Crypto_Clustering-Model19-screenshots

Module 19 Challenge – CryptoClustering
SMU DS – Raj Agrawal
Submitted on: 03-SEP-2023
Repository – https://github.com/RajAgrawal99/SMU_DS_Bootcamp_March2023_RA.git
Folder – CryptoClustering-Challenge

Data source - crypto_market_data.csv

```
# Create a DataFrame with the scaled data
num_cols = ['price_change_percentage_24h', 'price_change_percentage_7d',
       'price_change_percentage_14d', 'price_change_percentage_30d',
       'price_change_percentage_60d', 'price_change_percentage_200d',
       'price_change_percentage_1y']
df_num = df.loc[:, num_cols]
scaler = StandardScaler().fit(df_num)

df_scale = scaler.transform(df_num)
df_scale = pd.DataFrame(df_scale, columns=num_cols, index=df.index)
df_scale.head()
```

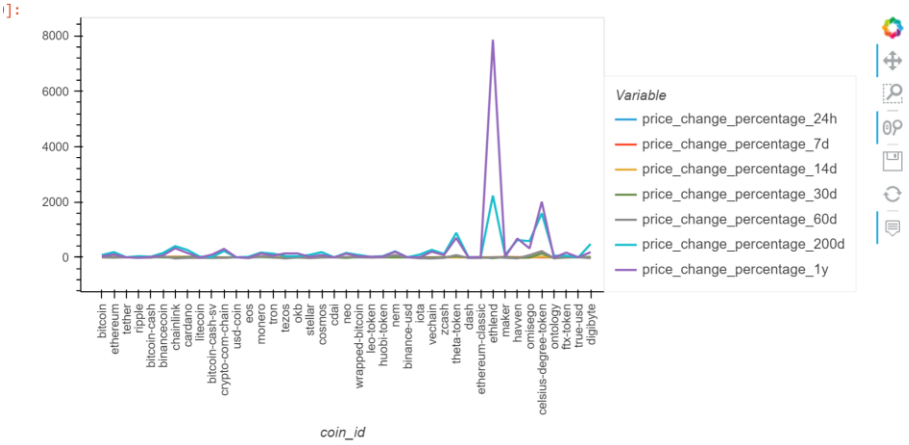
coin_id	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d	price_change_percentage_30d	price_change_percentage
bitcoin	0.508529	0.493193	0.772200	0.235460	-0.0
ethereum	0.185446	0.934445	0.558692	-0.054341	-0.2
tether	0.021774	-0.706337	-0.021680	-0.061030	0.0
ripple	-0.040764	-0.810928	0.249458	-0.050388	-0.3
bitcoin-cash	1.193036	2.000959	1.760610	0.545842	-0.2

```
# Generate summary statistics
df.describe()
```

	price_change_percentage_24h	price_change_percentage_7d	price_change_percentage_14d	price_change_percentage_30d	price_change_percentage_60
count	41.000000	41.000000	41.000000	41.000000	41.0000
mean	-0.269686	4.497147	0.185787	1.545693	-0.0941
std	2.694793	6.375218	8.376939	26.344218	47.3658
min	-13.527860	-6.094560	-18.158900	-34.705480	-44.8224
25%	-0.608970	0.047260	-5.026620	-10.438470	-25.9079
50%	-0.063410	3.296410	0.109740	-0.042370	-7.5445
75%	0.612090	7.602780	5.510740	4.578130	0.6572
max	4.840330	20.694590	24.239190	140.795700	223.0643

```
# Plotting data to see what's in the DataFrame
df.hvplot.line(
    width=800,
    height=400,
    rot=90
)
```

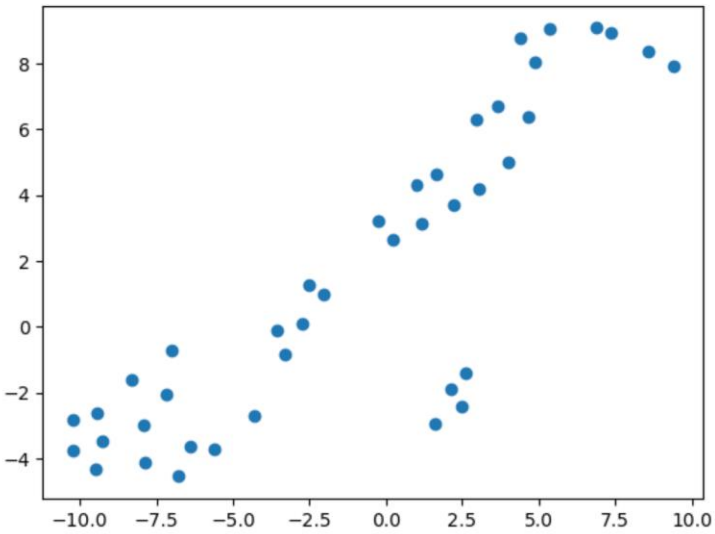
]:



