Clustering Crypto

You and Martha have done your research. You understand what unsupervised learning is used for, how to process data, how to cluster, how to reduce your dimensions, and how to reduce the principal components using PCA. It's time to put all these skills to use by creating an analysis for your clients who are preparing to get into the cryptocurrency market.

Martha is a senior manager for the Advisory Services Team at Accountability Accounting, one of your most important clients. Accountability Accounting, a prominent investment bank, is interested in offering a new cryptocurrency investment portfolio for its customers. The company, however, is lost in the vast universe of cryptocurrencies. So, they've asked you to create a report that includes what cryptocurrencies are on the trading market and how they could be grouped to create a classification system for this new investment.

The data Martha will be working with is not ideal, so it will need to be processed to fit the machine learning models. Since there is no known output for what Martha is looking for, she has decided to use unsupervised learning. To group the cryptocurrencies, Martha decided on a clustering algorithm. She'll use data visualizations to share her findings with the board.

- Deliverable 1: Preprocessing the Data for PCA
- Deliverable 2: Reducing Data Dimensions Using PCA
- Deliverable 3: Clustering Cryptocurrencies Using K-means
- Deliverable 4: Visualizing Cryptocurrencies Results

In [1]:

```
# Initial imports
import pandas as pd
import hvplot.pandas
from matplotlib import pyplot as plt
from path import Path
import plotly.express as px
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
```

Deliverable 1: Preprocessing the Data for PCA

In [34]:

```
# Load the crypto_data.csv dataset.
#df = pd.read_csv('crypto_data_GP.csv')
df = pd.read_csv('crypto_data_GP.csv', index_col=0)
df
```

Out[34]:

	CoinName	Algorithm	IsTrading	ProofType	TotalCoinsMined	TotalCoinSupply
42	42 Coin	Scrypt	True	PoW/PoS	4.199995e+01	42

	CoinName	Algorithm	IsTrading	ProofType	TotalCoinsMined	TotalCoinSupply
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]:

```
# Keep all the cryptocurrencies that are being traded.
df_trading = df.loc[df['IsTrading'] == True]
df_trading.tail(10)
```

Out[3]:

	CoinName	Algorithm	IsTrading	ProofType	TotalCoinsMined	TotalCoinSupply
ZEPH	ZEPHYR	SHA-256	True	DPoS	2.000000e+09	2000000000
XQN	Quotient	Scrypt	True	PoW/PoS	NaN	0
NETC	NetworkCoin	X13	True	PoW/PoS	NaN	400000
VPRC	VapersCoin	Scrypt	True	PoW	NaN	42750000000
GAP	Gapcoin	Scrypt	True	PoW/PoS	1.493105e+07	250000000
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

In [4]:

```
# Remove the "IsTrading" column.
df_omit_trade = df_trading.drop(columns=["IsTrading"])
#another way: df_trading.drop(['IsTrading'], axis='columns')

df_omit_trade.tail(10)
```

Out[4]:

	CoinName	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply
ZEPH	ZEPHYR	SHA-256	DPoS	2.000000e+09	2000000000
XQN	Quotient	Scrypt	PoW/PoS	NaN	0

	CoinName	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply
NETC	NetworkCoin	X13	PoW/PoS	NaN	400000
VPRC	VapersCoin	Scrypt	PoW	NaN	42750000000
GAP	Gapcoin	Scrypt	PoW/PoS	1.493105e+07	250000000
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

In [5]:

```
# Remove rows that have at least 1 null value.

# A. Find null values
for column in df_omit_trade.columns:
    print(f"Column {column} has {df_omit_trade[column].isnull().sum()} null values")
```

```
Column CoinName has 0 null values
Column Algorithm has 0 null values
Column ProofType has 0 null values
Column TotalCoinsMined has 459 null values
Column TotalCoinSupply has 0 null values
```

In [6]:

```
# B. Drop the null-value rows
df_dropnull = df_omit_trade.dropna(axis=0)      # "0" for rows

df_dropnull
```

Out[6]:

	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 [7]:
```
count = (df_dropnull['TotalCoinsMined'] == 0).sum()
print('Count of zeros in Column TotalCoinsMined : ', count)
```

```
Count of zeros in Column TotalCoinsMined :  152
```

In [8]:
```
# Keep the rows where coins are mined.
df_mined = df_dropnull.loc[df_dropnull['TotalCoinsMined'] != 0]

df_mined
```

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
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
...	...	...	...	...	...
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

533 rows × 5 columns

In [9]:
```
# See if it worked
count = (df_mined['TotalCoinsMined'] == 0).sum()
print('Count of zeros in Column TotalCoinsMined : ', count)
```

```
Count of zeros in Column TotalCoinsMined :  0
```

In [10]:
```
# Create a new DataFrame that holds only the cryptocurrency names, and
#   use the crypto_df DataFrame index as the index for this new DataFrame.

# Best Option (A):

# A1. Create new df w/ new index:
#    Go back tot where we red-in the csv and add `, index_col=0` fater the file name

# A2. df holds only cryptocurrencies names:
crypto_df = df_mined[['CoinName']]
crypto_df
```

Out[10]:

	CoinName
42	42 Coin
404	404Coin

	CoinName
1337	EliteCoin
BTC	Bitcoin
ETH	Ethereum
...	...
ZEPH	ZEPHYR
GAP	Gapcoin
BDX	Beldex
ZEN	Horizen
XBC	BitcoinPlus

533 rows × 1 columns

In [11]:

```
# # Alternate Option (B): (won't keep the changed index for the original/top df, just t

# # B1. Create new df w/ new index:
# coin_dfB1 = df_mined.set_index(['Unnamed: 0'])
# coin_dfB1.head()
```

In [12]:

```
# # B2. Holds only cryptocurrencies names.
# coin_dfB2 = coin_dfB1[['CoinName']]
# coin_dfB2.head()
```

In [13]:

```
# # B3. Holds only cryptocurrencies names.
# coin_dfB3 = coin_dfB2.rename_axis(None)
# coin_dfB3.head()
```

In [14]:

```
# Drop the 'CoinName' column since it's not going to be used on the clustering algorith
coin_drop_df = df_mined.drop(columns=["CoinName"])

#another way: df_mined.drop(['CoinName'], axis='columns')

coin_drop_df
```

Out[14]:

	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
...	...	...	...	...

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply
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

533 rows × 4 columns

In [15]:

```
# Use get_dummies() to create variables for text features , `Algorithm` and `ProofType`
#   and store the resulting data in a new DataFrame named X.
X = pd.get_dummies(coin_drop_df, columns=['Algorithm', 'ProofType'])
X.head(10)
```

Out[15]:

	TotalCoinsMined	TotalCoinSupply	Algorithm_1GB AES Pattern Search	Algorithm_536	Algorithm_Argon2d	Algorithr
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	
LTC	6.303924e+07	84000000	0	0	0	
DASH	9.031294e+06	22000000	0	0	0	
XMR	1.720114e+07	0	0	0	0	
ETC	1.133597e+08	210000000	0	0	0	
ZEC	7.383056e+06	21000000	0	0	0	

