Singular Value Decomposition Feature Analysis for Clustering Goldman Sachs Stock Valuation

AP Statistics 4th Period

Mitesh Shah, Hakimuddin Kitabwalla, Zain Arfoosh, Sunny Desai, Iliyan Gilani, Rachel Tran, Junmo Moon, Katelyn Ngo, Justin Lee, Aayan Mukul, Isaiah Iwayemi, Joan Leal
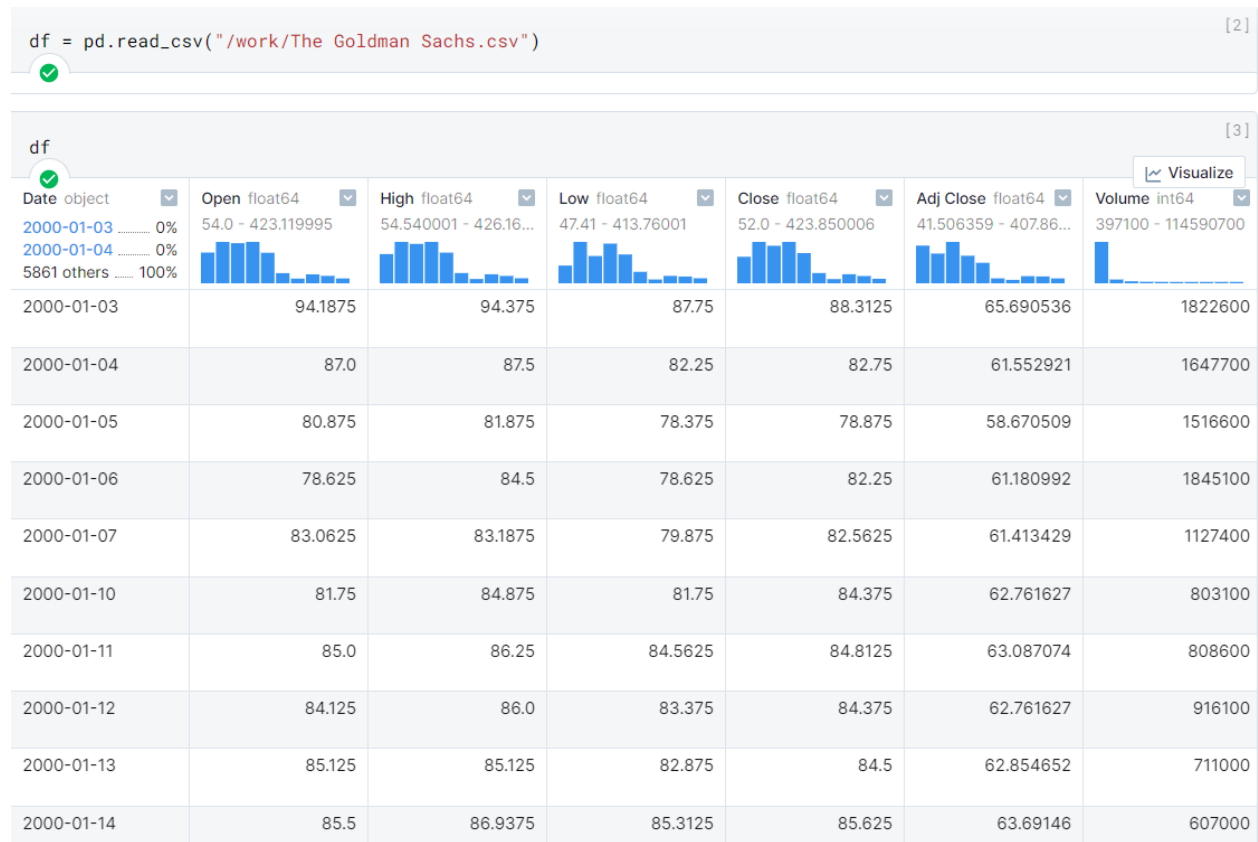
Github repository:

https://github.com/StudioMitesh/SVD-Clustering-Goldman-Sachs-SharePrices

Table of Contents

Data Set

```
df = pd.read_csv("/work/The Goldman Sachs.csv")
```
[2]

[3]

df

Date object	Open float64	High float64	Low float64	Close float64	Adj Close float64	Volume int64
2000-01-03 .... 0% 2000-01-04 .... 0% 5861 others 100%	54.0 - 423.119995	54.540001 - 426.16...	47.41 - 413.76001	52.0 - 423.850006	41.506359 - 407.86...	397100 - 114590700
2000-01-03	94.1875	94.375	87.75	88.3125	65.690536	1822600
2000-01-04	87.0	87.5	82.25	82.75	61.552921	1647700
2000-01-05	80.875	81.875	78.375	78.875	58.670509	1516600
2000-01-06	78.625	84.5	78.625	82.25	61.180992	1845100
2000-01-07	83.0625	83.1875	79.875	82.5625	61.413429	1127400
2000-01-10	81.75	84.875	81.75	84.375	62.761627	803100
2000-01-11	85.0	86.25	84.5625	84.8125	63.087074	808600
2000-01-12	84.125	86.0	83.375	84.375	62.761627	916100
2000-01-13	85.125	85.125	82.875	84.5	62.854652	711000
2000-01-14	85.5	86.9375	85.3125	85.625	63.69146	607000

Our dataset contains nearly 6000 days of price data and examines 6 unique factors of The Goldman Sachs stock, all of which are on a quantitative scale. We omitted the date after conversion because it was simply a counting variable and not actually a quantitative factor as we are purely focused on clustering quantitative factors.

As can be seen from the individual histograms in the pictures above, the 6 factors do not have completely normal distributions and all have right skews, however, this does not prove to be a major factor as all of the data points can be reasonably explained.

The unaltered dataset can be accessed through Kaggle.com from this link:
https://www.kaggle.com/datasets/varpit94/goldman-sachs-stock-data-updated-till-1jul2021

Data Dictionary

Variable Name	Data Type	Description
Open	Quantitative	Open price of the stock for the day
High	Quantitative	High price of the stock for the day
Low	Quantitative	Low price of the stock for the day
Close	Quantitative	Close price of the stock for the day
Adj Close	Quantitative	Closing price accounting for stock splits, dividends, and new stock offerings
Volume	Quantitative	Number of shares traded in the day

Research Questions

We chose to analyze the following 2 research questions:

1. How can the patterns and relationships between variables (open, high, low, close, adj close, volume) aid stock market analysis and prediction?

2. What is the association between the variables (open, high, low, close, adj close, volume)?

Preprocessing Data

First, we removed the date column from the dataset, leaving all of the data present in the dataset as truly quantitative. In terms of our singular value decomposition data mining algorithm, all of our factors needed to be quantified and standardized, so this dataset is validated for our data mining algorithm purposes.

Step 1: Removing Outliers

Our dataset contains chronological data, however, we see that it is not fit to remove outliers from the dataset because of the disconnections this would leave. All of the data points in our set can be reasonably explained based on the share prices and factors of the share price throughout the years as part of the dataset, leaving no data points to be considered "unusual" or as an outlier.

Step 2: Simplifying Objects

We did not aggregate any objects in our dataset as all of these factors are already individually quantified and unique.

Step 3: Simplifying Attributes

We utilized the singular value decomposition (SVD) to refactor the features of our data. We did not create any new features or subsect any other features in our dataset in order to preprocess the data.

Step 4: Altering Data Values

We had to examine for any missing data in our dataset and replace those missing values with proper values. First, we created a heatmap of the null values within the dataset using the Seaborn heatmap function.

```
sns.heatmap(df.isnull())
```
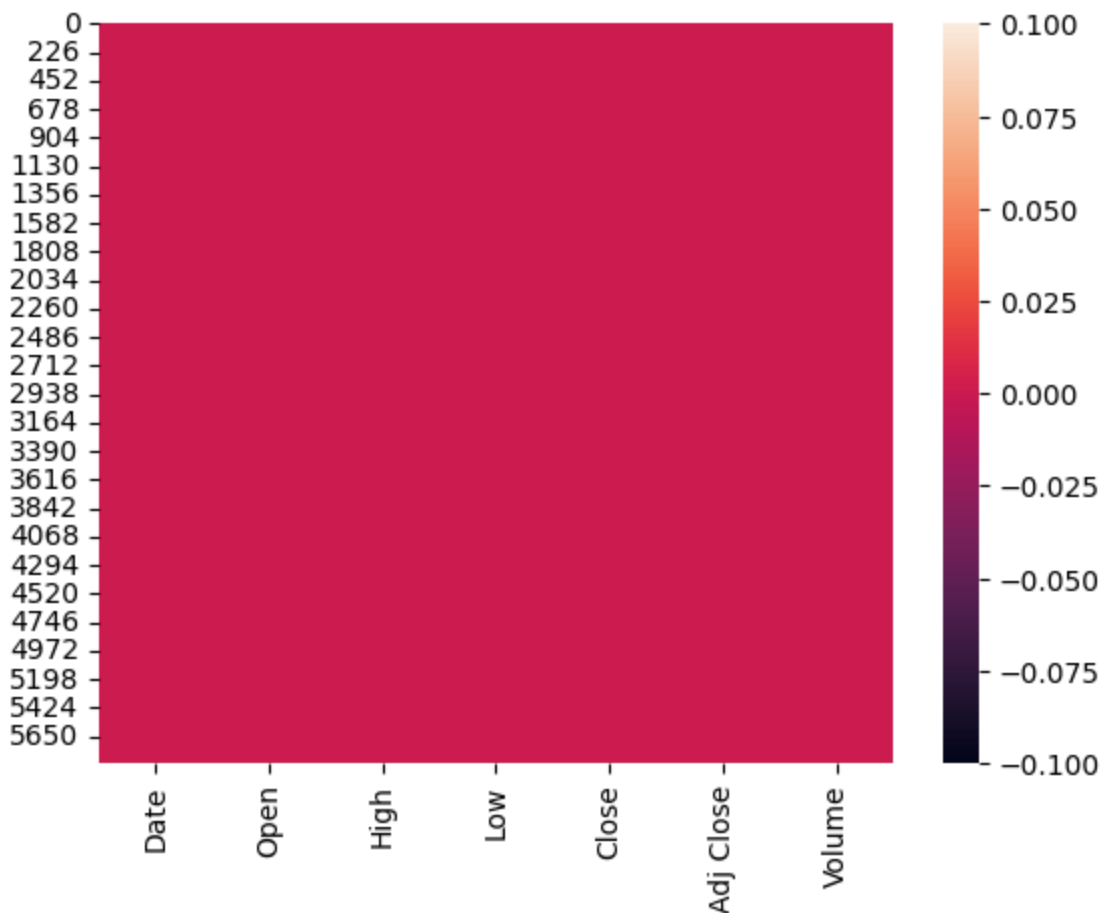

From this, we can see that our dataset contains no NULL values or missing data in any of the columns. Therefore, no altering of the data points was necessary.

Had we faced a problem with missing data, we would have created a series of impute functions to impute values in each of these columns based on numbers from the 5 number summary. For each variable, we'd have created a boxplot and imputed missing values with a random number between the first quartile and the third quartile, as shown in the corresponding boxplot. We then applied each impute function to each of the columns in the dataset with missing values to complete the dataset.

Step 5: Standardization of Data

The next part of the preprocessing of the data was to standardize all of the data into z-scores. In Python, the Scipy.stats library contains a z-score module for converting arrays of values into their respective z-scores by calculating the means and standard deviations and each individual value. Using this function, we standardized every single factor in the dataset.

```python
df2["Open"] = stats.zscore(df["Open"])
df2["High"] = stats.zscore(df["High"])
df2["Low"] = stats.zscore(df["Low"])
df2["Close"] = stats.zscore(df["Close"])
df2["Adj Close"] = stats.zscore(df["Adj Close"])
df2["Volume"] = stats.zscore(df["Volume"])
df2["Date"] = pd.to_datetime(df["Date"])
df2["Date"] = (df2["Date"] - df2["Date"].min()) / np.timedelta64(1, "D")
df2 = df2.drop(columns="Date")
```

In this step, you can also see where we removed the "Date" column, even after converting the object to a datetime object and then into a number of days since passed integer value, which ended up being a counter variable and therefore extraneous and a hindrance to the actual clustering algorithm.

Here is the final dataset with the standardized data instead of the actual values. This is the dataset we use for the actual singular value decomposition. In total, we have these 9 columns and 3276 rows of data in the dataframe. The full dataset can be referenced here as a file in the Github repository.

df2

	Open float64 -1.5127263583659...	High float64 -1.5197469134365...	Low float64 -1.5850412799113...	Close float64 -1.5391419202792...	Adj Close float64 -1.3379293733614...	Volume float64 -0.808555838226...
0	-0.99449384653954509	-1.0102371593586328	-1.060216627295719	-1.0707779765798167	-1.0346280925879574	-0.5714129506972031
1	-1.0871792880014166	-1.0981718812670112	-1.1317717980787072	-1.1425239163203542	-1.0865192083719528	-0.6005089107315081
2	-1.1661634033342223	-1.1701184719193207	-1.1821856684030854	-1.1925042338924143	-1.1226684331860493	-0.6223184039304641
3	-1.1951779763136203	-1.136543396281576	-1.1789331606402222	-1.1489729895554588	-1.0911836855653179	-0.5676699026653457
4	-1.137954790715363	-1.1533309341004483	-1.1626706218259066	-1.1449423187835186	-1.0882686209013788	-0.6870648169970822
5	-1.1548799582866787	-1.1317469569047556	-1.1382768136044334	-1.1215644283062647	-1.0713604507314254	-0.7410146159629206
6	-1.1129700195386594	-1.11416001252308	-1.1016861012722234	-1.1159214892255482	-1.0672789187477698	-0.7400996486662443
7	-1.124253464586203	-1.1173576387742936	-1.1171355131458234	-1.1215644283062647	-1.0713604507314254	-0.722216196958481
8	-1.1113580988175817	-1.1285493306535417	-1.1236405286715494	-1.1199521599974884	-1.0701937952920466	-0.7563361592399903
9	-1.1065223366543486	-1.1053665403322421	-1.0919285779836343	-1.1054417452185035	-1.0596991260642488	-0.7736373590316868

Data Mining Algorithms and Procedure

For this analysis, we used the Singular Value Decomposition Algorithm to refactor our data and mine the data to find new clusters. In addition to the Singular Value Decomposition, we utilized a K-Means Clustering Algorithm with a number of different clusters to cluster our data and search for different trends between the axes.

Singular Value Decomposition (SVD)

We started with the Singular Value Decomposition of our original matrix. Singular value decomposition is a matrix decomposition algorithm that changes the number of factors in the data to scale it and retain variability by altering the factors and corroborating different features.

Singular decomposition analysis(SVD)

$$C_{m \times n} = U_{m \times r} \times \Sigma_{r \times r} \times V^1_{r \times n}$$

Source: https://www.geeksforgeeks.org/singular-value-decomposition-svd/

The singular value decomposition works by making a N-dimensional scatter plot and finding multi-dimensional trend lines that keep the natural variability of the z-scores centered around 0 in order to find the optimum number of clusters based on the spread of the singular values of the dataset given.

