# Stock Market Analysis and Dimensionality Reduction Using PCA

[Author name]

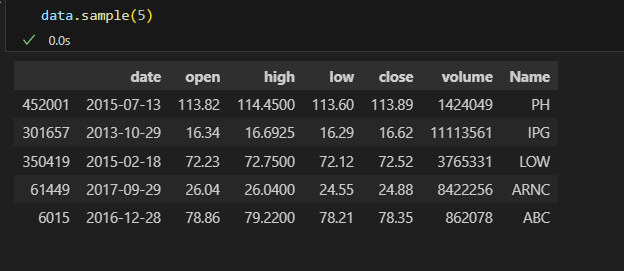
## Task 1

**Code**

import pandas as pd # importing pandas library

data = pd.read\_csv('stock\_data.csv', sep=',') # reading our CSV data file

**Output**



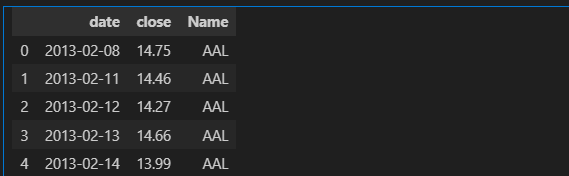
**Code**

# drop the unwanted columns and just keep date, close, Name columns

data = data.drop(['open', 'high', 'low', 'volume'], axis=1)

data.head()

**Output**



**Explanation**

Loading data using pandas

* This code is using the Pandas library in Python. Pandas is great for working with data, kind of like a super-powered Excel for programming.
* **import pandas as pd:** This line brings in the Pandas library and gives it the nickname "pd" so that it's easier to refer to later in the code.
* **data = pd.read\_csv('stock\_data.csv', sep=',')**: Here, it's reading a CSV (Comma Separated Values) file named 'stock\_data.csv'. The pd.read\_csv() function reads the CSV file and creates a DataFrame, which is like a table with rows and columns. The file name ('stock\_data.csv') is inside the parentheses, and sep=',' tells Pandas that the columns in the CSV file are separated by commas. The resulting DataFrame is stored in a variable named data.

## **Task 2**

**Code**

# Get unique names from a column and sort them alphabetically

unique\_names = sorted(data['Name'].unique())  # Collect unique names and sort them alphabetically

num\_names = len(unique\_names)  # Count the number of unique names

first\_5\_names = unique\_names[:5]  # Select the first five names from the sorted list

last\_5\_names = unique\_names[-5:]  # Select the last five names from the sorted list

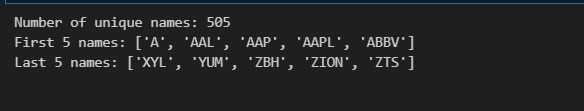
# Output the results

print("Number of unique names:", num\_names)

print("First 5 names:", first\_5\_names)

print("Last 5 names:", last\_5\_names)

**Output**



**Explanation**

This code takes a DataFrame (data), accesses the 'Name' column, finds unique names, sorts them alphabetically, and then prints out the total count of unique names, the first five names in alphabetical order, and the last five names in alphabetical order.

## **Task 3**

**Code**

# Convert 'Date' column to datetime format

data['date'] = pd.to\_datetime(data['date'])  # Convert the 'date' column to a datetime format

# Filter names based on date criteria

filtered\_data = data.groupby('Name').filter(

    lambda x: (x['date'].min() <= pd.Timestamp('2014-07-01')) and (x['date'].max() >= pd.Timestamp('2017-06-30'))

)  # Select names based on date criteria: between July 1, 2014, and June 30, 2017

# Find removed names and count remaining unique names

removed\_names = sorted(set(data['Name'].unique()) - set(filtered\_data['Name'].unique()))  # Names removed due to date criteria

num\_remaining\_names = len(filtered\_data['Name'].unique())  # Count of remaining unique names

# Output the results

print("Names removed:", removed\_names)

print("Number of remaining names:", num\_remaining\_names)

