HUDM6122 Homework_01

Chenguang Pan

Jan 28, 2023

0.1 Exercise 1.1

First, I made a xlsx. version of `Table 1.1` to let R read it directly using the package 'readxl". This table is in 10x7 size. The first column is just the index of each observation, so I drop it here. Finally this dataset is in 9x7 size.

One should notice that the **sex**, **depression**, and **health** are categorical variables. The Pearson Correlation Coefficient is used for continuous rather than categorical variables. Therefore, when calculate the correlation matrix we should drop the categorical ones.

Note, there are some parameters need to be set. Since the original dataset contains missing value, I construct the correlation matrix based on all complete observations.

```
> library(readxl)
> table_11 <- read_excel("table_1.1.xlsx")
> my_data <- table_11[,c(2:7)]
> # drop the discrete vars and use only the complete observations
> my_data_cor <- round(cor(my_data[,c(2,3,6)], use = "complete"),2)
> # the output is rounded in two decimals.
> my_data_cor
         age    IQ weight
age     1.00 -0.15  -0.12
IQ     -0.15  1.00   0.75
weight -0.12  0.75   1.00
```

0.2 Exercise 1.2

Fill the `NA` with the column's mean, and recalculate the correlation matrix.

```
> # to impute the NA with mean using a for-loop
> for (cols in c(2,3,6)) {
+   my_data[,cols][is.na(my_data[,cols])] <- mean(my_data[,cols], na.rm=T)
+ }
> # create the correlation matrix
> my_data_cor_2 <- round(cor(my_data[,c(2,3,6)]),2)
> my_data_cor_2
         age    IQ weight
age     1.00 -0.14  -0.10
IQ     -0.14  1.00   0.52
weight -0.10  0.52   1.00
```

0.3 Exercise 1.3

The authors may not clearly say where readers can find the original dataset. After seeing the code underlying in this book's R package `MVA`, and run this argument `demo("Ch-MVA")`, one can find the `table 1.3`'s dataset information at the paragraph named `code chunk number 5`. It is in another package called `HSAUR2`. First, I draw the normal probability plots of each variable.

```
> # load the Table 1.3's original dataset
> library(HSAUR2)
> # dim(pottery)
> # names(pottery) # the dataset is the same
> # layout(matrix(1:10, nc = 3))
> sapply(colnames(pottery)[1:9], function(x){
+   # use double bracket can directly return the col
+   qqnorm(pottery[[x]], main = x)
+   qqline(pottery[[x]])
+ })
```

