Poster project

Christoffer Mondrup Kramer

2023-05-24

Poster project

I will do a toy example of my poster project.

Step 1: Describe data

What kind of data are we dealing with? What are categorical, what is numerical, what are the attributes, what do we want to classify?

```
full_df <- read.table("T11-9.dat")
full_df$V8 <- as.numeric(as.character(full_df$V8))
```

```
## Warning: NAs introduced by coercion
```

```
full_df
```

```
##                         V1 V2  V3 V4 V5  V6  V7   V8 V9 V10 V11
## 1               ACCheerios  G 110  2  2 180 1.5 10.5 10  70   1
## 2                 Cheerios  G 110  6  2 290 2.0 17.0  1 105   1
## 3               CocoaPuffs  G 110  1  1 180 0.0 12.0 13  55   1
## 4             CountChocula  G 110  1  1 180 0.0 12.0 13  65   1
## 5            GoldenGrahams  G 110  1  1 280 0.0 15.0  9  45   1
## 6         HoneyNutCheerios  G 110  3  1 250 1.5 11.5 10  90   1
## 7                      Kix  G 110  2  1 260 0.0 21.0  3  40   1
## 8              LuckyCharms  G 110  2  1 180 0.0 12.0 12  55   1
## 9       MultiGrainCheerios  G 100  2  1 220 2.0 15.0  6  90   1
## 10      OatmealRaisinCrisp  G 130  3  2 170 1.5 13.5 10 120   1
## 11           RaisinNutBran  G 100  3  2 140 2.5   NA  8 140   1
## 12         TotalCornFlakes  G 110  2  1 200 0.0 21.0  3  35   1
## 13         TotalRaisinBran  G 140  3  1 190 4.0 15.0 14 230   1
## 14         TotalWholeGrain  G 100  3  1 200 3.0 16.0  3 110   1
## 15                    Trix  G 110  1  1 140 0.0 13.0 12  25   1
## 16                Cheaties  G 100  3  1 200 3.0 17.0  3 110   1
## 17       WheatiesHoneyGold  G 110  2  1 200 1.0 16.0  8  60   1
## 18                 AllBran  K  70  4  1 260 9.0  7.0  5 320   2
## 19              AppleJacks  K 110  2  0 125 1.0 11.0 14  30   2
## 20              CornFlakes  K 100  2  0 290 1.0 21.0  2  35   2
## 21                CornPops  K 110  1  0  90 1.0 13.0 12  20   2
## 22         CracklinOatBran  K 110  3  3 140 4.0 10.0  7 160   2
## 23                 Crispix  K 110  2  0 220 1.0 21.0  3  30   2
## 24              FrootLoops  K 110  2  1 125 1.0 11.0 13  30   2
## 25           FrostedFlakes  K 110  1  0 200 1.0 14.0 11  25   2
## 26       FrostedMiniWheats  K 100  3  0   0 3.0 14.0  7 100   2
## 27            FruitfulBran  K 120  3  0 240 5.0 14.0 12 190   2
## 28 JustRightCrunchyNuggets  K 110  2  1 170 1.0 17.0  6  60   2
## 29      MueslixCrispyBlend  K 160  3  2 150 3.0 17.0 13 160   2
## 30         NutNHoneyCrunch  K 120  2  1 190 0.0 15.0  9  40   2
## 31  NutriGrainAlmondRaisin  K 140  3  2 220 3.0 21.0  7 130   2
## 32         NutriGrainWheat  K  90  3  0 170 3.0 18.0  2  90   2
## 33               Product19  K 100  3  0 320 1.0 20.0  3  45   2
## 34              RaisinBran  K 120  3  1 210 5.0 14.0 12 240   2
## 35            RiceKrispies  K 110  2  0 290 0.0 22.0  3  35   2
## 36                  Smacks  K 110  2  1  70 1.0  9.0 15  40   2
## 37                SpecialK  K 110  6  0 230 1.0 16.0  3  55   2
## 38              CapNCrunch  Q 120  1  2 220 0.0 12.0 12  35   3
## 39          HoneyGrahamOhs  Q 120  1  2 220 1.0 12.0 11  45   3
## 40                    Life  Q 100  4  2 150 2.0 12.0  6  95   3
## 41              PuffedRice  Q  50  1  0   0 0.0 13.0  0  15   3
## 42             PuffedWheat  Q  50  2  0   0 1.0 10.0  0  50   3
## 43           QuakerOatmeal  Q 100  5  2   0 2.7  1.0  1 110   3
```

The dataset contains the following information:

- V1: Cereal brand
- Type: string
- V2: Manufacturer
- Type: Categorical String
- V3: Calories
- Type: Numerical
- V4: Protein
- Type: Numerical

- V5: Fat
- Type: Numerical
- V6: Sodium
- Type: Numerical
- V7: Fiber
- Type: Numerical
- V8: Carbohydrates
- Type: Numerical
- V9: Sugar
- Type: Numerical
- V10: Potasium
- Type: Numerical
- V11: Group
- Type: Numerical categorical

The first step will be to remove V1, V2, and V11, since V2 is not interesting for our purposes, and V1 or V2 will be our classification variable (here it will probably be V11). Moreover V8 needs to be converted to numerical

```
df <- full_df[, -c(1, 2, 11)]
df$V8 <- as.numeric(as.character(df$V8))
df <- df[-11,] # contains NA
df
```

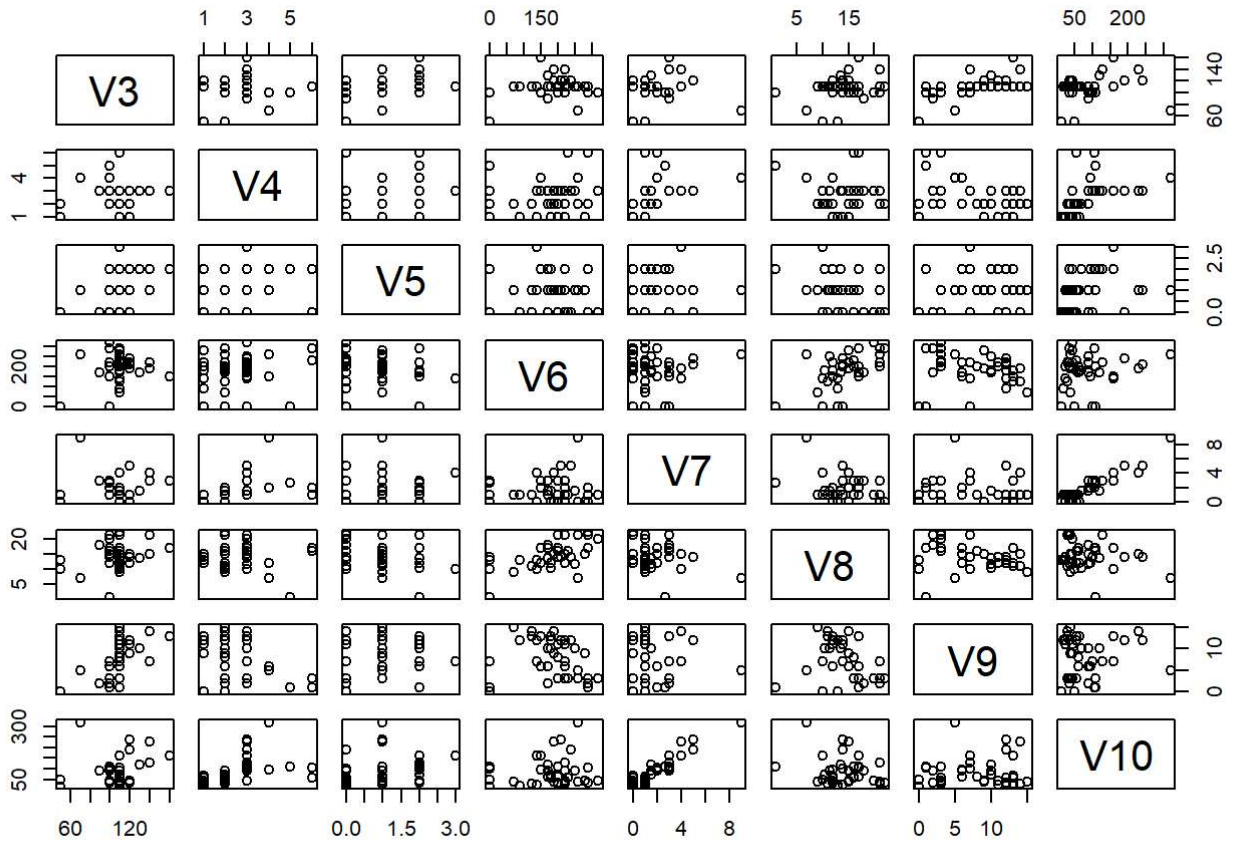
```
##     V3 V4 V5  V6  V7   V8 V9 V10
## 1  110  2  2 180 1.5 10.5 10  70
## 2  110  6  2 290 2.0 17.0  1 105
## 3  110  1  1 180 0.0 12.0 13  55
## 4  110  1  1 180 0.0 12.0 13  65
## 5  110  1  1 280 0.0 15.0  9  45
## 6  110  3  1 250 1.5 11.5 10  90
## 7  110  2  1 260 0.0 21.0  3  40
## 8  110  2  1 180 0.0 12.0 12  55
## 9  100  2  1 220 2.0 15.0  6  90
## 10 130  3  2 170 1.5 13.5 10 120
## 12 110  2  1 200 0.0 21.0  3  35
## 13 140  3  1 190 4.0 15.0 14 230
## 14 100  3  1 200 3.0 16.0  3 110
## 15 110  1  1 140 0.0 13.0 12  25
## 16 100  3  1 200 3.0 17.0  3 110
## 17 110  2  1 200 1.0 16.0  8  60
## 18  70  4  1 260 9.0  7.0  5 320
## 19 110  2  0 125 1.0 11.0 14  30
## 20 100  2  0 290 1.0 21.0  2  35
## 21 110  1  0  90 1.0 13.0 12  20
## 22 110  3  3 140 4.0 10.0  7 160
## 23 110  2  0 220 1.0 21.0  3  30
## 24 110  2  1 125 1.0 11.0 13  30
## 25 110  1  0 200 1.0 14.0 11  25
## 26 100  3  0   0 3.0 14.0  7 100
## 27 120  3  0 240 5.0 14.0 12 190
## 28 110  2  1 170 1.0 17.0  6  60
## 29 160  3  2 150 3.0 17.0 13 160
## 30 120  2  1 190 0.0 15.0  9  40
## 31 140  3  2 220 3.0 21.0  7 130
## 32  90  3  0 170 3.0 18.0  2  90
## 33 100  3  0 320 1.0 20.0  3  45
## 34 120  3  1 210 5.0 14.0 12 240
## 35 110  2  0 290 0.0 22.0  3  35
## 36 110  2  1  70 1.0  9.0 15  40
## 37 110  6  0 230 1.0 16.0  3  55
## 38 120  1  2 220 0.0 12.0 12  35
## 39 120  1  2 220 1.0 12.0 11  45
## 40 100  4  2 150 2.0 12.0  6  95
## 41  50  1  0   0 0.0 13.0  0  15
## 42  50  2  0   0 1.0 10.0  0  50
## 43 100  5  2   0 2.7  1.0  1 110
```

Step 2: Descriptive statistics

Scatter plot

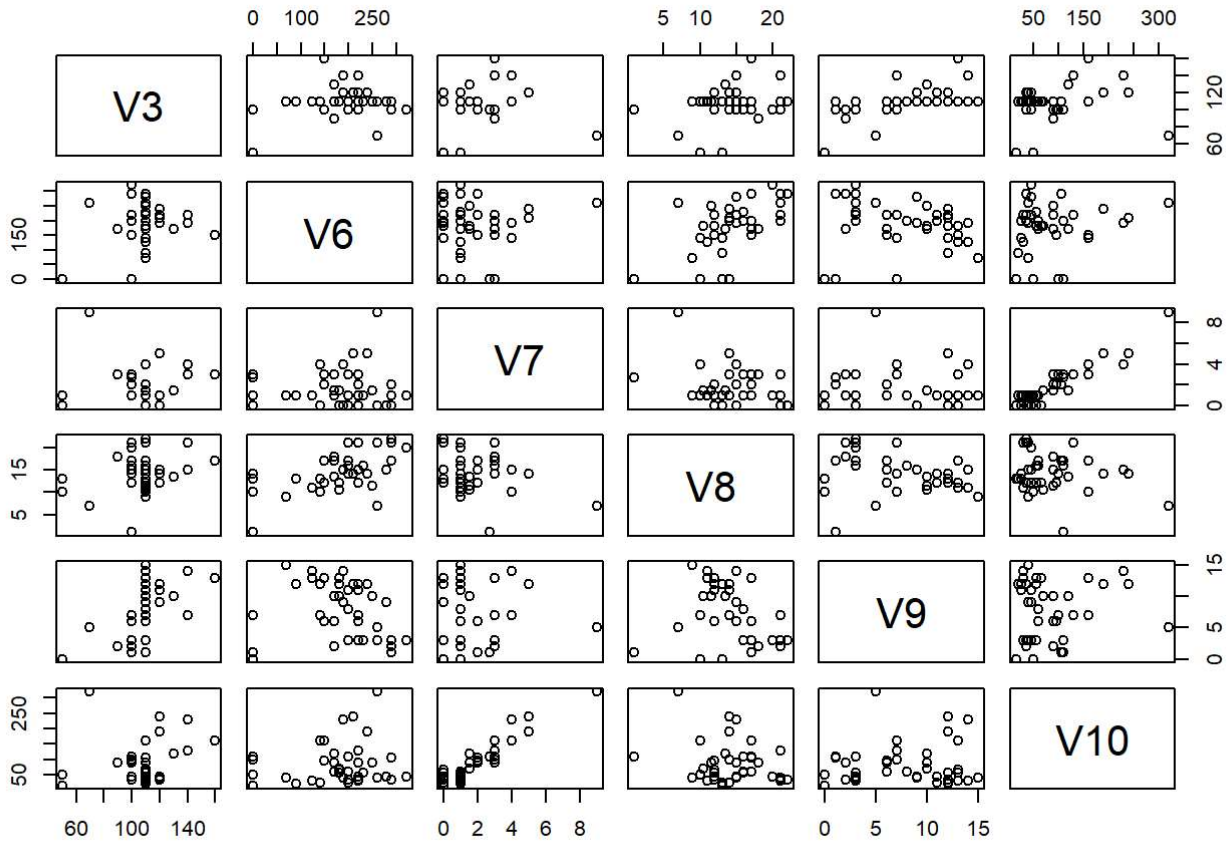
Before we start with anything let us make a scatterplot of the pairs:

```
pairs(df)
```



As the scatterplot above shows, V4 and V5 looks a lot like categorical variables, they might need to be excluded:

```
df <- df[, -c(2, 3)]
pairs(df)
```



At a quick glance some of the variables might be correlated:
- V7 and V10
- v6 and V8
- V6 and V9
- V3 and V10

Mean, variance, co-variance

Let us get some simple stats:

```
#mean(full_df)
colMeans(df)
```

```
##         V3         V6         V7         V8         V9        V10
## 108.095238 181.428571   1.695238  14.345238   7.595238  83.095238
```

```
cov(df)
```

```
##              V3         V6          V7         V8          V9        V10
## V3  367.0150987  510.10453  -0.5214866  20.429733  51.1614402  192.62485
## V6  510.1045296 6386.93380   7.1289199 192.177700 -17.5783972  669.86063
## V7   -0.5214866    7.12892   3.3009524  -1.815389  -0.2946574  112.08827
## V8   20.4297329  192.17770  -1.8153891  18.213269  -6.1983159  -59.14344
## V9   51.1614402  -17.57840  -0.2946574  -6.198316  21.0760743   24.45412
## V10 192.6248548  669.86063 112.0882695 -59.143438  24.4541231 4399.94193
```

let us also check the generalized variance:

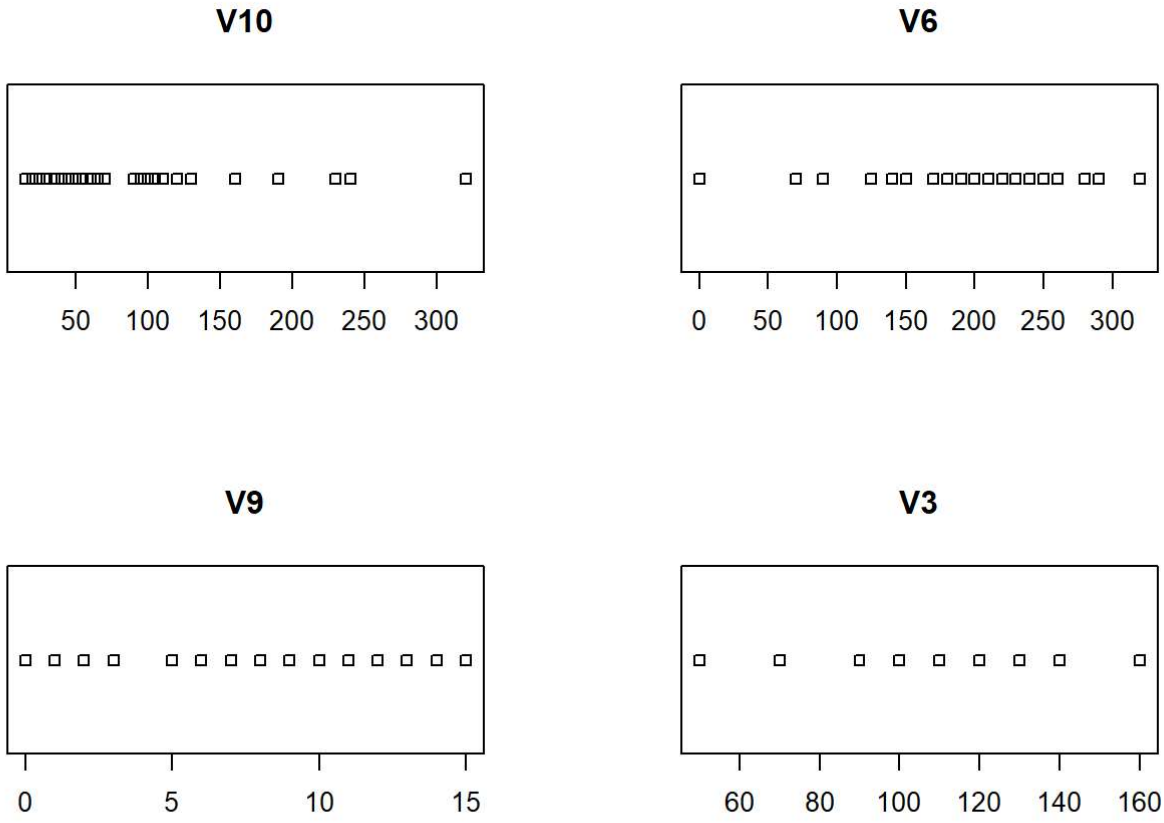
```
sum(diag(cov(df)))
```

```
## [1] 11196.48
```

This is pretty high, but not something that is unexpected given, that some variabels have a high variance.

Okay, we can now conclude that no variable appears to be completed unrelated, since none are 0. Moreover, V10, V6 have a very high variance, but a low mean, which might indicate, that there are some crazy outliers, the same goes for V3 and V9 to a lesser extend. Let's check them:

```
par(mfrow = c(2, 2))
stripchart(df$V10, main = "V10")
stripchart(df$V6, main = "V6")
stripchart(df$V9, main = "V9")
stripchart(df$V3, main = "V3")
```



Yep we need to keep an eye out for V10's outliers, V6 is not that bad, but still has an outlier. V3 and V9 looks good however.

Now let's check there correlation and let us keep an eye out on those we saw at the scatter plot:

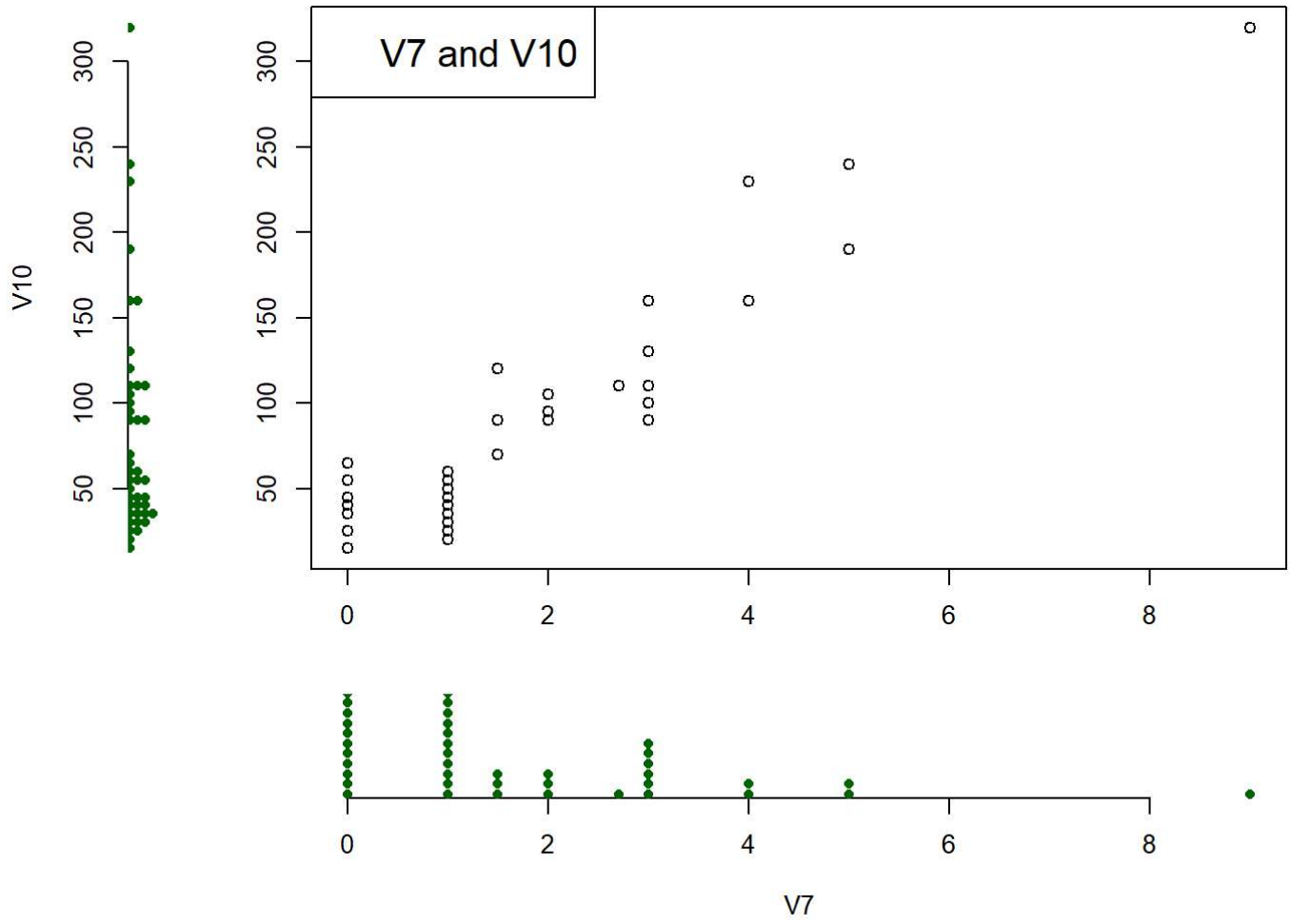
```
cor(df)
```

```
##              V3          V6          V7         V8          V9         V10
## V3   1.00000000  0.33317393 -0.01498241  0.2498773  0.58170920  0.15158174
## V6   0.33317393  1.00000000  0.04909733  0.5634597 -0.04791133  0.12636150
## V7  -0.01498241  0.04909733  1.00000000 -0.2341295 -0.03532668  0.93007305
## V8   0.24987728  0.56345966 -0.23412950  1.0000000 -0.31636242 -0.20892439
## V9   0.58170920 -0.04791133 -0.03532668 -0.3163624  1.00000000  0.08030337
## V10  0.15158174  0.12636150  0.93007305 -0.2089244  0.08030337  1.00000000
```

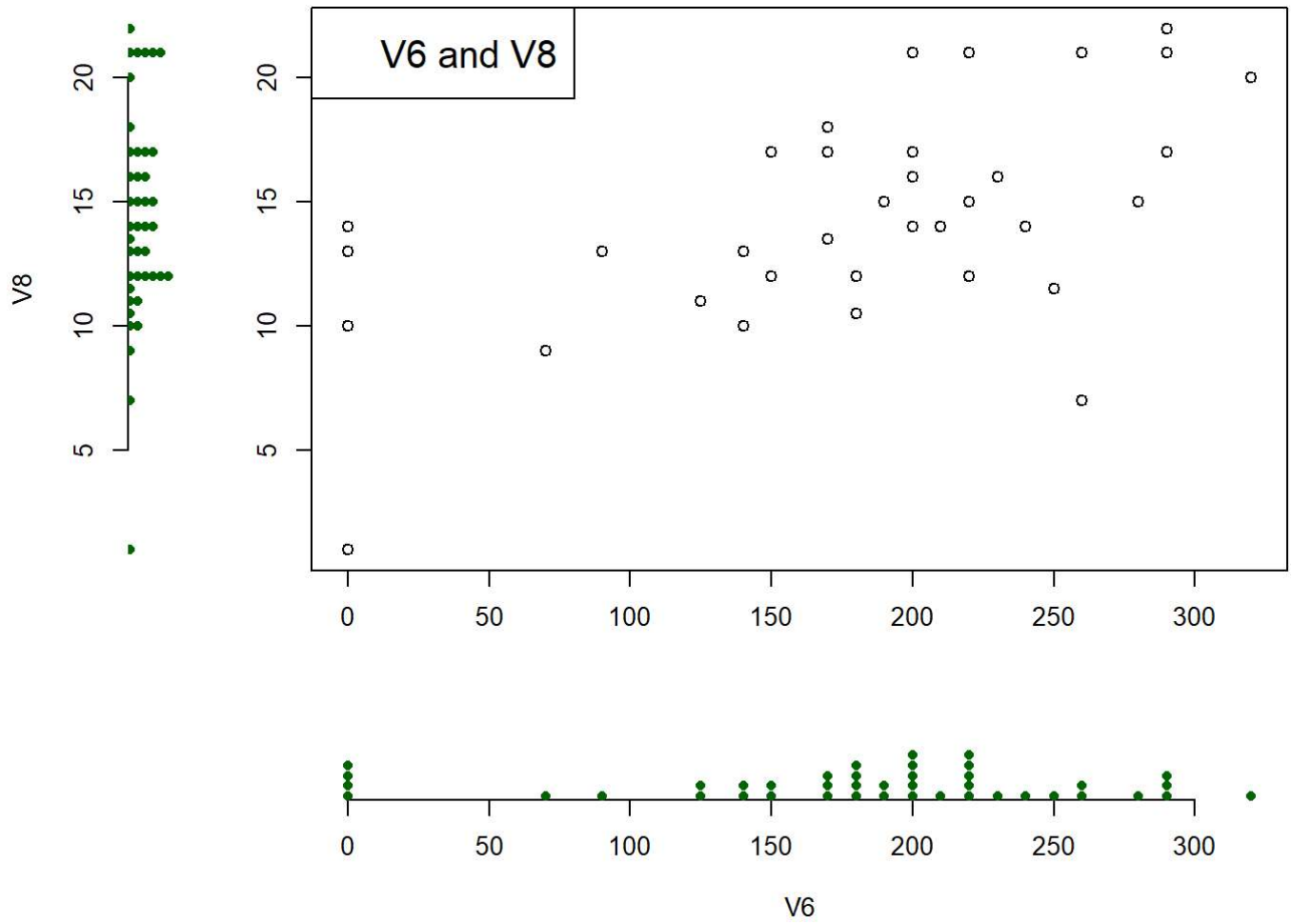
V7 and V10: 0.93 (Very strong)
V6 and V8: 0.56 (Strong)
V3 and V9: 0.58 (strong)
Let us plot these together

```
margin_dot_plot <- function(x, y, xlabel = "x", ylabel = "y", name = "") {
layout(mat = matrix(c(2, 0, # First column
                      1, 3), # Second column
                        nrow = 2,
                        ncol = 2),
       heights = c(6, 2),    # Heights of the two rows
       widths = c(1, 6))     # Widths of the two columns
par(mar = c(4, # Bottom
            4, # left
            0.1, # Top
            0.1))# Right

plot(x, y, xlab = "", ylab = "")
legend("topleft",
       name,
       cex = 1.5)

stripchart(y, method = "stack", at = 0,
           pch = 16, col = "darkgreen", frame = FALSE, vertical = TRUE, ylab = ylabel)
stripchart(x, method = "stack", at = 0,
           pch = 16, col = "darkgreen", frame = FALSE, xlab = xlabel)
}
```

```
margin_dot_plot(df$V7, df$V10, xlabel = "V7", ylabel = "V10", name = "V7 and V10")
```




```
margin_dot_plot(df$V6, df$V8, xlabel = "V6", ylabel = "V8", "V6 and V8")
```



V7 and V10 seems to highly correlated and so does V6 and V8.

Conclusion

V7 and V10 have a strong correlation, and the same goes for V6 and V8. Moreover V10 and V6 appears to have some large outliers, due to a high variance and low mean, which might need to be removed. Moreover, wether intended or not, V4 and V5 appears to behave more like categorical rather than numerical nature and they have been excluded.

Step 3: Check for normality

Each attribute

We start by checking the normality of each attribute:

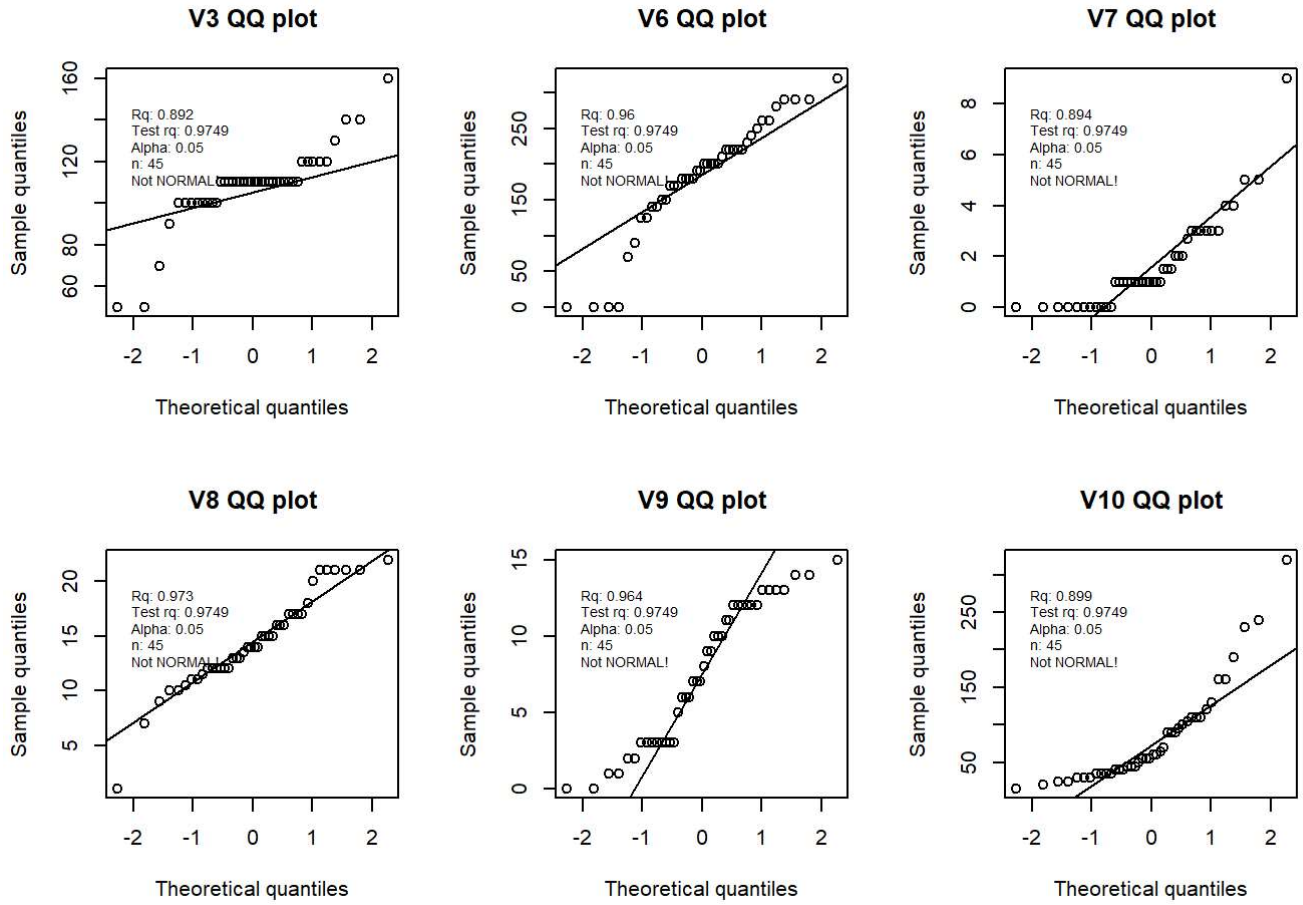
```
library(MASS)

get_best_lambda <- function(data_vector, name = "")  {
    # Transform data
    # Make box cox plot on our data
    df_box_cox<- boxcox(data_vector~1,lambda=seq(-2, 2, 1/10))# Can also be seq(-.5, 1.5,.01)

    # Get best lambda
    max_lambda <- df_box_cox$x[which.max(df_box_cox$y)]

    legend("left",
           paste0(name, " Best lambda", "\n",
                  "The best lambda is ", round(max_lambda, 3), "\n"),
           cex = 0.75,
           bty = "n")
    return(max_lambda)
}

box_cox_transformation <- function(data_vector, name = ""){

  lambda <- get_best_lambda(data_vector, name)
  if (lambda == 0){
    transformed_vector <- log(data_vector)
  }

  else {
  transformed_vector <- ( (data_vector^lambda) - 1)/lambda
  }

  return(transformed_vector)
}


test_norm <- function(data_vector, name = " ", signigicance = 0.05) {
  # You can chose the following significance levels
  # 0.01
  # 0.05
  # 0.10

  if (signigicance == 0.01){
    signigicance_col <- 2
  }

  else if (signigicance == 0.05){
    signigicance_col <- 3
  }

  else if (signigicance == 0.1){
    signigicance_col <- 4
  }


  # ----- QQ plot -----
  qq <- qqnorm(data_vector, plot.it = F)
```

```
  # ----- Hypothesis test ------
  # Create Testing table
  n <- c(5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 75, 100, 150, 200, 300)
  one <- c(0.8299, 0.8801, 0.9126, 0.9269, 0.9410,
        0.9479, 0.9538, 0.9599, 0.9632, 0.9671,
        0.9695, 0.9720, 0.9771, 0.9822, 0.9879, 0.9905, 0.9935)

  five <- c(0.8788, 0.9198, 0.9389,
        0.9508, 0.9591, 0.9652,
        0.9682, 0.9726, 0.9749,
        0.9768, 0.9787, 0.9801,
        0.9838, 0.9873, 0.9913, 0.9931, 0.9953)

  ten <- c(0.9032, 0.9351, 0.9503,
        0.9604, 0.9665, 0.9715,
        0.9740, 0.9771, 0.9792,
        0.9809, 0.9822, 0.9836,
        0.9866, 0.9895, 0.9928, 0.9942, 0.9960)
  testing_tbl <- data.frame(
  n,
  one,
  five,
  ten
  )

  # Find index of testing (n)
  sample_size <- length(data_vector)
  i <- 1
  prev_value = NaN
  for (n in testing_tbl$n){

    if (sample_size > testing_tbl$n[length(testing_tbl$n)]){
      i <- length(testing_tbl$n)
      break
    }

    if (n == sample_size){
      exact_sample_size <- TRUE
      break
    }

    else if ( n > sample_size & prev_value < sample_size){
      break
    }
    i <- i + 1
    prev_value <- n
  }

  # ----- Normal -------
  cor_coef <- cor(qq$x, qq$y)
  normality = FALSE
  if (cor_coef >= testing_tbl[i, signigicance_col]){
    plot(qq, xlab = "Theoretical quantiles", ylab = "Sample quantiles", main = paste0(name, "
QQ plot"))
    qqline(data_vector)
    legend("topleft",
```

```
            paste0("Rq: ", round(cor_coef, 3), "\n",
                   "Test rq: ", testing_tbl[i, signigicance_col], "\n",
                   "Alpha: ", signigicance, "\n",
                   "n: ", n, "\n",
                   "NORMAL!"),
            cex = 0.65,
            bty = "n")
    normality = TRUE
    return(normality)
  }

  # ----- Not normal -----
  else {

    plot(qq, xlab = "Theoretical quantiles", ylab = "Sample quantiles", main = paste0(name, "
QQ plot"))
    qqline(data_vector)
    legend("topleft",
            paste0("Rq: ", round(cor_coef, 3), "\n",
                   "Test rq: ", testing_tbl[i, signigicance_col], "\n",
                   "Alpha: ", signigicance, "\n",
                   "n: ", n, "\n",
                   "Not NORMAL!"),
            cex = 0.65,
            bty = "n")
  }
}
```

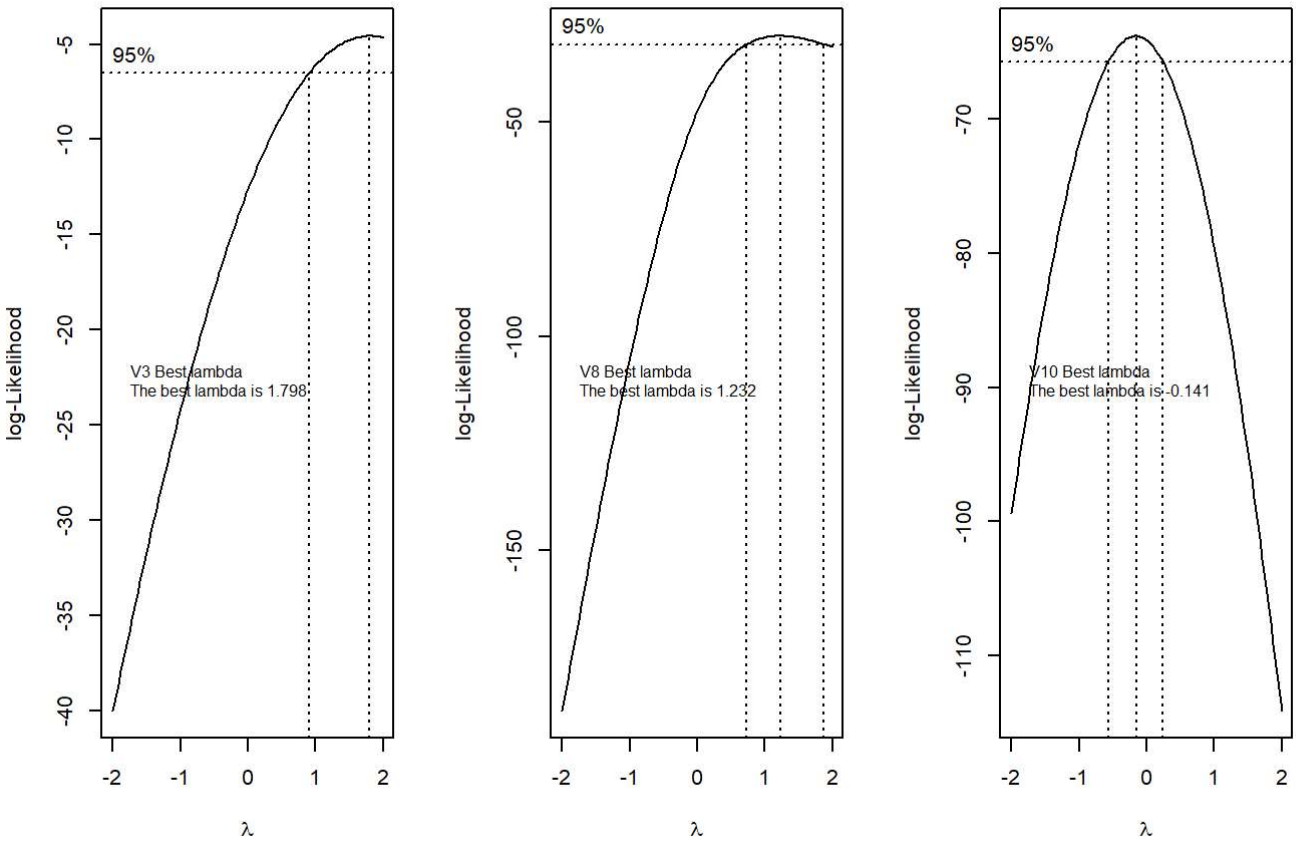
Now apply above function to each attribute

```
par(mfrow = c(2, 3))
i = 1
for (colname in colnames(df)) {
  test_norm(df[, i], name = colname)
  i = i + 1
}
```



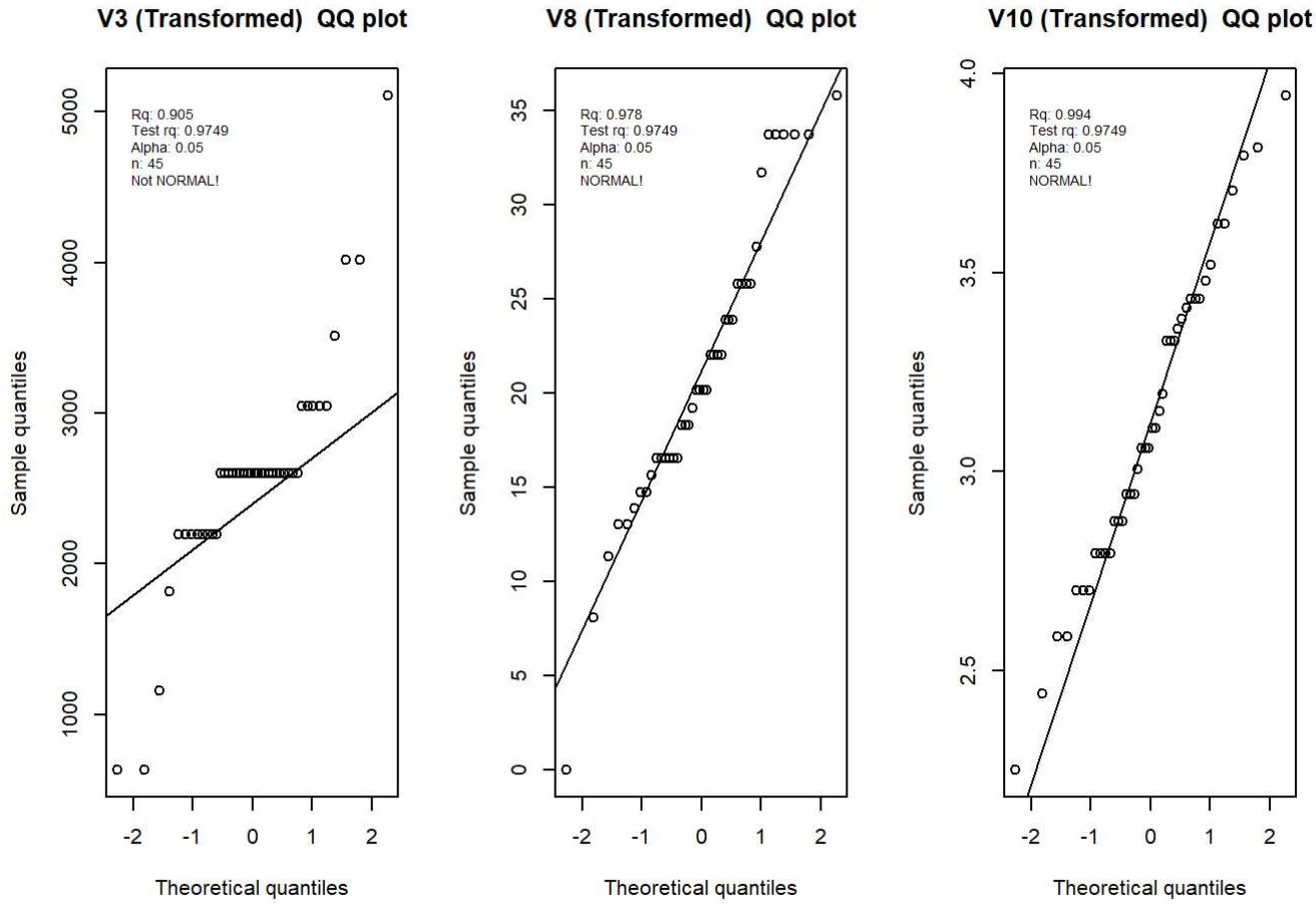
None of the data is normally distributed within alpha = 0.05, let us transform it then:
Now since the values must be positive, then we need to remove V6, V7, V9:

```
par(mfrow= c(1, 3))
z_df <- df[, c(1, 4, 6)]
i = 1
for (colname in colnames(z_df)){
  z_df[, i] <- box_cox_transformation(z_df[, i], name = colname)
  i = i + 1
}
```



Let us now test for normality:

```
i = 1
par(mfrow= c(1, 3))
for (colname in colnames(z_df)){
  test_norm(z_df[, i], name = paste0(colname, " (Transformed) "))
  i = i + 1
}
```



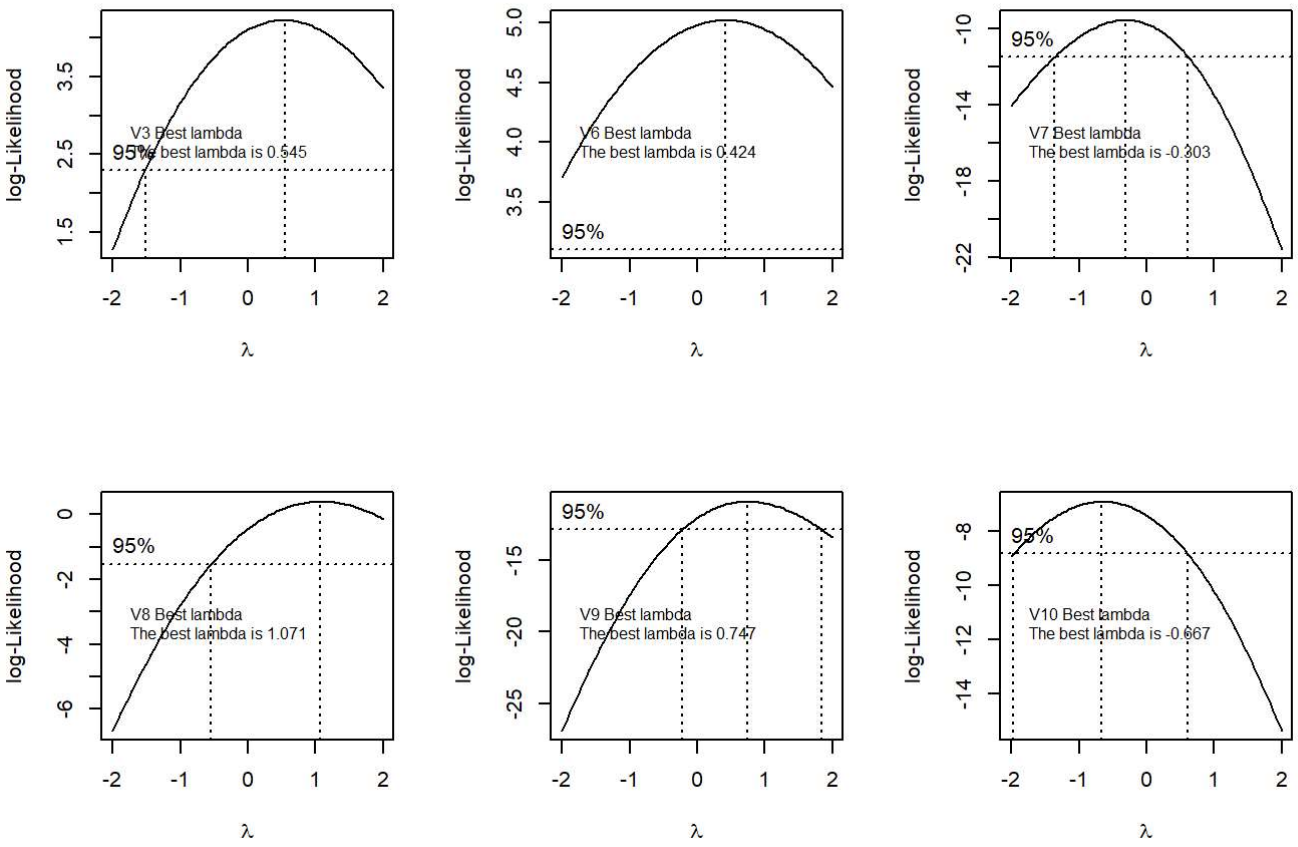
V8 and V10 are normal if we transform them, therefore, we either have to proceed with these two alone or use non normal transformations.
Howevever, we can also try to exlude the non-positive from V6, V7, V9

```
z_df <- df

i = 1
for (colname in colnames(df)){
  z_df <- subset(z_df, z_df[,i] > 1)
  i = i + 1
}
z_df
```

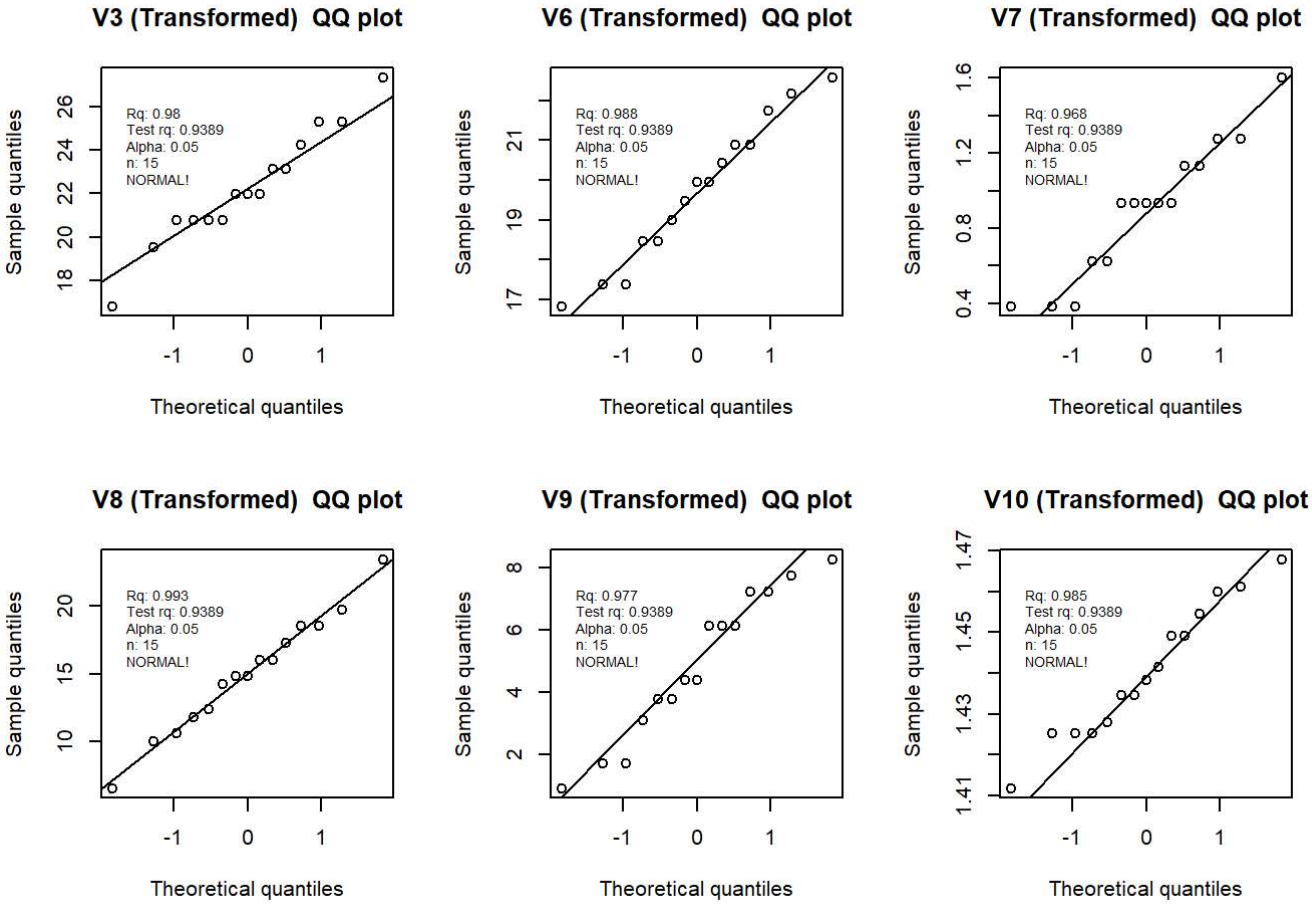
```
##     V3  V6  V7   V8 V9 V10
## 1  110 180 1.5 10.5 10  70
## 6  110 250 1.5 11.5 10  90
## 9  100 220 2.0 15.0  6  90
## 10 130 170 1.5 13.5 10 120
## 13 140 190 4.0 15.0 14 230
## 14 100 200 3.0 16.0  3 110
## 16 100 200 3.0 17.0  3 110
## 18  70 260 9.0  7.0  5 320
## 22 110 140 4.0 10.0  7 160
## 27 120 240 5.0 14.0 12 190
## 29 160 150 3.0 17.0 13 160
## 31 140 220 3.0 21.0  7 130
## 32  90 170 3.0 18.0  2  90
## 34 120 210 5.0 14.0 12 240
## 40 100 150 2.0 12.0  6  95
```

Let us then transform:

```
par(mfrow= c(2, 3))
i = 1
for (colname in colnames(z_df)){
  z_df[, i] <- box_cox_transformation(z_df[, i], name = colname)
  i = i + 1
}
```



```
i = 1
par(mfrow= c(2, 3))
for (colname in colnames(z_df)){
  test_norm(z_df[, i], name = paste0(colname, " (Transformed) "))
  i = i + 1
}
```



Now they are all normal. So if we want a normal classifer we need to both exclude zero variables from the data set and transform it, other we need a non-normal classifier. Let us continue with the transformed data and see if it is multivariate normal:

Attribute pairs

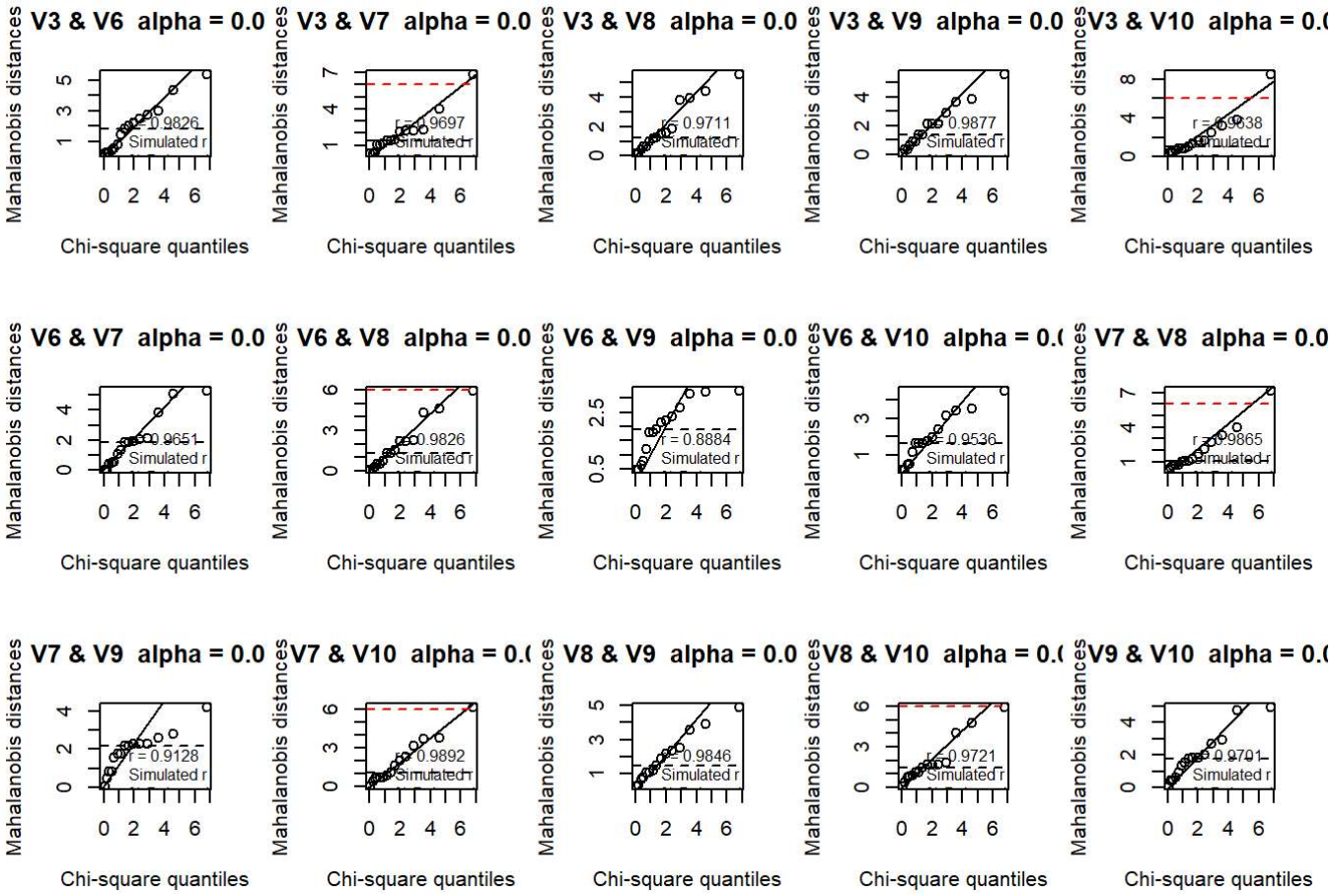
We will test bivariate normality for each pair:

```
FindcrikChi <- function(n=30, p=2, alpha= 0.05, n_simulations=10000){

    cricvec <- rep(0, n_simulations)  #vector for the rQ result collection#

    for(i in 1:n_simulations){
        #iteration to estimate rQ#
        numvec <- rchisq(n, p)  #generate a data set of size n, degree of freedom=p#
        d <- sort(numvec)
        q <- qchisq((1:n-0.5)/n, p)
        cricvec[i] <- cor(d,q)
    }

    scricvec <- sort(cricvec)
    cN <- ceiling(n_simulations* alpha) #to be on the safe side I use ceiling instead of floo
r(), take the 'worst' alpha*N cor as rQ, everything lower than that is deemed as rejection#
    cricvalue <- scricvec[cN]
    result <- list(cN, cricvalue, scricvec)
    return(result[[2]])
}

multi_var_norm <- function(df, sim_cor, alpha, name, remove_outlier = FALSE, n_outliers = 1)
{

  # Data and parameters
  n <- nrow(df) # observations
  p <- ncol(df) # number of variables
  D2 <- mahalanobis(df,
                    center  = colMeans(df),
                    cov = cov(df),
                    tol=1e-20) # generalized squared distance
  # Removes outliers if necessary
  if(remove_outlier == TRUE){

    i = 0
    while (i < n_outliers){
    #This is where we remove outliers. This will most likely change the correlation value and
the % number of points in the contour.
    D2 <- D2[-which.max(D2)]
    i = i + 1
    }
  }

  # Chi square qq plot
  chi_plot <- qqplot(qchisq(ppoints(n, a = .5), df = p), D2,
                     plot.it = F) # chi square plot values.
  my_cor <- cor(chi_plot$x, chi_plot$y) # correlation value
  critical_value <- qchisq(p = alpha,
                           df = p,
                           lower.tail = F) # calculate critical value

  # Proportion of points below alpha value
  prop_within_contour <- round(length(D2[D2 <= critical_value]) / length(D2),4) #

  quantiles <- quantile(D2)
  quantile_25 <- quantiles[2]
```

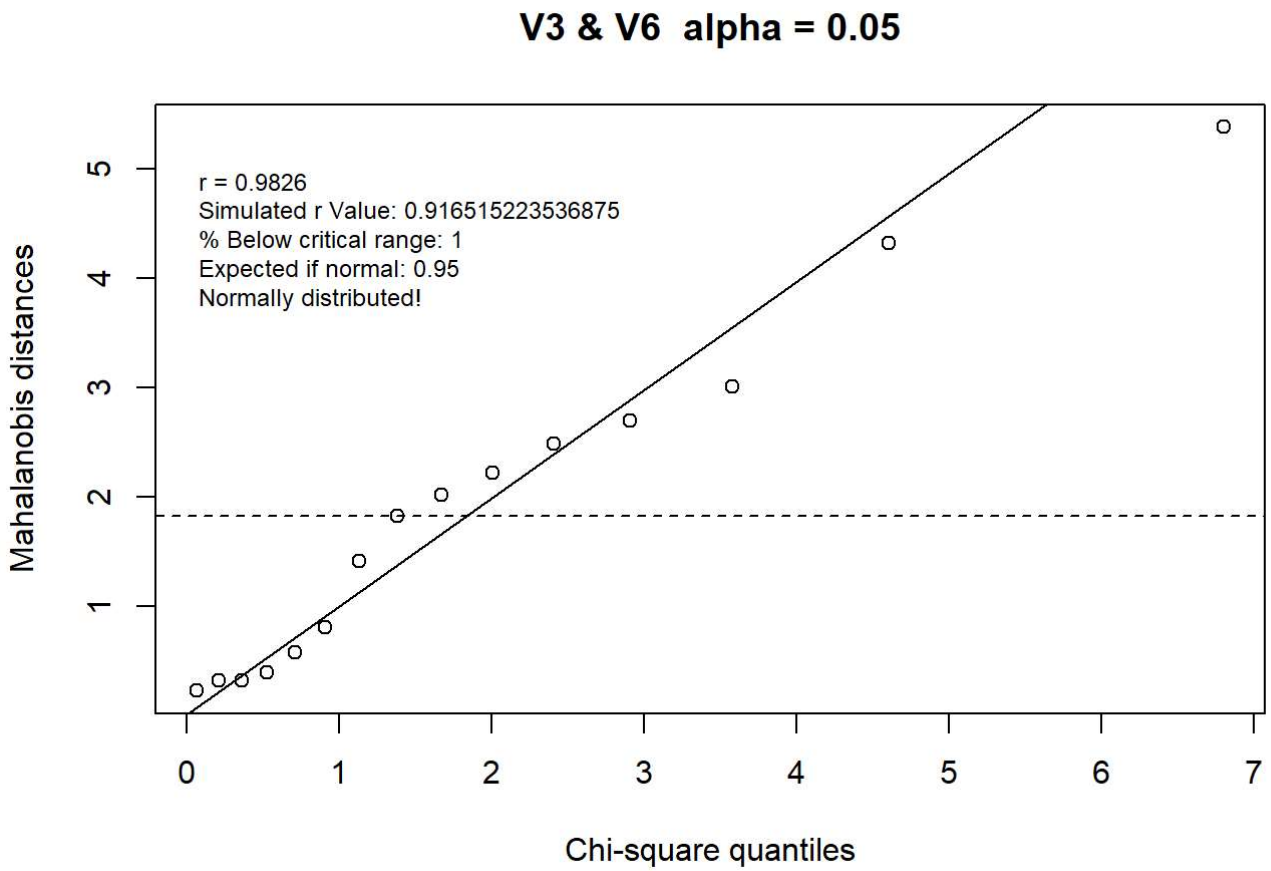
```
  quantile_50 <- quantiles[3]
  quantile_75 <- quantiles[4]

  plot(chi_plot, #From here and downwards it is only how you want the plot to look.
       ylab = 'Mahalanobis distances',
       xlab = 'Chi-square quantiles',
       main = paste0(name, ' alpha = ',alpha)) # plot chi square plot

  # Q line
  y <- rchisq(500, df = p)
  qqline(y, distribution = function(n) qchisq(n, df = p), prob = c(0.1, 0.6))

  # Lines for quantiles
  abline(h = quantile_50, lty = 2) # 50% quantiles becayse book p. 187
  abline(h = critical_value, lty = 2, col = "red") # Below Critical value

  if(my_cor > sim_cor){
    conclusion <- "Normally distributed!"
  }

  else {
    conclusion <- "NOT normally distributed!"
  }
  legend("topleft",
         paste0("r = ", round(my_cor,4), "\n",
                "Simulated r Value: ", sim_cor, "\n",
                "% Below critical range: ", prop_within_contour, "\n",
                "Expected if normal: ", 1-alpha, "\n",
                conclusion),
         cex = 0.75,
         bty = "n") # add legend to plot
}


bivar_pairs <- function(data_frame, significance = 0.05, remove_outliers = FALSE, n_outliers
= 1, n_simulations = 10000) {

sim_cor <- FindcrikChi(n = length(data_frame[,1]),
                           p = 2,
                           alpha = significance,
                           n_simulations)

prev <- c()
i = 1
for (col_name in colnames(data_frame)){

  j = 1
  for (col_name_inner in colnames(data_frame)){

    if (i == j){
      j = j + 1
      next
    }

    else if (j %in% prev){
      j = j + 1
```

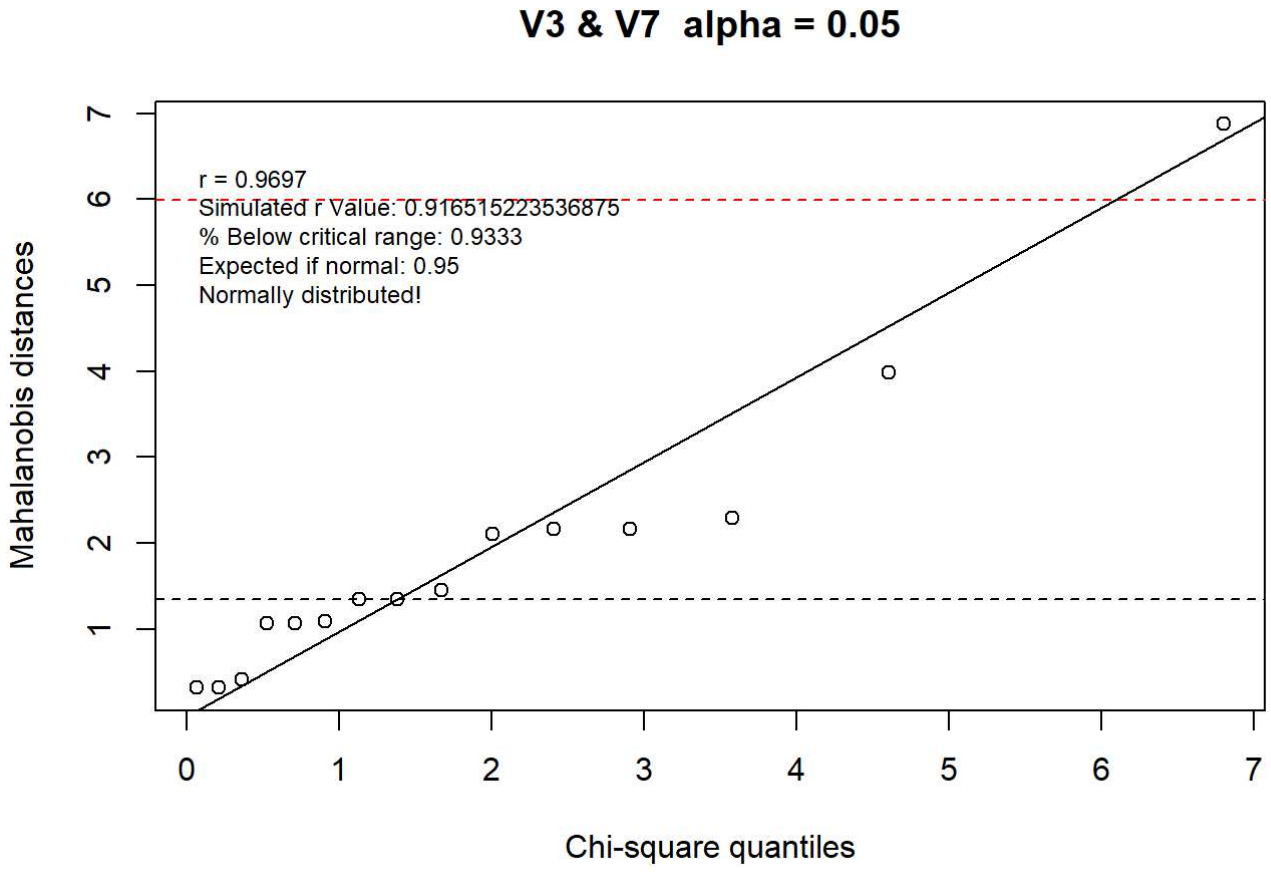
```
      next
    }

    else{
      multi_var_norm(df = data_frame[, c(i, j)],
                     sim_cor = sim_cor,
                     alpha = significance,
                     name = paste0(col_name, " & ", col_name_inner, " "),
                     remove_outlier = remove_outliers,
                     n_outliers = n_outliers)

    }
    j = j + 1
  }
  prev <- append(prev, i)
  i = i + 1
  }
}
```

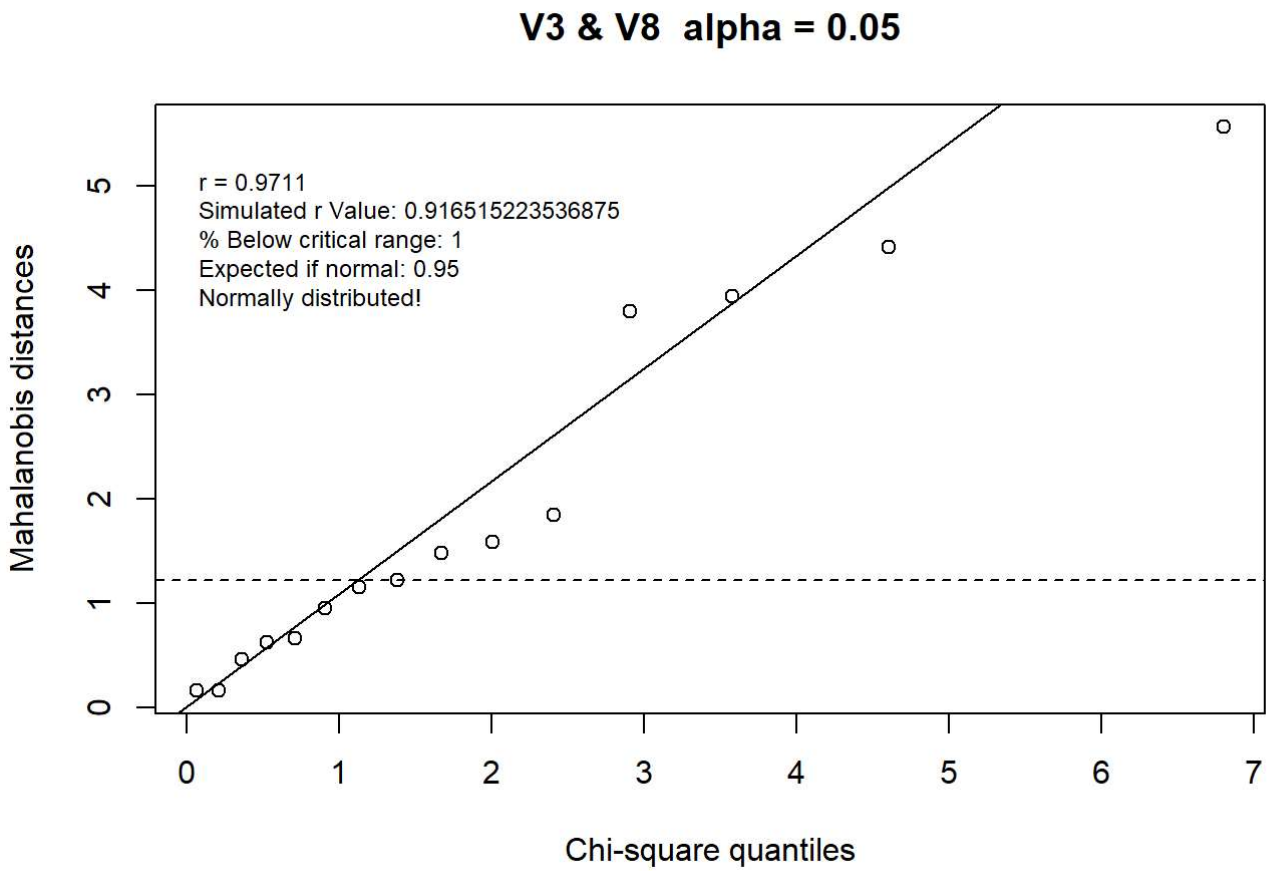
```
par(mfrow = c(3, 5))
bivar_pairs(z_df)
```

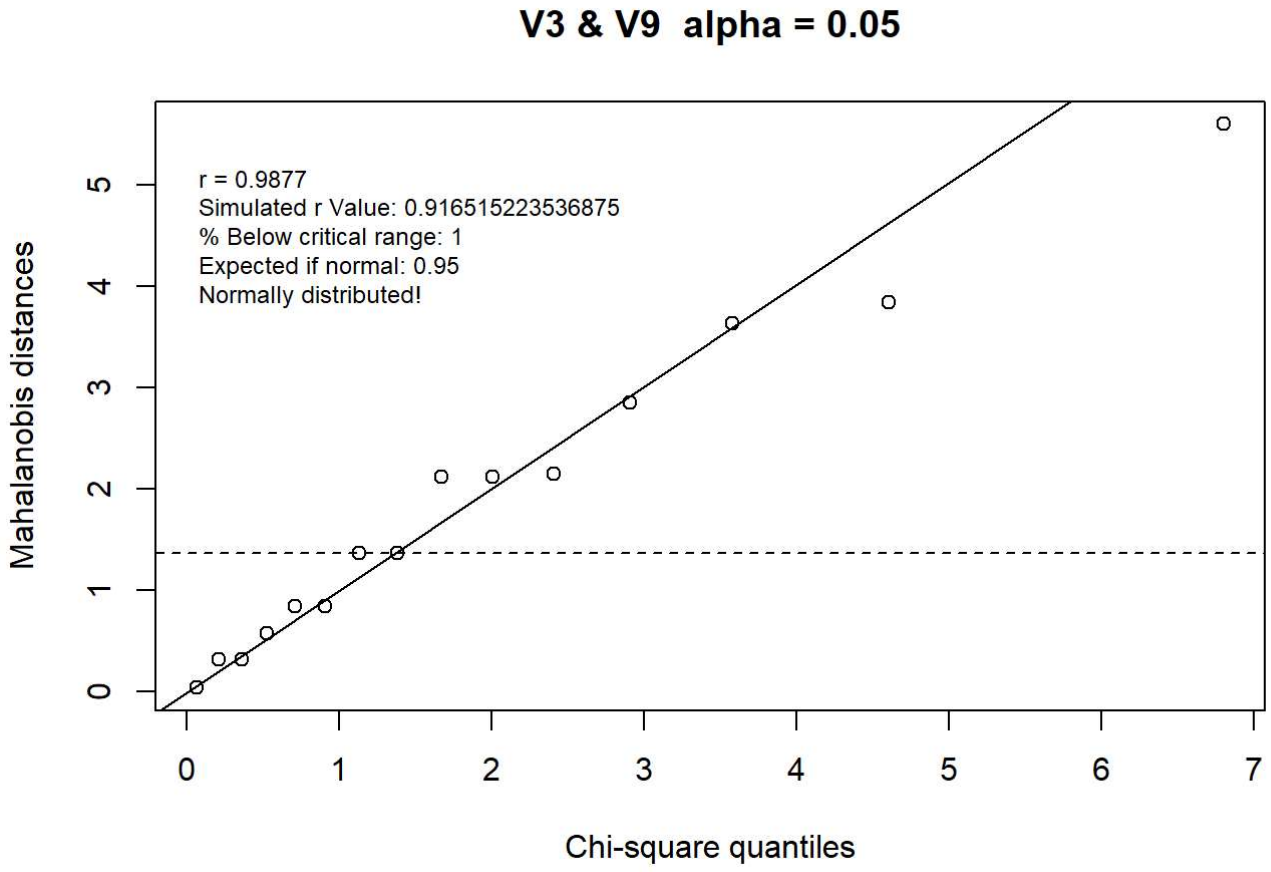


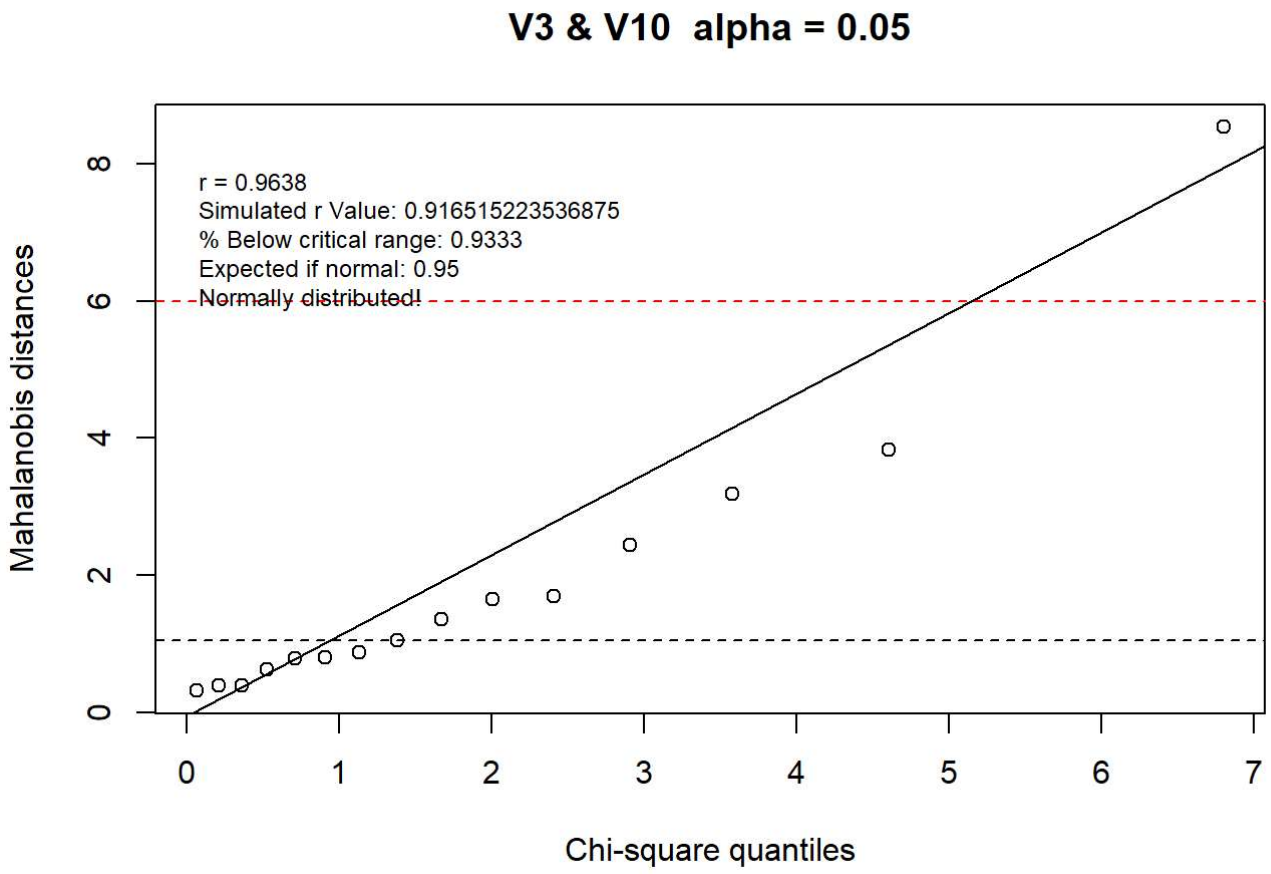

```
bivar_pairs(z_df)
```

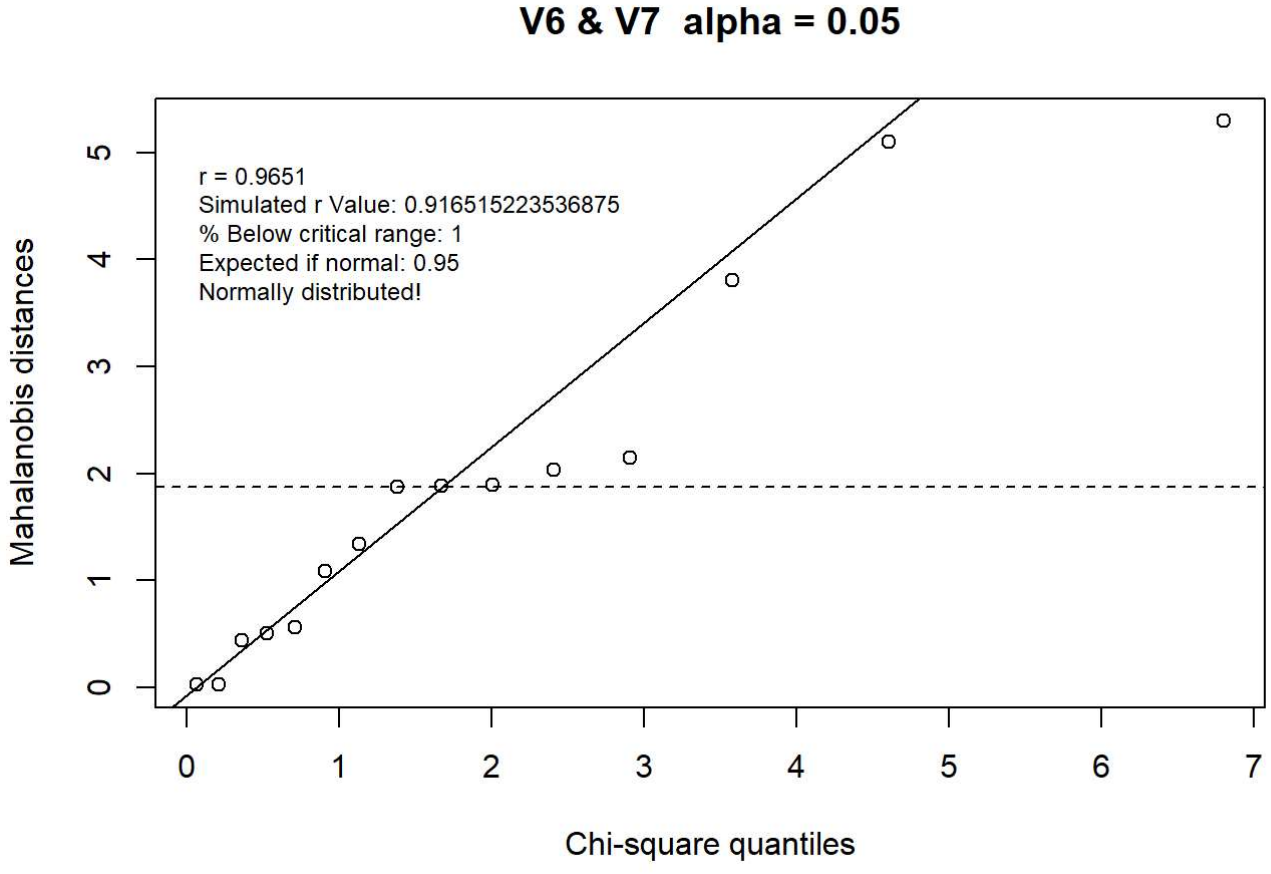

V3 & V6 alpha = 0.05
r = 0.9826
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

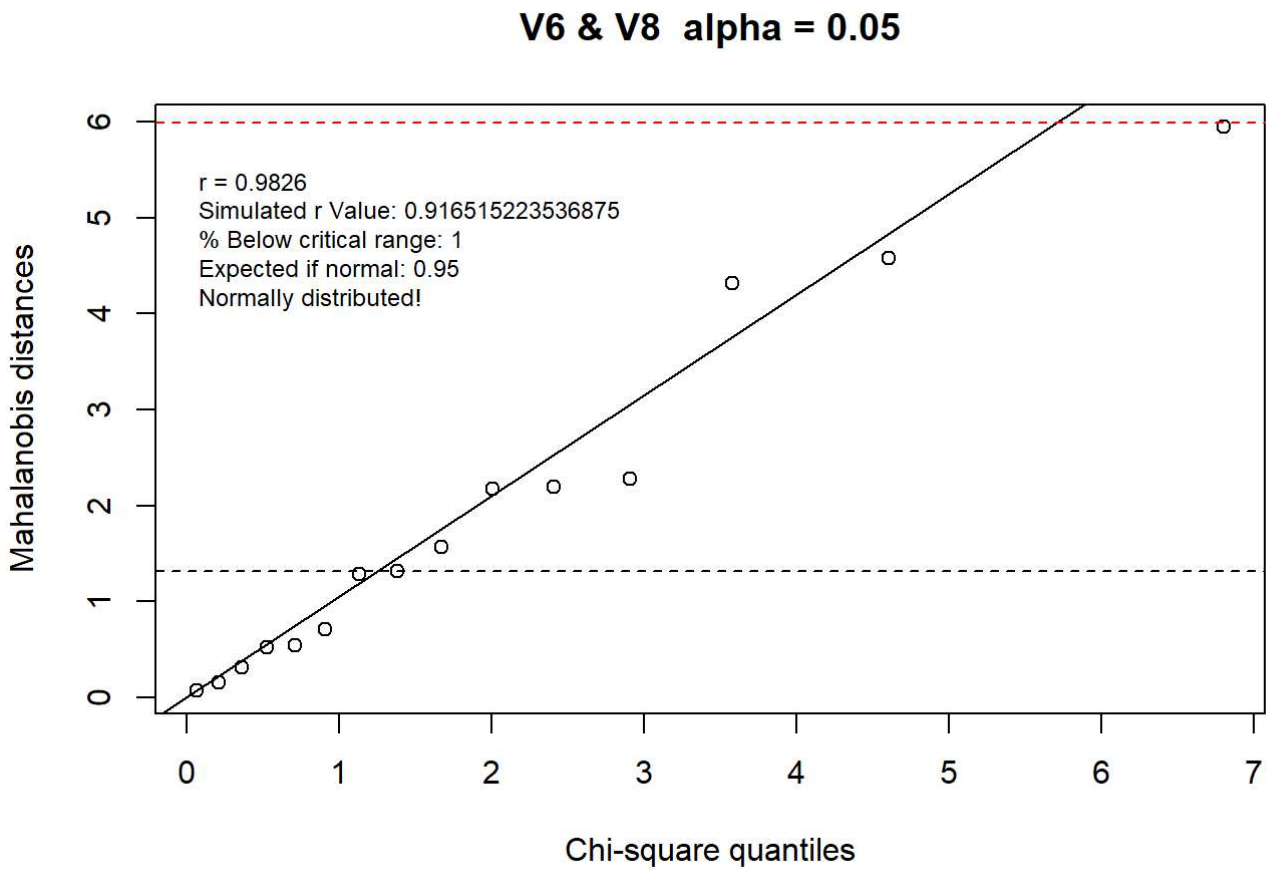

V3 & V7 alpha = 0.05
r = 0.9697
Simulated r Value: 0.916515223536875
% Below critical range: 0.9333
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

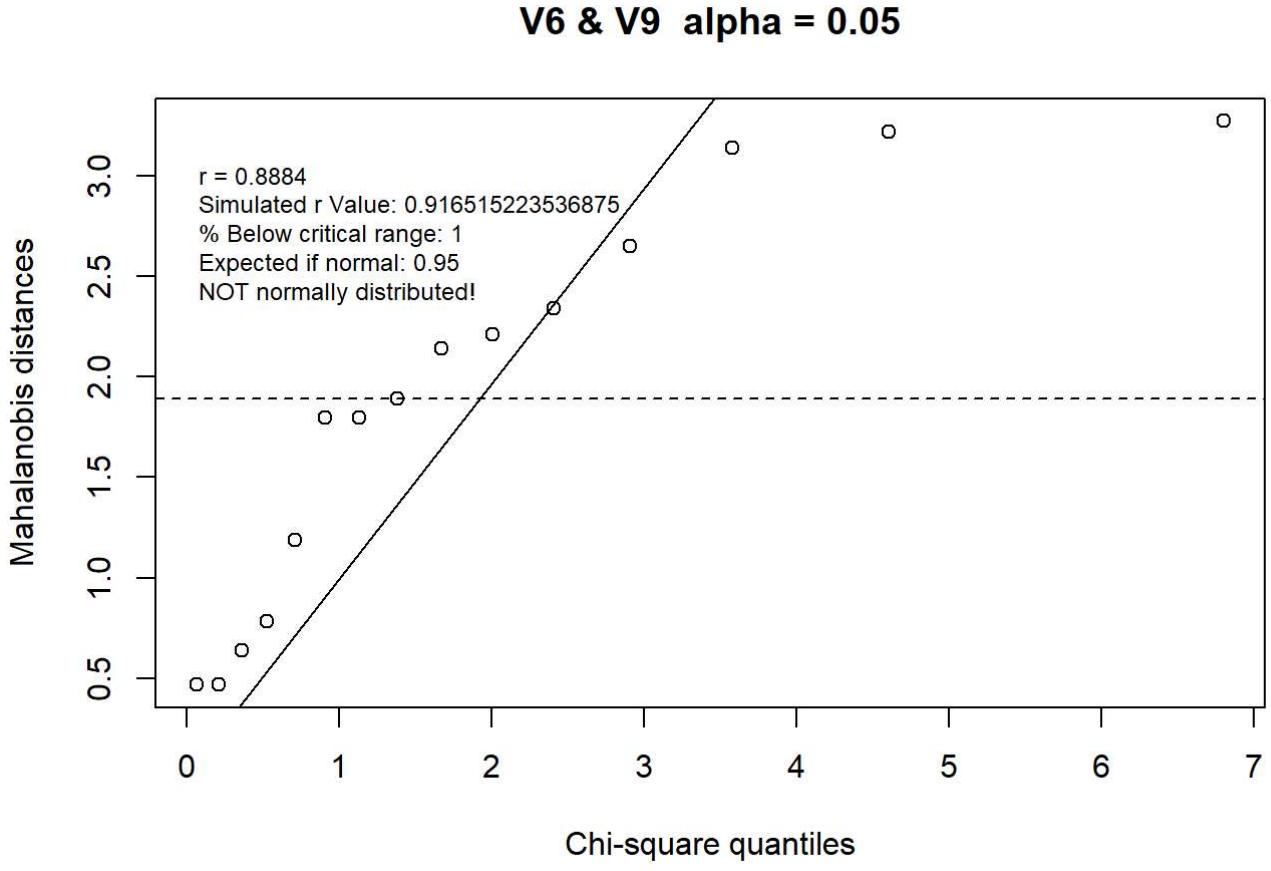

V3 & V8 alpha = 0.05
r = 0.9711
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

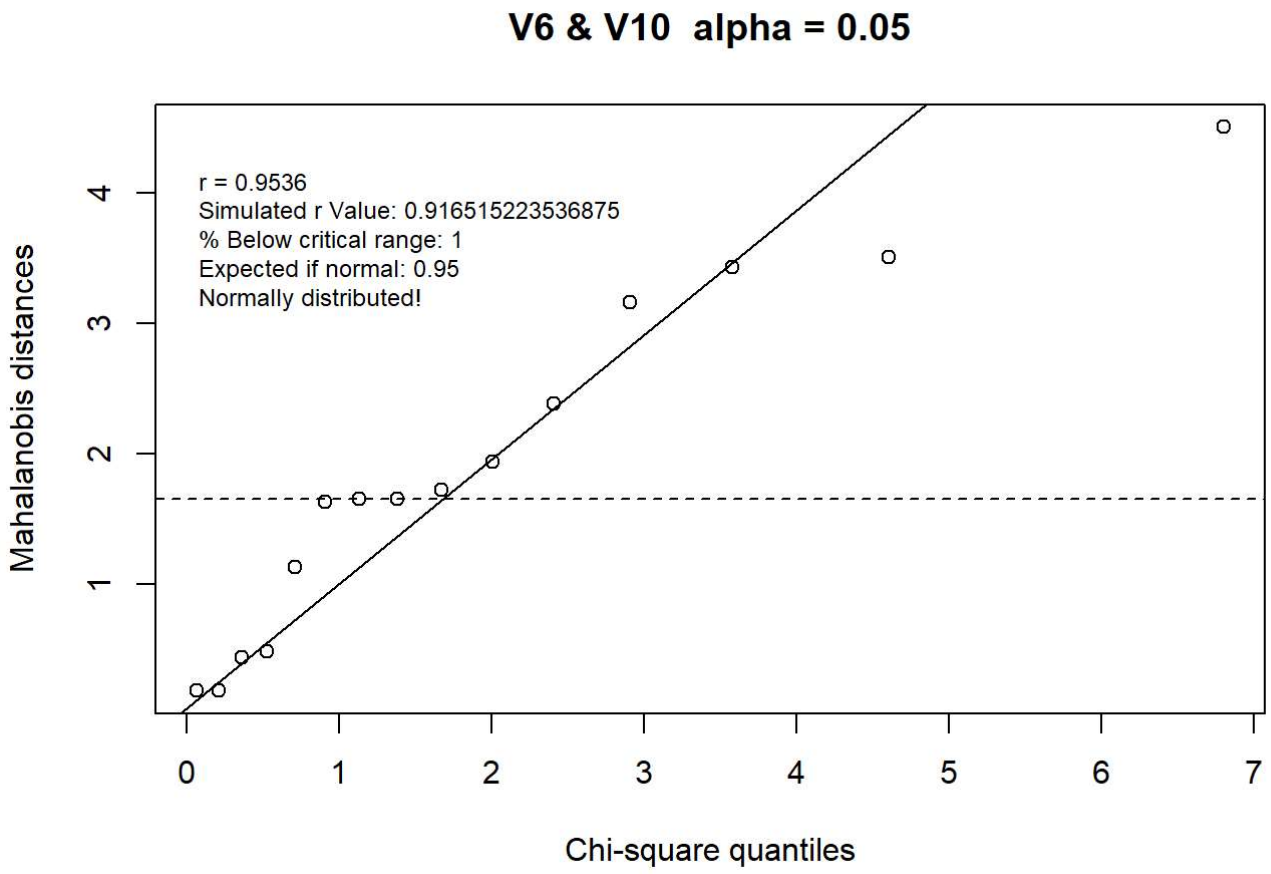

V3 & V9 alpha = 0.05
r = 0.9877
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

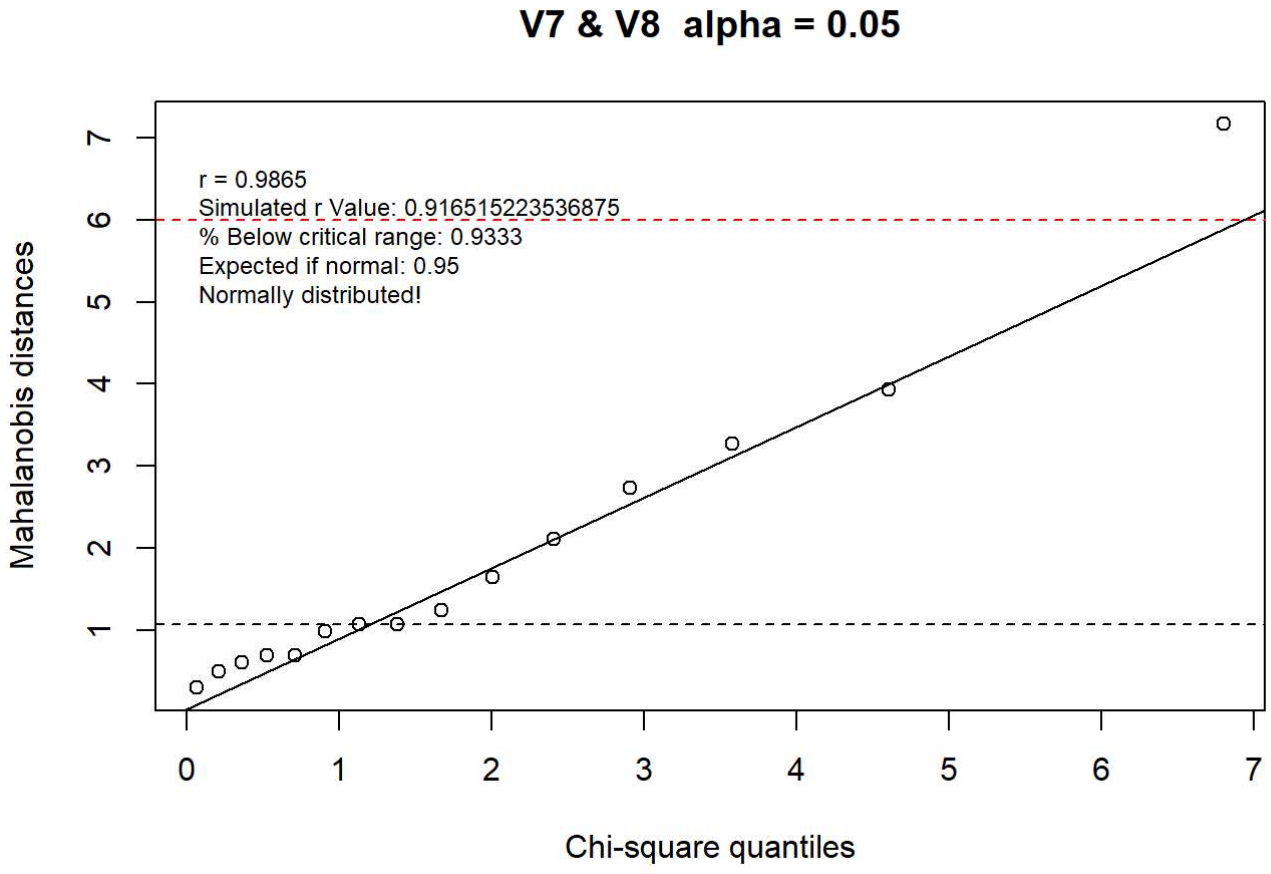

V3 & V10 alpha = 0.05
r = 0.9638
Simulated r Value: 0.916515223536875
% Below critical range: 0.9333
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7
0
2
4
6
8

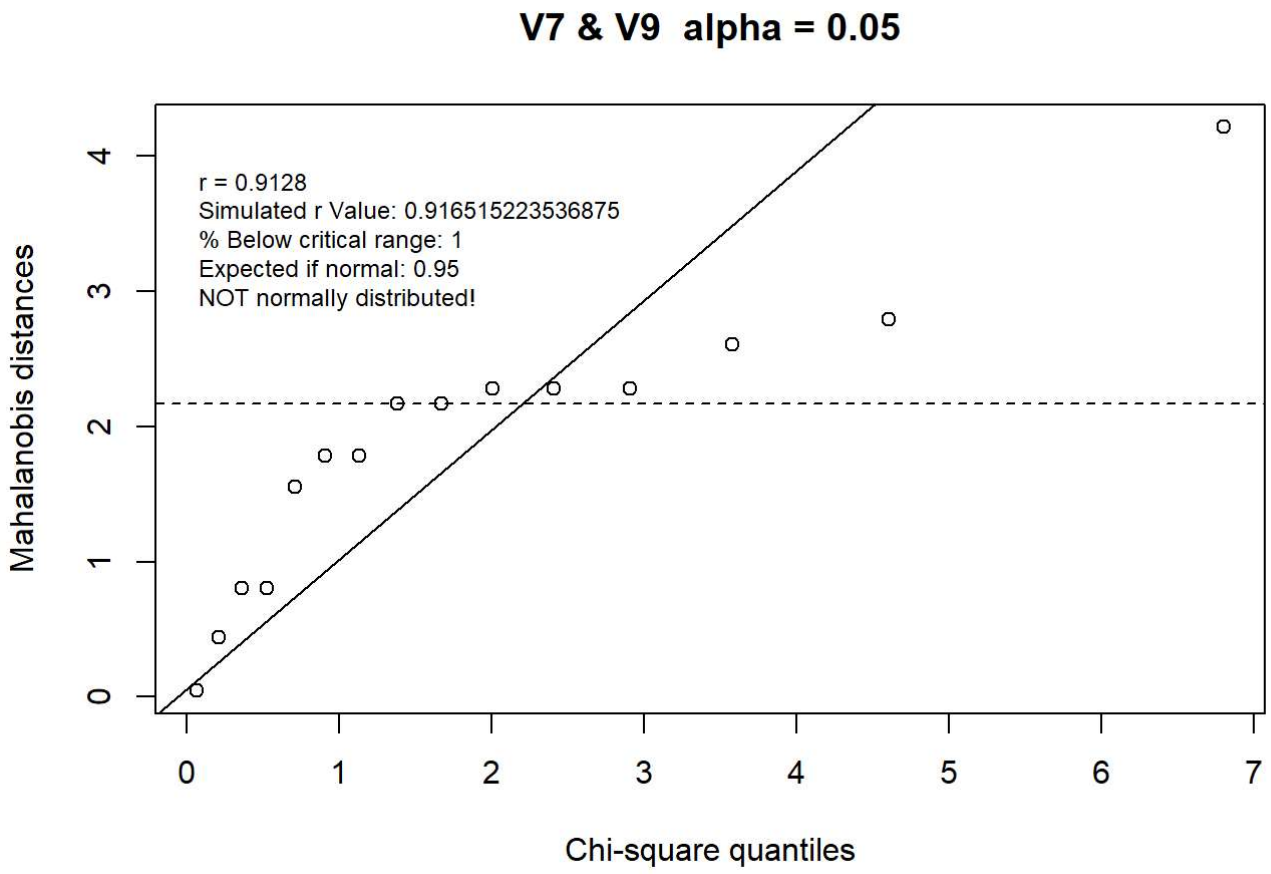

V6 & V7 alpha = 0.05
r = 0.9651
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7
0
1
2
3
4
5

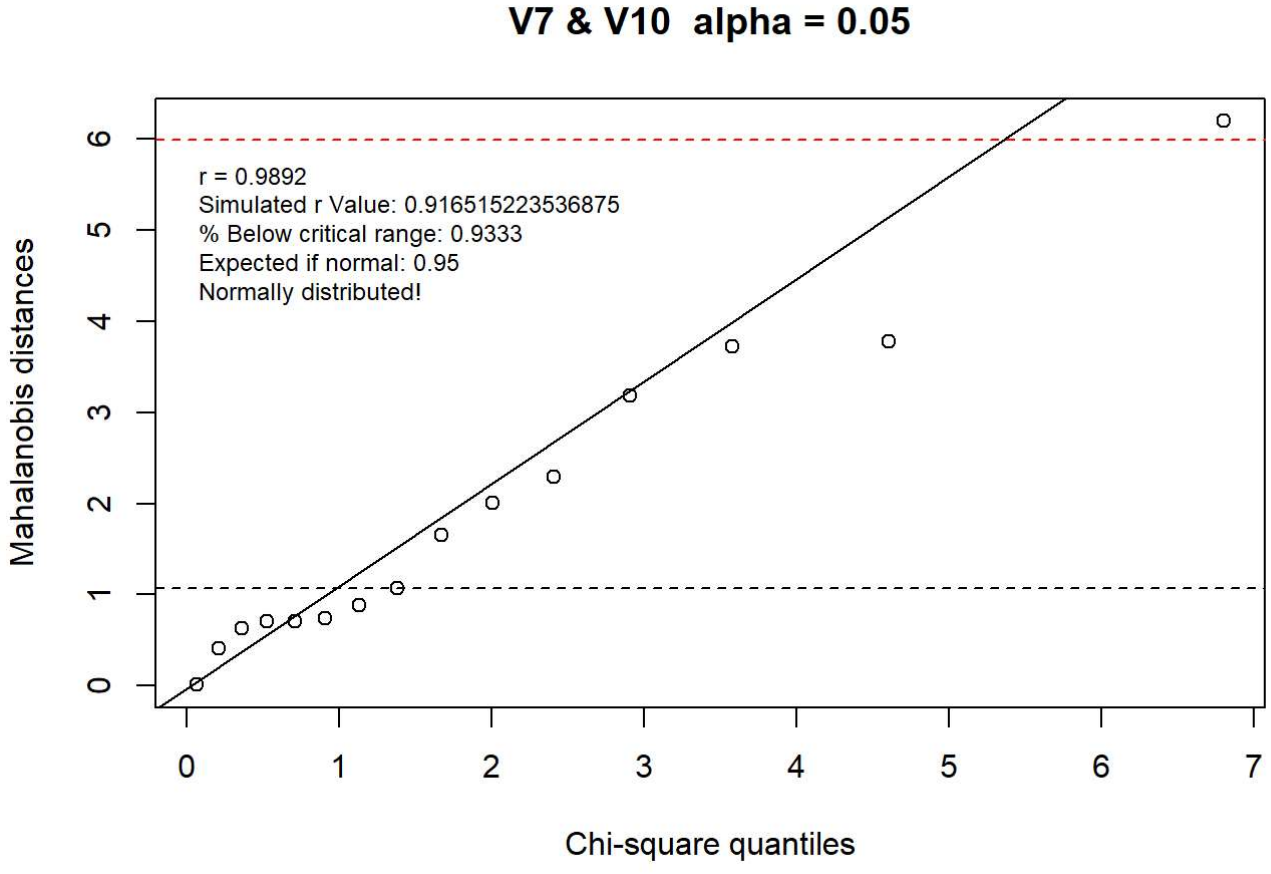

V6 & V8 alpha = 0.05
r = 0.9826
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

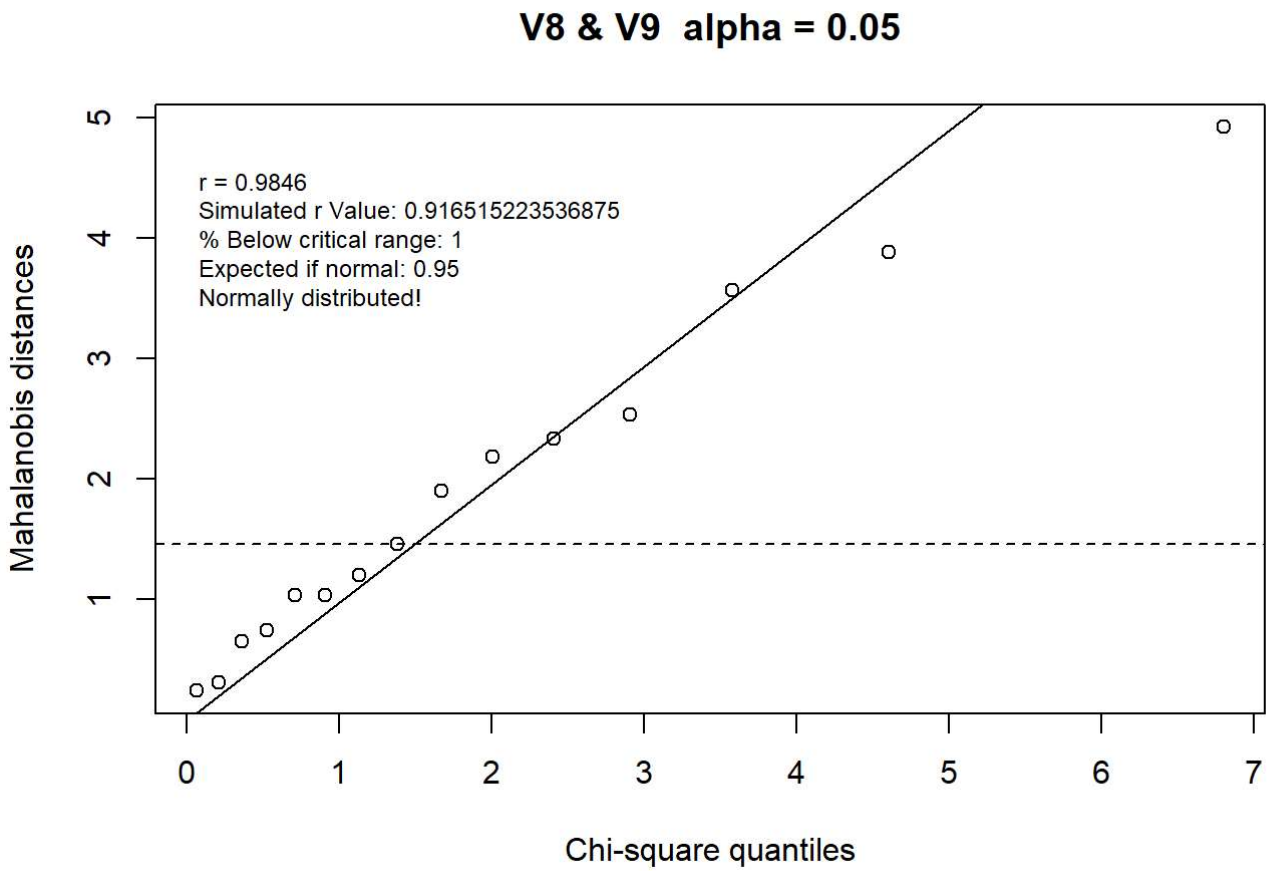

V6 & V9 alpha = 0.05
r = 0.8884
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
NOT normally distributed!
Mahalanobis distances
Chi-square quantiles
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7

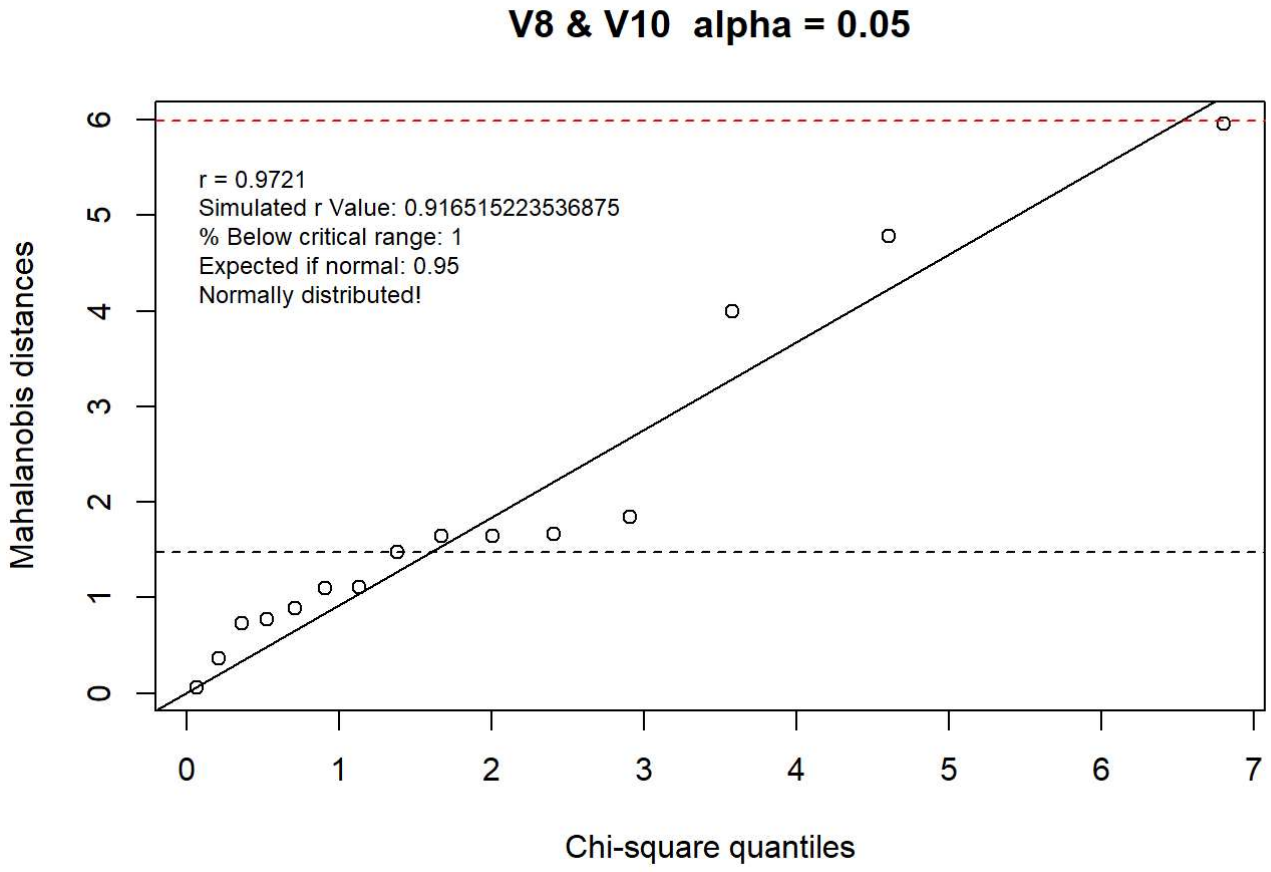

V6 & V10 alpha = 0.05
r = 0.9536
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

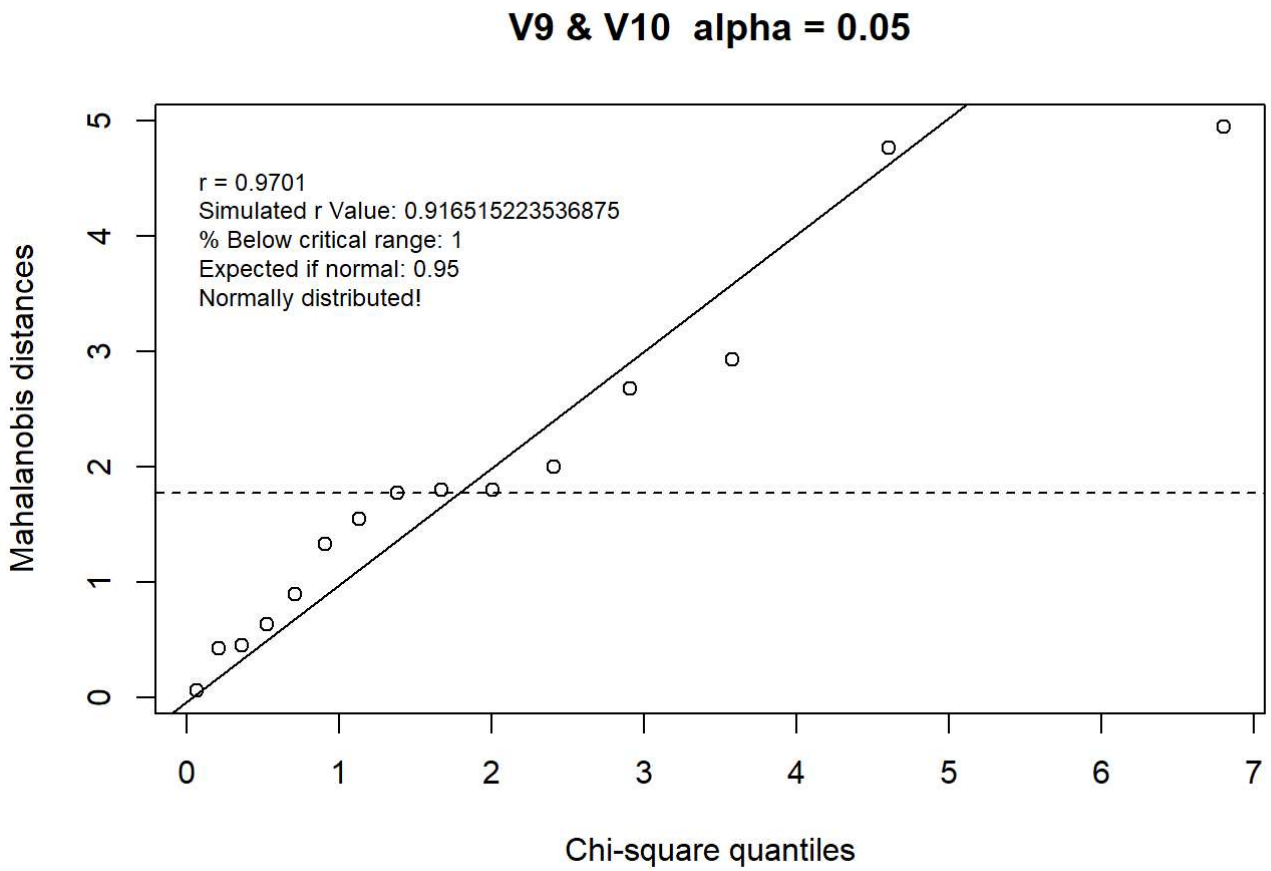

V7 & V8 alpha = 0.05
r = 0.9865
Simulated r Value: 0.916515223536875
% Below critical range: 0.9333
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7


V7 & V9 alpha = 0.05
r = 0.9128
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
NOT normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7


V7 & V10 alpha = 0.05
r = 0.9892
Simulated r Value: 0.916515223536875
% Below critical range: 0.9333
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7


V8 & V9 alpha = 0.05
r = 0.9846
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

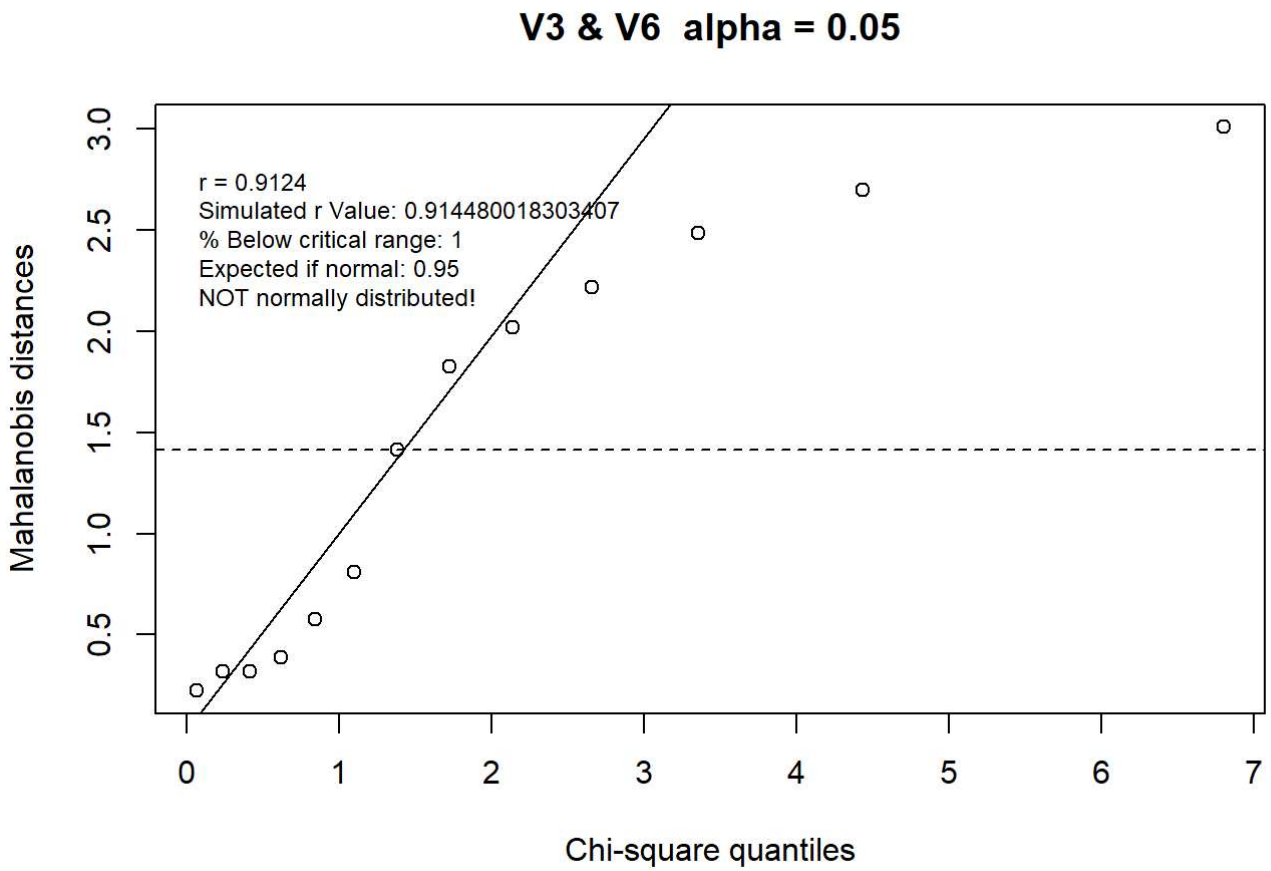

V8 & V10 alpha = 0.05
r = 0.9721
Simulated r Value: 0.916515223536875
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

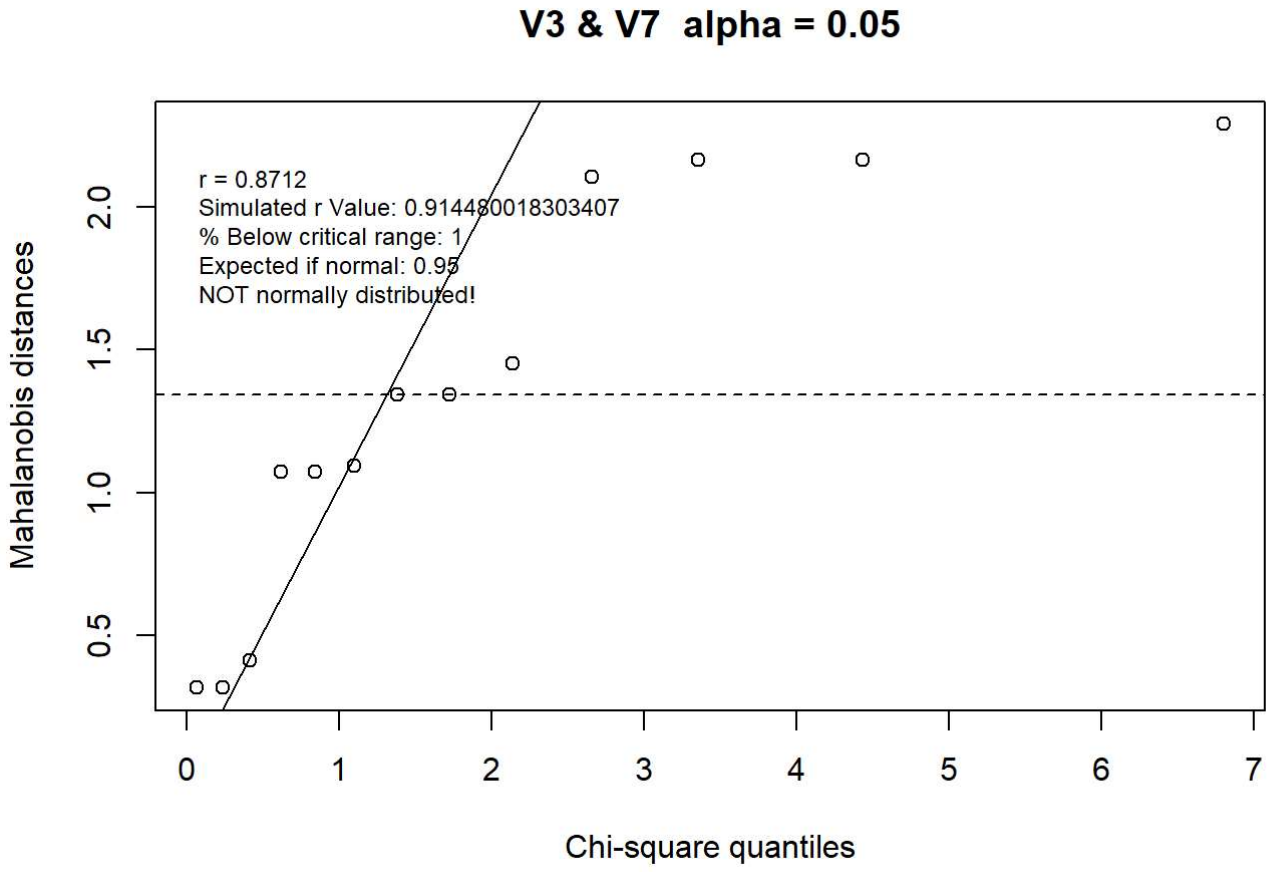


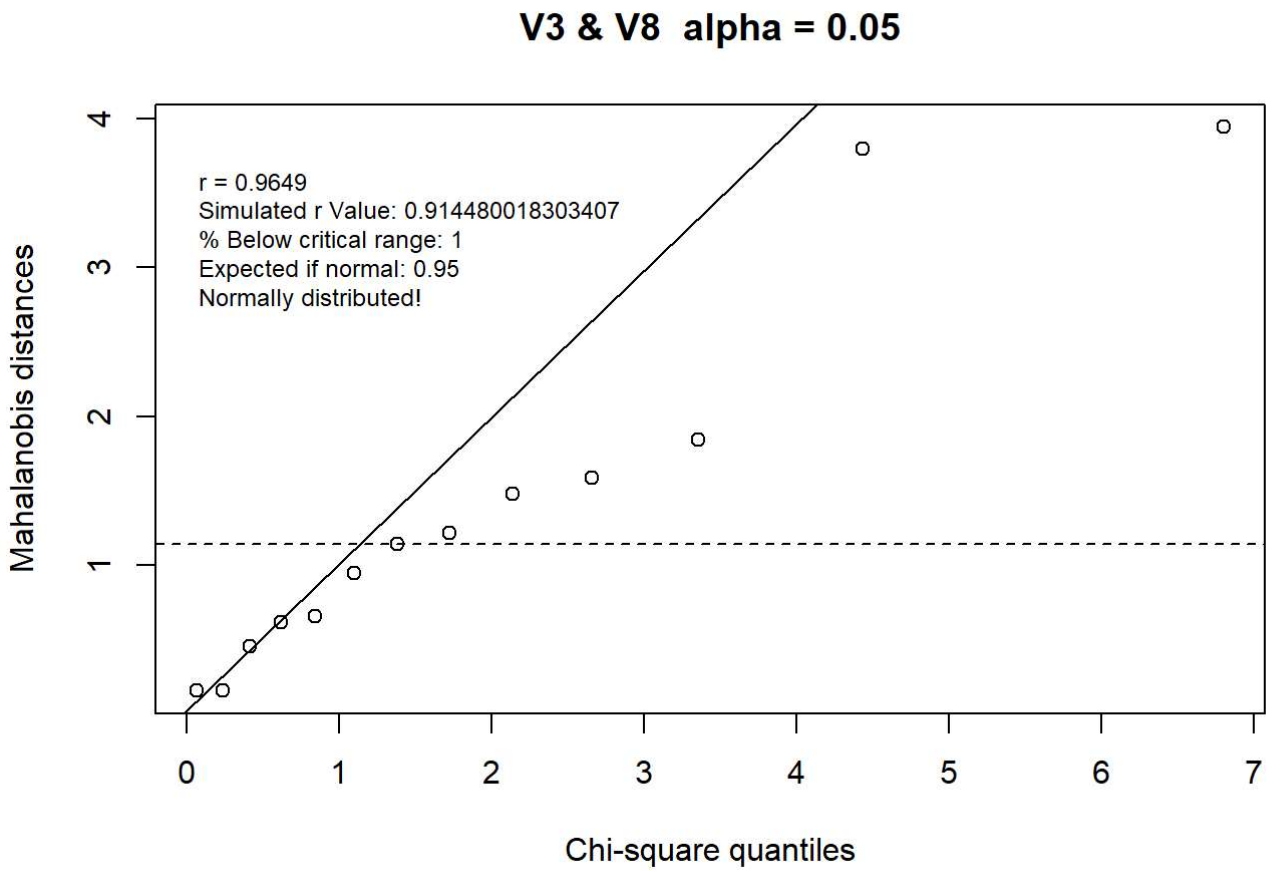
The following pairs are not normal:
- V6 and V9
- V7 and V9

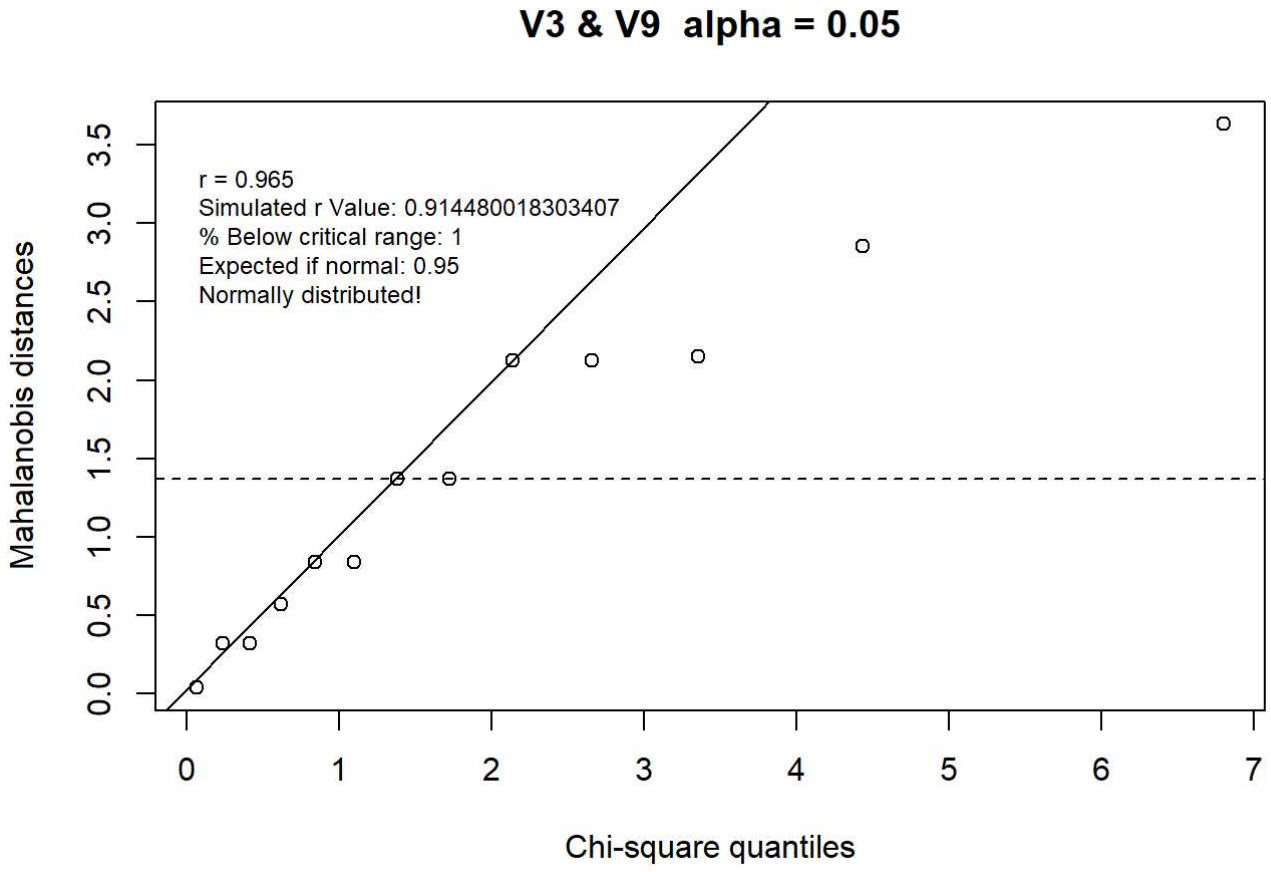
We either have to remove outliers from V9 or just remove the whole attribute:

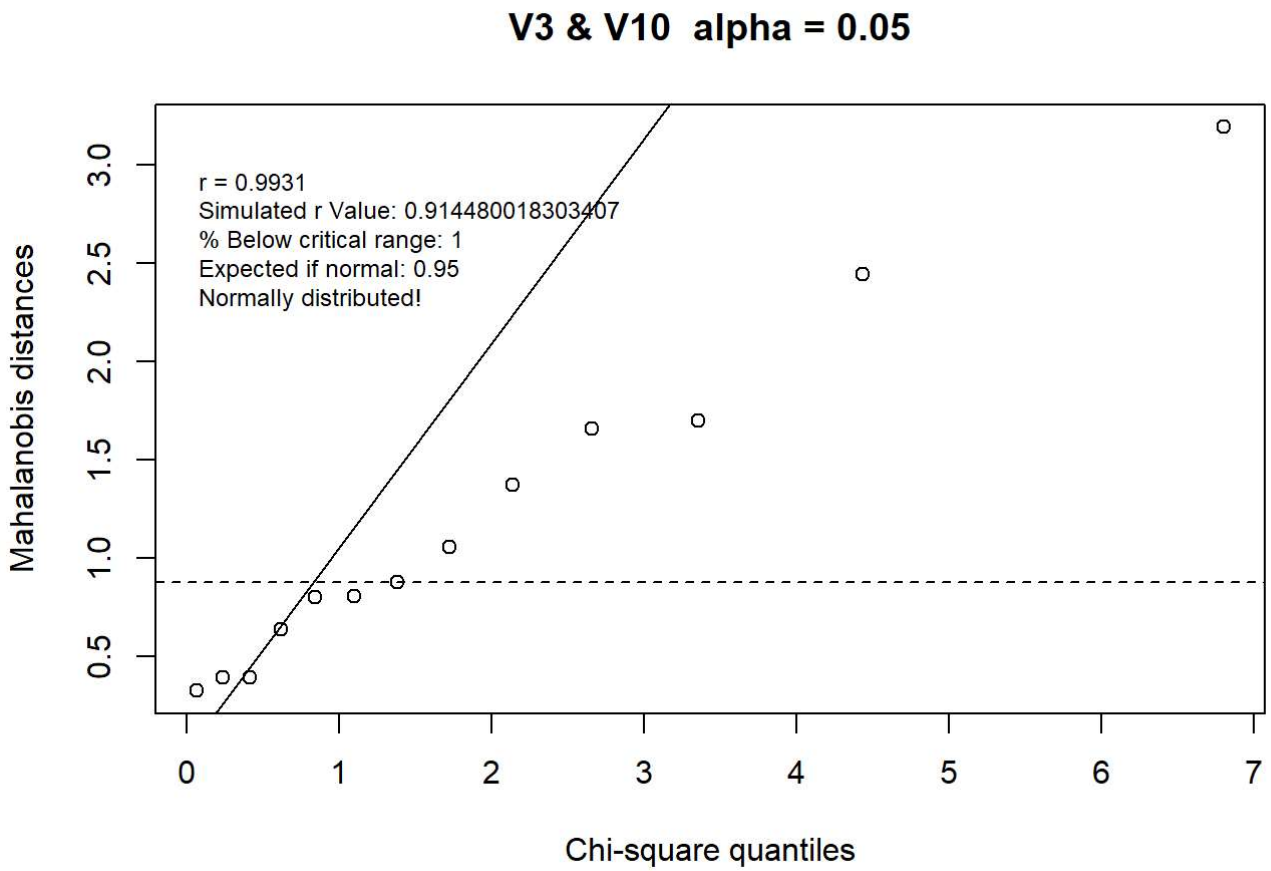
```
bivar_pairs(z_df, remove_outliers = TRUE, n_outliers = 2)
```

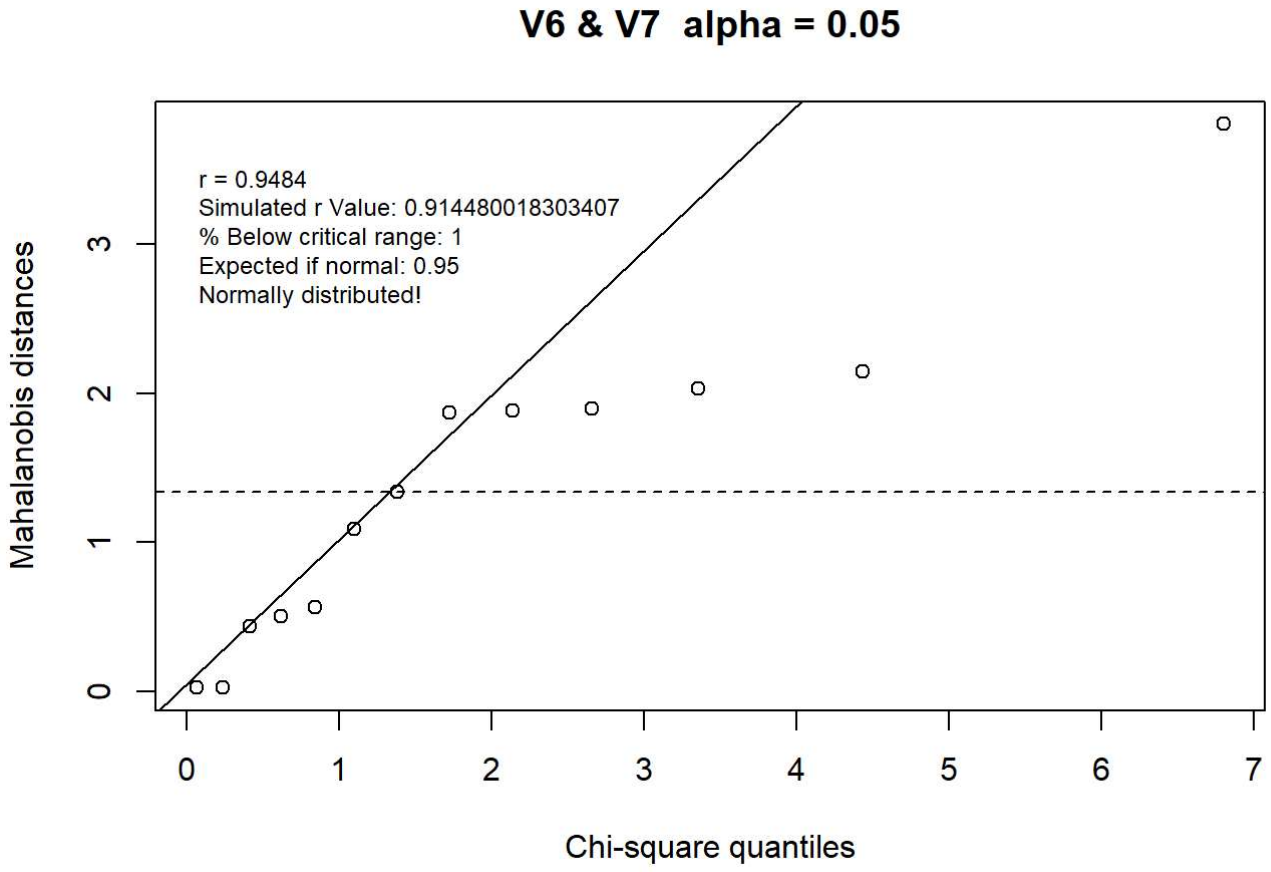

V3 & V6 alpha = 0.05
r = 0.9124
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
NOT normally distributed!
Mahalanobis distances
Chi-square quantiles
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7

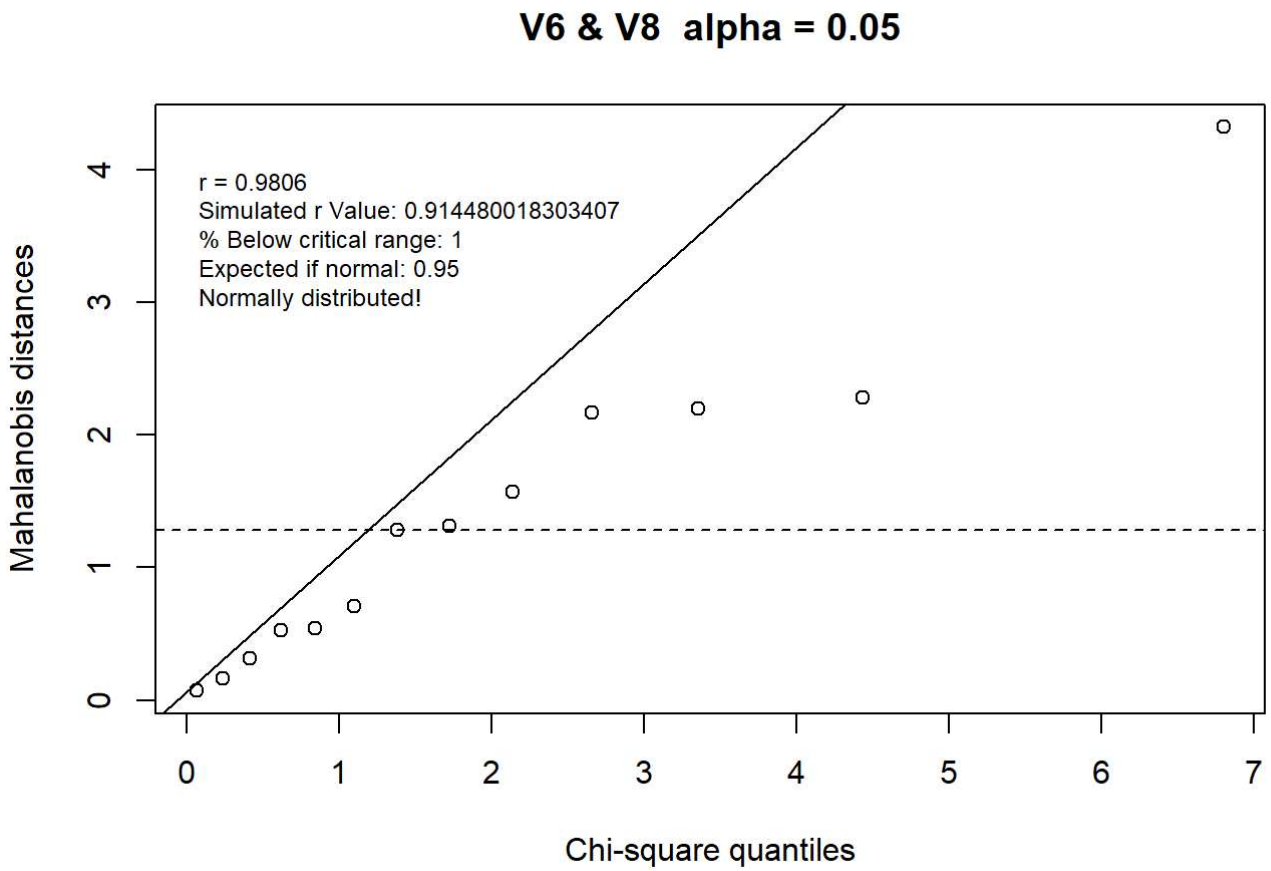

V3 & V7 alpha = 0.05
r = 0.8712
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
NOT normally distributed!
Mahalanobis distances
Chi-square quantiles
0.5
1.0
1.5
2.0
0
1
2
3
4
5
6
7

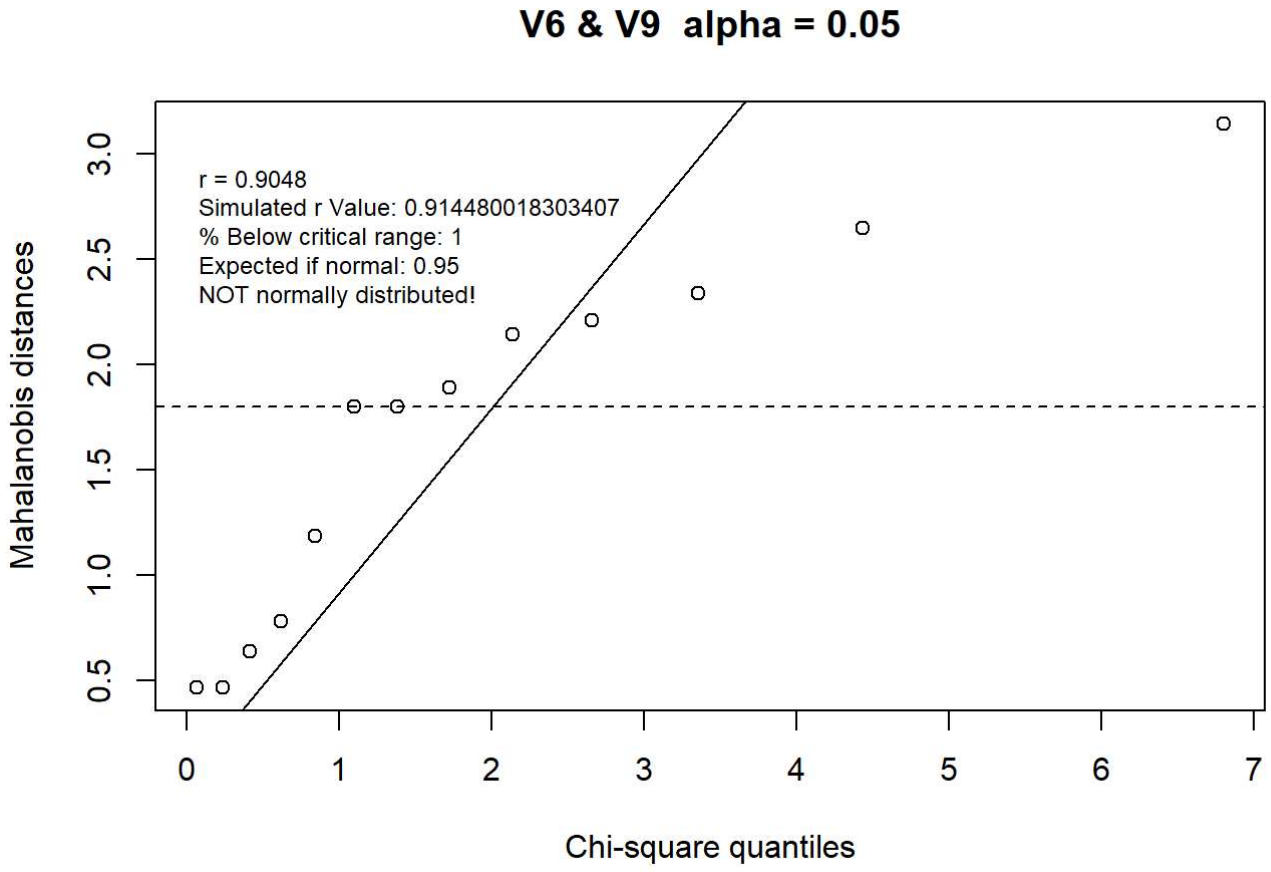

V3 & V8 alpha = 0.05
r = 0.9649
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

