Machine Learning Optimization

KAI RU

Keio university

ABSTRACT:

We compared all two tranditional methods of optimization, Black Litterman as maximum returns of a portfolio and Mean Variance Optimization with new methods of Machine Learning Optimization Nested Clustered Optimization (NCO), Convex Optimization(CVO), NCO with Monte Carlo Optimization Selection (MCOS) and CVO with MCOS in this paper by optimizing 92 stocks in US market.

We find that these new Machine learning optimization methods were not able to reach the high performamce as Mean Variance optimization and Black Litterman optimization as they do.

Mean Variance optimization and Black Litterman optimization are still the powerful and main ways to improve portfolio performance.

1. The goal: Build a portfolio from the US stock market, simulate a three-month short-term investment, and evaluate the actual return

by comparing the all 6 models. We use SP100 dataset which contain 92 stocks without missing data.The training period will be from June 16, 2010 to March 17, 2020. The simulation period is from March 18, 2020 to June 16, 2020. As reference to highest Return and sharpe ratio, we introduce Black Litterman model as maximum future return model after three months comparing these to the performance of other portfolios.

(1) As external information, it is first necessary to know the risk-free interest rate and market price. Measured using the 52 Week Treasury Bill as a risk-free interest rate.

In [38]:
```
#Simulation period
import datetime
datetime.datetime(2020, 3, 17)-datetime.datetime(2010, 6, 16)
```

Out[38]: datetime.timedelta(days=3562)

In [39]:
```
import datetime
datetime.datetime(2020, 6, 16)-datetime.datetime(2020, 3, 18)
```

Out[39]: datetime.timedelta(days=90)

In [40]:
```
3562/(52*7)
```

Out[40]: 9.785714285714286

In [41]:
```
import quandl
quandl.ApiConfig.api_key = 'DxKMsvF36hXo5BAMpeDK'
Wk_Bank_Discount_Rate_52=quandl.get("USTREASURY/BILLRATES" ,
                    start_date=datetime.datetime(2010, 6, 16),
                    end_date=datetime.datetime(2020, 3, 17))
```

In [42]:
```
#Downloading bond price
yield_list=[]
for i in range(10):
   yield_list.append(Wk_Bank_Discount_Rate_52[datetime.datetime(2010, 6, 16)+datetime.timedelta(days=364*i):]\
            ["52 Wk Bank Discount Rate"][0])
```

In [43]:
```
yield_list
```

Out[43]: [0.28, 0.18, 0.17, 0.13, 0.1, 0.26, 0.58, 1.14, 2.24, 1.99]

In [ ]:
```
```

Simulation period Yield from October 1, 2012 to September 12, 2019 S = (1 + S0) x (1 + S1) x (1 + S2) x (1 + S3) x (1 + S4) x(1 + S5)x(1 + S6)x(1 + S7)x(1 + S8)x(1 + S9) -1

In [44]:
```
S=(1+yield_list[0]/100)*(1+yield_list[1]/100)*(1+yield_list[2]/100)*\
(1+yield_list[3]/100)*(1+yield_list[4]/100)*(1+yield_list[5]/100)*(1+yield_list[6]/100)\
*(1+yield_list[7]/100)*(1+yield_list[8]/100)*(1+yield_list[9]*(3562/((52*7)*10))/100)-1
```

In [45]:
```
S
```

Out[45]: 0.07223607162892498

In [ ]:
```
```

If you invest $1 in the bond on June 16, 2010, you will have an asset of 1.072 on March 17, 2020. This is defined as a safe asset, and the interest rate of this safe asset is a risk-free interest rate.

In [46]:
```
risk_free=S
```

In [47]:
```
risk_free
```

Out[47]: 0.07223607162892498

In [48]:
```
risk_free_annual=risk_free/9.785714285714286
```

In [49]:
```
risk_free_annual
```

Out[49]: 0.007381788341641968

Risk-free interest rate for simulation period

In [50]:
```
import quandl
quandl.ApiConfig.api_key = 'DxKMsvF36hXo5BAMpeDK'
Wk_Bank_Discount_Rate_8=quandl.get("USTREASURY/BILLRATES" ,
                    start_date=datetime.datetime(2020, 3, 18),
                    end_date=datetime.datetime(2020, 6, 16))
```

In [51]:
```
#Downloading bond price
rate_free_simulation=Wk_Bank_Discount_Rate_8["8 Wk Bank Discount Rate"][0]/100
rate_free_simulation
```

Out[51]: 0.0003

(3) Download the selected stocks as Training Datasets

In [52]:
```
import pandas as pd
import numpy as np
data=pd.read_excel("S&P 100 constituents Aktienkurse-work.xlsx",encoding="SHIFT-JIS",header=3)
data=data.drop(labels=0)
data=data.reset_index(drop= True)
data.index=data["Name"]
data=data.drop(["Name"],axis=1)
data=data.dropna(axis=1)
symbols = data.columns
df = data[symbols]
df=df.astype("float")
data=df[datetime.datetime(2010, 6, 16):datetime.datetime(2020, 3, 17)]
```

In [53]:
```
data.head()
```

Out[53]:

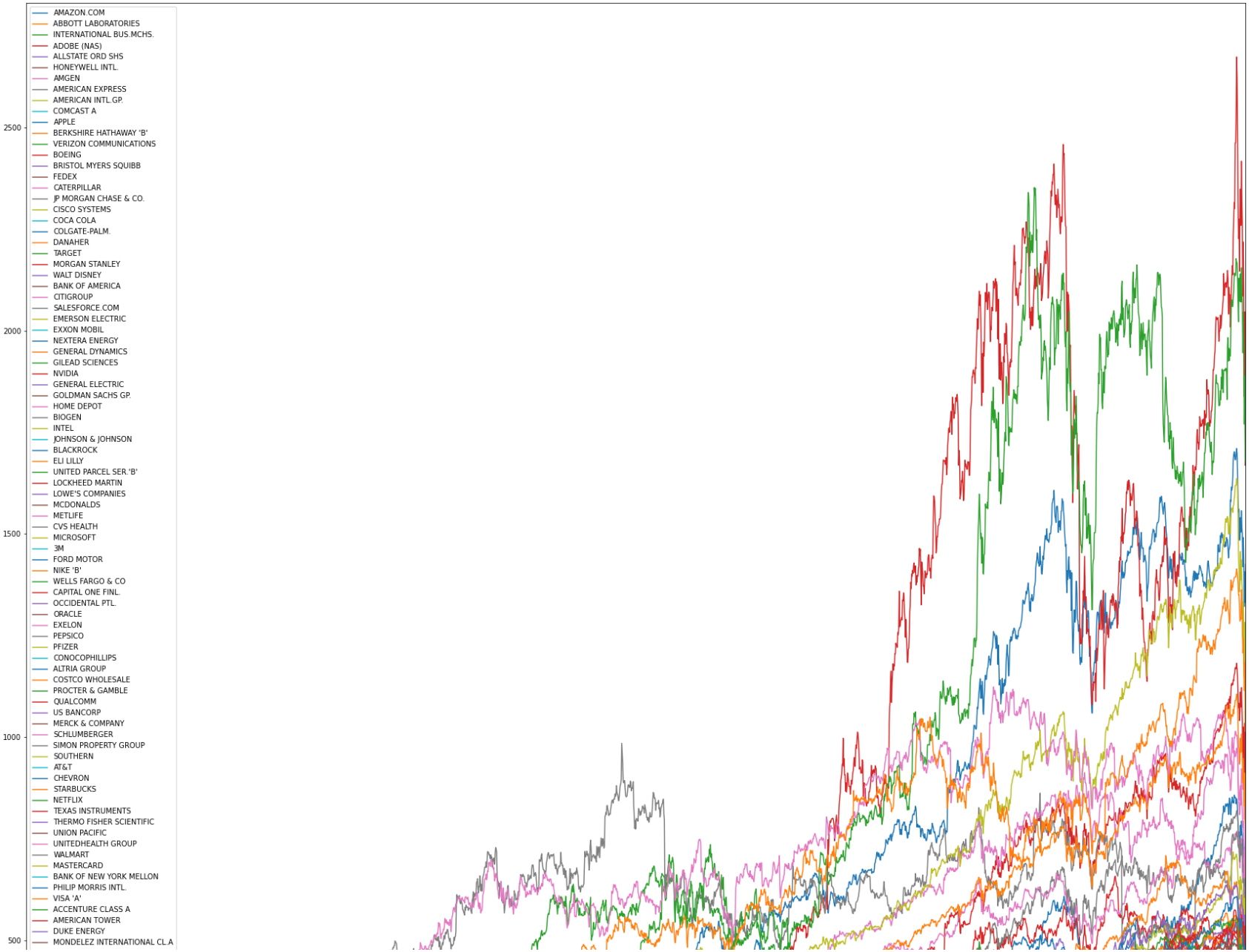
Name	AMAZON.COM	ABBOTT LABORATORIES	INTERNATIONAL BUS.MCHS.	ADOBE (NAS)	ALLSTATE ORD SHS	HONEYWELL INTL.	AMGEN	AMERICAN EXPRESS	AMERICAN INTL.GP.	COMCAS
2010-06-16	126.900	23.3447	130.35	32.4400	30.22	40.6987	55.22	42.34	31.7271	9.19
2010-06-17	125.890	23.2682	130.98	33.1200	30.06	40.6034	55.44	42.06	31.6433	9.29
2010-06-18	125.830	23.3351	130.15	33.5200	30.54	40.8605	55.20	42.03	31.7606	9.25
2010-06-21	122.550	23.1103	130.65	33.1300	30.27	40.8700	56.52	42.60	32.4727	9.17
2010-06-22	122.307	22.9619	129.30	32.7625	30.55	40.1750	56.12	41.94	32.0622	9.12

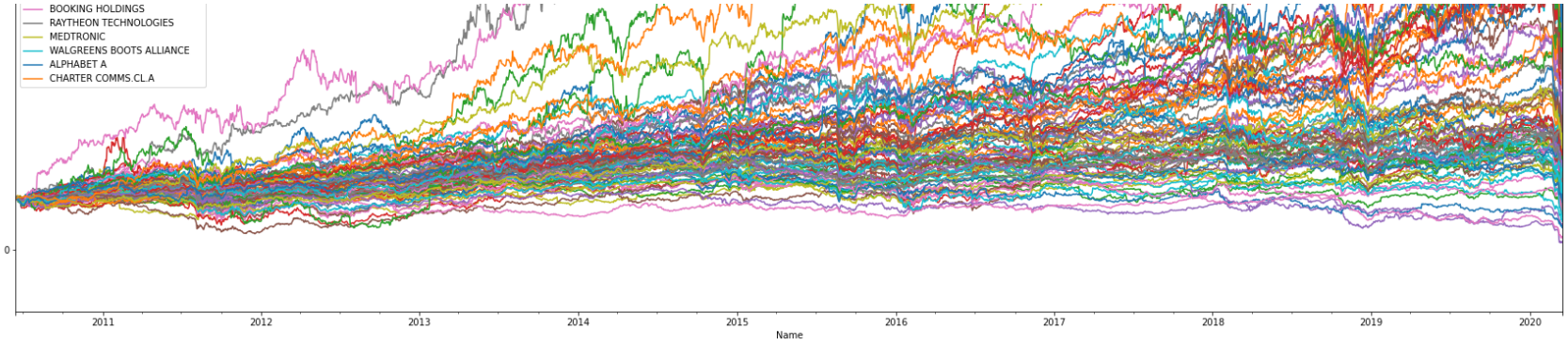
5 rows × 92 columns

(4) Plot time series transition and rate of return

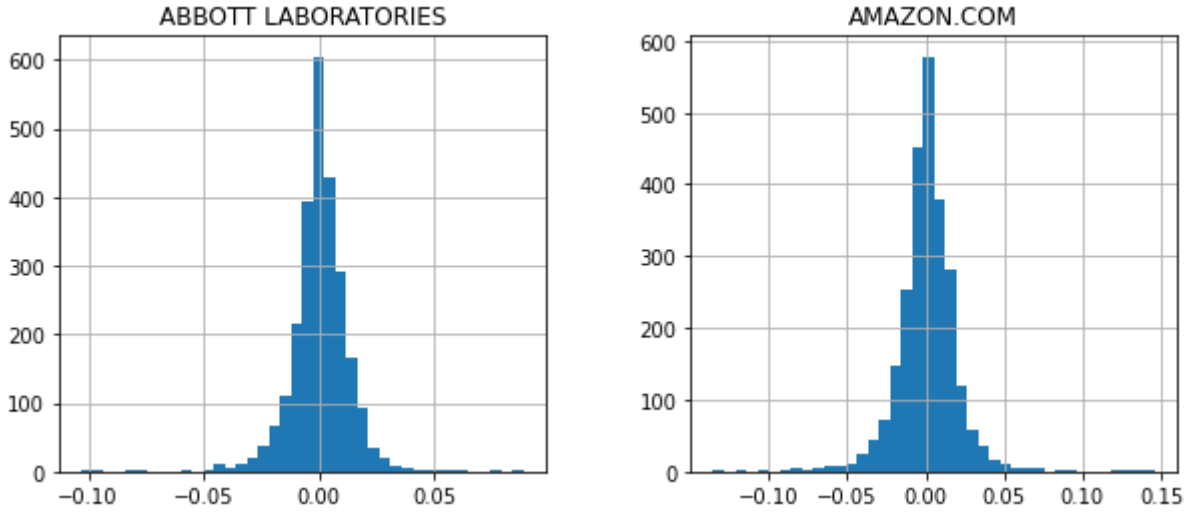
In [ ]:
```
```

In [54]:
```
import matplotlib.pyplot as plt
%matplotlib inline
(data / data.iloc[0] * 100).plot(figsize=(30, 30))
plt.savefig('stat_01.png')
```





In [55]:
```
noa = len(symbols)
data = data[symbols]
rets = np.log(data / data.shift(1))
rets[symbols[:2]].hist(bins=40, figsize=(10, 4))
plt.savefig('stat_2.png')
```



2, mean variance model

(1) Model optimization

In [56]:
```
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns

mu = expected_returns.mean_historical_return(data)
S = risk_models.sample_cov(data,frequency=252)

#mean variance model optimization
EF_min = EfficientFrontier(mu, S)

EF_min.min_volatility()
#portfolio performance
EF_min.portfolio_performance(verbose=True)
```

```
Expected annual return: 7.5%
Annual volatility: 11.9%
Sharpe Ratio: 0.46
```

Out[56]:
```
(0.07456143876531135, 0.11886679211421806, 0.4590133021582995)
```

In [57]:
```
#CAPM optimization
#Enter non-risky asset
EF = EfficientFrontier(mu, S)
weights = EF.max_sharpe(risk_free_rate=risk_free_annual)
#Portfolio ratio
EF.portfolio_performance(verbose=True)
```

```
Expected annual return: 23.1%
Annual volatility: 16.2%
Sharpe Ratio: 1.31
```

Out[57]:
```
(0.23130038471727615, 0.16153275450818833, 1.3080962146693602)
```

In [371]:
```
#Weights in each stock
EF_clean_weights=EF.weights
```

In [ ]:
```
```

(2) Download the data of each stock from March 18, 2020 to June 16, 2020 will

be collected for simulation.

In [59]:
```
import pandas as pd
import numpy as np
data2=pd.read_excel("S&P 100 constituents Aktienkurse-work.xlsx",encoding="SHIFT-JIS",header=3)
data2=data2.drop(labels=0)
data2=data2.reset_index(drop= True)
data2.index=data2["Name"]
data2=data2.drop(["Name"],axis=1)
data2=data2.dropna(axis=1)
symbols2 = data2.columns
df2 = data2[symbols]
df2=df2.astype("float")
data2=df2[datetime.datetime(2020, 3, 18):datetime.datetime(2020, 6, 16)]
```

(3) If managed from March 18, 2020 to June 16, 2020, the average return of the portfolio will be

R = 1r1 + w2r2 + ... + wn * rn

ri = Return of individual stock

wi = weight of individual stock

R = average revenue of the portfolio

In [374]:
```
Mean_variance_return=np.sum(EF.weights*np.array(expected_returns.mean_historical_return(data2, frequency=90*252/36
Mean_variance_return
```

Out[374]: 0.2778270880497722

(4) Volatility of the mean variance model portfolio

In [375]:
```
from pypfopt import objective_functions
Mean_variance_Volatility=np.sqrt(objective_functions.portfolio_variance(EF.weights, risk_models.sample_cov(data2, frequer
Mean_variance_Volatility
```

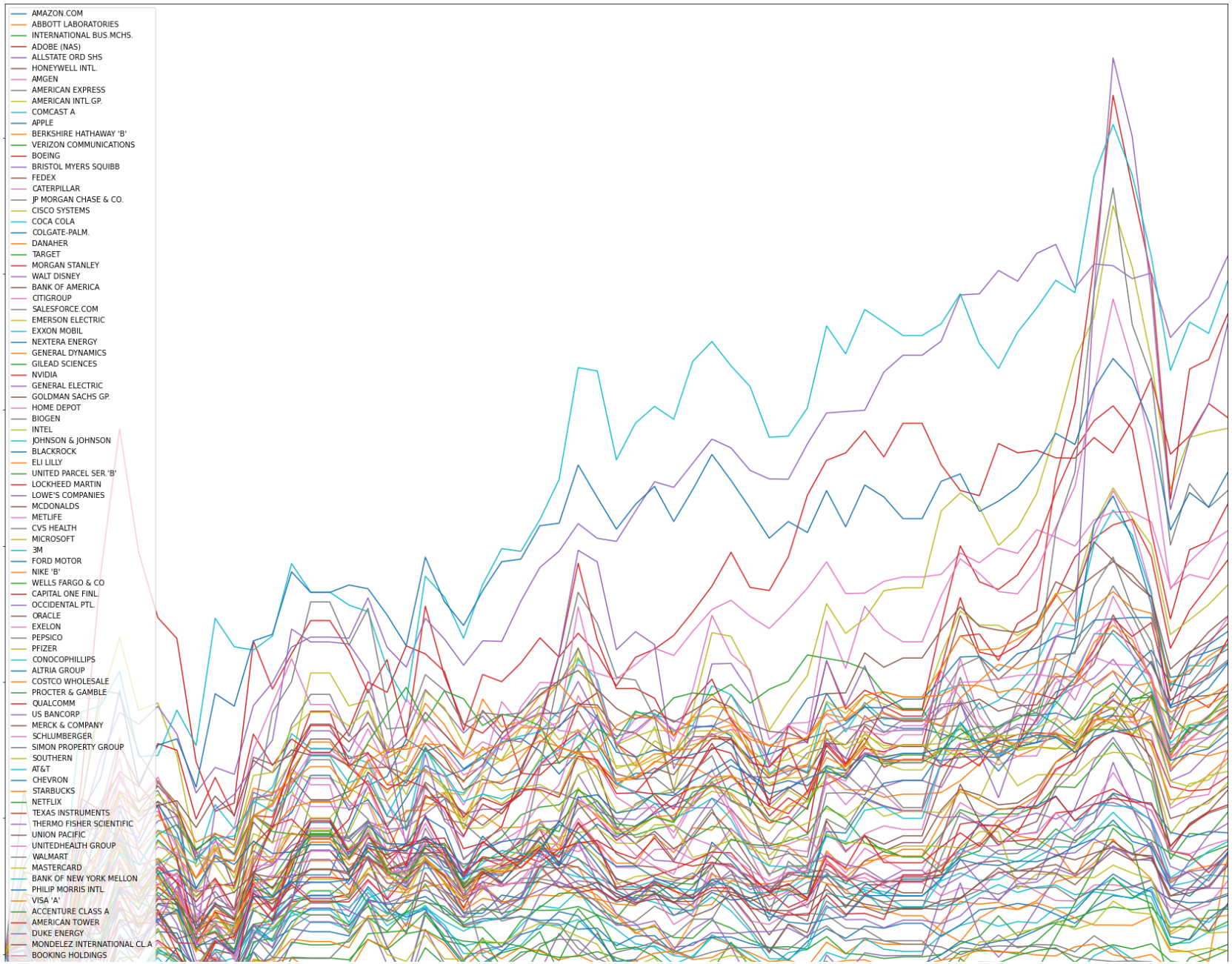
```
/usr/local/anaconda3/lib/python3.7/site-packages/pypfopt/risk_models.py:69: UserWarning: The covariance matrix is non
positive semidefinite. Amending eigenvalues.
  "The covariance matrix is non positive semidefinite. Amending eigenvalues."
/usr/local/anaconda3/lib/python3.7/site-packages/pypfopt/risk_models.py:88: UserWarning: Could not fix matrix. Please tr
y a different risk model.
  warnings.warn("Could not fix matrix. Please try a different risk model.")
```