For this, we looked at a number of different Python modules for Singular Value Decomposition, namely Numpy, Scipy, and Sci-kit Learn's Truncated SVD. Sci-kit Learn's Truncated SVD does not provide the individual matrices, however, it gave us insight into how to truncate our values correctly. In the end, we used the Scipy.linalg SVD function for creating our matrices.

```python
from scipy.linalg import svd
U, s, vt = svd(df2)
```

After creating our matrices, we printed each out to verify the proper placement. All of the matrices are included in the Github repository in full.

U

```
array([[-1.30950220e-02, -1.04628607e-02, -2.35254704e-03, ...,
         2.77642768e-02,  2.81935256e-02,  2.85975210e-02],
       [-1.40538172e-02, -1.10672410e-02,  1.20686628e-03, ...,
         5.88414644e-03,  7.77779654e-03,  1.28932224e-03],
       [-1.47874949e-02, -1.15258033e-02,  4.51744566e-03, ...,
         1.25556362e-02,  1.61758540e-02,  3.72223645e-03],

       ...,

       [ 2.82318347e-02,  2.16091592e-04,  3.33817192e-02, ...,
         9.97730666e-01, -2.17566400e-03, -2.20721846e-03],
       [ 2.86215393e-02, -6.04638875e-04,  3.25425607e-02, ...,
        -2.18444211e-03,  9.97798774e-01, -2.01980351e-03],
       [ 2.91040494e-02, -3.65790371e-04,  3.24949841e-02, ...,
        -2.20124578e-03, -2.00652480e-03,  9.97704506e-01]])
```

s

```
array([171.66064834,  75.18490959,   7.14495136,   2.03741497,
         1.45961466,   0.72063507])
```

vt

```
array([[ 4.45368186e-01,  4.45280535e-01,  4.45615630e-01,
         4.45455165e-01,  4.44336224e-01, -9.45206319e-02],
       [ 4.72004130e-02,  5.50255222e-02,  3.55841878e-02,
         4.57893393e-02,  2.76789000e-02,  9.95296586e-01],
       [-2.54607370e-01, -2.01143008e-01, -2.49410310e-01,
        -1.83040824e-01,  8.93734484e-01,  1.56781070e-02],
       [ 6.98656936e-01,  6.49990955e-02, -1.00973429e-01,
        -7.04093387e-01,  4.13154401e-02, -1.87285142e-03],
       [-5.65777595e-02,  6.92565968e-01, -7.09988468e-01,
         1.07531556e-01, -3.61114118e-02, -1.41648453e-02],
       [ 4.93225942e-01, -5.23799987e-01, -4.72943669e-01,
         5.08584857e-01, -5.18364580e-03, -7.76724226e-04]])
```

In the analysis section, we show how we altered the matrices and interpreted them.

K-Means Clustering Algorithm

On top of this, we used a K-Means clustering algorithm as our clustering algorithm. KMeans is a Sci-kit Learn module used for partitioning scattered data into k clusters, each with a respective centroid or mean value. This clustering algorithm provides a clear set of clusters and a set of labels that we can then put back into the initial dataset. We will go into further detail on how we implemented this with our analysis section.

Unlabelled Data   **Labelled Clusters**

K-means

X = Centroid
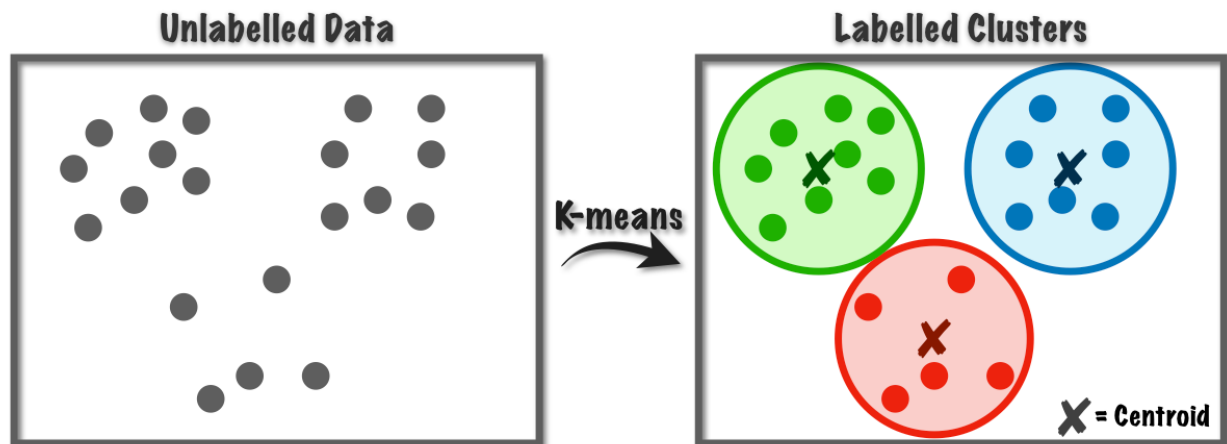
Source:

Results and Analysis

First, we examine the singular value decomposition matrices produced. The first thing we need to do is compute the eigenvalues and plot those on a line plot in order to find the optimal number of clusters as suggested by our SVD. This number will help us truncate the U matrix, determine the variability retained by the SVD, and select the vectors from the V^T matrix to interpret.

Eigenvalue Plot

The first step is squaring the singular values, which gives us the eigenvalues in an array of 6 numbers.

```
s2 = s*s
s2
```

```
array([2.94673782e+04, 5.65277063e+03, 5.10503299e+01, 4.15105977e+00,
       2.13047496e+00, 5.19314902e-01])
```

Then, we create a list of integers 1-6 to use as the pointer values for each of these eigenvalues for the plot.
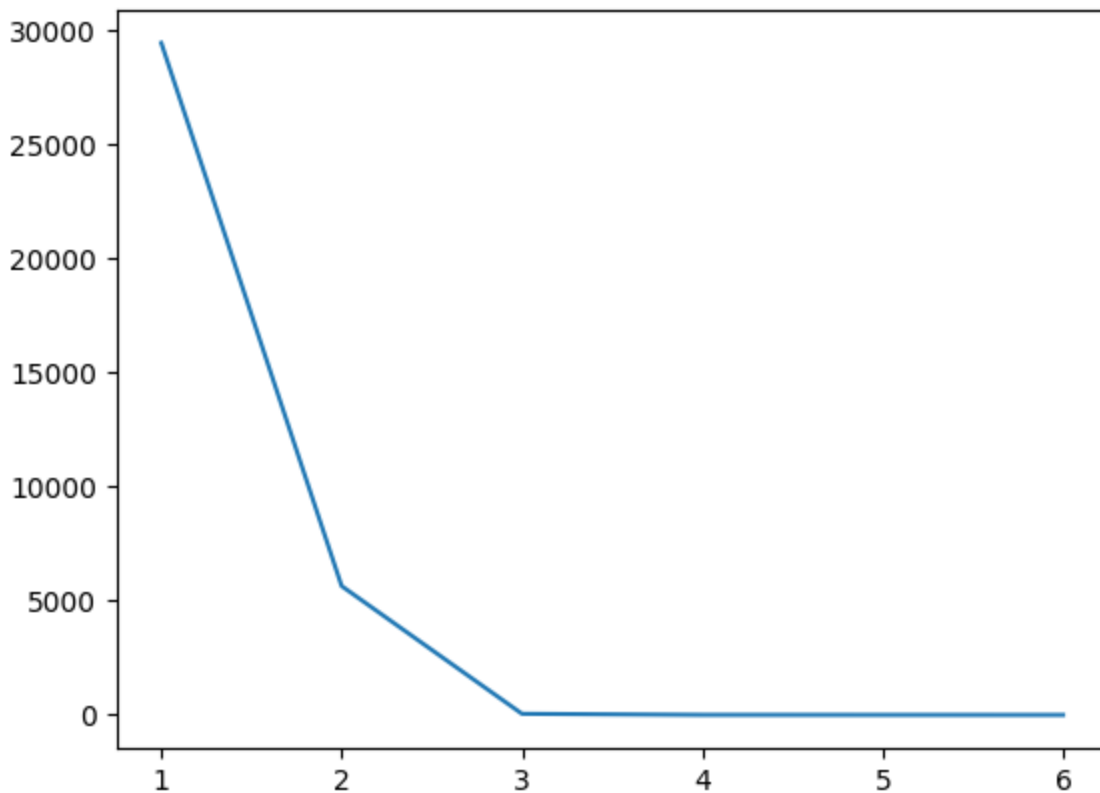
```
nums = [i for i in range(1, 7)]
nums
```

```
[1, 2, 3, 4, 5, 6]
```

