my_Rmarkdown

best so far

2025-02-09

```r
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.2.3

## Warning: package 'ggplot2' was built under R version 4.2.3

## Warning: package 'tibble' was built under R version 4.2.3

## Warning: package 'tidyr' was built under R version 4.2.3

## Warning: package 'readr' was built under R version 4.2.3

## Warning: package 'purrr' was built under R version 4.2.3

## Warning: package 'dplyr' was built under R version 4.2.3

## Warning: package 'forcats' was built under R version 4.2.3

## Warning: package 'lubridate' was built under R version 4.2.3

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(quantmod)
```

```
## Warning: package 'quantmod' was built under R version 4.2.3

## Loading required package: xts

## Warning: package 'xts' was built under R version 4.2.3

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 4.2.3

##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
```

```
##
## ######################### Warning from 'xts' package ##########################
## #                                                                             #
## # The dplyr lag() function breaks how base R's lag() function is supposed to  #
## # work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
## # source() into this session won't work correctly.                            #
## #                                                                             #
## # Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
## # conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
## # dplyr from breaking base R's lag() function.                                #
## #                                                                             #
## # Code in packages is not affected. It's protected by R's namespace mechanism #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                                             #
## ###############################################################################
##
## Attaching package: 'xts'
##
## The following objects are masked from 'package:dplyr':
##
##     first, last
##
## Loading required package: TTR
```

```
## Warning: package 'TTR' was built under R version 4.2.3
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(rgl)
```

```
## Warning: package 'rgl' was built under R version 4.2.3
```

```
library(ggpubr)
```

```
## Warning: package 'ggpubr' was built under R version 4.2.3
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.2.3
```

```
library(ks)
```

```
## Warning: package 'ks' was built under R version 4.2.3
```

```
library(goftest)
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 4.2.3
```

```
##
## Attaching package: 'plotly'
##
## The following object is masked from 'package:MASS':
##
##     select
##
## The following object is masked from 'package:ggplot2':
##
##     last_plot
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following object is masked from 'package:graphics':
##
##     layout
```

```r
#install.packages(c("fitdistrplus","metRology","copula"))
#install.packages("metRology")
library(copula)
```

```
## Warning: package 'copula' was built under R version 4.2.3
```

```
##
## Attaching package: 'copula'
##
## The following object is masked from 'package:lubridate':
##
##     interval
```

```r
#library(metRology)
library(fitdistrplus)
```

```
## Loading required package: survival
```

```
## Warning: package 'survival' was built under R version 4.2.3
```

```r
