# Import the Bloomberg Query Language (BQL) and bqfactor libraries

import bql

import bqport

# Import other data analytics and chatting libraries

import pandas as pd

import bqplot as bqp

import bqviz as bqv

import numpy as np

from numpy.linalg import inv

from collections import OrderedDict

import scipy

import bqwidgets as bqw

from ipywidgets import HBox, VBox, IntSlider, Text, Tab, FloatText, Label, Layout, FloatText, IntText, Checkbox, Button, FloatSlider, Dropdown, HTML

import bqplot as bqp

# Loading animation

loading\_html = HTML("""

<div style="font-size:14px; color:lightskyblue;">

<i class="fa fa-circle-o-notch fa-spin"></i><span>&nbsp;Loading...</span>

</div>""")

preload\_box = HBox([loading\_html])

preload\_box

# Instantiate an object to interface with the BQL service

bq = bql.Service()

#Default settings

security = OrderedDict()

security['1-5 years GILTS'] = 'LF56TRGU Index'

security['Cash'] = 'DBDCONIA Index'

security['Chinese Bonds'] = 'I32561US Index'

security['Chinese Equity'] = 'SHSZ300 Index'

security['Emerging Asia Equity'] = 'NDUEEGFA Index'

security['EU High Yield Bonds'] = 'EUNW GY Equity'

security['European Banks'] = 'SX7E Index'

security['European Corp'] = 'EUN5 GR Equity'

security['European Equity'] = 'SXXE Index'

security['German Equity'] = 'DAX Index'

security['Greek Equity'] = 'FTASE Index'

security['Greek Govies'] = 'BEGCGA Index'

security['Italian Equity'] = 'FTSEMIB Index'

security['MSCI Info tech'] = 'NDWUIT Index'

security['MSCI World'] = 'MACXUIGB Index'

security['Spanish Equity'] = 'IBEX Index'

security['US Equity'] = 'SPX Index'

security['US High Yield Bonds'] = 'IBXXHYCT Index'

#security["Crypto Currency"] = 'GBTC US Equity'

approximated\_mkt\_weight = [0.0112878580039961,0.164879596149528,0.0248550020344915,0.00957643167488187,0.010241765265639,0.398894134073001,0.00416351972379412,0.0967099088024052,0.0828703866165383,0.0235103219298358,0.0125595027532384,0.0120035820663699,0.0106296429781949,0.0202795023703381,0.035435880040154,0.00992384006540524,0.0311647410666334,0.0410143843855553]

rf = 0.015 # rf is the risk-free rate

num\_avail\_ticker=20

uncertainty = 0.025 # tau is a scalar indicating the uncertainty in the CAPM (Capital Asset Pricing Model)

import pickle

from collections import OrderedDict

from datetime import timedelta

try:

f = open('settings\_bl.pckl', 'rb')

dict\_settings = pickle.load(f)

f.close()

except:

dict\_settings = OrderedDict()

dict\_settings['security'] = security

dict\_settings['weight'] = approximated\_mkt\_weight

dict\_settings['confidence'] = 0.8

dict\_settings['scalar'] = uncertainty

dict\_settings['usemktcap'] = False

def save\_settings(caller=None):

temp\_sec, temp\_weight = loadtickerfrominput()

dict\_settings['security'] = temp\_sec

dict\_settings['weight'] = temp\_weight

dict\_settings['confidence'] = floattext\_confidence.value

dict\_settings['usemktcap'] = check\_usemktcap.value

f=open('settings\_bl.pckl','wb')

pickle.dump(dict\_settings, f)

f.close()

def loadtickerfrominput():

temp\_ticker = []

temp\_name = []

temp\_weight = []

dict\_missnametickers = OrderedDict()

flag\_missingname = False

for n in range(num\_avail\_ticker):

if bool(list\_sec\_input[n+1].children[0].value):

temp\_ticker.append(list\_sec\_input[n+1].children[0].value)

temp\_name.append(list\_sec\_input[n+1].children[1].value)

if list\_sec\_input[n+1].children[1].value.strip() == '':

dict\_missnametickers[list\_sec\_input[n+1].children[0].value] = n

flag\_missingname = True

temp\_weight.append(list\_sec\_input[n+1].children[2].value)

if flag\_missingname:

df\_name=bq\_ref\_data(dict\_missnametickers.keys(),{'name':bq.data.NAME()})

for index,row in df\_name.iterrows():

temp\_name[dict\_missnametickers[index]] = row['name']

temp\_sec=OrderedDict(zip(temp\_name,temp\_ticker))

return temp\_sec, temp\_weight

def bq\_ref\_data(security,datafields):

# Generate the request using the sercurity variable and data item

request = bql.Request(security, datafields)

response = bq.execute(request)

def merge(response):

return pd.concat([sir.df()[sir.name] for sir in response], axis=1)

result=merge(response)

return result

def bq\_series\_data(security,datafields):

request = bql.Request(security, datafields)

response = bq.execute(request)

result = response[0].df().reset\_index().pivot(index='DATE',columns='ID',values=response[0].name)[security]

return result

def \_port\_mean(weights, expected\_returns):

return((expected\_returns.T \* weights).sum())

def \_port\_var(weights, risk\_matrix):

return np.dot(np.dot(weights.T, risk\_matrix), weights)

def \_port\_mean\_var(weights, expected\_returns, risk\_matrix):

return \_port\_mean(weights, expected\_returns), \_port\_var(weights, risk\_matrix)

def \_find\_mean\_variance(weights, expected\_returns, covar, return\_target):

mean, var = \_port\_mean\_var(weights, expected\_returns, covar)

penalty = 10 \* abs(mean-return\_target)

return var + penalty

# Solve for optimal portfolio weights

def solve\_weights(R, C, rf, target\_r = None):

n = len(R)

W = np.ones([n])/n # Start optimization with equal weights

b\_ = [(0,1) for i in range(n)] # Bounds for decision variables

c\_ = ({'type':'eq', 'fun': lambda W: sum(W)-1. }) # Constraints - weights must sum to 1

r = np.sum(R\*W) if target\_r is None else target\_r

# 'target' return is the expected return on the market portfolio

optimized = scipy.optimize.minimize(\_find\_mean\_variance, W, (R, C, r), method='SLSQP', constraints=c\_, bounds=b\_)

if not optimized.success:

raise BaseException(optimized.message)

return optimized.x

def solve\_for\_frountier(R, C, rf):

frontier\_mean, frontier\_var , weights = [], [], []

for r in np.linspace(R.min(), R.max(), num=15):

weight = solve\_weights(R, C, rf, r)

frontier\_mean.append(r)

frontier\_var.append(\_port\_var(weight, C))

weights.append(weight)

weights = pd.DataFrame(weights)

weights.index.name = 'portolio'

frontier = pd.DataFrame([np.array(frontier\_mean), np.sqrt(frontier\_var)], index=['return', 'risk']).T

frontier.index.name = 'portfolio'

return frontier, weights

def solve\_intial\_opt\_weight():

global W\_opt, frontier, f\_weights, Pi, C, lmb, new\_mean, W, R, mean\_opt, var\_opt

security = dict\_settings['security']

univ = list(security.values())

datafields = OrderedDict()

datafields['return'] = bq.data.day\_to\_day\_total\_return(start='-5y',per='m')

day\_to\_day\_return=bq\_series\_data(univ,datafields)

R = day\_to\_day\_return.dropna().mean()\*12 #252 # R is the vector of expected returns

C = day\_to\_day\_return.cov() \*12 #252 # C is the covariance matrix

if dict\_settings['usemktcap']:

datafields = OrderedDict()

datafields['Mkt Cap'] = bq.data.cur\_mkt\_cap(currency='usd')

df\_mkt\_cap=bq\_ref\_data(univ,datafields)

W = np.array(df\_mkt\_cap/df\_mkt\_cap.sum()) # W is the market cap weight

else:

W = np.array(dict\_settings['weight']).reshape(len(R),1)

new\_mean = \_port\_mean(W.T[0],R)

new\_var = \_port\_var(W,C)

lmb = 0.5/np.sqrt(W.T.dot(C).dot(W))[0][0] # Compute implied risk adversion coefficient

Pi = np.dot(lmb \* C, W) # Compute equilibrium excess returns

frontier, f\_weights = solve\_for\_frountier(Pi+rf, C, rf)

frontier['sharpe']=frontier['return']/frontier['risk']

f\_weights.columns = R.keys()

# Solve for weights before incorporating views

W\_opt = np.array(f\_weights.iloc[frontier.loc[frontier['sharpe']==frontier['sharpe'].max()].index.values[0]])

mean\_opt, var\_opt = \_port\_mean\_var(W\_opt, Pi+rf, C) # calculate tangency portfolio

#return W\_opt, frontier, f\_weights, Pi, C

solve\_intial\_opt\_weight()

input\_header = HBox([Label(value='Ticker', layout=Layout(width='120px',height='22px')), Label(value='Name of Asset', layout=Layout(width='120px',height='22px')),

Label(value='Weight',layout=Layout(width='120px',height='22px'))])

list\_sec\_input = [input\_header]

lst\_name = list(dict\_settings['security'].keys())

lst\_ticker = list(dict\_settings['security'].values())

lst\_weight = dict\_settings['weight']

check\_usemktcap = Checkbox(description='Use Market Cap as Weight',value=dict\_settings['usemktcap'], layout=Layout(min\_width='15px'),style={'description\_width':'initial'})

label\_usemktcap = Label(value=' ',layout={'height':'22px'})

label\_usemktcap2 = Label(value='(not recommended when using ETF as proxy)',layout=Layout(min\_width='300px'))

port\_dict = {x['name']: x['id'].split(':')[2].replace('-','U') + '-' + x['id'].split(':')[3] for x in bqport.list\_portfolios()}

load\_button = Button(description='Load members')

portfolio\_dropdown = Dropdown(description='Portfolio:',options=sorted(set(port\_dict.keys())))

load\_members\_hbox = HBox([portfolio\_dropdown, load\_button])

for n in range(num\_avail\_ticker):

text\_name = Text(layout=Layout(width='120px'))

text\_ticker = Text(layout=Layout(width='120px'))

list\_sec\_input.append(HBox([text\_ticker, text\_name, FloatText(layout=Layout(width='120px'))]))

try:

if check\_usemktcap.value:

list\_sec\_input[n+1].children[2].disabled = True

list\_sec\_input[n+1].children[2].layout.visibility = 'hidden'

list\_sec\_input[n+1].children[0].value = lst\_ticker[n]

list\_sec\_input[n+1].children[1].value = lst\_name[n] if lst\_name[n] != '' else list\_sec\_input[n+1].children[0].data[0].split(":")[-1].strip()

list\_sec\_input[n+1].children[2].value = lst\_weight[n]

except:

pass

def updateinputboxes(obj=None):

lst\_name = list(dict\_settings['security'].keys())

lst\_ticker = list(dict\_settings['security'].values())

for n in range(num\_avail\_ticker):

if list\_sec\_input[n+1].children[0].value.strip() != '' and list\_sec\_input[n+1].children[1].value.strip() == '':

list\_sec\_input[n+1].children[1].value = lst\_name[n]

if check\_usemktcap.value:

list\_sec\_input[n+1].children[2].disabled = True

list\_sec\_input[n+1].children[2].layout.visibility = 'hidden'

else:

list\_sec\_input[n+1].children[2].disabled = False

list\_sec\_input[n+1].children[2].layout.visibility = 'visible'

check\_usemktcap.observe(updateinputboxes, 'value')

button\_applysettings=Button(description = 'Apply Settings')

def onclickapplysettings(obj=None):

save\_settings()

updateinputboxes()

solve\_intial\_opt\_weight()

updateviewcontrol()

updatecontrolinui()

run\_viewmodel({'new':0.})

button\_applysettings.on\_click(onclickapplysettings)

UI\_sec\_input = HBox([VBox(list\_sec\_input),VBox([load\_members\_hbox,label\_usemktcap,check\_usemktcap,label\_usemktcap2,button\_applysettings],layout={'margin':'0px 0px 0px 10px'})])

def on\_click\_load\_portfolio(obj=None):

global df\_portfolio\_weight

portfolio\_univ = bq.univ.members(port\_dict[portfolio\_dropdown.value],type='PORT')

id\_ = bq.data.id()

df\_portfolio\_weight = pd.concat([x.df() for x in bq.execute(bql.Request(portfolio\_univ, [bq.data.name(),id\_['Weights']/100]))],axis=1).reset\_index()

for x in range(1,num\_avail\_ticker+1):

if x - 1 < len(df\_portfolio\_weight):

list\_sec\_input[x].children[0].value = df\_portfolio\_weight.iloc[x-1,0]

list\_sec\_input[x].children[1].value = str(df\_portfolio\_weight.iloc[x-1,1])

list\_sec\_input[x].children[2].value = df\_portfolio\_weight.iloc[x-1,2]

else:

list\_sec\_input[x].children[0].value = ''

list\_sec\_input[x].children[1].value = ''

list\_sec\_input[x].children[2].value = 0

load\_button.on\_click(on\_click\_load\_portfolio)

def run\_viewmodel(change=None):

# VIEWS ON ASSET PERFORMANCE

# for troubleshoot

global sub\_a, sub\_b, sub\_c, sub\_d, tau, omega, P, Q, Pi\_new

list\_security=list(dict\_settings['security'].keys())

weights=OrderedDict()

P=np.identity(len(dict\_settings['security']))

if isinstance(change['new'],float):

Q=[]

for n in range(len(dict\_settings['security'])):

alpha = (list\_slider[n].value - Pi[n][0]) \* (floattext\_confidence.value)

Q.append(alpha + Pi[n][0])

for relative\_box in list\_relative\_controls:

sec1\_pos = relative\_box.children[0].value - 1

sec2\_pos = relative\_box.children[2].value - 1

if sec1\_pos >= 0 and sec2\_pos >= 0:

npselection = np.zeros(len(dict\_settings['security']))

npselection[sec1\_pos] = 1

npselection[sec2\_pos] = -1

P = np.array(pd.DataFrame(P).append(pd.DataFrame(npselection).T))

alpha = (relative\_box.children[-1].value - (Pi[sec1\_pos][0] - Pi[sec2\_pos][0])) \* (floattext\_confidence.value)

Q.append(alpha + (Pi[sec1\_pos][0] - Pi[sec2\_pos][0]))

Q=np.array([Q]).T

#tau = floattext\_uncertainty.value

tau = 1/(5\*12-len(list\_security)) #tau is a scalar indicating the uncertainty

omega = np.dot(np.dot(np.dot(tau, P), C), P.T)# omega represents uncertanity of views implied uncertainty from market parameters.

# Compute equilibrium excess returns taking into account views on assets

sub\_a = inv(np.dot(tau, C))

sub\_b = np.dot(np.dot(P.T, inv(omega)), P)

sub\_c = np.dot(inv(np.dot(tau, C)), Pi)

sub\_d = np.dot(np.dot(P.T, inv(omega)), Q)

Pi\_new = np.dot(inv(sub\_a + sub\_b), (sub\_c + sub\_d))

# Perform a mean-variance optimization taking into account views

new\_frontier, new\_f\_weights = solve\_for\_frountier(Pi\_new + rf, C, rf)

new\_frontier['sharpe']=new\_frontier['return']/new\_frontier['risk']

# Solve for weights before incorporating views

new\_weights = np.array(new\_f\_weights.iloc[new\_frontier.loc[new\_frontier['sharpe']==new\_frontier['sharpe'].max()].index.values[0]])

leverage = np.sum(abs(new\_weights))

weights['Opt Portfolio']=W\_opt[::-1]

weights['Opt Portfolio with View']=new\_weights[::-1]

output\_df = pd.DataFrame(weights, index=list\_security[::-1])

output\_df.to\_excel('output.xlsx')

mean, var = \_port\_mean\_var(new\_weights[::-1], Pi\_new + rf, C)

scatt.x = [np.sqrt(var\_opt)]

scatt.y = [mean\_opt]

scatt\_view.x = [np.sqrt(var)]

scatt\_view.y = [mean]

bar.x = list\_security[::-1]

bar.y = [weights[col] for col in weights]

line.x = frontier['risk']

line.y = [frontier['return'],new\_frontier['return']]

floattext\_confidence = FloatSlider(description='Confidence Level on Views', value=dict\_settings['confidence'],style={'description\_width':'initial'}, readout\_format='.2%', max=1, min=0,

layout={'margin':'20px 0px 0px 0px'}, step=0.5/100)

floattext\_confidence.observe(run\_viewmodel)

sv = pd.Series(np.sqrt(np.diag(Pi.T.dot(C.dot(Pi))).astype(float)), index=C.index)

def updateviewcontrol():

global UI\_viewcontrol, list\_slider, list\_relative\_controls, floattext\_uncertainty

list\_slider=[]

list\_security=list(dict\_settings['security'].keys())

for n in range(len(dict\_settings['security'])):

temp\_slider=FloatSlider(value=Pi[n], description=list\_security[n], max=0.2, min=-0.2, readout\_format='.2%', step=0.2/100,style={'description\_width':'100PX'})

temp\_slider.observe(run\_viewmodel)

list\_slider.append(temp\_slider)

list\_relative\_controls=[]

sec\_dropdown\_options = OrderedDict(zip(['None']+list(dict\_settings['security'].keys()),range(len(dict\_settings['security'])+1)))

for n in range(3):

dropdown1 = Dropdown(options=sec\_dropdown\_options,layout={'width':'100px'})

dropdown2 = Dropdown(options=sec\_dropdown\_options,layout={'width':'100px'})

label = Label(value='over',layout={'width':'30px'})

float\_value = FloatSlider(description='by', value=0, readout\_format='.2%', max=0.2, min=0,

style={'description\_width':'initial'}, step=0.1/100,layout={'width':'200px'})

float\_value.observe(run\_viewmodel)

relative\_box = HBox([dropdown1,label,dropdown2,float\_value])

list\_relative\_controls.append(relative\_box)

header\_abs\_html = HTML('<p style="color: white;">{}</p>'.format('Absolute Views'))

header\_rel\_html = HTML('<p style="color: white;">{}</p>'.format('Relative Views'), layout={'margin':'20px 0px 0px 0px'})

UI\_viewcontrol = [header\_abs\_html, VBox(list\_slider),header\_rel\_html, VBox(list\_relative\_controls), VBox([floattext\_confidence])]

def updatecontrolinui():

UI\_model.children[0].children = UI\_viewcontrol

updateviewcontrol()

# Build bar charts

x\_ord = bqp.OrdinalScale()

y\_sc = bqp.LinearScale()

bar = bqp.Bars(x=[],

y=[],

scales={'x': x\_ord, 'y': y\_sc},

orientation="horizontal", display\_legend=True, labels=['Mkt Efficient Portfolio','Efficient Portfolio with Views'],

colors=['#1B84ED','#F39F41'],

type='grouped')

#bar.type='grouped'

bar.tooltip = bqp.Tooltip(fields=['y'], labels=['Weights'], formats=['.3f'])

ax\_x = bqp.Axis(scale=x\_ord, orientation="vertical")

ax\_y = bqp.Axis(scale=y\_sc, label='Weight')

fig\_bar = bqp.Figure(marks=[bar], axes=[ax\_x, ax\_y], padding\_x=0.025, padding\_y=0.025,

layout=Layout(width='800px'), legend\_location='top-right',

fig\_margin={'top':20, 'bottom':30, 'left':80, 'right':20})

x\_lin = bqp.LinearScale()

y\_lin = bqp.LinearScale()

x\_ax = bqp.Axis(label='risk', scale=x\_lin, grid\_lines='solid')

x\_ay = bqp.Axis(label='return', scale=y\_lin, orientation='vertical', grid\_lines='solid')

def\_tt = bqp.Tooltip(fields=['x', 'y'], formats=['.3f', '.3f'])

scatt = bqp.Scatter(x=[],y=[], scales={'x': x\_lin, 'y': y\_lin}, tooltip=def\_tt,

display\_legend=True, labels=['Efficient Portfolio'], colors=['#1B84ED'])

scatt\_view = bqp.Scatter(x=[],y=[], scales={'x': x\_lin, 'y': y\_lin}, tooltip=def\_tt,

display\_legend=True, labels=['Efficient Portfolio with Views Portfolio'], colors=['#F39F41'])

line = bqp.Lines(x=[], y=[], scales={'x': x\_lin, 'y': y\_lin}, display\_legend=True, labels=['Mkt Efficient Portfolio','Efficient Portfolio with Views'], colors=['#1B84ED','#F39F41'])

fig\_line = bqp.Figure(marks=[line], axes=[x\_ax, x\_ay],

legend\_location='top-left', layout=Layout(width='800px'),

fig\_margin={'top':20, 'bottom':30, 'left':80, 'right':20})

run\_viewmodel({'new':0.})

UI\_model=HBox([VBox(UI\_viewcontrol,layout=Layout(width='450px')),VBox([fig\_bar,fig\_line])])

tab = Tab()

tab.children = [UI\_model, UI\_sec\_input]

tab.set\_title(0, 'B-L Model')

tab.set\_title(1, 'Settings')

#tab.set\_title(2, 'Reference Data')

def updatedseclist(obj=None):

if obj['old'] == 1:

save\_settings()

solve\_intial\_opt\_weight()

run\_viewmodel({'new':0.})

import bqcde

from datetime import date

def upload\_to\_cde(obj):

obj.description = 'Uploading...'

#lmb\_2 = 0.5/np.sqrt(np.asarray([W\_opt]).dot(C).dot(np.asarray([W\_opt]).T))[0][0]

#Pi\_2 = np.dot(lmb\_2 \* C, np.asarray([W\_opt]).T)

df = pd.DataFrame({'ID':list(dict\_settings['security'].values()),'Equil Return': [x[0] for x in Pi], 'BL Return':[x[0] for x in Pi\_new] })

as\_of\_date = date.today()

upload\_dict = {'Equil Return':'UD\_EQUIL\_RETURN','BL Return':'UD\_BL\_RETURN'}

#pd.DataFrame({'ID':list(dict\_settings['security'].keys()),'Equil Return': [x[0] for x in Pi]}).set\_index('ID')

try:

df = df.rename(columns=upload\_dict)

list\_to\_upload = list(upload\_dict.values())

fs = bqcde.get\_fields(mnemonics=list\_to\_upload)

df['AS\_OF\_DATE'] = as\_of\_date

df['AS\_OF\_DATE'] = pd.to\_datetime(df['AS\_OF\_DATE'])

bqcde.write(fields=fs, dataframe=df.fillna("N/A"))

except Exception as e:

print(e)

obj.description = 'Upload to CDE'

#upload\_to\_cde()

button = Button(description='Upload to CDE')

button.on\_click(upload\_to\_cde)

# Black-Litterman Asset-allocation Model

---

The Black-Litterman asset-allocation model is a Bayesian approach to combine the subjective views of an investor with the market equiilibrium returns to form a new, mixed estimate of expected returns.

preload\_box.children = []

VBox([button,tab])

[Open Weight](output.xlsx)

for slider in list\_slider:

print(slider.description, ": ", slider.value)