Finally, using Matplotlib.pyplot, we are able to plot the eigenvalues and examine the plot. We look for the elbow of the data, which will indicate where most of the variability would be retained reasonably in a good number of clusters.

```
plt.plot(nums, s2)
```

```
[<matplotlib.lines.Line2D at 0x7f0a9ff79c10>]
```

This eigenvalue plot shows that the elbow is around 2 eigenvalues, which is what we use as our parameter for each of the functions explained above.

New Vectors Componentized from Original Vectors

Using 2 as our parameter, we truncate the V^T matrix into only the first 2 rows of the 6 columns, each of which contain the vector quantity corresponding to each variable for the component.

```
vt_chosen = vt[0:2]
vt_chosen
```

```
array([[ 0.44536819,  0.44528053,  0.44561563,  0.44545517,  0.44433622,
        -0.09452063],
       [ 0.04720041,  0.05502552,  0.03558419,  0.04578934,  0.0276789 ,
         0.99529659]])
```

Interpretation

Each of the values in this V^T matrix corresponds to how much of the feature from the original dataset went into that component. This warranted some very interesting results as we can see that the first component has even proportions of the open, low, high, close, and adjusted close prices and a negative correlation to the volume of shares bought. For the second component, we see that the proportion of the component made up of volume is significantly greater than the other components as the open, low, and close prices have a little bit of impact, still miniscule to that of the volume. Essentially, we observe that our 2 components are made up of all of the price metrics and then the volume metric, meaning that the clusters will likely follow this same pattern. This shows that the low, open, close, and high prices are all correlated, which is an expected and not that significant discovery.

- Each of these corresponds to how much of that factor was maintained in the new component that the SVD created smth like that

Variability Retained

Using the value of 2 singular values, we can calculate the proportion of the **variability retained** by summing the first X eigenvalues, which are the singular values squared, and dividing that by the total of all of the eigenvalues.

```
variability = sum(s2[0:2]) / sum(s2)
print(variability)
```

✅

```
0.9983554727523813
```

Variability at = 0.9983554727523813 ~ 99.8%
This is a very interesting discovery, as it seems that, after only 2 of the SVD components being considered, almost all of the variability in the dataset is retained, meaning that there is some very strong linear correlation between the two components, which can be interpreted in part of our conclusion.

Truncated U-Matrix

For the clustering algorithm, we use the U matrix produced by the SVD as our new data points. However, the current U matrix has a shape of

```
U.shape
```
✔

```
(5863, 5863)
```

Therefore, we need to truncate the columns so there are only 2 columns of data points, representing the 2 dimensions of data that we are clustering in terms of now. Therefore, we use Python's matrix properties to truncate the matrix and reshape it.

```
U = U[:,0:2]
U
```
✔

```
array([[-0.01309502, -0.01046286],
       [-0.01405382, -0.01106724],
       [-0.01478749, -0.0115258 ],

       ...,

       [ 0.02823183,  0.00021609],
       [ 0.02862154, -0.00060464],
       [ 0.02910405, -0.00036579]])
```

After truncating the matrix, we check the shape to verify that we now have 2 columns of plottable data points.

```
U.shape
```
✔

```
(5863, 2)
```

Now, we have a valid U matrix that can be used for plotting our data points for our K-Means clustering algorithm.

Preliminary Attempt at K-Means Clustering

We first looked at the K-Means clustering algorithm in an attempt to see if we could force 2 clusters and demonstrate the linear trend specifically.

```
from sklearn.cluster import KMeans
from sklearn import metrics

km = KMeans(n_clusters=2)
```

After creating this model, we fit it for predictions to the U matrix using the KMeans function of .fit_predict(), which provides the labels to the data points.

```
label = km.fit_predict(U)
label
```
✅
```
array([0, 0, 0, ..., 1, 1, 1], dtype=int32)
```

We checked the length of this array to see if it truly was the labels for every single point in the original dataset, which it was.

```
len(label)
```
✅
```
5863
```

After fitting the K-Means clustering algorithm to the truncated U matrix, we calculated the silhouette score of the model to see if it was relatively high scoring on the labels. The silhouette score is a metric comparable to the R-squared value exhibited usually in linear regression trends as it represents the goodness of the clustering algorithm. If the value is closer to -1 or 1, it indicates a strong positive or negative correlation and it means the clusters are very uniquely independent and distinguishable from each other.

```
kms = metrics.silhouette_score(U, km.labels_)
kms
```
✅
```
0.397645419496646
```

This silhouette score is relatively low and we could do a better job of creating silhouettes. Therefore, this was only our preliminary attempt. We still created a plain scatter plot of the two label values, 0 and 1. These two scatter plots are *not* on the same scale as they are plain scatter plots to observe the variation of the data of these two components, not to observe the difference and compare between the two clusters.

```
label0 = U[label==0]
plt.scatter(label0[:,0], label0[:,1])
plt.show()
```
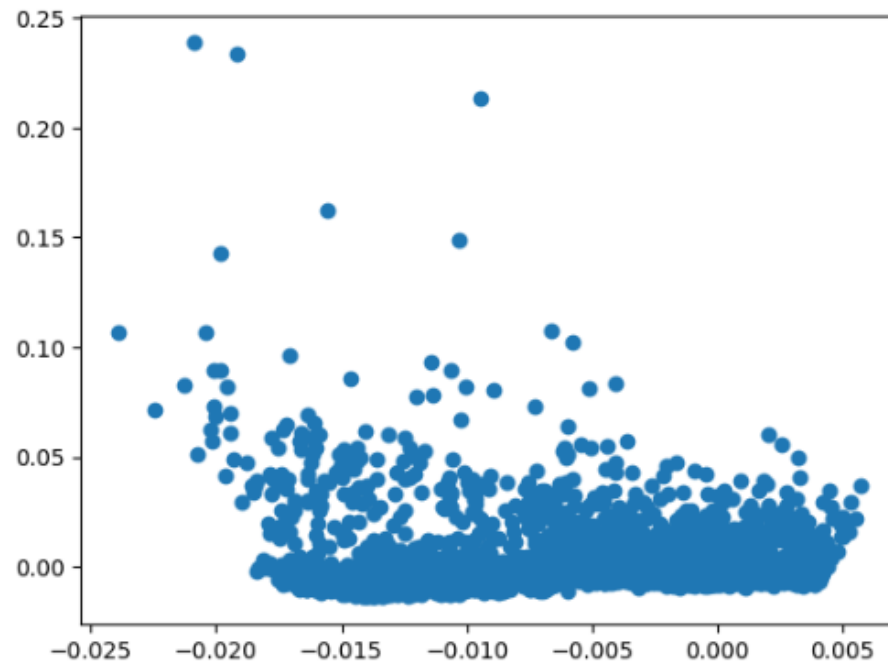


```
label1 = U[label==1]
plt.scatter(label1[:,0], label1[:,1])
plt.show()
```
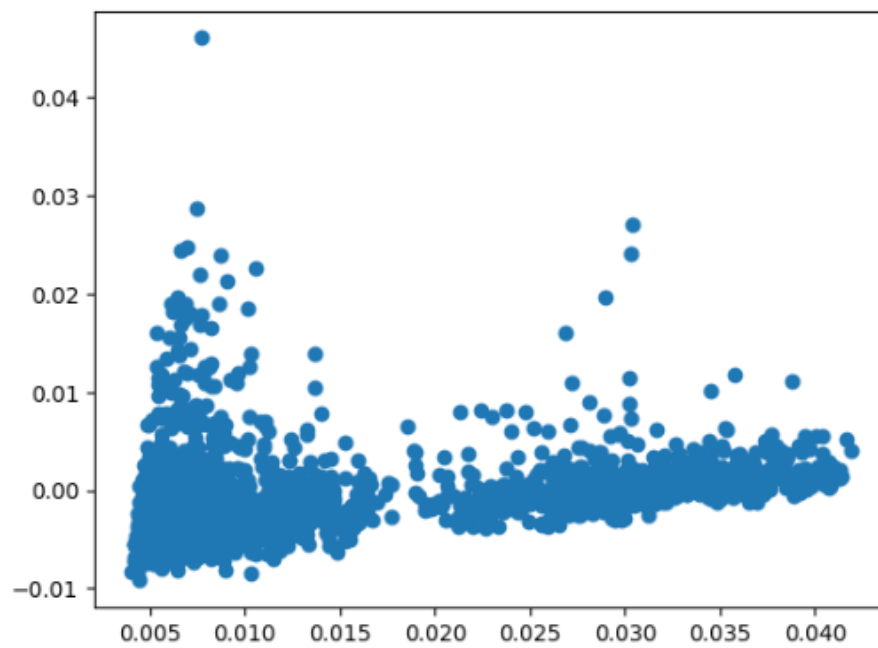
Then, we graphed the scatter plot of the clusters and observed the difference in the clusters.

```
for i in ulabels:
    plt.scatter(U[label == i , 0] , U[label == i , 1] , label = i)
plt.legend()
plt.show()
```
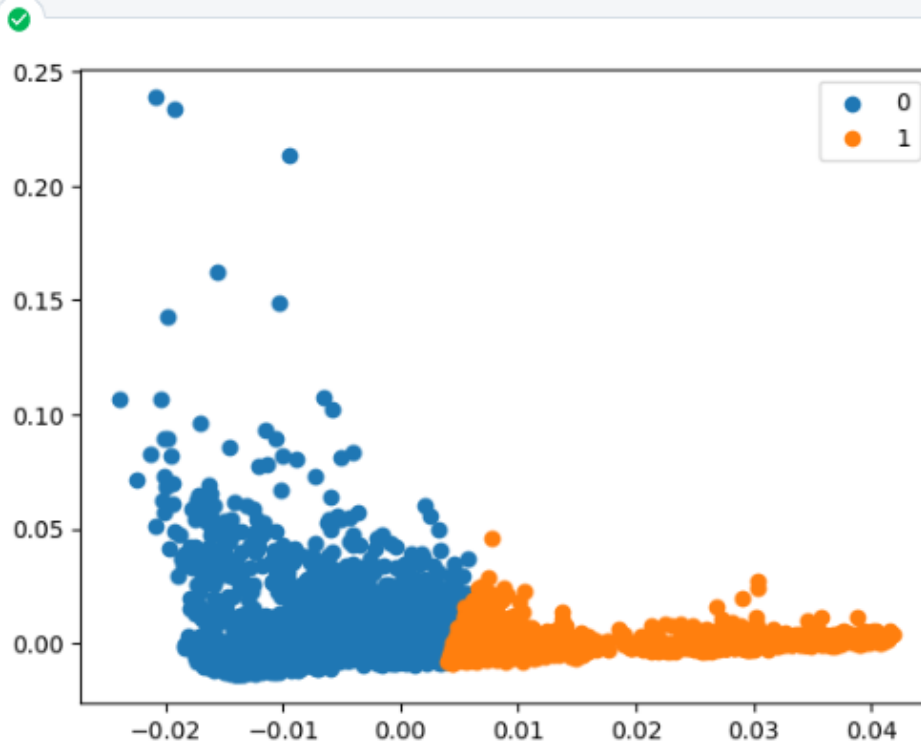

This concludes the preliminary attempt of computing silhouette scores and creating cluster plots. Based on what we received from this, we wanted to test for a range of different k_cluster values, specifically from 2 to 5. We also wanted to create silhouette plots in this portion of the analysis. Lastly, we wanted to add the centroids to the cluster plots to be able to see the differences in the average values of each of the clusters.

K-Means Analysis with Silhouettes and Iteration

For the K-Means analysis, we started off by creating the iterative function for 4 different k_clusters values that we wanted to test. Then, we created a brand new KMeans algorithm each time and re-fit it to the U matrix each time. We also computed the centroids using the cluster_centers_ function in Sci-kit Learn's KMeans module. Using these values, we then computed the unique silhouette samples to use in our

silhouette plots and clustered the y_pred values on each of these silhouette values using another iteration. After this, we used Matplotlib again to plot the silhouette plots. We also computed the average silhouette score for each of the 4 iterations to determine which controlled variability and showed the strongest correlation in it. Here is the entire iterative analysis for the K-Means algorithm:

```python
labelslabels = []

for i, k in enumerate([2, 3, 4, 5]):

    fig, ax = plt.subplots(1, 2, figsize=(15, 5))

    # Run the kmeans algorithm
    kmkm = KMeans(n_clusters=k)
    y_pred = kmkm.fit_predict(U)
    centroids = kmkm.cluster_centers_
    labelslabels.append(y_pred)
    # get silhouette
    silhouette_vals = metrics.silhouette_samples(df2, y_pred)
    # silhouette plot
    y_ticks = []
    y_lower = y_upper = 0
    for i, cluster in enumerate(np.unique(y_pred)):
        cluster_silhouette_vals = silhouette_vals[y_pred == cluster]
        cluster_silhouette_vals.sort()
        y_upper += len(cluster_silhouette_vals)

        ax[0].barh(range(y_lower, y_upper), cluster_silhouette_vals, height=1)
        ax[0].text(-0.03, (y_lower + y_upper) / 2, str(i + 1))
        y_lower += len(cluster_silhouette_vals)

        # Get the average silhouette score
        avg_score = np.mean(silhouette_vals)
        print(avg_score)
        ax[0].axvline(avg_score, linestyle="--", linewidth=2, color="green")
        ax[0].set_yticks([])
```

```
        ax[0].set_xlim([-0.1, 1])
        ax[0].set_xlabel("Silhouette coefficient values")
        ax[0].set_ylabel("Cluster labels")
        ax[0].set_title("Silhouette plot for the various clusters")
        # scatter plot of data colored with labels
        ax[1].scatter(U[:,0], U[:,1], c=y_pred)
        ax[1].scatter(centroids[:, 0], centroids[:, 1], marker="*", c="r",
s=250)
        ax[1].set_xlabel("component 1")
        ax[1].set_ylabel("Component 2")
        ax[1].set_title("Visualization of clustered data", y=1.02)

        plt.tight_layout()
        plt.suptitle(
            f" Silhouette analysis using k = {k}", fontsize=16,
fontweight="semibold"
        )
        plt.savefig(f"money_Silhouette_analysis_{k}.jpg")
```

This produced the following results:

Table of k_cluster values and their respective average silhouette scores

k_clusters	Average silhouette score
2	0.39766590180066763
3	0.47791736153949804
4	0.4829869133840531
5	0.5146515822137643

