**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:

日历

描述已自动生成

**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

print(df\_percent\_change)

Output:

日历

描述已自动生成

**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.

# 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]}")

    # print PC

    print(pca.components\_[index])

Output of the eigenvalue of first 5 PCs:

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.

# 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%

**图表, 折线图

描述已自动生成**

**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:

图表

描述已自动生成

**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]}")

    # print PC

    print(pca\_normalized.components\_[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)

# 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 of the eigenvalue of first 5 PCs:

Top 5 Principal Components (Ranked by Eigenvalue):

PC 1: Eigenvalue = 154.8400276159304

PC 2: Eigenvalue = 31.385086467658297

PC 3: Eigenvalue = 16.064844741335126

PC 4: Eigenvalue = 9.480418012795994

PC 5: Eigenvalue = 8.468571757545053

Output of percentage:

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

图表

描述已自动生成图表

描述已自动生成