$Al2O3 NULL

$Fe2O3 NULL

$MgO NULL

$CaO NULL

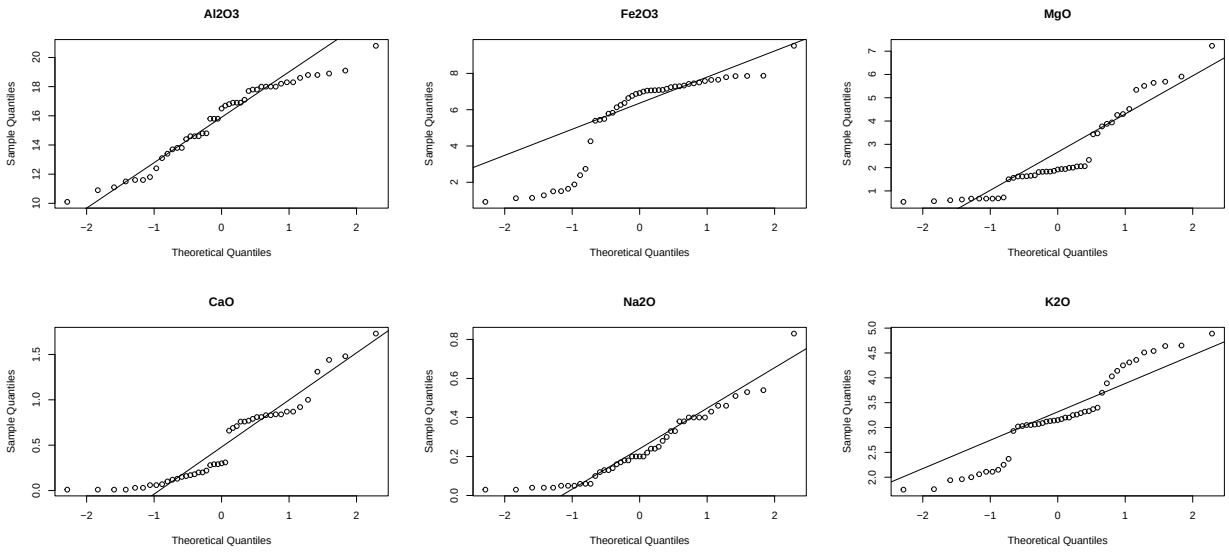
$Na2O NULL

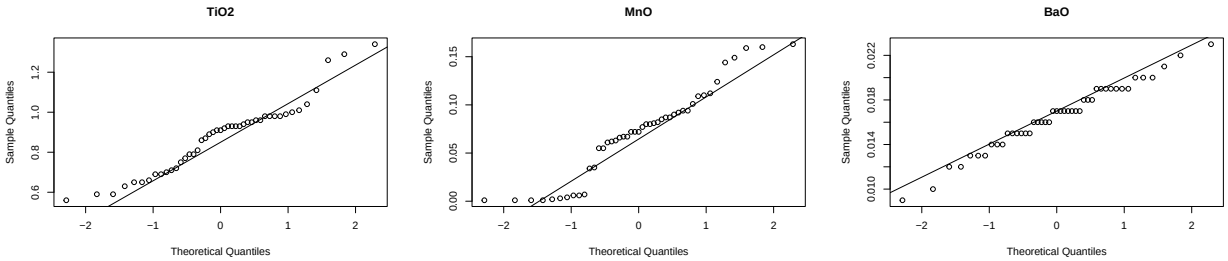
$K2O NULL

$TiO2 NULL

$MnO NULL

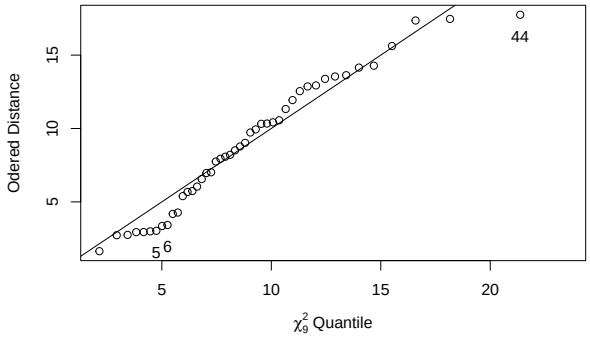
$BaO NULL





Second, I draw the chi-square plot of the data as followed.

```
> # to drop the last discrete variable
> X <- pottery[,1:9]
> # get each col's mean
> col_mean <-  colMeans(X)
> # get the cov matrix
> S <- cov(X)
> # solve() function can get inverse matrix directly
> # compute the generalised distance
> d <- apply(X, 1, # array; 1= row 2 = col
+            function(X)
+              t(X - col_mean) %*% solve(S) %*% (X - col_mean))
> plot(qc <- qchisq((1:nrow(X) - 1/2)/ nrow(X), df = 9),
+      sd<- sort(d),
+      xlab = expression(paste(chi[9]^2," Quantile")),
+      ylab = "Odered Distance", xlim = range(qc)*c(1, 1.1))
> oups <- which(rank(abs(qc-sd), ties="random") > nrow(X) - 3)
> text(qc[oups], sd[oups]-1.5, names(oups))
> abline(a=0, b=1)
```



From the Q-Q plots, one can see most of the elements do not perfectly follow the normal distribution except the `AL2O3` and the `BaO` present relatively good normal distribution. As for the Chi-square plot, if all the data are normally distributed, they should correspond to each others. Therefore, from the plot, we can easily find the outliers are the 5th, 6th, and the 44th observations.

0.4 Exercise 1.4

This question can be addressed in one line using `cov2cor` function

```
> # import the cov matrix
> a_matrix<- matrix(c(3.8778, 2.8110, 3.1480, 3.5062,
+                     2.8110, 2.1210, 2.2669, 2.5690,
+                     3.1480, 2.2669, 2.6550, 2.8341,
+                     3.5062, 2.5690, 2.8341, 3.2352),4,4, byrow=T)
> round(cov2cor(a_matrix),3)
      [,1]  [,2]  [,3]  [,4]
[1,] 1.000 0.980 0.981 0.990
[2,] 0.980 1.000 0.955 0.981
[3,] 0.981 0.955 1.000 0.967
[4,] 0.990 0.981 0.967 1.000
```

But here I provide another more specific code to achieve this goal. Could you please consider to give me extra credits? Just kidding..lol..

```
> # function to convert covariance matrix to correlation matrix
> cov2cor_test <- function(covmat) {
+     sd_vec <- sqrt(diag(covmat))
+     cormat <- covmat / outer(sd_vec, sd_vec)
+     return(cormat)
+ }
>
> round(cov2cor_test(a_matrix),3)
      [,1]  [,2]  [,3]  [,4]
[1,] 1.000 0.980 0.981 0.990
[2,] 0.980 1.000 0.955 0.981
[3,] 0.981 0.955 1.000 0.967
[4,] 0.990 0.981 0.967 1.000
```

Finally, the results are exactly the same.

0.5 Exercise 1.5

0.5.1 E1.5 Part 1 Create the Euclidean Distance Matrix

Here I write a function to get the Euclidean distance matrix for any shape of observed matrix. Since for any m * n matrix, the size of euclidean distance is always m * m.

```
> # write a function that can get Eu-dist matrix any shape of matrix
> eudis_matrix <- function(a_matrix){
+   # first define a function to get the eu distance of any two vectors
+   eu_distance <- function(vec_1, vec_2){
+     d_2_vec <- (vec_1-vec_2)^2
+     d_2 <- sum(d_2_vec)
+     d <- sqrt(d_2)
+     return(d)
+     }
+
+   m <- nrow(a_matrix)
+   dist_matrix <- matrix(nrow = m, ncol = m)
+   for (i in 1:m ) {
+     for (j in 1:m) {
```

```
+       dist_matrix[i,j] <- eu_distance(a_matrix[i,], a_matrix[j,])
+     }
+   }
+   return(dist_matrix)
+ }
```

The function seems good. Then, I tried it on the given dataset.

```
> # import the data
> multi_mat<- matrix(c(3,6,4,0,7,
+                      4,2,7,4,6,
+                      4,0,3,1,5,
+                      6,2,6,1,1,
+                      1,6,2,1,4,
+                      5,1,2,0,2,
+                      1,1,2,6,1,
+                      1,1,5,4,4,
+                      7,0,1,3,3,
+                      3,3,0,5,1),10,5, byrow=T)
> # create the euclidean dis-matrix on the given dataset.
> round(eudis_matrix(multi_mat),2)
       [,1] [,2] [,3] [,4] [,5] [,6]  [,7] [,8] [,9] [,10]
 [1,]  0.00 6.56 6.56 8.12 4.24 7.62 10.25 7.42 9.27  9.27
 [2,]  6.56 0.00 5.48 6.24 7.94 7.68  8.00 4.24 7.68  8.77
 [3,]  6.56 5.48 0.00 5.74 6.86 3.61  7.21 4.90 4.58  7.14
 [4,]  8.12 6.24 5.74 0.00 8.12 4.47  8.19 6.71 6.16  7.87
 [5,]  4.24 7.94 6.86 8.12 0.00 6.78  7.68 6.56 8.83  6.48
 [6,]  7.62 7.68 3.61 4.47 6.78 0.00  7.28 6.71 4.00  6.16
 [7,] 10.25 8.00 7.21 8.19 7.68 7.28  0.00 4.69 7.14  3.61
 [8,]  7.42 4.24 4.90 6.71 6.56 6.71  4.69 0.00 7.42  6.56
 [9,]  9.27 7.68 4.58 6.16 8.83 4.00  7.14 7.42 0.00  5.83
[10,]  9.27 8.77 7.14 7.87 6.48 6.16  3.61 6.56 5.83  0.00
```

Note each row of an empirical data n*p matrix represents an observation/vector with P dimensions. Therefore, the entry[i, j] on a n*n Euclidean distance matrix must be a scalar rather than a vector and represents the euclidean distance between ith observation/vector and jth observation/vector.

0.5.2 E1.5 Part 2 Create the City Block Distance Matrix

Here this question raise alternative concept called `city block distance`. It is the sum of the absolute differences of the blocks' coordinates.

```
> # write a function that can get city block distance
> city_matrix<-function(a_matrix){
+   # first define a function to get the eu distance of any two vectors
+   city_distance <- function(vec_1, vec_2){
+     city_dist <- sum(abs(vec_1-vec_2))
+     return(city_dist)
+     }
+
+   m <- nrow(a_matrix)
+   c_matrix <- matrix(nrow = m, ncol = m)
```

```
+   for (i in 1:m ) {
+     for (j in 1:m) {
+       c_matrix[i,j] <- city_distance(a_matrix[i,], a_matrix[j,])
+     }
+   }
+   return(c_matrix)
+ }
```

This function seems to work well. Try it on the given data:

```
> city_matrix(multi_mat)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    0   13   11   16    8   14   21   15   20    18
 [2,]   13    0   10   11   17   15   16    8   15    15
 [3,]   11   10    0   11   11    7   14   10    9    15
 [4,]   16   11   11    0   16    8   15   13   12    14
 [5,]    8   17   11   16    0   12   13   11   16    14
 [6,]   14   15    7    8   12    0   11   13    8    12
 [7,]   21   16   14   15   13   11    0    8   13     7
 [8,]   15    8   10   13   11   13    8    0   13    13
 [9,]   20   15    9   12   16    8   13   13    0    12
[10,]   18   15   15   14   14   12    7   13   12     0
```