10 rows × 100 columns

In [16]:

```
# Standardize features from the X df using StandardScaler() function + it's fit_transfo
X_scaled = StandardScaler().fit_transform(X)
print(X_scaled[0:5])
```

```
[[-0.11674788 -0.15286468 -0.0433555  -0.0433555  -0.0433555  -0.06137164
  -0.07523548 -0.0433555  -0.06137164 -0.06137164 -0.0433555  -0.0433555
  -0.19226279 -0.06137164 -0.09731237 -0.0433555  -0.11536024 -0.07523548
  -0.0433555  -0.0433555  -0.15176505 -0.0433555  -0.13105561 -0.0433555
  -0.0433555  -0.08695652 -0.0433555  -0.0433555  -0.0433555  -0.0433555
  -0.06137164 -0.0433555  -0.08695652 -0.08695652 -0.08695652 -0.0433555
  -0.13105561 -0.13827675 -0.13827675 -0.0433555  -0.06137164 -0.0433555
  -0.07523548 -0.1815096  -0.0433555  -0.0433555  -0.0433555  -0.07523548
  -0.15811388 -0.3145935  -0.0433555  -0.08695652 -0.07523548 -0.06137164
```

```
 -0.0433555   1.38873015 -0.0433555  -0.0433555  -0.06137164 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.39836623 -0.0433555  -0.1815096  -0.0433555  -0.08695652
 -0.08695652 -0.10670145 -0.0433555  -0.0433555  -0.13105561 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.07523548 -0.4386271  -0.0433555
 -0.06137164 -0.0433555  -0.0433555  -0.89480483 -0.0433555  -0.0433555
  1.42422228 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555 ]
[-0.09358885 -0.14499604 -0.0433555  -0.0433555  -0.0433555  -0.06137164
 -0.07523548 -0.0433555  -0.06137164 -0.06137164 -0.0433555  -0.0433555
 -0.19226279 -0.06137164 -0.09731237 -0.0433555  -0.11536024 -0.07523548
 -0.0433555  -0.0433555  -0.15176505 -0.0433555  -0.13105561 -0.0433555
 -0.0433555  -0.08695652 -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.06137164 -0.0433555  -0.08695652 -0.08695652 -0.08695652 -0.0433555
 -0.13105561 -0.13827675 -0.13827675 -0.0433555  -0.06137164 -0.0433555
 -0.07523548 -0.1815096  -0.0433555  -0.0433555  -0.0433555  -0.07523548
 -0.15811388 -0.3145935  -0.0433555  -0.08695652 -0.07523548 -0.06137164
 -0.0433555   1.38873015 -0.0433555  -0.0433555  -0.06137164 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.39836623 -0.0433555  -0.1815096  -0.0433555  -0.08695652
 -0.08695652 -0.10670145 -0.0433555  -0.0433555  -0.13105561 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.07523548 -0.4386271  -0.0433555
 -0.06137164 -0.0433555  -0.0433555  -0.89480483 -0.0433555  -0.0433555
  1.42422228 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555 ]
[ 0.52587231  4.4937636  -0.0433555  -0.0433555  -0.0433555  -0.06137164
 -0.07523548 -0.0433555  -0.06137164 -0.06137164 -0.0433555  -0.0433555
 -0.19226279 -0.06137164 -0.09731237 -0.0433555  -0.11536024 -0.07523548
 -0.0433555  -0.0433555  -0.15176505 -0.0433555  -0.13105561 -0.0433555
 -0.0433555  -0.08695652 -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.06137164 -0.0433555  -0.08695652 -0.08695652 -0.08695652 -0.0433555
 -0.13105561 -0.13827675 -0.13827675 -0.0433555  -0.06137164 -0.0433555
 -0.07523548 -0.1815096  -0.0433555  -0.0433555  -0.0433555  -0.07523548
 -0.15811388 -0.3145935  -0.0433555  -0.08695652 -0.07523548 -0.06137164
 -0.0433555  -0.7200823  -0.0433555  -0.0433555  -0.06137164 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.39836623 -0.0433555   5.50935034 -0.0433555  -0.08695652
 -0.08695652 -0.10670145 -0.0433555  -0.0433555  -0.13105561 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.07523548 -0.4386271  -0.0433555
 -0.06137164 -0.0433555  -0.0433555  -0.89480483 -0.0433555  -0.0433555
  1.42422228 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555 ]
[-0.11635442 -0.15255408 -0.0433555  -0.0433555  -0.0433555  -0.06137164
 -0.07523548 -0.0433555  -0.06137164 -0.06137164 -0.0433555  -0.0433555
 -0.19226279 -0.06137164 -0.09731237 -0.0433555  -0.11536024 -0.07523548
 -0.0433555  -0.0433555  -0.15176505 -0.0433555  -0.13105561 -0.0433555
 -0.0433555  -0.08695652 -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.06137164 -0.0433555  -0.08695652 -0.08695652 -0.08695652 -0.0433555
 -0.13105561 -0.13827675 -0.13827675 -0.0433555  -0.06137164 -0.0433555
 -0.07523548 -0.1815096  -0.0433555  -0.0433555  -0.0433555  -0.07523548
 -0.15811388  3.17870519 -0.0433555  -0.08695652 -0.07523548 -0.06137164
 -0.0433555  -0.7200823  -0.0433555  -0.0433555  -0.06137164 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.39836623 -0.0433555  -0.1815096  -0.0433555  -0.08695652
 -0.08695652 -0.10670145 -0.0433555  -0.0433555  -0.13105561 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.07523548 -0.4386271  -0.0433555
 -0.06137164 -0.0433555  -0.0433555   1.11756214 -0.0433555  -0.0433555
 -0.70213759 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555 ]
[-0.11438445 -0.15286468 -0.0433555  -0.0433555  -0.0433555  -0.06137164
```

```
 -0.07523548 -0.0433555  -0.06137164 -0.06137164 -0.0433555  -0.0433555
 -0.19226279 -0.06137164 -0.09731237 -0.0433555  -0.11536024 -0.07523548
 -0.0433555  -0.0433555  -0.15176505 -0.0433555   7.63034876 -0.0433555
 -0.0433555  -0.08695652 -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.06137164 -0.0433555  -0.08695652 -0.08695652 -0.08695652 -0.0433555
 -0.13105561 -0.13827675 -0.13827675 -0.0433555  -0.06137164 -0.0433555
 -0.07523548 -0.1815096  -0.0433555  -0.0433555  -0.0433555  -0.07523548
 -0.15811388 -0.3145935  -0.0433555  -0.08695652 -0.07523548 -0.06137164
 -0.0433555  -0.7200823  -0.0433555  -0.0433555  -0.06137164 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.39836623 -0.0433555  -0.1815096  -0.0433555  -0.08695652
 -0.08695652 -0.10670145 -0.0433555  -0.0433555  -0.13105561 -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.07523548 -0.4386271  -0.0433555
 -0.06137164 -0.0433555  -0.0433555   1.11756214 -0.0433555  -0.0433555
 -0.70213759 -0.0433555  -0.0433555  -0.0433555  -0.0433555  -0.0433555
 -0.0433555  -0.0433555  -0.0433555  -0.0433555 ]]
```

