TotalCoinSupply_scaled	TotalCoinsMined_scaled	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	3
ETH	0.000000e+00	0.000109	Ethereum	3
LTC	8.400000e-05	0.000064	Litecoin	3
DASH	2.200000e-05	0.000009	Dash	0
XMR	0.000000e+00	0.000017	Monero	3
ETC	2.100000e-04	0.000115	Ethereum Classic	3
ZEC	2.100000e-05	0.000007	ZCash	3

In [77]:

```
# Create a hvplot.scatter plot using x="TotalCoinsMined" and y="TotalCoinSupp
plot_df.hvplot.scatter(x="TotalCoinsMined_scaled", y="TotalCoinSupply_scaled"
                        xlabel="Total Cryptocurrency Coins Mined",
                        ylabel="Total Cryptocurrency Coin Supply",
                        )
```

Out[77]:



In [ ]:

```
# By Emmanuel Martinez
```