hw5

20989977 Zhang Mingtao

2023/10/27

1.

```
import pandas as pd
import yfinance as yf
import numpy as np

#1
start_date = "2022-10-26"
end_date1 = "2023-10-26"

tickers = ["AAPL", "MSFT", "AMZN", "NVDA", "GOOGL", "META", "GOOG", "TSLA", "BRK.B", "UNH", "^G
SPC"]
tickers2 = ["AAPL", "MSFT", "AMZN", "NVDA", "GOOGL", "META", "GOOG", "TSLA", "BRK.B", "UNH"]

data1 = yf.download(tickers, start=start_date, end=end_date1)

# 获取每个股票的调整收盘价
```

```
##
[                       0%%                       ]
[*********             18%%                       ]  2 of 11 completed
[*************         27%%                       ]  3 of 11 completed
[*****************     36%%                       ]  4 of 11 completed
[**********************45%%                       ]  5 of 11 completed
[**********************55%%                       ]  6 of 11 completed
[**********************64%%*****                  ]  7 of 11 completed
[**********************73%%*********              ]  8 of 11 completed
[**********************82%%*************          ]  9 of 11 completed
[**********************91%%******************     ]  10 of 11 completed
[*********************100%%**********************]  11 of 11 completed
##
##
## 1 Failed download:
## ['BRK.B']: Exception('%ticker%: No timezone found, symbol may be delisted')
```

```
adj_closing_prices1 = data1["Adj Close"]

# 搞到BRK.B的数据：

# Alpha Vantage: 9MV3V4OZJVR4JEN3
# Nasdaq: D1zHPDcxv3TyJRsoFVEr
# Polygon: SyBy5BXPYaNpQgvRklreByZcTvUhm0dw

import requests

# 替换为你的 Polygon.io API 密钥
api_key = 'SyBy5BXPYaNpQgvRklreByZcTvUhm0dw'

# 指定股票代码
symbol = 'BRK.B'

end_date2 = "2023-10-25"

# 构建请求URL
url = f"https://api.polygon.io/v2/aggs/ticker/{symbol}/range/1/day/{start_date}/{end_date2}?adj
usted=true&apiKey={api_key}"

# 发起GET请求获取数据
response = requests.get(url)

# 解析JSON响应
data = response.json()

# 将数据转换为DataFrame
df = pd.DataFrame(data["results"])

# 选择调整收盘价列
adj_closing_prices2 = df["c"]

adj_closing_prices1["BRK.B"] = list(adj_closing_prices2)
```

```
## <string>:1: SettingWithCopyWarning:
## A value is trying to be set on a copy of a slice from a DataFrame.
## Try using .loc[row_indexer,col_indexer] = value instead
##
## See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
```

```
pricedf = adj_closing_prices1.copy(deep=True)

# 初始化一个DataFrame来存储每个股票的日收益率
returns = pd.DataFrame()

# 计算每个股票的日收益率
for ticker in tickers:
  returns[ticker] = np.log(pricedf[ticker] / pricedf[ticker].shift(1))

returns.drop(pd.to_datetime('2022-10-26'),inplace=True)

print(returns.drop(columns=['^GSPC']))
```

```
##                 AAPL      MSFT      AMZN  ...      TSLA     BRK.B       UNH
## Date                                      ...                              
## 2022-10-27 -0.030939 -0.019954 -0.041485  ...  0.002001  0.004703 -0.002525
## 2022-10-28  0.072835  0.039433 -0.070468  ...  0.015123  0.033081  0.017273
## 2022-10-31 -0.015530 -0.015983 -0.009424  ... -0.004298 -0.015268  0.007068
## 2022-11-01 -0.017698 -0.017207 -0.056734  ...  0.001230 -0.003259 -0.014223
## 2022-11-02 -0.038019 -0.036009 -0.049452  ... -0.058011 -0.016005 -0.007114
## ...              ...       ...       ...  ...       ...       ...       ...
## 2023-10-19 -0.002163  0.003659  0.002105  ... -0.097616 -0.006563 -0.008298
## 2023-10-20 -0.014813 -0.014134 -0.025478  ... -0.037588 -0.008302 -0.008690
## 2023-10-23  0.000694  0.008079  0.011044  ...  0.000424  0.002914 -0.010414
## 2023-10-24  0.002540  0.003667  0.015679  ...  0.020719  0.005300  0.006555
## 2023-10-25 -0.013584  0.030217 -0.057387  ... -0.019118 -0.005122  0.009875
## 
## [250 rows x 10 columns]
```

2.

```
#2
pricedf0=returns.drop(columns=['^GSPC'])

# 使用corr方法计算相关性矩阵
correlation_matrix = pricedf0.corr()

# 创建样式来保留三位小数
styled_correlation_matrix = correlation_matrix.round(3)

# 打印样式化的相关性矩阵
print(styled_correlation_matrix)
```

```
##          AAPL   MSFT   AMZN   NVDA  GOOGL   META   GOOG   TSLA  BRK.B    UNH
## AAPL   1.000  0.648  0.478  0.540  0.628  0.514  0.631  0.466  0.573  0.174
## MSFT   0.648  1.000  0.618  0.610  0.606  0.517  0.612  0.351  0.417  0.157
## AMZN   0.478  0.618  1.000  0.466  0.647  0.532  0.656  0.403  0.366 -0.010
## NVDA   0.540  0.610  0.466  1.000  0.482  0.368  0.492  0.474  0.371 -0.008
## GOOGL  0.628  0.606  0.647  0.482  1.000  0.576  0.998  0.354  0.465  0.027
## META   0.514  0.517  0.532  0.368  0.576  1.000  0.571  0.280  0.329 -0.061
## GOOG   0.631  0.612  0.656  0.492  0.998  0.571  1.000  0.357  0.469  0.032
## TSLA   0.466  0.351  0.403  0.474  0.354  0.280  0.357  1.000  0.274  0.092
## BRK.B  0.573  0.417  0.366  0.371  0.465  0.329  0.469  0.274  1.000  0.205
## UNH    0.174  0.157 -0.010 -0.008  0.027 -0.061  0.032  0.092  0.205  1.000
```

3.

```
#3
# 标准化
mean = np.mean(pricedf0, axis=0)
std = np.std(pricedf0, axis=0)
pricedf0 = (pricedf0 - mean) / std

# 计算相关性矩阵的特征值和特征向量
eigenvalues, eigenvectors = np.linalg.eig(np.cov(pricedf0.T))

#三位小数精度不够因此我分别设置了4位和5位：

# 打印特征值
with np.printoptions(precision=5, suppress=True):
    print(f"Eigenvalues: {eigenvalues}")

# 打印特征向量
```

```
## Eigenvalues: [5.14598 1.1675  0.00206 0.89306 0.23337 0.33989 0.45422 0.65724 0.54988
##  0.59697]
```

```
with np.printoptions(precision=3, suppress=True):
    print(f"Eigenvectors: {eigenvectors}")
```

```
## Eigenvectors: [[-0.36  -0.195 -0.002 -0.034  0.529 -0.578 -0.368  0.233 -0.161 -0.038]
##  [-0.357 -0.067  0.    -0.028 -0.607 -0.399  0.07  -0.172  0.044 -0.551]
##  [-0.34   0.185 -0.013  0.007  0.407 -0.093  0.758 -0.316  0.017  0.013]
##  [-0.31  -0.004 -0.008 -0.481  0.238  0.525 -0.141  0.132  0.315 -0.451]
##  [-0.386  0.16  -0.703  0.294 -0.089  0.087 -0.217 -0.091  0.312  0.276]
##  [-0.306  0.255  0.009  0.173 -0.034  0.355 -0.159 -0.095 -0.798 -0.116]
##  [-0.388  0.154  0.711  0.287 -0.075  0.085 -0.202 -0.094  0.323  0.267]
##  [-0.247 -0.152  0.002 -0.692 -0.259 -0.039 -0.003 -0.152 -0.174  0.561]
##  [-0.275 -0.33   0.     0.185 -0.195  0.155  0.388  0.742 -0.065  0.122]
##  [-0.047 -0.823 -0.003  0.232  0.098  0.234 -0.032 -0.446 -0.031 -0.026]]
```

4.

```
#4
# 使用argsort函数对特征值降序排序并获取排序后的索引
sorted_indices = np.argsort(eigenvalues)[::-1]

# 根据排序后的索引重新排列特征值和特征向量
sorted_eigenvalues = eigenvalues[sorted_indices]
sorted_eigenvectors = eigenvectors[:, sorted_indices]
sorted_eigenvectors = sorted_eigenvectors.T

# 打印排序后的特征值
with np.printoptions(precision=5, suppress=True):
    print(f"Sorted Eigenvalues: {sorted_eigenvalues}")

# 打印排序后的特征向量
```

```
## Sorted Eigenvalues: [5.14598 1.1675  0.89306 0.65724 0.59697 0.54988 0.45422 0.33989 0.23337
##  0.00206]
```

```
with np.printoptions(precision=3, suppress=True):
    print(f"Sorted Eigenvectors: {sorted_eigenvectors}")



#对比scikit-learn库
```

```
## Sorted Eigenvectors: [[-0.36  -0.357 -0.34  -0.31  -0.386 -0.306 -0.388 -0.247 -0.275 -0.04
7]
##  [-0.195 -0.067  0.185 -0.004  0.16   0.255  0.154 -0.152 -0.33  -0.823]
##  [-0.034 -0.028  0.007 -0.481  0.294  0.173  0.287 -0.692  0.185  0.232]
##  [ 0.233 -0.172 -0.316  0.132 -0.091 -0.095 -0.094 -0.152  0.742 -0.446]
##  [-0.038 -0.551  0.013 -0.451  0.276 -0.116  0.267  0.561  0.122 -0.026]
##  [-0.161  0.044  0.017  0.315  0.312 -0.798  0.323 -0.174 -0.065 -0.031]
##  [-0.368  0.07   0.758 -0.141 -0.217 -0.159 -0.202 -0.003  0.388 -0.032]
##  [-0.578 -0.399 -0.093  0.525  0.087  0.355  0.085 -0.039  0.155  0.234]
##  [ 0.529 -0.607  0.407  0.238 -0.089 -0.034 -0.075 -0.259 -0.195  0.098]
##  [-0.002  0.    -0.013 -0.008 -0.703  0.009  0.711  0.002  0.    -0.003]]
```

```
from sklearn.decomposition import PCA

# 创建PCA对象
pca = PCA()

# 拟合数据并进行主成分分析
pca.fit(pricedf0)

# 查看主成分分析的结果
```