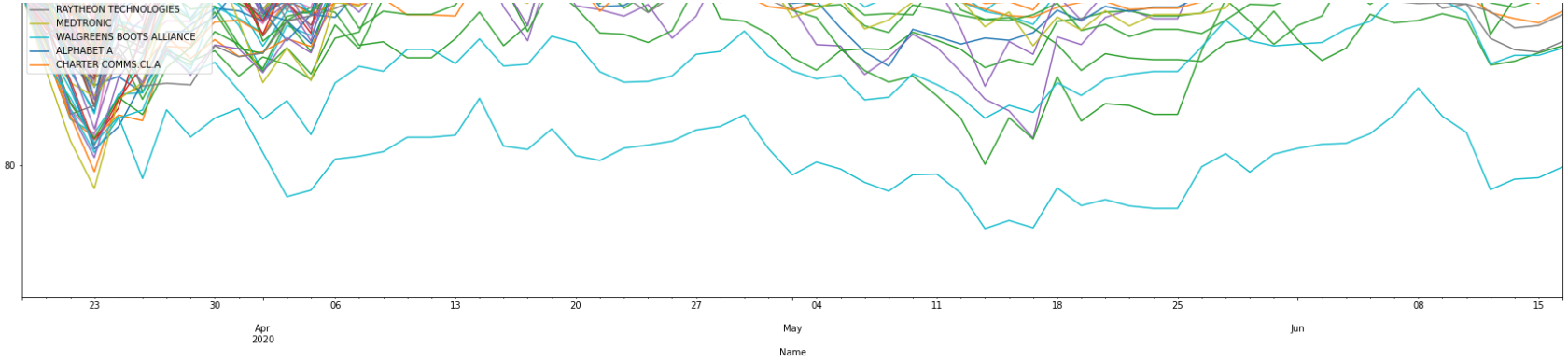
Out[375]: 0.1834288042950433

3. Black – Litterman model

(1) For the maxmize the Return of portfolio, calculate the actual return of each stock from March 18, 2020 to June 16, 2020

In [62]:
```
import matplotlib.pyplot as plt
%matplotlib inline
(data2 / data2.iloc[0] * 100).plot(figsize=(30, 30))
plt.savefig('stat_02.png')
```





In [63]:
```
expected_returns.mean_historical_return(data2, frequency=90*252/365)
```

Out[63]:
```
AMAZON.COM                0.414333
ABBOTT LABORATORIES       0.128138
INTERNATIONAL BUS.MCHS.   0.201948
ADOBE (NAS)               0.383795
ALLSTATE ORD SHS          0.301649
                            ...
RAYTHEON TECHNOLOGIES     0.432650
MEDTRONIC                 0.272630
WALGREENS BOOTS ALLIANCE  -0.196600
ALPHABET A                0.314758
CHARTER COMMS.CL.A        0.382126
Length: 92, dtype: float64
```

(3) Setting critic reviews for each brand

Referring to the above figures and data supposed that I could correctly predict the future returns after three months, exactly same as the actual returns above.

In [376]:
```
from pypfopt.black_litterman import BlackLittermanModel
S = risk_models.sample_cov(data)

viewdict = expected_returns.mean_historical_return(data2, frequency=90*252/365)
```

In [377]:
```
bl = BlackLittermanModel(S, absolute_views=viewdict)
rets = bl.bl_returns()
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/pypfopt/black_litterman.py:257: UserWarning: Running Black-Litterman with no prior.
  warnings.warn("Running Black-Litterman with no prior.")
```

(4) Calculate the return of each brand

In [378]:
```
rets
```

Out[378]:
```
AMAZON.COM                0.350793
ABBOTT LABORATORIES       0.154719
INTERNATIONAL BUS.MCHS.   0.211473
ADOBE (NAS)               0.365374
ALLSTATE ORD SHS          0.254778
                            ...
RAYTHEON TECHNOLOGIES     0.339975
MEDTRONIC                 0.245333
WALGREENS BOOTS ALLIANCE -0.002802
ALPHABET A                0.310377
CHARTER COMMS.CL.A        0.299549
Length: 92, dtype: float64
```

In [ ]:
```

```

(5) Introduce SP500 as market price

In [67]:
```
import pandas_datareader as pdr
SP500 = pdr.get_data_yahoo('^GSPC',
                           start=datetime.datetime(2010, 6, 16),
                           end=datetime.datetime(2020, 3, 17))
```

In [68]:
```
market_prices=SP500["Close"]
```

In [ ]:

(6) Black – Litterman model simulation

In [379]:
```
from pypfopt import black_litterman

delta = black_litterman.market_implied_risk_aversion(market_prices,risk_free_rate=risk_free_annual)

ef = EfficientFrontier(rets, S)

bl.bl_weights(delta)
weights = bl.clean_weights()
```

In [380]:
```
bl.portfolio_performance(verbose=True)
```

```
Expected annual return: 395.5%
Annual volatility: 98.8%
Sharpe Ratio: 3.98
```

Out[380]: (3.9552438689625338, 0.9884899400192397, 3.981066179475675)

In [381]:
```
Black_Litterman_weights=weights
```

In [382]:
```
sum(weights.values())
```

Out[382]: 0.9999999999999998

In [ ]:

(7) If managed from March 18, 2020 to June 16, 2020, the average return of the portfolio will be

R = 1r1 + w2r2 + ... + wn * rn

ri = Return of individual stock

wi = weight of individual stock

R = average revenue of the portfolio

In [383]:
```
BL_return=np.sum(np.array(bl.weights)*np.array(expected_returns.mean_historical_return(data2, frequency=90*252/365))
```

In [384]:
```
BL_return
```

Out[384]: 7.855041971946283

(8) Portfolio volatility

In [387]:
```
from pypfopt import objective_functions
Black_Litterman_volatility=np.sqrt(objective_functions.portfolio_variance(bl.weights, risk_models.sample_cov(data2, freque
```

In [388]:
```
Black_Litterman_volatility
```

Out[388]: 2.198686649036785

4. Machine Learning Optimization, Nested Clustered Optimization algorithm(NCO), Convex Optimization Solution(CVO) and Monte Carlo approach(MCOS)

(1)Calculate the Return and Variance of Data

In [159]:
```
from pypfopt.expected_returns import mean_historical_return
from pypfopt.risk_models import sample_cov
assets_mean=mean_historical_return(data,frequency=252)
assets_cov=sample_cov(data,frequency=252)
```

(2)Optimization of NCO & CVO

In [ ]:

In [182]:
```
import pandas as pd
from mlfinlab.portfolio_optimization import NCO


# Import dataframe of returns for assets in a portfolio
max_num_clusters=91

# Calculate empirical covariance of assets
assets_cov = np.array(assets_cov)

# Calculate empirical means of assets
assets_mean =  np.array(assets_mean).reshape(-1, 1)

# Class that contains needed functions
nco = NCO()

# Find optimal weights using the NCO algorithm
w_nco = nco.allocate_nco(assets_cov, assets_mean,max_num_clusters,n_init=10)

# Find optimal weights using the CVO algorithm
w_cvo = nco.allocate_cvo(assets_cov, assets_mean)
```

In [ ]:

In [183]:
```
nco_weights= w_nco/sum(w_nco)
```

In [188]:
```
cvo_weights=(w_cvo/sum(w_cvo))
```

(3)Return of NCO method

In [337]:
```
NCO_return=np.sum(nco_weights.flatten()*np.array(expected_returns.mean_historical_return(data2, frequency=90*252/36
```

In [338]:
```
NCO_return
```

Out[338]: -0.23033475252864208

(4)Return of CVO method

In [339]:
```
CVO_return=np.sum(cvo_weights.flatten()*np.array(expected_returns.mean_historical_return(data2, frequency=90*252/36
```

In [340]:
```
CVO_return
```

Out[340]: 0.013069365050046328

(5)Variance of NCO method

In [347]:
```
NCO_volatility=np.sqrt(objective_functions.portfolio_variance(nco_weights.flatten(), risk_models.sample_cov(data2, freque
```

In [342]:
```
NCO_volatility
```

Out[342]: 0.740664398246723

(6)Variance of CVO method

In [343]:
```
CVO_volatility=np.sqrt(objective_functions.portfolio_variance(cvo_weights.flatten(), risk_models.sample_cov(data2, freque
```

In [344]:
```
CVO_volatility
```

Out[344]: 1.0560104662365144

(7)Optimization of MCOS(Parameters are: 10 simulations, 2545 observations in a simulation)

In [277]:
```
# Compare the NCO solutions to the CVO ones using MCOS
# Parameters are: 1 simulations, 2545 observations in a simulation
# goal of maximum sharpe ratio, using LW shrinkage

w_cvo_mcos, w_nco_mcos = nco.allocate_mcos(assets_mean, assets_cov, 2545, 10, 0.01, False, False)

# Find the errors in estimations of NCO and CVO in simulations
err_cvo_mcos, err_nco_mcos = nco.estim_errors_mcos(w_cvo, w_nco, assets_mean, assets_cov, False)
```

(8)Summary the returns of each simulation

In [293]:
```
weight_nco_mcos=(w_cvo_mcos.sum(axis=0)/10)/sum((w_cvo_mcos.sum(axis=0)/10))
weight_cvo_mcos=(w_nco_mcos.sum(axis=0)/10)/sum((w_nco_mcos.sum(axis=0)/10))
```

In [354]:
```
nco_mcos_return=sum(np.array(weight_nco_mcos)*np.array(expected_returns.mean_historical_return(data2, frequency=9(
```

In [355]:
```
nco_mcos_return
```

Out[355]: 0.11099875049713796

In [356]:
```
cvo_mcos_return=sum(np.array(weight_cvo_mcos)*np.array(expected_returns.mean_historical_return(data2, frequency=9(
```

In [357]:
```
cvo_mcos_return
```

Out[357]: 0.03370507421840299

(10)Volatility of MCOS

In [358]:
```
nco_mcos_volatility=np.sqrt(objective_functions.portfolio_variance(np.array(weight_nco_mcos), (risk_models.sample_cov(c
```

In [359]:
```
nco_mcos_volatility
```

Out[359]: 0.9643433615472587

In [360]:
```
cvo_mcos_volatility=np.sqrt(objective_functions.portfolio_variance(np.array(weight_cvo_mcos), risk_models.sample_cov(d
```

In [361]:
```
cvo_mcos_volatility
```

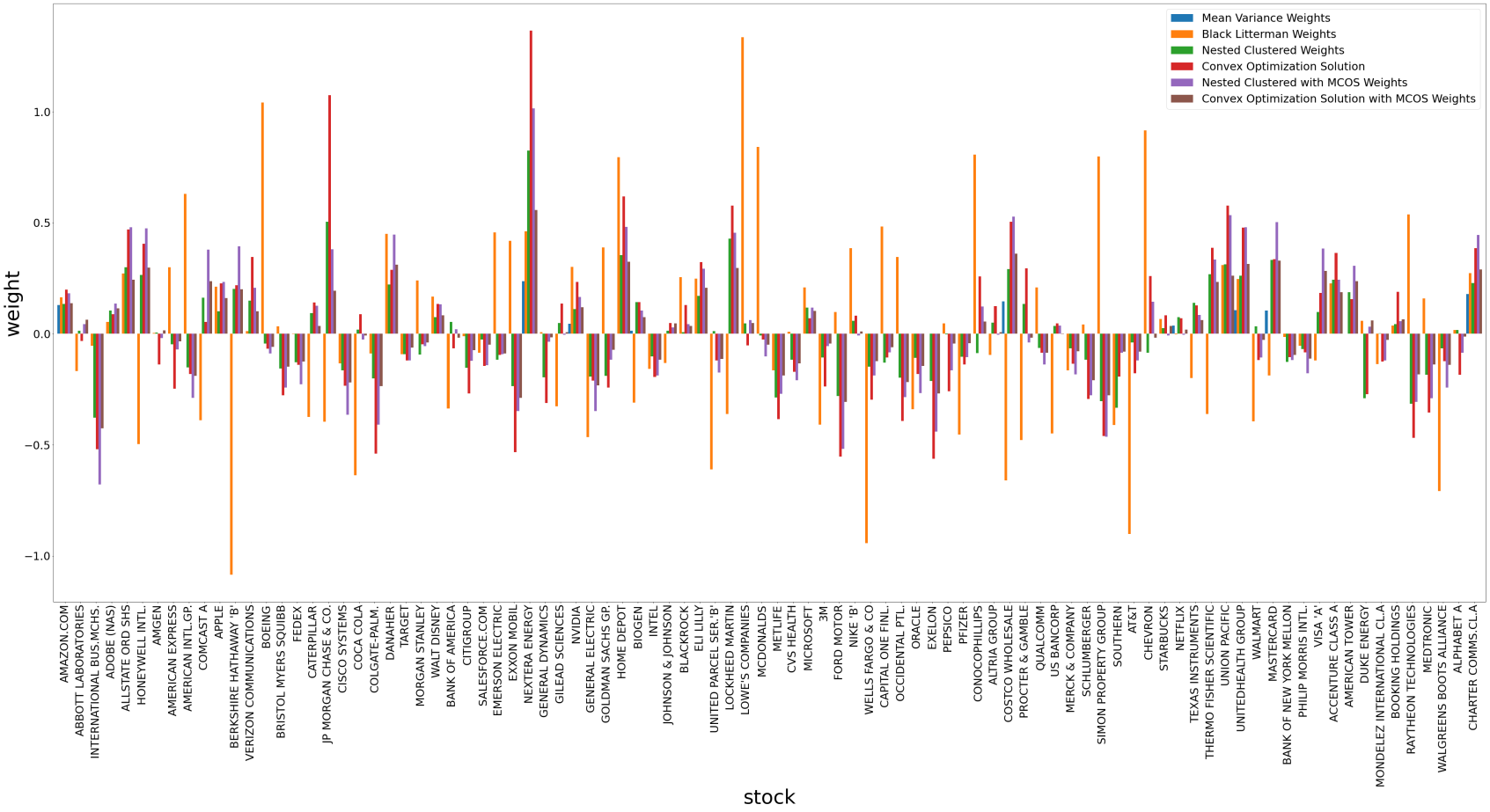
Out[361]: 0.6995856305646891

4, Portfolio comparison

(1) Portfolio weight comparison

In [389]:
```
df = pd.DataFrame([EF.weights.reshape(92),np.array(list(bl.weights)).reshape(92),(w_nco/sum(w_nco)).reshape(92),\
                   (w_cvo/sum(w_cvo)).reshape(92),weight_nco_mcos,weight_cvo_mcos],
                  columns=data.columns,
                  index=['Mean Variance Weights','Black Litterman Weights','Nested Clustered Weights','Convex Optimization Solution
                         'Nested Clustered with MCOS Weights','Convex Optimization Solution with MCOS Weights'])
plot=df.T.plot(kind='bar',figsize=(70, 30),legend=True, fontsize=30,width=1)
plot.legend(loc=1,fontsize=30)
plot.set_xlabel('stock',fontdict={'fontsize':54})
plot.set_ylabel('weight',fontdict={'fontsize':54})
```

Out[389]: Text(0, 0.5, 'weight')



(2)Comparison of trained portfolios average return and volatility

In [397]:
```
nco_por_ret=np.sum(nco_weights.flatten()*np.array(expected_returns.mean_historical_return(data, frequency=252)))
cvo_por_ret=np.sum(cvo_weights.flatten()*np.array(expected_returns.mean_historical_return(data, frequency=252)))
nco_mcos_por_ret=np.sum(np.array(weight_nco_mcos)*np.array(expected_returns.mean_historical_return(data, frequency
cvo_mcos_por_ret=np.sum(np.array(weight_cvo_mcos)*np.array(expected_returns.mean_historical_return(data, frequency


nco_por_var=np.sqrt(objective_functions.portfolio_variance(nco_weights.flatten(), (risk_models.sample_cov(data, frequenc
cvo_por_var=np.sqrt(objective_functions.portfolio_variance(cvo_weights.flatten(), risk_models.sample_cov(data, frequency
nco_mcos_por_var=np.sqrt(objective_functions.portfolio_variance(np.array(weight_nco_mcos), (risk_models.sample_cov(d
cvo_mcos_por_var=np.sqrt(objective_functions.portfolio_variance(np.array(weight_cvo_mcos), risk_models.sample_cov(da
```

In [398]:
```
df = pd.DataFrame([[EF.portfolio_performance(verbose=True)[0],bl.portfolio_performance(verbose=True)[0],nco_por_ret,cv
                    nco_mcos_por_ret,cvo_mcos_por_ret],\
                    [EF.portfolio_performance(verbose=True)[1],\
                    bl.portfolio_performance(verbose=True)[1]
                    ,nco_por_var,cvo_por_var,nco_mcos_por_var,cvo_mcos_por_var],\
                    [EF.portfolio_performance(verbose=True)[2],\
                    bl.portfolio_performance(verbose=True)[2]\
                     ,(nco_por_ret-risk_free_annual)/NCO_volatility,(cvo_por_ret-risk_free_annual)/cvo_por_var,\
                    (nco_mcos_por_ret-risk_free_annual)/nco_mcos_por_var,(cvo_mcos_por_ret-risk_free_annual)/cvo_mcos_por
                    ]],columns=["Mean Variance simulated value","Black Litterman simulated value",'Nested Clustered simulated value'
                        'Nested Clustered with MCOS simulated value','Convex Optimization Solution with MCOS simulated value'],index=

df
```

```
Expected annual return: 23.1%
Annual volatility: 16.2%
Sharpe Ratio: 1.31
Expected annual return: 395.5%
Annual volatility: 98.8%
Sharpe Ratio: 3.98
Expected annual return: 23.1%
Annual volatility: 16.2%
Sharpe Ratio: 1.31
Expected annual return: 395.5%
Annual volatility: 98.8%
Sharpe Ratio: 3.98
Expected annual return: 23.1%
Annual volatility: 16.2%
Sharpe Ratio: 1.31
Expected annual return: 395.5%
Annual volatility: 98.8%
Sharpe Ratio: 3.98
```

Out[398]:

	Mean Variance simulated value	Black Litterman simulated value	Nested Clustered simulated value	Convex Optimization simulated value	Nested Clustered with MCOS simulated value	Convex Optimization Solution with MCOS simulated value
Return	0.231300	3.955244	1.170317	1.672693	1.632810	1.164744
Volatility	0.161533	0.988490	0.366028	0.489439	0.511006	0.372097

	Mean Variance simulated value	Black Litterman simulated value	Nested Clustered simulated value	Convex Optimization simulated value	Nested Clustered with MCOS simulated value	Convex Optimization Solution with MCOS simulated value
Portfolio Sharpe Ratio	1.308096	3.981066	1.570125	3.402491	3.180838	3.110376

(3) Comparison of simulated portfolios average return and volatility

In [390]:
```
risk_free_3months=rate_free_simulation
```

In [391]:
```
df2 = pd.DataFrame([[Mean_variance_return,BL_return,NCO_return,CVO_return,\
                     nco_mcos_return,cvo_mcos_return],\
                     [Mean_variance_Volatility,\
                     Black_Litterman_volatility
                     ,NCO_volatility,CVO_volatility,nco_mcos_volatility,cvo_mcos_volatility],\
                     [(Mean_variance_return-risk_free_3months)/Mean_variance_Volatility,\
                      (BL_return-risk_free_3months)/Black_Litterman_volatility\
                      ,(NCO_return-risk_free_3months)/NCO_volatility,(CVO_return-risk_free_3months)/CVO_volatility,\
                     (nco_mcos_return-risk_free_3months)/nco_mcos_volatility,(cvo_mcos_return-risk_free_3months)/cvo_mcos_v
                     ]],columns=["Mean Variance simulated value","Black Litterman simulated value",'Nested Clustered simulated value'
                         'Nested Clustered with MCOS simulated value','Convex Optimization Solution with MCOS simulated value'],index=

df2
```

Out[391]:

	Mean Variance simulated value	Black Litterman simulated value	Nested Clustered simulated value	Convex Optimization simulated value	Nested Clustered with MCOS simulated value	Convex Optimization Solution with MCOS simulated value
Return	0.277827	7.855042	-0.230335	0.013069	0.110999	0.033705
Volatility	0.183429	2.198687	0.740664	1.056010	0.964343	0.699586
Portfolio Sharpe Ratio	1.512996	3.572470	-0.311389	0.012092	0.114792	0.047750

5, conclusion

We are comparing 6 models, the mean variance model ,the Black Litterman model, Nested Clustered simulated value, Convex Optimization simulated value, Nested Clustered with MCOS simulated value and Convex Optimization Solution with MCOS simulated value from March 18, 2020 to June 16, 2020,We improved Black Litterman model as maximum return of portfolio compare to other portfolios of optimization.

The average return of Black Litterman simulated value is much higher than other models.

Disappointly the new Machine learning optimization Nested Clustered simulated value and Convex Optimization resulted in lower simulated Return and Sharpe Ratio than the Black Litterman in maximum performance and Mean Variance optimization though these new models' training Return and Sharpe Ratio are higher.

This paper supposed that Mean Variance optimization and Black Litterman optimization are still the normal ways to improve portfolio optimization.

Nested Clustered with MCOS simulated value is the model that only have a positive return though its convergence is not stable if by several times of simulations.

6, reference list

translated by David G. Ruenberger, Hiroshi Konno, Kenichi Suzuki, Norio Bibiki, "Introduction to Financial Engineering: Second Edition," Nihon Keizai Shimbun (2015)

Takahiro Komatsu “Optimal Investment Strategy” Asakura Shoten (2018)

PyPortfolioOpt, https://pyportfolioopt.readthedocs.io/en/latest/ (https://pyportfolioopt.readthedocs.io/en/latest/)

Machine Learning Financial Laboratory (mlfinlab) https://mlfinlab.readthedocs.io/en/latest/index.html (https://mlfinlab.readthedocs.io/en/latest/index.html)

A ROBUST ESTIMATOR OF THE EFFICIENT FRONTIER https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3469961 (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3469961)

López de Prado Machine Learning for Asset Managers

López de Prado Advances in Financial Machine Learning

In [ ]: