Prob_Rmarkdown

best so far

2025-02-17

```r
##install.packages("plot3D")
library(plot3D)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- 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)
```

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

```r
library(rgl)
library(ggpubr)
library(MASS)
```

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

```r
library(e1071)
library(ks)
library(goftest)
library(plotly)
```

```
##
## 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)
```

```
##
```

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

```r
library(metRology)
```

```
##
## Attaching package: 'metRology'
##
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```r
library(fitdistrplus)
```

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

Q1 DOwnloading the Datasets

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

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

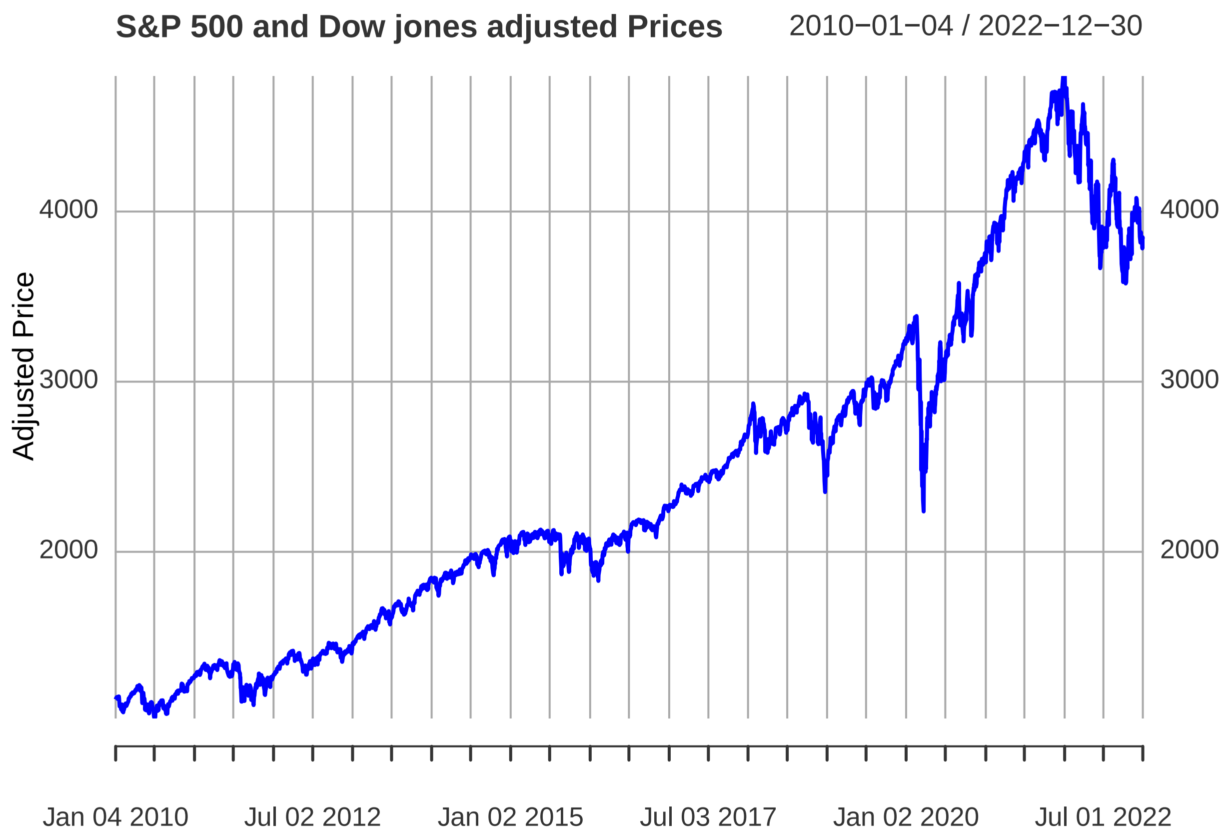
```r
getSymbols("^DJI", from="2010-01-01", to="2022-12-31")
```

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

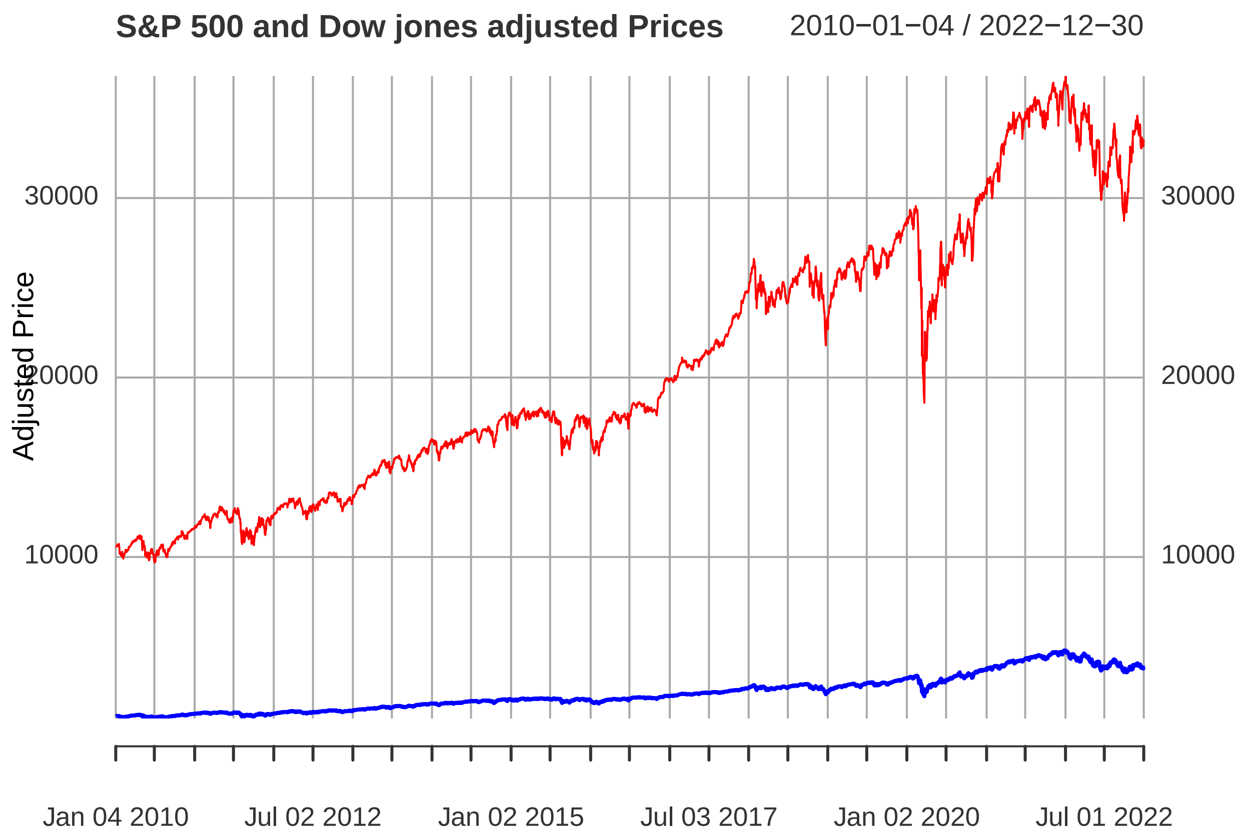
```r
GSPC_adj <- Ad(GSPC)
DJI_adj <- Ad(DJI)
```

Q2 Calculating STATS

```r
plot(GSPC_adj,type="l",col="blue",
     xlab="Year",ylab="Adjusted Price",
     main="S&P 500 and Dow jones adjusted Prices")
```




```
lines(DJI_adj,col="red")
legend("topleft",legend = c("S&P 500","Dow Jones"),
       col=c("blue","red"),lty=1)
```



Q2 Calculating STATS

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

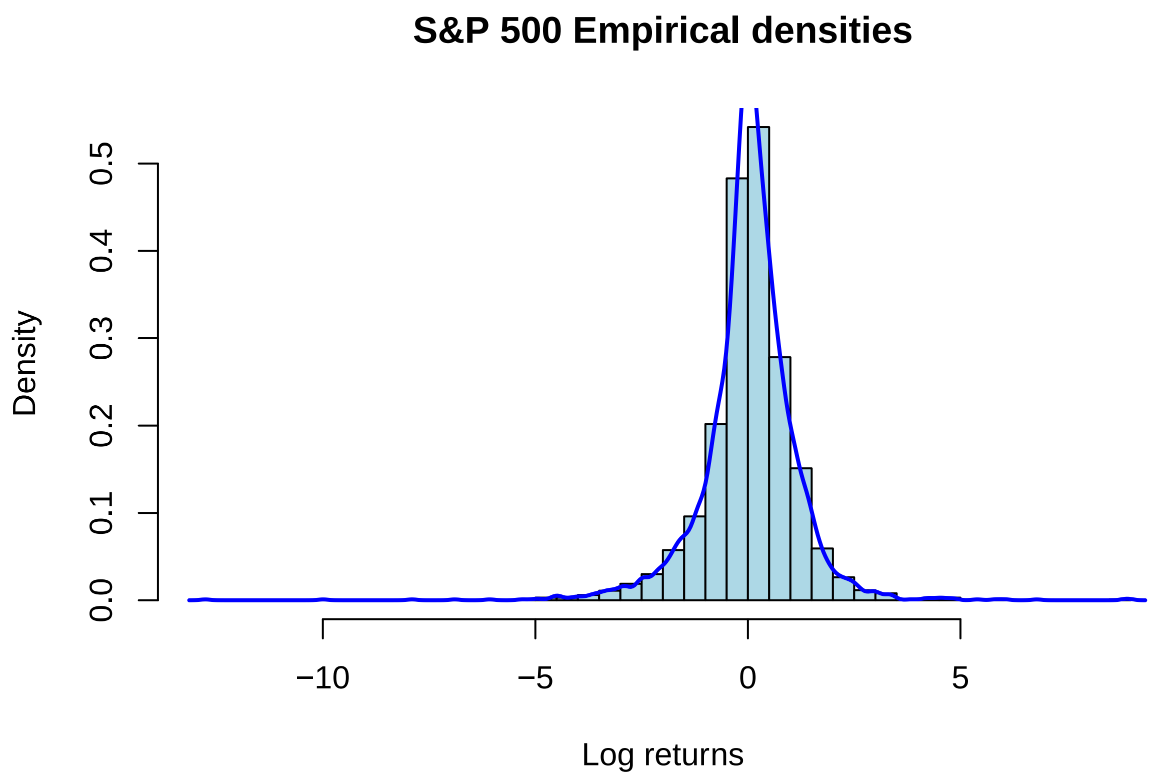
```
GSPC_adj<-na.omit(GSPC_adj)
DJI_adj <- na.omit(DJI_adj)
```

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

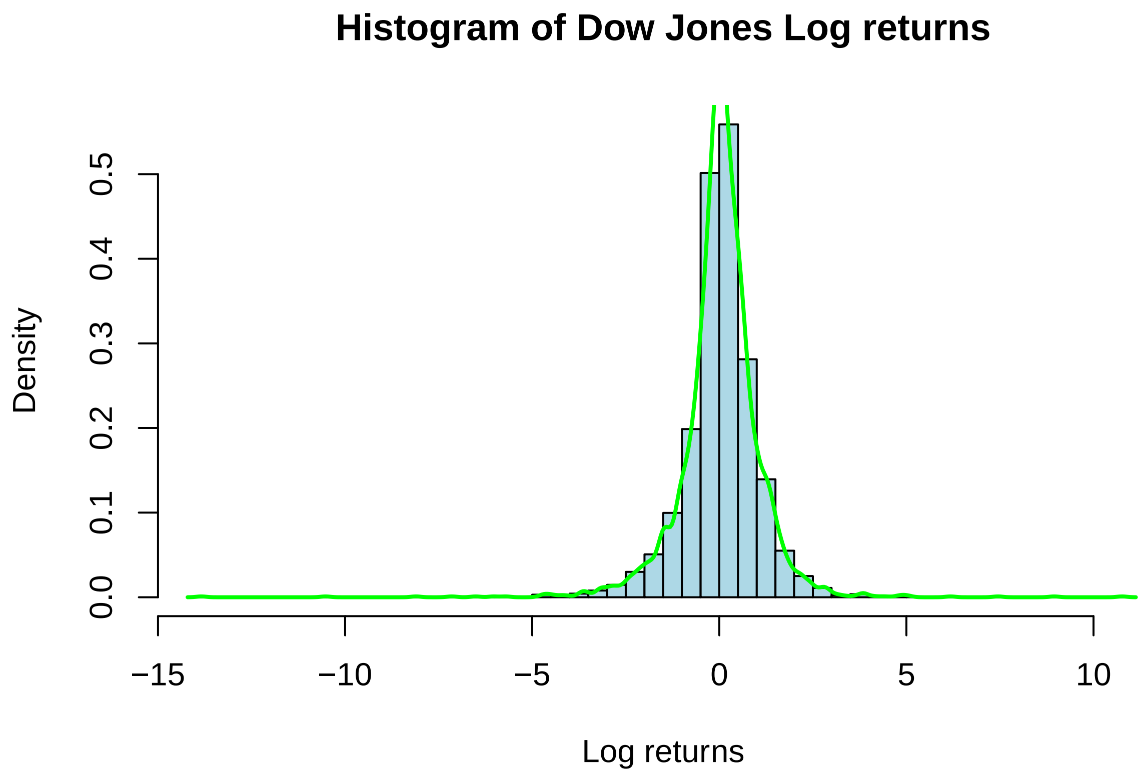
```
S_P_500_stats <- Compute_statistics(GSPC_adj$log_return)
Dow_jones_stats <- Compute_statistics(DJI_adj$log_return)

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
```

```r
hist(GSPC_adj$log_return,breaks = 36,prob=TRUE,col="lightblue",main="S&P 500 Empirical densities",xlab =
lines(density(GSPC_adj$log_return),col="blue",lwd=2)
```

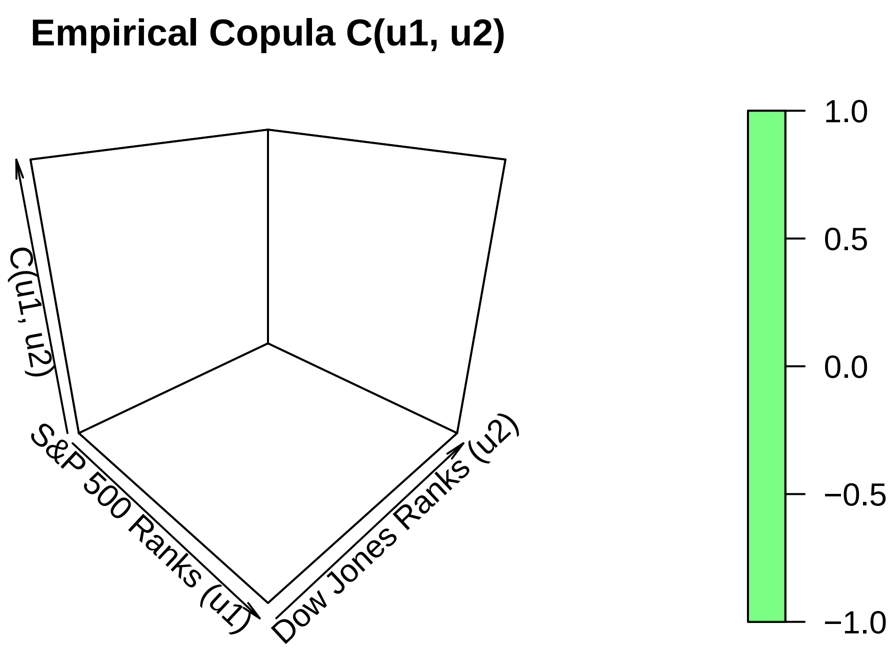



```r
hist(DJI_adj$log_return,breaks = 36,prob=TRUE,col="lightblue",main="Histogram of Dow Jones Log returns"
lines(density(DJI_adj$log_return),col="green",lwd=2)
```




```r
# Columns: 'GSPC_log_returns' and 'DJI_log_returns'
combined_data<-cbind(GSPC_adj$log_return,DJI_adj$log_return)
n <- nrow(combined_data)
u1 <- rank(combined_data$GSPC_log_returs) / (n + 1)
u2 <- rank(combined_data$DJI_log_returs) / (n + 1)
# Define a grid of points in [0, 1]
grid_resolution <- 50  # Adjust for finer/coarser resolution
grid_points <- seq(0, 1, length.out = grid_resolution)
Cn_matrix <- matrix(0, nrow = grid_resolution, ncol = grid_resolution)

for (i in 1:grid_resolution) {
  for (j in 1:grid_resolution) {
    # Calculate C_n at grid point (u1_grid[i], u2_grid[j])
    Cn_matrix[i, j] <- sum(u1 <= grid_points[i] & u2 <= grid_points[j]) / n
  }
}
library(plot3D)

persp3D(
  x = grid_points,
  y = grid_points,
  z = Cn_matrix,
  colkey = TRUE,
  xlab = "S&P 500 Ranks (u1)",
  ylab = "Dow Jones Ranks (u2)",
  zlab = "C(u1, u2)",
```

```
  main = "Empirical Copula C(u1, u2)",
  theta = 45,  # Adjust viewing angle
  phi = 25     # Adjust elevation angle
)
```




```
Normal_distr_GSpc <- fitdistr(GSPC_adj$log_return,"normal")
Normal_distr_Dow <- fitdistr(DJI_adj$log_return,"normal")

## Fit t distr
suppressWarnings(t_distr_GPSC <- fitdistr(GSPC_adj$log_return,"t"))
suppressWarnings(t_distr_DJI <- fitdistr(DJI_adj$log_return,"t"))
```

```
### KS test for normal
ks.ts_GSPC_norm <- ks.test(GSPC_adj$log_return,
                           "pnorm",mean=Normal_distr_GSpc$estimate[1],
                           sd=Normal_distr_GSpc$estimate[2])

ks.ts_Dow_norm <- ks.test(DJI_adj$log_return,
                          "pnorm",mean=Normal_distr_Dow$estimate[1],
                          sd=Normal_distr_Dow$estimate[2])
```

```
## Warning in ks.test.default(DJI_adj$log_return, "pnorm", mean =
## Normal_distr_Dow$estimate[1], : ties should not be present for the one-sample
## Kolmogorov-Smirnov test
```

```r
mu_hat_GSPC <- t_distr_GPSC$estimate["m"]
sd_hat <- t_distr_GPSC$estimate["s"]
df_hat <- t_distr_GPSC$estimate["df"]

standard_log <- (GSPC_adj$log_return-mu_hat_GSPC)/sd_hat
##Ks test for t_dist
ks.ts_GSPC_t <- ks.test(standard_log,"pt",df=df_hat)


mu_hat_DJI <- t_distr_DJI$estimate["m"]
sd_hat_DJ <- t_distr_DJI$estimate["s"]
df_hat_DJI <- t_distr_DJI$estimate["df"]

standard_DJI <- (DJI_adj$log_return-mu_hat_DJI)/sd_hat_DJ

ks.ts_Dow_t <- ks.test(standard_DJI,"pt",df=df_hat_DJI)
```

```
## Warning in ks.test.default(standard_DJI, "pt", df = df_hat_DJI): ties should
## not be present for the one-sample Kolmogorov-Smirnov test
```

```r
## CVM
cvm_test_GSPC_norm <- cvm.test(GSPC_adj$log_return, "pnorm", mean = Normal_distr_GSpc$estimate[1],
                                sd = Normal_distr_GSpc$estimate[2])
cvm_test_DJI_norm <- cvm.test(DJI_adj$log_return, "pnorm", mean = Normal_distr_Dow$estimate[1],
                                sd = Normal_distr_Dow$estimate[2])
```

```r
# CVM Test for t-Distribution
cvm_test_GSPC_t <- cvm.test(standard_log, "pt",df=df_hat)

cvm_test_DJI_t <- cvm.test(standard_DJI, "pt", df =df_hat_DJI)
```