Deliverable 2: Reducing Data Dimensions of X df Using PCA

In [17]:

```python
# Using PCA to reduce dimension to 3 principal components.

# Initialize PCA model
pca = PCA(n_components=3)

# Get 3 principal components for the X_scaled data.
X_scaled_pca = pca.fit_transform(X_scaled)
```

In [18]:

```python
# Create a new DataFrame named `pcs_df` that has 3 PC's: PC 1, PC 2, PC 3
#   and uses the index of the `crypto_df` DataFrame as the index.
pcs_df = pd.DataFrame(
    data=X_scaled_pca, columns=["PC1", "PC2", "PC3"], index = crypto_df.index
)

pcs_df.head()
```

Out[18]:

	PC1	PC2	PC3
42	-0.349434	1.043618	-0.625013
404	-0.332797	1.043956	-0.625464
1337	2.302559	1.722744	-0.745259
BTC	-0.141931	-1.335313	0.145641
ETH	-0.149956	-2.042581	0.456143

Deliverable 3: Clustering Crytocurrencies Using K-Means

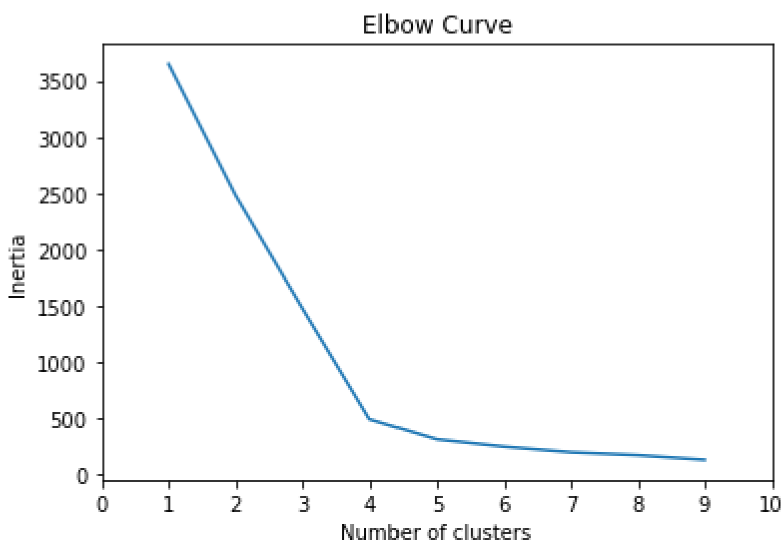
Finding the Best Value for `k` Using the Elbow Curve

In [19]:

```python
# Create an elbow curve for the pcs_df to find the best value for K.

# Finding the best value for k
inertia = []
k = list(range(1, 10))
```

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

# Creating the Elbow Curve
elbow_data = {"k": k, "inertia": inertia}
df_elbow = pd.DataFrame(elbow_data)

plt.plot(df_elbow['k'], df_elbow['inertia'])
plt.xticks(list(range(11)))
plt.title('Elbow Curve')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

```
C:\Users\c-hol\anaconda3\envs\mleny\lib\site-packages\sklearn\cluster\_kmeans.py:882: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=3.
  f"KMeans is known to have a memory leak on Windows "
```



Running K-Means with k=4

In [20]:

```
# Initialize the k-means model
model = KMeans(n_clusters=4, random_state=0)

# Fit the model
model.fit(pcs_df)                           # pass in the pca_df

# Predict clusters
predictions = model.predict(pcs_df)         # pass in the pca_df

#  Add a new column, "Class" to the clustered_df DataFrame that holds the predictions.
pcs_df["Class"] = model.labels_
pcs_df.head()
```

Out[20]:

	PC1	PC2	PC3	Class

	PC1	PC2	PC3	Class
42	-0.349434	1.043618	-0.625013	0
404	-0.332797	1.043956	-0.625464	0
1337	2.302559	1.722744	-0.745259	0
BTC	-0.141931	-1.335313	0.145641	1
ETH	-0.149956	-2.042581	0.456143	1

In [21]:

```
# Create a new DataFrame named clustered_df by concatenating the crypto_df and pcs_df d
#   on the same columns. The index should be the same as the crypto_df DataFrame.
clustered_df = pd.concat([coin_drop_df, pcs_df], axis=1)
clustered_df.head()
```

Out[21]:

