Task 1: Loading Data

I read a CSV file named 'stock_data.csv' into a pandas DataFrame, renaming its columns to match the specified format ('Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Name'). This prepares the dataset for further analysis, focusing on the Date, Close, and Name columns.

```
# Read the data from the csv file
stock_data = pd.read_csv('stock_data.csv')

# Name: The stock name
stock_data.rename(columns={'date': 'Date'}, inplace=True)
stock_data.rename(columns={'open': 'Open'}, inplace=True)
stock_data.rename(columns={'high': 'High'}, inplace=True)
stock_data.rename(columns={'low': 'Low'}, inplace=True)
stock_data.rename(columns={'close': 'Close'}, inplace=True)
stock_data.rename(columns={'volume': 'Volume'}, inplace=True)
stock_data.rename(columns={'Name': 'Name'}, inplace=True)
```

Task 2: Sorting Stock Names

Here, I extracted the unique stock names from the 'Name' column of the DataFrame and sorted them alphabetically. I then printed the total number of unique names and listed the first and last five names. This step helps identify the range of stocks in the dataset.

```
# Get the unique names of the stocks
names = stock_data['Name'].unique()

# Sort the names alphabetically
names.sort()

# Print the first five names
print('There are {} names in the data'.format(len(names)))
print('First 5 names: {}'.format(names[:5]))
print('Last 5 names: {}'.format(names[-5:]))
```

Output:

```
There are 505 names in the data

First 5 names: ['A' 'AAL' 'AAP' 'AAPL' 'ABBV']

Last 5 names: ['XYL' 'YUM' 'ZBH' 'ZION' 'ZTS']
```

Task 3: Filtering Stocks by Date

This task involved filtering out stocks that either started trading after July 1, 2014, or stopped trading before June 30, 2017. I achieved this by grouping the data by stock name and calculating the minimum and maximum dates for each stock. Stocks not meeting the date criteria were removed. This ensures that the remaining dataset only includes stocks with sufficient historical data for analysis.

```
df = stock_data.copy()
df_original = df.copy()
```

```
# convert the date column to datetime
df['first_date'] = pd.to_datetime(df['Date'])
df['last_date'] = pd.to_datetime(df['Date'])

# get the first and last date for each stock
df = df.groupby('Name').aggregate({'first_date': 'min', 'last_date':
'max'})

# remove stocks that do not have data for the entire period
remove_data = df[(df['first_date'] > '2014-07-01') | (df['last_date'] <
'2017-06-30')]
df = df.drop(remove_data.index)

# print the names of the removed stocks
print('Removed names: {}'.format(remove_data.index.values))

# print the number of names left
print('There are {} names left'.format(len(df)))

# get the names of the stocks that are left
df = df_original[df_original['Name'].isin(df.index.values)]
```

Output:

```
Removed names: ['APTV' 'BHF' 'BHGE' 'CFG' 'CSRA' 'DWDP' 'DXC' 'EVHC' 'FTV'
'HLT' 'HPE' 'HPQ' 'KHC' 'PYPL' 'QRVO' 'SYF' 'UA' 'WLTW' 'WRK']

There are 486 names left
```

Task 4: Identifying Common Dates

Here, I aimed to find common trading dates across the remaining stocks, focusing on the period between July 1, 2014, and June 30, 2017. I removed any dates outside this range and then identified dates when all stocks were traded. This step is crucial for comparative analysis across different stocks on the same dates.

```
# remove stocks that do not have data for the entire period
remove_date = df[(df['Date'] < '2014-07-01') | (df['Date'] > '2017-06-30')]
df = df.drop(remove_date.index)

# get the number of common dates
date_counts = df.groupby('Date')['Name'].nunique()
max_stock_count = df['Name'].nunique()

# get the stock names for the common dates
common_dates = date_counts[date_counts == max_stock_count].index

# get the first and last 5 common dates
count_common_dates = len(common_dates)
first_5_dates = common_dates[:5]
last_5_dates = common_dates[-5:]
```

```
# print the results
print("Number of common dates:", count_common_dates)
print("First 5 common dates:", first_5_dates.values)
print("Last 5 common dates:", last_5_dates.values)

# filter the data for the common dates
df = df[df['Date'].isin(common_dates)]
```