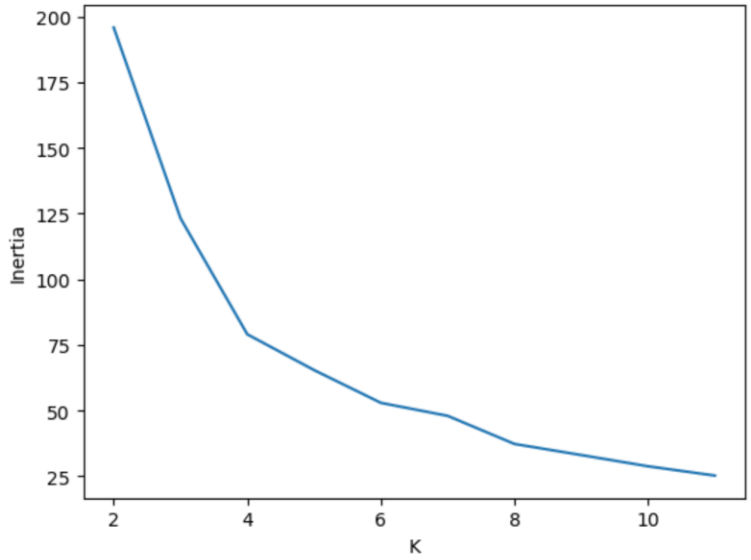
TSNE plots

```
# TSNE plots
# (UNSCALED)
X = df_num.to_numpy()
X_embedded = TSNE(perplexity=10).fit_transform(X)

plt.scatter(X_embedded[:, 0], X_embedded[:, 1])
plt.show()
```



```
plt.plot(df_elbow.k, df_elbow.inertia)
plt.xlabel("K")
plt.ylabel("Inertia")
plt.show()
```




```
# Make TSNE for 3
model = KMeans(n_clusters=3, random_state=1)
# Fit the model
model.fit(df_scale)
```

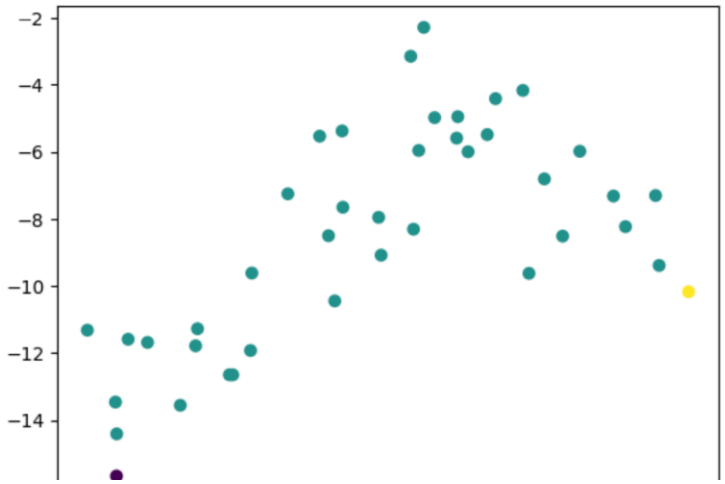
KMeans

```
KMeans(n_clusters=3, random_state=1)
```

```
# Make predictions
preds = model.predict(df_scale)

X = df_scale.to_numpy()
X_embedded = TSNE(perplexity=10).fit_transform(X)

plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=preds)
plt.show()
```