**Output**

*Names removed: ['APTV', 'BHF', 'BHGE', 'CFG', 'CSRA', 'DWDP', 'DXC', 'EVHC', 'FTV', 'HLT', 'HPE', 'HPQ', 'KHC', 'PYPL', 'QRVO', 'SYF', 'UA', 'WLTW', 'WRK']*

*Number of remaining names: 486*

**Explanation**

This code first converts the 'Date' column in the DataFrame data to a datetime format. Then, it filters the data based on specific date criteria (between July 1, 2014, and June 30, 2017) and creates a new DataFrame called filtered\_data. Finally, it identifies the names that were removed due to the date criteria and calculates the count of unique names that remain after filtering.

## **Task 4**

**Code**

# Filter dates based on specific criteria

filtered\_dates = filtered\_data[

    (filtered\_data['date'] >= pd.Timestamp('2014-07-01')) & (filtered\_data['date'] <= pd.Timestamp('2017-06-30'))

]  # Select dates between July 1, 2014, and June 30, 2017

# Find unique dates and count them

unique\_dates = sorted(filtered\_dates['date'].unique())  # Collect unique dates and sort them

num\_dates = len(unique\_dates)  # Count the number of unique dates

# Select the first five and last five dates

first\_5\_dates = unique\_dates[:5]  # Get the first five dates

last\_5\_dates = unique\_dates[-5:] # Get the last five dates

# Output the results

print("Number of dates:", num\_dates)

print("First 5 dates:", first\_5\_dates)

print("Last 5 dates:", last\_5\_dates)

**Output**

*Number of dates: 757*

*First 5 dates: [Timestamp('2014-07-01 00:00:00'), Timestamp('2014-07-02 00:00:00'), Timestamp('2014-07-03 00:00:00'), Timestamp('2014-07-07 00:00:00'), Timestamp('2014-07-08 00:00:00')]*

*Last 5 dates: [Timestamp('2017-06-26 00:00:00'), Timestamp('2017-06-27 00:00:00'), Timestamp('2017-06-28 00:00:00'), Timestamp('2017-06-29 00:00:00'), Timestamp('2017-06-30 00:00:00')]*

**Explanation**

This code takes the previously filtered data (filtered\_data), specifically focuses on the 'date' column, and filters dates within a certain range (from July 1, 2014, to June 30, 2017). Then, it collects unique dates, counts them, and displays the total count of unique dates along with the first five and last five dates in that range.

## **Task 5**

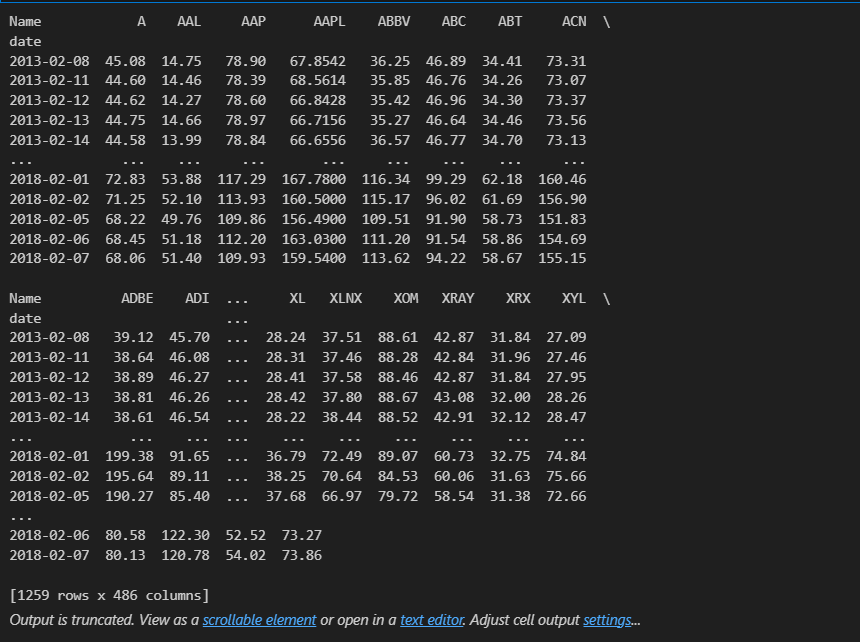
**Code**

# Build a new DataFrame with close values for each name and date

pivot\_data = filtered\_data.pivot(index='date', columns='Name', values='close')

print(pivot\_data)

**Output**



**Explanation**

It rearranges the data so that you have dates as rows, names as columns, and the 'close' prices as the values where the rows and columns intersect. The resulting DataFrame pivot\_data will show the close prices for each 'Name' across various dates. The print(pivot\_data) line displays this newly structured DataFrame.

## **Task 6**

**Code**

returns\_data = {}  # Initialize an empty dictionary to store returns for each stock

# Loop through unique 'Name's in filtered\_data

for name in filtered\_data['Name'].unique():

    temp = filtered\_data[filtered\_data['Name'] == name].sort\_values('date')  # Select data for each 'Name' and sort by 'date'

    temp['Return'] = (temp['close'] - temp['close'].shift(1)) / temp['close'].shift(1)  # Calculate daily returns

    returns\_data[name] = temp['Return'].values[1:]  # Store calculated returns in the dictionary

returns\_data\_2 = {}  # Initialize another dictionary for filtered returns

# Loop through items in returns\_data

for item, value in returns\_data.items():

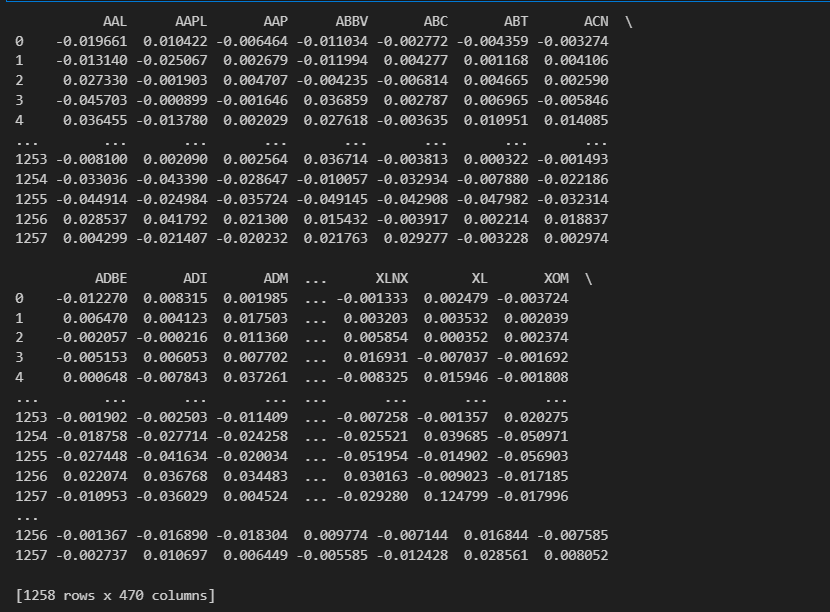
    if len(value) == 1258:  # Check if the number of returns is 1258 (assuming daily data for about 5 years)

        returns\_data\_2[item] = value  # Store returns with correct length in another dictionary

returns\_df = pd.DataFrame(returns\_data\_2)  # Create a DataFrame using the filtered returns

print(returns\_df)

**Output**



**Explanation**

* This code calculates daily returns for each stock based on the 'close' prices in the filtered\_data DataFrame. It iterates through each unique 'Name', computes daily returns using the formula `(today’s close price−yesterday’s close price)/yesterday’s close price` and stores these returns in dictionaries.
* Then it filters out stocks that don't have the expected number of returns (1258 assuming daily data for about 5 years) and creates a DataFrame returns\_df containing the calculated returns for those specific stocks.
* Finally, it displays this DataFrame.

## **Task 7**

**Code**

from sklearn.decomposition import PCA

# Calculate principal components

pca = PCA()  # Initialize PCA

pca.fit(returns\_df.dropna())  # Fit PCA model after dropping NaN values in returns\_df

# Print top five principal components by eigenvalue

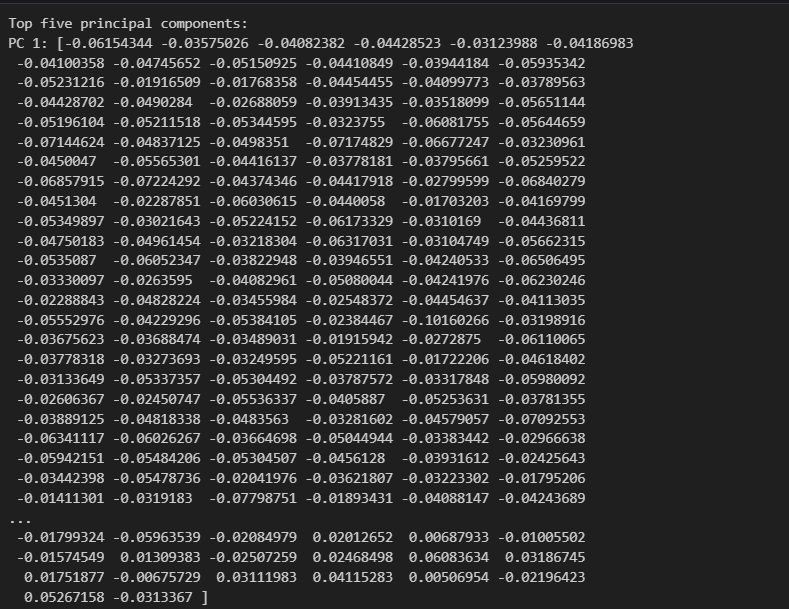
top\_five\_PCs = pca.components\_[:5]  # Extract the top five principal components

print("Top five principal components:")

for i, pc in enumerate(top\_five\_PCs, 1):  # Iterate through top five principal components

    print(f"PC {i}: {pc}")  # Print each principal component

**Output**



**Explanation**

* This code uses the scikit-learn library to perform Principal Component Analysis (PCA) on the returns\_df DataFrame. PCA is a technique used for reducing the dimensionality of the data while preserving important information.
* The code initializes a PCA model, fits it to the data after dropping any rows containing NaN values in the returns\_df, and then prints the top five principal components (PCs) based on their eigenvalues. Each principal component represents a combination of the original features (in this case, stock returns) that captures the most variance in the data.

## **Task 8**

**Code**

import matplotlib.pyplot as plt

# Extract explained variance ratios from PCA

explained\_variance\_ratios = pca.explained\_variance\_ratio\_  # Extract explained variance ratios

# Calculate percentage of variance explained by the first principal component

variance\_explained = explained\_variance\_ratios[0] \* 100  # Calculate variance explained by the first PC

print(f"Percentage of variance explained by the first PC: {variance\_explained:.2f}%")

# Plot explained variance ratios for the top 20 principal components

plt.figure(figsize=(8, 6))  # Set the figure size

plt.plot(range(1, 21), explained\_variance\_ratios[:20], marker='o')  # Plot explained variance ratios for the top 20 PCs

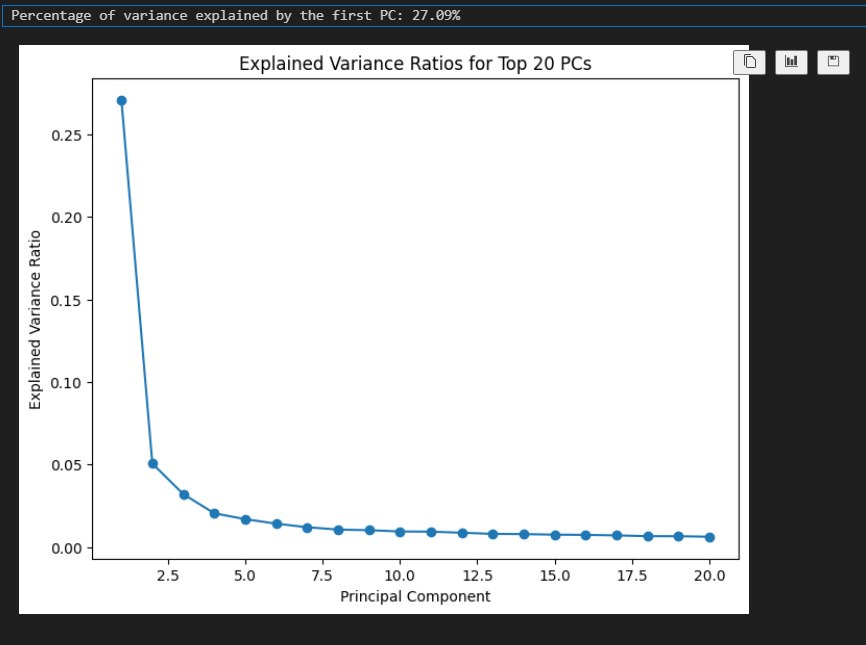
plt.xlabel('Principal Component')  # Set x-axis label

plt.ylabel('Explained Variance Ratio')  # Set y-axis label

plt.title('Explained Variance Ratios for Top 20 PCs')  # Set plot title

plt.show()  # Display the plot

**Output**



**Explanation**

* This code snippet uses Matplotlib, a plotting library in Python, to visualize the explained variance ratios for the principal components obtained from the PCA analysis.
* First, it calculates the explained variance ratios for each principal component obtained from the PCA (explained\_variance\_ratios). Then, it calculates and prints the percentage of variance explained by the first principal component.
* Finally, it creates a plot displaying the explained variance ratios for the top 20 principal components. The x-axis represents the principal components, and the y-axis represents the corresponding explained variance ratios. The plot shows how much variance in the data each principal component explains, helping to understand the significance of each component in capturing the data's variability.

## **Task 9**

**Code**

import numpy as np

# Calculate cumulative variance ratios

cumulative\_variance\_ratios = np.cumsum(explained\_variance\_ratios)  # Calculate cumulative sum of explained variance ratios

# Plot cumulative variance ratios

plt.figure(figsize=(8, 6))  # Set the figure size

plt.plot(range(1, len(cumulative\_variance\_ratios) + 1), cumulative\_variance\_ratios, marker='o')  # Plot cumulative variance ratios

plt.xlabel('Principal Component')  # Set x-axis label

plt.ylabel('Cumulative Variance Ratio')  # Set y-axis label

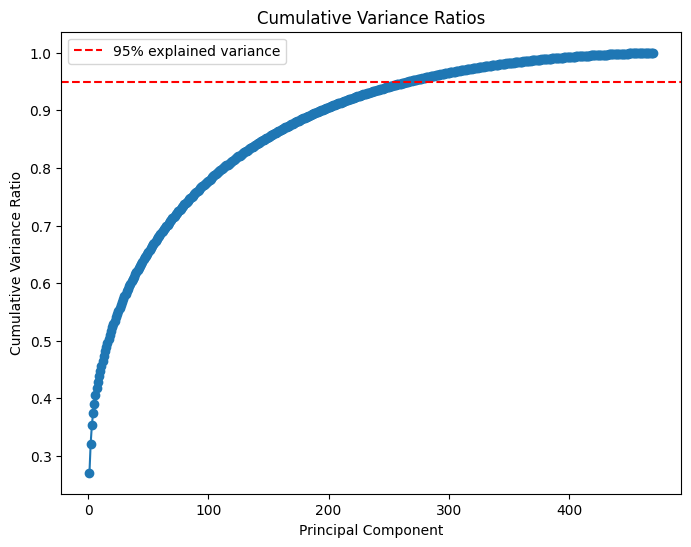
plt.title('Cumulative Variance Ratios')  # Set plot title

plt.axhline(y=0.95, color='r', linestyle='--', label='95% explained variance')  # Add a horizontal line at 95% explained variance

plt.legend()  # Show the legend

plt.show()  # Display the plot

**Output**



**Explanation**

* This code calculates the cumulative sum of the explained variance ratios obtained from PCA using NumPy's cumsum() function. Then, it creates a plot to visualize the cumulative variance ratios for each principal component.
* The x-axis represents the principal components, and the y-axis represents the cumulative variance ratios. This plot helps in understanding how many principal components are required to reach a certain level of cumulative variance. The red dashed line indicates 95% explained variance, showing the number of principal components needed to explain at least 95% of the variability in the data.

## **Task 10**

**Code**

from sklearn.preprocessing import StandardScaler  # Import StandardScaler for normalization

# Normalizing the returns DataFrame

scaler = StandardScaler()  # Initialize StandardScaler

normalized\_returns = scaler.fit\_transform(returns\_df.dropna())  # Normalize the returns data after dropping NaN values

# Applying PCA to normalized data

pca\_normalized = PCA()  # Initialize PCA for normalized data

pca\_normalized.fit(normalized\_returns)  # Fit PCA to the normalized returns data

# Extract explained variance ratios for normalized data

explained\_variance\_ratios\_normalized = pca\_normalized.explained\_variance\_ratio\_  # Extract explained variance ratios

# Calculate cumulative variance ratios for normalized data

cumulative\_variance\_ratios\_normalized = np.cumsum(explained\_variance\_ratios\_normalized)  # Calculate cumulative variance ratios

# Plot cumulative variance ratios for normalized data

plt.figure(figsize=(8, 6))  # Set the figure size

plt.plot(range(1, len(cumulative\_variance\_ratios\_normalized) + 1), cumulative\_variance\_ratios\_normalized, marker='o')  # Plot cumulative variance ratios

plt.xlabel('Principal Component')  # Set x-axis label

plt.ylabel('Cumulative Variance Ratio')  # Set y-axis label

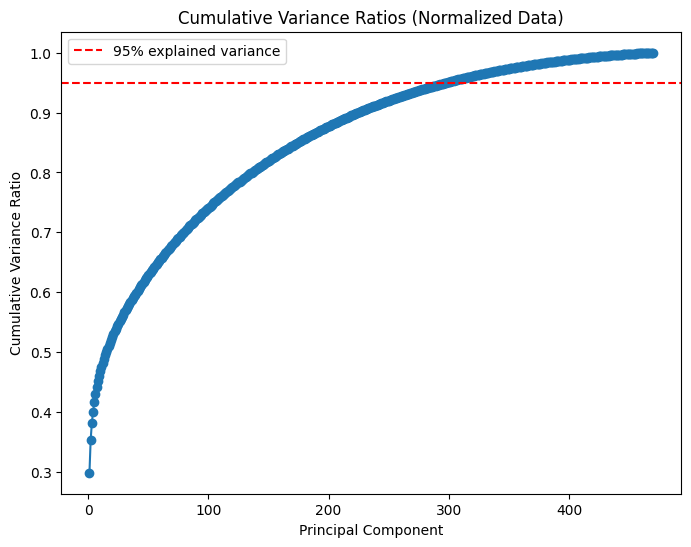
plt.title('Cumulative Variance Ratios (Normalized Data)')  # Set plot title

plt.axhline(y=0.95, color='r', linestyle='--', label='95% explained variance')  # Add a horizontal line at 95% explained variance

plt.legend()  # Show the legend

plt.show()  # Display the plot

**Output**



**Explanation**

* This code snippet introduces data normalization using StandardScaler from scikit-learn. It normalizes the returns data by transforming it to have a mean of 0 and a standard deviation of 1 for each feature (column).
* Then, it applies PCA to the normalized returns data. The subsequent steps, similar to the previous visualization, calculate and plot the cumulative variance ratios for the principal components derived from the PCA on the normalized data. The red dashed line represents the 95% explained variance threshold.
* This plot demonstrates the cumulative variance explained by each principal component after normalizing the data, providing insights into how normalization impacts the explained variance by the principal components.