Importing "stock_market_data.csv"

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load the dataset
data = pd.read_csv('stock_market_data.csv')

# Preview the data
print("Original Data:")
data.head()
```

```
Original Data:
```

	Date	Open	High	Low	Close	Volume	Ticker	Daily_Change	PercentageChange
0	2020-01-02 00:00:00-05:00	71.721026	72.776606	71.466820	72.716080	135480400	AAPL	0.995053	1.387394
1	2020-01-03 00:00:00-05:00	71.941328	72.771745	71.783962	72.009117	146322800	AAPL	0.067789	0.094228
2	2020-01-06 00:00:00-05:00	71.127873	72.621654	70.876083	72.582916	118387200	AAPL	1.455043	2.045672
3	2020-01-07 00:00:00-05:00	72.592601	72.849231	72.021238	72.241554	108872000	AAPL	-0.351047	-0.483585
4	2020-01-08 00:00:00-05:00	71.943751	73.706271	71.943751	73.403641	132079200	AAPL	1.459889	2.029209

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12770 entries, 0 to 12769
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              12770 non-null  object
 1   Open              12770 non-null  float64
 2   High              12770 non-null  float64
 3   Low               12770 non-null  float64
 4   Close             12770 non-null  float64
 5   Volume            12770 non-null  int64
 6   Ticker            12770 non-null  object
 7   Daily_Change      12770 non-null  float64
 8   PercentageChange  12770 non-null  float64
dtypes: float64(6), int64(1), object(2)
memory usage: 898.0+ KB
```

```
data.describe()
```

	Open	High	Low	Close	Volume	Daily_Change	PercentageChange
count	12770.000000	12770.000000	12770.000000	12770.000000	1.277000e+04	12770.000000	12770.000000
mean	204.904713	207.872366	201.892925	204.928753	8.944684e+07	0.024039	0.051338
std	144.454432	146.183162	142.646392	144.448858	1.415215e+08	4.759927	2.203755
min	4.984596	5.229715	4.500834	4.892426	1.144000e+06	-50.033356	-15.206961
25%	114.588469	116.556723	112.730003	114.270868	1.872388e+07	-1.740005	-1.107405
50%	170.933771	173.146500	168.683550	171.000000	4.078850e+07	0.050003	0.050615
75%	251.811650	255.319725	248.242954	251.953011	8.537245e+07	1.872762	1.206521
max	998.030029	999.000000	970.010010	984.859985	1.543911e+09	55.030029	15.780256

```
data['Profit_Category'] = pd.cut(data['PercentageChange'],
                                  bins=[-float('inf'), -10, 0, 10, float('inf')],
                                  labels=['High Loss', 'Loss', 'Moderate Profit', 'High Profit'],
                                  include_lowest=True,
                                  right=False

                                 )
data
```

	Date	Open	High	Low	Close	Volume	Ticker	Daily_Change	PercentageChange	Profit_Category
0	2020-01-02 00:00:00-05:00	71.721026	72.776606	71.466820	72.716080	135480400	AAPL	0.995053	1.387394	Moderate Profit
1	2020-01-03 00:00:00-05:00	71.941328	72.771745	71.783962	72.009117	146322800	AAPL	0.067789	0.094228	Moderate Profit
2	2020-01-06 00:00:00-05:00	71.127873	72.621654	70.876083	72.582916	118387200	AAPL	1.455043	2.045672	Moderate Profit
3	2020-01-07 00:00:00-05:00	72.592601	72.849231	72.021238	72.241554	108872000	AAPL	-0.351047	-0.483585	Loss
4	2020-01-08 00:00:00	71.943751	73.706271	71.943751	73.403641	132079200	AAPL	1.459889	2.029209	Moderate Profit

Next steps: Generate code with data View recommended plots New interactive sheet

```
# Keep only numeric columns
numeric_data = data.select_dtypes(include=[np.number])
print("Numeric Data:")
numeric_data
```

Numeric Data:

	Open	High	Low	Close	Volume	Daily_Change	PercentageChange
0	71.721026	72.776606	71.466820	72.716080	135480400	0.995053	1.387394
1	71.941328	72.771745	71.783962	72.009117	146322800	0.067789	0.094228
2	71.127873	72.621654	70.876083	72.582916	118387200	1.455043	2.045672
3	72.592601	72.849231	72.021238	72.241554	108872000	-0.351047	-0.483585
4	71.943751	73.706271	71.943751	73.403641	132079200	1.459889	2.029209
...	...	...	...	...	...	...	...
12765	176.000000	180.429993	174.369995	176.059998	9304200	0.059998	0.034090
12766	175.550003	178.179993	174.399994	175.160004	7148600	-0.389999	-0.222159
12767	181.309998	188.479996	174.020004	177.779999	22768900	-3.529999	-1.946941
12768	179.130005	182.550003	170.649994	173.660004	12263500	-5.470001	-3.053649
12769	174.589996	179.940002	173.720001	179.529999	6996300	4.940002	2.829488

12770 rows × 7 columns

Next steps: Generate code with numeric_data View recommended plots New interactive sheet

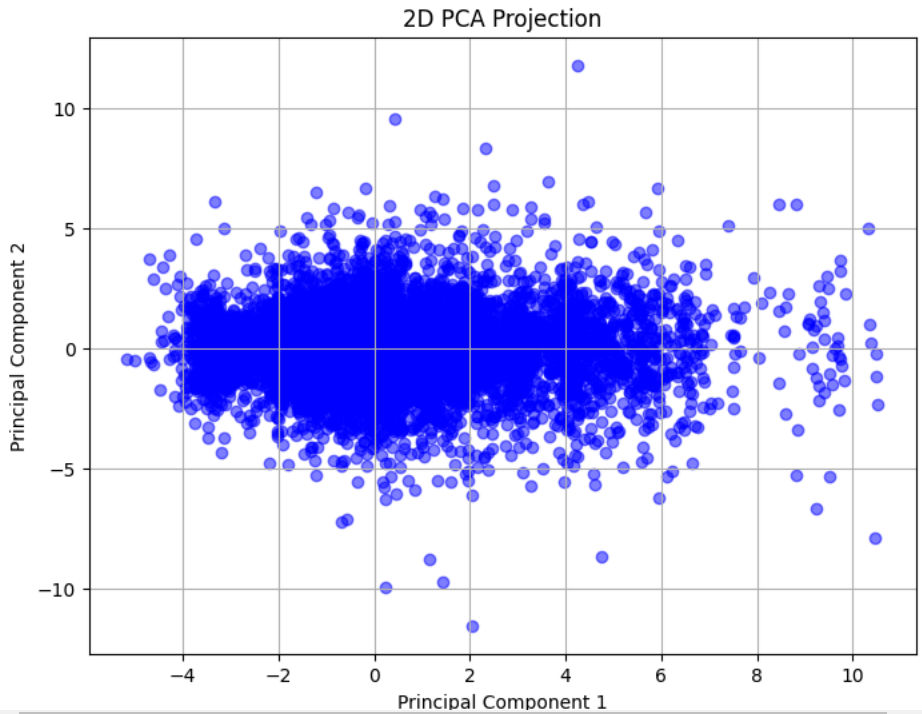
```
# Normalize the data using StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numeric_data)
pd.DataFrame(scaled_data, columns=numeric_data.columns).head()
```

	Open	High	Low	Close	Volume	Daily_Change	PercentageChange
0	-0.922013	-0.924190	-0.914367	-0.915326	0.325289	0.204006	0.606287
1	-0.920488	-0.924223	-0.912144	-0.920221	0.401905	0.009192	0.019463
2	-0.926120	-0.925250	-0.918509	-0.916248	0.204502	0.300647	0.905006
3	-0.915979	-0.923693	-0.910481	-0.918612	0.137265	-0.078804	-0.242742
4	-0.920471	-0.917830	-0.911024	-0.910566	0.301255	0.301666	0.897536

```
# PCA with 2 components
pca_2d = PCA(n_components=2)
data_pca_2d = pca_2d.fit_transform(scaled_data)
explained_var_2d = pca_2d.explained_variance_ratio_.sum() * 100
```

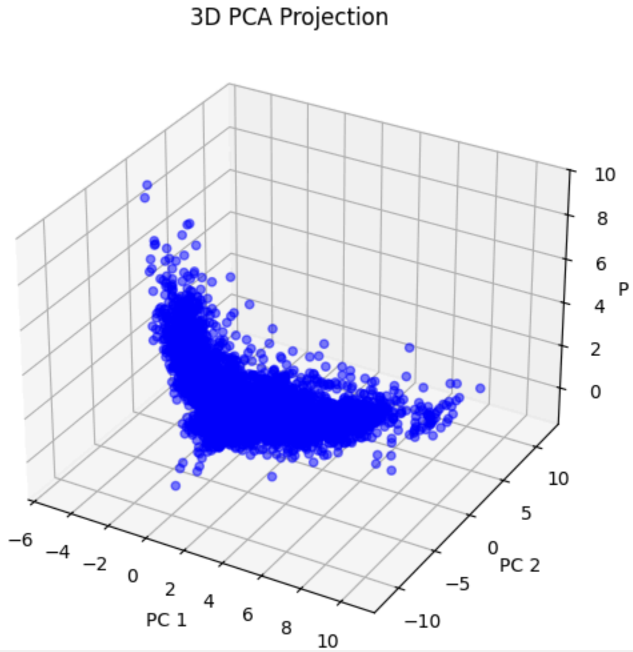
```
print(f"2D PCA retains {explained_var_2d:.2f}% of the variance")

# Plot 2D projection
plt.figure(figsize=(8, 6))
plt.scatter(data_pca_2d[:, 0], data_pca_2d[:, 1], alpha=0.5,c='blue')
plt.title('2D PCA Projection')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```

```
2D PCA retains 86.56% of the variance
```

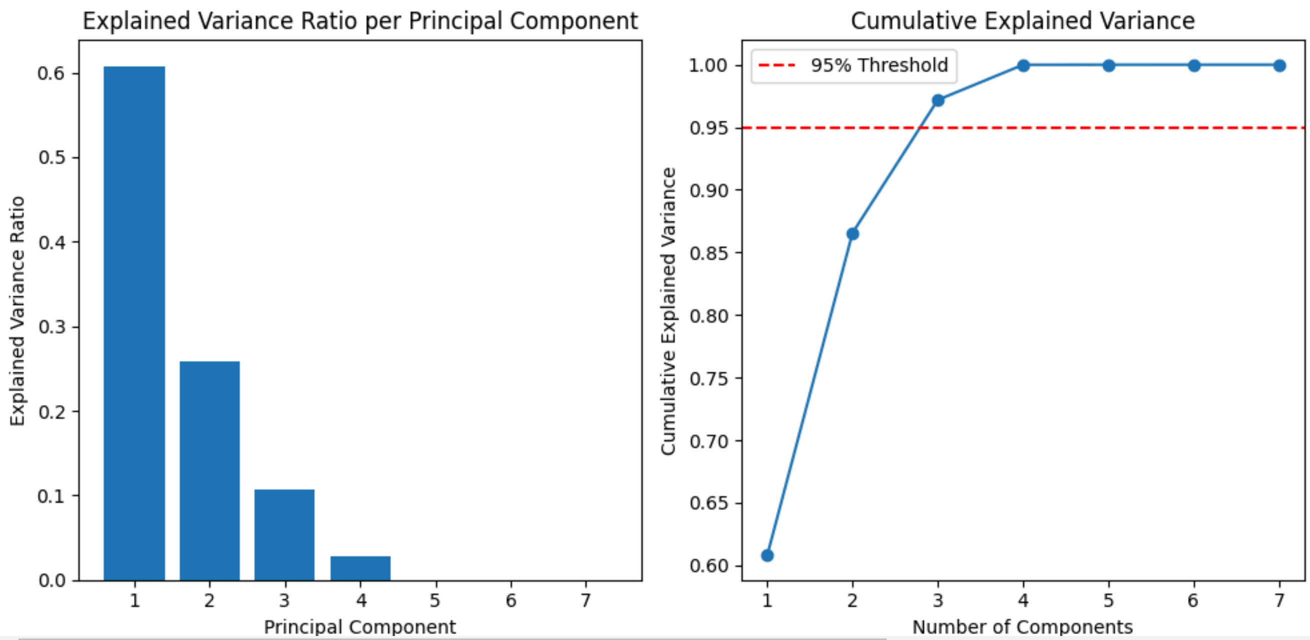



```
# PCA with 3 components
pca_3d = PCA(n_components=3)
data_pca_3d = pca_3d.fit_transform(scaled_data)
explained_var_3d = pca_3d.explained_variance_ratio_.sum() * 100
print(f"3D PCA retains {explained_var_3d:.2f}% of the variance")

# Plot 3D projection
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data_pca_3d[:, 0], data_pca_3d[:, 1], data_pca_3d[:, 2], alpha=0.5,c='blue')
ax.set_title('3D PCA Projection')
ax.set_xlabel('PC 1')
ax.set_ylabel('PC 2')
ax.set_zlabel('PC 3')
plt.show()
```

```
3D PCA retains 97.19% of the variance
```




```
# Calculate the number of components to retain at least 95% variance
pca_full = PCA().fit(scaled_data)

# Explained variance ratio
explained_variance_ratio = pca_full.explained_variance_ratio_

# Cumulative explained variance
cumulative_variance = np.cumsum(explained_variance_ratio)

# Plot explained variance
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.bar(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio)
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio per Principal Component')

# Plot cumulative explained variance
plt.subplot(1, 2, 2)
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance')
plt.axhline(y=0.95, color='r', linestyle='--', label='95% Threshold')  # Add 95% threshold line
plt.legend()

plt.tight_layout()
plt.show()
```




```
eigenvalues = pca_full.explained_variance_

# Get the top three eigenvalues
top_three_eigenvalues = eigenvalues[:3]

# Print the top three eigenvalues
print("Top three eigenvalues:", top_three_eigenvalues)
```

```
Top three eigenvalues: [4.25575302 1.80369308 0.74410938]
```

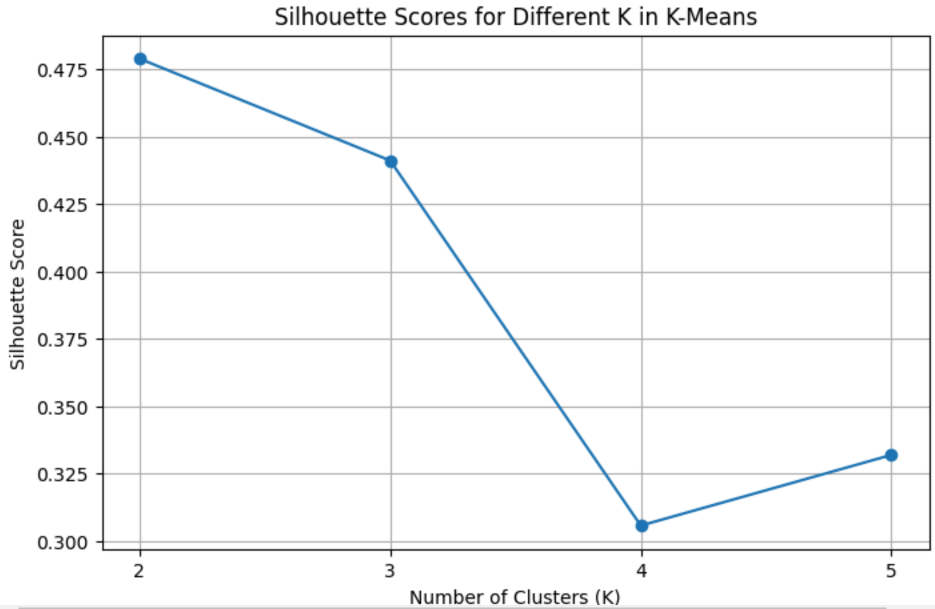
Clustering

```
#Reduced 3D pca data
df_cluster = data_pca_3d

# Finding the optimal K using the silhouette method
silhouette_scores = {}
for k in range(2, 6):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    cluster_labels = kmeans.fit_predict(df_cluster)
    silhouette_scores[k] = silhouette_score(df_cluster, cluster_labels)

silhouette_scores
```

```
{2: 0.47887121053535353,
 3: 0.44109592046218216,
 4: 0.3057366879085549,
 5: 0.3319351912529001}
```

```
optimal_k = max(silhouette_scores, key=silhouette_scores.get)
print(f"Optimal number of clusters for KMeans: {optimal_k}")

# Plot the Silhouette Scores for different K values
plt.figure(figsize=(8, 5))
plt.plot(list(silhouette_scores.keys()), list(silhouette_scores.values()), marker='o', linestyle='-')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Scores for Different K in K-Means")
plt.xticks(list(silhouette_scores.keys()))
plt.grid(True)
plt.show()
```

```
Optimal number of clusters for KMeans: 2
```



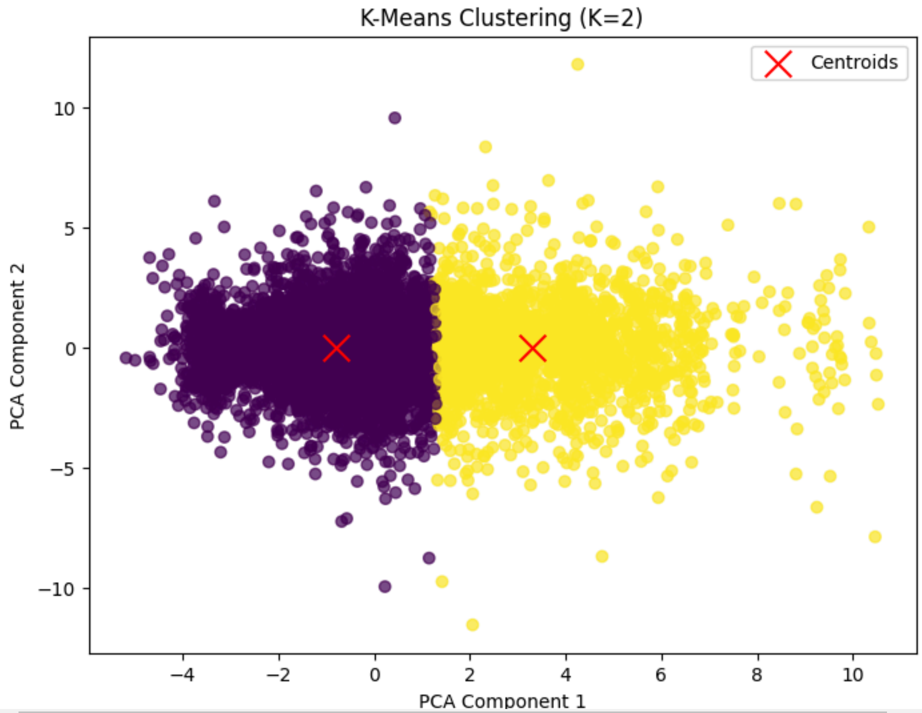

```
# Applying KMeans with optimal K
kmeans_final = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df_kmeans_labels = kmeans_final.fit_predict(df_cluster)

# Get the centroids in the PCA-reduced space
centroids_pca_space = kmeans_final.cluster_centers_

# Use pca_3d for inverse_transform
centroids_original_space = pca_3d.inverse_transform(centroids_pca_space)

# Print the centroids in the PCA-reduced space
print("Centroids in PCA-reduced space:")
print(centroids_pca_space)
```

```
Centroids in PCA-reduced space:
[[-0.80013832 -0.00668016 -0.0975436 ]
 [ 3.30833228  0.02762048  0.40331358]]
```

```
print("\nCentroids in original feature space:")
print(centroids_original_space)
```

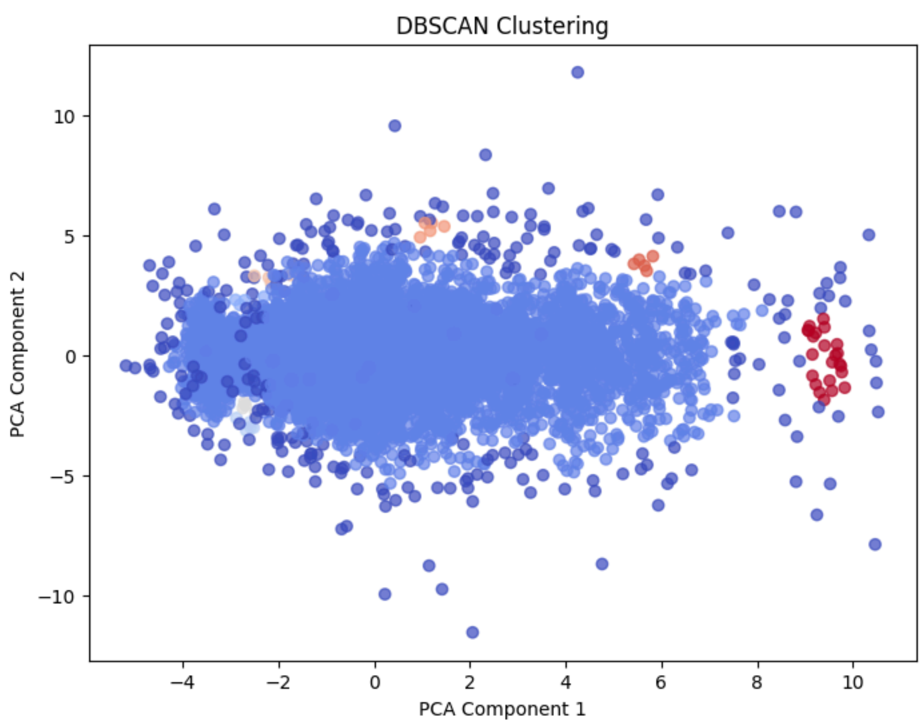
```

Centroids in original feature space:
[[-3.98369022e-01 -3.98467105e-01 -3.98311706e-01 -3.98398159e-01
   1.22181395e-01 -4.17759802e-04  2.24254615e-03]
 [ 1.64713657e+00  1.64754212e+00  1.64689959e+00  1.64725704e+00
  -5.05183467e-01  1.72731164e-03 -9.27225656e-03]]
```

```
from sklearn.cluster import KMeans
# Applying KMeans with optimal K
kmeans_final = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df_kmeans_labels = kmeans_final.fit_predict(df_cluster)

# Get centroids
centroids = kmeans_final.cluster_centers_

# Plot KMeans Clustering in 2D
plt.figure(figsize=(8, 6))
plt.scatter(df_cluster[:, 0], df_cluster[:, 1], c=df_kmeans_labels, cmap='viridis', alpha=0.7)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200, c='red', label='Centroids') # Plot centroids
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.title(f"K-Means Clustering (K={optimal_k})")
plt.legend() # Show legend to identify centroids
plt.show()
```

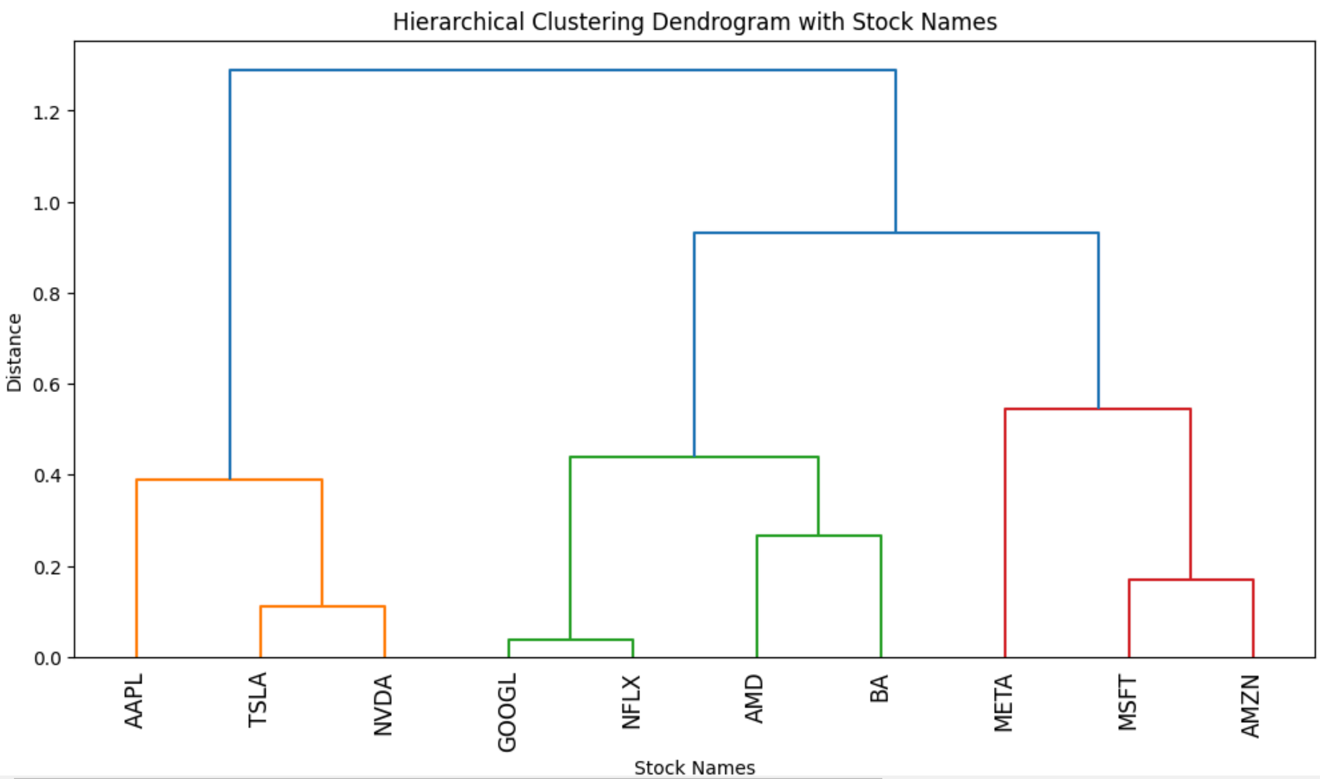


DBSCAN

```
from sklearn.cluster import DBSCAN

# Initialize and fit DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
df_dbscan_labels = dbscan.fit_predict(df_cluster)

plt.figure(figsize=(8, 6))
plt.scatter(df_cluster[:, 0], df_cluster[:, 1], c=df_dbscan_labels, cmap='coolwarm', alpha=0.7)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.title("DBSCAN Clustering")
plt.show()
```



```
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
import numpy as np

# Mapping of numerical labels to stock tickers
ticker_mapping = {
    1: 'AAPL', 2: 'MSFT', 3: 'GOOGL', 4: 'TSLA', 5: 'AMZN',
    6: 'NVDA', 7: 'META', 8: 'NFLX', 9: 'AMD', 10: 'BA'
}

# Generate dummy data for clustering
np.random.seed(42)
data = np.random.rand(len(ticker_mapping), 2)  # Assume 2D feature space
linkage_matrix = sch.linkage(data, method='ward')

# Plot dendrogram with meaningful labels
plt.figure(figsize=(12, 6))
dendro = sch.dendrogram(linkage_matrix, labels=[ticker_mapping.get(i+1, i+1) for i in range(len(ticker_mapping))], leaf_rotation=90)
plt.title("Hierarchical Clustering Dendrogram with Stock Names")
plt.xlabel("Stock Names")
plt.ylabel("Distance")
plt.show()
```



Preparing data (converting) for ARM

```
data = pd.read_csv('stock_market_data.csv')
data
```

	Date	Open	High	Low	Close	Volume	Ticker	Daily_Change	PercentageChange
0	2020-01-02 00:00:00-05:00	71.721026	72.776606	71.466820	72.716080	135480400	AAPL	0.995053	1.387394
1	2020-01-03 00:00:00-05:00	71.941328	72.771745	71.783962	72.009117	146322800	AAPL	0.067789	0.094228
2	2020-01-06 00:00:00-05:00	71.127873	72.621654	70.876083	72.582916	118387200	AAPL	1.455043	2.045672
3	2020-01-07 00:00:00-05:00	72.592601	72.849231	72.021238	72.241554	108872000	AAPL	-0.351047	-0.483585
4	2020-01-08 00:00:00-05:00	71.943751	73.706271	71.943751	73.403641	132079200	AAPL	1.459889	2.029209
...	...	...	...	...	...	...	...	...	...
12765	2025-01-24 00:00:00-05:00	176.000000	180.429993	174.369995	176.059998	9304200	BA	0.059998	0.034090
12766	2025-01-27 00:00:00-05:00	175.550003	178.179993	174.399994	175.160004	7148600	BA	-0.389999	-0.222159
12767	2025-01-28 00:00:00-05:00	181.309998	188.479996	174.020004	177.779999	22768900	BA	-3.529999	-1.946941
12768	2025-01-29 00:00:00-05:00	179.130005	182.550003	170.649994	173.660004	12263500	BA	-5.470001	-3.053649
12769	2025-01-30 00:00:00-05:00	174.589996	179.940002	173.720001	179.529999	6996300	BA	4.940002	2.829488

12770 rows × 9 columns

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data['Price_Change_Category'] = pd.cut(data['Daily_Change'],
                                       bins=[-float('inf'), -2, 0, 2, float('inf')],
                                       labels=['Big Drop', 'Small Drop', 'Small Rise', 'Big Rise'])

data
```

	Date	Open	High	Low	Close	Volume	Ticker	Daily_Change	PercentageChange	Price_Change_Category
0	2020-01-02 00:00:00-05:00	71.721026	72.776606	71.466820	72.716080	135480400	AAPL	0.995053	1.387394	Small Rise
1	2020-01-03 00:00:00-05:00	71.941328	72.771745	71.783962	72.009117	146322800	AAPL	0.067789	0.094228	Small Rise
2	2020-01-06 00:00:00-05:00	71.127873	72.621654	70.876083	72.582916	118387200	AAPL	1.455043	2.045672	Small Rise
3	2020-01-07 00:00:00-05:00	72.592601	72.849231	72.021238	72.241554	108872000	AAPL	-0.351047	-0.483585	Small Drop
	2020-01-									

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data['Trade_Volume'] = pd.qcut(data['Volume'], q=4,
                               labels=['Very Low', 'Low', 'High', 'Very High'])
data
```

	Date	Open	High	Low	Close	Volume	Ticker	Daily_Change	PercentageChange	Price_Change_Category
0	2020-01-02 00:00:00-05:00	71.721026	72.776606	71.466820	72.716080	135480400	AAPL	0.995053	1.387394	Small Rise
1	2020-01-03 00:00:00-05:00	71.941328	72.771745	71.783962	72.009117	146322800	AAPL	0.067789	0.094228	Small Rise
2	2020-01-06 00:00:00-05:00	71.127873	72.621654	70.876083	72.582916	118387200	AAPL	1.455043	2.045672	Small Rise
3	2020-01-07 00:00:00-05:00	72.592601	72.849231	72.021238	72.241554	108872000	AAPL	-0.351047	-0.483585	Small Drop
	2020-01-									

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data['Trend_Strength'] = pd.cut(data['PercentageChange'],
                               bins=[-float('inf'), -5, -2, 0, 2, 5, float('inf')],
                               labels=['Strong Loss', 'Moderate Loss', 'Small Loss', 'Small Gain', 'Moderate Gain', 'Strong Gain'])
data
```

	Date	Open	High	Low	Close	Volume	Ticker	Daily_Change	PercentageChange	Price_Change_Category
0	2020-01-02 00:00:00-05:00	71.721026	72.776606	71.466820	72.716080	135480400	AAPL	0.995053	1.387394	Small Rise
1	2020-01-03 00:00:00-05:00	71.941328	72.771745	71.783962	72.009117	146322800	AAPL	0.067789	0.094228	Small Rise
2	2020-01-06 00:00:00-05:00	71.127873	72.621654	70.876083	72.582916	118387200	AAPL	1.455043	2.045672	Small Rise
3	2020-01-07 00:00:00-05:00	72.592601	72.849231	72.021238	72.241554	108872000	AAPL	-0.351047	-0.483585	Small Drop
4	2020-01-08 00:00:00-05:00	71.943751	73.706271	71.943751	73.403641	132079200	AAPL	1.459889	2.029209	Small Rise
...	...	...	...	...	...	...	...	...	...	...
12765	2025-01-24 00:00:00-05:00	176.000000	180.429993	174.369995	176.059998	9304200	BA	0.059998	0.034090	Small Rise
12766	2025-01-27 00:00:00-05:00	175.550003	178.179993	174.399994	175.160004	7148600	BA	-0.389999	-0.222159	Small Drop
12767	2025-01-28 00:00:00-05:00	181.309998	188.479996	174.020004	177.779999	22768900	BA	-3.529999	-1.946941	Big Drop

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
tech_stocks = ['AAPL', 'MSFT', 'GOOGL', 'NVDA', 'META']
consumer_stocks = ['AMZN', 'NFLX']
industrial_stocks = ['BA', 'AMD']
```

```
data['Sector'] = data['Ticker'].apply(lambda x: 'Tech' if x in tech_stocks
                                      else 'Consumer' if x in consumer_stocks
                                      else 'Industrial')
```

	Date	Open	High	Low	Close	Volume	Ticker	Daily_Change	PercentageChange	Price_Change_Category
0	2020-01-02 00:00:00-05:00	71.721026	72.776606	71.466820	72.716080	135480400	AAPL	0.995053	1.387394	Small Rise
1	2020-01-03 00:00:00-05:00	71.941328	72.771745	71.783962	72.009117	146322800	AAPL	0.067789	0.094228	Small Rise
2	2020-01-06 00:00:00-05:00	71.127873	72.621654	70.876083	72.582916	118387200	AAPL	1.455043	2.045672	Small Rise
3	2020-01-07 00:00:00-05:00	72.592601	72.849231	72.021238	72.241554	108872000	AAPL	-0.351047	-0.483585	Small Drop
4	2020-01-08 00:00:00-05:00	71.943751	73.706271	71.943751	73.403641	132079200	AAPL	1.459889	2.029209	Small Rise
...	...	...	...	...	...	...	...	...	...	...
12765	2025-01-24 00:00:00-05:00	176.000000	180.429993	174.369995	176.059998	9304200	BA	0.059998	0.034090	Small Rise