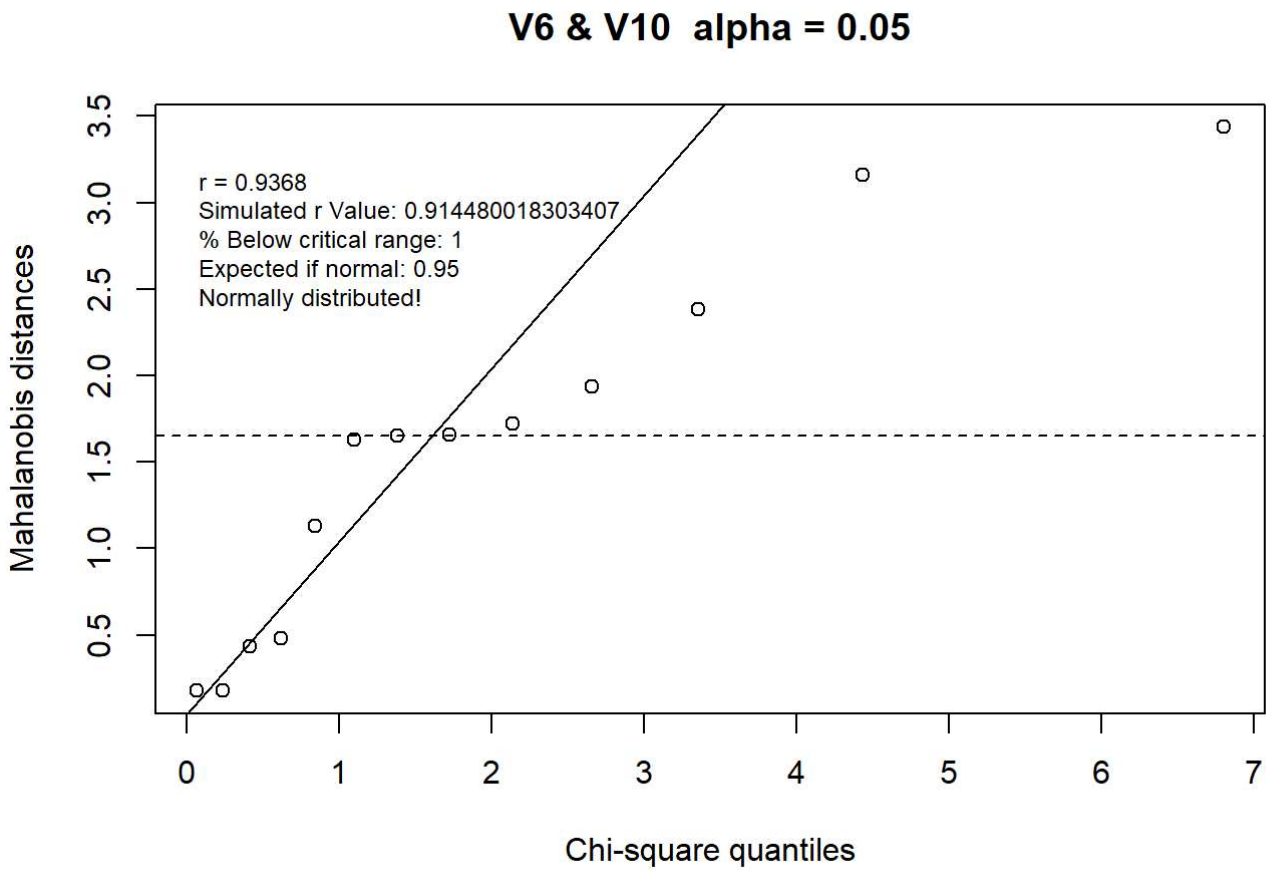

V3 & V9 alpha = 0.05
r = 0.965
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
1
2
3
4
5
6
7

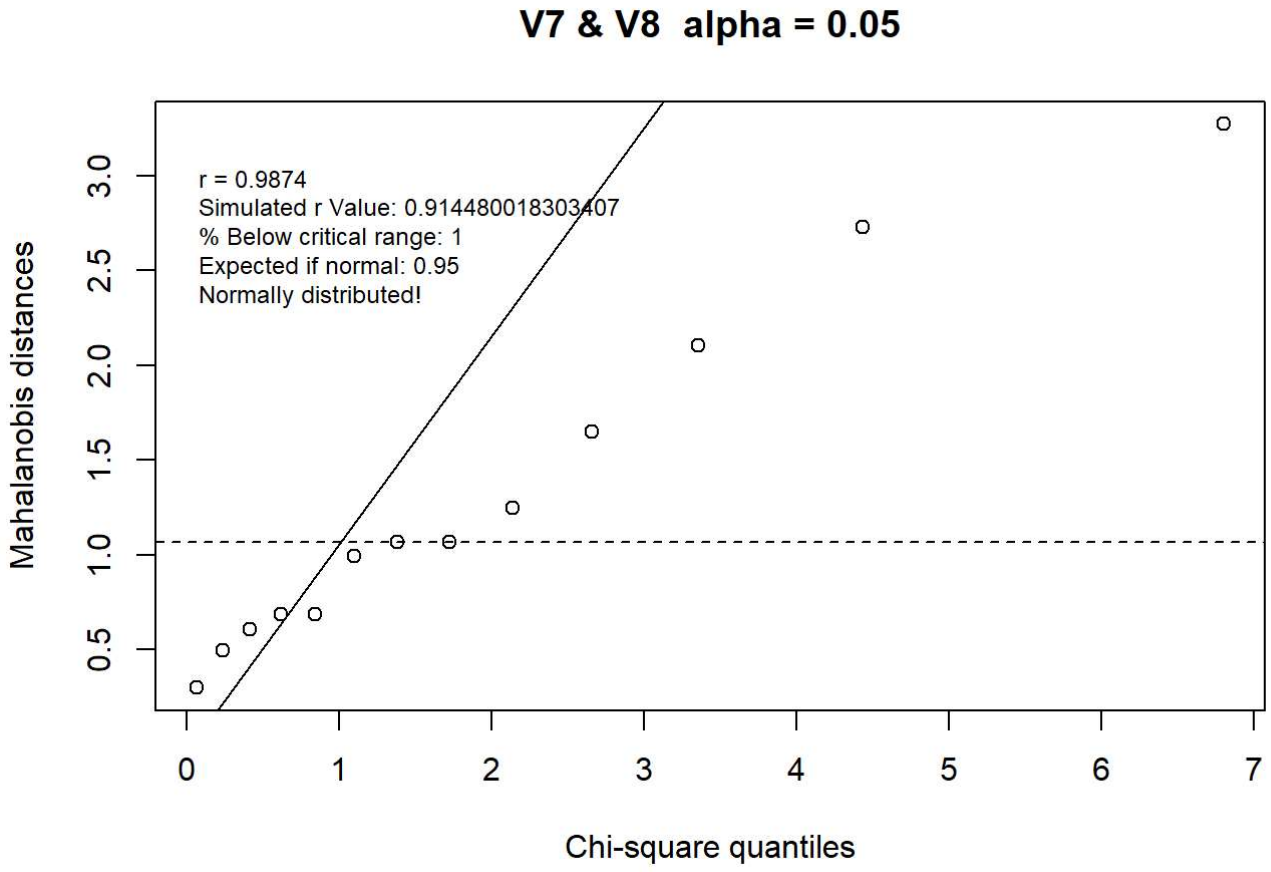

V3 & V10 alpha = 0.05
r = 0.9931
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7

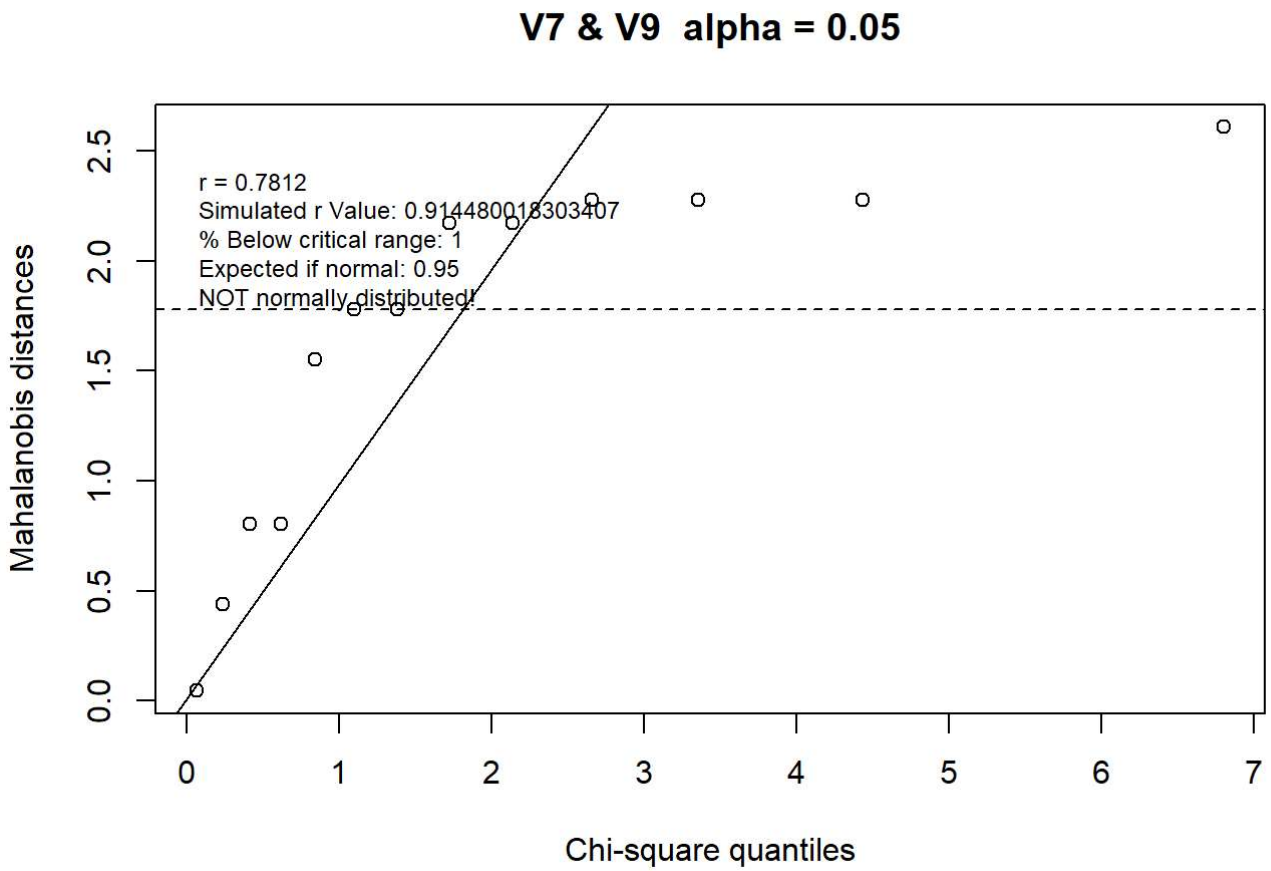

V6 & V7 alpha = 0.05
r = 0.9484
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
0
1
2
3
4
5
6
7

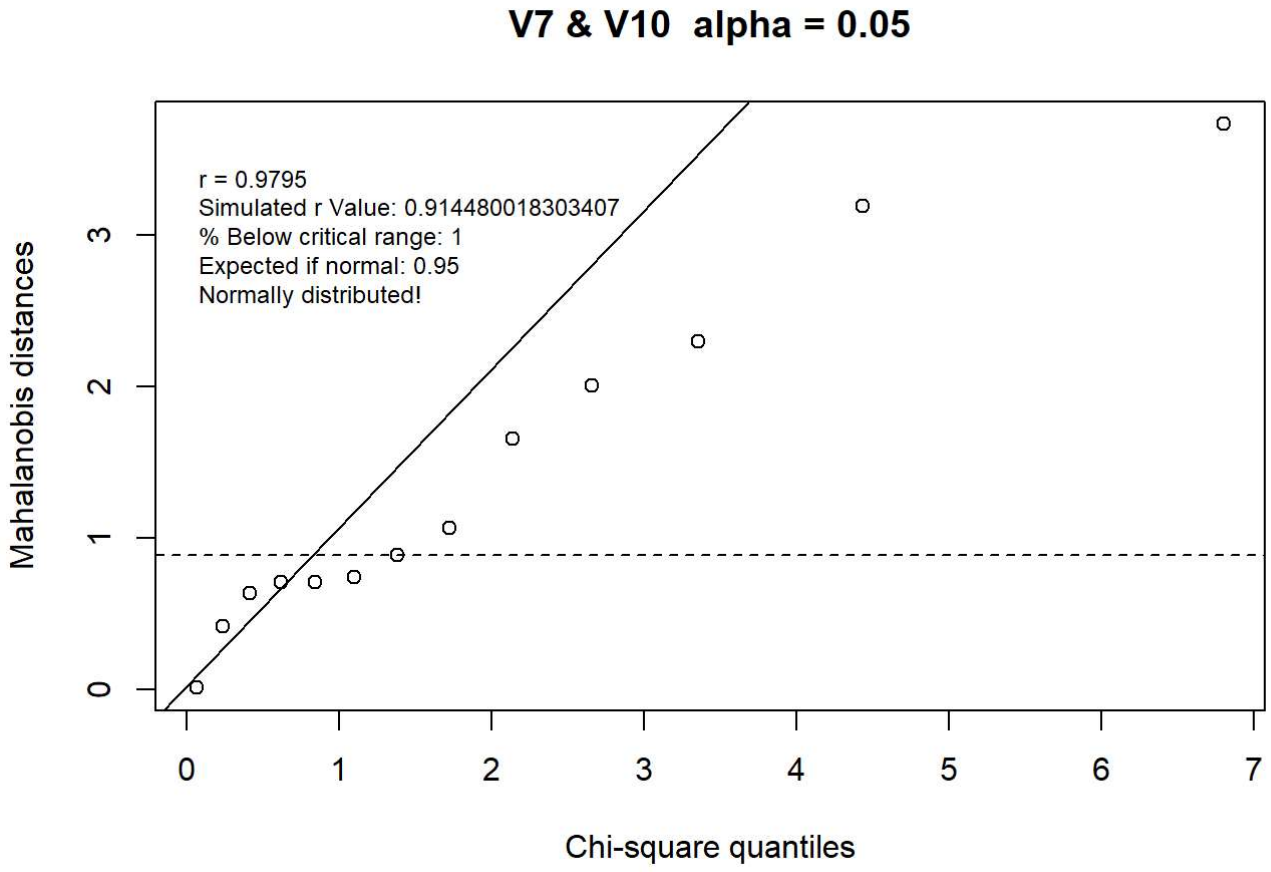

V6 & V8 alpha = 0.05
r = 0.9806
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

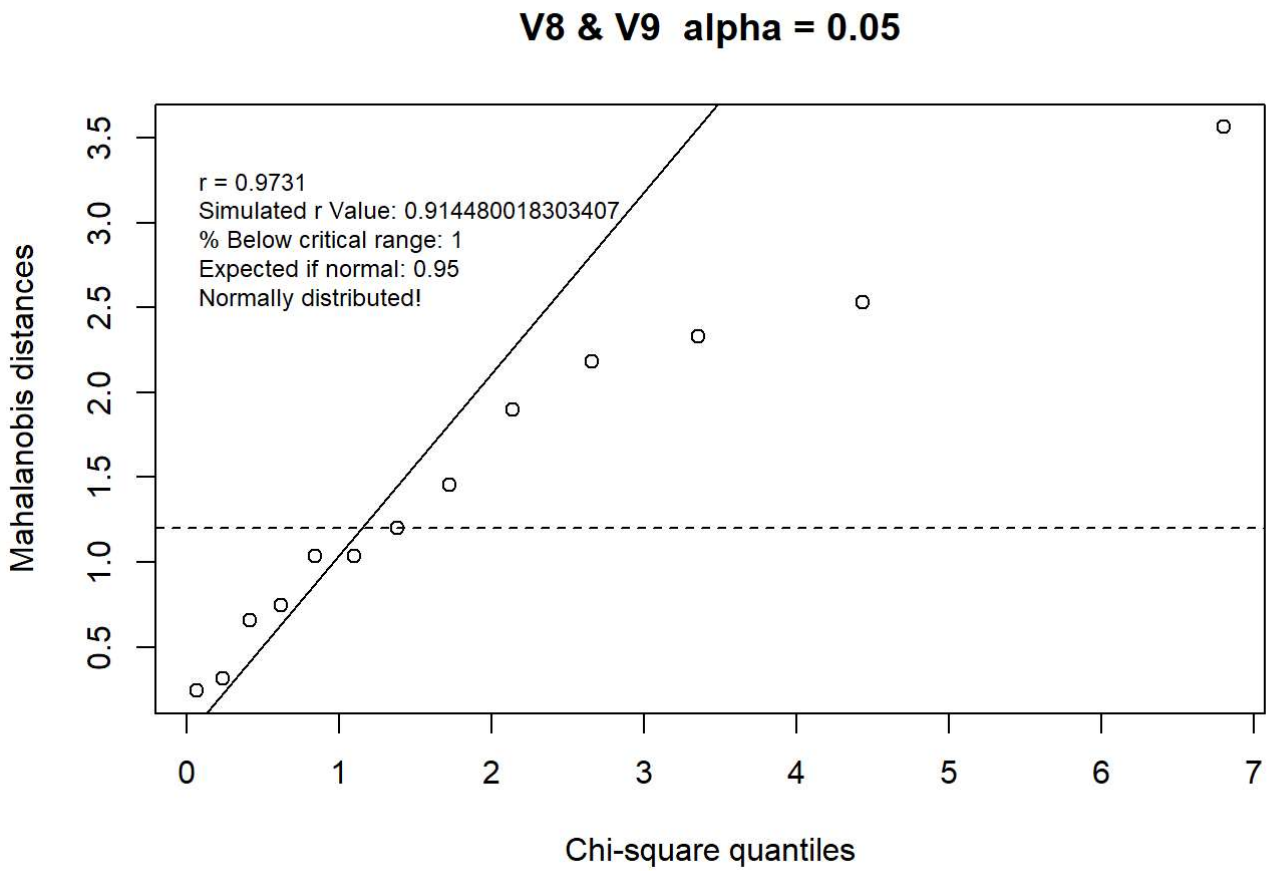

V6 & V9 alpha = 0.05
r = 0.9048
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
NOT normally distributed!
Mahalanobis distances
Chi-square quantiles
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7

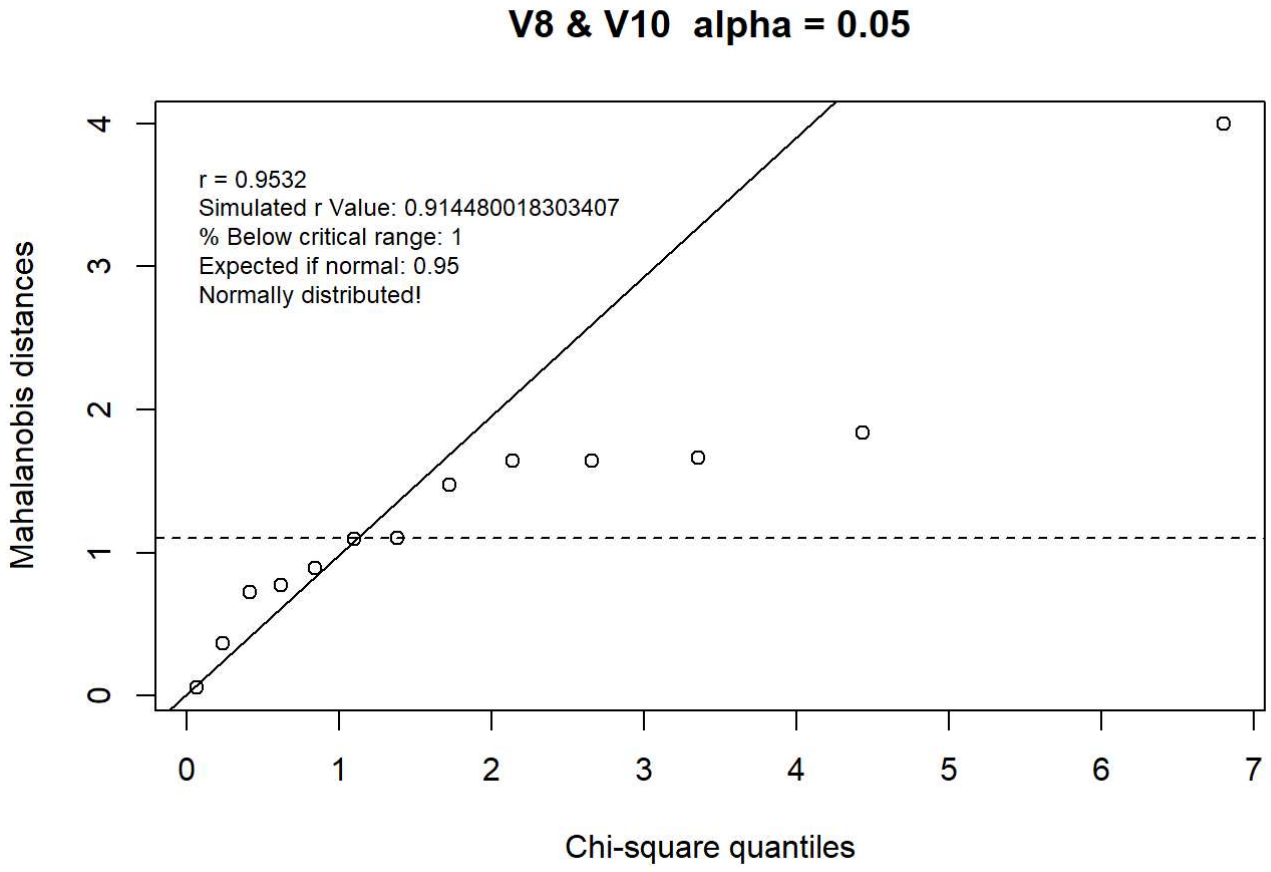

V6 & V10 alpha = 0.05
r = 0.9368
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
1
2
3
4
5
6
7

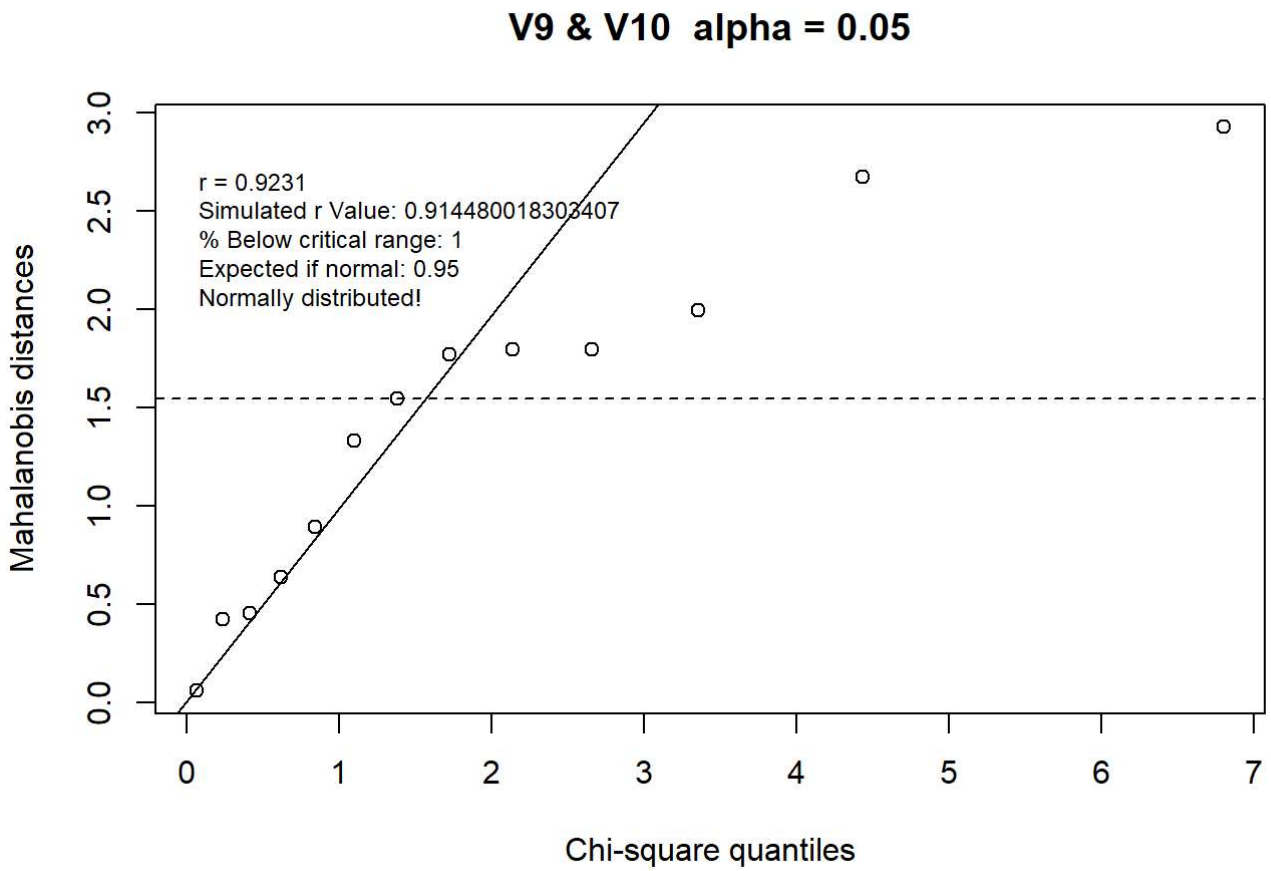

V7 & V8 alpha = 0.05
r = 0.9874
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7


V7 & V9 alpha = 0.05
r = 0.7812
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
NOT normally distributed!
Mahalanobis distances
Chi-square quantiles
0.0
0.5
1.0
1.5
2.0
2.5
0
1
2
3
4
5
6
7


V7 & V10 alpha = 0.05
r = 0.9795
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
0
1
2
3
4
5
6
7


V8 & V9 alpha = 0.05
r = 0.9731
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles


V8 & V10 alpha = 0.05
r = 0.9532
Simulated r Value: 0.914480018303407
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles



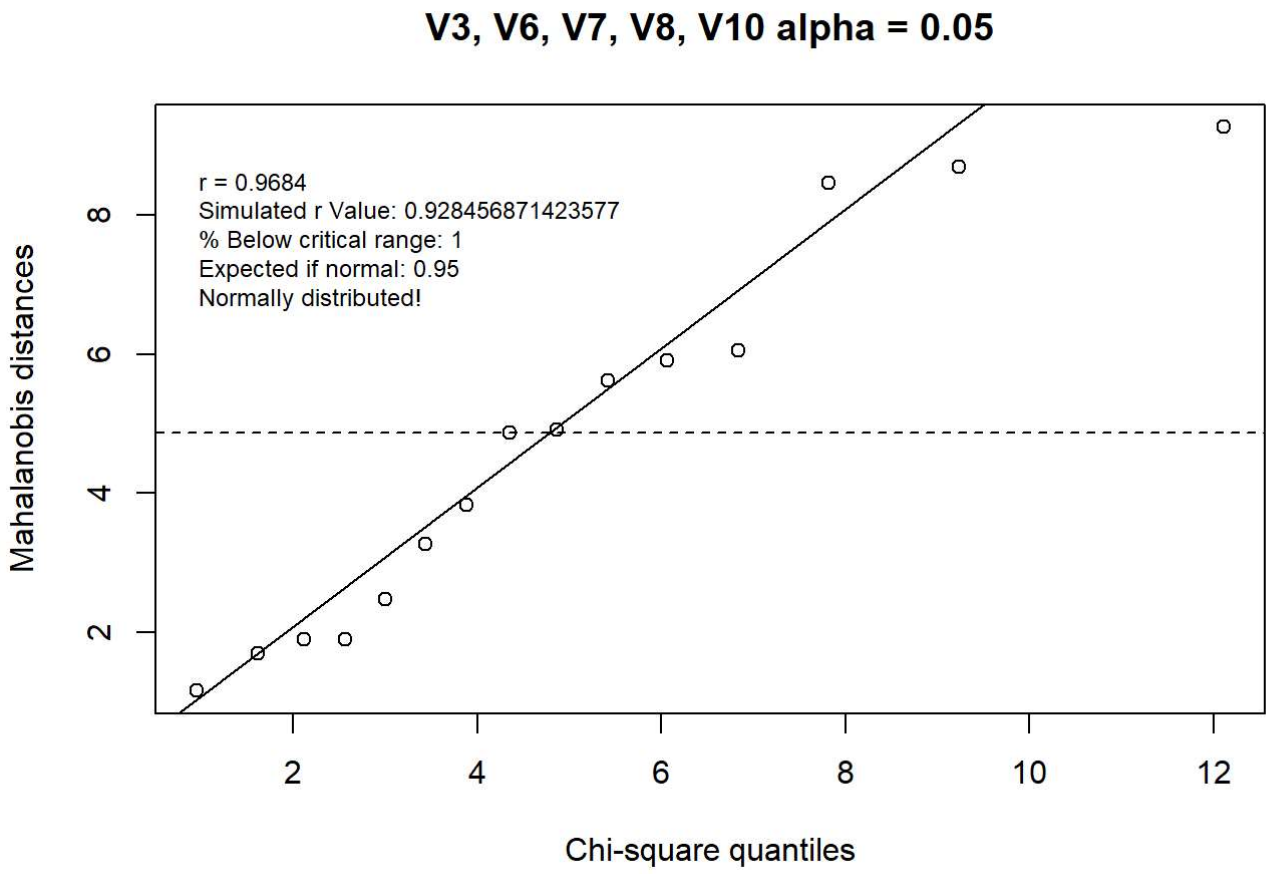
Okay, this suggest, that we need to remove V9 completely.

```
z_df <- z_df[, -5]
z_df
```

```
##          V3       V6        V7        V8      V10
## 1  21.97496 18.98157 0.3815453 10.646448 1.411687
## 6  21.97496 22.17264 0.3815453 11.831190 1.425310
## 9  20.76886 20.87777 0.6251913 16.032002 1.425310
## 10 24.24630 18.47035 0.3815453 14.222076 1.438345
## 13 25.32210 19.47668 1.1319389 16.032002 1.460042
## 14 20.76886 19.95701 0.9344521 17.245838 1.434662
## 16 20.76886 19.95701 0.9344521 18.465053 1.434662
## 18 16.77290 22.58421 1.6042980  6.568119 1.467938
## 22 21.97496 16.82356 1.1319389 10.057017 1.449105
## 27 23.13217 21.75148 1.2736949 14.823879 1.454614
## 29 27.37378 17.39327 0.9344521 18.465053 1.449105
## 31 25.32210 20.87777 0.9344521 23.390255 1.441548
## 32 19.50654 18.47035 0.9344521 19.689353 1.425310
## 34 23.13217 20.42370 1.2736949 14.823879 1.461160
## 40 20.76886 17.39327 0.6251913 12.426341 1.427954
```

All attributes

```
sim_cor <- FindcrikChi(n = length(z_df[,1]),
                       p = length(colnames(z_df)),
                       alpha = 0.05,
                       10000)

multi_var_norm(z_df, sim_cor, 0.05, "V3, V6, V7, V8, V10")
```

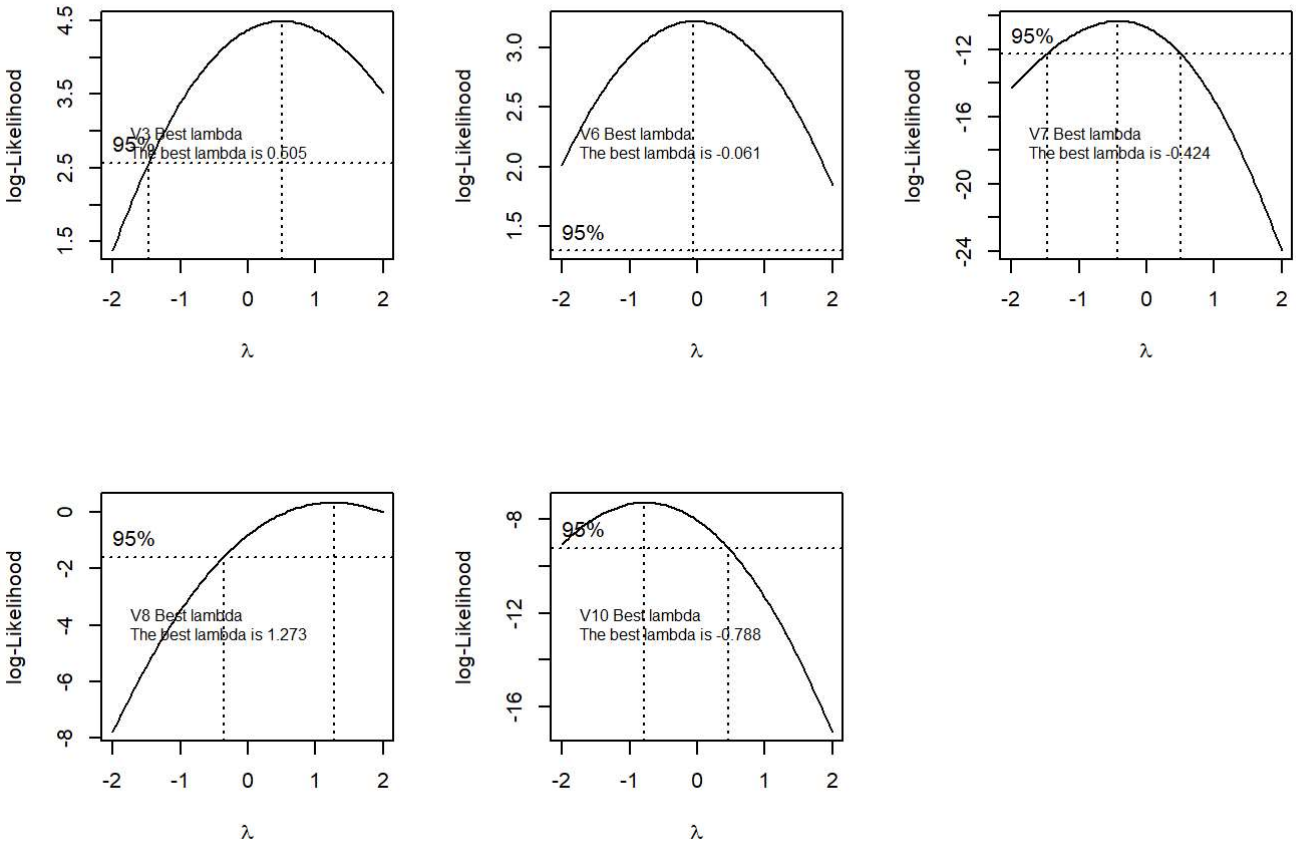


Okay, so we need to drop V9, and then we have a multivariate distribution, however, remember, that we removed some rows where V0 was 0, so let us now, start from the complete beginning and do the same test:

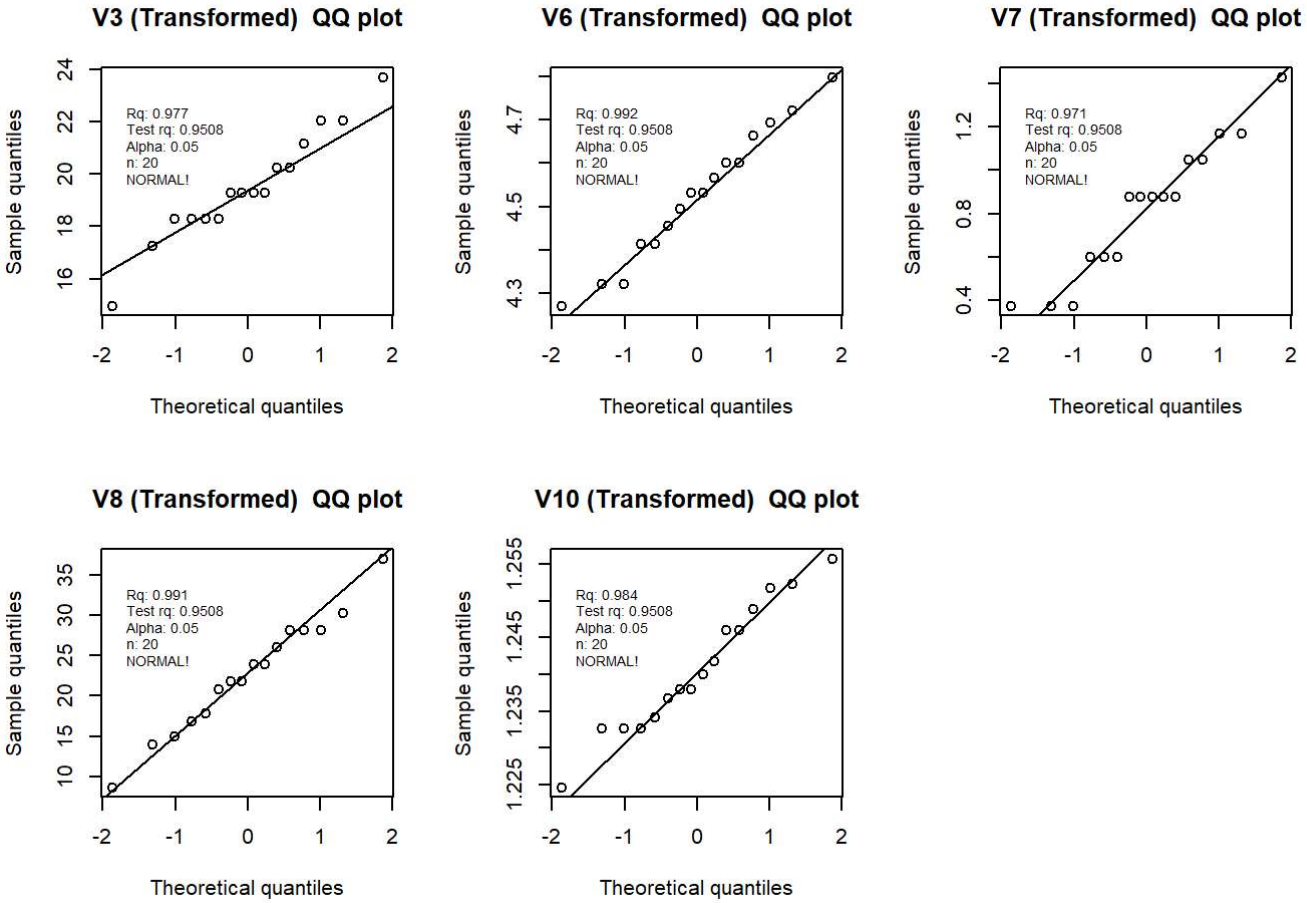
```
z_df <- df[, -5]

i = 1
for (colname in colnames(z_df)){
  z_df <- subset(z_df, z_df[,i] > 1)
  i = i + 1
}
z_df
```

```
##     V3  V6  V7   V8 V10
## 1  110 180 1.5 10.5  70
## 2  110 290 2.0 17.0 105
## 6  110 250 1.5 11.5  90
## 9  100 220 2.0 15.0  90
## 10 130 170 1.5 13.5 120
## 13 140 190 4.0 15.0 230
## 14 100 200 3.0 16.0 110
## 16 100 200 3.0 17.0 110
## 18  70 260 9.0  7.0 320
## 22 110 140 4.0 10.0 160
## 27 120 240 5.0 14.0 190
## 29 160 150 3.0 17.0 160
## 31 140 220 3.0 21.0 130
## 32  90 170 3.0 18.0  90
## 34 120 210 5.0 14.0 240
## 40 100 150 2.0 12.0  95
```

```
par(mfrow= c(2, 3))
i = 1
for (colname in colnames(z_df)){
  z_df[, i] <- box_cox_transformation(z_df[, i], name = colname)
  i = i + 1
}


i = 1
par(mfrow= c(2, 3))
```



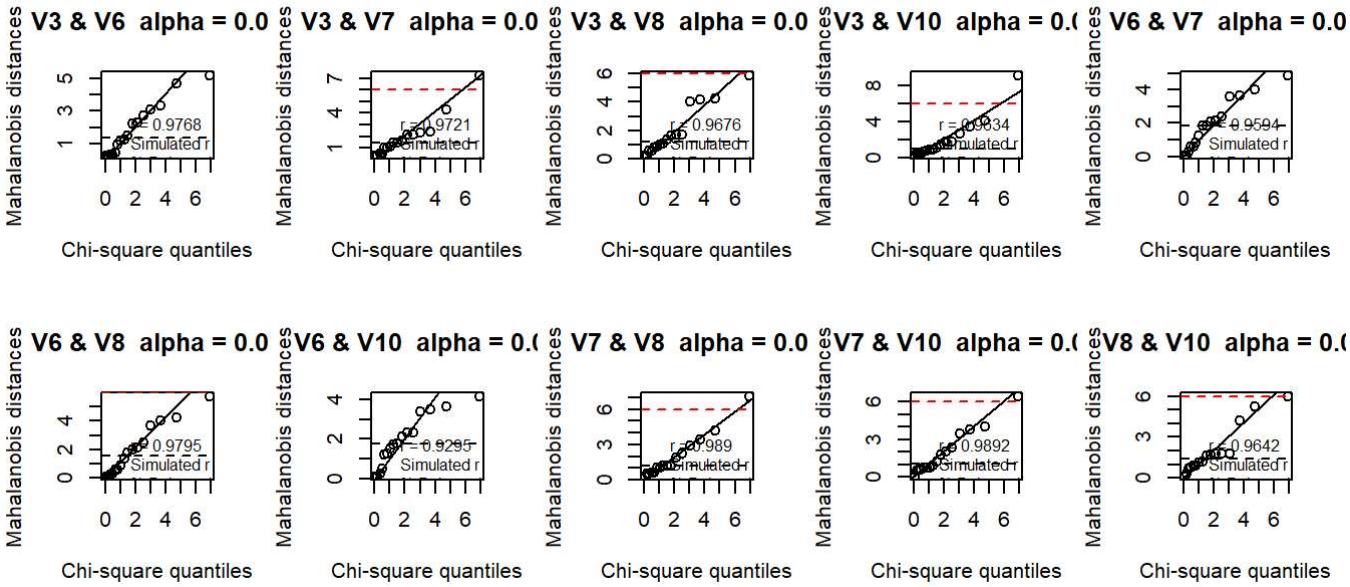
```
for (colname in colnames(z_df)){
  test_norm(z_df[, i], name = paste0(colname, " (Transformed) "))
  i = i + 1
}
```



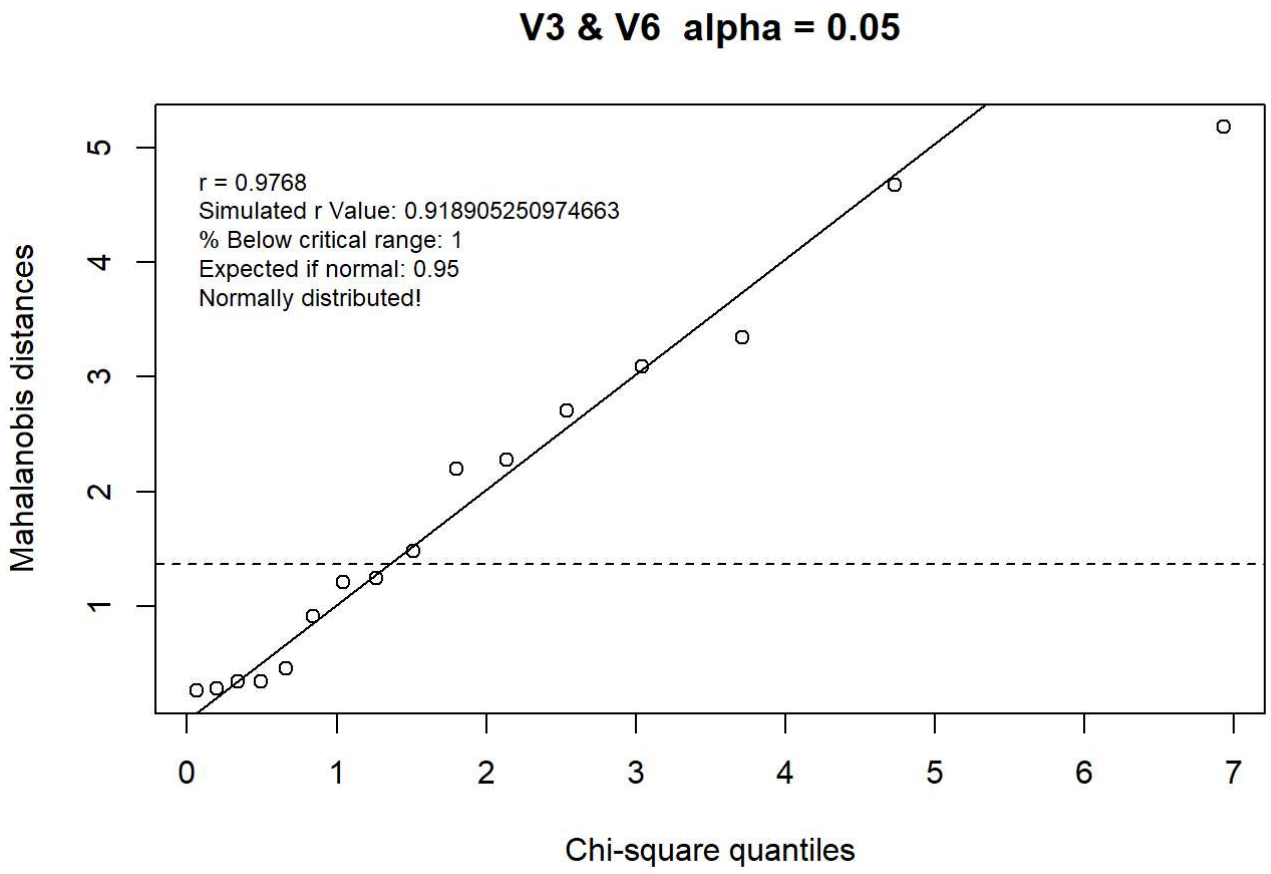
Yep they are still normal.

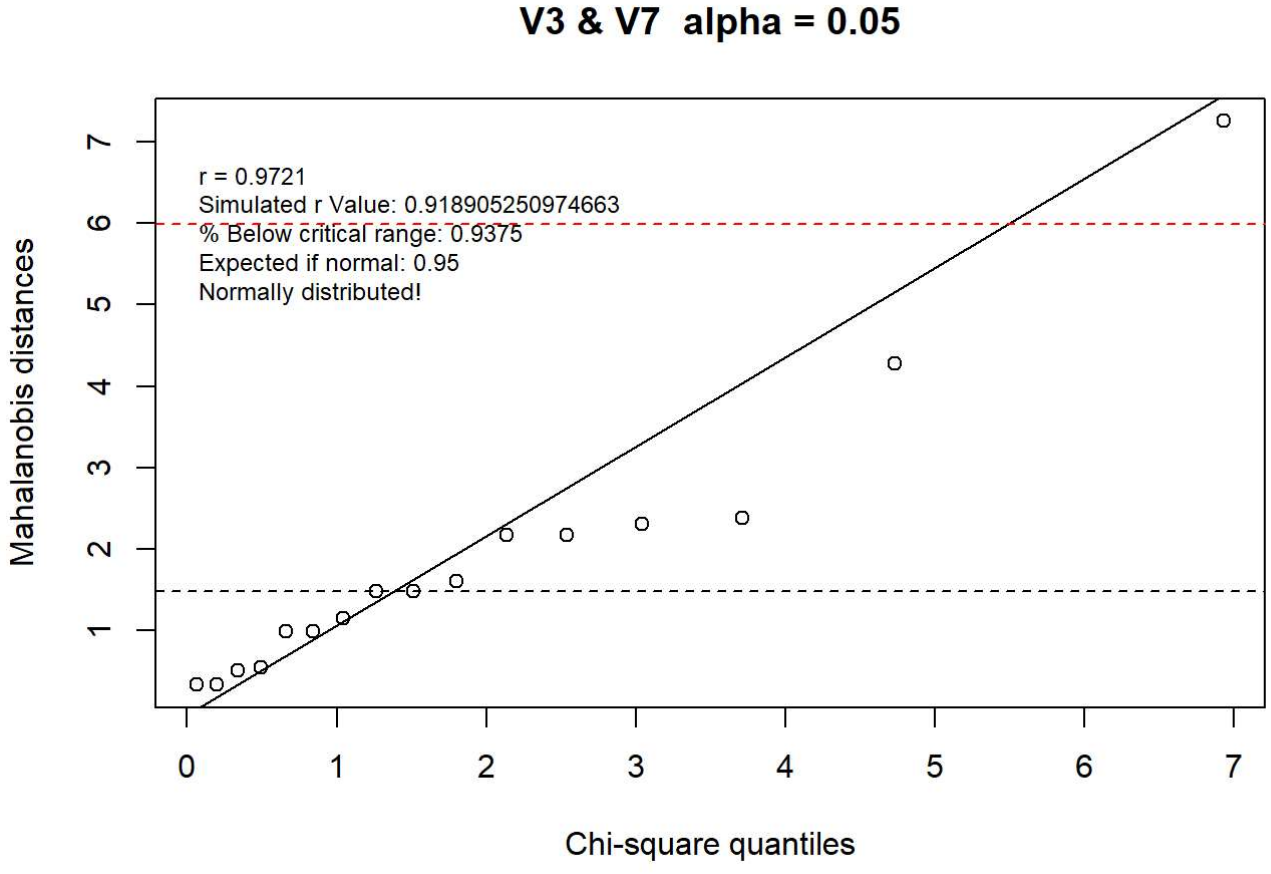
Let us test bivariate normality:

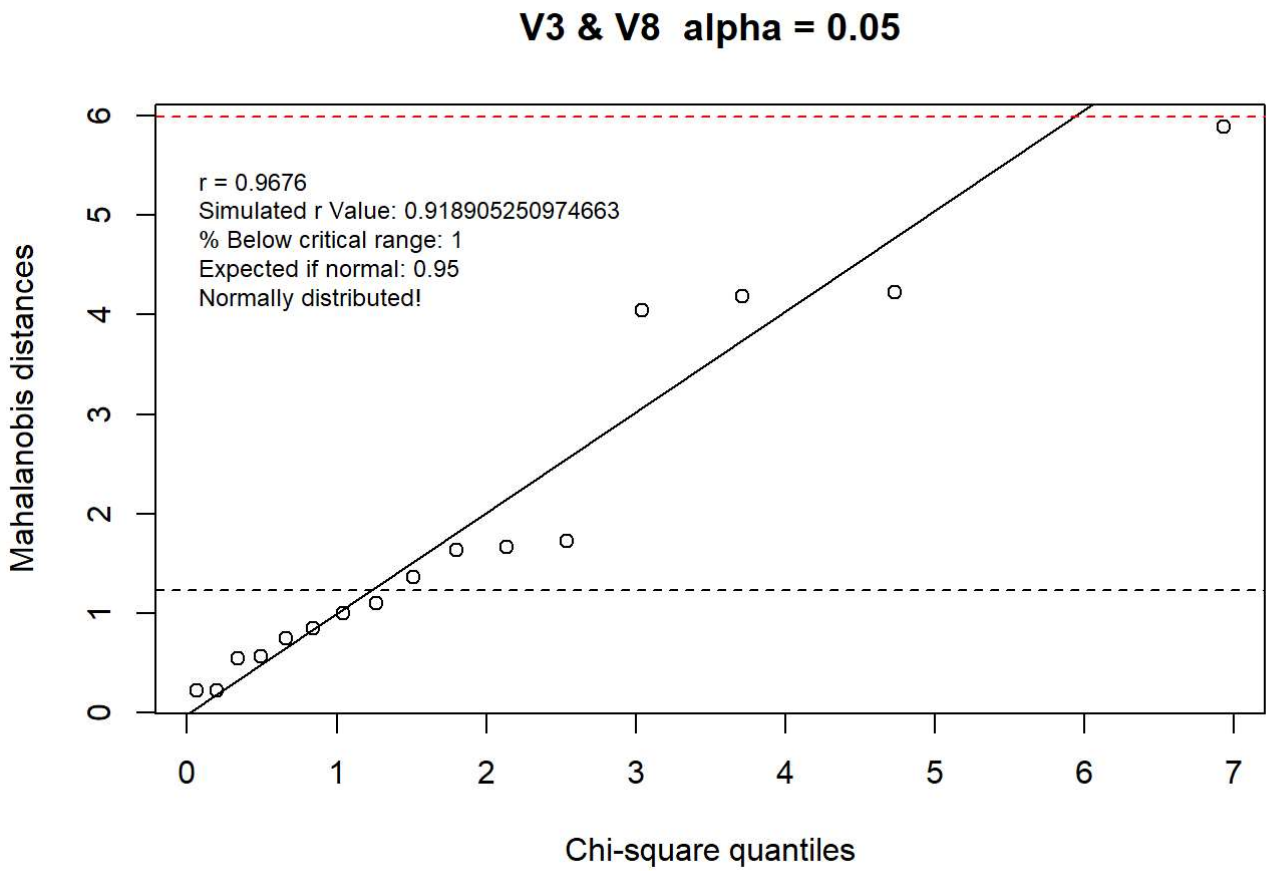
```
par(mfrow = c(3, 5))
bivar_pairs(z_df)
```

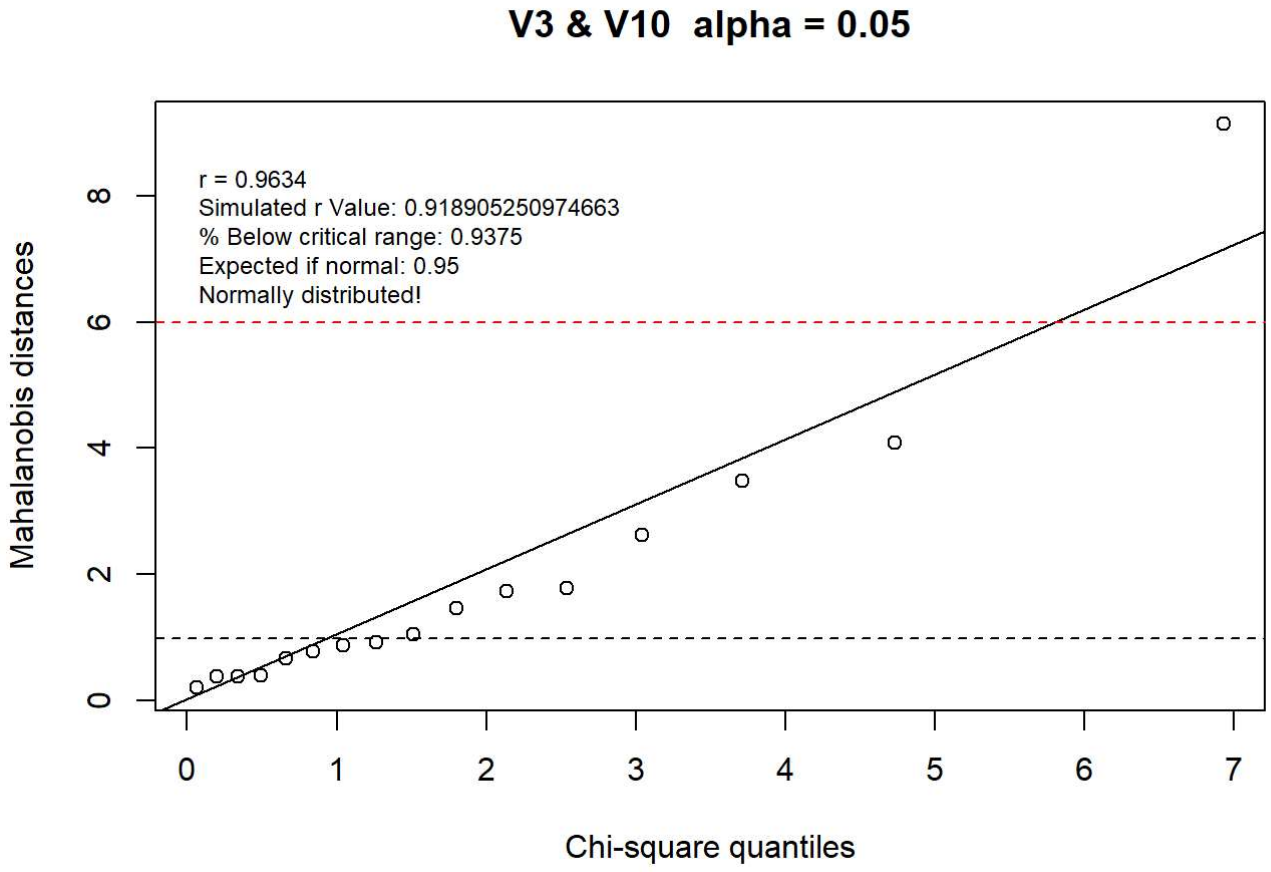


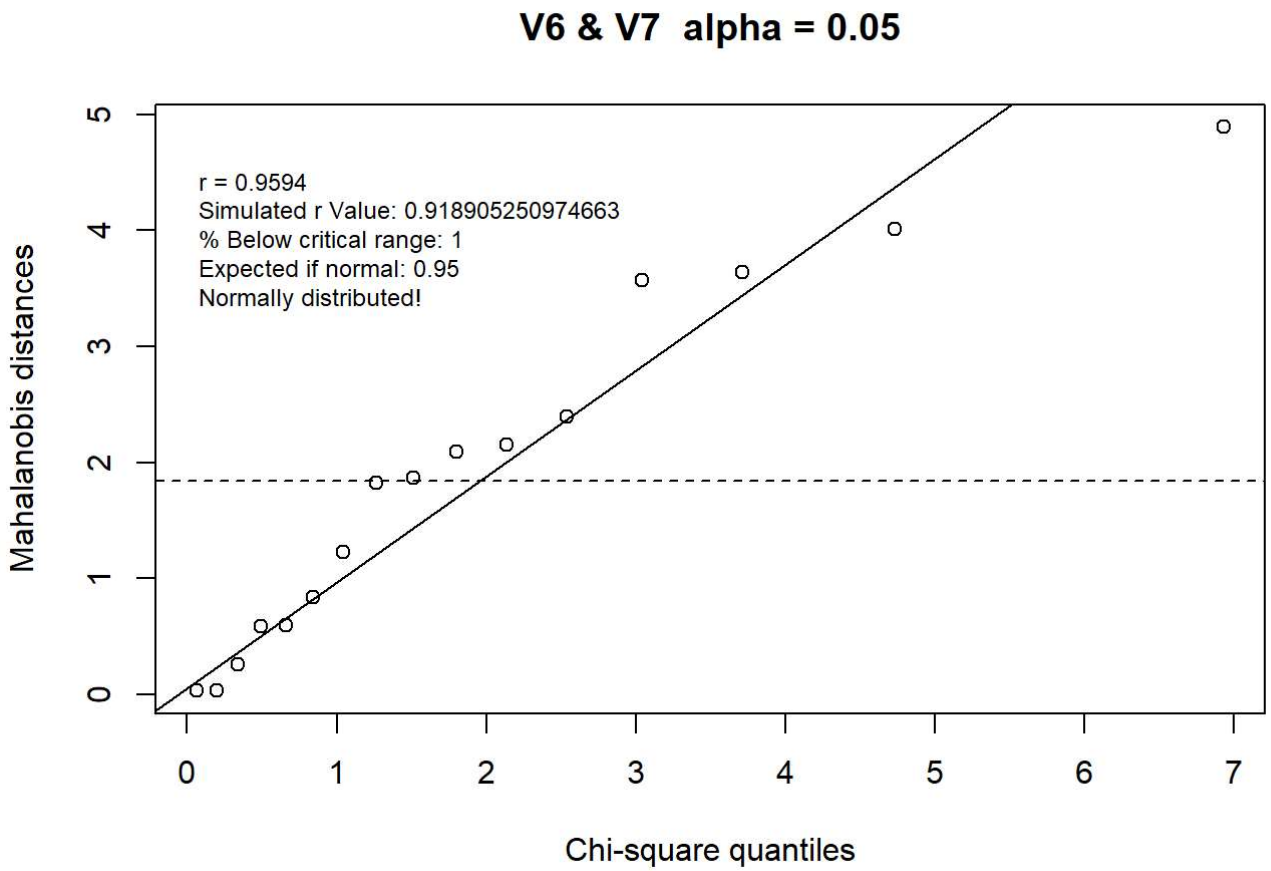

```
bivar_pairs(z_df)
```

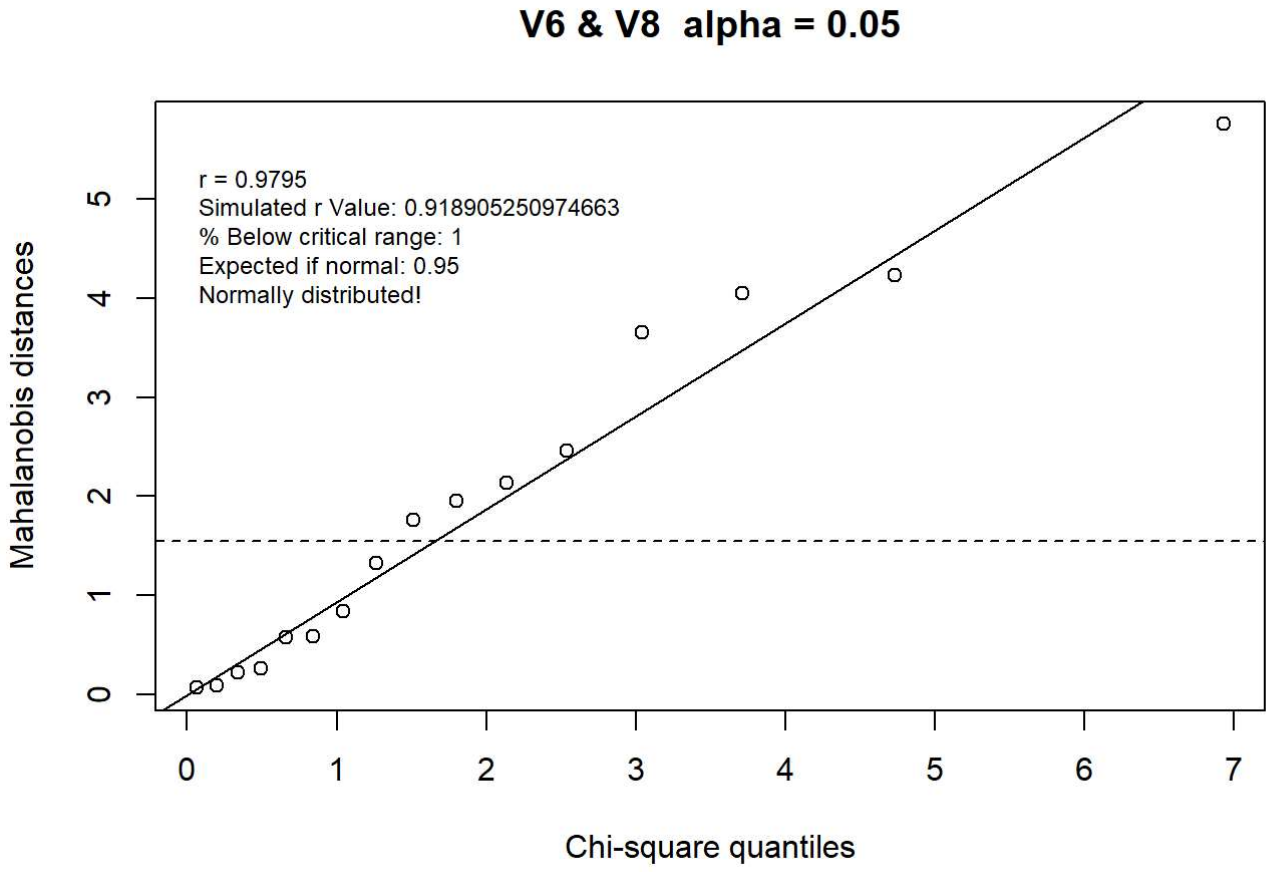

V3 & V6 alpha = 0.05
r = 0.9768
Simulated r Value: 0.918905250974663
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles

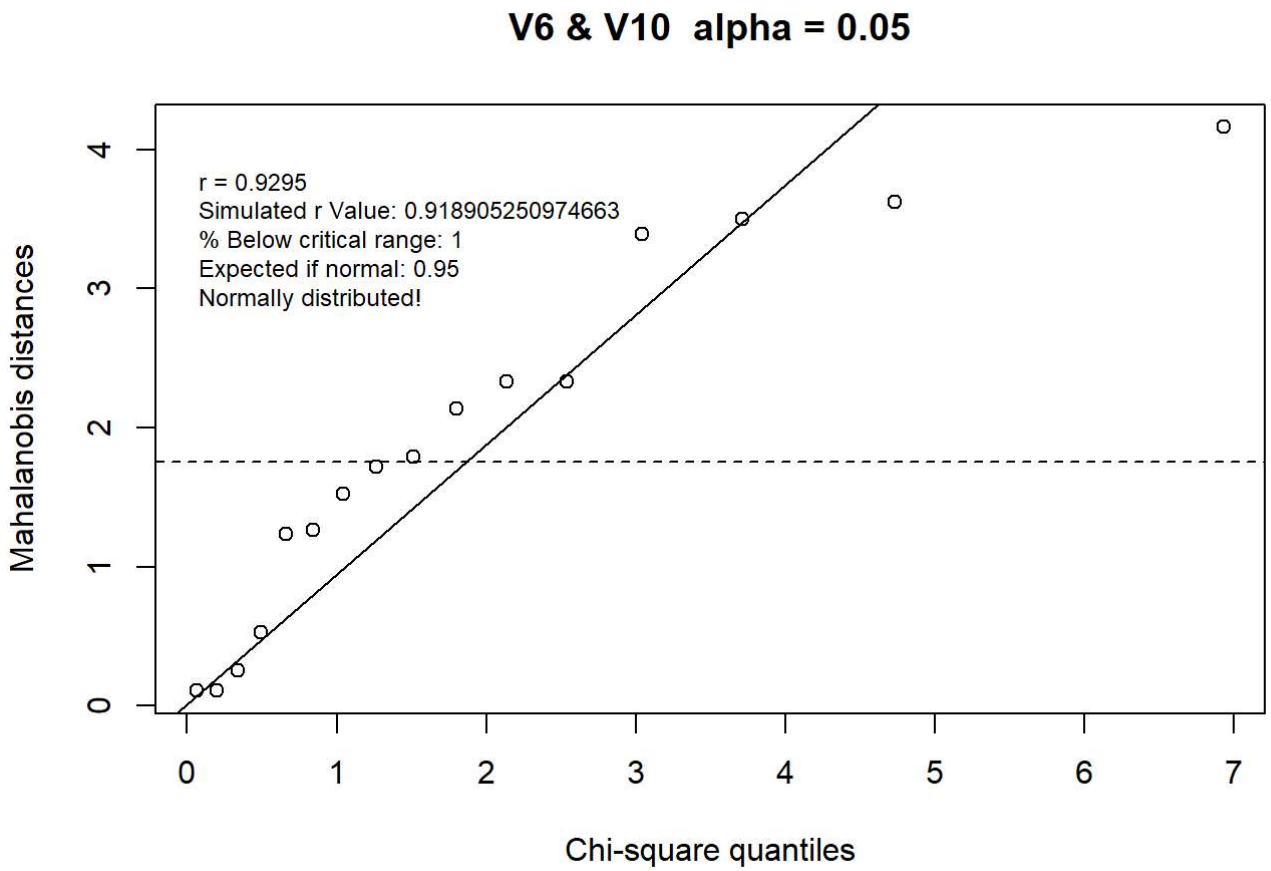

V3 & V7 alpha = 0.05
r = 0.9721
Simulated r Value: 0.918905250974663
% Below critical range: 0.9375
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles

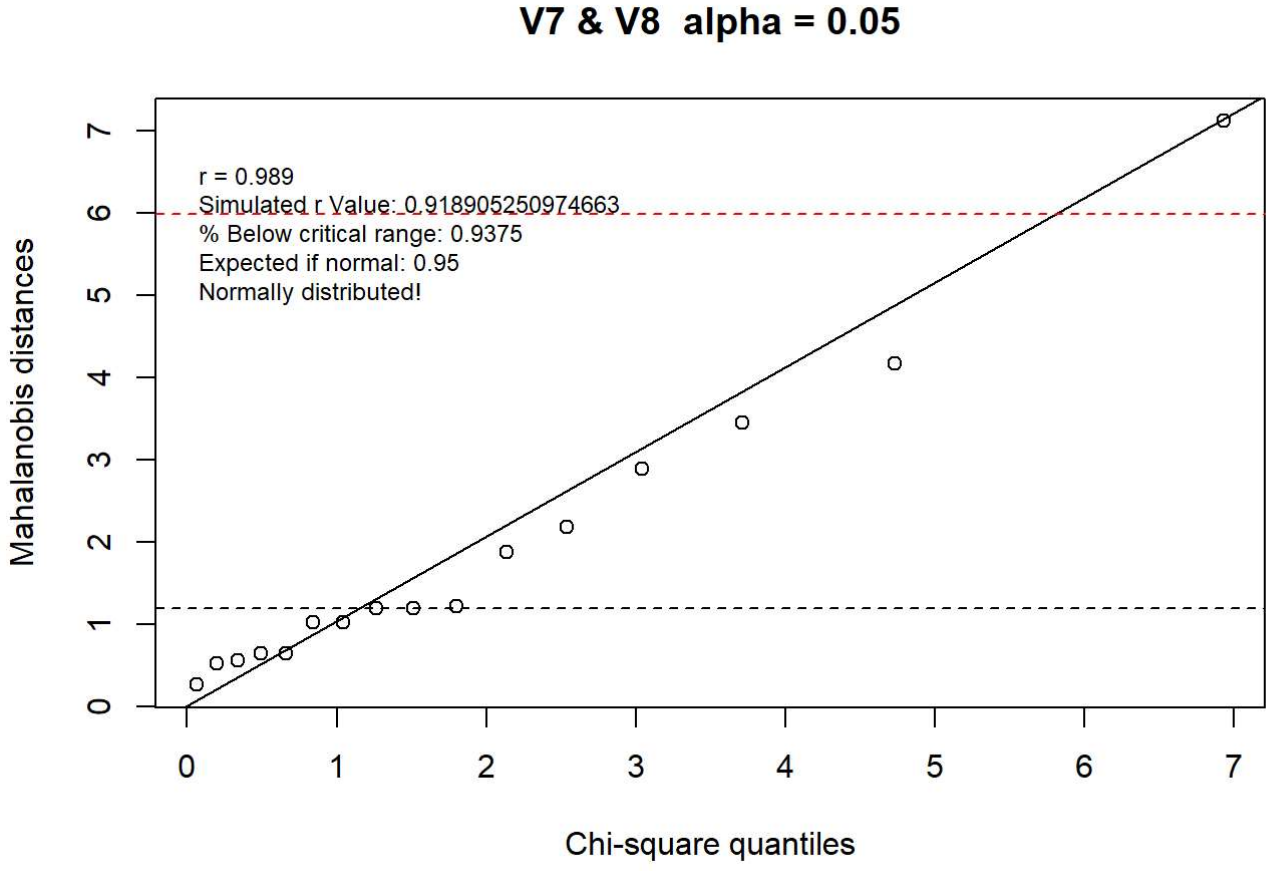

V3 & V8 alpha = 0.05
r = 0.9676
Simulated r Value: 0.918905250974663
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

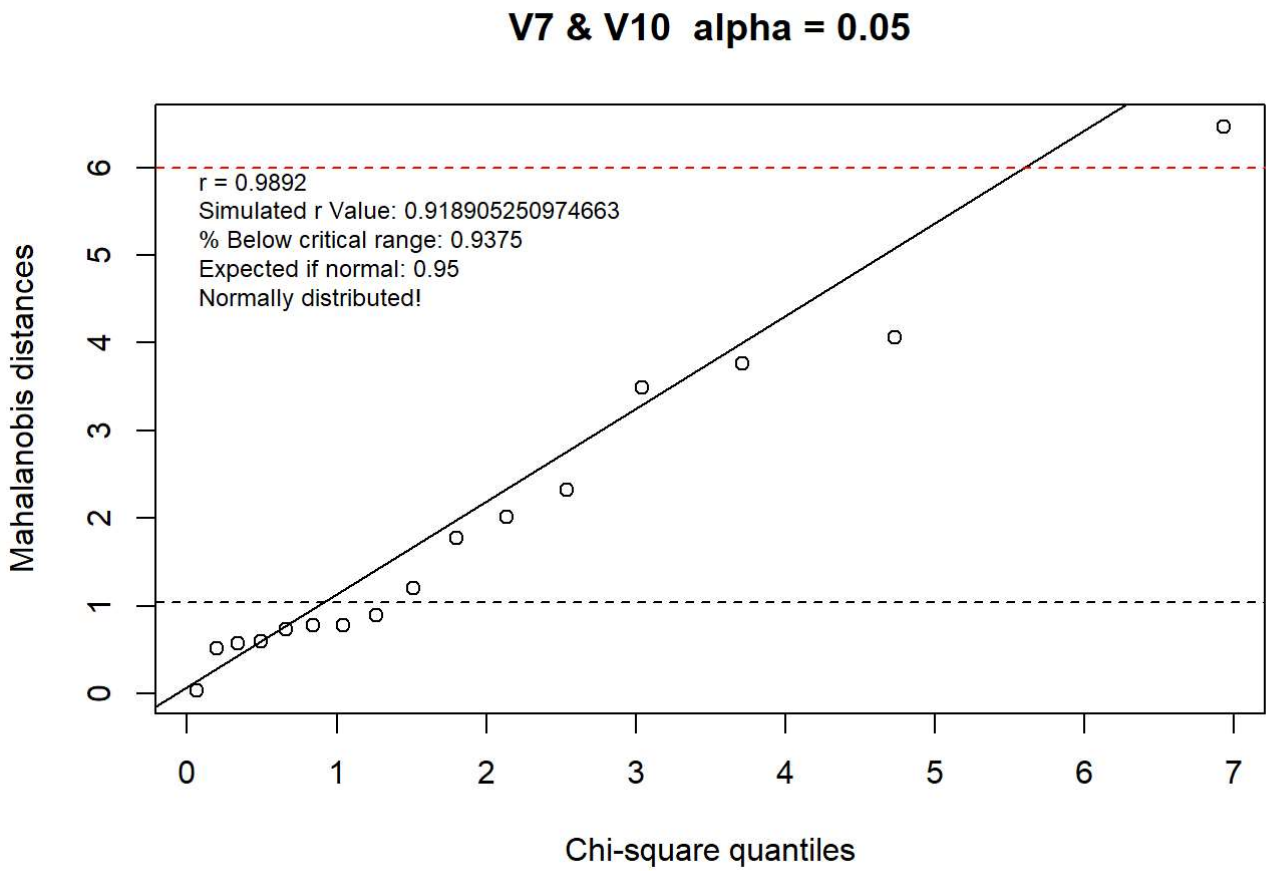

V3 & V10 alpha = 0.05
r = 0.9634
Simulated r Value: 0.918905250974663
% Below critical range: 0.9375
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
2
4
6
8
0
1
2
3
4
5
6
7

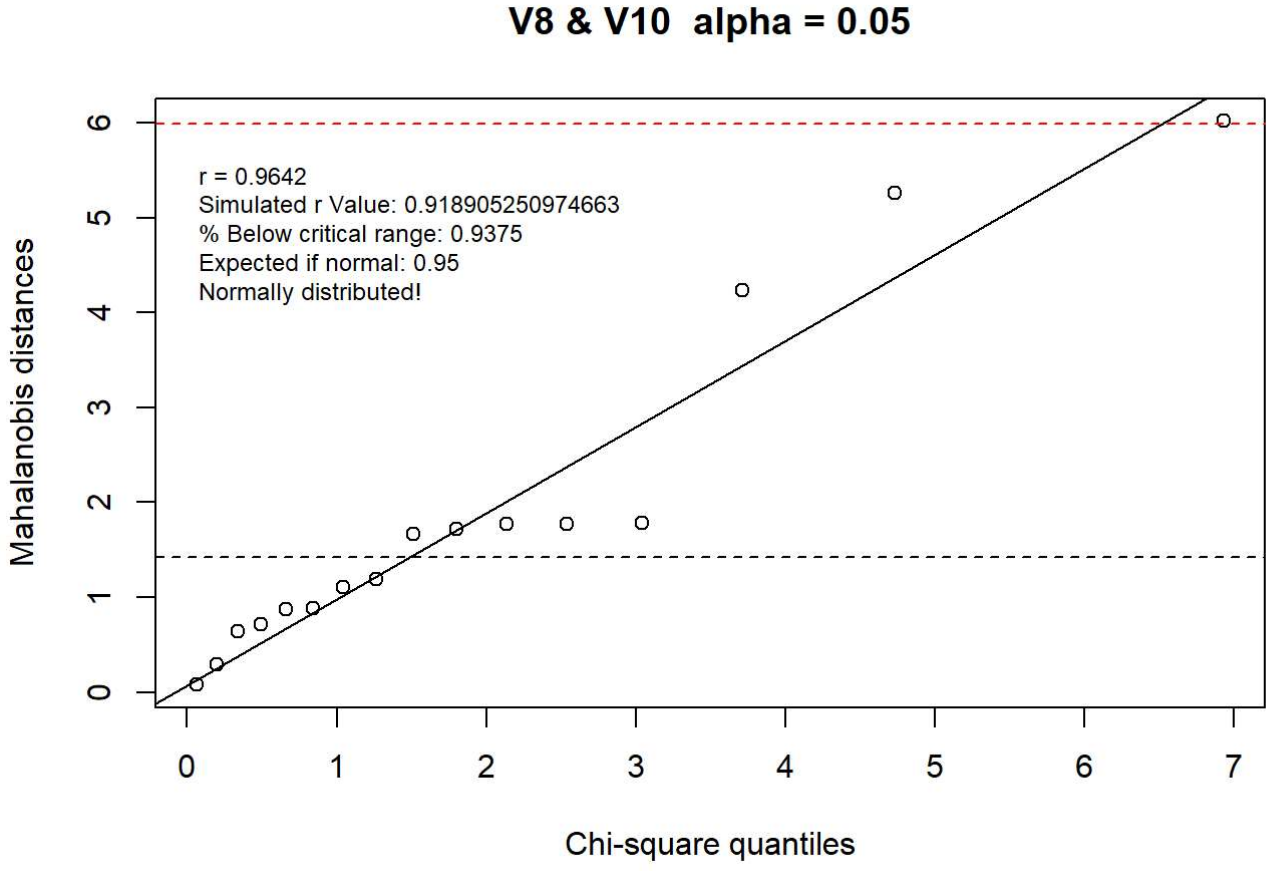

V6 & V7 alpha = 0.05
r = 0.9594
Simulated r Value: 0.918905250974663
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7


V6 & V8 alpha = 0.05
r = 0.9795
Simulated r Value: 0.918905250974663
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7


V6 & V10 alpha = 0.05
r = 0.9295
Simulated r Value: 0.918905250974663
% Below critical range: 1
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7

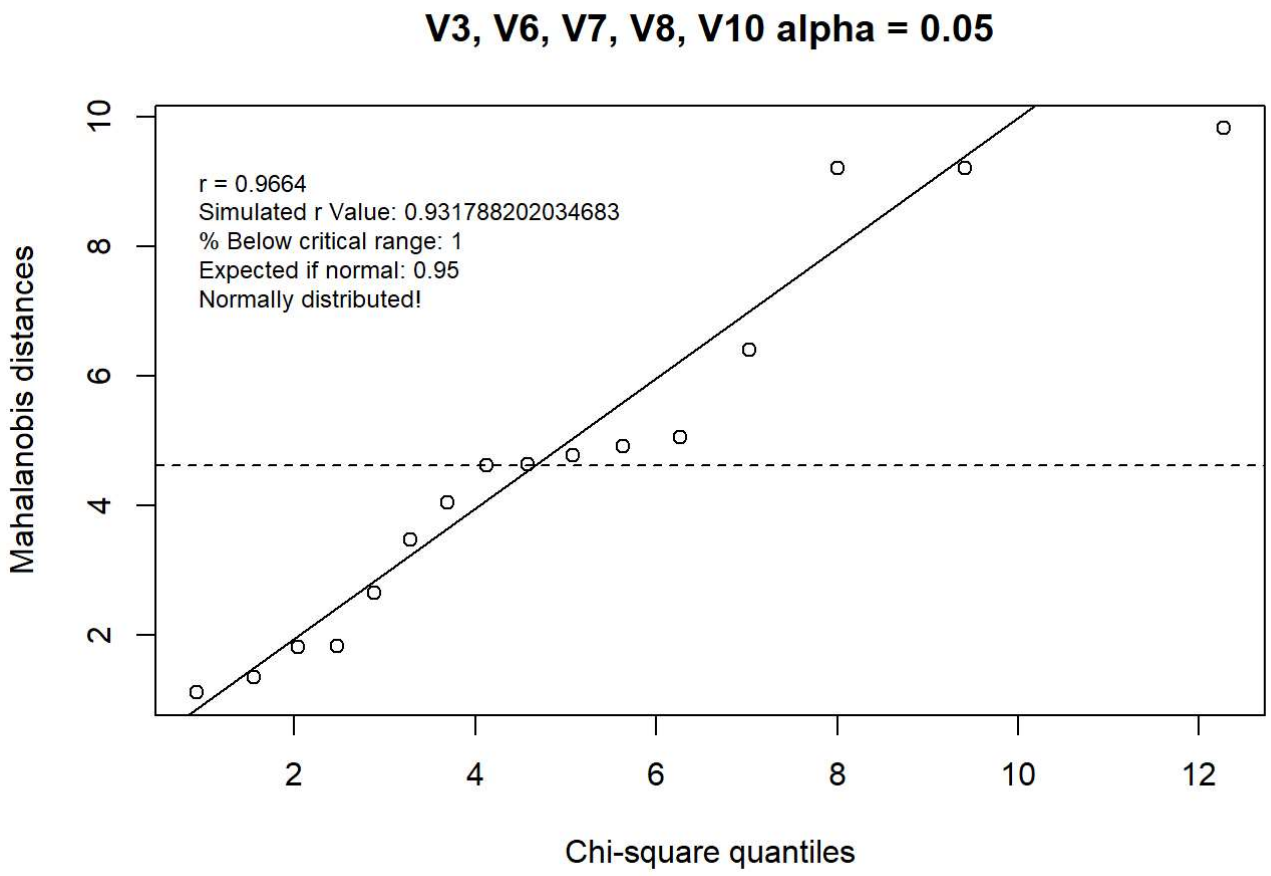

V7 & V8 alpha = 0.05
r = 0.989
Simulated r Value: 0.918905250974663
% Below critical range: 0.9375
Expected if normal: 0.95
Normally distributed!
Mahalanobis distances
Chi-square quantiles
0
1
2
3
4
5
6
7





All the pairs are normal. Let us check for all attributes:

```
sim_cor <- FindcrikChi(n = length(z_df[,1]),
                       p = length(colnames(z_df)),
                       alpha = 0.05,
                       10000)

multi_var_norm(z_df, sim_cor, 0.05, "V3, V6, V7, V8, V10")
```



They are likewise normally distributed.

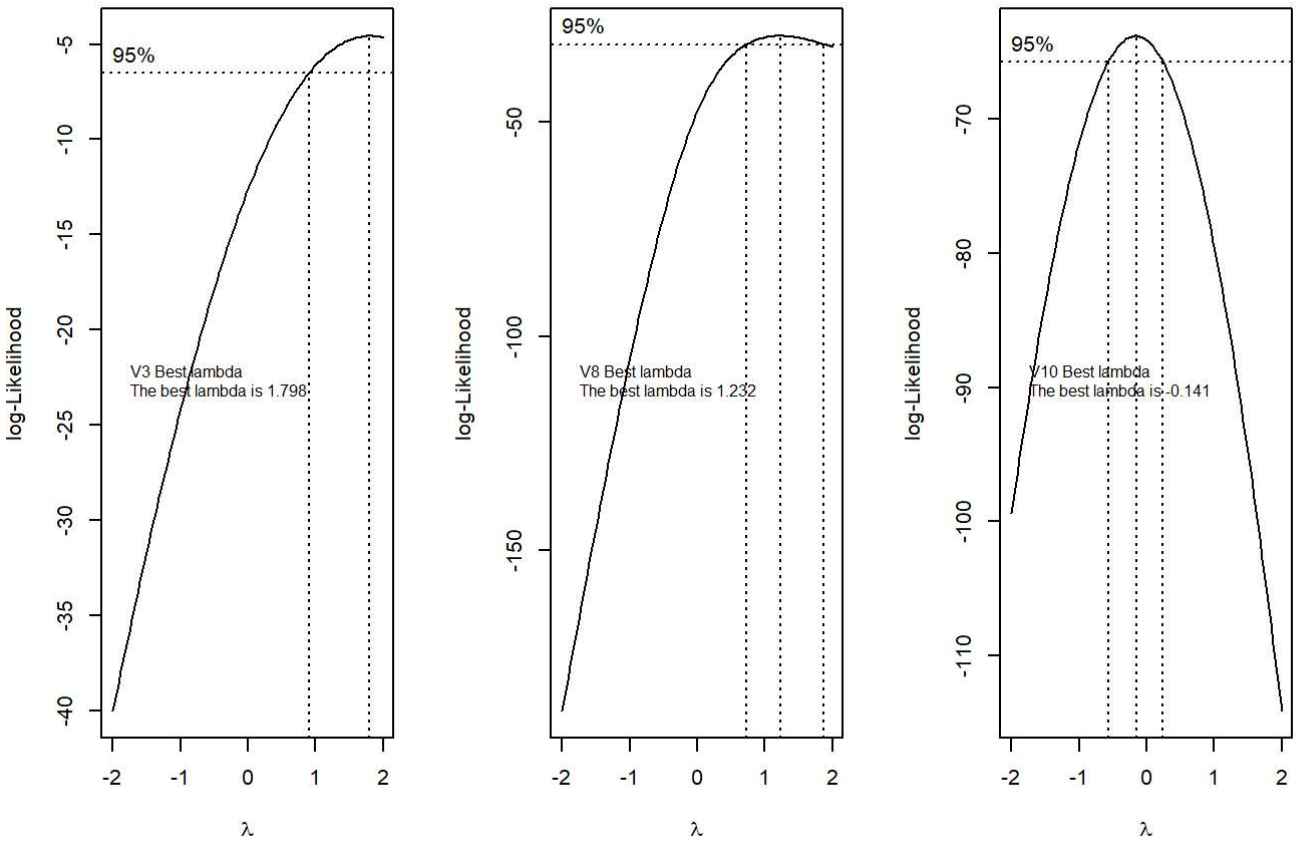
Conclusion

This gives us three options
- option 1: Use the dataframe with V3, V6, V7, V8, V9, V10, without any transformations. This is not normally distributed, but retains most of its attributes.
- Option 2: Use the data frame V3, V8, V10 transformed, since these are normally distributeed
- Option 3: Use the data frame V3, V6, V7, V8, V10 as transformed and where the 0s are removed from from V6, V7 since these are normally distributed.

```
# option 1
opt_1 <- df
opt_1
```

```
##     V3  V6  V7   V8 V9 V10
## 1  110 180 1.5 10.5 10  70
## 2  110 290 2.0 17.0  1 105
## 3  110 180 0.0 12.0 13  55
## 4  110 180 0.0 12.0 13  65
## 5  110 280 0.0 15.0  9  45
## 6  110 250 1.5 11.5 10  90
## 7  110 260 0.0 21.0  3  40
## 8  110 180 0.0 12.0 12  55
## 9  100 220 2.0 15.0  6  90
## 10 130 170 1.5 13.5 10 120
## 12 110 200 0.0 21.0  3  35
## 13 140 190 4.0 15.0 14 230
## 14 100 200 3.0 16.0  3 110
## 15 110 140 0.0 13.0 12  25
## 16 100 200 3.0 17.0  3 110
## 17 110 200 1.0 16.0  8  60
## 18  70 260 9.0  7.0  5 320
## 19 110 125 1.0 11.0 14  30
## 20 100 290 1.0 21.0  2  35
## 21 110  90 1.0 13.0 12  20
## 22 110 140 4.0 10.0  7 160
## 23 110 220 1.0 21.0  3  30
## 24 110 125 1.0 11.0 13  30
## 25 110 200 1.0 14.0 11  25
## 26 100   0 3.0 14.0  7 100
## 27 120 240 5.0 14.0 12 190
## 28 110 170 1.0 17.0  6  60
## 29 160 150 3.0 17.0 13 160
## 30 120 190 0.0 15.0  9  40
## 31 140 220 3.0 21.0  7 130
## 32  90 170 3.0 18.0  2  90
## 33 100 320 1.0 20.0  3  45
## 34 120 210 5.0 14.0 12 240
## 35 110 290 0.0 22.0  3  35
## 36 110  70 1.0  9.0 15  40
## 37 110 230 1.0 16.0  3  55
## 38 120 220 0.0 12.0 12  35
## 39 120 220 1.0 12.0 11  45
## 40 100 150 2.0 12.0  6  95
## 41  50   0 0.0 13.0  0  15
## 42  50   0 1.0 10.0  0  50
## 43 100   0 2.7  1.0  1 110
```

```
#option 2
par(mfrow = c(1, 3))
opt_2 <- df[, c(1, 4, 6)]
i = 1
for (colname in colnames(opt_2)){
  opt_2[, i] <- box_cox_transformation(opt_2[, i], name = colname)
  i = i + 1
}
```


95%
V3 Best lambda
The best lambda is 1.798
log-Likelihood
λ
95%
V8 Best lambda
The best lambda is 1.232
log-Likelihood
λ
95%
V10 Best lambda
The best lambda is -0.141
log-Likelihood
λ

```
opt_2
```

```
##           V3        V8      V10
## 1  2603.1861 13.901844 3.193639
## 2  2603.1861 25.831711 3.409732
## 3  2603.1861 16.533572 3.059111
## 4  2603.1861 16.533572 3.152787
## 5  2603.1861 22.023468 2.943620
## 6  2603.1861 15.647317 3.329034
## 7  2603.1861 33.756737 2.874291
## 8  2603.1861 16.533572 3.059111
## 9  2193.1315 22.023468 3.329034
## 10 3515.3934 19.243029 3.478227
## 12 2603.1861 33.756737 2.794282
## 13 4016.5198 22.023468 3.794059
## 14 2193.1315 23.913758 3.433741
## 15 2603.1861 18.331685 2.585848
## 16 2193.1315 25.831711 3.433741
## 17 2603.1861 23.913758 3.108179
## 18 1154.6625  8.115602 3.943596
## 19 2603.1861 14.769971 2.700021
## 20 2193.1315 33.756737 2.794282
## 21 2603.1861 18.331685 2.442046
## 22 2603.1861 13.043271 3.621474
## 23 2603.1861 33.756737 2.700021
## 24 2603.1861 14.769971 2.700021
## 25 2603.1861 20.162250 2.585848
## 26 2193.1315 20.162250 3.384380
## 27 3044.1197 20.162250 3.704304
## 28 2603.1861 25.831711 3.108179
## 29 5106.5912 25.831711 3.621474
## 30 3044.1197 22.023468 2.874291
## 31 4016.5198 33.756737 3.518670
## 32 1814.5571 27.776068 3.329034
## 33 2193.1315 31.739563 2.943620
## 34 3044.1197 20.162250 3.813725
## 35 2603.1861 35.796358 2.794282
## 36 2603.1861 11.356283 2.874291
## 37 2603.1861 23.913758 3.059111
## 38 3044.1197 16.533572 2.794282
## 39 3044.1197 16.533572 2.943620
## 40 2193.1315 16.533572 3.357538
## 41  630.2979 18.331685 2.249828
## 42  630.2979 13.043271 3.004666
## 43 2193.1315  0.000000 3.433741
```

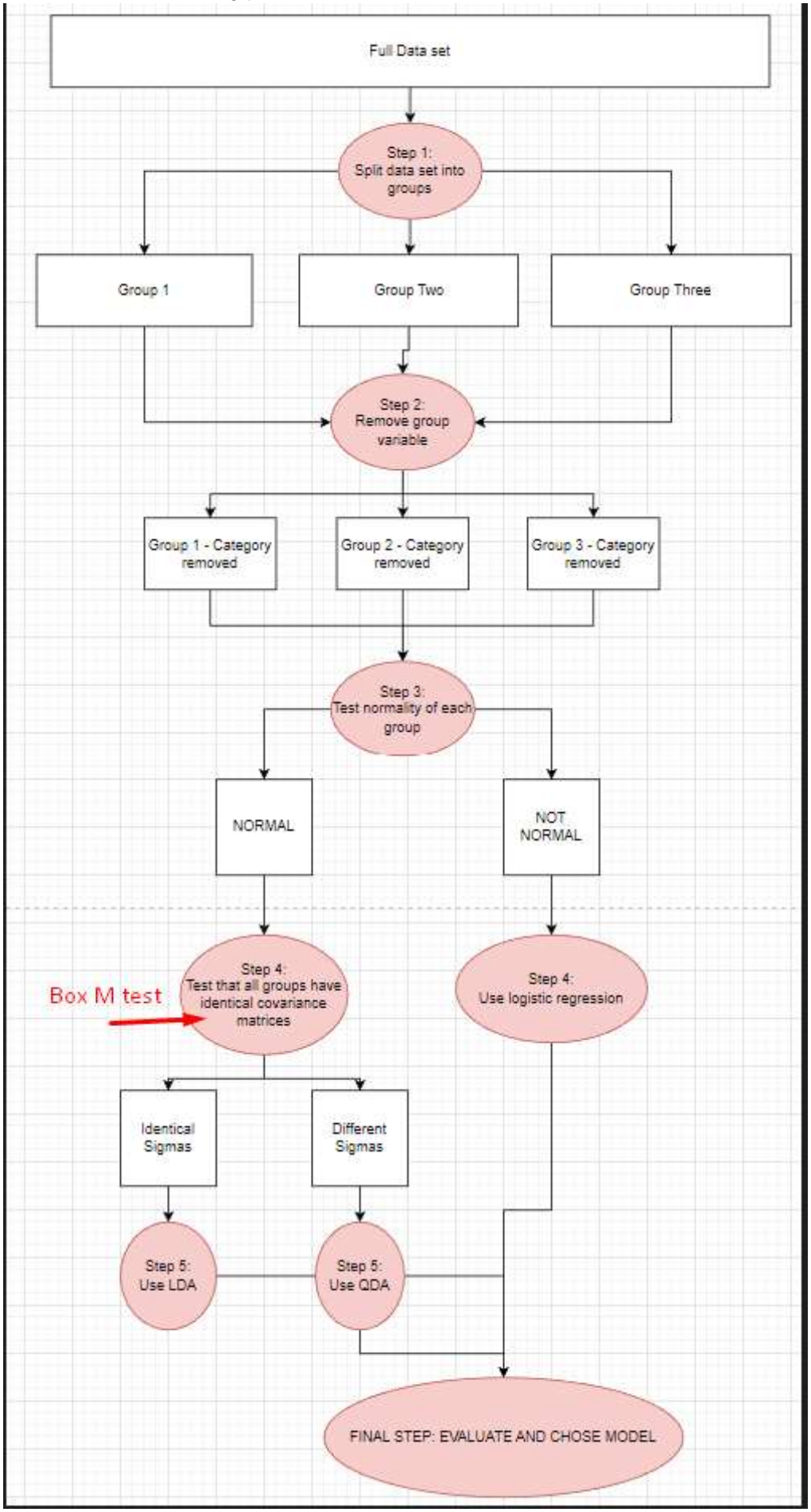
```
#option 3
opt_3 <- z_df
opt_3
```

```
##          V3       V6        V7        V8      V10
## 1  19.28530 4.455168 0.3725084 14.880358 1.224580
## 2  19.28530 4.798334 0.6005294 28.140430 1.236790
## 6  19.28530 4.692601 0.3725084 16.803390 1.232601
## 9  18.28591 4.600768 0.6005294 23.880816 1.232601
## 10 21.15734 4.413370 0.3725084 20.785335 1.240030
## 13 22.03974 4.494572 1.0480619 23.880816 1.251741
## 14 18.28591 4.531835 0.8781338 25.992462 1.237958
## 16 18.28591 4.531835 0.8781338 28.140430 1.237958
## 18 14.94516 4.720634 1.4291262  8.564982 1.255748
## 22 19.28530 4.270306 1.0480619 13.937137 1.245952
## 27 20.24065 4.663352 1.1663035 21.807242 1.248900
## 29 23.71550 4.321337 0.8781338 28.140430 1.245952
## 31 22.03974 4.600768 0.8781338 37.066318 1.241814
## 32 17.23571 4.413370 0.8781338 30.323153 1.232601
## 34 20.24065 4.567172 1.1663035 21.807242 1.252318
## 40 18.28591 4.321337 0.6005294 17.782411 1.234129
```

However, I have decided to only use option 2 for normal classifiers, since I believe that to many observatsion have been removed from option 3 (there are only 16 observations left out of the 42 observations).

Step 4: Find Optimal classifier

I will follow the following process:



Optimal Classifier

I Will first split and prepare my data:

```
z_opt_2 <- full_df
z_opt_2_cat <- full_df[, 11]

par(mfrow = c(1, 3))


opt_2_group_1 <- subset(z_opt_2, full_df[,11] == 1)
opt_2_group_2 <- subset(z_opt_2, full_df[,11] == 2)
opt_2_group_3 <- subset(z_opt_2, full_df[,11] == 3)

opt_2_group_1 <-na.omit(opt_2_group_1[, c(3, 8, 10, 11)])
opt_2_group_2 <-na.omit(opt_2_group_2[, c(3, 8, 10, 11)])
opt_2_group_3 <-na.omit(opt_2_group_3[, c(3, 8, 10, 11)])

opt_2_group_1
```

```
##     V3   V8 V10 V11
## 1  110 10.5  70   1
## 2  110 17.0 105   1
## 3  110 12.0  55   1
## 4  110 12.0  65   1
## 5  110 15.0  45   1
## 6  110 11.5  90   1
## 7  110 21.0  40   1
## 8  110 12.0  55   1
## 9  100 15.0  90   1
## 10 130 13.5 120   1
## 12 110 21.0  35   1
## 13 140 15.0 230   1
## 14 100 16.0 110   1
## 15 110 13.0  25   1
## 16 100 17.0 110   1
## 17 110 16.0  60   1
```

```
opt_2_group_2
```

```
##     V3 V8 V10 V11
## 18  70  7 320   2
## 19 110 11  30   2
## 20 100 21  35   2
## 21 110 13  20   2
## 22 110 10 160   2
## 23 110 21  30   2
## 24 110 11  30   2
## 25 110 14  25   2
## 26 100 14 100   2
## 27 120 14 190   2
## 28 110 17  60   2
## 29 160 17 160   2
## 30 120 15  40   2
## 31 140 21 130   2
## 32  90 18  90   2
## 33 100 20  45   2
## 34 120 14 240   2
## 35 110 22  35   2
## 36 110  9  40   2
## 37 110 16  55   2
```

```
opt_2_group_3
```

```
##     V3 V8 V10 V11
## 38 120 12  35   3
## 39 120 12  45   3
## 40 100 12  95   3
## 41  50 13  15   3
## 42  50 10  50   3
## 43 100  1 110   3
```

Now we need to test normality. However, it appears that V3 is causing problems, so we will remove it

```
opt_2_group_1
```

```
##     V3   V8 V10 V11
## 1  110 10.5  70   1
## 2  110 17.0 105   1
## 3  110 12.0  55   1
## 4  110 12.0  65   1
## 5  110 15.0  45   1
## 6  110 11.5  90   1
## 7  110 21.0  40   1
## 8  110 12.0  55   1
## 9  100 15.0  90   1
## 10 130 13.5 120   1
## 12 110 21.0  35   1
## 13 140 15.0 230   1
## 14 100 16.0 110   1
## 15 110 13.0  25   1
## 16 100 17.0 110   1
## 17 110 16.0  60   1
```

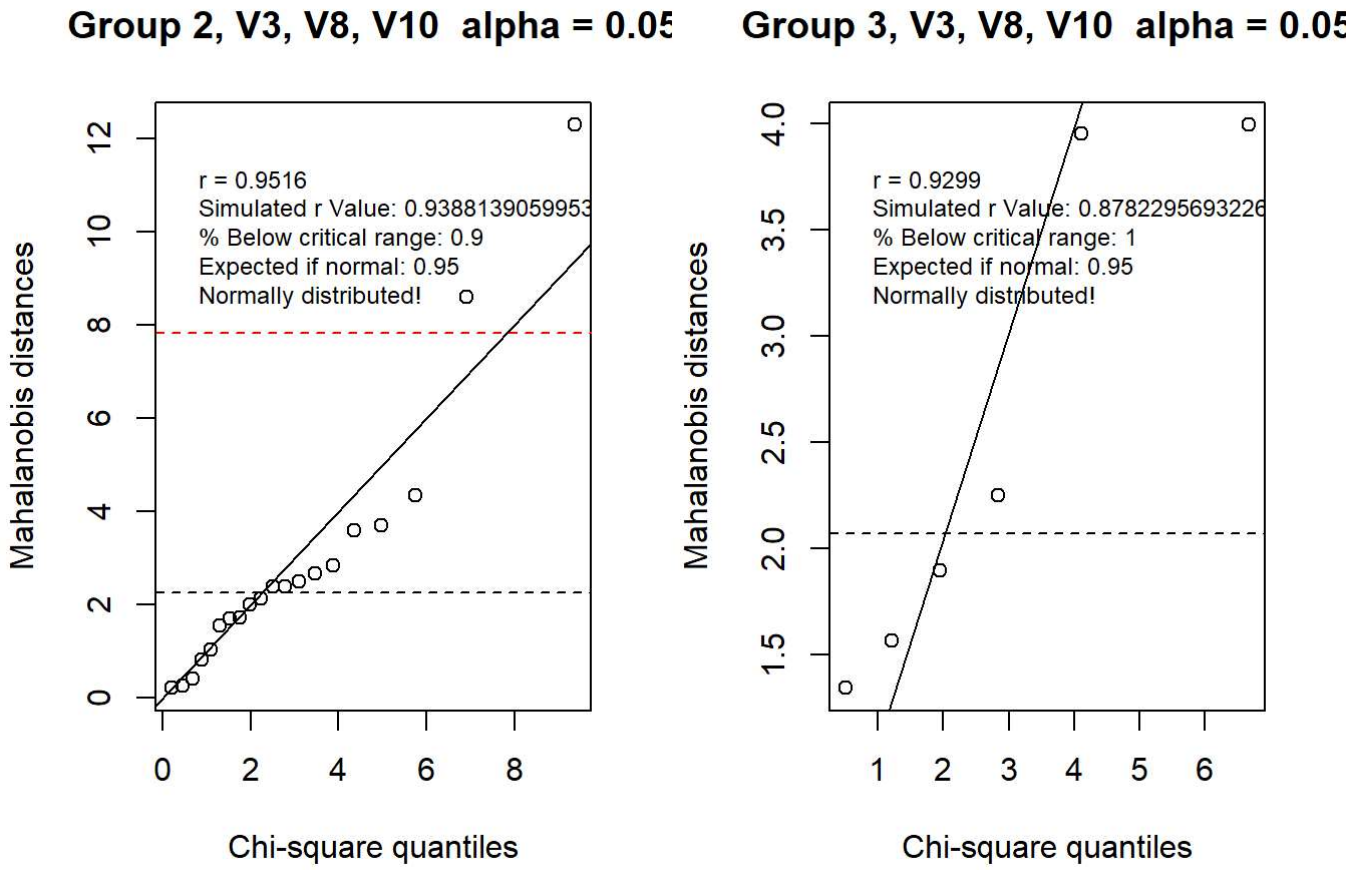
```
opt_2_group_2
```

```
##     V3 V8 V10 V11
## 18  70  7 320   2
## 19 110 11  30   2
## 20 100 21  35   2
## 21 110 13  20   2
## 22 110 10 160   2
## 23 110 21  30   2
## 24 110 11  30   2
## 25 110 14  25   2
## 26 100 14 100   2
## 27 120 14 190   2
## 28 110 17  60   2
## 29 160 17 160   2
## 30 120 15  40   2
## 31 140 21 130   2
## 32  90 18  90   2
## 33 100 20  45   2
## 34 120 14 240   2
## 35 110 22  35   2
## 36 110  9  40   2
## 37 110 16  55   2
```

```
opt_2_group_3
```

```
##     V3 V8 V10 V11
## 38 120 12  35   3
## 39 120 12  45   3
## 40 100 12  95   3
## 41  50 13  15   3
## 42  50 10  50   3
## 43 100  1 110   3
```

```
par(mfrow = c(1, 2))
# Group 1
sim_cor <- FindcrikChi(n = length(opt_2_group_1[,1]),
                       p = length(colnames(opt_2_group_1)),
                       alpha = 0.05,
                       10000)
#multi_var_norm(opt_2_group_1[, -4], sim_cor, 0.05, "Group 1, V3, V8, V10 ")

# Group 2
sim_cor <- FindcrikChi(n = length(opt_2_group_2[,1]),
                       p = length(colnames(opt_2_group_2)),
                       alpha = 0.05,
                       10000)
multi_var_norm(opt_2_group_2[, -4], sim_cor, 0.05, "Group 2, V3, V8, V10 ")

# Group 1
sim_cor <- FindcrikChi(n = length(opt_2_group_3[,1]),
                       p = length(colnames(opt_2_group_3)),
                       alpha = 0.05,
                       10000)
multi_var_norm(opt_2_group_3[, -4], sim_cor, 0.05, "Group 3, V3, V8, V10 ")
```



Since we can't check group one we should assume that it is not normally distributed, however in this case let us assume it is normal just for demonstration.

Now we know that the two groups are normal, so now we need to test their covariance matrices using the Box-m test for homogeneity
- h0: hypothesis: The covariance matrices are equal
- h1: The covariance matrices differ.

We calculate the C value and test if it is above a critical chi-square value using the function below

```
box_m_test <- function(df, group_index, significance = 0.95){
  #This takes the full dataframe and the index of the column which contains the groups

  n <- nrow(df)
  p <- ncol(df[, -group_index])
  groups <- unique(df[, group_index])
  g <- length(groups)

  # get w
  w <- 0
  for (group in groups){
    mask <- df[,group_index] == group
    group_df <- df[mask, ]
    n_g <- nrow(group_df)
    s_g <- cov(group_df[, -group_index])
    w <- w + (n_g - 1) * s_g
  }
  # get w
  spooled <- w/(n - g)

  # get M
  M_sum <- 0
  for (group in groups){
    mask <- df[,group_index] == group
    group_df <- df[mask, ]
    n_g <- nrow(group_df)
    s_g <- cov(group_df[, -group_index])
    M_sum <- ((n_g - 1) * log(det(s_g))) + M_sum
  }
  M <- (n-g)*log(det(spooled)) - M_sum

  # get u
  sum_u <- 0
  for (group in groups){
    mask <- df[,group_index] == group
    group_df <- df[mask, ]
    n_g <- nrow(group_df)
    s_g <- cov(group_df[, -group_index])
    sum_u <- sum_u + (1/(n_g - 1))
  }
  u <-  ( sum_u - 1/(n-g)) * ( (2*p^2 + 3*p - 1) / (6*(p+1)*(g-1)))

  # Test statistic
  C <- (1-u)*M

  # Critcal value v
  critvalue <- qchisq(significance,p*(p+1)*(g-1)/2)   #v=p*(p+1)*(g-1)/2#

  ### final decision ####
  decisionflag <- (C > critvalue)

  writeLines(paste0("u :                          ", u, "\n",
                    "M statistic:                 ", M, "\n",
                    "C value (Chi-squared value): ", C, "\n",
                    "Critival value (v):          ", critvalue, "\n",
```

```
                    "Equal covariance matrices:   ", decisionflag, "\n"))
}
```

Let us test the function, with significance 0.05

```
box_m_test(df = rbind(opt_2_group_1, opt_2_group_2, opt_2_group_3),
           group_index = 4)
```

```
## u :                          0.15906432748538
## M statistic:                 31.4002334672491
## C value (Chi-squared value): 26.4055764478972
## Critival value (v):          21.0260698174831
## Equal covariance matrices:   TRUE
```

With 0.01 significance:

```
box_m_test(df = rbind(opt_2_group_1, opt_2_group_2, opt_2_group_3),
           group_index = 4,
           significance = 0.99)
```

```
## u :                          0.15906432748538
## M statistic:                 31.4002334672491
## C value (Chi-squared value): 26.4055764478972
## Critival value (v):          26.2169673055358
## Equal covariance matrices:   TRUE
```

As we can see we have a c value, which is above the critical value, for significance of 0.06 and even 0.01 so we must accept the null hypothesis and conclude that the covariance matrices are true. This can likewise be implemented with a premade package:

```
library(heplots)
```

```
## Indlæser krævet pakke: car
```

```
## Indlæser krævet pakke: carData
```

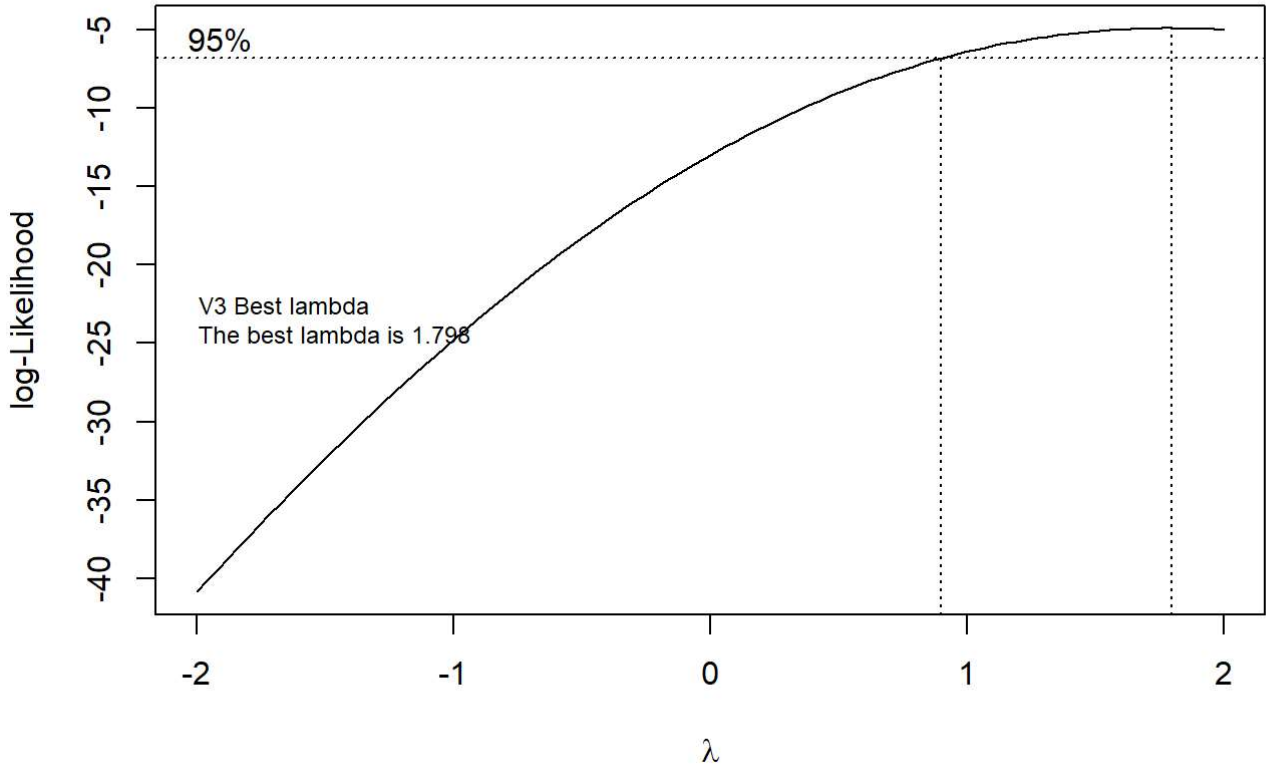
```
## Indlæser krævet pakke: broom
```

```
temp <- rbind(opt_2_group_1, opt_2_group_2, opt_2_group_3)
boxM(temp[, -4], temp$V11)
```

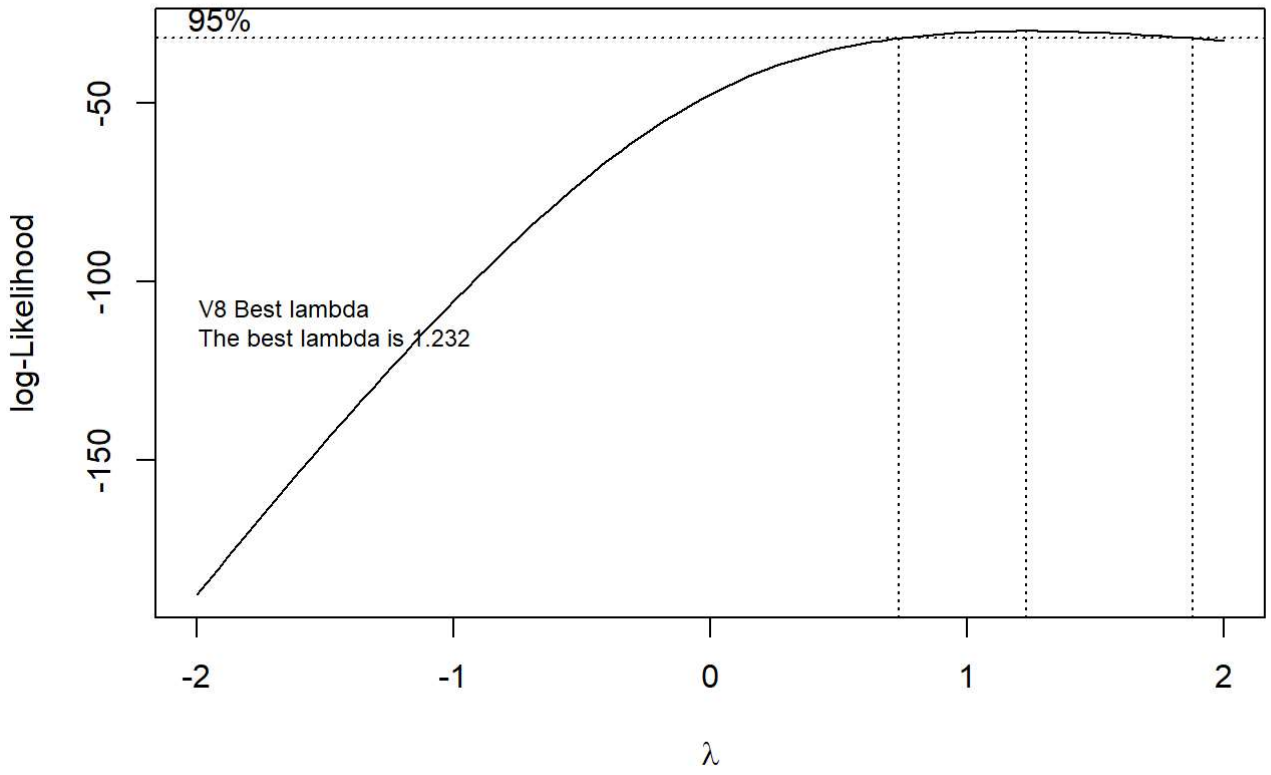
```
##
##  Box's M-test for Homogeneity of Covariance Matrices
##
## data:  temp[, -4]
## Chi-Sq (approx.) = 26.406, df = 12, p-value = 0.009401
```

This means, that we should use the LDA or Linear Discriminant Analysis, but it might not be best but in theory it is. What this does, like QDA, is that they try to Minimize ECM or the Expected costs of misclassification by creating a hyperplane which is used to seperate between the classes.

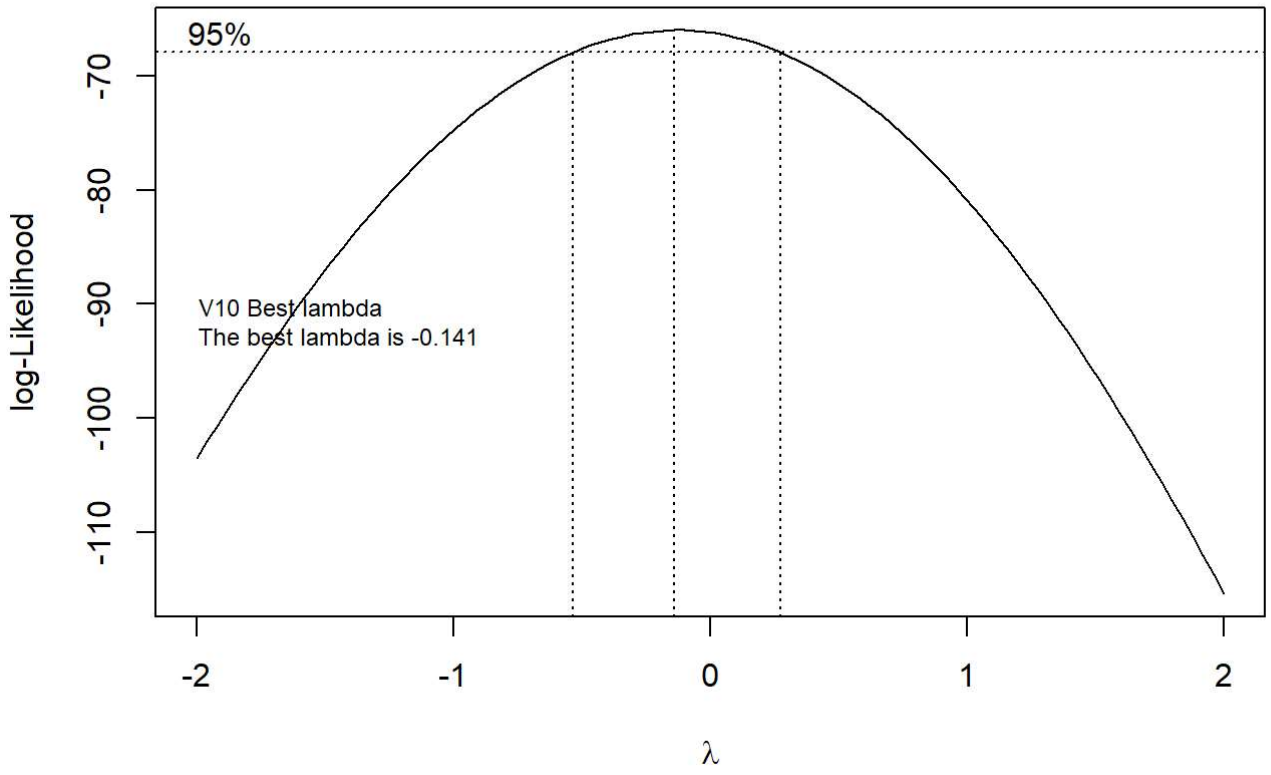
```
z_opt_2 <- full_df
z_opt_2[, 3] <- box_cox_transformation(z_opt_2[, 3], name = "V3")
```




```
z_opt_2[, 8] <- box_cox_transformation(z_opt_2[, 8], name = "V8")
```




```
z_opt_2[, 10] <- box_cox_transformation(z_opt_2[, 10], name = "V10")
```



```
z_opt_2 <- z_opt_2[-11,]
z_opt_2 <- z_opt_2[, c(3, 8, 10, 11)]
```

Creating classifers and evaluating

70 - 30 split and APER

We split into training and test data

```
z_opt_2_train <- z_opt_2[sample(1:nrow(z_opt_2), 30),] # 70%
z_opt_2_test <- z_opt_2[sample(1:nrow(z_opt_2), 12),] # 30%
```

We then have some simple evalutaion techniques such as APER (misclassifications divided by number of samples)

```
aper <- function(preds, obs){
  n_misclassifications <- 0
  i <- 1
  for (val in preds){

    if (val != obs[i]){
      n_misclassifications = n_misclassifications + 1
    }

    i = i + 1
  }
  return(n_misclassifications / length(preds))

}

acc <- function(preds, obs){
  correct <- 0
  i <- 1
  for (val in preds){

    if (val == obs[i]){
      correct = correct + 1
    }

    i = i + 1
  }
  return(correct / length(preds))
}

confusion_matrix <- function(preds, observed){
  # Alternatively user this code below
  # library(caret)
  # confusionMatrix(as.factor(predictions$class), as.factor(observed))
  return(table(preds, observed))
}
```

LDA

We start by creating the LDA model:

```
library(MASS)
lda_model <- lda(V11~., data = z_opt_2_train)
lda_model
```

```
## Call:
## lda(V11 ~ ., data = z_opt_2_train)
##
## Prior probabilities of groups:
##         1         2         3
## 0.3333333 0.4666667 0.2000000
##
## Group means:
##         V3       V8      V10
## 1 2794.735 21.67887 3.262363
## 2 2700.792 22.37316 3.071542
## 3 1955.850 13.49595 2.963946
##
## Coefficients of linear discriminants:
##               LD1           LD2
## V3  -0.0006687581 -0.0004536305
## V8  -0.1205319532 -0.0245463988
## V10 -0.8041670072  2.7739335512
##
## Proportion of trace:
##    LD1    LD2
## 0.9046 0.0954
```

The prior probabilities are used for the ECM, since they give weight to each of the groups, since group two is larger it has a Let us now predict.

```
lda_preds <- predict(lda_model, z_opt_2_test)
lda_preds
```

```
## $class
##  [1] 3 1 3 2 2 2 2 1 1 1 3 2
## Levels: 1 2 3
##
## $posterior
##             1          2           3
## 43 0.06003348 0.05638562 0.883580908
## 34 0.62744593 0.34847556 0.024078509
## 36 0.16441812 0.36409515 0.471486735
## 7  0.32589478 0.66552129 0.008583934
## 3  0.30290625 0.49856454 0.198529216
## 19 0.17026292 0.48400921 0.345727876
## 39 0.27762412 0.57821118 0.144164705
## 40 0.39448863 0.39139658 0.214114789
## 13 0.60224976 0.39206002 0.005690216
## 16 0.52045980 0.44523822 0.034301983
## 41 0.03786282 0.15870399 0.803433192
## 23 0.27084139 0.71883502 0.010323593
##
## $x
##           LD1         LD2
## 43  2.4581719  1.56479599
## 34 -0.8466987  1.73789943
## 36  1.2650411 -0.45185144
## 7  -1.4349293 -1.00170192
## 3   0.4923859 -0.06625547
## 19  0.9937250 -1.01905884
## 39  0.2903822 -0.58664121
## 40  0.5266282  0.94757427
## 13 -1.7055209  1.19655152
## 16 -0.6553746  0.93072052
## 41  2.2458396 -1.46032845
## 23 -1.2947870 -1.48511557
```

Evaluate LDA

```
aper(lda_preds$class, z_opt_2_test$V11)
```

```
## [1] 0.5
```

It has a very high misclassification rate, it actually misclassifies the majority of the classes!

```
acc(lda_preds$class, z_opt_2_test$V11)
```

```
## [1] 0.5
```

The accuracy is all the way down at 41,6% which is pathetic.

```
#library(caret)
# confusionMatrix(as.factor(lda_preds$class), as.factor(z_opt_2_test$V11))
confusion_matrix(lda_preds$class, z_opt_2_test$V11)
```

```
##      observed
## preds 1 2 3
##     1 2 1 1
##     2 2 2 1
##     3 0 1 2
```

We can see that class 1 is prone to misclassificaitons. ### QDA

```
library(MASS)
qda_model <- qda(V11~., data = z_opt_2_train)
qda_model
```

```
## Call:
## qda(V11 ~ ., data = z_opt_2_train)
##
## Prior probabilities of groups:
##         1         2         3
## 0.3333333 0.4666667 0.2000000
##
## Group means:
##         V3       V8      V10
## 1 2794.735 21.67887 3.262363
## 2 2700.792 22.37316 3.071542
## 3 1955.850 13.49595 2.963946
```

```
qda_preds <- predict(qda_model, z_opt_2_test)
qda_preds
```

```
## $class
##  [1] 3 1 2 2 1 2 2 1 1 2 3 2
## Levels: 1 2 3
##
## $posterior
##              1         2           3
## 43 0.007797458 0.0816206 0.910581945
## 34 0.509557855 0.4881568 0.002285376
## 36 0.109900596 0.4879259 0.402173523
## 7  0.332246251 0.6665895 0.001164262
## 3  0.494237755 0.3619359 0.143826373
## 19 0.052548849 0.5992991 0.348152101
## 39 0.167031117 0.6010966 0.231872282
## 40 0.449009995 0.4320487 0.118941261
## 13 0.578483379 0.4199623 0.001554357
## 16 0.333249146 0.6653488 0.001402068
## 41 0.001488332 0.1088926 0.889619026
## 23 0.155965712 0.8387635 0.005270748
```

Evaluate QDA

```
aper(qda_preds$class, z_opt_2_test$V11)
```

```
## [1] 0.4166667
```

It has a lower misclassification rate, it “only” misclassifes 30% of the classes

```
acc(qda_preds$class, z_opt_2_test$V11)
```

```
## [1] 0.5833333
```

The accuracy is 66% which is at least better than hald

```
#library(caret)
# confusionMatrix(as.factor(lda_preds$class), as.factor(z_opt_2_test$V11))
confusion_matrix(qda_preds$class, z_opt_2_test$V11)
```

```
##      observed
## preds 1 2 3
##     1 2 1 1
##     2 2 3 1
##     3 0 0 2
```

THis time class 3 is prone to misclassifications

Logistic Regression

```
library(nnet)
logistic_train <- z_opt_2_train
logistic_train$V11 <- as.factor(logistic_train$V11)
logistic <- multinom(V11~., data=logistic_train)
```

```
## # weights:  15 (8 variable)
## initial  value 32.958369
## iter  10 value 25.503968
## iter  20 value 24.361136
## iter  30 value 24.356581
## final  value 24.356363
## converged
```

```
logistic
```

```
## Call:
## multinom(formula = V11 ~ ., data = logistic_train)
##
## Coefficients:
##   (Intercept)            V3           V8       V10
## 2    5.228621  0.0001512352  0.002644872 -1.694624
## 3   13.925494 -0.0012877172 -0.226989242 -2.360012
##
## Residual Deviance: 48.71273
## AIC: 64.71273
```

```
logistic_preds <- predict(logistic, z_opt_2_test, type = "class")
logistic_preds
```

```
##  [1] 3 1 3 2 2 2 2 1 1 1 3 2
## Levels: 1 2 3
```

Evaluate Logistic

```
aper(logistic_preds, z_opt_2_test$V11)
```

```
## [1] 0.5
```

It is basically horrible, since it misclassifies more than 2/3 classes

```
acc(logistic_preds, z_opt_2_test$V11)
```

```
## [1] 0.5
```

The accuracy is 33% which is at least terrible

```
#library(caret)
# confusionMatrix(as.factor(lda_preds$class), as.factor(z_opt_2_test$V11))
confusion_matrix(logistic_preds, z_opt_2_test$V11)
```

```
##      observed
## preds 1 2 3
##     1 2 1 1
##     2 2 2 1
##     3 0 1 2
```

Most are classified wrongly

E(AER)

This is also known as cross-validation or hold-one-out. Let us define a function for it:

```
EAER <- function(data_frame, model_type = "lda") {

  n_misclass <- 0
  for (i in 1:nrow(data_frame)) {
    temp_data <- data_frame[-i,]
    if (model_type == "lda"){
      model <- lda(V11~., data = temp_data)
    }
    else if (model_type == "qda"){
      model <- qda(V11~., data = temp_data)
    }
    else {
      temp_data$V11 <- as.factor(temp_data$V11)
      model <- multinom(V11~., data=temp_data, trace = FALSE)
    }
    pred <- predict(model, data_frame[i,], type = "class")

    if (model_type != "lda" & model_type != "qda"){
      true_prediction = (pred != data_frame[i,4])

    }
    else{
      true_prediction = (pred$class != data_frame[i,4])
    }

    if (true_prediction){
      n_misclass <- n_misclass + 1
    }
  }
  return(n_misclass / length(data_frame[,1]))

}

# Below is how to do it with the built in function
```

Let us evaluate each model:
LDA

```
lda_eaer <- EAER(z_opt_2, "lda")
lda_acc <- 1 - lda_eaer
print(paste0("EAER: ", lda_eaer," Acc: ", lda_acc))
```

```
## [1] "EAER: 0.666666666666667 Acc: 0.333333333333333"
```

```
model <- lda(V11~., data = z_opt_2, CV = TRUE)
aper(model$class, z_opt_2$V11)
```

```
## [1] 0.6428571
```

It misclassifies around 66% and has an accuracy of 33%, so all in all pretty terrible.

QDA

```
qda_eaer <- EAER(z_opt_2, "qda")
qda_acc <- 1 - qda_eaer
print(paste0("EAER: ", qda_eaer," Acc: ", qda_acc))
```

```
## [1] "EAER: 0.619047619047619 Acc: 0.380952380952381"
```

```
model <- qda(V11~., data = z_opt_2, CV = TRUE)
aper(model$class, z_opt_2$V11)
```

```
## [1] 0.6190476
```

The QDA has a slightly lower misclassification rate, but still very close to just guessing.

Logistic

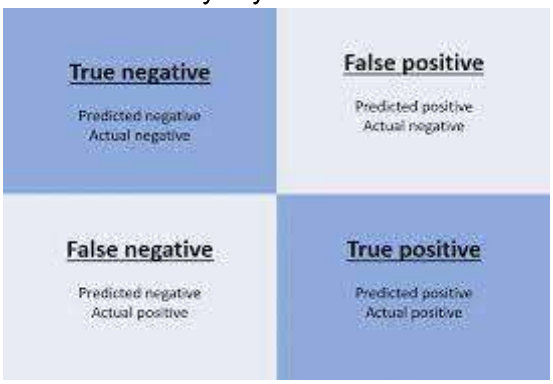
```
logi_eaer <- EAER(z_opt_2, "logistic")
logi_acc <- 1 - logi_eaer
print(paste0("EAER: ", logi_eaer," Acc: ", logi_acc))
```

```
## [1] "EAER: 0.714285714285714 Acc: 0.285714285714286"
```

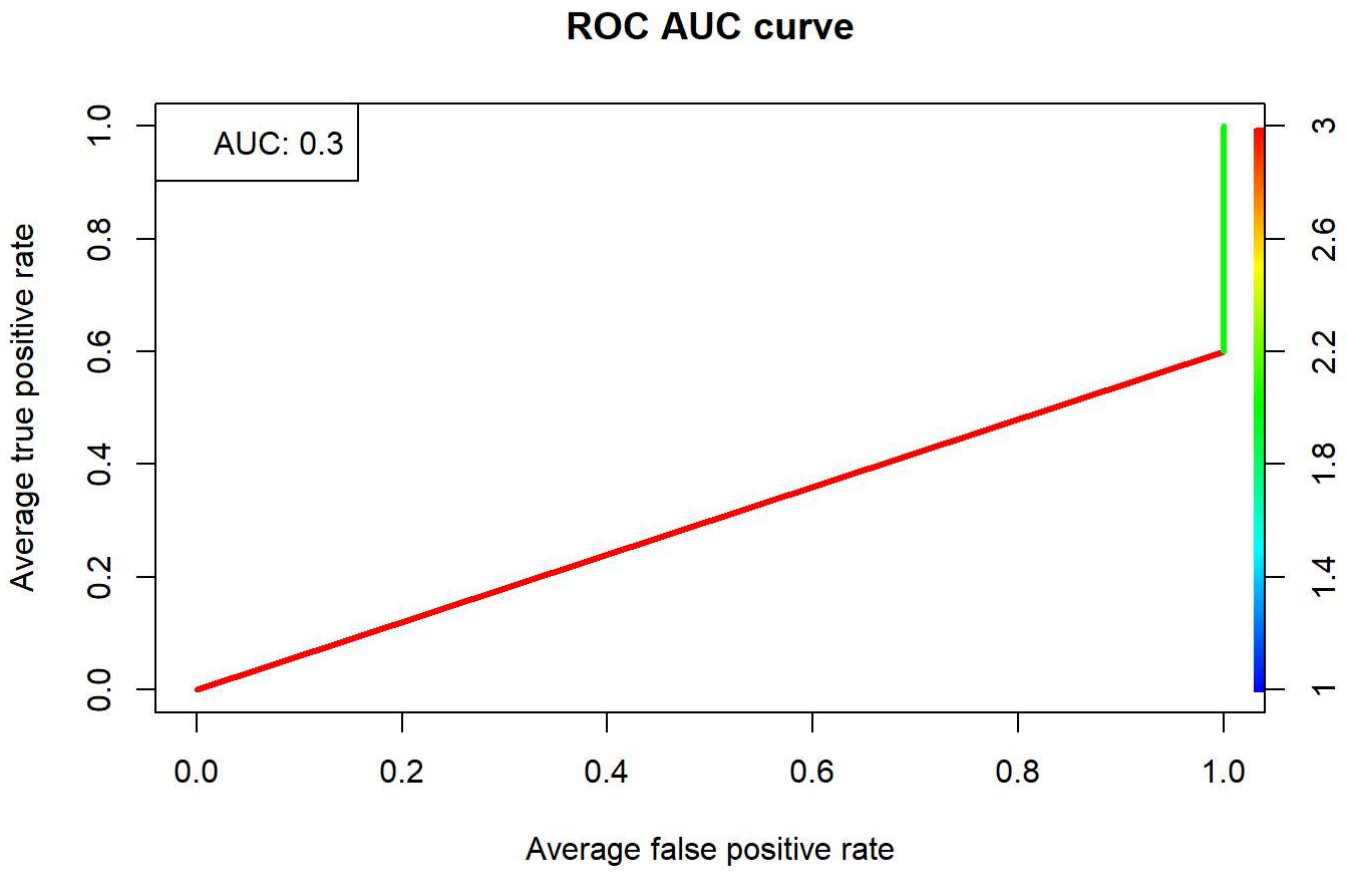
As before we can see that this model is terrible.

ROC

ROC (Receiver Operating Characteristic Curve). But it only works with two classes so we will remove class three since it anyways have so few observations. it works by looking at the true positives and false positives:



```
roc_data <- df2<-z_opt_2[!(z_opt_2$V11==3), ]
library(ROCR)
model <- lda(V11~., data = roc_data, CV = TRUE)
predictions <- as.numeric(model$class)
labels <- roc_data$V11
pred <- prediction(predictions, labels)
perf <- performance(pred, "tpr", "fpr")
auc <- performance(pred, measure = "auc")

plot(perf,
     avg= "threshold",
     colorize=TRUE,
     lwd= 3,
     main= "ROC AUC curve")
legend("topleft",
       paste0("AUC: ", auc@y.values[[1]]))
```




```
#per.auc
```

ROC is a way to summarize which threshold we should set for binary classification. It is a convenient way compared to having a bunch of different confusion matrices. It tells have many true positves and false positives we will have for each threshold. The larger the area under the curve (THE AUC) the better it can only be between 1 and 0, and mine is only 0.3 so pretty bad. O let us say that it is very importatn to classify true positives. I.e. that we need to classify group two correctly, all the time. Let us look at the confusion matrix to give us a clearer clue:

```
confusion_matrix(predictions, labels)
```

```
##      observed
## preds  1  2
##     1  0  8
##     2 16 12
```

In that case our threshold should be 1.8 (green line), because our true postive rate will be 100% and our false positives will likewise be 100%, that is we will classify all as two. If we go a bit higher to 3 (red line), we will end up classifying fever as true positives but almost have just as many false positives. Our current model almost exlusive classifying every as two.

Conclusion

In conclusion the best model for option 2 data is the QDA, but it is still at terrible model.

PCA

Lastly we will finish up with PCA on the dataframe from original but with characters and categorical data excluded.:

```
pca <- prcomp(df)
pca
```

```
## Standard deviations (1, .., p=6):
## [1] 81.5067583 64.8181842 18.1198380  4.2336884  2.2630476  0.5986941
##
## Rotation (n x k) = (6 x 6):
##               PC1          PC2         PC3         PC4          PC5
## V3  -0.0864188207 -0.009017834  0.98223490 -0.10893039 -0.125110725
## V6  -0.9523355728  0.292607516 -0.07907092  0.03224397 -0.012054665
## V7  -0.0059277867 -0.025023669 -0.01520712 -0.01173223 -0.006570497
## V8  -0.0252889931  0.026928314  0.01893617 -0.66653054  0.744318011
## V9   0.0008081525 -0.006968211  0.16502420  0.73656382  0.655711455
## V10 -0.2913990722 -0.955457820 -0.03375748 -0.01294704  0.013856660
##              PC6
## V3  -0.012105490
## V6  -0.000773935
## V7  -0.999463158
## V8   0.002118585
## V9  -0.015298061
## V10  0.026224697
```

This does not tell us much, we need the summary

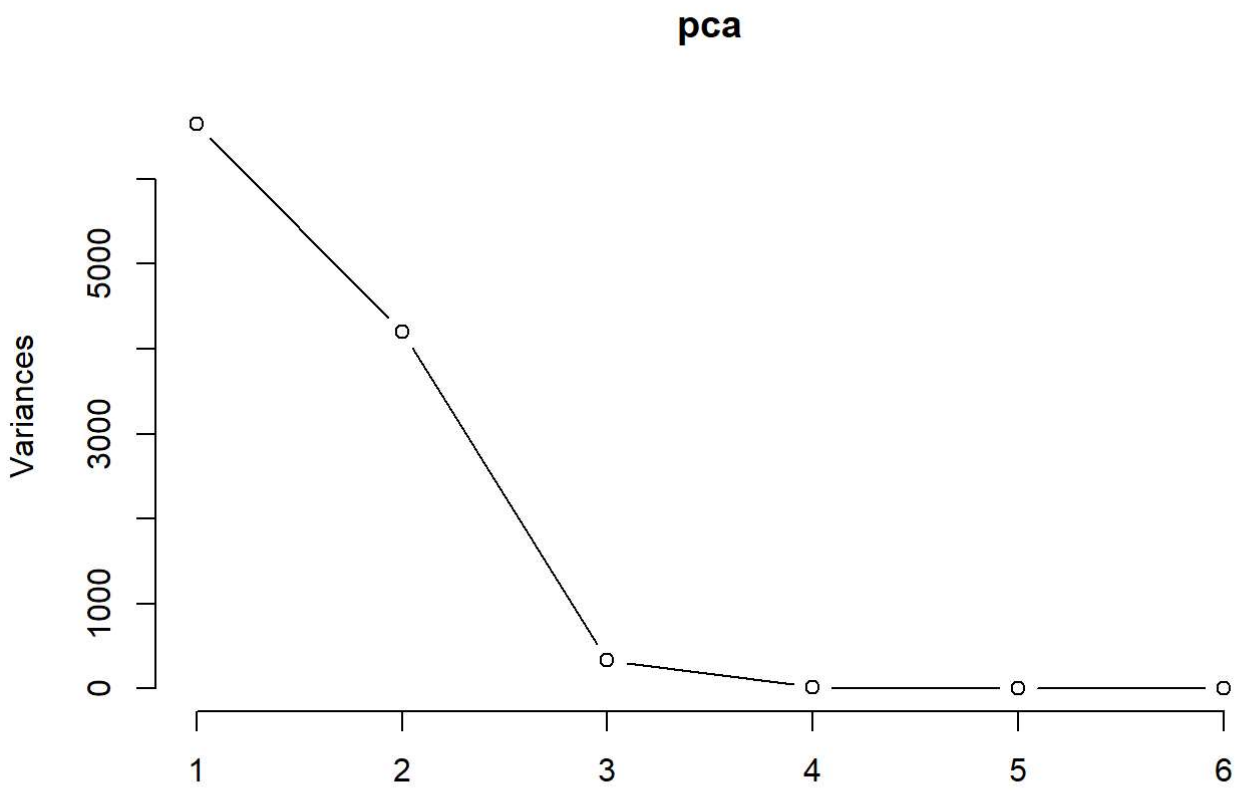
```
summary(pca)
```

```
## Importance of components:
##                            PC1     PC2      PC3    PC4     PC5     PC6
## Standard deviation     81.5068 64.8182 18.11984 4.2337 2.26305 0.59869
## Proportion of Variance  0.5933  0.3752  0.02932 0.0016 0.00046 0.00003
## Cumulative Proportion   0.5933  0.9686  0.99791 0.9995 0.99997 1.00000
```

So as we can see the first two principal components can explain 96.86% percent of the variance, and the first three can explain 99.7%. This is maybe not as usefull now, but imagine we had millions of variables, then we could reduce them to the principal componens and only work with these for the classification.

We can see how many we need with a screeplot:

```
screeplot(pca, type="l")
```



The goal is to find the elbow point, which is around three, so we need three principal compnents, not two.

We can also scale the components:

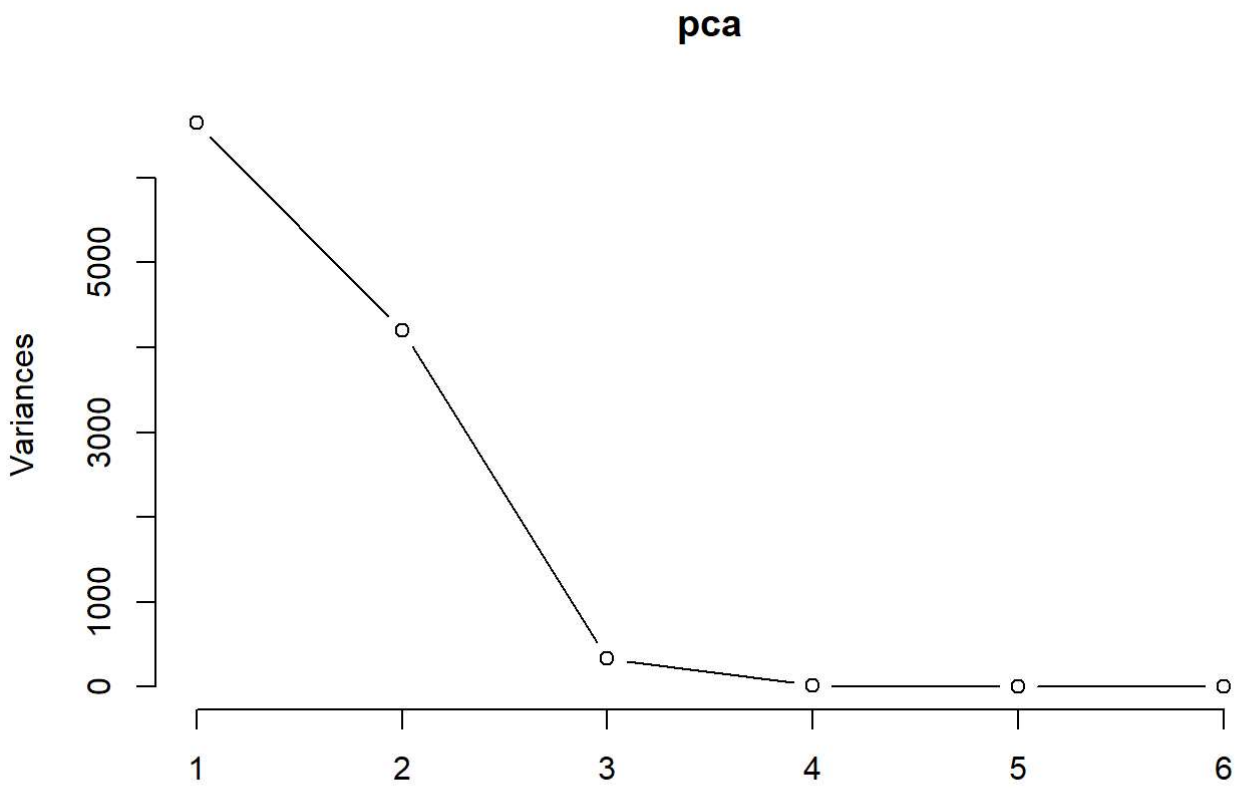
```
z_pca <- prcomp(df, scale = T)
z_pca
```

```
## Standard deviations (1, .., p=6):
## [1] 1.4326705 1.3388786 1.2255354 0.6404099 0.4346378 0.2321365
##
## Rotation (n x k) = (6 x 6):
##             PC1         PC2        PC3         PC4         PC5          PC6
## V3  -0.09114827 -0.58425519 -0.4009784  0.42078147  0.54493390 -0.124676477
## V6   0.04839871 -0.60063687  0.2899391 -0.73049746  0.13161004 -0.043364794
## V7  -0.64438451 -0.02361528  0.2734651  0.09658193 -0.16384987 -0.687933805
## V8   0.33229342 -0.48647493  0.3680098  0.46707321 -0.54508091  0.047131875
## V9  -0.18091351 -0.21026321 -0.7129601 -0.23065150 -0.60127428  0.004092771
## V10 -0.65648270 -0.12827733  0.1913773  0.09290235 -0.01533302  0.712098439
```

```
summary(z_pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     1.4327 1.3389 1.2255 0.64041 0.43464 0.23214
## Proportion of Variance 0.3421 0.2988 0.2503 0.06835 0.03149 0.00898
## Cumulative Proportion  0.3421 0.6409 0.8912 0.95953 0.99102 1.00000
```

```
screeplot(pca, type="l")
```



We likewise need three but we will end up accounting for less of the variance with the scaled version.