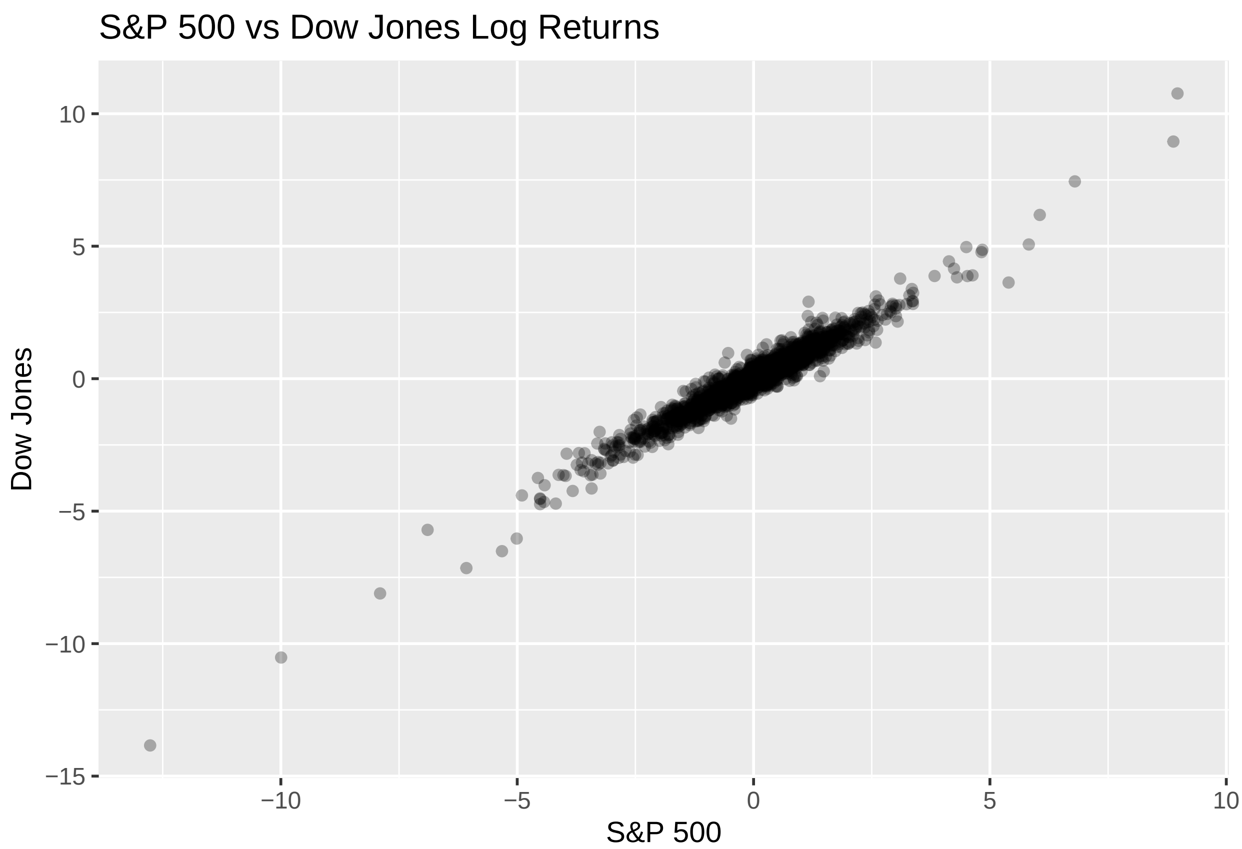
```r
###For normal
MLE_para <- data.frame(index=c("S&P 500","Dow Jone"),
                       Norma_mean=c(Normal_distr_GSpc$estimate[1],Normal_distr_Dow$estimate[1]),
                       Normal_sd=c(Normal_distr_GSpc$estimate[2],Normal_distr_Dow$estimate[2]),T_df=c(t_


Goodness_fit <- data.frame(index=c("S&P 500","Dow jones"),
                           Ks_Norm=c(ks.ts_GSPC_norm$p.value,ks.ts_Dow_norm$p.value),
                           Ks_T=c(ks.ts_GSPC_t$p.value,ks.ts_Dow_t$p.value),
                           CvM_norm=c(cvm_test_GSPC_norm$p.value,cvm_test_DJI_norm$p.value),cvm_t=c(cvm_

print(MLE_para)
```

```
##      index Norma_mean Normal_sd     T_df
## 1  S&P 500 0.03731220  1.125316 2.542262
## 2 Dow Jone 0.03490126  1.091330 2.556896
```

```r
print(Goodness_fit)
```

```
##       index Ks_Norm Ks_T CvM_norm      cvm_t
## 1   S&P 500       0    0        0 0.09654624
## 2 Dow jones       0    0        0 0.16271779
```

```r
# Scatter plot
combined_data<- data.frame(GSPC_adj$log_return,DJI_adj$log_return)
colnames(combined_data)<-c("S&P500","Dow Jones")
ggplot(combined_data,aes(x=`S&P500`,y=`Dow Jones`)) +
  geom_point(alpha = 0.3) +
  labs(title = "S&P 500 vs Dow Jones Log Returns", x = "S&P 500", y = "Dow Jones")
```




```r
# Fit copulas
N<-nrow(combined_data)
rank_SP500<-rank(combined_data$`S&P500`)/(N+1)
rank_DowJones<-(combined_data$`Dow Jones`)/(N+1)
emp_copula <- pobs(cbind(rank_SP500, rank_DowJones))
fit_gaussian <- fitCopula(normalCopula(dim = 2), emp_copula, method = "ml")
fit_clayton <- fitCopula(claytonCopula(dim = 2), emp_copula, method = "ml")
fit_gumbel <- fitCopula(gumbelCopula(dim = 2), emp_copula, method = "ml")
fit_tcopula<-fitCopula(tCopula (dim=2, dispstr = "un"),emp_copula, method="ml")


# Print copula parameters
print(fit_gaussian@estimate)
```

```
## [1] 0.9559643
```

```r
print(fit_clayton@estimate)
```

```
## [1] 8.364749
```

```r
print(fit_gumbel@estimate)
```

```
## [1] 5.07792
```

```r
print(fit_tcopula@estimate)
```

```
## [1] 0.9548318 2.6347075
```

```r
fit_gaussian_ifm <- fitCopula(normalCopula(dim = 2), emp_copula, method = "itau")
fit_clayton_ifm <- fitCopula(claytonCopula(dim = 2), emp_copula, method = "itau")
fit_gumbel_ifm <- fitCopula(gumbelCopula(dim = 2), emp_copula, method = "itau")
fit_tcopula_ifm<-fitCopula(tCopula (dim=2, dispstr = "un"),emp_copula, method="itau")
```

```
## Warning in fitCopula.icor(copula, x = data, method = method, estimate.variance
## = estimate.variance, : "itau" fitting ==> copula coerced to 'df.fixed=TRUE'
```

```r
# Print copula parameters
print(fit_gaussian_ifm@estimate)
```

```
## [1] 0.9544147
```

```r
print(fit_clayton_ifm@estimate)
```

```
## [1] 8.364749
```

```r
print(fit_gumbel_ifm@estimate)
```

```
## [1] 5.182374
```

```r
print(fit_tcopula_ifm@estimate)
```

```
## [1] 0.9544147
```

```r
# Fit using Omnibus Method (OM)
fit_gaussian_om <- fitCopula(normalCopula(dim=2), emp_copula, method = "mpl")
fit_clayton_om <- fitCopula(claytonCopula(dim=2), emp_copula, method = "mpl")
fit_gumbel_om <- fitCopula(gumbelCopula(dim=2), emp_copula, method = "mpl")
fit_t_om <- fitCopula(tCopula(dim=2,dispstr = "un"), emp_copula, method = "mpl")
```

```
## Warning in var.mpl(copula, u): the covariance matrix of the parameter estimates
## is computed as if 'df.fixed = TRUE' with df = 2.63470748182605
```

```r
# Print and compare estimated parameters
  print(fit_gaussian_om@estimate)
```

```
## [1] 0.9559643
```

```r
  print(fit_clayton_om@estimate)
```

```
## [1] 8.364749
```

```r
  print(fit_gumbel_om@estimate)
```

```
## [1] 5.07792
```

```r
  print(fit_t_om@estimate)
```

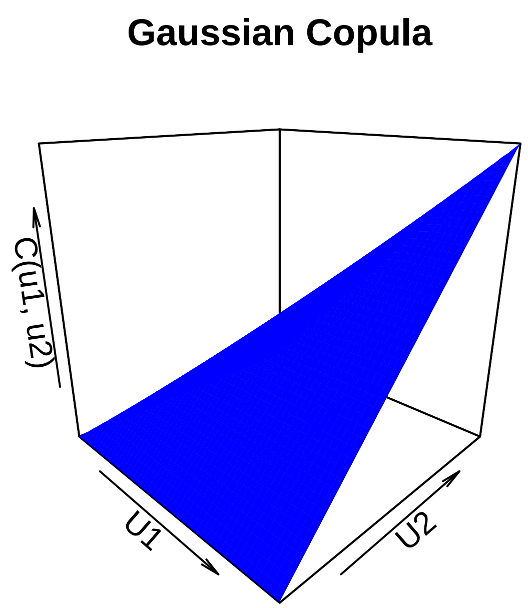
```
## [1] 0.9548318 2.6347075
```

```r
estimate_parameter_table <- data.frame(
  copula = c("gaussian", "clayton", "gumbel", "tcopula"),
  mle_estimate = c(fit_gaussian@estimate, fit_clayton@estimate, fit_gumbel@estimate, NA),  # Handle len
  ifm_estimates = c(fit_gaussian_ifm@estimate, fit_clayton_ifm@estimate, fit_gumbel_ifm@estimate, NA),
  om_estimates = c(fit_gaussian_om@estimate,fit_clayton_om@estimate,fit_gumbel_om@estimate, NA)
)
print(estimate_parameter_table)
```

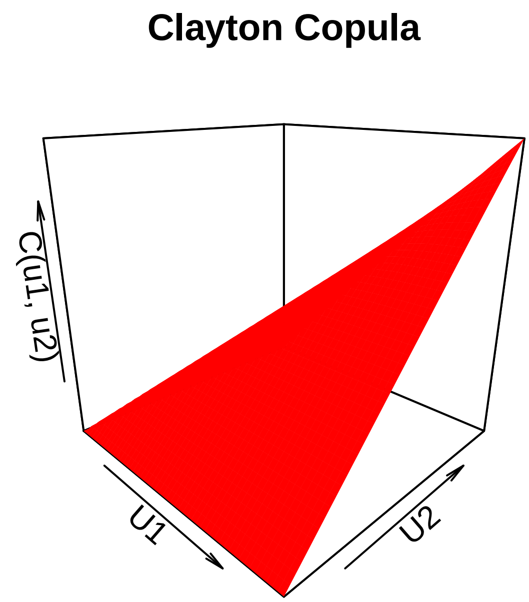
```
##     copula mle_estimate ifm_estimates om_estimates
## 1 gaussian    0.9559643     0.9544147    0.9559643
## 2  clayton    8.3647490     8.3647490    8.3647490
## 3   gumbel    5.0779197     5.1823745    5.0779197
## 4  tcopula           NA            NA           NA
```

```r
# Define grid size for visualization
grid_size <- 50
u_seq <- seq(0.01, 0.99, length.out = grid_size)
v_seq <- seq(0.01, 0.99, length.out = grid_size)

# Function to compute empirical copula surface
copula_surface <- function(copula_model, u_seq, v_seq) {
  surface <- outer(u_seq, v_seq, Vectorize(function(u, v) {
    pCopula(c(u, v), copula_model)
  }))
  return(surface)
}

# Generate surfaces for each copula
gaussian_surface <- copula_surface(fit_gaussian@copula, u_seq, v_seq)
clayton_surface  <- copula_surface(fit_clayton@copula, u_seq, v_seq)
gumbel_surface   <- copula_surface(fit_gumbel@copula, u_seq, v_seq)
#t_surface        <- copula_surface(fit_tcopula@copula, u_seq, v_seq)
```

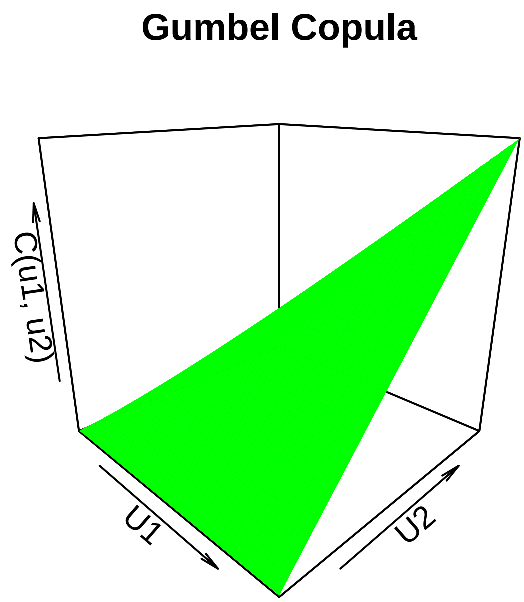
```r
# Plot Gaussian Copula
persp3D(u_seq, v_seq, gaussian_surface, col = "blue", theta = 45, phi = 20,
        main = "Gaussian Copula", xlab = "U1", ylab = "U2", zlab = "C(u1, u2)")
```




```r
# Plot Clayton Copula
# Plot Clayton Copula
persp3D(u_seq, v_seq, clayton_surface, col = "red", theta = 45, phi = 20,
        main = "Clayton Copula", xlab = "U1", ylab = "U2", zlab = "C(u1, u2)")
```

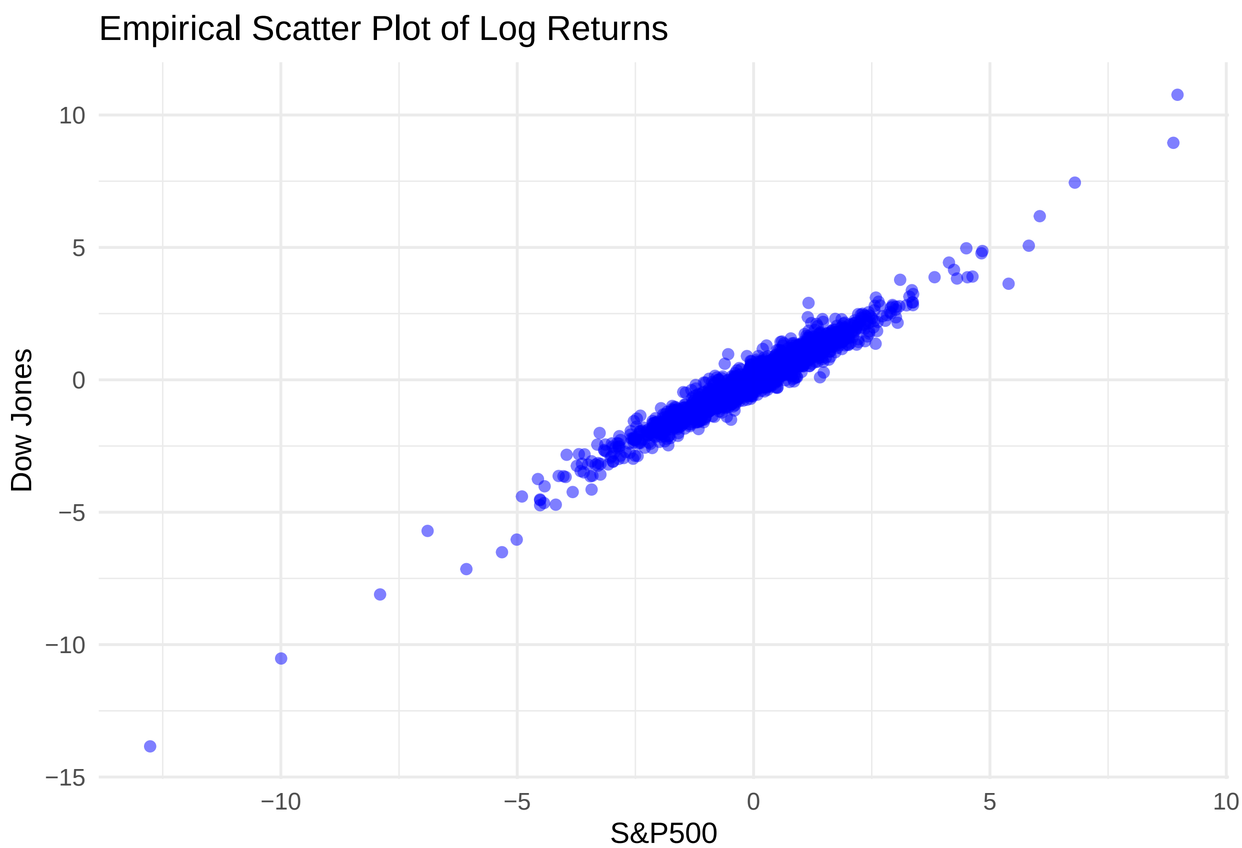



```r
# Plot Gumbel Copula
persp3D(u_seq, v_seq, gumbel_surface, col = "green", theta = 45, phi = 20,
        main = "Gumbel Copula", xlab = "U1", ylab = "U2", zlab = "C(u1, u2)")
```

