```python
### Portfolio Optiimization
###
#
# Finds an optimal allocation of stocks in a portfolio,
# satisfying a minimum expected return.
# The problem is posed as a Quadratic Program, and solved
# using the cvxopt library.
# Uses actual past stock data, obtained using the stocks module.
import math
import numpy as np
import pandas as pd
import datetime
import cvxopt
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt
##########################
import warnings
warnings.filterwarnings('ignore')
warnings.warn('DelftStack')
warnings.warn('Do not show this message')
#####################
solvers.options['show_progress'] = False        # !!!

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

#from cvxopt import solvers
#import stocks
import numpy as np
import pandas as pd
import datetime

import pandas as pd
import statsmodels.formula.api as smf
import statsmodels.api as sm

# c = cvxopt.matrix([0, -1], tc='d')
# print('c: ', c)
# c = numpy.matrix(c)
# print('c: ', c)
#
# c = cvxopt.matrix([0, -1])
# print('c: ', c)
# G = cvxopt.matrix([[-1, 1], [3, 2], [2, 3], [-1, 0], [0, -1]], tc='d')
# print('G: ', G)
##################
xDir = r'D:\\Users\\ggu\\Documents\\GU\\MultiRiskFactorModel\\DATA\\'
xSPXT = pd.read_csv(xDir + 'xSPXT.txt')
xSPXT['DATE'] = pd.to_datetime(xSPXT['DATE'], format='%m/%d/%Y')
xBondTR = pd.read_csv(xDir + 'xBondTR.txt')
xBondTR['DATE'] = pd.to_datetime(xBondTR['DATE'], format='%m/%d/%Y')
#xBondTR.rename(columns={'LBUSTRUU': 'BondTR'},inplace=True)
xAAPL = pd.read_csv(xDir + 'xAAPL.txt')
xAAPL['DATE'] = pd.to_datetime(xAAPL['DATE'], format='%m/%d/%Y')
xAGG = pd.read_csv(xDir + 'xAGG.txt')
xAGG['DATE'] = pd.to_datetime(xAGG['DATE'], format='%m/%d/%Y')
xCCY = pd.read_csv(xDir + 'xCCY.txt')
xCCY['DATE'] = pd.to_datetime(xCCY['DATE'], format='%m/%d/%Y')
xCOMM = pd.read_csv(xDir + 'xCOMM.txt')
```

```
xCOMM['DATE'] = pd.to_datetime(xCOMM['DATE'], format='%m/%d/%Y')
xCREDIT = pd.read_csv(xDir + 'xCREDIT.txt')
xCREDIT['DATE'] = pd.to_datetime(xCREDIT['DATE'], format='%m/%d/%Y')
xFTLS = pd.read_csv(xDir + 'xFTLS.txt')
xFTLS['DATE'] = pd.to_datetime(xFTLS['DATE'], format='%m/%d/%Y')
xHFRIEMNI = pd.read_csv(xDir + 'xHFRIEMNI.txt')
xHFRIEMNI['DATE'] = pd.to_datetime(xHFRIEMNI['DATE'], format='%m/%d/%Y')
xPRBAX = pd.read_csv(xDir + 'xPRBAX.txt')
xPRBAX['DATE'] = pd.to_datetime(xPRBAX['DATE'], format='%m/%d/%Y')
xPRWAX = pd.read_csv(xDir + 'xPRWAX.txt')
xPRWAX['DATE'] = pd.to_datetime(xPRWAX['DATE'], format='%m/%d/%Y')
xSPLPEQTY = pd.read_csv(xDir + 'xSPLPEQTY.txt')
xSPLPEQTY['DATE'] = pd.to_datetime(xSPLPEQTY['DATE'], format='%m/%d/%Y')
xSPX = pd.read_csv(xDir + 'xSPX.txt')
xSPX['DATE'] = pd.to_datetime(xSPX['DATE'], format='%m/%d/%Y')
xSPY = pd.read_csv(xDir + 'xSPY.txt')
xSPY['DATE'] = pd.to_datetime(xSPY['DATE'], format='%m/%d/%Y')
xTSLA = pd.read_csv(xDir + 'xTSLA.txt')
xTSLA['DATE'] = pd.to_datetime(xTSLA['DATE'], format='%m/%d/%Y')
xUS3M = pd.read_csv(xDir + 'xUS3M.txt')
xUS3M['DATE'] = pd.to_datetime(xUS3M['DATE'], format='%m/%d/%Y')
xUS10Y = pd.read_csv(xDir + 'xUS10Y.txt')
xUS10Y['DATE'] = pd.to_datetime(xUS10Y['DATE'], format='%m/%d/%Y')
xHYG = pd.read_csv(xDir + 'xHYG.txt')
xHYG['DATE'] = pd.to_datetime(xHYG['DATE'], format='%m/%d/%Y')
xCPI = pd.read_csv(xDir + 'xCPI.txt')
xCPI['DATE'] = pd.to_datetime(xCPI['DATE'], format='%m/%d/%Y')
xHYTR = pd.read_csv(xDir + 'xLF98TRUU.txt')
xHYTR['DATE'] = pd.to_datetime(xHYTR['DATE'], format='%m/%d/%Y')
xTIPS = pd.read_csv(xDir + 'xLBUTTRUU.txt')
xTIPS['DATE'] = pd.to_datetime(xTIPS['DATE'], format='%m/%d/%Y')
xGMWAX = pd.read_csv(xDir + 'xGMWAX.txt')
xGMWAX['DATE'] = pd.to_datetime(xGMWAX['DATE'], format='%m/%d/%Y')
xCashConst = pd.read_csv(xDir + 'xCashConst.txt')
xCashConst['DATE'] = pd.to_datetime(xCashConst['DATE'], format='%m/%d/%Y')
xS5INFT = pd.read_csv(xDir + 'xS5INFT.txt')
xS5INFT['DATE'] = pd.to_datetime(xS5INFT['DATE'], format='%m/%d/%Y')
x7030TR = pd.read_csv(xDir + 'x7030TR.txt')
x7030TR['DATE'] = pd.to_datetime(x7030TR['DATE'], format='%m/%d/%Y')
xUSCredit = pd.read_csv(xDir + 'xLUCRTRUU.txt')
xUSCredit['DATE'] = pd.to_datetime(xUSCredit['DATE'], format='%m/%d/%Y')
xSHY = pd.read_csv(xDir + 'xSHY.txt')
xSHY['DATE'] = pd.to_datetime(xSHY['DATE'], format='%m/%d/%Y')
xTIP = pd.read_csv(xDir + 'xTIP.txt')
xTIP['DATE'] = pd.to_datetime(xTIP['DATE'], format='%m/%d/%Y')
xAMZN = pd.read_csv(xDir + 'xAMZN.txt')
xAMZN['DATE'] = pd.to_datetime(xAMZN['DATE'], format='%m/%d/%Y')
xFB = pd.read_csv(xDir + 'xFB.txt')
xFB['DATE'] = pd.to_datetime(xFB['DATE'], format='%m/%d/%Y')
xVIAC = pd.read_csv(xDir + 'xVIAC.txt')
xVIAC['DATE'] = pd.to_datetime(xVIAC['DATE'], format='%m/%d/%Y')
xGOOG = pd.read_csv(xDir + 'xGOOG.txt')
xGOOG['DATE'] = pd.to_datetime(xGOOG['DATE'], format='%m/%d/%Y')
xLQD = pd.read_csv(xDir + 'xLQD.txt')
xLQD['DATE'] = pd.to_datetime(xLQD['DATE'], format='%m/%d/%Y')
xMDY = pd.read_csv(xDir + 'xMDY.txt')
xMDY['DATE'] = pd.to_datetime(xMDY['DATE'], format='%m/%d/%Y')
xMSFT = pd.read_csv(xDir + 'xMSFT.txt')
xMSFT['DATE'] = pd.to_datetime(xMSFT['DATE'], format='%m/%d/%Y')
xRLV = pd.read_csv(xDir + 'xRLV.txt')
```

```
xRLV['DATE'] = pd.to_datetime(xRLV['DATE'], format='%m/%d/%Y')
xRLG = pd.read_csv(xDir + 'xRLG.txt')
xRLG['DATE'] = pd.to_datetime(xRLG['DATE'], format='%m/%d/%Y')
xRIY = pd.read_csv(xDir + 'xRIY.txt')
xRIY['DATE'] = pd.to_datetime(xRIY['DATE'], format='%m/%d/%Y')
xRMV = pd.read_csv(xDir + 'xRMV.txt')
xRMV['DATE'] = pd.to_datetime(xRMV['DATE'], format='%m/%d/%Y')
xRMC = pd.read_csv(xDir + 'xRMC.txt')
xRMC['DATE'] = pd.to_datetime(xRMC['DATE'], format='%m/%d/%Y')
xRDG = pd.read_csv(xDir + 'xRDG.txt')
xRDG['DATE'] = pd.to_datetime(xRDG['DATE'], format='%m/%d/%Y')
xRUJ = pd.read_csv(xDir + 'xRUJ.txt')
xRUJ['DATE'] = pd.to_datetime(xRUJ['DATE'], format='%m/%d/%Y')
xRTY = pd.read_csv(xDir + 'xRTY.txt')
xRTY['DATE'] = pd.to_datetime(xRTY['DATE'], format='%m/%d/%Y')
xRUO = pd.read_csv(xDir + 'xRUO.txt')
xRUO['DATE'] = pd.to_datetime(xRUO['DATE'], format='%m/%d/%Y')

##########################################
xDF = xSPX.copy()
xDF = pd.merge(xDF, xSPXT, on=['DATE'], how='left')
xDF = pd.merge(xDF, xBondTR, on=['DATE'], how='left')
xDF = pd.merge(xDF, xAAPL, on=['DATE'], how='left')
xDF = pd.merge(xDF, xAGG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xCCY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xCOMM, on=['DATE'], how='left')
xDF = pd.merge(xDF, xCREDIT, on=['DATE'], how='left')
xDF = pd.merge(xDF, xFTLS, on=['DATE'], how='left')
###xDF = pd.merge(xDF, xHFRIEMNI, on=['DATE'], how='left')
xDF = pd.merge(xDF, xPRBAX, on=['DATE'], how='left')
xDF = pd.merge(xDF, xPRWAX, on=['DATE'], how='left')
xDF = pd.merge(xDF, xSPLPEQTY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xSPY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xTSLA, on=['DATE'], how='left')
xDF = pd.merge(xDF, xUS3M, on=['DATE'], how='left')
xDF = pd.merge(xDF, xUS10Y, on=['DATE'], how='left')
xDF = pd.merge(xDF, xHYG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xHYTR, on=['DATE'], how='left')
xDF = pd.merge(xDF, xTIPS, on=['DATE'], how='left')
xDF = pd.merge(xDF, xGMWAX, on=['DATE'], how='left')
xDF = pd.merge(xDF, xCashConst, on=['DATE'], how='left')
xDF = pd.merge(xDF, xS5INFT, on=['DATE'], how='left')
xDF = pd.merge(xDF, x7030TR, on=['DATE'], how='left')
xDF = pd.merge(xDF, xUSCredit, on=['DATE'], how='left')
xDF = pd.merge(xDF, xSHY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xTIP, on=['DATE'], how='left')
xDF = pd.merge(xDF, xAMZN, on=['DATE'], how='left')
xDF = pd.merge(xDF, xFB, on=['DATE'], how='left')
xDF = pd.merge(xDF, xVIAC, on=['DATE'], how='left')
xDF = pd.merge(xDF, xGOOG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xLQD, on=['DATE'], how='left')
xDF = pd.merge(xDF, xMDY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xMSFT, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRLV, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRLG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRIY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRMV, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRMC, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRDG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRUJ, on=['DATE'], how='left')
```

```
xDF = pd.merge(xDF, xRTY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRUO, on=['DATE'], how='left')

######################################################################
# xEndDate_0 = pd.to_datetime('10/1/2018')
# xDF = xDF.loc[xDF['DATE']<xEndDate_0]
# ################ forward fill the missing equity trading dates ############
xDF['BondTR'].fillna(method='ffill', inplace=True)
xDF['HYTR'].fillna(method='ffill', inplace=True)
xDF['TIPS'].fillna(method='ffill', inplace=True)
xDF['LQD'].fillna(method='ffill', inplace=True)
xDF['SPLPEQTY'].fillna(method='ffill', inplace=True)
##################################
###############Calculating SI returns here ##################################
for k in range(1,4):
    if k==1:
        # 2 year, buffer -10%, x1.5, cap = 21%, hard buffer note!
        xCap = 0.21 #0.21 #0.21  #1000 #0.21   #1000  #0.21
        xBuffer = -0.10000  ####-0.10  #-0.25   #-0.30   #-0.25
        xTerm = 2  #2  #4  #6   #4 #2  #3 # years
        xAmount = 100
        xLever = 1.500  #1.5  #1.15
        xBufferType = "H"  #"T"  # "H" for regular Buffer; "G" for Geared Buffer (or Barrier); 
"T" for Trigger Buffer!
    elif k == 2:
        # 4 years, buffer -25%, no leverage and no cap, barrier buffer note!
        xCap = 10000
        xBuffer = -0.250000
        xTerm = 4
        xAmount = 100
        xLever = 1.00
        xBufferType = "T"
    elif k==3:
        # 6 years, buffer -30%, x1.15 leverage and no cap, barrier buffer note!
        xCap = 10000  # 0.21 #0.21  #1000 #0.21   #1000  #0.21
        xBuffer = -0.300000  ####-0.10  #-0.25   #-0.30   #-0.25
        xTerm = 6
        xAmount = 100
        xLever = 1.1500  # 1.5  #1.15
        xBufferType = "T"  # "T"  # "H" for regular Buffer; "G" for Geared Buffer (or 
Barrier); "T" for Trigger Buffer!

    xDF['SPX_rtn_term'] = xDF['SPX'].pct_change(xTerm*252)
    xDF.loc[xDF['SPX_rtn_term'] > 0, 'SI' + (str)(xTerm) + '_rtn_term'] = xDF['SPX_rtn_term'] 
* xLever
    xDF.loc[xDF['SPX_rtn_term']* xLever > xCap, 'SI' + (str)(xTerm) + '_rtn_term'] = xCap
    xDF.loc[(xDF['SPX_rtn_term']<=0) & (xDF['SPX_rtn_term']>=xBuffer), 
'SI'+(str)(xTerm)+'_rtn_term'] = 0
    if (xBufferType=='H'):
        xDF.loc[(xDF['SPX_rtn_term']<xBuffer),'SI'+(str)(xTerm)+'_rtn_term'] = 
xDF['SPX_rtn_term'] - xBuffer
    elif (xBufferType=='T'):
        xDF.loc[(xDF['SPX_rtn_term']<xBuffer),'SI'+(str)(xTerm)+'_rtn_term'] = 
xDF['SPX_rtn_term']
    elif (xBufferType=='G'):
        xK = 1 / (1+xBuffer)
        xDF.loc[(xDF['SPX_rtn_term']<xBuffer),'SI'+(str)(xTerm)+'_rtn_term'] = xK * 
(xDF['SPX_rtn_term'] - xBuffer)

    xDF['SI'+(str)(xTerm)+'_pct_ch_Y'] = (1+xDF['SI'+(str)(xTerm)+'_rtn_term'])**(1/xTerm) - 1
```

```python
    xDF['SI'+(str)(xTerm)+'_pct_ch_Q'] = (1+xDF['SI'+(str)(xTerm)+'_rtn_term'])**(1/(xTerm*4))
- 1
    xDF['SI'+(str)(xTerm)+'_pct_ch_M'] =
(1+xDF['SI'+(str)(xTerm)+'_rtn_term'])**(1/(xTerm*12)) - 1
    xDF['SI'+(str)(xTerm)+'_pct_ch_D'] =
(1+xDF['SI'+(str)(xTerm)+'_rtn_term'])**(1/(xTerm*252)) - 1

###########################
#########################################
xSPX_2 = xSPX.copy()
xSPX_2['month']=xSPX_2.DATE.dt.month
xSPX_2['year']=xSPX_2.DATE.dt.year
xSPX_2['diff']=xSPX_2.month.diff(-1)
xSPX_2['month-year']=xSPX_2.month.astype('string')+'-'+xSPX_2.year.astype('string')
xSPX_2 = xSPX_2.loc[xSPX_2['diff']!=0]
##### equity market neutral index (monthly) ######
xHFRIEMNI['month']=xHFRIEMNI.DATE.dt.month
xHFRIEMNI['year']=xHFRIEMNI.DATE.dt.year
###xHFRIEMNI['diff']=xHFRIEMNI.month.diff(-1)
xHFRIEMNI['month-year']=xHFRIEMNI.month.astype('string')+'-'+xHFRIEMNI.year.astype('string')
xHFRIEMNI.rename(columns={'DATE': 'DATE0'},inplace=True)
#xHFRIEMNI['LS_1y_pct_ch']=xHFRIEMNI['HFRIEMNI'].pct_change(12)
xHFRIEMNI = pd.merge(xHFRIEMNI, xSPX_2[['DATE','month-year']], on=['month-year'],how='left')
xDF = pd.merge(xDF, xHFRIEMNI[['DATE','HFRIEMNI']], on=['DATE'], how='left')
##### CPI index (monthly) ######
xCPI['month']=xCPI.DATE.dt.month
xCPI['year']=xCPI.DATE.dt.year
xCPI['month-year']=xCPI.month.astype('string')+'-'+xCPI.year.astype('string')
xCPI.rename(columns={'DATE': 'DATE0'},inplace=True)
#xCPI['LS_1y_pct_ch']=xCPI['HFRIEMNI'].pct_change(12)
xCPI = pd.merge(xCPI, xSPX_2[['DATE','month-year']], on=['month-year'],how='left')
xDF = pd.merge(xDF, xCPI[['DATE','CPI']], on=['DATE'], how='left')
###########
xDF = pd.merge(xDF, xSPX_2[['DATE','month-year','diff']], on=['DATE'], how='left')
xDF_M = xDF.loc[xDF['diff']!=0]
xDF_M = xDF_M.loc[xDF_M['diff'].notnull()]
xDF_M.reset_index(drop=True, inplace=True)
xDF_M['RealBondTR'] = xDF_M['BondTR']/xDF_M['CPI']
xDF_M['quarter'] = xDF_M['DATE'].dt.quarter
xDF_M['diff_Q']=xDF_M.quarter.diff(-1)
######################################################

#########################################
xDF['SPXT_pct_ch_D']=xDF['SPXT'].pct_change()
xDF['BondTR_pct_ch_D']=xDF['BondTR'].pct_change()
xDF['AAPL_pct_ch_D']=xDF['AAPL'].pct_change()
xDF['AGG_pct_ch_D']=xDF['AGG'].pct_change()
xDF['CCY_pct_ch_D']=xDF['CCY'].pct_change()
xDF['COMM_pct_ch_D']=xDF['COMM'].pct_change()
xDF['CREDIT_pct_ch_D']=xDF['CREDIT'].pct_change()
xDF['FTLS_pct_ch_D']=xDF['FTLS'].pct_change()
xDF['HFRIEMNI_pct_ch_D']=xDF['HFRIEMNI'].pct_change()
xDF['PRBAX_pct_ch_D']=xDF['PRBAX'].pct_change()
xDF['PRWAX_pct_ch_D']=xDF['PRWAX'].pct_change()
xDF['SPLPEQTY_pct_ch_D']=xDF['SPLPEQTY'].pct_change()
xDF['SPX_pct_ch_D']=xDF['SPX'].pct_change()
xDF['SPY_pct_ch_D']=xDF['SPY'].pct_change()
xDF['TSLA_pct_ch_D']=xDF['TSLA'].pct_change()
xDF['US3M_pct_ch_D']=xDF['US3M'].pct_change()
xDF['US10Y_pct_ch_D']=xDF['US10Y'].pct_change()
```

```
xDF['HYG_pct_ch_D']=xDF['HYG'].pct_change()
xDF['HYTR_pct_ch_D']=xDF['HYTR'].pct_change()
xDF['TIPS_pct_ch_D']=xDF['TIPS'].pct_change()
xDF['GMWAX_pct_ch_D']=xDF['GMWAX'].pct_change()
xDF['CashConst_pct_ch_D']=xDF['CashConst'].pct_change()
xDF['S5INFT_pct_ch_D']=xDF['S5INFT'].pct_change()
xDF['7030TR_pct_ch_D']=xDF['7030TR'].pct_change()
xDF['USCredit_pct_ch_D']=xDF['USCredit'].pct_change()
xDF['SHY_pct_ch_D']=xDF['SHY'].pct_change()
xDF['TIP_pct_ch_D']=xDF['TIP'].pct_change()
xDF['AMZN_pct_ch_D']=xDF['AMZN'].pct_change()
xDF['FB_pct_ch_D']=xDF['FB'].pct_change()
xDF['VIAC_pct_ch_D']=xDF['VIAC'].pct_change()
xDF['GOOG_pct_ch_D']=xDF['GOOG'].pct_change()
xDF['LQD_pct_ch_D']=xDF['LQD'].pct_change()
xDF['MDY_pct_ch_D']=xDF['MDY'].pct_change()
xDF['MSFT_pct_ch_D']=xDF['MSFT'].pct_change()
xDF['RLV_pct_ch_D']=xDF['RLV'].pct_change()
xDF['RLG_pct_ch_D']=xDF['RLG'].pct_change()
xDF['RIY_pct_ch_D']=xDF['RIY'].pct_change()
xDF['RMV_pct_ch_D']=xDF['RMV'].pct_change()
xDF['RMC_pct_ch_D']=xDF['RMC'].pct_change()
xDF['RDG_pct_ch_D']=xDF['RDG'].pct_change()
xDF['RUJ_pct_ch_D']=xDF['RUJ'].pct_change()
xDF['RTY_pct_ch_D']=xDF['RTY'].pct_change()
xDF['RUO_pct_ch_D']=xDF['RUO'].pct_change()

############### new portfolio #########
################
xY_col = 'SPLPEQTY'
xY_col = 'FTLS'
xY_col = 'CashConst'
xY_col = 'PRWAX'
xY_col = 'GMWAX'
xY_col = 'PRBAX'
xY_col = 'SI'
xY_col = '7030TR'

xY_col = 'SPY'
xY_col = 'SHY'
xY_col = 'TIP'
xY_col = 'AGG'
xY_col = 'HYG'
xY_col = 'TSLA'
xY_col = 'AAPL'

xCoef_table = pd.DataFrame()
xRiskExp_Current = pd.DataFrame()
xRiskConcentration_Current = pd.DataFrame()
##################
xW1=0.75
xW2=0.25
xW3=0.0  #0.10
#xW4=0.15
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['BondTR_pct_ch_D']
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['TIPS_pct_ch_D']
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['BondTR_pct_ch_D']+
xW3*xDF['SI2_pct_ch_D']
xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['BondTR_pct_ch_D']+
xW3*xDF['SI2_pct_ch_D']
```

```
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['SPXT_pct_ch_D']+
xW3*xDF['SI2_pct_ch_D']
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['SI2_pct_ch_D']+
xW3*xDF['TIPS_pct_ch_D']+ xW4*xDF['HYTR_pct_ch_D']

xDF['NewPort'] = (1+xDF['NewPort_pct_ch_D']).cumprod()

############### Yearly ########################
xDF['SPXT_pct_ch_Y']=xDF['SPXT'].pct_change(252)
xDF['BondTR_pct_ch_Y']=xDF['BondTR'].pct_change(252)
xDF['AAPL_pct_ch_Y']=xDF['AAPL'].pct_change(252)
xDF['AGG_pct_ch_Y']=xDF['AGG'].pct_change(252)
xDF['CCY_pct_ch_Y']=xDF['CCY'].pct_change(252)
xDF['COMM_pct_ch_Y']=xDF['COMM'].pct_change(252)
xDF['CREDIT_pct_ch_Y']=xDF['CREDIT'].pct_change(252)
xDF['FTLS_pct_ch_Y']=xDF['FTLS'].pct_change(252)
xDF['HFRIEMNI_pct_ch_Y']=xDF['HFRIEMNI'].pct_change(252)   #need to calculate YoY from Monthly
data pct_change(12)!!!
xDF['PRBAX_pct_ch_Y']=xDF['PRBAX'].pct_change(252)
xDF['PRWAX_pct_ch_Y']=xDF['PRWAX'].pct_change(252)
xDF['SPLPEQTY_pct_ch_Y']=xDF['SPLPEQTY'].pct_change(252)
xDF['SPX_pct_ch_Y']=xDF['SPX'].pct_change(252)
xDF['SPY_pct_ch_Y']=xDF['SPY'].pct_change(252)
xDF['TSLA_pct_ch_Y']=xDF['TSLA'].pct_change(252)
xDF['US3M_pct_ch_Y']=xDF['US3M'].pct_change(252)
xDF['US10Y_pct_ch_Y']=xDF['US10Y'].pct_change(252)
xDF['HYG_pct_ch_Y']=xDF['HYG'].pct_change(252)
xDF['HYTR_pct_ch_Y']=xDF['HYTR'].pct_change(252)
xDF['TIPS_pct_ch_Y']=xDF['TIPS'].pct_change(252)
xDF['GMWAX_pct_ch_Y']=xDF['GMWAX'].pct_change(252)
xDF['CashConst_pct_ch_Y']=xDF['CashConst'].pct_change(252)
xDF['S5INFT_pct_ch_Y']=xDF['S5INFT'].pct_change(252)
xDF['7030TR_pct_ch_Y']=xDF['7030TR'].pct_change(252)
xDF['USCredit_pct_ch_Y']=xDF['USCredit'].pct_change(252)
xDF['SHY_pct_ch_Y']=xDF['SHY'].pct_change(252)
xDF['TIP_pct_ch_Y']=xDF['TIP'].pct_change(252)
xDF['AMZN_pct_ch_Y']=xDF['AMZN'].pct_change(252)
xDF['FB_pct_ch_Y']=xDF['FB'].pct_change(252)
xDF['VIAC_pct_ch_Y']=xDF['VIAC'].pct_change(252)
xDF['GOOG_pct_ch_Y']=xDF['GOOG'].pct_change(252)
xDF['LQD_pct_ch_Y']=xDF['LQD'].pct_change(252)
xDF['MDY_pct_ch_Y']=xDF['MDY'].pct_change(252)
xDF['MSFT_pct_ch_Y']=xDF['MSFT'].pct_change(252)
xDF['RLV_pct_ch_Y']=xDF['RLV'].pct_change(252)
xDF['RLG_pct_ch_Y']=xDF['RLG'].pct_change(252)
xDF['RIY_pct_ch_Y']=xDF['RIY'].pct_change(252)
xDF['RMV_pct_ch_Y']=xDF['RMV'].pct_change(252)
xDF['RMC_pct_ch_Y']=xDF['RMC'].pct_change(252)
xDF['RDG_pct_ch_Y']=xDF['RDG'].pct_change(252)
xDF['RUJ_pct_ch_Y']=xDF['RUJ'].pct_change(252)
xDF['RTY_pct_ch_Y']=xDF['RTY'].pct_change(252)
xDF['RUO_pct_ch_Y']=xDF['RUO'].pct_change(252)
############### overwrite to create the EXACT 70/30 returns #############
xDF['7030TR_pct_ch_Y']=0.7*xDF['SPXT_pct_ch_Y']+0.3*xDF['BondTR_pct_ch_Y']
xDF['3070TR_pct_ch_Y']=0.3*xDF['SPXT_pct_ch_Y']+0.7*xDF['BondTR_pct_ch_Y']
xDF['30AAPL30MSFT20AMZN20GOOGTR_pct_ch_Y'] = 0.3*xDF['AAPL_pct_ch_Y'] +
0.3*xDF['MSFT_pct_ch_Y'] +0.2*xDF['AMZN_pct_ch_Y'] +0.2*xDF['GOOG_pct_ch_Y']
xDF['30SPY30MDY20AGG20LQDTR_pct_ch_Y'] = 0.3*xDF['SPY_pct_ch_Y'] + 0.3*xDF['MDY_pct_ch_Y']
+0.2*xDF['AGG_pct_ch_Y'] +0.2*xDF['LQD_pct_ch_Y']
#xDF['85AAPL15SHY_pct_ch_Y'] =
```

```
0.70*xDF['AAPL_pct_ch_Y']+0.15*xDF['SHY_pct_ch_Y']+0.15*xDF['SI4_pct_ch_Y']
########################################################################
xDF['NewPort_pct_ch_Y']=xDF['NewPort'].pct_change(252)

xDF['Inflation_pct_ch_Y'] = xDF['BondTR_pct_ch_Y'] - xDF['TIPS_pct_ch_Y']
xDF['RealBondTR_pct_ch_Y'] = xDF['BondTR_pct_ch_Y'] - xDF['TIPS_pct_ch_Y']

# xDF['NewPort_pct_ch_Y']=xW1*xDF['SPY_pct_ch_Y'] + xW2*xDF['SI2_pct_ch_Y']
xDF['NewPort_pct_ch_Y']=xW1*xDF['3070TR_pct_ch_Y'] + xW2*xDF['SI2_pct_ch_Y']
#xDF['NewPort_pct_ch_Y']=xW1*xDF['SPY_pct_ch_Y'] + xW2*xDF['BondTR_pct_ch_Y']+
xW3*xDF['SI2_pct_ch_Y']  # case #1
#xDF['NewPort_pct_ch_Y']=xW1*xDF[xY_col+'_pct_ch_Y'] + xW2*xDF['BondTR_pct_ch_Y']+
xW3*xDF['SI2_pct_ch_Y']  # case #1
#xDF['NewPort_pct_ch_Y']=xW1*xDF['AGG_pct_ch_Y'] + xW2*xDF['SPXT_pct_ch_Y']+
xW3*xDF['SI2_pct_ch_Y']  # case #2
#xDF['NewPort_pct_ch_Y']=xW1*xDF['HYG_pct_ch_Y'] + xW2*xDF['SPXT_pct_ch_Y']+
xW3*xDF['SI2_pct_ch_Y']  # case #3
#xDF['NewPort_pct_ch_Y']=xW1*xDF[xY_col+'_pct_ch_Y'] + xW2*xDF['SPXT_pct_ch_Y']+
xW3*xDF['SI2_pct_ch_Y']+ xW4*xDF['BondTR_pct_ch_Y']  # case #3

# xDF['NewPort_pct_ch_Y']=xW1*xDF[xY_col+'_pct_ch_Y'] + xW2*xDF['SHY_pct_ch_Y']+
xW3*xDF['SI4_pct_ch_Y']  # case #1
# xDF['NewPort_pct_ch_Y']=xW1*xDF[xY_col+'_pct_ch_Y'] + xW2*xDF['SHY_pct_ch_Y'] #+
xW3*xDF['SI4_pct_ch_Y']  # case #1
# xDF['NewPort_pct_ch_Y']=xW1*xDF[xY_col+'_pct_ch_Y'] + xW2*xDF['AGG_pct_ch_Y'] #+
xW3*xDF['SI4_pct_ch_Y']  # case #1
#xDF['NewPort_pct_ch_Y']=xW1*xDF[xY_col+'_pct_ch_Y'] + xW2*xDF['SPY_pct_ch_Y'] +
xW3*xDF['SI6_pct_ch_Y']  # case #1

#xDF['NewPort_pct_ch_Y']=xW1*xDF['PRWAX_pct_ch_Y'] + xW2*xDF['BondTR_pct_ch_Y']+
xW3*xDF['SI2_pct_ch_Y']

############################
xDF_M = pd.merge(xDF_M,xDF[['DATE','NewPort']],on=['DATE'],how='left')
######################
xCols_pct = xDF.columns[xDF.columns.str.contains(pat = '_pct_ch')]

xDF2=xDF[xCols_pct].copy()

xCorrelations = xDF2.corr()
xStdDev=xDF2.std()
xMean = xDF2.mean()
xCorrelations.to_csv(xDir+'xCorrelations.txt')
xStdDev.to_csv(xDir+'xStdDev.txt')
xMean.to_csv(xDir+'xMean.txt')
########### model portfolios #########
CONS_E=0.2
MODCONS_E=0.4
MOD_E=0.6
MODGROW_E=0.75
GROW_E=0.9
MAXGROW_E=0.98

#xMP=xDF[['DATE','SPXT','SPXT_pct_ch_D','SPXT_pct_ch_Y','BondTR','BondTR_pct_ch_D','BondTR_pct
_ch_Y']].copy()
xMP=xDF[['DATE','SPXT_pct_ch_D','BondTR_pct_ch_D',xY_col+'_pct_ch_D','NewPort_pct_ch_D','NewPo
rt_pct_ch_Y']].copy()
xMP.rename(columns={xY_col+'_pct_ch_D':'Current_pct_ch_D','NewPort_pct_ch_D':'New_pct_ch_D'},i
nplace=True)
xMP['CONS_pct_ch_D']=CONS_E*xMP['SPXT_pct_ch_D']+(1-CONS_E)*xMP['BondTR_pct_ch_D']      #daily
```

```
rebalanced as benchmark index
xMP['MODCONS_pct_ch_D']=MODCONS_E*xMP['SPXT_pct_ch_D']+(1-MODCONS_E)*xMP['BondTR_pct_ch_D']
#daily rebalanced as benchmark index
xMP['MOD_pct_ch_D']=MOD_E*xMP['SPXT_pct_ch_D']+(1-MOD_E)*xMP['BondTR_pct_ch_D']     #daily
rebalanced as benchmark index
xMP['MODGROW_pct_ch_D']=MODGROW_E*xMP['SPXT_pct_ch_D']+(1-MODGROW_E)*xMP['BondTR_pct_ch_D']
#daily rebalanced as benchmark index
xMP['GROW_pct_ch_D']=GROW_E*xMP['SPXT_pct_ch_D']+(1-GROW_E)*xMP['BondTR_pct_ch_D']     #daily
rebalanced as benchmark index
xMP['MAXGROW_pct_ch_D']=MAXGROW_E*xMP['SPXT_pct_ch_D']+(1-MAXGROW_E)*xMP['BondTR_pct_ch_D']
#daily rebalanced as benchmark index

xMP['CONS_port']=(1 + xMP['CONS_pct_ch_D']).cumprod()
xMP['MODCONS_port']=(1 + xMP['MODCONS_pct_ch_D']).cumprod()
xMP['MOD_port']=(1 + xMP['MOD_pct_ch_D']).cumprod()
xMP['MODGROW_port']=(1 + xMP['MODGROW_pct_ch_D']).cumprod()
xMP['GROW_port']=(1 + xMP['GROW_pct_ch_D']).cumprod()
xMP['MAXGROW_port']=(1 + xMP['MAXGROW_pct_ch_D']).cumprod()
xMP['Current_port']=(1 + xMP['Current_pct_ch_D']).cumprod()
xMP['New_port']=(1 + xMP['New_pct_ch_D']).cumprod()

xMP['CONS_pct_ch_Y']=xMP['CONS_port'].pct_change(252)
xMP['MODCONS_pct_ch_Y']=xMP['MODCONS_port'].pct_change(252)
xMP['MOD_pct_ch_Y']=xMP['MOD_port'].pct_change(252)
xMP['MODGROW_pct_ch_Y']=xMP['MODGROW_port'].pct_change(252)
xMP['GROW_pct_ch_Y']=xMP['GROW_port'].pct_change(252)
xMP['MAXGROW_pct_ch_Y']=xMP['MAXGROW_port'].pct_change(252)
xMP['Current_pct_ch_Y']=xMP['Current_port'].pct_change(252)
xMP['New_pct_ch_Y']=xMP['New_port'].pct_change(252)

xMP['New_pct_ch_Y']=xMP['NewPort_pct_ch_Y']

############################
xMP_MQ = xDF_M[['DATE']].copy()
xMP_MQ = pd.merge(xMP_MQ,
xMP[['DATE','CONS_port','MODCONS_port','MOD_port','MODGROW_port','GROW_port','MAXGROW_port',
                               'Current_port','New_port']],on=['DATE'],how='left')
xMP_MQ['CONS_pct_ch_M']=xMP_MQ['CONS_port'].pct_change()
xMP_MQ['MODCONS_pct_ch_M']=xMP_MQ['MODCONS_port'].pct_change()
xMP_MQ['MOD_pct_ch_M']=xMP_MQ['MOD_port'].pct_change()
xMP_MQ['MODGROW_pct_ch_M']=xMP_MQ['MODGROW_port'].pct_change()
xMP_MQ['GROW_pct_ch_M']=xMP_MQ['GROW_port'].pct_change()
xMP_MQ['MAXGROW_pct_ch_M']=xMP_MQ['MAXGROW_port'].pct_change()
xMP_MQ['Current_pct_ch_M']=xMP_MQ['Current_port'].pct_change()
xMP_MQ['New_pct_ch_M']=xMP_MQ['New_port'].pct_change()

#####xMP_MQ['Current_pct_ch_M']=xMP_MQ['Current_port'].pct_change()

xMP_MQ['CONS_pct_ch_Q']=xMP_MQ['CONS_port'].pct_change(3)
xMP_MQ['MODCONS_pct_ch_Q']=xMP_MQ['MODCONS_port'].pct_change(3)
xMP_MQ['MOD_pct_ch_Q']=xMP_MQ['MOD_port'].pct_change(3)
xMP_MQ['MODGROW_pct_ch_Q']=xMP_MQ['MODGROW_port'].pct_change(3)
xMP_MQ['GROW_pct_ch_Q']=xMP_MQ['GROW_port'].pct_change(3)
xMP_MQ['MAXGROW_pct_ch_Q']=xMP_MQ['MAXGROW_port'].pct_change(3)
xMP_MQ['Current_pct_ch_Q']=xMP_MQ['Current_port'].pct_change(3)
xMP_MQ['New_pct_ch_Q']=xMP_MQ['New_port'].pct_change(3)
############################
xCols_pct_MP = xMP.columns[xMP.columns.str.contains(pat = '_pct_ch_Y')]
xMP2=xMP[xCols_pct_MP].copy()
xAnnRtn_MP_Y=xMP2.mean()
```

```python
xStdDev_MP_Y=xMP2.std()
xStdDev_MP_Y.to_csv(xDir+'xStdDev_MP_Y.txt')
xAnnRtn_MP_Y.to_csv(xDir+'xAnnRtn_MP_Y.txt')
#################
xCols_pct_MP = xMP.columns[xMP.columns.str.contains(pat = '_pct_ch_D')]
xMP2=xMP[xCols_pct_MP].copy()
xAnnRtn_MP_D=xMP2.mean() * 252 ####  try to annualized compounded annual return
xStdDev_MP_D=xMP2.std() * np.sqrt(252)
xStdDev_MP_D.to_csv(xDir+'xStdDev_MP_D.txt')
xAnnRtn_MP_D.to_csv(xDir+'xAnnRtn_MP_D.txt')

#xStdDev_MP[0] is the std dev of the conservative model portfolio
#xStdDev_MP[5] is the std dev of the MAX Growth model portfolio
# std dev > xStdDev_MP[5] is ACCESSIVE GROWTH portfolio!!!!

######## OLS HERE ANNUAL ###############
xCols_pct_ch_Y= xDF.columns[xDF.columns.str.contains(pat = '_pct_ch_Y')]
xCols_pct_ch_Y=xCols_pct_ch_Y.insert(0,'DATE')

xRiskFactorSet_Y=['SPXT_pct_ch_Y','BondTR_pct_ch_Y','CCY_pct_ch_Y','COMM_pct_ch_Y','USCredit_p
ct_ch_Y','HYTR_pct_ch_Y',
                  'TIPS_pct_ch_Y','Inflation_pct_ch_Y']
#'CPI_pct_ch_M','RealBondTR_pct_ch_Y'

xRiskFactorSet_Y=['SPXT_pct_ch_Y','USCredit_pct_ch_Y','TIPS_pct_ch_Y','COMM_pct_ch_Y']
#'CPI_pct_ch_M','RealBondTR_pct_ch_Y'
#xRiskFactorSet_Y=['SPXT_pct_ch_Y','USCredit_pct_ch_Y','TIPS_pct_ch_Y','COMM_pct_ch_Y','HYTR_p
ct_ch_Y']   #'CPI_pct_ch_M','RealBondTR_pct_ch_Y'

################## derive orthogonal risk factors ##################
xDF_orthog = xDF[['DATE']+xRiskFactorSet_Y]
xDF_orthog.dropna(inplace=True)
xDF_orthog.reset_index(drop=True,inplace=True)
## (1) derive orthog_SPXT #####
Y = xDF_orthog['SPXT_pct_ch_Y']
X = xDF_orthog['TIPS_pct_ch_Y']
X = sm.add_constant(X)
model = sm.OLS(Y, X)
result = model.fit()
xDF_orthog['orthog_SPXT_pct_ch_Y'] = result.params[0] + result.resid
print(xDF_orthog[['orthog_SPXT_pct_ch_Y', 'TIPS_pct_ch_Y']].corr())

## (2) derive orthog_USCredit #####
Y = xDF_orthog['USCredit_pct_ch_Y']
X = xDF_orthog[['SPXT_pct_ch_Y','TIPS_pct_ch_Y']]
#X = xDF_orthog['TIPS_pct_ch_Y']
X = sm.add_constant(X)
model = sm.OLS(Y, X)
result = model.fit()
xDF_orthog['orthog_USCredit_pct_ch_Y'] = result.params[0] + result.resid
print(xDF_orthog[['orthog_USCredit_pct_ch_Y','TIPS_pct_ch_Y']].corr())

## (3) derive orthog_COMM #####
Y = xDF_orthog['COMM_pct_ch_Y']
#X = xDF_orthog[['SPXT_pct_ch_Y','TIPS_pct_ch_Y']]
X = xDF_orthog[['SPXT_pct_ch_Y','TIPS_pct_ch_Y','USCredit_pct_ch_Y']]
X = sm.add_constant(X)
model = sm.OLS(Y, X)
result = model.fit()
xDF_orthog['orthog_COMM_pct_ch_Y'] = result.params[0] + result.resid
```

```
print(xDF_orthog[['orthog_COMM_pct_ch_Y','TIPS_pct_ch_Y']].corr())

xRiskFactorCorrelations_orthog =
(xDF_orthog[['orthog_SPXT_pct_ch_Y','orthog_USCredit_pct_ch_Y','orthog_COMM_pct_ch_Y','TIPS_pc
t_ch_Y']].corr()).round(4)
xRiskFactorCorrelations_raw =
(xDF_orthog[['SPXT_pct_ch_Y','USCredit_pct_ch_Y','COMM_pct_ch_Y','TIPS_pct_ch_Y']].corr()).rou
nd(4)

xDF =
pd.merge(xDF,xDF_orthog[['DATE','orthog_SPXT_pct_ch_Y','orthog_USCredit_pct_ch_Y','orthog_COMM
_pct_ch_Y']],on=['DATE'],how='left')
##################### bring in the rolling annual returns for Model Portfolios as a benchmarks
for Risk Exposures ###########
xDF =
pd.merge(xDF,xMP[['DATE','CONS_pct_ch_Y','MODCONS_pct_ch_Y','MOD_pct_ch_Y','MODGROW_pct_ch_Y',
'GROW_pct_ch_Y','MAXGROW_pct_ch_Y']],on=['DATE'],how='left')
#########################################################################################
xOrthogonal = 'orthog'
#xOrthogonal = ''

if xOrthogonal == 'orthog':
    xRiskFactorSet_Y =
['orthog_SPXT_pct_ch_Y','orthog_USCredit_pct_ch_Y','TIPS_pct_ch_Y','orthog_COMM_pct_ch_Y']
    #xRiskFactorSet_Y = ['orthog_SPXT_pct_ch_Y', 'TIPS_pct_ch_Y', 'orthog_COMM_pct_ch_Y']
else:
    xOrthogonal = ''
    xRiskFactorSet_Y = ['SPXT_pct_ch_Y', 'USCredit_pct_ch_Y', 'TIPS_pct_ch_Y',
                        'COMM_pct_ch_Y']  # 'CPI_pct_ch_M','RealBondTR_pct_ch_Y'
    #xRiskFactorSet_Y = ['SPXT_pct_ch_Y',
'TIPS_pct_ch_Y','RealBondTR_pct_ch_Y','COMM_pct_ch_Y']  # 'CPI_pct_ch_M','RealBondTR_pct_ch_Y'

#xRiskFactorSet_Y =
['orthog_SPXT_pct_ch_Y','orthog_USCredit_pct_ch_Y','TIPS_pct_ch_Y','orthog_COMM_pct_ch_Y','HYT
R_pct_ch_Y']
############# in REAL terms #####################
xDF['RealSPXT_pct_ch_Y'] = xDF['SPXT_pct_ch_Y'] - xDF['TIPS_pct_ch_Y']
xDF['RealUSCredit_pct_ch_Y'] = xDF['USCredit_pct_ch_Y'] - xDF['TIPS_pct_ch_Y']
xDF['RealCOMM_pct_ch_Y'] = xDF['COMM_pct_ch_Y'] - xDF['TIPS_pct_ch_Y']
##xRiskFactorSet_Y=['RealSPXT_pct_ch_Y','RealUSCredit_pct_ch_Y','TIPS_pct_ch_Y','RealCOMM_pct_
ch_Y']
###################################

xDescriptive_Y=xDF[[xY_col+'_pct_ch_Y']+xRiskFactorSet_Y].describe(include='all').to_string()
xCorrelations_Y=xDF[[xY_col+'_pct_ch_Y']+xRiskFactorSet_Y].corr().to_string()

xDescriptive_Y = xDescriptive_Y + '\n\n' + xCorrelations_Y
f = open(xDir + 'xDescriptive_Y_'+xY_col+'.txt','w')
f.write(xDescriptive_Y + '\r\n')
f.close()

xDep_var = ['SPY','AGG','HYG','SHY','MSFT','AMZN','FB','GOOG','VIAC',
            'LQD','30AAPL30MSFT20AMZN20GOOGTR','30SPY30MDY20AGG20LQDTR','TSLA',
            '7030TR','SI2','SI4','SI6','SPLPEQTY','CONS','MODCONS',
            'MOD','MODGROW','GROW','MAXGROW','CashConst','AAPL']
xDep_var = ['SPY','AGG','HYG','SHY','MSFT','AMZN','FB','GOOG','VIAC',
            'LQD','30AAPL30MSFT20AMZN20GOOGTR','30SPY30MDY20AGG20LQDTR','AAPL',
            '7030TR','3070TR','SI2','SI4','SI6','SPLPEQTY','CONS','MODCONS',
            'MOD','MODGROW','GROW','MAXGROW','CashConst','RLV','RIY','RMV',
            'RMC','RDG','RUJ','RTY','RUO','TSLA']  #### 'RLG',
```

```
#xDep_var = ['TSLA']  #### 'RLG',

### xDep_var = ['RTY','RUO','AAPL']

#xDep_var = ['RLV','RLG','RIY','RMV','RMC','RDG','RUJ','RTY','RUO','AAPL']

#xDep_Var = [xY_col]

xRiskFactorSet_Y = ['SPXT_pct_ch_Y','BondTR_pct_ch_Y']
#xRiskFactorSet_Y = ['RLG_pct_ch_Y']

xRisk_concentration_Current = pd.DataFrame()
xRisk_concentration_New = pd.DataFrame()
for xY_col in xDep_var:
    ####xDF_OLS_Y=xDF[xCols_pct_ch_Y].copy()
    xDF_OLS_Y = xDF[['DATE'] + xRiskFactorSet_Y +
[xY_col+'_pct_ch_Y','NewPort_pct_ch_Y']].copy()  #'CPI_pct_ch_Y',

    xDF_OLS_Y.dropna(inplace=True)
    xDF_OLS_Y.reset_index(drop=True,inplace=True)
    ########### set up weightings for WLS here ####################
    xDF_OLS_Y['w'] = np.exp(-(-xDF_OLS_Y.index+xDF_OLS_Y.index.max()) / (len(xDF_OLS_Y) / 5))
# exponential
    #xDF_OLS_Y['w'] = (xDF_OLS_Y.index) / xDF_OLS_Y.index.max()  # linear - the latest has
more weights!
    ###########################
    xStartDate_Y = xDF_OLS_Y['DATE'].min()
    xEndDate_Y = xDF_OLS_Y['DATE'].max()
    #################### model portfolios for Rooling annual rate ############
    xMP_Y = xDF_OLS_Y[['DATE']].copy()
    xMP_Y = pd.merge(xMP_Y,
xMP[['DATE','CONS_pct_ch_Y','MODCONS_pct_ch_Y','MOD_pct_ch_Y','MODGROW_pct_ch_Y','GROW_pct_ch_
Y',
                        'MAXGROW_pct_ch_Y','Current_pct_ch_Y','New_pct_ch_Y']], on=['DATE'],
how='left')
    xMP_Y_AnnRtn = xMP_Y.mean().reset_index()
    xMP_Y_AnnRisk = xMP_Y.std().reset_index()
    xMP_Y_AnnRtn.rename(columns={0: 'AnnRtn'},inplace=True)
    xMP_Y_AnnRisk.rename(columns={0: 'AnnRisk'},inplace=True)

    xMP_Y_AnnRtnRisk = pd.merge(xMP_Y_AnnRtn,xMP_Y_AnnRisk,on=['index'],how='left')
    ################ OLS ################
    xInd_Vars =
['SPXT_pct_ch_Y','BondTR_pct_ch_Y','TIPS_pct_ch_Y','CCY_pct_ch_Y','COMM_pct_ch_Y','USCredit_pc
t_ch_Y','HYTR_pct_ch_Y']
    xInd_Vars =
['SPXT_pct_ch_Y','BondTR_pct_ch_Y','TIPS_pct_ch_Y','CCY_pct_ch_Y','USCredit_pct_ch_Y']
    xInd_Vars =
['SPXT_pct_ch_Y','BondTR_pct_ch_Y','TIPS_pct_ch_Y','CCY_pct_ch_Y','USCredit_pct_ch_Y','HYTR_pc
t_ch_Y']
    xInd_Vars =
['SPXT_pct_ch_Y','Inflation_pct_ch_Y','TIPS_pct_ch_Y','CCY_pct_ch_Y','USCredit_pct_ch_Y','HYTR
_pct_ch_Y']
    xInd_Vars =
['SPXT_pct_ch_Y','Inflation_pct_ch_Y','TIPS_pct_ch_Y','CCY_pct_ch_Y','USCredit_pct_ch_Y']
    xInd_Vars = ['SPXT_pct_ch_Y']
    xInd_Vars = ['SPXT_pct_ch_Y','BondTR_pct_ch_Y']
    xInd_Vars =
['SPXT_pct_ch_Y','BondTR_pct_ch_Y','TIPS_pct_ch_Y','CCY_pct_ch_Y','COMM_pct_ch_Y','USCredit_pc
t_ch_Y','HYTR_pct_ch_Y']
```

```
    xInd_Vars = xRiskFactorSet_Y

    ###xInd_Vars = ['S5INFT_pct_ch_Y']
    #xInd_Vars =
['SPXT_pct_ch_Y','Inflation_pct_ch_Y','TIPS_pct_ch_Y','CCY_pct_ch_Y','USCredit_pct_ch_Y']
    X = xDF_OLS_Y[xInd_Vars]
    ############### risk factors annual returns and std dev ##########
    X_StdDev_Y = xDF_OLS_Y[xInd_Vars].std().reset_index()
    X_Rtn_Y = xDF_OLS_Y[xInd_Vars].mean().reset_index()
    ##################################################################
    xVersion=['Current','New']
    #xVersion=['Current']
    xRisk_exposures_Current=pd.DataFrame()
    xRisk_exposures_New=pd.DataFrame()
    for x in xVersion:
        if x=='Current':
            Y = xDF_OLS_Y[xY_col + '_pct_ch_Y']
            xY_col2 = xY_col
            xCorrelations_Y = xDF_OLS_Y[[xY_col + '_pct_ch_Y'] + xInd_Vars].corr().to_string()
        elif x=='New':
            Y = xDF_OLS_Y['NewPort_pct_ch_Y']
            xY_col2 = 'New'
            xCorrelations_Y = xDF_OLS_Y[[xY_col2 + 'Port_pct_ch_Y'] +
xInd_Vars].corr().to_string()
        #xInd_Vars =
['SPXT_pct_ch_Y','RealBondTR_pct_ch_Y','TIPS_pct_ch_Y','CPI_pct_ch_Y','CCY_pct_ch_Y','COMM_pct
_ch_Y','USCredit_pct_ch_Y','HYTR_pct_ch_Y']

        #xCorrelations_Y = xDF_OLS_Y[[xY_col + '_pct_ch_Y']+xInd_Vars].corr().to_string()
        f = open(xDir + 'xCorrelations_Y_' + xY_col + '_' + x + '.txt','w')
        f.write(xCorrelations_Y + '\r\n')
        f.close()

        X = sm.add_constant(X)
        xStart_time = datetime.datetime.now() #time.time_ns()*1000000
        xRegressionType ='WLS'    #'OLS' #'WLS'
        if xRegressionType == 'OLS':
            model = sm.OLS(Y,X)
        elif xRegressionType == 'WLS':
            model = sm.WLS(Y, X, weights=xDF_OLS_Y['w'])
        result = model.fit()
        # for i in range(1,9999):
        #     print(i)
        xEnd_time = datetime.datetime.now() #time.time_ns()*1000000
        globals()['xSecond_Y_'+x] = 'Start: '+(str)(xStart_time) +'; End: '+(str)(xEnd_time) +
'; Duration: ' +(str)((xEnd_time - xStart_time))
        xOLS_Summary_Y = result.summary()
        xOLS_text = xOLS_Summary_Y.as_text()

        f = open(xDir + 'xOLS_Y_' + xY_col +  '_' + x + '.txt','w')
        f.write(globals()['xSecond_Y_'+x] + '\n\n' + xOLS_text + '\r\n')
        f.close()

        # ########## calc annualized return YEARLY ##########
        xAnnRtn_Y_Y = Y.mean()
        xAnnRisk_Y_Y = np.sqrt(Y.var())
        # xMP_Y_AnnRtnRisk=xMP_Y_AnnRtnRisk.append({'index':'Current
Portfolio','AnnRtn':xAnnRtn_Y_Y,'AnnRisk':xAnnRisk_Y_Y}, ignore_index=True)
        # ##############################################
```

```
        xVar_X = np.array(X.var())
        xVar_Y = Y.var()
        xCoef_sq = result.params**2
        xVar_resid = result.resid.var()
        xVar_CoefX = xCoef_sq * xVar_X
        xDelta_var = xVar_Y - np.sum(xVar_CoefX) - xVar_resid       #this is the
diversificaation effect
        xDelta_varX = xDelta_var * xVar_CoefX / np.sum(xVar_CoefX)
        xVar_X_adj = xVar_CoefX + xDelta_varX
        xVar_X_adj_pct = xVar_X_adj / xVar_Y
        xVar_resid_pct = xVar_resid / xVar_Y
        print (xVar_X_adj_pct, xVar_resid_pct)
        print(np.sum(xVar_X_adj_pct)+xVar_resid_pct)

        xVar_X_adj_pct = pd.DataFrame(xVar_X_adj_pct)
        xVar_X_adj_pct.reset_index(inplace=True)

        xVar_X_adj_pct.rename(columns={0: 'Risk_Concentration(%)'},inplace=True)
        xVar_X_adj_pct.rename(columns={'index': 'Risk_Factor'},inplace=True)
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Idiosyncratic',
'Risk_Concentration(%)': xVar_resid_pct}, ignore_index=True)

        xVar_X_adj_pct=xVar_X_adj_pct.loc[~xVar_X_adj_pct['Risk_Factor'].isin({'const'})]
        xSum = xVar_X_adj_pct['Risk_Concentration(%)'].sum()
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Sum', 'Risk_Concentration(%)':
xSum}, ignore_index=True)
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual
StDev)', 'Risk_Concentration(%)': xAnnRisk_Y_Y}, ignore_index=True)
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual Rtn)',
'Risk_Concentration(%)': xAnnRtn_Y_Y}, ignore_index=True)

        xVar_X_adj_pct['Risk_Concentration(%)'] =
xVar_X_adj_pct['Risk_Concentration(%)'].astype(float).map("{:.2%}".format)
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Sharpe
Ratio)', 'Risk_Concentration(%)': np.round(xAnnRtn_Y_Y/xAnnRisk_Y_Y,2)}, ignore_index=True)

        # for x in (result.tvalues.index):
        #     if x=='const':
        #         continue
        #     else:
        #         #print (x, result.tvalues[x])
        #         if (abs(result.tvalues[x]) <1.5):
        #             xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor'] == x]['Risk_Exposure(%)'] =
'NA'
        #             #print (x,
xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor']==x]['Risk_Exposure(%)'])

        #####globals()['xRisk_concentration_'+x] = xVar_X_adj_pct

        xRisk_Exposure_Y = xVar_X_adj_pct.to_string()

        f = open(xDir + 'xRisk_Concentration_Y_' + xY_col + '_' + x + '.txt', 'w')
        f.write(xRisk_Exposure_Y + '\r\n')
        f.close()

        ################# store Risk Concentration for the Current Portfolio ###############
        xRiskConcentration_temp = xVar_X_adj_pct[['Risk_Factor',
'Risk_Concentration(%)']].copy()
        xRiskConcentration_temp.rename(columns={'Risk_Concentration(%)': xY_col},
```

```
inplace=True)
        xRiskConcentration_temp['Risk_Factor'][len(xRiskConcentration_temp) - 1] =
'Sharpe_Ratio'
        xRiskConcentration_temp['Risk_Factor'][len(xRiskConcentration_temp) - 2] =
'Annual_Rtn'
        xRiskConcentration_temp['Risk_Factor'][len(xRiskConcentration_temp) - 3] =
'Annual_StdDev'
        if len(globals()['xRisk_concentration_'+x]) == 0:
            globals()['xRisk_concentration_'+x] = xRiskConcentration_temp.copy()
        else:
            globals()['xRisk_concentration_'+x] =
pd.merge(globals()['xRisk_concentration_'+x], xRiskConcentration_temp, on=['Risk_Factor'],
how='left')

        # ######################################
        ############### the following is working on the Std Dev (RISK) ANNUALLY #######
        xStdDev_X = np.array(X.std())   #these are already annualized std dev
        xStdDev_Y = Y.std()     #these are already annualized std dev
        xCoef = result.params.abs()
        xStdDev_resid = result.resid.std()
        xStdDev_CoefX = xCoef * xStdDev_X
        xDelta_StdDev = xStdDev_Y - np.sum(xStdDev_CoefX) - xStdDev_resid  #this is the
diversification benefit...
        print('xDelta_StdDev = ', xDelta_StdDev)
        xAdj_StdDev_resid = False
        if (xAdj_StdDev_resid == False):
            xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / np.sum(xStdDev_CoefX)
        else:
            xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / (np.sum(xStdDev_CoefX) +
xStdDev_resid)
            xStdDev_resid = xStdDev_resid + xDelta_StdDev * xStdDev_resid /
(np.sum(xStdDev_CoefX) + xStdDev_resid)
        xStdDev_X_adj = xStdDev_CoefX + xDelta_StdDevX

        xStdDev_X_adj = pd.DataFrame(xStdDev_X_adj)
        xStdDev_X_adj.reset_index(inplace=True)

        xStdDev_X_adj.rename(columns={0: 'Risk_Exposure(%)'},inplace=True)
        xStdDev_X_adj.rename(columns={'index': 'Risk_Factor'},inplace=True)
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Idiosyncratic', 'Risk_Exposure(%)':
xStdDev_resid}, ignore_index=True)

        xStdDev_X_adj=xStdDev_X_adj.loc[~xStdDev_X_adj['Risk_Factor'].isin({'const'})]
        xSum = xStdDev_X_adj['Risk_Exposure(%)'].sum()
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sum', 'Risk_Exposure(%)': xSum},
ignore_index=True)
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual StDev)',
'Risk_Exposure(%)': xAnnRisk_Y_Y}, ignore_index=True)
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual Rtn)',
'Risk_Exposure(%)': xAnnRtn_Y_Y}, ignore_index=True)

        #xStdDev_X_adj['Risk_Exposure(%)'] =
xStdDev_X_adj['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)

        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sharpe Ratio (Rtn/Risk)',
'Risk_Exposure(%)': np.round(xAnnRtn_Y_Y / xAnnRisk_Y_Y,2)}, ignore_index=True)
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Diversification benefit',
'Risk_Exposure(%)': xDelta_StdDev}, ignore_index=True)

        if x== 'Current':
```

```
            xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (' + xY_col + ')'},
inplace=True)
        else:
            xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (proposed)'},
inplace=True)
        globals()['xRisk_exposures_' + x] = xStdDev_X_adj

        xIndex_StdDev=globals()['xRisk_exposures_' + x][
            globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names + '(Annual
StdDev)'].index.values[0]
        xIndex_Rtn = globals()['xRisk_exposures_' + x][
            globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names + '(Annual
Rtn)'].index.values[0]
        globals()['xRisk_exposures_' + x].loc[globals()['xRisk_exposures_' + x].index ==
xIndex_StdDev, 'Risk_Factor'] = 'Annual StdDev'
        globals()['xRisk_exposures_' + x].loc[
            globals()['xRisk_exposures_' + x].index == xIndex_Rtn, 'Risk_Factor'] = 'Annual
Rtn'

        globals()['xIdio_exp_' + x] = xStdDev_resid / xSum

        if x=='New':  # second time and last time!
            xStdDev_X_adj = pd.merge(xRisk_exposures_Current, xRisk_exposures_New,
on='Risk_Factor', how='left')

        xRisk_Exposure_Y_StdDev = xStdDev_X_adj.to_string()

        ######################################################################
        #xRisk_Exposure_Y.to_csv (xDir + 'xRisk_Exposure_Y.txt')

        f = open(xDir + 'xRisk_Exposure_Y_' + xY_col +  '_' + x + '.txt','w')
        f.write(xRisk_Exposure_Y_StdDev + '\r\n')
        f.close()
        ############### store oefficients for 'Current" portfolio ########
        if x=='Current':
            xCoef_temp = pd.DataFrame(result.params).reset_index()
            xCoef_temp.rename(columns={0: xY_col}, inplace=True)
            xCoef_temp[xY_col] = xCoef_temp[xY_col].round(4)
            xCoef_temp.rename(columns={'index': 'Risk_Factor'}, inplace=True)
            if len(xCoef_table)==0:
                xCoef_table = xCoef_temp.copy()
            else:
                xCoef_table = pd.merge(xCoef_table, xCoef_temp, on=['Risk_Factor'],how='left')
        ############### the following is working on the Std Dev (RISK) ANNUALLY with AR(1)
error term #######
        if False:
            ###################### the following is AR(1) error term ########################
            from statsmodels.tsa.arima.model import ARIMA as ARIMA

            X2 = X.drop('const', axis=1)
            sarimax_model = ARIMA(endog=Y, exog=X2, order=(1, 0, 0))  # X already has a
constant term, trend='c')  # , seasonal_order=(0,1,1,24))
            sarimax_results = sarimax_model.fit()
            sarimax_results.summary()

            xOLS_AR1_Summary_Y = sarimax_results.summary()
            xOLS_AR1_text = xOLS_AR1_Summary_Y.as_text()

            f = open(xDir + 'xOLS_AR1_Y_' + xY_col + '_' + x + '.txt', 'w')
            f.write(xOLS_AR1_text + '\r\n')
```

```
            f.close()

            xStdDev_X = np.array(X.std())   #these are already annualized std dev
            xStdDev_Y = Y.std()     #these are already annualized std dev
            xCoef = sarimax_results.params[:len(X.columns)].abs()
            xStdDev_resid = np.sqrt(sarimax_results.params[len(X.columns):].values[1] / (1-
sarimax_results.params[len(X.columns):].values[0]**2)) #result.resid.std()
            xStdDev_CoefX = xCoef * xStdDev_X
            xDelta_StdDev = xStdDev_Y - np.sum(xStdDev_CoefX) - xStdDev_resid
            print('xDelta_StdDev = ', xDelta_StdDev)
            xAdj_StdDev_resid = False
            if (xAdj_StdDev_resid == False):
                xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / np.sum(xStdDev_CoefX)
            else:
                xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / (np.sum(xStdDev_CoefX) +
xStdDev_resid)
                xStdDev_resid = xStdDev_resid + xDelta_StdDev * xStdDev_resid /
(np.sum(xStdDev_CoefX) + xStdDev_resid)
            xStdDev_X_adj = xStdDev_CoefX + xDelta_StdDevX

            xStdDev_X_adj = pd.DataFrame(xStdDev_X_adj)
            xStdDev_X_adj.reset_index(inplace=True)

            xStdDev_X_adj.rename(columns={0: 'Risk_Exposure(%)'},inplace=True)
            xStdDev_X_adj.rename(columns={'index': 'Risk_Factor'},inplace=True)
            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Idiosyncratic',
'Risk_Exposure(%)': xStdDev_resid}, ignore_index=True)

            xStdDev_X_adj=xStdDev_X_adj.loc[~xStdDev_X_adj['Risk_Factor'].isin({'const'})]
            xSum = xStdDev_X_adj['Risk_Exposure(%)'].sum()
            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sum', 'Risk_Exposure(%)':
xSum}, ignore_index=True)
            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual
StDev)', 'Risk_Exposure(%)': xAnnRisk_Y_Y}, ignore_index=True)
            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual
Rtn)', 'Risk_Exposure(%)': xAnnRtn_Y_Y}, ignore_index=True)

            #xStdDev_X_adj['Risk_Exposure(%)'] =
xStdDev_X_adj['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)

            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sharpe Ratio (Rtn/Risk)',
'Risk_Exposure(%)': np.round(xAnnRtn_Y_Y / xAnnRisk_Y_Y,2)}, ignore_index=True)

            if x== 'Current':
                xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (' + xY_col +
')'}, inplace=True)
            else:
                xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (proposed)'},
inplace=True)
            globals()['xRisk_exposures_' + x] = xStdDev_X_adj

            xIndex_StdDev=globals()['xRisk_exposures_' + x][
                globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names +
'(Annual StDev)'].index.values[0]
            xIndex_Rtn = globals()['xRisk_exposures_' + x][
                globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names +
'(Annual Rtn)'].index.values[0]
            globals()['xRisk_exposures_' + x].loc[globals()['xRisk_exposures_' + x].index ==
xIndex_StdDev, 'Risk_Factor'] = 'Annual StDev'
            globals()['xRisk_exposures_' + x].loc[
```

```
                globals()['xRisk_exposures_' + x].index == xIndex_Rtn, 'Risk_Factor'] =
'Annual Rtn'

            globals()['xIdio_exp_' + x] = xStdDev_resid / xSum

            if x=='New':  # second time and last time!
                xStdDev_X_adj = pd.merge(xRisk_exposures_Current, xRisk_exposures_New,
on='Risk_Factor', how='left')


            xRisk_Exposure_Y_StdDev = xStdDev_X_adj.to_string()

            ######################################################################
            #xRisk_Exposure_Y.to_csv (xDir + 'xRisk_Exposure_Y.txt')

            f = open(xDir + 'xRisk_Exposure_Y_AR(1)_' + xY_col +  '_' + x + '.txt','w')
            f.write(xRisk_Exposure_Y_StdDev + '\r\n')
            f.close()

    ######################
    xStdDev_X_Y = pd.DataFrame(X.std()).reset_index()
    xStdDev_X_Y.rename(columns={0:'Risk_Factor_AnnStdDev'}, inplace=True)
    xStdDev_X_Y.rename(columns={'index':'Risk_Factor'}, inplace=True)
    xStdDev_X_adj = pd.merge(xStdDev_X_adj,xStdDev_X_Y, on=['Risk_Factor'],how='left')
    xStdDev_X_adj['Risk_Exp (Current)'] = xStdDev_X_adj['Current Risk
('+xY_col+')']/xStdDev_X_adj['Risk_Factor_AnnStdDev']
    xStdDev_X_adj['Risk_Exp (New)'] = xStdDev_X_adj['New Risk
(proposed)']/xStdDev_X_adj['Risk_Factor_AnnStdDev']
    xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Idiosyncratic','Risk_Exp
(Current)']=xIdio_exp_Current
    xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Idiosyncratic','Risk_Exp
(New)']=xIdio_exp_New
    xSum_Risk_Exp_Current = xStdDev_X_adj['Risk_Exp (Current)'].sum()
    xSum_Risk_Exp_New = xStdDev_X_adj['Risk_Exp (New)'].sum()
    xStdDev_X_adj['Risk_Exp (Current)'] = xStdDev_X_adj['Risk_Exp
(Current)']/xSum_Risk_Exp_Current
    xStdDev_X_adj['Risk_Exp (New)'] = xStdDev_X_adj['Risk_Exp (New)']/xSum_Risk_Exp_New
    xSum_Risk_Exp_Current = xStdDev_X_adj['Risk_Exp (Current)'].sum()
    xSum_Risk_Exp_New = xStdDev_X_adj['Risk_Exp (New)'].sum()
    xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Sum','Risk_Exp
(Current)']=xSum_Risk_Exp_Current
    xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Sum','Risk_Exp (New)']=xSum_Risk_Exp_New

    xPart_1=xStdDev_X_adj.loc[xStdDev_X_adj.index<len(xStdDev_X_adj)-1]
    xPart_2=xStdDev_X_adj.loc[xStdDev_X_adj.index==len(xStdDev_X_adj)-1]

    xPart_1['Current Risk ('+xY_col+')'] = xPart_1['Current Risk
('+xY_col+')'].astype(float).map("{:.2%}".format)
    xPart_1['New Risk (proposed)'] = xPart_1['New Risk
(proposed)'].astype(float).map("{:.2%}".format)
    xPart_1['Risk_Factor_AnnStdDev'] =
xPart_1['Risk_Factor_AnnStdDev'].astype(float).map("{:.2%}".format)
    xPart_1['Risk_Exp (Current)'] = xPart_1['Risk_Exp
(Current)'].astype(float).map("{:.2%}".format)
    xPart_1['Risk_Exp (New)'] = xPart_1['Risk_Exp (New)'].astype(float).map("{:.2%}".format)

    xStdDev_X_adj=xPart_1.append(xPart_2, ignore_index=True)

    xStdDev_X_adj = xStdDev_X_adj.replace({'nan%': ''})
    xStdDev_X_adj = xStdDev_X_adj.replace({np.nan: ''})
```

```
    xRisk_Exposure_Y_StdDev = xStdDev_X_adj.to_string()
    #xStdDev_X_adj['Risk_Exposure(%)'] =
xStdDev_X_adj['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)
    ######################
    f = open(xDir + 'xRisk_Exposure_Y_' + xY_col + '.txt','w')
    f.write('From '+xStartDate_Y.strftime('%Y-%m-%d') +' to ' + xEndDate_Y.strftime('%Y-%m-
%d') + '\n\n' +xRisk_Exposure_Y_StdDev + '\r\n')
    f.close()
    ################# store Risk Exposures for the Current Portfolio ###############
    xRiskExp_Current_temp = xStdDev_X_adj[['Risk_Factor','Risk_Exp (Current)']].copy()
    xRiskExp_Current_temp.rename(columns={'Risk_Exp (Current)':xY_col}, inplace=True)
    if len(xRiskExp_Current)==0:
        xRiskExp_Current = xRiskExp_Current_temp.copy()
    else:
        xRiskExp_Current =
pd.merge(xRiskExp_Current,xRiskExp_Current_temp,on=['Risk_Factor'],how='left')

#res = pd.concat([xRiskExp_Current, xCoef_table], axis=1, keys=["Risk_Exp", "Coefs"])
d={} #dictionary of dataframe
xRiskExp_Current2 =
xRiskExp_Current.loc[xRiskExp_Current['Risk_Factor'].isin(list(xRiskFactorSet_Y+['Idiosyncrati
c']))]
d['Current_Risk_Exposures']=xRiskExp_Current2.set_index('Risk_Factor')
d=pd.concat(d, axis=1)

A={}
xCoef_table2 = xCoef_table.loc[xCoef_table['Risk_Factor'].isin(xRiskFactorSet_Y)]
A['Coefficients']=xCoef_table2.set_index('Risk_Factor')
A=pd.concat(A, axis=1)

B={}
##xRisk_concentration_Current2 =
xRisk_concentration_Current.loc[xRisk_concentration_Current['Risk_Factor'].isin(xRiskFactorSet
_Y)]
B['Current_Risk_Concentration']=xRisk_concentration_Current.set_index('Risk_Factor')
B=pd.concat(B, axis=1)

C={}
###xRisk_concentration_New2 =
xRisk_concentration_New.loc[xRisk_concentration_New['Risk_Factor'].isin(xRiskFactorSet_Y)]
C['New_Risk_Concentration']=xRisk_concentration_New.set_index('Risk_Factor')
C=pd.concat(C, axis=1)

# A2=pd.merge(A,d,on=['Risk_Factor'],how='outer')
# A2.reset_index(inplace=True)
#
# A2_text = A2.to_csv()   #.to_string()
#
# if xOrthogonal == 'orthog':
#     f = open(xDir + 'xCoef_Risk_Exposure_Y_' + xOrthogonal+'.csv','w')
# else:
#     f = open(xDir + 'xCoef_Risk_Exposure_Y.csv', 'w')
# f.write(A2_text + '\r\n')
# f.close()
# ############################
# A2_text = A2.to_string()
#
# if xOrthogonal == 'orthog':
#     f = open(xDir + 'xCoef_Risk_Exposure_Y_' + xOrthogonal+'.txt','w')
```

```
# else:
#     f = open(xDir + 'xCoef_Risk_Exposure_Y.txt', 'w')
# f.write(A2_text + '\r\n')
# f.close()

# create excel writer
if xOrthogonal == 'orthog':
    writer = pd.ExcelWriter(xDir + 'xRiskFactorExp_Corre_orthog.xlsx')
    d.reset_index().to_excel(writer, 'Risk_Exposures_orthog')
    A.reset_index().to_excel(writer, 'Coefficients_orthog')
    xRiskFactorCorrelations_orthog.to_excel(writer, 'Corre_RiskFactor_orthog')
    B.reset_index().to_excel(writer, 'Risk_Concentration_orthog')
else:
    writer = pd.ExcelWriter(xDir + 'xRiskFactorExp_Corre_raw.xlsx')
    d.reset_index().to_excel(writer, 'Risk_Exposures_raw')
    A.reset_index().to_excel(writer, 'Coefficients_raw')
    xRiskFactorCorrelations_raw.to_excel(writer, 'Corre_RiskFactor_raw')
    B.reset_index().to_excel(writer, 'Risk_Concentration_raw')
#writer = pd.ExcelWriter(xDir + 'xRiskFactorExp_Corre.xlsx')
# write dataframe to excel sheet named 'marks'
# xRiskFactorCorrelations_orthog.to_excel(writer, 'Corre_RiskFactor_orthog')
# xRiskFactorCorrelations_raw.to_excel(writer, 'Corre_RiskFactor_raw')
# d.reset_index().to_excel(writer, 'Risk_Exposures')
# A.reset_index().to_excel(writer, 'Coefficients')
# save the excel file
writer.save()
writer.close()
###
#
#
##################################################################################
########################################
############### DAILY ###################
xCols_pct_ch_D= xDF.columns[xDF.columns.str.contains(pat = '_pct_ch_D')]
xCols_pct_ch_D=xCols_pct_ch_D.insert(0,'DATE')

xDF_OLS_D=xDF[xCols_pct_ch_D].copy()

xDF_OLS_D.dropna(inplace=True)
xDF_OLS_D.reset_index(drop=True,inplace=True)
#############################
xStartDate_D = xDF_OLS_D['DATE'].min()
xEndDate_D = xDF_OLS_D['DATE'].max()
#############################
xMean_OLS_D=xDF_OLS_D.mean()*252
xStdDev_OLS_D=xDF_OLS_D.std()*np.sqrt(252)
xStdDev_OLS_D.to_csv(xDir+'xStdDev_OLS_D.txt')
xMean_OLS_D.to_csv(xDir+'xMean_OLS_D.txt')

#################### model portfolios for daily returns ############
xMP_D = xDF_OLS_D[['DATE']].copy()
xMP_D = pd.merge(xMP_D,
xMP[['DATE','CONS_pct_ch_D','MODCONS_pct_ch_D','MOD_pct_ch_D','MODGROW_pct_ch_D',
                 'GROW_pct_ch_D','MAXGROW_pct_ch_D','Current_pct_ch_D','New_pct_ch_D']],
on=['DATE'], how='left')

xMP_D_CumRtn =
(1+xMP_D[['CONS_pct_ch_D','MODCONS_pct_ch_D','MOD_pct_ch_D','MODGROW_pct_ch_D','GROW_pct_ch_D'
,
                          'MAXGROW_pct_ch_D','Current_pct_ch_D','New_pct_ch_D']]).cumprod()
```

```
xMP_D_AnnRtn = (xMP_D_CumRtn.iloc[len(xMP_D_CumRtn)-
1]/xMP_D_CumRtn.iloc[0])**(1/(len(xMP_D_CumRtn)/252))-1

xMP_D_AnnRtn = xMP_D_AnnRtn.reset_index()
xMP_D_AnnRisk =
xMP_D[['CONS_pct_ch_D','MODCONS_pct_ch_D','MOD_pct_ch_D','MODGROW_pct_ch_D','GROW_pct_ch_D',

'MAXGROW_pct_ch_D','Current_pct_ch_D','New_pct_ch_D']].std().reset_index()
xMP_D_AnnRtn.rename(columns={0: 'AnnRtn'},inplace=True)
xMP_D_AnnRisk.rename(columns={0: 'AnnRisk'},inplace=True)
xMP_D_AnnRisk['AnnRisk']=xMP_D_AnnRisk['AnnRisk']*np.sqrt(252)

xMP_D_AnnRtnRisk = pd.merge(xMP_D_AnnRtn,xMP_D_AnnRisk,on=['index'],how='left')
####################
Y = xDF_OLS_D[xY_col + '_pct_ch_D']
#xInd_Vars =
['SPXT_pct_ch_D','BondTR_pct_ch_D','CCY_pct_ch_D','COMM_pct_ch_D','USCredit_pct_ch_D','HYTR_pc
t_ch_D']
xInd_Vars =
['SPXT_pct_ch_D','BondTR_pct_ch_D','TIPS_pct_ch_D','CCY_pct_ch_D','COMM_pct_ch_D','USCredit_pc
t_ch_D','HYTR_pct_ch_D']
xInd_Vars = ['SPXT_pct_ch_D','BondTR_pct_ch_D']
xInd_Vars = ['RLG_pct_ch_D']

X = xDF_OLS_D[xInd_Vars]

xCorrelations_D = xDF_OLS_D[[xY_col + '_pct_ch_D']+xInd_Vars].corr().to_string()
f = open(xDir + 'xCorrelations_D_' + xY_col + '.txt','w')
f.write(xCorrelations_D + '\r\n')
f.close()

X = sm.add_constant(X)
model = sm.OLS(Y,X)
result = model.fit()
xOLS_Summary_D = result.summary()
xOLS_text = xOLS_Summary_D.as_text()

f = open(xDir + 'xOLS_D_' + xY_col + '.txt','w')
f.write(xOLS_text + '\r\n')
f.close()

# ########## calc annualized return Daily ##########
xCumRtn_Y = (1+Y).cumprod()
xAnnRtn_Y_D = (xCumRtn_Y[len(xCumRtn_Y)-1]/xCumRtn_Y[0])**(1/(len(xCumRtn_Y)/252))-1
xAnnRisk_Y_D = np.sqrt(252*Y.var())
# xMP_D_AnnRtnRisk=xMP_D_AnnRtnRisk.append({'index':'Current
Portfolio','AnnRtn':xAnnRtn_Y_D,'AnnRisk':xAnnRisk_Y_D}, ignore_index=True)
############################################
xVar_X = np.array(X.var())
xVar_Y = Y.var()
xCoef_sq = result.params**2
xVar_resid = result.resid.var()
xVar_CoefX = xCoef_sq * xVar_X
xDelta_var = xVar_Y - np.sum(xVar_CoefX) - xVar_resid
xDelta_varX = xDelta_var * xVar_CoefX / np.sum(xVar_CoefX)
xVar_X_adj = xVar_CoefX + xDelta_varX
xVar_X_adj_pct = xVar_X_adj / xVar_Y
xVar_resid_pct = xVar_resid / xVar_Y
print (xVar_X_adj_pct, xVar_resid_pct)
print(np.sum(xVar_X_adj_pct)+xVar_resid_pct)
```

```python
xVar_X_adj_pct = pd.DataFrame(xVar_X_adj_pct)
xVar_X_adj_pct.reset_index(inplace=True)

xVar_X_adj_pct.rename(columns={0: 'Risk_Concentration(%)'},inplace=True)
xVar_X_adj_pct.rename(columns={'index': 'Risk_Factor'},inplace=True)
xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Idiosyncratic', 'Risk_Concentration(%)':
xVar_resid_pct}, ignore_index=True)

xVar_X_adj_pct=xVar_X_adj_pct.loc[~xVar_X_adj_pct['Risk_Factor'].isin({'const'})]
xSum = xVar_X_adj_pct['Risk_Concentration(%)'].sum()
xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Sum', 'Risk_Concentration(%)': xSum},
ignore_index=True)
xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual StDev)',
'Risk_Concentration(%)': xAnnRisk_Y_D}, ignore_index=True)
xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual Rtn)',
'Risk_Concentration(%)': xAnnRtn_Y_D}, ignore_index=True)

xVar_X_adj_pct['Risk_Concentration(%)'] =
xVar_X_adj_pct['Risk_Concentration(%)'].astype(float).map("{:.2%}".format)
# for x in (result.tvalues.index):
#     if x=='const':
#         continue
#     else:
#         #print (x, result.tvalues[x])
#         if (abs(result.tvalues[x]) < 1.5):
#             xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor'] == x]['Risk_Exposure(%)'] = 'NA'
#             #print (x, xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor']==x]['Risk_Exposure(%)'])

xRisk_Exposure_D = xVar_X_adj_pct.to_string()

f = open(xDir + 'xRisk_Concentration_D_' + xY_col + '.txt','w')
f.write(xRisk_Exposure_D + '\r\n')
f.close()

##########################
xCols_pct_MP = xMP.columns[xMP.columns.str.contains(pat = '_pct_ch_D')]
xCols_pct_MP=xCols_pct_MP.insert(0,'DATE')
xMP_D = xMP[xCols_pct_MP].copy()
xMP_D = xMP_D.loc[(xMP_D['DATE']>=xStartDate_D) & (xMP_D['DATE']<=xEndDate_D)]

xMP_D_StDev = pd.DataFrame(xMP_D.std()*np.sqrt(252))
xMP_D_StDev.reset_index(inplace=True)

xThisName=''
i=0
for x in xMP_D_StDev['index']:
    i=i+1
    if (i<=2):
        continue
    xName = xMP_D_StDev['index'][i-1]
    xAnnStDev = xMP_D_StDev[0]
    if (np.sqrt(252*xVar_Y) > xAnnStDev[i-1]):
        xThisName=xName
    xPreviousName=xName

if (xThisName=='MAXGROW_pct_ch_D'):
    xThisName = 'Accessive Growth Risk'
elif (xThisName == 'GROW_pct_ch_D'):
    xThisName = 'Growth Risk'
```

```python
    elif (xThisName == 'MODGROW_pct_ch_D'):
        xThisName = 'Moderate Growth Risk'
    elif (xThisName=='MOD_pct_ch_D'):
        xThisName = 'Moderate Risk'
    elif (xThisName=='MODCONS_pct_ch_D'):
        xThisName = 'Moderate Conservative Risk'
    elif (xThisName=='CONS_pct_ch_D'):
        xThisName = 'Conservative Risk'

    ################### MONTHLY #################
    ##################################################
    xDF_M['SPXT_pct_ch_M']=xDF_M['SPXT'].pct_change()
    xDF_M['BondTR_pct_ch_M']=xDF_M['BondTR'].pct_change()
    xDF_M['AAPL_pct_ch_M']=xDF_M['AAPL'].pct_change()
    xDF_M['AGG_pct_ch_M']=xDF_M['AGG'].pct_change()
    xDF_M['CCY_pct_ch_M']=xDF_M['CCY'].pct_change()
    xDF_M['COMM_pct_ch_M']=xDF_M['COMM'].pct_change()
    xDF_M['CREDIT_pct_ch_M']=xDF_M['CREDIT'].pct_change()
    xDF_M['FTLS_pct_ch_M']=xDF_M['FTLS'].pct_change()
    xDF_M['HFRIEMNI_pct_ch_M']=xDF_M['HFRIEMNI'].pct_change()
    xDF_M['PRBAX_pct_ch_M']=xDF_M['PRBAX'].pct_change()
    xDF_M['PRWAX_pct_ch_M']=xDF_M['PRWAX'].pct_change()
    xDF_M['SPLPEQTY_pct_ch_M']=xDF_M['SPLPEQTY'].pct_change()
    xDF_M['SPX_pct_ch_M']=xDF_M['SPX'].pct_change()
    xDF_M['SPY_pct_ch_M']=xDF_M['SPY'].pct_change()
    xDF_M['TSLA_pct_ch_M']=xDF_M['TSLA'].pct_change()
    xDF_M['US3M_pct_ch_M']=xDF_M['US3M'].pct_change()
    xDF_M['US10Y_pct_ch_M']=xDF_M['US10Y'].pct_change()
    xDF_M['HYG_pct_ch_M']=xDF_M['HYG'].pct_change()
    xDF_M['HYTR_pct_ch_M']=xDF_M['HYTR'].pct_change()
    xDF_M['RealBondTR_pct_ch_M']=xDF_M['RealBondTR'].pct_change()
    xDF_M['CPI_pct_ch_M']=xDF_M['CPI'].pct_change()
    xDF_M['CPI_pct_ch_Y']=xDF_M['CPI'].pct_change(12)
    xDF_M['TIPS_pct_ch_M']=xDF_M['TIPS'].pct_change()
    xDF_M['GMWAX_pct_ch_M']=xDF_M['GMWAX'].pct_change()
    xDF_M['CashConst_pct_ch_M']=xDF_M['CashConst'].pct_change()
    xDF_M['S5INFT_pct_ch_M']=xDF_M['S5INFT'].pct_change()
    xDF_M['7030TR_pct_ch_M']=xDF_M['7030TR'].pct_change()
    xDF_M['USCredit_pct_ch_M']=xDF_M['USCredit'].pct_change()
    xDF_M['SHY_pct_ch_M']=xDF_M['SHY'].pct_change()
    xDF_M['TIP_pct_ch_M']=xDF_M['TIP'].pct_change()
    xDF_M['GOOG_pct_ch_M']=xDF_M['GOOG'].pct_change()

    xDF_M['Inflation_pct_ch_M'] = xDF_M['BondTR_pct_ch_M'] - xDF_M['TIPS_pct_ch_M']

    ############### overwrite to create the EXACT 70/30 returns #############
    xDF_M['7030TR_pct_ch_M']=0.7*xDF_M['SPXT_pct_ch_M']+0.3*xDF_M['BondTR_pct_ch_M']
    xDF_M['3070TR_pct_ch_M']=0.3*xDF_M['SPXT_pct_ch_M']+0.7*xDF_M['BondTR_pct_ch_M']
    #############################################

    xY_col='3070TR' # for Monthly ony here!

    xW1=.75
    xW2=0.25
    xDF_M['New_pct_ch_M']=xDF_M['NewPort'].pct_change()
    xDF_M['New_pct_ch_M']=0.2*xDF_M['SPXT_pct_ch_M']+0.60*xDF_M['BondTR_pct_ch_M']+0.2*xDF_M['HFRI
    EMNI_pct_ch_M']

    xRiskFactorSet_M=['SPXT_pct_ch_M','BondTR_pct_ch_M','CCY_pct_ch_M','COMM_pct_ch_M','USCredit_p
    ct_ch_M','HYTR_pct_ch_M',
```

```
                  'CPI_pct_ch_M','TIPS_pct_ch_M','Inflation_pct_ch_M','RealBondTR_pct_ch_M']

xRiskFactorSet_M=['SPXT_pct_ch_M','TIPS_pct_ch_M','USCredit_pct_ch_M','CPI_pct_ch_M','COMM_pct
_ch_M']
#xRiskFactorSet_M=['SPXT_pct_ch_M','TIPS_pct_ch_M','USCredit_pct_ch_M','CPI_pct_ch_M','COMM_pc
t_ch_M','HYTR_pct_ch_M']

xRiskFactorSet_M=['SPXT_pct_ch_M','TIPS_pct_ch_M','USCredit_pct_ch_M','COMM_pct_ch_M']

xDescriptive_M=xDF_M[[xY_col+'_pct_ch_M']+xRiskFactorSet_M].describe(include='all').to_string(
)
xCorrelations_M=xDF_M[[xY_col+'_pct_ch_M']+xRiskFactorSet_M].corr().to_string()

xDescriptive_M = xDescriptive_M + '\n\n' + xCorrelations_M
f = open(xDir + 'xDescriptive_M_'+xY_col+'.txt','w')
f.write(xDescriptive_M + '\r\n')
f.close()

##############################
################### (ROLLING here) QUARTERLY #################
##################################################
xDF_M['SPXT_pct_ch_Q']=xDF_M['SPXT'].pct_change(3)
xDF_M['BondTR_pct_ch_Q']=xDF_M['BondTR'].pct_change(3)
xDF_M['AAPL_pct_ch_Q']=xDF_M['AAPL'].pct_change(3)
xDF_M['AGG_pct_ch_Q']=xDF_M['AGG'].pct_change(3)
xDF_M['CCY_pct_ch_Q']=xDF_M['CCY'].pct_change(3)
xDF_M['COMM_pct_ch_Q']=xDF_M['COMM'].pct_change(3)
xDF_M['CREDIT_pct_ch_Q']=xDF_M['CREDIT'].pct_change(3)
xDF_M['FTLS_pct_ch_Q']=xDF_M['FTLS'].pct_change(3)
xDF_M['HFRIEMNI_pct_ch_Q']=xDF_M['HFRIEMNI'].pct_change(3)
xDF_M['PRBAX_pct_ch_Q']=xDF_M['PRBAX'].pct_change(3)
xDF_M['PRWAX_pct_ch_Q']=xDF_M['PRWAX'].pct_change(3)
xDF_M['SPLPEQTY_pct_ch_Q']=xDF_M['SPLPEQTY'].pct_change(3)
xDF_M['SPX_pct_ch_Q']=xDF_M['SPX'].pct_change(3)
xDF_M['SPY_pct_ch_Q']=xDF_M['SPY'].pct_change(3)
xDF_M['TSLA_pct_ch_Q']=xDF_M['TSLA'].pct_change(3)
xDF_M['US3M_pct_ch_Q']=xDF_M['US3M'].pct_change(3)
xDF_M['US10Y_pct_ch_Q']=xDF_M['US10Y'].pct_change(3)
xDF_M['HYG_pct_ch_Q']=xDF_M['HYG'].pct_change(3)
xDF_M['HYTR_pct_ch_Q']=xDF_M['HYTR'].pct_change(3)
xDF_M['RealBondTR_pct_ch_Q']=xDF_M['RealBondTR'].pct_change(3)
xDF_M['CPI_pct_ch_Q']=xDF_M['CPI'].pct_change(3)
xDF_M['TIPS_pct_ch_Q']=xDF_M['TIPS'].pct_change(3)
xDF_M['GMWAX_pct_ch_Q']=xDF_M['GMWAX'].pct_change(3)
xDF_M['CashConst_pct_ch_Q']=xDF_M['CashConst'].pct_change(3)
xDF_M['S5INFT_pct_ch_Q']=xDF_M['S5INFT'].pct_change(3)
xDF_M['7030TR_pct_ch_Q']=xDF_M['7030TR'].pct_change(3)
xDF_M['USCredit_pct_ch_Q']=xDF_M['USCredit'].pct_change(3)
xDF_M['SHY_pct_ch_Q']=xDF_M['SHY'].pct_change(3)
xDF_M['TIP_pct_ch_Q']=xDF_M['TIP'].pct_change(3)
xDF_M['GOOG_pct_ch_Q']=xDF_M['GOOG'].pct_change(3)

xDF_M['3070TR_pct_ch_Q']=0.3*xDF_M['SPXT_pct_ch_Q']+0.7*xDF_M['BondTR_pct_ch_Q']
xDF_M['New_pct_ch_Q']=xDF_M['NewPort'].pct_change(3)
######################
###xDF_OLS_M = xDF_M.copy()
xDF_OLS_M = xDF_M[['DATE'] + xRiskFactorSet_M +
[xY_col+'_pct_ch_M','CPI_pct_ch_Y','New_pct_ch_M','HFRIEMNI_pct_ch_M']].copy()
###################
xDF_OLS_M.dropna(inplace=True)
```

```
xDF_OLS_M.reset_index(drop=True,inplace=True)
##################
xStartDate_M = xDF_OLS_M['DATE'].min()
xEndDate_M = xDF_OLS_M['DATE'].max()
#################### model portfolios for Monthly returns ############
xMP_M = xDF_OLS_M[['DATE']].copy()
xMP_M = pd.merge(xMP_M,
xMP_MQ[['DATE','CONS_pct_ch_M','MODCONS_pct_ch_M','MOD_pct_ch_M','MODGROW_pct_ch_M','GROW_pct_
ch_M',
                'MAXGROW_pct_ch_M','Current_pct_ch_M','New_pct_ch_M']], on=['DATE'],
how='left')
xMP_M_CumRtn =
(1+xMP_M[['CONS_pct_ch_M','MODCONS_pct_ch_M','MOD_pct_ch_M','MODGROW_pct_ch_M','GROW_pct_ch_M'
,
                         'MAXGROW_pct_ch_M','Current_pct_ch_M','New_pct_ch_M']]).cumprod()
xMP_M_AnnRtn = (xMP_M_CumRtn.iloc[len(xMP_M_CumRtn)-
1]/xMP_M_CumRtn.iloc[0])**(1/(len(xMP_M_CumRtn)/12))-1

xMP_M_AnnRtn = xMP_M_AnnRtn.reset_index()
xMP_M_AnnRisk =
xMP_M[['CONS_pct_ch_M','MODCONS_pct_ch_M','MOD_pct_ch_M','MODGROW_pct_ch_M','GROW_pct_ch_M','M
AXGROW_pct_ch_M',
                       'Current_pct_ch_M','New_pct_ch_M']].std().reset_index()
xMP_M_AnnRtn.rename(columns={0: 'AnnRtn'},inplace=True)
xMP_M_AnnRisk.rename(columns={0: 'AnnRisk'},inplace=True)
xMP_M_AnnRisk['AnnRisk']=xMP_M_AnnRisk['AnnRisk']*np.sqrt(12)

xMP_M_AnnRtnRisk = pd.merge(xMP_M_AnnRtn,xMP_M_AnnRisk,on=['index'],how='left')

import time
from datetime import timedelta
#start_time = time.monotonic()
#end_time = time.monotonic()

############## OLS MONHTLY ###################
xVersion=['Current','New']
for x in xVersion:
    if x=='Current':
        ##xY_col = 'HFRIEMNI' #### special test!!!!
        Y = xDF_OLS_M[xY_col + '_pct_ch_M']
        xY_col2 = xY_col
    elif x=='New':
        Y = xDF_OLS_M['New_pct_ch_M']
        xY_col2 = 'New'
    #xInd_Vars =
['SPXT_pct_ch_M','BondTR_pct_ch_M','CCY_pct_ch_M','COMM_pct_ch_M','USCredit_pct_ch_M','HYTR_pc
t_ch_M']
    #xInd_Vars =
['SPXT_pct_ch_M','RealBondTR_pct_ch_M','TIPS_pct_ch_M','CPI_pct_ch_Y','CCY_pct_ch_M','COMM_pct
_ch_M','USCredit_pct_ch_M','HYTR_pct_ch_M']
    xInd_Vars = ['SPXT_pct_ch_M', 'Inflation_pct_ch_M', 'TIPS_pct_ch_M', 'CCY_pct_ch_M',
'USCredit_pct_ch_M']
    xInd_Vars = ['SPXT_pct_ch_M', 'Inflation_pct_ch_M', 'TIPS_pct_ch_M', 'CCY_pct_ch_M',
'USCredit_pct_ch_M']
    xInd_Vars = ['SPXT_pct_ch_M']
    xInd_Vars = ['SPXT_pct_ch_M','BondTR_pct_ch_M']
    xInd_Vars =
['SPXT_pct_ch_M','RealBondTR_pct_ch_M','TIPS_pct_ch_M','CPI_pct_ch_Y','CCY_pct_ch_M','COMM_pct
_ch_M','USCredit_pct_ch_M','HYTR_pct_ch_M']
    ##xInd_Vars = ['S5INFT_pct_ch_M']
```

```python
    #xInd_Vars = ['SPXT_pct_ch_M', 'BondTR_pct_ch_M', 'TIPS_pct_ch_M', 'CCY_pct_ch_M',
'USCredit_pct_ch_M']

    xInd_Vars = xRiskFactorSet_M
    xInd_Vars = ['SPXT_pct_ch_M']

    X = xDF_OLS_M[xInd_Vars]
    #X =
xDF_OLS_M[['SPXT_pct_ch_M','BondTR_pct_ch_M','CCY_pct_ch_M','COMM_pct_ch_M','USCredit_pct_ch_M
']]

    xCorrelations_M = xDF_OLS_M[[xY_col2 + '_pct_ch_M']+xInd_Vars].corr().to_string()
    f = open(xDir + 'xCorrelations_M_' + xY_col + '_'+ x +'.txt','w')
    f.write(xCorrelations_M + '\r\n')
    f.close()

    X = sm.add_constant(X)
    xStart_time = datetime.datetime.now() #time.time_ns()*1000000
    model = sm.OLS(Y,X)
    result = model.fit()
    # for i in range(1,9999):
    #     print(i)
    xEnd_time = datetime.datetime.now() #time.time_ns()*1000000
    globals()['xSecond_M_' + x] = 'Start: ' + (str)(xStart_time) + '; End: ' +
(str)(xEnd_time) + '; Duration: ' + (
        str)((xEnd_time - xStart_time))
    xOLS_Summary_M = result.summary()
    xOLS_text = xOLS_Summary_M.as_text()

    f = open(xDir + 'xOLS_M_' + xY_col + '_' + x + '.txt','w')
    f.write(globals()['xSecond_M_' + x] +'\n\n' + xOLS_text + '\r\n')
    f.close()

    ########## calc annualized return from Monthly returns ##########
    xCumRtn_Y = (1+Y).cumprod()
    xAnnRtn_Y_M = (xCumRtn_Y[len(xCumRtn_Y)-1]/xCumRtn_Y[0])**(1/(len(xCumRtn_Y)/12))-1
    xAnnRisk_Y_M = np.sqrt(12*Y.var())
    #xMP_M_AnnRtnRisk=xMP_M_AnnRtnRisk.append({'index':'Current
Portfolio','AnnRtn':xAnnRtn_Y_M,'AnnRisk':xAnnRisk_Y_M}, ignore_index=True)

    ############################################

    xVar_X = np.array(X.var())
    xVar_Y = Y.var()
    xCoef_sq = result.params**2
    xVar_resid = result.resid.var()
    xVar_CoefX = xCoef_sq * xVar_X
    xDelta_var = xVar_Y - np.sum(xVar_CoefX) - xVar_resid
    xDelta_varX = xDelta_var * xVar_CoefX / np.sum(xVar_CoefX)
    xVar_X_adj = xVar_CoefX + xDelta_varX
    xVar_X_adj_pct = xVar_X_adj / xVar_Y
    xVar_resid_pct = xVar_resid / xVar_Y
    print (xVar_X_adj_pct, xVar_resid_pct)
    print(np.sum(xVar_X_adj_pct)+xVar_resid_pct)

    xVar_X_adj_pct = pd.DataFrame(xVar_X_adj_pct)
    xVar_X_adj_pct.reset_index(inplace=True)

    xVar_X_adj_pct.rename(columns={0: 'Risk_Exposure(%)'},inplace=True)
    xVar_X_adj_pct.rename(columns={'index': 'Risk_Factor'},inplace=True)
```

```
    xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Idiosyncratic', 'Risk_Exposure(%)':
xVar_resid_pct}, ignore_index=True)

    xVar_X_adj_pct=xVar_X_adj_pct.loc[~xVar_X_adj_pct['Risk_Factor'].isin({'const'})]
    xSum = xVar_X_adj_pct['Risk_Exposure(%)'].sum()
    xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Sum', 'Risk_Exposure(%)': xSum},
ignore_index=True)
    xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual StDev)',
'Risk_Exposure(%)': xAnnRisk_Y_M}, ignore_index=True)
    xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual Rtn)',
'Risk_Exposure(%)': xAnnRtn_Y_M}, ignore_index=True)

    xVar_X_adj_pct['Risk_Exposure(%)'] =
xVar_X_adj_pct['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)
    # for x in (result.tvalues.index):
    #     if x=='const':
    #         continue
    #     else:
    #         #print (x, result.tvalues[x])
    #         if (abs(result.tvalues[x]) <1.5):
    #             xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor'] == x]['Risk_Exposure(%)'] =
'NA'
    #             #print (x,
xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor']==x]['Risk_Exposure(%)'])

    xRisk_Exposure_M = xVar_X_adj_pct.to_string()
    #xRisk_Exposure_M.to_csv (xDir + 'xRisk_Exposure_M.txt')

    f = open(xDir + 'xRisk_Concentration_M_' + xY_col + '_'+ x + '.txt','w')
    f.write(xRisk_Exposure_M + '\r\n')
    f.close()
    ################## the following is working on StdDev monthly ###############
    ############### the following is working on the Std Dev (RISK) MONTHLY #######
    xStdDev_X = np.array(X.std()) * np.sqrt(12)
    xStdDev_Y = Y.std() * np.sqrt(12)
    xCoef = result.params.abs()
    xStdDev_resid = result.resid.std() * np.sqrt(12)
    xStdDev_CoefX = xCoef * xStdDev_X
    xDelta_StdDev = xStdDev_Y - np.sum(xStdDev_CoefX) - xStdDev_resid
    ####### debug ########
    print('Monthly '+x+': xDelta_StdDev = ', xDelta_StdDev,'\n')
    print('Monthly '+x+': xStdDev_Y = ', xStdDev_Y,'\n')
    print('Monthly '+x+': xStdDev_X = ', xStdDev_X,'\n')
    print('Monthly '+x+': xStdDev_CoefX = ', xStdDev_CoefX,'\n')
    print('Monthly '+x+': xStdDev_resid = ', xStdDev_resid,'\n')
    ######################
    xAdj_StdDev_resid = False
    if (xAdj_StdDev_resid == False):
        xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / np.sum(xStdDev_CoefX)
    else:
        xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / (np.sum(xStdDev_CoefX) +
xStdDev_resid)
        xStdDev_resid = xStdDev_resid + xDelta_StdDev * xStdDev_resid / (np.sum(xStdDev_CoefX)
+ xStdDev_resid)
    xStdDev_X_adj = xStdDev_CoefX + xDelta_StdDevX

    xStdDev_X_adj = pd.DataFrame(xStdDev_X_adj)
    xStdDev_X_adj.reset_index(inplace=True)

    xStdDev_X_adj.rename(columns={0: 'Risk_Exposure(%)'},inplace=True)
```

```
    xStdDev_X_adj.rename(columns={'index': 'Risk_Factor'},inplace=True)
    xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Idiosyncratic', 'Risk_Exposure(%)':
xStdDev_resid}, ignore_index=True)

    xStdDev_X_adj=xStdDev_X_adj.loc[~xStdDev_X_adj['Risk_Factor'].isin({'const'})]
    xSum = xStdDev_X_adj['Risk_Exposure(%)'].sum()
    xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sum', 'Risk_Exposure(%)': xSum},
ignore_index=True)
    xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual StDev)',
'Risk_Exposure(%)': xAnnRisk_Y_M}, ignore_index=True)
    xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual Rtn)',
'Risk_Exposure(%)': xAnnRtn_Y_M}, ignore_index=True)

    #xStdDev_X_adj['Risk_Exposure(%)'] =
xStdDev_X_adj['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)

    xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sharpe Ratio (Rtn/Risk)',
'Risk_Exposure(%)': np.round(xAnnRtn_Y_M / xAnnRisk_Y_M,2)}, ignore_index=True)

    if x== 'Current':
        xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (' + xY_col + ')'},
inplace=True)
    else:
        xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (proposed)'},
inplace=True)
    globals()['xRisk_exposures_' + x] = xStdDev_X_adj

    xIndex_StdDev=globals()['xRisk_exposures_' + x][
        globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names + '(Annual
StDev)'].index.values[0]
    xIndex_Rtn = globals()['xRisk_exposures_' + x][
        globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names + '(Annual
Rtn)'].index.values[0]
    globals()['xRisk_exposures_' + x].loc[globals()['xRisk_exposures_' + x].index ==
xIndex_StdDev, 'Risk_Factor'] = 'Annual StdDev'
    globals()['xRisk_exposures_' + x].loc[
        globals()['xRisk_exposures_' + x].index == xIndex_Rtn, 'Risk_Factor'] = 'Annual Rtn'

    globals()['xIdio_exp_' + x] = xStdDev_resid / xSum

    if x=='New':  # second time and last time!
        xStdDev_X_adj = pd.merge(xRisk_exposures_Current, xRisk_exposures_New,
on='Risk_Factor', how='left')

    xRisk_Exposure_M_StdDev = xStdDev_X_adj.to_string()

    ######################################################################
    #xRisk_Exposure_Y.to_csv (xDir + 'xRisk_Exposure_Y.txt')

    f = open(xDir + 'xRisk_Exposure_M_' + xY_col +  '_' + x + '.txt','w')
    f.write(xRisk_Exposure_M_StdDev + '\r\n')
    f.close()
######################
xStdDev_X_M = pd.DataFrame(X.std() * np.sqrt(12)).reset_index()
xStdDev_X_M.rename(columns={0:'Risk_Factor_AnnStdDev'}, inplace=True)
xStdDev_X_M.rename(columns={'index':'Risk_Factor'}, inplace=True)
xStdDev_X_adj = pd.merge(xStdDev_X_adj,xStdDev_X_M, on=['Risk_Factor'],how='left')
xStdDev_X_adj['Risk_Exp (Current)'] = xStdDev_X_adj['Current Risk
('+xY_col+')']/xStdDev_X_adj['Risk_Factor_AnnStdDev']
xStdDev_X_adj['Risk_Exp (New)'] = xStdDev_X_adj['New Risk
```

```
(proposed)']/xStdDev_X_adj['Risk_Factor_AnnStdDev']
xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Idiosyncratic','Risk_Exp
(Current)']=xIdio_exp_Current
xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Idiosyncratic','Risk_Exp
(New)']=xIdio_exp_New
xSum_Risk_Exp_Current = xStdDev_X_adj['Risk_Exp (Current)'].sum()
xSum_Risk_Exp_New = xStdDev_X_adj['Risk_Exp (New)'].sum()
xStdDev_X_adj['Risk_Exp (Current)'] = xStdDev_X_adj['Risk_Exp
(Current)']/xSum_Risk_Exp_Current
xStdDev_X_adj['Risk_Exp (New)'] = xStdDev_X_adj['Risk_Exp (New)']/xSum_Risk_Exp_New
xSum_Risk_Exp_Current = xStdDev_X_adj['Risk_Exp (Current)'].sum()
xSum_Risk_Exp_New = xStdDev_X_adj['Risk_Exp (New)'].sum()
xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Sum','Risk_Exp
(Current)']=xSum_Risk_Exp_Current
xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Sum','Risk_Exp (New)']=xSum_Risk_Exp_New

xPart_1=xStdDev_X_adj.loc[xStdDev_X_adj.index<len(xStdDev_X_adj)-1]
xPart_2=xStdDev_X_adj.loc[xStdDev_X_adj.index==len(xStdDev_X_adj)-1]

xPart_1['Current Risk ('+xY_col+')'] = xPart_1['Current Risk
('+xY_col+')'].astype(float).map("{:.2%}".format)
xPart_1['New Risk (proposed)'] = xPart_1['New Risk
(proposed)'].astype(float).map("{:.2%}".format)
xPart_1['Risk_Factor_AnnStdDev'] =
xPart_1['Risk_Factor_AnnStdDev'].astype(float).map("{:.2%}".format)
xPart_1['Risk_Exp (Current)'] = xPart_1['Risk_Exp
(Current)'].astype(float).map("{:.2%}".format)
xPart_1['Risk_Exp (New)'] = xPart_1['Risk_Exp (New)'].astype(float).map("{:.2%}".format)

xStdDev_X_adj=xPart_1.append(xPart_2, ignore_index=True)

xStdDev_X_adj = xStdDev_X_adj.replace({'nan%': ''})
xStdDev_X_adj = xStdDev_X_adj.replace({np.nan: ''})

xRisk_Exposure_M_StdDev = xStdDev_X_adj.to_string()
#xStdDev_X_adj['Risk_Exposure(%)'] =
xStdDev_X_adj['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)
######################
f = open(xDir + 'xRisk_Exposure_M_' + xY_col + '.txt','w')
f.write('From '+xStartDate_M.strftime('%Y-%m-%d') +' to ' + xEndDate_M.strftime('%Y-%m-%d') +
'\n\n' + xRisk_Exposure_M_StdDev + '\r\n')
f.close()
#


########################## QUARTERLY ######################
######################
xDF_OLS_Q = xDF_M.copy()
####################
xNoRolling_Q = True  #True
if (xNoRolling_Q):
    xDF_OLS_Q = xDF_OLS_Q.loc[xDF_OLS_Q['diff_Q'].isin({-1,3})]
    xDF_OLS_Q = xDF_OLS_Q.loc[xDF_OLS_Q['diff_Q'].notnull()]
###################
xDF_OLS_Q.dropna(inplace=True)
xDF_OLS_Q.reset_index(drop=True,inplace=True)
######################
xStartDate_Q = xDF_OLS_Q['DATE'].min()
xEndDate_Q = xDF_OLS_Q['DATE'].max()
#################### model portfolios for Quarterly returns ############
```

```
xMP_Q = xDF_OLS_Q[['DATE']].copy()
xMP_Q = pd.merge(xMP_Q,
xMP_MQ[['DATE','CONS_pct_ch_Q','MODCONS_pct_ch_Q','MOD_pct_ch_Q','MODGROW_pct_ch_Q','GROW_pct_
ch_Q',
                'MAXGROW_pct_ch_Q','Current_pct_ch_Q','New_pct_ch_Q']], on=['DATE'],
how='left')
xMP_Q_CumRtn =
(1+xMP_Q[['CONS_pct_ch_Q','MODCONS_pct_ch_Q','MOD_pct_ch_Q','MODGROW_pct_ch_Q','GROW_pct_ch_Q'
,
                         'MAXGROW_pct_ch_Q','Current_pct_ch_Q','New_pct_ch_Q']]).cumprod()
xMP_Q_AnnRtn = (xMP_Q_CumRtn.iloc[len(xMP_Q_CumRtn)-
1]/xMP_Q_CumRtn.iloc[0])**(1/(len(xMP_Q_CumRtn)/4))-1

xMP_Q_AnnRtn = xMP_Q_AnnRtn.reset_index()
xMP_Q_AnnRisk =
xMP_Q[['CONS_pct_ch_Q','MODCONS_pct_ch_Q','MOD_pct_ch_Q','MODGROW_pct_ch_Q','GROW_pct_ch_Q',

'MAXGROW_pct_ch_Q','Current_pct_ch_Q','New_pct_ch_Q']].std().reset_index()
xMP_Q_AnnRtn.rename(columns={0: 'AnnRtn'},inplace=True)
xMP_Q_AnnRisk.rename(columns={0: 'AnnRisk'},inplace=True)
xMP_Q_AnnRisk['AnnRisk']=xMP_Q_AnnRisk['AnnRisk']*np.sqrt(4)

xMP_Q_AnnRtnRisk = pd.merge(xMP_Q_AnnRtn,xMP_Q_AnnRisk,on=['index'],how='left')
####################

Y = xDF_OLS_Q[xY_col + '_pct_ch_Q']
xInd_Vars =
['SPXT_pct_ch_Q','RealBondTR_pct_ch_Q','TIPS_pct_ch_Q','CPI_pct_ch_Y','CCY_pct_ch_Q','COMM_pct
_ch_Q','USCredit_pct_ch_Q','HYTR_pct_ch_Q']
#xInd_Vars =
['SPXT_pct_ch_Q','TIPS_pct_ch_Q','CPI_pct_ch_Y','CCY_pct_ch_Q','COMM_pct_ch_Q','USCredit_pct_c
h_Q','HYTR_pct_ch_Q']
xInd_Vars = ['SPXT_pct_ch_Q','BondTR_pct_ch_Q']
X = xDF_OLS_Q[xInd_Vars]

xCorrelations_Q = xDF_OLS_Q[[xY_col + '_pct_ch_Q']+xInd_Vars].corr().to_string()
f = open(xDir + 'xCorrelations_Q_' + xY_col + '.txt','w')
f.write(xCorrelations_Q + '\r\n')
f.close()

X = sm.add_constant(X)
model = sm.OLS(Y,X)
result = model.fit()
xOLS_Summary_Q = result.summary()
xOLS_text = xOLS_Summary_Q.as_text()

f = open(xDir + 'xOLS_Q_' + xY_col + '.txt','w')
f.write(xOLS_text + '\r\n')
f.close()

########## calc annualized return from Monthly returns #########
xCumRtn_Y = (1+Y).cumprod()
xAnnRtn_Y_Q = (xCumRtn_Y[len(xCumRtn_Y)-1]/xCumRtn_Y[0])**(1/(len(xCumRtn_Y)/4))-1
xAnnRisk_Y_Q = np.sqrt(4*Y.var())
#xMP_Q_AnnRtnRisk=xMP_Q_AnnRtnRisk.append({'index':'Current
Portfolio','AnnRtn':xAnnRtn_Y_Q,'AnnRisk':xAnnRisk_Y_Q}, ignore_index=True)
############################################

xVar_X = np.array(X.var())
xVar_Y = Y.var()
```

```
xCoef_sq = result.params**2
xVar_resid = result.resid.var()
xVar_CoefX = xCoef_sq * xVar_X
xDelta_var = xVar_Y - np.sum(xVar_CoefX) - xVar_resid
xDelta_varX = xDelta_var * xVar_CoefX / np.sum(xVar_CoefX)
xVar_X_adj = xVar_CoefX + xDelta_varX
xVar_X_adj_pct = xVar_X_adj / xVar_Y
xVar_resid_pct = xVar_resid / xVar_Y
print (xVar_X_adj_pct, xVar_resid_pct)
print(np.sum(xVar_X_adj_pct)+xVar_resid_pct)

xVar_X_adj_pct = pd.DataFrame(xVar_X_adj_pct)
xVar_X_adj_pct.reset_index(inplace=True)

xVar_X_adj_pct.rename(columns={0: 'Risk_Exposure(%)'},inplace=True)
xVar_X_adj_pct.rename(columns={'index': 'Risk_Factor'},inplace=True)
xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Idiosyncratic', 'Risk_Exposure(%)':
xVar_resid_pct}, ignore_index=True)

xVar_X_adj_pct=xVar_X_adj_pct.loc[~xVar_X_adj_pct['Risk_Factor'].isin({'const'})]
xSum = xVar_X_adj_pct['Risk_Exposure(%)'].sum()
xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Sum', 'Risk_Exposure(%)': xSum},
ignore_index=True)
xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual StDev)',
'Risk_Exposure(%)': xAnnRisk_Y_Q}, ignore_index=True)
xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual Rtn)',
'Risk_Exposure(%)': xAnnRtn_Y_Q}, ignore_index=True)

xVar_X_adj_pct['Risk_Exposure(%)'] =
xVar_X_adj_pct['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)
# for x in (result.tvalues.index):
#     if x=='const':
#         continue
#     else:
#         #print (x, result.tvalues[x])
#         if (abs(result.tvalues[x]) <1.5):
#             xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor'] == x]['Risk_Exposure(%)'] = 'NA'
#             #print (x, xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor']==x]['Risk_Exposure(%)'])

xRisk_Exposure_Q = xVar_X_adj_pct.to_string()
#xRisk_Exposure_M.to_csv (xDir + 'xRisk_Exposure_M.txt')

f = open(xDir + 'xRisk_Exposure_Q_' + xY_col + '.txt','w')
f.write(xRisk_Exposure_Q + '\r\n')
f.close()

#### Scatter plots for Current Portfolio vs 6 Model Portfolios (Using Daily, Monthly,
Qquarterly and Annual Rtns #####
import matplotlib.pyplot as plt

xFreq = ''
for k in range(0,4):
    if k==0:
        xRtn_Risk_Name = 'xMP_D_AnnRtnRisk'
        xFreq = '(using daily data)'
        xStartDate = xStartDate_D
        xEndDate = xEndDate_D

    elif k == 1:
        xRtn_Risk_Name = 'xMP_M_AnnRtnRisk'
```

```
        xFreq = '(using monthly data)'
        xStartDate = xStartDate_M
        xEndDate = xEndDate_M
    elif k == 2:
        xRtn_Risk_Name = 'xMP_Q_AnnRtnRisk'
        xFreq = '(using quarterly data)'
        xStartDate = xStartDate_Q
        xEndDate = xEndDate_Q
    elif k == 3:
        xRtn_Risk_Name = 'xMP_Y_AnnRtnRisk'
        xFreq = '(using annual data)'
        xStartDate = xStartDate_Y
        xEndDate = xEndDate_Y

    xMP_Rtn_Risk=globals()[xRtn_Risk_Name].copy()
    #xMP_Rtn_Risk=xMP_M_AnnRtnRisk.copy()

    xMP_name = pd.DataFrame()
    xMP_name['name']=''
    xMP_name['Rtn_Risk']=''
    xRtn= xMP_Rtn_Risk['AnnRtn'][0]
    xRisk= xMP_Rtn_Risk['AnnRisk'][0]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Conservative','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][1]
    xRisk= xMP_Rtn_Risk['AnnRisk'][1]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Moderate Conservative','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][2]
    xRisk= xMP_Rtn_Risk['AnnRisk'][2]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Moderate','Rtn_Risk': xRtn_Risk}, ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][3]
    xRisk= xMP_Rtn_Risk['AnnRisk'][3]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Moderate Growth','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][4]
    xRisk= xMP_Rtn_Risk['AnnRisk'][4]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Growth','Rtn_Risk': xRtn_Risk}, ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][5]
    xRisk= xMP_Rtn_Risk['AnnRisk'][5]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Maximum Growth','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][6]
    xRisk= xMP_Rtn_Risk['AnnRisk'][6]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Current Portfoio','Rtn_Risk': xRtn_Risk},
ignore_index=True)
```

```
    xRtn= xMP_Rtn_Risk['AnnRtn'][7]
    xRisk= xMP_Rtn_Risk['AnnRisk'][7]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'New Portfoio','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    #######
    xMP_Rtn_Risk['Lable']=''
    xMP_Rtn_Risk['Rtn_Risk']=''
    i=0
    for x in xMP_name['name']:
        xMP_Rtn_Risk['Lable'][i]=xMP_name['name'][i]
        xMP_Rtn_Risk['Rtn_Risk'][i]=xMP_name['Rtn_Risk'][i]
        i=i+1
    ##############
    x = xMP_Rtn_Risk['AnnRisk'].values
    y = xMP_Rtn_Risk['AnnRtn'].values
    #types = xMP_Rtn_Risk.reset_index()['index'].values
    #types = xMP_Rtn_Risk['index'].values
    types = xMP_Rtn_Risk['Lable'].values

    fig, ax = plt.subplots()
    #ax.plot(risks, returns, color='red', label='Equity/Bond')      # this is a line
(efficient frontier)
    xSubText = xFreq + ' from ' + xStartDate.strftime('%m/%d/%Y') + ' to ' +
xEndDate.strftime('%m/%d/%Y')
    fig.suptitle('Return and Risk of the Current Portfolio (' + xY_col+') vs Model Portfolios
\n' + xSubText, fontsize=13,y=0.98)
    #ax.set_xlabel('Risk (Annualized Std)', fontsize=10)
    #ax.set_ylabel('Annualized Return', fontsize=10)

    #fig, ax = plt.subplots(figsize=(10,10))
    ax.scatter(x, y)

    ax.set_xlabel('Annualized Risk', fontsize=12)
    ax.set_ylabel('Annualized Return', fontsize=12)
    #ax.set_title('(Return and Risk) of the Current Portfolio vs Model Portfolios ' +
xSubText, fontsize=18)

    for i, txt in enumerate(types):
        ax.annotate(txt + '\n' + xMP_Rtn_Risk['Rtn_Risk'][i], (x[i], y[i]), xytext=(-18,-18),
textcoords='offset points',ha="left", size=8)
        #ax.annotate(txt + '\n', (x[i], y[i]), xytext=(10, 10), textcoords='offset points')
        plt.scatter(x, y, marker='o', color='blue')

    plt.savefig(xDir + xRtn_Risk_Name +'_'+xY_col+'.png')
    plt.show()

################## SI and SPXT and BondTR: Rolling Annual Returns and Calendar Monthly Returns
###########
xSI_Y =
xDF[['DATE','SPLPEQTY_pct_ch_Y','SI2_pct_ch_Y','SI4_pct_ch_Y','SI6_pct_ch_Y','SPXT_pct_ch_Y','
BondTR_pct_ch_Y']].copy()
xSI_Y.dropna(inplace=True)
xSI_Y.reset_index(drop=True,inplace=True)
xStartDate_Y_SI= xSI_Y['DATE'].min()
xEndDate_Y_SI= xSI_Y['DATE'].max()

xSI_Y_AnnStdDev=pd.DataFrame(xSI_Y.std())
xSI_Y_AnnStdDev.reset_index(inplace=True)
```

```
xSI_Y_AnnStdDev.rename(columns={0: 'AnnStdDev(%)'},inplace=True)
xSI_Y_AnnRtn=pd.DataFrame(xSI_Y.mean())
xSI_Y_AnnRtn.reset_index(inplace=True)
xSI_Y_AnnRtn.rename(columns={0: 'AnnRtn(%)'},inplace=True)
xSI_Y_RtnRisk = pd.merge(xSI_Y_AnnRtn,xSI_Y_AnnStdDev,on=['index'],how='left')
xSI_Y_RtnRisk['Sharpe Ratio (Rtn/Risk)'] = xSI_Y_RtnRisk['AnnRtn(%)'] /
xSI_Y_RtnRisk['AnnStdDev(%)']

xSI_Y_RtnRisk['AnnRtn(%)'] = xSI_Y_RtnRisk['AnnRtn(%)'].astype(float).map("{:.2%}".format)
xSI_Y_RtnRisk['AnnStdDev(%)'] =
xSI_Y_RtnRisk['AnnStdDev(%)'].astype(float).map("{:.2%}".format)
xSI_Y_RtnRisk['Sharpe Ratio (Rtn/Risk)'] = xSI_Y_RtnRisk['Sharpe Ratio
(Rtn/Risk)'].astype(float).map("{:.2f}".format)

xText_RtnRisk = xSI_Y_RtnRisk.to_string()
xText_corr = xSI_Y.corr().to_string()

f = open(xDir + 'xSI_Y_AnnRtnRisk_corr.txt','w')
f.write(xStartDate_Y_SI.strftime('%Y/%m/%d') + ' to ' + xEndDate_Y_SI.strftime('%Y/%m/%d') +
'\n\n'
        + xText_RtnRisk + '\n\n' + xText_corr)
f.close()
##############################
xSI_M =
xDF_M[['DATE','SPLPEQTY_pct_ch_M','HFRIEMNI_pct_ch_M','SI2_pct_ch_M','SI4_pct_ch_M','SI6_pct_c
h_M','SPXT_pct_ch_M','BondTR_pct_ch_M']].copy()
xSI_M.dropna(inplace=True)
xSI_M.reset_index(drop=True,inplace=True)
xStartDate_M_SI= xSI_M['DATE'].min()
xEndDate_M_SI= xSI_M['DATE'].max()

xSI_M_AnnStdDev=pd.DataFrame(xSI_M.std())*np.sqrt(12)
xSI_M_AnnStdDev.reset_index(inplace=True)
xSI_M_AnnStdDev.rename(columns={0: 'AnnStdDev(%)'},inplace=True)

#########
# xSI_M_AnnRtn=pd.DataFrame(xSI_M.mean())*12
# xSI_M_AnnRtn.reset_index(inplace=True)
# xSI_M_AnnRtn.rename(columns={0: 'AnnRtn(%)'},inplace=True)
##########
########## calc annualized return from Monthly returns ##########
#xCumRtn_Y = (1 + Y).cumprod()
xCumRtn_Y = (1 +
xSI_M[['SPLPEQTY_pct_ch_M','HFRIEMNI_pct_ch_M','SI2_pct_ch_M','SI4_pct_ch_M','SI6_pct_ch_M','S
PXT_pct_ch_M','BondTR_pct_ch_M']]).cumprod()
xSI_M_AnnRtn = (xCumRtn_Y.iloc[len(xCumRtn_Y) - 1] / xCumRtn_Y.iloc[0]) ** (1 /
(len(xCumRtn_Y) / 12)) - 1
xSI_M_AnnRtn=pd.DataFrame(xSI_M_AnnRtn)
xSI_M_AnnRtn.reset_index(inplace=True)
xSI_M_AnnRtn.rename(columns={0: 'AnnRtn(%)'},inplace=True)
############################################
xSI_M_RtnRisk = pd.merge(xSI_M_AnnRtn,xSI_M_AnnStdDev,on=['index'],how='left')
xSI_M_RtnRisk['Sharpe Ratio (Rtn/Risk)'] = xSI_M_RtnRisk['AnnRtn(%)'] /
xSI_M_RtnRisk['AnnStdDev(%)']

xSI_M_RtnRisk['AnnRtn(%)'] = xSI_M_RtnRisk['AnnRtn(%)'].astype(float).map("{:.2%}".format)
xSI_M_RtnRisk['AnnStdDev(%)'] =
xSI_M_RtnRisk['AnnStdDev(%)'].astype(float).map("{:.2%}".format)
xSI_M_RtnRisk['Sharpe Ratio (Rtn/Risk)'] = xSI_M_RtnRisk['Sharpe Ratio
(Rtn/Risk)'].astype(float).map("{:.2f}".format)
```

```
xText_RtnRisk = xSI_M_RtnRisk.to_string()
xText_corr = xSI_M.corr().to_string()

f = open(xDir + 'xSI_M_AnnRtnRisk_corr.txt','w')
f.write(xStartDate_M_SI.strftime('%Y/%m/%d') + ' to ' + xEndDate_M_SI.strftime('%Y/%m/%d') +
'\n\n'
        + xText_RtnRisk + '\n\n' + xText_corr)
f.close()
```

```
#2
### Portfolio Optiimization
###
#
# Finds an optimal allocation of stocks in a portfolio,
# satisfying a minimum expected return.
# The problem is posed as a Quadratic Program, and solved
# using the cvxopt library.
# Uses actual past stock data, obtained using the stocks module.
import math
import numpy as np
import pandas as pd
import datetime
import cvxopt
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt
##########################
import warnings
warnings.filterwarnings('ignore')
warnings.warn('DelftStack')
warnings.warn('Do not show this message')
#####################
solvers.options['show_progress'] = False        # !!!

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

#from cvxopt import solvers
#import stocks
import numpy as np
import pandas as pd
import datetime

import pandas as pd
import statsmodels.formula.api as smf
import statsmodels.api as sm

# c = cvxopt.matrix([0, -1], tc='d')
# print('c: ', c)
# c = numpy.matrix(c)
# print('c: ', c)
#
# c = cvxopt.matrix([0, -1])
# print('c: ', c)
# G = cvxopt.matrix([[-1, 1], [3, 2], [2, 3], [-1, 0], [0, -1]], tc='d')
# print('G: ', G)
##################
xDir = r'D:\\Users\\ggu\\Documents\\GU\\MultiRiskFactorModel\\DATA\\'
xSPXT = pd.read_csv(xDir + 'xSPXT.txt')
xSPXT['DATE'] = pd.to_datetime(xSPXT['DATE'], format='%m/%d/%Y')
xBondTR = pd.read_csv(xDir + 'xBondTR.txt')
xBondTR['DATE'] = pd.to_datetime(xBondTR['DATE'], format='%m/%d/%Y')
#xBondTR.rename(columns={'LBUSTRUU': 'BondTR'},inplace=True)
xAAPL = pd.read_csv(xDir + 'xAAPL.txt')
xAAPL['DATE'] = pd.to_datetime(xAAPL['DATE'], format='%m/%d/%Y')
xAGG = pd.read_csv(xDir + 'xAGG.txt')
xAGG['DATE'] = pd.to_datetime(xAGG['DATE'], format='%m/%d/%Y')
xCCY = pd.read_csv(xDir + 'xCCY.txt')
xCCY['DATE'] = pd.to_datetime(xCCY['DATE'], format='%m/%d/%Y')
```

```
xCOMM = pd.read_csv(xDir + 'xCOMM.txt')
xCOMM['DATE'] = pd.to_datetime(xCOMM['DATE'], format='%m/%d/%Y')
xCREDIT = pd.read_csv(xDir + 'xCREDIT.txt')
xCREDIT['DATE'] = pd.to_datetime(xCREDIT['DATE'], format='%m/%d/%Y')
xFTLS = pd.read_csv(xDir + 'xFTLS.txt')
xFTLS['DATE'] = pd.to_datetime(xFTLS['DATE'], format='%m/%d/%Y')
xHFRIEMNI = pd.read_csv(xDir + 'xHFRIEMNI.txt')
xHFRIEMNI['DATE'] = pd.to_datetime(xHFRIEMNI['DATE'], format='%m/%d/%Y')
xPRBAX = pd.read_csv(xDir + 'xPRBAX.txt')
xPRBAX['DATE'] = pd.to_datetime(xPRBAX['DATE'], format='%m/%d/%Y')
xPRWAX = pd.read_csv(xDir + 'xPRWAX.txt')
xPRWAX['DATE'] = pd.to_datetime(xPRWAX['DATE'], format='%m/%d/%Y')
xSPLPEQTY = pd.read_csv(xDir + 'xSPLPEQTY.txt')
xSPLPEQTY['DATE'] = pd.to_datetime(xSPLPEQTY['DATE'], format='%m/%d/%Y')
xSPX = pd.read_csv(xDir + 'xSPX.txt')
xSPX['DATE'] = pd.to_datetime(xSPX['DATE'], format='%m/%d/%Y')
xSPY = pd.read_csv(xDir + 'xSPY.txt')
xSPY['DATE'] = pd.to_datetime(xSPY['DATE'], format='%m/%d/%Y')
xTSLA = pd.read_csv(xDir + 'xTSLA.txt')
xTSLA['DATE'] = pd.to_datetime(xTSLA['DATE'], format='%m/%d/%Y')
xUS3M = pd.read_csv(xDir + 'xUS3M.txt')
xUS3M['DATE'] = pd.to_datetime(xUS3M['DATE'], format='%m/%d/%Y')
xUS10Y = pd.read_csv(xDir + 'xUS10Y.txt')
xUS10Y['DATE'] = pd.to_datetime(xUS10Y['DATE'], format='%m/%d/%Y')
xHYG = pd.read_csv(xDir + 'xHYG.txt')
xHYG['DATE'] = pd.to_datetime(xHYG['DATE'], format='%m/%d/%Y')
xCPI = pd.read_csv(xDir + 'xCPI.txt')
xCPI['DATE'] = pd.to_datetime(xCPI['DATE'], format='%m/%d/%Y')
xHYTR = pd.read_csv(xDir + 'xLF98TRUU.txt')
xHYTR['DATE'] = pd.to_datetime(xHYTR['DATE'], format='%m/%d/%Y')
xTIPS = pd.read_csv(xDir + 'xLBUTTRUU.txt')
xTIPS['DATE'] = pd.to_datetime(xTIPS['DATE'], format='%m/%d/%Y')
xGMWAX = pd.read_csv(xDir + 'xGMWAX.txt')
xGMWAX['DATE'] = pd.to_datetime(xGMWAX['DATE'], format='%m/%d/%Y')
xCashConst = pd.read_csv(xDir + 'xCashConst.txt')
xCashConst['DATE'] = pd.to_datetime(xCashConst['DATE'], format='%m/%d/%Y')
xS5INFT = pd.read_csv(xDir + 'xS5INFT.txt')
xS5INFT['DATE'] = pd.to_datetime(xS5INFT['DATE'], format='%m/%d/%Y')
x7030TR = pd.read_csv(xDir + 'x7030TR.txt')
x7030TR['DATE'] = pd.to_datetime(x7030TR['DATE'], format='%m/%d/%Y')
xUSCredit = pd.read_csv(xDir + 'xLUCRTRUU.txt')
xUSCredit['DATE'] = pd.to_datetime(xUSCredit['DATE'], format='%m/%d/%Y')
xSHY = pd.read_csv(xDir + 'xSHY.txt')
xSHY['DATE'] = pd.to_datetime(xSHY['DATE'], format='%m/%d/%Y')
xTIP = pd.read_csv(xDir + 'xTIP.txt')
xTIP['DATE'] = pd.to_datetime(xTIP['DATE'], format='%m/%d/%Y')
xAMZN = pd.read_csv(xDir + 'xAMZN.txt')
xAMZN['DATE'] = pd.to_datetime(xAMZN['DATE'], format='%m/%d/%Y')
xFB = pd.read_csv(xDir + 'xFB.txt')
xFB['DATE'] = pd.to_datetime(xFB['DATE'], format='%m/%d/%Y')
xVIAC = pd.read_csv(xDir + 'xVIAC.txt')
xVIAC['DATE'] = pd.to_datetime(xVIAC['DATE'], format='%m/%d/%Y')
xGOOG = pd.read_csv(xDir + 'xGOOG.txt')
xGOOG['DATE'] = pd.to_datetime(xGOOG['DATE'], format='%m/%d/%Y')
xLQD = pd.read_csv(xDir + 'xLQD.txt')
xLQD['DATE'] = pd.to_datetime(xLQD['DATE'], format='%m/%d/%Y')
xMDY = pd.read_csv(xDir + 'xMDY.txt')
xMDY['DATE'] = pd.to_datetime(xMDY['DATE'], format='%m/%d/%Y')
xMSFT = pd.read_csv(xDir + 'xMSFT.txt')
xMSFT['DATE'] = pd.to_datetime(xMSFT['DATE'], format='%m/%d/%Y')
```

```
xRLV = pd.read_csv(xDir + 'xRLV.txt')
xRLV['DATE'] = pd.to_datetime(xRLV['DATE'], format='%m/%d/%Y')
xRLG = pd.read_csv(xDir + 'xRLG.txt')
xRLG['DATE'] = pd.to_datetime(xRLG['DATE'], format='%m/%d/%Y')
xRIY = pd.read_csv(xDir + 'xRIY.txt')
xRIY['DATE'] = pd.to_datetime(xRIY['DATE'], format='%m/%d/%Y')
xRMV = pd.read_csv(xDir + 'xRMV.txt')
xRMV['DATE'] = pd.to_datetime(xRMV['DATE'], format='%m/%d/%Y')
xRMC = pd.read_csv(xDir + 'xRMC.txt')
xRMC['DATE'] = pd.to_datetime(xRMC['DATE'], format='%m/%d/%Y')
xRDG = pd.read_csv(xDir + 'xRDG.txt')
xRDG['DATE'] = pd.to_datetime(xRDG['DATE'], format='%m/%d/%Y')
xRUJ = pd.read_csv(xDir + 'xRUJ.txt')
xRUJ['DATE'] = pd.to_datetime(xRUJ['DATE'], format='%m/%d/%Y')
xRTY = pd.read_csv(xDir + 'xRTY.txt')
xRTY['DATE'] = pd.to_datetime(xRTY['DATE'], format='%m/%d/%Y')
xRUO = pd.read_csv(xDir + 'xRUO.txt')
xRUO['DATE'] = pd.to_datetime(xRUO['DATE'], format='%m/%d/%Y')
xSCHP = pd.read_csv(xDir + 'xSCHP.txt')
xSCHP['DATE'] = pd.to_datetime(xSCHP['DATE'], format='%m/%d/%Y')
xIEF = pd.read_csv(xDir + 'xIEF.txt')
xIEF['DATE'] = pd.to_datetime(xIEF['DATE'], format='%m/%d/%Y')
xMUB = pd.read_csv(xDir + 'xMUB.txt')
xMUB['DATE'] = pd.to_datetime(xMUB['DATE'], format='%m/%d/%Y')
xSH = pd.read_csv(xDir + 'xSH.txt')
xSH['DATE'] = pd.to_datetime(xSH['DATE'], format='%m/%d/%Y')
xSSO = pd.read_csv(xDir + 'xSSO.txt')
xSSO['DATE'] = pd.to_datetime(xSSO['DATE'], format='%m/%d/%Y')
xTREASURY = pd.read_csv(xDir + 'xLUATTRUU.txt')
xTREASURY['DATE'] = pd.to_datetime(xTREASURY['DATE'], format='%m/%d/%Y')

##########################################
xDF = xSPX.copy()
xDF = pd.merge(xDF, xSPXT, on=['DATE'], how='left')
xDF = pd.merge(xDF, xBondTR, on=['DATE'], how='left')
xDF = pd.merge(xDF, xAAPL, on=['DATE'], how='left')
xDF = pd.merge(xDF, xAGG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xCCY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xCOMM, on=['DATE'], how='left')
xDF = pd.merge(xDF, xCREDIT, on=['DATE'], how='left')
xDF = pd.merge(xDF, xFTLS, on=['DATE'], how='left')
###xDF = pd.merge(xDF, xHFRIEMNI, on=['DATE'], how='left')
xDF = pd.merge(xDF, xPRBAX, on=['DATE'], how='left')
xDF = pd.merge(xDF, xPRWAX, on=['DATE'], how='left')
xDF = pd.merge(xDF, xSPLPEQTY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xSPY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xTSLA, on=['DATE'], how='left')
xDF = pd.merge(xDF, xUS3M, on=['DATE'], how='left')
xDF = pd.merge(xDF, xUS10Y, on=['DATE'], how='left')
xDF = pd.merge(xDF, xHYG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xHYTR, on=['DATE'], how='left')
xDF = pd.merge(xDF, xTIPS, on=['DATE'], how='left')
xDF = pd.merge(xDF, xGMWAX, on=['DATE'], how='left')
xDF = pd.merge(xDF, xCashConst, on=['DATE'], how='left')
xDF = pd.merge(xDF, xS5INFT, on=['DATE'], how='left')
xDF = pd.merge(xDF, x7030TR, on=['DATE'], how='left')
xDF = pd.merge(xDF, xUSCredit, on=['DATE'], how='left')
xDF = pd.merge(xDF, xSHY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xTIP, on=['DATE'], how='left')
xDF = pd.merge(xDF, xAMZN, on=['DATE'], how='left')
```

```
xDF = pd.merge(xDF, xFB, on=['DATE'], how='left')
xDF = pd.merge(xDF, xVIAC, on=['DATE'], how='left')
xDF = pd.merge(xDF, xGOOG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xLQD, on=['DATE'], how='left')
xDF = pd.merge(xDF, xMDY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xMSFT, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRLV, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRLG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRIY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRMV, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRMC, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRDG, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRUJ, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRTY, on=['DATE'], how='left')
xDF = pd.merge(xDF, xRUO, on=['DATE'], how='left')
xDF = pd.merge(xDF, xSCHP, on=['DATE'], how='left')
xDF = pd.merge(xDF, xIEF, on=['DATE'], how='left')
xDF = pd.merge(xDF, xMUB, on=['DATE'], how='left')
xDF = pd.merge(xDF, xSH, on=['DATE'], how='left')
xDF = pd.merge(xDF, xSSO, on=['DATE'], how='left')
xDF = pd.merge(xDF, xTREASURY, on=['DATE'], how='left')

####################################################################
# xEndDate_0 = pd.to_datetime('10/1/2018')
# xDF = xDF.loc[xDF['DATE']<xEndDate_0]
# ################ forward fill the missing equity trading dates ############
xDF['BondTR'].fillna(method='ffill', inplace=True)
xDF['HYTR'].fillna(method='ffill', inplace=True)
xDF['TIPS'].fillna(method='ffill', inplace=True)
xDF['LQD'].fillna(method='ffill', inplace=True)
xDF['SPLPEQTY'].fillna(method='ffill', inplace=True)
#################################
################Calculating SI returns here #################################
for k in range(1,4):
    if k==1:
        # 2 year, buffer -10%, x1.5, cap = 21%, hard buffer note!
        xCap = 0.21 #0.21 #0.21  #1000 #0.21   #1000  #0.21
        xBuffer = -0.10000  ####-0.10  #-0.25   #-0.30   #-0.25
        xTerm = 2  #2  #4  #6   #4 #2  #3 # years
        xAmount = 100
        xLever = 1.500  #1.5  #1.15
        xBufferType = "H"  #"T"  # "H" for regular Buffer; "G" for Geared Buffer (or Barrier); 
"T" for Trigger Buffer!
    elif k == 2:
        # 4 years, buffer -25%, no leverage and no cap, barrier buffer note!
        xCap = 10000
        xBuffer = -0.250000
        xTerm = 4
        xAmount = 100
        xLever = 1.00
        xBufferType = "T"
    elif k==3:
        # 6 years, buffer -30%, x1.15 leverage and no cap, barrier buffer note!
        xCap = 10000  # 0.21 #0.21  #1000 #0.21   #1000  #0.21
        xBuffer = -0.300000  ####-0.10  #-0.25   #-0.30   #-0.25
        xTerm = 6
        xAmount = 100
        xLever = 1.1500  # 1.5  #1.15
        xBufferType = "T"  # "T"  # "H" for regular Buffer; "G" for Geared Buffer (or 
Barrier); "T" for Trigger Buffer!
```

```
    xDF['SPX_rtn_term'] = xDF['SPX'].pct_change(xTerm*252)
    xDF.loc[xDF['SPX_rtn_term'] > 0, 'SI' + (str)(xTerm) + '_rtn_term'] = xDF['SPX_rtn_term']
* xLever
    xDF.loc[xDF['SPX_rtn_term']* xLever > xCap, 'SI' + (str)(xTerm) + '_rtn_term'] = xCap
    xDF.loc[(xDF['SPX_rtn_term']<=0) & (xDF['SPX_rtn_term']>=xBuffer),
'SI'+(str)(xTerm)+'_rtn_term'] = 0
    if (xBufferType=='H'):
        xDF.loc[(xDF['SPX_rtn_term']<xBuffer),'SI'+(str)(xTerm)+'_rtn_term'] =
xDF['SPX_rtn_term'] - xBuffer
    elif (xBufferType=='T'):
        xDF.loc[(xDF['SPX_rtn_term']<xBuffer),'SI'+(str)(xTerm)+'_rtn_term'] =
xDF['SPX_rtn_term']
    elif (xBufferType=='G'):
        xK = 1 / (1+xBuffer)
        xDF.loc[(xDF['SPX_rtn_term']<xBuffer),'SI'+(str)(xTerm)+'_rtn_term'] = xK *
(xDF['SPX_rtn_term'] - xBuffer)

    #xDF['SI'+(str)(xTerm)+'_pct_ch_Y'] = (1+xDF['SI'+(str)(xTerm)+'_rtn_term'])**(1/xTerm) -
1
    #xDF['SI'+(str)(xTerm)+'_pct_ch_Q'] =
(1+xDF['SI'+(str)(xTerm)+'_rtn_term'])**(1/(xTerm*4)) - 1
    xDF['SI'+(str)(xTerm)+'_pct_ch_M'] = (1+xDF['SI'+(str)(xTerm)+'_rtn_term'])**(1/xTerm) - 1
#(1+xDF['SI'+(str)(xTerm)+'_rtn_term'])**(1/(xTerm*12)) - 1
    #xDF['SI'+(str)(xTerm)+'_pct_ch_D'] =
(1+xDF['SI'+(str)(xTerm)+'_rtn_term'])**(1/(xTerm*252)) - 1

###########################
#########################################
xSPX_2 = xSPX.copy()
xSPX_2['month']=xSPX_2.DATE.dt.month
xSPX_2['year']=xSPX_2.DATE.dt.year
xSPX_2['diff']=xSPX_2.month.diff(-1)
xSPX_2['month-year']=xSPX_2.month.astype('string')+'-'+xSPX_2.year.astype('string')
xSPX_2 = xSPX_2.loc[xSPX_2['diff']!=0]
##### equity market neutral index (monthly) ######
xHFRIEMNI['month']=xHFRIEMNI.DATE.dt.month
xHFRIEMNI['year']=xHFRIEMNI.DATE.dt.year
###xHFRIEMNI['diff']=xHFRIEMNI.month.diff(-1)
xHFRIEMNI['month-year']=xHFRIEMNI.month.astype('string')+'-'+xHFRIEMNI.year.astype('string')
xHFRIEMNI.rename(columns={'DATE': 'DATE0'},inplace=True)
#xHFRIEMNI['LS_1y_pct_ch']=xHFRIEMNI['HFRIEMNI'].pct_change(12)
xHFRIEMNI = pd.merge(xHFRIEMNI, xSPX_2[['DATE','month-year']], on=['month-year'],how='left')
xDF = pd.merge(xDF, xHFRIEMNI[['DATE','HFRIEMNI']], on=['DATE'], how='left')
##### CPI index (monthly) ######
xCPI['month']=xCPI.DATE.dt.month
xCPI['year']=xCPI.DATE.dt.year
xCPI['month-year']=xCPI.month.astype('string')+'-'+xCPI.year.astype('string')
xCPI.rename(columns={'DATE': 'DATE0'},inplace=True)
#xCPI['LS_1y_pct_ch']=xCPI['HFRIEMNI'].pct_change(12)
xCPI = pd.merge(xCPI, xSPX_2[['DATE','month-year']], on=['month-year'],how='left')
xDF = pd.merge(xDF, xCPI[['DATE','CPI']], on=['DATE'], how='left')
###########
xDF = pd.merge(xDF, xSPX_2[['DATE','month-year','diff']], on=['DATE'], how='left')
xDF_M = xDF.loc[xDF['diff']!=0]
xDF_M = xDF_M.loc[xDF_M['diff'].notnull()]
xDF_M.reset_index(drop=True, inplace=True)
xDF_M['RealBondTR'] = xDF_M['BondTR']/xDF_M['CPI']
xDF_M['quarter'] = xDF_M['DATE'].dt.quarter
xDF_M['diff_Q']=xDF_M.quarter.diff(-1)
```

```
######################################################

#########################################
xDF['SPXT_pct_ch_D']=xDF['SPXT'].pct_change()
xDF['BondTR_pct_ch_D']=xDF['BondTR'].pct_change()
# #########################

############### new portfolio #########
################
xY_col = 'SPLPEQTY'
xY_col = 'FTLS'
xY_col = 'CashConst'
xY_col = 'PRWAX'
xY_col = 'GMWAX'
xY_col = 'PRBAX'
xY_col = 'SI'
xY_col = '7030TR'

xY_col = 'SPY'
xY_col = 'SHY'
xY_col = 'TIP'
xY_col = 'AGG'
xY_col = 'HYG'
xY_col = 'TSLA'
xY_col = 'AAPL'

xCoef_table = pd.DataFrame()
xRiskExp_Current = pd.DataFrame()
xRiskConcentration_Current = pd.DataFrame()
xCoefxStdDev = pd.DataFrame()
xStdDev_indep_table=pd.DataFrame()
##################
xW1=0.75
xW2=0.25
xW3=0.0  #0.10
#xW4=0.15
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['BondTR_pct_ch_D']
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['TIPS_pct_ch_D']
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['BondTR_pct_ch_D']+
xW3*xDF['SI2_pct_ch_D']
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['BondTR_pct_ch_D']+
xW3*xDF['SI2_pct_ch_D']
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['SPXT_pct_ch_D']+
xW3*xDF['SI2_pct_ch_D']
#xDF['NewPort_pct_ch_D'] =xW1* xDF[xY_col + '_pct_ch_D'] + xW2*xDF['SI2_pct_ch_D']+
xW3*xDF['TIPS_pct_ch_D']+ xW4*xDF['HYTR_pct_ch_D']

#xDF['NewPort'] = (1+xDF['NewPort_pct_ch_D']).cumprod()

############################
#xDF_M = pd.merge(xDF_M,xDF[['DATE','NewPort']],on=['DATE'],how='left')
######################
xCols_pct = xDF.columns[xDF.columns.str.contains(pat = '_pct_ch')]

xDF2=xDF[xCols_pct].copy()

xCorrelations = xDF2.corr()
xStdDev=xDF2.std()
xMean = xDF2.mean()
xCorrelations.to_csv(xDir+'xCorrelations.txt')
```

```
xStdDev.to_csv(xDir+'xStdDev.txt')
xMean.to_csv(xDir+'xMean.txt')
########### model portfolios #########
CONS_E=0.2
MODCONS_E=0.4
MOD_E=0.6
MODGROW_E=0.75
GROW_E=0.9
MAXGROW_E=0.98

#xMP=xDF[['DATE','SPXT','SPXT_pct_ch_D','SPXT_pct_ch_Y','BondTR','BondTR_pct_ch_D','BondTR_pct
_ch_Y']].copy()
#xMP=xDF[['DATE','SPXT_pct_ch_D','BondTR_pct_ch_D',xY_col+'_pct_ch_D','NewPort_pct_ch_D','NewP
ort_pct_ch_Y']].copy()
xMP=xDF[['DATE','SPXT_pct_ch_D','BondTR_pct_ch_D']].copy()
xMP.rename(columns={xY_col+'_pct_ch_D':'Current_pct_ch_D','NewPort_pct_ch_D':'New_pct_ch_D'},i
nplace=True)
xMP['CONS_pct_ch_D']=CONS_E*xMP['SPXT_pct_ch_D']+(1-CONS_E)*xMP['BondTR_pct_ch_D']     #daily
rebalanced as benchmark index
xMP['MODCONS_pct_ch_D']=MODCONS_E*xMP['SPXT_pct_ch_D']+(1-MODCONS_E)*xMP['BondTR_pct_ch_D']
#daily rebalanced as benchmark index
xMP['MOD_pct_ch_D']=MOD_E*xMP['SPXT_pct_ch_D']+(1-MOD_E)*xMP['BondTR_pct_ch_D']     #daily
rebalanced as benchmark index
xMP['MODGROW_pct_ch_D']=MODGROW_E*xMP['SPXT_pct_ch_D']+(1-MODGROW_E)*xMP['BondTR_pct_ch_D']
#daily rebalanced as benchmark index
xMP['GROW_pct_ch_D']=GROW_E*xMP['SPXT_pct_ch_D']+(1-GROW_E)*xMP['BondTR_pct_ch_D']     #daily
rebalanced as benchmark index
xMP['MAXGROW_pct_ch_D']=MAXGROW_E*xMP['SPXT_pct_ch_D']+(1-MAXGROW_E)*xMP['BondTR_pct_ch_D']
#daily rebalanced as benchmark index

xMP['CONS_port']=(1 + xMP['CONS_pct_ch_D']).cumprod()
xMP['MODCONS_port']=(1 + xMP['MODCONS_pct_ch_D']).cumprod()
xMP['MOD_port']=(1 + xMP['MOD_pct_ch_D']).cumprod()
xMP['MODGROW_port']=(1 + xMP['MODGROW_pct_ch_D']).cumprod()
xMP['GROW_port']=(1 + xMP['GROW_pct_ch_D']).cumprod()
xMP['MAXGROW_port']=(1 + xMP['MAXGROW_pct_ch_D']).cumprod()
#xMP['Current_port']=(1 + xMP['Current_pct_ch_D']).cumprod()
#xMP['New_port']=(1 + xMP['New_pct_ch_D']).cumprod()

#
# xMP['New_pct_ch_Y']=xMP['NewPort_pct_ch_Y']
############################
xMP_MQ = xDF_M[['DATE']].copy()
# xMP_MQ = pd.merge(xMP_MQ,
xMP[['DATE','CONS_port','MODCONS_port','MOD_port','MODGROW_port','GROW_port','MAXGROW_port',
#                                'Current_port','New_port']],on=['DATE'],how='left')
xMP_MQ = pd.merge(xMP_MQ,
xMP[['DATE','CONS_port','MODCONS_port','MOD_port','MODGROW_port','GROW_port','MAXGROW_port'
                              ]],on=['DATE'],how='left')
xMP_MQ['CONS_pct_ch_M']=xMP_MQ['CONS_port'].pct_change()
xMP_MQ['MODCONS_pct_ch_M']=xMP_MQ['MODCONS_port'].pct_change()
xMP_MQ['MOD_pct_ch_M']=xMP_MQ['MOD_port'].pct_change()
xMP_MQ['MODGROW_pct_ch_M']=xMP_MQ['MODGROW_port'].pct_change()
xMP_MQ['GROW_pct_ch_M']=xMP_MQ['GROW_port'].pct_change()
xMP_MQ['MAXGROW_pct_ch_M']=xMP_MQ['MAXGROW_port'].pct_change()
#xMP_MQ['Current_pct_ch_M']=xMP_MQ['Current_port'].pct_change()
#xMP_MQ['New_pct_ch_M']=xMP_MQ['New_port'].pct_change()


############################
```

```
xCols_pct_MP = xMP.columns[xMP.columns.str.contains(pat = '_pct_ch_M')]
xMP2=xMP[xCols_pct_MP].copy()
xAnnRtn_MP_Y=xMP2.mean()
xStdDev_MP_Y=xMP2.std()
xStdDev_MP_Y.to_csv(xDir+'xStdDev_MP_M.txt')
xAnnRtn_MP_Y.to_csv(xDir+'xAnnRtn_MP_M.txt')
#################
# xCols_pct_MP = xMP.columns[xMP.columns.str.contains(pat = '_pct_ch_D')]
# xMP2=xMP[xCols_pct_MP].copy()
# xAnnRtn_MP_D=xMP2.mean() * 252 ####  try to annualized compounded annual return
# xStdDev_MP_D=xMP2.std() * np.sqrt(252)
# xStdDev_MP_D.to_csv(xDir+'xStdDev_MP_D.txt')
# xAnnRtn_MP_D.to_csv(xDir+'xAnnRtn_MP_D.txt')

#xStdDev_MP[0] is the std dev of the conservative model portfolio
#xStdDev_MP[5] is the std dev of the MAX Growth model portfolio
# std dev > xStdDev_MP[5] is ACCESSIVE GROWTH portfolio!!!!
################### MONTHLY #################
##################################################
xDF_M['SPXT_pct_ch_M']=xDF_M['SPXT'].pct_change()
xDF_M['BondTR_pct_ch_M']=xDF_M['BondTR'].pct_change()
xDF_M['AAPL_pct_ch_M']=xDF_M['AAPL'].pct_change()
xDF_M['MSFT_pct_ch_M']=xDF_M['MSFT'].pct_change()
xDF_M['AMZN_pct_ch_M']=xDF_M['AMZN'].pct_change()
xDF_M['FB_pct_ch_M']=xDF_M['FB'].pct_change()
xDF_M['AGG_pct_ch_M']=xDF_M['AGG'].pct_change()
xDF_M['CCY_pct_ch_M']=xDF_M['CCY'].pct_change()
xDF_M['COMM_pct_ch_M']=xDF_M['COMM'].pct_change()
xDF_M['CREDIT_pct_ch_M']=xDF_M['CREDIT'].pct_change()
xDF_M['FTLS_pct_ch_M']=xDF_M['FTLS'].pct_change()
xDF_M['HFRIEMNI_pct_ch_M']=xDF_M['HFRIEMNI'].pct_change()
xDF_M['PRBAX_pct_ch_M']=xDF_M['PRBAX'].pct_change()
xDF_M['PRWAX_pct_ch_M']=xDF_M['PRWAX'].pct_change()
xDF_M['SPLPEQTY_pct_ch_M']=xDF_M['SPLPEQTY'].pct_change()
xDF_M['SPX_pct_ch_M']=xDF_M['SPX'].pct_change()
xDF_M['SPY_pct_ch_M']=xDF_M['SPY'].pct_change()
xDF_M['TSLA_pct_ch_M']=xDF_M['TSLA'].pct_change()
xDF_M['US3M_pct_ch_M']=xDF_M['US3M'].pct_change()
xDF_M['US10Y_pct_ch_M']=xDF_M['US10Y'].pct_change()
xDF_M['HYG_pct_ch_M']=xDF_M['HYG'].pct_change()
xDF_M['HYTR_pct_ch_M']=xDF_M['HYTR'].pct_change()
xDF_M['RealBondTR_pct_ch_M']=xDF_M['RealBondTR'].pct_change()
xDF_M['CPI_pct_ch_M']=xDF_M['CPI'].pct_change()
xDF_M['CPI_pct_ch_Y']=xDF_M['CPI'].pct_change(12)
xDF_M['TIPS_pct_ch_M']=xDF_M['TIPS'].pct_change()
xDF_M['GMWAX_pct_ch_M']=xDF_M['GMWAX'].pct_change()
xDF_M['CashConst_pct_ch_M']=xDF_M['CashConst'].pct_change()
xDF_M['S5INFT_pct_ch_M']=xDF_M['S5INFT'].pct_change()
xDF_M['7030TR_pct_ch_M']=xDF_M['7030TR'].pct_change()
xDF_M['USCredit_pct_ch_M']=xDF_M['USCredit'].pct_change()
xDF_M['SHY_pct_ch_M']=xDF_M['SHY'].pct_change()
xDF_M['TIP_pct_ch_M']=xDF_M['TIP'].pct_change()
xDF_M['GOOG_pct_ch_M']=xDF_M['GOOG'].pct_change()
xDF_M['VIAC_pct_ch_M']=xDF_M['VIAC'].pct_change()
xDF_M['LQD_pct_ch_M']=xDF_M['LQD'].pct_change()
xDF_M['MDY_pct_ch_M']=xDF_M['MDY'].pct_change()
xDF_M['RLV_pct_ch_M']=xDF_M['RLV'].pct_change()
xDF_M['RIY_pct_ch_M']=xDF_M['RIY'].pct_change()
xDF_M['RLG_pct_ch_M']=xDF_M['RLG'].pct_change()
xDF_M['RMV_pct_ch_M']=xDF_M['RMV'].pct_change()
```

```
xDF_M['RMC_pct_ch_M']=xDF_M['RMC'].pct_change()
xDF_M['RDG_pct_ch_M']=xDF_M['RDG'].pct_change()
xDF_M['RUJ_pct_ch_M']=xDF_M['RUJ'].pct_change()
xDF_M['RTY_pct_ch_M']=xDF_M['RTY'].pct_change()
xDF_M['RUO_pct_ch_M']=xDF_M['RUO'].pct_change()
xDF_M['SCHP_pct_ch_M']=xDF_M['SCHP'].pct_change()
xDF_M['IEF_pct_ch_M']=xDF_M['IEF'].pct_change()
xDF_M['MUB_pct_ch_M']=xDF_M['MUB'].pct_change()
xDF_M['SH_pct_ch_M']=xDF_M['SH'].pct_change()
xDF_M['SSO_pct_ch_M']=xDF_M['SSO'].pct_change()
xDF_M['TREASURY_pct_ch_M']=xDF_M['TREASURY'].pct_change()

#xDF_M['Inflation_pct_ch_M'] = xDF_M['BondTR_pct_ch_M'] - xDF_M['TIPS_pct_ch_M']

############### overwrite to create the EXACT 70/30 returns #############
xDF_M['7030TR_pct_ch_M']=0.7*xDF_M['SPXT_pct_ch_M']+0.3*xDF_M['BondTR_pct_ch_M']
xDF_M['3070TR_pct_ch_M']=0.3*xDF_M['SPXT_pct_ch_M']+0.7*xDF_M['BondTR_pct_ch_M']
xDF_M['30AAPL30MSFT20AMZN20GOOGTR_pct_ch_M']= 0.3 * xDF_M['AAPL_pct_ch_M'] +
0.3*xDF_M['MSFT_pct_ch_M'] \
                                              + 0.2*xDF_M['AMZN_pct_ch_M'] +
0.2*xDF_M['GOOG_pct_ch_M']
xDF_M['30SPY30MDY20AGG20LQDTR_pct_ch_M']= 0.3 * xDF_M['SPY_pct_ch_M'] +
0.3*xDF_M['MDY_pct_ch_M'] \
                                              + 0.2*xDF_M['AGG_pct_ch_M'] +
0.2*xDF_M['LQD_pct_ch_M']
#############################################

######## OLS HERE MONTHLY ###############
# xCols_pct_ch_M= xDF_M.columns[xDF_M.columns.str.contains(pat = '_pct_ch_M')]
# xCols_pct_ch_M=xCols_pct_ch_M.insert(0,'DATE')

#
xRiskFactorSet_M=['SPXT_pct_ch_M','BondTR_pct_ch_M','CCY_pct_ch_M','COMM_pct_ch_M','USCredit_p
ct_ch_M','HYTR_pct_ch_M',
#                    'TIPS_pct_ch_M','Inflation_pct_ch_M']
#'CPI_pct_ch_M','RealBondTR_pct_ch_M'

xRiskFactorSet_M=['SPXT_pct_ch_M','USCredit_pct_ch_M','TREASURY_pct_ch_M','COMM_pct_ch_M']
#'CPI_pct_ch_M','RealBondTR_pct_ch_M'
#xRiskFactorSet_M=['SPXT_pct_ch_M','USCredit_pct_ch_M','TIPS_pct_ch_M','COMM_pct_ch_M','HYTR_p
ct_ch_M']   #'CPI_pct_ch_M','RealBondTR_pct_ch_M'

################## derive orthogonal risk factors ##################
xDF_orthog = xDF_M[['DATE']+xRiskFactorSet_M]
xDF_orthog.dropna(inplace=True)
xDF_orthog.reset_index(drop=True,inplace=True)
## (1) derive orthog_SPXT #####
Y = xDF_orthog['USCredit_pct_ch_M']# xDF_orthog['SPXT_pct_ch_M']
X = xDF_orthog['TREASURY_pct_ch_M'] #xDF_orthog['TIPS_pct_ch_M']
X = sm.add_constant(X)
model = sm.OLS(Y, X)
result = model.fit()
xDF_orthog['orthog_USCredit_pct_ch_M'] = result.params[0] + result.resid

## (2) derive orthog_USCredit #####
Y = xDF_orthog['SPXT_pct_ch_M'] #xDF_orthog['USCredit_pct_ch_M']
X = xDF_orthog[['orthog_USCredit_pct_ch_M','TREASURY_pct_ch_M']]
#xDF_orthog[['SPXT_pct_ch_M','TIPS_pct_ch_M']]
X = sm.add_constant(X)
model = sm.OLS(Y, X)
```

```
result = model.fit()
xDF_orthog['orthog_SPXT_pct_ch_M'] = result.params[0] + result.resid

## (3) derive orthog_COMM #####
Y = xDF_orthog['COMM_pct_ch_M']
X = xDF_orthog[['orthog_USCredit_pct_ch_M','orthog_SPXT_pct_ch_M','TREASURY_pct_ch_M']]
#xDF_orthog[['SPXT_pct_ch_M','TIPS_pct_ch_M','USCredit_pct_ch_M']]
X = sm.add_constant(X)
model = sm.OLS(Y, X)
result = model.fit()
xDF_orthog['orthog_COMM_pct_ch_M'] = result.params[0] + result.resid

#xRiskFactorCorrelations_orthog =
(xDF_orthog[['orthog_SPXT_pct_ch_M','orthog_USCredit_pct_ch_M','orthog_COMM_pct_ch_M','TIPS_pc
t_ch_M']].corr()).round(4)
#xRiskFactorCorrelations_raw =
(xDF_orthog[['SPXT_pct_ch_M','USCredit_pct_ch_M','COMM_pct_ch_M','TIPS_pct_ch_M']].corr()).rou
nd(4)
xRiskFactorCorrelations_orthog =
(xDF_orthog[['orthog_SPXT_pct_ch_M','orthog_USCredit_pct_ch_M','orthog_COMM_pct_ch_M','TREASUR
Y_pct_ch_M']].corr()).round(4)
xRiskFactorCorrelations_raw =
(xDF_orthog[['SPXT_pct_ch_M','USCredit_pct_ch_M','COMM_pct_ch_M','TREASURY_pct_ch_M']].corr())
.round(4)

xDF_M =
pd.merge(xDF_M,xDF_orthog[['DATE','orthog_SPXT_pct_ch_M','orthog_USCredit_pct_ch_M','orthog_CO
MM_pct_ch_M']],on=['DATE'],how='left')
##################### bring in the rolling annual returns for Model Portfolios as a benchmarks
for Risk Exposures ###########
xDF_M =
pd.merge(xDF_M,xMP_MQ[['DATE','CONS_pct_ch_M','MODCONS_pct_ch_M','MOD_pct_ch_M','MODGROW_pct_c
h_M','GROW_pct_ch_M','MAXGROW_pct_ch_M']],on=['DATE'],how='left')
#########################################################################################
xOrthogonal = 'orthog'
#xOrthogonal = ''

if xOrthogonal == 'orthog':
    xRiskFactorSet_M = ['orthog_SPXT_pct_ch_M', 'orthog_USCredit_pct_ch_M',
'orthog_COMM_pct_ch_M', 'TREASURY_pct_ch_M']
    #xRiskFactorSet_M = ['orthog_SPXT_pct_ch_M', 'orthog_USCredit_pct_ch_M',
'orthog_COMM_pct_ch_M', 'TIPS_pct_ch_M']
    #xRiskFactorSet_M = ['orthog_SPXT_pct_ch_M', 'TIPS_pct_ch_M', 'orthog_COMM_pct_ch_M']
else:
    xOrthogonal = ''
    xRiskFactorSet_M = ['SPXT_pct_ch_M', 'USCredit_pct_ch_M', 'TIPS_pct_ch_M',
                        'COMM_pct_ch_M']  # 'CPI_pct_ch_M','RealBondTR_pct_ch_M'
    #xRiskFactorSet_M = ['SPXT_pct_ch_M',
'TIPS_pct_ch_M','RealBondTR_pct_ch_M','COMM_pct_ch_M']  # 'CPI_pct_ch_M','RealBondTR_pct_ch_M'

#xRiskFactorSet_M =
['orthog_SPXT_pct_ch_M','orthog_USCredit_pct_ch_M','TIPS_pct_ch_M','orthog_COMM_pct_ch_M','HYT
R_pct_ch_M']
############# in REAL terms #####################
xDF_M['RealSPXT_pct_ch_M'] = xDF_M['SPXT_pct_ch_M'] - xDF_M['TIPS_pct_ch_M']
xDF_M['RealUSCredit_pct_ch_M'] = xDF_M['USCredit_pct_ch_M'] - xDF_M['TIPS_pct_ch_M']
xDF_M['RealCOMM_pct_ch_M'] = xDF_M['COMM_pct_ch_M'] - xDF_M['TIPS_pct_ch_M']
##xRiskFactorSet_M=['RealSPXT_pct_ch_M','RealUSCredit_pct_ch_M','TIPS_pct_ch_M','RealCOMM_pct_
ch_M']
##################################
```

```
xDescriptive_M=xDF_M[[xY_col+'_pct_ch_M']+xRiskFactorSet_M].describe(include='all').to_string(
)
xCorrelations_M=xDF_M[[xY_col+'_pct_ch_M']+xRiskFactorSet_M].corr().to_string()

xDescriptive_M = xDescriptive_M + '\n\n' + xCorrelations_M
f = open(xDir + 'xDescriptive_M_'+xY_col+'.txt','w')
f.write(xDescriptive_M + '\r\n')
f.close()

xDep_var = ['SPY','AGG','HYG','SHY','MSFT','AMZN','FB','GOOG','VIAC',
            'LQD','30AAPL30MSFT20AMZN20GOOGTR','30SPY30MDY20AGG20LQDTR','TSLA',
            '7030TR','SI2','SI4','SI6','SPLPEQTY','CONS','MODCONS',
            'MOD','MODGROW','GROW','MAXGROW','CashConst','AAPL']
xDep_var = ['SPY','AGG','HYG','SHY','MSFT','AMZN','FB','GOOG','VIAC',
            'LQD','30AAPL30MSFT20AMZN20GOOGTR','30SPY30MDY20AGG20LQDTR','AAPL',
            '7030TR','3070TR','SI2','SI4','SI6','SPLPEQTY','CONS','MODCONS',
            'MOD','MODGROW','GROW','MAXGROW','CashConst','RLV','RIY','RMV',
            'RMC','RDG','RUJ','RTY','RUO','TSLA']  #### 'RLG',

xDep_var = ['SPY','IEF','AGG','LQD','HYG','MUB','SH','SSO','RIY','RLG',
            'RLV','RTY','RUO','RUJ']
#xDep_var = ['TSLA']  #### 'RLG',

### xDep_var = ['RTY','RUO','AAPL']

#xDep_var = ['RLV','RLG','RIY','RMV','RMC','RDG','RUJ','RTY','RUO','AAPL']

#xDep_Var = [xY_col]

#xRiskFactorSet_M = ['SPXT_pct_ch_M','BondTR_pct_ch_M']
#xRiskFactorSet_M = ['RLG_pct_ch_M']

xRisk_concentration_Current = pd.DataFrame()
xRisk_concentration_New = pd.DataFrame()
for xY_col in xDep_var:
    xDF_M['NewPort_pct_ch_M'] = 0.75 * xDF_M[xY_col + '_pct_ch_M'] + 0.25 *
xDF_M['SI4_pct_ch_M']
    xDF_OLS_M = xDF_M[['DATE'] + xRiskFactorSet_M + [xY_col + '_pct_ch_M',
'NewPort_pct_ch_M']].copy()  #'CPI_pct_ch_M',
    ####xDF_OLS_M=xDF[xCols_pct_ch_M].copy()

    xDF_OLS_M = xDF_OLS_M.loc[xDF_OLS_M['DATE']>='2005-01-01']

    xDF_OLS_M.dropna(inplace=True)
    xDF_OLS_M.reset_index(drop=True,inplace=True)
    ########### set up weightings for WLS here ####################
    xDF_OLS_M['w'] = np.exp(-(-xDF_OLS_M.index+xDF_OLS_M.index.max()) / (len(xDF_OLS_M) / 5))
# exponential
    #xDF_OLS_M['w'] = (xDF_OLS_M.index) / xDF_OLS_M.index.max()  # linear - the latest has
more weights!
    ###########################
    xStartDate_M = xDF_OLS_M['DATE'].min()
    xEndDate_M = xDF_OLS_M['DATE'].max()
    ################### model portfolios for Rooling annual rate ###########
    xMP_M = xDF_OLS_M[['DATE',xY_col + '_pct_ch_M','NewPort_pct_ch_M']].copy()
    xColumns_temp =
['DATE','CONS_pct_ch_M','MODCONS_pct_ch_M','MOD_pct_ch_M','MODGROW_pct_ch_M','GROW_pct_ch_M',
                          'MAXGROW_pct_ch_M']
    if (xY_col + '_pct_ch_M' in xColumns_temp):
```

```
        xColumns_temp.remove(xY_col + '_pct_ch_M')
    xMP_M = pd.merge(xMP_M, xMP_MQ[xColumns_temp], on=['DATE'], how='left')
#,'Current_pct_ch_M','New_pct_ch_M'

    xMP_M_CumRtn = (1 + xMP_M[['CONS_pct_ch_M', 'MODCONS_pct_ch_M', 'MOD_pct_ch_M',
'MODGROW_pct_ch_M', 'GROW_pct_ch_M',
                               'MAXGROW_pct_ch_M']]).cumprod()  #, 'Current_pct_ch_M',
'New_pct_ch_M'
    xMP_M_AnnRtn = (xMP_M_CumRtn.iloc[len(xMP_M_CumRtn) - 1] / xMP_M_CumRtn.iloc[0]) ** (
                1 / (len(xMP_M_CumRtn) / 12)) - 1

    xMP_M_AnnRtn = xMP_M_AnnRtn.reset_index()
    xMP_M_AnnRisk = xMP_M[
        ['CONS_pct_ch_M', 'MODCONS_pct_ch_M', 'MOD_pct_ch_M', 'MODGROW_pct_ch_M',
'GROW_pct_ch_M', 'MAXGROW_pct_ch_M'
         ]].std().reset_index()  #'Current_pct_ch_M', 'New_pct_ch_M'
    xMP_M_AnnRtn.rename(columns={0: 'AnnRtn'}, inplace=True)
    xMP_M_AnnRisk.rename(columns={0: 'AnnRisk'}, inplace=True)
    xMP_M_AnnRisk['AnnRisk'] = xMP_M_AnnRisk['AnnRisk'] * np.sqrt(12)

    xMP_M_AnnRtnRisk = pd.merge(xMP_M_AnnRtn, xMP_M_AnnRisk, on=['index'], how='left')

    ################ OLS ################
    xInd_Vars =
['SPXT_pct_ch_M','BondTR_pct_ch_M','TIPS_pct_ch_M','CCY_pct_ch_M','COMM_pct_ch_M','USCredit_pc
t_ch_M','HYTR_pct_ch_M']
    xInd_Vars =
['SPXT_pct_ch_M','BondTR_pct_ch_M','TIPS_pct_ch_M','CCY_pct_ch_M','USCredit_pct_ch_M']
    xInd_Vars =
['SPXT_pct_ch_M','BondTR_pct_ch_M','TIPS_pct_ch_M','CCY_pct_ch_M','USCredit_pct_ch_M','HYTR_pc
t_ch_M']
    xInd_Vars =
['SPXT_pct_ch_M','Inflation_pct_ch_M','TIPS_pct_ch_M','CCY_pct_ch_M','USCredit_pct_ch_M','HYTR
_pct_ch_M']
    xInd_Vars =
['SPXT_pct_ch_M','Inflation_pct_ch_M','TIPS_pct_ch_M','CCY_pct_ch_M','USCredit_pct_ch_M']
    xInd_Vars = ['SPXT_pct_ch_M']
    xInd_Vars = ['SPXT_pct_ch_M','BondTR_pct_ch_M']
    xInd_Vars =
['SPXT_pct_ch_M','BondTR_pct_ch_M','TIPS_pct_ch_M','CCY_pct_ch_M','COMM_pct_ch_M','USCredit_pc
t_ch_M','HYTR_pct_ch_M']

    xInd_Vars = xRiskFactorSet_M

    ###xInd_Vars = ['S5INFT_pct_ch_M']
    #xInd_Vars =
['SPXT_pct_ch_M','Inflation_pct_ch_M','TIPS_pct_ch_M','CCY_pct_ch_M','USCredit_pct_ch_M']
    X = xDF_OLS_M[xInd_Vars]
    ############### risk factors annual returns and std dev ##########
    X_StdDev_M = xDF_OLS_M[xInd_Vars].std().reset_index()
    X_Rtn_M = xDF_OLS_M[xInd_Vars].mean().reset_index()
    ##################################################################
    xVersion=['Current','New']
    #xVersion=['Current']
    xRisk_exposures_Current=pd.DataFrame()
    xRisk_exposures_New=pd.DataFrame()
    for x in xVersion:
        if x=='Current':
            Y = xDF_OLS_M[xY_col + '_pct_ch_M']
            xY_col2 = xY_col
```

```
            xCorrelations_M = xDF_OLS_M[[xY_col + '_pct_ch_M'] + xInd_Vars].corr().to_string()
        elif x=='New':
            Y = xDF_OLS_M['NewPort_pct_ch_M']
            xY_col2 = 'New'
            xCorrelations_M = xDF_OLS_M[[xY_col2 + 'Port_pct_ch_M'] + 
xInd_Vars].corr().to_string()
        #xInd_Vars = 
['SPXT_pct_ch_M','RealBondTR_pct_ch_M','TIPS_pct_ch_M','CPI_pct_ch_M','CCY_pct_ch_M','COMM_pct
_ch_M','USCredit_pct_ch_M','HYTR_pct_ch_M']

        #xCorrelations_M = xDF_OLS_M[[xY_col + '_pct_ch_M']+xInd_Vars].corr().to_string()
        f = open(xDir + 'xCorrelations_M_' + xY_col + '_' + x + '.txt','w')
        f.write(xCorrelations_M + '\r\n')
        f.close()

        X = sm.add_constant(X)
        xStart_time = datetime.datetime.now() #time.time_ns()*1000000
        xRegressionType ='OLS'    #'OLS' #'WLS'
        if xRegressionType == 'OLS':
            model = sm.OLS(Y,X)
        elif xRegressionType == 'WLS':
            model = sm.WLS(Y, X, weights=xDF_OLS_M['w'])
        result = model.fit()
        # for i in range(1,9999):
        #     print(i)
        xEnd_time = datetime.datetime.now() #time.time_ns()*1000000
        globals()['xSecond_M_'+x] = 'Start: '+(str)(xStart_time) +'; End: '+(str)(xEnd_time) + 
'; Duration: ' +(str)((xEnd_time - xStart_time))
        xOLS_Summary_M = result.summary()
        xOLS_text = xOLS_Summary_M.as_text()

        f = open(xDir + 'xOLS_M_' + xY_col +  '_' + x + '.txt','w')
        f.write(globals()['xSecond_M_'+x] + '\n\n' + xOLS_text + '\r\n')
        f.close()

        ########## calc annualized return from Monthly returns ##########
        xCumRtn_Y = (1 + Y).cumprod()
        xAnnRtn_Y_M = (xCumRtn_Y[len(xCumRtn_Y) - 1] / xCumRtn_Y[0]) ** (1 / (len(xCumRtn_Y) / 
12)) - 1
        xAnnRisk_Y_M = np.sqrt(12 * Y.var())
        # xMP_M_AnnRtnRisk=xMP_M_AnnRtnRisk.append({'index':'Current 
Portfolio','AnnRtn':xAnnRtn_M_M,'AnnRisk':xAnnRisk_M_M}, ignore_index=True)
        # ############################################

        xVar_X = np.array(X.var())
        xVar_Y = Y.var()
        xCoef_sq = result.params**2
        xVar_resid = result.resid.var()
        xVar_CoefX = xCoef_sq * xVar_X
        xDelta_var = xVar_Y - np.sum(xVar_CoefX) - xVar_resid       #this is the 
diversificaation effect
        xDelta_varX = xDelta_var * xVar_CoefX / np.sum(xVar_CoefX)
        xVar_X_adj = xVar_CoefX + xDelta_varX
        xVar_X_adj_pct = xVar_X_adj / xVar_Y
        xVar_resid_pct = xVar_resid / xVar_Y
        print (xVar_X_adj_pct, xVar_resid_pct)
        print(np.sum(xVar_X_adj_pct)+xVar_resid_pct)

        xVar_X_adj_pct = pd.DataFrame(xVar_X_adj_pct)
        xVar_X_adj_pct.reset_index(inplace=True)
```

```
        xVar_X_adj_pct.rename(columns={0: 'Risk_Concentration(%)'},inplace=True)
        xVar_X_adj_pct.rename(columns={'index': 'Risk_Factor'},inplace=True)
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Idiosyncratic',
'Risk_Concentration(%)': xVar_resid_pct}, ignore_index=True)

        xVar_X_adj_pct=xVar_X_adj_pct.loc[~xVar_X_adj_pct['Risk_Factor'].isin({'const'})]
        xSum = xVar_X_adj_pct['Risk_Concentration(%)'].sum()
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':'Sum', 'Risk_Concentration(%)':
xSum}, ignore_index=True)
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual
StDev)', 'Risk_Concentration(%)': xAnnRisk_Y_M}, ignore_index=True)
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Annual Rtn)',
'Risk_Concentration(%)': xAnnRtn_Y_M}, ignore_index=True)

        xVar_X_adj_pct['Risk_Concentration(%)'] =
xVar_X_adj_pct['Risk_Concentration(%)'].astype(float).map("{:.2%}".format)
        xVar_X_adj_pct=xVar_X_adj_pct.append({'Risk_Factor':model.endog_names+'(Sharpe
Ratio)', 'Risk_Concentration(%)': np.round(xAnnRtn_Y_M/xAnnRisk_Y_M,2)}, ignore_index=True)

        # for x in (result.tvalues.index):
        #     if x=='const':
        #         continue
        #     else:
        #         #print (x, result.tvalues[x])
        #         if (abs(result.tvalues[x]) <1.5):
        #             xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor'] == x]['Risk_Exposure(%)'] =
'NA'
        #             #print (x,
xVar_X_adj_pct[xVar_X_adj_pct['Risk_Factor']==x]['Risk_Exposure(%)'])

        #####globals()['xRisk_concentration_'+x] = xVar_X_adj_pct

        xRisk_Exposure_M = xVar_X_adj_pct.to_string()

        f = open(xDir + 'xRisk_Concentration_M_' + xY_col + '_' + x + '.txt', 'w')
        f.write(xRisk_Exposure_M + '\r\n')
        f.close()

        ################# store Risk Concentration for the Current Portfolio ###############
        xRiskConcentration_temp = xVar_X_adj_pct[['Risk_Factor',
'Risk_Concentration(%)']].copy()
        xRiskConcentration_temp.rename(columns={'Risk_Concentration(%)': xY_col},
inplace=True)
        xRiskConcentration_temp['Risk_Factor'][len(xRiskConcentration_temp) - 1] =
'Sharpe_Ratio'
        xRiskConcentration_temp['Risk_Factor'][len(xRiskConcentration_temp) - 2] =
'Annual_Rtn'
        xRiskConcentration_temp['Risk_Factor'][len(xRiskConcentration_temp) - 3] =
'Annual_StdDev'
        if len(globals()['xRisk_concentration_'+x]) == 0:
            globals()['xRisk_concentration_'+x] = xRiskConcentration_temp.copy()
        else:
            globals()['xRisk_concentration_'+x] =
pd.merge(globals()['xRisk_concentration_'+x], xRiskConcentration_temp, on=['Risk_Factor'],
how='left')

        # ######################################
        ############### the following is working on the Std Dev (RISK) ANNUALLY #######
        xStdDev_indep = (X.std() * np.sqrt(12)).reset_index()
```

```
        xStdDev_indep.rename(columns={0: xY_col, 'index':'Risk_Factor'}, inplace=True)
        xStdDev_indep = xStdDev_indep.loc[~xStdDev_indep['Risk_Factor'].isin({'const'})]
        xStdDev_indep[xY_col] = xStdDev_indep[xY_col].astype(float).map("{:.2%}".format)
        xStdDev_indep = xStdDev_indep.append({'Risk_Factor': 'Start_Date', xY_col:
xStartDate_M.strftime('%m/%d/%Y') }, ignore_index=True)
        xStdDev_indep = xStdDev_indep.append({'Risk_Factor': 'End_Date', xY_col:
xEndDate_M.strftime('%m/%d/%Y')}, ignore_index=True)
        ########
        xStdDev_X = np.array(X.std()) * np.sqrt(12)
        xStdDev_Y = Y.std() * np.sqrt(12)
        xCoef = result.params.abs()
        xStdDev_resid = result.resid.std() * np.sqrt(12)
        xStdDev_CoefX = xCoef * xStdDev_X
        xDelta_StdDev = xStdDev_Y - np.sum(xStdDev_CoefX) - xStdDev_resid #this is the
diversification benefit...
        print('xDelta_StdDev = ', xDelta_StdDev)
        xAdj_StdDev_resid = False
        if (xAdj_StdDev_resid == False):
            xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / np.sum(xStdDev_CoefX)
        else:
            xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / (np.sum(xStdDev_CoefX) +
xStdDev_resid)
            xStdDev_resid = xStdDev_resid + xDelta_StdDev * xStdDev_resid /
(np.sum(xStdDev_CoefX) + xStdDev_resid)
        xStdDev_X_adj = xStdDev_CoefX + xDelta_StdDevX

        xStdDev_X_adj = pd.DataFrame(xStdDev_X_adj)
        xStdDev_X_adj.reset_index(inplace=True)

        xStdDev_X_adj.rename(columns={0: 'Risk_Exposure(%)'},inplace=True)
        xStdDev_X_adj.rename(columns={'index': 'Risk_Factor'},inplace=True)
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Idiosyncratic', 'Risk_Exposure(%)':
xStdDev_resid}, ignore_index=True)

        xStdDev_X_adj=xStdDev_X_adj.loc[~xStdDev_X_adj['Risk_Factor'].isin({'const'})]
        xSum = xStdDev_X_adj['Risk_Exposure(%)'].sum()
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sum', 'Risk_Exposure(%)': xSum},
ignore_index=True)
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual StDev)',
'Risk_Exposure(%)': xAnnRisk_Y_M}, ignore_index=True)
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual Rtn)',
'Risk_Exposure(%)': xAnnRtn_Y_M}, ignore_index=True)

        #xStdDev_X_adj['Risk_Exposure(%)'] =
xStdDev_X_adj['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)

        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sharpe Ratio (Rtn/Risk)',
'Risk_Exposure(%)': np.round(xAnnRtn_Y_M / xAnnRisk_Y_M,2)}, ignore_index=True)
        xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Diversification benefit',
'Risk_Exposure(%)': xDelta_StdDev}, ignore_index=True)

        if x== 'Current':
            xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (' + xY_col + ')'},
inplace=True)
        else:
            xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (proposed)'},
inplace=True)
        globals()['xRisk_exposures_' + x] = xStdDev_X_adj

        xIndex_StdDev=globals()['xRisk_exposures_' + x][
```

```
            globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names + '(Annual
StdDev)'].index.values[0]
        xIndex_Rtn = globals()['xRisk_exposures_' + x][
            globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names + '(Annual
Rtn)'].index.values[0]
        globals()['xRisk_exposures_' + x].loc[globals()['xRisk_exposures_' + x].index ==
xIndex_StdDev, 'Risk_Factor'] = 'Annual StdDev'
        globals()['xRisk_exposures_' + x].loc[
            globals()['xRisk_exposures_' + x].index == xIndex_Rtn, 'Risk_Factor'] = 'Annual
Rtn'

        globals()['xIdio_exp_' + x] = xStdDev_resid / xSum

        if x=='New':  # second time and last time!
            xStdDev_X_adj = pd.merge(xRisk_exposures_Current, xRisk_exposures_New,
on='Risk_Factor', how='left')

        xRisk_Exposure_M_StdDev = xStdDev_X_adj.to_string()

        ######################################################################
        #xRisk_Exposure_M.to_csv (xDir + 'xRisk_Exposure_M.txt')

        f = open(xDir + 'xRisk_Exposure_M_' + xY_col +  '_' + x + '.txt','w')
        f.write(xRisk_Exposure_M_StdDev + '\r\n')
        f.close()
        ############### store oefficients for 'Current" portfolio ########
        if x=='Current':
            xCoef_temp = pd.DataFrame(result.params).reset_index()
            xCoef_temp.rename(columns={0: xY_col}, inplace=True)
            xCoef_temp[xY_col] = xCoef_temp[xY_col].round(4)
            xCoef_temp.rename(columns={'index': 'Risk_Factor'}, inplace=True)
            if len(xCoef_table)==0:
                xCoef_table = xCoef_temp.copy()
            else:
                xCoef_table = pd.merge(xCoef_table, xCoef_temp, on=['Risk_Factor'],how='left')
            ######################
            xCoefxStdDev_temp = pd.DataFrame(xStdDev_CoefX).reset_index()
            xCoefxStdDev_temp.rename(columns={0: xY_col}, inplace=True)
            xCoefxStdDev_temp[xY_col] = xCoefxStdDev_temp[xY_col].round(4)
            xCoefxStdDev_temp.rename(columns={'index': 'Risk_Factor'}, inplace=True)
            xCoefxStdDev_temp =
xCoefxStdDev_temp.loc[xCoefxStdDev_temp['Risk_Factor']!='const']
            if len(xCoefxStdDev)==0:
                xCoefxStdDev = xCoefxStdDev_temp.copy()
            else:
                xCoefxStdDev = pd.merge(xCoefxStdDev, xCoefxStdDev_temp,
on=['Risk_Factor'],how='left')
            ###################
            if len(xStdDev_indep_table)==0:
                xStdDev_indep_table = xStdDev_indep.copy()
            else:
                xStdDev_indep_table = pd.merge(xStdDev_indep_table, xStdDev_indep,
on=['Risk_Factor'],how='left')
        ############### the following is working on the Std Dev (RISK) ANNUALLY with AR(1)
error term #######
        if False:
            ###################### the following is AR(1) error term ########################
            from statsmodels.tsa.arima.model import ARIMA as ARIMA

            X2 = X.drop('const', axis=1)
```

```
            sarimax_model = ARIMA(endog=Y, exog=X2, order=(1, 0, 0))  # X already has a
constant term, trend='c')  # , seasonal_order=(0,1,1,24))
            sarimax_results = sarimax_model.fit()
            sarimax_results.summary()

            xOLS_AR1_Summary_M = sarimax_results.summary()
            xOLS_AR1_text = xOLS_AR1_Summary_M.as_text()

            f = open(xDir + 'xOLS_AR1_M_' + xY_col + '_' + x + '.txt', 'w')
            f.write(xOLS_AR1_text + '\r\n')
            f.close()

            xStdDev_X = np.array(X.std())   #these are already annualized std dev
            xStdDev_M = Y.std()     #these are already annualized std dev
            xCoef = sarimax_results.params[:len(X.columns)].abs()
            xStdDev_resid = np.sqrt(sarimax_results.params[len(X.columns):].values[1] / (1-
sarimax_results.params[len(X.columns):].values[0]**2)) #result.resid.std()
            xStdDev_CoefX = xCoef * xStdDev_X
            xDelta_StdDev = xStdDev_M - np.sum(xStdDev_CoefX) - xStdDev_resid
            print('xDelta_StdDev = ', xDelta_StdDev)
            xAdj_StdDev_resid = False
            if (xAdj_StdDev_resid == False):
                xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / np.sum(xStdDev_CoefX)
            else:
                xDelta_StdDevX = xDelta_StdDev * xStdDev_CoefX / (np.sum(xStdDev_CoefX) +
xStdDev_resid)
                xStdDev_resid = xStdDev_resid + xDelta_StdDev * xStdDev_resid /
(np.sum(xStdDev_CoefX) + xStdDev_resid)
            xStdDev_X_adj = xStdDev_CoefX + xDelta_StdDevX

            xStdDev_X_adj = pd.DataFrame(xStdDev_X_adj)
            xStdDev_X_adj.reset_index(inplace=True)

            xStdDev_X_adj.rename(columns={0: 'Risk_Exposure(%)'},inplace=True)
            xStdDev_X_adj.rename(columns={'index': 'Risk_Factor'},inplace=True)
            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Idiosyncratic',
'Risk_Exposure(%)': xStdDev_resid}, ignore_index=True)

            xStdDev_X_adj=xStdDev_X_adj.loc[~xStdDev_X_adj['Risk_Factor'].isin({'const'})]
            xSum = xStdDev_X_adj['Risk_Exposure(%)'].sum()
            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sum', 'Risk_Exposure(%)':
xSum}, ignore_index=True)
            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual
StDev)', 'Risk_Exposure(%)': xAnnRisk_M_M}, ignore_index=True)
            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':model.endog_names+'(Annual
Rtn)', 'Risk_Exposure(%)': xAnnRtn_M_M}, ignore_index=True)

            #xStdDev_X_adj['Risk_Exposure(%)'] =
xStdDev_X_adj['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)

            xStdDev_X_adj=xStdDev_X_adj.append({'Risk_Factor':'Sharpe Ratio (Rtn/Risk)',
'Risk_Exposure(%)': np.round(xAnnRtn_M_M / xAnnRisk_M_M,2)}, ignore_index=True)

            if x== 'Current':
                xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (' + xY_col +
')'}, inplace=True)
            else:
                xStdDev_X_adj.rename(columns={'Risk_Exposure(%)': x + ' Risk (proposed)'},
inplace=True)
            globals()['xRisk_exposures_' + x] = xStdDev_X_adj
```

```
            xIndex_StdDev=globals()['xRisk_exposures_' + x][
                globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names +
'(Annual StdDev)'].index.values[0]
            xIndex_Rtn = globals()['xRisk_exposures_' + x][
                globals()['xRisk_exposures_' + x]['Risk_Factor'] == model.endog_names +
'(Annual Rtn)'].index.values[0]
            globals()['xRisk_exposures_' + x].loc[globals()['xRisk_exposures_' + x].index ==
xIndex_StdDev, 'Risk_Factor'] = 'Annual StdDev'
            globals()['xRisk_exposures_' + x].loc[
                globals()['xRisk_exposures_' + x].index == xIndex_Rtn, 'Risk_Factor'] =
'Annual Rtn'

            globals()['xIdio_exp_' + x] = xStdDev_resid / xSum

            if x=='New':  # second time and last time!
                xStdDev_X_adj = pd.merge(xRisk_exposures_Current, xRisk_exposures_New,
on='Risk_Factor', how='left')


            xRisk_Exposure_M_StdDev = xStdDev_X_adj.to_string()

            ######################################################################
            #xRisk_Exposure_M.to_csv (xDir + 'xRisk_Exposure_M.txt')

            f = open(xDir + 'xRisk_Exposure_M_AR(1)_' + xY_col +  '_' + x + '.txt','w')
            f.write(xRisk_Exposure_M_StdDev + '\r\n')
            f.close()

    ######################
    xStdDev_X_M = pd.DataFrame(X.std()).reset_index()
    xStdDev_X_M.rename(columns={0:'Risk_Factor_AnnStdDev'}, inplace=True)
    xStdDev_X_M.rename(columns={'index':'Risk_Factor'}, inplace=True)
    xStdDev_X_adj = pd.merge(xStdDev_X_adj,xStdDev_X_M, on=['Risk_Factor'],how='left')
    xStdDev_X_adj['Risk_Exp (Current)'] = xStdDev_X_adj['Current Risk
('+xY_col+')']/xStdDev_X_adj['Risk_Factor_AnnStdDev']
    xStdDev_X_adj['Risk_Exp (New)'] = xStdDev_X_adj['New Risk
(proposed)']/xStdDev_X_adj['Risk_Factor_AnnStdDev']
    xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Idiosyncratic','Risk_Exp
(Current)']=xIdio_exp_Current
    xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Idiosyncratic','Risk_Exp
(New)']=xIdio_exp_New
    xSum_Risk_Exp_Current = xStdDev_X_adj['Risk_Exp (Current)'].sum()
    xSum_Risk_Exp_New = xStdDev_X_adj['Risk_Exp (New)'].sum()
    xStdDev_X_adj['Risk_Exp (Current)'] = xStdDev_X_adj['Risk_Exp
(Current)']/xSum_Risk_Exp_Current
    xStdDev_X_adj['Risk_Exp (New)'] = xStdDev_X_adj['Risk_Exp (New)']/xSum_Risk_Exp_New
    xSum_Risk_Exp_Current = xStdDev_X_adj['Risk_Exp (Current)'].sum()
    xSum_Risk_Exp_New = xStdDev_X_adj['Risk_Exp (New)'].sum()
    xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Sum','Risk_Exp
(Current)']=xSum_Risk_Exp_Current
    xStdDev_X_adj.loc[xStdDev_X_adj['Risk_Factor']=='Sum','Risk_Exp (New)']=xSum_Risk_Exp_New

    xPart_1=xStdDev_X_adj.loc[xStdDev_X_adj.index<len(xStdDev_X_adj)-1]
    xPart_2=xStdDev_X_adj.loc[xStdDev_X_adj.index==len(xStdDev_X_adj)-1]

    xPart_1['Current Risk ('+xY_col+')'] = xPart_1['Current Risk
('+xY_col+')'].astype(float).map("{:.2%}".format)
    xPart_1['New Risk (proposed)'] = xPart_1['New Risk
(proposed)'].astype(float).map("{:.2%}".format)
```

```
    xPart_1['Risk_Factor_AnnStdDev'] =
xPart_1['Risk_Factor_AnnStdDev'].astype(float).map("{:.2%}".format)
    xPart_1['Risk_Exp (Current)'] = xPart_1['Risk_Exp
(Current)'].astype(float).map("{:.2%}".format)
    xPart_1['Risk_Exp (New)'] = xPart_1['Risk_Exp (New)'].astype(float).map("{:.2%}".format)

    xStdDev_X_adj=xPart_1.append(xPart_2, ignore_index=True)

    xStdDev_X_adj = xStdDev_X_adj.replace({'nan%': ''})
    xStdDev_X_adj = xStdDev_X_adj.replace({np.nan: ''})

    xRisk_Exposure_M_StdDev = xStdDev_X_adj.to_string()
    #xStdDev_X_adj['Risk_Exposure(%)'] =
xStdDev_X_adj['Risk_Exposure(%)'].astype(float).map("{:.2%}".format)
    ######################
    f = open(xDir + 'xRisk_Exposure_M_' + xY_col + '.txt','w')
    f.write('From '+xStartDate_M.strftime('%Y-%m-%d') +' to ' + xEndDate_M.strftime('%Y-%m-
%d') + '\n\n' +xRisk_Exposure_M_StdDev + '\r\n')
    f.close()
    ################# store Risk Exposures for the Current Portfolio ###############
    xRiskExp_Current_temp = xStdDev_X_adj[['Risk_Factor','Risk_Exp (Current)']].copy()
    xRiskExp_Current_temp.rename(columns={'Risk_Exp (Current)':xY_col}, inplace=True)
    if len(xRiskExp_Current)==0:
        xRiskExp_Current = xRiskExp_Current_temp.copy()
    else:
        xRiskExp_Current =
pd.merge(xRiskExp_Current,xRiskExp_Current_temp,on=['Risk_Factor'],how='left')

#res = pd.concat([xRiskExp_Current, xCoef_table], axis=1, keys=["Risk_Exp", "Coefs"])
d={} #dictionary of dataframe
xRiskExp_Current2 =
xRiskExp_Current.loc[xRiskExp_Current['Risk_Factor'].isin(list(xRiskFactorSet_M+['Idiosyncrati
c']))]
d['Current_Risk_Exposures']=xRiskExp_Current2.set_index('Risk_Factor')
d=pd.concat(d, axis=1)

A={}
xCoef_table2 = xCoef_table.loc[xCoef_table['Risk_Factor'].isin(xRiskFactorSet_M)]
A['Coefficients']=xCoef_table2.set_index('Risk_Factor')
A=pd.concat(A, axis=1)

B={}
##xRisk_concentration_Current2 =
xRisk_concentration_Current.loc[xRisk_concentration_Current['Risk_Factor'].isin(xRiskFactorSet
_M)]
B['Current_Risk_Concentration']=xRisk_concentration_Current.set_index('Risk_Factor')
B=pd.concat(B, axis=1)

C={}
###xRisk_concentration_New2 =
xRisk_concentration_New.loc[xRisk_concentration_New['Risk_Factor'].isin(xRiskFactorSet_M)]
C['New_Risk_Concentration']=xRisk_concentration_New.set_index('Risk_Factor')
C=pd.concat(C, axis=1)

# create excel writer
if xOrthogonal == 'orthog':
    writer = pd.ExcelWriter(xDir + 'xRiskFactorExp_Corre_orthog.xlsx')

    A.reset_index().to_excel(writer, 'Coefficients_orthog')
    xStdDev_indep_table.to_excel(writer, 'Ann_StdDev_RiskFactor_orthog')
```

```
    xCoefxStdDev.to_excel(writer, 'Coef_x_StdDev_orthog')
    xRiskFactorCorrelations_orthog.to_excel(writer, 'Corre_RiskFactor_orthog')

    d.reset_index().to_excel(writer, 'Risk_Exposures_orthog')
    B.reset_index().to_excel(writer, 'Risk_Concentration_orthog')
else:
    writer = pd.ExcelWriter(xDir + 'xRiskFactorExp_Corre_raw.xlsx')
    d.reset_index().to_excel(writer, 'Risk_Exposures_raw')
    A.reset_index().to_excel(writer, 'Coefficients_raw')
    xRiskFactorCorrelations_raw.to_excel(writer, 'Corre_RiskFactor_raw')
    B.reset_index().to_excel(writer, 'Risk_Concentration_raw')

# save the excel file
writer.save()
writer.close()
###
#
#
#### Scatter plots for Current Portfolio vs 6 Model Portfolios (Using Daily, Monthly,
Qquarterly and Annual Rtns #####
import matplotlib.pyplot as plt

xFreq = ''
for k in range(1,2):
    if k==0:
        xRtn_Risk_Name = 'xMP_D_AnnRtnRisk'
        xFreq = '(using daily data)'
        xStartDate = xStartDate_D
        xEndDate = xEndDate_D

    elif k == 1:
        xRtn_Risk_Name = 'xMP_M_AnnRtnRisk'
        xFreq = '(using monthly data)'
        xStartDate = xStartDate_M
        xEndDate = xEndDate_M
    elif k == 2:
        xRtn_Risk_Name = 'xMP_Q_AnnRtnRisk'
        xFreq = '(using quarterly data)'
        xStartDate = xStartDate_Q
        xEndDate = xEndDate_Q
    elif k == 3:
        xRtn_Risk_Name = 'xMP_Y_AnnRtnRisk'
        xFreq = '(using annual data)'
        xStartDate = xStartDate_Y
        xEndDate = xEndDate_Y

    xMP_Rtn_Risk=globals()[xRtn_Risk_Name].copy()
    #xMP_Rtn_Risk=xMP_M_AnnRtnRisk.copy()

    xMP_name = pd.DataFrame()
    xMP_name['name']=''
    xMP_name['Rtn_Risk']=''
    xRtn= xMP_Rtn_Risk['AnnRtn'][0]
    xRisk= xMP_Rtn_Risk['AnnRisk'][0]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Conservative','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][1]
    xRisk= xMP_Rtn_Risk['AnnRisk'][1]
```

```
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Moderate Conservative','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][2]
    xRisk= xMP_Rtn_Risk['AnnRisk'][2]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Moderate','Rtn_Risk': xRtn_Risk}, ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][3]
    xRisk= xMP_Rtn_Risk['AnnRisk'][3]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Moderate Growth','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][4]
    xRisk= xMP_Rtn_Risk['AnnRisk'][4]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Growth','Rtn_Risk': xRtn_Risk}, ignore_index=True)
    xRtn= xMP_Rtn_Risk['AnnRtn'][5]
    xRisk= xMP_Rtn_Risk['AnnRisk'][5]
    xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    xMP_name = xMP_name.append({'name':'Maximum Growth','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    # xRtn= xMP_Rtn_Risk['AnnRtn'][6]
    # xRisk= xMP_Rtn_Risk['AnnRisk'][6]
    # xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    # xMP_name = xMP_name.append({'name':'Current Portfoio','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    # xRtn= xMP_Rtn_Risk['AnnRtn'][7]
    # xRisk= xMP_Rtn_Risk['AnnRisk'][7]
    # xRtn_Risk = '('+f'{round(xRtn*100,1)}%' +
','+f'{round(xRisk*100,1)}%'+','+f'{round(xRtn/xRisk,2)}'+')'
    # xMP_name = xMP_name.append({'name':'New Portfoio','Rtn_Risk': xRtn_Risk},
ignore_index=True)
    #######
    xMP_Rtn_Risk['Lable']=''
    xMP_Rtn_Risk['Rtn_Risk']=''
    i=0
    for x in xMP_name['name']:
        xMP_Rtn_Risk['Lable'][i]=xMP_name['name'][i]
        xMP_Rtn_Risk['Rtn_Risk'][i]=xMP_name['Rtn_Risk'][i]
        i=i+1
    ##############
    x = xMP_Rtn_Risk['AnnRisk'].values
    y = xMP_Rtn_Risk['AnnRtn'].values
    #types = xMP_Rtn_Risk.reset_index()['index'].values
    #types = xMP_Rtn_Risk['index'].values
    types = xMP_Rtn_Risk['Lable'].values

    fig, ax = plt.subplots()
    #ax.plot(risks, returns, color='red', label='Equity/Bond')      # this is a line
(efficient frontier)
    xSubText = xFreq + ' from ' + xStartDate.strftime('%m/%d/%Y') + ' to ' +
xEndDate.strftime('%m/%d/%Y')
    fig.suptitle('Return and Risk of the Current Portfolio (' + xY_col+') vs Model Portfolios
\n' + xSubText, fontsize=13,y=0.98)
```

```
    #ax.set_xlabel('Risk (Annualized Std)', fontsize=10)
    #ax.set_ylabel('Annualized Return', fontsize=10)

    #fig, ax = plt.subplots(figsize=(10,10))
    ax.scatter(x, y)

    ax.set_xlabel('Annualized Risk', fontsize=12)
    ax.set_ylabel('Annualized Return', fontsize=12)
    #ax.set_title('(Return and Risk) of the Current Portfolio vs Model Portfolios ' +
xSubText, fontsize=18)

    for i, txt in enumerate(types):
        ax.annotate(txt + '\n' + xMP_Rtn_Risk['Rtn_Risk'][i], (x[i], y[i]), xytext=(-18,-18),
textcoords='offset points',ha="left", size=8)
        #ax.annotate(txt + '\n', (x[i], y[i]), xytext=(10, 10), textcoords='offset points')
        plt.scatter(x, y, marker='o', color='blue')

    plt.savefig(xDir + xRtn_Risk_Name +'_'+xY_col+'.png')
    plt.show()

################## SI and SPXT and BondTR: Rolling Annual Returns and Calendar Monthly Returns
###########
##############################
xSI_Y = xDF_M[['DATE','SI2_pct_ch_M','SI4_pct_ch_M','SI6_pct_ch_M','SPXT_pct_ch_M']].copy()
xSI_Y.dropna(inplace=True)
xSI_Y.reset_index(drop=True,inplace=True)
xStartDate_M_SI= xSI_Y['DATE'].min()
xEndDate_M_SI= xSI_Y['DATE'].max()

xSI_Y_AnnStdDev=pd.DataFrame(xSI_Y.std())   #*np.sqrt(12)
xSI_Y_AnnStdDev.reset_index(inplace=True)
xSI_Y_AnnStdDev.rename(columns={0: 'AnnStdDev(%)'},inplace=True)

#########
xSI_Y_AnnRtn=pd.DataFrame(xSI_Y.mean())
xSI_Y_AnnRtn.reset_index(inplace=True)
xSI_Y_AnnRtn.rename(columns={0: 'AnnRtn(%)'},inplace=True)
##########
xSI_Y_RtnRisk = pd.merge(xSI_Y_AnnRtn,xSI_Y_AnnStdDev,on=['index'],how='left')
xSI_Y_RtnRisk['Sharpe Ratio (Rtn/Risk)'] = xSI_Y_RtnRisk['AnnRtn(%)'] /
xSI_Y_RtnRisk['AnnStdDev(%)']

xSI_Y_RtnRisk['AnnRtn(%)'] = xSI_Y_RtnRisk['AnnRtn(%)'].astype(float).map("{:.2%}".format)
xSI_Y_RtnRisk['AnnStdDev(%)'] =
xSI_Y_RtnRisk['AnnStdDev(%)'].astype(float).map("{:.2%}".format)
xSI_Y_RtnRisk['Sharpe Ratio (Rtn/Risk)'] = xSI_Y_RtnRisk['Sharpe Ratio
(Rtn/Risk)'].astype(float).map("{:.2f}".format)

########## calc annualized return from Monthly returns ##########
#xCumRtn_Y = (1 + Y).cumprod()
xCumRtn_Y = (1 +
xDF_M[['SPLPEQTY_pct_ch_M','HFRIEMNI_pct_ch_M','SPXT_pct_ch_M','BondTR_pct_ch_M']]).cumprod()
xSI_M_AnnRtn = (xCumRtn_Y.iloc[len(xCumRtn_Y) - 1] / xCumRtn_Y.iloc[0]) ** (1 /
(len(xCumRtn_Y) / 12)) - 1
xSI_M_AnnRtn=pd.DataFrame(xSI_M_AnnRtn)
xSI_M_AnnRtn.reset_index(inplace=True)
xSI_M_AnnRtn.rename(columns={0: 'AnnRtn(%)'},inplace=True)
#######
xSI_M_AnnStdDev=pd.DataFrame(xDF_M[['SPLPEQTY_pct_ch_M','HFRIEMNI_pct_ch_M','SPXT_pct_ch_M','B
ondTR_pct_ch_M']].std())*np.sqrt(12)
```

```
xSI_M_AnnStdDev.reset_index(inplace=True)
xSI_M_AnnStdDev.rename(columns={0: 'AnnStdDev(%)'},inplace=True)

############################################
xSI_M_RtnRisk = pd.merge(xSI_M_AnnRtn,xSI_M_AnnStdDev,on=['index'],how='left')
xSI_M_RtnRisk['Sharpe Ratio (Rtn/Risk)'] = xSI_M_RtnRisk['AnnRtn(%)'] /
xSI_M_RtnRisk['AnnStdDev(%)']

xSI_M_RtnRisk['AnnRtn(%)'] = xSI_M_RtnRisk['AnnRtn(%)'].astype(float).map("{:.2%}".format)
xSI_M_RtnRisk['AnnStdDev(%)'] =
xSI_M_RtnRisk['AnnStdDev(%)'].astype(float).map("{:.2%}".format)
xSI_M_RtnRisk['Sharpe Ratio (Rtn/Risk)'] = xSI_M_RtnRisk['Sharpe Ratio
(Rtn/Risk)'].astype(float).map("{:.2f}".format)

#####################
#xSI_M_RtnRisk=pd.concat([xSI_Y_RtnRisk,xSI_M_RtnRisk],axis=0).reset_index(drop=True)
#####################
xText_RtnRisk = xSI_M_RtnRisk.to_string()
xText_corr = xDF_M[['SPLPEQTY_pct_ch_M','HFRIEMNI_pct_ch_M','SPXT_pct_ch_M','BondTR_pct_ch_M',
                    'SI2_pct_ch_M','SI4_pct_ch_M','SI6_pct_ch_M']].corr().to_string()

f = open(xDir + 'xSI_M_AnnRtnRisk_corr.txt','w')
f.write(xStartDate_M_SI.strftime('%Y/%m/%d') + ' to ' + xEndDate_M_SI.strftime('%Y/%m/%d') +
'\n\n'
        + xText_RtnRisk + '\n\n' + xText_corr)
f.close()
```

#3

```
### Portfolio Optiimization
###
#
# Finds an optimal allocation of stocks in a portfolio,
# satisfying a minimum expected return.
# The problem is posed as a Quadratic Program, and solved
# using the cvxopt library.
# Uses actual past stock data, obtained using the stocks module.
import math
import numpy as np
import pandas as pd
import datetime
import cvxopt
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt
##########################
import warnings
warnings.filterwarnings('ignore')
warnings.warn('DelftStack')
warnings.warn('Do not show this message')
#####################
solvers.options['show_progress'] = False        # !!!

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

#from cvxopt import solvers
#import stocks
import numpy
import pandas as pd

# c = cvxopt.matrix([0, -1], tc='d')
# print('c: ', c)
# c = numpy.matrix(c)
# print('c: ', c)
#
# c = cvxopt.matrix([0, -1])
# print('c: ', c)
# G = cvxopt.matrix([[-1, 1], [3, 2], [2, 3], [-1, 0], [0, -1]], tc='d')
# print('G: ', G)
##################
xDir = r'D:\\Users\\ggu\\Documents\\GU\\MeanVarianceOptimization\\'
xSPXT = pd.read_csv(xDir + 'SPXT.txt')
xSPXT['DATE'] = pd.to_datetime(xSPXT['DATE'], format='%m/%d/%Y')
xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')
xAggregateBondTR['DATE'] = pd.to_datetime(xAggregateBondTR['DATE'], format='%m/%d/%Y')

# xSI = pd.read_csv(xDir + 'SI.txt')
# xSI['DATE'] = pd.to_datetime(xSI['DATE'], format='%m/%d/%Y')

xSPX = pd.read_csv(xDir + 'SPX.txt')
xSPX['DATE'] = pd.to_datetime(xSPX['DATE'], format='%m/%d/%Y')

##xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')

print(xSPXT.head())
print(xAggregateBondTR.head())
```

```
#print(xSI.head())
print(xSPX.head())

# xSPXT = pd.merge(xSPXT, xSI, on=['DATE'], how='left')
xSPXT = pd.merge(xSPXT, xAggregateBondTR, on=['DATE'], how='left')
xSPXT = pd.merge(xSPXT, xSPX, on=['DATE'], how='left')

# xMinDateSI = xSI['DATE'].min()
# xMaxDateSI = xSI['DATE'].max()

###xSPXT = xSPXT.loc[(xSPXT['DATE'] >= xMinDateSI) & (xSPXT['DATE'] <= xMaxDateSI)]

#xSPXT['intrinsic_value'].fillna(method='ffill', inplace=True) #fill N/As with previous
prices!!!!
xSPXT['LBUSTRUU'].fillna(method='ffill', inplace=True) #fill N/As with previous prices!!!!
xSPXT['SPX'].fillna(method='ffill', inplace=True)

xSPXT['SPXT_rtn'] = xSPXT['SPXT'].pct_change()
#xSPXT['SI_rtn'] = xSPXT['intrinsic_value'].pct_change()
xSPXT['Bond_rtn'] = xSPXT['LBUSTRUU'].pct_change()
xSPXT['SPX_rtn'] = xSPXT['SPX'].pct_change()

xSPXT.to_csv(xDir + 'xSPXT.txt')

xSPXT = xSPXT.dropna()
########################
xUnderlier = 'SPX'

xDF0 = xSPXT[['DATE', xUnderlier]]
print('xDF0 = ', xDF0.head())

### These are the generic products we used in learning center.
#- 2Y, 10% hard buffer, 1.5x upside up to 21%
#- 4Y, 25% barrier, 1x upside no-cap
#- 6Y, 30% barrier, 1.15x upside no-cap
#################################################################

xCap = 1000  #0.21
xBuffer = -0.10   #-0.25
xDate = '2000-01-01'
xTerm = 2  #2  #4  #6   #4 #2  #3 # years
xAmount = 100000
xLever = 1.50  #1.15
xBufferType = "H"  #"T"  # "H" for regular Buffer; "G" for Geared Buffer (or Barrier); "T" for
Trigger Buffer!
xStartDate = datetime.date.fromisoformat(xDate)
print('start date = ', xStartDate)
xPortfolio = pd.DataFrame()

####################

xEndDate = xStartDate + datetime.timedelta(days = 365*xTerm)
print('xEndDate = ', xEndDate)
xDF = xDF0.loc[(xDF0['DATE'] >= xStartDate.strftime('%Y-%m-%d')) & (xDF0['DATE'] <=
xEndDate.strftime('%Y-%m-%d'))]
xDF.reset_index(drop=True, inplace=True)
###### in case xEndDate does NOT exist in xDF, then reassign the latest date less than the
original xEndDate ###
xEndDate = xDF.loc[xDF.index == (len(xDF)-1)]['DATE'][len(xDF)-1]
```

```
xTime = 0
xString3 = 'Structure: ' + 'Buffer Type = ' + xBufferType + '; Term = ' + (str)(xTerm) + '
years; ' + (str)(xLever) + 'x Underlier; Cap = '  + (str)(xCap) + '; Buffer = ' +
(str)(xBuffer)
xStartDate0 = xStartDate
###while (xDF.empty != True):  #this may not work properly because xStartDate = xEndDate = 1
row onlu!!!!
while (xStartDate != xEndDate):
   print('start date = ', xStartDate, ';    end date = ', xEndDate)
   xTime = xTime + 1

   xStartValue = xDF.loc[xDF.index==0][xUnderlier][0]
   xDF['CumRtn_UL'] = xDF[xUnderlier] / xStartValue - 1
   xDF['CumRtn_SI'] = xDF['CumRtn_UL'].copy()
   ############# simple buffer for DOWNSIDE ########
   xDF.loc[(xDF['CumRtn_UL']<0) & (xDF['CumRtn_UL']>xBuffer), 'CumRtn_SI'] = 0
   if (xBufferType == "H"):
      xDF.loc[(xDF['CumRtn_UL'] <= xBuffer), 'CumRtn_SI'] = xDF['CumRtn_UL'] - xBuffer
   elif (xBufferType == "T"):
      # do nothing here for trigger buffer
      print("Trigger Buffer here...")
   else:
      # do geared buffer here
      print("Geared Buffer here...")

   ############# leverage for UPSIDE ##############
   xDF.loc[(xDF['CumRtn_SI'] > 0), 'CumRtn_SI'] = xDF['CumRtn_SI'] * xLever
   ############# simple cap for UPSIDE (after LEVERAGE) ###########
   xDF.loc[(xDF['CumRtn_SI'] >= xCap), 'CumRtn_SI'] = xCap

   ############# calculate IV and Portfolio Values (PV) ########
   xDF['IV'] = xStartValue * (1 + xDF['CumRtn_SI'])
   xDF['PV_SI'] = xDF['IV'] / xStartValue * xAmount / 2
   xDF['PV_UL'] = xDF[xUnderlier] / xStartValue * xAmount / 2

   xDF['PV'] = xDF['PV_SI'] + xDF['PV_UL']
   ############### calculate daily returns ##################
   xDF[xUnderlier+'_rtn'] = xDF[xUnderlier].pct_change()
   xDF['IV_rtn'] = xDF['IV'].pct_change()
   xDF['SI_rtn'] = xDF['PV_SI'].pct_change()
   xDF['UL_rtn'] = xDF['PV_UL'].pct_change()
   xDF['PV_rtn'] = xDF['PV'].pct_change()
   ########### calculate the downside risks ################
   xDF[xUnderlier + '_rtnSQ'] = xDF[xUnderlier+'_rtn'] - xDF[xUnderlier+'_rtn'].mean()
   xDF['IV_rtnSQ'] = xDF['IV_rtn'] - xDF['IV_rtn'].mean()
   xDF['SI_rtnSQ'] = xDF['SI_rtn'] - xDF['SI_rtn'].mean()
   xDF['UL_rtnSQ'] = xDF['UL_rtn'] - xDF['UL_rtn'].mean()
   xDF['PV_rtnSQ'] = xDF['PV_rtn'] - xDF['PV_rtn'].mean()

   xDF.loc[(xDF[xUnderlier + '_rtnSQ'] > 0), xUnderlier + '_rtnSQ'] = 0
   xDF.loc[(xDF['IV_rtnSQ'] > 0), 'IV_rtnSQ'] = 0
   xDF.loc[(xDF['SI_rtnSQ'] > 0), 'SI_rtnSQ'] = 0
   xDF.loc[(xDF['UL_rtnSQ'] > 0), 'UL_rtnSQ'] = 0
   xDF.loc[(xDF['PV_rtnSQ'] > 0), 'PV_rtnSQ'] = 0

   xDF[xUnderlier + '_rtnSQ'] = xDF[xUnderlier + '_rtnSQ'] ** 2
   xDF['IV_rtnSQ'] = xDF['IV_rtnSQ'] ** 2
   xDF['SI_rtnSQ'] = xDF['SI_rtnSQ'] ** 2
   xDF['UL_rtnSQ'] = xDF['UL_rtnSQ'] ** 2
   xDF['PV_rtnSQ'] = xDF['PV_rtnSQ'] ** 2
```

```
    globals()['xDnRisk_' + xUnderlier] = np.sqrt(xDF[xUnderlier + '_rtnSQ'].mean() * 252)
    xDnRisk_IV = np.sqrt(xDF['IV_rtnSQ'].mean() * 252)
    xDnRisk_SI = np.sqrt(xDF['SI_rtnSQ'].mean() * 252)
    xDnRisk_UL = np.sqrt(xDF['UL_rtnSQ'].mean() * 252)
    xDnRisk_PV = np.sqrt(xDF['PV_rtnSQ'].mean() * 252)

    ######### calculate days and compounded returns, std of returns, correlation, sharp ratio, 
etc...###
    ######### calculate other statistics here .............
    xDF.reset_index(drop=True, inplace=True)
    xDays = (xDF.loc[xDF.index == (len(xDF)-1)]['DATE'][len(xDF)-1] - xDF.loc[xDF.index == 
0]['DATE'][0]).days
    # ############### the following has some problem by using start value and end value with 
rebalancing #####
    # ######## it must be using daily returns ###############
    # xFirst_PV_SI = xDF.loc[xDF.index == 0]['PV_SI'][0]
    # xLast_PV_SI = xDF.loc[xDF.index == (len(xDF) - 1)]['PV_SI'][len(xDF) - 1]
    # xFirst_PV_UL = xDF.loc[xDF.index == 0]['PV_UL'][0]
    # xLast_PV_UL = xDF.loc[xDF.index == (len(xDF) - 1)]['PV_UL'][len(xDF) - 1]
    # xFirst_PV = xDF.loc[xDF.index == 0]['PV'][0]
    # xLast_PV = xDF.loc[xDF.index == (len(xDF) - 1)]['PV'][len(xDF) - 1]
    #globals()['xFirst_'+xUnderlier] = xDF.loc[xDF.index == 0][xUnderlier][0]
    #globals()['xLast_'+xUnderlier] = xDF.loc[xDF.index == (len(xDF) - 1)][xUnderlier][len(xDF) 
- 1]
    #########################
    xDF['temp'] = (1 + xDF[xUnderlier + '_rtn']).cumprod()
    globals()['xCM_rtn_' + xUnderlier] = xDF['temp'][len(xDF) - 1]
    xDF['temp'] = (1 + xDF['IV_rtn']).cumprod()
    xCM_rtn_IV = xDF['temp'][len(xDF) - 1]
    xDF['temp'] = (1 + xDF['SI_rtn']).cumprod()
    xCM_rtn_SI = xDF['temp'][len(xDF) - 1]
    xDF['temp'] = (1 + xDF['UL_rtn']).cumprod()
    xCM_rtn_UL = xDF['temp'][len(xDF) - 1]
    xDF['temp'] = (1 + xDF['PV_rtn']).cumprod()
    xCM_rtn_PV = xDF['temp'][len(xDF) - 1]

    xCAGR_SI = (xCM_rtn_SI) ** (1 / (xDays / 365)) - 1
    xCAGR_UL = (xCM_rtn_UL) ** (1 / (xDays / 365)) - 1
    xCAGR_PV = (xCM_rtn_PV) ** (1 / (xDays / 365)) - 1
    globals()['xCAGR_' + xUnderlier] = (globals()['xCM_rtn_' + xUnderlier]) ** (1 / (xDays / 
365)) - 1

    xSimple_rtn_SI = xCM_rtn_SI - 1.0
    xSimple_rtn_UL = xCM_rtn_UL - 1.0
    xSimple_rtn_PV = xCM_rtn_PV - 1.0
    globals()['xSimple_rtn_' + xUnderlier] = globals()['xCM_rtn_' + xUnderlier] - 1.0

    xMean = pd.DataFrame(xDF[['SI_rtn', 'UL_rtn', 'PV_rtn', 'SPX_rtn']].mean(), 
columns=['AvgDlyRtn'])
    xStd = pd.DataFrame(xDF[['SI_rtn', 'UL_rtn', 'PV_rtn', 'SPX_rtn']].std() * math.sqrt(252), 
columns=['AnnStd'])
    xMax = pd.DataFrame(xDF[['SI_rtn', 'UL_rtn', 'PV_rtn', 'SPX_rtn']].max(), columns=['Max'])
    xMin = pd.DataFrame(xDF[['SI_rtn', 'UL_rtn', 'PV_rtn', 'SPX_rtn']].min(), columns=['Min'])

    xStats = pd.merge(xMean, xStd, left_index=True, right_index=True)
    xStats = pd.merge(xStats, xMax, left_index=True, right_index=True)
    xStats = pd.merge(xStats, xMin, left_index=True, right_index=True)

    xStats['AnnRtn'] = xCAGR_SI
```

```
    xStats['AnnRtn']['UL_rtn'] = xCAGR_UL
    xStats['AnnRtn']['PV_rtn'] = xCAGR_PV
    xStats['AnnRtn'][xUnderlier + '_rtn'] = globals()['xCAGR_' + xUnderlier]

    xStats['SimpleRtn'] = xSimple_rtn_SI
    xStats['SimpleRtn']['UL_rtn'] = xSimple_rtn_UL
    xStats['SimpleRtn']['PV_rtn'] = xSimple_rtn_PV
    xStats['SimpleRtn'][xUnderlier + '_rtn'] = globals()['xSimple_rtn_' + xUnderlier]

    xStats['AnnDnRisk'] = xDnRisk_SI
    xStats['AnnDnRisk']['UL_rtn'] = xDnRisk_UL
    xStats['AnnDnRisk']['PV_rtn'] = xDnRisk_PV
    xStats['AnnDnRisk'][xUnderlier + '_rtn'] = globals()['xDnRisk_' + xUnderlier]

    xStats['Sharpe'] = xStats['AnnRtn'] / xStats['AnnStd']
    xStats['SharpeDnRisk'] = xStats['AnnRtn'] / xStats['AnnDnRisk']

    ############# format output #############
    xStats['AvgDlyRtn'] = xStats['AvgDlyRtn'].astype(float).map("{:.3%}".format)
    xStats['AnnStd'] = xStats['AnnStd'].astype(float).map("{:.2%}".format)
    xStats['Max'] = xStats['Max'].astype(float).map("{:.2%}".format)
    xStats['Min'] = xStats['Min'].astype(float).map("{:.2%}".format)
    xStats['AnnRtn'] = xStats['AnnRtn'].astype(float).map("{:.3%}".format)
    xStats['SimpleRtn'] = xStats['SimpleRtn'].astype(float).map("{:.2%}".format)
    xStats['AnnDnRisk'] = xStats['AnnDnRisk'].astype(float).map("{:.2%}".format)

    xStats = np.round(xStats, 4)
    xCorrMatrix = np.round(xDF[['SI_rtn', (xUnderlier+'_rtn'), 'PV_rtn']].corr(), 4)

    xString0 = 'From ' + xStartDate.strftime('%m/%d/%Y') + ' to ' +
xEndDate.strftime('%m/%d/%Y')
    xString1 = xStats.astype('string')
    xString2 = xCorrMatrix.astype('string')

    xString = (str)(xString0) + '\n\n' + (str)(xString1) + '\n\n'+(str)(xString2)
    xString3 = xString3 + '\n\n' + xString
    ############ end of calculating statistics ##################
    ############################################################
    xDF.to_csv(xDir + 'xBufferIV_' + str(xTime) + '_' + xBufferType + '.txt')
    ##############
    ####### combine together while rolling over #####
    if xTime > 1:
       xDF = xDF[1:]  # remove the first row, it is duplicated with the last row of previous
xDF !!!
       xDF.reset_index(drop=True, inplace=True)
    xPortfolio = pd.concat([xPortfolio, xDF], ignore_index=True)
    ### reassign or reset for the next new SI to start ########
    xAmount = xDF['PV'][len(xDF)-1]
    xStartDate = xEndDate

    ############## NEW xDF here from the original xDF0 ###############
    xDF = xDF0.loc[xDF0['DATE']>=xStartDate.strftime('%Y-%m-%d')]
    #xDF.reset_index(drop=True, inplace=True)
    #xStartValue = xDF.loc[xDF.index == 0][xUnderlier][0]

    xEndDate = xStartDate + datetime.timedelta(days = 365 * xTerm)
    xDF = xDF.loc[(xDF['DATE'] <= xEndDate.strftime('%Y-%m-%d'))]
    xDF.reset_index(drop=True, inplace=True)
    xEndDate = xDF.loc[xDF.index == (len(xDF)-1)]['DATE'][len(xDF)-1]
```

```
##########################################################
xPortfolio.to_csv(xDir + 'xBufferIV' + '_' + xBufferType + '.txt')
if True:
   ######### calculate days and compounded returns, std of returns, correlation, sharp ratio,
etc...###
   ######### calculate other statistics here ............
   xPortfolio.reset_index(drop=True, inplace=True)
   ########### calculate the downside risks ################
   xPortfolio[xUnderlier + '_rtnSQ'] = xPortfolio[xUnderlier + '_rtn'] - xPortfolio[xUnderlier
+ '_rtn'].mean()
   xPortfolio['IV_rtnSQ'] = xPortfolio['IV_rtn'] - xPortfolio['IV_rtn'].mean()
   xPortfolio['SI_rtnSQ'] = xPortfolio['SI_rtn'] - xPortfolio['SI_rtn'].mean()
   xPortfolio['UL_rtnSQ'] = xPortfolio['UL_rtn'] - xPortfolio['UL_rtn'].mean()
   xPortfolio['PV_rtnSQ'] = xPortfolio['PV_rtn'] - xPortfolio['PV_rtn'].mean()

   xPortfolio.loc[(xPortfolio[xUnderlier + '_rtnSQ'] > 0), xUnderlier + '_rtnSQ'] = 0
   xPortfolio.loc[(xPortfolio['IV_rtnSQ'] > 0), 'IV_rtnSQ'] = 0
   xPortfolio.loc[(xPortfolio['SI_rtnSQ'] > 0), 'SI_rtnSQ'] = 0
   xPortfolio.loc[(xPortfolio['UL_rtnSQ'] > 0), 'UL_rtnSQ'] = 0
   xPortfolio.loc[(xPortfolio['PV_rtnSQ'] > 0), 'PV_rtnSQ'] = 0

   xPortfolio[xUnderlier + '_rtnSQ'] = xPortfolio[xUnderlier + '_rtnSQ'] ** 2
   xPortfolio['IV_rtnSQ'] = xPortfolio['IV_rtnSQ'] ** 2
   xPortfolio['SI_rtnSQ'] = xPortfolio['SI_rtnSQ'] ** 2
   xPortfolio['UL_rtnSQ'] = xPortfolio['UL_rtnSQ'] ** 2
   xPortfolio['PV_rtnSQ'] = xPortfolio['PV_rtnSQ'] ** 2

   globals()['xDnRisk_' + xUnderlier] = np.sqrt(xPortfolio[xUnderlier + '_rtnSQ'].mean() *
252)
   xDnRisk_IV = np.sqrt(xPortfolio['IV_rtnSQ'].mean() * 252)
   xDnRisk_SI = np.sqrt(xPortfolio['SI_rtnSQ'].mean() * 252)
   xDnRisk_UL = np.sqrt(xPortfolio['UL_rtnSQ'].mean() * 252)
   xDnRisk_PV = np.sqrt(xPortfolio['PV_rtnSQ'].mean() * 252)
   ###########################
   xDays = (xPortfolio.loc[xPortfolio.index == (len(xPortfolio)-1)]['DATE'][len(xPortfolio)-1]
- xPortfolio.loc[xPortfolio.index == 0]['DATE'][0]).days
   # xFirst_PV_SI = xPortfolio.loc[xPortfolio.index == 0]['PV_SI'][0]
   # xLast_PV_SI = xPortfolio.loc[xPortfolio.index == (len(xPortfolio) -
1)]['PV_SI'][len(xPortfolio) - 1]
   # xFirst_PV_UL = xPortfolio.loc[xPortfolio.index == 0]['PV_UL'][0]
   # xLast_PV_UL = xPortfolio.loc[xPortfolio.index == (len(xPortfolio) -
1)]['PV_UL'][len(xPortfolio) - 1]
   # xFirst_PV = xPortfolio.loc[xPortfolio.index == 0]['PV'][0]
   # xLast_PV = xPortfolio.loc[xPortfolio.index == (len(xPortfolio) -
1)]['PV'][len(xPortfolio) - 1]
   # globals()['xFirst_'+xUnderlier] = xPortfolio.loc[xPortfolio.index == 0][xUnderlier][0]
   # globals()['xLast_'+xUnderlier] = xPortfolio.loc[xPortfolio.index == (len(xPortfolio) -
1)][xUnderlier][len(xPortfolio) - 1]

   ########################
   xPortfolio['temp'] = (1 + xPortfolio[xUnderlier + '_rtn']).cumprod()
   globals()['xCM_rtn_' + xUnderlier] = xPortfolio['temp'][len(xPortfolio) - 1]
   xPortfolio['temp'] = (1 + xPortfolio['IV_rtn']).cumprod()
   xCM_rtn_IV = xPortfolio['temp'][len(xPortfolio) - 1]
   xPortfolio['temp'] = (1 + xPortfolio['SI_rtn']).cumprod()
   xCM_rtn_SI = xPortfolio['temp'][len(xPortfolio) - 1]
   xPortfolio['temp'] = (1 + xPortfolio['UL_rtn']).cumprod()
   xCM_rtn_UL = xPortfolio['temp'][len(xPortfolio) - 1]
   xPortfolio['temp'] = (1 + xPortfolio['PV_rtn']).cumprod()
   xCM_rtn_PV = xPortfolio['temp'][len(xPortfolio) - 1]
```

```
    xCAGR_SI = (xCM_rtn_SI) ** (1 / (xDays / 365)) - 1
    xCAGR_UL = (xCM_rtn_UL) ** (1 / (xDays / 365)) - 1
    xCAGR_PV = (xCM_rtn_PV) ** (1 / (xDays / 365)) - 1
    globals()['xCAGR_' + xUnderlier] = (globals()['xCM_rtn_' + xUnderlier]) ** (1 / (xDays /
365)) - 1

    xSimple_rtn_SI = xCM_rtn_SI - 1.0
    xSimple_rtn_UL = xCM_rtn_UL - 1.0
    xSimple_rtn_PV = xCM_rtn_PV - 1.0
    globals()['xSimple_rtn_' + xUnderlier] = globals()['xCM_rtn_' + xUnderlier] - 1.0

    xMean = pd.DataFrame(xPortfolio[['SI_rtn', 'UL_rtn', 'PV_rtn', 'SPX_rtn']].mean(),
columns=['AvgDlyRtn'])
    xStd = pd.DataFrame(xPortfolio[['SI_rtn', 'UL_rtn', 'PV_rtn', 'SPX_rtn']].std() *
math.sqrt(252), columns=['AnnStd'])
    xMax = pd.DataFrame(xPortfolio[['SI_rtn', 'UL_rtn', 'PV_rtn', 'SPX_rtn']].max(),
columns=['Max'])
    xMin = pd.DataFrame(xPortfolio[['SI_rtn', 'UL_rtn', 'PV_rtn', 'SPX_rtn']].min(),
columns=['Min'])

    xStats = pd.merge(xMean, xStd, left_index=True, right_index=True)
    xStats = pd.merge(xStats, xMax, left_index=True, right_index=True)
    xStats = pd.merge(xStats, xMin, left_index=True, right_index=True)

    xStats['AnnRtn'] = xCAGR_SI
    xStats['AnnRtn']['UL_rtn'] = xCAGR_UL
    xStats['AnnRtn']['PV_rtn'] = xCAGR_PV
    xStats['AnnRtn'][xUnderlier + '_rtn'] = globals()['xCAGR_' + xUnderlier]

    xStats['SimpleRtn'] = xSimple_rtn_SI
    xStats['SimpleRtn']['UL_rtn'] = xSimple_rtn_UL
    xStats['SimpleRtn']['PV_rtn'] = xSimple_rtn_PV
    xStats['SimpleRtn'][xUnderlier + '_rtn'] = globals()['xSimple_rtn_' + xUnderlier]

    xStats['AnnDnRisk'] = xDnRisk_SI
    xStats['AnnDnRisk']['UL_rtn'] = xDnRisk_UL
    xStats['AnnDnRisk']['PV_rtn'] = xDnRisk_PV
    xStats['AnnDnRisk'][xUnderlier + '_rtn'] = globals()['xDnRisk_' + xUnderlier]

    xStats['Sharpe'] = xStats['AnnRtn'] / xStats['AnnStd']
    xStats['SharpeDnRisk'] = xStats['AnnRtn'] / xStats['AnnDnRisk']
    ############# format output #############
    xStats['AvgDlyRtn'] = xStats['AvgDlyRtn'].astype(float).map("{:.3%}".format)
    xStats['AnnStd'] = xStats['AnnStd'].astype(float).map("{:.2%}".format)
    xStats['Max'] = xStats['Max'].astype(float).map("{:.2%}".format)
    xStats['Min'] = xStats['Min'].astype(float).map("{:.2%}".format)
    xStats['AnnRtn'] = xStats['AnnRtn'].astype(float).map("{:.3%}".format)
    xStats['SimpleRtn'] = xStats['SimpleRtn'].astype(float).map("{:.2%}".format)
    xStats['AnnDnRisk'] = xStats['AnnDnRisk'].astype(float).map("{:.2%}".format)

    xStats = np.round(xStats, 4)
    xCorrMatrix = np.round(xPortfolio[['SI_rtn',(xUnderlier+'_rtn'),'PV_rtn']].corr(), 4)

    xString0 = '****** From ' + xStartDate0.strftime('%m/%d/%Y') + ' to ' +
xEndDate.strftime('%m/%d/%Y') + ' ******'
    xString1 = xStats.astype('string')
    xString2 = xCorrMatrix.astype('string')

    xString = (str)(xString0) + '\n\n' + (str)(xString1) + '\n\n'+(str)(xString2)
```

```
xString3 = xString3 + '\n\n\n\n' + xString
############ end of calculating statistics ##################
#############################################################
f_w = open(xDir + 'xStats_all_' + xBufferType + '.txt','w')
f_w.write(xString3)
f_w.close()
```

#4

```
### Portfolio Optiimization
###
#
# Finds an optimal allocation of stocks in a portfolio,
# satisfying a minimum expected return.
# The problem is posed as a Quadratic Program, and solved
# using the cvxopt library.
# Uses actual past stock data, obtained using the stocks module.
import math
import sys

import numpy as np
import pandas as pd
import datetime
import cvxopt
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt

##########################
import warnings
warnings.filterwarnings('ignore')
warnings.warn('DelftStack')
warnings.warn('Do not show this message')
#####################
solvers.options['show_progress'] = False        # !!!

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

#from cvxopt import solvers
#import stocks
import numpy
import pandas as pd

# c = cvxopt.matrix([0, -1], tc='d')
# print('c: ', c)
# c = numpy.matrix(c)
# print('c: ', c)
#
# c = cvxopt.matrix([0, -1])
# print('c: ', c)
# G = cvxopt.matrix([[-1, 1], [3, 2], [2, 3], [-1, 0], [0, -1]], tc='d')
# print('G: ', G)
##################
xDir = r'D:\\Users\\ggu\\Documents\\GU\\MeanVarianceOptimization\\'
xSPXT = pd.read_csv(xDir + 'SPXT.txt')
xSPXT['DATE'] = pd.to_datetime(xSPXT['DATE'], format='%m/%d/%Y')
xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')
xAggregateBondTR['DATE'] = pd.to_datetime(xAggregateBondTR['DATE'], format='%m/%d/%Y')

# xSI = pd.read_csv(xDir + 'SI.txt')
# xSI['DATE'] = pd.to_datetime(xSI['DATE'], format='%m/%d/%Y')

xSPX = pd.read_csv(xDir + 'SPX.txt')
xSPX['DATE'] = pd.to_datetime(xSPX['DATE'], format='%m/%d/%Y')

##xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')
```

```
print(xSPXT.head())
print(xAggregateBondTR.head())
#print(xSI.head())
print(xSPX.head())

# xSPXT = pd.merge(xSPXT, xSI, on=['DATE'], how='left')
xSPXT = pd.merge(xSPXT, xAggregateBondTR, on=['DATE'], how='left')
xSPXT = pd.merge(xSPXT, xSPX, on=['DATE'], how='left')

# xMinDateSI = xSI['DATE'].min()
# xMaxDateSI = xSI['DATE'].max()

###xSPXT = xSPXT.loc[(xSPXT['DATE'] >= xMinDateSI) & (xSPXT['DATE'] <= xMaxDateSI)]

#xSPXT['intrinsic_value'].fillna(method='ffill', inplace=True) #fill N/As with previous
prices!!!!
xSPXT['LBUSTRUU'].fillna(method='ffill', inplace=True) #fill N/As with previous prices!!!!
xSPXT['SPX'].fillna(method='ffill', inplace=True)

xSPXT['SPXT_rtn'] = xSPXT['SPXT'].pct_change()
#xSPXT['SI_rtn'] = xSPXT['intrinsic_value'].pct_change()
xSPXT['BondTR_rtn'] = xSPXT['LBUSTRUU'].pct_change()
xSPXT['SPX_rtn'] = xSPXT['SPX'].pct_change()

xSPXT.to_csv(xDir + 'xDailyIndexes.txt')

xSPXT.rename(columns={'LBUSTRUU': 'BondTR'},inplace=True)

xSPXT = xSPXT.dropna()
########################
xUnderlier = 'SPX'  #'SPXT'   #'SPX'

#xDF0 = xSPXT[['DATE', xUnderlier]]
xDF0 = xSPXT[['DATE', 'SPX', 'SPXT', 'SPX_rtn', 'SPXT_rtn','BondTR_rtn','BondTR']]
print('xDF0 = ', xDF0.head())

####### These are the generic products we used in learning center. ####
#- 2Y, 10% hard buffer, 1.5x upside up to 21%
#- 4Y, 25% barrier, 1x upside no-cap
#- 6Y, 30% barrier, 1.15x upside no-cap
#######################################################################

xCap = 0.21 #0.21 #0.21  #1000 #0.21   #1000  #0.21
xBuffer = -0.10000  ####-0.10  #-0.25   #-0.30   #-0.25
xTerm = 2  #2  #4  #6   #4 #2  #3 # years
xAmount = 100
xLever = 1.500  #1.5  #1.15
xBufferType = "H"  #"T"  # "H" for regular Buffer; "G" for Geared Buffer (or Barrier); "T" for
Trigger Buffer!
###########################

xTime = 0
xString3 = 'Structure: ' + 'Buffer Type = ' + xBufferType + '; Term = ' + str(xTerm) + '
years; ' + (str)(xLever) + 'x ' + xUnderlier + '; Cap = '  + (str)(xCap) + '; Buffer = ' +
(str)(xBuffer)
xLastDate = xDF0['DATE'].max()
xStartDate0 = xDF0['DATE'].min()
xDF0[xUnderlier + '_UL_rtn_term'] = np.nan
xDF0['SI_rtn_term'] = np.nan
```

```
xDF0['SPXT_rtn_term'] = np.nan
xDF0['BondTR_rtn_term'] = np.nan
xDF0['SPX_rtn_term'] = np.nan

xNew = 1
##xDF0[xUnderlier + '_rtn'] = xDF0[xUnderlier].pct_change()
### debug
#xLastDate = xDF0['DATE'][10]

xTempDF = pd.DataFrame()
######### this way to get xStartDate; it will NOT miss a single date!!!!
xDF0['StartDate'] = xDF0['DATE'].shift(xTerm * 252)    # assume 252 trading days per year ####
############################################
for xTempDate in xDF0['DATE']:
   xEndDate = xTempDate
   #####xStartDate = xEndDate + datetime.timedelta(days=-365 * xTerm)
   xStartDate = xDF0.loc[xDF0['DATE'] == xEndDate]['StartDate'].values[0]
   #if (xStartDate < xStartDate0):
   xDF0.loc[xDF0['DATE'] == xTempDate, 'SI_Cycle'] = xNew
   if pd.isna(xStartDate):
      #sys.exit()
      #break
      continue
   xDF = xDF0.loc[(xDF0['DATE'] >= xStartDate) & (xDF0['DATE'] <= xEndDate)]
   xDF.reset_index(drop=True, inplace=True)
   #xEndDate = xDF['DATE'][len(xDF) - 1]
   xStartDate = xDF['DATE'][0]
   xTime = xTime + 1

   #xDF0['TradingDays'] = len(xDF) - 1
   xDF0.loc[(xDF0['DATE'] >= xStartDate) & (xDF0['DATE'] <= xEndDate), 'TradingDays'] =
len(xDF) - 1

   xStartValue = xDF[xUnderlier][0]
   xEndValue = xDF[xUnderlier][len(xDF)-1]
   xPctChange = xEndValue / xStartValue - 1.0

   xStartValueBondTR = xDF['BondTR'][0]
   xEndValueBondTR = xDF['BondTR'][len(xDF) - 1]
   xPctChangeBondTR = xEndValueBondTR / xStartValueBondTR - 1.0

   xStartValueSPXT = xDF['SPXT'][0]
   xEndValueSPXT = xDF['SPXT'][len(xDF) - 1]
   xPctChangeSPXT = xEndValueSPXT / xStartValueSPXT - 1.0

   xStartValueSPX = xDF['SPX'][0]
   xEndValueSPX = xDF['SPX'][len(xDF) - 1]
   xPctChangeSPX = xEndValueSPX / xStartValueSPX - 1.0

   print('start date = ', xStartDate, ';    end date = ', xEndDate, '; pch change = ',
xPctChange)

   xDF0.loc[xDF0['DATE'] == xEndDate, xUnderlier + '_UL_rtn_term'] = xPctChange
   xDF0.loc[xDF0['DATE'] == xEndDate, 'BondTR_rtn_term'] = xPctChangeBondTR
   xDF0.loc[xDF0['DATE'] == xEndDate, 'SPXT_rtn_term'] = xPctChangeSPXT
   xDF0.loc[xDF0['DATE'] == xEndDate, 'SPX_rtn_term'] = xPctChangeSPX

   if (xBufferType == 'T'):
      if (xPctChange < xBuffer):
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = xPctChange
```

```
      elif (xPctChange <= 0):
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = 0
      elif (xPctChange * xLever > xCap):  #(((xPctChange + 1) * xLever - 1)> xCap):
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = xCap
      else:
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = xPctChange * xLever
   elif (xBufferType == 'H'):
      if (xPctChange < xBuffer):
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = xPctChange - xBuffer
      elif (xPctChange <= 0):
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = 0
      elif (xPctChange * xLever > xCap):  #(((xPctChange + 1) * xLever - 1)> xCap):
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = xCap
      else:
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = xPctChange * xLever
   elif (xBufferType == 'G'):
      if (xPctChange < xBuffer):
         xK = 1 / (1 + xBuffer)  # 100/(100-30) = 10/7
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = xK * (xPctChange - xBuffer)
      elif (xPctChange <= 0):
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = 0
      elif (xPctChange * xLever > xCap):  #(((xPctChange + 1) * xLever - 1)> xCap):
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = xCap
      else:
         xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term'] = xPctChange * xLever

###############################################################################################
####################
   #i could have added here to calculate the single SI growth over the entire history (20 or
30 years).  but this calculation
   # does not look good because it depends on where/when this specific single SI started!!!!
the start date and the maturity
   # date for this single SI can be critical and is of NO representative power.  the same is
true to calculate the valuation of
   # this specific single SI, the value does NOT have any representative power.
   if xNew == 1:
      xPreviousMaturityDate = xStartDate
      xDF0.loc[xDF0['DATE'] == xStartDate, 'MaturityDate'] = xStartDate
      xDF0.loc[xDF0['DATE'] == xStartDate, 'LaunchDate'] = xStartDate
      xDF0.loc[xDF0['DATE'] == xStartDate, 'SI_Cycle'] = 0   #over this first date
      ###xNew = xNew + 1
   if xStartDate == xPreviousMaturityDate:   # note: it is funny that maybe there is no
xStartDate = xPreviousMaturityDate!!!!
      xDF0.loc[xDF0['DATE'] == xEndDate, 'MaturityDate'] = xEndDate
      xDF0.loc[xDF0['DATE'] == xEndDate, 'LaunchDate'] = xStartDate
      xDays = len(xDF) - 1
      xDF0.loc[xDF0['DATE'] == xEndDate, 'SI_rtn_term_specific'] = \
         xDF0.loc[xDF0['DATE'] == xEndDate]['SI_rtn_term'].values[0]
      xDF0.loc[(xDF0['DATE'] > xStartDate) & (xDF0['DATE'] < xEndDate), \
             'SI_rtn_term_specific'] = 0
      xTempCmpRtn = (1 + xDF0.loc[xDF0['DATE'] == xEndDate]['SI_rtn_term'].values[0])**(1 /
xDays) - 1.0
      xDF0.loc[(xDF0['DATE'] > xStartDate) & (xDF0['DATE'] <= xEndDate), \
             'SI_rtn_term_specific_cmp'] = xTempCmpRtn
      xPreviousMaturityDate = xEndDate
      xNew = xNew + 1
   xTempDF = xTempDF.append({'start_date':xStartDate, 'end_date':xEndDate,
'previous_maturity_date':xPreviousMaturityDate}, \
                      ignore_index=True)
###############################################################################################
```

```
##################################
if (True):
   xDF0['SI_DailyRtn'] = (1 + xDF0['SI_rtn_term'])**(1 / (252*xTerm)) - 1.0
   xDF0[xUnderlier + '_UL_DailyRtn'] = (1 + xDF0[xUnderlier + '_UL_rtn_term'])**(1 /
(252*xTerm)) - 1.0
   xDF0['BondTR_DailyRtn'] = (1 + xDF0['BondTR_rtn_term'])**(1 / (252*xTerm)) - 1.0
   xDF0['SPXT_DailyRtn'] = (1 + xDF0['SPXT_rtn_term']) ** (1 / (252 * xTerm)) - 1.0
   xDF0['SPX_DailyRtn'] = (1 + xDF0['SPX_rtn_term']) ** (1 / (252 * xTerm)) - 1.0
else:
   ##### by using actural number of trading days ##########
   xDF0['SI_DailyRtn'] = (1 + xDF0['SI_rtn_term'])**(1 / (xDF0['TradingDays'])) - 1.0
   xDF0[xUnderlier + '_UL_DailyRtn'] = (1 + xDF0[xUnderlier + '_UL_rtn_term'])**(1 /
(xDF0['TradingDays'])) - 1.0
   xDF0['BondTR_DailyRtn'] = (1 + xDF0['BondTR_rtn_term'])**(1 / (xDF0['TradingDays'])) - 1.0
   xDF0['SPXT_DailyRtn'] = (1 + xDF0['SPXT_rtn_term']) ** (1 / (xDF0['TradingDays'])) - 1.0
   xDF0['SPX_DailyRtn'] = (1 + xDF0['SPX_rtn_term']) ** (1 / (xDF0['TradingDays'])) - 1.0
#############################################################################################
##################################
############################ i immediately calculate the equivalent 1 year return
#########################
xDF0['SI_rtn_1_year'] = (1 + xDF0['SI_rtn_term']) ** (1 / xTerm) - 1
xDF0['SPXT_rtn_1_year_roll'] = (1 + xDF0['SPXT_rtn_term']) ** (1 / xTerm) - 1
xDF0['BondTR_rtn_1_year_roll'] = (1 + xDF0['BondTR_rtn_term']) ** (1 / xTerm) - 1
##################
#xDF0.to_csv(xDir + 'xCalcRtnsOverTerm.txt')

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
xDF0['Year'] = xDF0['DATE'].dt.year
xDF0['cum_rtn_SPX'] = xDF0.groupby('Year')['SPX_rtn'].apply(lambda x: np.cumprod(1 + x) - 1)
xDF0['cum_rtn_SPXT'] = xDF0.groupby('Year')['SPXT_rtn'].apply(lambda x: np.cumprod(1 + x) - 1)
xDF0['cum_rtn_SI'] = xDF0.groupby('Year')['SI_DailyRtn'].apply(lambda x: np.cumprod(1 + x) -
1)
xDF0['cum_rtn2_UL_' + xUnderlier] = xDF0.groupby('Year')[xUnderlier +
'_UL_DailyRtn'].apply(lambda x: np.cumprod(1 + x) - 1)
xDF0['cum_rtn_BondTR'] = xDF0.groupby('Year')['BondTR_rtn'].apply(lambda x: np.cumprod(1 + x)
- 1)
xDF0['cum_rtn2_BondTR'] = xDF0.groupby('Year')['BondTR_DailyRtn'].apply(lambda x: np.cumprod(1
+ x) - 1)
xDF0['cum_rtn2_SPXT'] = xDF0.groupby('Year')['SPXT_DailyRtn'].apply(lambda x: np.cumprod(1 +
x) - 1)
xDF0['cum_rtn2_SPX'] = xDF0.groupby('Year')['SPX_DailyRtn'].apply(lambda x: np.cumprod(1 + x)
- 1)

################## calculate 1 year returns for SPXT, BondTR ##############
################## SI_rtn_1_year was already done #########################
xDF0['SPXT_rtn_1_year'] = xDF0['SPXT'].pct_change(252)
xDF0['BondTR_rtn_1_year'] = xDF0['BondTR'].pct_change(252)
#### this following is for the ENTIRE term period #######
xDF2 = xDF0[['DATE', 'SPXT_rtn', 'SI_DailyRtn', 'BondTR_rtn', 'SPX_rtn', xUnderlier +
'_UL_DailyRtn', \
          'BondTR_DailyRtn', 'SPXT_DailyRtn', 'SPX_DailyRtn']].copy()
xDF2.dropna(inplace=True)
xDF2['SPXT_cum'] = (1+xDF2['SPXT_rtn']).cumprod()
xDF2['SPX_cum'] = (1+xDF2['SPX_rtn']).cumprod()
xDF2['SI_cum'] = (1+xDF2['SI_DailyRtn']).cumprod()
xDF2['BondTR_cum'] = (1+xDF2['BondTR_rtn']).cumprod()
xDF2[xUnderlier + '_UL_cum2'] = (1+xDF2[xUnderlier + '_UL_DailyRtn']).cumprod()
xDF2['BondTR_cum2'] = (1+xDF2['BondTR_DailyRtn']).cumprod()
xDF2['SPXT_cum2'] = (1+xDF2['SPXT_DailyRtn']).cumprod()
xDF2['SPX_cum2'] = (1+xDF2['SPX_DailyRtn']).cumprod()
```

```
xFirstDate = xDF2['DATE'].min()
xFirstDate = xFirstDate + datetime.timedelta(days=-1)

#### initial values of $1 #############
xNew_row = pd.DataFrame([[xFirstDate, 1, 1, 1, 1, 1, 1, 1, 1]], columns=['DATE', 'SPXT_cum',
'SI_cum','BondTR_cum', \
                              xUnderlier + '_UL_cum2',
'SPX_cum','BondTR_cum2','SPXT_cum2','SPX_cum2'])
xDF2 = pd.concat([xDF2, pd.DataFrame(xNew_row)], ignore_index=True)
xDF2.sort_values(by=['DATE'], ascending=True,inplace=True)
xDF2.reset_index(drop=True,inplace=True)

xChartTitle = xString3
xDF2.plot(x='DATE', y=['SI_cum','SPXT_cum','BondTR_cum', xUnderlier + '_UL_cum2',
'SPX_cum','BondTR_cum2','SPXT_cum2','SPX_cum2'])
plt.title('Performance Comparison: SI vs Equity and Bond\n' + xChartTitle, fontsize=9,
ha='center')
#plt.figtext(0.5,0.9,'Performance Comparison: SI vs S&P 500 Index (TR)', fontsize=15,
ha='center')
#plt.figtext(0.5,0.8,xString3,fontsize=9,ha='center')
#plt.subplot().yaxis.set_major_formatter('${x:1.2f}')
plt.subplot().yaxis.set_major_formatter('${x:1.0f}')
plt.minorticks_on()
plt.grid(which='both')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Investment Growth')
plt.savefig(xDir + 'xPerformanceChart_' + xUnderlier + '.png')
#plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()

############################
xDF0['Year_diff'] = xDF0['Year'] - xDF0['Year'].shift(-1)

################### calculate portfolio value with equal weightings betweem SI and SPXT
##############
xDF0a = xDF0.loc[xDF0['SI_DailyRtn'].isna()][['DATE', 'SPXT_rtn', 'SI_DailyRtn','BondTR_rtn',
'SPX_rtn', \
                        xUnderlier +
'_UL_DailyRtn','BondTR_DailyRtn','SPXT_DailyRtn','SPX_DailyRtn']].copy()
xLastDateNA = xDF0a['DATE'].max()

xDF0a = xDF0.loc[xDF0['DATE'] >= xLastDateNA][['DATE', 'Year', 'SPXT_rtn', 'SI_DailyRtn',
'BondTR_rtn', \
               'SPX_rtn', xUnderlier +
'_UL_DailyRtn','BondTR_DailyRtn','SPXT_DailyRtn','SPX_DailyRtn','Year_diff']].copy()

xDF0a['PV_SI'] = np.nan
xDF0a['PV_SPXT'] = np.nan
xDF0a['PV'] = np.nan
xDF0a['SPXT_100'] = np.nan
xDF0a['SI_100'] = np.nan
xDF0a['BondTR_100'] = np.nan
xDF0a[xUnderlier + '_UL_term_100'] = np.nan
xDF0a['SPX_100'] = np.nan
xDF0a['BondTR_term_100'] = np.nan
xDF0a['SPXT_term_100'] = np.nan
xDF0a['SPX_term_100'] = np.nan
```

```
################
########## calculate two portfolios ######
### 1) 70% Equity SPXT and 30% Aggr Bond ####
### 2) 70% Equity SPXT and 15% Aggr Bond and 15% SI ####
xDF02 = xDF0.loc[xDF0['SI_DailyRtn'].isna()][['DATE', 'SPXT_rtn', 'SI_DailyRtn','BondTR_rtn',
\
                        'BondTR_DailyRtn','SPXT_DailyRtn','SPX_DailyRtn']].copy()
xLastDateNA = xDF02['DATE'].max()
xDF02 = xDF0.loc[xDF0['DATE'] >= xLastDateNA][['DATE', 'Year', 'SPXT_rtn', 'SI_DailyRtn',
'BondTR_rtn', \
                              'BondTR_DailyRtn','SPXT_DailyRtn','SPX_DailyRtn']].copy()
xDF02.reset_index(drop=True, inplace=True)

xDates = xDF02[['DATE']].copy()

xP1W1_EQ=0.70
xP1W2_BD=0.30
xDF02['PV1'] = np.nan
xDF02['PV1_SPXT'] = np.nan
xDF02['PV1_Bond'] = np.nan

xP2W1_EQ=0.70
xP2W2_BD=0.15
xP2W3_SI=0.15
xDF02['PV2'] = np.nan
xDF02['PV2_SPXT'] = np.nan
xDF02['PV2_SI'] = np.nan
xDF02['PV2_Bond'] = np.nan

xDF02['PV_SPXT_100'] = np.nan
xDF02['PV_BondTR_100'] = np.nan

xTime = 1
for xTempDate in xDF0a['DATE']:
   print(xTempDate)
   xThisDate = xTempDate
   xThisYear = xThisDate.year
   if (xTime == 1):   # on the start date
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'PV_SI'] = xAmount / 2
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'PV_SPXT'] = xAmount / 2
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'PV'] = xAmount

      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SPXT_100'] = xAmount
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SPX_100'] = xAmount
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SI_100'] = xAmount
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'BondTR_100'] = xAmount
      xDF0a.loc[xDF0a['DATE'] == xThisDate, xUnderlier + '_UL_term_100'] = xAmount
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'BondTR_term_100'] = xAmount
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SPXT_term_100'] = xAmount
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SPX_term_100'] = xAmount

      #################### two portfolos ###################
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_Bond'] = xAmount * xP1W2_BD
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_SPXT'] = xAmount * xP1W1_EQ
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1'] = xAmount

      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SPXT'] = xAmount * xP2W1_EQ
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_Bond'] = xAmount * xP2W2_BD
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SI'] = xAmount * xP2W3_SI
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2'] = xAmount
```

```
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV_SPXT_100'] = xAmount
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV_BondTR_100'] = xAmount

      xExpirationDate = xThisDate + datetime.timedelta(days=365 * xTerm)
      xExpirationDate = xDates.loc[xDates['DATE'] <= xExpirationDate]['DATE'].max() ## set the
expiraton = a trading date
      ######################################################
      xTime = xTime + 1
      xPreviousDate = xThisDate
      xPreviousYear = xPreviousDate.year
      continue
   else:
      xPV_SI_PreviousDate = xDF0a[xDF0a['DATE'] == xPreviousDate]['PV_SI'].values[0]
      xPV_SPXT_PreviousDate = xDF0a[xDF0a['DATE'] == xPreviousDate]['PV_SPXT'].values[0]
      xPV_PreviousDate = xDF0a[xDF0a['DATE'] == xPreviousDate]['PV'].values[0]

      xSPXT_100_PreviousDate = xDF0a[xDF0a['DATE'] == xPreviousDate]['SPXT_100'].values[0]
      xSI_100_PreviousDate = xDF0a[xDF0a['DATE'] == xPreviousDate]['SI_100'].values[0]
      xBondTR_100_PreviousDate = xDF0a[xDF0a['DATE'] == xPreviousDate]['BondTR_100'].values[0]
      xSPX_100_PreviousDate = xDF0a[xDF0a['DATE'] == xPreviousDate]['SPX_100'].values[0]
      globals()[xUnderlier + '_UL_term_100_PreviousDate'] = xDF0a[xDF0a['DATE'] ==
xPreviousDate][xUnderlier + '_UL_term_100'].values[0]
      xBondTR_term_100_PreviousDate = xDF0a[xDF0a['DATE'] ==
xPreviousDate]['BondTR_term_100'].values[0]
      xSPXT_term_100_PreviousDate = xDF0a[xDF0a['DATE'] ==
xPreviousDate]['SPXT_term_100'].values[0]
      xSPX_term_100_PreviousDate = xDF0a[xDF0a['DATE'] ==
xPreviousDate]['SPX_term_100'].values[0]

      xPV_SI_ThisDate = xPV_SI_PreviousDate * (1 + xDF0a[xDF0a['DATE'] ==
xThisDate]['SI_DailyRtn'].values[0])
      xPV_SPXT_ThisDate = xPV_SPXT_PreviousDate * (1 + xDF0a[xDF0a['DATE'] ==
xThisDate]['SPXT_rtn'].values[0])
      xPV_ThisDate = xPV_SI_ThisDate + xPV_SPXT_ThisDate

      xSPXT_100_ThisDate = xSPXT_100_PreviousDate * (1 + xDF0a[xDF0a['DATE'] ==
xThisDate]['SPXT_rtn'].values[0])
      xSI_100_ThisDate = xSI_100_PreviousDate * (1 + xDF0a[xDF0a['DATE'] ==
xThisDate]['SI_DailyRtn'].values[0])
      xBondTR_100_ThisDate = xBondTR_100_PreviousDate * (1 + xDF0a[xDF0a['DATE'] ==
xThisDate]['BondTR_rtn'].values[0])
      xSPX_100_ThisDate = xSPX_100_PreviousDate * (1 + xDF0a[xDF0a['DATE'] ==
xThisDate]['SPX_rtn'].values[0])
      globals()[xUnderlier + '_UL_term_100_ThisDate'] = globals()[xUnderlier +
'_UL_term_100_PreviousDate'] * \
                           (1 + xDF0a[xDF0a['DATE'] == xThisDate][xUnderlier +
'_UL_DailyRtn'].values[0])
      xBondTR_term_100_ThisDate = xBondTR_term_100_PreviousDate * \
                           (1 + xDF0a[xDF0a['DATE'] ==
xThisDate]['BondTR_DailyRtn'].values[0])
      xSPXT_term_100_ThisDate = xSPXT_term_100_PreviousDate * (
            1 + xDF0a[xDF0a['DATE'] == xThisDate]['SPXT_DailyRtn'].values[0])
      xSPX_term_100_ThisDate = xSPX_term_100_PreviousDate * (
            1 + xDF0a[xDF0a['DATE'] == xThisDate]['SPX_DailyRtn'].values[0])

      ######if (xPreviousYear != xThisYear):
      ### rebalanced on the last date of the year
      if (len(xDF0a.loc[((xDF0a['DATE'] == xThisDate) & (xDF0a['Year_diff'] == -1))]) != 0):
         xPV_SI_ThisDate = xPV_ThisDate / 2
```

```
         xPV_SPXT_ThisDate = xPV_ThisDate / 2
      ################################

      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'PV_SI'] = xPV_SI_ThisDate
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'PV_SPXT'] = xPV_SPXT_ThisDate
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'PV'] = xPV_ThisDate

      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SI_100'] = xSI_100_ThisDate
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SPXT_100'] = xSPXT_100_ThisDate
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'BondTR_100'] = xBondTR_100_ThisDate
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SPX_100'] = xSPX_100_ThisDate
      xDF0a.loc[xDF0a['DATE'] == xThisDate, xUnderlier + '_UL_term_100'] =
globals()[xUnderlier + '_UL_term_100_ThisDate']
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'BondTR_term_100'] = xBondTR_term_100_ThisDate
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SPXT_term_100'] = xSPXT_term_100_ThisDate
      xDF0a.loc[xDF0a['DATE'] == xThisDate, 'SPX_term_100'] = xSPX_term_100_ThisDate

      #################### two portfolios ##################
      xPV1_Bond_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV1_Bond'].values[0]
      xPV1_SPXT_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV1_SPXT'].values[0]
      xPV1_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV1'].values[0]

      xPV2_SPXT_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2_SPXT'].values[0]
      xPV2_SI_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2_SI'].values[0]
      xPV2_Bond_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2_Bond'].values[0]
      xPV2_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2'].values[0]

      xPV_SPXT_100_PreviousDate = xDF02[xDF02['DATE'] ==
xPreviousDate]['PV_SPXT_100'].values[0]
      xPV_BondTR_100_PreviousDate = xDF02[xDF02['DATE'] ==
xPreviousDate]['PV_BondTR_100'].values[0]

      xPV1_Bond_ThisDate = xPV1_Bond_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['BondTR_rtn'].values[0])
      xPV1_SPXT_ThisDate = xPV1_SPXT_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SPXT_rtn'].values[0])
      xPV1_ThisDate = xPV1_Bond_ThisDate + xPV1_SPXT_ThisDate

      xPV2_SPXT_ThisDate = xPV2_SPXT_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SPXT_rtn'].values[0])
      xPV2_SI_ThisDate = xPV2_SI_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SI_DailyRtn'].values[0])
      xPV2_Bond_ThisDate = xPV2_Bond_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['BondTR_rtn'].values[0])
      xPV2_ThisDate = xPV2_SPXT_ThisDate + xPV2_Bond_ThisDate + xPV2_SI_ThisDate

      xPV_SPXT_100_ThisDate = xPV_SPXT_100_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SPXT_rtn'].values[0])
      xPV_BondTR_100_ThisDate = xPV_BondTR_100_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['BondTR_rtn'].values[0])

      ### rebalanced on expiration date #########
      if (xThisDate == xExpirationDate):
         xPV1_SPXT_ThisDate = xPV1_ThisDate * xP1W1_EQ
         xPV1_Bond_ThisDate = xPV1_ThisDate * xP1W2_BD

         xPV2_SPXT_ThsDate = xPV2_ThisDate * xP2W1_EQ
         xPV2_Bond_ThsDate = xPV2_ThisDate * xP2W2_BD
         xPV2_SI_ThsDate = xPV2_ThisDate * xP2W3_SI
```

```
        xExpirationDate = xThisDate + datetime.timedelta(days=365 * xTerm)
        xExpirationDate = xDates.loc[xDates['DATE'] <= xExpirationDate]['DATE'].max()  ## set
the expiraton = a trading date
      ################################

      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_SPXT'] = xPV1_SPXT_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_Bond'] = xPV1_Bond_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1'] = xPV1_ThisDate

      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SPXT'] = xPV2_SPXT_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_Bond'] = xPV2_Bond_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SI'] = xPV2_SI_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2'] = xPV2_ThisDate

      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV_SPXT_100'] = xPV_SPXT_100_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV_BondTR_100'] = xPV_BondTR_100_ThisDate
      ####################################################
      xTime = xTime + 1
      xPreviousDate = xThisDate
      xPreviousYear = xPreviousDate.year

xDF0a = pd.merge(xDF0a, xDF02[['DATE','PV1','PV2']], on=['DATE'],how='left')
xDF0a['PV_rtn'] = xDF0a['PV'].pct_change()
xDF0a['cum_rtn_PV'] = xDF0a.groupby('Year')['PV_rtn'].apply(lambda x: np.cumprod(1 + x) - 1)
xDF0a['PV1_rtn'] = xDF0a['PV1'].pct_change()
xDF0a['cum_rtn_PV1'] = xDF0a.groupby('Year')['PV1_rtn'].apply(lambda x: np.cumprod(1 + x) - 1)
xDF0a['PV2_rtn'] = xDF0a['PV2'].pct_change()
xDF0a['cum_rtn_PV2'] = xDF0a.groupby('Year')['PV2_rtn'].apply(lambda x: np.cumprod(1 + x) - 1)

xDF0 = pd.merge(xDF0, xDF0a[['DATE', 'PV_SI', 'PV_SPXT', 'PV', 'PV_rtn', 'cum_rtn_PV',\
                     'SI_100', 'SPXT_100', 'BondTR_100', 'SPX_100', xUnderlier +
'_UL_term_100',\
                      'cum_rtn_PV1','cum_rtn_PV2', 'PV1_rtn',
'PV2_rtn','PV1','PV2','BondTR_term_100',\
                      'SPXT_term_100', 'SPX_term_100']], on=['DATE'], how='left')

#################### calculate statisics ###############
xAnnRtnByYear = xDF0.loc[xDF0['Year_diff'] != 0][['Year', 'cum_rtn_SI', 'cum_rtn_SPXT',
'cum_rtn_PV', 'cum_rtn_BondTR', \
                                      'cum_rtn_SPX', 'cum_rtn2_UL_' +
xUnderlier,'cum_rtn_PV1','cum_rtn_PV2', \
                                      'cum_rtn2_BondTR', 'cum_rtn2_SPXT', 'cum_rtn2_SPX']]
xAnnRtnByYear.dropna(inplace=True)
xAnnRtnByYear.reset_index(drop=True, inplace=True)

xStdByYear = xDF0.groupby('Year')['SI_DailyRtn', 'SPXT_rtn', 'PV_rtn', 'BondTR_rtn','SPX_rtn',
\
         xUnderlier + '_UL_DailyRtn','PV1_rtn','PV2_rtn', 'BondTR_DailyRtn', 'SPXT_DailyRtn',
\
                          'SPX_DailyRtn'].std() * np.sqrt(252)
xCorrByYear = xDF0.groupby('Year')['SI_DailyRtn', 'SPXT_rtn', 'BondTR_rtn', 'SPX_rtn',
xUnderlier + '_UL_DailyRtn',\

'PV1_rtn','PV2_rtn','BondTR_DailyRtn','SPXT_DailyRtn','SPX_DailyRtn'].corr()
xCorrByYear = xCorrByYear.round(4)

xStdByYear.reset_index(drop=False, inplace=True)
xCorrByYear.reset_index(drop=False, inplace=True)

xStdByYear.dropna(inplace=True)
```

```
xCorrByYear.dropna(inplace=True)

xStdByYear.reset_index(drop=True, inplace=True)
xCorrByYear.reset_index(drop=True, inplace=True)

xStdByYear.rename(columns={'SI_DailyRtn': 'SI_AnnStd', 'SPXT_rtn': 'SPXT_AnnStd', 'PV_rtn': 
'PV_AnnStd',\
   'BondTR_rtn': 'BondTR_AnnStd', 'SPX_rtn': 'SPX_AnnStd', xUnderlier + '_UL_DailyRtn': 
xUnderlier + '_UL_AnnStd2', \
   'PV1_rtn': 'PV1_AnnStd','PV2_rtn':'PV2_AnnStd', 'BondTR_DailyRtn': 'BondTR_AnnStd2', \
                      'SPXT_DailyRtn': 'SPXT_AnnStd2', 'SPX_DailyRtn': 
'SPX_AnnStd2'},inplace=True)

xAnnRtnByYear = pd.merge(xAnnRtnByYear, xStdByYear, on=['Year'], how='left')

xAnnRtnByYear['Sharpe_SI'] = xAnnRtnByYear['cum_rtn_SI'] / xAnnRtnByYear['SI_AnnStd']
xAnnRtnByYear['Sharpe_PV'] = xAnnRtnByYear['cum_rtn_PV'] / xAnnRtnByYear['PV_AnnStd']
xAnnRtnByYear['Sharpe_SPXT'] = xAnnRtnByYear['cum_rtn_SPXT'] / xAnnRtnByYear['SPXT_AnnStd']
xAnnRtnByYear['Sharpe_BondTR'] = xAnnRtnByYear['cum_rtn_BondTR'] / 
xAnnRtnByYear['BondTR_AnnStd']
xAnnRtnByYear['Sharpe_SPX'] = xAnnRtnByYear['cum_rtn_SPX'] / xAnnRtnByYear['SPX_AnnStd']
xAnnRtnByYear['Sharpe2_UL_' + xUnderlier] = xAnnRtnByYear['cum_rtn2_UL_' + xUnderlier] / 
xAnnRtnByYear[xUnderlier + '_UL_AnnStd2']
xAnnRtnByYear['Sharpe_PV1'] = xAnnRtnByYear['cum_rtn_PV1'] / xAnnRtnByYear['PV1_AnnStd']
xAnnRtnByYear['Sharpe_PV2'] = xAnnRtnByYear['cum_rtn_PV2'] / xAnnRtnByYear['PV2_AnnStd']
xAnnRtnByYear['Sharpe2_BondTR'] = xAnnRtnByYear['cum_rtn2_BondTR'] / 
xAnnRtnByYear['BondTR_AnnStd2']
xAnnRtnByYear['Sharpe2_SPXT'] = xAnnRtnByYear['cum_rtn2_SPXT'] / xAnnRtnByYear['SPXT_AnnStd2']
xAnnRtnByYear['Sharpe2_SPX'] = xAnnRtnByYear['cum_rtn2_SPX'] / xAnnRtnByYear['SPX_AnnStd2']

xAnnRtnByYear.rename(columns={'cum_rtn_SI':'SI_AnnRtn', 'cum_rtn_SPXT':'SPXT_AnnRtn', 
'cum_rtn_PV':'PV_AnnRtn', \
   'cum_rtn_BondTR':'BondTR_AnnRtn', 'cum_rtn_SPX':'SPX_AnnRtn', 'cum_rtn2_UL_' + xUnderlier: 
xUnderlier + '_UL_AnnRtn2', \
   'cum_rtn_PV1':'PV1_AnnRtn', 'cum_rtn_PV2':'PV2_AnnRtn', 'cum_rtn2_BondTR':'BondTR_AnnRtn2', 
\
                        'cum_rtn2_SPXT':'SPXT_AnnRtn2', 
'cum_rtn2_SPX':'SPX_AnnRtn2'},inplace=True)

xAnnRtnByYear['SI_AnnRtn'] = xAnnRtnByYear['SI_AnnRtn'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['PV_AnnRtn'] = xAnnRtnByYear['PV_AnnRtn'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['SPXT_AnnRtn'] = xAnnRtnByYear['SPXT_AnnRtn'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['BondTR_AnnRtn'] = 
xAnnRtnByYear['BondTR_AnnRtn'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['SPX_AnnRtn'] = xAnnRtnByYear['SPX_AnnRtn'].astype(float).map("{:.2%}".format)
xAnnRtnByYear[xUnderlier + '_UL_AnnRtn2'] = xAnnRtnByYear[xUnderlier + 
'_UL_AnnRtn2'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['PV1_AnnRtn'] = xAnnRtnByYear['PV1_AnnRtn'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['PV2_AnnRtn'] = xAnnRtnByYear['PV2_AnnRtn'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['BondTR_AnnRtn2'] = 
xAnnRtnByYear['BondTR_AnnRtn2'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['SPXT_AnnRtn2'] = 
xAnnRtnByYear['SPXT_AnnRtn2'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['SPX_AnnRtn2'] = xAnnRtnByYear['SPX_AnnRtn2'].astype(float).map("{:.2%}".format)

xAnnRtnByYear['SI_AnnStd'] = xAnnRtnByYear['SI_AnnStd'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['PV_AnnStd'] = xAnnRtnByYear['PV_AnnStd'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['SPXT_AnnStd'] = xAnnRtnByYear['SPXT_AnnStd'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['BondTR_AnnStd'] = 
xAnnRtnByYear['BondTR_AnnStd'].astype(float).map("{:.2%}".format)
```

```
xAnnRtnByYear['SPX_AnnStd'] = xAnnRtnByYear['SPX_AnnStd'].astype(float).map("{:.2%}".format)
xAnnRtnByYear[xUnderlier + '_UL_AnnStd2'] = xAnnRtnByYear[xUnderlier +
'_UL_AnnStd2'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['PV1_AnnStd'] = xAnnRtnByYear['PV1_AnnStd'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['PV2_AnnStd'] = xAnnRtnByYear['PV2_AnnStd'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['BondTR_AnnStd2'] =
xAnnRtnByYear['BondTR_AnnStd2'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['SPXT_AnnStd2'] =
xAnnRtnByYear['SPXT_AnnStd2'].astype(float).map("{:.2%}".format)
xAnnRtnByYear['SPX_AnnStd2'] = xAnnRtnByYear['SPX_AnnStd2'].astype(float).map("{:.2%}".format)

xAnnRtnByYear['Sharpe_SI'] = xAnnRtnByYear['Sharpe_SI'].round(2)
xAnnRtnByYear['Sharpe_PV'] = xAnnRtnByYear['Sharpe_PV'].round(2)
xAnnRtnByYear['Sharpe_SPXT'] = xAnnRtnByYear['Sharpe_SPXT'].round(2)
xAnnRtnByYear['Sharpe_BondTR'] = xAnnRtnByYear['Sharpe_BondTR'].round(2)
xAnnRtnByYear['Sharpe_SPX'] = xAnnRtnByYear['Sharpe_SPX'].round(2)
xAnnRtnByYear['Sharpe2_UL_' + xUnderlier] = xAnnRtnByYear['Sharpe2_UL_' + xUnderlier].round(2)
xAnnRtnByYear['Sharpe_PV1'] = xAnnRtnByYear['Sharpe_PV1'].round(2)
xAnnRtnByYear['Sharpe_PV2'] = xAnnRtnByYear['Sharpe_PV2'].round(2)
xAnnRtnByYear['Sharpe2_BondTR'] = xAnnRtnByYear['Sharpe2_BondTR'].round(2)
xAnnRtnByYear['Sharpe2_SPXT'] = xAnnRtnByYear['Sharpe2_SPXT'].round(2)
xAnnRtnByYear['Sharpe2_SPX'] = xAnnRtnByYear['Sharpe2_SPX'].round(2)

############## calculate OVERALL ENTIRE PERIOD ##########
xDF0b = xDF0[['DATE', 'SPXT_rtn', 'SI_DailyRtn', 'PV_rtn', 'BondTR_rtn', 'SPX_rtn', xUnderlier
+ '_UL_DailyRtn', \
           'PV1_rtn', 'PV2_rtn', 'BondTR_DailyRtn', 'SPXT_DailyRtn', 'SPX_DailyRtn']].copy()
xDF0b.dropna(inplace=True)

xStdDev = xDF0b[['SPXT_rtn', 'SI_DailyRtn', 'PV_rtn', 'BondTR_rtn', 'SPX_rtn', xUnderlier +
'_UL_DailyRtn', \
             'PV1_rtn', 'PV2_rtn', 'BondTR_DailyRtn', 'SPXT_DailyRtn',
'SPX_DailyRtn']].std()*np.sqrt(252)

xAnnStD_SPXT = xStdDev['SPXT_rtn']
xAnnStD_SI = xStdDev['SI_DailyRtn']
xAnnStD_PV = xStdDev['PV_rtn']
xAnnStD_BondTR = xStdDev['BondTR_rtn']
xAnnStD_SPX = xStdDev['SPX_rtn']
globals()['xAnnStD2_UL_' + xUnderlier] = xStdDev[xUnderlier + '_UL_DailyRtn']
xAnnStD_PV1 = xStdDev['PV1_rtn']
xAnnStD_PV2 = xStdDev['PV2_rtn']
xAnnStD2_BondTR = xStdDev['BondTR_DailyRtn']
xAnnStD2_SPXT = xStdDev['SPXT_DailyRtn']
xAnnStD2_SPX = xStdDev['SPX_DailyRtn']

xDF0c = xDF0[['DATE', 'SPXT', 'SI_100', 'SPXT_100', 'PV', 'BondTR_100', 'SPX_100', xUnderlier
+ '_UL_term_100', \
           'PV1', 'PV2', 'BondTR_term_100', 'SPXT_term_100', 'SPX_term_100']].copy()
xDF0c.dropna(inplace=True)
xDF0c.reset_index(drop=True,inplace=True)

xDF0c['SI_growth'] = xDF0c['SI_100'].pct_change(len(xDF0c)-1)
xDF0c['SPXT_growth'] = xDF0c['SPXT_100'].pct_change(len(xDF0c)-1)
xDF0c['PV_growth'] = xDF0c['PV'].pct_change(len(xDF0c)-1)
xDF0c['BondTR_growth'] = xDF0c['BondTR_100'].pct_change(len(xDF0c)-1)
xDF0c['SPX_growth'] = xDF0c['SPX_100'].pct_change(len(xDF0c)-1)
xDF0c[xUnderlier + '_UL_term_growth'] = xDF0c[xUnderlier +
'_UL_term_100'].pct_change(len(xDF0c)-1)
xDF0c['PV1_growth'] = xDF0c['PV1'].pct_change(len(xDF0c)-1)
```

```
xDF0c['PV2_growth'] = xDF0c['PV2'].pct_change(len(xDF0c)-1)
xDF0c['BondTR_term_growth'] = xDF0c['BondTR_term_100'].pct_change(len(xDF0c)-1)
xDF0c['SPXT_term_growth'] = xDF0c['SPXT_term_100'].pct_change(len(xDF0c)-1)
xDF0c['SPX_term_growth'] = xDF0c['SPX_term_100'].pct_change(len(xDF0c)-1)

xTermRtn_SI =xDF0c.loc[xDF0c.index == len(xDF0c)-1]['SI_growth'].values[0]
xTermRtn_SPXT = xDF0c.loc[xDF0c.index == len(xDF0c)-1]['SPXT_growth'].values[0]
xTermRtn_PV = xDF0c.loc[xDF0c.index == len(xDF0c)-1]['PV_growth'].values[0]
xTermRtn_BondTR = xDF0c.loc[xDF0c.index == len(xDF0c)-1]['BondTR_growth'].values[0]
xTermRtn_SPX = xDF0c.loc[xDF0c.index == len(xDF0c)-1]['SPX_growth'].values[0]
globals()['xTermRtn2_UL_' + xUnderlier] = xDF0c.loc[xDF0c.index == len(xDF0c)-1][xUnderlier +
'_UL_term_growth'].values[0]
xTermRtn_PV1 = xDF0c.loc[xDF0c.index == len(xDF0c)-1]['PV1_growth'].values[0]
xTermRtn_PV2 = xDF0c.loc[xDF0c.index == len(xDF0c)-1]['PV2_growth'].values[0]
xTermRtn2_BondTR = xDF0c.loc[xDF0c.index == len(xDF0c)-1]['BondTR_term_growth'].values[0]
xTermRtn2_SPXT = xDF0c.loc[xDF0c.index == len(xDF0c)-1]['SPXT_term_growth'].values[0]
xTermRtn2_SPX = xDF0c.loc[xDF0c.index == len(xDF0c)-1]['SPX_term_growth'].values[0]

xAnnRtn_SI = (1+xTermRtn_SI)**(1/(len(xDF0c)/252)) - 1
xAnnRtn_SPXT = (1+xTermRtn_SPXT)**(1/(len(xDF0c)/252)) - 1
xAnnRtn_PV = (1+xTermRtn_PV)**(1/(len(xDF0c)/252)) - 1
xAnnRtn_BondTR = (1+xTermRtn_BondTR)**(1/(len(xDF0c)/252)) - 1
xAnnRtn_SPX = (1+xTermRtn_SPX)**(1/(len(xDF0c)/252)) - 1
globals()['xAnnRtn2_UL_' + xUnderlier] = (1+(globals()['xTermRtn2_UL_' +
xUnderlier]))**(1/(len(xDF0c)/252)) - 1
xAnnRtn_PV1 = (1+xTermRtn_PV1)**(1/(len(xDF0c)/252)) - 1
xAnnRtn_PV2 = (1+xTermRtn_PV2)**(1/(len(xDF0c)/252)) - 1
xAnnRtn2_BondTR = (1+xTermRtn2_BondTR)**(1/(len(xDF0c)/252)) - 1
xAnnRtn2_SPXT = (1+xTermRtn2_SPXT)**(1/(len(xDF0c)/252)) - 1
xAnnRtn2_SPX = (1+xTermRtn2_SPX)**(1/(len(xDF0c)/252)) - 1

xSharpe_SI = np.round(xAnnRtn_SI / xAnnStD_SI, 4)
xSharpe_SPXT = np.round( xAnnRtn_SPXT / xAnnStD_SPXT, 4)
xSharpe_PV = np.round(xAnnRtn_PV / xAnnStD_PV, 4)
xSharpe_BondTR = np.round(xAnnRtn_BondTR / xAnnStD_BondTR, 4)
xSharpe_SPX = np.round( xAnnRtn_SPX / xAnnStD_SPX, 4)
globals()['xSharpe2_UL_' + xUnderlier] = np.round((globals()['xAnnRtn2_UL_' + xUnderlier]) /
(globals()['xAnnStD2_UL_' + xUnderlier]), 4)
xSharpe_PV1 = np.round(xAnnRtn_PV1 / xAnnStD_PV1, 4)
xSharpe_PV2 = np.round(xAnnRtn_PV2 / xAnnStD_PV2, 4)
xSharpe2_BondTR = np.round(xAnnRtn2_BondTR / xAnnStD2_BondTR, 4)
xSharpe2_SPXT = np.round(xAnnRtn2_SPXT / xAnnStD2_SPXT, 4)
xSharpe2_SPX = np.round(xAnnRtn2_SPX / xAnnStD2_SPX, 4)

xAnnRtn_SI = '{:.2%}'.format(xAnnRtn_SI)
xAnnRtn_SPXT = '{:.2%}'.format(xAnnRtn_SPXT)
xAnnRtn_PV = '{:.2%}'.format(xAnnRtn_PV)
xAnnRtn_BondTR = '{:.2%}'.format(xAnnRtn_BondTR)
xAnnRtn_SPX = '{:.2%}'.format(xAnnRtn_SPX)
globals()['xAnnRtn2_UL_' + xUnderlier] = '{:.2%}'.format(globals()['xAnnRtn2_UL_' +
xUnderlier])
xAnnRtn_PV1 = '{:.2%}'.format(xAnnRtn_PV1)
xAnnRtn_PV2 = '{:.2%}'.format(xAnnRtn_PV2)
xAnnRtn2_BondTR = '{:.2%}'.format(xAnnRtn2_BondTR)
xAnnRtn2_SPXT = '{:.2%}'.format(xAnnRtn2_SPXT)
xAnnRtn2_SPX = '{:.2%}'.format(xAnnRtn2_SPX)

xTermRtn_SI = '{:.2%}'.format(xTermRtn_SI)
xTermRtn_SPXT = '{:.2%}'.format(xTermRtn_SPXT)
xTermRtn_PV = '{:.2%}'.format(xTermRtn_PV)
```

```
xTermRtn_BondTR = '{:.2%}'.format(xTermRtn_BondTR)
xTermRtn_SPX = '{:.2%}'.format(xTermRtn_SPX)
globals()['xTermRtn2_UL_' + xUnderlier] = '{:.2%}'.format(globals()['xTermRtn2_UL_' +
xUnderlier])
xTermRtn_PV1 = '{:.2%}'.format(xTermRtn_PV1)
xTermRtn_PV2 = '{:.2%}'.format(xTermRtn_PV2)
xTermRtn2_BondTR = '{:.2%}'.format(xTermRtn2_BondTR)
xTermRtn2_SPXT = '{:.2%}'.format(xTermRtn2_SPXT)
xTermRtn2_SPX = '{:.2%}'.format(xTermRtn2_SPX)

xAnnStD_SPXT = '{:.2%}'.format(xAnnStD_SPXT)
xAnnStD_SI = '{:.2%}'.format(xAnnStD_SI)
xAnnStD_PV = '{:.2%}'.format(xAnnStD_PV)
xAnnStD_BondTR = '{:.2%}'.format(xAnnStD_BondTR)
xAnnStD_SPX = '{:.2%}'.format(xAnnStD_SPX)
globals()['xAnnStD2_UL_' + xUnderlier] = '{:.2%}'.format(globals()['xAnnStD2_UL_' +
xUnderlier])
xAnnStD_PV1 = '{:.2%}'.format(xAnnStD_PV1)
xAnnStD_PV2 = '{:.2%}'.format(xAnnStD_PV2)
xAnnStD2_BondTR = '{:.2%}'.format(xAnnStD2_BondTR)
xAnnStD2_SPXT = '{:.2%}'.format(xAnnStD2_SPXT)
xAnnStD2_SPX = '{:.2%}'.format(xAnnStD2_SPX)

xDF_stats = pd.DataFrame(columns=['Name', 'Rtn_Term',' AnnRtn','AnnStd','Sharpe'])
xDF_stats.loc[0] = ['SI',xTermRtn_SI, xAnnRtn_SI, xAnnStD_SI, xSharpe_SI]
xDF_stats.loc[1] = ['SPXT',xTermRtn_SPXT, xAnnRtn_SPXT, xAnnStD_SPXT, xSharpe_SPXT]
xDF_stats.loc[2] = ['PV',xTermRtn_PV, xAnnRtn_PV, xAnnStD_PV, xSharpe_PV]
xDF_stats.loc[3] = ['BondTR',xTermRtn_BondTR, xAnnRtn_BondTR, xAnnStD_BondTR, xSharpe_BondTR]
xDF_stats.loc[4] = ['SPX',xTermRtn_SPX, xAnnRtn_SPX, xAnnStD_SPX, xSharpe_SPX]
xDF_stats.loc[5] = [xUnderlier + '_term',globals()['xTermRtn2_' + xUnderlier],
globals()['xAnnRtn2_' + xUnderlier], \
               globals()['xAnnStD2_' + xUnderlier], globals()['xSharpe2_' + xUnderlier]]
xDF_stats.loc[6] = ['PV1',xTermRtn_PV1, xAnnRtn_PV1, xAnnStD_PV1, xSharpe_PV1]
xDF_stats.loc[7] = ['PV2',xTermRtn_PV2, xAnnRtn_PV2, xAnnStD_PV2, xSharpe_PV2]
xDF_stats.loc[8] = ['BondTR_term',xTermRtn2_BondTR, xAnnRtn2_BondTR, xAnnStD2_BondTR,
xSharpe2_BondTR]
xDF_stats.loc[9] = ['SPXT_term',xTermRtn2_SPXT, xAnnRtn2_SPXT, xAnnStD2_SPXT, xSharpe2_SPXT]
xDF_stats.loc[10] = ['SPX_term',xTermRtn2_SPX, xAnnRtn2_SPX, xAnnStD2_SPX, xSharpe2_SPX]

xString5 = np.round(xDF0b[['SPXT_rtn','SI_DailyRtn', 'PV_rtn', 'BondTR_rtn','SPX_rtn',
xUnderlier + '_UL_DailyRtn', \
                     'PV1_rtn', 'PV2_rtn', 'BondTR_DailyRtn', 'SPXT_DailyRtn',
'SPX_DailyRtn']].corr(),4).astype('string')
xString4 = xDF_stats.astype('string')
xString1 = xAnnRtnByYear.astype('string')
xString2 = xCorrByYear.astype('string')

xStartDate = xDF0b['DATE'].min().strftime('%Y-%m-%d')
xEndDate = xDF0b['DATE'].max().strftime('%Y-%m-%d')

xString = '*** from ' + (str)(xStartDate) + ' to ' + (str)(xEndDate) + ' ***\n\n' +
(str)(xString4) + '\n\n' + \
        (str)(xString5) +'\n\n' + (str)(xString1) + '\n\n'+(str)(xString2)
xString3a = xString3 + '\n\n' + xString

f_w = open(xDir + 'xStats_Term_' + xBufferType + '_' + xUnderlier + '.txt','w')
f_w.write(xString3a)
f_w.close()

################
```

```
import matplotlib.pyplot as plt2

xDF0a.plot(x='DATE', y=['SI_100','SPXT_100', 'PV', 'BondTR_100', 'SPX_100', xUnderlier
+'_UL_term_100', \
                    'BondTR_term_100', 'SPXT_term_100', 'SPX_term_100'])
plt2.title('Performance Comparison: SI vs S&P 500 Index (TR)\n' + xChartTitle, fontsize=9,
ha='center')
#plt.figtext(0.5,0.9,'Performance Comparison: SI vs S&P 500 Index (TR)', fontsize=15,
ha='center')
#plt.figtext(0.5,0.8,xString3,fontsize=9,ha='center')
#plt.subplot().yaxis.set_major_formatter('${x:1.2f}')
plt2.subplot().yaxis.set_major_formatter('${x:1.0f}')
plt2.minorticks_on()
plt2.grid(which='both')
plt2.legend()
plt2.xlabel('Time')
plt2.ylabel('Investment Growth')
plt2.savefig(xDir + 'xPerformanceChart2_' + xUnderlier + '.png')
plt2.show()
###################

# ########## calculate two portfolios ######
# ### 1) 70% Equity SPXT and 30% Aggr Bond ####
# ### 2) 70% Equity SPXT and 15% Aggr Bond and 15% SI ####
# xDF02 = xDF0.loc[xDF0['SI_DailyRtn'].isna()][['DATE', 'SPXT_rtn',
'SI_DailyRtn','BondTR_rtn']].copy()
# xLastDateNA = xDF02['DATE'].max()
# xDF02 = xDF0.loc[xDF0['DATE'] >= xLastDateNA][['DATE', 'Year', 'SPXT_rtn', 'SI_DailyRtn',
'BondTR_rtn']].copy()
# xDF02.reset_index(drop=True, inplace=True)
#
# xDates = xDF02[['DATE']].copy()
#
# xP1W1_EQ=0.70
# xP1W2_BD=0.30
# xDF02['PV1'] = np.nan
# xDF02['PV1_SPXT'] = np.nan
# xDF02['PV1_Bond'] = np.nan
#
# xP2W1_EQ=0.70
# xP2W2_BD=0.15
# xP2W3_SI=0.15
# xDF02['PV2'] = np.nan
# xDF02['PV2_SPXT'] = np.nan
# xDF02['PV2_SI'] = np.nan
# xDF02['PV2_Bond'] = np.nan
#
# xDF02['PV_SPXT_100'] = np.nan
# xDF02['PV_BondTR_100'] = np.nan
#
# xTime = 1
# for xTempDate in xDF02['DATE']:
#  print(xTempDate)
#  xThisDate = xTempDate
#  xThisYear = xThisDate.year
#  if (xTime == 1):   # on the start date
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_Bond'] = xAmount * xP1W2_BD
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_SPXT'] = xAmount * xP1W1_EQ
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1'] = xAmount
#
```

```
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SPXT'] = xAmount * xP2W1_EQ
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_Bond'] = xAmount * xP2W2_BD
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SI'] = xAmount * xP2W3_SI
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2'] = xAmount
#
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV_SPXT_100'] = xAmount
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV_BondTR_100'] = xAmount
#
#     xExpirationDate = xThisDate + datetime.timedelta(days=365 * xTerm)
#     xExpirationDate = xDates.loc[xDates['DATE'] <= xExpirationDate]['DATE'].max() ## set the
expiraton = a trading date
#
#     xTime = xTime + 1
#     xPreviousDate = xThisDate
#     xPreviousYear = xPreviousDate.year
#     continue
#  else:
#     xPV1_Bond_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV1_Bond'].values[0]
#     xPV1_SPXT_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV1_SPXT'].values[0]
#     xPV1_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV1'].values[0]
#
#     xPV2_SPXT_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2_SPXT'].values[0]
#     xPV2_SI_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2_SI'].values[0]
#     xPV2_Bond_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2_Bond'].values[0]
#     xPV2_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2'].values[0]
#
#     xPV_SPXT_100_PreviousDate = xDF02[xDF02['DATE'] ==
xPreviousDate]['PV_SPXT_100'].values[0]
#     xPV_BondTR_100_PreviousDate = xDF02[xDF02['DATE'] ==
xPreviousDate]['PV_BondTR_100'].values[0]
#
#     xPV1_Bond_ThisDate = xPV1_Bond_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['BondTR_rtn'].values[0])
#     xPV1_SPXT_ThisDate = xPV1_SPXT_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SPXT_rtn'].values[0])
#     xPV1_ThisDate = xPV1_Bond_ThisDate + xPV1_SPXT_ThisDate
#
#     xPV2_SPXT_ThisDate = xPV2_SPXT_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SPXT_rtn'].values[0])
#     xPV2_SI_ThisDate = xPV2_SI_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SI_DailyRtn'].values[0])
#     xPV2_Bond_ThisDate = xPV2_Bond_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['BondTR_rtn'].values[0])
#     xPV2_ThisDate = xPV2_SPXT_ThisDate + xPV2_Bond_ThisDate + xPV2_SI_ThisDate
#
#     xPV_SPXT_100_ThisDate = xPV_SPXT_100_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SPXT_rtn'].values[0])
#     xPV_BondTR_100_ThisDate = xPV_BondTR_100_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['BondTR_rtn'].values[0])
#
#     ### rebalanced on expiration date #########
#     if (xThisDate == xExpirationDate):
#        xPV1_SPXT_ThisDate = xPV1_ThisDate * xP1W1_EQ
#        xPV1_Bond_ThisDate = xPV1_ThisDate * xP1W2_BD
#
#        xPV2_SPXT_ThsDate = xPV2_ThisDate * xP2W1_EQ
#        xPV2_Bond_ThsDate = xPV2_ThisDate * xP2W2_BD
#        xPV2_SI_ThsDate = xPV2_ThisDate * xP2W3_SI
#
#        xExpirationDate = xThisDate + datetime.timedelta(days=365 * xTerm)
```

```
#        xExpirationDate = xDates.loc[xDates['DATE'] <= xExpirationDate]['DATE'].max()  ## set
the expiraton = a trading date
#     ################################
#
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_SPXT'] = xPV1_SPXT_ThisDate
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_Bond'] = xPV1_Bond_ThisDate
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1'] = xPV1_ThisDate
#
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SPXT'] = xPV2_SPXT_ThisDate
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_Bond'] = xPV2_Bond_ThisDate
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SI'] = xPV2_SI_ThisDate
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2'] = xPV2_ThisDate
#
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV_SPXT_100'] = xPV_SPXT_100_ThisDate
#     xDF02.loc[xDF02['DATE'] == xThisDate, 'PV_BondTR_100'] = xPV_BondTR_100_ThisDate
#
#     xTime = xTime + 1
#     xPreviousDate = xThisDate
#     xPreviousYear = xPreviousDate.year
# ###############################################################
#################### calculate the risk and returns for PV1, PV2, PV_SPXT_100, PV_BondTR_100
##########
#################### actually these resutls are already calculated above ###########
xDF02.reset_index(drop=True,inplace=True)
xDF02['PV1_rtn'] = xDF02['PV1'].pct_change()
xDF02['PV2_rtn'] = xDF02['PV2'].pct_change()
xDF02['PV_SPXT_100_rtn'] = xDF02['PV_SPXT_100'].pct_change()
xDF02['PV_BondTR_100_rtn'] = xDF02['PV_BondTR_100'].pct_change()

xPV1_AnnStd = xDF02['PV1_rtn'].std()*np.sqrt(252)
xPV2_AnnStd = xDF02['PV2_rtn'].std()*np.sqrt(252)
xSPXT_AnnStd = xDF02['PV_SPXT_100_rtn'].std()*np.sqrt(252)
xBondTR_AnnStd = xDF02['PV_BondTR_100_rtn'].std()*np.sqrt(252)

xPV1_total_growth = xDF02['PV1'][len(xDF02)-1]/xDF02['PV1'][0] - 1
xPV2_total_growth = xDF02['PV2'][len(xDF02)-1]/xDF02['PV2'][0] - 1
xSPXT_total_growth = xDF02['PV_SPXT_100'][len(xDF02)-1]/xDF02['PV_SPXT_100'][0] - 1
xBondTR_total_growth = xDF02['PV_BondTR_100'][len(xDF02)-1]/xDF02['PV_BondTR_100'][0] - 1

xPV1_AnnRtn = (1 + xPV1_total_growth)**(1/((len(xDF02)-1)/252)) - 1
xPV2_AnnRtn = (1 + xPV2_total_growth)**(1/((len(xDF02)-1)/252)) - 1
xSPXT_AnnRtn = (1 + xSPXT_total_growth)**(1/((len(xDF02)-1)/252)) - 1
xBondTR_AnnRtn = (1 + xBondTR_total_growth)**(1/((len(xDF02)-1)/252)) - 1

xPV1_Sharpe = xPV1_AnnRtn / xPV1_AnnStd
xPV2_Sharpe = xPV2_AnnRtn / xPV2_AnnStd
xSPXT_Sharpe = xSPXT_AnnRtn / xSPXT_AnnStd
xBondTR_Sharpe = xBondTR_AnnRtn / xBondTR_AnnStd

xDF_stats.loc[11] = ['xPV1',xPV1_total_growth, xPV1_AnnRtn, xPV1_AnnStd, xPV1_Sharpe]
xDF_stats.loc[12] = ['xPV2',xPV2_total_growth, xPV2_AnnRtn, xPV2_AnnStd, xPV2_Sharpe]
xDF_stats.loc[13] = ['xSPXT_100',xSPXT_total_growth, xSPXT_AnnRtn, xSPXT_AnnStd, xSPXT_Sharpe]
xDF_stats.loc[14] = ['xBondTR_100',xBondTR_total_growth, xBondTR_AnnRtn, xBondTR_AnnStd,
xBondTR_Sharpe]

#### it proves that these results are IDENTICAL with the results calculated before!!!!
xDF_stats.to_csv(xDir + 'xDF_stats_debug.txt')
###############################################################################################
#########
############### plot ############
```

```
################
import matplotlib.pyplot as plt3

xDF02.rename(columns={'PV1': 'PV1:70% Equity/30% Bond', 'PV2': 'PV2:70% Equity/15% Bond/15%
SI'},inplace=True)

#xDF02.plot(x='DATE', y=['PV1:70% Equity/30% Bond', 'PV2:70% Equity/15% Bond/15%
SI','PV_SPXT_100', 'PV_BondTR_100'])
xDF02.plot(x='DATE', y=['PV1:70% Equity/30% Bond', 'PV2:70% Equity/15% Bond/15% SI'])
if xBufferType=='H':
   xChartTitle3 = '(Hard Buffer Note #1)'
elif xBufferType=='T':
   if xTerm == 4:
      xChartTitle3='(Barrier Buffer Note #2)'
   elif xTerm == 6:
      xChartTitle3 = '(Barrier Buffer Note #3)'
else:
   xChartTitle3 = '(Geared Buffer Note)'
plt3.title('Performance Comparison\n' + xChartTitle3, fontsize=9, ha='center')
#plt3.title('Performance Comparison: {70% Equity/30% Bond} vs {70% Equity/15% Bond/15% SI}\n'
+ xChartTitle, fontsize=9, ha='center')
#plt.figtext(0.5,0.9,'Performance Comparison: SI vs S&P 500 Index (TR)', fontsize=15,
ha='center')
#plt.figtext(0.5,0.8,xString3,fontsize=9,ha='center')
#plt.subplot().yaxis.set_major_formatter('${x:1.2f}')
plt3.subplot().yaxis.set_major_formatter('${x:1.0f}')
plt3.minorticks_on()
plt3.grid(which='both')
plt3.legend()
plt3.xlabel('Time')
plt3.ylabel('Investment Growth')
plt3.savefig(xDir + 'xPerformanceChart3_' + xUnderlier + '.png')
plt3.show()
###################

xDF0 = pd.merge(xDF0, xDF02[['DATE','PV1_SPXT','PV1_Bond','PV1:70% Equity/30%
Bond','PV2_SPXT','PV2_Bond','PV2_SI','PV2:70% Equity/15% Bond/15%
SI','PV_SPXT_100','PV_BondTR_100']], on=['DATE'],how='left')
xDF0.to_csv(xDir + 'xCalcRtnsOverTerm4SI_'+ xUnderlier + '.txt')
################# compounded return group by SI_Cycle ##########
xSI_cum_DailyRtn_vs_term =
xDF0.groupby(['SI_Cycle'])[['SI_DailyRtn','SI_rtn_term_specific']].apply(lambda x:
(np.cumprod(1 + x) - 1).iloc[-1])
xSI_cum_DailyRtn_vs_term.reset_index(inplace=True)
xSI_cum_DailyRtn_vs_term['SI_DailyRtn'] =
xSI_cum_DailyRtn_vs_term['SI_DailyRtn'].astype(float).map("{:.2%}".format)
xSI_cum_DailyRtn_vs_term['SI_rtn_term_specific'] =
xSI_cum_DailyRtn_vs_term['SI_rtn_term_specific'].astype(float).map("{:.2%}".format)
xTempDF = xDF0[['SI_Cycle','LaunchDate','MaturityDate']].copy()
xTempDF.dropna(inplace=True)
xSI_cum_DailyRtn_vs_term =
pd.merge(xSI_cum_DailyRtn_vs_term,xTempDF,on=['SI_Cycle'],how='left')
xSI_cum_DailyRtn_vs_term.to_csv(xDir + 'xSI_cum_DailyRtn_vs_term_'+ xUnderlier + '.txt')
############
################ maximum drawdowns ################
def max_dd(returns):
    """Assumes returns is a pandas Series"""
    r = returns.add(1).cumprod()
    dd = r.div(r.cummax()).sub(1)
    mdd = dd.min()
```

```
    end = returns.index[dd.argmin()]
    start = returns.index[r[:end].argmax()]
    return mdd, start, end

#########
xRtns = xDF0[['DATE','SI_DailyRtn']].copy()
xRtns.dropna(inplace=True)
xRtns.set_index('DATE',inplace=True)
xRtns.index.name = None
s = pd.Series(xRtns['SI_DailyRtn'], index=xRtns.index)

xMDD,xStart, xEnd = max_dd(s)

xStartValueMax = xDF0.loc[xDF0['DATE']==xStart]['SI_100']
xEndValueMin = xDF0.loc[xDF0['DATE']==xEnd]['SI_100']
print('maxDD:', xMDD, 'start: ', xStart, 'start value:', xStartValueMax, '; end: ', xEnd,
';end value: ', xEndValueMin)

######################### TWO PORTFOLIOS 70/30 and 70/15/15 ###############
######################### simply using the 6/4/2 years returns ############

################
########## calculate two portfolios ######
### 1) 70% Equity SPXT and 30% Aggr Bond ####
### 2) 70% Equity SPXT and 15% Aggr Bond and 15% SI ####
xDF02 = xDF0.loc[~xDF0['MaturityDate'].isna()][['DATE',
'LaunchDate','MaturityDate','SPX','SPXT','BondTR','SPXT_rtn_term', \
                                   'SI_rtn_term','BondTR_rtn_term', 'SPX_rtn_term']].copy()
xDF02 = xDF02.dropna(axis=0, subset=['MaturityDate'])
xDF02.reset_index(drop=True, inplace=True)

xP1W1_EQ=0.70
xP1W2_BD=0.30
xDF02['PV1'] = np.nan
xDF02['PV1_SPXT'] = np.nan
xDF02['PV1_Bond'] = np.nan

xP2W1_EQ=0.70
xP2W2_BD=0.15
xP2W3_SI=0.15
xDF02['PV2'] = np.nan
xDF02['PV2_SPXT'] = np.nan
xDF02['PV2_SI'] = np.nan
xDF02['PV2_Bond'] = np.nan

xTime = 1
for xTempDate in xDF02['DATE']:
   print(xTempDate)
   xThisDate = xTempDate
   xThisYear = xThisDate.year
   if (xTime == 1):   # on the start date
      #################### two portfolos ###################
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_Bond'] = xAmount * xP1W2_BD
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_SPXT'] = xAmount * xP1W1_EQ
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1'] = xAmount

      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SPXT'] = xAmount * xP2W1_EQ
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_Bond'] = xAmount * xP2W2_BD
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SI'] = xAmount * xP2W3_SI
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2'] = xAmount
```

```
      #xExpirationDate = xThisDate + datetime.timedelta(days=365 * xTerm)
      #xExpirationDate = xDates.loc[xDates['DATE'] <= xExpirationDate]['DATE'].max() ## set
the expiraton = a trading date
      #######################################################
      xTime = xTime + 1
      xPreviousDate = xThisDate
      #xPreviousYear = xPreviousDate.year
      continue
   else:
      #################### two portfolios ##################
      xPV1_Bond_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV1_Bond'].values[0]
      xPV1_SPXT_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV1_SPXT'].values[0]
      xPV1_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV1'].values[0]

      xPV2_SPXT_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2_SPXT'].values[0]
      xPV2_SI_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2_SI'].values[0]
      xPV2_Bond_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2_Bond'].values[0]
      xPV2_PreviousDate = xDF02[xDF02['DATE'] == xPreviousDate]['PV2'].values[0]

      xPV1_Bond_ThisDate = xPV1_Bond_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['BondTR_rtn_term'].values[0])
      xPV1_SPXT_ThisDate = xPV1_SPXT_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SPXT_rtn_term'].values[0])
      xPV1_ThisDate = xPV1_Bond_ThisDate + xPV1_SPXT_ThisDate

      xPV2_SPXT_ThisDate = xPV2_SPXT_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SPXT_rtn_term'].values[0])
      xPV2_SI_ThisDate = xPV2_SI_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['SI_rtn_term'].values[0])
      xPV2_Bond_ThisDate = xPV2_Bond_PreviousDate * (1 + xDF02[xDF02['DATE'] ==
xThisDate]['BondTR_rtn_term'].values[0])
      xPV2_ThisDate = xPV2_SPXT_ThisDate + xPV2_Bond_ThisDate + xPV2_SI_ThisDate
      ### rebalanced on expiration date #########
      if (True):  #(xThisDate == xExpirationDate):  # every day is expiration date
         xPV1_SPXT_ThisDate = xPV1_ThisDate * xP1W1_EQ
         xPV1_Bond_ThisDate = xPV1_ThisDate * xP1W2_BD

         xPV2_SPXT_ThsDate = xPV2_ThisDate * xP2W1_EQ
         xPV2_Bond_ThsDate = xPV2_ThisDate * xP2W2_BD
         xPV2_SI_ThsDate = xPV2_ThisDate * xP2W3_SI

         #xExpirationDate = xThisDate + datetime.timedelta(days=365 * xTerm)
         #xExpirationDate = xDates.loc[xDates['DATE'] <= xExpirationDate]['DATE'].max()  ##
set the expiraton = a trading date
      ################################

      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_SPXT'] = xPV1_SPXT_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1_Bond'] = xPV1_Bond_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV1'] = xPV1_ThisDate

      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SPXT'] = xPV2_SPXT_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_Bond'] = xPV2_Bond_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2_SI'] = xPV2_SI_ThisDate
      xDF02.loc[xDF02['DATE'] == xThisDate, 'PV2'] = xPV2_ThisDate
      ######################################################
      xTime = xTime + 1
      xPreviousDate = xThisDate
      #xPreviousYear = xPreviousDate.year
```

```
xDF02.to_csv(xDir + 'xTwoPortfolios.txt')

xTempDF =
xDF0[['DATE','SI_rtn_1_year','SPXT_rtn_1_year','BondTR_rtn_1_year','SPXT_rtn_1_year_roll','Bon
dTR_rtn_1_year_roll']].copy()
xTempDF.dropna(inplace=True)

xAnnRtn_roll =
xTempDF[['SI_rtn_1_year','SPXT_rtn_1_year','BondTR_rtn_1_year','SPXT_rtn_1_year_roll','BondTR_
rtn_1_year_roll']].mean()
xAnnStd_roll =
xTempDF[['SI_rtn_1_year','SPXT_rtn_1_year','BondTR_rtn_1_year','SPXT_rtn_1_year_roll','BondTR_
rtn_1_year_roll']].std()

xAnnRtn_roll=pd.DataFrame(xAnnRtn_roll, columns = ["AnnRtn"])
xAnnStd_roll=pd.DataFrame(xAnnStd_roll, columns = ["AnnStd"])

xAnnRtn_roll.reset_index(inplace=True)
xAnnStd_roll.reset_index(inplace=True)

xAnnRtn_Std_roll = pd.merge(xAnnRtn_roll,xAnnStd_roll,on=['index'],how='left')
xAnnRtn_Std_roll['Sharpe'] = xAnnRtn_Std_roll['AnnRtn'] / xAnnRtn_Std_roll['AnnStd']
xAnnRtn_Std_roll['AnnRtn'] = xAnnRtn_Std_roll['AnnRtn'].astype(float).map("{:.2%}".format)
xAnnRtn_Std_roll['AnnStd'] = xAnnRtn_Std_roll['AnnStd'].astype(float).map("{:.2%}".format)
xAnnRtn_Std_roll['Sharpe'] = xAnnRtn_Std_roll['Sharpe'].astype(float).map("{:.4}".format)

xCorrAnnRtn_roll
=round(xTempDF[['SI_rtn_1_year','SPXT_rtn_1_year','BondTR_rtn_1_year','SPXT_rtn_1_year_roll','
BondTR_rtn_1_year_roll']].corr(),4)

xString_roll1 = xAnnRtn_Std_roll.astype('string')
xString_roll2 = xCorrAnnRtn_roll.astype('string')

xString_roll = str(xString3) + '\n\n' +str(xString_roll1) + '\n\n' + str(xString_roll2)
f_w = open(xDir + 'xStats_roll_' + xBufferType + '_' + xUnderlier + '.txt','w')
f_w.write(xString_roll)
f_w.close()
```

#5

```
### Portfolio Optiimization
###
#
# Finds an optimal allocation of stocks in a portfolio,
# satisfying a minimum expected return.
# The problem is posed as a Quadratic Program, and solved
# using the cvxopt library.
# Uses actual past stock data, obtained using the stocks module.
import math

import numpy as np
import pandas as pd
import cvxopt
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt

solvers.options['show_progress'] = False        # !!!

#from cvxopt import solvers
#import stocks
import numpy
import pandas as pd

c = cvxopt.matrix([0, -1], tc='d')
print('c: ', c)
c = numpy.matrix(c)
print('c: ', c)

c = cvxopt.matrix([0, -1])
print('c: ', c)
G = cvxopt.matrix([[-1, 1], [3, 2], [2, 3], [-1, 0], [0, -1]], tc='d')
print('G: ', G)
##################
xDir = r'D:\\Users\\ggu\\Documents\\GU\\MeanVarianceOptimization\\'
xSPXT = pd.read_csv(xDir + 'SPXT.txt')
xSPXT['DATE'] = pd.to_datetime(xSPXT['DATE'], format='%m/%d/%Y')
xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')
xAggregateBondTR['DATE'] = pd.to_datetime(xAggregateBondTR['DATE'], format='%m/%d/%Y')
#xSI = pd.read_csv(xDir + 'SI.txt')

xSI = pd.read_csv(xDir + 'xCalcRtnsOverTerm4SI.txt',usecols = ['DATE','SI_100'])
xSI['DATE'] = pd.to_datetime(xSI['DATE'], format='%Y-%m-%d')

print(xSPXT.head())
print(xAggregateBondTR.head())
print(xSI.head())

xSPXT = pd.merge(xSPXT, xSI, on=['DATE'], how='left')
xSPXT = pd.merge(xSPXT, xAggregateBondTR, on=['DATE'], how='left')
#xTest3=pd.merge(xTest2,xTestDATE2[['YEAR','WK','DATE2']],on=['YEAR','WK'],how='left')

xSPXT.rename(columns={'SI_100':'SI', 'LBUSTRUU':'BondTR'},inplace=True)

xMinDateSI = xSI['DATE'].min()
xMaxDateSI = xSI['DATE'].max()

xSPXT = xSPXT.loc[(xSPXT['DATE'] >= xMinDateSI) & (xSPXT['DATE'] <= xMaxDateSI)]
```

```
xSPXT['SI'].fillna(method='ffill', inplace=True) #fill N/As with previous prices!!!!
xSPXT['BondTR'].fillna(method='ffill', inplace=True) #fill N/As with previous prices!!!!

xSPXT['SPXT_rtn'] = xSPXT['SPXT'].pct_change()
xSPXT['SI_rtn'] = xSPXT['SI'].pct_change()
xSPXT['BondTR_rtn'] = xSPXT['BondTR'].pct_change()

xSPXT.to_csv(xDir + 'xSPXT.txt')
#############################################################
xUnderlier = 'SPX'  #'SPX'
xSubDir1 = r'2YearsHardBufferNote\\'
xSubDir2 = r'4YearsBarrierNote\\'
xSubDir3 = r'6YearsTriggerBuffer\\'

xSubText1 = 'Hard Buffer Note #1'
xSubText2 = 'Barrier Buffer Note #2'
xSubText3 = 'Barrier Buffer Note #3'

xBufferNoteNumber = '2' ###'2'  ###'1'  # 3

if xBufferNoteNumber =='1':
   xTerm='2 years'
elif xBufferNoteNumber == '2':
   xTerm='4 years'
elif xBufferNoteNumber == '3':
   xTerm='6 years'

xSubDir = globals()['xSubDir' + xBufferNoteNumber]
xSubText = globals()['xSubText' + xBufferNoteNumber]

xSI2 = pd.read_csv(xDir + xSubDir + 'xCalcRtnsOverTerm4SI_' + xUnderlier + '.txt',usecols =
['DATE','SI_100','SPXT_100', \

'BondTR_100','SPX_100','SPX_term_100','BondTR_term_100','SPXT_term_100','SPX_term_100',\
            'BondTR_rtn_term','SPXT_rtn_term','SI_rtn_term', 'BondTR_rtn_1_year',\

'SPXT_rtn_1_year','SI_rtn_1_year','SPXT_rtn_1_year_roll','BondTR_rtn_1_year_roll'])
xSI2['DATE'] = pd.to_datetime(xSI2['DATE'], format='%Y-%m-%d')

xSPXT = xSI2.copy()
xSPXT.rename(columns={'SI_100':'SI','BondTR_100':'BondTR','SPXT_100':'SPXT','SPX_100':'SPX', \

'SPX_term_100':'SPX_term','BondTR_term_100':'BondTR_term','SPXT_term_100':'SPXT_term','SPX_ter
m_100':'SPX_term'},inplace=True)
xSPXT['SPXT_rtn'] = xSPXT['SPXT'].pct_change()
xSPXT['SI_rtn'] = xSPXT['SI'].pct_change()
xSPXT['BondTR_rtn'] = xSPXT['BondTR'].pct_change()
xSPXT['SPX_rtn'] = xSPXT['SPX'].pct_change()
xSPXT['SPX_term_rtn'] = xSPXT['SPX_term'].pct_change()
xSPXT['BondTR_term_rtn'] = xSPXT['BondTR_term'].pct_change()
xSPXT['SPXT_term_rtn'] = xSPXT['SPXT_term'].pct_change()
xSPXT['SPX_term_rtn'] = xSPXT['SPX_term'].pct_change()

xSPXT = xSPXT.dropna()

############################ EQUITY AND BOND ONLY #################
xSI_indicator = False
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn', 'SI_rtn']]
else:
```

```
   xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = xRtns.std(axis=0) * numpy.sqrt(252)

print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() * (252 ^ 2)
print('covs: ', covs)

covs = covs.values

print('covs: ', covs)
avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret * 252    #annualized
print('avg_ret: ', avg_ret)
#########
############

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)


# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

####################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
```

```
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   G = cvxopt.matrix(numpy.concatenate((
      -numpy.transpose(numpy.array(avg_ret)),
      -numpy.identity(n)), 0))
   h = cvxopt.matrix(numpy.concatenate((
      -numpy.ones((1, 1))*r_min,
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   A = cvxopt.matrix(1.0, (1, n))
   b = cvxopt.matrix(1.0)
   print('P = ', P)
   print('q = ', q)
   print('G = ', G)
   print('h = ', h)
   print('A = ', A)
   print('b = ', b)
   # A = numpy.matrix(1.0, (1, n))
   # print('A = ', A)
   sol = solvers.qp(P, q, G, h, A, b)
   return sol

# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = 0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
   print('delta_r: ', delta_r)
```

```
    w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
    print('w: ', w)
    print('w.T', w.T)
    w2 = numpy.matrix(w.T)
    print('w2.T', w2)
    return2 = (w.T * avg_ret)[0]
    risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
    print('return2: ', return2)
    print('risk2: ', risk2)

    returns.append(return2)
    risks.append(risk2)

    w2 = numpy.insert(w2, w2.size, [risk2, return2])
    print('w2:', w2)
    df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_Equity_Bond.txt')


fig, ax = plt.subplots()
ax.plot(risks, returns, color='red', label='Equity/Bond')
fig.suptitle('Efficient Frontiers for ' + xSubText, fontsize=16,y=0.95)
ax.set_xlabel('Risk (Annualized Std)', fontsize=10)
ax.set_ylabel('Annualized Return', fontsize=10)

# plt.ylabel('mean')
# plt.xlabel('std')
# plt.title('Efficient Frontier xx with underlying index ' + xUnderlier)
# #plt.plot(risks, returns, 'y-o')
# plt.plot(risks, returns, color='red',label='Equity/Bond')
# plt.legend(loc='lower right')
# import matplotlib.ticker as mtick
# plt.axis()

xStock_scater = plt.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
xBond_scatter = plt.scatter(std_vec[1], avg_ret[1], marker='*', color='green',label='Bond')
#bond
if xSI_indicator:
    if xCash:
        xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
        xCash_scatter = plt.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
    else:
        xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
    if xCash:
        xCash_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
    else:
        print ('nothing here')
#plt.show()
#plt.show(block=False)
```

```
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
############################

##plt.xlim(xmin=0)
##plt.ylim(ymin=0.02)

##plt.show()
############# CASE 2 ###########################
############################ EQUITY, BOND ONLY AND SI #################
xSI_indicator = True
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn', 'SI_rtn']]
else:
   xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = cvxopt.matrix(xRtns.std(axis=0)) * numpy.sqrt(252)
################
##std_vec[2] = 0.06
#############
print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() * (252 ^ 2)
print('covs: ', covs)

covs = covs.values
print('covs: ', covs)

corr = xRtns.corr()
corr = corr.values

################ alternative way to calculate covs ########
xL = [std_vec[0],std_vec[1],std_vec[2]]
xDiag_std = np.diag(xL)
#covs = std_vec * corr * std_vec.T
covs_2 = cvxopt.matrix(xDiag_std) * cvxopt.matrix(corr) * cvxopt.matrix(xDiag_std)
### note:  this calculation is slightly different from the
##################
```

```
avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret * 252    #annualized
print('avg_ret: ', avg_ret)
######### testing #########
###avg_ret[2] = avg_ret[2] / 3
##########################

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)


# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

###################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   G = cvxopt.matrix(numpy.concatenate((
      -numpy.transpose(numpy.array(avg_ret)),
      -numpy.identity(n)), 0))
   h = cvxopt.matrix(numpy.concatenate((
      -numpy.ones((1, 1))*r_min,
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   A = cvxopt.matrix(1.0, (1, n))
   b = cvxopt.matrix(1.0)
   print('P = ', P)
   print('q = ', q)
   print('G = ', G)
   print('h = ', h)
   print('A = ', A)
   print('b = ', b)
```

```
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol

# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = 0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
    print('delta_r: ', delta_r)
    w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
    print('w: ', w)
    print('w.T', w.T)
    w2 = numpy.matrix(w.T)
    print('w2.T', w2)
    return2 = (w.T * avg_ret)[0]
    risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
    print('return2: ', return2)
    print('risk2: ', risk2)

    returns.append(return2)
    risks.append(risk2)

    w2 = numpy.insert(w2, w2.size, [risk2, return2])
    print('w2:', w2)
    df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_equity_bond_SI.txt')

ax.plot(risks, returns,color='blue',label='Equity/Bond/SI')
import matplotlib.ticker as mtick
#plt.axis()

#xStock_scater = plt.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
#xBond_scatter = plt.scatter(std_vec[1], avg_ret[1], marker='*', color='green')    #bond
if xSI_indicator:
```

```
   if xCash:
      xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black',lable='SI')
# SI
      xCash_scatter = plt.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
   else:
      xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
   else:
      print('nothing here')
#plt.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
############################
############# CASE 3 : THIS HAS 15% cap on SI WEIGHT!!! ###########################
############################ EQUITY, BOND ONLY AND SI #################
xSI_indicator = True
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn', 'SI_rtn']]
else:
   xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = cvxopt.matrix(xRtns.std(axis=0)) * numpy.sqrt(252)
################
##std_vec[2] = 0.06
#############
print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() * (252 ^ 2)
print('covs: ', covs)

covs = covs.values
print('covs: ', covs)
```

```
corr = xRtns.corr()
corr = corr.values

################ alternative way to calculate covs ########
xL = [std_vec[0],std_vec[1],std_vec[2]]
xDiag_std = np.diag(xL)
#covs = std_vec * corr * std_vec.T
covs_2 = cvxopt.matrix(xDiag_std) * cvxopt.matrix(corr) * cvxopt.matrix(xDiag_std)
### note:  this calculation is slightly different from the
##################

avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret * 252    #annualized
print('avg_ret: ', avg_ret)
######### testing #########
###avg_ret[2] = avg_ret[2] / 3
##########################

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)


# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

##################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   G = cvxopt.matrix(numpy.concatenate((
      -numpy.transpose(numpy.array(avg_ret)),
      -numpy.identity(n),
      ), 0))
```

```
    v = (G[G.size[0] - 1, :])
    v[0, v.size[1] - 1] = 1
    G = cvxopt.matrix(numpy.concatenate((G, v), 0))

    h = cvxopt.matrix(numpy.concatenate((
        -numpy.ones((1, 1))*r_min,
        numpy.zeros((n, 1)),
        numpy.ones((1, 1)) * 0.15), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    A = cvxopt.matrix(1.0, (1, n))
    b = cvxopt.matrix(1.0)
    print('P = ', P)
    print('q = ', q)
    print('G = ', G)
    print('h = ', h)
    print('A = ', A)
    print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol

# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = 0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
    print('delta_r: ', delta_r)
    w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
    print('w: ', w)
    print('w.T', w.T)
    w2 = numpy.matrix(w.T)
    print('w2.T', w2)
    return2 = (w.T * avg_ret)[0]
    risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
    print('return2: ', return2)
    print('risk2: ', risk2)

    returns.append(return2)
    risks.append(risk2)

    w2 = numpy.insert(w2, w2.size, [risk2, return2])
```

```
    print('w2:', w2)
    df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_equity_bond_SI_15pct.txt')

import matplotlib.ticker as mtick

#fig = plt.figure(1)
#fig.add_subplot(111)
#ax = fig.add_subplot(111)

#ax.plot(perc, data)

fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
##xticks = mtick.FormatStrFormatter(fmt)
xticks = mtick.FuncFormatter("{:.0%}".format)
ax.xaxis.set_major_formatter(xticks)
ax.yaxis.set_major_formatter(xticks)

#plt.ylabel('mean')
#plt.xlabel('std')
#plt.title('Efficient Frontier xxxx with underlying index ' + xUnderlier)
#plt.plot(risks, returns, 'y-o')
ax.plot(risks, returns,color='black',label='Equity/Bond/SI with max 15% on SI')
import matplotlib.ticker as mtick
#plt.axis()

#xStock_scater = plt.scatter(std_vec[0], avg_ret[0], marker='x', color='red')  #stock
#xBond_scatter = plt.scatter(std_vec[1], avg_ret[1], marker='*', color='green')    #bond
# if xSI_indicator:
#   if xCash:
#      xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black', label='SI')
# SI
#      xCash_scatter = plt.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
#   else:
#      #print('hererrrrrrrr')
#      xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
# else:
#   if xCash:
#      xCash_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
#   else:
#      print('nothing here')
#plt.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
############################
plt.grid(which='both')
plt.legend(loc='best', ncol=2,facecolor='white')
plt.xlim(xmin=0)
plt.ylim(ymin=0.02)
```

```
plt.savefig(xDir + 'EfficientFrontier_'+xSubText+'.png')
plt.show()
################################
############################ EQUITY AND BOND ONLY TERM RETURNS #################
xSI_indicator = False
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn', 'SI_rtn']]
else:
   #xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn']]
   xRtns = xSPXT[['SPXT_term_rtn', 'BondTR_term_rtn']]
   #xRtns = xSPXT[['SPX_term_rtn', 'BondTR_term_rtn']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = xRtns.std(axis=0) * numpy.sqrt(252)

print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() * (252 ^ 2)
print('covs: ', covs)

covs = covs.values

print('covs: ', covs)
avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret * 252    #annualized
print('avg_ret: ', avg_ret)
#########
############

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)


# from numpy.linalg import eig
# values, vectors = eig(covs)
```

```
# print('values: ', values)
# print('eigen vector: ', vectors)

###################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   G = cvxopt.matrix(numpy.concatenate((
      -numpy.transpose(numpy.array(avg_ret)),
      -numpy.identity(n)), 0))
   h = cvxopt.matrix(numpy.concatenate((
      -numpy.ones((1, 1))*r_min,
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   A = cvxopt.matrix(1.0, (1, n))
   b = cvxopt.matrix(1.0)
   print('P = ', P)
   print('q = ', q)
   print('G = ', G)
   print('h = ', h)
   print('A = ', A)
   print('b = ', b)
   # A = numpy.matrix(1.0, (1, n))
   # print('A = ', A)
   sol = solvers.qp(P, q, G, h, A, b)
   return sol

# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve
```

```
P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = 0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
   print('delta_r: ', delta_r)
   w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
   print('w: ', w)
   print('w.T', w.T)
   w2 = numpy.matrix(w.T)
   print('w2.T', w2)
   return2 = (w.T * avg_ret)[0]
   risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
   print('return2: ', return2)
   print('risk2: ', risk2)

   returns.append(return2)
   risks.append(risk2)

   w2 = numpy.insert(w2, w2.size, [risk2, return2])
   print('w2:', w2)
   df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_ALL_CM.txt')

#import matplotlib as plt2
import matplotlib.pyplot as plt2

#import matplotlib.pyplot as plt2

fig, ax = plt2.subplots()
ax.plot(risks, returns, label='Equity/Bond')
fig.suptitle('Efficient Frontier (all based on CM) with underlying index ' + xUnderlier,
fontsize=12)
ax.set_xlabel('Std', fontsize=10)
ax.set_ylabel('Mean', fontsize=10)

#plt2.ylabel('mean')
#plt2.xlabel('std')
#plt2.title('Efficient Frontier x (all based on CM) with underlying index ' + xUnderlier)
#plt2.plot(risks, returns, 'y-o')
#plt2.plot(risks, returns)
import matplotlib.ticker as mtick
#plt2.axis()

xStock_scater = plt2.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='stock')
#stock
xBond_scatter = plt2.scatter(std_vec[1], avg_ret[1], marker='*', color='green',label='bond')
#bond
if xSI_indicator:
   if xCash:
      xSI_scatter = plt2.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
```

```
# SI
      xCash_scatter = plt2.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='cash')  # cash
   else:
      #print('hererrrrrrrrr')
      xSI_scatter = plt2.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt2.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='cash')  # cash
   else:
      print('nothing here')
#################################################
############################ EQUITY, BOND AND SI TERM RETURNS #################
xSI_indicator = True
if (xSI_indicator):
   #xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn', 'SI_rtn']]
   xRtns = xSPXT[['SPXT_term_rtn', 'BondTR_term_rtn', 'SI_rtn']]
   #xRtns = xSPXT[['SPX_term_rtn', 'BondTR_term_rtn', 'SI_rtn']]
else:
   #xRtns = xSPXT[['SPXT_rtn', 'BondTR_rtn']]
   xRtns = xSPXT[['SPXT_term_rtn', 'BondTR_term_rtn']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = xRtns.std(axis=0) * numpy.sqrt(252)

print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() * (252 ^ 2)
print('covs: ', covs)

covs = covs.values

print('covs: ', covs)
avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret * 252    #annualized
print('avg_ret: ', avg_ret)
#########
############
```

```
n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)


# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

###################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   G = cvxopt.matrix(numpy.concatenate((
      -numpy.transpose(numpy.array(avg_ret)),
      -numpy.identity(n)), 0))
   h = cvxopt.matrix(numpy.concatenate((
      -numpy.ones((1, 1))*r_min,
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   A = cvxopt.matrix(1.0, (1, n))
   b = cvxopt.matrix(1.0)
   print('P = ', P)
   print('q = ', q)
   print('G = ', G)
   print('h = ', h)
   print('A = ', A)
   print('b = ', b)
   # A = numpy.matrix(1.0, (1, n))
   # print('A = ', A)
   sol = solvers.qp(P, q, G, h, A, b)
   return sol

# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
```

```
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = 0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
   print('delta_r: ', delta_r)
   w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
   print('w: ', w)
   print('w.T', w.T)
   w2 = numpy.matrix(w.T)
   print('w2.T', w2)
   return2 = (w.T * avg_ret)[0]
   risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
   print('return2: ', return2)
   print('risk2: ', risk2)

   returns.append(return2)
   risks.append(risk2)

   w2 = numpy.insert(w2, w2.size, [risk2, return2])
   print('w2:', w2)
   df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_term.txt')

#import matplotlib as plt2
#import matplotlib.pyplot as plt2

#fig, ax = plt2.subplots()
ax.plot(risks, returns, label='Equity/Bond/SI')
#fig.suptitle('Efficient Frontier (all based on CM) with underlying index ' + xUnderlier,
fontsize=12)
#ax.set_xlabel('Std', fontsize=10)
#ax.set_ylabel('Mean', fontsize=10)

# # title and labels, setting initial sizes
# fig.suptitle('test title', fontsize=12)
# ax.set_xlabel('xlabel', fontsize=10)
# ax.set_ylabel('ylabel', fontsize='medium')   # relative to plt.rcParams['font.size']
#
```

```
# # setting label sizes after creation
# ax.xaxis.label.set_size(20)
# plt.draw()
# plt.show()

#plt2.ylabel('mean')
#plt2.xlabel('std')
#plt2.title('Efficient Frontier xx (all based on CM) with underlying index ' + xUnderlier)
#plt2.plot(risks, returns, 'y-o')
#plt2.plot(risks, returns)
import matplotlib.ticker as mtick
#plt2.axis()

#xStock_scater = ax.scatter(std_vec[0], avg_ret[0], marker='x', color='red', label='stock')
#stock
#xBond_scatter = ax.scatter(std_vec[1], avg_ret[1], marker='*', color='green',label='bond')
#bond
if xSI_indicator:
   if xCash:
      xSI_scatter = ax.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
      xCash_scatter = plt2.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='cash')  # cash
   else:
      #print('hererrrrrrrr')
      xSI_scatter = ax.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt2.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='cash')  # cash
   else:
      print('nothing')

# plt2.xlim(xmin=0)
# plt2.ylim(ymin=0.02)

plt2.legend(loc='best')

plt2.show()

################################# the following are efficient frontiers with 2/4//6 years
rolling returns and std #######
############################ EQUITY AND BOND ONLY #################
import matplotlib.pyplot as plt3
xSI_indicator = False
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn_term', 'BondTR_rtn_term', 'SI_rtn_term']]
else:
   xRtns = xSPXT[['SPXT_rtn_term', 'BondTR_rtn_term']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)
```

```
print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = xRtns.std(axis=0)  ############# * numpy.sqrt(252)

print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov()  ########## * (252 ^ 2)
print('covs: ', covs)

covs = covs.values

print('covs: ', covs)
avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret ################ * 252   #annualized
print('avg_ret: ', avg_ret)
#########
############

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)

# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

###################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
```

```
    # if the lowest return has a higher risk, this constraint will find a
    # higher return corresponding to the lowest risk!!!  that is why there
    # is no line (or no curve) on the efficient frontier from the return
    # corresponding to the minimal risk to the lowest return.
    G = cvxopt.matrix(numpy.concatenate((
       -numpy.transpose(numpy.array(avg_ret)),
       -numpy.identity(n)), 0))
    h = cvxopt.matrix(numpy.concatenate((
       -numpy.ones((1, 1))*r_min,
       numpy.zeros((n, 1))), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    A = cvxopt.matrix(1.0, (1, n))
    b = cvxopt.matrix(1.0)
    print('P = ', P)
    print('q = ', q)
    print('G = ', G)
    print('h = ', h)
    print('A = ', A)
    print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol

# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = (r_max2 - r_min2) / 100  ###############0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
   print('delta_r: ', delta_r)
   w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
   print('w: ', w)
   print('w.T', w.T)
   w2 = numpy.matrix(w.T)
   print('w2.T', w2)
   return2 = (w.T * avg_ret)[0]
   risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
   print('return2: ', return2)
   print('risk2: ', risk2)

   returns.append(return2)
   risks.append(risk2)
```

```
    w2 = numpy.insert(w2, w2.size, [risk2, return2])
    print('w2:', w2)
    df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_Equity_Bond_term.txt')

fig3, ax3 = plt3.subplots()
ax3.plot(risks, returns, color='red', label='Equity/Bond')
fig3.suptitle('Efficient Frontiers (term) for ' + xSubText, fontsize=16,y=0.95)
ax3.set_xlabel('Risk ('+xTerm+')', fontsize=10)
ax3.set_ylabel('Return ('+xTerm+')', fontsize=10)

# plt.ylabel('mean')
# plt.xlabel('std')
# plt.title('Efficient Frontier xx with underlying index ' + xUnderlier)
# #plt.plot(risks, returns, 'y-o')
# plt.plot(risks, returns, color='red',label='Equity/Bond')
# plt.legend(loc='lower right')
# import matplotlib.ticker as mtick
# plt.axis()

xStock_scater = plt3.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
xBond_scatter = plt3.scatter(std_vec[1], avg_ret[1], marker='*', color='green',label='Bond')
#bond
if xSI_indicator:
    if xCash:
        xSI_scatter = plt3.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
        xCash_scatter = plt3.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
    else:
        xSI_scatter = plt3.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
    if xCash:
        xCash_scatter = plt3.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
    else:
        print ('nothing here')
#plt.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
############################

##plt.xlim(xmin=0)
##plt.ylim(ymin=0.02)

##plt.show()
############# CASE 2 ###########################
############################ EQUITY, BOND ONLY AND SI #################
xSI_indicator = True
if (xSI_indicator):
```

```
   xRtns = xSPXT[['SPXT_rtn_term', 'BondTR_rtn_term', 'SI_rtn_term']]
else:
   xRtns = xSPXT[['SPXT_rtn_term', 'BondTR_rtn_term']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = cvxopt.matrix(xRtns.std(axis=0)) ################ * numpy.sqrt(252)
################
##std_vec[2] = 0.06
#############
print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() ############### * (252 ^ 2)
print('covs: ', covs)
########## debug testing #########
#covs['SI_rtn_term'][0] = covs['SI_rtn_term'][0]*(-1)
#covs['SPXT_rtn_term'][2] = covs['SPXT_rtn_term'][2]*(-1)
##########################
covs = covs.values
print('covs: ', covs)

corr = xRtns.corr()
corr = corr.values

################ alternative way to calculate covs ########
xL = [std_vec[0],std_vec[1],std_vec[2]]
xDiag_std = np.diag(xL)
#covs = std_vec * corr * std_vec.T
covs_2 = cvxopt.matrix(xDiag_std) * cvxopt.matrix(corr) * cvxopt.matrix(xDiag_std)
### note:  this calculation is slightly different from the
##########################################################

avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret ################# * 252  #annualized
print('avg_ret: ', avg_ret)
######### testing debug #########
####avg_ret[2] = avg_ret[2] * 1.2
##########################

n = len(avg_ret)
print('n = ', n)
```

```
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)

# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

###################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   G = cvxopt.matrix(numpy.concatenate((
      -numpy.transpose(numpy.array(avg_ret)),
      -numpy.identity(n)), 0))
   h = cvxopt.matrix(numpy.concatenate((
      -numpy.ones((1, 1))*r_min,
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   A = cvxopt.matrix(1.0, (1, n))
   b = cvxopt.matrix(1.0)
   print('P = ', P)
   print('q = ', q)
   print('G = ', G)
   print('h = ', h)
   print('A = ', A)
   print('b = ', b)
   # A = numpy.matrix(1.0, (1, n))
   # print('A = ', A)
   sol = solvers.qp(P, q, G, h, A, b)
   return sol

# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
```

```
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = (r_max2 - r_min2) / 100 ###########0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
   print('delta_r: ', delta_r)
   w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
   print('w: ', w)
   print('w.T', w.T)
   w2 = numpy.matrix(w.T)
   print('w2.T', w2)
   return2 = (w.T * avg_ret)[0]
   risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
   print('return2: ', return2)
   print('risk2: ', risk2)

   returns.append(return2)
   risks.append(risk2)

   w2 = numpy.insert(w2, w2.size, [risk2, return2])
   print('w2:', w2)
   df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_equity_bond_SI_term.txt')

ax3.plot(risks, returns,color='blue',label='Equity/Bond/SI')
import matplotlib.ticker as mtick
#plt.axis()

#xStock_scater = plt.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
#xBond_scatter = plt.scatter(std_vec[1], avg_ret[1], marker='*', color='green')    #bond
if xSI_indicator:
   if xCash:
      xSI_scatter = plt3.scatter(std_vec[2], avg_ret[2], marker='X', color='black',lable='SI')
# SI
      xCash_scatter = plt3.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
   else:
      xSI_scatter = plt3.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt3.scatter(std_vec[2], avg_ret[2], marker='+',
```

```
color='blue',label='Cash')  # cash
   else:
      print('nothing here')
#plt3.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
import matplotlib.ticker as mtick3

#fig = plt.figure(1)
#fig.add_subplot(111)
#ax = fig.add_subplot(111)

#ax.plot(perc, data)

fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
##xticks = mtick.FormatStrFormatter(fmt)
xticks = mtick3.FuncFormatter("{:.0%}".format)
ax3.xaxis.set_major_formatter(xticks)
ax3.yaxis.set_major_formatter(xticks)

plt3.grid(which='both')
plt3.legend(loc='best', ncol=3,facecolor='white')
plt3.xlim(xmin=0)
plt3.ylim(ymin=0.02)

plt3.savefig(xDir + 'EfficientFrontier_'+xSubText+'_term.png')
plt3.show()
############################

#################### the following are efficient frontiers based on 1-YEAR returns for SPXT,
BondTR and SI #######
########## 1 year return for SI derived from 2/4/6 years return; 1-year returns for SPXT and
BondTR from daily prices ###########
############################ EQUITY AND BOND ONLY #################
import matplotlib.pyplot as plt4
xSI_indicator = False
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year', 'SI_rtn_1_year']]
else:
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = xRtns.std(axis=0)  ############# * numpy.sqrt(252)

print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
```

```
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov()  ########## * (252 ^ 2)
print('covs: ', covs)

covs = covs.values

print('covs: ', covs)
avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret ################ * 252   #annualized
print('avg_ret: ', avg_ret)
#########
############

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)

# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

####################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
##########<=mmodified = R ################
def optimize_portfolio_modified(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.transpose(numpy.array(avg_ret)),
   #  -numpy.identity(n)), 0))
```

```
    #G = cvxopt.matrix(numpy.concatenate((
    #  -numpy.identity(n)), 0))
    G = cvxopt.matrix(-np.diag(np.ones(n),0))
    # h = cvxopt.matrix(numpy.concatenate((
    #  -numpy.ones((1, 1))*r_min,
    #  numpy.zeros((n, 1))), 0))
    h = cvxopt.matrix(numpy.concatenate((
       numpy.zeros((n, 1))), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    #-numpy.transpose(numpy.array(avg_ret)),
    #A = cvxopt.matrix(1.0, (1, n))
    A = cvxopt.matrix(numpy.concatenate((
       numpy.transpose(numpy.array(avg_ret)),
       cvxopt.matrix(1.0, (1, n)))))
    #b = cvxopt.matrix(1.0)
    b = cvxopt.matrix(numpy.concatenate((
       numpy.ones((1, 1)) * r_min,
       cvxopt.matrix(1.0))))
    print('P = ', P)
    print('q = ', q)
    print('G = ', G)
    print('h = ', h)
    print('A = ', A)
    print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol

############## original version ##################
def optimize_portfolio(n, avg_ret, covs, r_min):
    P = cvxopt.matrix(covs)
    # x = variable(n)
    q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
    # inequality constraints Gx <= h
    # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
    # note: the loop starts from the lowest return to the highest return
    # if the lowest return has a higher risk, this constraint will find a
    # higher return corresponding to the lowest risk!!!  that is why there
    # is no line (or no curve) on the efficient frontier from the return
    # corresponding to the minimal risk to the lowest return.
    G = cvxopt.matrix(numpy.concatenate((
       -numpy.transpose(numpy.array(avg_ret)),
       -numpy.identity(n)), 0))
    h = cvxopt.matrix(numpy.concatenate((
       -numpy.ones((1, 1))*r_min,
       numpy.zeros((n, 1))), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    A = cvxopt.matrix(1.0, (1, n))
    b = cvxopt.matrix(1.0)
    print('P = ', P)
    print('q = ', q)
    print('G = ', G)
    print('h = ', h)
    print('A = ', A)
    print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol
```

```
################################################
# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = (r_max2 - r_min2) / 100  ###############0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
   print('delta_r: ', delta_r)
   w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
   print('w: ', w)
   print('w.T', w.T)
   w2 = numpy.matrix(w.T)
   print('w2.T', w2)
   return2 = (w.T * avg_ret)[0]
   risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
   print('return2: ', return2)
   print('risk2: ', risk2)

   returns.append(return2)
   risks.append(risk2)

   w2 = numpy.insert(w2, w2.size, [risk2, return2])
   print('w2:', w2)
   df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_Equity_Bond_term.txt')

fig4, ax4 = plt4.subplots()
ax4.plot(risks, returns, color='red', label='Equity/Bond')
fig4.suptitle('Efficient Frontiers for ' + xSubText, fontsize=16,y=0.95)
ax4.set_xlabel('Annual Risk', fontsize=10)
ax4.set_ylabel('Annual Return', fontsize=10)

# plt.ylabel('mean')
# plt.xlabel('std')
# plt.title('Efficient Frontier xx with underlying index ' + xUnderlier)
# #plt.plot(risks, returns, 'y-o')
# plt.plot(risks, returns, color='red',label='Equity/Bond')
# plt.legend(loc='lower right')
```

```
# import matplotlib.ticker as mtick
# plt.axis()

xStock_scater = plt4.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
xBond_scatter = plt4.scatter(std_vec[1], avg_ret[1], marker='*', color='green',label='Bond')
#bond
if xSI_indicator:
   if xCash:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
      xCash_scatter = plt4.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
   else:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
   else:
      print ('nothing here')
#plt.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
############################

##plt.xlim(xmin=0)
##plt.ylim(ymin=0.02)

##plt.show()
############# CASE 2 ###########################
############################ EQUITY, BOND ONLY AND SI #################
xSI_indicator = True
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year', 'SI_rtn_1_year']].copy()
else:
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year']].copy()

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = cvxopt.matrix(xRtns.std(axis=0)) ################ * numpy.sqrt(252)
################
##std_vec[2] = 0.06
#############
print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
```

```
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() ############### * (252 ^ 2)
print('covs: ', covs)
########## debug testing #########
#covs['SI_rtn_term'][0] = covs['SI_rtn_term'][0]*(-1)
#covs['SPXT_rtn_term'][2] = covs['SPXT_rtn_term'][2]*(-1)
##########################
covs = covs.values
print('covs: ', covs)

corr = xRtns.corr()
corr = corr.values

################ alternative way to calculate covs ########
xL = [std_vec[0],std_vec[1],std_vec[2]]
xDiag_std = np.diag(xL)
#covs = std_vec * corr * std_vec.T
covs_2 = cvxopt.matrix(xDiag_std) * cvxopt.matrix(corr) * cvxopt.matrix(xDiag_std)
### note:  this calculation is slightly different from the
##########################################################

avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret ################# * 252  #annualized
print('avg_ret: ', avg_ret)
######### testing debug #########
####avg_ret[2] = avg_ret[2] * 1.2
##########################

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)

# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

##################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
```

```
##########<=mmodified = R ###############
def optimize_portfolio_modified(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.transpose(numpy.array(avg_ret)),
   #  -numpy.identity(n)), 0))
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.identity(n)), 0))
   G = cvxopt.matrix(-np.diag(np.ones(n),0))
   # h = cvxopt.matrix(numpy.concatenate((
   #  -numpy.ones((1, 1))*r_min,
   #  numpy.zeros((n, 1))), 0))
   h = cvxopt.matrix(numpy.concatenate((
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   #-numpy.transpose(numpy.array(avg_ret)),
   #A = cvxopt.matrix(1.0, (1, n))
   A = cvxopt.matrix(numpy.concatenate((
      numpy.transpose(numpy.array(avg_ret)),
      cvxopt.matrix(1.0, (1, n)))))
   #b = cvxopt.matrix(1.0)
   b = cvxopt.matrix(numpy.concatenate((
      numpy.ones((1, 1)) * r_min,
      cvxopt.matrix(1.0))))
   print('P = ', P)
   print('q = ', q)
   print('G = ', G)
   print('h = ', h)
   print('A = ', A)
   print('b = ', b)
   # A = numpy.matrix(1.0, (1, n))
   # print('A = ', A)
   sol = solvers.qp(P, q, G, h, A, b)
   return sol

############## original version #################
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   G = cvxopt.matrix(numpy.concatenate((
      -numpy.transpose(numpy.array(avg_ret)),
      -numpy.identity(n)), 0))
   h = cvxopt.matrix(numpy.concatenate((
```

```
        -numpy.ones((1, 1))*r_min,
        numpy.zeros((n, 1))), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    A = cvxopt.matrix(1.0, (1, n))
    b = cvxopt.matrix(1.0)
    print('P = ', P)
    print('q = ', q)
    print('G = ', G)
    print('h = ', h)
    print('A = ', A)
    print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol
###############################################
# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = (r_max2 - r_min2) / 100 ############0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
    print('delta_r: ', delta_r)
    w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
    print('w: ', w)
    print('w.T', w.T)
    w2 = numpy.matrix(w.T)
    print('w2.T', w2)
    return2 = (w.T * avg_ret)[0]
    risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
    print('return2: ', return2)
    print('risk2: ', risk2)

    returns.append(return2)
    risks.append(risk2)

    w2 = numpy.insert(w2, w2.size, [risk2, return2])
    print('w2:', w2)
    df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())
```

```
df.to_csv(xDir + 'xOptimalPortfolio_equity_bond_SI_1_year.txt')

ax4.plot(risks, returns,color='blue',label='Equity/Bond/SI')
import matplotlib.ticker as mtick
#plt.axis()

#xStock_scater = plt.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
#xBond_scatter = plt.scatter(std_vec[1], avg_ret[1], marker='*', color='green')    #bond
if xSI_indicator:
   if xCash:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',lable='SI')
# SI
      xCash_scatter = plt4.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
   else:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
   else:
      print('nothing here')
#plt3.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
import matplotlib.ticker as mtick4

#fig = plt.figure(1)
#fig.add_subplot(111)
#ax = fig.add_subplot(111)

#ax.plot(perc, data)

fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
##xticks = mtick.FormatStrFormatter(fmt)
xticks = mtick4.FuncFormatter("{:.0%}".format)
ax4.xaxis.set_major_formatter(xticks)
ax4.yaxis.set_major_formatter(xticks)

############# CASE 3a : THIS HAS 25% cap on SI WEIGHT!!! ###########################
#############1 year returns from daily prices for EQUITY, BOND; AND SI uses 1 year return
2/4/6 year returns ###########
xSI_indicator = True
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year', 'SI_rtn_1_year']]
else:
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
```

```
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = cvxopt.matrix(xRtns.std(axis=0)) * numpy.sqrt(252)
################
##std_vec[2] = 0.06
#############
print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() ############ * (252 ^ 2)
print('covs: ', covs)

covs = covs.values
print('covs: ', covs)

corr = xRtns.corr()
corr = corr.values

################ alternative way to calculate covs ########
xL = [std_vec[0],std_vec[1],std_vec[2]]
xDiag_std = np.diag(xL)
#covs = std_vec * corr * std_vec.T
covs_2 = cvxopt.matrix(xDiag_std) * cvxopt.matrix(corr) * cvxopt.matrix(xDiag_std)
### note:  this calculation is slightly different from the above #########
##################

avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret ####### no more * 252    #annualized
print('avg_ret: ', avg_ret)
######### testing #########
###avg_ret[2] = avg_ret[2] / 3
##########################

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)


# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

###################################################################
# solves the QP, where x is the allocation of the portfolio:
```

```
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
##########<=mmodified = R ################
def optimize_portfolio_modified(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.transpose(numpy.array(avg_ret)),
   #  -numpy.identity(n)), 0))
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.identity(n)), 0))
   G = cvxopt.matrix(-np.diag(np.ones(n),0))
   # h = cvxopt.matrix(numpy.concatenate((
   #  -numpy.ones((1, 1))*r_min,
   #  numpy.zeros((n, 1))), 0))
   h = cvxopt.matrix(numpy.concatenate((
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   #-numpy.transpose(numpy.array(avg_ret)),
   #A = cvxopt.matrix(1.0, (1, n))
   A = cvxopt.matrix(numpy.concatenate((
      numpy.transpose(numpy.array(avg_ret)),
      cvxopt.matrix(1.0, (1, n)))))
   #b = cvxopt.matrix(1.0)
   b = cvxopt.matrix(numpy.concatenate((
      numpy.ones((1, 1)) * r_min,
      cvxopt.matrix(1.0))))
   print('P = ', P)
   print('q = ', q)
   print('G = ', G)
   print('h = ', h)
   print('A = ', A)
   print('b = ', b)
   # A = numpy.matrix(1.0, (1, n))
   # print('A = ', A)
   sol = solvers.qp(P, q, G, h, A, b)
   return sol

############## original version ##################
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
```

```
    # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
    # note: the loop starts from the lowest return to the highest return
    # if the lowest return has a higher risk, this constraint will find a
    # higher return corresponding to the lowest risk!!!  that is why there
    # is no line (or no curve) on the efficient frontier from the return
    # corresponding to the minimal risk to the lowest return.
    G = cvxopt.matrix(numpy.concatenate((
       -numpy.transpose(numpy.array(avg_ret)),
       -numpy.identity(n)), 0))
    h = cvxopt.matrix(numpy.concatenate((
       -numpy.ones((1, 1))*r_min,
       numpy.zeros((n, 1))), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    A = cvxopt.matrix(1.0, (1, n))
    b = cvxopt.matrix(1.0)
    print('P = ', P)
    print('q = ', q)
    print('G = ', G)
    print('h = ', h)
    print('A = ', A)
    print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol
###############################################
# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = 0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
    print('delta_r: ', delta_r)
    w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
    print('w: ', w)
    print('w.T', w.T)
    w2 = numpy.matrix(w.T)
    print('w2.T', w2)
    return2 = (w.T * avg_ret)[0]
    risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
    print('return2: ', return2)
    print('risk2: ', risk2)
```

```
   returns.append(return2)
   risks.append(risk2)

   w2 = numpy.insert(w2, w2.size, [risk2, return2])
   print('w2:', w2)
   df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_equity_bond_SI_1_year_25pct.txt')

#import matplotlib.ticker as mtick

#fig = plt.figure(1)
#fig.add_subplot(111)
#ax = fig.add_subplot(111)

#ax.plot(perc, data)

# fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
# ##xticks = mtick.FormatStrFormatter(fmt)
# xticks = mtick.FuncFormatter("{:.0%}".format)
# ax4.xaxis.set_major_formatter(xticks)
# ax4.yaxis.set_major_formatter(xticks)

#plt.ylabel('mean')
#plt.xlabel('std')
#plt.title('Efficient Frontier xxxx with underlying index ' + xUnderlier)
#plt.plot(risks, returns, 'y-o')
ax4.plot(risks, returns,color='black',label='Equity/Bond/SI with max 25% on SI')
import matplotlib.ticker as mtick
#plt.axis()

#xStock_scater = plt.scatter(std_vec[0], avg_ret[0], marker='x', color='red')  #stock
#xBond_scatter = plt.scatter(std_vec[1], avg_ret[1], marker='*', color='green')    #bond
# if xSI_indicator:
#  if xCash:
#     xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black', label='SI')
# SI
#     xCash_scatter = plt.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
#  else:
#     #print('hererrrrrrrr')
#     xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
# else:
#  if xCash:
#     xCash_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
#  else:
#     print('nothing here')
#plt.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
############################
################################
```

```
plt4.grid(which='both')
plt4.legend(loc='best', ncol=2,facecolor='white')
plt4.xlim(xmin=0)
plt4.ylim(ymin=0.02)

plt4.savefig(xDir + 'EfficientFrontier_'+xSubText+'_1_year.png')
plt4.show()
############################
##################### the following are efficient frontiers based on 1-YEAR returns for SPXT,
BondTR and SI #######
############### ALL 1-year returns are derived from 2/4/6 years ROLLING ####################
############################ EQUITY AND BOND ONLY #################
import matplotlib.pyplot as plt5
xSI_indicator = False
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn_1_year_roll', 'BondTR_rtn_1_year_roll', 'SI_rtn_1_year']]
else:
   xRtns = xSPXT[['SPXT_rtn_1_year_roll', 'BondTR_rtn_1_year_roll']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = xRtns.std(axis=0)  ############# * numpy.sqrt(252)

print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov()  ########## * (252 ^ 2)
print('covs: ', covs)

covs = covs.values

print('covs: ', covs)
avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret ################ * 252   #annualized
print('avg_ret: ', avg_ret)
#########
############

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)
```

```
r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)

# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

####################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   G = cvxopt.matrix(numpy.concatenate((
      -numpy.transpose(numpy.array(avg_ret)),
      -numpy.identity(n)), 0))
   h = cvxopt.matrix(numpy.concatenate((
      -numpy.ones((1, 1))*r_min,
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   A = cvxopt.matrix(1.0, (1, n))
   b = cvxopt.matrix(1.0)
   print('P = ', P)
   print('q = ', q)
   print('G = ', G)
   print('h = ', h)
   print('A = ', A)
   print('b = ', b)
   # A = numpy.matrix(1.0, (1, n))
   # print('A = ', A)
   sol = solvers.qp(P, q, G, h, A, b)
   return sol

# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
```

```
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = (r_max2 - r_min2) / 100  ###############0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
   print('delta_r: ', delta_r)
   w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
   print('w: ', w)
   print('w.T', w.T)
   w2 = numpy.matrix(w.T)
   print('w2.T', w2)
   return2 = (w.T * avg_ret)[0]
   risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
   print('return2: ', return2)
   print('risk2: ', risk2)

   returns.append(return2)
   risks.append(risk2)

   w2 = numpy.insert(w2, w2.size, [risk2, return2])
   print('w2:', w2)
   df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_Equity_Bond_1_year_roll.txt')

fig5, ax5 = plt5.subplots()
ax5.plot(risks, returns, color='red', label='Equity/Bond')
fig5.suptitle('Efficient Frontiers (ALL ROLLING) for ' + xSubText, fontsize=14,y=0.95)
ax5.set_xlabel('Annual Risk', fontsize=10)
ax5.set_ylabel('Annual Return', fontsize=10)

# plt.ylabel('mean')
# plt.xlabel('std')
# plt.title('Efficient Frontier xx with underlying index ' + xUnderlier)
# #plt.plot(risks, returns, 'y-o')
# plt.plot(risks, returns, color='red',label='Equity/Bond')
# plt.legend(loc='lower right')
# import matplotlib.ticker as mtick
# plt.axis()

xStock_scater = plt4.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
xBond_scatter = plt4.scatter(std_vec[1], avg_ret[1], marker='*', color='green',label='Bond')
#bond
if xSI_indicator:
   if xCash:
```

```
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
      xCash_scatter = plt4.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
   else:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
   else:
      print ('nothing here')
#plt.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
############################

##plt.xlim(xmin=0)
##plt.ylim(ymin=0.02)

##plt.show()
############## CASE 2 ###########################
############################ EQUITY, BOND ONLY AND SI #################
xSI_indicator = True
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn_1_year_roll', 'BondTR_rtn_1_year_roll', 'SI_rtn_1_year']]
else:
   xRtns = xSPXT[['SPXT_rtn_1_year_roll', 'BondTR_rtn_1_year_roll']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = cvxopt.matrix(xRtns.std(axis=0)) ################ * numpy.sqrt(252)
################
##std_vec[2] = 0.06
#############
print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() ############### * (252 ^ 2)
```

```
print('covs: ', covs)
########## debug testing #########
#covs['SI_rtn_term'][0] = covs['SI_rtn_term'][0]*(-1)
#covs['SPXT_rtn_term'][2] = covs['SPXT_rtn_term'][2]*(-1)
##########################
covs = covs.values
print('covs: ', covs)

corr = xRtns.corr()
corr = corr.values

################ alternative way to calculate covs ########
xL = [std_vec[0],std_vec[1],std_vec[2]]
xDiag_std = np.diag(xL)
#covs = std_vec * corr * std_vec.T
covs_2 = cvxopt.matrix(xDiag_std) * cvxopt.matrix(corr) * cvxopt.matrix(xDiag_std)
### note:  this calculation is slightly different from the
##########################################################

avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret ################# * 252  #annualized
print('avg_ret: ', avg_ret)
######### testing debug #########
####avg_ret[2] = avg_ret[2] * 1.2
##########################

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)

# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

###################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
```

```
    # is no line (or no curve) on the efficient frontier from the return
    # corresponding to the minimal risk to the lowest return.
    G = cvxopt.matrix(numpy.concatenate((
        -numpy.transpose(numpy.array(avg_ret)),
        -numpy.identity(n)), 0))
    h = cvxopt.matrix(numpy.concatenate((
        -numpy.ones((1, 1))*r_min,
        numpy.zeros((n, 1))), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    A = cvxopt.matrix(1.0, (1, n))
    b = cvxopt.matrix(1.0)
    print('P = ', P)
    print('q = ', q)
    print('G = ', G)
    print('h = ', h)
    print('A = ', A)
    print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol

# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = (r_max2 - r_min2) / 100 ############0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
    print('delta_r: ', delta_r)
    w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
    print('w: ', w)
    print('w.T', w.T)
    w2 = numpy.matrix(w.T)
    print('w2.T', w2)
    return2 = (w.T * avg_ret)[0]
    risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
    print('return2: ', return2)
    print('risk2: ', risk2)

    returns.append(return2)
    risks.append(risk2)

    w2 = numpy.insert(w2, w2.size, [risk2, return2])
```

```
   print('w2:', w2)
   df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_equity_bond_SI_1_year_roll.txt')

ax5.plot(risks, returns,color='blue',label='Equity/Bond/SI')
import matplotlib.ticker as mtick
#plt.axis()

#xStock_scater = plt.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
#xBond_scatter = plt.scatter(std_vec[1], avg_ret[1], marker='*', color='green')    #bond
if xSI_indicator:
   if xCash:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',lable='SI')
# SI
      xCash_scatter = plt4.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
   else:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
   else:
      print('nothing here')
#plt3.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
import matplotlib.ticker as mtick5

#fig = plt.figure(1)
#fig.add_subplot(111)
#ax = fig.add_subplot(111)

#ax.plot(perc, data)

fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
##xticks = mtick.FormatStrFormatter(fmt)
xticks = mtick5.FuncFormatter("{:.0%}".format)
ax5.xaxis.set_major_formatter(xticks)
ax5.yaxis.set_major_formatter(xticks)

plt5.grid(which='both')
plt5.legend(loc='best', ncol=3,facecolor='white')
plt5.xlim(xmin=0)
plt5.ylim(ymin=0.02)

plt5.savefig(xDir + 'EfficientFrontier_'+xSubText+'_1_year_roll.png')
plt5.show()
############################
```

#6

```
### Portfolio Optiimization
###
#
# Finds an optimal allocation of stocks in a portfolio,
# satisfying a minimum expected return.
# The problem is posed as a Quadratic Program, and solved
# using the cvxopt library.
# Uses actual past stock data, obtained using the stocks module.
import math
import numpy as np
import pandas as pd
import datetime
import cvxopt
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt
##########################
import warnings
warnings.filterwarnings('ignore')
warnings.warn('DelftStack')
warnings.warn('Do not show this message')
#####################
solvers.options['show_progress'] = False        # !!!

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

#from cvxopt import solvers
#import stocks
import numpy
import pandas as pd
import datetime

# c = cvxopt.matrix([0, -1], tc='d')
# print('c: ', c)
# c = numpy.matrix(c)
# print('c: ', c)
#
# c = cvxopt.matrix([0, -1])
# print('c: ', c)
# G = cvxopt.matrix([[-1, 1], [3, 2], [2, 3], [-1, 0], [0, -1]], tc='d')
# print('G: ', G)
##################
xDir = r'D:\\Users\\ggu\\Documents\\GU\\MeanVarianceOptimization\\'
xSPXT = pd.read_csv(xDir + 'SPXT.txt')
xSPXT['DATE'] = pd.to_datetime(xSPXT['DATE'], format='%m/%d/%Y')
xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')
xAggregateBondTR['DATE'] = pd.to_datetime(xAggregateBondTR['DATE'], format='%m/%d/%Y')

# xSI = pd.read_csv(xDir + 'SI.txt')
# xSI['DATE'] = pd.to_datetime(xSI['DATE'], format='%m/%d/%Y')

xSPX = pd.read_csv(xDir + 'SPX.txt')
xSPX['DATE'] = pd.to_datetime(xSPX['DATE'], format='%m/%d/%Y')

##xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')

print(xSPXT.head())
```

```
print(xAggregateBondTR.head())
#print(xSI.head())
print(xSPX.head())

# xSPXT = pd.merge(xSPXT, xSI, on=['DATE'], how='left')
xSPXT = pd.merge(xSPXT, xAggregateBondTR, on=['DATE'], how='left')
xSPXT = pd.merge(xSPXT, xSPX, on=['DATE'], how='left')

# xMinDateSI = xSI['DATE'].min()
# xMaxDateSI = xSI['DATE'].max()

###xSPXT = xSPXT.loc[(xSPXT['DATE'] >= xMinDateSI) & (xSPXT['DATE'] <= xMaxDateSI)]

#xSPXT['intrinsic_value'].fillna(method='ffill', inplace=True) #fill N/As with previous
prices!!!!
xSPXT['LBUSTRUU'].fillna(method='ffill', inplace=True) #fill N/As with previous prices!!!!
xSPXT['SPX'].fillna(method='ffill', inplace=True)

xSPXT.rename(columns={'LBUSTRUU': 'BondTR'},inplace=True)

xSPXT['SPXT_rtn'] = xSPXT['SPXT'].pct_change()
#xSPXT['SI_rtn'] = xSPXT['intrinsic_value'].pct_change()
xSPXT['BondTR_rtn'] = xSPXT['BondTR'].pct_change()
xSPXT['SPX_rtn'] = xSPXT['SPX'].pct_change()

xSPXT.to_csv(xDir + 'xSPXT.txt')

xSPXT = xSPXT.dropna()
########################
xUnderlier = 'SPX'

#xDF0 = xSPXT[['DATE', xUnderlier,'SPXT','SPXT_rtn','BondTR','BondTR_rtn']]
xDF0 = xSPXT[['DATE', xUnderlier,'SPXT','BondTR']]
print('xDF0 = ', xDF0.head())

### These are the generic products we used in learning center.
#- 2Y, 10% hard buffer, 1.5x upside up to 21%
#- 4Y, 25% barrier, 1x upside no-cap
#- 6Y, 30% barrier, 1.15x upside no-cap
#################################################################

xCap = 100000 #0.21  #10000  #100000 #0.21   #10000 #0.21 #0.21
xBuffer = -0.30 #-0.30  #250  #-0.25   #-0.30   # -0.10   #-0.25

xTerm = 6  #2  #4  #6   #4 #2  #3 # years
xAmount = 100000
xLever = 1.15  #1.15
xBufferType = "T"  #"T"  # "H" for regular Buffer; "G" for Geared Buffer (or Barrier); "T" for
Trigger Buffer!

xPortfolio = pd.DataFrame()

####################

xDate = '2007-10-09'   #'2000-01-01'
xStartDate = pd.to_datetime(xDate)   #datetime.date.fromisoformat(xDate)
##########################################################################
print('xStartDate = ', xStartDate)
xEndDate = xStartDate + datetime.timedelta(days = 365*xTerm)
print('xEndDate = ', xEndDate)
```

```
#################### retrieve the stress start and end dates ############################
xStressDates = pd.read_csv(xDir + 'xMajorDeclineDate.txt', usecols=['StartDate','EndDate'])
xStressDates['StartDate'] = pd.to_datetime(xStressDates['StartDate'], format='%Y-%m-%d')
xStressDates['EndDate'] = pd.to_datetime(xStressDates['EndDate'], format='%Y-%m-%d')

############## select stress period ####################
#xI = 2
xString0 =''
for xI in range(0,3): #range(1,2):   #range(0,3)
   xStressStartDate = pd.to_datetime(xStressDates.StartDate.values[xI],format='%Y-%m-%d')
   xStressEndDate = pd.to_datetime(xStressDates.EndDate.values[xI],format='%Y-%m-%d')
   #xScenario = 1
   for xScenario in range(1,7): #range(1,2):  #range(1,7) #7 is NOT included
      if xScenario == 1:
         xStartDate = xStressStartDate
         xEffectiveStressStartDate = xStressStartDate
         xEndDate = xStartDate + datetime.timedelta(days=365 * xTerm)
         xDF0['Days'] = (xDF0['DATE'] - xEndDate).dt.days
         xTemp = xDF0.loc[xDF0['Days']<=0]
         xTemp.reset_index(drop=True,inplace=True)
         xEndDate = xTemp['DATE'][len(xTemp)-1] # this is the trading date!
         xEffectiveStressEndDate = min(xEndDate, xStressEndDate)
      elif xScenario == 2:
         xStartDate = xStressStartDate + datetime.timedelta(days=-365 * round(xTerm / 3,0))
         xDF0['Days'] = (xDF0['DATE'] - xStartDate).dt.days
         xTemp = xDF0.loc[xDF0['Days'] >= 0]
         xTemp.reset_index(drop=True, inplace=True)
         xStartDate = xTemp['DATE'][0]  # this is the trading date!
         xEffectiveStressStartDate = xStressStartDate
         xEndDate = xStartDate + datetime.timedelta(days=365 * xTerm)
         xDF0['Days'] = (xDF0['DATE'] - xEndDate).dt.days
         xTemp = xDF0.loc[xDF0['Days'] <= 0]
         xTemp.reset_index(drop=True, inplace=True)
         xEndDate = xTemp['DATE'][len(xTemp) - 1]  # this is the trading date!
         xEffectiveStressEndDate = min(xEndDate, xStressEndDate)
      elif xScenario == 3:
         xStartDate = xStressStartDate + datetime.timedelta(days=-365 * round(xTerm / 2, 0))
         xDF0['Days'] = (xDF0['DATE'] - xStartDate).dt.days
         xTemp = xDF0.loc[xDF0['Days'] >= 0]
         xTemp.reset_index(drop=True, inplace=True)
         xStartDate = xTemp['DATE'][0]  # this is the trading date!
         xEffectiveStressStartDate = xStressStartDate
         xEndDate = xStartDate + datetime.timedelta(days=365 * xTerm)
         xDF0['Days'] = (xDF0['DATE'] - xEndDate).dt.days
         xTemp = xDF0.loc[xDF0['Days'] <= 0]
         xTemp.reset_index(drop=True, inplace=True)
         xEndDate = xTemp['DATE'][len(xTemp) - 1]  # this is the trading date!
         xEffectiveStressEndDate = min(xEndDate, xStressEndDate)
      elif xScenario == 4:
         xEndDate = xStressEndDate
         xEffectiveStressEndDate = xStressEndDate
         xStartDate = xEndDate + datetime.timedelta(days=-365 * xTerm)
         xDF0['Days'] = (xDF0['DATE'] - xStartDate).dt.days
         xTemp = xDF0.loc[xDF0['Days'] >= 0]
         xTemp.reset_index(drop=True, inplace=True)
         xStartDate = xTemp['DATE'][0]  # this is the trading date!
         xEffectiveStressStartDate = max(xStressStartDate, xStartDate)
      elif xScenario == 5:
         xEndDate = xStressEndDate + datetime.timedelta(days=365 * round(xTerm / 3, 0))
         xDF0['Days'] = (xDF0['DATE'] - xEndDate).dt.days
```

```
            xTemp = xDF0.loc[xDF0['Days'] <= 0]
            xTemp.reset_index(drop=True, inplace=True)
            xEndDate = xTemp['DATE'][len(xTemp) - 1]  # this is the trading date!
            xEffectiveStressEndDate = xStressEndDate
            xStartDate = xEndDate + datetime.timedelta(days=-365 * xTerm)
            xDF0['Days'] = (xDF0['DATE'] - xStartDate).dt.days
            xTemp = xDF0.loc[xDF0['Days'] >= 0]
            xTemp.reset_index(drop=True, inplace=True)
            xStartDate = xTemp['DATE'][0]  # this is the trading date!
            xEffectiveStressStartDate = max(xStressStartDate, xStartDate)
        elif xScenario == 6:
            xEndDate = xStressEndDate + datetime.timedelta(days=365 * round(xTerm / 2, 0))
            xDF0['Days'] = (xDF0['DATE'] - xEndDate).dt.days
            xTemp = xDF0.loc[xDF0['Days'] <= 0]
            xTemp.reset_index(drop=True, inplace=True)
            xEndDate = xTemp['DATE'][len(xTemp) - 1]  # this is the trading date!
            xEffectiveStressEndDate = xStressEndDate
            xStartDate = xEndDate + datetime.timedelta(days=-365 * xTerm)
            xDF0['Days'] = (xDF0['DATE'] - xStartDate).dt.days
            xTemp = xDF0.loc[xDF0['Days'] >= 0]
            xTemp.reset_index(drop=True, inplace=True)
            xStartDate = xTemp['DATE'][0]  # this is the trading date!
            xEffectiveStressStartDate = max(xStressStartDate, xStartDate)
        #
        # ############# SI starts at the peak!!! #######
        # xStartDate = xStressStartDate
        # xEndDate = xStressEndDate
        # ############# SI ends at the trough ##########
        # if False:
        #  xStartDate = xEndDate + datetime.timedelta(days = -365*xTerm)
        # ######### this is to set the start date as the trough 1 year ago #########
        # x1YearAgo = 0
        # if x1YearAgo == 1:
        #  xStartDate2 = xStartDate + datetime.timedelta(days = -365) # one year ago from the
stress start date!
        #  xDF = xDF0.loc[(xDF0['DATE'] >= xStartDate2) & (xDF0['DATE'] <= xStartDate)]
        #  xMin_SPX = xDF['SPX'].min()
        #  xStartDate = pd.to_datetime(xDF.loc[xDF['SPX']==xMin_SPX]['DATE'].values[0])   # this
is trough...lowest point
        # #####################
        # xSIEndDate = xStartDate + datetime.timedelta(days = 365*xTerm)
        ########### debug ############
        print('Stress Cycle: ' + (str)(xI) + ';  Scenario: ' + (str)(xScenario))
        print('Start date: ', xStartDate, '; End date:', xEndDate)
        ##############################
        #xDF = xDF0.loc[(xDF0['DATE'] >= xStartDate.strftime('%Y-%m-%d')) & (xDF0['DATE'] <=
xEndDate.strftime('%Y-%m-%d'))]
        xDF = xDF0.loc[(xDF0['DATE'] >= xStartDate) & (xDF0['DATE'] <= xEndDate)]
        #xDF = xDF0.loc[(xDF0['DATE'] >= xStartDate)]
        xDF.reset_index(drop=True, inplace=True)
        ###### in case xEndDate does NOT exist in xDF, then reassign the latest date less than
the original xEndDate ###
        xEndDate = pd.to_datetime(xDF.loc[xDF.index == (len(xDF)-1)]['DATE'].values[0])

        xDF[xUnderlier+'_rtn'] = xDF[xUnderlier].pct_change()
        xDF['SPXT_rtn'] = xDF['SPXT'].pct_change()
        xDF['BondTR_rtn'] = xDF['BondTR'].pct_change()
        xDF['CumRtn_SPXT'] = (1 + xDF['SPXT_rtn']).cumprod() - 1
        xDF['CumRtn_BondTR'] = (1 + xDF['BondTR_rtn']).cumprod() - 1
        xDF['CumRtn_UL'] = (1 + xDF[xUnderlier+'_rtn']).cumprod() - 1
```

```
      xDF['CumRtn_SI'] = xDF['CumRtn_UL'].copy()

      xTime = 0
      xString3 = 'Structure: ' + 'Buffer Type = ' + xBufferType + '; Term = ' + (str)(xTerm) +
' years; ' + (str)(xLever) + 'x Underlier; Cap = '  + (str)(xCap) + '; Buffer = ' +
(str)(xBuffer)
      xStartDate0 = xStartDate
      #xStartValue = xDF.loc[xDF.index==0][xUnderlier][0]
      xStartValue = xAmount
      xW_equity_pv1 = 0.7
      xW_bond_pv1 = 0.3
      xW_equity_pv2 = 0.7
      xW_bond_pv2 = 0.15
      xW_SI_pv2 = 0.15

      ###while (xDF.empty != True):  #this may not work properly because xStartDate = xEndDate
= 1 row onlu!!!!
      #while (xStartDate != xEndDate):
      for xTempDate in xDF['DATE']:
         print('date = ', xTempDate)
         xTime = xTime + 1

         xCumRtn_UL = xDF.loc[xDF['DATE']==xTempDate]['CumRtn_UL'].values[0]
         if (xBufferType == 'T'):
            if (xCumRtn_UL < xBuffer):
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = xCumRtn_UL
            elif (xCumRtn_UL <= 0):
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = 0
            elif (xCumRtn_UL * xLever > xCap):  #(((xCumRtn_UL + 1) * xLever - 1)> xCap):  #
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = xCap
            else:
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = xCumRtn_UL * xLever
         elif (xBufferType == 'H'):
            if (xCumRtn_UL < xBuffer):
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = xCumRtn_UL - xBuffer
            elif (xCumRtn_UL <= 0):
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = 0
            elif (xCumRtn_UL * xLever > xCap):   # (((xCumRtn_UL + 1) * xLever - 1)> xCap):
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = xCap
            else:
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = xCumRtn_UL * xLever
         elif (xBufferType == 'G'):
            if (xCumRtn_UL < xBuffer):
               xK = 1 / (1 + xBuffer)  # 100/(100-30) = 10/7
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = xK * (xCumRtn_UL - xBuffer)
            elif (xCumRtn_UL <= 0):
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = 0
            elif (xCumRtn_UL * xLever > xCap):   #(((xCumRtn_UL + 1) * xLever - 1)> xCap): #
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = xCap
            else:
               xDF.loc[xDF['DATE'] == xTempDate, 'CumRtn_SI'] = xCumRtn_UL * xLever

##############################################################################################
####################
         ############# calculate IV and Portfolio Values (PV) ########
         if xTime == 1:
            xDF.loc[xDF['DATE'] == xTempDate, 'IV'] = xAmount

            xDF.loc[xDF['DATE'] == xTempDate, 'PV1_SPXT'] = xAmount * xW_equity_pv1
            xDF.loc[xDF['DATE'] == xTempDate, 'PV1_BondTR'] = xAmount * xW_bond_pv1
```

```
            xDF.loc[xDF['DATE'] == xTempDate, 'PV1'] = xAmount

            xDF.loc[xDF['DATE'] == xTempDate, 'PV2_SPXT'] = xAmount * xW_equity_pv2
            xDF.loc[xDF['DATE'] == xTempDate, 'PV2_BondTR'] = xAmount * xW_bond_pv2
            xDF.loc[xDF['DATE'] == xTempDate, 'PV2_SI'] = xAmount * xW_SI_pv2
            xDF.loc[xDF['DATE'] == xTempDate, 'PV2'] = xAmount

            xDF.loc[xDF['DATE'] == xTempDate, 'SPXT_100'] = xAmount
         else:
            xDF.loc[xDF['DATE'] == xTempDate, 'IV'] = (1+xDF.loc[xDF['DATE'] ==
xTempDate]['CumRtn_SI'].values[0]) * xAmount

            xDF.loc[xDF['DATE'] == xTempDate, 'PV1_SPXT'] = (1+xDF.loc[xDF['DATE'] ==
xTempDate]['SPXT_rtn'].values[0]) * \
                                                xDF.loc[xDF['DATE'] ==
xPreviousDate]['PV1_SPXT'].values[0]
            xDF.loc[xDF['DATE'] == xTempDate, 'PV1_BondTR'] = (1+xDF.loc[xDF['DATE'] ==
xTempDate]['BondTR_rtn'].values[0]) * \
                                                xDF.loc[xDF['DATE'] ==
xPreviousDate]['PV1_BondTR'].values[0]
            xDF.loc[xDF['DATE'] == xTempDate, 'PV1'] = xDF.loc[xDF['DATE'] ==
xTempDate]['PV1_SPXT'].values[0] + \
                                                xDF.loc[xDF['DATE'] ==
xTempDate]['PV1_BondTR'].values[0]

            xDF.loc[xDF['DATE'] == xTempDate, 'PV2_SPXT'] = (1+xDF.loc[xDF['DATE'] ==
xTempDate]['SPXT_rtn'].values[0]) * \
                                                xDF.loc[xDF['DATE'] ==
xPreviousDate]['PV2_SPXT'].values[0]
            xDF.loc[xDF['DATE'] == xTempDate, 'PV2_BondTR'] = (1 + xDF.loc[xDF['DATE'] ==
xTempDate]['BondTR_rtn'].values[0]) * \
                                                  xDF.loc[xDF['DATE'] ==
xPreviousDate]['PV2_BondTR'].values[0]
            xDF.loc[xDF['DATE'] == xTempDate, 'PV2_SI'] = (1 + xDF.loc[xDF['DATE'] ==
xTempDate]['CumRtn_SI'].values[0]) * \
                                               (xAmount * xW_SI_pv2)

            xDF.loc[xDF['DATE'] == xTempDate, 'PV2'] = xDF.loc[xDF['DATE'] ==
xTempDate]['PV2_SPXT'].values[0] +\
                                             xDF.loc[xDF['DATE'] ==
xTempDate]['PV2_BondTR'].values[0] +\
                                             xDF.loc[xDF['DATE'] ==
xTempDate]['PV2_SI'].values[0]

            xDF.loc[xDF['DATE'] == xTempDate, 'SPXT_100'] = (1 + xDF.loc[xDF['DATE'] ==
xTempDate]['SPXT_rtn'].values[0]) * \
                                             xDF.loc[xDF['DATE'] ==
xPreviousDate]['SPXT_100'].values[0]

         xPreviousDate = xTempDate
         xTime = xTime + 1

      xDF['SPX_growth'] = xDF['SPX'].pct_change(len(xDF)-1)
      xDF['SPXT_growth'] = xDF['SPXT'].pct_change(len(xDF)-1)
      xDF['BondTR_growth'] = xDF['BondTR'].pct_change(len(xDF)-1)
      xDF['IV_growth'] = xDF['IV'].pct_change(len(xDF)-1)
      xDF['PV1_SPXT_growth'] = xDF['PV1_SPXT'].pct_change(len(xDF)-1)
      xDF['PV1_BondTR_growth'] = xDF['PV1_BondTR'].pct_change(len(xDF)-1)
      xDF['PV1_growth'] = xDF['PV1'].pct_change(len(xDF)-1)
      xDF['PV2_SPXT_growth'] = xDF['PV2_SPXT'].pct_change(len(xDF)-1)
```

```
       xDF['PV2_BondTR_growth'] = xDF['PV2_BondTR'].pct_change(len(xDF)-1)
       xDF['PV2_SI_growth'] = xDF['PV2_SI'].pct_change(len(xDF)-1)
       xDF['PV2_growth'] = xDF['PV2'].pct_change(len(xDF)-1)
       xDF['SPXT_100_growth'] = xDF['SPXT_100'].pct_change(len(xDF)-1)

       xGrowth =
xDF[['SPX_growth','SPXT_growth','BondTR_growth','IV_growth','PV1_SPXT_growth','PV1_BondTR_grow
th',\

'PV1_growth','PV2_SPXT_growth','PV2_BondTR_growth','PV2_SI_growth','PV2_growth','SPXT_100_grow
th']].copy()

       xGrowth.dropna(inplace=True)
       xG = xGrowth.T
       xDF.to_csv(xDir + 'xStressTest_'+(str)(xTerm)+'.txt')
       xG.to_csv(xDir + 'xStressTest_Growth'+(str)(xTerm)+'.txt')
       xColName = xG.columns[0]
       xSPXT_exp = xG[xColName]['SPXT_growth']
       xSI_exp = xG[xColName]['IV_growth']
       ########################### find SI value on peak date and trough date
##################
       xSI_peak = xDF.loc[xDF['DATE']==xEffectiveStressStartDate]['IV'].values[0]
       xSI_trough = xDF.loc[xDF['DATE'] == xEffectiveStressEndDate]['IV'].values[0]
       xSI_decline = xSI_trough / xSI_peak - 1.0
       xSPXT_peak = xDF.loc[xDF['DATE'] == xEffectiveStressStartDate]['SPXT'].values[0]
       xSPXT_trough = xDF.loc[xDF['DATE'] == xEffectiveStressEndDate]['SPXT'].values[0]
       xSPXT_decline = xSPXT_trough / xSPXT_peak - 1.0
       ########### we only compare the performance during the stree perio!!!
######################
       ####xDF2 =
xDF.loc[(xDF['DATE']>=xStressStartDate)&(xDF['DATE']<=xStressEndDate)][['DATE','CumRtn_SPXT','
CumRtn_SI','CumRtn_UL']].copy()
       xDF2 =
xDF.loc[(xDF['DATE']>=xEffectiveStressStartDate)&(xDF['DATE']<=xEffectiveStressEndDate)][['DAT
E','CumRtn_SPXT','CumRtn_SI','CumRtn_UL']].copy()

##############################################################################################
       xDF2['Category'] = 'Full Protection'  # = 0 is fully protected!
       xDF2.loc[xDF2['CumRtn_SI']>0,'Category']='Upside Gain'
       xDF2.loc[xDF2['CumRtn_SI']<0,'Category']='No/Partial Protection'

       xPerformance = xDF2.groupby('Category')['CumRtn_SPXT','CumRtn_SI'].mean()
       xDays = xDF2.groupby('Category')['CumRtn_SPXT','CumRtn_SI'].count()
       xPerformance.reset_index(inplace=True)
       xDays.reset_index(inplace=True)

       xDays.rename(columns={'CumRtn_SPXT': 'Days'},inplace=True)

       xPerformance =
pd.merge(xPerformance,xDays[['Category','Days']],on=['Category'],how='left')

       if len(xPerformance.loc[xPerformance['Category']=='Full Protection'])!=0:
          xIndex = xPerformance.loc[xPerformance['Category']=='Full
Protection'].index.values[0]
          xSPXT_FP = xPerformance.values[xIndex][1]
          xSI_FP = xPerformance.values[xIndex][2]
          xDays_FP = xDays.values[xIndex][1]
       else:
          xSPXT_FP = 0.0000000001
          xSI_FP = 0.00000000001
```

```
            xDays_FP = 0.00000000001
            xPerformance = xPerformance.append({'Category': 'Full Protection',
                                        'CumRtn_SPXT': xSPXT_FP, 'CumRtn_SI': xSI_FP, 'Days':
xDays_FP}, \
                                        ignore_index=True)
        #########
        if len(xPerformance.loc[xPerformance['Category']=='No/Partial Protection'])!=0:
            xIndex = xPerformance.loc[xPerformance['Category']=='No/Partial
Protection'].index.values[0]
            xSPXT_NP = xPerformance.values[xIndex][1]
            xSI_NP = xPerformance.values[xIndex][2]
            xDays_NP = xDays.values[xIndex][1]
        else:
            xSPXT_NP = 0.0000000001
            xSI_NP = 0.00000000001
            xDays_NP = 0.00000000001
            xPerformance = xPerformance.append({'Category': 'No/Partial Protection',
                                        'CumRtn_SPXT': xSPXT_NP, 'CumRtn_SI': xSI_NP, 'Days':
xDays_NP}, \
                                        ignore_index=True)
        ###############
        if len(xPerformance.loc[xPerformance['Category']=='Upside Gain'])!=0:
            xIndex = xPerformance.loc[xPerformance['Category']=='Upside Gain'].index.values[0]
            xSPXT_UG = xPerformance.values[xIndex][1]
            xSI_UG = xPerformance.values[xIndex][2]
            xDays_UG = xDays.values[xIndex][1]
        else:
            xSPXT_UG = 0.0000000001
            xSI_UG = 0.00000000001
            xDays_UG = 0.00000000001
            xPerformance = xPerformance.append({'Category': 'Upside Gain',
                                        'CumRtn_SPXT': xSPXT_UG, 'CumRtn_SI': xSI_UG, 'Days':
xDays_UG}, \
                                        ignore_index=True)

        #############
        xPerformance = xPerformance.sort_values(by=['Category'], ascending=True)
        ################
        xPerformanceAll = xDF2[['CumRtn_SPXT','CumRtn_SI']].mean()
        xDaysAll = xDF2[['CumRtn_SPXT','CumRtn_SI']].count()

        xSPXT_all = xPerformanceAll[0]
        xSI_all = xPerformanceAll[1]

        xDays_all = xDaysAll[0]

        xPerformance = xPerformance.append({'Category':'Overall Average',
                    'CumRtn_SPXT':xSPXT_all, 'CumRtn_SI':xSI_all, 'Days':xDays_all}, \
                            ignore_index=True)

        xDays_peak2trough = (xEffectiveStressEndDate-xEffectiveStressStartDate).days

        xPerformance = xPerformance.append({'Category': 'From Peak to Trough',
                                     'CumRtn_SPXT': xSPXT_decline, 'CumRtn_SI': xSI_decline,
'Days': xDays_peak2trough}, \
                                     ignore_index=True)
        # xPerformance = xPerformance.append({'Category':'On Expiration Date',
        #           'CumRtn_SPXT':xSPXT_exp, 'CumRtn_SI':xSI_exp, 'Days':0.00000000001}, \
        #                   ignore_index=True)
```

```
        xPerformance['CumRtn_SPXT'] =
xPerformance['CumRtn_SPXT'].astype(float).map("{:.2%}".format)
        xPerformance['CumRtn_SI'] = xPerformance['CumRtn_SI'].astype(float).map("{:.2%}".format)
        xPerformance['Days'] = xPerformance['Days'].round(0) #.astype(int).map("{:.0}".format)

        xPerformance.rename(columns={'CumRtn_SPXT': 'S&P 500 TR Index','CumRtn_SI':
'SI_'+(str)(xTerm)},inplace=True)

        xResult_String = (str)(xPerformance.astype('string'))

        xPerformance.to_csv(xDir+'xStressTestResult_'+(str)(xTerm)+'.txt')

        globals()['xString_' + (str)(xScenario) + '_' + (str)(xI)] = 'Stress Period #' +
(str)(xI) + ' and Scenario #' +(str)(xScenario) + ':' + \
        '\nStress period from ' + xStressStartDate.strftime('%Y-%m-%d') + ' to ' +
xStressEndDate.strftime('%Y-%m-%d') + \
        '\nSI start date: ' + xStartDate.strftime('%Y-%m-%d') +'; SI maturity date:'
+xEndDate.strftime('%Y-%m-%d') + \
        '\nEffective Stress period from ' + xEffectiveStressStartDate.strftime('%Y-%m-%d') + '
to ' + \
                xEffectiveStressEndDate.strftime('%Y-%m-%d')
        #xString1 = 'From ' + xStartDate.strftime('%Y-%m-%d') + ' to ' + xEndDate.strftime('%Y-
%m-%d') +':'

        xString0 = xString0 + '\n' + globals()['xString_' + (str)(xScenario) + '_' + (str)(xI)]
+ \
            '\n\n' + xResult_String +'\n'

f_w = open(xDir + 'xStressTestResult_' + xBufferType + '_' + (str)(xTerm) + '.txt','w')
f_w.write(xString0)
f_w.close()


#xPerformanceALl.reset_index(inplace=True)
#xDaysAll.reset_index(inplace=True)
```


```
#xPerformanceALl.reset_index(inplace=True)
#xDaysAll.reset_index(inplace=True)
```

```
#################### plot bar chart ########################
from matplotlib import pyplot as plt

def mk_groups(data):
   try:
      newdata = data.items()
   except:
      return

   thisgroup = []
   groups = []
   for key, value in newdata:
      newgroups = mk_groups(value)
      if newgroups is None:
         thisgroup.append((key, value))
      else:
         thisgroup.append((key, len(newgroups[-1])))
         if groups:
            groups = [g + n for n, g in zip(newgroups, groups)]
         else:
            groups = newgroups
   return [thisgroup] + groups

def add_line(ax, xpos, ypos):
   line = plt.Line2D([xpos, xpos], [ypos + .1, ypos],
                     transform=ax.transAxes, color='black')
   line.set_clip_on(False)
   ax.add_line(line)

def label_group_bar(ax, data):
   groups = mk_groups(data)
   xy = groups.pop()
   x, y = zip(*xy)
   ly = len(y)
   xticks = range(1, ly + 1)

   ax.bar(xticks, y, align='center')
   ax.set_xticks(xticks)
   ax.set_xticklabels(x)
   ax.set_xlim(.5, ly + .5)
   ax.yaxis.grid(True)

   scale = 1. / ly
   #for pos in xrange(ly + 1):  # change xrange to range for python3
   for pos in range(ly + 1):
      add_line(ax, pos * scale, -.1)
   ypos = -.2
   while groups:
      group = groups.pop()
      pos = 0
      for label, rpos in group:
         lxpos = (pos + .5 * rpos) * scale
         ax.text(lxpos, ypos, label, ha='center', transform=ax.transAxes)
         add_line(ax, pos * scale, ypos)
         pos += rpos
      add_line(ax, pos * scale, ypos)
      ypos -= .1
###################
```

```
# data = {'Room A':
#                {'Shelf 1':
#                    {'Milk': 10,
#                     'Water': 20},
#                 'Shelf 2':
#                    {'Sugar': 5,
#                     'Honey': 6},
#          'Shelf 2a':
#                    {'Sugar': 7,
#                     'Honey': 8}
#                },
#            'Room B':
#                {'Shelf 1':
#                    {'Wheat': 4,
#                     'Corn': 7},
#                 'Shelf 2':
#                    {'Chicken': 2,
#                     'Cow': 1}
#                }
#            }
data = {'Mar-to-Market':
         {'Full Protection ('+(str)(xDays_FP)+')':
             {'SPXT': xSPXT_FP,
              'SI': xSI_FP},
          'No/Partial Protection ('+(str)(xDays_NP)+')':
             {'SPXT': xSPXT_NP,
              'SI': xSI_NP},
          'Upside Gain ('+(str)(xDays_UG)+')':
             {'SPXT': xSPXT_UG,
              'SI': xSI_UG},
          'Overall Average ('+(str)(xDays_all)+')':
             {'SPXT': xSPXT_all,
              'SI': xSI_all}
          },
      'On Expiration Date':
         {'SPXT': xSPXT_exp,
          'SI': xSI_exp}
      }
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
label_group_bar(ax, data)
fig.subplots_adjust(bottom=0.3)
fig.savefig(xDir + 'xStressTestBarChart_' + (str)(xTerm) + '.png')
fig.show()



#############################
```

#7

```
### Portfolio Optiimization
###
#
# Finds an optimal allocation of stocks in a portfolio,
# satisfying a minimum expected return.
# The problem is posed as a Quadratic Program, and solved
# using the cvxopt library.
# Uses actual past stock data, obtained using the stocks module.
import math

import numpy as np
import pandas as pd
import cvxopt
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt

solvers.options['show_progress'] = False        # !!!

#from cvxopt import solvers
#import stocks
import numpy
import pandas as pd
import datetime

xDir = r'D:\\Users\\ggu\\Documents\\GU\\MeanVarianceOptimization\\'
################## stress test dates #############
xStressDates = pd.read_csv(xDir + 'xMajorDeclineDate.txt')
xStressDates['EndDate'] = pd.to_datetime(xStressDates['EndDate'], format='%Y-%m-%d')

xEndDate = xStressDates['EndDate'][13]  # 2009-03-09
xYears = 3
xStartDate = xEndDate + datetime.timedelta(days=-365 * xYears)

# xSPXT = pd.read_csv(xDir + 'SPXT.txt')
# xSPXT['DATE'] = pd.to_datetime(xSPXT['DATE'], format='%m/%d/%Y')
# xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')
# xAggregateBondTR['DATE'] = pd.to_datetime(xAggregateBondTR['DATE'], format='%m/%d/%Y')
#xSI = pd.read_csv(xDir + 'SI.txt')
#############################################################
xUnderlier = 'SPX'  #'SPX'
xSubDir1 = r'2YearsHardBufferNote\\'
xSubDir2 = r'4YearsBarrierNote\\'
xSubDir3 = r'6YearsTriggerBuffer\\'

xSubText1 = 'Hard Buffer Note #1'
xSubText2 = 'Barrier Buffer Note #2'
xSubText3 = 'Barrier Buffer Note #3'

xBufferNoteNumber = '1' ###'2'  ###'1'  # 3

if xBufferNoteNumber =='1':
   xTerm='2 years'
elif xBufferNoteNumber == '2':
   xTerm='4 years'
elif xBufferNoteNumber == '3':
   xTerm='6 years'

xSubDir = globals()['xSubDir' + xBufferNoteNumber]
```

```
xSubText = globals()['xSubText' + xBufferNoteNumber]

xSPXT = pd.read_csv(xDir + xSubDir + 'xCalcRtnsOverTerm4SI_' + xUnderlier + '.txt',usecols =
['DATE','SI_100','SPXT_100', \

'BondTR_100','SPX_100','SPX_term_100','BondTR_term_100','SPXT_term_100','SPX_term_100',\
            'BondTR_rtn_term','SPXT_rtn_term','SI_rtn_term', 'BondTR_rtn_1_year',\

'SPXT_rtn_1_year','SI_rtn_1_year','SPXT_rtn_1_year_roll','BondTR_rtn_1_year_roll'])
xSPXT['DATE'] = pd.to_datetime(xSPXT['DATE'], format='%Y-%m-%d')

#xSPXT = xSI2.copy()
xSPXT.rename(columns={'SI_100':'SI','BondTR_100':'BondTR','SPXT_100':'SPXT','SPX_100':'SPX', \

'SPX_term_100':'SPX_term','BondTR_term_100':'BondTR_term','SPXT_term_100':'SPXT_term','SPX_ter
m_100':'SPX_term'},inplace=True)
xSPXT['SPXT_rtn'] = xSPXT['SPXT'].pct_change()
xSPXT['SI_rtn'] = xSPXT['SI'].pct_change()
xSPXT['BondTR_rtn'] = xSPXT['BondTR'].pct_change()
xSPXT['SPX_rtn'] = xSPXT['SPX'].pct_change()
xSPXT['SPX_term_rtn'] = xSPXT['SPX_term'].pct_change()
xSPXT['BondTR_term_rtn'] = xSPXT['BondTR_term'].pct_change()
xSPXT['SPXT_term_rtn'] = xSPXT['SPXT_term'].pct_change()
xSPXT['SPX_term_rtn'] = xSPXT['SPX_term'].pct_change()

xSPXT = xSPXT.dropna()

#####xSPXT = xSPXT.loc[(xSPXT['DATE'] >= xStartDate) & (xSPXT['DATE'] <= xEndDate)]

##################### the following are efficient frontiers based on 1-YEAR returns for SPXT,
BondTR and SI #######
########## 1 year return for SI derived from 2/4/6 years return; 1-year returns for SPXT and
BondTR from daily prices ###########
############################ EQUITY AND BOND ONLY #################
import matplotlib.pyplot as plt4
xSI_indicator = False
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year', 'SI_rtn_1_year']]
else:
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

xAnnStd_Equity_Bond = xRtns.std(axis=0)  ############# * numpy.sqrt(252)

std_vec = xAnnStd_Equity_Bond
print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
```

```
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
xCov_Equity_Bond = xRtns.cov()  ########## * (252 ^ 2)
print('xCov_Equity_Bond: ', xCov_Equity_Bond)

covs = xCov_Equity_Bond.values

print('xCov_Equity_Bond: ', covs)
xAnnRtn_Equity_Bond = xRtns.mean(axis=0)

avg_ret = cvxopt.matrix(xAnnRtn_Equity_Bond)    #.T
avg_ret = avg_ret ################ * 252   #annualized
print('avg_ret: ', avg_ret)

#########################
xCorr_Equity_Bond = xRtns.corr()
corr = xCorr_Equity_Bond.values
################ alternative way to calculate covs ########
xL = [std_vec[0], std_vec[1]]
xDiag_std = np.diag(xL)
# covs = std_vec * corr * std_vec.T
covs_2 = cvxopt.matrix(xDiag_std) * cvxopt.matrix(corr) * cvxopt.matrix(xDiag_std)
### note:  this calculation is slightly different from the
############################################################
xRisk_Rtn_Corr_Eqy_Bnd = 'AnnStd: \n' + (str)(round(xAnnStd_Equity_Bond,4).astype('string')) + 
\
            '\nAnnRtn: \n' + (str)(round(xAnnRtn_Equity_Bond,4).astype('string')) + \
            '\nCorr: \n' + (str)(round(xCorr_Equity_Bond,4).astype('string'))
#########
############

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)

# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

###################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
```

```
# Output: sol - cvxopt solution object
##########<=mmodified = R ################
def optimize_portfolio_modified(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.transpose(numpy.array(avg_ret)),
   #  -numpy.identity(n)), 0))
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.identity(n)), 0))
   G = cvxopt.matrix(-np.diag(np.ones(n),0))
   # h = cvxopt.matrix(numpy.concatenate((
   #  -numpy.ones((1, 1))*r_min,
   #  numpy.zeros((n, 1))), 0))
   h = cvxopt.matrix(numpy.concatenate((
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   #-numpy.transpose(numpy.array(avg_ret)),
   #A = cvxopt.matrix(1.0, (1, n))
   A = cvxopt.matrix(numpy.concatenate((
      numpy.transpose(numpy.array(avg_ret)),
      cvxopt.matrix(1.0, (1, n)))))
   #b = cvxopt.matrix(1.0)
   b = cvxopt.matrix(numpy.concatenate((
      numpy.ones((1, 1)) * r_min,
      cvxopt.matrix(1.0))))
   # print('P = ', P)
   # print('q = ', q)
   # print('G = ', G)
   # print('h = ', h)
   # print('A = ', A)
   # print('b = ', b)
   # A = numpy.matrix(1.0, (1, n))
   # print('A = ', A)
   sol = solvers.qp(P, q, G, h, A, b)
   return sol

############## original version ##################
def optimize_portfolio(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   G = cvxopt.matrix(numpy.concatenate((
      -numpy.transpose(numpy.array(avg_ret)),
      -numpy.identity(n)), 0))
```

```
    h = cvxopt.matrix(numpy.concatenate((
       -numpy.ones((1, 1))*r_min,
       numpy.zeros((n, 1))), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    A = cvxopt.matrix(1.0, (1, n))
    b = cvxopt.matrix(1.0)
    # print('P = ', P)
    # print('q = ', q)
    # print('G = ', G)
    # print('h = ', h)
    # print('A = ', A)
    # print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol
###############################################
# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = (r_max2 - r_min2) / 100  ###############0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
    print('delta_r: ', delta_r)
    w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
    print('w: ', w)
    print('w.T', w.T)
    w2 = numpy.matrix(w.T)
    print('w2.T', w2)
    return2 = (w.T * avg_ret)[0]
    risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
    print('return2: ', return2)
    print('risk2: ', risk2)

    returns.append(return2)
    risks.append(risk2)

    w2 = numpy.insert(w2, w2.size, [risk2, return2])
    print('w2:', w2)
    df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
```

```
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_Equity_Bond_1_year.txt')

fig4, ax4 = plt4.subplots()
ax4.plot(risks, returns, color='red', label='Equity/Bond')
fig4.suptitle('Efficient Frontiers for ' + xSubText, fontsize=16,y=0.95)
ax4.set_xlabel('Annual Risk', fontsize=10)
ax4.set_ylabel('Annual Return', fontsize=10)

# plt.ylabel('mean')
# plt.xlabel('std')
# plt.title('Efficient Frontier xx with underlying index ' + xUnderlier)
# #plt.plot(risks, returns, 'y-o')
# plt.plot(risks, returns, color='red',label='Equity/Bond')
# plt.legend(loc='lower right')
# import matplotlib.ticker as mtick
# plt.axis()

xStock_scater = plt4.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
xBond_scatter = plt4.scatter(std_vec[1], avg_ret[1], marker='*', color='green',label='Bond')
#bond
if xSI_indicator:
   if xCash:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
      xCash_scatter = plt4.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
   else:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
   else:
      print ('nothing here')
#plt.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
############################

##plt.xlim(xmin=0)
##plt.ylim(ymin=0.02)

##plt.show()
############# CASE 2 ###########################
############################ EQUITY, BOND ONLY AND SI #################
xSI_indicator = True
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year', 'SI_rtn_1_year']].copy()
else:
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year']].copy()

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252
```

```
#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

xAnnStd_Equity_Bond = xRtns.std(axis=0)  ############# * numpy.sqrt(252)

std_vec = xAnnStd_Equity_Bond
print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
xCov_Equity_Bond = xRtns.cov()  ########## * (252 ^ 2)
print('xCov_Equity_Bond: ', xCov_Equity_Bond)

covs = xCov_Equity_Bond.values

print('xCov_Equity_Bond: ', covs)
xAnnRtn_Equity_Bond = xRtns.mean(axis=0)

avg_ret = cvxopt.matrix(xAnnRtn_Equity_Bond)    #.T
avg_ret = avg_ret ################ * 252   #annualized
print('avg_ret: ', avg_ret)

#########################
xCorr_Equity_Bond = xRtns.corr()
corr = xCorr_Equity_Bond.values
################ alternative way to calculate covs ########
xL = [std_vec[0],std_vec[1],std_vec[2]]
xDiag_std = np.diag(xL)
#covs = std_vec * corr * std_vec.T
covs_2 = cvxopt.matrix(xDiag_std) * cvxopt.matrix(corr) * cvxopt.matrix(xDiag_std)
### note:  this calculation is slightly different from the
#########################################################
xRisk_Rtn_Corr_Eqy_Bnd_SI = 'AnnStd: \n' +
(str)(round(xAnnStd_Equity_Bond,4).astype('string')) + \
            '\nAnnRtn: \n' + (str)(round(xAnnRtn_Equity_Bond,4).astype('string')) + \
            '\nCorr: \n' + (str)(round(xCorr_Equity_Bond,4).astype('string'))
f_w = open(xDir + 'xRisk_Rtn_Corr_Eqy_Bnd_SI_1_year_' + xSubText + '.txt','w')
f_w.write(xRisk_Rtn_Corr_Eqy_Bnd_SI)
f_w.close()
######### testing debug #########
####avg_ret[2] = avg_ret[2] * 1.2
##########################

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)
```

```
r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)

# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

####################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
##########<=mmodified = R ################
def optimize_portfolio_modified(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.transpose(numpy.array(avg_ret)),
   #  -numpy.identity(n)), 0))
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.identity(n)), 0))
   G = cvxopt.matrix(-np.diag(np.ones(n),0))
   # h = cvxopt.matrix(numpy.concatenate((
   #  -numpy.ones((1, 1))*r_min,
   #  numpy.zeros((n, 1))), 0))
   h = cvxopt.matrix(numpy.concatenate((
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   #-numpy.transpose(numpy.array(avg_ret)),
   #A = cvxopt.matrix(1.0, (1, n))
   A = cvxopt.matrix(numpy.concatenate((
      numpy.transpose(numpy.array(avg_ret)),
      cvxopt.matrix(1.0, (1, n)))))
   #b = cvxopt.matrix(1.0)
   b = cvxopt.matrix(numpy.concatenate((
      numpy.ones((1, 1)) * r_min,
      cvxopt.matrix(1.0))))
   # print('P = ', P)
   # print('q = ', q)
   # print('G = ', G)
   # print('h = ', h)
   # print('A = ', A)
   # print('b = ', b)
```

```
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol

############## original version ##################
def optimize_portfolio(n, avg_ret, covs, r_min):
    P = cvxopt.matrix(covs)
    # x = variable(n)
    q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
    # inequality constraints Gx <= h
    # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
    # note: the loop starts from the lowest return to the highest return
    # if the lowest return has a higher risk, this constraint will find a
    # higher return corresponding to the lowest risk!!!  that is why there
    # is no line (or no curve) on the efficient frontier from the return
    # corresponding to the minimal risk to the lowest return.
    G = cvxopt.matrix(numpy.concatenate((
       -numpy.transpose(numpy.array(avg_ret)),
       -numpy.identity(n)), 0))
    h = cvxopt.matrix(numpy.concatenate((
       -numpy.ones((1, 1))*r_min,
       numpy.zeros((n, 1))), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    A = cvxopt.matrix(1.0, (1, n))
    b = cvxopt.matrix(1.0)
    # print('P = ', P)
    # print('q = ', q)
    # print('G = ', G)
    # print('h = ', h)
    # print('A = ', A)
    # print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol
###############################################
# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = (r_max2 - r_min2) / 100 ############0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
```

```
   print('delta_r: ', delta_r)
   w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
   print('w: ', w)
   print('w.T', w.T)
   w2 = numpy.matrix(w.T)
   print('w2.T', w2)
   return2 = (w.T * avg_ret)[0]
   risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
   print('return2: ', return2)
   print('risk2: ', risk2)

   returns.append(return2)
   risks.append(risk2)

   w2 = numpy.insert(w2, w2.size, [risk2, return2])
   print('w2:', w2)
   df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_equity_bond_SI_1_year.txt')

ax4.plot(risks, returns,color='blue',label='Equity/Bond/SI')
import matplotlib.ticker as mtick
#plt.axis()

#xStock_scater = plt.scatter(std_vec[0], avg_ret[0], marker='x', color='red',label='Stock')
#stock
#xBond_scatter = plt.scatter(std_vec[1], avg_ret[1], marker='*', color='green')    #bond
if xSI_indicator:
   if xCash:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',lable='SI')
# SI
      xCash_scatter = plt4.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
   else:
      xSI_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
else:
   if xCash:
      xCash_scatter = plt4.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
   else:
      print('nothing here')
#plt3.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
import matplotlib.ticker as mtick4

#fig = plt.figure(1)
#fig.add_subplot(111)
#ax = fig.add_subplot(111)

#ax.plot(perc, data)

fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
```

```
##xticks = mtick.FormatStrFormatter(fmt)
xticks = mtick4.FuncFormatter("{:.0%}".format)
ax4.xaxis.set_major_formatter(xticks)
ax4.yaxis.set_major_formatter(xticks)

############# CASE 3a : THIS HAS 25% cap on SI WEIGHT!!! ###########################
#############1 year returns from daily prices for EQUITY, BOND; AND SI uses 1 year return
2/4/6 year returns ###########
xSI_indicator = True
if (xSI_indicator):
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year', 'SI_rtn_1_year']]
else:
   xRtns = xSPXT[['SPXT_rtn_1_year', 'BondTR_rtn_1_year']]

xCash = False
if xCash:
   xRtns['cash_rtn'] = 0.025 / 252

#WKPRICE['rtn_w'] = WKPRICE.groupby('CUSIP')['PRICE'].pct_change()
#WKPRICE['std_w'] =
WKPRICE.groupby('CUSIP')['rtn_w'].apply(pd.rolling_std,window=52*2,min_periods=26)
#WKPRICE.rename(columns={'DATE':'TueDATE'},inplace=True)

print(xSPXT.head())
print(xRtns.head())
print(xRtns.tail())

std_vec = cvxopt.matrix(xRtns.std(axis=0)) * numpy.sqrt(252)
################
##std_vec[2] = 0.06
#############
print('daily obs:\n', xRtns.count(axis=0))
print('daily mean:\n', xRtns.mean(axis=0))
print('daily Std:\n', xRtns.std(axis=0))
print('correlation:\n', xRtns.corr())
print('covariance:\n', xRtns.cov())
print(xRtns.describe())

#A = xRtns.values
print('xRtns: ', xRtns.head())
##print('A: ', A)
covs = xRtns.cov() ############ * (252 ^ 2)
print('covs: ', covs)

covs = covs.values
print('covs: ', covs)

corr = xRtns.corr()
corr = corr.values

################ alternative way to calculate covs ########
xL = [std_vec[0],std_vec[1],std_vec[2]]
xDiag_std = np.diag(xL)
#covs = std_vec * corr * std_vec.T
covs_2 = cvxopt.matrix(xDiag_std) * cvxopt.matrix(corr) * cvxopt.matrix(xDiag_std)
### note:  this calculation is slightly different from the above #########
##################

avg_ret = cvxopt.matrix(xRtns.mean(axis=0))    #.T
avg_ret = avg_ret ####### no more * 252    #annualized
```

```
print('avg_ret: ', avg_ret)
######### testing #########
###avg_ret[2] = avg_ret[2] / 3
##########################

n = len(avg_ret)
print('n = ', n)
r_min2 = min(avg_ret)
print('r_min2 = ', r_min2)

r_max2 = max(avg_ret)
print('r_max2 = ', r_max2)


# from numpy.linalg import eig
# values, vectors = eig(covs)
# print('values: ', values)
# print('eigen vector: ', vectors)

##################################################################
# solves the QP, where x is the allocation of the portfolio:
# minimize   x'Px + q'x
# subject to Gx <= h
#            Ax == b
#
# Input:  n       - # of assets
#         avg_ret - nx1 matrix of average returns
#         covs    - nxn matrix of return covariance
#         r_min   - the minimum expected return that you'd
#                   like to achieve
# Output: sol - cvxopt solution object
##########<=mmodified = R ################
def optimize_portfolio_modified(n, avg_ret, covs, r_min):
   P = cvxopt.matrix(covs)
   # x = variable(n)
   q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
   # inequality constraints Gx <= h
   # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
   # note: the loop starts from the lowest return to the highest return
   # if the lowest return has a higher risk, this constraint will find a
   # higher return corresponding to the lowest risk!!!  that is why there
   # is no line (or no curve) on the efficient frontier from the return
   # corresponding to the minimal risk to the lowest return.
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.transpose(numpy.array(avg_ret)),
   #  -numpy.identity(n)), 0))
   #G = cvxopt.matrix(numpy.concatenate((
   #  -numpy.identity(n)), 0))
   G = cvxopt.matrix(-np.diag(np.ones(n),0))
   # h = cvxopt.matrix(numpy.concatenate((
   #  -numpy.ones((1, 1))*r_min,
   #  numpy.zeros((n, 1))), 0))
   h = cvxopt.matrix(numpy.concatenate((
      numpy.zeros((n, 1))), 0))
   # equality constraint Ax = b; captures the constraint sum(x) == 1
   #-numpy.transpose(numpy.array(avg_ret)),
   #A = cvxopt.matrix(1.0, (1, n))
   A = cvxopt.matrix(numpy.concatenate((
      numpy.transpose(numpy.array(avg_ret)),
      cvxopt.matrix(1.0, (1, n)))))
```

```
    #b = cvxopt.matrix(1.0)
    b = cvxopt.matrix(numpy.concatenate((
       numpy.ones((1, 1)) * r_min,
       cvxopt.matrix(1.0))))
    # print('P = ', P)
    # print('q = ', q)
    # print('G = ', G)
    # print('h = ', h)
    # print('A = ', A)
    # print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol

############## original version ##################
def optimize_portfolio(n, avg_ret, covs, r_min):
    P = cvxopt.matrix(covs)
    # x = variable(n)
    q = cvxopt.matrix(numpy.zeros((n, 1)), tc='d')
    # inequality constraints Gx <= h
    # captures the constraints (avg_ret'x >= r_min) and (x >= 0)
    # note: the loop starts from the lowest return to the highest return
    # if the lowest return has a higher risk, this constraint will find a
    # higher return corresponding to the lowest risk!!!  that is why there
    # is no line (or no curve) on the efficient frontier from the return
    # corresponding to the minimal risk to the lowest return.
    G = cvxopt.matrix(numpy.concatenate((
       -numpy.transpose(numpy.array(avg_ret)),
       -numpy.identity(n),
       ), 0))
    v = (G[G.size[0] - 1, :])
    v[0, v.size[1] - 1] = 1
    G = cvxopt.matrix(numpy.concatenate((G, v), 0))

    h = cvxopt.matrix(numpy.concatenate((
       -numpy.ones((1, 1))*r_min,
       numpy.zeros((n, 1)),
       numpy.ones((1, 1)) * 0.25), 0))
    # equality constraint Ax = b; captures the constraint sum(x) == 1
    A = cvxopt.matrix(1.0, (1, n))
    b = cvxopt.matrix(1.0)
    # print('P = ', P)
    # print('q = ', q)
    # print('G = ', G)
    # print('h = ', h)
    # print('A = ', A)
    # print('b = ', b)
    # A = numpy.matrix(1.0, (1, n))
    # print('A = ', A)
    sol = solvers.qp(P, q, G, h, A, b)
    return sol
###############################################
# ### setup the parameters
# symbols = ['GOOG', 'AIMC', 'CE', 'BH', 'AHGP', 'AB', 'HLS', 'BKH', 'LUV']
# # pull data from this date range
# start   = '1/1/2010'
# end     = '1/1/2014'
# n       = len(symbols)
# # average yearly return for each stock
```

```
# avg_ret = matrix(map(lambda s: stocks.avg_return(s, start, end, 'y'), symbols))
# # covariance of asset returns
# covs    = matrix(numpy.array(stocks.cov_matrix(symbols, start, end, 'y')))
# # minimum expected return threshold

### solve

P = cvxopt.matrix(covs)
returns = []
risks = []
portfolios = []
df = pd.DataFrame()
columns = ['w_{}'.format(x) for x in range(1, n + 1)] + ['risk', 'return']

xStep = 0.001   #0.001
for delta_r in numpy.arange(r_min2, r_max2, xStep):
   print('delta_r: ', delta_r)
   w = optimize_portfolio(n, avg_ret, covs, delta_r)['x']
   print('w: ', w)
   print('w.T', w.T)
   w2 = numpy.matrix(w.T)
   print('w2.T', w2)
   return2 = (w.T * avg_ret)[0]
   risk2 = numpy.asscalar(numpy.sqrt(w.T * P * w))
   print('return2: ', return2)
   print('risk2: ', risk2)

   returns.append(return2)
   risks.append(risk2)

   w2 = numpy.insert(w2, w2.size, [risk2, return2])
   print('w2:', w2)
   df = df.append(pd.DataFrame(w2, columns=[columns]), ignore_index=True)

print('df_portfolios: \n', df)
# print('df_portfolios: \n', df.head())
# print('df_portfolios: \n', df.tail())

df.to_csv(xDir + 'xOptimalPortfolio_equity_bond_SI_1_year_25pct.txt')

#import matplotlib.ticker as mtick

#fig = plt.figure(1)
#fig.add_subplot(111)
#ax = fig.add_subplot(111)

#ax.plot(perc, data)

# fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
# ##xticks = mtick.FormatStrFormatter(fmt)
# xticks = mtick.FuncFormatter("{:.0%}".format)
# ax4.xaxis.set_major_formatter(xticks)
# ax4.yaxis.set_major_formatter(xticks)

#plt.ylabel('mean')
#plt.xlabel('std')
#plt.title('Efficient Frontier xxxx with underlying index ' + xUnderlier)
#plt.plot(risks, returns, 'y-o')
ax4.plot(risks, returns,color='black',label='Equity/Bond/SI with max 25% on SI')
import matplotlib.ticker as mtick
```

```
#plt.axis()

#xStock_scater = plt.scatter(std_vec[0], avg_ret[0], marker='x', color='red')  #stock
#xBond_scatter = plt.scatter(std_vec[1], avg_ret[1], marker='*', color='green')    #bond
# if xSI_indicator:
#  if xCash:
#     xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black', label='SI')
# SI
#     xCash_scatter = plt.scatter(std_vec[3], avg_ret[3], marker='+',
color='blue',label='Cash')  # cash
#  else:
#     #print('hererrrrrrrr')
#     xSI_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='X', color='black',label='SI')
# SI
# else:
#  if xCash:
#     xCash_scatter = plt.scatter(std_vec[2], avg_ret[2], marker='+',
color='blue',label='Cash')  # cash
#  else:
#     print('nothing here')
#plt.show()
#plt.show(block=False)
#plt.interactive(False)
#plt.show(block=True)
#plt.interactive(False)
############################
################################
plt4.grid(which='both')
plt4.legend(loc='best', ncol=2,facecolor='white')
plt4.xlim(xmin=0)
plt4.ylim(ymin=0)

plt4.savefig(xDir + 'EfficientFrontier_'+xSubText+'_1_year.png')
plt4.show()
############################
```

#8

```
### Portfolio Optiimization
###
#
# Finds an optimal allocation of stocks in a portfolio,
# satisfying a minimum expected return.
# The problem is posed as a Quadratic Program, and solved
# using the cvxopt library.
# Uses actual past stock data, obtained using the stocks module.
import math
import numpy as np
import pandas as pd
import datetime
import cvxopt
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt
##########################
import warnings
warnings.filterwarnings('ignore')
warnings.warn('DelftStack')
warnings.warn('Do not show this message')
#####################
solvers.options['show_progress'] = False        # !!!

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

#from cvxopt import solvers
#import stocks
import numpy
import pandas as pd
import datetime

# c = cvxopt.matrix([0, -1], tc='d')
# print('c: ', c)
# c = numpy.matrix(c)
# print('c: ', c)
#
# c = cvxopt.matrix([0, -1])
# print('c: ', c)
# G = cvxopt.matrix([[-1, 1], [3, 2], [2, 3], [-1, 0], [0, -1]], tc='d')
# print('G: ', G)
##################
xDir = r'D:\\Users\\ggu\\Documents\\GU\\MeanVarianceOptimization\\'
xSPXT = pd.read_csv(xDir + 'SPXT.txt')
xSPXT['DATE'] = pd.to_datetime(xSPXT['DATE'], format='%m/%d/%Y')
xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')
xAggregateBondTR['DATE'] = pd.to_datetime(xAggregateBondTR['DATE'], format='%m/%d/%Y')

# xSI = pd.read_csv(xDir + 'SI.txt')
# xSI['DATE'] = pd.to_datetime(xSI['DATE'], format='%m/%d/%Y')

xSPX = pd.read_csv(xDir + 'SPX.txt')
xSPX['DATE'] = pd.to_datetime(xSPX['DATE'], format='%m/%d/%Y')

##xAggregateBondTR = pd.read_csv(xDir + 'AggregateBondTR.txt')

print(xSPXT.head())
```

```
print(xAggregateBondTR.head())
#print(xSI.head())
print(xSPX.head())

# xSPXT = pd.merge(xSPXT, xSI, on=['DATE'], how='left')
xSPXT = pd.merge(xSPXT, xAggregateBondTR, on=['DATE'], how='left')
xSPXT = pd.merge(xSPXT, xSPX, on=['DATE'], how='left')

# xMinDateSI = xSI['DATE'].min()
# xMaxDateSI = xSI['DATE'].max()

###xSPXT = xSPXT.loc[(xSPXT['DATE'] >= xMinDateSI) & (xSPXT['DATE'] <= xMaxDateSI)]

#xSPXT['intrinsic_value'].fillna(method='ffill', inplace=True) #fill N/As with previous
prices!!!!
xSPXT['LBUSTRUU'].fillna(method='ffill', inplace=True) #fill N/As with previous prices!!!!
xSPXT['SPX'].fillna(method='ffill', inplace=True)

xSPXT.rename(columns={'LBUSTRUU': 'BondTR'},inplace=True)

xSPXT['SPXT_rtn'] = xSPXT['SPXT'].pct_change()
#xSPXT['SI_rtn'] = xSPXT['intrinsic_value'].pct_change()
xSPXT['BondTR_rtn'] = xSPXT['BondTR'].pct_change()
xSPXT['SPX_rtn'] = xSPXT['SPX'].pct_change()

xSPXT.to_csv(xDir + 'xSPXT.txt')

xSPXT = xSPXT.dropna()
########################
xUnderlier = 'SPX'

#xDF0 = xSPXT[['DATE', xUnderlier,'SPXT','SPXT_rtn','BondTR','BondTR_rtn']]
xDF0 = xSPXT[['DATE', xUnderlier,'SPXT','BondTR', 'SPX_rtn','SPXT_rtn']]
print('xDF0 = ', xDF0.head())

### These are the generic products we used in learning center.
#- 2Y, 10% hard buffer, 1.5x upside up to 21%
#- 4Y, 25% barrier, 1x upside no-cap
#- 6Y, 30% barrier, 1.15x upside no-cap
#################################################################

xCap = 100000 #0.21  #10000  #100000 #0.21   #10000 #0.21 #0.21
xBuffer = -0.30 #-0.30  #250  #-0.25   #-0.30   # -0.10   #-0.25

xTerm = 6  #2  #4  #6   #4 #2  #3 # years
xAmount = 100000
xLever = 1.15  #1.15
xBufferType = "T"  #"T"  # "H" for regular Buffer; "G" for Geared Buffer (or Barrier); "T" for
Trigger Buffer!

xPortfolio = pd.DataFrame()

####################

xDate = '2007-10-09'   #'2000-01-01'
xStartDate = pd.to_datetime(xDate)   #datetime.date.fromisoformat(xDate)
#########################################################################
print('xStartDate = ', xStartDate)
xEndDate = xStartDate + datetime.timedelta(days = 365*xTerm)
print('xEndDate = ', xEndDate)
```

```
    #################### retrieve the stress start and end dates ############################
    xStressDates = pd.read_csv(xDir + 'xMajorDeclineDate.txt', usecols=['StartDate','EndDate'])
    xStressDates['StartDate'] = pd.to_datetime(xStressDates['StartDate'], format='%Y-%m-%d')
    xStressDates['EndDate'] = pd.to_datetime(xStressDates['EndDate'], format='%Y-%m-%d')

    xStressDateSet = []
    #xI = 0
    xResult_string = ''
    for xI in xStressDates.index:
        xStressStartDate = pd.to_datetime(xStressDates.StartDate.values[xI],format='%Y-%m-%d')
        xStressEndDate = pd.to_datetime(xStressDates.EndDate.values[xI],format='%Y-%m-%d')

        xActualDF = xDF0.loc[(xDF0['DATE'] >= xStressStartDate) & (xDF0['DATE'] < xStressEndDate)]

        xPeakSPX = xDF0.loc[xDF0['DATE']==xStressStartDate]['SPX'].values[0]
        xTroughSPX = xDF0.loc[xDF0['DATE']==xStressEndDate]['SPX'].values[0]

        xMDD = (xTroughSPX - xPeakSPX) / xPeakSPX

        ################ calculate mean and std dev for 6 years (xTerm) back from xPeak Date
    ########
        xSampleStartDate = xStressStartDate + datetime.timedelta(days = -365*xTerm)
        xTemp = xDF0.loc[(xDF0['DATE']>=xSampleStartDate) & (xDF0['DATE']<xStressStartDate)]
        if True:    # 15 years from 2005 to 2020
            xTemp = xDF0.loc[(xDF0['DATE'] >= pd.to_datetime('2005-01-01')) & (xDF0['DATE'] <
    pd.to_datetime('2020-12-31'))]
        else:
            xStressDateSet = xStressDateSet + pd.date_range(xStressStartDate, xStressEndDate,
    freq='B').tolist()
            if False:
               xTemp = xDF0.loc[~xDF0['DATE'].isin(xStressDateSet)]
            else:
                xTemp = xDF0.loc[~xDF0['DATE'].isin(xStressDateSet) & (xDF0['DATE'] <
    xStressStartDate)]
        xMu = xTemp['SPX_rtn'].mean() * 252 #annualized
        xSigma = xTemp['SPX_rtn'].std() * np.sqrt(252) #annualized
        xS0 = xTroughSPX

        ##################
        xEndDate0 = xStressStartDate + datetime.timedelta(days = 365*xTerm)
        xDF0['Days'] = (xDF0['DATE'] - xEndDate0).dt.days
        xTemp = xDF0.loc[xDF0['Days'] <= 0]
        xTemp.reset_index(drop=True, inplace=True)
        xEndDate = xTemp['DATE'][len(xTemp) - 1]  # this is the trading date!

        ############### if the SI term is less than the stress period ##########
        if xTerm < ((xStressEndDate - xStressStartDate).days / 365):
            xIndexValueOnEndDate = xDF0.loc[xDF0['DATE'] == xEndDate]['SPX'].values[0]
            xSPXTOnEndDate = xDF0.loc[xDF0['DATE'] == xEndDate]['SPXT'].values[0]
            xSPXTOnStartDate = xDF0.loc[xDF0['DATE'] == xStressStartDate]['SPXT'].values[0]

            xIndexGrowth = xIndexValueOnEndDate / xPeakSPX - 1
            xSPXTGrowth = xSPXTOnEndDate / xSPXTOnStartDate - 1
            if xBufferType == 'H':
                xSIGrowth = xIndexGrowth - xBuffer
            elif xBufferType == 'T':
                if xIndexGrowth >= xBuffer:
                    xSIGrowth = 0
                else:
                    xSIGrowth = xIndexGrowth
```

```
        elif xBufferType == 'G':
            xK = 1 / (1 + xBuffer)  # 100/(100-30) = 10/7
            xSIGrowth = xK * (xIndexGrowth - xBuffer)
        xString1 = ''
        xSubTitle = ''
        if xI == 0:
            xSubTitle = 'Stress Period: Dotcom bubbles burst (' \
                        + xStressStartDate.strftime('%m/%d/%Y') + ' - ' +
xStressEndDate.strftime('%m/%d/%Y') + ')'
        elif xI == 1:
            xSubTitle = 'Stress Period: Financial crisis (' \
                        + xStressStartDate.strftime('%m/%d/%Y') + ' - ' +
xStressEndDate.strftime('%m/%d/%Y') + ')'
        elif xI == 2:
            xSubTitle = 'Stress Period: COVID-19 meltdown (' \
                        + xStressStartDate.strftime('%m/%d/%Y') + ' - ' +
xStressEndDate.strftime('%m/%d/%Y') + ')\n'
        if xBufferType == 'H':
            xString3 = 'Structure: ' + 'Buffer Type = ' + 'Hard Buffer' + '; Term = ' +
(str)(xTerm) + ' years; ' + (
                str)(xLever) + 'x Underlier; Cap = ' + '{:.1%}'.format(xCap) + '; Buffer = ' +
'{:.1%}'.format(xBuffer) + '\n'
        elif xBufferType == 'T':
            xString3 = 'Structure: ' + 'Buffer Type = ' + 'Barrier Buffer' + '; Term = ' +
(str)(xTerm) + ' years; ' + (
                str)(xLever) + 'x Underlier; Cap = ' + '{:.1%}'.format(xCap)+ '; Buffer = ' +
'{:.1%}'.format(xBuffer) + '\n'
        elif xBufferType == 'G':
            xString3 = 'Structure: ' + 'Buffer Type = ' + 'Geared Buffer' + '; Term = ' +
(str)(xTerm) + ' years; ' + (
                str)(xLever) + 'x Underlier; Cap = ' + '{:.1%}'.format(xCap) + '; Buffer = ' +
'{:.1%}'.format(xBuffer) + '\n'
        xString2 = 'From ' + xStressStartDate.strftime('%m/%d/%Y') + ' to ' +
xEndDate.strftime('%m/%d/%Y') + ':\n' +\
             'SI: ' + '{:.1%}'.format(xSIGrowth) + '\n' + 'SPXT: ' +
'{:.1%}'.format(xSPXTGrowth) + '\n'

        xString1 = xSubTitle + '\n' + xString3 + xString2
        f_w = open(xDir + 'xActualResult_' + xBufferType + '_' + (str)(xTerm) + '_' +
(str)(xI) + '.txt', 'w')
        f_w.write(xString1)
        f_w.close()
        continue
    ####################### end of term < stress period #################################
    if len(xDF0.loc[xDF0['Days']>0]) == 0:
        xFutureDates = pd.bdate_range(start= (xEndDate + datetime.timedelta(days =
1)),end=xEndDate0)
        for xTempDate in xFutureDates:
            xDF0 = xDF0.append({'DATE': xTempDate}, ignore_index = True)
        xDF0['Days'] = (xDF0['DATE'] - xEndDate0).dt.days
        xTemp = xDF0.loc[xDF0['Days']<=0]
        xTemp.reset_index(drop=True,inplace=True)
        xEndDate = xTemp['DATE'][len(xTemp)-1] # this is the trading date!
    xTemp = xDF0.loc[(xDF0['DATE']>=xStressEndDate) & (xDF0['DATE']<=xEndDate)]
    xCounts = len(xTemp)

    xDates_axis = xTemp['DATE'].tolist()
    xActual = xTemp['SPX'].tolist()

    # Creates a list containing 5 lists, each of 8 items, all set to 0
```

```
    #w, h = 8, 5
    #Matrix = [[0 for x in range(w)] for y in range(h)]

    xBucketDF = pd.DataFrame()
    xBucketDF = xBucketDF.append({'Name': 'Above Peak','Level': xPeakSPX}, ignore_index=True)
    #xBucketDF = xBucketDF.append({'Name': 'ActualAtEnd','Level': xActual[len(xActual) - 1]},
ignore_index=True)
    xBucketDF = xBucketDF.append({'Name': 'Between Peak and Buffer','Level': xPeakSPX * (1 +
xBuffer)}, ignore_index=True)
    #xBucketDF = xBucketDF.append({'Name': 'Trough','Level': xTroughSPX}, ignore_index=True)

    xBucketDF.sort_values(by=['Level'], ascending=False,  inplace=True)
    xBucketDF.reset_index(drop=True,inplace=True)

    xTotalNo = 0
    xAboveNo_0 = 0
    xNo_0_1 = 0
    #xNo_1_2 = 0
    #xNo_2_3 = 0
    #xBelowNo_3 = 0
    xBelowNo_1 = 0

    xAbove_0 = 0
    x0_1 = 0
    #x1_2 = 0
    #x2_3 = 0
    #xBelow_3 = 0
    xBelow_1 = 0

    xAboveAvg_0 = 0
    xAvg_0_1 = 0
    #xAvg_1_2 = 0
    #xAvg_2_3 = 0
    #xAvg_Below_3 = 0
    xAvg_Below_1 = 0

    xSet_above_0 = set()
    xSet_0_1 = set()
    xSet_below_1 = set()

    xPaths = 5001
    xP = [[0 for x in range(xCounts)] for y in range(xPaths)]

    xPath = 0
    for xPath in range(0,xPaths):
        xN = np.random.normal(0, 1, xCounts + 1)
        for i in range(0,xCounts):
            print (xPath, i)
            if i==0:
                xP[xPath][i] = xS0
                continue
            else:
                xSt_1 = xP[xPath][i-1]
                xDeltaS = xSt_1 * (xMu * 1 / 252 + xSigma * xN[i] * np.sqrt(1/252))
                xP[xPath][i] = xSt_1 + xDeltaS
            ######### calc stats ########
            if i == (xCounts - 1):
                if xP[xPath][i] > xBucketDF['Level'][0]:  #np.max(xActual[len(xActual) - 1],
xPeakSPX):
                    xAboveNo_0 = xAboveNo_0 + 1
```

```
                    xAbove_0 = xAbove_0 + xP[xPath][i]
                    xSet_above_0.add(xPath)
                if (xP[xPath][i] < xBucketDF['Level'][0]) & (xP[xPath][i] >
xBucketDF['Level'][1]):
                    xNo_0_1 = xNo_0_1 + 1
                    x0_1 = x0_1 + xP[xPath][i]
                    xSet_0_1.add(xPath)
                # if (xP[xPath][i] < xBucketDF['Level'][1]) & (xP[xPath][i] >
xBucketDF['Level'][2]):
                #     xNo_1_2 = xNo_1_2 + 1
                #     x1_2 = x1_2 + xP[xPath][i]
                # if (xP[xPath][i] < xBucketDF['Level'][2]) & (xP[xPath][i] >
xBucketDF['Level'][3]):
                #     xNo_2_3 = xNo_2_3 + 1
                #     x2_3 = x2_3 + xP[xPath][i]
                if xP[xPath][i] < xBucketDF['Level'][1]:
                    xBelowNo_1 = xBelowNo_1 + 1
                    xBelow_1 = xBelow_1 + xP[xPath][i]
                    xSet_below_1.add(xPath)

    try:
        xAboveAvg_0 = xAbove_0 / xAboveNo_0
    except:
        {}
    try:
        xAvg_0_1 = x0_1 / xNo_0_1
    except:
        {}
    try:
        xBelowAvg_1 = xBelow_1 / xBelowNo_1
    except:
        {}

    xTotalNo = xAboveNo_0 + xNo_0_1 + xBelowNo_1

    xAboveNo_0_pct = xAboveNo_0 / xTotalNo
    xNo_0_1_pct = xNo_0_1 / xTotalNo
    xBelowNo_1_pct = xBelowNo_1 / xTotalNo

    xBucketDF['Pct'] = np.nan
    xBucketDF['Pct'][0] = xAboveNo_0_pct
    xBucketDF['Pct'][1] = xNo_0_1_pct
    #xBucketDF = xBucketDF.append({'Name': ('Below ' + xBucketDF['Name'][1]), 'Level': np.nan,
'Pct': xBelowNo_1_pct}, ignore_index=True)
    xBucketDF = xBucketDF.append({'Name': 'Below Buffer', 'Level': np.nan, 'Pct':
xBelowNo_1_pct},
                                 ignore_index=True)
    xBucketDF['Pct'] = xBucketDF['Pct'].astype(float).map("{:.1%}".format)

    ###xBucketDF[['Name','Level','Pct']].to_csv(xDir + 'xSimulations_' + (str)(xTerm) + '_' +
(str)(xI) + '.txt')
    xSubTitle = ''
    if xI == 0:
        xSubTitle = 'Stress Period: Dotcom bubbles burst (' \
                    + xStressStartDate.strftime('%m/%d/%Y') + ' - ' +
xStressEndDate.strftime('%m/%d/%Y') +')'
    elif xI == 1:
        xSubTitle = 'Stress Period: Financial crisis (' \
                    + xStressStartDate.strftime('%m/%d/%Y') + ' - ' +
xStressEndDate.strftime('%m/%d/%Y') +')'
```

```
    elif xI == 2:
        xSubTitle = 'Stress Period: COVID-19 meltdown (' \
                     + xStressStartDate.strftime('%m/%d/%Y') + ' - ' +
xStressEndDate.strftime('%m/%d/%Y') +')'
    if xTerm == 2:
        xResult_string = xResult_string + 'Simulation Results for ' +(str)(xTerm) + ' Hard
Buffer Note over ' + xSubTitle +':\n'
    else:
        xResult_string = xResult_string + 'Simulation Results for ' +(str)(xTerm) + ' Barrier
Buffer Note over ' + xSubTitle +':\n\n'
    xResult_string = xResult_string +
(str)(xBucketDF[['Name','Level','Pct']].astype('string')) \
                      + '\n\n' + 'SI expiration date: ' + xEndDate.strftime('%m/%d/%Y') +'\n\n'

    xAvgList = []
    xAvgList.append(xTroughSPX)
    xAvgList_above_0 = []
    xAvgList_above_0.append(xTroughSPX)
    xAvgList_0_1 = []
    xAvgList_0_1.append(xTroughSPX)
    xAvgList_below_1 = []
    xAvgList_below_1.append(xTroughSPX)
    for i in range(0,xCounts-1):   #1154
        xSum = 0
        xNo = 0
        xSum_above_0 = 0
        xSum_0_1 = 0
        xSum_below_1 = 0
        xNo_above_0 = 0  # these numbers are calculated already; here recalculate them for
double checking
        xNo_0_1 = 0
        xNo_below_1 = 0
        for j in range(0, xPaths): #5001
            xSum = xSum + xP[j][i]
            xNo = xNo + 1
            if j in xSet_above_0:
                if xNo_above_0 < 1:
                    xSum_above_0 = xSum_above_0 + xP[j][i]
                    xNo_above_0 = xNo_above_0 + 1
            elif j in xSet_0_1:
                if xNo_0_1 < 1:
                    xSum_0_1 = xSum_0_1 + xP[j][i]
                    xNo_0_1 = xNo_0_1 + 1
            elif j in xSet_below_1:
                if xNo_below_1 < 1:
                    xSum_below_1 = xSum_below_1 + xP[j][i]
                    xNo_below_1 = xNo_below_1 + 1

        try:
            xAvgSum = xSum / xNo
            xAvgList.append(xAvgSum)
        except:
            xAvgList.append(np.nan)
        try:
            xAvgSum_above_0 = xSum_above_0 / xNo_above_0
            xAvgList_above_0.append(xAvgSum_above_0)
        except:
            xAvgList_above_0.append(np.nan)
        try:
            xAvgSum_0_1 = xSum_0_1 / xNo_0_1
```

```
            xAvgList_0_1.append(xAvgSum_0_1)
        except:
            xAvgList_0_1.append(np.nan)
        try:
            xAvgSum_below_1 = xSum_below_1 / xNo_below_1
            xAvgList_below_1.append(xAvgSum_below_1)
        except:
            xAvgList_below_1.append(np.nan)
    #################
    import matplotlib.pyplot as plt
    import matplotlib.dates as mdates
    import matplotlib.transforms as transforms

    #plt.figure()
    fig, ax = plt.subplots()
    #############
    if False:
        #plt.locator_params(axis='x', nbins =7)
        plt.plot(xDates_axis,xP[0],label='sample path_1')
        plt.plot(xDates_axis,xP[1000],label='sample path_2')

        plt.plot(xDates_axis,xActual, color='black',label='Actual')
        plt.plot(xDates_axis,xAvgList, color='red',label='Simulated Avg')

    else:
        xActualDates = xActualDF['DATE'].to_list()
        xActualDF['NAS'] = np.nan
        xActualNAS = xActualDF['NAS'].to_list()
        xActual0 = xActualDF['SPX'].to_list()

        xDates_axis = xActualDates + xDates_axis
        xP[0] = xActualNAS + xP[0]
        xP[1000] = xActualNAS + xP[1000]
        xActual = xActual0 + xActual
        xAvgList = xActualNAS + xAvgList
        xAvgList_above_0 = xActualNAS + xAvgList_above_0
        xAvgList_0_1 = xActualNAS + xAvgList_0_1
        xAvgList_below_1 = xActualNAS + xAvgList_below_1

        xPeakLine = [xPeakSPX]*len(xDates_axis)
        xBufferLine = [xPeakSPX*(1+xBuffer)]*len(xDates_axis)
        #plt.plot(xDates_axis, xP[0], label='sample path_1')
        #plt.plot(xDates_axis, xP[1000], label='sample path_2')

        ax.plot(xDates_axis, xActual, color='black', label='Actual')
        #plt.plot(xDates_axis, xPeakLine, color='cyan', label='Peak')
        ax.axhline(y=xPeakSPX, color='cyan', linestyle='--') #, label='Peak')
        ax.axhline(y=xPeakSPX*(1+xBuffer), color='magenta', linestyle='--') #,label='Buffer')
        #plt.plot(xDates_axis, xBufferLine, color='magenta', label='Buffer')
        #plt.plot(xDates_axis, xAvgList, color='red', label='Simulated Avg')
        ax.plot(xDates_axis, xAvgList_above_0, color='red', \
                 label='Sample ' + xBucketDF['Name'][0] + '(with Prob of ' +
xBucketDF['Pct'][0]+')')
        ax.plot(xDates_axis, xAvgList_0_1, color='blue', \
                 label='Sample ' + xBucketDF['Name'][1]+ '(with Prob of ' +
xBucketDF['Pct'][1]+')')
        ax.plot(xDates_axis, xAvgList_below_1, color='orange', \
                 label='Sample ' + xBucketDF['Name'][2]+ '(with Prob of ' +
xBucketDF['Pct'][2]+')')
```

```
        trans = transforms.blended_transform_factory(
            ax.get_yticklabels()[0].get_transform(), ax.transData)
        ax.text(0, xPeakSPX, 'Peak', color="cyan", transform=trans,
                ha="right", va="center")
        ax.text(0, xPeakSPX*(1+xBuffer), 'Buffer', color="magenta",
                transform=trans, ha="right", va="center")
    plt.legend(loc='best')
    ax = plt.gca()
    #ax.xaxis.set_major_locator(mdates.YearLocator(2, month=1, day=1))
    ax.xaxis.set_major_locator(mdates.MonthLocator(interval=6))
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.ylabel('The S&P 500 Index')
    plt.gcf().autofmt_xdate()
    if xTerm == 2:
        plt.suptitle('Simulation Results for ' + (str)(xTerm) + ' Years Hard Buffer Note\n'
                + xSubTitle)
    elif xTerm in {4,6}:
        plt.suptitle('Simulation Results for ' + (str)(xTerm) + ' Years Barrier Buffer Note\n'
                + xSubTitle)
    plt.savefig(xDir + 'xSimulationResults_' + (str)(xTerm) + '_' + (str)(xI)+'.png')
    plt.show()
    print("i am done")

f_w = open(xDir + 'xSimulationResults_' + xBufferType + '_' + (str)(xTerm) + '.txt','w')
f_w.write(xResult_string)
f_w.close()
```