Appendix A: Log Return and Volatility

```
import csv
import datetime
import math
from nlib import *

#Get the mu, sigma, log returns, historical stock price for each firm from combined csv. file
def Port_return(filename = 'combined_stocks.csv',
             start_date = datetime.datetime(2013,5,28),
             end_date = datetime.datetime(2018,5,25)):
    now = datetime.datetime.now()
    history = []
    log_portfolio = []
    log_aapl=[]
    log_goog=[]
    log_msft=[]
    log_amzn=[]
    log_fb=[]
    with open(filename) as myfile:
        reader = csv.reader(myfile)
        header = reader.next()
        rows = [dict(zip(header,row)) for row in reader]
        rows.reverse()
        for k,row in enumerate(rows):
            date = datetime.datetime.strptime(row['Date'], '%m/%d/%Y')
            close1, close2, close3, close4, close5 =
            float(row['Adj_appl']),float(row['Adj_goog']),float(row['Adj_msft']),float(row['Adj_amzn']),
            float(row['Adj_fb'])

            if k>1:
                log_return1,log_return2,log_return3,log_return4,log_return5 = math.log(close1/previous1),
                math.log(close2/previous2),math.log(close3/previous3),math.log(close4/previous4),math.log(
                close5/previous5)
            else:
                log_return1,log_return2,log_return3,log_return4,log_return5 = 0.0,0.0,0.0,0.0,0.0
            if start_date < date < end_date:
                history.append([(date-start_date).days,close1, close2, close3, close4, close5])
                log_portfolio.append([log_return1,log_return2,log_return3,log_return4,log_return5])
                log_aapl.append(log_return1)
                log_goog.append(log_return2)
                log_msft.append(log_return3)
                log_amzn.append(log_return4)
                log_fb.append(log_return5)
            previous1,previous2,previous3,previous4,previous5 = close1, close2, close3, close4, close5

    mu1,mu2,mu3,mu4,mu5 = mean(log_aapl),mean(log_goog),mean(log_msft),mean(log_amzn),mean(log_fb)
    sigma1,sigma2,sigma3,sigma4,sigma5 = sd(log_aapl),sd(log_goog),sd(log_msft),sd(log_amzn),sd(log_fb)

    return  mu1,mu2,mu3,mu4,mu5,sigma1,sigma2,sigma3,sigma4,sigma5, log_portfolio, history, log_aapl,
    log_goog,log_msft, log_amzn, log_fb


#Get the mu, sigma, log returns, current stock price for each firm
mu1,mu2,mu3,mu4,mu5,sigma1,sigma2,sigma3,sigma4,sigma5, log_portfolio, hist_port, log_aapl,
log_goog,log_msft, log_amzn, log_fb = Port_return()
current_price=hist_port[-1]
current_day,S_aapl1,S_goog1,S_msft1,S_amzn1, S_fb1 = current_price


#Calculate annual log returns and volatility for 5 stocks
T =252
def year_ret_std(log_returns=log_aapl,T=T):
    yearly_return = mean(log_returns)*T
    yearly_std = sd(log_returns)*math.sqrt(T)
    return yearly_return,yearly_std
```

Appendix B: Portfolio Value and 5% VAR

```
#Get the number of stocks invested 1M for each firm
Inv=1000000.0
numS1,numS2,numS3,numS4,numS5= Inv/S_aapl1, Inv/S_goog1, Inv/S_msft1,Inv/S_amzn1 ,Inv/S_fb1
T =252

#Monte Carlo Simulation for stock prices in one year, then return portfolio value
def Monte_once():
        future = []
        P_aapl,P_goog,P_msft,P_amzn, P_fb = S_aapl1,S_goog1,S_msft1,S_amzn1, S_fb1
        for k in range(1,T+1):
            R_aapl =  random.gauss(mu1,sigma1)
            R_goog =  random.gauss(mu2,sigma2)
            R_msft =  random.gauss(mu3,sigma3)
            R_amzn =  random.gauss(mu4,sigma4)
            R_fb =  random.gauss(mu5,sigma5)

            P_aapl = P_aapl*exp(R_aapl)
            P_goog = P_goog*exp(R_goog)
            P_msft = P_msft*exp(R_msft)
            P_amzn = P_amzn*exp(R_amzn)
            P_fb = P_fb*exp(R_fb)
            future.append([ P_aapl,P_goog,P_msft,P_amzn, P_fb])
        F_appl, F_goog, F_msft, F_amzn,F_fb =future[-1]
        Port_value = numS1*F_appl + numS2*F_goog + numS3*F_msft + numS4*F_amzn + numS5*F_fb
        return Port_value

def Monte_many(ap=0.1, rp=0.1, ns=1000):
        results = []
        s1=s2=0.0
        convergence=False
        for k in xrange(1,ns):
            x = Monte_once()
            results.append(x)
            s1 += x
            s2 += x*x
            mu = float(s1)/k
            variance = float(s2)/k-mu*mu
            dmu = sqrt(variance/k)
            if k>10:
                if abs(dmu)<max(ap,abs(mu)*rp):
                    converence = True
                    break
        results.sort()
        return bootstrap(results)


# 5% VAR of portfolio value in one year (1000 simulations)
def VAR_monte():
    scenarios = []
    for k in range(1000):
        y = Monte_once()
        scenarios.append(y)
        scenarios.sort()
    print '5% VaR =',scenarios[int(1000*0.05)]
    #Canvas().hist(scenarios).save('scenariosMonte1000.png')
```

Appendix C: Call Option Price

```
class Option_Value(MCEngine):
    def __init__(self, parameters):
        self.parameters = parameters

# Monte Carlo
    def simulate_once(self):
            future = []
            P_aapl,P_goog,P_msft,P_amzn, P_fb = S_aapl1,S_goog1,S_msft1,S_amzn1, S_fb1
            T = self.parameters['expiration']
            for k in range(1,T+1):
                R_aapl =  random.gauss(self.parameters['mu1'],self.parameters['sigma1'])
                R_goog =  random.gauss(self.parameters['mu2'],self.parameters['sigma2'])
                R_msft =  random.gauss(self.parameters['mu3'],self.parameters['sigma3'])
                R_amzn =  random.gauss(self.parameters['mu4'],self.parameters['sigma4'])
                R_fb =  random.gauss(self.parameters['mu5'],self.parameters['sigma5'])

                P_aapl = P_aapl*exp(R_aapl)
                P_goog = P_goog*exp(R_goog)
                P_msft = P_msft*exp(R_msft)
                P_amzn = P_amzn*exp(R_amzn)
                P_fb = P_fb*exp(R_fb)
                future.append([ P_aapl,P_goog,P_msft,P_amzn, P_fb])
            discount = exp(-self.parameters['Rfree']/365*T)
            return self.payoff(future) * discount

# Resampling
    def simulate_once(self):
            future = []
            P_aapl,P_goog,P_msft,P_amzn, P_fb = S_aapl1,S_goog1,S_msft1,S_amzn1, S_fb1
            T = self.parameters['expiration']
            for t in range (1,T+1):
                Rt= random.choice(log_portfolio)
                R_aapl=Rt[0]
                R_goog = Rt[1]
                R_msft = Rt[2]
                R_amzn = Rt[3]
                R_fb   = Rt[4]
                P_aapl = P_aapl*exp(R_aapl)
                P_goog = P_goog*exp(R_goog)
                P_msft = P_msft*exp(R_msft)
                P_amzn = P_amzn*exp(R_amzn)
                P_fb   = P_fb*exp(R_fb)
                future.append([ P_aapl,P_goog,P_msft,P_amzn, P_fb])
            discount = exp(-self.parameters['Rfree']/365*T)
            return self.payoff(future) * discount

 # Option Payoff and conditions (payoff =1 m if 4 of 5 stocks below current value, S)
    def payoff(self,future):
        F_appl, F_goog, F_msft, F_amzn,F_fb =future[-1]
        if F_appl<S_aapl1 and F_goog<S_goog1 and F_msft<S_msft1 and F_amzn<S_amzn1 and F_fb>S_fb1:
            return 1000000.0
        elif F_appl<S_aapl1 and F_goog<S_goog1 and F_msft<S_msft1 and F_amzn>S_amzn1 and F_fb<S_fb1:
            return 1000000.0
        elif F_appl<S_aapl1 and F_goog<S_goog1 and F_msft>S_msft1 and F_amzn<S_amzn1 and F_fb<S_fb1:
            return 1000000.0
        elif F_appl<S_aapl1 and F_goog>S_goog1 and F_msft<S_msft1 and F_amzn<S_amzn1 and F_fb<S_fb1:
            return 1000000.0
        elif F_appl>S_aapl1 and F_goog<S_goog1 and F_msft<S_msft1 and F_amzn<S_amzn1 and F_fb<S_fb1:
            return 1000000.0
        else:
            return 0.0
pricer =
Option_Value({'mu1':mu1,'mu2':mu2,'mu3':mu3,'mu4':mu4,'mu5':mu5,'sigma1':sigma1,'sigma2':sigma2,'sigma3':s
igma3,'sigma4':sigma4,'sigma5':sigma5,'expiration':90,'Rfree':0.026})
print pricer.simulate_many(ap=0.1, rp=0.1, ns=1000)
```