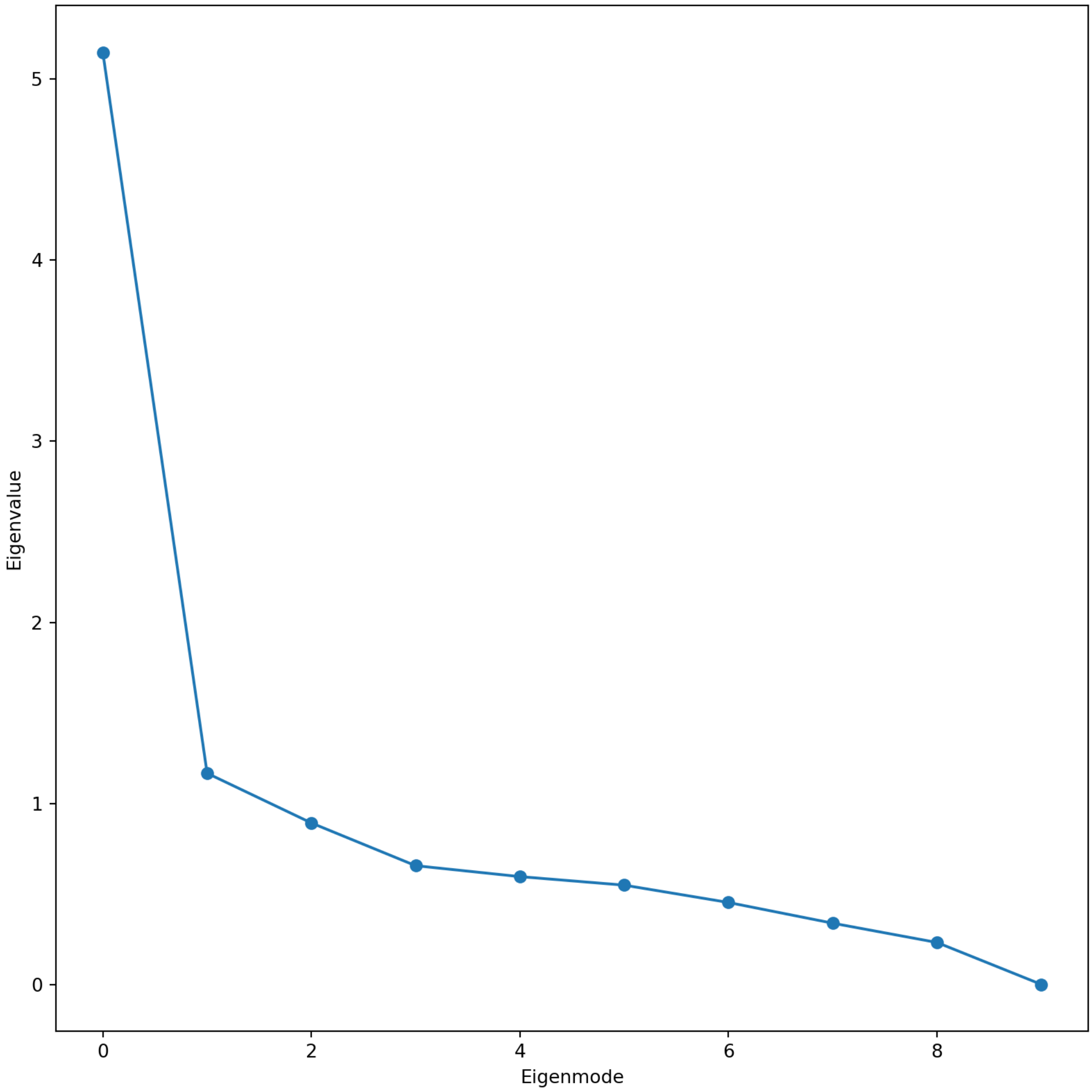
▾ PCA

PCA()

```
explained_variance_ratio = pca.explained_variance_ratio_  # 主成分的方差解释比例
components = pca.components_  # 主成分的载荷（特征向量）

# 转换数据到主成分空间
transformed_data = pca.transform(pricedf0)

# 对比R
# library(reticulate)
# setwd("C:\\Users\\张铭韬\\Desktop")
# py <- import("hw5.py")
# # 从Python中传递DataFrame到R
# df <- py$pricedf0
# pca <- prcomp(df)
# summary(pca)
```

5.

```
#5
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(10,10), dpi=300)
plt.plot(range(0,10), sorted_eigenvalues, marker='o')
plt.xlabel('Eigenmode')
plt.ylabel('Eigenvalue')

plt.show()
```



7.

```
#7
cumulative_variance=np.cumsum(sorted_eigenvalues) / np.sum(sorted_eigenvalues)
cumulative_variance

# 选取N=4即前三个主成分用于后续的建模
```

```
## array([0.51253977, 0.62882322, 0.71777168, 0.78323238, 0.84269016,
##        0.89745844, 0.94269861, 0.97655137, 0.99979461, 1.        ])
```

8.

```
#8

fig, ax = plt.subplots(figsize=(12, 9))

np.random.seed(1)

for i, ticker in enumerate(tickers2):
  plt.scatter(components[1][i], components[2][i], color=np.random.rand(3,), marker='o')

plt.xlabel('2nd')
plt.ylabel('3rd')
plt.title('2nd - 3rd PCA plot')

legend = plt.legend(tickers2, loc='upper left', title='names')
plt.setp(legend.get_title(), fontsize=9)

# plt.axis('off')

# 将坐标轴移动到原点位置
```

```
## [None, None]
```

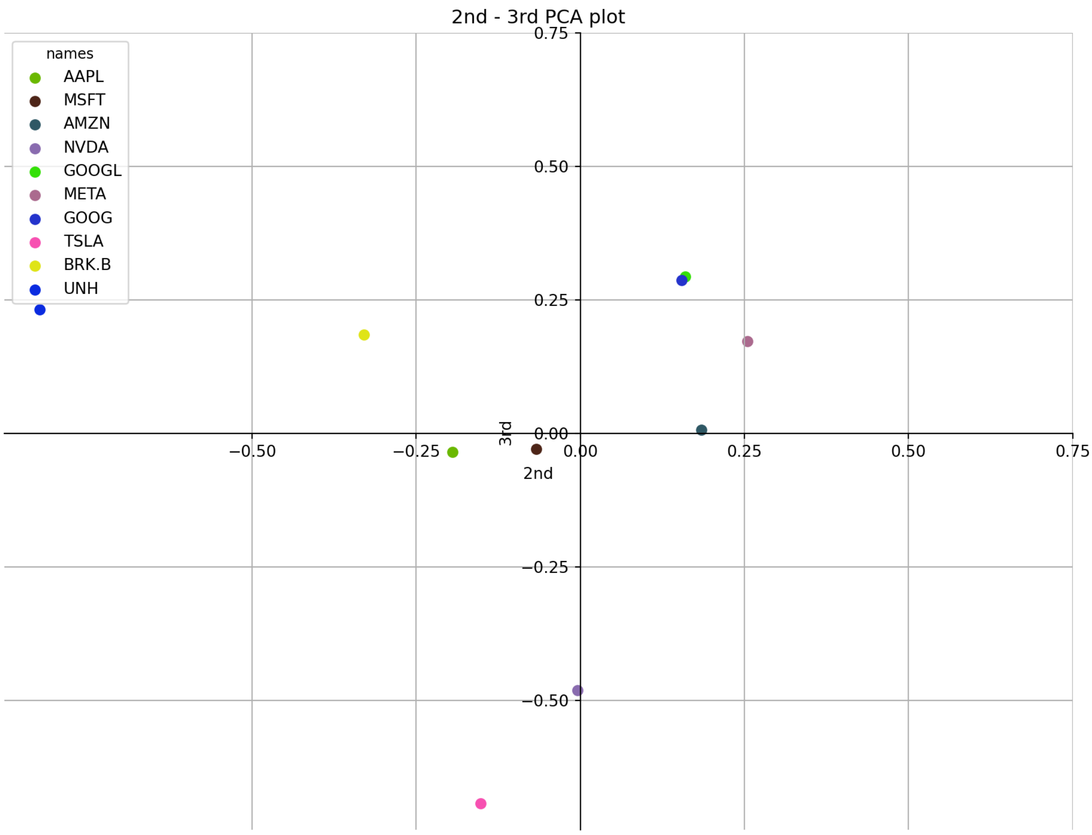
```
ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')

# 去掉上面和右面的线
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

plt.xticks(np.arange(-0.5, 1, 0.25))
```

```
## ([<matplotlib.axis.XTick object at 0x00000000E1A87A90>, <matplotlib.axis.XTick object at 0x000000006B2B3710>, <matplotlib.axis.XTick object at 0x00000000E1A94BD0>, <matplotlib.axis.XTick object at 0x000000006B2EE9D0>, <matplotlib.axis.XTick object at 0x00000000E1ABEA50>, <matplotlib.axis.XTick object at 0x00000000E1AC4DD0>], [Text(-0.5, 0, '-0.50'), Text(-0.25, 0, '-0.25'), Text(0.0, 0, '0.00'), Text(0.25, 0, '0.25'), Text(0.5, 0, '0.50'), Text(0.75, 0, '0.75')])
```

```
plt.yticks(np.arange(-0.5, 1, 0.25))
```

```
## ([<matplotlib.axis.YTick object at 0x000000006B2DF6D0>, <matplotlib.axis.YTick object at 0x00000000E1A236D0>, <matplotlib.axis.YTick object at 0x00000000E1AC8F50>, <matplotlib.axis.YTick object at 0x00000000E1AC4450>, <matplotlib.axis.YTick object at 0x00000000E1ACC110>, <matplotlib.axis.YTick object at 0x00000000E1ACE3D0>], [Text(0, -0.5, '-0.50'), Text(0, -0.25, '-0.25'), Text(0, 0.0, '0.00'), Text(0, 0.25, '0.25'), Text(0, 0.5, '0.50'), Text(0, 0.75, '0.75')])
```

```
plt.grid(True)

plt.show()
```



9.

```
#A

#9
names=tickers2.copy()
weight_EWS=np.array([0.1]*10)
price=pricedf.drop(columns=['^GSPC']).iloc[0].values
volume_EWS=weight_EWS/price

Series_EWS=np.zeros(len(pricedf))
for k in range(len(pricedf)):
  Series_EWS[k]=np.dot(pricedf.drop(columns=['^GSPC']).iloc[k],volume_EWS)
```

10.

```
#B

#10
SD=np.array(np.std(returns.drop(columns=['^GSPC']), axis=0))
inverse=1/SD
weight_RP=inverse/sum(inverse)
volume_RP=weight_RP/price

Series_RP=np.zeros(len(pricedf))
for k in range(len(pricedf)):
  Series_RP[k]=np.dot(pricedf.drop(columns=['^GSPC']).iloc[k],volume_RP)
```

11.

```
#C

#11
equal_wt=np.array([0.1]*10)
Projection_4=np.dot(components[0:4],equal_wt)
Projection_4
```

```
## array([ 0.30151555, -0.08169917, -0.00566217, -0.02610388])
```

12.

```
#12
# 第二三四主成分符号为负，进行更改
Projection_4[1]=-Projection_4[1]
Projection_4[2]=-Projection_4[2]
Projection_4[3]=-Projection_4[3]
Projection_4
```

```
## array([0.30151555, 0.08169917, 0.00566217, 0.02610388])
```

```
components[1]=-components[1]
components[2]=-components[2]
components[3]=-components[3]
components[0:4]
```

```
## array([[ 0.36020049,  0.35650699,  0.34011671,  0.30981028,  0.38572232,
##          0.30639755,  0.38764776,  0.24665797,  0.27473972,  0.04735571],
##        [ 0.19454949,  0.06725041, -0.18469206,  0.00401192, -0.15979121,
##         -0.25468573, -0.15420942,  0.1517548 ,  0.32960782,  0.82319572],
##        [ 0.03394312,  0.028368  , -0.0065637 ,  0.48080939, -0.29422409,
##         -0.17342247, -0.28733659,  0.69239542, -0.18505105, -0.2322963 ],
##        [-0.23282993,  0.17234481,  0.31617951, -0.13159238,  0.09114163,
##          0.0949877 ,  0.09440808,  0.15190513, -0.74181603,  0.44631034]])
```

```
stock_wt_EWP=np.mean(components[0:4], axis=0)
stock_wt_EWP
```

```
## array([ 0.08896579,  0.15611755,  0.11626011,  0.1657598 ,  0.00571216,
##        -0.00668074,  0.01012746,  0.31067833, -0.08062988,  0.27114137])
```

13.

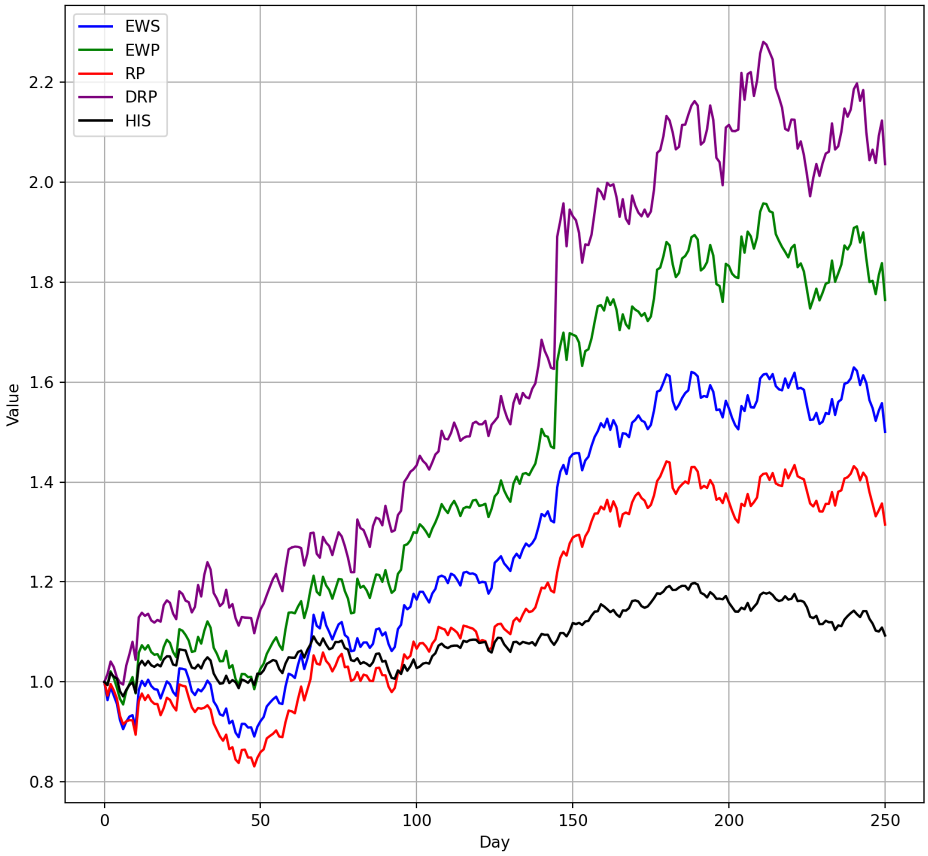
```
#13
stock_wt_EWP_norm = stock_wt_EWP/np.sum(stock_wt_EWP)
volume_EWP=stock_wt_EWP_norm/price

Series_EWP=np.zeros(len(pricedf))
for k in range(len(pricedf)):
  Series_EWP[k]=np.dot(pricedf.drop(columns=['^GSPC']).iloc[k],volume_EWP)
```

14.

```
#D

#14
Eigenvalue=sorted_eigenvalues[0:4]
Risk_wt=1/np.sqrt(Eigenvalue)

stock_wt_DRP=np.zeros(10)
for j in range(10):
  stock_wt_DRP[j]=np.dot(components[0:4][:,j], Risk_wt)/sum(Risk_wt)

stock_wt_DRP_norm=stock_wt_DRP/np.sum(stock_wt_DRP)
volume_DRP=stock_wt_DRP_norm/price

Series_DRP=np.zeros(len(pricedf))
for k in range(len(pricedf)):
  Series_DRP[k]=np.dot(pricedf.drop(columns=['^GSPC']).iloc[k],volume_DRP)
```

15.

```
#15
Series_HIS=np.zeros(len(pricedf))
for k in range(len(pricedf)):
  Series_HIS[k]=pricedf["^GSPC"].iloc[k]/pricedf["^GSPC"].iloc[0]



fig,ax=plt.subplots(figsize=(12,9), dpi=300)

x=np.arange(251)
ax.plot(x, Series_EWS, label='EWS', color='blue')
ax.plot(x, Series_EWP, label='EWP', color='green')
ax.plot(x, Series_RP, label='RP', color='red')
ax.plot(x, Series_DRP, label='DRP', color='purple')
ax.plot(x, Series_HIS, label='HIS', color='black')



ax.set_xlabel('Day')
ax.set_ylabel('Value')
ax.legend()

ax.set_aspect(160)
plt.grid(True)

plt.show()
```



16.

```
#16
data = {
    'EWS': Series_EWS,
    'RP': Series_RP,
    'EWP': Series_EWP,
    'DRP': Series_DRP,
    'HIS': Series_HIS
}

df = pd.DataFrame(data)

Gain=df.iloc[250]/df.iloc[0]
SD_mean=df.std()/df.mean()
mini=df.min()

df.loc["Gain"]=Gain
df.loc["SD/mean"]=SD_mean
df.loc["Minimum"]=mini

print(df.iloc[251:254])
```

```
##               EWS        RP       EWP       DRP       HIS
## Gain     1.500074  1.314808  1.764145  2.035738  1.092980
## SD/mean  0.195793  0.166017  0.226531  0.248955  0.053717
## Minimum  0.889606  0.831102  0.955059  0.994838  0.971098
```

17.

```
df1=pd.DataFrame(data)

beta=np.zeros(5)
for i in range(5):
  beta[i]=np.cov(df1.iloc[:,i], df1.iloc[:,4])[0,1]/np.var(df1.iloc[:,4])

df.loc["beta"]=beta
print(df.iloc[251:255])
```

```
##               EWS        RP       EWP       DRP       HIS
## Gain     1.500074  1.314808  1.764145  2.035738  1.092980
## SD/mean  0.195793  0.166017  0.226531  0.248955  0.053717
## Minimum  0.889606  0.831102  0.955059  0.994838  0.971098
## beta     4.046591  3.126747  5.287739  6.449641  1.004000
```