Multivaraite Statistical Methods - Lab 01

Maximilian Pfundstein (maxpf364) and Hector and Aashana and Lakshidaa

2019-11-08

Contents

1 Describing Individual Variables

Consider the data set in the `T1-9.dat` file, National track records for women. For 55 different countries we have the national records for 7 variables (100, 200, 400, 800, 1500, 3000m and marathon). Use R to do the following analyses.

1.1 Describing the Variables

Task: Describe the 7 variables with mean values, standard deviations e.t.c.

Answer: First we import, name and look at the track times.

```
track_times = read.table("data/T1-9.dat")
colnames(track_times) = c("country", "100m", "200m", "400m",
                    "800m", "1500m", "3000m", "marathon")
head(track_times)
```

```
##   country  100m  200m  400m 800m 1500m 3000m marathon
## 1     ARG 11.57 22.94 52.50 2.05  4.25  9.19   150.32
## 2     AUS 11.12 22.23 48.63 1.98  4.02  8.63   143.51
## 3     AUT 11.15 22.70 50.62 1.94  4.05  8.78   154.35
## 4     BEL 11.14 22.48 51.45 1.97  4.08  8.82   143.05
## 5     BER 11.46 23.05 53.30 2.07  4.29  9.81   174.18
## 6     BRA 11.17 22.60 50.62 1.97  4.17  9.04   147.41
```

We see that the times for the 100m, 200m and 400m are in seconds, whereas the times for 1500m, 3000m and the marathon are measured in minutes. So first we have to adjust that.

```
# It seems like we are not supposed to correct this, so it's commented out.
#track_times[,5:8] = track_times[,5:8] * 60
head(track_times)
```

```
##   country  100m  200m  400m 800m 1500m 3000m marathon
## 1     ARG 11.57 22.94 52.50 2.05  4.25  9.19   150.32
## 2     AUS 11.12 22.23 48.63 1.98  4.02  8.63   143.51
## 3     AUT 11.15 22.70 50.62 1.94  4.05  8.78   154.35
## 4     BEL 11.14 22.48 51.45 1.97  4.08  8.82   143.05
## 5     BER 11.46 23.05 53.30 2.07  4.29  9.81   174.18
## 6     BRA 11.17 22.60 50.62 1.97  4.17  9.04   147.41
```

```
track_times_mean = apply(track_times[,2:8], 2, mean)
track_times_median = apply(track_times[,2:8], 2, median)
track_times_sd = apply(track_times[,2:8], 2, sd)

track_times_mean
```

```
##       100m       200m       400m       800m      1500m      3000m
##  11.357778  23.118519  51.989074   2.022407   4.189444   9.080741
##   marathon
## 153.619259
```

```
track_times_median
```

```
##     100m     200m     400m     800m    1500m    3000m marathon
##   11.325   22.980   51.645    2.005    4.100    8.845  148.430
```

```
track_times_sd
```

```
##        100m        200m        400m        800m       1500m       3000m
##  0.39410116  0.92902547  2.59720188  0.08687304  0.27236502  0.81532689
##    marathon
## 16.43989508
```

We see that the mean exceeds the anticipated scaled times. This means that if the time for 100m is 11.3577778, one could assume that for 200m twice the time, 22.7155556 is needed. But due to human exhaustion we see that the means increase by more than that, in this case the mean for the 200m marathon is 23.1185185. For the median the effect is a bit weaker as it is not as heavily effected by outliers as the mean. Still the effect is visible. Also the deviations increase, here the described effect is even larger.

1.2 Illustrate the Variables

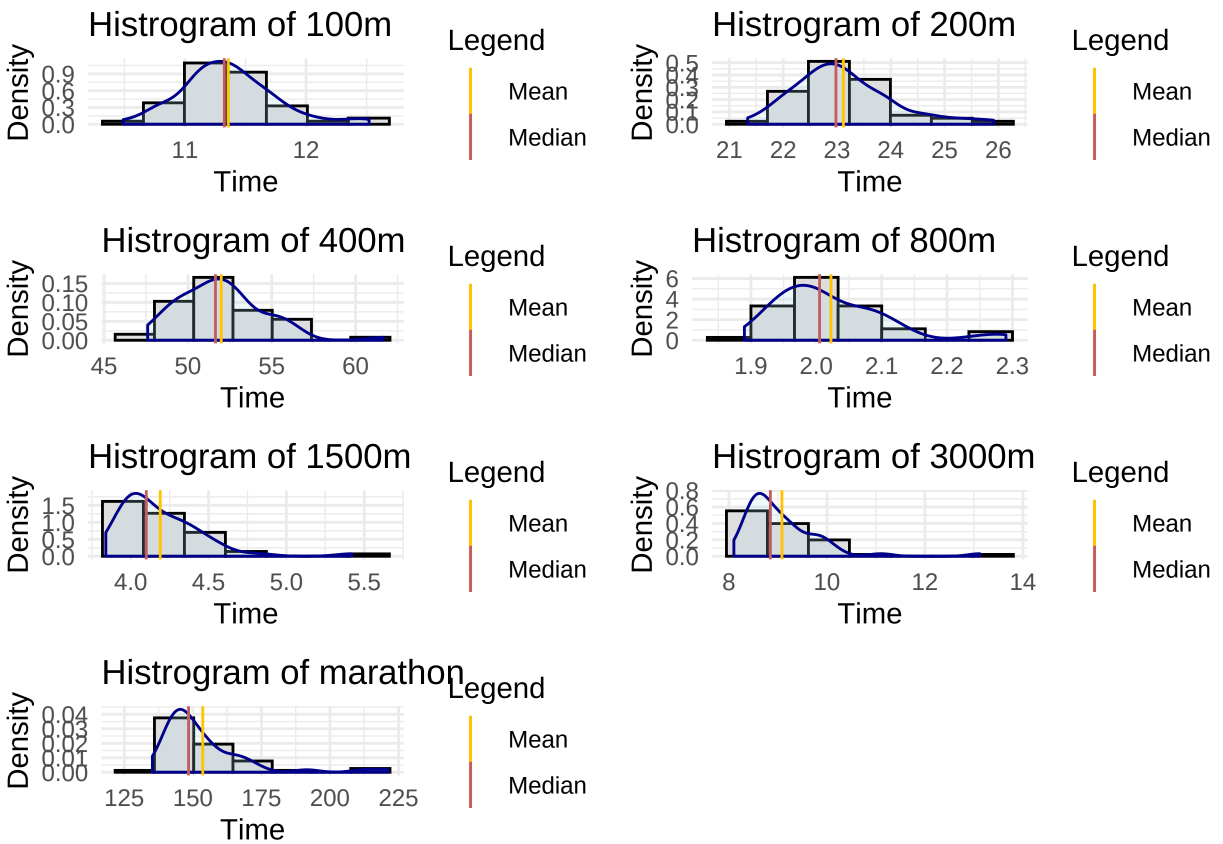
Task: Illustrate the variables with different graphs (explore what plotting possibilities R has). Make sure that the graphs look attractive (it is absolutely necessary to look at the labels, font sizes, point types). Are there any apparent extreme values? Do the variables seem normally distributed? Plot the best fitting (match the mean and standard deviation, i.e. method of moments) Gaussian density curve on the data's histogram. For the last part you may be interested in the `hist()` and `density()` functions.

