modernactuarialmodels

```r
### load packages and data

sub_wkd<-"5 - Unsupervised Learning What is a Sports Car"
source(paste(sub_wkd,"./00_a functions and tools.R",sep=""))
```

```
## Loading required package: MASS

##
## Attaching package: 'matrixStats'

## The following object is masked from 'package:plyr':
##
##     count
```

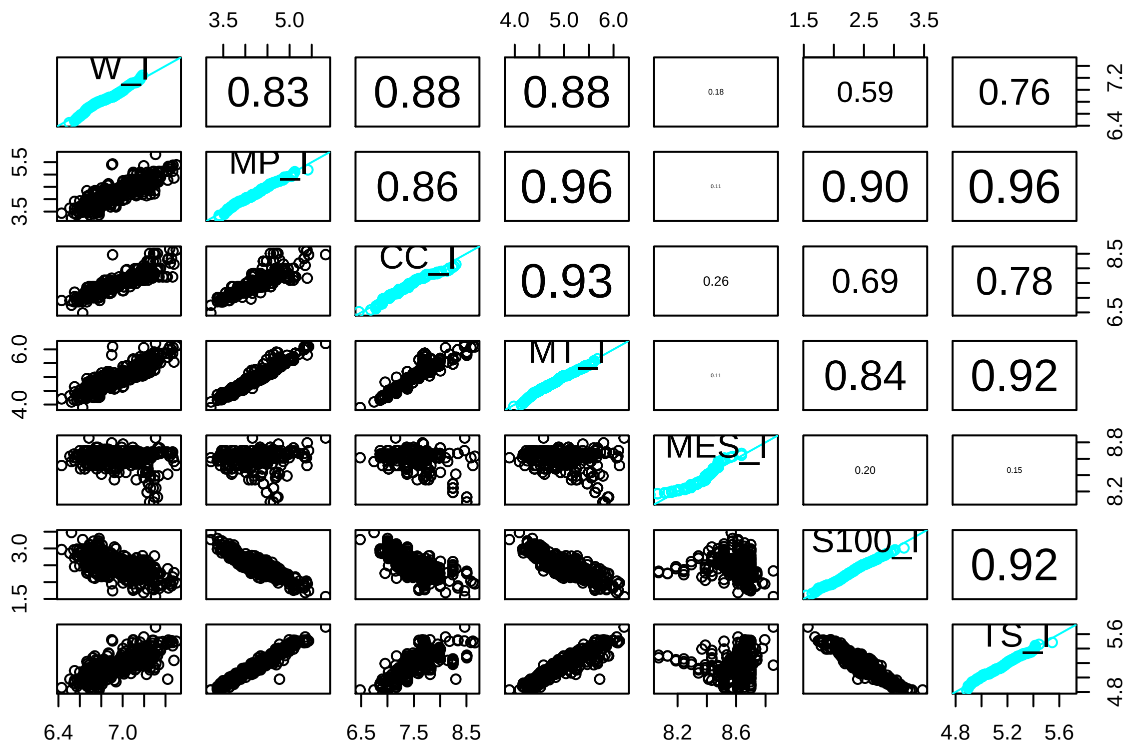
```r
d.data.org<-read.table("5 - Unsupervised Learning What is a Sports Car/SportsCars.csv",sep=";",header=T
d.data<-d.data.org
str(d.data)
```

```
## 'data.frame':    475 obs. of  13 variables:
##  $ brand           : chr  "Austin" "Citroen" "Citroen" "Fiat" ...
##  $ type            : chr  "Rovermini" "Visa" "2CV" "Panda" ...
##  $ model           : chr  "Ehlemayair" "Baseclub" "Specialton" "34" ...
##  $ cubic_capacity  : int  998 652 602 850 1598 845 956 1588 1596 992 ...
##  $ max_power       : int  31 25 21 25 41 21 31 40 40 37 ...
##  $ max_torque      : num  67 49 39 60 96 56 65 100 100 98 ...
##  $ seats           : int  4 5 4 5 5 4 5 5 5 5 ...
##  $ weight          : int  620 755 585 680 1015 695 695 900 1030 920 ...
##  $ max_engine_speed: int  5000 5500 5750 5250 4600 4500 5750 4500 4800 4250 ...
##  $ seconds_to_100  : num  19.5 26.2 NA 32.3 21 NA 19.3 18.7 20 NA ...
##  $ top_speed       : int  129 125 115 125 143 115 137 148 140 130 ...
##  $ sports_car      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ tau             : num  23.3 34.1 28.6 32.8 35 ...
```

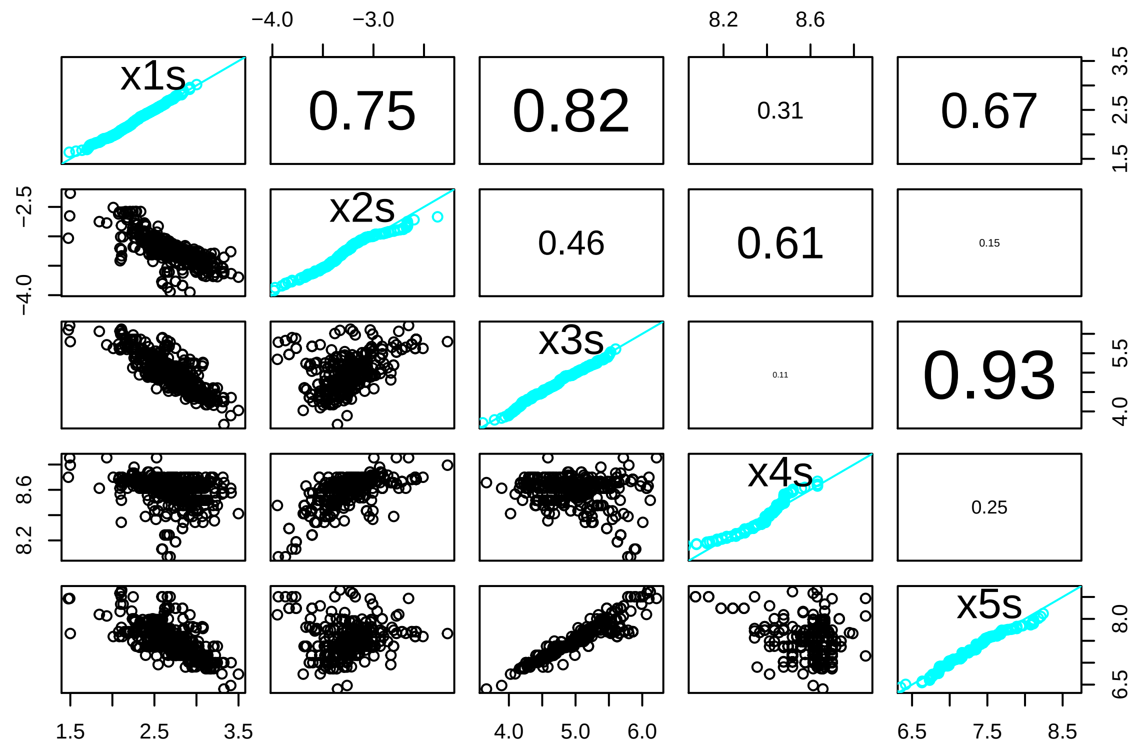
```r
###  log

# log
d.data$W_l    <- log(d.data$weight)
d.data$MP_l   <- log(d.data$max_power)
d.data$CC_l   <- log(d.data$cubic_capacity)
d.data$MT_l   <- log(d.data$max_torque)
d.data$MES_l  <- log(d.data$max_engine_speed)
d.data$S100_l <- log(d.data$seconds_to_100)
d.data$TS_l   <- log(d.data$top_speed)

#  Ingenbleek-Lemaire (ASTIN Bulletin 1988)
d.data$x1s  <- d.data$W_l-d.data$MP_l
d.data$x2s  <- d.data$MP_l-d.data$CC_l
d.data$x3s  <- d.data$MT_l
d.data$x4s  <- d.data$MES_l
```

```
d.data$x5s  <- d.data$CC_l
```