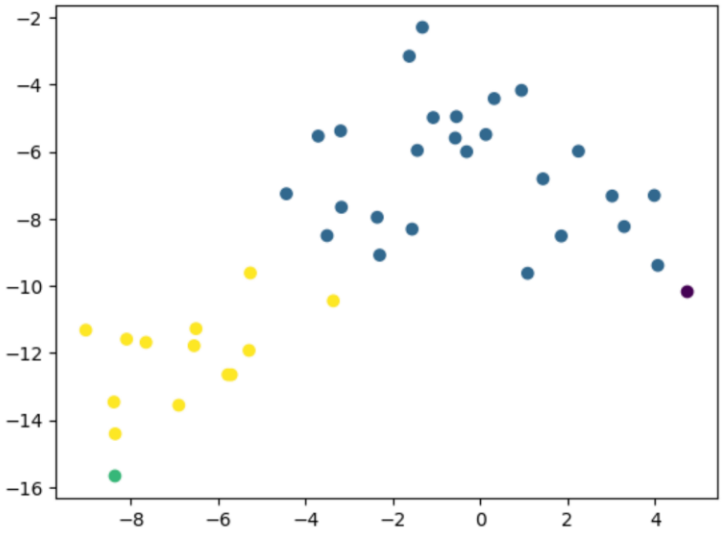
```
# Make TSNE for 4
model = KMeans(n_clusters=4, random_state=1)

# Fit the model
model.fit(df_scale)

# Make predictions
preds = model.predict(df_scale)

X = df_scale.to_numpy()
X_embedded = TSNE(perplexity=10).fit_transform(X)
```

```
plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=preds)
plt.show()
```



Answer the following question:

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

Answer: From the scatter plots, it's a little hard to tell given the variability and quantity of the data (zooming in helps), but it appears that the optimal value for k, the nubmer of clusters, is 4.

```
df_pca = pd.DataFrame(df_pca, columns=[f"PCA_{x}" for x in range(1, n_comps + 1)], index=df.index)
df_pca.head()
```

:

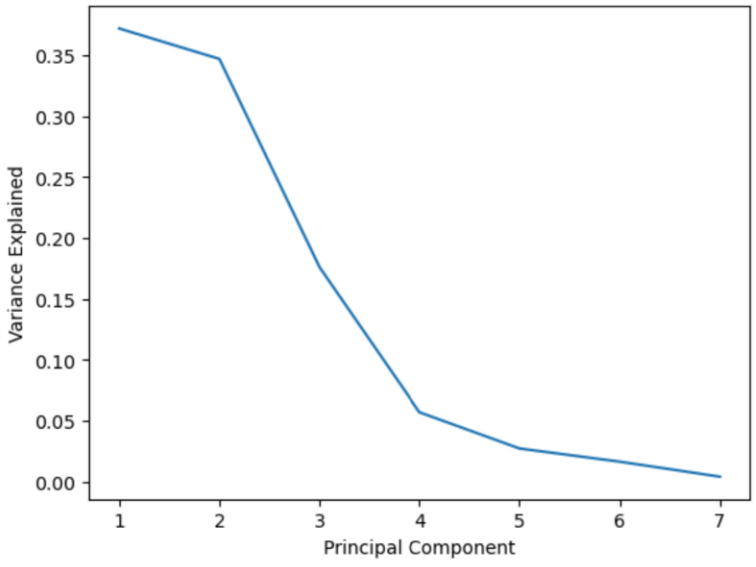
coin_id	PCA_1	PCA_2	PCA_3	PCA_4	PCA_5	PCA_6	PCA_7
bitcoin	-0.600667	0.842760	0.461595	-0.109151	-0.033786	-0.225703	0.006595
ethereum	-0.458261	0.458466	0.952877	0.095100	0.014588	0.034158	0.109593
tether	-0.433070	-0.168126	-0.641752	-0.470282	0.115300	-0.127710	-0.086857
ripple	-0.471835	-0.222660	-0.479053	-0.737473	-0.148641	-0.273472	0.134870
bitcoin-cash	-1.157800	2.041209	1.859715	0.236479	-0.191787	-0.411513	-0.070411

```
df_pca.corr()
```

:

	PCA_1	PCA_2	PCA_3	PCA_4	PCA_5	PCA_6	PCA_7
PCA_1	1.000000e+00	2.599562e-16	1.095974e-16	7.434809e-17	-1.343605e-17	-2.088502e-16	1.230570e-17
PCA_2	2.599562e-16	1.000000e+00	-3.322026e-16	-1.326485e-16	-2.980973e-17	-1.292298e-16	-2.548176e-18
PCA_3	1.095974e-16	-3.322026e-16	1.000000e+00	2.315917e-16	-2.232150e-17	-2.892237e-16	4.650926e-17
PCA_4	7.434809e-17	-1.326485e-16	2.315917e-16	1.000000e+00	-5.881178e-17	-1.199061e-16	1.256828e-17
PCA_5	-1.343605e-17	-2.980973e-17	-2.232150e-17	-5.881178e-17	1.000000e+00	-3.649550e-17	1.817052e-17
PCA_6	-2.088502e-16	-1.292298e-16	-2.892237e-16	-1.199061e-16	-3.649550e-17	1.000000e+00	-1.374611e-16
PCA_7	1.230570e-17	-2.548176e-18	4.650926e-17	1.256828e-17	1.817052e-17	-1.374611e-16	1.000000e+00

```
plt.plot(range(1, n_comps + 1), pca.explained_variance_ratio_)
plt.xlabel("Principal Component")
plt.ylabel("Variance Explained")
plt.show()
```

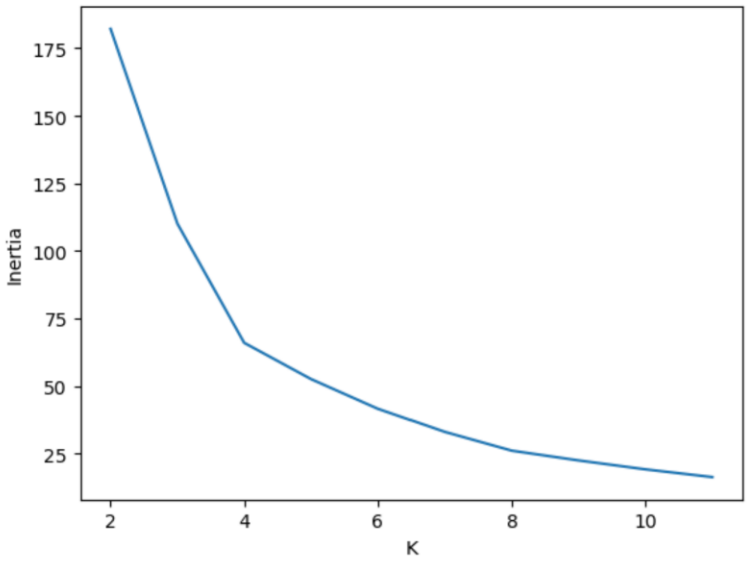


Answer the following question:

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

Answer: 0.9520883911037916

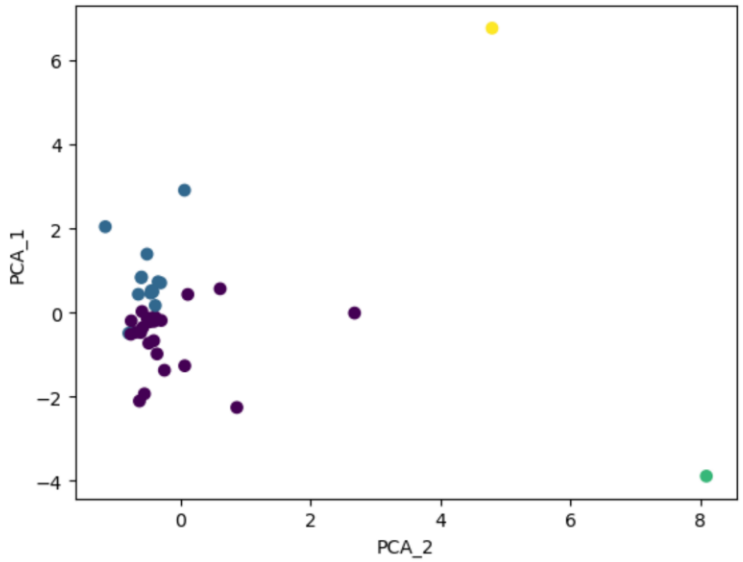
```
plt.plot(df_elbow.k, df_elbow.inertia)
plt.xlabel("K")
plt.ylabel("Inertia")
plt.show()
```



Answer the following questions:

- **Question:** What is the best value for `k` when using the PCA data?
  - **Answer:** The best k-value is k=4 when using PCA data
- **Question:** Does it differ from the best k value found using the original data?
  - **Answer:** No, it is the same k value as found using the original data

```
plt.scatter(df_copy2.PCA_1, df_copy2.PCA_2, c=df_copy2.cluster)
plt.ylabel("PCA_1")
plt.xlabel("PCA_2")
plt.show()
```



Visualize and Compare the Results

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

Answer the following question:

- **Question:** After visually analyzing the cluster analysis results, what is the impact of using fewer features to cluster the data using K-Means?
- **Answer:** The impact of using PCA data resulted in tighter clusters, it also resulted in more entries within cluster 0 and cluster 1 than the original analysis did.