Answer: From the histograms we can try to make assumptions, if the data is normally distributed. Some of them are cleary not, but we will use a statistical test for checking: For $p > 0.05$ we cannot reject the hypothesis, that the data is normal. It looks like that only the data from the 100m track is normal, after that the likelihood being from a normal decreases with track length quite fast.

```r
shapiro_wilk_res = apply(track_times[,2:8], 2, shapiro.test)
shapiro_wilk_res
```

```
## $`100m`
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.96973, p-value = 0.1875
##
##
## $`200m`
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.9538, p-value = 0.0365
##
##
## $`400m`
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.94455, p-value = 0.0145
##
##
## $`800m`
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.90487, p-value = 0.0004189
##
##
## $`1500m`
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.83583, p-value = 3.257e-06
##
##
## $`3000m`
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.76319, p-value = 5.99e-08
##
##
## $marathon
##
##  Shapiro-Wilk normality test
```

```
##
## data:  newX[, i]
## W = 0.74612, p-value = 2.621e-08
```



2 Relationships Between the Variables

2.1 Covariance and Correlation Matrices

Task: Compute the covariance and correlation matrices for the 7 variables. Is there any apparent structure in them? Save these matrices for future use.

Answer: So far we looked at the data as independent processes. If we assume them as from multivariate process, our analysis will change in the following way.

Mean: Actually the calculation stays the same:

```
as.vector(track_times_mean)
```

```
## [1]  11.357778  23.118519  51.989074   2.022407   4.189444   9.080741
## [7] 153.619259
```

Covariance:

```
cov(track_times[,2:8])
```

```
##                100m      200m      400m        800m      1500m
## 100m     0.15531572 0.3445608 0.8912960 0.027703564 0.08389119
## 200m     0.34456080 0.8630883 2.1928363 0.066165898 0.20276331
```

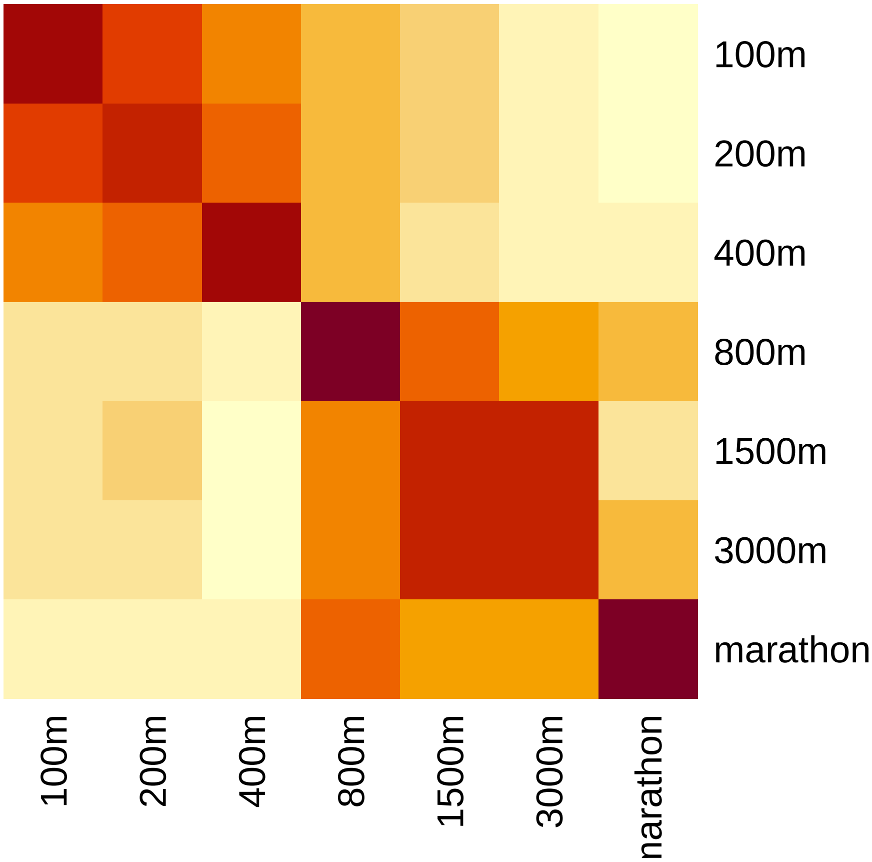
```
## 400m      0.89129602  2.1928363  6.7454576 0.181807932 0.50917683
## 800m      0.02770356  0.0661659  0.1818079 0.007546925 0.02141457
## 1500m     0.08389119  0.2027633  0.5091768 0.021414570 0.07418270
## 3000m     0.23388281  0.5543502  1.4268158 0.061379315 0.21615514
## marathon 4.33417757 10.3849876 28.9037314 1.219654647 3.53983732
##                3000m   marathon
## 100m       0.23388281   4.334178
## 200m       0.55435017  10.384988
## 400m       1.42681579  28.903731
## 800m       0.06137932   1.219655
## 1500m      0.21615514   3.539837
## 3000m      0.66475793  10.706091
## marathon 10.70609113 270.270150
```