Silhouette Analyses Plots and Clustered Scatter Plots

Silhouette analysis using k = 2
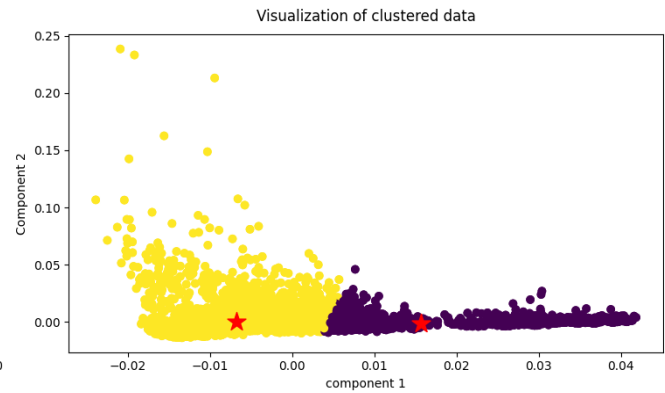
Silhouette analysis using k = 3
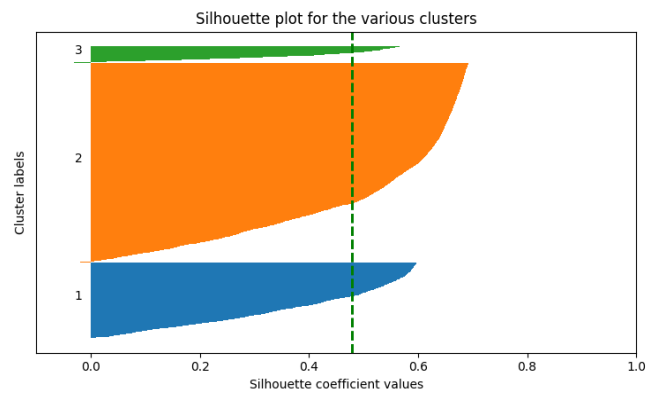
Silhouette analysis using k = 4

Silhouette plot for the various clusters

Visualization of clustered data

Silhouette analysis using k = 5

Silhouette plot for the various clusters

Visualization of clustered data
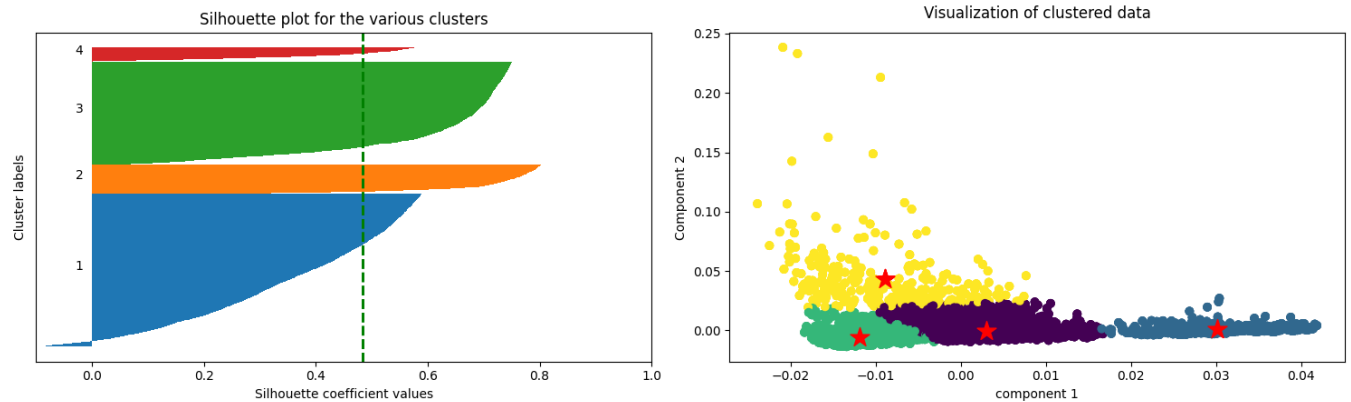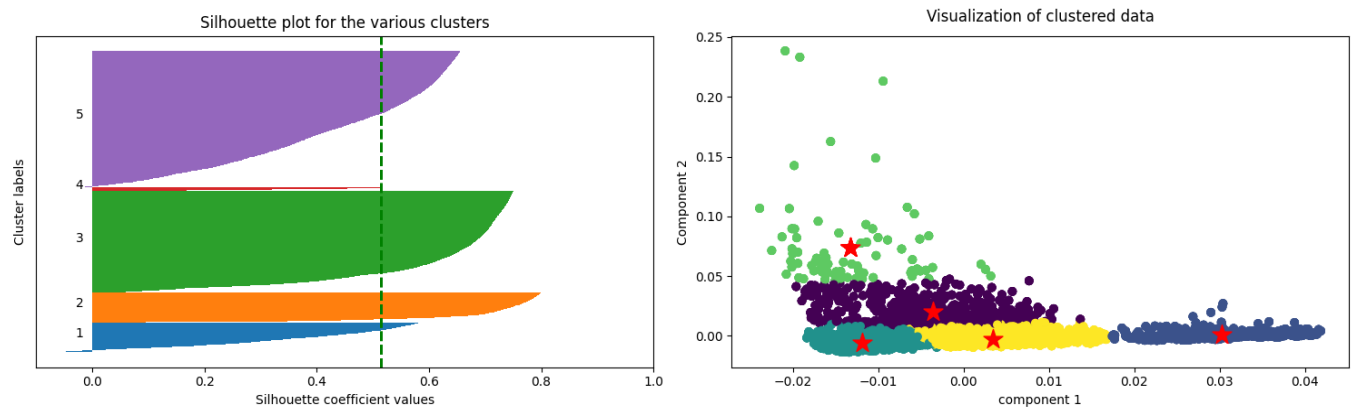
Interpretation of the Clusters

So, now we can start viewing these clusters and analyzing which size of clusters clustered the best and what the different clusters actually represent. For interpreting silhouette plots, there are two main criteria that we examine. The first is that all of the silhouettes of the clusters pass the average silhouette score threshold. This verifies that the clusters are all valid and actually do contain a trend that is correspondent with the other trends visible in the other clusters, ensuring that the values placed in the same cluster do actually exhibit the same tendencies. The second is that the c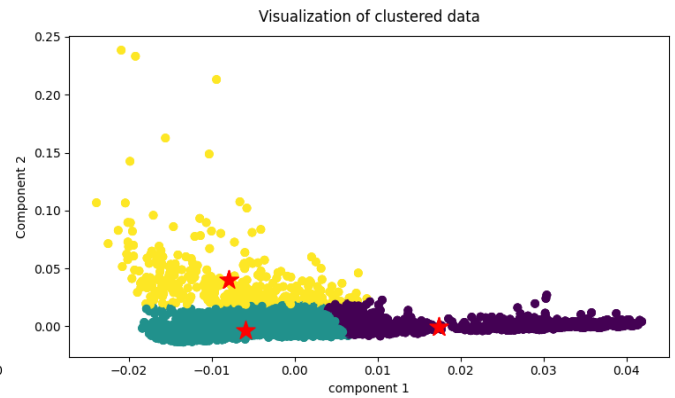lusters themselves are not ridiculously fluctuating in size. This condition allows us to see that the clusters are not abnormally sized and that they all do contain relatively similar amounts of data, however, this is not an end-all-be-all criteria like the first one is, rather a guiding principle for the selection of the optimal n_cluster count.

Clustering Tendency

The clustering tendency of this K-Means clustering algorithm for the Goldman Sachs dataset is slightly weak but still does have some value. In the cluster plots, no matter the number of clusters used, we can see the clear definition of the clusters and the distinction between each of them. However, it is weak because of the differences in the sizes as seen in the silhouette plots and due to the mediocre average silhouette scores. The silhouette scores actually give an interesting interpretation as the n_clusters=5 has the greatest silhouette score, but this is actually not a true representation of the correct clusters because one cluster, cluster number 4 (the green color in the scatter plot), contains very little data and has an abnormally small size along with a very low correlation that doesn't meet this average threshold. It may seem like the strength of the other 4 clusters counteracts this, however, this is not the case because the 4th cluster fails to meet either set of criterion for optimal clustering count.

Overall, we can say that the clustering tendency is weak and that the stock price characteristics of the Goldman Sachs stock has clusters, but these clusters are not particularly meaningful.

Silhouette analysis using k = 4
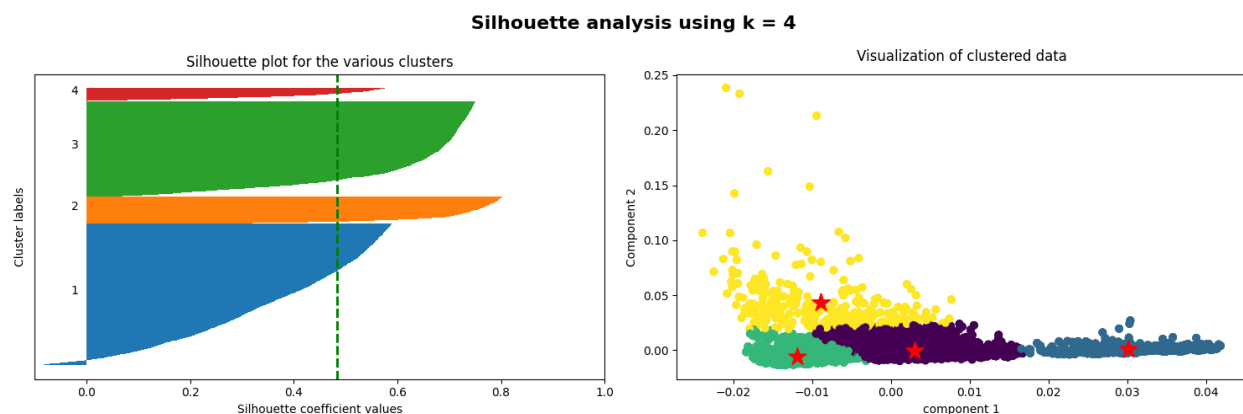
Optimal Cluster Amount

Based on the criteria listed earlier, it is reasonable to conclude that n_clusters=4 is a feasible estimate for the optimal number of clusters based on the silhouette plots. The silhouette plot from these values has all 4 of the clusters passing the average silhouette score threshold and they have relatively even sizes, even though there are two that are noticeably larger than the other 2 clusters, these clusters balance each other out. The cluster plot for this amount also indicates specifically distinct centroids

and there are clear boundaries between each cluster, further adding to how this amount of clusters is the most optimal for our purpose.

Cluster Labels

The clusters are as follows:
1. Cluster 1 in the silhouette plot is cluster 0 and is the green colored cluster.
2. Cluster 2 in the silhouette plot is cluster 1 and is the purple colored cluster.
3. Cluster 3 in the silhouette plot is cluster 2 and is the blue colored cluster.
4. Cluster 4 in the silhouette plot is cluster 3 and is the yellow colored cluster.

Final Actual Interpretation of the Clusters

The scatter plots are the price metrics versus the volume metrics plotted against each other. However, there is still no significant clustering trend or meaningful clusters with differences that we can see. The singular value decomposition created 2 components based on the V^T vector components. Component 1, which was on every factor evenly except volume, showed that there were 3 different clusters that cut at certain points on this scale. As for Component 2, which was primarily based on the volume and very little of the low and high prices, those 3 clusters all had relatively similar values for this component, but there was one cluster, the yellow cluster, that had higher values of the component, indicating very high volumes for those values and relatively low price metrics.

Defining Characteristics of the Clusters

To determine the defining characteristics, we took our clusters and plugged them back into our original DataFrame containing the original standardized dataset, before it was truncated and decomposed by the SVD. This started with taking the "labelslabels" variable, which was a 2-dimensional array containing the long list of labels for each n_cluster value, and extracting each set of labels for each n_cluster.

```
labelslabels
```
✓

```
[array([0, 0, 0, ..., 1, 1, 1], dtype=int32),
 array([1, 1, 1, ..., 0, 0, 0], dtype=int32),
 array([0, 0, 0, ..., 3, 3, 3], dtype=int32),
 array([2, 2, 2, ..., 0, 0, 0], dtype=int32)]
```

```
setlabels2 = labelslabels[0]
setlabels3 = labelslabels[1]
setlabels4 = labelslabels[2]
setlabels5 = labelslabels[3]
```
✓

After taking these labels out, we used the setlabels4 and created a new DataFrame and added this as a column to the DataFrame, with each label corresponding to the cluster that that row of data, or that data point, falls into.

```
df2c = df3c = df4c = df5c = df2
```
✓

[60]
```
df4c["Clusters"] = setlabels4
df4c
```
✓

~ Visualize

Open float64	High float64	Low float64	Close float64	Adj Close float64	Volume float64	Clusters int32
-1.5127263583659...	-1.5197469134365...	-1.5850412799113...	-1.5391419202792...	-1.3379293733614...	-0.808555838226...	0 - 3
-0.9944938465394509	-1.0102371593586328	-1.060216627295719	-1.0707779765798167	-1.0346280925879574	-0.5714129506972031	0
-1.0871792880014166	-1.0981718812670112	-1.1317717980787072	-1.1425239163203542	-1.0865192083719528	-0.6005089107315081	0
-1.1661634033342223	-1.1701184719193207	-1.1821856684030854	-1.1925042338924143	-1.1226684331860493	-0.6223184039304641	0
-1.1951779763136203	-1.1365433962815762	-1.1789331606402222	-1.1489729895554588	-1.0911836855653179	-0.5676699026653457	0
-1.1379547907153630	-1.1533309341004830	-1.1626706218259066	-1.1449423187835186	-1.0882686209013788	-0.6870648169970822	0
-1.1548799582866787	-1.1317469569047556	-1.1382768136044334	-1.1215644283062647	-1.0713604507314254	-0.7410146159629206	0
-1.1129700195386594	-1.1141600125230834	-1.1016861012722234	-1.1159214892255482	-1.0672789187477698	-0.7400996486662443	0
-1.1242534645862403	-1.1173576387742936	-1.1171355131458234	-1.1215644283062647	-1.0713604507314254	-0.7222161969584810	0
-1.1113580988175817	-1.1285493306535417	-1.1236405286715494	-1.1199521599974884	-1.0701937952920466	-0.7563361592399903	0
-1.1065223366543486	-1.1053665403322421	-1.0919285779836343	-1.1054417452185035	-1.0596991260642488	-0.7736373590316868	0

◄ ◄ ►

We then sorted the DataFrame based on the cluster values.

```
df4c.sort_values(by=['Clusters'])
```

Then, we extract a new DataFrame containing just the values for each of the clusters using the filtration habits of Pandas DataFrames.

```
df4c0 = df4c[df4c['Clusters'] == 0]
df4c1 = df4c[df4c['Clusters'] == 1]
df4c2 = df4c[df4c['Clusters'] == 2]
df4c3 = df4c[df4c['Clusters'] == 3]
```

From this, we can now take the average values and standard deviations of each attribute for each cluster and examine the defining characteristics of the clusters to see how these are distinguishable from each other. Using np.mean() and np.std() on the .iloc[index] DataFrames, we calculated the means and standard deviation for each attribute and compiled them in these tables here.

Average Value and St. Dev Tables for Each Cluster and Attribute

Code Used to Calculate Mean and Standard Deviation

```
print(np.mean(df4c0.iloc[0]))
print(np.std(df4c0.iloc[0]))
```

Cluster 0: Very low open, high, low, and close prices, very low volume
Cluster 0 is the green colored cluster.