Output:

```
Number of common dates: 745

First 5 common dates: ['2014-07-01' '2014-07-03' '2014-07-07' '2014-07-08'
'2014-07-09']
Last 5 common dates: ['2017-06-26' '2017-06-27' '2017-06-28' '2017-06-29'
'2017-06-30']
```

Task 5: Creating a Pivot Table

I transformed the DataFrame into a pivot table, with stock names as columns, dates as rows, and closing prices as values. This structure is helpful for analyzing stock prices across different stocks and dates.

```
# Pivot the data so that the stock names are the columns, the dates are the rows, and the values are the closing prices
daily_close = df.pivot(index='Date', columns='Name', values='Close')

# Print the first and last five rows of daily_close
print(daily_close)
```

Output:

```
Name            A     AAL     AAP     AAPL   ABBV    ABC    ABT     ACN    ADBE     ADI    ADM  ...    XEL     XL   XLNX     XOM   XRAY    XRX    XYL    YUM     ZBH   ZION    ZTS
Date                                                                                          ...
                                                                                                        
2014-07-01  58.25  43.860  134.53   93.520  56.89  72.98  41.18   81.25   73.01  54.520  44.85  ...  32.04  33.05  48.16  101.36  47.51  49.52  39.13  81.54  104.89  29.50  32.51
2014-07-03  58.46  41.620  134.94   94.030  58.22  73.23  41.89   81.55   73.57  54.825  45.77  ...  31.53  33.16  49.12  102.59  47.83  49.80  39.02  82.49  105.13  29.77  32.91
2014-07-07  58.12  40.100  134.00   95.968  57.40  73.01  41.51   81.05   72.68  54.660  46.50  ...  31.65  33.25  48.79  102.65  47.81  48.84  38.00  82.29  104.48  29.77  32.56
2014-07-08  57.15  40.260  132.27   95.350  55.69  72.87  41.05   81.11   71.14  54.450  45.94  ...  31.61  33.02  48.57  102.83  47.49  48.80  37.36  82.30  103.36  29.39  32.48
2014-07-09  56.90  41.985  133.58   95.390  55.01  72.98  41.27   80.64   71.57  54.740  46.15  ...  31.64  33.39  48.82  103.55  47.58  49.88  37.54  83.23  103.79  29.74  32.46
...           ...     ...     ...      ...    ...    ...    ...     ...     ...     ...    ...  ...    ...    ...    ...     ...    ...    ...    ...    ...     ...    ...    ...
2017-06-26  59.24  48.780  121.79  145.820  72.74  95.99  48.91  122.34  144.96  80.570  41.30  ...  47.38  43.62  67.36   81.24  64.07  29.19  54.68  74.74  129.06  42.34  62.72
2017-06-27  58.88  48.500  121.96  143.730  72.39  95.53  48.67  122.19  142.54  78.140  41.00  ...  47.01  44.00  65.10   81.11  64.43  28.98  54.50  73.97  128.25  42.54  62.76
2017-06-28  59.40  49.250  116.71  145.830  72.92  96.38  48.48  123.74  143.81  79.200  41.14  ...  46.47  44.17  66.01   81.53  65.25  29.29  54.97  74.15  128.51  42.97  62.95
2017-06-29  58.80  49.620  116.05  143.680  72.48  95.76  48.73  122.99  141.24  77.740  40.63  ...  45.96  44.14  64.18   80.70  64.53  29.12  54.90  73.45  127.89  43.99  62.50
2017-06-30  59.31  50.320  116.59  144.020  72.51  94.53  48.61  123.68  141.44  77.800  41.38  ...  45.88  43.80  64.32   80.73  64.84  28.73  55.43  73.76  128.40  43.91  62.38

[745 rows x 486 columns]
```

Task 6: Calculating Returns

In this step, I calculated daily returns for each stock by comparing the closing price with the previous day's closing price. This new DataFrame has one less row than the previous one since the first date can't have a preceding date for comparison. Daily returns are a common measure in financial analysis.

```
# Calculate the daily percentage change for `daily_close`
daily_close_shift = daily_close.shift(1)
daily_close = (daily_close - daily_close_shift) / daily_close_shift

# Remove the NaN values from daily_close
df_percent_change = daily_close.dropna()

# Print the first five rows of df_percent_change
```

```python
print(df_percent_change)
```

Output:

```
Name               A       AAL       AAP      AAPL      ABBV       ABC       ABT       ACN  ...       XOM      XRAY       XRX       XYL       YUM       ZBH      ZION       ZTS
Date                                                                                        ...
2014-07-03  0.003605 -0.051072  0.003048  0.005453  0.023378  0.003426  0.017241  0.003692  ...  0.012135  0.006735  0.005654 -0.002811  0.011651  0.002288  0.009153  0.012304
2014-07-07 -0.005816 -0.036521 -0.006966  0.020610 -0.014085 -0.003004 -0.009071 -0.006131  ...  0.000585 -0.000418 -0.019277 -0.026140 -0.002425 -0.006183  0.000000 -0.010635
2014-07-08 -0.016690  0.003990 -0.012910 -0.006440 -0.029791 -0.001918 -0.011082  0.000740  ...  0.001754 -0.006693 -0.000819 -0.016842  0.000122 -0.010720 -0.012765 -0.002457
2014-07-09 -0.004374  0.042846  0.009904  0.000420 -0.012210  0.001510  0.005359 -0.005795  ...  0.007002  0.001895  0.022131  0.004818  0.011300  0.004160  0.011909 -0.000616
2014-07-10 -0.007206  0.019888 -0.003144 -0.003722  0.014179 -0.000822 -0.000727 -0.010789  ... -0.009464 -0.001681 -0.005613 -0.007991 -0.009131 -0.001156 -0.012441  0.001848
...              ...       ...       ...       ...       ...       ...       ...       ...  ...       ...       ...       ...       ...       ...       ...       ...       ...
2017-06-26 -0.008867  0.003085  0.017120 -0.003145  0.001377  0.001774 -0.003464 -0.003259  ... -0.004534  0.003603  0.026371  0.006442 -0.001069 -0.000232  0.009778 -0.000956
2017-06-27 -0.006077 -0.005740  0.001396 -0.014333 -0.004812 -0.004792 -0.004907 -0.001226  ... -0.001600  0.005619 -0.007194 -0.003292 -0.010302 -0.006276  0.004724  0.000638
2017-06-28  0.008832  0.015464 -0.043047  0.014611  0.007321  0.008898 -0.003904  0.012685  ...  0.005178  0.012727  0.010697  0.008624  0.002433  0.002027  0.010108  0.003027
2017-06-29 -0.010101  0.007513 -0.005655 -0.014743 -0.006034 -0.006433  0.005157 -0.006061  ... -0.010180 -0.011034 -0.005804 -0.001273 -0.009440 -0.004825  0.023737 -0.007149
2017-06-30  0.008673  0.014107  0.004653  0.002366  0.000414 -0.012845 -0.002463  0.005610  ...  0.000372  0.004804 -0.013393  0.009654  0.004221  0.003988 -0.001819 -0.001920

[744 rows x 486 columns]
```

Task 7: Principal Component Analysis (PCA)

Using the sklearn library, I performed PCA on the returns DataFrame. I extracted and printed the top five principal components based on their eigenvalues. PCA is used here to identify patterns and reduce the dimensionality of the dataset, focusing on the components that explain the most variance.

```python
# Create a PCA model with 20 components
pca = PCA()
pca.fit(df_percent_change)

# Get the eigenvalues (explained variance) and sort them in descending
order
eigenvalues = pca.explained_variance_
sorted_indices = eigenvalues.argsort()[::-1]

# Print the top five principal components according to their eigenvalues
print("Top 5 Principal Components (Ranked by Eigenvalue):")
for i in range(5):
    index = sorted_indices[i]
    print(f"PC {i + 1}: Eigenvalue = {eigenvalues[index]}")
```

Output:

```
Top 5 Principal Components (Ranked by Eigenvalue):

PC 1: Eigenvalue = 0.03979689799696103

PC 2: Eigenvalue = 0.008719696371567708

PC 3: Eigenvalue = 0.005061543903121703

PC 4: Eigenvalue = 0.0027954041820193874

PC 5: Eigenvalue = 0.002541362365751441
```

Task 8: Explained Variance Ratios

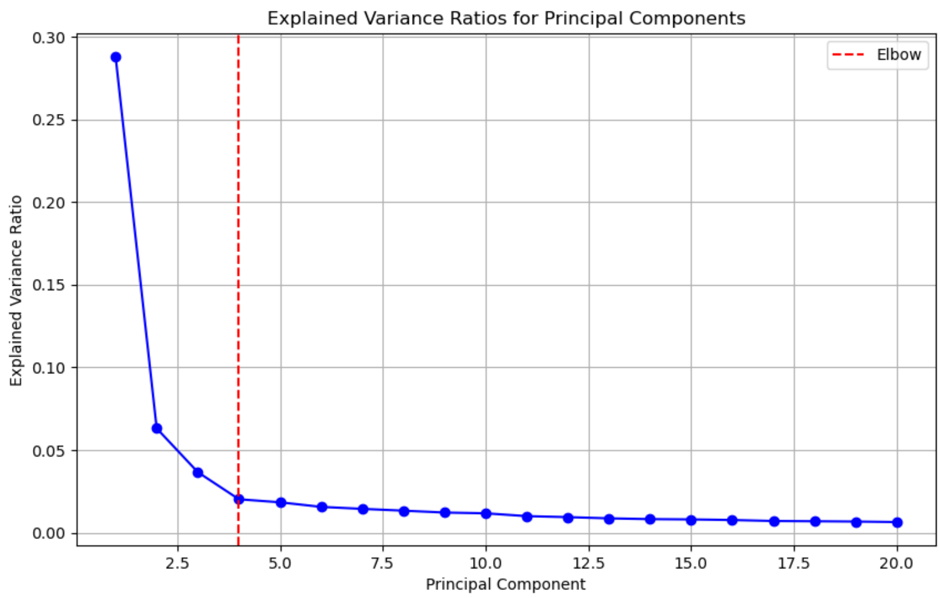
I plotted the explained variance ratios for the first 20 principal components. This shows how much variance each principal component accounts for. An 'elbow' in the plot was identified, indicating an optimal number of components. This step is important for understanding the effectiveness of PCA in reducing dimensions while retaining significant information.

```python
# Get the explained variance ratios from the PCA object
explained_variance_ratios = pca.explained_variance_ratio_
```

```
# Plot the explained variance ratios
plt.figure(figsize=(10, 6))
plt.plot(range(1, 21), explained_variance_ratios[:20], marker='o',
linestyle='-', color='b')
plt.title('Explained Variance Ratios for Principal Components')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.grid(True)

# Find the elbow point
kneedle = KneeLocator(range(1, 21), explained_variance_ratios[:20],
curve='convex', direction='decreasing')
plt.axvline(x=kneedle.knee, color='r', linestyle='--', label='Elbow')

# Show the plot
plt.legend()
plt.show()

# percentage of variance is explained by the first principal component
print('Percentage of variance is explained by the first principal
component: {}%'.format(explained_variance_ratios[0]*100))
```

Output:

```
Percentage of variance is explained by the first principal component:
28.810731163487752%
```

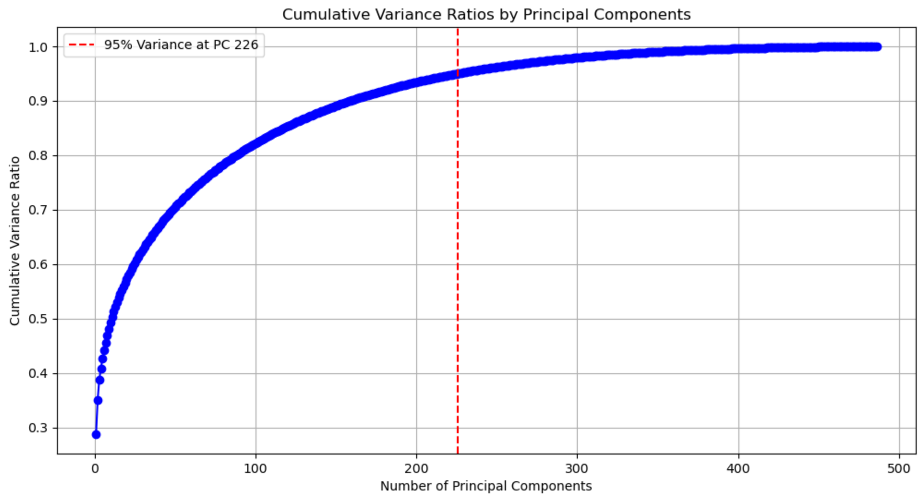


Task 9: Cumulative Variance Ratios

Here, I calculated and plotted the cumulative variance ratios. This helps determine how many principal components are needed to explain a certain percentage (e.g., 95%) of the variance in the dataset. This is a crucial step in deciding how many principal components to retain for further analysis.

```
# Calculate cumulative variance ratios
cumulative_variance_ratios = np.cumsum(explained_variance_ratios)

# Find the number of components needed to reach 95% cumulative variance
num_components_95 = np.where(cumulative_variance_ratios >= 0.95)[0][0] + 1

# Plot the cumulative variance ratios
plt.figure(figsize=(12, 6))
plt.plot(range(1, len(cumulative_variance_ratios) + 1),
cumulative_variance_ratios, marker='o', linestyle='-',
         color='b')
plt.title('Cumulative Variance Ratios by Principal Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Ratio')
plt.grid(True)

# Plot a vertical line at the number of components needed for 95% variance
plt.axvline(x=num_components_95, color='r', linestyle='--', label=f'95%
Variance at PC {num_components_95}')

# Show the plot
plt.legend()
plt.show()
```

Output:



Task 10: Normalization and PCA

Finally, I normalized the returns DataFrame so that each column has zero mean and unit variance. I then repeated the PCA process and the steps for plotting explained and cumulative variance ratios. Normalization is often performed before PCA to ensure that all variables are on the same scale, especially when they have different units of measurement.

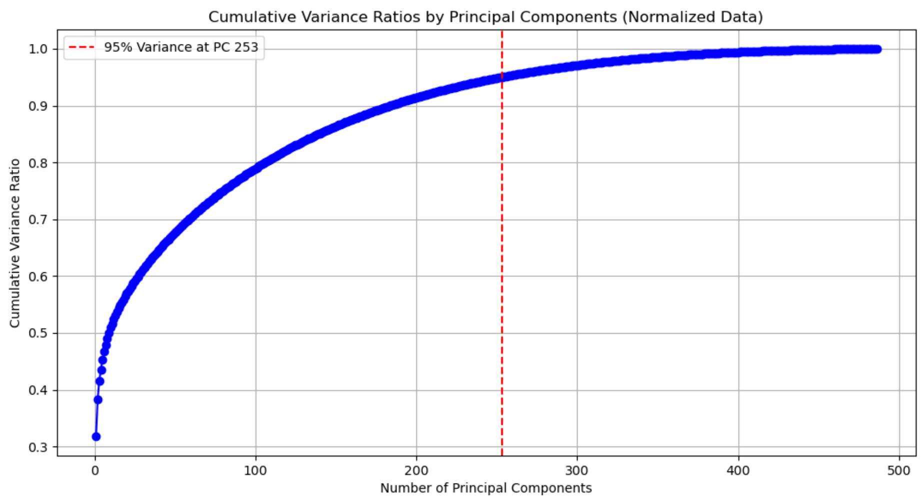
```
# standardize the data
scaler = StandardScaler()
```

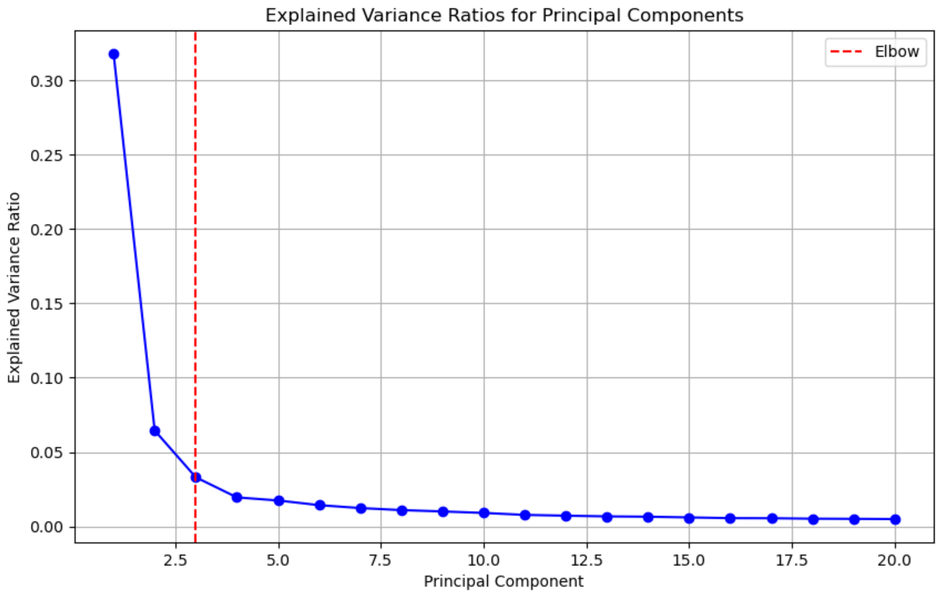
```
normalized_data = scaler.fit_transform(df_percent_change)
normalized_df = pd.DataFrame(normalized_data,
columns=df_percent_change.columns)

# Create a PCA model and fit it with the standardized data
pca_normalized = PCA()
pca_normalized.fit(normalized_df)

# Get the eigenvalues (explained variance) and sort them in descending
order
eigenvalues_normalized = pca_normalized.explained_variance_

# Print the top five principal components according to their eigenvalues
print("Top 5 Principal Components (Ranked by Eigenvalue):")
for i in range(5):
    print(f"PC {i+1}: Eigenvalue = {eigenvalues_normalized[i]}")

# Get the explained variance ratios from the PCA object
explained_variance_ratios_normalized =
pca_normalized.explained_variance_ratio_

# Plot the explained variance ratios
plt.figure(figsize=(10, 6))
plt.plot(range(1, 21), explained_variance_ratios_normalized[:20],
marker='o', linestyle='-', color='b')
plt.title('Explained Variance Ratios for Principal Components')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.grid(True)

# Find the elbow point
kneedle_normalized = KneeLocator(range(1, 21),
explained_variance_ratios_normalized[:20], curve='convex',
direction='decreasing')
plt.axvline(x=kneedle_normalized.knee, color='r', linestyle='--',
label='Elbow')

# Show the plot
plt.legend()
plt.show()

# percentage of variance is explained by the first principal component
print('Percentage of variance is explained by the first principal
component: {}%'.format(explained_variance_ratios_normalized[0]*100))

# Calculate cumulative variance ratios
cumulative_variance_ratios_normalized =
np.cumsum(explained_variance_ratios_normalized)
```

```python
# Find the number of components needed to reach 95% cumulative variance
num_components_95_normalized =
np.where(cumulative_variance_ratios_normalized >= 0.95)[0][0] + 1

# Plot the cumulative variance ratios
plt.figure(figsize=(12, 6))
plt.plot(range(1, len(cumulative_variance_ratios_normalized) + 1),
cumulative_variance_ratios_normalized, marker='o', linestyle='-',
color='b')
plt.title('Cumulative Variance Ratios by Principal Components (Normalized
Data)')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Ratio')

# Plot a vertical line at the number of components needed for 95% variance
plt.axvline(x=num_components_95_normalized, color='r', linestyle='--',
label=f'95% Variance at PC {num_components_95_normalized}')

# Show the plot
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
Percentage of variance is explained by the first principal component:
31.81726528790996%
```




Explained Variance Ratios for Principal Components
Elbow
Explained Variance Ratio
0.30
0.25
0.20
0.15
0.10
0.05
0.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Principal Component