Lab 5

This week, we will be going through the codes from week 5 and 7 lectures. That is, continue learning about PCA. The codes throughout this notebook comes from “R Code Lecture Slides Part III; IV A and IV B”.

## PCA

Recall that we use PCA to reduce the dimension for continuous explanatory variables using principal components and complicated mathematics. We have seen in the previous Lab that there are many packages that runs PCA for us. But how can we trust that the packages work properly? I would say that R its CRAN repositories is pretty trustworthy, it is managed by a large multinational corporation Posit, which is used in many many companies in the world. However, in actuarial work, we need to explain how we got to the specific answer in as much detail as possible because the regulators tell us to… Therefore, we need to try and understand how the PCs are computed mathematically. It is also important that our work can be reproduced using a different dataset to mtcars, so let us perform PCA again on the SportsCars dataset.

Remove all objects from the R memory. What is the command for this?

rm(list=ls())

### R Code Lecture Slides Part III

Load the tidyverse, ggplot and GGally libraries

library(tidyverse)

Warning: package 'tidyverse' was built under R version 4.2.3

Warning: package 'ggplot2' was built under R version 4.2.3

Warning: package 'tibble' was built under R version 4.2.3

Warning: package 'tidyr' was built under R version 4.2.3

Warning: package 'readr' was built under R version 4.2.3

Warning: package 'purrr' was built under R version 4.2.3

Warning: package 'dplyr' was built under R version 4.2.3

Warning: package 'stringr' was built under R version 4.2.3

Warning: package 'forcats' was built under R version 4.2.3

Warning: package 'lubridate' was built under R version 4.2.3

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ dplyr 1.1.4 ✔ readr 2.1.4  
✔ forcats 1.0.0 ✔ stringr 1.5.1  
✔ ggplot2 3.5.1 ✔ tibble 3.2.1  
✔ lubridate 1.9.3 ✔ tidyr 1.3.1  
✔ purrr 1.0.2   
── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag() masks stats::lag()  
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(ggplot2)  
library(GGally)

Registered S3 method overwritten by 'GGally':  
 method from   
 +.gg ggplot2

Go onto canvas and copy and paste the codes in the chunck below:

# Read the SportsCars.csv and name this SC  
SC <- read.csv('SportsCars.csv', sep = ';', header = TRUE)  
  
# Display the structure of this data frame  
str(SC)

'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 ...

# Head and tails  
head(SC)

brand type model cubic\_capacity max\_power max\_torque seats weight  
1 Austin Rovermini Ehlemayair 998 31 67 4 620  
2 Citroen Visa Baseclub 652 25 49 5 755  
3 Citroen 2CV Specialton 602 21 39 4 585  
4 Fiat Panda 34 850 25 60 5 680  
5 Opel Ascona2 LS1.6D 1598 41 96 5 1015  
6 Renault R4 Berlinetl 845 21 56 4 695  
 max\_engine\_speed seconds\_to\_100 top\_speed sports\_car tau  
1 5000 19.5 129 0 23.33892  
2 5500 26.2 125 0 34.13032  
3 5750 NA 115 0 28.64863  
4 5250 32.3 125 0 32.84696  
5 4600 21.0 143 0 35.00643  
6 4500 NA 115 0 37.04656

tail(SC)

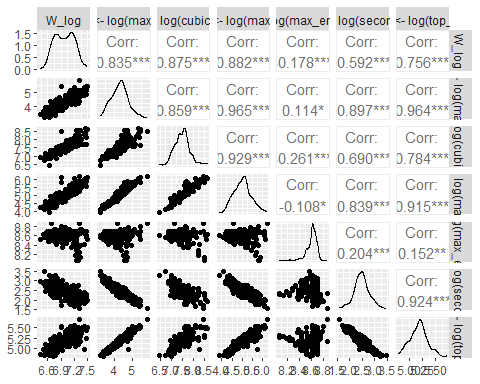
brand type model cubic\_capacity max\_power  
470 Ford Thunderbird 9 generation 4942 103  
471 Ford Thunderbird 9 generation 4942 110  
472 Ford Thunderbird 9 generation [Facelift] 3797 103  
473 Ford Thunderbird 9 generation [Facelift] 4942 114  
474 Ford Thunderbird 9 generation [Facelift] 2301 110  
475 Ford Thunderbird 9 generation [Facelift] 2301 140  
 max\_torque seats weight max\_engine\_speed seconds\_to\_100 top\_speed  
470 339 5 1464 3200 9.8 202  
471 366 5 1475 3400 9.5 200  
472 292 4 1458 3800 10.1 192  
473 359 4 1517 3400 9.6 200  
474 271 4 1533 4400 9.7 177  
475 326 4 1533 4400 7.8 232  
 sports\_car tau  
470 0 26.65330  
471 0 25.14470  
472 0 23.07005  
473 0 23.16459  
474 0 20.03996  
475 1 15.74568

#Summary  
summary(SC)

brand type model cubic\_capacity  
 Length:475 Length:475 Length:475 Min. : 602   
 Class :character Class :character Class :character 1st Qu.:1456   
 Mode :character Mode :character Mode :character Median :1796   
 Mean :1954   
 3rd Qu.:2226   
 Max. :5763   
   
 max\_power max\_torque seats weight   
 Min. : 21.00 Min. : 39.0 Min. :2.000 Min. : 585   
 1st Qu.: 54.00 1st Qu.:109.0 1st Qu.:5.000 1st Qu.: 885   
 Median : 74.00 Median :153.0 Median :5.000 Median :1065   
 Mean : 79.76 Mean :163.9 Mean :4.794 Mean :1092   
 3rd Qu.: 97.00 3rd Qu.:189.0 3rd Qu.:5.000 3rd Qu.:1266   
 Max. :335.00 Max. :500.0 Max. :7.000 Max. :1940   
   
 max\_engine\_speed seconds\_to\_100 top\_speed sports\_car   
 Min. :3200 Min. : 4.80 Min. :115.0 Min. :0.0000   
 1st Qu.:5200 1st Qu.: 9.90 1st Qu.:159.0 1st Qu.:0.0000   
 Median :5500 Median :11.70 Median :178.0 Median :0.0000   
 Mean :5436 Mean :12.38 Mean :178.4 Mean :0.1516   
 3rd Qu.:5750 3rd Qu.:14.10 3rd Qu.:195.0 3rd Qu.:0.0000   
 Max. :7000 Max. :32.30 Max. :298.0 Max. :1.0000   
 NA's :24   
 tau   
 Min. : 5.981   
 1st Qu.:18.809   
 Median :21.644   
 Mean :21.899   
 3rd Qu.:24.647   
 Max. :37.047

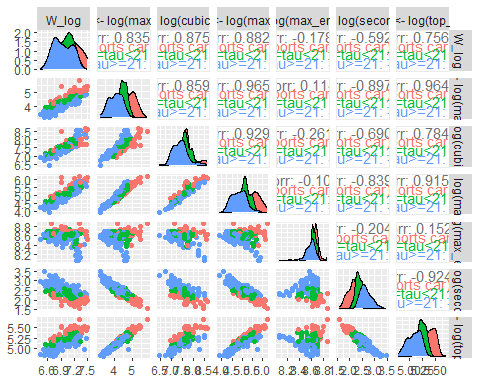
# Add the log-transformation  
SC <- SC %>% mutate(W\_log = log(weight),  
MP\_log <- log(max\_power),  
CC\_log <- log(cubic\_capacity),  
MT\_log <- log(max\_torque),  
MES\_log <- log(max\_engine\_speed),  
S100\_log <- log(seconds\_to\_100),  
TS\_log <- log(top\_speed))  
  
S100\_log <- log(SC$seconds\_to\_100)  
  
# Use ggpairs() to produce a scatterplot  
ggpairs(SC %>% filter(!is.na(S100\_log)), columns = 14:20)

Warning in geom\_point(): All aesthetics have length 1, but the data has 49 rows.  
ℹ Please consider using `annotate()` or provide this layer with data containing  
 a single row.



# Classify the cars into the groups of tau  
SC <- SC %>% mutate(sports\_type = cut(tau, breaks = c(0, 17, 21, 100),   
 labels = c('tau<17 (sports car)',   
 '17<=tau<21', 'tau>=21')))  
  
# Use ggpairs() to construct a scatter plot matrix with different colors based on the three groups of cars:  
ggpairs(data = SC %>% filter(!is.na(S100\_log)), columns = 14:20,  
 mapping = aes(colour = sports\_type),  
 upper = list(continous = wrap('cor', size = 3)))

Warning in geom\_point(): All aesthetics have length 1, but the data has 49 rows.  
ℹ Please consider using `annotate()` or provide this layer with data containing  
 a single row.



### R Code Lecture Slides Part IV A

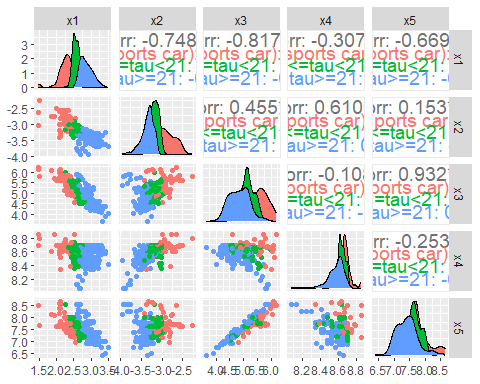
Define new features

SC <- SC %>% mutate(x1 = log(weight/max\_power),  
 x2 = log(max\_power/cubic\_capacity),  
 x3 = log(max\_torque),  
 x4 = log(max\_engine\_speed),  
 x5 = log(cubic\_capacity))

Use the ggpairs () function for constructing a scatter plot matrix with different colors based on the three groups of cars :

ggpairs(SC , columns = 22:26 , mapping = aes(colour = sports\_type ), upper = list( continuous = wrap ("cor", size = 5)))

Warning in geom\_point(): All aesthetics have length 1, but the data has 25 rows.  
ℹ Please consider using `annotate()` or provide this layer with data containing  
 a single row.

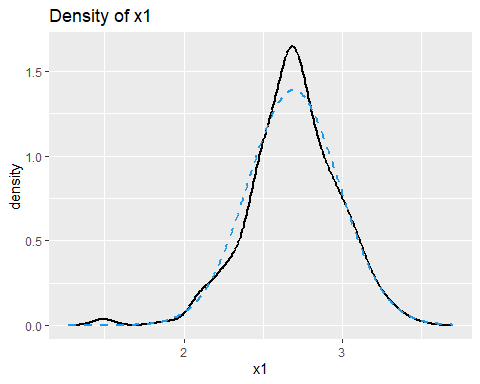


Now paste the custom function for the empirical density plots:

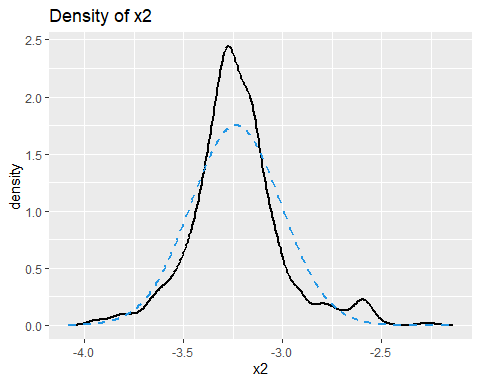
density\_plots <- function(i) {  
 # Step 1: Select the variables for which you will plot their empirical density plots  
 # Convert the string to a symbol and then evaluate it inside select  
 var <- SC %>% dplyr::select(paste0('x', i)) %>% pull  
   
 # Step 2: Compute the empirical density estimates  
 dens <- density(var)  
   
 # Step 3: Create a data frame for the Gaussian approximation  
 dat\_norm <- data.frame(x = dens$x, y = dens$y,   
 z = dnorm(dens$x, mean = mean(var), sd = sd(var)))  
   
 # Step 4: The plot with the empirical density in red and the Gaussian approximation in blue  
 p <- ggplot(data = dat\_norm) +   
 geom\_line(aes(x=x, y=y), col=1, lwd=1) +   
 geom\_line(aes(x=x, y=z), col=4, lwd=1, linetype = "dashed") +  
 labs(x = paste0("x", i), y = "density",  
 title = paste0("Density of x", i))  
 p  
}

Let us see what they tell us. Call each of the 5 density plots below:

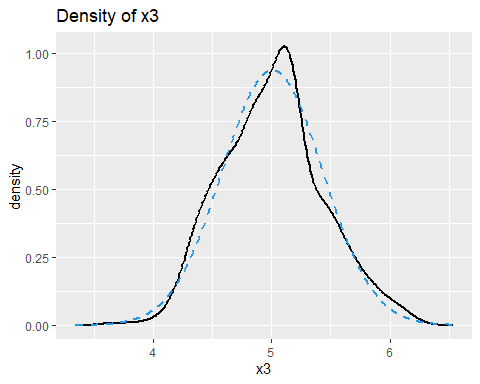
density\_plots(1)



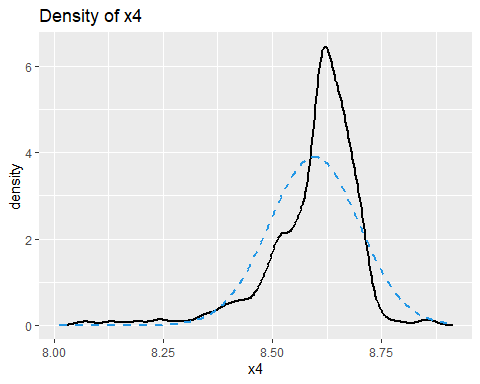
density\_plots(2)



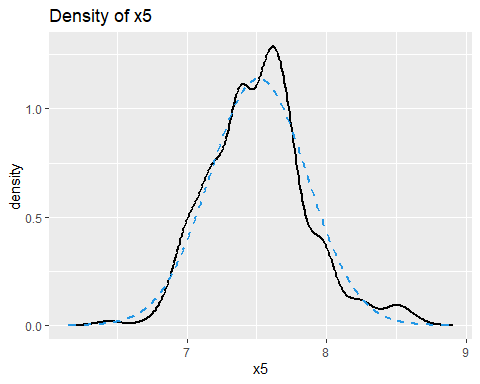
density\_plots(3)



density\_plots(4)

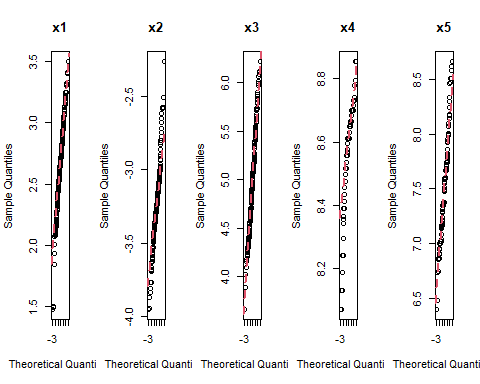


density\_plots(5)



Alternatively, we can check the QQ plots to see if the variables are normally distributed: par(mfrow = c(a,b)) allows to split the screen into a grid of a rows and b columns.

par(mfrow = c(1,5))  
for(i in 1:5){  
 var <- SC %>% dplyr::select(paste0('x',i)) %>% pull  
 qqnorm(var,main = paste0('x',i));qqline(var,col=2,lty=2,lwd=2)  
}



#### Principal Component Analysis

Step 1: Use the new set of features:

SC <- SC %>% mutate(x1 = log(weight/max\_power),  
 x2 = log(max\_power/cubic\_capacity),  
 x3 = log(max\_torque),  
 x4 = log(max\_engine\_speed),  
 x5 = log(cubic\_capacity))

Step 2: Create the raw design matrix and name it X\_raw, which includes x1 to x5.

X\_raw <- SC %>% dplyr::select(x1,x2,x3,x4,x5)

Step 3: Standardize the raw design matrix Xraw by column means and column standard deviations:

X <- apply(X\_raw, 2, function(x) (x - mean(x))/sd(x))

Now check the correlation matrix

#The correlation matrix:  
var(X)

x1 x2 x3 x4 x5  
x1 1.0000000 -0.7484100 -0.8172558 -0.3073629 -0.6690386  
x2 -0.7484100 1.0000000 0.4551775 0.6100167 0.1530871  
x3 -0.8172558 0.4551775 1.0000000 -0.1075713 0.9316912  
x4 -0.3073629 0.6100167 -0.1075713 1.0000000 -0.2532848  
x5 -0.6690386 0.1530871 0.9316912 -0.2532848 1.0000000

#OR  
cor(X)

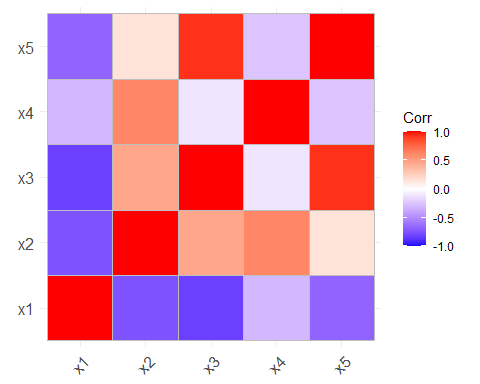
x1 x2 x3 x4 x5  
x1 1.0000000 -0.7484100 -0.8172558 -0.3073629 -0.6690386  
x2 -0.7484100 1.0000000 0.4551775 0.6100167 0.1530871  
x3 -0.8172558 0.4551775 1.0000000 -0.1075713 0.9316912  
x4 -0.3073629 0.6100167 -0.1075713 1.0000000 -0.2532848  
x5 -0.6690386 0.1530871 0.9316912 -0.2532848 1.0000000

We can also visualize it using the ggcorplot library

library(ggcorrplot)

Warning: package 'ggcorrplot' was built under R version 4.2.3

ggcorrplot(cor(X))



### R Code Lecture Slides Part IV B

Step 1: Run the codes above (redundant as we are in a notebook)

Step 2: Compute the SVD of the scaled matrix X

svdX <- svd(X)

Step 3: Display the structure of the svdX object using the str() function:

str(svdX)

List of 3  
 $ d: num [1:5] 37.49 28.04 11.46 6.47 2.12  
 $ u: num [1:475, 1:5] -0.0781 -0.1116 -0.1211 -0.1044 -0.0724 ...  
 $ v: num [1:5, 1:5] -0.558 0.412 0.539 0.126 0.461 ...

Step 4: Extract the V matrix:

V <- svdX$v

Step 5: Check the orthogonality of the matrices V and t(V):

t(V) %\*% V

[,1] [,2] [,3] [,4] [,5]  
[1,] 1.000000e+00 0.000000e+00 -1.110223e-16 1.804112e-16 -2.220446e-16  
[2,] 0.000000e+00 1.000000e+00 2.498002e-16 4.163336e-17 -5.551115e-17  
[3,] -1.110223e-16 2.498002e-16 1.000000e+00 -1.249001e-16 1.110223e-16  
[4,] 1.804112e-16 4.163336e-17 -1.249001e-16 1.000000e+00 -1.942890e-16  
[5,] -2.220446e-16 -5.551115e-17 1.110223e-16 -1.942890e-16 1.000000e+00

Note that the off-diagonal elements are not exactly 0 but they are very close to 0 which is fine.

#### PC Scratch

Now, we construct the principal components and summarize the results.

Step 1: Compute the principal components (PCs) as Y=XV. Each column of Y represents a PC and each column of V provides the coefficients for x1,...,x5.

Y <- X %\*% V

Step 2: Extract the singular values for each PC and compute the standard deviation of each principal component, the variation explained by each PC and the cumulative variation:

pca\_summary <- data.frame(  
 p = 1:ncol(X),  
 sng\_values = svdX$d,  
 sd\_pcs = sqrt(svdX$d^2/(nrow(X)-1)),  
 explained\_variation = svdX$d^2/sum(svdX$d^2),  
 cum\_variation = cumsum(svdX$d^2/sum(svdX$d^2))  
 )

Step 3: Summarize the results:

pca\_summary

p sng\_values sd\_pcs explained\_variation cum\_variation  
1 1 37.493278 1.72212331 0.593141738 0.5931417  
2 2 28.044232 1.28811423 0.331847653 0.9249894  
3 3 11.463386 0.52653074 0.055446924 0.9804363  
4 4 6.470324 0.29719183 0.017664596 0.9981009  
5 5 2.121518 0.09744456 0.001899088 1.0000000

#### princomp

Perform PCA using the princomp() function

Step 1: Use the princomp() function for the scaled matrix X and set cor=TRUE which means that the calculation will be based on eigen-decomposition of the matrix A = t(X) %\*% X:

pca <- princomp(X,cor=TRUE)

Step 2: Summarize the results:

summary(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

Step 3: Compare with the previous results:

pca\_summary

p sng\_values sd\_pcs explained\_variation cum\_variation  
1 1 37.493278 1.72212331 0.593141738 0.5931417  
2 2 28.044232 1.28811423 0.331847653 0.9249894  
3 3 11.463386 0.52653074 0.055446924 0.9804363  
4 4 6.470324 0.29719183 0.017664596 0.9981009  
5 5 2.121518 0.09744456 0.001899088 1.0000000

Display the structure of the pca object using the str() function in R:

str(pca)

List of 7  
 $ sdev : Named num [1:5] 1.7221 1.2881 0.5265 0.2972 0.0974  
 ..- attr(\*, "names")= chr [1:5] "Comp.1" "Comp.2" "Comp.3" "Comp.4" ...  
 $ loadings: 'loadings' num [1:5, 1:5] 0.558 -0.412 -0.539 -0.126 -0.461 ...  
 ..- attr(\*, "dimnames")=List of 2  
 .. ..$ : chr [1:5] "x1" "x2" "x3" "x4" ...  
 .. ..$ : chr [1:5] "Comp.1" "Comp.2" "Comp.3" "Comp.4" ...  
 $ center : Named num [1:5] -1.53e-16 -7.88e-16 -9.70e-17 2.07e-15 -7.15e-17  
 ..- attr(\*, "names")= chr [1:5] "x1" "x2" "x3" "x4" ...  
 $ scale : Named num [1:5] 0.999 0.999 0.999 0.999 0.999  
 ..- attr(\*, "names")= chr [1:5] "x1" "x2" "x3" "x4" ...  
 $ n.obs : int 475  
 $ scores : num [1:475, 1:5] 2.93 4.19 4.54 3.92 2.72 ...  
 ..- attr(\*, "dimnames")=List of 2  
 .. ..$ : NULL  
 .. ..$ : chr [1:5] "Comp.1" "Comp.2" "Comp.3" "Comp.4" ...  
 $ call : language princomp(x = X, cor = TRUE)  
 - attr(\*, "class")= chr "princomp"

Now we compare the eigenvectors which are computed using the svd() function to those which are calculated via the princomp() function.

#scd() vs pca() for the 1st eigenvectors:  
V[,1]; pca$loadings[,1]

[1] -0.5577027 0.4122188 0.5389689 0.1262176 0.4611129

x1 x2 x3 x4 x5   
 0.5577027 -0.4122188 -0.5389689 -0.1262176 -0.4611129

#### Plots

Next, we plot a scatter plot to show the cars represented by the five principal components coordinates.

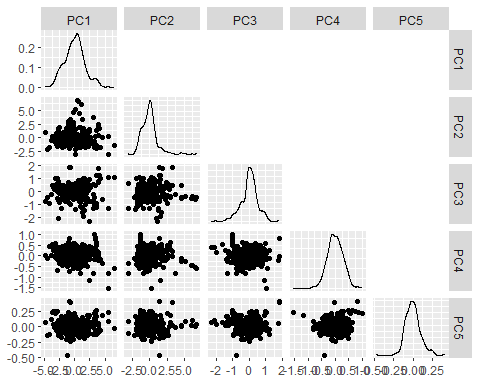
Step 1: Give names to the PCs and combine thim with sports car data set using cbind():

colnames(Y) <- names(c(PC = 1:5))  
SC <- cbind(SC, Y)

Step 2: Plot the scatter plots using the ggpairs() function:

ggpairs(data.frame(Y), upper = NULL)

Warning in geom\_point(): All aesthetics have length 1, but the data has 25 rows.  
ℹ Please consider using `annotate()` or provide this layer with data containing  
 a single row.



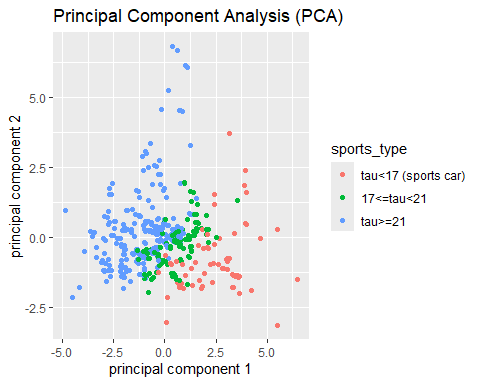
Next, we graphically illustrate all cars along the first two principal component axis:

Step 1: Classify the cars using the values for tau:

SC <- SC %>% mutate(sports\_type = cut(tau,breaks = c(0, 17, 21, 100),  
labels = c("tau<17 (sports car)","17<=tau<21", "tau>=21")))

Step 2: Plot the three types of cars using the first two principal components:

ggplot(SC, aes(x = PC1, y = PC2, colour = sports\_type)) + geom\_point() +  
labs(y = paste0("principal component ", 2),  
x = paste0("principal component ", 1),  
title = "Principal Component Analysis (PCA)")

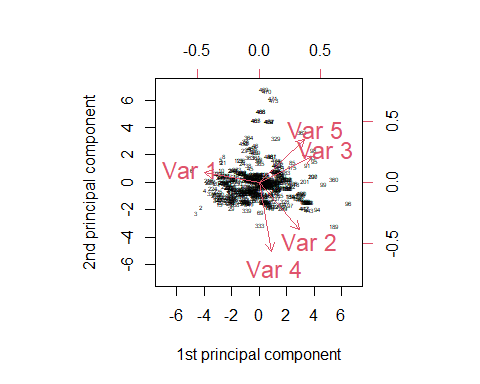


We can also graphically illustrate the same results using the biplot() function in R that we also used in the previous class.

Step 1: Use biplot() function with the first two PCs in the first argument and the laodings (eigenvectors) in the second argument:

Step 2: Add labels to the graph for the first two PCs:

biplot(Y[,1:2],V[,1:2], expand = 2,  
xlab = "1st principal component", ylab = "2nd principal component",  
cex = c(0.4, 1.5), ylim = c(-7, 7), xlim = c(-7, 7))



#### Reconstruction Errors

The reconstruction error can be seen as average squared distance between the original data points and respective projections onto principal subspace.

The reconstruction errors are scaled by . We use the for() loop to compute these and we observe that for p = 2 PCs we receive a reconstruction error of .

recon\_err <- 0  
for(p in 1:5){  
Xp <- X %\*% V[,1:p] %\*% t(V[,1:p])  
recon\_err[p] <- sqrt(sum( (X - Xp)^2)/nrow(X))  
 }  
round(recon\_err,5)

[1] 1.42478 0.61177 0.31243 0.09734 0.00000