```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 21 18:29:41 2018
@author: Group F
"""

from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
from scipy import stats
import math
from datetime import datetime
import matplotlib.mlab as mlab

# Task 1.1
Oct18 = pd.read_csv('DJIA_components.csv')
TotalPrice = Oct18[' Price:D-1 '].sum()
divisor = 0.14748071991788

print(Oct18.iloc[:, 0:3])
print(TotalPrice/divisor)

# Task 1.3
DJI = pd.read_csv('DJI.csv')
GSPC = pd.read_csv('GSPC.csv')

DJI_close = DJI['Close']
GSPC_close = GSPC['Close']

Dates = []

Dates=[]
for i in list(DJI['Date']):
    Dates.append(datetime.strptime(i,'%Y-%m-%d'))

fig, ax1 = plt.subplots(figsize=(8,6))
ax2 = ax1.twinx()

ax1.plot(Dates, DJI_close, color='b', label='Dow Jones Industrial
Average',linewidth=0.5)
ax1.set_xlabel('Year')
ax1.set_ylabel('Dow Jones Ind Avg Price')

ax2.plot(Dates, GSPC_close, color='r', label='S&P 500',linewidth=0.5)
ax2.set_ylabel('S&P 500 Price')

fig.tight_layout()
fig.legend(loc=(0.16, 0.82))
plt.grid()
plt.savefig('time_series.png', dpi=900)
plt.show()


# Task 1.4, 1.5
# Dow Jones Index
DJI_close = np.array(DJI_close)
DJI_close1 = DJI_close[:-1]
DJI_close2 = DJI_close[1:]

DJIret = DJI_close2/DJI_close1
```

```
DJIlogret = np.log(DJIret)

plt.figure(figsize=(8,6))
plt.plot(DJIlogret, label='Dow Jones Ind Avg Log Returns',color='b')
plt.xlabel('Period (t)')
plt.ylabel('Log Return (%)')
plt.legend(loc='upper left')
plt.axis([0,8500,-0.28,0.28])
plt.grid(True)
plt.savefig('DJI_logret.png', dpi=900)
plt.show()

# S&P 500
GSPC_close = np.array(GSPC_close)
GSPC_close1 = GSPC_close[:-1]
GSPC_close2 = GSPC_close[1:]

GSPCret = GSPC_close2/GSPC_close1
GSPClogret = np.log(GSPCret)

plt.figure(figsize=(8,6))
plt.plot(GSPClogret, label='S&P 500 Log Returns',color='r')
plt.xlabel('Period (t)')
plt.ylabel('Log Return (%)')
plt.legend(loc='upper left')
plt.axis([0,8500,-0.28,0.28])
plt.grid(True)
plt.savefig('SP_logret.png', dpi=900)
plt.show()

# Task 1.6
DJImean = DJIlogret.mean()
GSPCmean = GSPClogret.mean()
print('------------------------------------------------------------------------------
')
print('Dow Jones Ind Avg Mean Daily Log Return: ' +str(DJImean))
print('S&P 500 Mean Log Daily Return: ' +str(GSPCmean))


DJIvar = np.var(DJIlogret, ddof=1)
GSPCvar = np.var(GSPClogret, ddof=1)
print('------------------------------------------------------------------------------
')
print('Dow Jones Ind Avg Sample Variance of Returns: ' +str(DJIvar))
print('S&P 500 Sample Variance of Returns: ' +str(GSPCvar))

# Task 1.7
DJIAmean = 252*DJImean
GSPCAmean = 252*GSPCmean


DJIAstd = math.sqrt(252*DJIvar)
GSPCAstd = math.sqrt(252*GSPCvar)

print('------------------------------------------------------------------------------
')
print('Dow Jones Ind Avg Annualised Return: ' + str(round(DJIAmean,8)))
print('S&P 500 Annualised Return: ' + str(round(GSPCAmean,8)))
print('------------------------------------------------------------------------------
')
```

```
print('Dow Jones Ind Avg Annualised Volatility: ' + str(round(DJIAstd,8)))
print('S&P 500 Annualised Volatility: ' + str(round(GSPCAstd,8)))


# Task 1.8

def Skewness(ret_list):
    mean_ret=np.mean(ret_list)
    std_ret=np.std(ret_list,ddof=0)
    skew_num_list=[((i-(mean_ret))**3) for i in ret_list]
    return np.sum(skew_num_list)/(len(ret_list)*(std_ret**3))

def Kurtosis(ret_list):
    mean_ret=np.mean(ret_list)
    std_ret=np.std(ret_list,ddof=0)
    kurt_num_list=[((i-(mean_ret))**4) for i in ret_list]
    return np.sum(kurt_num_list)/(len(ret_list)*(std_ret**4))

DJI_Skew=Skewness(DJIlogret)
GSPC_Skew=Skewness(GSPClogret)

DJI_Kurt=Kurtosis(DJIlogret)
GSPC_Kurt=Kurtosis(GSPClogret)

print('--------------------------------------------------------------------------------
')
print('Skewness of Dow Jones Ind Avg\'s Returns: ' + str(round(DJI_Skew,8)))
print('Skewness of S&P 500\'s Returns: ' + str(round(GSPC_Skew,8)))
print('--------------------------------------------------------------------------------
')
print('Kurtosis of Dow Jones Ind Avg\'s Returns: ' + str(round(DJI_Kurt,8)))
print('Kurtosis of S&P 500\'s Returns: ' + str(round(GSPC_Kurt,8)))

# Additional chart
# frequency distribution histogram overlay with normal with same mean and variance
plt.figure(figsize=(8,6))
plt.hist([i*100 for i in DJIlogret],bins=1000,density=True,label='Dist. of Dow Jones
Ind Avg Log Returns',color='b')

DJI_NormMean = np.mean([i*100 for i in DJIlogret])
DJI_NormSigma = np.std([i*100 for i in DJIlogret],ddof=1)
DJI_NormSpace = np.linspace(DJI_NormMean - 6*DJI_NormSigma, DJI_NormMean +
6*DJI_NormSigma, 8500)

plt.plot(DJI_NormSpace,mlab.normpdf(DJI_NormSpace, DJI_NormMean,
DJI_NormSigma),label='Normal Dist with Same Mean and Std Dev',color='k')
plt.xlabel('Log Returns (%)')
plt.ylabel('Probability')
plt.legend(loc='upper left')
plt.grid(True)
plt.axis([-30,15,0,0.8])
plt.savefig('DJInorm.png', dpi=900)
plt.show()

plt.figure(figsize=(8,6))
plt.hist([i*100 for i in GSPClogret],bins=1000,density=True,label='Dist. of S&P 500
Log Returns',color='r')

GSPC_NormMean = np.mean([i*100 for i in GSPClogret])
GSPC_NormSigma = np.std([i*100 for i in GSPClogret],ddof=1)
```

```
GSPC_NormSpace = np.linspace(GSPC_NormMean - 6*GSPC_NormSigma, GSPC_NormMean +
6*GSPC_NormSigma, 8500)

plt.plot(GSPC_NormSpace,mlab.normpdf(GSPC_NormSpace, GSPC_NormMean,
GSPC_NormSigma),label='Normal Dist with Same Mean and Std Dev',color='k')
plt.xlabel('Log Returns (%)')
plt.ylabel('Probability')
plt.legend(loc='upper left')
plt.grid(True)
plt.axis([-25,15,0,0.9])
plt.savefig('GSPCnorm.png', dpi=900)
plt.show()

# Task 1.9
def Jarque_Beta(ret_list):
    JB = len(ret_list)*((Skewness(ret_list)**2)/6 + ((Kurtosis(ret_list)-3)**2)/24)
    return JB

DJI_JBstat = ('Dow Jones Ind Avg', float(Jarque_Beta(DJIlogret)))
GSPC_JBstat = ('S&P500', float(Jarque_Beta(GSPClogret)))
Chisquare_value = stats.chi2.ppf(1-0.05,2)

print('-------------------------------------------------------------------------------
')

for (i,j) in [DJI_JBstat, GSPC_JBstat]:
    print(i+" Jarque-Beta Statistics : "+str(round(j,4))+". critical value:
"+str(round(Chisquare_value,4)))
    if j>Chisquare_value:
        print(i+" Jarque-Beta Statistics exceeds critical value. \
              \n - Reject null hypothesis, JB not equal to 0.\
              \n - Data does not fit a normal distribution at 95% confidence")
    else:
        print(i+" Jarque-Beta Statistics does not exceed critical valu. \
              \n - Do not reject null hypothesis, JB equal to 0.\
              \n - Data does fits a normal distribution at 95% confidence")
    print("\n")

# Additional metrics
# n-day rolling correlation // drc
drc = 120
F_Rho = lambda i: stats.pearsonr(DJIlogret[i-drc:i], GSPClogret[i-drc:i])
Correl = [F_Rho(i)[0] for i in range(drc,len(DJI_close))]
plt.figure(figsize=(8,6))
plt.plot(Dates[drc:len(DJI_close)],Correl, label='252-day Rolling
Correlation',color='m')
plt.xlabel('Period')
plt.ylabel('Correlation Coefficient')
plt.legend(loc='lower left')
plt.grid(True)
plt.savefig('Correlation.png', dpi=900)
plt.show()


# Task 2.1
IndexCorr=np.corrcoef(DJIlogret,GSPClogret)[1,0]

# Task 2.2
def tStat_MeanDiff(ret_list1,ret_list2):
    mean_ret1=np.mean(ret_list1)
```

```
    mean_ret2=np.mean(ret_list2)
    var_ret1=np.var(ret_list1,ddof=1)
    var_ret2=np.var(ret_list2,ddof=1)
    n1=len(ret_list1)
    n2=len(ret_list2)
    return ((mean_ret1-mean_ret2)/
            (np.sqrt(var_ret1/n1+var_ret2/n2)))

def tStat_MeanDiff_DF(ret_list1,ret_list2):
    var_ret1=np.var(ret_list1,ddof=1)
    var_ret2=np.var(ret_list2,ddof=1)
    n1=len(ret_list1)
    n2=len(ret_list2)
    return (((var_ret1/n1+var_ret2/n2)**2)/
            ((((var_ret1/n1)**2)/(n1-1))+((var_ret2/n2)**2)/(n2-1)))

DJI_GPSC_MeanDiff_tStat=tStat_MeanDiff(DJIlogret,GSPClogret)
DJI_GPSC_MeanDiff_DF=tStat_MeanDiff_DF(DJIlogret,GSPClogret)
DJI_GPSC_MeanDiff_tStat_abs = abs(DJI_GPSC_MeanDiff_tStat)

Tstats_R = stats.t.isf(0.025, df=DJI_GPSC_MeanDiff_DF)
Tstats_L = stats.t.isf(1-0.025, df=DJI_GPSC_MeanDiff_DF)

print('------------------------------------------------------------------------------
')
print('t Statistic: '+str(round(DJI_GPSC_MeanDiff_tStat,6)))
print('')
print('Critical t Values: +/-'+ str(round(stats.t.ppf(1-0.025,DJI_GPSC_MeanDiff_DF),
6)))
print('')

if DJI_GPSC_MeanDiff_tStat_abs > Tstats_R:
    print("Test statistic (abs) above critical T value. DJIA mean log return is not
equal to SP500 mean log return.\n")
    if DJI_GPSC_MeanDiff_tStat > Tstats_R:
        print("Test statistic above critical T value. DJIA mean log return is bigger
than SP500 mean log return.")
    elif DJI_GPSC_MeanDiff_tStat < Tstats_L:
        print("Test statistic below critical T value. DJIA mean log return is smaller
than SP500 mean log return.")
elif DJI_GPSC_MeanDiff_tStat_abs <= Tstats_R:
    print("Test statistic (abs) below critical T value. DJIA mean log return is equal
to SP500 mean log return.\n")

# Task 2.3
def F_Stat(ret_list1,ret_list2):
    var_ret1=np.var(ret_list1,ddof=1)
    var_ret2=np.var(ret_list2,ddof=1)
    return max(var_ret1/var_ret2,var_ret2/var_ret1)

def F_Stat_DFs(ret_list1,ret_list2):
    var_ret1=np.var(ret_list1,ddof=1)
    var_ret2=np.var(ret_list2,ddof=1)
    n1=len(ret_list1)
    n2=len(ret_list2)
    if var_ret1>var_ret2:
        return [n1-1,n2-1]
    elif var_ret1<var_ret2:
        return [n2-1,n1-1]
    else:
```

```
        print('ERROR')

DJI_GPSC_FStat=F_Stat(DJIlogret,GSPClogret)
DJI_GPSC_FStat_DFs=F_Stat_DFs(DJIlogret,GSPClogret)

Fstats_R = stats.f.isf(0.15/2, DJI_GPSC_FStat_DFs[0], DJI_GPSC_FStat_DFs[1])
Fstats_L = stats.f.isf(1-0.15/2, DJI_GPSC_FStat_DFs[0], DJI_GPSC_FStat_DFs[1])

print('-------------------------------------------------------------------------------
')
print('F Statistic: '+str(round(DJI_GPSC_FStat,6)))
print('')
print('Critical F Values: '+
str(round(stats.f.ppf(0.05/2,DJI_GPSC_FStat_DFs[0],DJI_GPSC_FStat_DFs[1]),6))+', '+
                              str(round(stats.f.ppf(1-0.05/2,DJI_GPSC_FStat_DFs[0],DJI
_GPSC_FStat_DFs[1]),6)))
print('')

if Fstats_L < DJI_GPSC_FStat <Fstats_R:
    print("Test statistic not beyond critical F values. DJIA log return sample
variance is equal to SP500 log return sample variance.")
else:
    print("Test statistic beyond critical F values. DJIA log return sample variance
is not equal to SP500 log return sample variance.")
```