Attribute	Average Value	Standard Deviation
Open	-0.8202523790083971	0.3717703196849677
High	-0.8780964289672786	0.3996373640883967
Low	-0.9222798020950795	0.4212579795441019
Close	-0.9026401587173629	0.4208528374982642
Adj Close	-0.9106045861890996	0.40351861432315467
Volume	-0.9084061748280682	0.39436806429465865

Cluster 1: Close to average open, high, low, and close prices, low volume

Cluster 1 is the purple colored cluster.

Attribute	Average Value	Standard Deviation
Open	0.05686153187916016	0.6021943193862952
High	-0.026578168246097964	0.4844506700061175
Low	-0.0659122980895379	0.46328174990516885
Close	-0.039053914948476355	0.4642611998828505
Adj Close	-0.06664210002697112	0.4436739815088534
Volume	-0.0858218313831 2025	0.4488745606293788

Cluster 2: Relatively high open and close prices, nearly average low and high prices, and high volume, lots of variation due to higher standard deviations
Cluster 2 is the blue colored cluster.

Attribute	Average Value	Standard Deviation
Open	0.5995156038927488	3.3370396865137213
High	-0.29209048235563045	1.5114634981560986
Low	0.24574090633654988	1.0287154251145136
Close	0.8501794168687707	0.8207872102089308
Adj Close	0.77714245243439	0.7104519564528767
Volume	1.1676893464397955	1.4581319047680812

Cluster 3: Very high open, high, low, and close prices, high volume
Cluster 3 is the yellow colored cluster.

Attribute	Average Value	Standard Deviation
Open	1.30334939023241	0.8650405842975702
High	1.2983382091604967	0.8997882827577454
Low	1.3100700186091931	0.8526035675128699
Close	1.483254646940889	0.7576597484284854
Adj Close	1.508580333625244	0.8608464085423041
Volume	1.4724686315354734	0.9182628335268447

Based on these values in the tables, we can now view a couple distinguishing factors between the different clusters. For the open price, it seems that cluster 2 has a standard deviation of over 3, so a wide variety of open prices, leaving open price as a factor that can't accurately distinguish that cluster from others. Additionally, we can see that the high price can be used as a distinguishing feature because cluster 3 contains a much higher average value than the other 3 clusters. Another feature that

seems to have a correlation is the adjusted close price, all of which have different values ranging from negative to positive and relatively lower standard deviations, even though they are not statistically different. Within each cluster, we see that the average values are close to each other and the differences between the first 5 attributes, excluding volume, between different clusters are very similar, which can also indicate the differences between these clusters. The main conclusion or interpretation we can see is that the volume is a major difference, which corresponds legitimately to the V^T matrix components as the 2nd component was primarily based on volume and none of the other factors, while the 1st component was primarily based on the other 5 factors. In our exploration of this dataset, we have found that the volume is not correlated with the other 5 factors reasonably, but the other 5 factors are all very similar in each cluster, meaning that they correlate with each other. This is expected as the high, low, open, and close prices of the stock are all going to be related to each other on each day and therefore will all affect the clusters in similar ways as they all have similar differences between each cluster or each unique data point.

It is important to note, however, that these trends are not statistically significant as the standard deviations of every attribute across every cluster are large and overlapping with one another because of a potential lack of general variability in the data that comes from the chronological dataset. However, we can be sure that volume is independent of the other 4 factors due to the way the SVD decided to cluster the components and then the way the KMeans showed a similar trend in clustering between the price factors and the volume.

Synthesis/Conclusions

Based on our analysis of the Goldman Sachs share prices over time, we can see that there are 4 major clusters of data that can be observed, each with their own corresponding characteristics, and that these characteristics are all unequivocally related to each other as well as they all have similar trends. The 4 clusters are as follows:

1. Cluster 0: Very low open, high, low, and close prices, very low volume
2. Cluster 1: Close to average open, high, low, and close prices, low volume
3. Cluster 2: Relatively high open and close prices, nearly average low and high prices, and high volume, lots of variation due to higher standard deviations
4. Cluster 3: Very high open, high, low, and close prices, high volume

These clusters are relatively weak as they each had mediocre silhouette scores, ranging from 0.39 to 0.51, as explained above in the Analysis section. This analysis allows us to examine the new way to cluster the share prices and relate certain share prices with one another on a scale less than 6 dimensions. We can apply this knowledge into the market by looking at the share price metrics vs the volume of the market to classify the market's volatility and riskiness and its relative reward. If the share price metrics are low and the volume is high, the market is classified as highly risky and volatile; however, if the share price metrics are high and the volume is low, the market is classified as low risk, low reward, and not very volatile.

Through our analysis, we were not able to find significant clustering associations between the volume and the share price metrics, which is a conclusion as it shows that there may not be any significance between the volume of the market and the respective share price. However, this connects to certain limitations, which will be explained in the next section.

Research Questions

1. How can the patterns and relationships between variables (open, high, low, close, adj close, volume) aid stock market analysis and prediction?
   a. **The observed patterns between open, high, low, close, and adj close can aid in determining the relative reward of betting on the stock and**

the volume can aid in determining the volatility, however, these two are independent.

2. What is the association between the variables (open, high, low, close, adj close, volume)?

   a. **There is an association found through the clustering between open, high, low, close, and adj close prices and there are no other clustering differences found in the data.**

Most of our conclusions can be drawn from our analysis of the clusters and from our SVD. Our SVD technically predicted the number of clusters to be used, however, it was an interesting conclusion due to the sheer amount of variability that was captured in simply 2 components initially. The eigenvalue plot shows us that there are only 2 major components that correlate to clusters in stocks, those being the share price metrics, as mentioned several times before, and the volume of shares.

Limitations and avenues for further research

Throughout our project, we faced a number of limitations regarding our dataset. For starters, the dataset does not contain the necessary randomness that is needed in a dataset and was rather a chronological ordered dataset, which proved to be a factor in the highly direct correlation between all of the share price metrics as they were all connected over time due to the inability of the stock market to have undefined jumps in the share prices. Additionally, the dataset was limited to only the share prices for Goldman Sachs, a major investment bank part of the New York Stock Exchange. This means our analysis is not properly applicable to any other stocks or the market as a whole and we cannot generalize our cluster findings to other stocks without further analysis. We also did not factor in outliers, which could have proved to be a limitation, however, these only seemed to be outliers because of the disparity between the volume and the price metrics, and they cannot be reasonably considered as outliers due to the chronological nature of the data. We can be sure of the strength of the price metrics as a way to see the connection between low price, high price, open price, close price, and adjusted close price, but they cannot be used as a part of the volume because of the outliers that were not understood. The fact that this data is chronological makes it quite difficult to reasonably conclude that our results are significant, especially when we look at the standard deviations, which lead us further toward this same interpretation. Therefore, these all played roles in limiting our ability to gain significant and strong results or clusters. This does not fully question our results in some ways, but it also leaves a wide field for future exploration and research with new data and similar metrics, perhaps with more data points and more features to see if there are more than just these 2 components that actually mattered.
Our data mining analysis allows us to also explore the uses of SVD as a way to refactor the data and find the optimal number of dimensions in a clustering algorithm. In contrast to the classification models that I am more familiar with, this experience allowed me to learn more about clustering algorithms and regression data and how dimensions and clustering works on a large scale. This is applicable to the real world as more valid quantitative data can be gathered, such as this stock data. This knowledge is valuable to me specifically because data science and analytics is what I would like to do in the future, and this project sets me up well for future research

projects in the discipline. Overall, our analysis, despite its limitations, proved to be a valuable insight to have for future research.