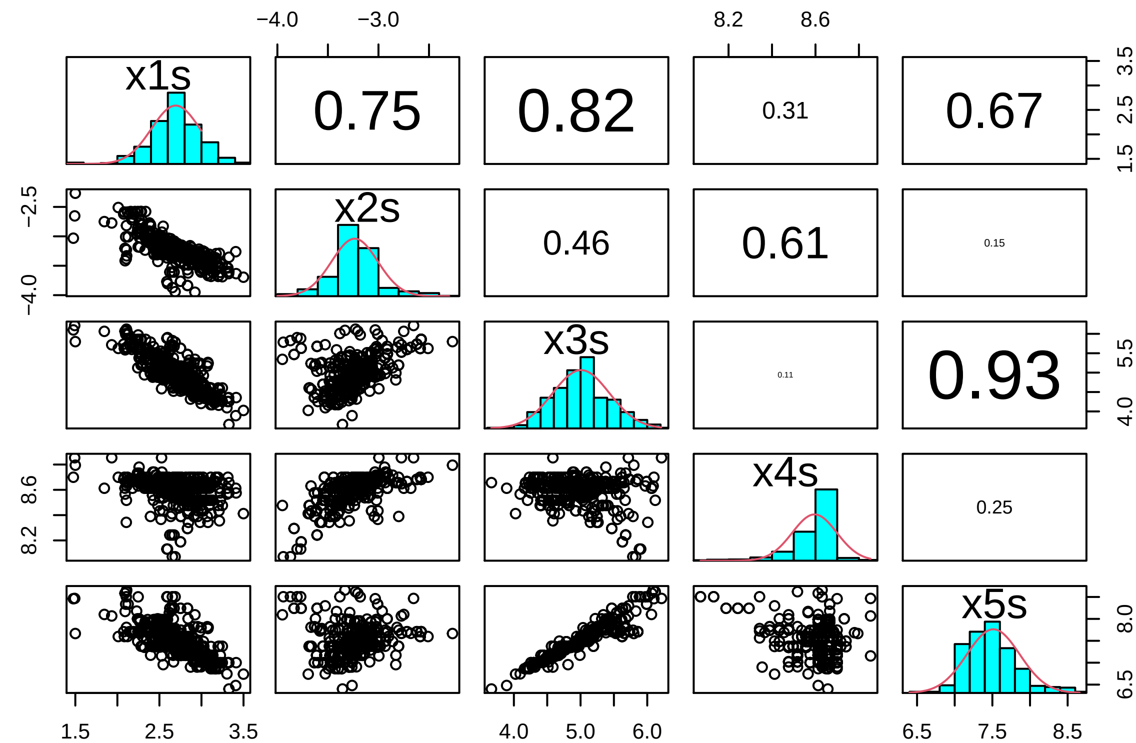
```
### data illustration

# scatter plots with  QQ plots
t.data.streu<-d.data[!is.na(d.data$S100_l),c("W_l","MP_l","CC_l","MT_l","MES_l","S100_l","TS_l")]
pairs(t.data.streu,diag.panel=panel.qq,upper.panel=panel.cor)
```




```
t.data.streu<-d.data[,c("x1s","x2s","x3s","x4s","x5s")]
pairs(t.data.streu,diag.panel=panel.qq,upper.panel=panel.cor)
```




```
# scatter plots with histogram
t.data.streu<-d.data[,c("x1s","x2s","x3s","x4s","x5s")]
pairs(t.data.streu,diag.panel=panel.hist.norm,upper.panel=panel.cor)
```



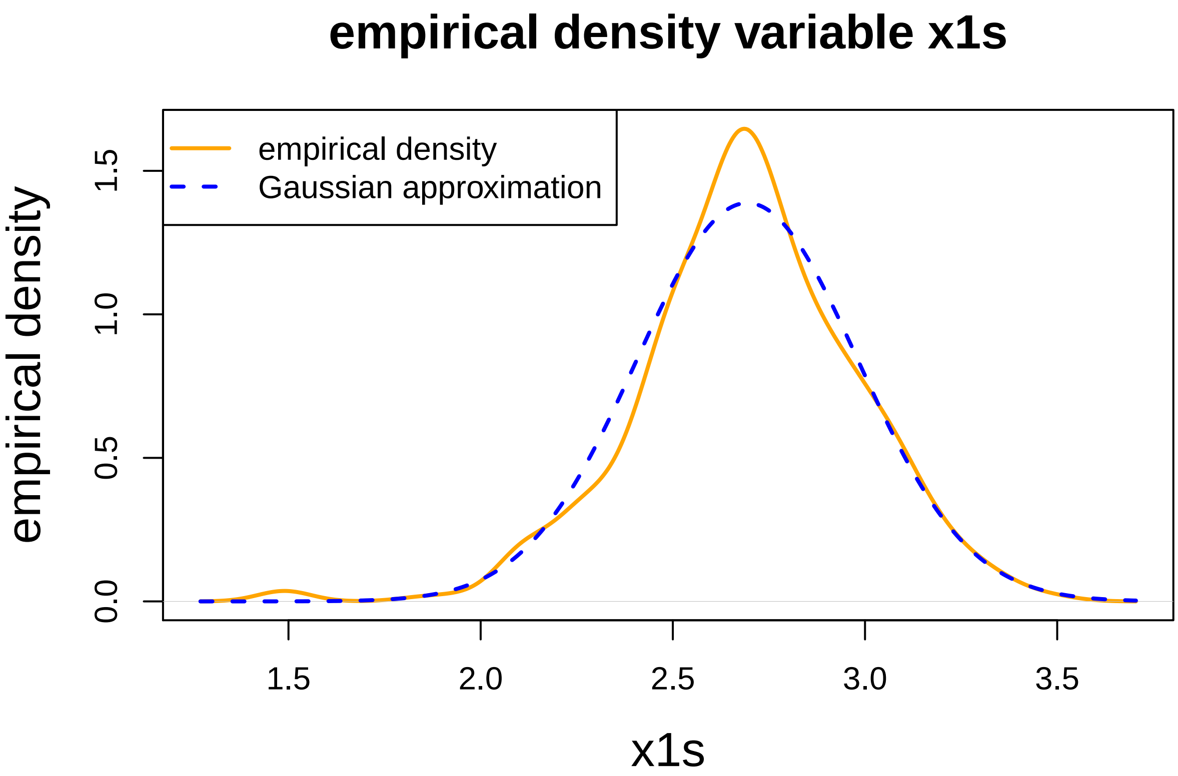

```
# empirical density plots
t.data.streu <-d.data[,c("x1s","x2s","x3s","x4s","x5s")]
(m0 <- colMeans(t.data.streu))
```

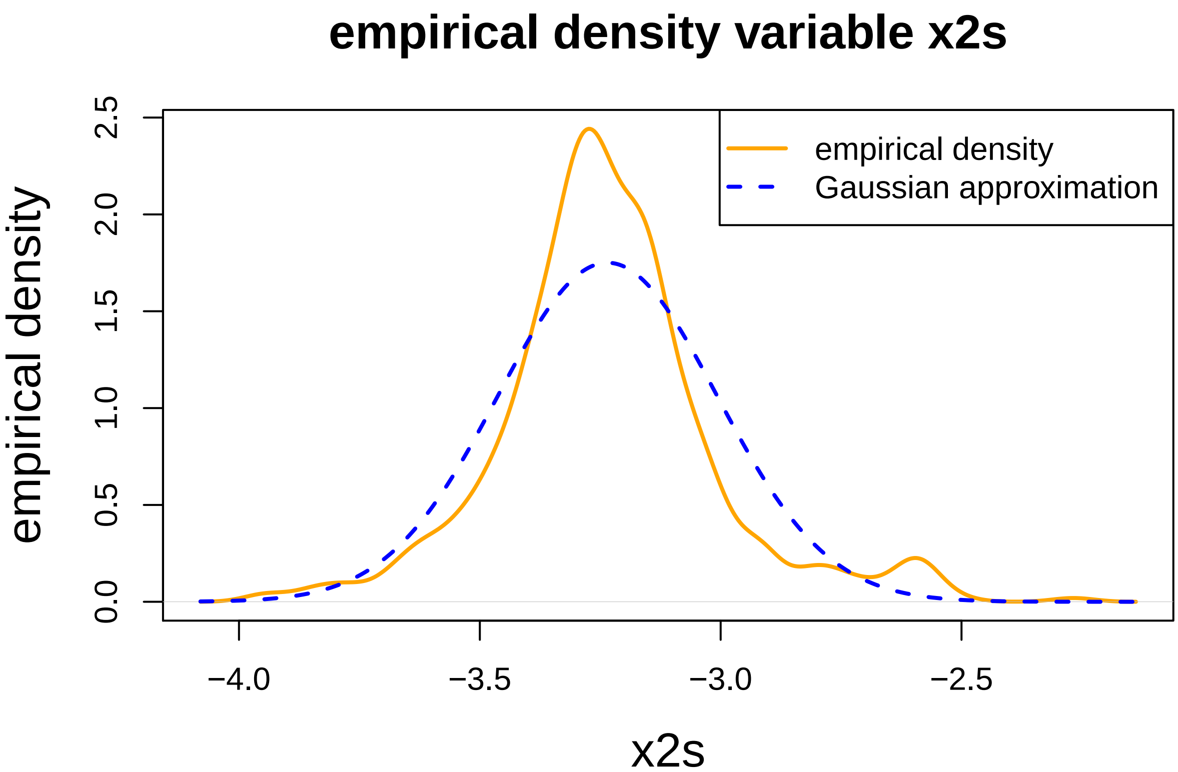
```
##        x1s       x2s       x3s       x4s       x5s
##   2.693725 -3.234989  5.007223  8.595787  7.512765
```

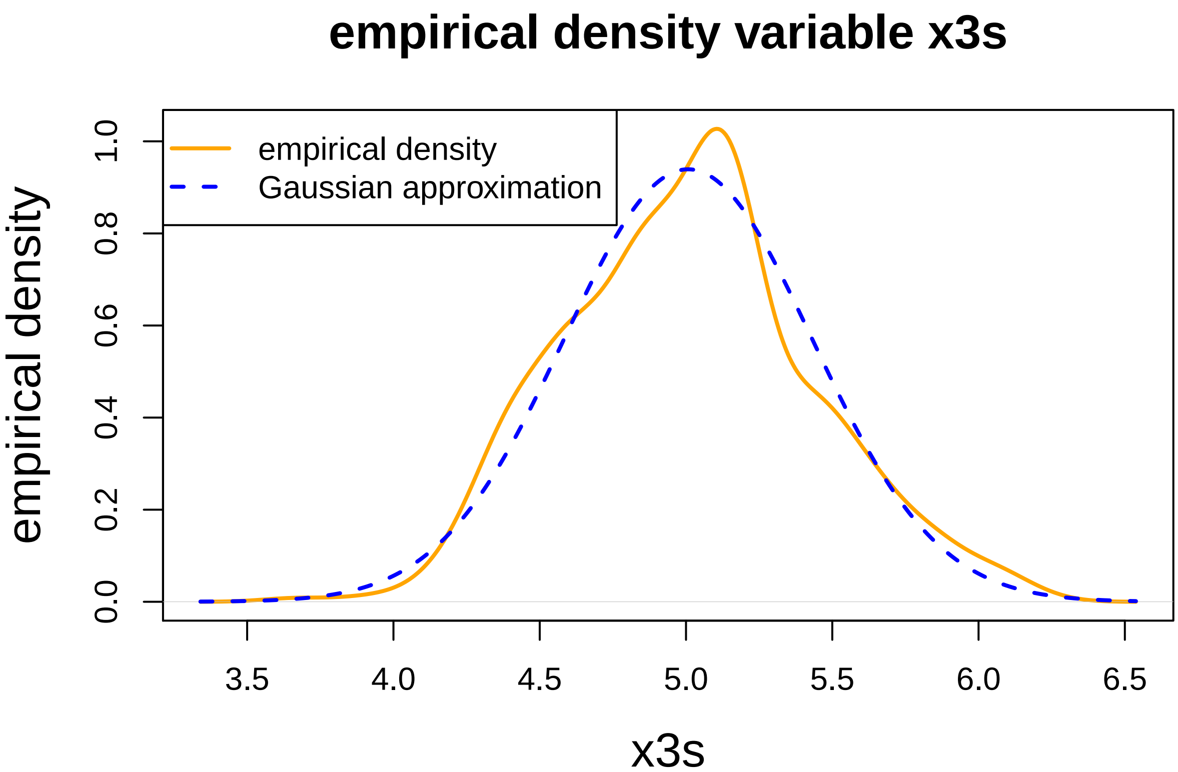
```
X01 <- t.data.streu-colMeans(t.data.streu)[col(t.data.streu)]
(sds <- sqrt(colMeans(X01^2))*sqrt(nrow(t.data.streu)/(nrow(t.data.streu)-1)))
```

```
##        x1s       x2s       x3s       x4s       x5s
## 0.2874374 0.2278949 0.4247271 0.1023802 0.3506195
```

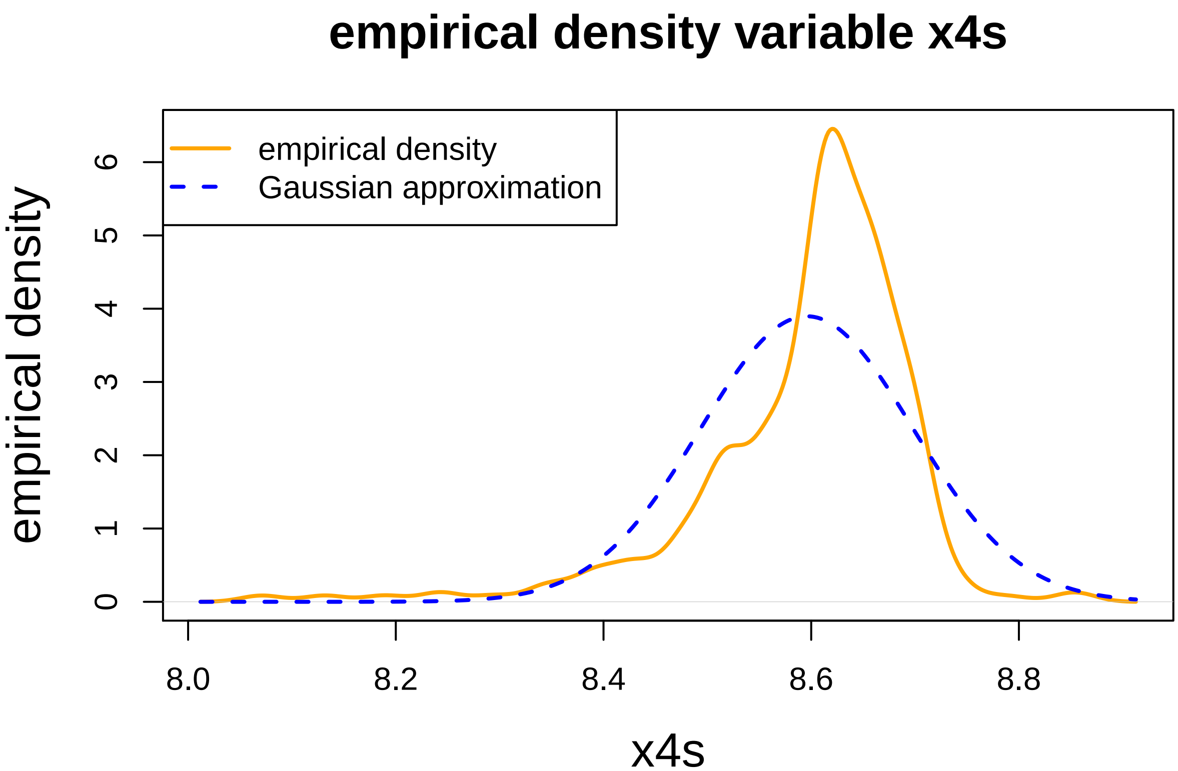
```
for (i1 in 1:5) {
  #i1 <- 1  # should be in 1:5 for xs1 to xs5
  position <- c("topleft","topright","topleft","topleft","topleft")
  plot(density(t.data.streu[,i1]), col="orange", lwd=2, ylab="empirical density", xlab=paste("x",i1, "s
  lines(density(t.data.streu[,i1])$x, dnorm(density(t.data.streu[,i1])$x, mean=m0[i1], sd=sds[i1]), col=
  legend(position[i1], c("empirical density", "Gaussian approximation"), col=c("orange", "blue"), lty=c
}
```


empirical density variable x1s
empirical density
Gaussian approximation
0.0
0.5
1.0
1.5
1.5
2.0
2.5
3.0
3.5
x1s
empirical density

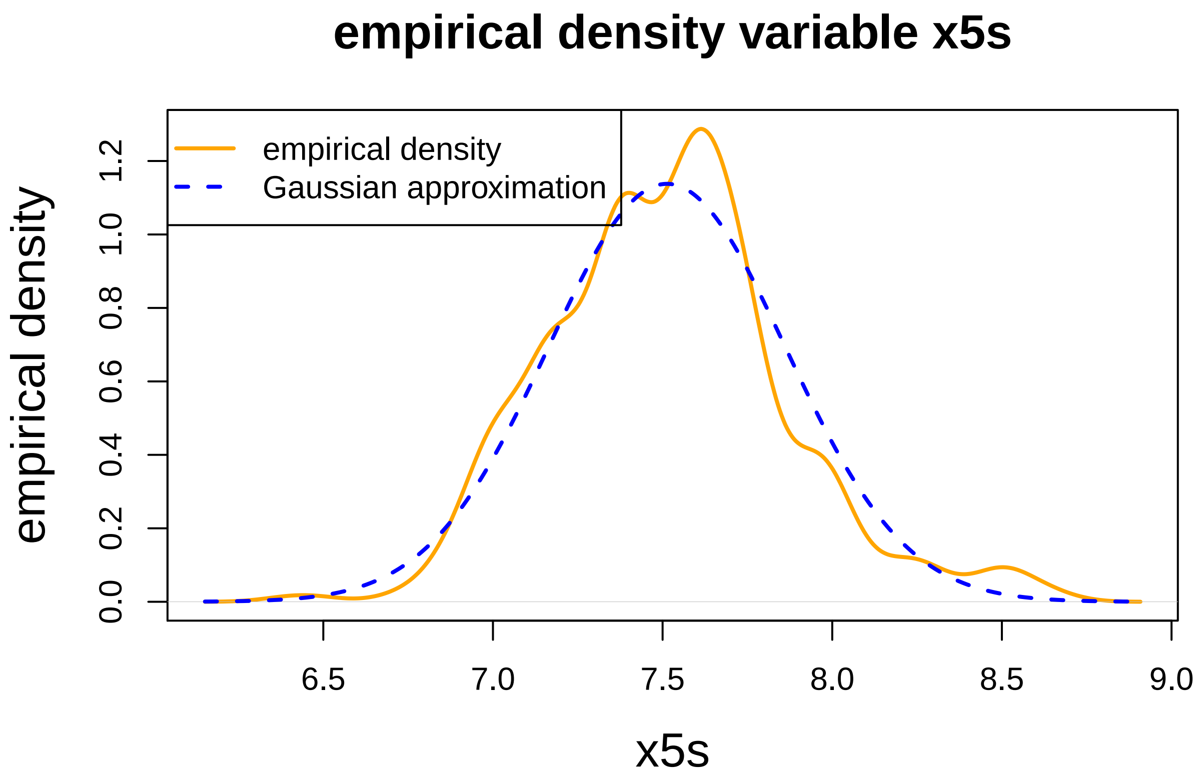

empirical density variable x2s
empirical density
Gaussian approximation
empirical density
x2s
0.0
0.5
1.0
1.5
2.0
2.5
−4.0
−3.5
−3.0
−2.5



empirical density variable x3s
empirical density
Gaussian approximation
empirical density
0.0
0.2
0.4
0.6
0.8
1.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
x3s



empirical density variable x4s
empirical density
Gaussian approximation
empirical density
0
1
2
3
4
5
6
8.0
8.2
8.4
8.6
8.8
x4s




```
### principal components analysis

# standardize matrix
X <- X01/sqrt(colMeans(X01^2))[col(X01)]

# eigenvectors and eigenvalues
X1 <- as.matrix(X)
A <-  t(X1) %*% X1
sqrt(eigen(A)$value)       # singular values
```

```
## [1] 37.532807 28.073799 11.475471  6.477146  2.123755
```

```
eigen(A)$value/nrow(X1)    # scaled eigenvalues
```

```
## [1] 2.965708690 1.659238265 0.277234620 0.088322982 0.009495442
```

```
# singular value decomposition
SVD <- svd(X1)
SVD$d                           # singular values
```

```
## [1] 37.532807 28.073799 11.475471  6.477146  2.123755
```

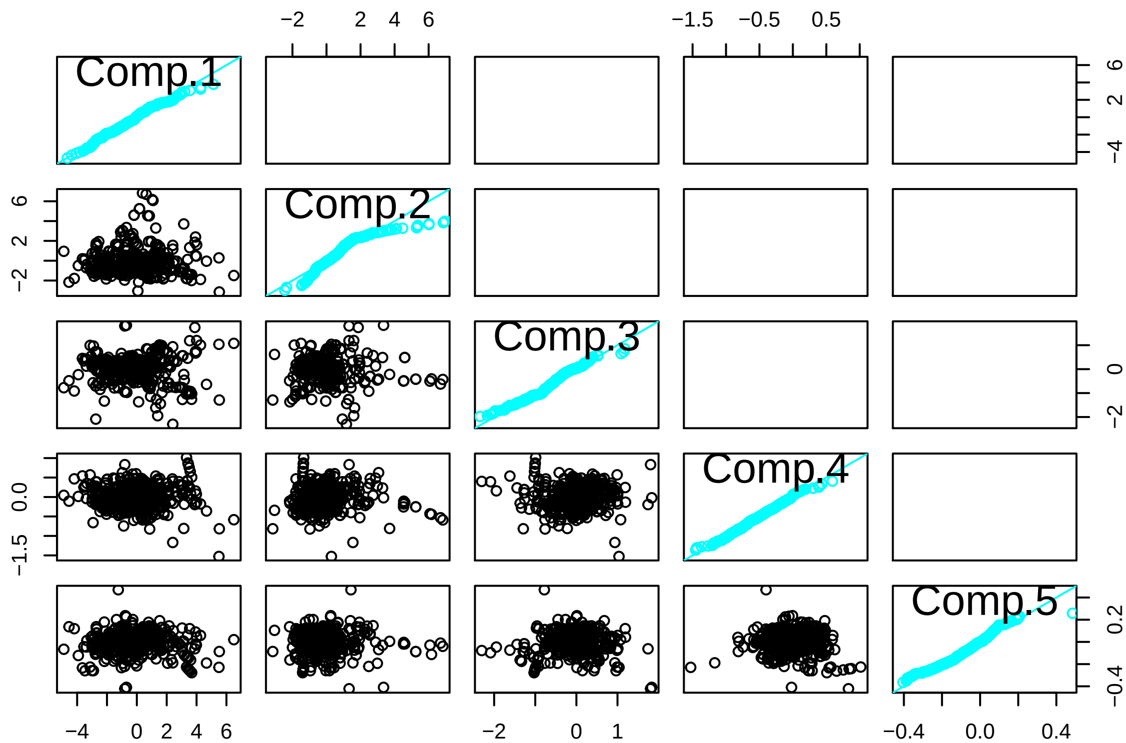
```
rbind(SVD$v[,1],SVD$v[,2])  # first two right singular vectors
```

```
##            [,1]       [,2]      [,3]       [,4]      [,5]
## [1,] -0.5577027  0.4122188 0.5389689  0.1262176 0.4611129
## [2,]  0.1025226 -0.4821905 0.2684738 -0.7045882 0.4341183
```

```r
# PCA with package PCA
t.pca <- princomp(X1,cor=TRUE)
summary(t.pca)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3    Comp.4      Comp.5
## Standard deviation     1.7221233 1.2881142 0.52653074 0.2971918 0.097444561
## Proportion of Variance 0.5931417 0.3318477 0.05544692 0.0176646 0.001899088
## Cumulative Proportion  0.5931417 0.9249894 0.98043632 0.9981009 1.000000000
```

```r
# scatter plot
switch_sign <- -1          # switch sign of the first component to make svd and princomp compatible
tt.pca <- t.pca$scores
tt.pca[,1] <- switch_sign *tt.pca[,1]
pairs(tt.pca,diag.panel=panel.qq,upper.panel=panel.cor)
```




```r
t.pca$loadings
```

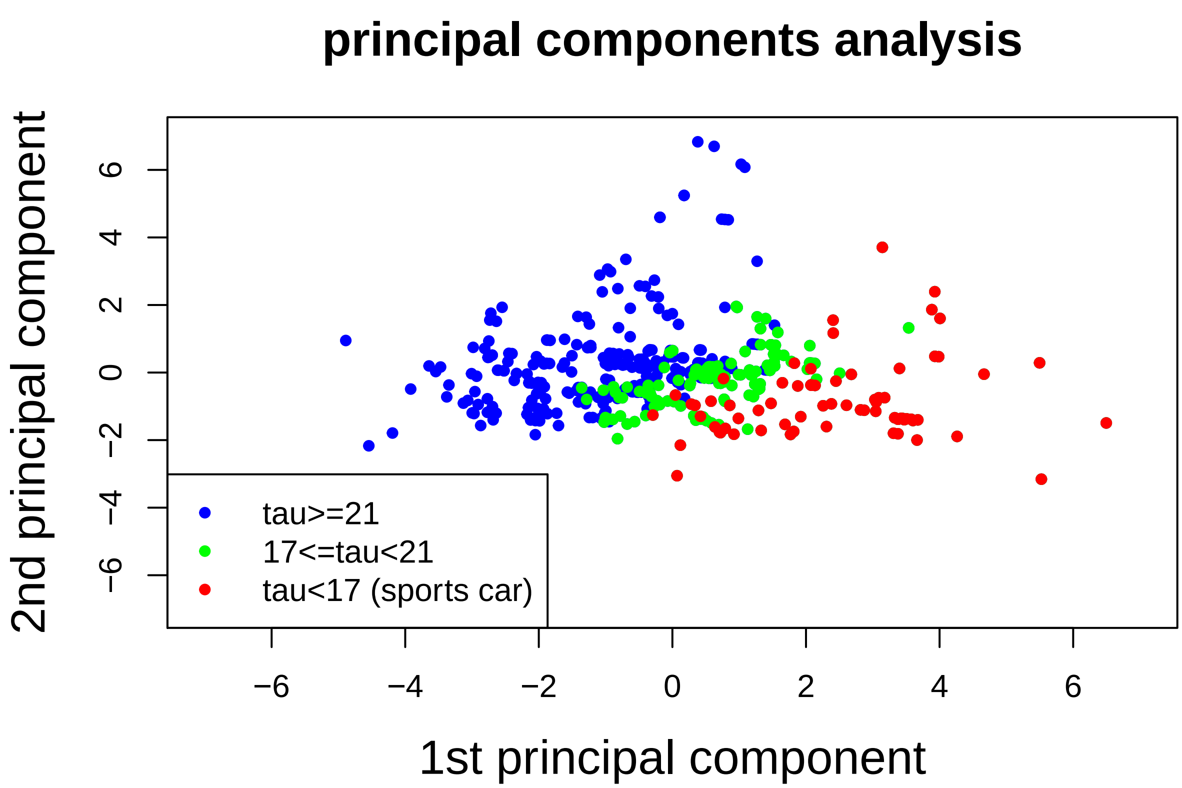
```
##
## Loadings:
##     Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## x1s  0.558  0.103         0.817
## x2s -0.412 -0.482 -0.595  0.353  0.345
## x3s -0.539  0.268         0.404 -0.689
## x4s -0.126 -0.705  0.678  0.133 -0.102
## x5s -0.461  0.434  0.427  0.165  0.624
##
```

```
##                   Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## SS loadings          1.0    1.0    1.0    1.0    1.0
## Proportion Var       0.2    0.2    0.2    0.2    0.2
## Cumulative Var       0.2    0.4    0.6    0.8    1.0
```

```
# PCA Sports Cars weights
alpha <- SVD$v[,1]/sds
(alpha_star <- c(alpha[1],alpha[2]-alpha[1], alpha[3], alpha[4], alpha[5]-alpha[2])/alpha[1])
```

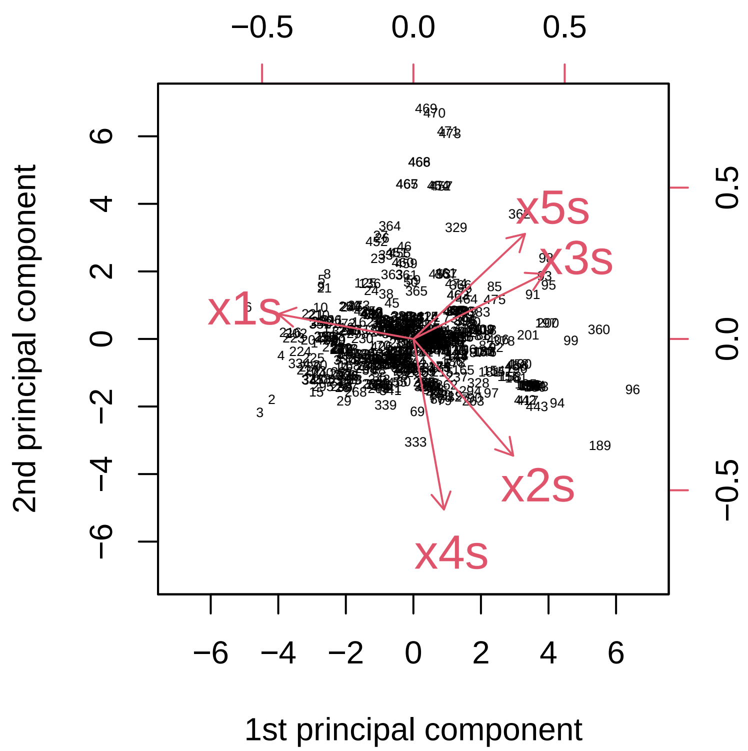
```
##        x1s        x2s        x3s        x4s        x5s
##  1.0000000 -1.9322527 -0.6540250 -0.6353958  0.2544368
```

```
# plot first two principal components
dat3 <- d.data
dat3$v1 <- X1 %*% SVD$v[,1]
dat3$v2 <- X1 %*% SVD$v[,2]

plot(x=dat3$v1, y=dat3$v2, col="blue",pch=20, ylim=c(-7,7), xlim=c(-7,7), ylab="2nd principal component
dat0 <- dat3[which(dat3$tau<21),]
points(x=dat0$v1, y=dat0$v2, col="green",pch=20)
dat0 <- dat3[which(dat3$tau<17),]
points(x=dat0$v1, y=dat0$v2, col="red",pch=20)
legend("bottomleft", c("tau>=21", "17<=tau<21", "tau<17 (sports car)"), col=c("blue", "green", "red"),
```




```
# biplot
tt.pca <- t.pca
tt.pca$scores[,1] <-  switch_sign * tt.pca$scores[,1]
tt.pca$loadings[1:5,1] <- switch_sign * tt.pca$loadings[1:5,1]
```

```
biplot(tt.pca,choices=c(1,2),scale=0, expand=2, xlab="1st principal component", ylab="2nd principal comp
```




```
# reconstruction error
reconstruction.PCA <- array(NA, c(5))

for (p in 1:5){
  Xp <- SVD$v[,1:p] %*% t(SVD$v[,1:p]) %*% t(X)
  Xp <- t(Xp)
  reconstruction.PCA[p] <- sqrt(sum(as.matrix((X-Xp)^2))/nrow(X))
             }
round(reconstruction.PCA,2)
```

```
## [1] 1.43 0.61 0.31 0.10 0.00
```

```
##### load data and pre-process data

source(file="5 - Unsupervised Learning What is a Sports Car/00_b functions bottleneck.R")

dat1 <- read.table(file="5 - Unsupervised Learning What is a Sports Car/SportsCars.csv", header=TRUE, se
str(dat1)
```

```
## 'data.frame':    475 obs. of  13 variables:
##  $ brand             : chr  "Austin" "Citroen" "Citroen" "Fiat" ...
##  $ type              : chr  "Rovermini" "Visa" "2CV" "Panda" ...
##  $ model             : chr  "Ehlemayair" "Baseclub" "Specialton" "34" ...
##  $ cubic_capacity    : int  998 652 602 850 1598 845 956 1588 1596 992 ...
##  $ max_power         : int  31 25 21 25 41 21 31 40 40 37 ...
```

```
##  $ max_torque      : num  67 49 39 60 96 56 65 100 100 98 ...
##  $ seats           : int  4 5 4 5 5 4 5 5 5 5 ...
##  $ weight          : int  620 755 585 680 1015 695 695 900 1030 920 ...
##  $ max_engine_speed: int  5000 5500 5750 5250 4600 4500 5750 4500 4800 4250 ...
##  $ seconds_to_100  : num  19.5 26.2 NA 32.3 21 NA 19.3 18.7 20 NA ...
##  $ top_speed       : int  129 125 115 125 143 115 137 148 140 130 ...
##  $ sports_car      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ tau             : num  23.3 34.1 28.6 32.8 35 ...
```

```
dat2 <- dat1
dat2$x1 <- log(dat2$weight/dat2$max_power)
dat2$x2 <- log(dat2$max_power/dat2$cubic_capacity)
dat2$x3 <- log(dat2$max_torque)
dat2$x4 <- log(dat2$max_engine_speed)
dat2$x5 <- log(dat2$cubic_capacity)
dat2 <- dat2[, c("x1","x2","x3","x4","x5")]

# normalization of design matrix
X01 <- dat2-colMeans(dat2)[col(dat2)]
X <- X01/sqrt(colMeans(X01^2))[col(X01)]
```

```
##### deep BNN Hinton-Salakhugdinov (2006) calibration

# bottleneck architecture
q1 <- 7
q2 <- 2
q0 <- ncol(X)

# pre-training 1: merging layers 1 and 3 (skipping bottleneck)
model.1 <- bottleneck.1(q0, q1)
model.1
```

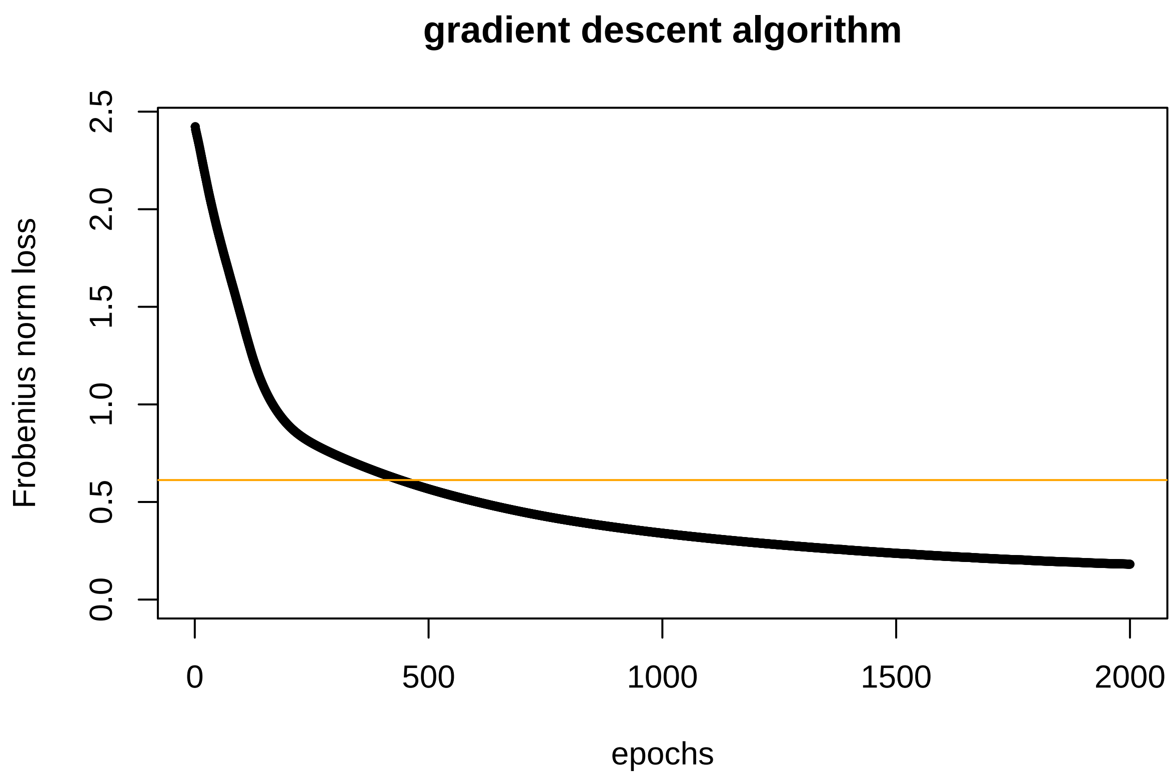
```
## Model
## Model: "model"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## Input (InputLayer)                  [(None, 5)]                         0
## ________________________________________________________________________________
## Bottleneck (Dense)                  (None, 7)                           35
## ________________________________________________________________________________
## Output (Dense)                      (None, 5)                           35
## ================================================================================
## Total params: 70
## Trainable params: 70
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
epochs <- 2000
batch_size <- nrow(X)
```

```
# fit the merged model
{t1 <- proc.time()
  fit <- model.1 %>% fit(as.matrix(X), as.matrix(X), epochs=epochs, batch_size=batch_size, verbose=0)
proc.time()-t1}
```

```
##    user  system elapsed
##    7.27    0.13    6.41
```

```
plot(x=c(1:length(fit[[2]]$loss)), y=sqrt(fit[[2]]$loss*q0),  ylim=c(0,max(sqrt(fit[[2]]$loss*q0))),pch=
abline(h=c(0.6124), col="orange")
```




```
# neuron activations in the central layer
zz <- keras_model(inputs=model.1$input, outputs=get_layer(model.1, 'Bottleneck')$output)
yy <- zz %>% predict(as.matrix(X))

# pre-training 2: middlepart
model.2 <- bottleneck.1(q1, q2)
model.2
```

```
## Model
## Model: "model_2"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## Input (InputLayer)                  [(None, 7)]                         0
## ________________________________________________________________________________
## Bottleneck (Dense)                  (None, 2)                           14
## ________________________________________________________________________________
## Output (Dense)                      (None, 7)                           14
## ================================================================================
## Total params: 28
## Trainable params: 28
```

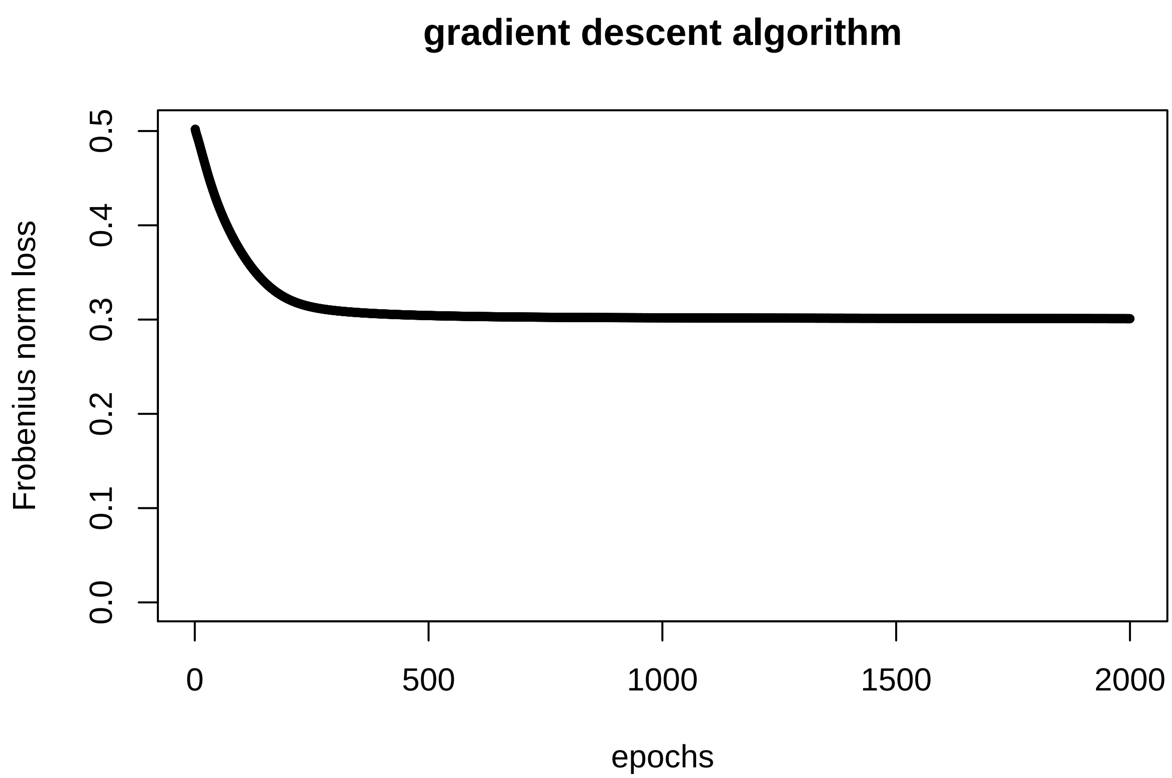
```
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
epochs <- 2000
```

```
# fit the merged model
{t1 <- proc.time()
  fit <- model.2 %>% fit(as.matrix(yy), as.matrix(yy), epochs=epochs, batch_size=batch_size, verbose=0)
proc.time()-t1}
```

```
##    user  system elapsed
##    6.50    0.14    5.10
```

```
plot(x=c(1:length(fit[[2]]$loss)), y=sqrt(fit[[2]]$loss*q0),  ylim=c(0,max(sqrt(fit[[2]]$loss*q0))),pch=
```




```
# fitting the full model
model.3 <- bottleneck.3(q0, q1, q2)
model.3
```

```
## Model
## Model: "model_3"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## Input (InputLayer)                  [(None, 5)]                         0
## ________________________________________________________________________________
## Layer1 (Dense)                      (None, 7)                           35
## ________________________________________________________________________________
```

```
## Bottleneck (Dense)                 (None, 2)                     14
## ___________________________________________________________________________
## Layer3 (Dense)                     (None, 7)                     14
## ___________________________________________________________________________
## Output (Dense)                     (None, 5)                     35
## ===========================================================================
## Total params: 98
## Trainable params: 98
## Non-trainable params: 0
## ___________________________________________________________________________
```

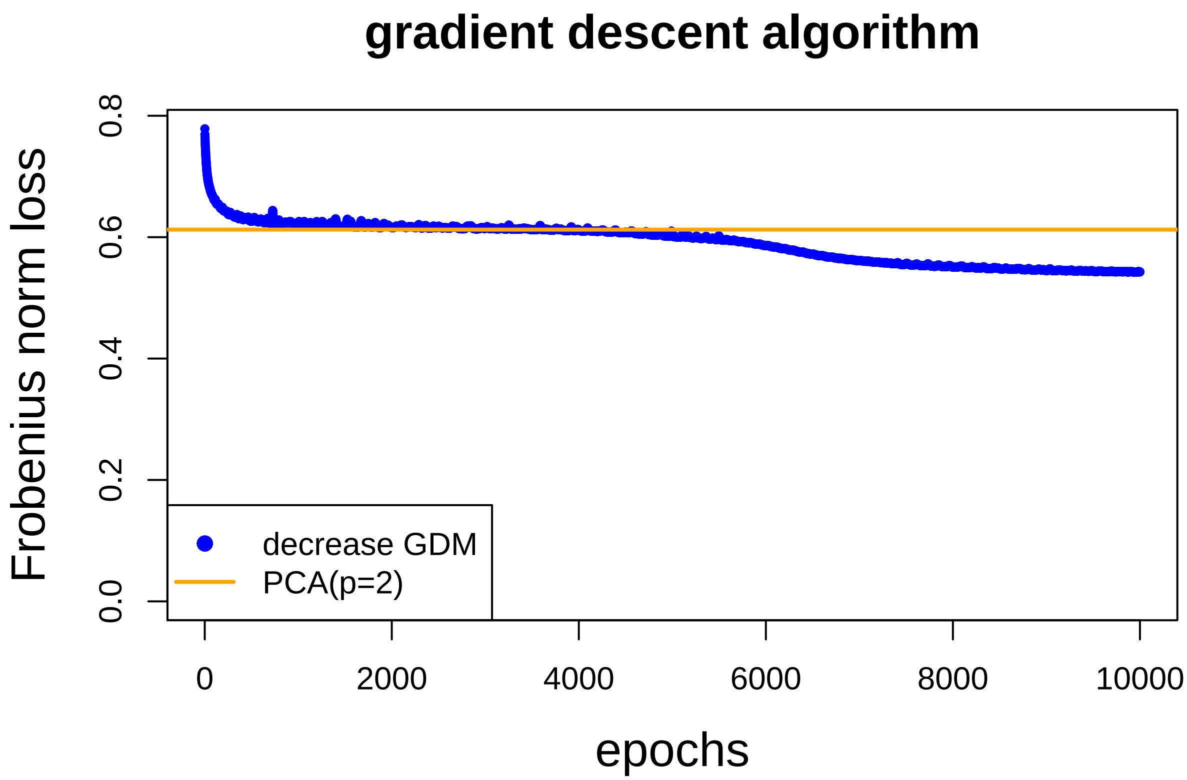
```
# set weights
weight.3 <- get_weights(model.3)
weight.1 <- get_weights(model.1)
weight.2 <- get_weights(model.2)
weight.3[[1]] <- weight.1[[1]]
weight.3[[4]] <- weight.1[[2]]
weight.3[[2]] <- weight.2[[1]]
weight.3[[3]] <- weight.2[[2]]
set_weights(model.3, weight.3)
fit0 <- model.3 %>% predict(as.matrix(X))

# reconstruction error of the pre-calibrated network
# note that this error may differ from the tutorial because we did not set a seed
round(Frobenius.loss(X,fit0),4)
```

```
## [1] 0.7786
```

```
# calibrate full bottleneck network
epochs <- 10000
batch_size <- nrow(X)
{t1 <- proc.time()
  fit <- model.3 %>% fit(as.matrix(X), as.matrix(X), epochs=epochs, batch_size=batch_size, verbose=0)
proc.time()-t1}
```

```
##    user  system elapsed
##   38.55    1.06   29.97
```

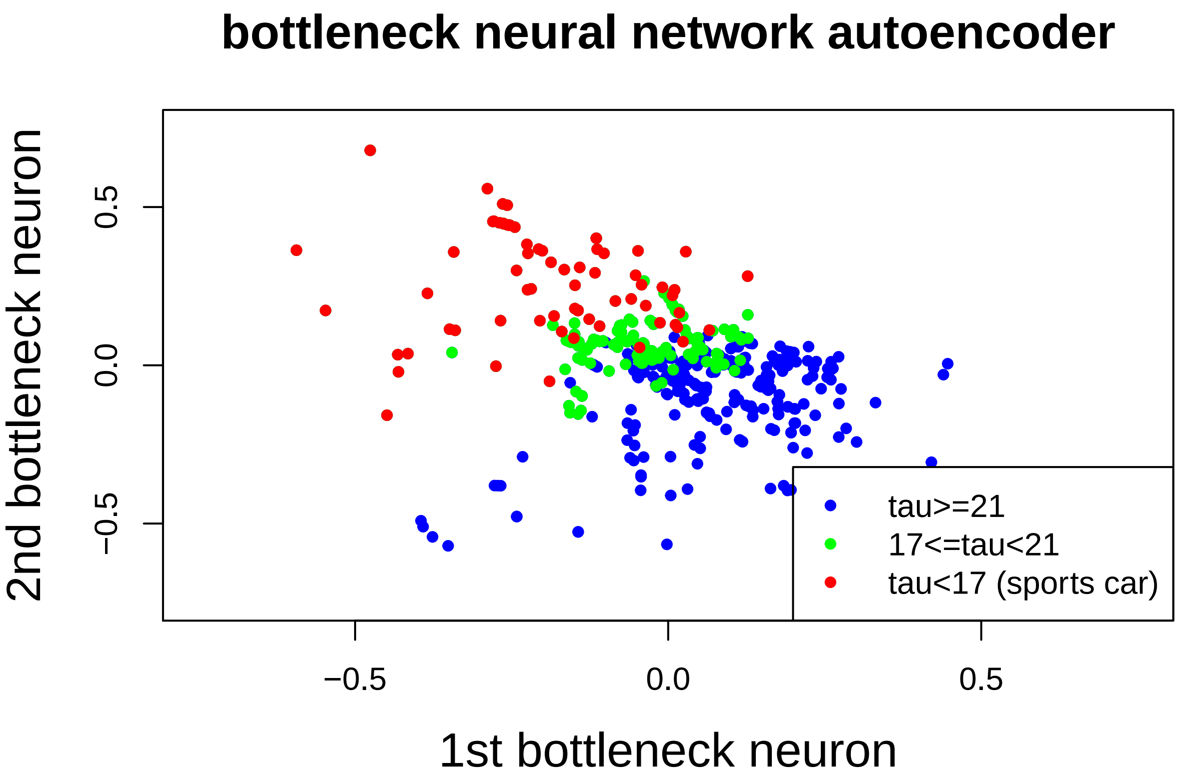
```
plot(x=c(1:length(fit[[2]]$loss)), y=sqrt(fit[[2]]$loss*q0), col="blue", ylim=c(0,max(sqrt(fit[[2]]$loss
abline(h=c(0.6124), col="orange", lwd=2)
legend("bottomleft", c("decrease GDM", "PCA(p=2)"), col=c("blue", "orange"), lty=c(-1,1), lwd=c(-1,2), 
```




```r
# reconstruction error (slightly differs from 0.5611 because of missing seed)
fit0 <- model.3 %>% predict(as.matrix(X))
round(Frobenius.loss(X,fit0),4)
```

```
## [1] 0.5428
```

```r
# read off the bottleneck activations
encoder <- keras_model(inputs=model.3$input, outputs=get_layer(model.3, 'Bottleneck')$output)
y <- encoder %>% predict(as.matrix(X))
y0 <- max(abs(y))*1.1

# note that we may need sign switches to make it comparable to PCA
plot(x=y[,1], y=y[,2], col="blue",pch=20, ylim=c(-y0,y0), xlim=c(-y0,y0), ylab="2nd bottleneck neuron",
dat0 <- y[which(dat1$tau<21),]
points(x=dat0[,1], y=dat0[,2], col="green",pch=20)
dat0 <- y[which(dat1$tau<17),]
points(x=dat0[,1], y=dat0[,2], col="red",pch=20)
legend("bottomright", c("tau>=21", "17<=tau<21", "tau<17 (sports car)"), col=c("blue", "green", "red"),
```


bottleneck neural network autoencoder
2nd bottleneck neuron
1st bottleneck neuron
0.5
0.0
−0.5
−0.5
0.0
0.5
tau>=21
17<=tau<21
tau<17 (sports car)