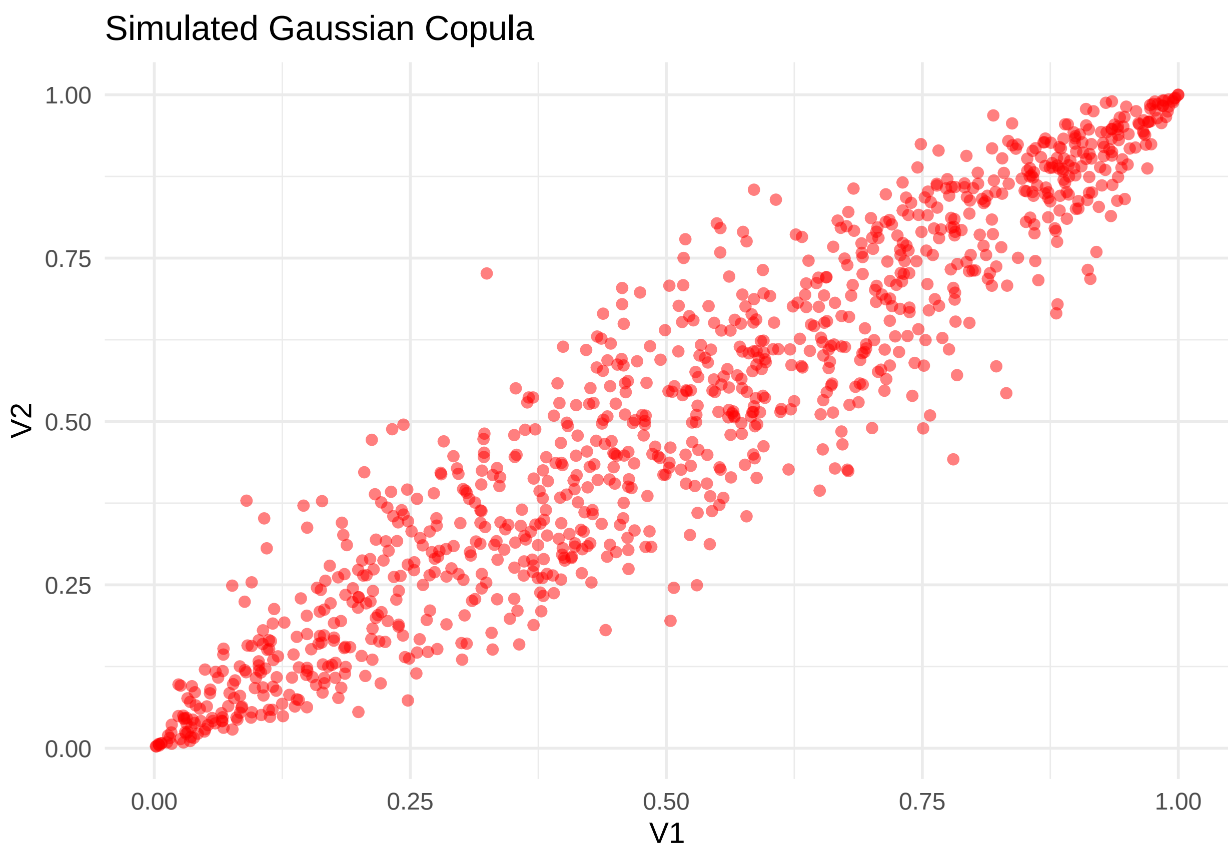



```r
# Set seed for reproducibility
set.seed(123)

# Generate 1000 random samples from each fitted copula
sim_gaussian <- rCopula(1000, fit_gaussian@copula)
sim_clayton  <- rCopula(1000, fit_clayton@copula)
sim_gumbel   <- rCopula(1000, fit_gumbel@copula)
#sim_t        <- rCopula(1000, fit_t@copula)

# Convert to data frames
sim_data <- list(
  Gaussian = as.data.frame(sim_gaussian),
  Clayton  = as.data.frame(sim_clayton),
  Gumbel   = as.data.frame(sim_gumbel)
  #T_Copula = as.data.frame(sim_t)
)
```

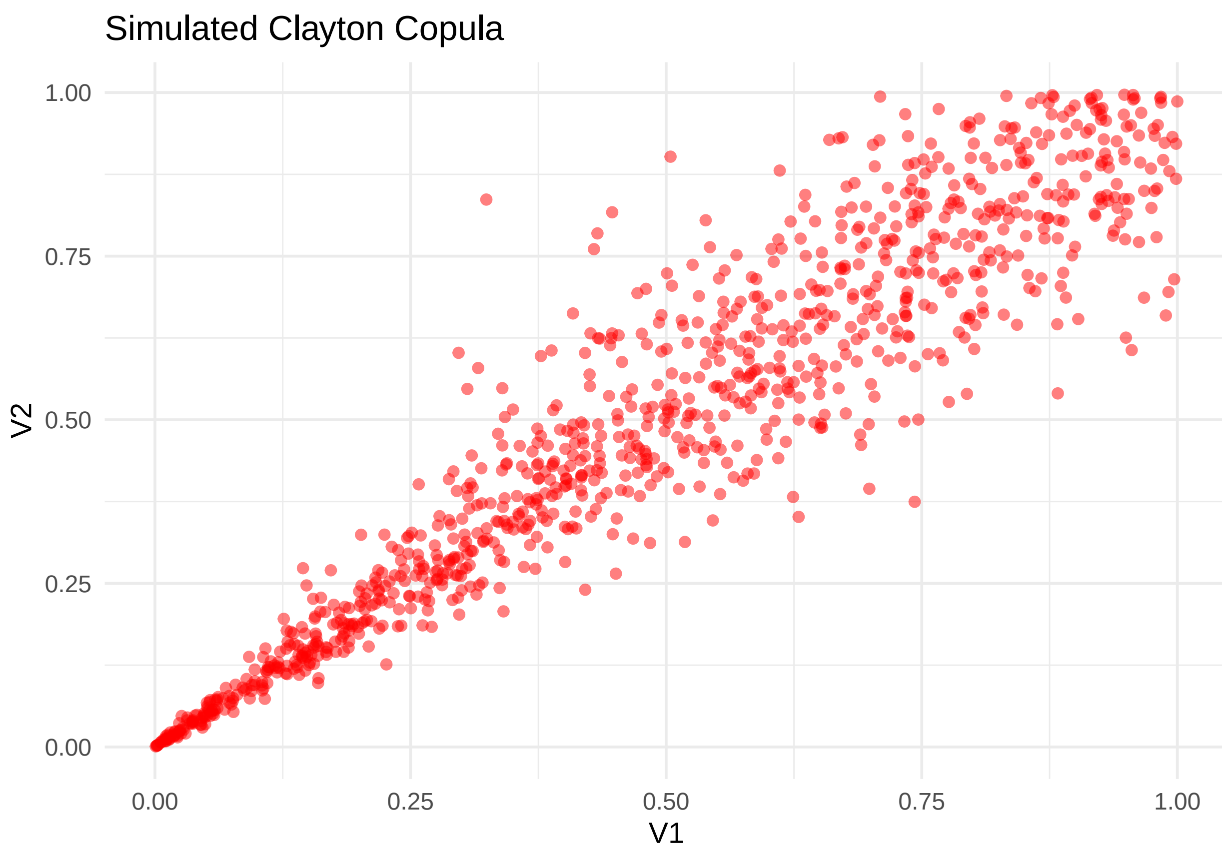
```r
# Scatter plot of actual data
ggplot(combined_data, aes(x = `S&P500`, y = `Dow Jones`)) +
  geom_point(alpha = 0.5, color = "blue") +
  ggtitle("Empirical Scatter Plot of Log Returns") +
  theme_minimal()
```



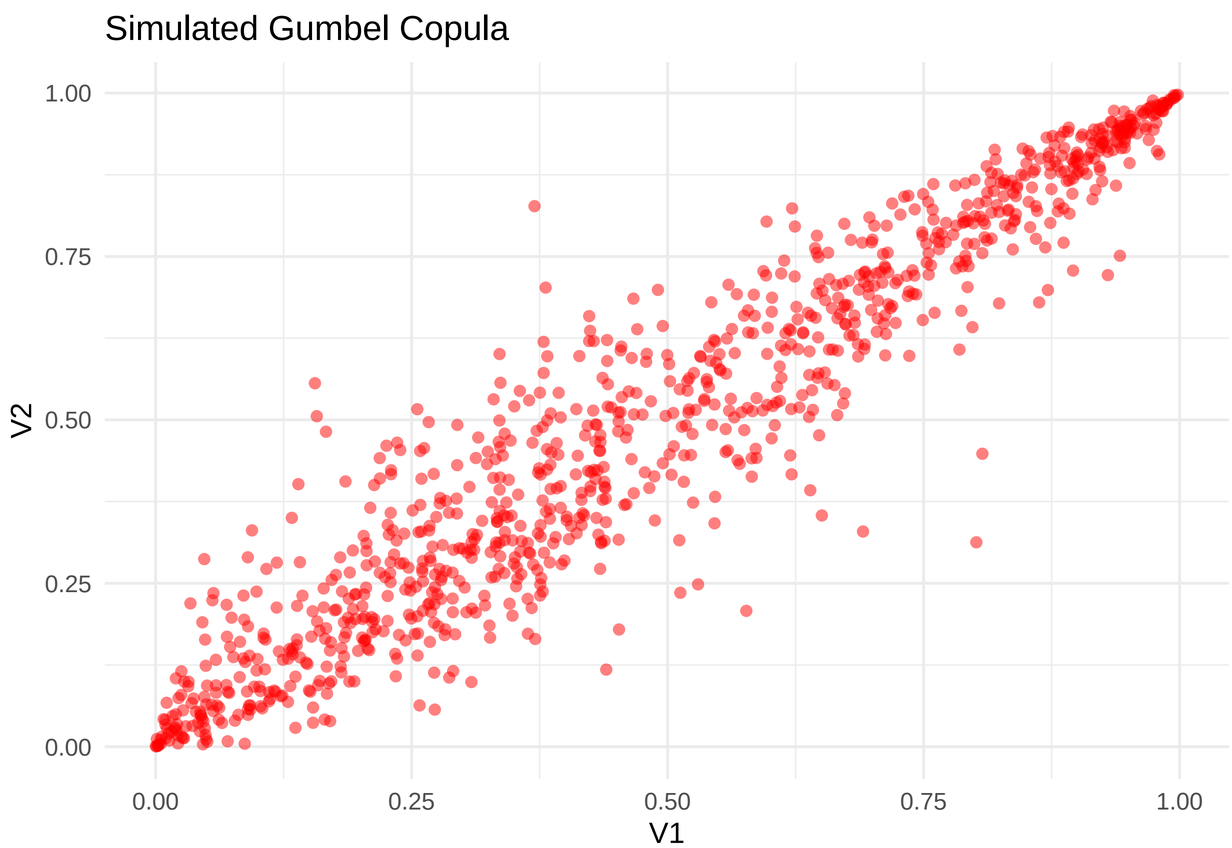

```r
# Function to plot simulated data
plot_simulated <- function(data, title) {
  ggplot(data, aes(x = V1, y = V2)) +
    geom_point(alpha = 0.5, color = "red") +
    ggtitle(title) +
    theme_minimal()
}

# Plot simulated scatter plots
plot_simulated(sim_data$Gaussian, "Simulated Gaussian Copula")
```




```
plot_simulated(sim_data$Clayton, "Simulated Clayton Copula")
```




```
plot_simulated(sim_data$Gumbel, "Simulated Gumbel Copula")
```




```r
#plot_simulated(sim_data$T_Copula, "Simulated T-Copula")
```

```r
# Compute Log-Likelihood values
loglik_values <- data.frame(
  Copula = c("Gaussian", "Clayton", "Gumbel", "T-Copula"),
  LogLikelihood = c(
    logLik(fit_gaussian),
    logLik(fit_clayton),
    logLik(fit_gumbel),
   NA
  )
)

# Print table
library(knitr)
kable(loglik_values, caption = "Log-Likelihood Values for Copula Models")
```

Table 1: Log-Likelihood Values for Copula Models

Copula	LogLikelihood
Gaussian	4003.680
Clayton	3385.342
Gumbel	3942.990
T-Copula	NA