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC1	PC2	PC3	Class
42	Scrypt	PoW/PoS	4.199995e+01	42	-0.349434	1.043618	-0.625013	0
404	Scrypt	PoW/PoS	1.055185e+09	532000000	-0.332797	1.043956	-0.625464	0
1337	X13	PoW/PoS	2.927942e+10	314159265359	2.302559	1.722744	-0.745259	0
BTC	SHA-256	PoW	1.792718e+07	21000000	-0.141931	-1.335313	0.145641	1
ETH	Ethash	PoW	1.076842e+08	0	-0.149956	-2.042581	0.456143	1

In [22]:

```
#  Add a new column, "CoinName" to the clustered_df DataFrame that holds the names of t
clustered_df["CoinName"] = crypto_df["CoinName"]

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

```
(533, 9)
```

In [23]:

```
clustered_df.head(10)
```

Out[23]:

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC1	PC2	PC3	Clas
42	Scrypt	PoW/PoS	4.199995e+01	42	-0.349434	1.043618	-0.625013	
404	Scrypt	PoW/PoS	1.055185e+09	532000000	-0.332797	1.043956	-0.625464	
1337	X13	PoW/PoS	2.927942e+10	314159265359	2.302559	1.722744	-0.745259	
BTC	SHA-256	PoW	1.792718e+07	21000000	-0.141931	-1.335313	0.145641	
ETH	Ethash	PoW	1.076842e+08	0	-0.149956	-2.042581	0.456143	
LTC	Scrypt	PoW	6.303924e+07	84000000	-0.174122	-1.125077	-0.008066	
DASH	X11	PoW/PoS	9.031294e+06	22000000	-0.395166	1.242682	-0.526264	
XMR	CryptoNight-V7	PoW	1.720114e+07	0	-0.148035	-2.278438	0.460726	

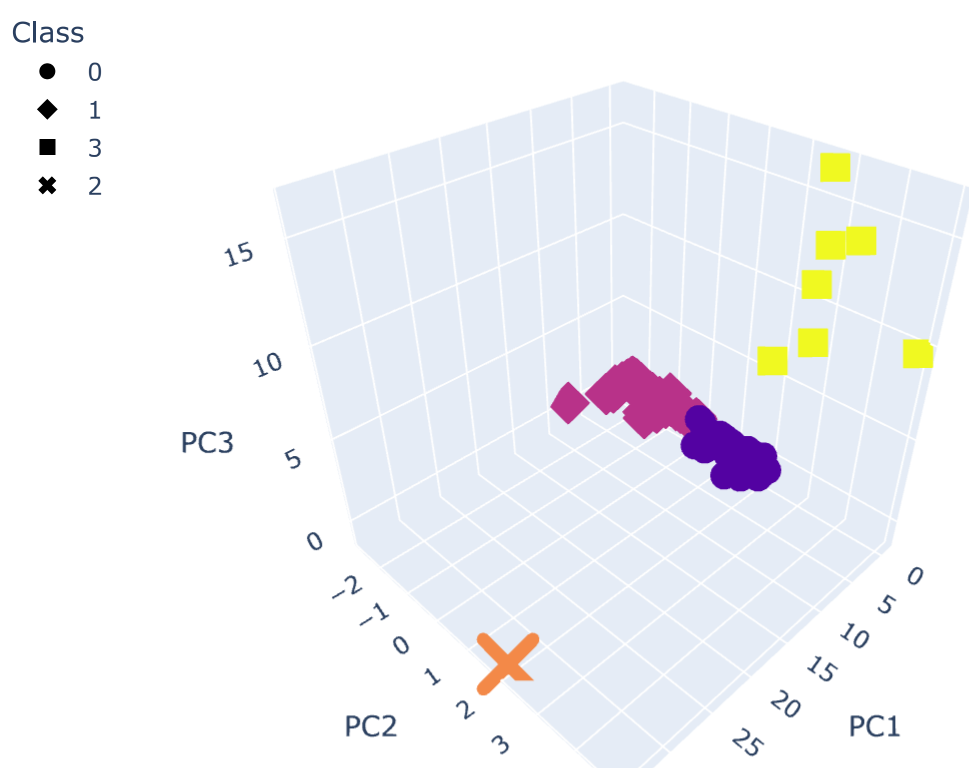
	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC1	PC2	PC3	Clas
ETC	Ethash	PoW	1.133597e+08	210000000	-0.148402	-2.042661	0.456124	
ZEC	Equihash	PoW	7.383056e+06	21000000	-0.128730	-1.946880	0.519222	

Deliverable 4: Visualizing Cryptocurrencies Results

3D-Scatter with Clusters

In [24]:

