*### 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  
 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'**,**'NewPort\_pct\_ch\_Y'**]].copy()  
xMP.rename(columns={xY\_col+**'\_pct\_ch\_D'**:**'Current\_pct\_ch\_D'**,**'NewPort\_pct\_ch\_D'**:**'New\_pct\_ch\_D'**},inplace=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()  
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\_pct\_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\_pct\_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\_pct\_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()).round(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','HYTR\_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\_pct\_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\_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'**,**'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\_pct\_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 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\_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 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\_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+[**'Idiosyncratic'**]))]  
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\_pct\_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\_pct\_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)  
  
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'  
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[**'HFRIEMNI\_pct\_ch\_M'**]  
  
xRiskFactorSet\_M=[**'SPXT\_pct\_ch\_M'**,**'BondTR\_pct\_ch\_M'**,**'CCY\_pct\_ch\_M'**,**'COMM\_pct\_ch\_M'**,**'USCredit\_pct\_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\_pct\_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'**,**'MAXGROW\_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\_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 = [**'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']  
 #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\_ch\_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\_ch\_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'**,**'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\_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','NewPort\_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'},inplace=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\_pct\_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\_pct\_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\_pct\_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()).round(4)  
xRiskFactorCorrelations\_orthog = (xDF\_orthog[['orthog\_SPXT\_pct\_ch\_M','orthog\_USCredit\_pct\_ch\_M','orthog\_COMM\_pct\_ch\_M','TREASURY\_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\_COMM\_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\_ch\_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','HYTR\_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\_pct\_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\_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','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\_pct\_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 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\_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 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\_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+['Idiosyncratic']))]  
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','BondTR\_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','BondTR\_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\_term\_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('hererrrrrrrrr')  
# 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('hererrrrrrrrr')  
 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('hererrrrrrrrr')  
# 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\_growth',\  
 'PV1\_growth','PV2\_SPXT\_growth','PV2\_BondTR\_growth','PV2\_SI\_growth','PV2\_growth','SPXT\_100\_growth']].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)][['DATE','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\_term\_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('hererrrrrrrrr')  
# 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()