Correlation:

```r
cor(track_times[,2:8])
```

```
##               100m      200m      400m      800m     1500m     3000m
## 100m     1.0000000 0.9410886 0.8707802 0.8091758 0.7815510 0.7278784
## 200m     0.9410886 1.0000000 0.9088096 0.8198258 0.8013282 0.7318546
## 400m     0.8707802 0.9088096 1.0000000 0.8057904 0.7197996 0.6737991
## 800m     0.8091758 0.8198258 0.8057904 1.0000000 0.9050509 0.8665732
## 1500m    0.7815510 0.8013282 0.7197996 0.9050509 1.0000000 0.9733801
## 3000m    0.7278784 0.7318546 0.6737991 0.8665732 0.9733801 1.0000000
## marathon 0.6689597 0.6799537 0.6769384 0.8539900 0.7905565 0.7987302
##           marathon
## 100m     0.6689597
## 200m     0.6799537
## 400m     0.6769384
## 800m     0.8539900
## 1500m    0.7905565
## 3000m    0.7987302
## marathon 1.0000000
```

We can also show the heatmap of the correlation to get a better overview:



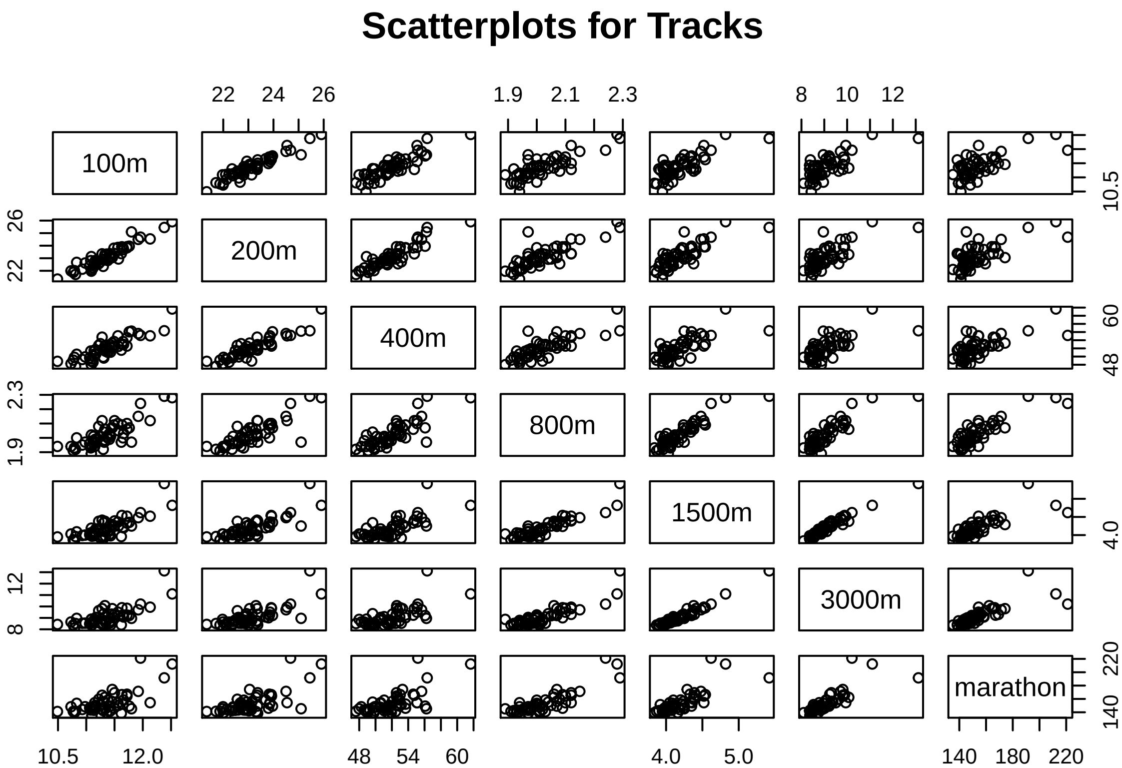
From the heatmap it looks like that we have two groups of tracks that are cmore correlated than the others. It seems that 100m, 200m and 400 belong to ond group and 800m, 1500m, 3000m and the marathon belong to the other.

2.2 Scatterplots

Task: Generate and study the scatterplots between each pair of variables. Any extreme values?

Answer: While the lower tracks seem to have a linear dependence to each other, this behaviour vanishes as the track length of one increases. So a track long track compares to a short track have more like a normal density around a shifted mean. If both tracks are long, this we see a mixture of these behaviours: Somehow they cluster up, but still remain some linear dependency (which makes sense to a degree that the track lenghts actually increase linearly). Also it seems more likely to have outliers for the longer tracks. This can also be seen in the histograms of the previous exercise.

```
pairs(track_times[,2:8], main="Scatterplots for Tracks")
```

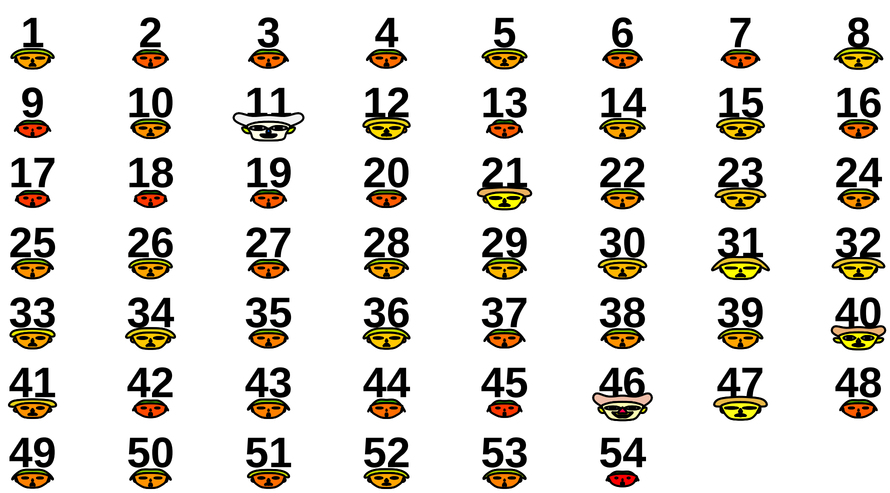


2.3 More Graphs

Task: Explore what other plotting possibilities R offers for multivariate data. Present other (at least two) graphs that you find interesting with respect to this data set.

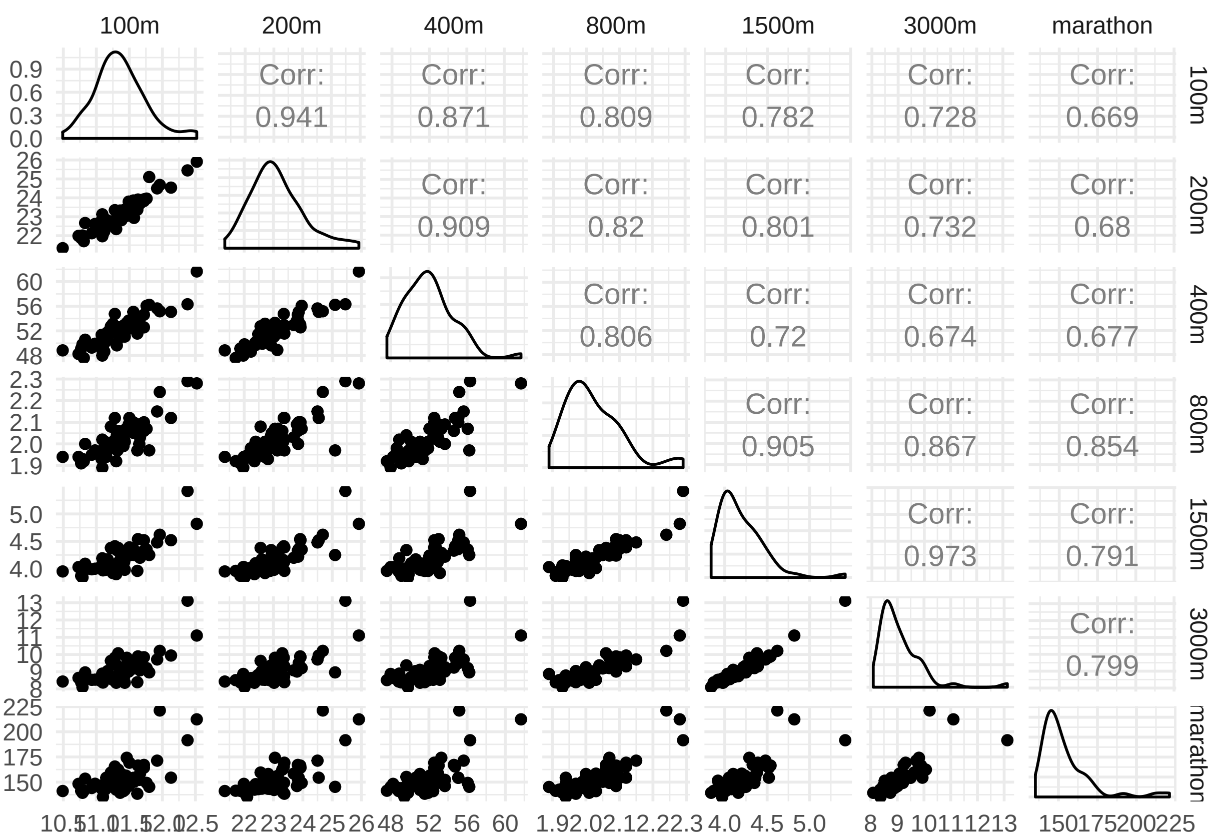
Answer: See the following plots.

```
faces(track_times[,2:8], face.type=1)
```




```
## effect of variables:
##  modified item       Var
##  "height of face   " "100m"
##  "width of face    " "200m"
##  "structure of face" "400m"
```

```
## "height of mouth  " "800m"
## "width of mouth   " "1500m"
## "smiling          " "3000m"
## "height of eyes   " "marathon"
## "width of eyes    " "100m"
## "height of hair   " "200m"
## "width of hair   "  "400m"
## "style of hair   "  "800m"
## "height of nose  "  "1500m"
## "width of nose   "  "3000m"
## "width of ear    "  "marathon"
## "height of ear   "  "100m"
```



3 Examining for Extreme Values

3.1 Most Extreme Countries

Task: Look at the plots (esp. scatterplots) generated in the previous question. Which 3–4 countries appear most extreme? Why do you consider them extreme?

Answer: Looking at the scatterplots we see that the outliers are usually those that are extremely slow. It does not really happen that we have one outlier that is way faster than the others (which makes sense in a way that in a fierce competition minimal times matter a lot). We therefore focus on outliers that are slower than 95% of the other nations. In this analysis USA was found two times and GER one time.

```
upper = function(x) {
  return(mean(x) + 1.645 * sd(x))
```

```r
}

lower = function(x) {
  return(mean(x) - 1.645 * sd(x))
}

uppers = apply(track_times[,2:8], 2, upper)
lowers = apply(track_times[,2:8], 2, lower)

for (i in 1:7) {
  name = colnames(track_times[,2:8])[i]
  #outliers_fast = track_times$country[track_times[,i] > uppers[i]]
  outliers_slow = track_times$country[track_times[,i+1] < lowers[i]]

  #print("##########")
  print(name)
  #print("Fast outliers:")
  #print(as.vector(outliers_fast))
  print("Slow outliers:")
  print(as.vector(outliers_slow))
  print("##########")
}
```

```
## [1] "100m"
## [1] "Slow outliers:"
## [1] "USA"
## [1] "##########"
## [1] "200m"
## [1] "Slow outliers:"
## [1] "USA"
## [1] "##########"
## [1] "400m"
## [1] "Slow outliers:"
## [1] "GER"
## [1] "##########"
## [1] "800m"
## [1] "Slow outliers:"
## character(0)
## [1] "##########"
## [1] "1500m"
## [1] "Slow outliers:"
## character(0)
## [1] "##########"
## [1] "3000m"
## [1] "Slow outliers:"
## character(0)
## [1] "##########"
## [1] "marathon"
## [1] "Slow outliers:"
## character(0)
## [1] "##########"
```

3.2 Multivariate Residual

One approach to measuring "extremism" is to look at the distance (needs to be defined!) between an observation and the sample mean vector, i.e. we look how far one is from the average. Such a distance can be called an *multivariate residual* for the given observation.

Task: The most common residual is the Euclidean distance between the observation and sample mean vector, i.e.

$$d(\vec{x}, \bar{x}) = \sqrt{(\vec{x} - \bar{x})^T (\vec{x} - \bar{x})}.$$

This distance can be immediately generalized to the $L^r, r > 0$ distance as

$$d_{L^r}(\vec{x} - \bar{x}) = \left(\sum_{i=1}^{p} |\vec{x}_i - \bar{x}_i|^r \right)^{1/r},$$

where p is the dimension of the observation (here $p = 7$).

Compute the squared Euclidean distance (i.e. r = 2) of the observation from the sample mean for all 55 countries using R's matrix operations. First center the raw data by the means to get $\vec{x} - \bar{x}$ for each country. Then do a calculation with matrices that will result in a matrix that has on its diagonal the requested squared distance for each country. Copy this diagonal to a vector and report on the five most extreme countries. In this questions you **MAY NOT** use any loops.

Answer: The following function implement the functionality. The function `euclidean` calculates the euclean residual. The function `distance` also handles arbitrary values for r, but utilised more built-in R functions. `get_outliers` orders the result and selects the n top countries. This functions are resued in the following exercises.

```
euclidean = function(X) {

  # Preprocessing
  X = as.matrix(X)

  # X - X_bar
  #ident = matrix(-1, nrow=nrow(X), ncol=nrow(X))
  #diag(ident) = nrow(X)
  #X_bar = 1/N *(t(ident) %*% X)

  # X - X_bar
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
  mu = 1/nrow(X) * (t(ident) %*% X)
  X_centered = X - mu

  res = sqrt(diag(X_centered %*% t(X_centered)))

  return(res)
}

distance = function(X, r = 2) {

  # Preprocessing
  X = as.matrix(X)

  # X - X_bar
```

```r
  #ident = matrix(-1, nrow=nrow(X), ncol=nrow(X))
  #diag(ident) = nrow(X)
  #X_bar = 1/N *(t(ident) %*% X)

  # X - X_bar
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
  mu = 1/nrow(X) * (t(ident) %*% X)
  X_centered = X - mu

  res = rowSums(X_centered^r)^(1/r)

  return(res)
}

get_outliers = function(data, n=5) {
  data = as.matrix(data)
  return(track_times$country[order(data, decreasing = TRUE)[1:n]])
}
```

```r
euclidean(track_times[,2:8])
```

```
##  [1]  3.352526 10.703279  1.656308 10.608378 20.616060  6.382646  5.718796
##  [8]  2.329972 14.486679  2.851967 59.615175 10.794572  9.408675  4.402629
## [15] 12.939488  5.978177  6.670036 13.039467 18.591462  1.610905 18.158217
## [22]  5.173356  3.287141  4.588885 11.451565  2.755565 10.201641 14.215076
## [29] 15.177242  7.717411  9.548406  6.069117 15.677070 13.750148 10.054448
## [36]  4.940436 10.231711  7.177852 12.582623 67.627962 12.204487  9.896828
## [43] 10.351343 11.370744 12.740366 38.524758  3.695017  7.546571  3.275672
## [50]  8.157574  6.014398  8.851508  2.587504 13.023696
```

```r
distance(track_times[,2:8])
```

```
##  [1]  3.352526 10.703279  1.656308 10.608378 20.616060  6.382646  5.718796
##  [8]  2.329972 14.486679  2.851967 59.615175 10.794572  9.408675  4.402629
## [15] 12.939488  5.978177  6.670036 13.039467 18.591462  1.610905 18.158217
## [22]  5.173356  3.287141  4.588885 11.451565  2.755565 10.201641 14.215076
## [29] 15.177242  7.717411  9.548406  6.069117 15.677070 13.750148 10.054448
## [36]  4.940436 10.231711  7.177852 12.582623 67.627962 12.204487  9.896828
## [43] 10.351343 11.370744 12.740366 38.524758  3.695017  7.546571  3.275672
## [50]  8.157574  6.014398  8.851508  2.587504 13.023696
```

```r
get_outliers(euclidean(track_times[,2:8]))
```

```
## [1] PNG COK SAM BER GBR
## 54 Levels: ARG AUS AUT BEL BER BRA CAN CHI CHN COK COL CRC CZE DEN ... USA
```

3.3 Squared Distance

Task: The different variables have different scales so it is possible that the distances can be dominated by some few variables. To avoid this we can use the squared distance

$$d^2_{\boldsymbol{V}}(\vec{x} - \bar{x}) = (\vec{x} - \bar{x})\boldsymbol{V}^{-1}(\vec{x} - \bar{x}),$$

where $\mathbf{V}$ is a diagonal matrix with variances of the appropriate variables on the diagonal. The effect, is that for each variable the squared distance is divided by its variance and we have a scaled independent distance. It is simple to compute this measure by standardizing the raw data with both means (centring) and standard deviations (scaling), and then compute the Euclidean distance for the normalized data. Carry out these computations and conclude which countries are the most extreme ones. How do your conclusions compare with the unnormalized ones?

Answer: In the following, both approaches are implemented and it can be seen, that they yield the same result. `sample_variance` is a helper function that calcualtes teh sample variance. The function `normalize` normalises the data (actually standardises, not normalises).

The function `squared_distance` directly calculates the result using vectorisation and matrix multiplication.

```
sample_variance = function(X) {

  X = as.matrix(X)

  identity = diag(nrow(X))
  one_n = matrix(1, nrow=nrow(X), ncol=1)

  inter =  identity - 1/nrow(X) * (one_n %*% t(one_n))

  return(1/nrow(X) * (t(X) %*% inter %*% X))
}

normalize = function(X) {
  X = as.matrix(X)

  sigma = sqrt(diag(sample_variance(X)))
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
  mu = 1/nrow(X) * (t(ident) %*% X)

  X_centered = X - mu
  X_norm = X_centered / matrix(sigma, nrow=nrow(X), ncol=length(sigma), byrow = TRUE)

  return(X_norm)
}

X_norm = normalize(track_times[,2:8])
X_euc = euclidean(X_norm)

get_outliers(X_euc)
```

```
## [1] SAM COK PNG USA SIN
## 54 Levels: ARG AUS AUT BEL BER BRA CAN CHI CHN COK COL CRC CZE DEN ... USA
```

```
squared_distance = function(X) {
  X = as.matrix(X)

  V = matrix(0, nrow=ncol(X), ncol=ncol(X))
  diag(V) = diag(sample_variance(X))
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
  mu = 1/nrow(X) * (t(ident) %*% X)

  X_centered = X - mu

  return(diag(X_centered %*% solve(V) %*% t(X_centered)))
```

```
}

get_outliers(squared_distance(track_times[,2:8]))
```

```
## [1] SAM COK PNG USA SIN
## 54 Levels: ARG AUS AUT BEL BER BRA CAN CHI CHN COK COL CRC CZE DEN ... USA
```

3.4 Mahalanobis Distance

Task: The most common statistical distance is the *Mahalanobis distance*

$$d_M^2(\vec{x} - \bar{x}) = (\vec{x} - \bar{x})^T \boldsymbol{C}^{-1}(\vec{x} - \bar{x}),$$

where **C** is the sample covariance matrix calculated from the data. With this measure we also use the relationships (covariances) between the variables (and not only the marginal variances as $d_V(\cdot, \cdot)$ does). Compute the Mahalanobis distance, which countries are most extreme now?

Answer: The function `mahalanobis_distance` is basically the same, just that this time the whole sample variance is taken into account.

```
mahalanobis_distance = function(X) {
  X = as.matrix(X)

  V = sample_variance(X)
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
  mu = 1/nrow(X) * (t(ident) %*% X)

  X_centered = X - mu

  return(diag(X_centered %*% solve(V) %*% t(X_centered)))
}

get_outliers(mahalanobis_distance(track_times[,2:8]))
```

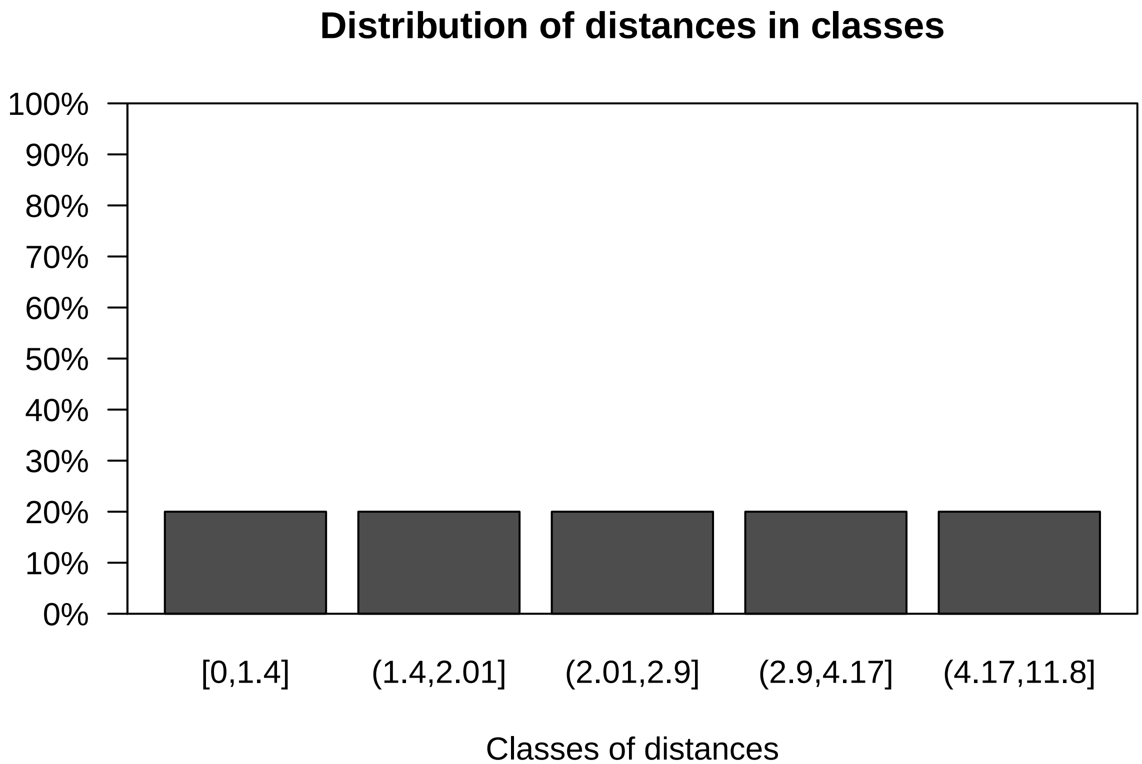
```
## [1] SAM  PNG  KORN COK  MEX
## 54 Levels: ARG AUS AUT BEL BER BRA CAN CHI CHN COK COL CRC CZE DEN ... USA
```

3.5 Czekanowski's Diagram

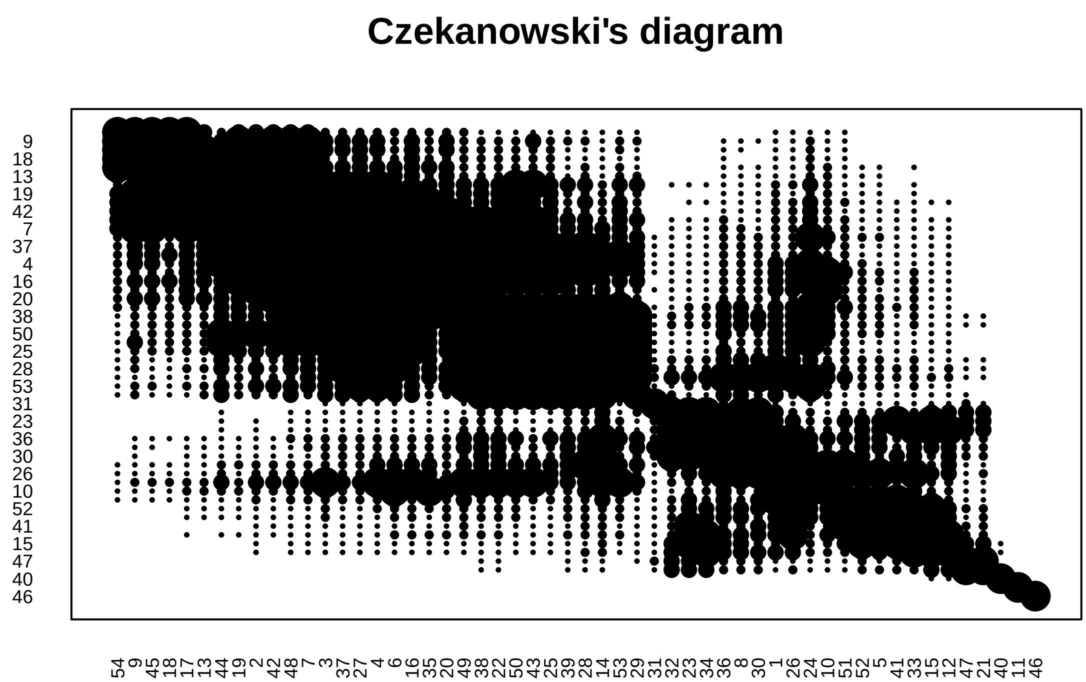
Task: Compare the results in b)–d). Some of the countries are in the upper end with all the measures and perhaps they can be classified as extreme. Discuss this. But also notice the different measures give rather different results (how does Sweden behave?). Summarize this graphically. Produce Czekanowski's diagram using e.g. the `RMaCzek` package. In case of problems please describe them.

Answer:

```
# ,n_classes = nrow(track_times)
czek_mat = czek_matrix(track_times[,2:8],
                       scale_data = TRUE,
                       monitor = TRUE)
```




```r
plot(czek_mat)
```



4 Source Code

```r
library(gridExtra)
library(ggplot2)
```

```r
library(GGally)
library(aplpack)
library(lattice)
library(RMaCzek)
knitr::opts_chunk$set(echo = TRUE)

track_times = read.table("data/T1-9.dat")
colnames(track_times) = c("country", "100m", "200m", "400m",
                          "800m", "1500m", "3000m", "marathon")
head(track_times)


# It seems like we are not supposed to correct this, so it's commented out.
#track_times[,5:8] = track_times[,5:8] * 60
head(track_times)


track_times_mean = apply(track_times[,2:8], 2, mean)
track_times_median = apply(track_times[,2:8], 2, median)
track_times_sd = apply(track_times[,2:8], 2, sd)

track_times_mean
track_times_median
track_times_sd


shapiro_wilk_res = apply(track_times[,2:8], 2, shapiro.test)
shapiro_wilk_res


p1 = ggplot() +
  geom_histogram(aes(x = track_times$`100m`, y=..density..),
                 color = "black", fill = "#dedede", bins = sqrt(nrow(track_times))) +
  geom_density(aes(x = track_times$`100m`, y=..density..),
               color="darkblue", fill="lightblue", alpha = 0.2) +
  geom_vline(aes(xintercept = track_times_mean[1], color="Mean")) +
  geom_vline(aes(xintercept = track_times_median[1], color ="Median")) +
  labs(title = "Histrogram of 100m",
       y = "Density",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("#FFC300", "#C25E5E")) +
  theme_minimal()

p2 = ggplot() +
  geom_histogram(aes(x = track_times$`200m`, y=..density..),
                 color = "black", fill = "#dedede", bins = sqrt(nrow(track_times))) +
  geom_density(aes(x = track_times$`200m`, y=..density..),
               color="darkblue", fill="lightblue", alpha = 0.2) +
  geom_vline(aes(xintercept = track_times_mean[2], color = "Mean")) +
  geom_vline(aes(xintercept = track_times_median[2], color = "Median")) +
  labs(title = "Histrogram of 200m",
       y = "Density",
       x = "Time", color = "Legend") +
```

```r
  scale_color_manual(values = c("#FFC300", "#C25E5E")) +
  theme_minimal()

p3 = ggplot() +
  geom_histogram(aes(x = track_times$`400m`, y=..density..),
                 color = "black", fill = "#dedede", bins = sqrt(nrow(track_times))) +
  geom_density(aes(x = track_times$`400m`, y=..density..),
               color="darkblue", fill="lightblue", alpha = 0.2) +
  geom_vline(aes(xintercept = track_times_mean[3], color = "Mean")) +
  geom_vline(aes(xintercept = track_times_median[3], color = "Median")) +
  labs(title = "Histrogram of 400m",
       y = "Density",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("#FFC300", "#C25E5E")) +
  theme_minimal()

p4 = ggplot() +
  geom_histogram(aes(x = track_times$`800m`, y=..density..),
                 color = "black", fill = "#dedede", bins = sqrt(nrow(track_times))) +
  geom_density(aes(x = track_times$`800m`, y=..density..),
               color="darkblue", fill="lightblue", alpha = 0.2) +
  geom_vline(aes(xintercept = track_times_mean[4],  color = "Mean")) +
  geom_vline(aes(xintercept = track_times_median[4],  color = "Median")) +
  labs(title = "Histrogram of 800m",
       y = "Density",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("#FFC300", "#C25E5E")) +
  theme_minimal()

p5 = ggplot() +
  geom_histogram(aes(x = track_times$`1500m`, y=..density..),
                 color = "black", fill = "#dedede", bins = sqrt(nrow(track_times))) +
  geom_density(aes(x = track_times$`1500m`, y=..density..),
               color="darkblue", fill="lightblue", alpha = 0.2) +
  geom_vline(aes(xintercept = track_times_mean[5], color = "Mean")) +
  geom_vline(aes(xintercept = track_times_median[5], color = "Median")) +
  labs(title = "Histrogram of 1500m",
       y = "Density",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("#FFC300", "#C25E5E")) +
  theme_minimal()

p6 = ggplot() +
  geom_histogram(aes(x = track_times$`3000m`, y=..density..),
                 color = "black", fill = "#dedede", bins = sqrt(nrow(track_times))) +
  geom_density(aes(x = track_times$`3000m`, y=..density..),
               color="darkblue", fill="lightblue", alpha = 0.2) +
  geom_vline(aes(xintercept = track_times_mean[6], color = "Mean")) +
  geom_vline(aes(xintercept = track_times_median[6], color = "Median")) +
  labs(title = "Histrogram of 3000m",
       y = "Density",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("#FFC300", "#C25E5E")) +
```

```
  theme_minimal()

p7 = ggplot() +
  geom_histogram(aes(x = track_times$`marathon`, y=..density..),
                 color = "black", fill = "#dedede", bins = sqrt(nrow(track_times))) +
  geom_density(aes(x = track_times$`marathon`, y=..density..),
               color="darkblue", fill="lightblue", alpha = 0.2) +
  geom_vline(aes(xintercept = track_times_mean[7], color = "Mean")) +
  geom_vline(aes(xintercept = track_times_median[7], color = "Median")) +
  labs(title = "Histrogram of marathon",
       y = "Density",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("#FFC300", "#C25E5E")) +
  theme_minimal()

#p1
#p2
#p3
#p4
#p5
#p6
#p7

grid.arrange(p1, p2, p3, p4, p5, p6, p7, ncol = 2)

as.vector(track_times_mean)
cov(track_times[,2:8])
cor(track_times[,2:8])
heatmap(cor(track_times[,2:8]), Rowv=NA, Colv=NA, revC=TRUE)
pairs(track_times[,2:8], main="Scatterplots for Tracks")

faces(track_times[,2:8], face.type=1)


p = ggpairs(track_times[,2:8], aes(label=track_times$country)) + theme_minimal()
# Change color manually.
# Loop through each plot changing relevant scales
for(i in 1:p$nrow) {
  for(j in 1:p$ncol){
    p[i,j] <- p[i,j] +
        scale_fill_manual(values=c("#00AFBB", "#E7B800", "#FC4E07")) +
        scale_color_manual(values=c("#00AFBB", "#E7B800", "#FC4E07"))
  }
}
p


upper = function(x) {
  return(mean(x) + 1.645 * sd(x))
}

lower = function(x) {
  return(mean(x) - 1.645 * sd(x))
```

```
}

uppers = apply(track_times[,2:8], 2, upper)
lowers = apply(track_times[,2:8], 2, lower)

for (i in 1:7) {
  name = colnames(track_times[,2:8])[i]
  #outliers_fast = track_times$country[track_times[,i] > uppers[i]]
  outliers_slow = track_times$country[track_times[,i+1] < lowers[i]]

  #print("##########")
  print(name)
  #print("Fast outliers:")
  #print(as.vector(outliers_fast))
  print("Slow outliers:")
  print(as.vector(outliers_slow))
  print("##########")
}


euclidean = function(X) {

  # Preprocessing
  X = as.matrix(X)

  # X - X_bar
  #ident = matrix(-1, nrow=nrow(X), ncol=nrow(X))
  #diag(ident) = nrow(X)
  #X_bar = 1/N *(t(ident) %*% X)

  # X - X_bar
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
  mu = 1/nrow(X) * (t(ident) %*% X)
  X_centered = X - mu

  res = sqrt(diag(X_centered %*% t(X_centered)))

  return(res)
}

distance = function(X, r = 2) {

  # Preprocessing
  X = as.matrix(X)

  # X - X_bar
  #ident = matrix(-1, nrow=nrow(X), ncol=nrow(X))
  #diag(ident) = nrow(X)
  #X_bar = 1/N *(t(ident) %*% X)

  # X - X_bar
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
  mu = 1/nrow(X) * (t(ident) %*% X)
```

```r
  X_centered = X - mu

  res = rowSums(X_centered^r)^(1/r)

  return(res)
}

get_outliers = function(data, n=5) {
  data = as.matrix(data)
  return(track_times$country[order(data, decreasing = TRUE)[1:n]])
}

euclidean(track_times[,2:8])
distance(track_times[,2:8])
get_outliers(euclidean(track_times[,2:8]))

sample_variance = function(X) {

  X = as.matrix(X)

  identity = diag(nrow(X))
  one_n = matrix(1, nrow=nrow(X), ncol=1)

  inter =  identity - 1/nrow(X) * (one_n %*% t(one_n))

  return(1/nrow(X) * (t(X) %*% inter %*% X))
}

normalize = function(X) {
  X = as.matrix(X)

  sigma = sqrt(diag(sample_variance(X)))
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
  mu = 1/nrow(X) * (t(ident) %*% X)

  X_centered = X - mu
  X_norm = X_centered / matrix(sigma, nrow=nrow(X), ncol=length(sigma), byrow = TRUE)

  return(X_norm)
}

X_norm = normalize(track_times[,2:8])
X_euc = euclidean(X_norm)

get_outliers(X_euc)


squared_distance = function(X) {
  X = as.matrix(X)

  V = matrix(0, nrow=ncol(X), ncol=ncol(X))
  diag(V) = diag(sample_variance(X))
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
```

```
  mu = 1/nrow(X) * (t(ident) %*% X)

  X_centered = X - mu

  return(diag(X_centered %*% solve(V) %*% t(X_centered)))
}

get_outliers(squared_distance(track_times[,2:8]))


mahalanobis_distance = function(X) {
  X = as.matrix(X)

  V = sample_variance(X)
  ident = matrix(1, nrow=nrow(X), ncol=nrow(X))
  mu = 1/nrow(X) * (t(ident) %*% X)

  X_centered = X - mu

  return(diag(X_centered %*% solve(V) %*% t(X_centered)))
}

get_outliers(mahalanobis_distance(track_times[,2:8]))


# ,n_classes = nrow(track_times)
czek_mat = czek_matrix(track_times[,2:8],
                       scale_data = TRUE,
                       monitor = TRUE)
plot(czek_mat)
```