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

# 1.2: Portfolio Returns #

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

# can set stock ticker macros here

# objective: to decide on reliable etfs to invest in

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

# importing libraries #

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

import numpy as np

import pandas as pd

from pandas\_datareader import data as wb

import matplotlib.pyplot as plt

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

# importing stock data of choice: spy, qqq #

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

tickers = ['SPY', 'QQQ']

sec\_data = pd.DataFrame()

# examining behavior over ten years '07 to '17

for t in tickers:

sec\_data[t] = wb.DataReader(t, data\_source='yahoo', start='2007-1-1')['Adj Close']

# print(sec\_data.tail())

# storing logarithmic returns data in a new table

sec\_returns = np.log(sec\_data / sec\_data.shift(1))

# print(sec\_returns)

#######

# SPY #

#######

sec\_returns['SPY'].mean()

# annualizing returns

sec\_returns['SPY'].mean()\*250

# checking standard deviation

sec\_returns['SPY'].std()

# annualizing volatility

sec\_returns['SPY'].std() \* 250 \*\* 0.5

#######

# QQQ #

#######

sec\_returns['QQQ'].mean()

# annualizing returns

sec\_returns['QQQ'].mean()\*250

# checking standard deviation

sec\_returns['QQQ'].std()

# annualizing volatility

sec\_returns['QQQ'].std() \* 250 \*\* 0.5

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

# mean - volatility comparison #

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

# printing consecutive

sec\_returns['SPY'].mean()\*250

sec\_returns['QQQ'].mean()\*250

# printing returns together, adding extra bracket to increase dimension

print(sec\_returns[['SPY', 'QQQ']].mean()\*250)

# printing volatility together

print(sec\_returns[['SPY', 'QQQ']].std()\*250\*0.5)

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

# 2.1: Portfolio Risk #

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

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

# importing libraries #

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

import numpy as np

import pandas as pd

from pandas\_datareader import data as wb

import matplotlib.pyplot as plt

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

# importing stock data of choice #

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

tickers = ['QQQ', 'VOO', 'SPY']

sec\_data = pd.DataFrame()

# examining behavior over ten years '07 to '17

for t in tickers:

sec\_data[t] = wb.DataReader(t, data\_source='yahoo', start='2007-1-1')['Adj Close']

# storing logarithmic returns data in a new table

sec\_returns = np.log(sec\_data / sec\_data.shift(1))

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

# SPY: Variance #

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

SPY\_var = sec\_returns['SPY'].var()

SPY\_var

# spy variance annualized

SPY\_var\_a = sec\_returns['SPY'].var() \*250

SPY\_var\_a

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

# QQQ: Variance #

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

QQQ\_var = sec\_returns['QQQ'].var()

QQQ\_var

# qqq variance annualized

QQQ\_var\_a = sec\_returns['QQQ'].var() \*250

QQQ\_var\_a

# Covariance between 2 stocks: Cov(SPY,QQQ)

cov\_matrix = sec\_returns.cov()

cov\_matrix

# covariance annualized

cov\_matrix\_a = sec\_returns.cov() \* 250

cov\_matrix\_a

# Correlation between 2 stocks: Corr(SPY,QQQ)

# remember this is the correlation bw returns, not prices

# though returns is what we care most about

corr\_matrix = sec\_returns.corr()

print(corr\_matrix)

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

# Portfolio Risk: 3 stock example #

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

# assigning current portfolio's weights

weights = np.array([0.36, 0.31, 0.30]) # update weights as they vary

# calculating portfolio variance

pfolio\_var = np.dot(weights.T, np.dot(sec\_returns.cov() \* 250, weights))

print(pfolio\_var)

# checking portfolio's volatility

pfolio\_vol = (np.dot(weights.T, np.dot(sec\_returns.cov() \* 250, weights))) \*\* 0.5

print(pfolio\_vol)

# printing volatility as a percentage

print (str(round(pfolio\_vol, 5) \* 100) + ' %')

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

# 3.1: Efficient Frontier: Part I #

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

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

# importing libraries #

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

import numpy as np

import pandas as pd

from pandas\_datareader import data as wb

import matplotlib.pyplot as plt

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

# importing stock data of choice #

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

assets = ['SPY', 'QQQ', 'VOO']

pf\_data = pd.DataFrame()

# examining behavior over 8 years: '12 to present (voo only started in 2012,

# can go further back if need be)

for a in assets:

pf\_data[a] = wb.DataReader(a, data\_source='yahoo', start='2012-1-1')['Adj Close']

# printing closing prices

print(pf\_data.tail())

# checking dimensions

print(pf\_data.shape)

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

# plotting stock data: normalized #

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

# checking % gains from t = 0, here 2012 (normalizing data)

print(pf\_data / pf\_data.iloc[0] \* 100)

# plotting % changes (normalizing data)

(pf\_data / pf\_data.iloc[0] \* 100).plot(figsize=(10, 5))

plt.show()

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

# log returns #

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

# to obtain efficient frontier, will need log returns

log\_returns = np.log(pf\_data / pf\_data.shift(1))

# avg log returns over 8 yrs

print(log\_returns.mean() \* 250)

# covariance matrix between log returns over 8 yrs

print(log\_returns.cov() \* 250)

# correlation matrix between log returns over 8 yrs

print(log\_returns.corr())

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

# generating weights #

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

# storing number of assets in a variable

num\_assets = len(assets)

print(num\_assets)

# creating n random weights for n assets

# arr = np.random.random(3)

# print(arr)

# this following line of manual code may or may not add to 1

# arr[0] + arr[1] + arr[2]

# want weights, randomly assigned, that add to 1

weights = np.random.random(num\_assets)

weights /= np.sum(weights)

print(weights)

# checking if summation adds to 1:

print(weights[0] + weights[1] + weights[2])

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

# 3.2: Efficient Frontier: Part II #

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

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

# importing libraries #

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

import numpy as np

import pandas as pd

from pandas\_datareader import data as wb

import matplotlib.pyplot as plt

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

# importing stock data of choice #

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

assets = ['SPY', 'QQQ', 'VOO', 'IWF']

pf\_data = pd.DataFrame()

# examining behavior over 8 years: '12 to present (voo only started in 2012,

# can go further back if need be)

for a in assets:

pf\_data[a] = wb.DataReader(a, data\_source='yahoo', start='2012-1-1')['Adj Close']

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

# log returns #

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

# to obtain efficient frontier, will need log returns

log\_returns = np.log(pf\_data / pf\_data.shift(1))

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

# generating weights #

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

# storing number of assets in a variable

num\_assets = len(assets)

# want weights, randomly assigned, that add to 1

weights = np.random.random(num\_assets)

weights /= np.sum(weights)

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

# expected portfolio return #

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

print(np.sum(weights \* log\_returns.mean() \* 250))

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

# expected portfolio variance #

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

print(np.dot(weights.T, np.dot(log\_returns.cov() \* 250, weights)))

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

# expected portfolio volatility #

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

print(np.sqrt(np.dot(weights.T, np.dot(log\_returns.cov() \* 250, weights))))

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

# simulating weights #

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

pfolio\_returns = []

pfolio\_volatilities = []

# simulating weights

for x in range (1000):

weights = np.random.random(num\_assets)

weights /= np.sum(weights)

# append method helps generate and store simulations

pfolio\_returns.append(np.sum(weights \* log\_returns.mean()) \* 250)

pfolio\_volatilities.append(np.sqrt(np.dot(weights.T, np.dot(log\_returns.cov() \* 250, weights))))

# converting weights generated to a numpy array

pfolio\_returns = np.array(pfolio\_returns)

pfolio\_volatilities = np.array(pfolio\_volatilities)

print(pfolio\_returns, pfolio\_volatilities)

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

# 3.3: Efficient Frontier: Part III #

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

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

# importing libraries #

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

import numpy as np

import pandas as pd

from pandas\_datareader import data as wb

import matplotlib.pyplot as plt

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

# importing stock data of choice #

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

assets = ['SPY', 'QQQ', 'VOO', 'IWF']

pf\_data = pd.DataFrame()

# examining behavior over 8 years: '12 to present (voo only started in 2012,

# can go further back if need be)

for a in assets:

pf\_data[a] = wb.DataReader(a, data\_source='yahoo', start='2012-1-1')['Adj Close']

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

# log returns #

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

# to obtain efficient frontier, will need log returns

log\_returns = np.log(pf\_data / pf\_data.shift(1))

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

# generating weights #

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

# storing number of assets in a variable

num\_assets = len(assets)

# want weights, randomly assigned, that add to 1

weights = np.random.random(num\_assets)

weights /= np.sum(weights)

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

# simulating weights #

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

pfolio\_returns = []

pfolio\_volatilities = []

# simulating weights

for x in range (1000):

weights = np.random.random(num\_assets)

weights /= np.sum(weights)

# append method helps generate and store simulations

pfolio\_returns.append(np.sum(weights \* log\_returns.mean()) \* 250)

pfolio\_volatilities.append(np.sqrt(np.dot(weights.T, np.dot(log\_returns.cov() \* 250, weights))))

# converting weights generated to a numpy array

pfolio\_returns = np.array(pfolio\_returns)

pfolio\_volatilities = np.array(pfolio\_volatilities)

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

# simulated dataframe #

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

# assigning simulated weights to a dictionary

portfolios = pd.DataFrame({'Return': pfolio\_returns, 'Volatility': pfolio\_volatilities})

# printing dataframe head, and tail

print(portfolios.head())

print(portfolios.tail())

portfolios.plot(x='Volatility', y='Return', kind='scatter', figsize=(10,6));

plt.xlabel('Expected Volatility')

plt.ylabel('Expected Return')

plt.show()

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

# 4.1: Stock Beta's #

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

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

# importing libraries #

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

import numpy as np

import pandas as pd

from pandas\_datareader import data as wb

# import matplotlib.pyplot as plt

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

# importing stock data of choice #

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

tickers = ['SPY', 'QQQ', 'VOO', 'IWF']

data = pd.DataFrame()

# stock beta: variation of stock wrt to mkt, is calculated for 5 yrs at a time

# beta = cov (stock/mkt)/ var(mkt)

for t in tickers:

data[t] = wb.DataReader(t, data\_source='yahoo', start='2015-1-1', end='2019-12-31')['Adj Close']

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

# sec returns #

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

sec\_returns = np.log(data / data.shift(1))

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

# covariance of data #

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

cov = sec\_returns.cov() \* 250

print(cov)

# pulling cov wrt mkt for qqq

cov\_with\_market = cov.iloc[0,1]

print(cov\_with\_market)

market\_var = sec\_returns['QQQ'].var() \* 250

print(market\_var)

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

# beta of stock #

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

# stock beta measures volatility

qqq\_beta = cov\_with\_market / market\_var

print(qqq\_beta)

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

# stock: expected return #

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

# using 5% as a value for risk premium for stock

qqq\_er = 0.025 + qqq\_beta \* 0.05

print(qqq\_er)

# returned value is roi for given stock

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

# obtaining Sharpe ratio #

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

# subtracting 10 year government bonds from numerator

# denominator = annualized std deviation of stock

Sharpe = (qqq\_er - 0.025) / (sec\_returns['QQQ'].std() \* 250 \*\* 0.5)

print(Sharpe)

# sharpe ratio of qqq is roughly 21%

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

# 5.1: Predicting Gross Profit #

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

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

# importing libraries #

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

import numpy as np

import matplotlib.pyplot as plt

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

# simulation using last years revenue #

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

# revenue and std dev as variables

rev\_m = 170

rev\_stdev = 20

# number of iterations

iterations =1000

# generating random normal distribution

rev = np.random.normal(rev\_m, rev\_stdev, iterations)

rev

# plotting our revenue simulations

plt.figure(figsize=(15,6))

plt.plot(rev)

plt.show()

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

# cogs calculations #

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

# since cogs is money spent, we make it a negative value

# setting roughly 60% of the revenue to cogs

COGS = - (rev \* np.random.normal(0.6,0.1))

plt.figure(figsize=(15,6))

plt.plot(COGS)

plt.show()

COGS.mean()

COGS.std()

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

# 5.1: Predicting Gross Profit, pt. II #

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

# computing gross profit: revenue - cogs

Gross\_Profit = rev + COGS

Gross\_Profit

plt.figure(figsize=(15,6))

plt.plot(Gross\_Profit)

plt.show()

print(max(Gross\_Profit))

print(min(Gross\_Profit))

print(Gross\_Profit.mean())

print(Gross\_Profit.std())

# plotting the simulation, 1: with cuts

plt.figure(figsize=(10,6));

plt.hist(Gross\_Profit, bins = [40, 50, 60, 70, 90, 100, 110, 120]);

plt.show()

# plotting the simulation, 2: with bins assigned

plt.figure(figsize=(10,6));

plt.hist(Gross\_Profit, bins = 20);

plt.show()

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

# 5.2: Predicting Stock Prices #

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

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

# importing libraries #

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

import numpy as np

import pandas as pd

from pandas\_datareader import data as wb

import matplotlib.pyplot as plt

from scipy.stats import norm

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

# Importing and Storing Stock Prices #

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

ticker = 'SPY'

data = pd.DataFrame()

data[ticker] = wb.DataReader(ticker, data\_source='yahoo', start='2007-1-1')['Adj Close']

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

# Plotting Historical Data: Past 10 yrs #

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

# estimating historical log returns over past 10 yrs

log\_returns = np.log(1 + data.pct\_change())

print(log\_returns.tail())

# plotting SPY's price, past 10 yrs

data.plot(figsize=(10,6));

plt.show()

# plotting log returns, past 10 yrs

log\_returns.plot(figsize=(10,6));

plt.show()

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

# Preparing for Brownian Motion #

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

# calculating mean

u = log\_returns.mean()

print(u)

# calculating variance

var = log\_returns.var()

print(var)

# not annualizing, predicting daily instead

# calculating 'drift' from mean and var

drift = u - (0.5 \* var)

print(drift)

# std dev for brownian motion

stdev = log\_returns.std()

print(stdev)

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

# Creating Random Simulated Matrix Arrays #

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

# all withing 95% confidence interval

print(type(drift))

print(type(drift))

np.array(drift)

print(drift.values)

print(stdev.values)

# checking width in std devs. of 95% conf interval

norm.ppf(0.95)

# generating 10 x 2 matrix for arrays

x = np.random.rand(10, 2)

norm.ppf(x)

# matrix of values showing dist from mean

Z = norm.ppf(np.random.rand(10,2))

Z

# upcoming 1000 days

t\_intervals = 1000

# 10 simulations

iterations = 10

# stock price prediction formula

daily\_returns = np.exp(drift.values + stdev.values \* norm.ppf(np.random.rand(t\_intervals, iterations)))

# matrix containing daily returns

print(daily\_returns)

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

# Predicting a Daily Stock Price #

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

# creating a price list, using 1st stock price

S0 = data.iloc[-1]

print(S0)

price\_list = np.zeros\_like(daily\_returns)

print(price\_list)

# replacing daily stock price - with zeros - then simulations

# simulating row 1

price\_list[0] = S0

print(price\_list)

# completing price list and verifying

for t in range(1, t\_intervals):

price\_list[t] = price\_list[t-1] \* daily\_returns[t]

print(price\_list)

# plotting 10 simulations of SPY stock price

plt.figure(figsize=(10,6))

plt.plot(price\_list);

plt.show()