1. (20 pts) Python environment and data

a. (5 pts) Install cvxpy

I installed 'cvxpy' to solve out Markowitz problem and 'FinanceDataReader' to call data.

```
(t_f2.2) C:₩Users₩김건우>pip list
Package                       Version
----------------------------- -------------------
absl-py                       0.9.0
arch                          4.15
argon2-cffi                   20.1.0
astunparse                    1.6.3
atomicwrites                  1.4.0
attrs                         19.3.0
backcall                      0.2.0
bayesian-optimization         1.2.0
beautifulsoup4                4.6.0
bleach                        3.1.5
blis                          0.7.4
cachetools                    4.1.1
catalogue                     2.0.1
catboost                      0.24.4
certifi                       2020.12.5
cffi                          1.14.0
chardet                       3.0.4
chart-studio                  1.1.0
click                         7.1.2
cloudpickle                   1.6.0
colorama                      0.4.3
colorlover                    0.3.0
cufflinks                     0.17.3
cvxpy                         1.1.11
cycler                        0.10.0
cymem                         2.0.5
Cython                        0.29.14
decorator                     4.4.2
defusedxml                    0.6.0
dill                          0.3.3
docopt                        0.6.2
```

1-a)

```
import pandas as pd
import numpy as np
import FinanceDataReader  as fdr
import cvxpy as cp
```

b. (5 pts) Pick 5 stocks you like. Please try not to choose companies that have shown extreme behavior (e.g. Tesla, Gamestop, ...). Download their price data can calculate their daily returns in 2018.01.01 ~ 2020.12.31.

I downloaded price data of 'Samsung, Hyundai, Sk Hynix, KaKao, Naver' by using library FinanceDataReader which is available to call data by API. In order to use them to calculate daily returns, I used data from 2017 not 2018 since we may lose first row data when we shift data by '1'.

1-b)

```
samsung_df = fdr.DataReader('005930', '2017')
hyundai_df = fdr.DataReader('005380', '2017')
sk_hynix_df = fdr.DataReader('000660', '2017')
kakao_df = fdr.DataReader('035720', '2017')
naver_df = fdr.DataReader('035420', '2017')
```

```
close_df = pd.DataFrame({'Samsung':samsung_df['Close'],
                         'Hyundai':hyundai_df['Close'],
                         'SK hynix':sk_hynix_df['Close'],
                           'KaKao' : kakao_df['Close'],
                           'Naver':naver_df['Close']})
```

```
close_df_2018 = close_df["2017-12-31":"2019-01-01"]
close_df_2019 = close_df["2018-12-31":"2020-01-01"]
close_df_2020 = close_df["2019-12-31":"2021-01-01"]
```

c. (10 pts) Calculate their expected returns and covariance matrices in years 2018, 2019, and 2020. That is, you should calculate 3 expected return vectors (in R5) and 3 covariance matrices (in R5×5). Make sure that you annualize them.

I calculated 3 expected return vectors (in R5) and 3 covariance matrices (in R5X5) each on 2018, 2019 and 2020 years. Since we lose first row data as 'None' on each log return data, I replace 'None' data to '0'. Also, to annualize expected return vectors and covariance matrices, I multiply the length of each year.

1-c)

```
#2018
log_df_2018 = np.log(close_df_2018/close_df_2018.shift(1))
log_df_2018.loc[0:1,list(log_df_2018.columns)]=0.0 # shift1하면서 생긴 none 데이터 결측치 처리

mu_2018 = log_df_2018.mean() * len(log_df_2018) #annualization 데이터가 1년단위면 길이만큼 곱해줌, 1년 보
sigma_2018 = log_df_2018.cov() * len(log_df_2018) # daily T=251, monthly T=12, weekly T=52
```

```
<ipython-input-43-2c9d58a29e84>:3: FutureWarning: Slicing a positional slice with .loc is not supported, and will raise TypeError in a future version.  Use .loc with labels or .iloc with positions instead.
  log_df_2018.loc[0:1,list(log_df_2018.columns)]=0.0 # shift1하면서 생긴 none 데이터 결측치 처리
```

```
mu_2018
```

```
Samsung    -0.276378
Hyundai    -0.232383
SK hynix   -0.235954
KaKao      -0.352336
Naver      -0.373546
dtype: float64
```

```
sigma_2018
```

	Samsung	Hyundai	SK hynix	KaKao	Naver
Samsung	0.069588	0.003577	0.060458	0.030400	0.024737
Hyundai	0.003577	0.089869	-0.004189	0.024834	0.011415
SK hynix	0.060458	-0.004189	0.124256	0.034097	0.016693
KaKao	0.030400	0.024834	0.034097	0.131883	0.048776
Naver	0.024737	0.011415	0.016693	0.048776	0.090944

20161342 KeonWooKim

```
#2019
log_df_2019 = np.log(close_df_2019/close_df_2019.shift(1))
log_df_2019.loc[0:1,list(log_df_2019.columns)]=0.0 # shift1하면서 생긴 none 데이터 결측치 처리

mu_2019 = log_df_2019.mean() * len(log_df_2019) #annualization 데이터가 1년단위면 길이만큼 곱해줌, 1년 ㄷ
sigma_2019 = log_df_2019.cov() * len(log_df_2019) # daily T=251, monthly T=12, weekly T=52
```

```
<ipython-input-44-9ad3001e2ffe>:3: FutureWarning: Slicing a positional slice with .loc is not supporte
d, and will raise TypeError in a future version.  Use .loc with labels or .iloc with positions instead.
  log_df_2019.loc[0:1,list(log_df_2019.columns)]=0.0 # shift1하면서 생긴 none 데이터 결측치 처리
```

```
mu_2019
```

```
Samsung     0.364643
Hyundai     0.055451
SK hynix    0.440063
KaKao       0.408732
Naver       0.457747
dtype: float64
```

```
sigma_2019
```

	Samsung	Hyundai	SK hynix	KaKao	Naver
Samsung	0.051675	0.012462	0.055106	0.008036	0.009020
Hyundai	0.012462	0.055022	0.017534	0.005146	0.011152
SK hynix	0.055106	0.017534	0.120113	0.007239	0.010537
KaKao	0.008036	0.005146	0.007239	0.077861	0.022141
Naver	0.009020	0.011152	0.010537	0.022141	0.101445

```
#2020
log_df_2020= np.log(close_df_2020/close_df_2020.shift(1))
log_df_2020.loc[0:1,list(log_df_2020.columns)]=0.0 # shift1하면서 생긴 none 데이터 결측치 처리

mu_2020 = log_df_2020.mean() * len(log_df_2020) #annualization 데이터가 1년단위면 길이만큼 곱해줌, 1년 ㄷ
sigma_2020 = log_df_2020.cov() * len(log_df_2020) # daily T=251, monthly T=12, weekly T=52
```

```
<ipython-input-45-b44b393f61e4>:3: FutureWarning: Slicing a positional slice with .loc is not supporte
d, and will raise TypeError in a future version.  Use .loc with labels or .iloc with positions instead.
  log_df_2020.loc[0:1,list(log_df_2020.columns)]=0.0 # shift1하면서 생긴 none 데이터 결측치 처리
```

```
mu_2020
```

```
Samsung     0.383486
Hyundai     0.486811
SK hynix    0.224199
KaKao       0.937707
Naver       0.471714
dtype: float64
```

```
sigma_2020
```

	Samsung	Hyundai	SK hynix	KaKao	Naver
Samsung	0.107974	0.088395	0.103681	0.045781	0.049261
Hyundai	0.088395	0.231081	0.096193	0.048717	0.053302
SK hynix	0.103681	0.096193	0.165009	0.058325	0.065012
KaKao	0.045781	0.048717	0.058325	0.161585	0.123059
Naver	0.049261	0.053302	0.065012	0.123059	0.155526

2. (40 pts) Classical Markowitz model

a. (10 pts) Calculate optimal portfolio weights by solving the Markowitz model with a nonnegativity constraint using expected returns and covariance matrices of year 2018.

I wrote the code a little differently from the way the professor taught us during 'implementation' lecture because the version of library I installed 'cvxpy' is little bit different so the shape of the expected return to be utilized should be different. So I reshaped the expected return by using '.reshape((5,))' after calling values of expected returns (Series -> Array). Then, I set five variables which will be used as portfolio weights on each stock. And the scaling factor lambda was set as 0.1. The return on 2018 was calculated by multiplying by transpose of expected return which I solved out on '1-c' and five variables which I set above. The risk on 2018 was calculated by using 'cvxpy' library. The objective function on Markowitz problem is set as minimizing 'risk – return on 2018'. Finally, after setting the values, I solved out objective function of Markowitz problem with two constraints which are non-negativity constraint and sum of weight equals to '1'. Thus, I solved out the expected return and standard deviation from this Markowitz problem as '-0.271' and '0.188'. And the weights on each stock is (Samsung:0.296, Hyundai:0.386, Sk Hynix:0.143, KaKao:0, Naver:0.176)

2-a)

```
mu_2018 = mu_2018.values
mu_2018 = mu_2018.reshape((5,))

w_2018 = cp.Variable(5)

lambda_ = 0.1

ret_2018 = mu_2018.T * w_2018
risk_2018 = cp.quad_form(w_2018,sigma_2018)

obj_2018 =cp.Minimize(risk_2018-lambda_*ret_2018)

prob_2018 = cp.Problem(obj_2018,
                 [sum(w_2018)==1,
                 w_2018>=0])

prob_2018.solve()
print('expected return', mu_2018.dot(w_2018.value)) # expected retrun #u^T * w^*(optimal one)
print('standard deviation',np.sqrt(w_2018.value.dot(sigma_2018).dot(w_2018.value)))
print('weights', w_2018.value)
```

```
expected return -0.27068996187790 69
standard deviation 0.18766886625011261
weights [2.95450148e-01 3.85809049e-01 1.43074397e-01 2.98496450e-23
 1.75666406e-01]
```

```
C:₩ana3₩envs₩t_f2.2₩lib₩site-packages₩cvxpy₩expressions₩expression.py:556: UserWarning:
This use of ``*`` has resulted in matrix multiplication.
Using ``*`` for matrix multiplication has been deprecated since CVXPY 1.1.
    Use ``*`` for matrix-scalar and vector-scalar multiplication.
    Use ``@`` for matrix-matrix and matrix-vector multiplication.
    Use ``multiply`` for elementwise multiplication.
This code path has been hit 3 times so far.

  warnings.warn(msg, UserWarning)
```

b. (10 pts) Suppose that you make investment using the portfolio you calculated in Problem 2-a. You invest $100 from 2019.01.01 to 2020.12.31 without rebalancing (i.e. buy-and-hold). How much would you have at the end of the year? Also, calculate annual return and annual return standard deviation.0020

Since we use buy-and-hold method investment, I solved out the value of each stock's return percentage and multiplied it with each following stock's optimal weight and initial investment cost ($100). Thus, since I invest $100 from 2019.01.01, the amount of current money on 2020.12.31 is calculated by adding each stock's gain and initial investment cost ($100). Then, the current money on 2020.12.31 is ($198.259). Also, the annual return on my account is calculated by 'A*(1+r)^t = B (A: invested money($100), r: annual return, t: number of years(2), B: current money($198.259)), so the value of annual return is 40.804%. And the annual return standard deviation is calculated by $(w^T \Sigma w)^{1/2}$ and while calculating covariance we should multiply average number of trading days on 2019 and 2020 in a year, so the value of annual return standard deviation is 0.262.

2-b) ¶

```
gain_list = []
for i in range(5):
    gap_price = close_df_2020.iloc[-1][i] - close_df_2019.iloc[0][i] # (20년 가격 - 19년 가격)
    return_pct = (gap_price / close_df_2019.iloc[0][i]) # (20년 가격 - 19년 가격)/(19년 가격)
    print(return_pct)
    #gain_list.append(return_pct * w_2018.value[i]*100) # 수익률 x 비중
    gain_list.append((1+return_pct) * w_2018.value[i]*100)
```

```
1.0903225806451613
0.6842105263157895
0.9554455445544554
2.8186391833146094
1.478813559322034
```

```
# 2019년 1월1일에 100$ 주어 놓아서, 2020년 12월31일에 198$ 됐음
all_gain = 0
for rtn in gain_list:
    all_gain += rtn
all_gain
```

```
198.2588238597681
```

```
all_ann_rtn = np.sqrt(all_gain/100)-1
all_ann_rtn
```

```
0.4080441181290029
```

```
# annual return standard deviation
days = int(len(log_df_2019) + len(log_df_2020))/2

concat_df = pd.concat([log_df_2019,log_df_2020])
sdv = np.sqrt(w_2018.value.dot(concat_df.cov()*days).dot(w_2018.value))
sdv
```

```
0.2623031924043395
```

c. (10 pts) Plot daily growth of your investment in 2-b. (i.e., horizontal axis: days, vertical axis: account value)

With having $100, I invested on each stock on optimal portfolio weight solved out on problem 2-a. Thus, I invested Samsung about $29.6, Hyundai about $38.6, Sk Hynix about $14.3, Kakao about $0 and Naver about 17.6$. And I used a dictionary structure to show the change of my total account value. For each day, I applied daily return of each stock starting from the price value written above. Then, I added them all and treat it as one price, and do the same thing for number of days from 2019.01.01 to 2020.12.31. Its graph of daily growth on my investment is calculated by using log return.

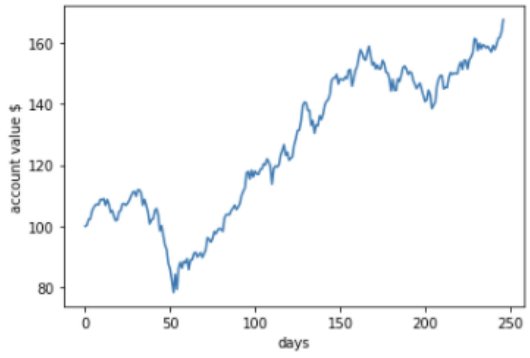
2-c)

```
invested_money = []
for w in w_2018.value:
    invested_money.append(100*w)
invested_money
```

```
[29.545014805660806,
 38.5809048824135,
 14.3074397146635,
 2.9849645044896876e-21,
 17.566640597262197]
```

```
invested_ = {}
close_df = close_df['2018-12-28':'2020-12-31']
log_close = np.log(close_df/close_df.shift(1))
log_close = log_close.dropna()
for i in range(len(log_close)):
    invested_money[0] = invested_money[0] * (1+log_close['Samsung'].iloc[i])
    invested_money[1] = invested_money[1] * (1+log_close['Hyundai'].iloc[i])
    invested_money[2] = invested_money[2] * (1+log_close['SK hynix'].iloc[i])
    invested_money[3] = invested_money[3] * (1+log_close['KaKao'].iloc[i])
    invested_money[4] = invested_money[4] * (1+log_close['Naver'].iloc[i])
    invested_[i] = invested_money[0] +invested_money[1] +invested_money[2] +invested_money[3] + invested_money[4]
```

```
import matplotlib.pyplot as plt
plt.plot(range(len(invested_)), invested_.values())
plt.xlabel('days')
plt.ylabel('account value $')
plt.show()
```



d. (10 pts) Repeat a, b, and c for year 2020. That is, you make investment in 2020 based on the optimal portfolio calculated using parameters estimated in year 2019.

First of all, the optimal weights calculated from Markowitz Problem are (Samsung: 0.42, Hyundai: 0.06, SK Hynix: 0.03, Kakao: 0.279 and Naver: 0.216). If I made investment using this portfolio with initial investment cost $100 on 2020.01.01, the amount of current money on 2020.12.31 is calculated by adding each stock's gain and initial investment cost. Thus, the current money on 2020.12.31 is ($180.231). Also, the annual return on my account is calculated by 'A*(1+r)^t = B (A: invested money($100), r: annual return, t: number of years(1), B: current money($180.231)), so the value of total annual return is 80.231%. And the annual return standard deviation is calculated by $(w^T \Sigma w)^{1/2}$ and while calculating covariance we should multiply number of trading days in 2020, and the value of annual return standard deviation is 0.293. And as same method on 2-c, I plot daily growth of my investment.

2-d)

```
mu_2019 = mu_2019.values
mu_2019 = mu_2019.reshape((5,))

w_2019 = cp.Variable(5)

lambda_ =0.1

ret_2019 = mu_2019.T * w_2019
risk_2019 = cp.quad_form(w_2019,sigma_2019)

obj_2019 =cp.Minimize(risk_2019-lambda_*ret_2019)

prob_2019 = cp.Problem(obj_2019,
                [sum(w_2019)==1,
                w_2019>=0])

prob_2019.solve()
print('expected return', mu_2019.dot(w_2019.value))
print('standard deviation',np.sqrt(w_2019.value.dot(sigma_2019).dot(w_2019.value)))
```

```
expected return 0.381769478878326
standard deviation 0.1703546643717708
```

```
C:\ana3\envs\t_f2.2\lib\site-packages\cvxpy\expressions\expression.py:556: UserWarning:
This use of ``*`` has resulted in matrix multiplication.
Using ``*`` for matrix multiplication has been deprecated since CVXPY 1.1.
    Use ``*`` for matrix-scalar and vector-scalar multiplication.
    Use ``@`` for matrix-matrix and matrix-vector multiplication.
    Use ``multiply`` for elementwise multiplication.
This code path has been hit 2 times so far.

  warnings.warn(msg, UserWarning)
```

```
w_2019.value
```

```
array([0.41840464, 0.05673786, 0.02987155, 0.27886979, 0.21611616])
```

```
gain_list = []
for i in range(5):
    gap_price = close_df_2020.iloc[-1][i] - close_df_2020.iloc[0][i] # (20년 1월 가격 - 20년  12월 가격)
    return_pct = (gap_price / close_df_2020.iloc[0][i])# (20년 1월 가격 - 20년  12월 가격)/(19년 가격)
    print(return_pct)
    gain_list.append(return_pct * w_2019.value[i]*100)# 수익률 x 비중
```

```
0.4673913043478261
0.6271186440677966
0.25131995776135163
1.5541180698487373
0.6027397260273972
```

```
# 2020년 1월1일에 100$ 집어 넣어서, 2020년 12월31일에 180$ 됐을
all_gain = 0
for rtn in gain_list:
    all_gain += rtn
current_money = 100 + all_gain # 100은 원래 자본, all_gain은 수익
current_money
```

180.23057498946946

```
all_ann_rtn = (current_money/100)-1
all_ann_rtn
```

0.8023057498946946

```
# annual return standard deviation
days = int(len(log_df_2020))

sdv = np.sqrt(w_2019.value.dot(log_df_2020.cov()*days).dot(w_2019.value))
sdv
```

0.29295591875839316

```
invested_money = []
for w in w_2019.value:
    invested_money.append(100*w)
invested_money
```

```
[41.84046422127436,
 5.673785625555312,
 2.987155495982261,
 27.886979024856384,
 21.611615632331677]
```

```
invested_ = {}
for i in range(1,len(log_df_2020)):
    invested_money[0] = invested_money[0] * (1+log_df_2020['Samsung'].iloc[i])
    invested_money[1] = invested_money[1] * (1+log_df_2020['Hyundai'].iloc[i])
    invested_money[2] = invested_money[2] * (1+log_df_2020['SK hynix'].iloc[i])
    invested_money[3] = invested_money[3] * (1+log_df_2020['KaKao'].iloc[i])
    invested_money[4] = invested_money[4] * (1+log_df_2020['Naver'].iloc[i])
    invested_[i] = invested_money[0] +invested_money[1] +invested_money[2] +invested_money[3] + invested_money[4]
```

```
plt.plot(range(len(log_df_2020)-1), invested_.values())
plt.xlabel('days')
plt.ylabel('account value $')
plt.show()
```