Time Series Clustering for Minute Univariate Intraday Stock Price for Trend Estimation

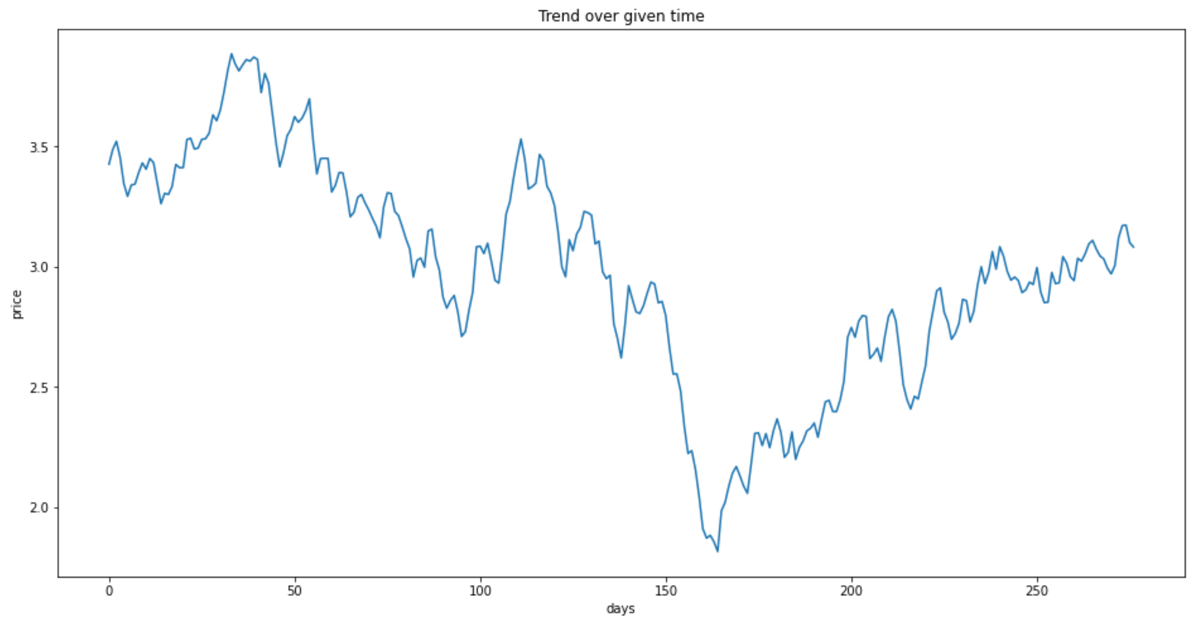
Some facts about the given data:

- The data is an unlabeled collection of daily variations for a singular stock logged every minute
- There are 277 days worth of data and 1440 data points, corresponding to one observation every minute
- All days have an equal number of data (1440) and no data point is absent

Here the data is graphed over its entire lifetime and the mean for each day is plotted

In [ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('simulated_data.csv')

df_daily_mean = df.groupby('day')['price'].mean()
df_daily_mean.plot(figsize=(16, 8))
plt.xlabel("days")
plt.ylabel("price")
plt.title("Trend over given time")
```

Out[ ]:
```
Text(0.5, 1.0, 'Trend over given time')
```



Data Preprocessing and Cleaning

Cleaning

Cleaning the data and getting rows of time series, where each row is a time series of price and each column is its corresponding time (in minutes) during the day. The orignal data can be interpreted as a multiindex pandas dataframe and unstacking it gives a 2D array where each row is a seperate series

```
In [ ]: df = pd.read_csv('simulated_data.csv', index_col=[0, 1])

        df = df.unstack(level=0)
        df = df.T
        print(df.shape)
        df = df.reset_index()
        df = df.drop(['level_0', 'day'], axis=1)

        df.head()
```

```
(277, 1440)
```

Out[ ]:

time	0	1	2	3	4	5	6	7	8	
0	3.477110	3.475698	3.475874	3.478345	3.476757	3.478169	3.475874	3.475522	3.471992	3.4
1	3.422223	3.419752	3.417987	3.421341	3.418693	3.418340	3.420988	3.423458	3.420105	3.4
2	3.474110	3.474816	3.477463	3.479757	3.479581	3.476757	3.473580	3.473404	3.470756	3.4
3	3.504995	3.507112	3.504995	3.505877	3.507112	3.508171	3.504642	3.502700	3.502700	3.5
4	3.426106	3.424870	3.422399	3.421517	3.419046	3.417105	3.415870	3.416046	3.415164	3.4

5 rows × 1440 columns

Preprocessing

To extract meaningful insights from the data it needs to obtained in a form that is most helpful to the insight that is desired, to achieve this two tasks are carried out:

1. **Rolling Average**: To smooth out extreme minute variations that may lead to noisy interpretations a rolling window of 30 is applied that averages 30 values and uses that average as a new data point for the time series, this leads to a more clear 'trend' as sudden sharp spikes are shown only if the subsequent data point reflects it too. The `.rolling(window='x')` can be modifed to further smooth the data. The rolling average can be changed to any other statistical feature to display the trend over that specific feature

   > Making an **assumption** here that any strong variation within the day takes atleast 30 minutes to build up and affect the trend for the day, hence the rolling window of 30, this value can be easily changed

2. **Normalization**: Since the desired outcome is trend clustering, the Open and Close values do not mean much going from one day to another, to remove this variable all values for a day are subtracted from the Opening (1st) value of that day. The data obtained after this all starts from the same point in the y-axis and fluctuates around the 1st value. This ensures that all our time series start from the value of 0

Both of these can be applied in unison to give less noisy and more meaningful data

In [ ]:
```
print(df.shape)

df_rolling = df.copy()

window_size = 30 # window_size can be changed here <<<<<<<<<<<<<<<<<<

df_rolling = df_rolling.T
for i in range(0, len(df_rolling.columns)):
    df_rolling[i] = df_rolling[i].rolling(window=window_size).mean()


df_rolling = df_rolling.T
df_rolling = df_rolling.drop(list(range(0, window_size-1)), axis=1)


df_normalized_rolling = df_rolling.sub(df_rolling[window_size-1], axis=0)

print(df_normalized_rolling.shape)
df_normalized_rolling.head()
```

```
(277, 1440)
(277, 1411)
```

Out[ ]:

time	29	30	31	32	33	34	35	36	37
0	0.0	-0.000335	-0.000647	-0.001053	-0.001647	-0.002188	-0.002700	-0.003224	-0.003765
1	0.0	0.000194	0.000359	0.000647	0.000765	0.000994	0.001147	0.001188	0.001153
2	0.0	0.000053	0.000124	0.000212	0.000341	0.000512	0.000777	0.001124	0.001382
3	0.0	-0.000582	-0.001259	-0.001824	-0.002483	-0.003283	-0.004077	-0.004730	-0.005306
4	0.0	-0.000612	-0.001177	-0.001624	-0.002112	-0.002412	-0.002712	-0.002930	-0.003224

5 rows × 1411 columns

Clustering

The clustering problem is split into two parts:

1. Determination of the number of clusters
2. Assigning the data to those clusters

For this problem Kmeans Clustering and Hierarchical Clustering will be used

Determining the number of clusters

Three methods to determine n_clusters will be used:

1. **Elbow Method** using Kmeans classification from which inertia and clusters are plotted and n_clusters is determined based on diminishing returns. Elbow Method gives a quick

way to estimate the number of clusters before adding more clusters will tend `K -> number of days`, as in an ideal clustering each day graph has its own cluster.

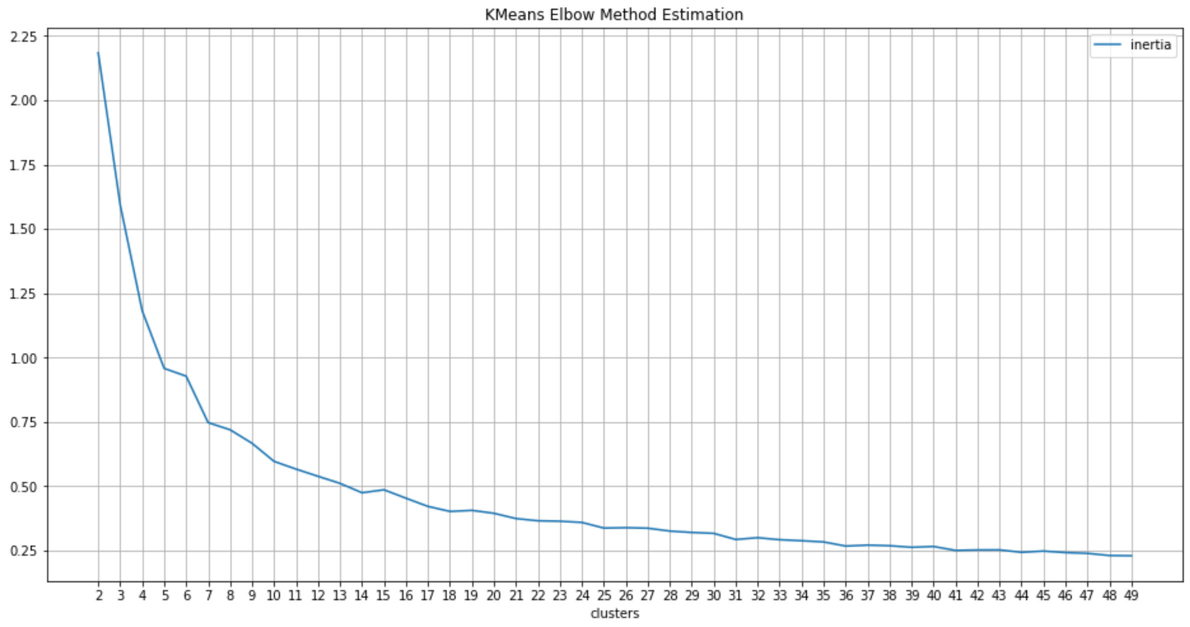
2. A **Complete Linkage Dendogram** based on Hierachical Clustering, the ideal application for this clustering algorithm as clusters with similar members are required

3. **Ward's Method** of minimizing sum of squares, this has the benefits of both of the above algorithms

These three methods are used together because as a scientific mind once said, "measure thrice, average once"

In [ ]:
```
from tslearn.clustering import TimeSeriesKMeans

np.random.seed(0)
elbow_data = []
clusters_data = []

for i in range(2, 50):

    elbow_model = TimeSeriesKMeans(
        n_clusters=i, metric="euclidean", max_iter=20)
    y_pred = elbow_model.fit_predict(df_normalized_rolling)
    elbow_data.append((i, elbow_model.inertia_))
    clusters_data.append(y_pred)

#fig = plt.figure()
df_elbow = pd.DataFrame(elbow_data, columns=['clusters', 'inertia'])
df_elbow.plot(x='clusters', y='inertia', figsize=(16, 8), title="KMeans Elbow Metho
plt.grid(True, axis='both')
plt.xticks(df_elbow['clusters'])
plt.show()
```

```
c:\Users\ayush\AppData\Local\Programs\Python\Python37\lib\site-packages\tslearn\ba
ses\bases.py:15: UserWarning: h5py not installed, hdf5 features will not be suppor
ted.
Install h5py to use hdf5 features: http://docs.h5py.org/
  warn(h5py_msg)
```



In [ ]:
```
for i in elbow_data[6:15]:
    print(i)
```

```
(8, 0.7197902398247988)
(9, 0.6672009543667904)
(10, 0.5970362042258951)
(11, 0.5670390127800196)
(12, 0.5391039833609608)
(13, 0.5116079173985734)
(14, 0.475202436189879)
(15, 0.486632425526135)
(16, 0.4539707633678821)
```

Elbow Method estimation does not give a sharp curve but it gives a nice range to make the estimation from. Here the Euclidean metric is used as the variations in time throughout the day need to be preserved, a Dynamic Time Warping metric would have skewed the intra day time variations unfavorably. The K estimation starts at 2 because the most basic clustering and intuitive knowledge dictates that either a stock closes higher than what it opens or vice versa

For Hierarchical Clustering a distance matrix needs to be created

In [ ]:
```
df_normalized_rolling_T = df_normalized_rolling.copy().T
distance_matrix = df_normalized_rolling_T.corr()
print(distance_matrix.shape)

distance_matrix.head(15)
```

```
(277, 277)
```

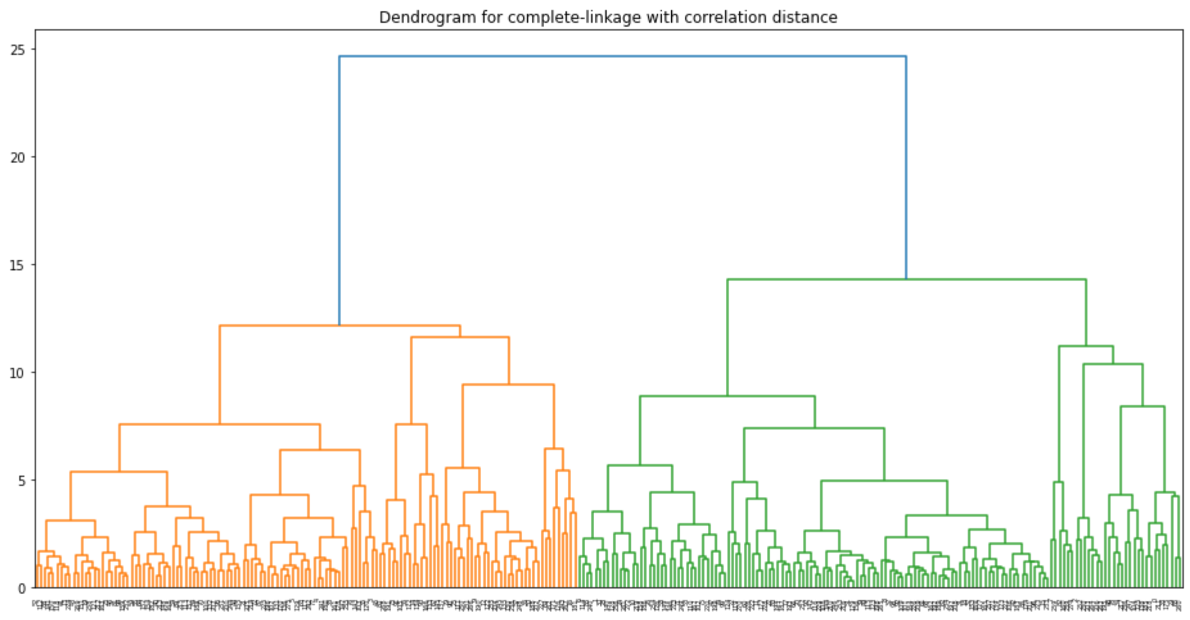
Out[ ]:

	0	1	2	3	4	5	6	7	
0	1.000000	-0.311427	-0.604379	-0.053320	-0.300280	-0.353506	-0.245560	-0.275817	0.48293
1	-0.311427	1.000000	0.778662	-0.348666	-0.566487	-0.504351	0.828968	0.057220	0.4013
2	-0.604379	0.778662	1.000000	-0.296211	-0.295455	-0.283558	0.696216	0.177984	0.0722
3	-0.053320	-0.348666	-0.296211	1.000000	0.688388	0.749831	-0.410371	0.174616	-0.6406
4	-0.300280	-0.566487	-0.295455	0.688388	1.000000	0.917589	-0.615920	0.387395	-0.9028
5	-0.353506	-0.504351	-0.283558	0.749831	0.917589	1.000000	-0.528358	0.347614	-0.8553
6	-0.245560	0.828968	0.696216	-0.410371	-0.615920	-0.528358	1.000000	-0.130356	0.5003
7	-0.275817	0.057220	0.177984	0.174616	0.387395	0.347614	-0.130356	1.000000	-0.3307
8	0.482938	0.401369	0.072284	-0.640665	-0.902895	-0.855322	0.500383	-0.330770	1.0000
9	-0.309671	-0.047695	0.010880	0.523640	0.263513	0.460539	-0.042057	-0.251071	-0.5086
10	-0.246312	0.879936	0.700238	-0.446801	-0.605201	-0.585321	0.770096	0.060608	0.4849
11	0.207626	0.532401	0.252321	-0.510632	-0.860816	-0.790550	0.593499	-0.275559	0.7511
12	-0.287181	-0.059457	0.036485	0.433437	0.308786	0.393981	-0.045764	-0.379393	-0.5153
13	-0.052970	-0.599589	-0.476517	0.705532	0.618283	0.722478	-0.519066	-0.148216	-0.6413
14	-0.277919	-0.343963	-0.124448	0.393920	0.761711	0.594015	-0.491693	0.752770	-0.6934

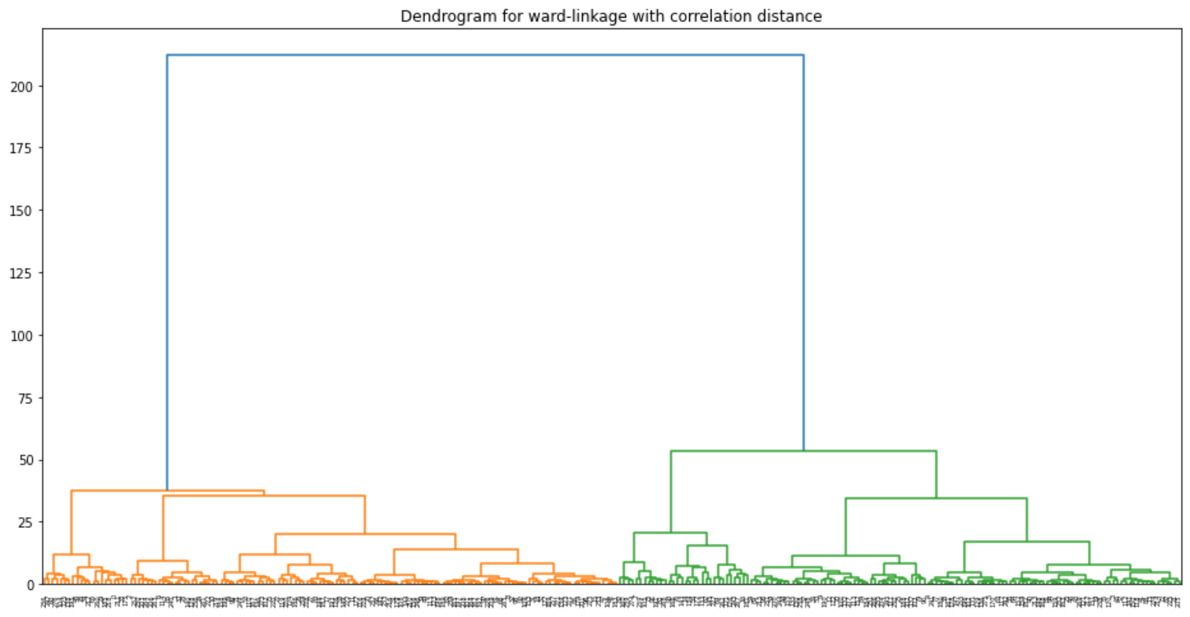
15 rows × 277 columns

In [ ]:
```
from scipy.cluster.hierarchy import complete, dendrogram, ward
```

```
linkage_matrix_complete = complete(distance_matrix)
#linkage_matrix = ward(distance_matrix)
fig = plt.figure(figsize=(16, 8))
dn = dendrogram(linkage_matrix_complete)
plt.title(f"Dendrogram for complete-linkage with correlation distance")
plt.show()
```



In [ ]:
```
linkage_matrix_ward = ward(distance_matrix)
fig = plt.figure(figsize=(16, 8))
dn = dendrogram(linkage_matrix_ward)
plt.title(f"Dendrogram for ward-linkage with correlation distance")
plt.show()
```



n_clusters:

K-Estimation Method	n_clusters with window_size = 5	n_clusters with window_size = 30
Elbow Method	12	14
Complete Linkage	12	13
Ward's Method	11	8

Slicing of the dendograms was done manually

> `12` is a good middle ground between resolution and comprehensibility for this intraday univariate stock data

In [ ]:
```
n_clusters = 12 # change clusters here <<<<<<<<<<<<<<<<
```

Fitting to n_clusters

Four models will be used to cluster the data:

1. **KMeans**: as a baseline for the n_clusters estimated using the Elbow Method

In [ ]:
```
model1 = TimeSeriesKMeans(n_clusters=n_clusters, metric="euclidean", max_iter=50)
model1.fit(df_normalized_rolling)

y_KMeans = model1.predict(df_normalized_rolling)
```

```
c:\Users\ayush\AppData\Local\Programs\Python\Python37\lib\site-packages\tslearn\ut
ils\utils.py:89: UserWarning: 2-Dimensional data passed. Assuming these are 277 1-
dimensional timeseries
  '{} 1-dimensional timeseries'.format(X.shape[0]))
```

2. **KShape**: as an imporvement on KMeans [1]

In [ ]:
```
from tslearn.clustering import KShape

model2 = KShape(n_clusters=n_clusters, verbose=True, max_iter=50)
model2.fit(df_normalized_rolling)

y_KShape = model2.predict(df_normalized_rolling)
```

```
0.134 --> 0.108 --> 0.105 --> 0.105 --> 0.105 --> 0.105 --> 0.105 --> 0.105 --> 0.
105 --> 0.105 --> 0.104 --> 0.104 --> 0.104 --> 0.104 --> 0.104 --> 0.104 --> 0.10
4 --> 0.104 --> 0.104 -->
```

```
c:\Users\ayush\AppData\Local\Programs\Python\Python37\lib\site-packages\tslearn\ut
ils\utils.py:89: UserWarning: 2-Dimensional data passed. Assuming these are 277 1-
dimensional timeseries
  '{} 1-dimensional timeseries'.format(X.shape[0]))
```

3. **FCluster**: to utilize the Complete Linkage Matrix

In [ ]:
```
from scipy.cluster.hierarchy import fcluster

y_FCluster_complete = fcluster(linkage_matrix_complete, n_clusters, criterion='max
```

4. **FCluster**: to utilize the Ward's Linkage Matrix

In [ ]:
```
y_FCluster_ward = fcluster(linkage_matrix_ward, n_clusters, criterion='maxclust')
```

Adding the cluster groups to the initial non preprocessed dataframe as a coloumn and exporting it as a CSV for vizualization

For final vizualizations use the `data_playground_clustered.py` and `cluster_visualize.py` scripts, there are comments within these scripts that let one change the model outputs they are viewing

In [ ]:

```
df_clustered = df.copy()
df_clustered = df_clustered.sub(df_clustered[0], axis=0)

df_clustered['y_KMeans'] = y_KMeans
df_clustered['y_KShape'] = y_KShape
df_clustered['y_FCluster_complete'] = [i - 1 for i in y_FCluster_complete]
df_clustered['y_FCluster_ward'] = [i - 1 for i in y_FCluster_ward]

df_clustered.to_csv('simulated_data_clustered.csv')
df_clustered.head(15)
```

Out[ ]:

time	0	1	2	3	4	5	6	7	8
0	0.0	-0.001412	-0.001235	0.001235	-0.000353	0.001059	-0.001235	-0.001588	-0.005118
1	0.0	-0.002471	-0.004236	-0.000882	-0.003530	-0.003883	-0.001235	0.001235	-0.002118
2	0.0	0.000706	0.003353	0.005648	0.005471	0.002647	-0.000529	-0.000706	-0.003353
3	0.0	0.002118	0.000000	0.000882	0.002118	0.003177	-0.000353	-0.002294	-0.002294
4	0.0	-0.001235	-0.003706	-0.004589	-0.007059	-0.009001	-0.010236	-0.010060	-0.010942
5	0.0	-0.001765	-0.003883	-0.006000	-0.006530	-0.007412	-0.005295	-0.006000	-0.003177
6	0.0	-0.001765	-0.004236	-0.000706	-0.002824	-0.002294	-0.000882	0.002471	0.003530
7	0.0	0.002824	0.003000	0.000882	0.004236	0.000882	0.001941	0.000000	-0.001059
8	0.0	0.001941	0.000176	-0.002471	-0.004412	-0.007589	-0.010236	-0.008118	-0.005295
9	0.0	-0.003353	-0.004765	-0.006177	-0.006000	-0.007942	-0.008295	-0.010766	-0.013060
10	0.0	0.003530	0.003000	0.002471	0.003883	0.001412	-0.002118	-0.004412	-0.006000
11	0.0	0.000529	0.001941	0.000882	0.000176	-0.001588	-0.002118	-0.001765	0.000706
12	0.0	0.001588	0.004765	0.001412	0.000529	0.003353	0.001059	-0.001765	0.001588
13	0.0	0.000000	0.001235	0.003000	0.000176	-0.001765	-0.004765	-0.003353	-0.004942
14	0.0	-0.002118	0.000353	0.002647	0.004412	0.004412	0.003883	0.006000	0.003530

15 rows × 1444 columns

In [ ]:

```
import pickle

with open('n_clusters.pkl', 'wb') as pklfile:
    pickle.dump(n_clusters, pklfile)
```

Areas of Improvement

- n_clusters estimation can be done using computer intelligence rather than human intervention
- More statistical features can be taken and accounted for in the rolling window (std, var)
- The rolling window size can be justified using real world data

- Another similar stock price graph that is already classified can be used to train a model and predictions for this data can be made from it
- Vizualization should have better colours but that is a drawback of the vizualization dynamically scaling with any change in n_clusters

References

- [1] https://www.sciencedirect.com/science/article/pii/S2666827020300013
- [2] https://towardsdatascience.com/how-to-apply-hierarchical-clustering-to-time-series-a5fe2a7d8447
- [3] https://www.kaggle.com/code/albertferre/timeserieskmeans-clustering/notebook
- [4] https://towardsdatascience.com/hands-on-climate-time-series-clustering-using-machine-learning-with-python-6a12ce1607f9