Table of Contents

1 Import Libraries and Given Functions

In [1]:

```python
from __future__ import print_function
import pandas as pd
import numpy as np
from numpy import linalg

"""
Loads financial data as a pandas dataframe
"""
```

```
executed in 810ms, finished 15:54:53 2021-12-04
```

Out[1]:

```
'\nLoads financial data as a pandas dataframe\n'
```

In [2]:

```
def load_dataframe(filename):
    return pd.read_csv(filename, index_col=0)


"""
Loads financial data as a tuple: names,data.
names is a list of the stock names represented in each column.
data is a 2d numpy array.  Each row of data corresponds to a trading day.
data[i,j] is the price (technically the adjusted closing price) of
instrument names[j] on the ith day.  The days are ordered chronologically.
"""


def load_data(filename):↔


"""
Given a 1d numpy array vec of n values, and a list of n names,
prints the values and their associated names.
"""


def pretty_print(vec, names):↔


"""
Given a 1d numpy array vec of n values, and a list of n names,
prints the values and their associated names in a LaTeX friendly
format.
"""


def pretty_print_latex(vec, names, num_col=6):↔


def main():↔
    # pretty_print_latex(data[0,:],names)


if __name__ == "__main__":↔
```

executed in 60ms, finished 15:54:53 2021-12-04

```
# of stocks = 18, # of days = 433
      AAPL     AMZN     MSFT     GOOG      XOM      APC      CVX        C  \
  112.6208  753.67  60.1715  786.14  83.316  68.7428  109.136  58.0193

        GS      JPM       AET       JNJ      DGX       SPY      XLF      SS
O  \
  235.3232  82.7023  120.0301  109.9246  88.5165  216.8377  22.7081  76.400
6

      SDS    USO
  58.5837  11.44
```

In [3]:

```
names, df = load_data('stockprices.csv')
df = load_dataframe('stockprices.csv')
```

executed in 29ms, finished 15:54:54 2021-12-04

In [4]:

```
len_df = len(df)
df1 = df.iloc[0:len(df)-1, :]
df2 = df.iloc[1:len(df), :]
df3 = df2.reset_index(drop=True) - df1.reset_index(drop=True)
print(df2.shape, df1.shape)
```

executed in 44ms, finished 15:54:54 2021-12-04

```
(432, 18) (432, 18)
```

2 PCA Function Below

In [5]:

```
def pca(data, stand=False, k=None, var=False):

    cols = list(data.iloc[:0])
    data_copy = data.copy
    # Center Data at 0, Each Column must have mean 0

    # Loop through every column
    data_copy = data - data.mean()

    # If we should standardize, then standardize the dataset
    if stand == True:
        data_copy = data_copy / data_copy.std()

    # Compute Covariance Matrix
    Cov_matrix = data_copy.T @ data_copy

    # Calculate eigendecomp for Cov_Matrix
    vals, vectors = np.linalg.eigh(Cov_matrix)

    # Sort the eigenvalues descending order
    vals = vals[::-1]
    vectors = vectors[:, ::-1]

    # If we wanted the least dimensions for a certain amount of variance ex

    # See if Var was passed to function and its valid
    if var > 0 and var <= 1:
        tracker = 0
        eig_vals_of_interest, eig_vectors_of_interest = [], []
        total_var = vals.sum()
        for i in range(len(vals)):
            if tracker < var:
                tracker += vals[i] / total_var
                eig_vals_of_interest.append(vals[i])
        eig_vectors_of_interest = vectors[:, :len(eig_vals_of_interest)]

    # Check if we just wanted k dimensions
    elif k > 0 and k <= len(cols):

        # If we wanted k dimensions, set the appropriate amount
        eig_vals_of_interest = vals[:k]
        eig_vectors_of_interest = vectors[:, :k]
    else:
        return vals, vectors

    # Make sure data types are compatible
    eig_vectors_of_interest = np.array(eig_vectors_of_interest)

    # Compute the data projected onto the pcas
    new_data = data_copy@eig_vectors_of_interest

    #Returns the data projected onto the pcas, the eigenvalues, and the eig
    return Cov_matrix, eig_vals_of_interest, eig_vectors_of_interest
```

executed in 14ms, finished 15:54:54 2021-12-04

3 Prolem A

In [6]:

```
new, vals, vec = pca(df3, stand=False, k=18)
print('PCA 1: \n')
pretty_print(vec[:, 0], names)
print('\n','PCA 2: \n')
pretty_print(vec[:, 1], names)
```

executed in 93ms, finished 15:54:54 2021-12-04

```
PCA 1: 

      AAPL     AMZN      MSFT      GOOG       XOM       APC       CVX  \
 -0.054553 -0.86793 -0.036651 -0.482672 -0.007916 -0.009795 -0.013876   

         C        GS       JPM       AET       JNJ       DGX       SPY  \
 -0.012407 -0.053403 -0.020728 -0.008575 -0.013319 -0.011993 -0.054358   

       XLF       SSO       SDS       USO  
 -0.004979 -0.044236  0.016887 -0.001485  

 PCA 2: 

      AAPL      AMZN      MSFT      GOOG       XOM       APC      CVX  \
  0.041894 -0.494995  0.026756  0.851087  0.028004  0.009165  0.02881   

        C        GS       JPM       AET       JNJ       DGX       SPY  \
  0.02591  0.114993  0.037115  0.028005  0.041139  0.011263  0.069472   

       XLF       SSO       SDS       USO  
  0.009083  0.056721 -0.021421  0.000401  
```

4 Problem B

In [7]:

```
new1, vals, vec = pca(df3, k=18)
print('PCA 1: \n')
pretty_print(vec[:, 0], names)
print('\n','PCA 2: \n')
pretty_print(vec[:, 1], names)
```

executed in 42ms, finished 15:54:54 2021-12-04

```
PCA 1: 

     AAPL      AMZN      MSFT      GOOG       XOM       APC       CVX  \
-0.054553 -0.86793 -0.036651 -0.482672 -0.007916 -0.009795 -0.013876   

        C        GS       JPM       AET       JNJ       DGX       SPY  \
-0.012407 -0.053403 -0.020728 -0.008575 -0.013319 -0.011993 -0.054358   

      XLF       SSO       SDS       USO  
-0.004979 -0.044236  0.016887 -0.001485  

 PCA 2: 

     AAPL      AMZN      MSFT      GOOG       XOM       APC       CVX  \
 0.041894 -0.494995  0.026756  0.851087  0.028004  0.009165  0.02881   

       C        GS       JPM       AET       JNJ       DGX       SPY  \
 0.02591  0.114993  0.037115  0.028005  0.041139  0.011263  0.069472   

      XLF       SSO       SDS       USO  
 0.009083  0.056721 -0.021421  0.000401  
```

5 Problem C ¶

In [8]:

```
shares_1 = [200]*4
shares_2 = [100]*14
shares = shares_1 + shares_2
shares = np.array(shares)
```

executed in 12ms, finished 15:54:54 2021-12-04

In [9]:

```

new1 = new1 / 431
number1 = shares.T @ new1
number1 = number1 @ shares
```

executed in 13ms, finished 15:54:54 2021-12-04

In [22]:

```
std_dev = number1 **.5
print(std_dev)
```

executed in 17ms, finished 16:04:45 2021-12-04

```
6962.072274462166
```

6 Problem D

In [17]:

```
#Initialize variables
list_of_means = []

#Iterate through the columns and get the mean return
for i in range(len(df3.iloc[0,:])):
    list_of_means.append(np.mean(df3.iloc[:,i]))

#Calculate expected value
expected_value = shares.T @ list_of_means
```

executed in 12ms, finished 15:59:50 2021-12-04

In [18]:

```
#Print expected value of our portfolio
print(expected_value)
```

executed in 14ms, finished 15:59:50 2021-12-04

```
879.7824537037035
```

In [23]:

```
#Calculate Z-Score
z_score = (-1000 - expected_value) / std_dev
```

executed in 16ms, finished 16:04:58 2021-12-04

In [24]:

```
z_score
```

executed in 13ms, finished 16:04:59 2021-12-04

Out[24]:

```
-0.270003294938348
```

In [ ]: