testing-copy-of-gwp3

November 7, 2023

1 STEP 2: Pseudocode of the multi-armed bandit problem

Define

Define N (number of trials)

Download historical data for the prices of a list of stocks that we are interested in

Turn the prices into returns.

Estimate the correlation structure and risk level.

Filter the list of the stocks to select a basket of K assets

For 1 to N do:

Choose portfolio of stocks and give them weights.

Find the returns that this portfolio gives.

[ ]:

2 STEP 3: Collect data

2.1 Financial companies

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import yfinance as yf
import seaborn as sns
from numpy.random import rand, seed
```

Choose tickers

[ ]:
```
tickers_fin = ["JPM","WFC","BAC", "C", "GS", "USB", "MS", "KEY", "PNC", "COF",␣
↪"AXP",
          "PRU", "SCHW"]
```

Collect data for the period Sept and Oct 2008

[ ]:
```
df_fin = pd.DataFrame()
for i in tickers_fin:
  ydata = yf.download(i, start = '2008-09-01', end = '2008-11-01')
  df_fin[i] = ydata['Adj Close']
df_fin.index = pd.to_datetime(ydata.index, format='%Y%m%d')
df_fin.head()
```

```
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
```

[ ]:

Date	JPM	WFC	BAC	C	GS	\
2008-09-02	26.480583	20.872437	25.531290	148.928192	129.483078	
2008-09-03	26.969582	20.738686	26.305449	152.824829	131.276657	
2008-09-04	25.747093	19.842529	24.421926	142.615677	126.021240	
2008-09-05	26.894867	20.865753	25.722836	148.616455	127.853897	
2008-09-08	28.219240	22.444059	27.718094	158.357986	132.937119	

Date	USB	MS	KEY	PNC	COF	AXP	\
2008-09-02	21.174292	30.126410	8.322878	49.117054	34.988831	31.773907	
2008-09-03	21.553684	30.761030	8.395541	49.571815	35.565228	31.992884	
2008-09-04	20.703318	29.426130	7.873707	48.655552	33.750340	30.303696	
2008-09-05	21.416332	30.170176	8.567280	49.685520	34.825249	30.812017	
2008-09-08	22.201281	31.563429	9.075903	51.350849	37.956490	31.687895	

Date	PRU	SCHW
2008-09-02	44.671402	19.997490
2008-09-03	45.859718	20.022301
2008-09-04	44.394524	19.484735
2008-09-05	45.421326	19.923058
2008-09-08	48.986256	20.857588

2.2 Non-financial companies

[ ]:
```
tickers_nfin = ["KR", "PFE", "XOM", "WMT", "DAL", "CSCO", "EQIX", "DUK",
                "NFLX", "GE", "APA", "F", "REGN", "CMS"]
```

[ ]:
```
df_nfin = pd.DataFrame()
for i in tickers_nfin:
  ydata = yf.download(i, start = '2008-09-01', end = '2008-11-01')
  df_nfin[i] = ydata['Adj Close']
df_nfin.index = pd.to_datetime(ydata.index, format='%Y%m%d')
df_nfin.head()
```

```
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
```

[ ]:

Date	KR	PFE	XOM	WMT	DAL	CSCO
2008-09-02	10.590397	10.190771	44.533047	42.738346	8.105466	16.363979
2008-09-03	10.586588	10.206717	44.936188	42.838665	8.052430	16.060818
2008-09-04	10.380877	9.924971	43.853394	42.831505	7.919844	15.351137
2008-09-05	10.236117	9.839912	43.553890	43.519333	7.787258	15.337356
2008-09-08	10.399925	10.174821	44.216248	44.422092	7.592797	16.102158

Date	EQIX	DUK	NFLX	GE	APA	F
2008-09-02	63.139462	26.190973	4.405714	115.751549	84.531815	2.665931
2008-09-03	62.515472	25.858681	4.415714	115.913864	85.438019	2.701398
2008-09-04	60.448521	26.115454	4.267143	112.384117	87.465034	2.594997
2008-09-05	60.440723	26.145662	4.237143	113.114403	88.577942	2.606820
2008-09-08	60.456345	26.991510	4.307143	118.023643	87.258400	2.689576

Date	REGN	CMS
2008-09-02	20.580000	8.152999
2008-09-03	21.799999	8.057152
2008-09-04	20.379999	8.069131

```
2008-09-05  19.070000  7.991254
2008-09-08  18.900000  8.236865
```

Couldn't get data for non financial tickers("HCP"), and from financial tickers("BBT", "STI").

2.3 Compute financial returns

Financial Companies

[ ]:
```
df_fin_returns = df_fin.pct_change(axis=0) # daily returns
df_fin_returns = df_fin_returns.dropna()
df_fin_returns
```

[ ]:

Date	JPM	WFC	BAC	C	GS	USB
2008-09-03	0.018466	-0.006408	0.030322	0.026165	0.013852	0.017918
2008-09-04	-0.045328	-0.043212	-0.071602	-0.066803	-0.040033	-0.039453
2008-09-05	0.044579	0.051567	0.053268	0.042077	0.014542	0.034440
2008-09-08	0.049243	0.075641	0.077568	0.065548	0.039758	0.036652
2008-09-09	-0.050060	-0.071216	-0.063634	-0.070866	-0.047488	-0.055392
2008-09-10	-0.001773	0.017004	-0.003690	-0.010593	-0.025236	-0.006551
2008-09-11	0.057107	0.067824	0.020370	-0.003748	-0.003554	0.056201
2008-09-12	-0.011525	0.012998	0.020569	-0.034927	-0.017959	0.005648
2008-09-15	-0.101287	-0.095946	-0.213101	-0.151448	-0.121328	-0.023943
2008-09-16	0.101081	0.126774	0.112995	0.033465	-0.018377	0.068141
2008-09-17	-0.121993	-0.042943	-0.079526	-0.109207	-0.139162	-0.054721
2008-09-18	0.126642	0.106791	0.124264	0.186743	-0.056769	0.102880
2008-09-19	0.167494	0.075676	0.225638	0.240240	0.201853	0.033179
2008-09-22	-0.132838	-0.116081	-0.088847	-0.030993	-0.069492	-0.078705
2008-09-23	-0.005882	-0.028709	-0.024890	-0.000999	0.035353	-0.028286
2008-09-24	-0.001480	0.002926	-0.006907	-0.051526	0.063574	-0.004705
2008-09-25	0.073086	-0.004377	0.039310	0.023734	0.018798	0.043132
2008-09-26	0.109986	0.093494	0.067792	0.038125	0.018376	0.037696
2008-09-29	-0.150083	-0.108818	-0.175749	-0.119106	-0.125299	-0.095304
2008-09-30	0.139025	0.128722	0.157025	0.155493	0.060481	0.099847
2008-10-01	0.062741	-0.022116	0.089429	0.121404	0.050781	0.018323
2008-10-02	0.012182	-0.041962	-0.046158	-0.021739	-0.022007	-0.010088
2008-10-03	-0.079238	-0.017065	-0.051966	-0.184445	-0.026912	-0.034426
2008-10-06	-0.041394	-0.026620	-0.065545	-0.051226	-0.031250	-0.007416
2008-10-07	-0.106364	-0.090369	-0.262260	-0.129811	-0.072580	-0.086207
2008-10-08	-0.000509	0.042484	-0.070257	-0.049505	-0.017392	-0.030503
2008-10-09	-0.066667	-0.145768	-0.111765	-0.102084	-0.103097	-0.061628
2008-10-10	0.135224	0.038899	0.063169	0.091261	-0.123828	0.049429
2008-10-13	0.008405	0.073825	0.091998	0.116230	0.250000	0.029315
2008-10-14	-0.030483	0.102632	0.164107	0.182222	0.107207	0.006720
2008-10-15	-0.054532	-0.005072	-0.102149	-0.128357	-0.079333	-0.042276
2008-10-16	0.051961	0.016492	0.018052	-0.020333	-0.006628	0.024561
2008-10-17	-0.028649	-0.054277	-0.041650	-0.064151	0.016904	-0.010366

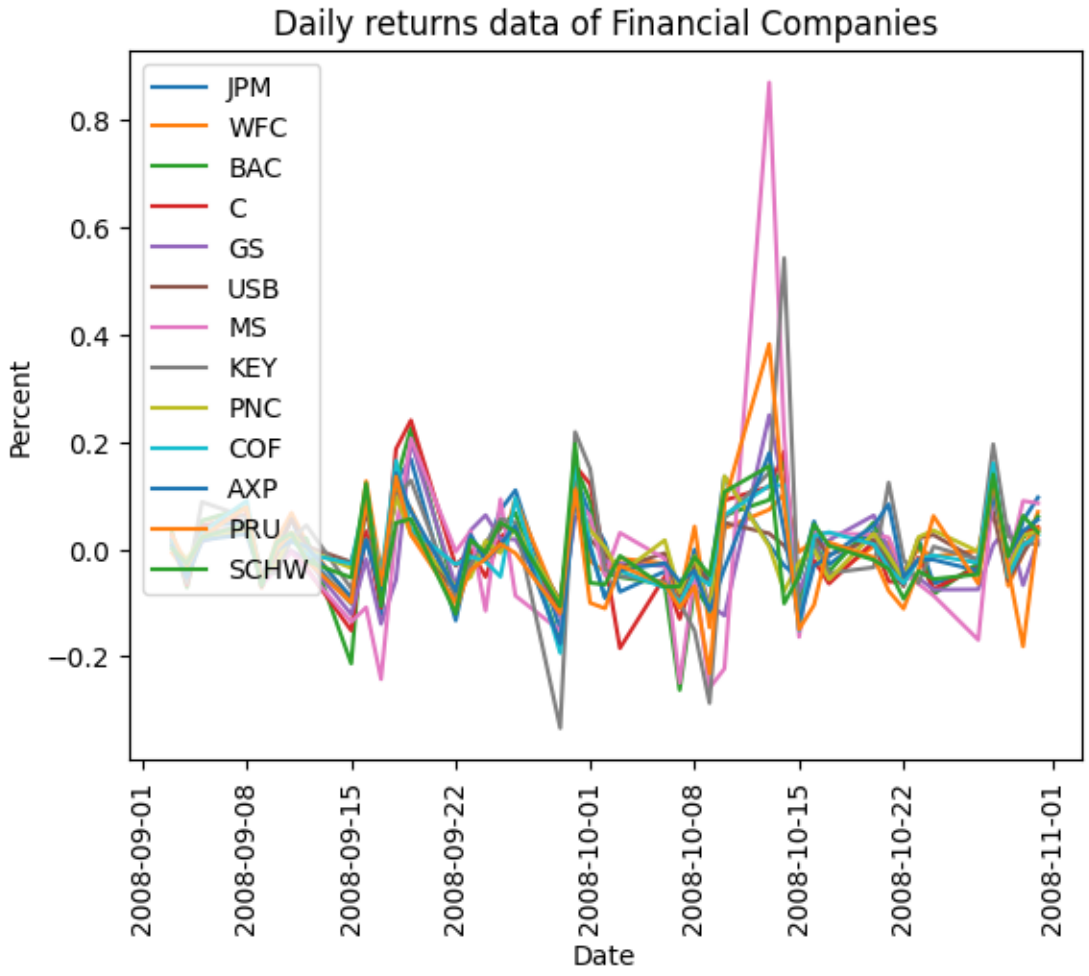
2008-10-20	0.033816	0.005303	0.049915	0.014113	0.062992	0.018658
2008-10-21	-0.022626	0.012721	-0.017623	-0.060304	-0.001235	-0.029563
2008-10-22	-0.064671	-0.041054	-0.054651	-0.060649	-0.051833	-0.069205
2008-10-23	0.018294	0.000959	0.015004	-0.015765	-0.053439	0.020278
2008-10-24	-0.063936	-0.013406	-0.083913	-0.073989	-0.075336	0.026848
2008-10-27	-0.040361	-0.002588	-0.025629	-0.033773	-0.074901	-0.021392
2008-10-28	0.105883	0.117743	0.121287	0.143222	0.007429	0.069396
2008-10-29	-0.050266	-0.068195	-0.030408	-0.037286	0.043711	-0.058079
2008-10-30	0.053487	-0.008408	0.020609	0.028235	-0.067070	-0.007923
2008-10-31	0.096491	0.069410	0.061019	0.041190	0.015257	0.035070

Date	MS	KEY	PNC	COF	AXP	PRU	\
2008-09-03	0.021065	0.008731	0.009259	0.016474	0.006892	0.026601	
2008-09-04	-0.043396	-0.062156	-0.018484	-0.051030	-0.052799	-0.031949	
2008-09-05	0.025285	0.088087	0.021169	0.031849	0.016774	0.023129	
2008-09-08	0.046180	0.059368	0.033517	0.089913	0.028426	0.078486	
2008-09-09	-0.066327	-0.042940	-0.051185	-0.066284	-0.056269	-0.060292	
2008-09-10	-0.036634	-0.044867	-0.027316	0.001758	-0.002092	-0.000251	
2008-09-11	-0.005396	0.019108	0.030482	0.016674	0.015723	0.031086	
2008-09-12	-0.038233	0.044531	-0.000684	-0.008632	0.004903	-0.020301	
2008-09-15	-0.135375	-0.073298	-0.034946	-0.026992	-0.089089	-0.099889	
2008-09-16	-0.108419	0.019370	0.090315	0.097092	0.016629	0.091536	
2008-09-17	-0.242160	-0.048298	-0.062777	-0.055057	-0.084003	-0.065926	
2008-09-18	0.036781	0.097338	0.097832	0.165300	0.141647	0.135614	
2008-09-19	0.206652	0.128128	0.027974	0.037963	0.071050	0.026908	
2008-09-22	-0.004410	-0.092742	-0.069080	-0.028546	-0.076980	-0.098435	
2008-09-23	0.033592	-0.028148	-0.051852	-0.013407	0.026549	-0.039095	
2008-09-24	-0.114643	0.009909	0.015764	-0.022896	-0.020376	-0.017933	
2008-09-25	0.093183	0.056603	-0.007004	-0.051248	0.008800	0.009403	
2008-09-26	-0.086716	0.050000	0.044260	0.095381	0.044145	-0.007425	
2008-09-29	-0.151919	-0.333334	-0.099338	-0.193400	-0.175949	-0.119287	
2008-09-30	0.095760	0.218368	0.098529	0.159090	0.088479	0.111969	
2008-10-01	0.061739	0.149916	0.036145	0.008235	0.002553	-0.100000	
2008-10-02	-0.049550	-0.037873	0.003617	-0.090237	-0.090549	-0.110339	
2008-10-03	0.030591	-0.049962	-0.045443	-0.040188	-0.039515	-0.030876	
2008-10-06	-0.017559	-0.060558	0.016858	-0.071492	-0.025915	-0.065689	
2008-10-07	-0.248936	-0.100085	-0.080902	-0.099065	-0.060525	-0.109003	
2008-10-08	-0.048159	-0.151744	-0.012966	-0.047124	-0.040000	-0.069232	
2008-10-09	-0.258929	-0.286667	-0.116605	-0.066499	-0.115044	-0.231462	
2008-10-10	-0.222490	0.057633	0.136174	0.063753	-0.035417	0.085963	
2008-10-13	0.869835	0.142857	0.005147	0.118176	0.179265	0.382507	
2008-10-14	0.212155	0.542526	-0.077688	0.119024	0.032601	0.083083	
2008-10-15	-0.163360	-0.130326	-0.024429	-0.150889	-0.134090	-0.149168	
2008-10-16	0.030888	0.013449	-0.001626	0.024895	-0.031544	-0.103845	
2008-10-17	0.029427	-0.044550	-0.056351	0.031524	-0.013114	0.012848	
2008-10-20	0.027547	-0.033730	0.029686	0.009018	0.043721	-0.015079	

```
2008-10-21  0.021750  0.124230 -0.011230 -0.034757  0.083778 -0.077521
2008-10-22 -0.043069 -0.018265 -0.057128 -0.063786 -0.051913 -0.110906
2008-10-23 -0.064666 -0.063256  0.022653 -0.018956 -0.019585 -0.038518
2008-10-24 -0.086283  0.004965  0.035162 -0.011481 -0.019568  0.061941
2008-10-27 -0.168886 -0.019763 -0.004246 -0.025496 -0.040333 -0.064132
2008-10-28  0.107065  0.195564  0.116152  0.159883  0.103553  0.131783
2008-10-29 -0.028947  0.024452 -0.040037 -0.051378 -0.010208 -0.034246
2008-10-30  0.090109  0.012346  0.026265  0.007133  0.033717 -0.180993
2008-10-31  0.085768  0.008943  0.034124  0.026233  0.055258  0.039141

                SCHW
Date
2008-09-03  0.001241
2008-09-04 -0.026848
2008-09-05  0.022496
2008-09-08  0.046907
2008-09-09 -0.068596
2008-09-10  0.012772
2008-09-11  0.029844
2008-09-12 -0.021633
2008-09-15 -0.052983
2008-09-16  0.122467
2008-09-17 -0.105966
2008-09-18  0.049166
2008-09-19  0.056486
2008-09-22 -0.120396
2008-09-23  0.017560
2008-09-24 -0.006637
2008-09-25  0.049888
2008-09-26  0.031820
2008-09-29 -0.106497
2008-09-30  0.196502
2008-10-01 -0.062692
2008-10-02 -0.065654
2008-10-03 -0.012297
2008-10-06 -0.070698
2008-10-07 -0.068900
2008-10-08 -0.014388
2008-10-09 -0.048488
2008-10-10  0.106849
2008-10-13  0.154951
2008-10-14 -0.101157
2008-10-15 -0.049118
2008-10-16  0.046640
2008-10-17  0.006229
2008-10-20 -0.020000
2008-10-21 -0.036928
```

```
2008-10-22 -0.092331
2008-10-23 -0.041134
2008-10-24 -0.055652
2008-10-27 -0.046655
2008-10-28  0.139730
2008-10-29 -0.014124
2008-10-30  0.062464
2008-10-31  0.031283
```

[ ]:
```
plt.plot(df_fin_returns.index,df_fin_returns)
plt.legend(tickers_fin, loc='upper left')
plt.xlabel("Date")
plt.ylabel("Percent")
plt.title("Daily returns data of Financial Companies")
plt.xticks(rotation=90)
plt.show()
```



Non-financial companies

```
[ ]: df_nfin_returns = df_nfin.pct_change(axis=0) # daily returns
     df_nfin_returns = df_nfin_returns.dropna()
     df_nfin_returns
```

```
[ ]:                  KR       PFE       XOM       WMT       DAL      CSCO  \
     Date
     2008-09-03 -0.000360  0.001565  0.009053  0.002347 -0.006543 -0.018526
     2008-09-04 -0.019431 -0.027604 -0.024096 -0.000167 -0.016465 -0.044187
     2008-09-05 -0.013945 -0.008570 -0.006830  0.016059 -0.016741 -0.000898
     2008-09-08  0.016003  0.034036  0.015208  0.020744 -0.024972  0.049865
     2008-09-09  0.023810 -0.047022 -0.045721 -0.014032 -0.041909 -0.017116
     2008-09-10  0.001073 -0.002193  0.027163  0.014559 -0.043743  0.000871
     2008-09-11  0.007863  0.010440  0.004120  0.018542  0.027954  0.002610
     2008-09-12 -0.010993  0.012507  0.025675 -0.012031  0.004944  0.017788
     2008-09-15 -0.046611 -0.030612 -0.054839 -0.012498 -0.009840 -0.046036
     2008-09-16  0.052651 -0.013296  0.043413  0.008275  0.234782  0.018767
     2008-09-17 -0.038942 -0.035935 -0.015047 -0.040231 -0.094567 -0.043860
     2008-09-18 -0.002974  0.047758  0.032811  0.030852  0.050000  0.045872
     2008-09-19  0.008203  0.031128  0.023923 -0.028953  0.023281  0.065351
     2008-09-22 -0.011835 -0.025876 -0.009169 -0.013568 -0.100311 -0.048579
     2008-09-23 -0.003742 -0.003321 -0.015087 -0.008321  0.017242 -0.016443
     2008-09-24 -0.010143 -0.001110  0.004377  0.008904 -0.092656  0.003080
     2008-09-25  0.020114  0.036131  0.033833  0.020367  0.003736  0.029824
     2008-09-26  0.036087  0.001073 -0.000248  0.009814 -0.013648  0.014480
     2008-09-29 -0.036625 -0.054126 -0.081711 -0.037226 -0.077988 -0.085222
     2008-09-30  0.024227  0.044759  0.048609  0.024637  0.016371  0.035337
     2008-10-01  0.002911  0.027115  0.011847 -0.003841  0.147651 -0.027039
     2008-10-02  0.007983 -0.007920 -0.013744 -0.013577 -0.074854 -0.034168
     2008-10-03 -0.020879  0.011176  0.005677  0.014953 -0.006321  0.002359
     2008-10-06 -0.025000 -0.003158 -0.007954 -0.030638 -0.078880 -0.037177
     2008-10-07 -0.028657 -0.068110 -0.016167 -0.052850 -0.211326 -0.079179
     2008-10-08 -0.034161 -0.029462  0.012226 -0.005288 -0.012259 -0.026539
     2008-10-09 -0.032155 -0.085231 -0.116884 -0.057928  0.003547 -0.062705
     2008-10-10 -0.012458 -0.033822 -0.082941 -0.008562  0.065371  0.002327
     2008-10-13  0.066863  0.101717  0.171905  0.069676  0.127695  0.118398
     2008-10-14  0.011825  0.026979 -0.008484 -0.001101  0.080882 -0.037882
     2008-10-15 -0.043241 -0.050204 -0.139526 -0.080639  0.012245 -0.106257
     2008-10-16  0.070440  0.043024  0.113874  0.091309  0.188172  0.071213
     2008-10-17 -0.040319 -0.003536 -0.020302 -0.015562  0.039593  0.009014
     2008-10-20  0.037257  0.025429  0.102145  0.012274  0.023939  0.058626
     2008-10-21  0.002293  0.000000 -0.046540 -0.013962  0.037194 -0.058017
     2008-10-22  0.022874 -0.034602 -0.096923 -0.026085  0.017418 -0.025196
     2008-10-23  0.004846  0.021505  0.090135  0.009374 -0.113797 -0.009764
     2008-10-24 -0.021143 -0.030994 -0.019179 -0.025777 -0.055682 -0.053944
     2008-10-27 -0.034104 -0.010863 -0.042729 -0.033657 -0.078219 -0.013489
```

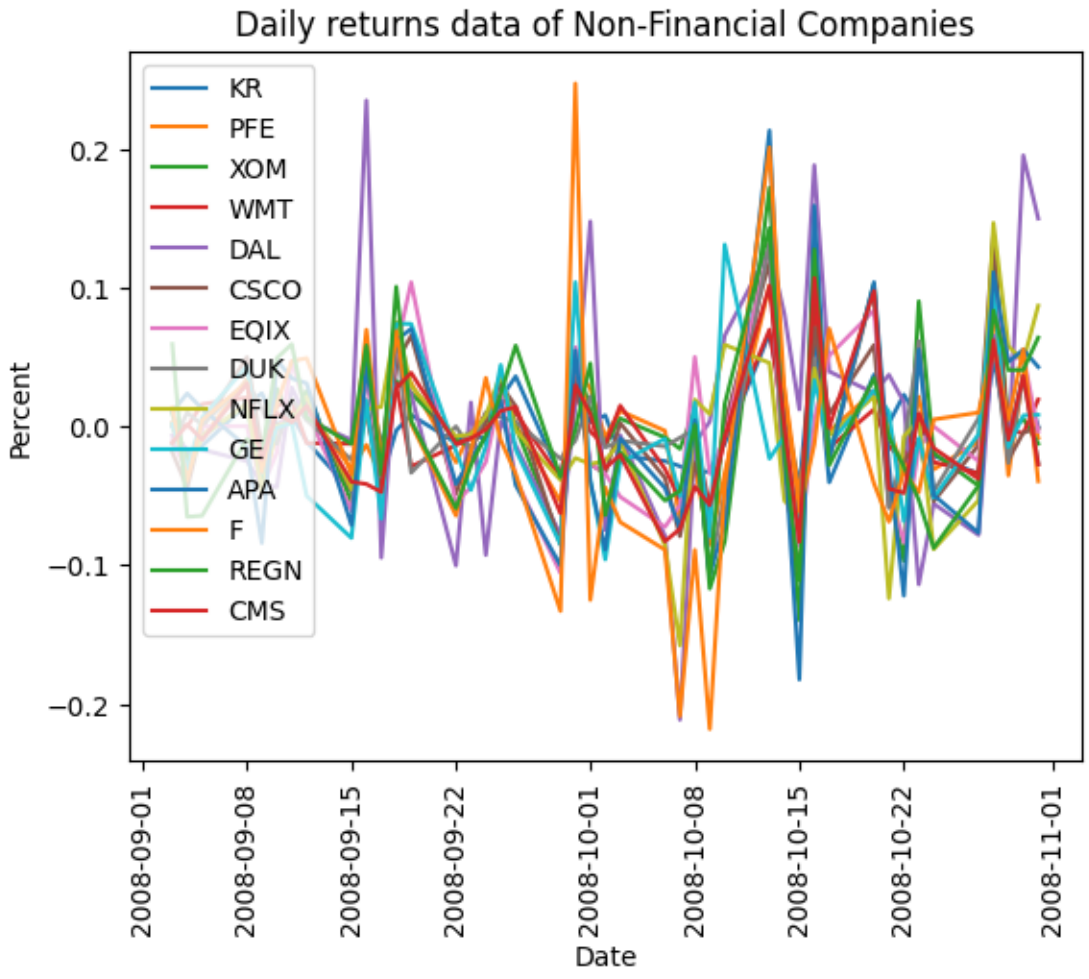
```
2008-10-28  0.049039  0.087248  0.132698  0.110731  0.065274  0.137974
2008-10-29 -0.008975 -0.035353 -0.002805 -0.002719 -0.020833 -0.024030
2008-10-30  0.036604  0.038976  0.005358 -0.004907  0.195244 -0.004477
2008-10-31 -0.000365 -0.008398 -0.012392  0.019361  0.149739 -0.001124

                EQIX       DUK      NFLX        GE       APA         F  \
Date
2008-09-03 -0.009883 -0.012687  0.002270  0.001402  0.010720  0.013304
2008-09-04 -0.033063  0.009930 -0.033646 -0.030451  0.023725 -0.039387
2008-09-05 -0.000129  0.001157 -0.007030  0.006498  0.012724  0.004556
2008-09-08  0.000258  0.032351  0.016521  0.043401 -0.014897  0.031746
2008-09-09 -0.044253  0.012870 -0.046766 -0.033345 -0.083994 -0.032967
2008-09-10 -0.019034 -0.012707  0.000348 -0.001067  0.047837  0.015909
2008-09-11  0.047062  0.019586  0.002782  0.002492  0.034643  0.046980
2008-09-12  0.011040  0.004391  0.021506 -0.050071  0.031006  0.049145
2008-09-15 -0.060835 -0.024044 -0.045501 -0.080374 -0.071181 -0.034623
2008-09-16  0.036263  0.010078  0.012451  0.018700  0.042629  0.069620
2008-09-17 -0.029651 -0.036586  0.014055 -0.066641 -0.020764 -0.025641
2008-09-18  0.055058  0.069621  0.075537  0.074090  0.062019  0.068826
2008-09-19  0.103849 -0.033351  0.028995  0.073820  0.070324  0.001894
2008-09-22 -0.055194  0.000000 -0.007201 -0.017656 -0.041436 -0.064272
2008-09-23 -0.043533 -0.016694 -0.005676 -0.045889 -0.023422 -0.018182
2008-09-24 -0.025634  0.010187  0.007612 -0.014429 -0.007231  0.034979
2008-09-25  0.025101  0.006723  0.037772  0.044327  0.019007 -0.009940
2008-09-26 -0.037842  0.003339 -0.003033 -0.016745 -0.042273 -0.034136
2008-09-29 -0.105743 -0.023295 -0.038637 -0.085148 -0.100200 -0.133056
2008-09-30  0.057069 -0.010221 -0.022785  0.103897  0.054718  0.247002
2008-10-01 -0.026922  0.021801 -0.027202 -0.039216 -0.039605 -0.125000
2008-10-02 -0.034029 -0.015160 -0.024634 -0.095919 -0.088767 -0.043956
2008-10-03 -0.050697 -0.010262 -0.011263 -0.026185 -0.008547 -0.068966
2008-10-06 -0.072443 -0.014401 -0.085606 -0.008808 -0.044430 -0.088889
2008-10-07 -0.056706 -0.008183 -0.157795 -0.050515 -0.073560 -0.208672
2008-10-08  0.049972 -0.002357  0.019274  0.017242  0.004370 -0.089041
2008-10-09 -0.034071 -0.105139  0.008795 -0.079419 -0.107520 -0.218045
2008-10-10 -0.007636 -0.047525  0.058849  0.130984 -0.069638 -0.043269
2008-10-13  0.130634  0.133056  0.045698 -0.023256  0.213173  0.201005
2008-10-14 -0.048452 -0.038532 -0.053543 -0.007143 -0.014808  0.025104
2008-10-15 -0.085831 -0.057888 -0.063644 -0.076739 -0.182364 -0.061225
2008-10-16  0.005216  0.048616  0.041759  0.033247  0.159007 -0.013043
2008-10-17  0.050963 -0.024469 -0.006397 -0.013072 -0.023526  0.070484
2008-10-20  0.083583  0.100990  0.021459  0.025981  0.103893 -0.041152
2008-10-21 -0.043938 -0.058753 -0.123950  0.010427 -0.027645 -0.068670
2008-10-22 -0.083745 -0.015924 -0.007194 -0.068305 -0.121936 -0.032258
2008-10-23 -0.015419  0.061489  0.004348 -0.008439  0.055260 -0.047619
2008-10-24  0.000000 -0.046952 -0.088504 -0.051596 -0.049502  0.005000
2008-10-27 -0.025094  0.005759 -0.053298 -0.005608 -0.076614  0.009950
2008-10-28  0.101413  0.061068  0.146600  0.099267  0.110939  0.059113
```

```
2008-10-29  0.047443 -0.026379  0.057851 -0.014880  0.046293  0.004651
2008-10-30  0.050999  0.036330  0.046875  0.007813  0.055340  0.055556
2008-10-31 -0.003671 -0.026737  0.086918  0.008269  0.042812 -0.039474
```

```
                REGN       CMS
Date
2008-09-03  0.059281 -0.011756
2008-09-04 -0.065138  0.001487
2008-09-05 -0.064279 -0.009651
2008-09-08 -0.008915  0.030735
2008-09-09 -0.020106 -0.024000
2008-09-10  0.049136  0.009687
2008-09-11  0.059187  0.002952
2008-09-12  0.006317  0.014716
2008-09-15 -0.012554 -0.039883
2008-09-16  0.058191 -0.041542
2008-09-17 -0.035120 -0.047281
2008-09-18  0.100575  0.027295
2008-09-19  0.003916  0.038648
2008-09-22 -0.058951 -0.012403
2008-09-23 -0.028098 -0.008634
2008-09-24 -0.006161 -0.002375
2008-09-25  0.023367  0.011111
2008-09-26  0.058248  0.014128
2008-09-29 -0.036548 -0.062693
2008-09-30 -0.002285  0.029728
2008-10-01  0.045350  0.008821
2008-10-02 -0.063979 -0.030207
2008-10-03 -0.018727 -0.020492
2008-10-06 -0.053435 -0.082845
2008-10-07 -0.046371 -0.073905
2008-10-08  0.001057 -0.043350
2008-10-09 -0.109292 -0.056643
2008-10-10  0.016005 -0.007642
2008-10-13  0.142941  0.101210
2008-10-14 -0.017356 -0.000999
2008-10-15 -0.111169 -0.083000
2008-10-16  0.127411  0.106870
2008-10-17 -0.027994 -0.000985
2008-10-20  0.034667  0.097633
2008-10-21 -0.007732 -0.044923
2008-10-22 -0.029091 -0.047978
2008-10-23 -0.052969  0.008893
2008-10-24 -0.087571 -0.015671
2008-10-27 -0.042724 -0.037811
2008-10-28  0.083441  0.062047
2008-10-29  0.040597 -0.009737
```

```
2008-10-30   0.040734   0.036381
2008-10-31   0.063947 -0.027514
```

[ ]:
```
plt.plot(df_nfin_returns.index,df_nfin_returns)
plt.legend(tickers_nfin, loc='upper left')
plt.xlabel("Date")
plt.ylabel("Percent")
plt.title("Daily returns data of Non-Financial Companies")
plt.xticks(rotation=90)
plt.show()
```



3 Step 4: Correlation matrix

We combine the two list of returns:

[ ]:
```
df_returns = pd.DataFrame()
```

[ ]:
```
df_returns = pd.concat([df_nfin_returns, df_fin_returns],axis=1)
```

[ ]:
```
df_returns
```

[ ]:

Date	KR	PFE	XOM	WMT	DAL	CSCO
2008-09-03	-0.000360	0.001565	0.009053	0.002347	-0.006543	-0.018526
2008-09-04	-0.019431	-0.027604	-0.024096	-0.000167	-0.016465	-0.044187
2008-09-05	-0.013945	-0.008570	-0.006830	0.016059	-0.016741	-0.000898
2008-09-08	0.016003	0.034036	0.015208	0.020744	-0.024972	0.049865
2008-09-09	0.023810	-0.047022	-0.045721	-0.014032	-0.041909	-0.017116
2008-09-10	0.001073	-0.002193	0.027163	0.014559	-0.043743	0.000871
2008-09-11	0.007863	0.010440	0.004120	0.018542	0.027954	0.002610
2008-09-12	-0.010993	0.012507	0.025675	-0.012031	0.004944	0.017788
2008-09-15	-0.046611	-0.030612	-0.054839	-0.012498	-0.009840	-0.046036
2008-09-16	0.052651	-0.013296	0.043413	0.008275	0.234782	0.018767
2008-09-17	-0.038942	-0.035935	-0.015047	-0.040231	-0.094567	-0.043860
2008-09-18	-0.002974	0.047758	0.032811	0.030852	0.050000	0.045872
2008-09-19	0.008203	0.031128	0.023923	-0.028953	0.023281	0.065351
2008-09-22	-0.011835	-0.025876	-0.009169	-0.013568	-0.100311	-0.048579
2008-09-23	-0.003742	-0.003321	-0.015087	-0.008321	0.017242	-0.016443
2008-09-24	-0.010143	-0.001110	0.004377	0.008904	-0.092656	0.003080
2008-09-25	0.020114	0.036131	0.033833	0.020367	0.003736	0.029824
2008-09-26	0.036087	0.001073	-0.000248	0.009814	-0.013648	0.014480
2008-09-29	-0.036625	-0.054126	-0.081711	-0.037226	-0.077988	-0.085222
2008-09-30	0.024227	0.044759	0.048609	0.024637	0.016371	0.035337
2008-10-01	0.002911	0.027115	0.011847	-0.003841	0.147651	-0.027039
2008-10-02	0.007983	-0.007920	-0.013744	-0.013577	-0.074854	-0.034168
2008-10-03	-0.020879	0.011176	0.005677	0.014953	-0.006321	0.002359
2008-10-06	-0.025000	-0.003158	-0.007954	-0.030638	-0.078880	-0.037177
2008-10-07	-0.028657	-0.068110	-0.016167	-0.052850	-0.211326	-0.079179
2008-10-08	-0.034161	-0.029462	0.012226	-0.005288	-0.012259	-0.026539
2008-10-09	-0.032155	-0.085231	-0.116884	-0.057928	0.003547	-0.062705
2008-10-10	-0.012458	-0.033822	-0.082941	-0.008562	0.065371	0.002327
2008-10-13	0.066863	0.101717	0.171905	0.069676	0.127695	0.118398
2008-10-14	0.011825	0.026979	-0.008484	-0.001101	0.080882	-0.037882
2008-10-15	-0.043241	-0.050204	-0.139526	-0.080639	0.012245	-0.106257
2008-10-16	0.070440	0.043024	0.113874	0.091309	0.188172	0.071213
2008-10-17	-0.040319	-0.003536	-0.020302	-0.015562	0.039593	0.009014
2008-10-20	0.037257	0.025429	0.102145	0.012274	0.023939	0.058626
2008-10-21	0.002293	0.000000	-0.046540	-0.013962	0.037194	-0.058017
2008-10-22	0.022874	-0.034602	-0.096923	-0.026085	0.017418	-0.025196
2008-10-23	0.004846	0.021505	0.090135	0.009374	-0.113797	-0.009764
2008-10-24	-0.021143	-0.030994	-0.019179	-0.025777	-0.055682	-0.053944
2008-10-27	-0.034104	-0.010863	-0.042729	-0.033657	-0.078219	-0.013489
2008-10-28	0.049039	0.087248	0.132698	0.110731	0.065274	0.137974
2008-10-29	-0.008975	-0.035353	-0.002805	-0.002719	-0.020833	-0.024030

```
2008-10-30  0.036604  0.038976  0.005358 -0.004907  0.195244 -0.004477
2008-10-31 -0.000365 -0.008398 -0.012392  0.019361  0.149739 -0.001124

                EQIX       DUK      NFLX        GE  …         C        GS  \
Date                                                …
2008-09-03 -0.009883 -0.012687  0.002270  0.001402  …  0.026165  0.013852
2008-09-04 -0.033063  0.009930 -0.033646 -0.030451  … -0.066803 -0.040033
2008-09-05 -0.000129  0.001157 -0.007030  0.006498  …  0.042077  0.014542
2008-09-08  0.000258  0.032351  0.016521  0.043401  …  0.065548  0.039758
2008-09-09 -0.044253  0.012870 -0.046766 -0.033345  … -0.070866 -0.047488
2008-09-10 -0.019034 -0.012707  0.000348 -0.001067  … -0.010593 -0.025236
2008-09-11  0.047062  0.019586  0.002782  0.002492  … -0.003748 -0.003554
2008-09-12  0.011040  0.004391  0.021506 -0.050071  … -0.034927 -0.017959
2008-09-15 -0.060835 -0.024044 -0.045501 -0.080374  … -0.151448 -0.121328
2008-09-16  0.036263  0.010078  0.012451  0.018700  …  0.033465 -0.018377
2008-09-17 -0.029651 -0.036586  0.014055 -0.066641  … -0.109207 -0.139162
2008-09-18  0.055058  0.069621  0.075537  0.074090  …  0.186743 -0.056769
2008-09-19  0.103849 -0.033351  0.028995  0.073820  …  0.240240  0.201853
2008-09-22 -0.055194  0.000000 -0.007201 -0.017656  … -0.030993 -0.069492
2008-09-23 -0.043533 -0.016694 -0.005676 -0.045889  … -0.000999  0.035353
2008-09-24 -0.025634  0.010187  0.007612 -0.014429  … -0.051526  0.063574
2008-09-25  0.025101  0.006723  0.037772  0.044327  …  0.023734  0.018798
2008-09-26 -0.037842  0.003339 -0.003033 -0.016745  …  0.038125  0.018376
2008-09-29 -0.105743 -0.023295 -0.038637 -0.085148  … -0.119106 -0.125299
2008-09-30  0.057069 -0.010221 -0.022785  0.103897  …  0.155493  0.060481
2008-10-01 -0.026922  0.021801 -0.027202 -0.039216  …  0.121404  0.050781
2008-10-02 -0.034029 -0.015160 -0.024634 -0.095919  … -0.021739 -0.022007
2008-10-03 -0.050697 -0.010262 -0.011263 -0.026185  … -0.184445 -0.026912
2008-10-06 -0.072443 -0.014401 -0.085606 -0.008808  … -0.051226 -0.031250
2008-10-07 -0.056706 -0.008183 -0.157795 -0.050515  … -0.129811 -0.072580
2008-10-08  0.049972 -0.002357  0.019274  0.017242  … -0.049505 -0.017392
2008-10-09 -0.034071 -0.105139  0.008795 -0.079419  … -0.102084 -0.103097
2008-10-10 -0.007636 -0.047525  0.058849  0.130984  …  0.091261 -0.123828
2008-10-13  0.130634  0.133056  0.045698 -0.023256  …  0.116230  0.250000
2008-10-14 -0.048452 -0.038532 -0.053543 -0.007143  …  0.182222  0.107207
2008-10-15 -0.085831 -0.057888 -0.063644 -0.076739  … -0.128357 -0.079333
2008-10-16  0.005216  0.048616  0.041759  0.033247  … -0.020333 -0.006628
2008-10-17  0.050963 -0.024469 -0.006397 -0.013072  … -0.064151  0.016904
2008-10-20  0.083583  0.100990  0.021459  0.025981  …  0.014113  0.062992
2008-10-21 -0.043938 -0.058753 -0.123950  0.010427  … -0.060304 -0.001235
2008-10-22 -0.083745 -0.015924 -0.007194 -0.068305  … -0.060649 -0.051833
2008-10-23 -0.015419  0.061489  0.004348 -0.008439  … -0.015765 -0.053439
2008-10-24  0.000000 -0.046952 -0.088504 -0.051596  … -0.073989 -0.075336
2008-10-27 -0.025094  0.005759 -0.053298 -0.005608  … -0.033773 -0.074901
2008-10-28  0.101413  0.061068  0.146600  0.099267  …  0.143222  0.007429
2008-10-29  0.047443 -0.026379  0.057851 -0.014880  … -0.037286  0.043711
2008-10-30  0.050999  0.036330  0.046875  0.007813  …  0.028235 -0.067070
```

```
2008-10-31 -0.003671 -0.026737  0.086918  0.008269  …  0.041190  0.015257

                USB        MS       KEY       PNC       COF       AXP  \
Date
2008-09-03  0.017918  0.021065  0.008731  0.009259  0.016474  0.006892
2008-09-04 -0.039453 -0.043396 -0.062156 -0.018484 -0.051030 -0.052799
2008-09-05  0.034440  0.025285  0.088087  0.021169  0.031849  0.016774
2008-09-08  0.036652  0.046180  0.059368  0.033517  0.089913  0.028426
2008-09-09 -0.055392 -0.066327 -0.042940 -0.051185 -0.066284 -0.056269
2008-09-10 -0.006551 -0.036634 -0.044867 -0.027316  0.001758 -0.002092
2008-09-11  0.056201 -0.005396  0.019108  0.030482  0.016674  0.015723
2008-09-12  0.005648 -0.038233  0.044531 -0.000684 -0.008632  0.004903
2008-09-15 -0.023943 -0.135375 -0.073298 -0.034946 -0.026992 -0.089089
2008-09-16  0.068141 -0.108419  0.019370  0.090315  0.097092  0.016629
2008-09-17 -0.054721 -0.242160 -0.048298 -0.062777 -0.055057 -0.084003
2008-09-18  0.102880  0.036781  0.097338  0.097832  0.165300  0.141647
2008-09-19  0.033179  0.206652  0.128128  0.027974  0.037963  0.071050
2008-09-22 -0.078705 -0.004410 -0.092742 -0.069080 -0.028546 -0.076980
2008-09-23 -0.028286  0.033592 -0.028148 -0.051852 -0.013407  0.026549
2008-09-24 -0.004705 -0.114643  0.009909  0.015764 -0.022896 -0.020376
2008-09-25  0.043132  0.093183  0.056603 -0.007004 -0.051248  0.008800
2008-09-26  0.037696 -0.086716  0.050000  0.044260  0.095381  0.044145
2008-09-29 -0.095304 -0.151919 -0.333334 -0.099338 -0.193400 -0.175949
2008-09-30  0.099847  0.095760  0.218368  0.098529  0.159090  0.088479
2008-10-01  0.018323  0.061739  0.149916  0.036145  0.008235  0.002553
2008-10-02 -0.010088 -0.049550 -0.037873  0.003617 -0.090237 -0.090549
2008-10-03 -0.034426  0.030591 -0.049962 -0.045443 -0.040188 -0.039515
2008-10-06 -0.007416 -0.017559 -0.060558  0.016858 -0.071492 -0.025915
2008-10-07 -0.086207 -0.248936 -0.100085 -0.080902 -0.099065 -0.060525
2008-10-08 -0.030503 -0.048159 -0.151744 -0.012966 -0.047124 -0.040000
2008-10-09 -0.061628 -0.258929 -0.286667 -0.116605 -0.066499 -0.115044
2008-10-10  0.049429 -0.222490  0.057633  0.136174  0.063753 -0.035417
2008-10-13  0.029315  0.869835  0.142857  0.005147  0.118176  0.179265
2008-10-14  0.006720  0.212155  0.542526 -0.077688  0.119024  0.032601
2008-10-15 -0.042276 -0.163360 -0.130326 -0.024429 -0.150889 -0.134090
2008-10-16  0.024561  0.030888  0.013449 -0.001626  0.024895 -0.031544
2008-10-17 -0.010366  0.029427 -0.044550 -0.056351  0.031524 -0.013114
2008-10-20  0.018658  0.027547 -0.033730  0.029686  0.009018  0.043721
2008-10-21 -0.029563  0.021750  0.124230 -0.011230 -0.034757  0.083778
2008-10-22 -0.069205 -0.043069 -0.018265 -0.057128 -0.063786 -0.051913
2008-10-23  0.020278 -0.064666 -0.063256  0.022653 -0.018956 -0.019585
2008-10-24  0.026848 -0.086283  0.004965  0.035162 -0.011481 -0.019568
2008-10-27 -0.021392 -0.168886 -0.019763 -0.004246 -0.025496 -0.040333
2008-10-28  0.069396  0.107065  0.195564  0.116152  0.159883  0.103553
2008-10-29 -0.058079 -0.028947  0.024452 -0.040037 -0.051378 -0.010208
2008-10-30 -0.007923  0.090109  0.012346  0.026265  0.007133  0.033717
2008-10-31  0.035070  0.085768  0.008943  0.034124  0.026233  0.055258
```

Date	PRU	SCHW
2008-09-03	0.026601	0.001241
2008-09-04	-0.031949	-0.026848
2008-09-05	0.023129	0.022496
2008-09-08	0.078486	0.046907
2008-09-09	-0.060292	-0.068596
2008-09-10	-0.000251	0.012772
2008-09-11	0.031086	0.029844
2008-09-12	-0.020301	-0.021633
2008-09-15	-0.099889	-0.052983
2008-09-16	0.091536	0.122467
2008-09-17	-0.065926	-0.105966
2008-09-18	0.135614	0.049166
2008-09-19	0.026908	0.056486
2008-09-22	-0.098435	-0.120396
2008-09-23	-0.039095	0.017560
2008-09-24	-0.017933	-0.006637
2008-09-25	0.009403	0.049888
2008-09-26	-0.007425	0.031820
2008-09-29	-0.119287	-0.106497
2008-09-30	0.111969	0.196502
2008-10-01	-0.100000	-0.062692
2008-10-02	-0.110339	-0.065654
2008-10-03	-0.030876	-0.012297
2008-10-06	-0.065689	-0.070698
2008-10-07	-0.109003	-0.068900
2008-10-08	-0.069232	-0.014388
2008-10-09	-0.231462	-0.048488
2008-10-10	0.085963	0.106849
2008-10-13	0.382507	0.154951
2008-10-14	0.083083	-0.101157
2008-10-15	-0.149168	-0.049118
2008-10-16	-0.103845	0.046640
2008-10-17	0.012848	0.006229
2008-10-20	-0.015079	-0.020000
2008-10-21	-0.077521	-0.036928
2008-10-22	-0.110906	-0.092331
2008-10-23	-0.038518	-0.041134
2008-10-24	0.061941	-0.055652
2008-10-27	-0.064132	-0.046655
2008-10-28	0.131783	0.139730
2008-10-29	-0.034246	-0.014124
2008-10-30	-0.180993	0.062464
2008-10-31	0.039141	0.031283

```
[43 rows x 27 columns]
```

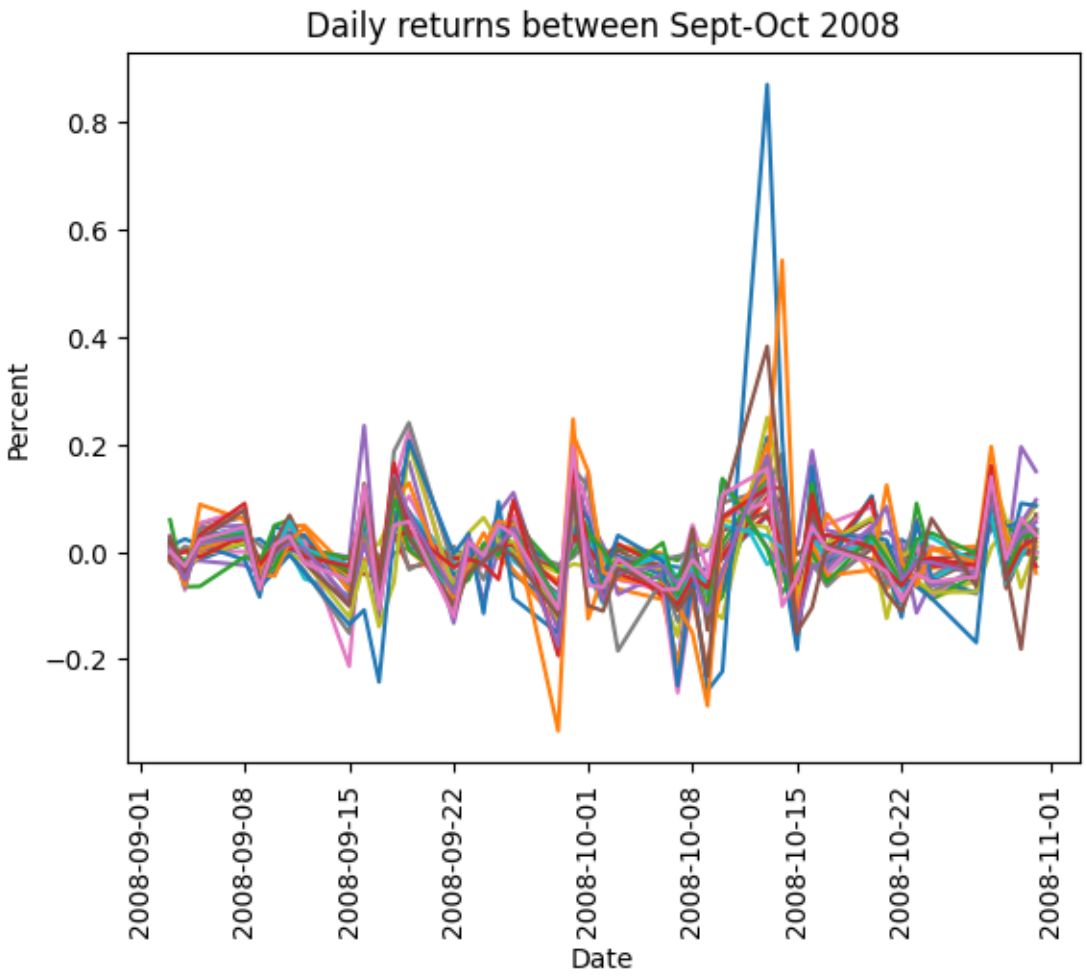
[ ]:
```python
zero_values = (df_returns == 0).sum().sum()

if zero_values > 0:
    print(f"There are {zero_values} zero values in the dataset.")
else:
    print("There are no zero values in the dataset.")
```

```
There are 3 zero values in the dataset.
```

[ ]:
```python
df_returns = df_returns.replace(0, 0.01)

zero_values = (df_returns == 0).sum().sum()

if zero_values > 0:
    print(f"There are {zero_values} zero values in the dataset.")
else:
    print("There are no zero values in the dataset.")
```

```
There are no zero values in the dataset.
```

[ ]:
```python
plt.plot(df_returns.index,df_returns)
plt.xlabel("Date")
plt.ylabel("Percent")
plt.title("Daily returns between Sept-Oct 2008")
plt.xticks(rotation=90)
plt.show()
```



Compute the correlation matrix

[ ]:
```
correlation_matrix = df_returns.corr()
```

[ ]:
```
correlation_matrix
```

[ ]:

	KR	PFE	XOM	WMT	DAL	CSCO	EQIX
KR	1.000000	0.678727	0.682164	0.719101	0.582101	0.718785	0.498876
PFE	0.678727	1.000000	0.814427	0.784545	0.470394	0.827935	0.651960
XOM	0.682164	0.814427	1.000000	0.813531	0.286487	0.810033	0.707328
WMT	0.719101	0.784545	0.813531	1.000000	0.459232	0.833895	0.572994
DAL	0.582101	0.470394	0.286487	0.459232	1.000000	0.445254	0.431524
CSCO	0.718785	0.827935	0.810033	0.833895	0.445254	1.000000	0.786882
EQIX	0.498876	0.651960	0.707328	0.572994	0.431524	0.786882	1.000000
DUK	0.620161	0.697747	0.807550	0.679876	0.224743	0.688168	0.530777
NFLX	0.430027	0.476899	0.461526	0.626155	0.481221	0.699343	0.610168
GE	0.423593	0.556435	0.451754	0.558394	0.346717	0.657158	0.605572

APA	0.650520	0.772125	0.909246	0.804323	0.453345	0.802754	0.788227
F	0.480844	0.670726	0.526334	0.542605	0.374166	0.635382	0.624723
REGN	0.690884	0.672397	0.677183	0.761742	0.611956	0.725780	0.595803
CMS	0.722943	0.796494	0.799214	0.777455	0.445378	0.840849	0.712909
JPM	0.573978	0.601153	0.445903	0.525894	0.546640	0.673530	0.622760
WFC	0.546407	0.671584	0.516368	0.574850	0.483720	0.643414	0.556777
BAC	0.623475	0.736074	0.528891	0.547682	0.568516	0.708609	0.621178
C	0.548134	0.682374	0.484337	0.484066	0.461114	0.635401	0.602125
GS	0.526926	0.654293	0.582451	0.440333	0.322715	0.599888	0.582868
USB	0.513936	0.692298	0.554304	0.618618	0.486203	0.688679	0.627353
MS	0.566829	0.767202	0.634495	0.562038	0.439652	0.617077	0.573751
KEY	0.477899	0.651672	0.408785	0.459337	0.397063	0.459416	0.372756
PNC	0.442177	0.541611	0.433090	0.511406	0.403470	0.582907	0.536895
COF	0.568240	0.707037	0.586769	0.675682	0.505415	0.742749	0.636392
AXP	0.610402	0.811972	0.672514	0.641287	0.462684	0.744538	0.719869
PRU	0.439126	0.635997	0.601032	0.599701	0.280200	0.679455	0.623644
SCHW	0.578730	0.623299	0.546693	0.647081	0.542431	0.757432	0.723779

	DUK	NFLX	GE	…	C	GS	USB	\
KR	0.620161	0.430027	0.423593	…	0.548134	0.526926	0.513936	
PFE	0.697747	0.476899	0.556435	…	0.682374	0.654293	0.692298	
XOM	0.807550	0.461526	0.451754	…	0.484337	0.582451	0.554304	
WMT	0.679876	0.626155	0.558394	…	0.484066	0.440333	0.618618	
DAL	0.224743	0.481221	0.346717	…	0.461114	0.322715	0.486203	
CSCO	0.688168	0.699343	0.657158	…	0.635401	0.599888	0.688679	
EQIX	0.530777	0.610168	0.605572	…	0.602125	0.582868	0.627353	
DUK	1.000000	0.398252	0.297068	…	0.354352	0.392924	0.395745	
NFLX	0.398252	1.000000	0.498654	…	0.476185	0.231463	0.470364	
GE	0.297068	0.498654	1.000000	…	0.696722	0.311549	0.719808	
APA	0.696941	0.576916	0.528884	…	0.518175	0.598577	0.557362	
F	0.435539	0.388444	0.458467	…	0.526942	0.496853	0.636061	
REGN	0.607500	0.601975	0.500273	…	0.522259	0.454663	0.554617	
CMS	0.723251	0.573619	0.514222	…	0.582580	0.588003	0.561152	
JPM	0.308987	0.550412	0.776209	…	0.798621	0.458783	0.874924	
WFC	0.360507	0.400030	0.687628	…	0.737301	0.508426	0.845237	
BAC	0.347014	0.542831	0.704379	…	0.904392	0.662752	0.797876	
C	0.354352	0.476185	0.696722	…	1.000000	0.624715	0.737274	
GS	0.392924	0.231463	0.311549	…	0.624715	1.000000	0.398360	
USB	0.395745	0.470364	0.719808	…	0.737274	0.398360	1.000000	
MS	0.559593	0.306551	0.249676	…	0.552784	0.815326	0.365876	
KEY	0.212803	0.161626	0.494459	…	0.740573	0.594278	0.585584	
PNC	0.382474	0.444965	0.739120	…	0.618893	0.197672	0.865550	
COF	0.425307	0.495256	0.700754	…	0.791585	0.488959	0.821383	
AXP	0.528129	0.404196	0.626061	…	0.749052	0.695363	0.721559	
PRU	0.482296	0.375403	0.526077	…	0.618039	0.597330	0.672177	
SCHW	0.383075	0.563079	0.728629	…	0.586964	0.449161	0.761063	

	MS	KEY	PNC	COF	AXP	PRU	SCHW
KR	0.566829	0.477899	0.442177	0.568240	0.610402	0.439126	0.578730
PFE	0.767202	0.651672	0.541611	0.707037	0.811972	0.635997	0.623299
XOM	0.634495	0.408785	0.433090	0.586769	0.672514	0.601032	0.546693
WMT	0.562038	0.459337	0.511406	0.675682	0.641287	0.599701	0.647081
DAL	0.439652	0.397063	0.403470	0.505415	0.462684	0.280200	0.542431
CSCO	0.617077	0.459416	0.582907	0.742749	0.744538	0.679455	0.757432
EQIX	0.573751	0.372756	0.536895	0.636392	0.719869	0.623644	0.723779
DUK	0.559593	0.212803	0.382474	0.425307	0.528129	0.482296	0.383075
NFLX	0.306551	0.161626	0.444965	0.495256	0.404196	0.375403	0.563079
GE	0.249676	0.494459	0.739120	0.700754	0.626061	0.526077	0.728629
APA	0.669160	0.405639	0.417116	0.609583	0.727876	0.607886	0.636193
F	0.583979	0.563755	0.503500	0.691583	0.656880	0.725533	0.686454
REGN	0.538296	0.399615	0.478011	0.609008	0.672142	0.544712	0.613827
CMS	0.627008	0.454761	0.424733	0.636121	0.647476	0.518944	0.578905
JPM	0.314179	0.507706	0.831551	0.716379	0.683871	0.484241	0.783256
WFC	0.435234	0.688072	0.747619	0.803338	0.752643	0.705200	0.709024
BAC	0.542733	0.749551	0.656818	0.799234	0.766054	0.621735	0.642980
C	0.552784	0.740573	0.618893	0.791585	0.749052	0.618039	0.586964
GS	0.815326	0.594278	0.197672	0.488959	0.695363	0.597330	0.449161
USB	0.365876	0.585584	0.865550	0.821383	0.721559	0.672177	0.761063
MS	1.000000	0.530135	0.174717	0.494553	0.713739	0.681716	0.465019
KEY	0.530135	1.000000	0.422968	0.729291	0.695863	0.583882	0.375968
PNC	0.174717	0.422968	1.000000	0.670653	0.590836	0.540139	0.718197
COF	0.494553	0.729291	0.670653	1.000000	0.820632	0.747273	0.722565
AXP	0.713739	0.695863	0.590836	0.820632	1.000000	0.750030	0.689332
PRU	0.681716	0.583882	0.540139	0.747273	0.750030	1.000000	0.658963
SCHW	0.465019	0.375968	0.718197	0.722565	0.689332	0.658963	1.000000

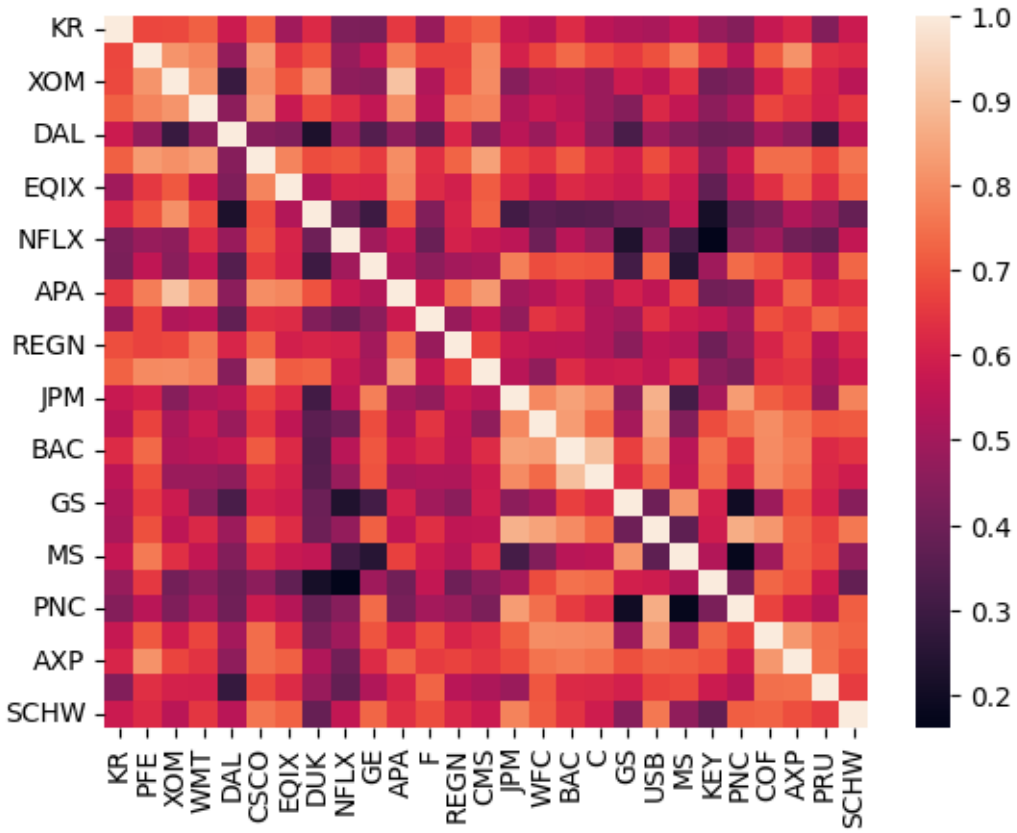
[27 rows x 27 columns]

Heatmap of the correlation matrix

1)

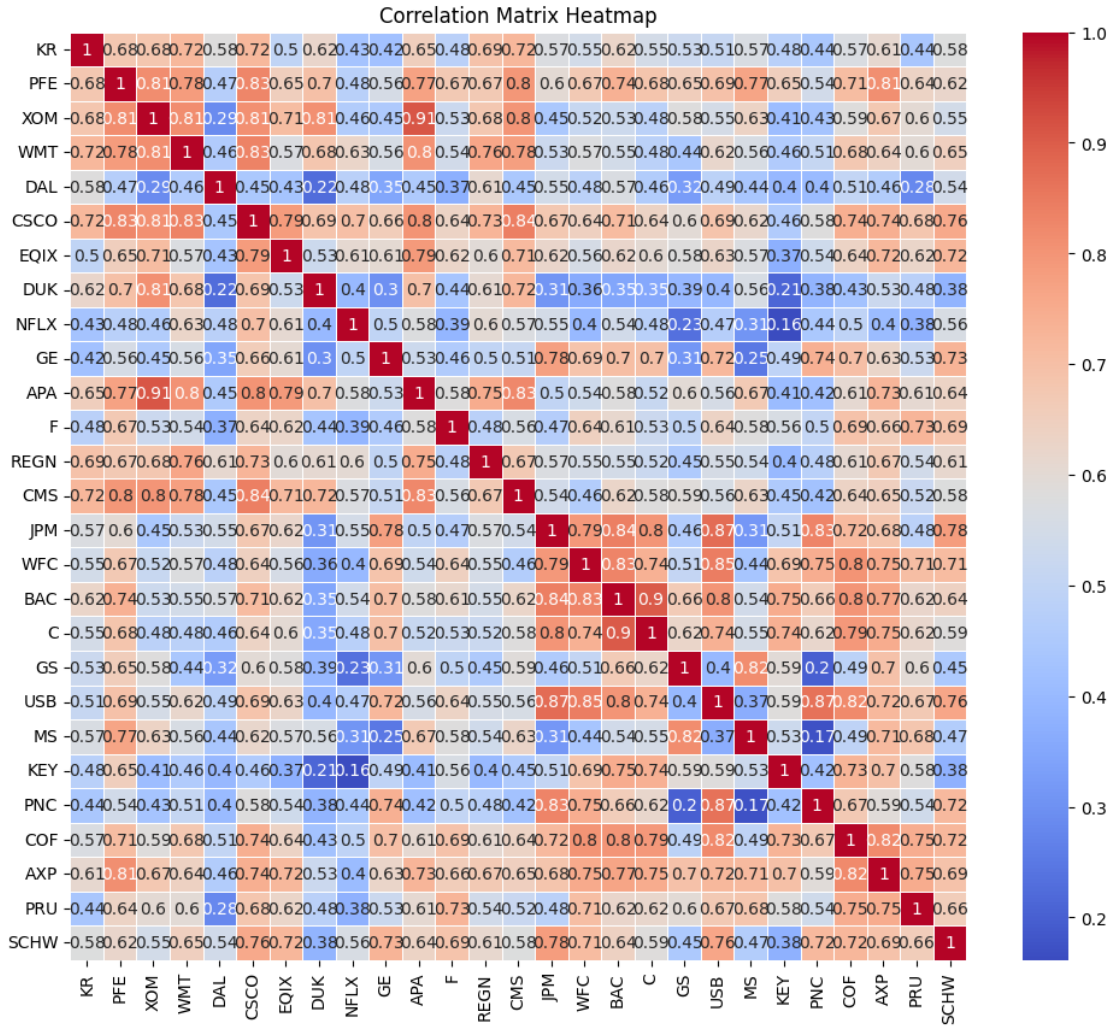
[ ]:
```
sns.heatmap(correlation_matrix)
```

[ ]: <Axes: >



2)

[ ]:
```
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Sort the securities:

We first calculate the average correlation for each security

[ ]:
```
maximum_corr = []
```

[ ]:
```
for i in correlation_matrix.columns:
  max_corr = correlation_matrix[i].drop(i).max()
  maximum_corr.append([i,f'{max_corr*100:.2f}%'])
```

[ ]:
```
maximum_corr = pd.DataFrame(maximum_corr,columns=['Tickers', 'Max Correlation'])
```

[ ]:
```
maximum_corr
```

[ ]:
```
    Tickers Max Correlation
0        KR          72.29%
```

1	PFE	82.79%
2	XOM	90.92%
3	WMT	83.39%
4	DAL	61.20%
5	CSCO	84.08%
6	EQIX	78.82%
7	DUK	80.75%
8	NFLX	69.93%
9	GE	77.62%
10	APA	90.92%
11	F	72.55%
12	REGN	76.17%
13	CMS	84.08%
14	JPM	87.49%
15	WFC	84.52%
16	BAC	90.44%
17	C	90.44%
18	GS	81.53%
19	USB	87.49%
20	MS	81.53%
21	KEY	74.96%
22	PNC	86.56%
23	COF	82.14%
24	AXP	82.06%
25	PRU	75.00%
26	SCHW	78.33%

Then we sort the securities based on average correlation

[ ]:
```
sort = maximum_corr.sort_values(by='Max Correlation', ascending=False)
```

[ ]:
```
sort
```

[ ]:

	Tickers	Max Correlation
10	APA	90.92%
2	XOM	90.92%
17	C	90.44%
16	BAC	90.44%
14	JPM	87.49%
19	USB	87.49%
22	PNC	86.56%
15	WFC	84.52%
5	CSCO	84.08%
13	CMS	84.08%
3	WMT	83.39%
1	PFE	82.79%
23	COF	82.14%

```
24     AXP          82.06%
18      GS          81.53%
20      MS          81.53%
7      DUK          80.75%
6     EQIX          78.82%
26    SCHW          78.33%
9       GE          77.62%
12    REGN          76.17%
25     PRU          75.00%
21     KEY          74.96%
11       F          72.55%
0       KR          72.29%
8     NFLX          69.93%
4      DAL          61.20%
```

Create two list for the high-correlated, and not correlated stocks.

First, we would set a threshold over the mean:

[ ]:
```
maximum_corr['Max Correlation'].str.rstrip('%').astype(float)
```

[ ]:
```
0     72.29
1     82.79
2     90.92
3     83.39
4     61.20
5     84.08
6     78.82
7     80.75
8     69.93
9     77.62
10    90.92
11    72.55
12    76.17
13    84.08
14    87.49
15    84.52
16    90.44
17    90.44
18    81.53
19    87.49
20    81.53
21    74.96
22    86.56
23    82.14
24    82.06
25    75.00
26    78.33
```

```
Name: Max Correlation, dtype: float64
```

[ ]:
```
maximum_corr['Numerical correlation'] = maximum_corr['Max Correlation'].str.
  ↪rstrip('%').astype(float)
maximum_corr
```

[ ]:

	Tickers	Max Correlation	Numerical correlation
0	KR	72.29%	72.29
1	PFE	82.79%	82.79
2	XOM	90.92%	90.92
3	WMT	83.39%	83.39
4	DAL	61.20%	61.20
5	CSCO	84.08%	84.08
6	EQIX	78.82%	78.82
7	DUK	80.75%	80.75
8	NFLX	69.93%	69.93
9	GE	77.62%	77.62
10	APA	90.92%	90.92
11	F	72.55%	72.55
12	REGN	76.17%	76.17
13	CMS	84.08%	84.08
14	JPM	87.49%	87.49
15	WFC	84.52%	84.52
16	BAC	90.44%	90.44
17	C	90.44%	90.44
18	GS	81.53%	81.53
19	USB	87.49%	87.49
20	MS	81.53%	81.53
21	KEY	74.96%	74.96
22	PNC	86.56%	86.56
23	COF	82.14%	82.14
24	AXP	82.06%	82.06
25	PRU	75.00%	75.00
26	SCHW	78.33%	78.33

[ ]:
```
threshold = maximum_corr['Numerical correlation'].mean()
threshold
```

[ ]:
```
81.03703703703704
```

Then, we create the two lists, for high correlated, and low correlated:

[ ]:
```
high_corr_assets = maximum_corr[maximum_corr['Numerical correlation'] >␣
  ↪threshold]['Tickers'].tolist()
low_corr_assets = maximum_corr[maximum_corr['Numerical correlation'] <=␣
  ↪threshold]['Tickers'].tolist()
```

[ ]:
```
print("high_corr_assets",high_corr_assets)
print("low_corr_assets",low_corr_assets)
```

```
high_corr_assets ['PFE', 'XOM', 'WMT', 'CSCO', 'APA', 'CMS', 'JPM', 'WFC',
'BAC', 'C', 'GS', 'USB', 'MS', 'PNC', 'COF', 'AXP']
low_corr_assets ['KR', 'DAL', 'EQIX', 'DUK', 'NFLX', 'GE', 'F', 'REGN', 'KEY',
'PRU', 'SCHW']
```

4 Step 6: UBC algorithm

4.0.1 a) pseudocode

Initialize:

N(a) = 0 for all a

Q(a) = 0 for all a

For each round:

For each arm a:

If N(a) > 0:

UCB(a) = Q(a) + sqrt((2 * log(total count of rounds)) / N(a))

Else:

UCB(a) = Infinity

a_max = argmax_a UCB(a) # Choose the arm which has maximum UCB

Reward = pullBandit(a_max) # Pull the chosen arm and get the reward

N(a_max) = N(a_max) + 1 # Increment the count of chosen arm

Q(a_max) = Q(a_max) + (Reward - Q(a_max)) / N(a_max) # Update the estimated value of chosen arm

In this pseudocode:

- N(a) is the number of times action a has been selected.
- Q(a) is the estimated value of action a.
- UCB(a) is the upper confidence bound of action a.
- pullBandit(a) is a function to pull arm a of the bandit and it returns a reward.

4.1 b) Python implementation

[ ]:
```
# Read stock price information

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

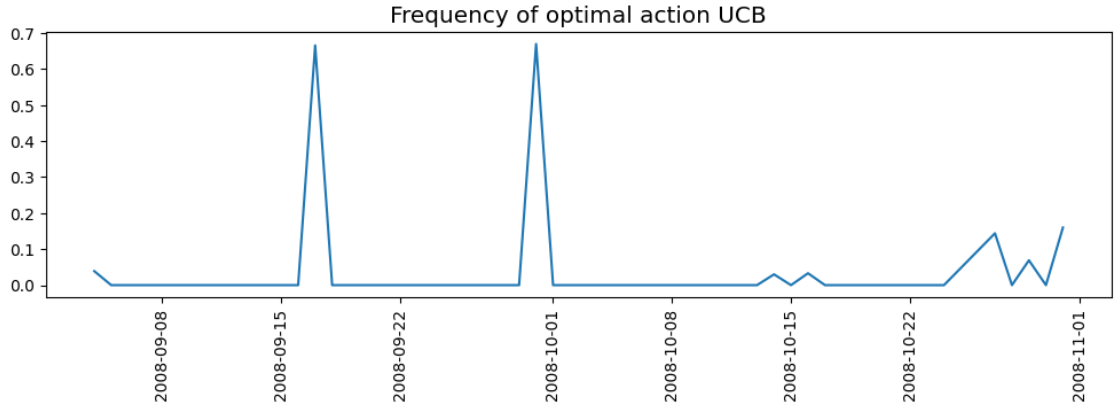
```
from numpy.random import rand, seed


def optimal_action(qvalue, eps):  # noQA E203
    """
    Determines what is the action to take given a measure of past
    expected rewards across actions. With probability eps the action
    is not the greedy one
    """
    nactions = qvalue.shape[0]
    action_hat = np.where(qvalue == np.max(qvalue))

    if rand() <= eps:
        randnum = rand()
        for aa in range(nactions):
            if randnum < (aa + 1) / nactions:  # noQA E203
                break
    elif action_hat[0].shape[0] > 1:  # noQA E203
        # Randomize action when ties
        randnum = rand()
        for aa in range(action_hat[0].shape[0]):  # noQA E203
            if randnum < (aa + 1) / action_hat[0].shape[0]:  # noQA E203
                break
        aa = action_hat[0][aa]
    else:
        aa = np.argmax(qvalue)

    return aa


def reward_update(action, reward, qvalue_old, alpha):  # noQA E203
    qvalue_new = qvalue_old.copy()

    qvalue_new[action] = qvalue_old[action] + alpha * (reward -␣
 ↪qvalue_old[action])

    return qvalue_new
```

[ ]:
```
pdata = df_returns.to_numpy()
pdata_dates = pd.to_datetime(df_returns.index, format='%Y-%m-%d')


#        Bandit problem for stock selection
NK = pdata.shape[1]

EPSILON = 0.0    # e-greedy setting TURNED OFF
ALPHA =0.9
```

```
NEPISODES = 1000
HOLD = 1
TMAX = pdata.shape[0] - HOLD

#        NEW PARAMETER
UCB_WEIGHT =1.0    # UCB setting TURNED ON
seed(1234)
reward_avg = np.zeros((TMAX))
optimal_avg = np.zeros((TMAX))
reward_queue = np.zeros((HOLD,2))

for run in range(NEPISODES):
# Initialize q function and actions record
  qvalue = np.zeros((NK))
  qvalue_up = np.zeros((NK))
  nactions = np.zeros((NK))

  for tt in range(TMAX):
    aa_opt = optimal_action(qvalue_up,EPSILON)
    nactions[aa_opt] += 1

  #        Compute reward as return over holding period
    reward_queue[HOLD-1,0] = (pdata[tt+HOLD,aa_opt]-pdata[tt,aa_opt])/
 ↪pdata[tt,aa_opt]
    reward_queue[HOLD-1,1] = aa_opt

#        Update Q function using action chosen HOLD days before

    qvalue = reward_update(int(reward_queue[0,1]), reward_queue[0,0], qvalue,␣
 ↪ALPHA)

    #qvalue = reward_update(int(reward_queue[0,1]), reward_queue[0,0], qvalue,␣
 ↪1/nactions[aa_opt])
    #print(qvalue)

   #        Upper-confidence adjustment

    qvalue_up = np.zeros((NK))
    for aa in range(NK):
      # If an action has not been visited simply give it the maximum value␣
 ↪across actions
      if nactions[aa] == 0:
        qvalue_up[aa] = np.max(qvalue_up) +1.0
      else:
        qvalue_up[aa] = qvalue[aa] + UCB_WEIGHT * np.sqrt(np.log(tt+1)/
 ↪nactions[aa])
```

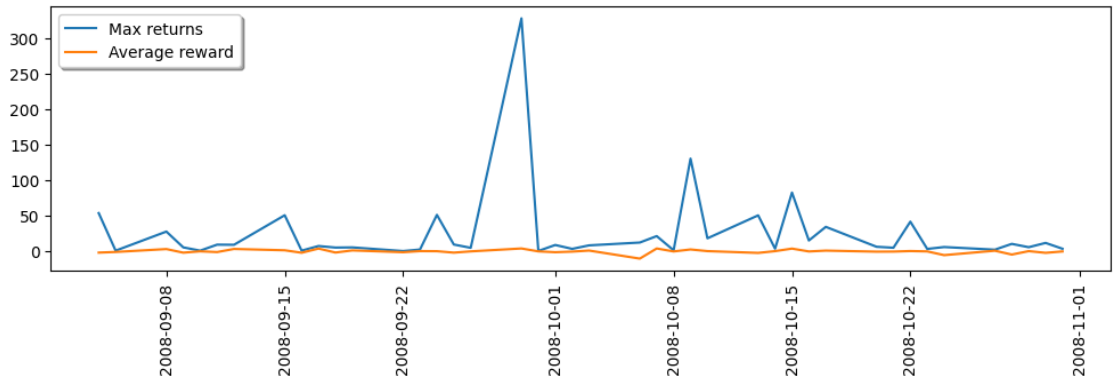
```
    reward_queue[0:HOLD-1,:] = reward_queue[1:HOLD,:]
    reward_avg[tt] += reward_queue[HOLD-1,0]/NEPISODES
    optimal_avg[tt] += (aa_opt==np.argmax((pdata[tt+HOLD,:]-pdata[tt,:])/
 ↪pdata[tt,:]))/NEPISODES
```

[ ]:
```
plt.plot(pdata_dates[HOLD:pdata.shape [0] ],optimal_avg)

plt.title("Frequency of optimal action UCB", fontsize='x-large')
plt.xticks(rotation=90)
fig = plt.gcf()

fig.set_size_inches(12, 3)

plt.show()
```



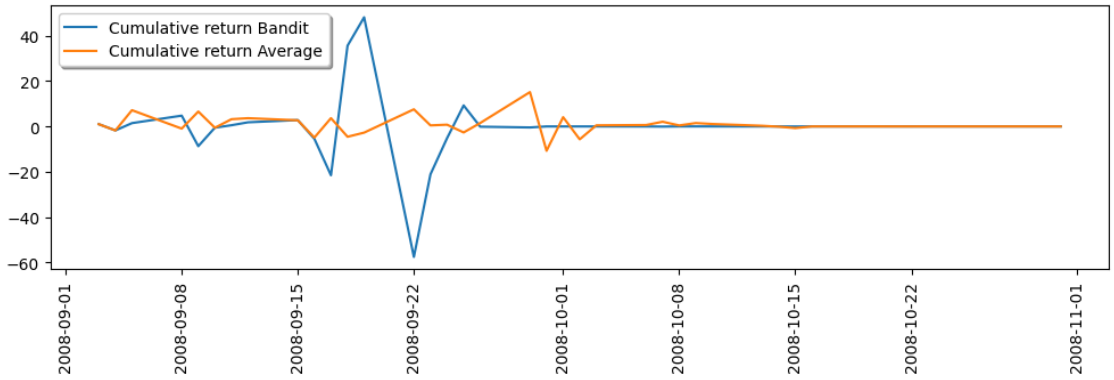
[ ]:
```
plt.plot(pdata_dates[HOLD:pdata.shape [0] ],np.max((pdata[HOLD:pdata.shape[0],:
 ↪]-pdata[0:TMAX,:])/pdata [0:TMAX,:] , axis=1), label='Max returns')

plt.plot(pdata_dates[HOLD:pdata.shape [0] ],reward_avg, label='Average reward')
plt.xticks(rotation=90)
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
fig.set_size_inches(12, 3)
plt.show()

plt.plot(pdata_dates[HOLD:pdata.shape[0] ],np.mean((pdata[HOLD:pdata.shape[0],:
 ↪]-pdata[0:TMAX,:])/pdata[0:TMAX,:], axis=1), label='Average returns')
plt.plot(pdata_dates[HOLD:pdata.shape[0]],reward_avg, label='Average reward')
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
plt.xticks(rotation=90)
fig.set_size_inches(12, 3)
```

```
plt.show()

#        Average frequency of optimal action
print("Optimal Rewards",np.mean(optimal_avg))

#        Average reward across steps
print("Average Rewards",np.mean(reward_avg))

#        Average annualized return from holding the equally-weighted portfolio
print("Average annualized return from holding the equally-weighted␣
 ↪portfolio",(1 + np.mean((pdata[HOLD:pdata.shape[0] ,:]-pdata[0:TMAX,:])/
 ↪pdata[0:TMAX,:])) ** (250 / HOLD) - 1,np.sqrt(250 / HOLD) * np.std(np.
 ↪mean((pdata))))

#        Average annualized return from holding the Bandit portfolio
print("Average annualized return from holding the Bandit portfolio",(1+np.
 ↪mean(reward_avg)) ** (250 / HOLD) - 1 , np.sqrt(250 / HOLD) * np.
 ↪std(reward_avg))
```



```
Optimal Rewards 0.04311904761904765
Average Rewards -1.1251549221198045
Average annualized return from holding the equally-weighted portfolio -1.0 0.0
Average annualized return from holding the Bandit portfolio -1.0
41.16853993417384
```

[ ]:
```
return_cumulative = np.zeros((TMAX+1,2))
return_cumulative[0,0] = 1
return_cumulative[0,1] = 1
for tt in range(1,TMAX+1):
  return_cumulative[tt,0] = return_cumulative[tt-1,0] * ( 1 + reward_avg[tt-1] )
  rmean = np.mean((pdata[tt+HOLD-1,:]-pdata[tt-1,:])/pdata[tt-1,:])
  return_cumulative[tt,1] = return_cumulative[tt-1,1] * ( 1 + rmean )

plt.plot(pdata_dates[HOLD-1:pdata.shape[0]],return_cumulative[:,0],␣
 ↪label='Cumulative return Bandit')
plt.plot(pdata_dates[HOLD-1:pdata.shape[0]],return_cumulative[:,1],␣
 ↪label='Cumulative return Average')
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
fig.set_size_inches(12, 3)
plt.xticks(rotation=90)
plt.show()
```



5 Step 8: Epsilon-greedy algorithm

5.1 a) Pseudocode

First step of algorithm: Create a function to obtain the optimal action choice depending on a measure of past expected rewards.

if rand() <= eps:

```
random action
```

else

`greedy action`

Second step of Algorithm: Create a function to update the expected reward Set the new qvalue equal to the old qvalue + a variance 'alpha'

Third step: Estimate the Algorithm over a number of steps

1) Create a for loop over the number of steps
2) Initialize the parameters
3) Use the optimal action algorithm to determine the type of action (random or greedy).
4) Use the expected reward algorithm based on the choice that comes from the previous function.
5) Calculate the average and optimal rewards across episodes

5.2 b) Python implementation

We are reusing the UCB code but this time we are defining the variable UCB_WEIGHT =0.0 so that we turn off the UCB algorithm and turn it into e-greedy algorithm

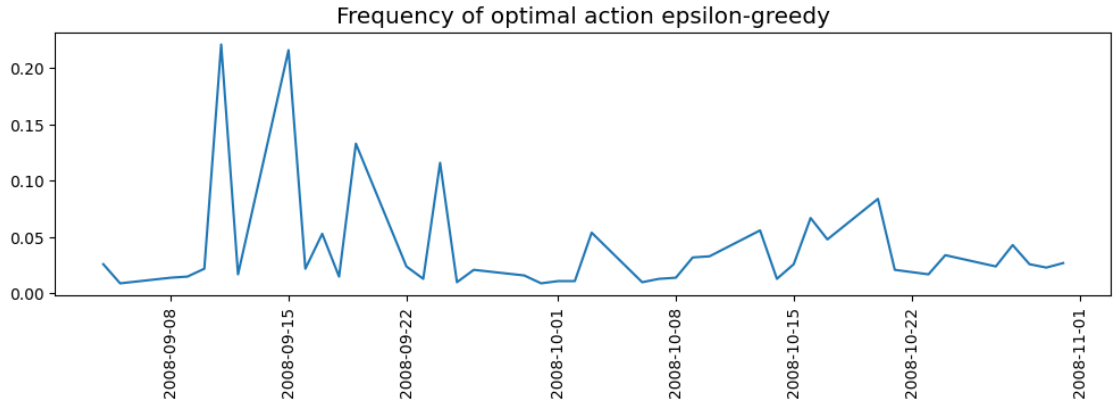
Set the initial parameters

[ ]:
```
pdata = df_returns.to_numpy()
pdata_dates = pd.to_datetime(df_returns.index, format='%Y-%m-%d')


#        Bandit problem for stock selection
NK = pdata.shape[1]

EPSILON = 0.4    # e-greedy setting TURNED ON
ALPHA =0.9
NEPISODES = 1000
HOLD = 1
TMAX = pdata.shape[0] - HOLD

#        NEW PARAMETER
UCB_WEIGHT =0.0   # UCB turned off
seed(1234)
reward_avg = np.zeros((TMAX))
optimal_avg = np.zeros((TMAX))
reward_queue = np.zeros((HOLD,2))

for run in range(NEPISODES):
# Initialize q function and actions record
  qvalue = np.zeros((NK))
  qvalue_up = np.zeros((NK))
  nactions = np.zeros((NK))
```

```
  for tt in range(TMAX):
    aa_opt = optimal_action(qvalue_up,EPSILON)
    nactions[aa_opt] += 1

  #        Compute reward as return over holding period
    reward_queue[HOLD-1,0] = (pdata[tt+HOLD,aa_opt]-pdata[tt,aa_opt])/
 ↪pdata[tt,aa_opt]
    reward_queue[HOLD-1,1] = aa_opt

#         Update Q function using action chosen HOLD days before

    #qvalue = reward_update(int(reward_queue[0,1]), reward_queue[0,0], qvalue,␣
 ↪ALPHA)

    qvalue = reward_update(int(reward_queue[0,1]), reward_queue[0,0], qvalue, 1/
 ↪nactions[aa_opt])
    #print(qvalue)

   #        Upper-confidence adjustment

    qvalue_up = np.zeros((NK))
    for aa in range(NK):
      # If an action has not been visited simply give it the maximum value␣
 ↪across actions
      if nactions[aa] == 0:
        qvalue_up[aa] = np.max(qvalue_up) +1.0
      else:
        qvalue_up[aa] = qvalue[aa] + UCB_WEIGHT * np.sqrt(np.log(tt+1)/
 ↪nactions[aa])

    reward_queue[0:HOLD-1,:] = reward_queue[1:HOLD,:]
    reward_avg[tt] += reward_queue[HOLD-1,0]/NEPISODES
    optimal_avg[tt] += (aa_opt==np.argmax((pdata[tt+HOLD,:]-pdata[tt,:])/
 ↪pdata[tt,:]))/NEPISODES
```

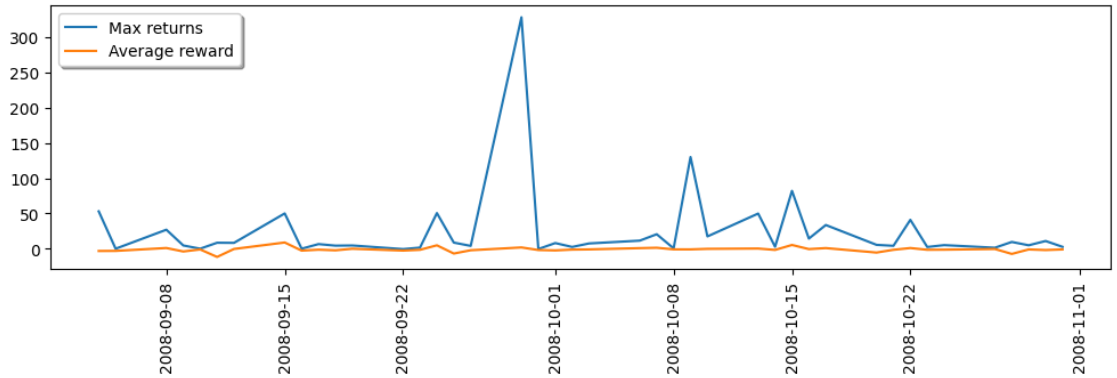
Visualize the results

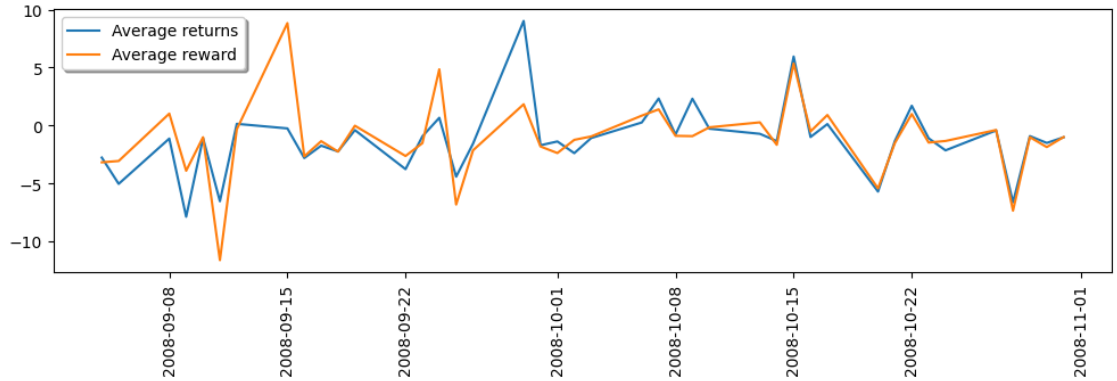
[ ]:
```
plt.plot(pdata_dates[HOLD:pdata.shape [0] ],optimal_avg)

plt.title("Frequency of optimal action epsilon-greedy", fontsize='x-large')
plt.xticks(rotation=90)
fig = plt.gcf()

fig.set_size_inches(12, 3)

plt.show()
```



[ ]:
```
plt.plot(pdata_dates[HOLD:pdata.shape [0] ],np.max((pdata[HOLD:pdata.shape[0],:
 ↪]-pdata[0:TMAX,:])/pdata [0:TMAX,:] , axis=1), label='Max returns')

plt.plot(pdata_dates[HOLD:pdata.shape [0] ],reward_avg, label='Average reward')
plt.xticks(rotation=90)
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
fig.set_size_inches(12, 3)
plt.show()

plt.plot(pdata_dates[HOLD:pdata.shape[0] ],np.mean((pdata[HOLD:pdata.shape[0],:
 ↪]-pdata[0:TMAX,:])/pdata[0:TMAX,:], axis=1), label='Average returns')
plt.plot(pdata_dates[HOLD:pdata.shape[0]],reward_avg, label='Average reward')
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
plt.xticks(rotation=90)
fig.set_size_inches(12, 3)
plt.show()

#        Average frequency of optimal action
print("optimal action",np.mean(optimal_avg))

#        Average reward across steps
print("Average reward ",np.mean(reward_avg))

#        Average annualized return from holding the equally-weighted portfolio
print("Average annualized return from holding the equally-weighted␣
 ↪portfolio",(1 + np.mean((pdata[HOLD:pdata.shape[0] ,:]-pdata[0:TMAX,:])/
 ↪pdata[0:TMAX,:])) ** (250 / HOLD) - 1,np.sqrt(250 / HOLD) * np.std(np.
 ↪mean((pdata))))

#        Average annualized return from holding the Bandit portfolio
```

```
print("Average annualized return from holding the Bandit portfolio",(1+np.
  ↪mean(reward_avg)) ** (250 / HOLD) - 1 , np.sqrt(250 / HOLD) * np.
  ↪std(reward_avg))
```

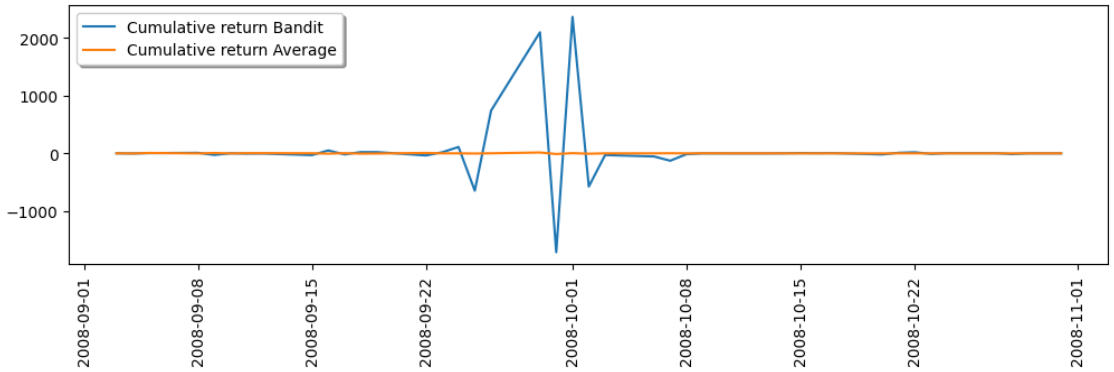





```
optimal action 0.039952380952380975
Average reward  -1.1487193922232553
Average annualized return from holding the equally-weighted portfolio -1.0 0.0
Average annualized return from holding the Bandit portfolio -1.0
51.23520428538145
```

[ ]:
```
return_cumulative = np.zeros((TMAX+1,2))
return_cumulative[0,0] = 1
return_cumulative[0,1] = 1
for tt in range(1,TMAX+1):
  return_cumulative[tt,0] = return_cumulative[tt-1,0] * ( 1 + reward_avg[tt-1] )
  rmean = np.mean((pdata[tt+HOLD-1,:]-pdata[tt-1,:])/pdata[tt-1,:])
  return_cumulative[tt,1] = return_cumulative[tt-1,1] * ( 1 + rmean )
```

```
plt.plot(pdata_dates[HOLD-1:pdata.shape[0]],return_cumulative[:,0],␣
 ↪label='Cumulative return Bandit')
plt.plot(pdata_dates[HOLD-1:pdata.shape[0]],return_cumulative[:,1],␣
 ↪label='Cumulative return Average')
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
fig.set_size_inches(12, 3)
plt.xticks(rotation=90)
plt.show()
```



6 Step 10: Update data

6.1 Import 15 financial companies

Choose tickers

[ ]:
```
tickers_fin_b = ["JPM","WFC","BAC", "C", "GS", "USB", "MS", "KEY", "PNC",␣
 ↪"COF", "AXP",
           "PRU", "SCHW"]
```

Collect data using a data frame for the period between Mar and Apr 2020

[ ]:
```
df_fin_b = pd.DataFrame()
for i in tickers_fin:
  ydata = yf.download(i, start = '2020-03-01', end = '2020-05-01')
  df_fin_b[i] = ydata['Adj Close']
#df_fin.index = pd.to_datetime(ydata.index, format='%Y-%m-%d')
df_fin_b.head()
```

```
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```

```
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```

[ ]:
```
                  JPM        WFC        BAC          C          GS  \
Date
2020-03-02  108.370621  38.377655  26.808290  58.315212  192.560318
2020-03-03  104.304047  36.806580  25.329586  56.123753  187.007889
2020-03-04  106.881332  37.596661  25.913763  58.142658  191.889252
2020-03-05  101.637589  35.326328  24.600166  54.777817  182.742462
2020-03-06   96.384933  33.682610  23.617258  52.871082  177.282013

                 USB         MS        KEY         PNC        COF  \
Date
2020-03-02  41.078987  41.254646  14.371768  115.400002  85.763748
2020-03-03  39.321865  39.408482  13.728129  109.179337  81.028786
2020-03-04  39.799534  40.154053  14.134640  112.581116  83.757240
2020-03-05  37.342979  37.801964  13.533342  105.003220  79.772369
2020-03-06  36.208542  37.136292  12.593292   99.382843  77.156425

                 AXP        PRU       SCHW
Date
2020-03-02  107.949593  66.210289  40.004227
2020-03-03  102.394264  62.465351  36.497421
2020-03-04  109.684441  64.221329  35.411083
2020-03-05  105.171928  60.135197  33.162148
2020-03-06  102.612312  58.096283  32.523682
```

6.2 Import 15 non-financial companies

[ ]:
```
tickers_nfin_b = ["KR", "PFE", "XOM", "WMT", "DAL", "CSCO", "EQIX", "DUK",
                  "NFLX", "GE", "APA", "F", "REGN", "CMS"]
```

[ ]:
```
df_nfin_b = pd.DataFrame()
for i in tickers_nfin_b:
  ydata = yf.download(i, start = '2020-03-01', end = '2020-05-01')
  df_nfin_b[i] = ydata['Adj Close']
#df_nfin.index = pd.to_datetime(ydata.index, format='%Y%m%d')
df_nfin_b.head()
```

```
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```

```
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
```

[ ]:

Date	KR	PFE	XOM	WMT	DAL	CSCO
2020-03-02	27.611349	29.024046	44.751831	109.201492	46.929810	36.643845
2020-03-03	27.295046	28.541418	42.608925	106.402649	45.954594	35.638077
2020-03-04	28.802132	30.288855	43.539169	110.040199	48.263268	36.839649
2020-03-05	31.137192	29.506672	41.620525	109.239204	44.790302	35.219742
2020-03-06	29.806860	29.140537	39.610516	110.473671	45.666008	35.317657

Date	EQIX	DUK	NFLX	GE	APA	F
2020-03-02	583.485107	83.872688	381.049988	68.972809	24.162060	6.278498
2020-03-03	579.977844	82.953018	368.769989	66.942398	23.660265	6.077935
2020-03-04	608.573181	88.193405	383.790009	67.373077	23.792810	6.173856
2020-03-05	581.401428	86.874649	372.779999	62.020157	23.044849	5.877372
2020-03-06	573.859009	85.937622	368.970001	57.893692	19.598537	5.659368

Date	REGN	CMS
2020-03-02	464.750000	57.414967
2020-03-03	461.549988	57.576096
2020-03-04	493.480011	61.183552
2020-03-05	488.170013	60.628559
2020-03-06	494.429993	60.771778

[ ]:

6.3 Merge into a single data-structure and compute the returns

We combine the two list of time series:

[ ]:
```
df_b = pd.DataFrame()
```

[ ]:
```
df_b = pd.concat([df_nfin_b, df_fin_b],axis=1)
df_b.head()
```

[ ]:
```
                  KR        PFE        XOM         WMT        DAL       CSCO  \
Date
2020-03-02  27.611349  29.024046  44.751831  109.201492  46.929810  36.643845
2020-03-03  27.295046  28.541418  42.608925  106.402649  45.954594  35.638077
2020-03-04  28.802132  30.288855  43.539169  110.040199  48.263268  36.839649
2020-03-05  31.137192  29.506672  41.620525  109.239204  44.790302  35.219742
2020-03-06  29.806860  29.140537  39.610516  110.473671  45.666008  35.317657

                 EQIX        DUK        NFLX         GE  …          C  \
Date                                                      …
2020-03-02  583.485107  83.872688  381.049988  68.972809  …  58.315212
2020-03-03  579.977844  82.953018  368.769989  66.942398  …  56.123753
2020-03-04  608.573181  88.193405  383.790009  67.373077  …  58.142658
2020-03-05  581.401428  86.874649  372.779999  62.020157  …  54.777817
2020-03-06  573.859009  85.937622  368.970001  57.893692  …  52.871082

                  GS        USB         MS        KEY         PNC  \
Date
2020-03-02  192.560318  41.078987  41.254646  14.371768  115.400002
2020-03-03  187.007889  39.321865  39.408482  13.728129  109.179337
2020-03-04  191.889252  39.799534  40.154053  14.134640  112.581116
2020-03-05  182.742462  37.342979  37.801964  13.533342  105.003220
2020-03-06  177.282013  36.208542  37.136292  12.593292   99.382843

                 COF         AXP        PRU       SCHW
Date
2020-03-02  85.763748  107.949593  66.210289  40.004227
2020-03-03  81.028786  102.394264  62.465351  36.497421
2020-03-04  83.757240  109.684441  64.221329  35.411083
2020-03-05  79.772369  105.171928  60.135197  33.162148
2020-03-06  77.156425  102.612312  58.096283  32.523682

[5 rows x 27 columns]
```

Comput the returns

[ ]:
```
df_b_returns = df_b.pct_change(axis=0) # daily returns
df_b_returns = df_b_returns.dropna()
df_b_returns.head()
```

[ ]:
```
                  KR       PFE       XOM       WMT       DAL      CSCO  \
Date
2020-03-03 -0.011456 -0.016629 -0.047884 -0.025630 -0.020780 -0.027447
2020-03-04  0.055215  0.061225  0.021832  0.034187  0.050238  0.033716
2020-03-05  0.081072 -0.025824 -0.044067 -0.007279 -0.071959 -0.043972
2020-03-06 -0.042725 -0.012409 -0.048294  0.011301  0.019551  0.002780
2020-03-09 -0.024657 -0.035979 -0.122248 -0.000597 -0.051645 -0.043347
```

```
                EQIX       DUK      NFLX        GE  …         C        GS  \
Date                                                 …
2020-03-03 -0.006011 -0.010965 -0.032227 -0.029438  … -0.037580 -0.028835
2020-03-04  0.049304  0.063173  0.040730  0.006434  …  0.035972  0.026102
2020-03-05 -0.044648 -0.014953 -0.028688 -0.079452  … -0.057872 -0.047667
2020-03-06 -0.012973 -0.010786 -0.010220 -0.066534  … -0.034809 -0.029881
2020-03-09 -0.056862 -0.045129 -0.060926 -0.126596  … -0.161717 -0.103915

                USB        MS       KEY       PNC       COF       AXP  \
Date
2020-03-03 -0.042774 -0.044750 -0.044785 -0.053905 -0.055209 -0.051462
2020-03-04  0.012148  0.018919  0.029611  0.031158  0.033673  0.071197
2020-03-05 -0.061723 -0.058577 -0.042541 -0.067311 -0.047576 -0.041141
2020-03-06 -0.030379 -0.017609 -0.069462 -0.053526 -0.032793 -0.024337
2020-03-09 -0.144405 -0.103729 -0.182246 -0.135516 -0.112043 -0.091925

                PRU      SCHW
Date
2020-03-03 -0.056561 -0.087661
2020-03-04  0.028111 -0.029765
2020-03-05 -0.063626 -0.063509
2020-03-06 -0.033905 -0.019253
2020-03-09 -0.165735 -0.113097

[5 rows x 27 columns]
```

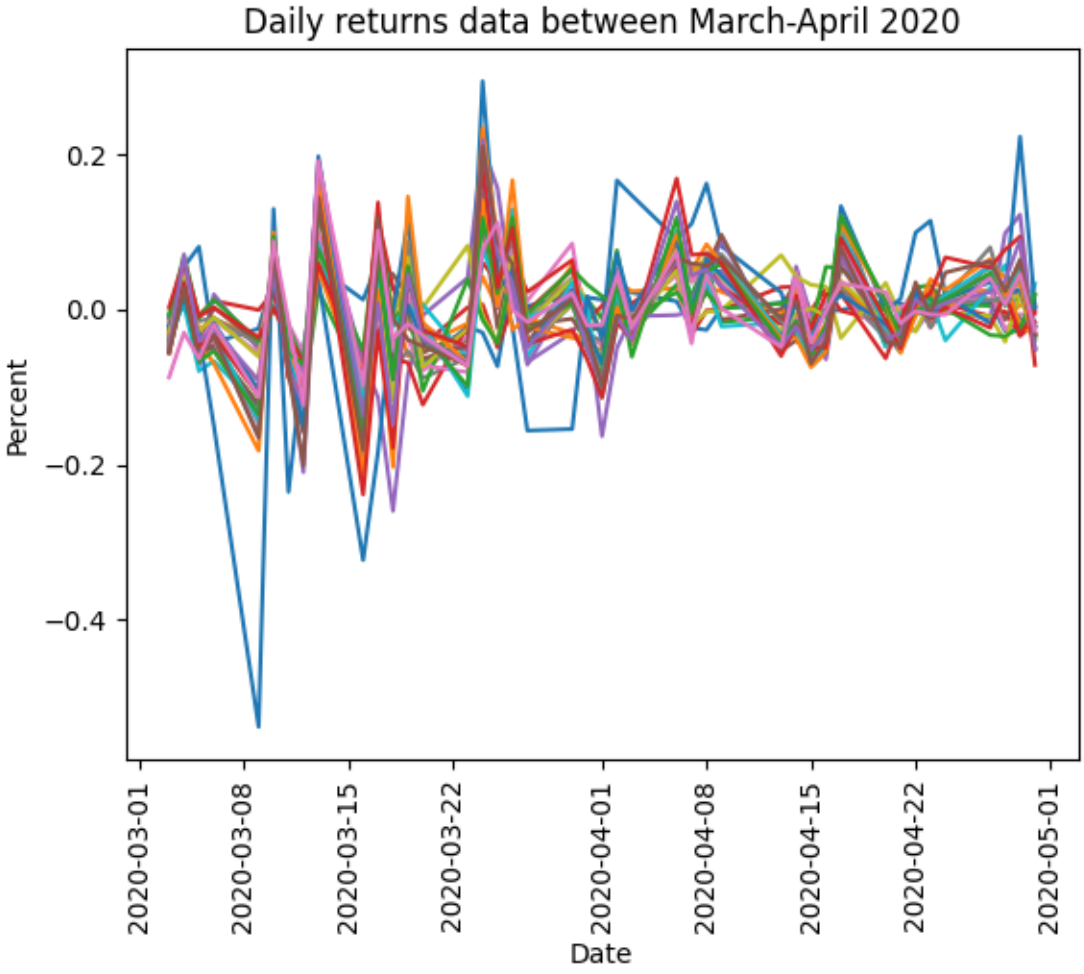
[ ]:
```python
zero_values = (df_b_returns == 0).sum().sum()

if zero_values > 0:
    print(f"There are {zero_values} zero values in the dataset.")
else:
    print("There are no zero values in the dataset.")
```

```
There are 3 zero values in the dataset.
```

[ ]:
```python
df_b_returns = df_b_returns.replace(0, 0.01)


zero_values = (df_b_returns == 0).sum().sum()

if zero_values > 0:
    print(f"There are {zero_values} zero values in the dataset.")
else:
    print("There are no zero values in the dataset.")
```

```
There are no zero values in the dataset.
```

[ ]:
```
plt.plot(df_b_returns.index,df_b_returns)
plt.xlabel("Date")
plt.ylabel("Percent")
plt.title("Daily returns data between March-April 2020")
plt.xticks(rotation=90)
plt.show()
```



7 Step 11

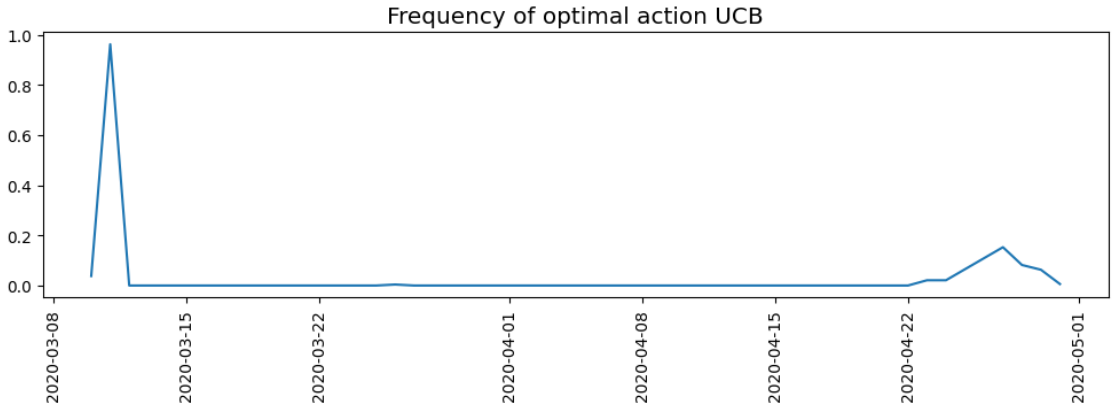
7.0.1 a) UCB algorithm

[ ]:
```
pdata = df_b_returns.to_numpy()
pdata_dates = pd.to_datetime(df_b_returns.index, format='%Y-%m-%d')


#        Bandit problem for stock selection
```

```
NK = pdata.shape[1]

EPSILON = 0.0    # e-greedy setting TURNED OFF
ALPHA =0.9
NEPISODES = 1000
HOLD = 5
TMAX = pdata.shape[0] - HOLD

#        NEW PARAMETER
UCB_WEIGHT =2.0    # UCB setting TURNED ON
seed(1234)
reward_avg = np.zeros((TMAX))
optimal_avg = np.zeros((TMAX))
reward_queue = np.zeros((HOLD,2))

for run in range(NEPISODES):
# Initialize q function and actions record
  qvalue = np.zeros((NK))
  qvalue_up = np.zeros((NK))
  nactions = np.zeros((NK))

  for tt in range(TMAX):
    aa_opt = optimal_action(qvalue_up,EPSILON)
    nactions[aa_opt] += 1

  #        Compute reward as return over holding period
    reward_queue[HOLD-1,0] = (pdata[tt+HOLD,aa_opt]-pdata[tt,aa_opt])/
 ↪pdata[tt,aa_opt]
    reward_queue[HOLD-1,1] = aa_opt

#        Update Q function using action chosen HOLD days before

    qvalue = reward_update(int(reward_queue[0,1]), reward_queue[0,0], qvalue,␣
 ↪ALPHA)

    #qvalue = reward_update(int(reward_queue[0,1]), reward_queue[0,0], qvalue,␣
 ↪1/nactions[aa_opt])
    #print(qvalue)

   #        Upper-confidence adjustment

    qvalue_up = np.zeros((NK))
    for aa in range(NK):
      # If an action has not been visited simply give it the maximum value␣
 ↪across actions
      if nactions[aa] == 0:
        qvalue_up[aa] = np.max(qvalue_up) +1.0
```

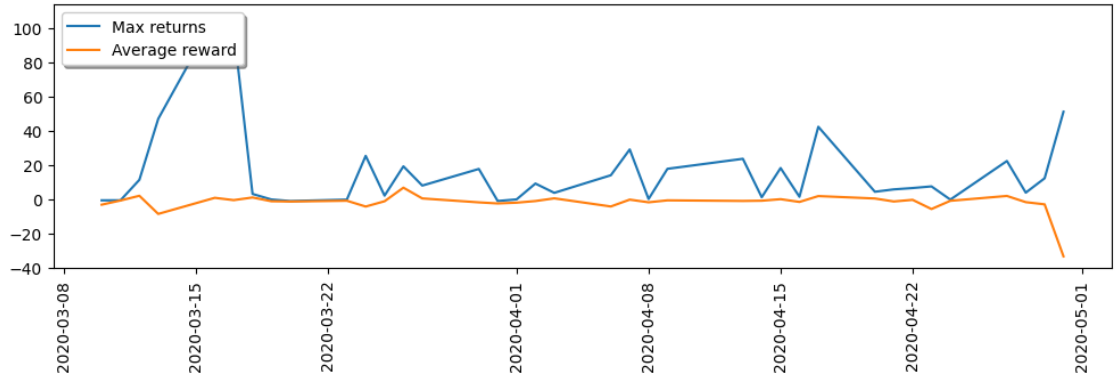
```
      else:
        qvalue_up[aa] = qvalue[aa] + UCB_WEIGHT * np.sqrt(np.log(tt+1)/
↪nactions[aa])

    reward_queue[0:HOLD-1,:] = reward_queue[1:HOLD,:]
    reward_avg[tt] += reward_queue[HOLD-1,0]/NEPISODES
    optimal_avg[tt] += (aa_opt==np.argmax((pdata[tt+HOLD,:]-pdata[tt,:])/
↪pdata[tt,:]))/NEPISODES
```

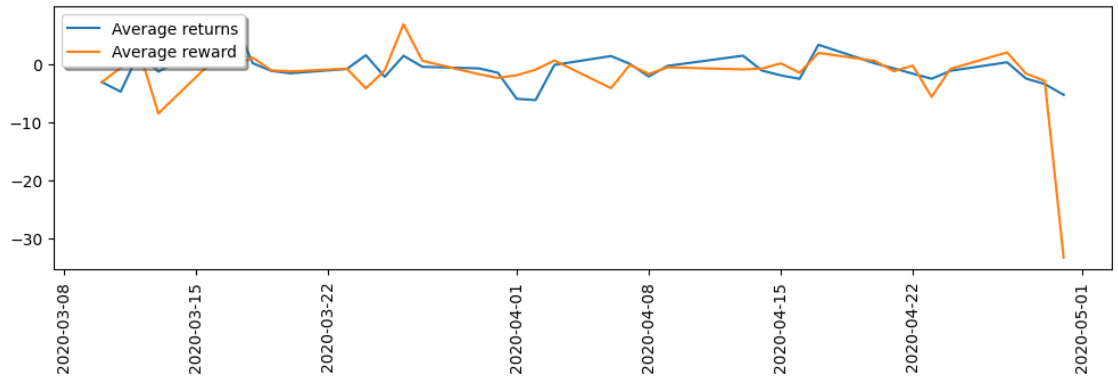
[ ]:
```
plt.plot(pdata_dates[HOLD:pdata.shape [0] ],optimal_avg)

plt.title("Frequency of optimal action UCB", fontsize='x-large')
plt.xticks(rotation=90)
fig = plt.gcf()

fig.set_size_inches(12, 3)

plt.show()
```



[ ]:
```
plt.plot(pdata_dates[HOLD:pdata.shape [0] ],np.max((pdata[HOLD:pdata.shape[0],:
↪]-pdata[0:TMAX,:])/pdata [0:TMAX,:] , axis=1), label='Max returns')

plt.plot(pdata_dates[HOLD:pdata.shape [0] ],reward_avg, label='Average reward')
plt.xticks(rotation=90)
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
fig.set_size_inches(12, 3)
plt.show()

plt.plot(pdata_dates[HOLD:pdata.shape[0] ],np.mean((pdata[HOLD:pdata.shape[0],:
↪]-pdata[0:TMAX,:])/pdata[0:TMAX,:], axis=1), label='Average returns')
plt.plot(pdata_dates[HOLD:pdata.shape[0]],reward_avg, label='Average reward')
```

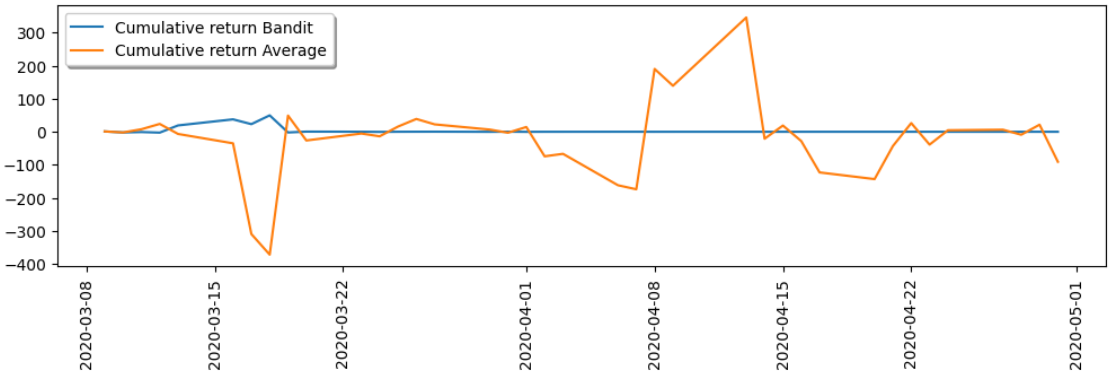
```
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
plt.xticks(rotation=90)
fig.set_size_inches(12, 3)
plt.show()

#        Average frequency of optimal action
print(np.mean(optimal_avg))

#        Average annualized return from holding the equally-weighted portfolio
print((1 + np.mean((pdata[HOLD:pdata.shape[0] ,:]-pdata[0:TMAX,:])/pdata[0:
  ↪TMAX,:])) ** (250 / HOLD) - 1,np.sqrt(250 / HOLD) * np.std(np.mean((pdata))))

#        Average annualized return from holding the Bandit portfolio
print((1+np.mean(reward_avg)) ** (250 / HOLD) - 1 , np.sqrt(250 / HOLD) * np.
  ↪std(reward_avg))
```






```
0.03651351351351353
-1.0 0.0
-0.9999986984228465 40.87261201565781
```

[ ]:
```
return_cumulative = np.zeros((TMAX+1,2))
return_cumulative[0,0] = 1
return_cumulative[0,1] = 1
for tt in range(1,TMAX+1):
  return_cumulative[tt,0] = return_cumulative[tt-1,0] * ( 1 + reward_avg[tt-1] )
  rmean = np.mean((pdata[tt+HOLD-1,:]-pdata[tt-1,:])/pdata[tt-1,:])
  return_cumulative[tt,1] = return_cumulative[tt-1,1] * ( 1 + rmean )

plt.plot(pdata_dates[HOLD-1:pdata.shape[0]],return_cumulative[:,0],␣
 ↪label='Cumulative return Bandit')
plt.plot(pdata_dates[HOLD-1:pdata.shape[0]],return_cumulative[:,1],␣
 ↪label='Cumulative return Average')
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
fig.set_size_inches(12, 3)
plt.xticks(rotation=90)
plt.show()
```

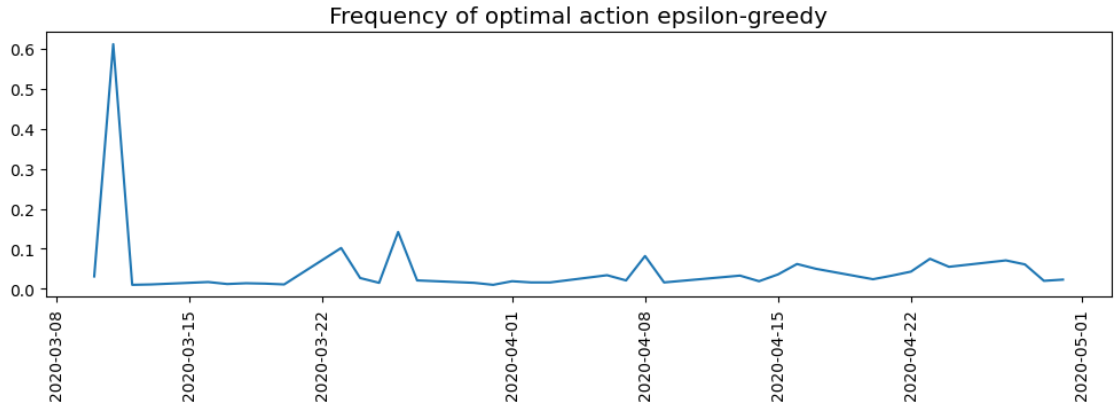


7.0.2 b) e-greedy algorithm

[ ]:
```
pdata = df_b_returns.to_numpy()
pdata_dates = pd.to_datetime(df_b_returns.index, format='%Y-%m-%d')


#        Bandit problem for stock selection
NK = pdata.shape[1]

EPSILON = 0.4
ALPHA =0.9
NEPISODES = 1000
HOLD = 5
TMAX = pdata.shape[0] - HOLD
```

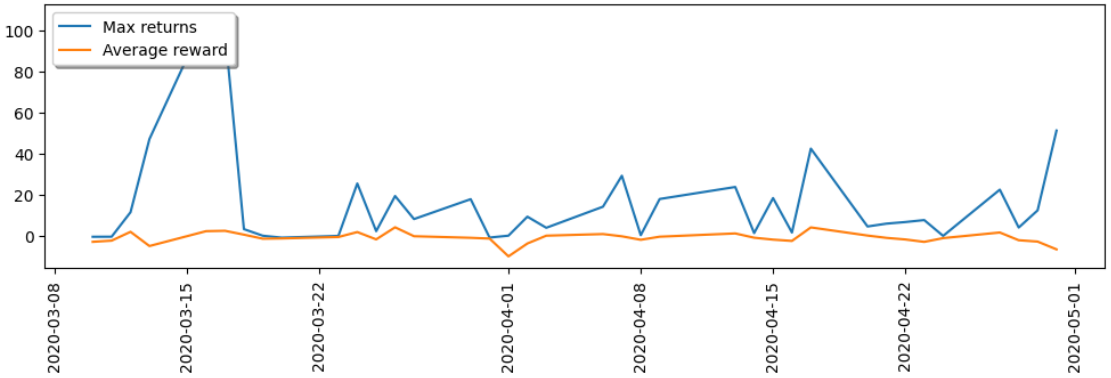
```
#        NEW PARAMETER
UCB_WEIGHT =0.0
seed(1234)
reward_avg = np.zeros((TMAX))
optimal_avg = np.zeros((TMAX))
reward_queue = np.zeros((HOLD,2))

for run in range(NEPISODES):
# Initialize q function and actions record
  qvalue = np.zeros((NK))
  qvalue_up = np.zeros((NK))
  nactions = np.zeros((NK))

  for tt in range(TMAX):
    aa_opt = optimal_action(qvalue_up,EPSILON)
    nactions[aa_opt] += 1

  #        Compute reward as return over holding period
    reward_queue[HOLD-1,0] = (pdata[tt+HOLD,aa_opt]-pdata[tt,aa_opt])/
 ↪pdata[tt,aa_opt]
    reward_queue[HOLD-1,1] = aa_opt

#        Update Q function using action chosen HOLD days before

    #qvalue = reward_update(int(reward_queue[0,1]), reward_queue[0,0], qvalue,␣
 ↪ALPHA)

    qvalue = reward_update(int(reward_queue[0,1]), reward_queue[0,0], qvalue, 1/
 ↪nactions[aa_opt])
    #print(qvalue)

   #        Upper-confidence adjustment

    qvalue_up = np.zeros((NK))
    for aa in range(NK):
      # If an action has not been visited simply give it the maximum value␣
 ↪across actions
      if nactions[aa] == 0:
        qvalue_up[aa] = np.max(qvalue_up) +1.0
      else:
        qvalue_up[aa] = qvalue[aa] + UCB_WEIGHT * np.sqrt(np.log(tt+1)/
 ↪nactions[aa])

    reward_queue[0:HOLD-1,:] = reward_queue[1:HOLD,:]
    reward_avg[tt] += reward_queue[HOLD-1,0]/NEPISODES
    optimal_avg[tt] += (aa_opt==np.argmax((pdata[tt+HOLD,:]-pdata[tt,:])/
 ↪pdata[tt,:]))/NEPISODES
```

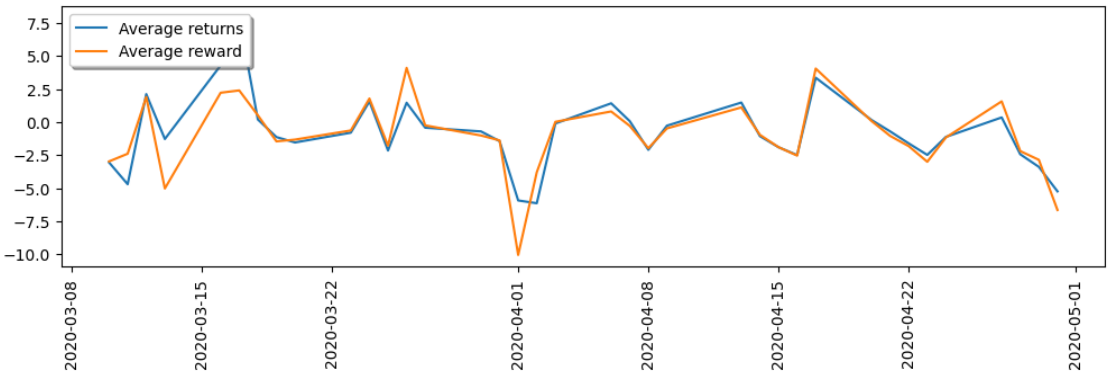
[ ]:
```
plt.plot(pdata_dates[HOLD:pdata.shape [0] ],optimal_avg)

plt.title("Frequency of optimal action epsilon-greedy", fontsize='x-large')
plt.xticks(rotation=90)
fig = plt.gcf()

fig.set_size_inches(12, 3)

plt.show()
```



[ ]:
```
plt.plot(pdata_dates[HOLD:pdata.shape [0] ],np.max((pdata[HOLD:pdata.shape[0],:
 ↪]-pdata[0:TMAX,:])/pdata [0:TMAX,:] , axis=1), label='Max returns')

plt.plot(pdata_dates[HOLD:pdata.shape [0] ],reward_avg, label='Average reward')
plt.xticks(rotation=90)
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
fig.set_size_inches(12, 3)
plt.show()

plt.plot(pdata_dates[HOLD:pdata.shape[0] ],np.mean((pdata[HOLD:pdata.shape[0],:
 ↪]-pdata[0:TMAX,:])/pdata[0:TMAX,:], axis=1), label='Average returns')
plt.plot(pdata_dates[HOLD:pdata.shape[0]],reward_avg, label='Average reward')
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
plt.xticks(rotation=90)
fig.set_size_inches(12, 3)
plt.show()

#        Average frequency of optimal action
print(np.mean(optimal_avg))
```

```
#        Average annualized return from holding the equally-weighted portfolio
print((1 + np.mean((pdata[HOLD:pdata.shape[0] ,:]-pdata[0:TMAX,:])/pdata[0:
  ↪TMAX,:])) ** (250 / HOLD) - 1,np.sqrt(250 / HOLD) * np.std(np.mean((pdata))))

#        Average annualized return from holding the Bandit portfolio
print((1+np.mean(reward_avg)) ** (250 / HOLD) - 1 , np.sqrt(250 / HOLD) * np.
  ↪std(reward_avg))
```

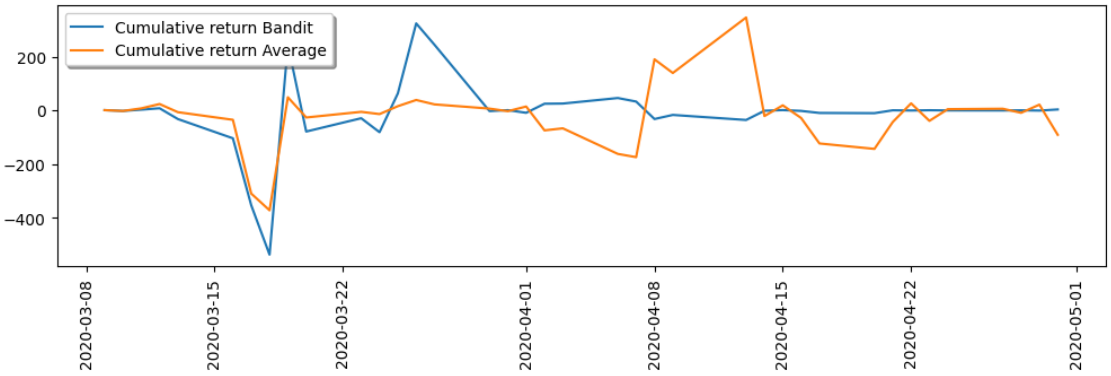





```
0.05056756756756759
-1.0 0.0
-1.0 19.05730139077147
```

[ ]:
```
return_cumulative = np.zeros((TMAX+1,2))
return_cumulative[0,0] = 1
return_cumulative[0,1] = 1
for tt in range(1,TMAX+1):
  return_cumulative[tt,0] = return_cumulative[tt-1,0] * ( 1 + reward_avg[tt-1] )
  rmean = np.mean((pdata[tt+HOLD-1,:]-pdata[tt-1,:])/pdata[tt-1,:])
  return_cumulative[tt,1] = return_cumulative[tt-1,1] * ( 1 + rmean )
```

```
plt.plot(pdata_dates[HOLD-1:pdata.shape[0]],return_cumulative[:,0],␣
 ↪label='Cumulative return Bandit')
plt.plot(pdata_dates[HOLD-1:pdata.shape[0]],return_cumulative[:,1],␣
 ↪label='Cumulative return Average')
legend = plt.legend(loc='upper left', shadow=True)
fig = plt.gcf()
fig.set_size_inches(12, 3)
plt.xticks(rotation=90)
plt.show()
```



[ ]: