Grouping Financial instruments.

Objective

Here in this project we choose a financial data from yahoo finance. We create a data set by using this data and some feature engineering.

The main objective of this project is to capture behaviour of different financial instruments in the market. Mainly to cluster the instruments into three clusters which may be

- High risk and high return
- Low risk and low return
- Neutral

We create out data capturing the return, volatility of the prices and volume and group them into clusters and see if there is any groups.

This is usefull in portfolio creation and analysis. You can analyse how the funds are allocated to different sectors to eliminate risk.

Data

The data contain `Price`, `Open`, `High`, `Low`, `Close` and `Volume` for different stocks in SP&500 and additional instruments such as bonds, crypto and indexes (tickers) from 2018-01-01 to 2023-01-01.

Using this dataset, we create our own dataset. We take the `Close` from the dataset for all the tickers. We create new features such as

1. `Returns`: annual average of the close price.
2. `Volatility`: annual standard deviation of the stocks.
3. `Sharpe_ratio`: the ratio of the above two.
4. `Momentum_30` : thirty day average of the returns.

These four should capture the price movements fairly.

Exploratory Data Analysis

Data Cleaning

There is not much to do here. There are some missing values, which we have dropped.

```
First five rows of the raw data:
```

	Ticker					GEN			
	Price	**Open**	**High**	**Low**	**Close**	**Volume**	**Open**	**High**	**L**
0	Date	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
1	2018-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2	2018-01-02	14.200767	14.597745	14.095240	14.527394	5539600.0	16.920730	17.224896	16.7351
3	2018-01-03	14.477144	14.658045	14.421869	14.527394	4273700.0	17.081929	17.413468	17.0515
4	2018-01-04	14.673120	14.698245	14.467094	14.557544	5283000.0	17.443888	17.629429	17.2279

5 rows × 2622 columns

```
/Users/basava/Documents/Coursera/coursera/lib/python3.13/site-packages/numpy/lib/_fun
ction_base_impl.py:2999: RuntimeWarning: invalid value encountered in divide
  c /= stddev[:, None]
/Users/basava/Documents/Coursera/coursera/lib/python3.13/site-packages/numpy/lib/_fun
ction_base_impl.py:3000: RuntimeWarning: invalid value encountered in divide
  c /= stddev[None, :]
/Users/basava/Documents/Coursera/coursera/lib/python3.13/site-packages/pandas/core/na
nops.py:1016: RuntimeWarning: invalid value encountered in subtract
  sqr = _ensure_numeric((avg - values) ** 2)
```

```
First five rows of the data created:
```

	Returns	Volatility	Sharpe_ratio	Momentum_30
Ticker				
MET	0.117824	-1.241169	0.407631	0.013181
EQIX	0.097080	-1.379629	0.385740	0.011473
GDDY	0.109694	-1.137203	0.342030	0.010984
TPL	0.328377	-0.846534	0.765629	0.040343
REGN	0.125569	-1.284607	0.453714	0.015315

Data scaling

We standardise the data using standard scarar.

```
First five columns of the scaled data:
```

	Returns	Volatility	Sharpe_ratio	Momentum_30
Ticker				
MET	0.073623	0.033037	0.144373	0.026625
EQIX	-0.149529	-0.416896	0.047982	-0.118527
GDDY	-0.013835	0.370879	-0.144488	-0.160098
TPL	2.338598	1.315417	1.720744	2.335009
REGN	0.156941	-0.108117	0.347288	0.208007

There is some skewness in the distribution. We will not do anything about it because it might be important for clustering.

```
Returns         2.671394
Volatility      0.245427
Sharpe_ratio    0.250710
Momentum_30     3.780119
dtype: float64
```

Now we train four models.

1. Kmeans
2. DBSCAN
3. Hirarchical Agglomerative Clustering and
4. Meanshift

We fit and predict the and add the results as columns to dataframe.

```
# Kmeans
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5, random_state=22)
df['Cluster_Kmeans'] = kmeans.fit_predict(df_scaled)

# DBSCAN
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.09, min_samples=4, p=2)
df['Cluster_DBSCAN'] = dbscan.fit_predict(df_scaled)

#HAC
from sklearn.cluster import AgglomerativeClustering
hac = AgglomerativeClustering(n_clusters=5, linkage='ward')
df['Cluster_Hierarchical'] = hac.fit_predict(df_scaled)

# Meanshift
from sklearn.cluster import MeanShift, estimate_bandwidth
bandwidth = estimate_bandwidth(df_scaled, quantile=.03, n_samples=3000)
ms = MeanShift(bandwidth=bandwidth,bin_seeding=True)
df['Cluster_Meanshift'] = ms.fit_predict(df_scaled)
```

We use silhouette score for scoring.

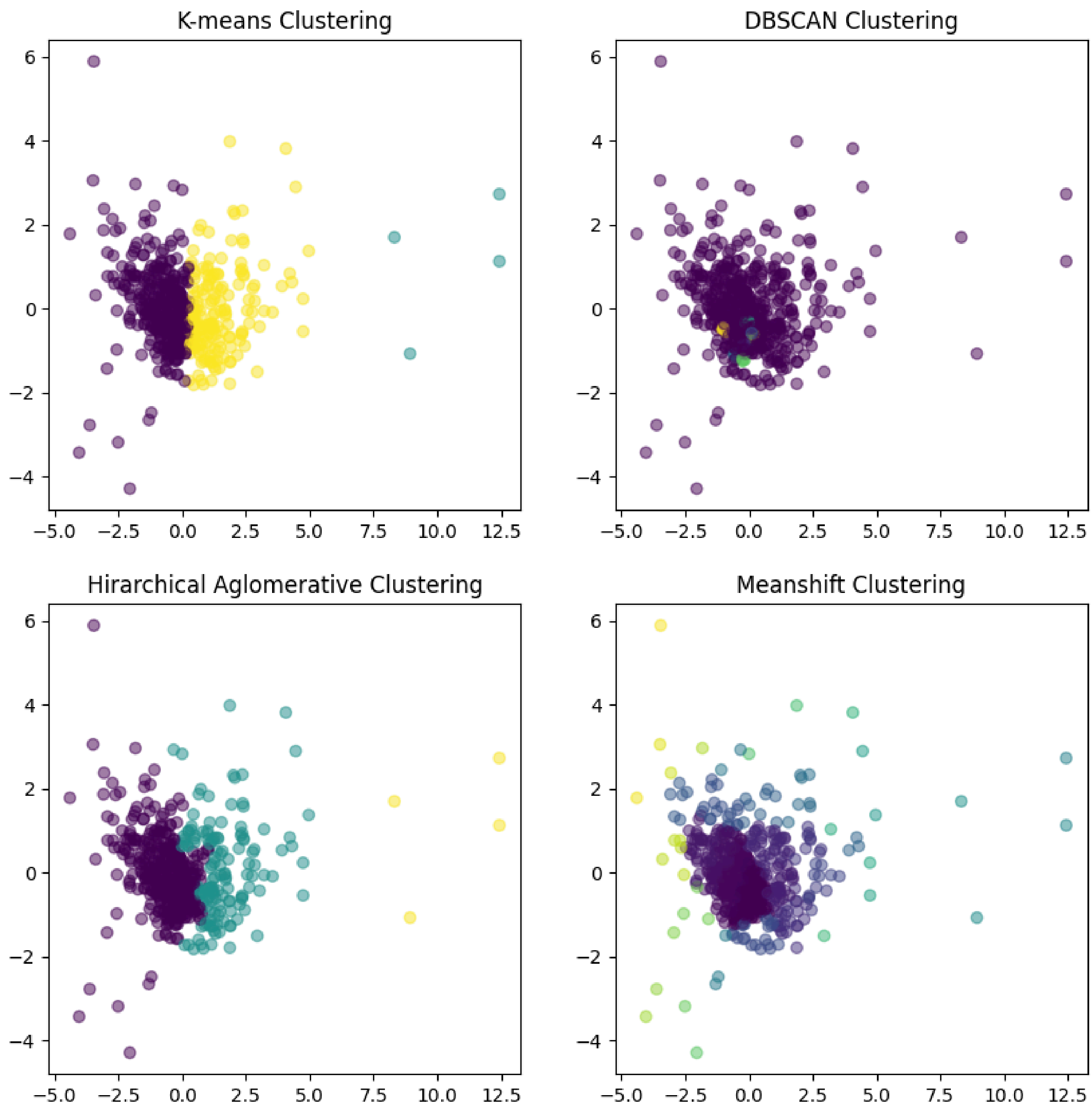
```python
from sklearn.metrics import silhouette_score

print("K-Means Silhouette:", silhouette_score(df_scaled,
df['Cluster_Kmeans']))
print("DBSCAN Silhouette:", silhouette_score(df_scaled,
df['Cluster_DBSCAN']))
print("Hierarchical Agglomatative Silhouette:", silhouette_score(df_scaled,
df['Cluster_Hierarchical']))
print("Meanshift Silhouette:", silhouette_score(df_scaled,
df['Cluster_Meanshift']))
```

```
K-Means Silhouette: 0.36297545497101297
DBSCAN Silhouette: -0.4275794744532688
Hierarchical Agglomatative Silhouette: 0.36886651731733344
Meanshift Silhouette: 0.2261673564546393
```

Data reduction for visualization

Now we use PCA and reduce the data dimension to 2D and plot to see how clusters are formed.

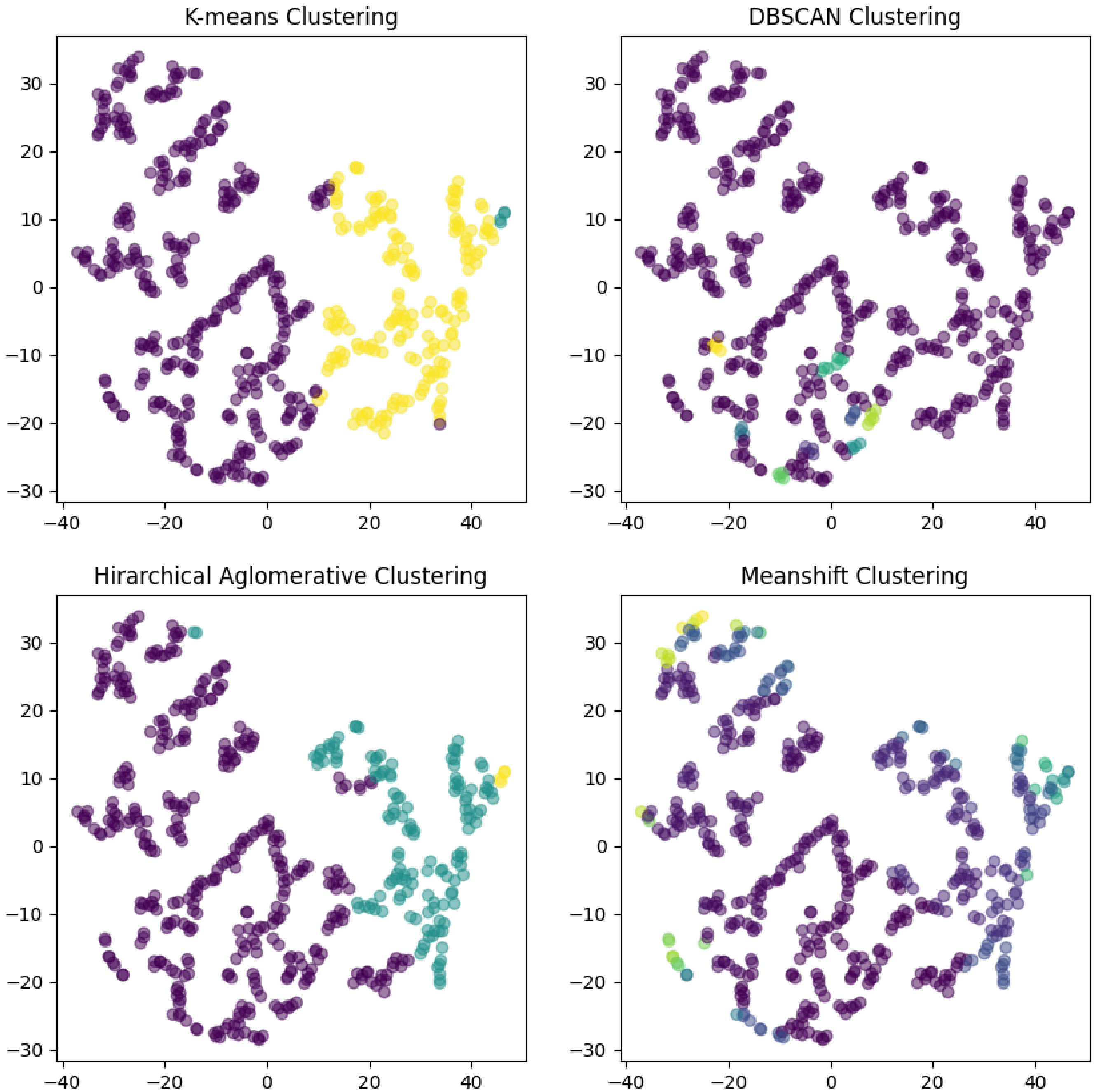
```python
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)
```



As expected the Kmeans and HAC has done a better job than DBSCAN and Meanshift.

Now we do the same i.e., data reduction via TSNE and plot the clusters.

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2,
        init="pca",
        n_iter=500,
        n_iter_without_progress=150,
        perplexity=12,
        random_state=2)

df_tsne = tsne.fit_transform(df_scaled)
```



Again here we clearly see that Meanshift and HAC has done a better job compared to the other two.

Clustering after reducing data dimension using TSNE

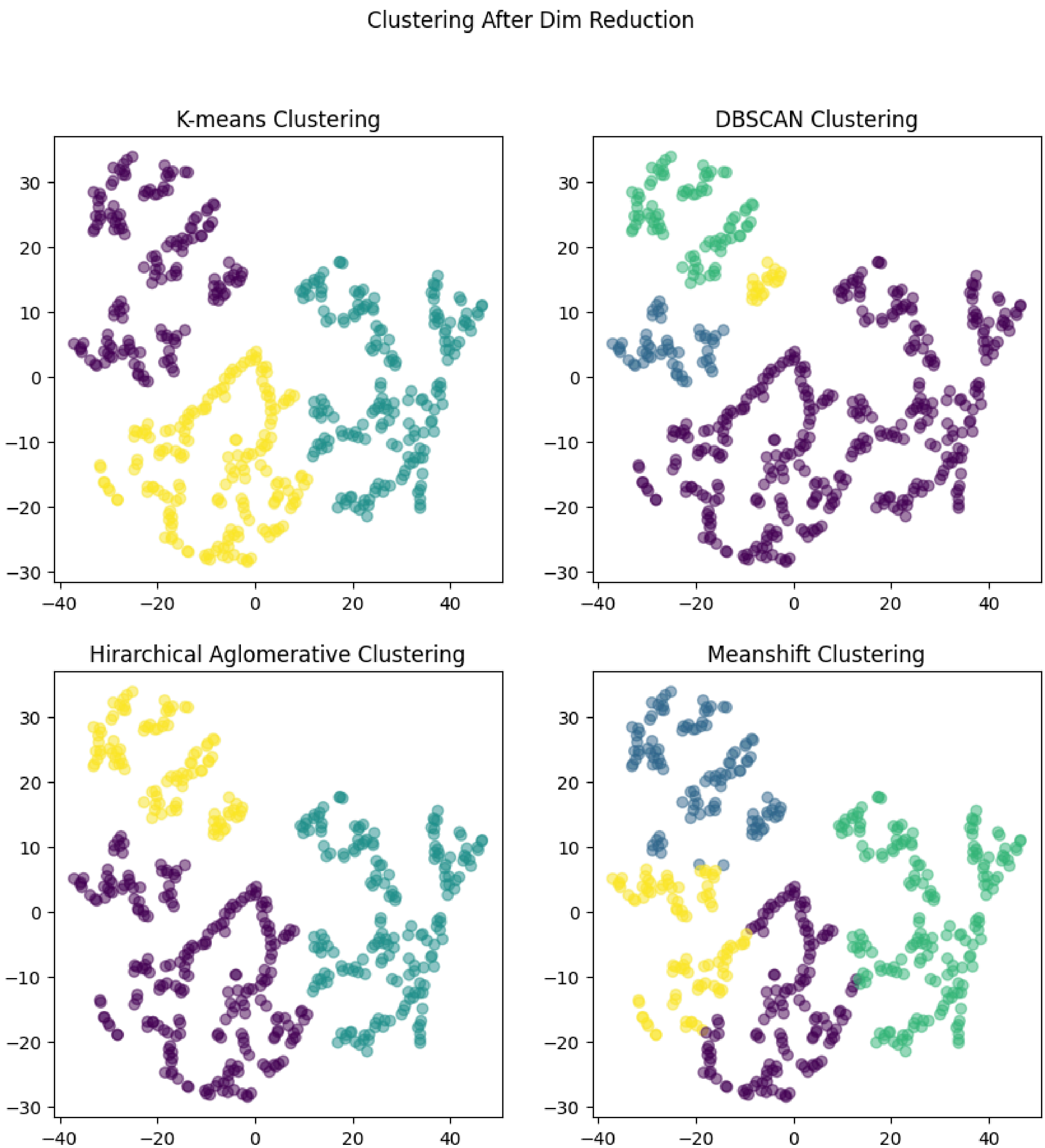
Now we use the dimension reduced dataset `data_tsne` and apply clustering algorithms to this dataset.

```
# Kmeans
df['Cluster_tsne_Kmeans'] = kmeans.fit_predict(df_tsne)

# DBSCAN
dbscan_tsne = DBSCAN(eps=6, min_samples=5, p=2)
```

```python
df['Cluster_tsne_DBSCAN'] = dbscan_tsne.fit_predict(df_tsne)

# HAC
df['Cluster_tsne_Hierarchical'] =  hac.fit_predict(df_tsne)

# Meanshift
bandwidth = estimate_bandwidth(df_tsne, quantile=.2, n_samples=5000)
ms_tsne = MeanShift(bandwidth=bandwidth,bin_seeding=True)
df['Cluster_tsne_Meanshift'] = ms_tsne.fit_predict(df_tsne)
```

```
K-Means Silhouette: 0.47651485
DBSCAN Silhouette: 0.17220163
Hierarchical Agglomatative Silhouette: 0.44451645
Meanshift Silhouette: 0.41535842
```

Again silhouette scores are calculated. Now we see that Kmeans, HAC, Meanshift performs better in this dimension reduced data. However DBSCAN eventhough has good score, identifies only two clusters.



Observations

- With the entire data, the Kmeans and HAC does a pretty good job in clustering. This is mainly because we are kind of know how many clusters we are lookin for.
- With the low dimension data, all the model performs better except DBSCAN which could not identify the third cluster.

Recommendation

I recommend the Kmeans algorithm for the problem concerned here. This is because of three reasons.

1. It is intuitive and easy to understand and explain.
2. We already know the number of clusters. So it is better to use Kmeans as it is simple.
3. It does a eqivalently good job compared to other models both in full and reduced data.

Further steps

- We can engineer more features like 'Average volume traded', 'Volume volatility', 'Cumulative returns' etc. to better capture risk and return relations more accurately.
- We can increase the data set with more tickers.
- We can tune the parameters of the model to even perform better.
- We can analyse the clusters formed after dimension reduction and look for the meaning for those cluster. Right now we only know the data forms three clusters, what they mean we do not know. They can be as our problem stated are the clusters of
    - High risk and high return
    - Low risk and low return
    - Neutral

```
[NbConvertApp] Converting notebook yahoo_fin.ipynb to webpdf
[NbConvertApp] WARNING | Alternative text is missing on 3 image(s).
[NbConvertApp] Building PDF
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 981765 bytes to yahoo_fin.pdf
```