```
# Create a 3D scatter plot using the Plotly Express scatter_3d() function
#    to plot the three clusters from the clustered_df DataFrame
fig = px.scatter_3d(
    clustered_df,
    x="PC1",
    y="PC2",
    z="PC3",
    color="Class",
    symbol="Class",
    width=800,
    hover_name="CoinName",
    hover_data=["Algorithm"],
)
fig.update_layout(legend=dict(x=0, y=1))
fig.show()
```



In [25]:

```
# Create a table with tradable cryptocurrencies using the hvplot.table() function
clustered_df.hvplot.table(columns=['CoinName','Algorithm','ProofType','TotalCoinSupply'
```

Out[25]:

#	CoinName	Algorithm	ProofType	TotalCoinSupply	TotalCoinsMined	Class
0	42 Coin	Scrypt	PoW/PoS	42	41.999954	0
1	404Coin	Scrypt	PoW/PoS	532000000	1,055,184,902.04	0
2	EliteCoin	X13	PoW/PoS	314159265359	29,279,424,622.5027	0
3	Bitcoin	SHA-256	PoW	21000000	17,927,175.0	1
4	Ethereum	Ethash	PoW	0	107,684,222.6865	1
5	Litecoin	Scrypt	PoW	84000000	63,039,243.300005	1
6	Dash	X11	PoW/PoS	22000000	9,031,294.375634	0
7	Monero	CryptoNight-V7	PoW	0	17,201,143.144913	1
8	Ethereum Classic	Ethash	PoW	210000000	113,359,703.0	1
9	ZCash	Equihash	PoW	21000000	7,383,056.25	1
10	Bitshares	SHA-512	PoS	3600570502	2,741,570,000.0	0

In [26]:

```
# Print the total number of tradable cryptocurrencies.
print('There are', clustered_df['CoinName'].count(), 'tradable cryptocurrencies')
```

```
There are 533 tradable cryptocurrencies
```

In [27]:

```
# Scaling data to create the scatter plot with tradable cryptocurrencies.
X_scaled = MinMaxScaler().fit_transform(clustered_df[['TotalCoinSupply','TotalCoinsMine
X_scaled
```

Out[27]:

```
array([[4.20000000e-11, 5.94230127e-03],
       [5.32000000e-04, 7.00182308e-03],
       [3.14159265e-01, 3.53420682e-02],
       ...,
       [1.40022261e-03, 6.92655266e-03],
       [2.10000000e-05, 5.94962775e-03],
       [1.00000000e-06, 5.94243008e-03]])
```

In [28]:

```
# Create a new DataFrame that has the scaled data with the clustered_df DataFrame index
# YOUR CODE HERE

scaled_clustered_df = pd.DataFrame(
    X_scaled,columns=['TotalCoinSupply','TotalCoinsMined'], index=clustered_df.index)

scaled_clustered_df.head()
```

Out[28]:

	TotalCoinSupply	TotalCoinsMined
42	4.200000e-11	0.005942
404	5.320000e-04	0.007002
1337	3.141593e-01	0.035342
BTC	2.100000e-05	0.005960
ETH	0.000000e+00	0.006050

In [29]:

```
# Add the "CoinName" column from the clustered_df DataFrame to the new DataFrame.
scaled_clustered_df['CoinName'] = clustered_df['CoinName']

scaled_clustered_df.head()
```

Out[29]:

	TotalCoinSupply	TotalCoinsMined	CoinName
42	4.200000e-11	0.005942	42 Coin
404	5.320000e-04	0.007002	404Coin
1337	3.141593e-01	0.035342	EliteCoin
BTC	2.100000e-05	0.005960	Bitcoin
ETH	0.000000e+00	0.006050	Ethereum

In [33]:

```
# Add the "Class" column from the clustered_df DataFrame to the new DataFrame.
scaled_clustered_df['Class'] = clustered_df['Class']

scaled_clustered_df
```

Out[33]:

	TotalCoinSupply	TotalCoinsMined	CoinName	Class
42	4.200000e-11	0.005942	42 Coin	0
404	5.320000e-04	0.007002	404Coin	0
1337	3.141593e-01	0.035342	EliteCoin	0
BTC	2.100000e-05	0.005960	Bitcoin	1
ETH	0.000000e+00	0.006050	Ethereum	1
...	...	...	...	...
ZEPH	2.000000e-03	0.007951	ZEPHYR	0
GAP	2.500000e-04	0.005957	Gapcoin	0
BDX	1.400223e-03	0.006927	Beldex	1
ZEN	2.100000e-05	0.005950	Horizen	1
XBC	1.000000e-06	0.005942	BitcoinPlus	0

533 rows × 4 columns

In [31]:
```python
# Create a hvplot.scatter plot using x="TotalCoinsMined" and y="TotalCoinSupply".
# http://holoviews.org/user_guide/Style_Mapping.html
hvplot.help('line')
```

```
Line plot

Parameters
----------
x, y : string, optional
    Field name to draw x- and y-positions from
**kwds : optional
    Keyword arguments to pass on to
    :py:meth:`hvplot.converter.HoloViewsConverter`.
Returns
-------
HoloViews object: Object representing the requested visualization


Generic options
---------------
clim: tuple
    Lower and upper bound of the color scale
cnorm (default='linear'): str
    Color scaling which must be one of 'linear', 'log' or 'eq_hist'
colorbar (default=False): boolean
    Enables a colorbar
fontscale: number
    Scales the size of all fonts by the same amount, e.g. fontscale=1.5
    enlarges all fonts (title, xticks, labels etc.) by 50%
fontsize: number or dict
    Set title, label and legend text to the same fontsize. Finer control
    by using a dict: {'title': '15pt', 'ylabel': '5px', 'ticks': 20}
flip_xaxis/flip_yaxis: boolean
    Whether to flip the axis left to right or up and down respectively
grid (default=False): boolean
    Whether to show a grid
hover : boolean
    Whether to show hover tooltips, default is True unless datashade is
    True in which case hover is False by default
hover_cols (default=[]): list or str
    Additional columns to add to the hover tool or 'all' which will
    includes all columns (including indexes if use_index is True).
invert (default=False): boolean
    Swaps x- and y-axis
frame_width/frame_height: int
    The width and height of the data area of the plot
legend (default=True): boolean or str
    Whether to show a legend, or a legend position
    ('top', 'bottom', 'left', 'right')
logx/logy (default=False): boolean
    Enables logarithmic x- and y-axis respectively
logz (default=False): boolean
    Enables logarithmic colormapping
loglog (default=False): boolean
    Enables logarithmic x- and y-axis
max_width/max_height: int
    The maximum width and height of the plot for responsive modes
min_width/min_height: int
    The minimum width and height of the plot for responsive modes
padding: number or tuple
```

```
    Fraction by which to increase auto-ranged extents to make
    datapoints more visible around borders. Supports tuples to
    specify different amount of padding for x- and y-axis and
    tuples of tuples to specify different amounts of padding for
    upper and lower bounds.
responsive: boolean
    Whether the plot should responsively resize depending on the
    size of the browser. Responsive mode will only work if at
    least one dimension of the plot is left undefined, e.g. when
    width and height or width and aspect are set the plot is set
    to a fixed size, ignoring any responsive option.
rot: number
    Rotates the axis ticks along the x-axis by the specified
    number of degrees.
shared_axes (default=True): boolean
    Whether to link axes between plots
transforms (default={}): dict
    A dictionary of HoloViews dim transforms to apply before plotting
title (default=''): str
    Title for the plot
tools (default=[]): list
    List of tool instances or strings (e.g. ['tap', box_select'])
xaxis/yaxis: str or None
    Whether to show the x/y-axis and whether to place it at the
    'top'/'bottom' and 'left'/'right' respectively.
xformatter/yformatter (default=None): str or TickFormatter
    Formatter for the x-axis and y-axis (accepts printf formatter,
    e.g. '%.3f', and bokeh TickFormatter)
xlabel/ylabel/clabel (default=None): str
    Axis labels for the x-axis, y-axis, and colorbar
xlim/ylim (default=None): tuple or list
    Plot limits of the x- and y-axis
xticks/yticks (default=None): int or list
    Ticks along x- and y-axis specified as an integer, list of
    ticks positions, or list of tuples of the tick positions and labels
width (default=700)/height (default=300): int
    The width and height of the plot in pixels
attr_labels (default=None): bool
    Whether to use an xarray object's attributes as labels, defaults to
    None to allow best effort without throwing a warning. Set to True
    to see warning if the attrs can't be found, set to False to disable
    the behavior.
sort_date (default=True): bool
    Whether to sort the x-axis by date before plotting
symmetric (default=None): bool
    Whether the data are symmetric around zero. If left unset, the data
    will be checked for symmetry as long as the size is less than
    ``check_symmetric_max``.
check_symmetric_max (default=1000000):
    Size above which to stop checking for symmetry by default on the data.