getSymbols("^GSPC", from="2010-01-01", to="2022-12-31")
```

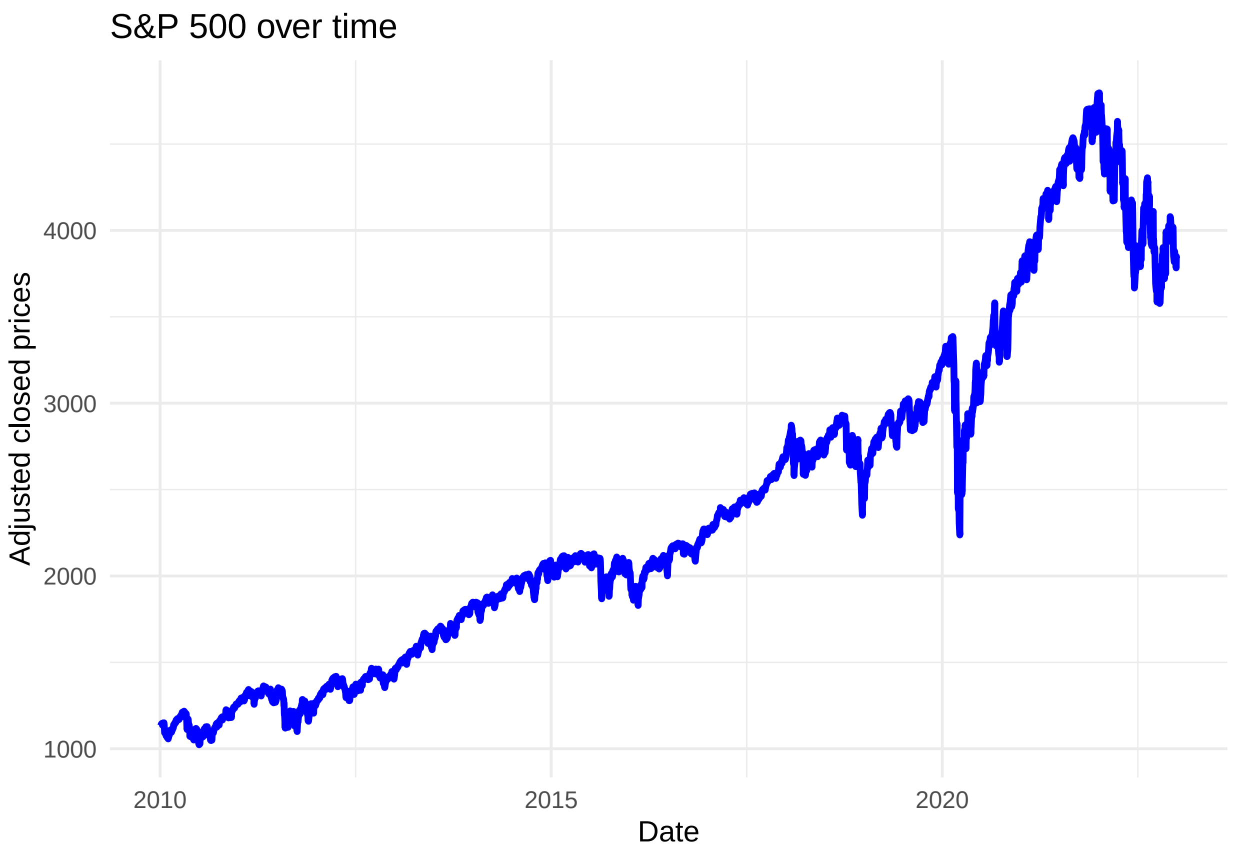
```
## [1] "GSPC"
```

```r
getSymbols("^DJI", from="2010-01-01", to="2022-12-31")
```

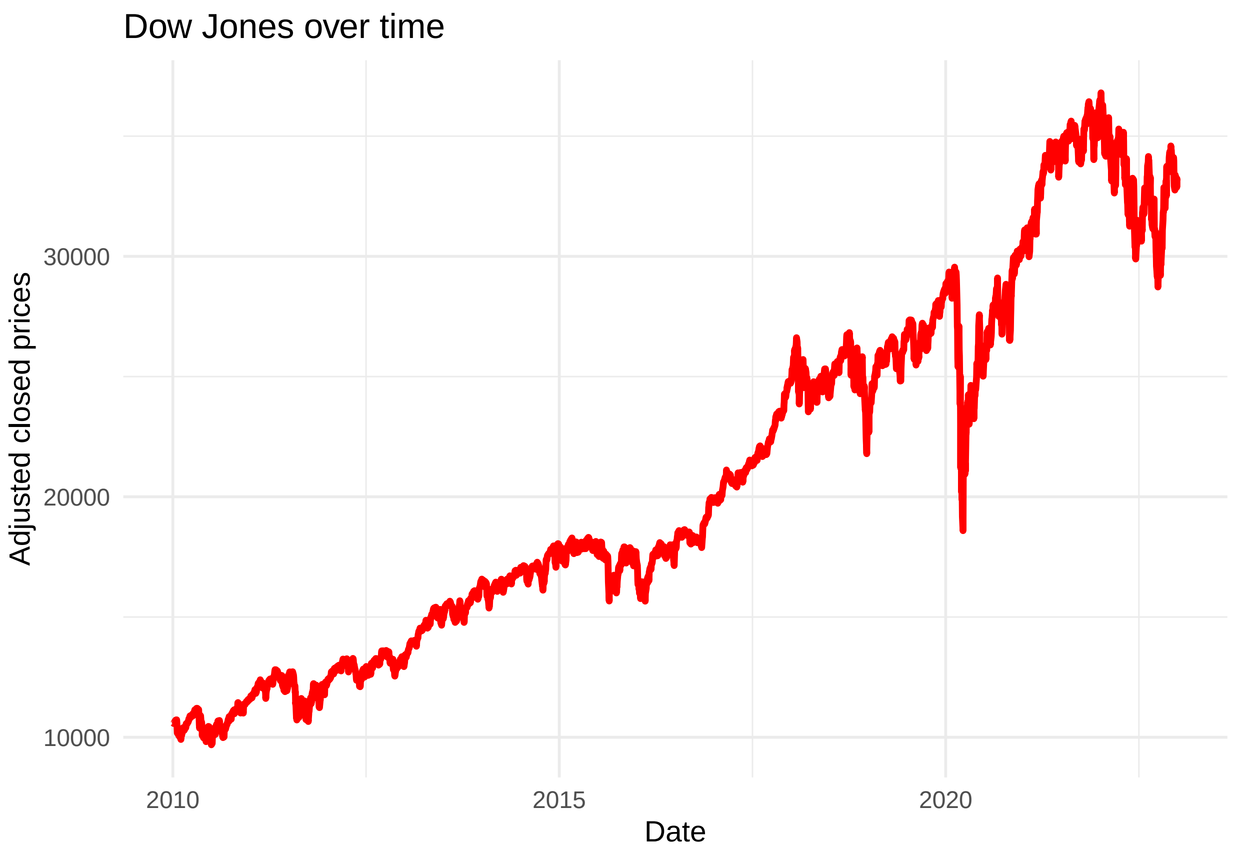
```
## [1] "DJI"
```

```r
GSPC_adj <- Ad(GSPC)
DJI_adj <- Ad(DJI)
```

```r
ggplot(GSPC_adj,aes(x=index(GSPC_adj),y=GSPC.Adjusted))+
  geom_line(linewidth=1.2,color="blue")+
  ggtitle("S&P 500 over time")+
  labs(x="Date",y="Adjusted closed prices")+
  theme_minimal()
```



```
ggplot(DJI_adj,aes(x=index(DJI_adj),y=DJI.Adjusted))+
  geom_line(linewidth=1.2,color="red")+
  ggtitle("Dow Jones over time")+
  labs(x="Date",y="Adjusted closed prices")+
  theme_minimal()
```



```
GSPC_adj$returns <- diff(log(GSPC_adj)*100)
DJI_adj$returns <- diff(log(DJI_adj)*100)
```

```
Compute_statistics <- function(returns){
  return(data.frame(mean=mean(returns,na.rm=T),
                    std_dev=sd(returns,na.rm = T),
                    skewness= skewness(returns,na.rm = T),
                    kurtosis=kurtosis(returns,na.rm = T)))
}
```

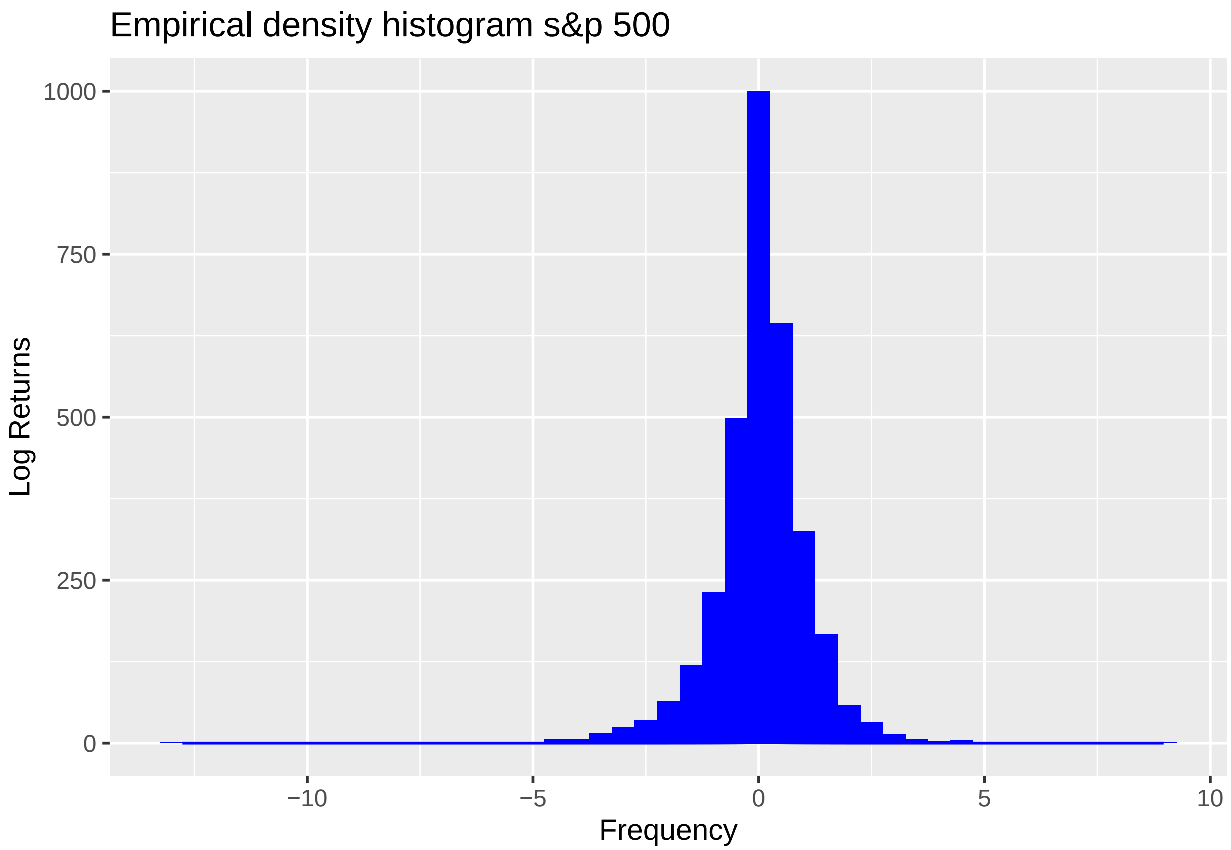
```
S_P_500_stats <- Compute_statistics(GSPC_adj$returns)
Dow_jones_stats <- Compute_statistics(DJI_adj$returns)
```

```
Stats_table <- rbind(S_P_500_stats,Dow_jones_stats)
rownames(Stats_table)<- c("S&P_500","Dow Jones")
print(Stats_table)
```

```
##                  mean  std_dev   skewness kurtosis
## S&P_500    0.03731220 1.125488 -0.7333844 13.19015
## Dow Jones 0.03490126 1.091497 -0.8571841 19.26713
```

```
ggplot(GSPC_adj,aes(x=returns))+
  geom_histogram(binwidth = 0.5,fill="blue")+
  geom_density(color="blue")+
  ggtitle("Empirical density histogram s&p 500")+
  labs(x="Frequency",y="Log Returns")
```

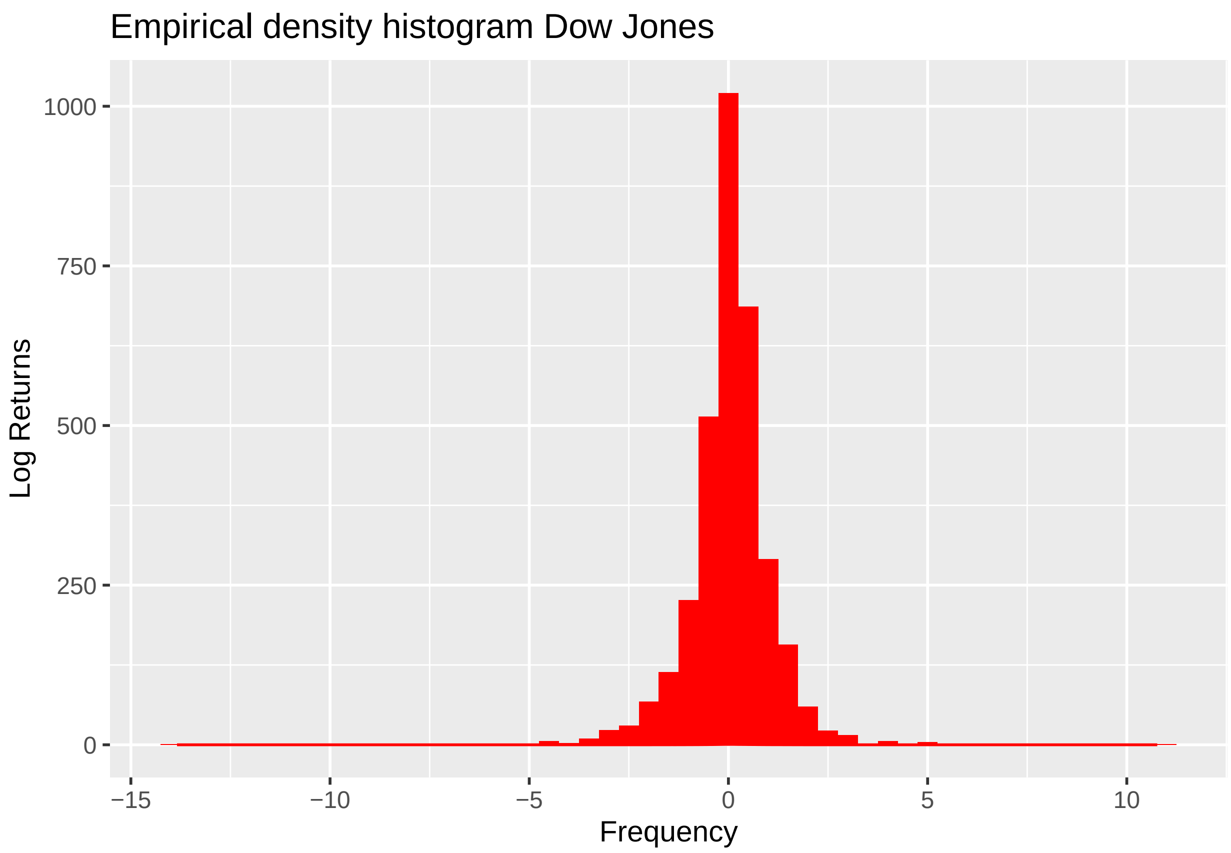
```
## Warning: Removed 1 rows containing non-finite values (`stat_bin()`).
## Warning: Removed 1 rows containing non-finite values (`stat_density()`).
```



```
ggplot(DJI_adj,aes(x=returns))+
  geom_histogram(binwidth = 0.5,fill="red")+
  geom_density(aes(x=returns),color="red")+
  ggtitle("Empirical density histogram Dow Jones")+
  labs(x="Frequency",y="Log Returns")
```

```
## Warning: Removed 1 rows containing non-finite values (`stat_bin()`).

## Warning: Removed 1 rows containing non-finite values (`stat_density()`).
```



```
returns_data <- na.omit(data.frame(GSPC_adj$returns,DJI_adj$returns))
colnames(returns_data) <- c("S&P 500","Dow Jones")

n <- nrow(returns_data)

ranks_sp500 <- rank(returns_data$`S&P 500`/(n+1))
rank_Dowjone <- rank(returns_data$`Dow Jones`/(n+1))
```

```
Cn_values <- numeric(n)

for(i in 1:n){
  Cn_values[i]<- mean(ranks_sp500<=ranks_sp500[i]&rank_Dowjone<=rank_Dowjone[i])
}

copula_data <- data.frame(u1=ranks_sp500,u2=rank_Dowjone,
                          Cn=Cn_values)
```

```
range(copula_data$Cn)
```

```
## [1] 0.0003057169 1.0000000000
```

```
plot3d(copula_data$u1,copula_data$u2,copula_data$Cn,
       col = "blue",size = 6, type = "s",
       xlab = "U1 (S&P 500)",
       ylab = "U2 (Dow Jones)",
       zlab = "Cn(u1,u2)")
```

```
##Normal
normal_s_p_500 <-fitdistr(na.omit(GSPC_adj$returns),"normal")
normal_dow_jone <-fitdistr(na.omit(DJI_adj$returns),"normal")

#T-distribution
t_S_P_500 <- fitdistr(na.omit(GSPC_adj$returns),"t",start = list(m=mean(GSPC_adj$returns,na.rm=T),
                                                     s=sd(GSPC_adj$returns,na.rm=T),df=5))
```

```
## Warning in log(s): NaNs produced

## Warning in log(s): NaNs produced

## Warning in log(s): NaNs produced

## Warning in log(s): NaNs produced

## Warning in log(s): NaNs produced
## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced
## Warning in log(s): NaNs produced
## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced
## Warning in log(s): NaNs produced
## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced
## Warning in log(s): NaNs produced
## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced
```

```
T_Dow_jones <- fitdistr(na.omit(DJI_adj$returns),"t",start = list(m=mean(DJI_adj$returns,na.rm=T),
                                                                  s=sd(DJI_adj$returns,na.rm=T),df=5))
```

```
## Warning in log(s): NaNs produced
## Warning in log(s): NaNs produced

## Warning in log(s): NaNs produced

## Warning in log(s): NaNs produced

## Warning in log(s): NaNs produced
## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced
## Warning in log(s): NaNs produced
## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced
## Warning in log(s): NaNs produced

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced

## Warning in log(s): NaNs produced

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced

## Warning in log(s): NaNs produced

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produced

## Warning in log(s): NaNs produced
```

```
# 2. Goodness-of-fit tests (Cramer-von Mises & Kolmogorov-Smirnov)
# Scaling data for t-distribution test
sp500_scaled <- (na.omit(GSPC_adj$returns) - t_S_P_500$estimate[1]) / t_S_P_500$estimate[2]
dj_scaled <- (na.omit(DJI_adj$returns) - T_Dow_jones$estimate[1]) / T_Dow_jones$estimate[2]

# Kolmogorov-Smirnov test
ks_sp500_normal <- ks.test(na.omit(GSPC_adj$returns), "pnorm", mean = normal_s_p_500$estimate[1], sd = n
ks_dj_normal <- ks.test(na.omit(DJI_adj$returns), "pnorm", mean = normal_dow_jone$estimate[1], sd = norm
```

```
## Warning in ks.test.default(na.omit(DJI_adj$returns), "pnorm", mean =
## normal_dow_jone$estimate[1], : ties should not be present for the
## Kolmogorov-Smirnov test
```

```
ks_sp500_t <- ks.test(sp500_scaled, "pt", df = t_S_P_500$estimate[3])
ks_dj_t <- ks.test(dj_scaled, "pt", df = T_Dow_jones$estimate[3])
```

```
## Warning in ks.test.default(dj_scaled, "pt", df = T_Dow_jones$estimate[3]): ties
## should not be present for the Kolmogorov-Smirnov test
```

```
# Cramer-von Mises test
cvm_sp500_normal <- cvm.test(na.omit(GSPC_adj$returns), null = "pnorm", mean = normal_s_p_500$estimate[
cvm_dj_normal <- cvm.test(na.omit(DJI_adj$returns), null = "pnorm", mean = normal_dow_jone$estimate[1],
cvm_sp500_t <- cvm.test(sp500_scaled, null = "pt", df = t_S_P_500$estimate[3])
cvm_dj_t <- cvm.test(dj_scaled, null = "pt", df = T_Dow_jones$estimate[3])

# 3. Create summary tables for estimated parameters
results_table <- data.frame(
  Distribution = c("Normal", "Normal", "t", "t"),
  Index = c("S&P 500", "Dow Jones", "S&P 500", "Dow Jones"),
  Mean = c(normal_s_p_500$estimate[1], normal_dow_jone$estimate[1], t_S_P_500$estimate[1], T_Dow_jones$e
  StdDev = c(normal_s_p_500$estimate[2], normal_dow_jone$estimate[2], t_S_P_500$estimate[2], T_Dow_jone
  DF = c(NA, NA, t_S_P_500$estimate[3], T_Dow_jones$estimate[3])
)

# 4. Create summary tables for goodness-of-fit test results
gof_table <- data.frame(
  Test = c("KS Normal", "KS Normal", "KS t", "KS t", "CVM Normal", "CVM Normal", "CVM t", "CVM t"),
  Index = c("S&P 500", "Dow Jones", "S&P 500", "Dow Jones", "S&P 500", "Dow Jones", "S&P 500", "Dow Jone
  Statistic = c(ks_sp500_normal$statistic, ks_dj_normal$statistic, ks_sp500_t$statistic, ks_dj_t$statist
                cvm_sp500_normal$statistic, cvm_dj_normal$statistic, cvm_sp500_t$statistic, cvm_dj_t$sta
  P_Value = c(ks_sp500_normal$p.value, ks_dj_normal$p.value, ks_sp500_t$p.value, ks_dj_t$p.value,
              cvm_sp500_normal$p.value, cvm_dj_normal$p.value, cvm_sp500_t$p.value, cvm_dj_t$p.value)
)
```

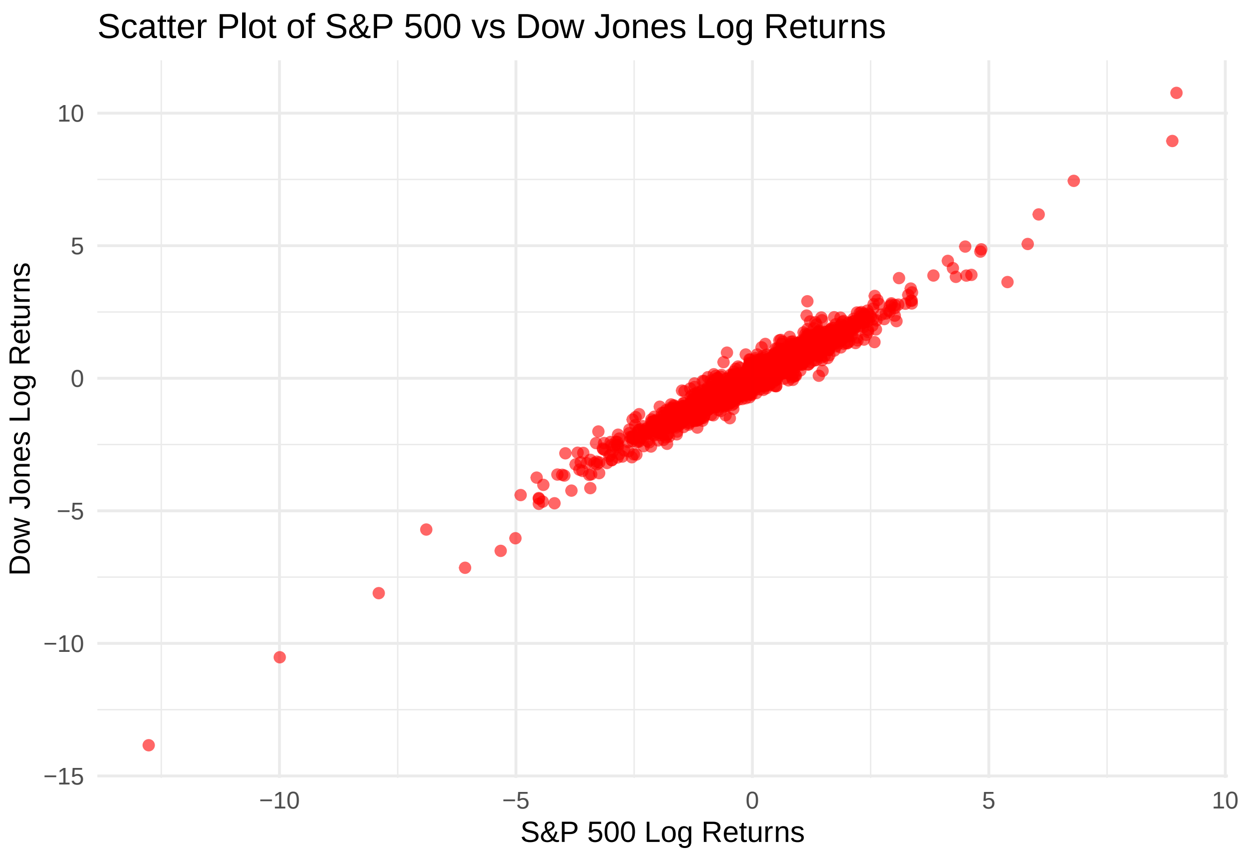
```
# Print results
print(results_table)
```

```
##   Distribution     Index       Mean    StdDev       DF
## 1       Normal   S&P 500 0.03731220 1.1253164       NA
## 2       Normal Dow Jones 0.03490126 1.0913299       NA
## 3            t   S&P 500 0.08145979 0.6330081 2.547594
## 4            t Dow Jones 0.07251199 0.6017421 2.556887
```

```
print(gof_table)
```

```
##         Test     Index  Statistic    P_Value
## 1  KS Normal   S&P 500  0.9855156 0.00000000
## 2  KS Normal Dow Jones  0.9798099 0.00000000
## 3       KS t   S&P 500  0.9792652 0.00000000
## 4       KS t Dow Jones  0.9771123 0.00000000
## 5 CVM Normal   S&P 500 12.3930899 0.00000000
## 6 CVM Normal Dow Jones 13.3928759 0.00000000
## 7      CVM t   S&P 500  0.3549327 0.09533975
## 8      CVM t Dow Jones  0.2717543 0.16272362
```

```
#Scatter plot of log returns
ggplot(returns_data, aes(x = `S&P 500`, y = `Dow Jones`)) +
  geom_point(alpha = 0.6,color="red") +
  labs(title = "Scatter Plot of S&P 500 vs Dow Jones Log Returns",
       x = "S&P 500 Log Returns",
       y = "Dow Jones Log Returns") +
  theme_minimal()
```


Scatter Plot of S&P 500 vs Dow Jones Log Returns
Dow Jones Log Returns
10
5
0
−5
−10
−15
−10
−5
0
5
10
S&P 500 Log Returns