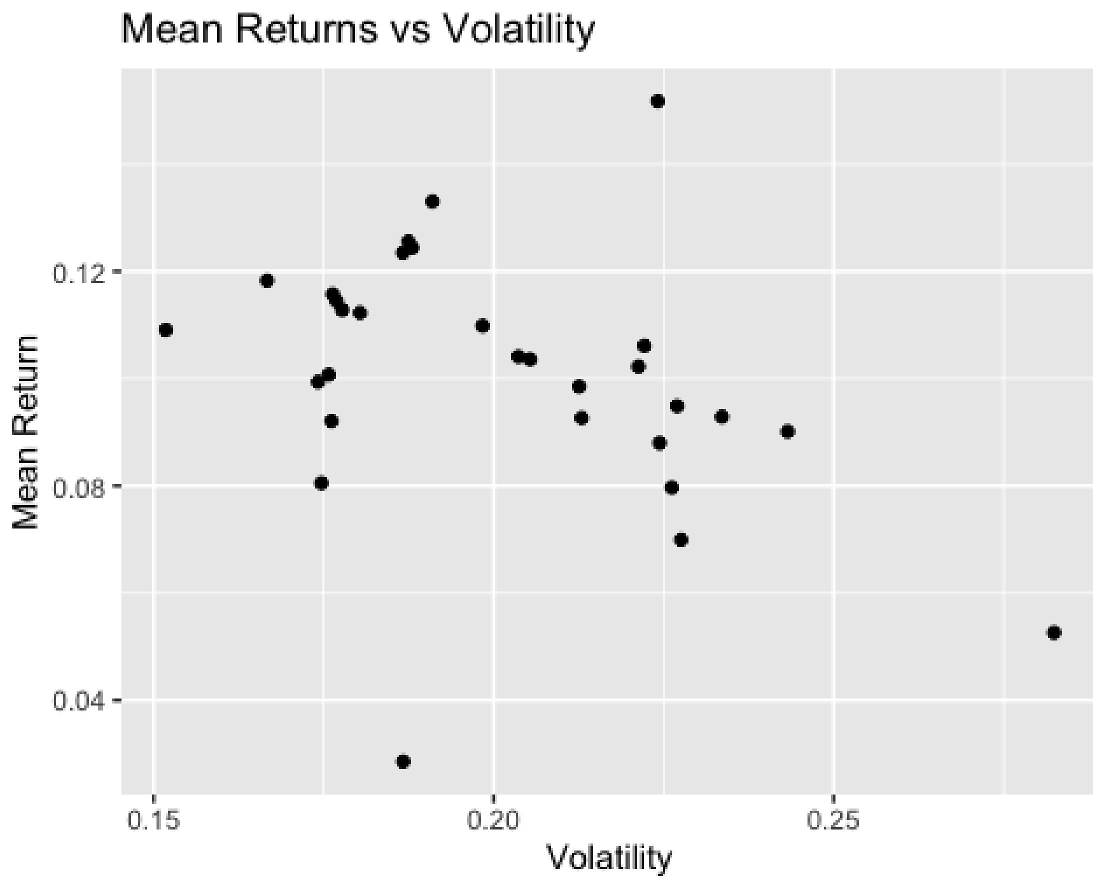
Project 1

Atharva Rane and Mrinal Gupta

2024-04-22

```
library(quantmod)

library(ggplot2)

# Exercise 1

# Task 1

symbols <- c("IVV", "IJH", "IWF", "IJR", "IWM", "IWD", "IVW", "IWB",
             "IVE", "IUSG", "IUSV","IWV", "IYW", "IWN", "IWO",
             "IJK", "IJJ", "IJS", "IJT", "IYH", "IYF", "IYY", "IYK", "IYE",
             "IYG", "IYJ", "IDU", "IYC", "IYM", "IYZ")


summary_stats <- function(x) {
  return(c(mean = mean(x), Q1 = quantile(x, 0.25), median = median(x), Q3 =
quantile(x, 0.75)))
}

mean_summary <- summary_stats(Mu)
volatility_summary <- summary_stats(sigma)
SR_summary <- summary_stats(Sr)

# Combine the summaries into a single data frame
summary_table <- data.frame(mean_summary, volatility_summary, SR_summary)
colnames(summary_table) <- c("Mean_Return", "Volatility", "Sharpe Ratio")
summary_table <- t(summary_table)

summary_table <- round(summary_table, 4)
print(summary_table)

##                mean Q1.25% median Q3.75%
## Mean_Return  0.1009 0.0922 0.1029 0.1140
## Volatility   0.2012 0.1771 0.1947 0.2237
## Sharpe Ratio 0.5156 0.4222 0.5169 0.6540

# Task 2
```

```
plot <- ggplot(results, aes(x=Volatility, y=Mean_Return)) +
  geom_point() +
  labs(title="Mean Returns vs Volatility", x="Volatility", y="Mean Return")

print(plot)
```



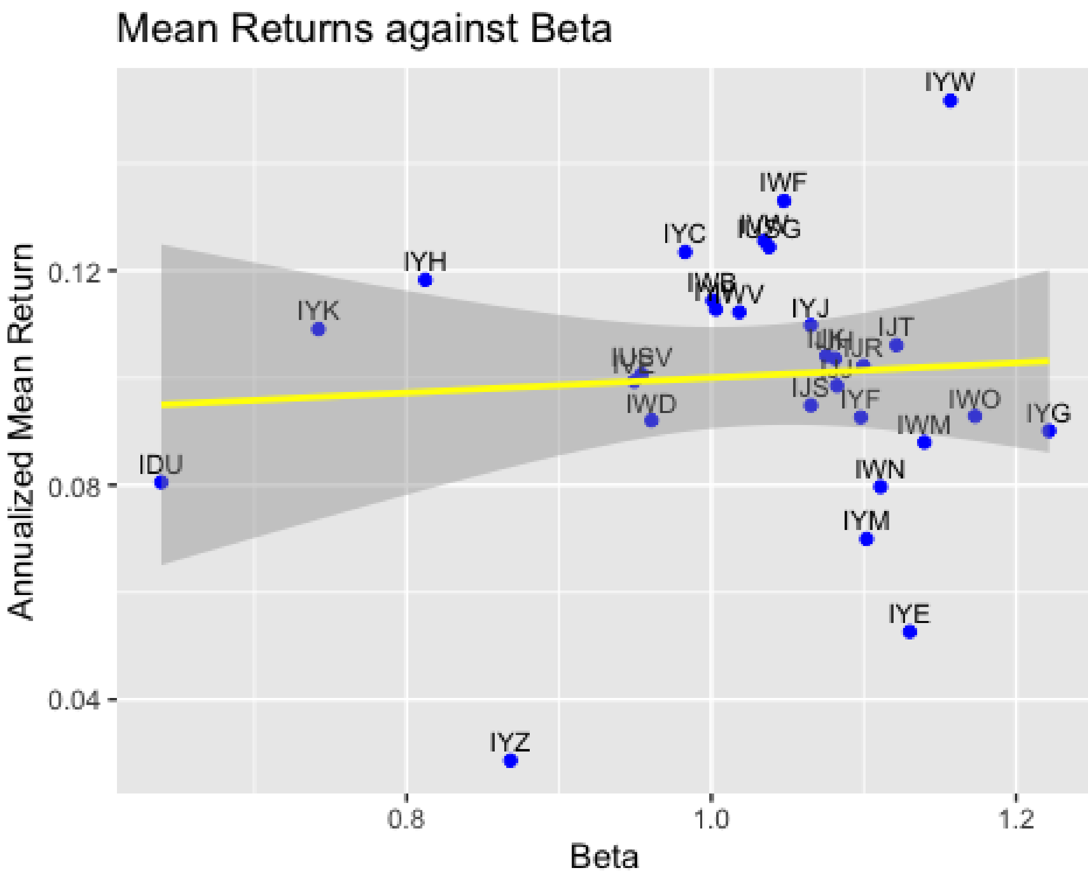

```
# Task 3

Market_IVV <- log_ret[,1]
ETFs_all <- log_ret[,-1]

Beta <- sapply(ETFs_all,FUN=function(x) cov(x,Market_IVV)/var(Market_IVV))

jensen_alpha <- apply(ETFs_all,2,FUN=function(x) mean(x)*252)-
Beta*mean(Market_IVV)*252

Treynor_ratio <- sapply(ETFs_all,FUN=function(x) mean(x)*252)/Beta

Track.Error <- sapply(ETFs_all,FUN=function(x) sqrt(var(x-
Market_IVV)*sqrt(252)))

Info_ratio <- (sapply(ETFs_all,FUN=function(x) mean(x)*252)-
mean(Market_IVV)*252) /Track.Error
```

```r
results_R <-
data.frame(cbind(Beta,jensen_alpha,Treynor_ratio,Track.Error,Info_ratio))
colnames(results_R) <- c("Beta", "Jensen_Alpha",
"Treynor_Ratio","Tracking_Error","Information_Ratio")

summary_stats_ratios <- function(x) {
  return(c(mean = mean(x), Q1 = quantile(x, 0.25), median = median(x), Q3 =
quantile(x, 0.75)))
}
summary_table_ratios <- t(summary_table_ratios)


summary_table_ratios <- round(summary_table_ratios, 4)


print(summary_table_ratios)

##                              mean  Q1.25%  median  Q3.75%
## jensen_alpha_summary   -0.0184 -0.0344 -0.0191  0.0043
## Beta_summary            1.0267  0.9828  1.0652  1.1018
## Treynor_ratio_summary   0.0993  0.0844  0.0968  0.1199
## Tracking_Error_summary  0.0212  0.0124  0.0227  0.0275
## Info_ratio_summary     -0.5938 -1.0751 -0.6624 -0.3628

# Relative Performance
max(Treynor_ratio)

## [1] 0.1469901

min(Treynor_ratio)

## [1] 0.03295736

max(Track.Error)

## [1] 0.05055439

min(Track.Error)

## [1] 0.003192108

max(Info_ratio)

## [1] 1.479608

min(Info_ratio)

## [1] -3.173363
```

(b) Looking at the perfrmance meterics we can evaluate that infact IYK comes out to be a better performing ETF where as IYZ is at the bottom of the results of performance metrics.

(c) The expense ratio of an ETF is the measure of the annual costs incured for investing in that specific ETF. Understanding with an example we say the ETF - IYK has an expense ratio is 0.33 this is the cost the investor is already paying after investing in IYK. While we look at an ETF's expense ratio plays an important role as it can affect the perofmance of the ETF.

```
# Task 4

data <- data.frame(Return = Mu[-1], Beta, Symbols = symbols[-1])

#Scatter plot
plot_1 <- ggplot(data = data, aes(x=Beta, y=Return))+
  geom_point(color = "blue") +
  labs(x = "Beta" , y = "Annualized Mean Return", title = "Mean Returns
against Beta")+
  geom_text(aes(label = Symbols), hjust = 0.5, vjust = -0.5, size = 3)+
  geom_smooth(method = "lm", color = "yellow")

plot(plot_1)

## `geom_smooth()` using formula = 'y ~ x'
```



(b)

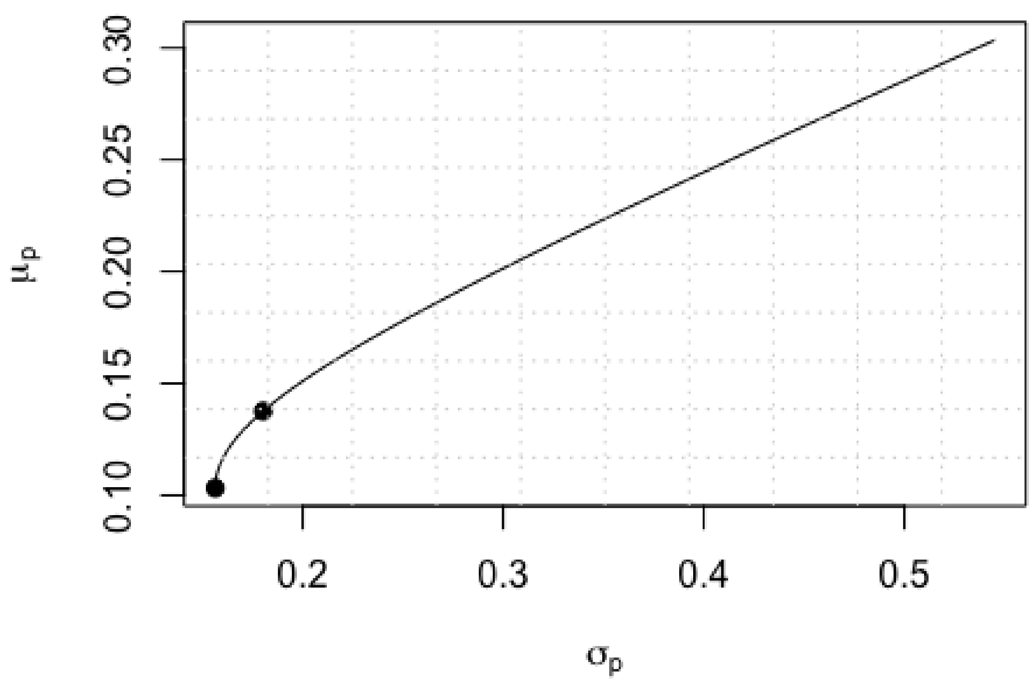
Observing the above graph it proves that CAPM doesnt holds quiet perfectly the reason may be that every assets have varrying mean returns and betas so that we a the regression line doesnt coincide with each perfectly.

```
# Exercise 2

# ETFs IVV, IYW and IYF
# Task 1

symbols <- c("IVV", "IYW", "IYF")
log_rets <- log_ret[,symbols]

Mu_2 <- Mu[symbols]
sigma_2 <- sigma[symbols]
Sr_2 <- Mu_2/sigma_2
result_2 <- data.frame(cbind(Mu_2,sigma_2,Sr_2))
colnames(result_2) <- c("Mean_ret", "Volatitliy", "SR")
round(result_2,4)

##      Mean_ret Volatitliy     SR
## IVV    0.1157     0.1763 0.6564
## IYW    0.1517     0.2242 0.6769
## IYF    0.0926     0.2129 0.4351
```

```
A_seq<-1/A_seq
sample(A_seq,5)
```

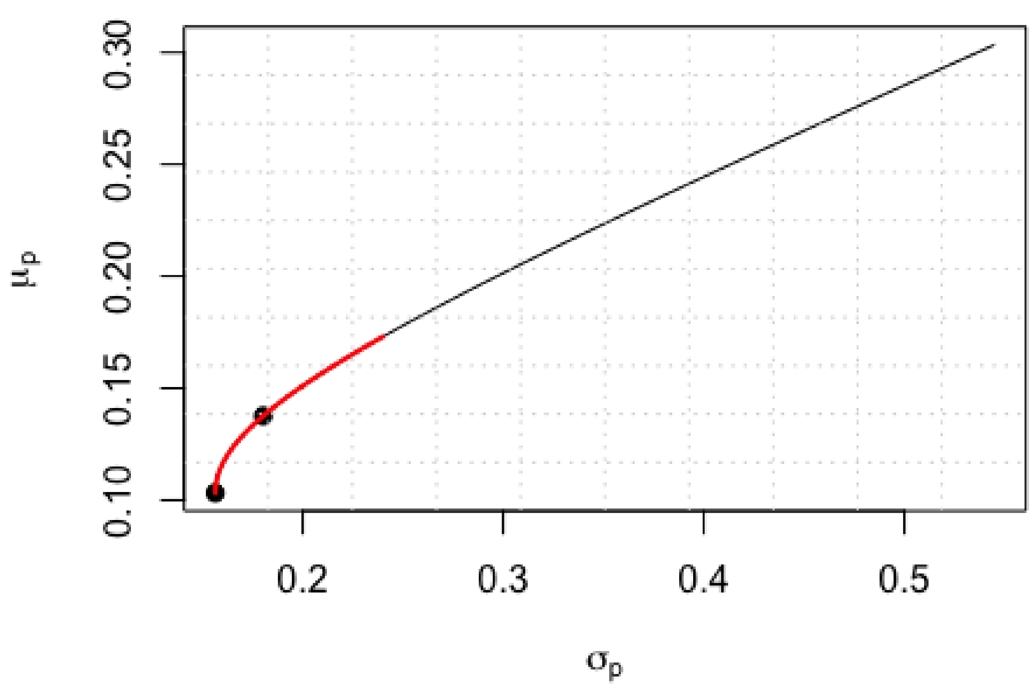
```
## [1] 2.9092107 2.5975095 0.7820459 0.8081141 1.0243700
```

```
# MVEF plot
plot(mu_p ~ sig_p,data = ds_A,
     type = "l", ylab = expression(mu[p]),
     xlab = expression(sigma[p]),
     xlim = range(ds_A$sig_p),
     ylim = range(ds_A$mu_p))
points(mu_p~sig_p,data = ds_A[which(ds_A$sig_p == imp['sig','w_0']),],
       col = 1,pch = 20,cex = 1.5)
points(mu_p~sig_p,data = ds_A[which.max(ds_A$Sr),],
       col = 1,pch = 20,cex = 1.5)
grid(10)
```



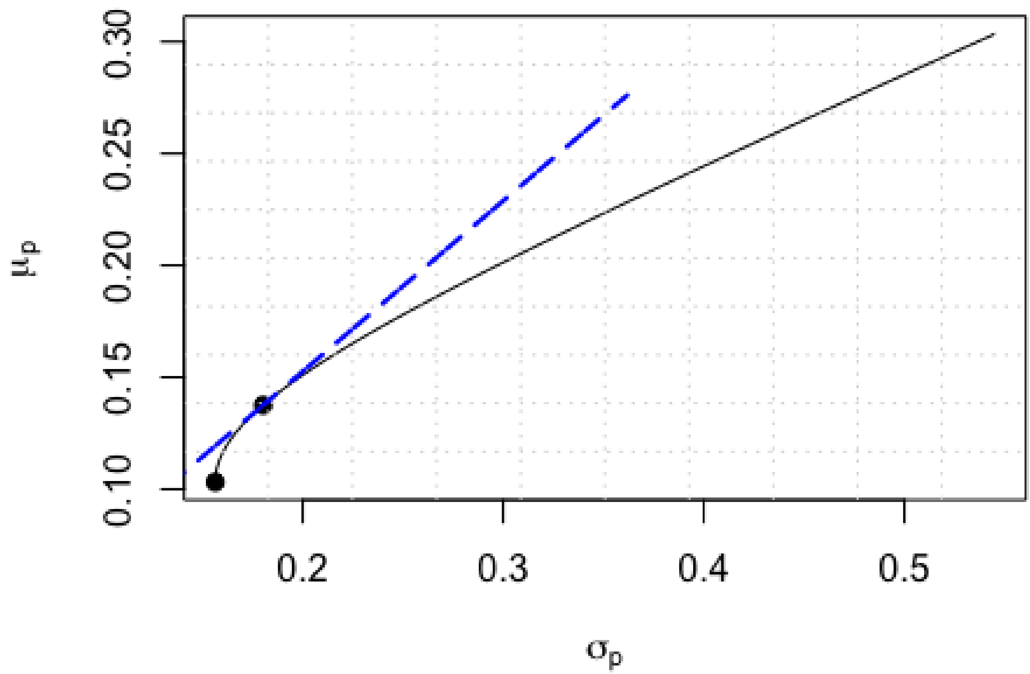

```
# Task 2

lambda<-seq(-1,1,by=0.001)
w_weight<-matrix(NA,2001,3)
```

```r
for (i in lambda){
  w_weight[which(lambda==i),]<-i*w_0+(1-i)*Sr_p
}

# Task 3

mvef<-apply(w_weight, 1, w_function)
rownames(mvef) <- c("mu_p","sig_p")
mvef<-t(mvef)
mvef<-data.frame(mvef)

plot(mu_p ~ sig_p,data = ds_A,
     type = "l", ylab = expression(mu[p]),
     xlab = expression(sigma[p]),
     xlim = range(ds_A$sig_p),
     ylim = range(ds_A$mu_p))
points(mu_p~sig_p,data = ds_A[which(ds_A$sig_p == imp['sig','w_0']),],
       col = 1,pch = 20,cex = 1.5)
points(mu_p~sig_p,data = ds_A[which.max(ds_A$Sr),],
       col = 1,pch = 20,cex = 1.5)
grid(10)
lines(mu_p ~ sig_p,data = mvef,col = 'red',lty = 2,lwd = 2)
```



(b) In two-fund separation theorem we understand that lambda is the part which is investors captial allocated to sharpe portfolio where as 1 - lambda is the part allocated to the GMV portfolio. If lambda is less than 0 this means that investors entire sum of capital is allocated in GMV portfolio and they might be shorting SR potfolio. This can mean that investors are risk averse. Negative lambda can also mean GMV portfolio is borrowed and sold acquiring the SR portfolio.

```
# Task 4

lambda<-seq(-1,1,by=0.001)
w_weight<-matrix(NA,2001,3)
for (i in lambda){
  w_weight[which(lambda==i),]<-(1-i)*Sr_p
}

mvef_0<-apply(w_weight, 1, w_function)
rownames(mvef_0) <- c("mu_p","sig_p")
mvef_0<-t(mvef_0)
mvef_0<-data.frame(mvef_0)

plot(mu_p ~ sig_p,data = ds_A,
     type = "l", ylab = expression(mu[p]),
     xlab = expression(sigma[p]),
     xlim = range(ds_A$sig_p),
     ylim = range(ds_A$mu_p))
points(mu_p~sig_p,data = ds_A[which(ds_A$sig_p == imp['sig','w_0']),],
       col = 1,pch = 20,cex = 1.5)
points(mu_p~sig_p,data = ds_A[which.max(ds_A$Sr),],
       col = 1,pch = 20,cex = 1.5)
grid(10)
lines(mu_p ~ sig_p,data = mvef_0,col = 'blue',lty = 2,lwd = 2)
```

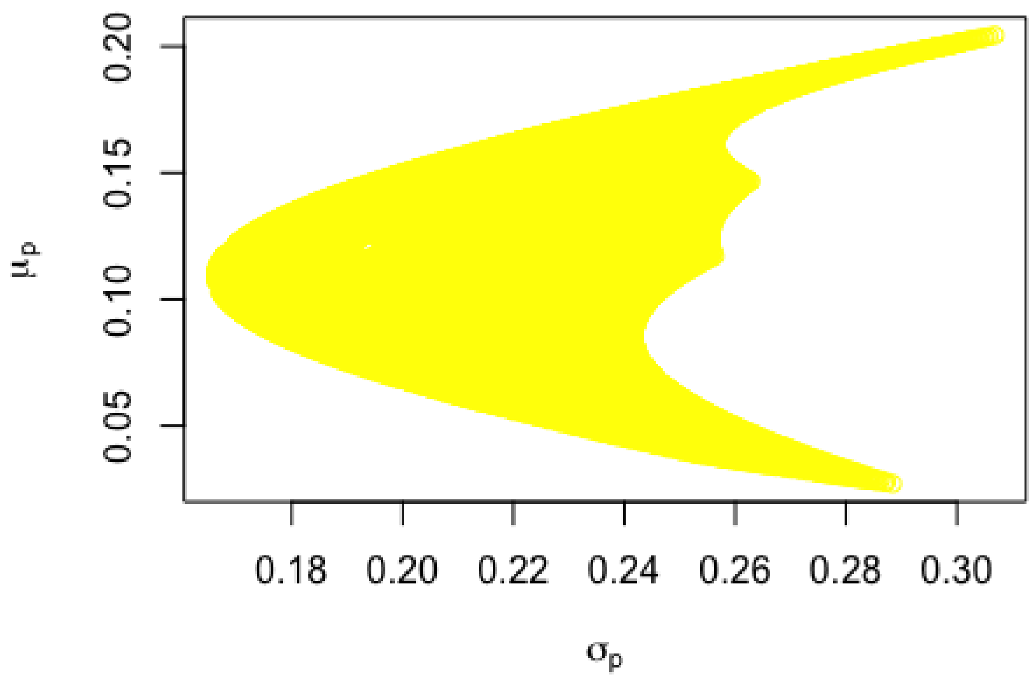



```
model<-lm(mu_p ~ sig_p,mvef_0)
summary(model)
```

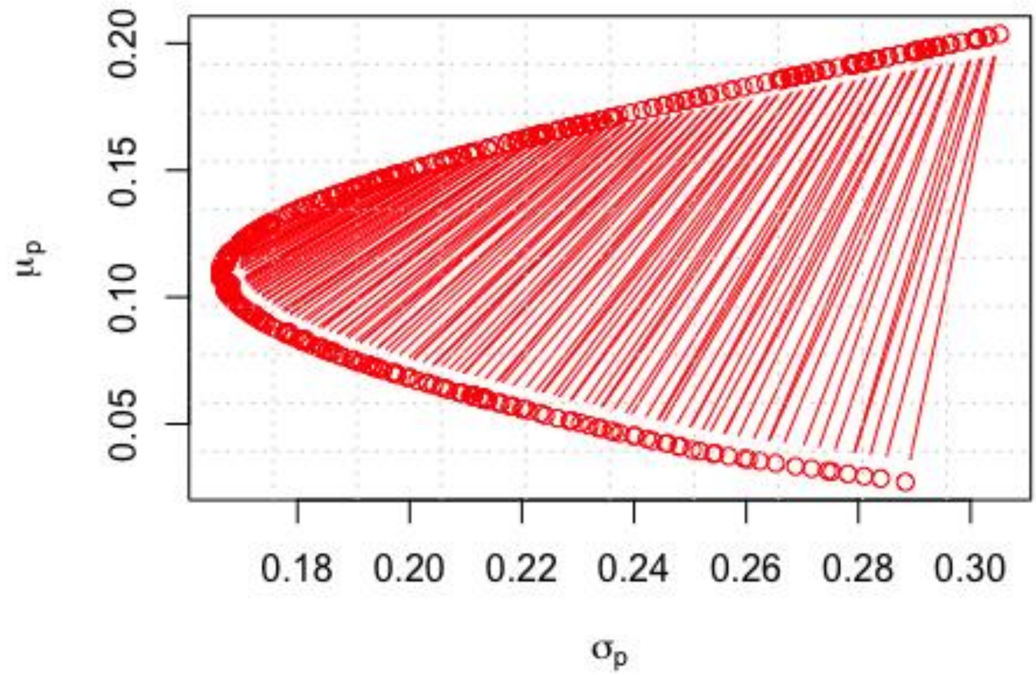
```
##
## Call:
## lm(formula = mu_p ~ sig_p, data = mvef_0)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -9.242e-15 -1.600e-17  2.000e-19  2.050e-17  3.124e-15
##
## Coefficients:
##              Estimate Std. Error   t value Pr(>|t|)
## (Intercept) 4.379e-16  9.882e-18 4.431e+01   <2e-16 ***
## sig_p       7.632e-01  4.730e-17 1.613e+16   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.211e-16 on 1999 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 2.603e+32 on 1 and 1999 DF,  p-value: < 2.2e-16
```

(c) The slope of the regression can pe interpreted as the market risk and the reward per unit of risk investor is willing to take. The intercept represents the risk free rate.

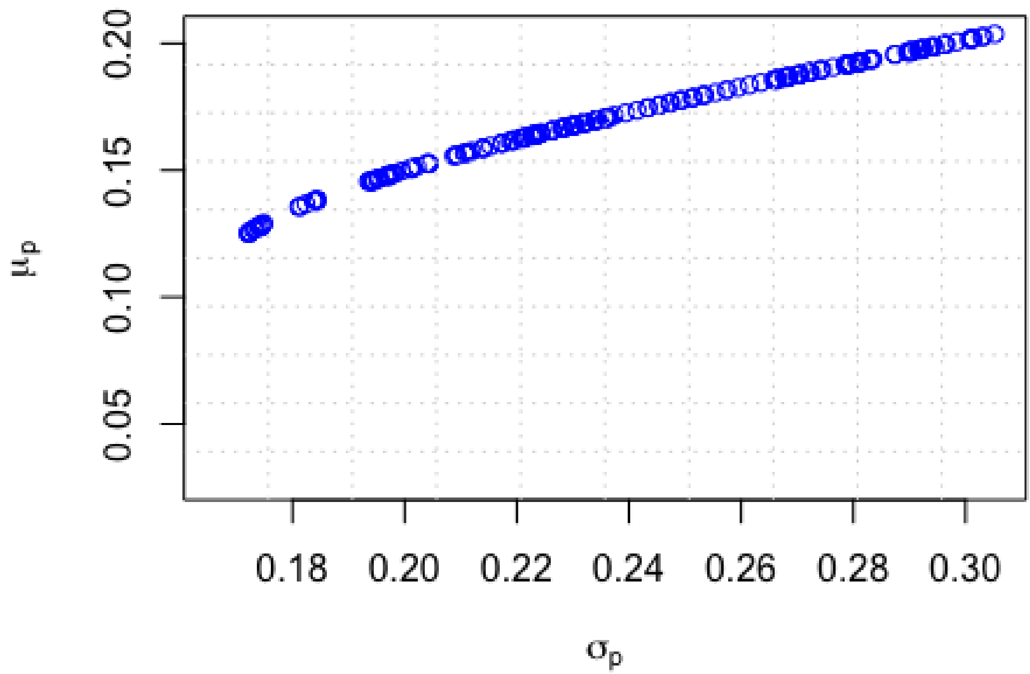
```
# Task 5
plot(mu_p ~ sig_p, data = results,
     type = "p", ylab = expression(mu[p]),
     xlab = expression(sigma[p]),
     xlim = range(results$sig_p),
     ylim = range(results$mu_p),
     col = "yellow")
```



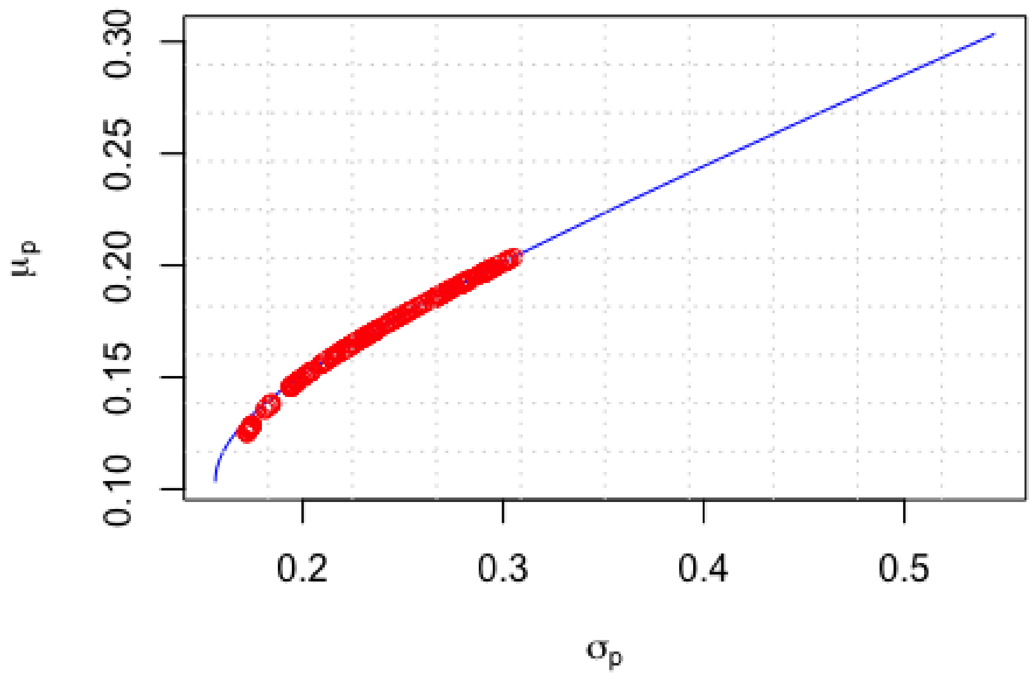

```
plot(mu_p ~ sig_p, data = result_0,
     type = "b", ylab = expression(mu[p]),
     xlab = expression(sigma[p]),
     xlim = range(result_0$sig_p),
     ylim = range(result_0$mu_p),
     col = "red")
grid(10)
```




```
plot(mu_p ~ sig_p, data = result_df2,
     type = "p", ylab = expression(mu[p]),
     xlab = expression(sigma[p]),
     xlim = range(result_0$sig_p),
     ylim = range(result_0$mu_p),
     col = "blue")
grid(10)
```




```
plot(mu_p ~ sig_p, data = ds_A,
     type = "l", ylab = expression(mu[p]),
     xlab = expression(sigma[p]),
     xlim = range(ds_A$sig_p),
     ylim = range(ds_A$mu_p),
     col = "blue")
lines(mu_p ~ sig_p, data = result_df2, type = "p", col = "red")
grid(10)
```



#Q3

#Part 1

In this experiment we are trying to find the cost we bear per coin toss which is k.

Firstly we declare a function coin_toss which runs a while loop and simulates a single run of the game,it counts the number of tosses until three consecutive heads are seen.

```
coin_toss <- function(n) {
  heads<- 0
  tosses <- 0

  while (heads< 3) {
    toss <- sample(c(0, 1), size = 1)
    tosses <- tosses + 1

    if (toss == 1) {
      heads<- heads+ 1
    } else {
      heads<- 0
    }
  }
```

```
  return(tosses)
}
```

In simulations part here we use a function replicate which runs the coin_toss function 100000 times giving us large enough sample set to estimate the value of X which is the average number of tosses to get three consecutive heads.

simulations

```
# simulations
num_simulations <- 100000
tosses_data <- replicate(num_simulations, coin_toss(n = num_simulations))

#### In the follwoing part we compute the expected value of X and solve for k uisng 1 − E[X] × k = 0.

# expected value of X
E_X <- mean(tosses_data)
E_X

## [1] 14.03357

# Solve for k such that E[P&L] = 0
k <- 1 / E_X
print(k)

## [1] 0.07125771
```

Part 2

We write a function which simulates a single run for the experiment.

```
tur_exp <- function(num_turt) {
  turtles <- sample(num_turt)
  groups <- unique(cummin(turtles))
  return(length(groups))
}
# We chose the number of turtles to be 100 and to sample them we replicate it 10^5 times. Finally we take the mean to
# obtain the average number of groups.
num_turt <- 100
num_groups <- replicate(10^5, tur_exp(num_turt))
mean_num_groups <- mean(num_groups)
mean_num_groups

## [1] 5.19022
```

Part 3

In the first line we create a vector of all the prices at P2 step.

```
# a-
price_2 <- c(120,100,80)
#### We define the probabilities leading to P2.
prob <- c(0.55*0.55,0.45*0.55*2,0.45*0.45)
#### We calculate the expectation of P2 and Variance in the next steps.
exp <- sum(price_2*prob)
var <- sum((price_2-exp)^2*prob)
print(c(exp,var))

## [1] 102 198

# b -
# Parameters
initial_price <- 100
num_steps <- 2  # For M = 10 steps
num_simulations <- 10000
p_up <- 0.55  # Probability of price going up
change_up <- 10  # Price change when it goes up
change_down <- -10  # Price change when it goes down

# Run the simulation
simulation_results <- run_simulation(initial_price, num_steps,
num_simulations, p_up, change_up, change_down)

# Output the results
cat("Expected price E[P10] =", simulation_results$mean, "\n")

## Expected price E[P10] = 102.056

cat("Variance of price V[P10] =", simulation_results$variance, "\n")

## Variance of price V[P10] = 192.5121

initial_price <- 100
num_steps <- 10  # For M = 10 steps
num_simulations <- 10000
p_up <- 0.55  # Probability of price going up
change_up <- 10  # Price change when it goes up
change_down <- -10  # Price change when it goes down

simulation_results <- run_simulation(initial_price, num_steps,
num_simulations, p_up, change_up, change_down)

# Output the results
cat("Expected price E[P10] =", simulation_results$mean, "\n")

## Expected price E[P10] = 109.922
```

```
cat("Variance of price V[P10] =", simulation_results$variance, "\n")

## Variance of price V[P10] = 992.9732
```

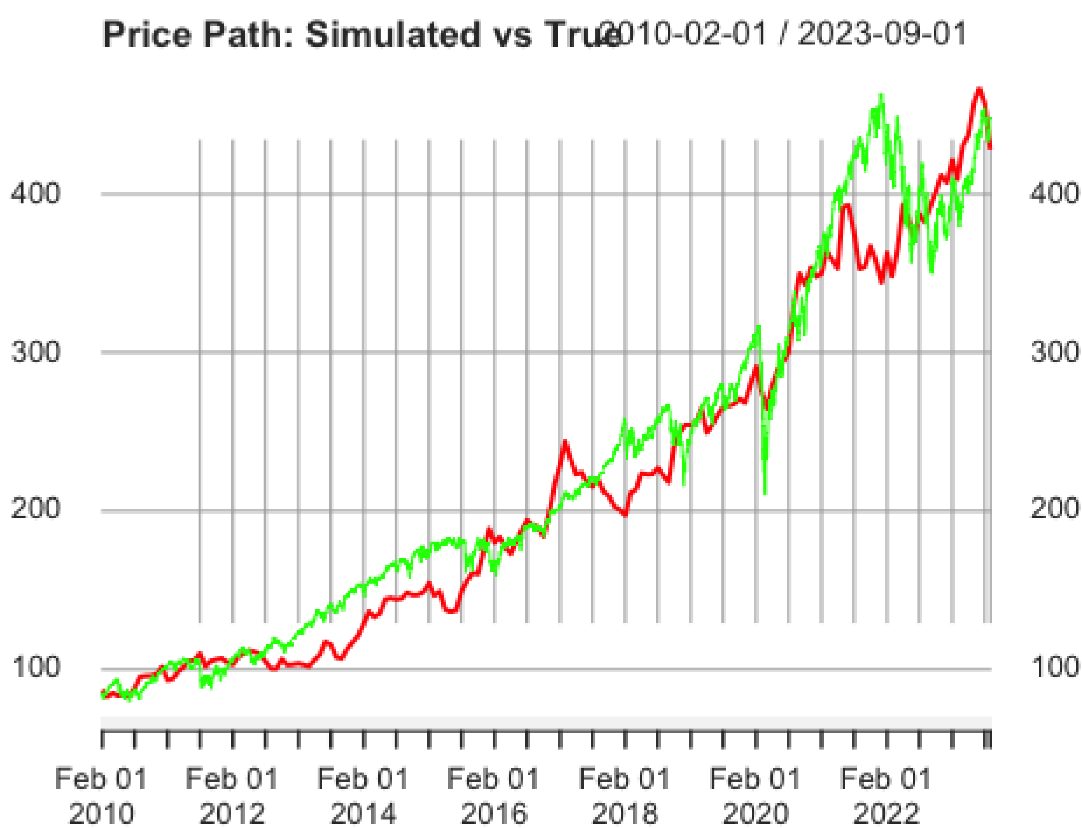
```
As we see our simulations works for 2 steps so we get near perform for 10
steps.
```

Q4

Part a

```
library(quantmod)

getSymbols('IVV', from = "2010-01-01", to = "2023-09-30")

## [1] "IVV"

data <- to.monthly(IVV)
adj_p <- data$IVV.Adjusted
S0 <- data$IVV.Adjusted[[1]]
log_ret1 <-  na.omit(log(adj_p/lag(adj_p)))

log_ret <- mean(log_ret1) # log returns

sigma <- sd(log_ret1)* sqrt(12)
mu <- mean(log_ret1) * 12 + sigma^2 / 2
table <- data.frame(mu,sigma)
print(table)

##          mu     sigma
## 1 0.1309751 0.1470194

# Part b


simu_vs_true <- data.frame(Mean = c(simu_expe, simu_expe_true), Sigma =
c(simu_sigma,simu_sigma_true))
rownames(simu_vs_true) <- c('Simulation', 'True Value')
simu_vs_true

##                Mean    Sigma
## Simulation 489.0634 287.8034
## True Value 494.2063 289.7183
```

```r
# Part c

norm<-function(p){
  norm<-c()
  for (i in 1:1000){
    norm<- c(norm, sum(abs(s_matrix[,i]-adj_p)^p)^(1/p))
  }
  norm<-matrix(norm,nrow=1000)
  norm<-data.frame(norm)
  return(norm)
}


x<-xts(s_matrix[,i],order.by=seq.Date(from = as.Date("2010-02-01"), to =
as.Date("2023-09-30"), by = "months"))
plot(x, type="l",main="Price Path: Simulated vs True", pch=1, col = 'red')

lines(IVV$IVV.Adjusted, pch=2, col = 'green')
```



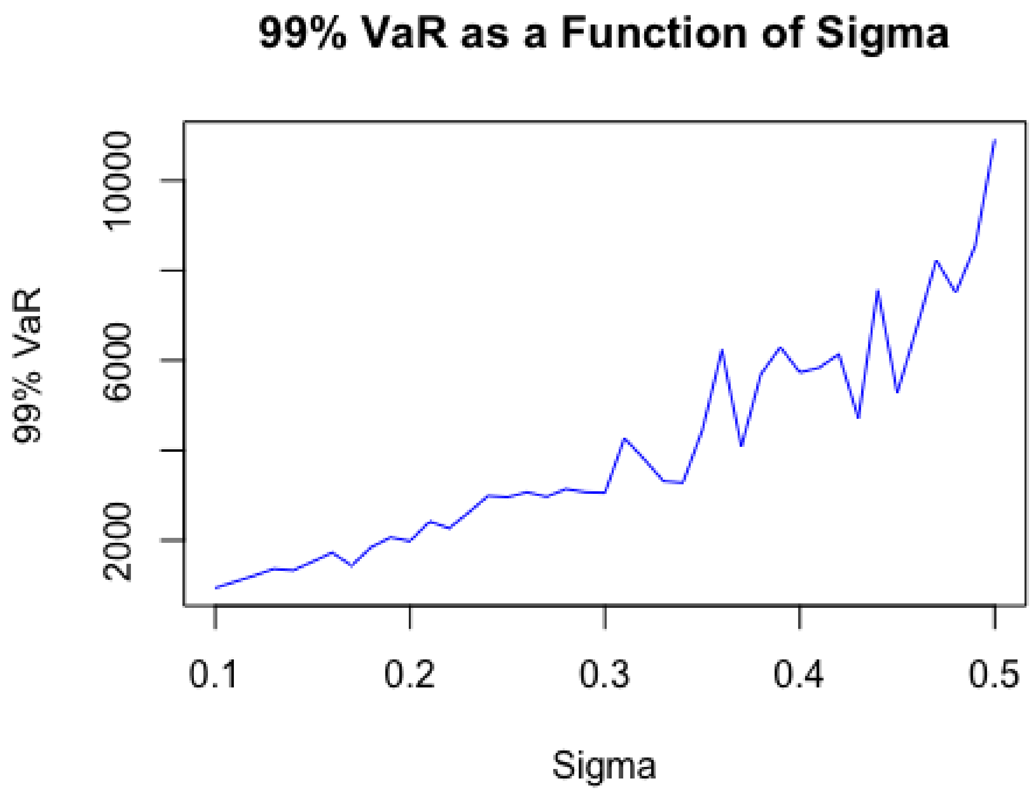
```r
# Part d -

quant_99 <- quantile(simu*100,0.01)
quant_99
```

```
##       1%
## 10596.28
```

```r
VaR <- mean(simu)*100 - quant_99
print(VaR)
```

```
##       1%
## 38310.06
```

```r
# Part e -
num_periods <- 165

gbm <- function(S0, mu, sigma, dt, num_periods) {
  drt <- rnorm(num_periods - 1, mean = (mu - sigma^2 / 2) * dt, sd = sigma *
sqrt(dt))
  St <- S0 * exp(cumsum(c(0, drt)))
  return(St)
}

# Plot VaR against sigma
plot(sigma_range, VaR_values, type = 'l', col = 'blue', xlab = 'Sigma', ylab
= '99% VaR', main = '99% VaR as a Function of Sigma')
```



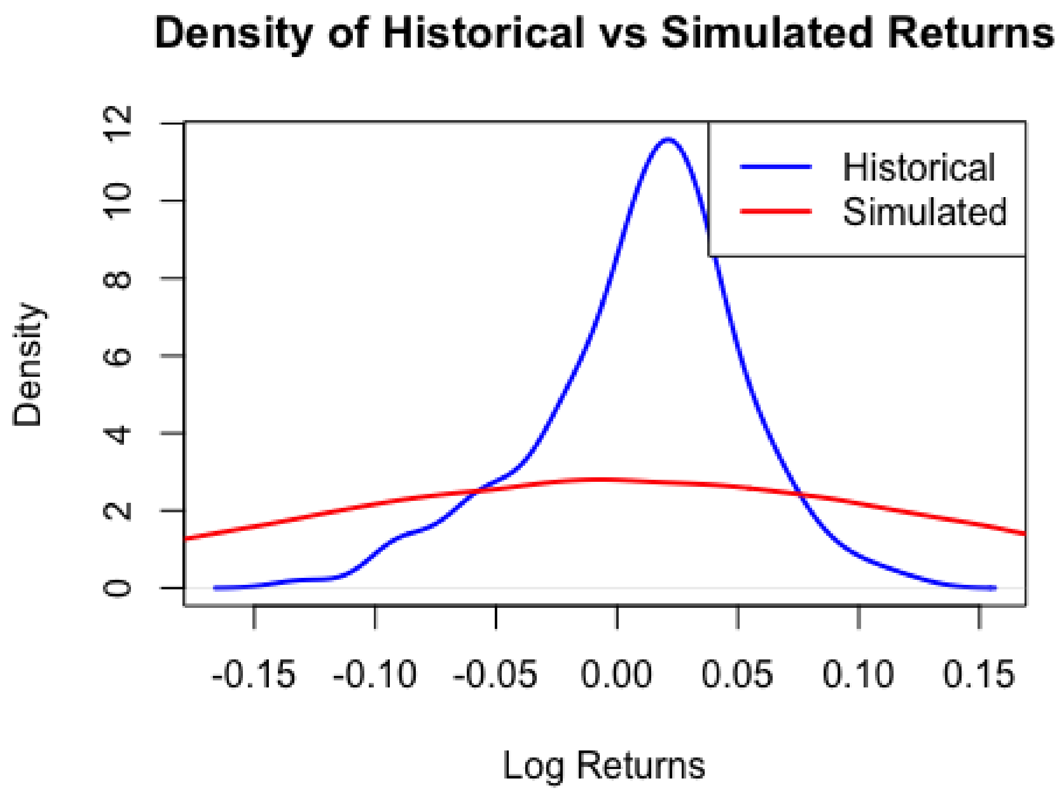
ii) We can observe how VaR changes as a function of volatility. It suggest that the VaR is highly sensitive to the model's input parameters. Therefore model risk is associated with the possibility that the parameter estimation can lead to significant errors in model's calculation.

iii) The relationship between VaR and sigma we find it to be nonlinear, as sigma increases the VaR does not increase at constant rate.

In comparison to IID Gaussian process we expect the VaR to increase linearly with sigma because the VaR in a Gaussian framework is a direct multiple of sigma.

```
# Part f -
# Historical -
quant <- quantile(log_ret1,0.01)
print(quant)

##          1%
## -0.09440559
```

```r
VaR <- log_ret - quant
print(VaR)

##        1%
## 0.1044196

# Parametric -

# Parametric approach

para<-matrix(NA,nrow=164,ncol=1000)
for (i in 1:1000){
  for (j in 2:165){
    para[j-1,i]<-log(s_matrix[j,i]/s_matrix[j-1,i])}}
quant <- quantile(para,0.01)
VaR_par <- mean(para) - quant
print(VaR_par)

##        1%
## 0.3353513

plot(historical_density, main = "Density of Historical vs Simulated Returns",
     xlab = "Log Returns", ylab = "Density", col = "blue", lwd = 2)
lines(simulated_density, col = "red", lwd = 2)
legend("topright", legend = c("Historical", "Simulated"), col = c("blue",
"red"), lwd = 2)
```



iii) We see that parametric approach yields much larger value than the historical approach.

v) We have learned that –

a) Model outputs such as VaR are highly sensitive to estimated parameters especially sigma.

b) Financial data often show nonlinearity and the real world data exhibit skewness and kurtosis.