Datashader options
------------------
aggregator (default=None):
    Aggregator to use when applying rasterize or datashade operation
    (valid options include 'mean', 'count', 'min', 'max' and more, and
    datashader reduction objects)
dynamic (default=True):
    Whether to return a dynamic plot which sends updates on widget and
    zoom/pan events or whether all the data should be embedded
```

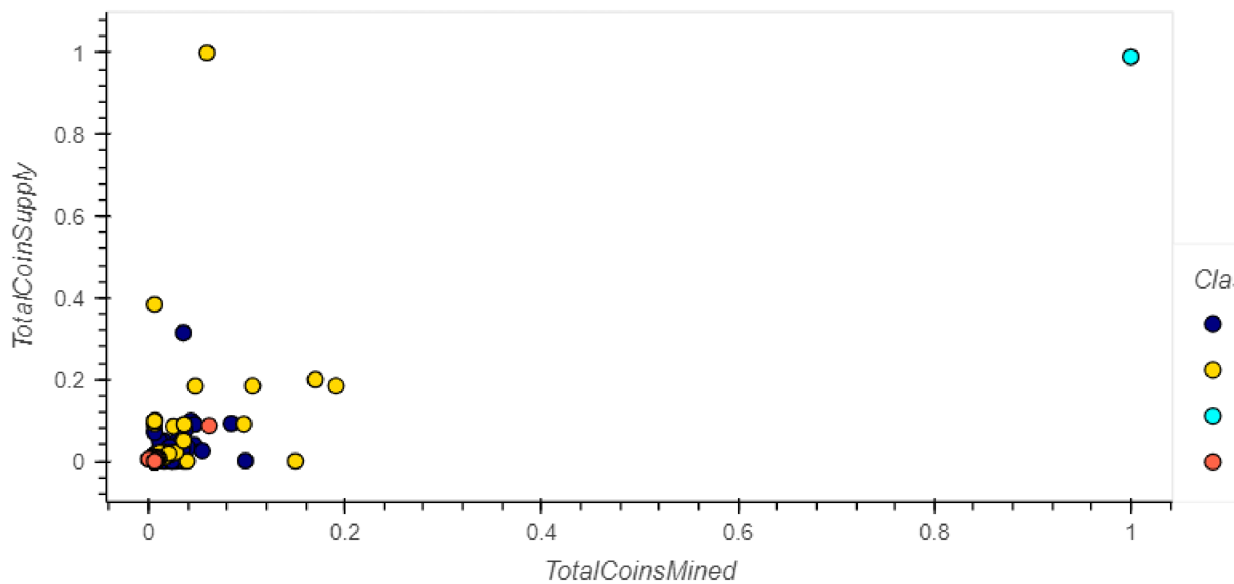
```
    (warning: for large groupby operations embedded data can become
    very large if dynamic=False)
datashade (default=False):
    Whether to apply rasterization and shading using datashader
    library returning an RGB object
dynspread (default=False):
    Allows plots generated with datashade=True or rasterize=True
    to increase the point size to make sparse regions more visible
rasterize (default=False):
    Whether to apply rasterization using the datashader library
    returning an aggregated Image
x_sampling/y_sampling (default=None):
    Specifies the smallest allowed sampling interval along the x/y axis.

Geographic options
------------------
coastline (default=False):
    Whether to display a coastline on top of the plot, setting
    coastline='10m'/'50m'/'110m' specifies a specific scale.
crs (default=None):
    Coordinate reference system of the data specified as Cartopy
    CRS object, proj.4 string or EPSG code.
features (default=None): dict or list
    A list of features or a dictionary of features and the scale
    at which to render it. Available features include 'borders',
    'coastline', 'lakes', 'land', 'ocean', 'rivers' and 'states'.
    Available scales include '10m'/'50m'/'110m'.
geo (default=False):
    Whether the plot should be treated as geographic (and assume
    PlateCarree, i.e. lat/lon coordinates).
global_extent (default=False):
    Whether to expand the plot extent to span the whole globe.
project (default=False):
    Whether to project the data before plotting (adds initial
    overhead but avoids projecting data when plot is dynamically
    updated).
tiles (default=False):
    Whether to overlay the plot on a tile source. Tiles sources
    can be selected by name or a tiles object or class can be passed,
    the default is 'Wikipedia'.

Style options
-------------

alpha
color
hover_alpha
hover_color
hover_line_alpha
hover_line_color
line_alpha
line_cap
line_color
line_dash
line_join
line_width
muted
muted_alpha
muted_color
muted_line_alpha
```

```
muted_line_color
nonselection_alpha
nonselection_color
nonselection_line_alpha
nonselection_line_color
selection_alpha
selection_color
selection_line_alpha
selection_line_color
visible
```

In [32]:

```
scaled_clustered_df.hvplot(
    x='TotalCoinsMined',
    y='TotalCoinSupply',
    kind='scatter',
    by="Class",
    size=60,
    line_color='black',
    color=scaled_clustered_df["Class"].map({0: "navy", 1: "gold", 2: "aqua",3: "tomato"
)
```

Out[32]:



In [ ]: