Task 0

Transforming the data:

A new Data Frame is created with columns Date, and the 10 companies. The Data Frame is indexed on the date, which allows us to locate and insert data into specific cells. We iterate over the Task 0 csv file from Phase 2. For each iteration (row), we compute the difference between Close and Open. If the result is non-negative, we traverse to the location corresponding to the Date (index) and Company name (column). In this location, we insert the Volume. If the result is negative, we insert 0. For instances where a company's stocks are not traded on a specific day, we have filled these values with 0. However, these account for fewer than 20 cells in a table of 782 records. The results of the procedure are shown below. The full csv file is included in the source code submission, along with the implementation in Python (Google Collaboratory). The duplicate Dates below are not an error, but the Data Frame displaying the index (Date) and the Date column concurrently.

	Date	NTDOY	SONY	MSFT	ATVI	EA	TTWO	\
2020-01-02	2020-01-02	1180500	856500	22622100	0	0	0	
2020-01-03	2020-01-03	506000	0	21116200	4466300	1840300	729100	
2020-01-06	2020-01-06	1162500	755500	20813700	4910800	2934200	1678700	
2020-01-07	2020-01-07	0	0	0	4920300	0	1179300	
2020-01-08	2020-01-08	2783000	0	27746500	0	2651600	1665900	
...	...	...	...	...	...	...	...	
2022-12-26	2022-12-26	0	0	0	0	0	0	
2022-12-27	2022-12-27	0	0	0	0	0	1225100	
2022-12-28	2022-12-28	0	0	0	3049900	0	1928800	
2022-12-29	2022-12-29	684300	574500	19770700	3150100	1222800	1526400	
2022-12-30	2022-12-30	862900	413500	21938500	2710400	1164400	1749700	

	NCBDY	CCOEY	TCEHY	SQNXF
2020-01-02	0	0	4293200	0
2020-01-03	0	0	2728500	0
2020-01-06	0	0	4712100	0
2020-01-07	917700	917700	4667000	0
2020-01-08	0	0	2925200	0
...	...	...	...	...
2022-12-26	295300	295300	0	0
2022-12-27	0	0	4433500	100
2022-12-28	0	0	0	200
2022-12-29	541200	541200	4546700	0
2022-12-30	0	0	1505500	100

Task 1

Finding 10 Frequent Item-Sets:

The csv file from Task 0 requires attribute transformation before we can generate item-sets. Specifically, the Volume attribute must be transformed from a continuous to binary attribute. To achieve this, we have applied a lambda function to the data frame. The function returns 1 if the parameter num is greater than 0, or 0 otherwise. Lambda functions are automatically applied to every cell in the data frame, so we have added an integer type requirement to receive the binary outputs above. This ensures we are not replacing fields such as the Date with binary values. This step was necessary because the $apriori(\,)$ function – which will be used to generate item-sets – requires the data to be in a binary. That includes all the data in the Data Frame, which is why we have dropped the table index and Date column. This leaves the 10 columns, one for each company, and binary values in each column. Finally, we can call the $apriori(\,)$ function by passing the Data Frame and minimum support value. For minimum support, we have chosen an arbitrarily small value so that all item-sets are generated. In return, the function generates a new Data Frame containing the item-sets and their corresponding support values. On the new Data Frame, we append a column 'length' indicating the number of items in the item-set. The length attribute is used to extract item-sets with at least 2 items into another Data Frame. The resulting item-sets are sorted by their support in ascending order, and the first 10 are extracted into the $frequent\_itemset$ Data Frame. The $frequent\_itemset$ is shown below and included as a csv file in the source code. As suggested by the Apriori Principle, the most frequent item-sets contain the fewest items.

	support	itemsets	length
49	0.483995	(NCBDY, CCOEY)	2
40	0.370038	(TTWO, EA)	2
19	0.367478	(MSFT, SONY)	2
35	0.366197	(ATVI, TTWO)	2
29	0.346991	(MSFT, TTWO)	2
34	0.344430	(ATVI, EA)	2
25	0.332907	(TCEHY, SONY)	2
32	0.331626	(MSFT, TCEHY)	2
11	0.330346	(MSFT, NTDOY)	2
27	0.329065	(MSFT, ATVI)	2

Generating 10 Association Rules:

To generate rules from the item-sets, the $association\_rules(\,)$ function will be invoked. The function requires the Data Frame containing all item-sets, and the minimum confidence threshold. For minimum confidence, we have chosen an arbitrarily small value so that all rules are generated. In return, the function generates a new Data Frame containing the rules and corresponding metrics for each item-set. Rules for the 10 frequent item-sets are extracted by sorting the data by 'support' and retrieving the first 20 rows. Since the most frequent item-sets contain two items, there are two possible rules for each. Additionally, support is independent of which items are the antecedent (left side of the rule) and consequent (right side of the rule). Thus, the first 20 rows contain rules for the 10 most frequent item-sets. From the set of 20, we extract the 5 rules with highest confidence by sorting in descending order of confidence and retrieving the first 5. Conversely, we extract the 5 rules with lowest confidence by sorting in ascending

order of confidence and retrieving the first 5. The highest confidence rules are shown below on the left, and the lowest confidence rules are shown on the right. Individual csv files containing the highest and lowest confidence rules are included in the csv, along with the metrics which are not included below.

	antecedents	consequents
79	(CCOEY)	(NCBDY)
78	(NCBDY)	(CCOEY)
61	(EA)	(TTWO)
19	(SONY)	(MSFT)
50	(ATVI)	(TTWO)

	antecedents	consequents
34	(MSFT)	(ATVI)
2	(MSFT)	(NTDOY)
44	(MSFT)	(TCEHY)
3	(NTDOY)	(MSFT)
38	(MSFT)	(TTWO)

Computing Measures of Interest:

The measures of interest we selected are $Odds\ Ratio, Jaccard, Cosine,$ and $Correlation$. To compute these measures, we iterate over the highest confidence rules. For each iteration (rule), we compute f_{11}, f_{10}, f_{01}, and f_{00} from the $antecedent\ support$ and $consequent\ support$. From these figures, we compute the f_{+1}, f_{1+}, f_{0+}, f_{+0} values. Lastly, we compute N from the previous values. Following the formulae on Chapter 5 slide 50, we compute the measures of interest and add the results to a list. After iterating over the highest confidence rules, we insert the list into the Data Frame as a new column. The results of the procedure are shown below. The '\' represents a continuation of the current row in the sub-table below. As described above, the computed values are appended as the final columns in the row. The highest confidence rules are shown below and included as a csv file in the source code.

	antecedents	consequents	antecedent support	consequent support	support	\
79	(CCOEY)	(NCBDY)	0.483995	0.483995	0.483995	
78	(NCBDY)	(CCOEY)	0.483995	0.483995	0.483995	
61	(EA)	(TTWO)	0.480154	0.505762	0.370038	
19	(SONY)	(MSFT)	0.478873	0.517286	0.367478	
50	(ATVI)	(TTWO)	0.477593	0.505762	0.366197	

	confidence	lift	leverage	conviction	zhangs_metric	odds ratio	\
79	1.000000	2.066138	0.249744	inf	1.000000	0.998975	
78	1.000000	2.066138	0.249744	inf	1.000000	0.998975	
61	0.770667	1.523774	0.127195	2.155108	0.661224	1.000458	
19	0.767380	1.483474	0.119763	2.075117	0.625388	1.001463	
50	0.766756	1.516042	0.124649	2.118975	0.651575	1.000517	

	jaccard	cosine	interest	correlation
79	0.326143	0.491867	0.999735	-0.000256
78	0.326143	0.491867	0.999735	-0.000256
61	0.330189	0.496495	1.000116	0.000114
19	0.332479	0.499130	1.000367	0.000365
50	0.329614	0.495854	1.000131	0.000129

Task 2

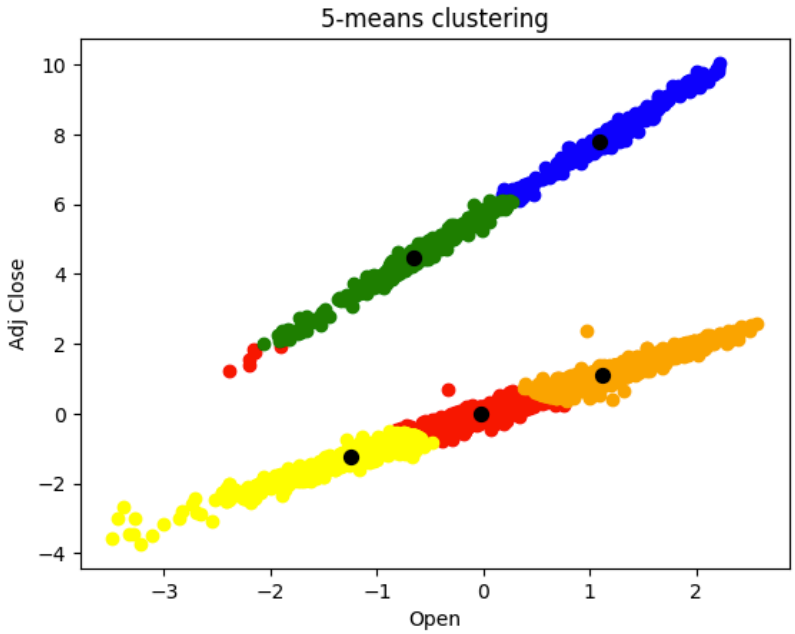
Preprocessing:

The data used in Phase 2 Task 6 requires normalization and feature reduction preprocessing before clustering. Normalization is achieved in Excel before the csv file is imported into the Google Collaboratory workspace. The Excel formula used to compute normalized values is listed below. The normalization preprocessing is included in the Task 2 Preprocessing.csv file within the source code submission. Feature reduction is achieved at runtime by extracting the *Open* and *Adj Close* features from the Data Frame containing *Open, Adj Close, Volume, Company, Direction* features.

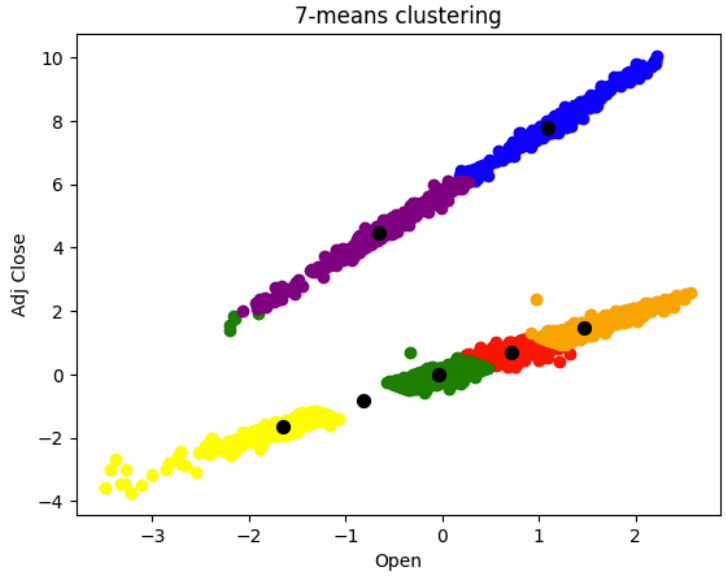
$$B_{norm} = (\$B2 - AVERAGE(\$B\$2{:}\ \$B\$757))/STDEV(\$B\$2{:}\ \$B\$757)$$

Performing k-means Clustering:

With the preprocessing complete, we can perform the clustering. The $sklearn.cluster$ library provides functionality for k-means clustering through the $KMeans$ class. We initialize an instance of the $KMeans$ class with parameters $n\_clusters$ and n_init. $n\_clusters$ specifies the number k of clusters the data will be grouped into. n_init specifies the number of times the k-means algorithm is performed with a random set of k centroids (cluster centers). To receive accurate results, we set n_init to an arbitrarily large number, 100. On the instance of the $KMeans$ class, we call the $fit(\ )$ function and pass the features as a parameter. This function applies the k-means algorithm to the data. The k-means clustering is computed, now it needs to be plotted. We create a new Data Frame with columns *Open*, *Adj Close*, and *Cluster* – which denotes the cluster each row was labeled as. For the i^{th} cluster, we extract every row with $Cluster\ = i$ to a new Data Frame. Given k clusters, we have k Data Frames. The $matplotlib$ library provides functionality for creating custom scatter plots. Each of the k Data Frames are inserted into the scatter plot with a different colour. The centroids (cluster centers) are highlighted in the graph by plotting them at a larger size and different colour. We access the centroids by retrieving the $cluster\_centers\_$ from the instance of the $KMeans$ class. The graph is labeled and displayed. The sum of squared errors (SSE) value is accessed by retrieving $inertia\_$ from the instance of the $KMeans$ class. The results for the 5-means clustering and 7-means clustering are shown below. k = 7 produces a lower SSE because a higher number of clusters produces more segments in the data, and fewer data points overlap in the wrong cluster. To illustrate, consider a set of 100 data points. The variance within each cluster would be lower if we had 100 clusters compared to 5.




```
SSE = 2936.2305308307514
```

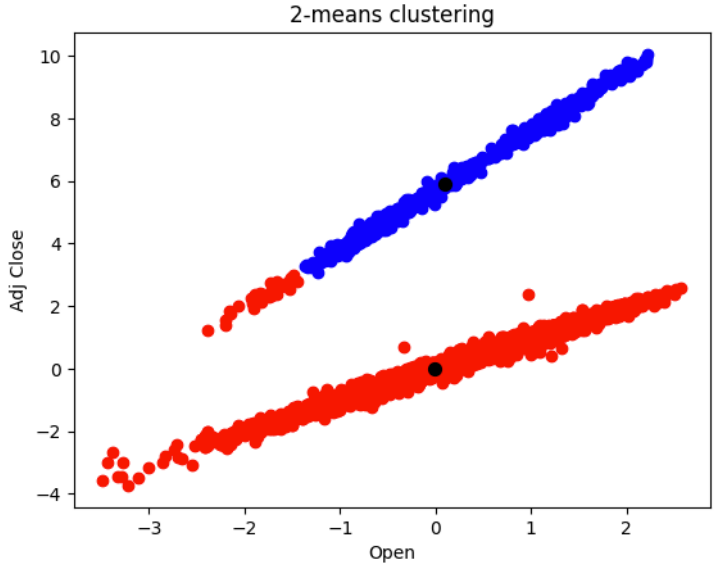



```
SSE = 1744.6931071873446
```

Performing 2-means Clustering:

The class from Phase 2 Task 6 was *Direction* which had value 0 if the stock price dropped (*Adj Close* – *Open* < 0) or 1 if the stock price rose (*Adj Close* – *Open* > 0). Performing 2-means clustering produces the results below. The implementation of 2-means clustering is as simple as modifying the *k* value from the previous clustering. Computing the accuracy of the clustering requires the actual and predicted values of *Direction*. We cannot compute the actual values on the current Data Frame because the values are normalized. So, we import the original data set from Phase 2 Task 6 once more. The actual values are computed by applying a Boolean condition to

each row in the Data Frame and mapping the results to 0 if False and 1 if True. From there, we append the *Cluster* labels to each row. Iterating over the Data Frame, we count every row where $Direction = Count$, and divide the count by the total number of rows. The result is accuracy of the clustering analysis which is shown below. The accuracy of 2-means clustering is 74.34%.



	Open	Adj Close	Direction	Cluster
0	10.136000	10.138000	1	1
1	9.946000	10.050000	1	1
2	9.880000	9.910000	1	1
3	9.880000	9.832000	0	1
4	9.820000	9.922000	1	1
...	...	...	...	...
7505	45.310001	45.310001	1	0
7506	45.299999	45.299999	1	0
7507	45.299999	45.299999	1	0
7508	45.299999	45.299999	1	0
7509	45.310001	45.310001	1	0

[7510 rows x 4 columns]
Accuracy: 74.34087882822902