Module 2: Recommended Exercises - Solution

TMA4268 Statistical Learning V2022

Emma Skarstein, Daesoo Lee, Stefanie Muff
Department of Mathematical Sciences, NTNU

January 17, 2022

Last changes: (14.01.2022)

Problem 1

a) Classification

   Example 1: Cancer diagnostics. Response: cancer (yes/no). Predictors: smoking, age, family history, gene expression ect. Goal: prediction.

   Example 2: Stock market price direction. Response: up/down. Predictors: yesterday's price movement change, two previous days price movement ect. Goal: inference.

   Example 3: Flower species. Response: species. Predictors: color, height, leafes ect. Goal: prediction

b) Regression

   Example 1: Illness classification. Response: age of death. Predictors: current age, gender, resting heart rate, resting breath rate ect. Goal: prediction

   Example 2: House price. Response: Price. Predictors: age of house, price of neighbourhood, crime rate, distance to town, distance to school, ect. Goal: prediction

   Example 3: What affects O2-uptake. Response: O2-uptake. Predictors: gender, age, amount of weekly exercise, type of exercise, smoking, heart disease, ect. Goal: inference

Problem 2

a) Based on the three different models compared here, a rigid method will have the highest test MSE. However, this does not always have to be the case. For instance, in figure 2.10, we see that the most flexible model has the highest test MSE. See also figure 2.12 that summarizes the last three figures and shows some of the variation. So in general we cannot say anything certain about how the test MSE will vary based on model flexibility, this will depend on how the training data compares to the test data, and other model characteristics. However (though the question does not ask this), the *training* MSE will always decrease as the flexibility increases.
b) A small (test) variance implies an underfit to the data.
c) See figure 2.12. Underfit - low variance - high bias. Overfit - high variance - low bias. We wish to find a model that lies somewhere inbetween, with low variance and low bias.

Problem 3

```r
#install.packages("ISLR")
library(ISLR)
data(Auto)
```

a)

```r
str(Auto)
```

```
## 'data.frame':    392 obs. of  9 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : num  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : num  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ name        : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 14 161 141 54 223 241 2
```

```r
summary(Auto)
```

```
##       mpg          cylinders      displacement     horsepower        weight
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140
##
##   acceleration        year           origin                      name
##  Min.   : 8.00   Min.   :70.00   Min.   :1.000   amc matador       :  5
##  1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5
##  Median :15.50   Median :76.00   Median :1.000   toyota corolla    :  5
##  Mean   :15.54   Mean   :75.98   Mean   :1.577   amc gremlin       :  4
##  3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4
##  Max.   :24.80   Max.   :82.00   Max.   :3.000   chevrolet chevette:  4
##                                                  (Other)           :365
```

The dimensions are 392 observations (rows) of 9 variables (columns). See from looking at the structure (`str()`) and `summary()` that cylinders (taking values 3,4,5,6,8), origin (taking values 1,2,3) and name (name of the cars) are qualitative predictors. The rest of the predictors are quantitative.

b) To see the range of the quantitative predictors, either apply the `range()` function to each column with a quantitative predictor separately

```r
range(Auto[,1])
```

```
## [1]  9.0 46.6
```

```r
range(Auto[,3])
```

```
## [1]  68 455
```

```r
range(Auto[,4])
```

```
## [1]  46 230
```

```r
range(Auto[,5])
```

```
## [1] 1613 5140
```

```r
range(Auto[,6])
```

```
## [1]  8.0 24.8
```

```r
range(Auto[,7])
```

```
## [1] 70 82
```

or use the `sapply()` function to run the `range()` function on the specified columns with a single line of code:

```r
quant = c(1,3,4,5,6,7)
sapply(Auto[, quant], range)
```

```
##       mpg displacement horsepower weight acceleration year
## [1,]  9.0           68         46   1613          8.0   70
## [2,] 46.6          455        230   5140         24.8   82
```

c) To get the and standard deviation of the quantitative predictors, we can again either use the `sapply()` function in the same manner as above, or apply the `mean()` and `sd()` commands columnwise.

```r
#mean
sapply(Auto[, quant], mean)
```

```
##          mpg displacement   horsepower       weight acceleration         year
##     23.44592    194.41199    104.46939   2977.58418     15.54133     75.97959
```

```r
#or
mean(Auto[,1])
```

```
## [1] 23.44592
```

```r
mean(Auto[,3])
```

```
## [1] 194.412
```

```r
mean(Auto[,4])
```

```
## [1] 104.4694
```

```r
mean(Auto[,5])
```

```
## [1] 2977.584
```

```r
mean(Auto[,6])
```

```
## [1] 15.54133
```

```r
mean(Auto[,7])
```

```
## [1] 75.97959
```

```r
#or
colMeans(Auto[,quant])
```

```
##          mpg displacement   horsepower       weight acceleration         year 
##     23.44592    194.41199    104.46939   2977.58418     15.54133     75.97959
```

```r
#sd
sapply(Auto[, quant], sd)
```

```
##          mpg displacement   horsepower       weight acceleration         year 
##     7.805007   104.644004    38.491160   849.402560     2.758864     3.683737
```

```r
#or
sd(Auto[,1])
```

```
## [1] 7.805007
```

```r
sd(Auto[,3])
```

```
## [1] 104.644
```

```r
sd(Auto[,4])
```

```
## [1] 38.49116
```

```r
sd(Auto[,5])
```

```
## [1] 849.4026
```

```r
sd(Auto[,6])
```

```
## [1] 2.758864
```

```r
sd(Auto[,7])
```

```
## [1] 3.683737
```

d) Remove 10th to 85th observations and look at the range, mean and standard deviation of the reduced set. We now only show the solutions using `sapply()` to save space.

```r
#remove observations
ReducedAuto = Auto[-c(10:85),]

#range, mean and sd
sapply(ReducedAuto[, quant], range)
```

```
##       mpg displacement horsepower weight acceleration year
## [1,] 11.0           68         46   1649          8.5   70
## [2,] 46.6          455        230   4997         24.8   82
```

```r
sapply(ReducedAuto[, quant], mean)
```

```
##          mpg displacement   horsepower       weight acceleration         year
##     24.40443    187.24051    100.72152   2935.97152     15.72690     77.14557
```

```r
sapply(ReducedAuto[, quant], sd)
```

```
##          mpg displacement   horsepower       weight acceleration         year
##     7.867283    99.678367    35.708853   811.300208     2.693721     3.106217
```

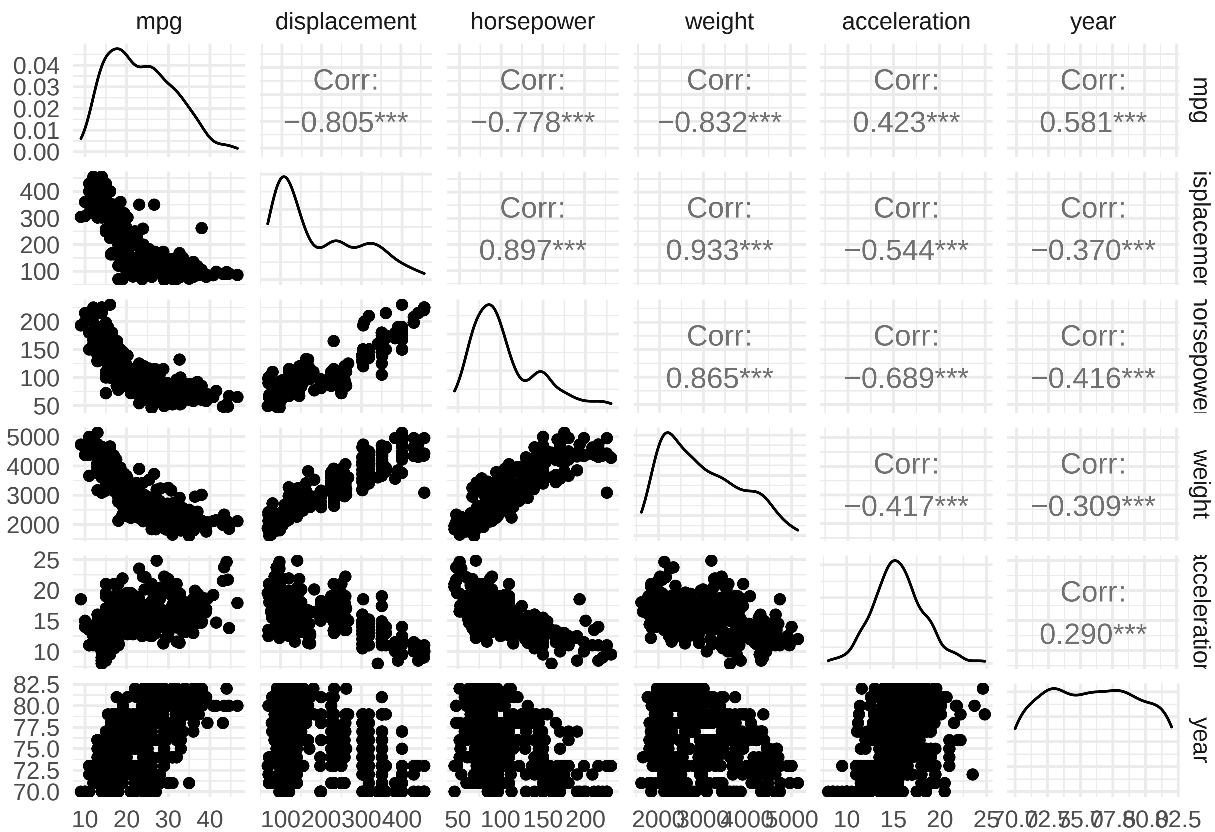
e) Make a scatterplot of the full dataset using the `ggpairs()` function.

```r
library(GGally)
```

```
## Loading required package: ggplot2
```

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

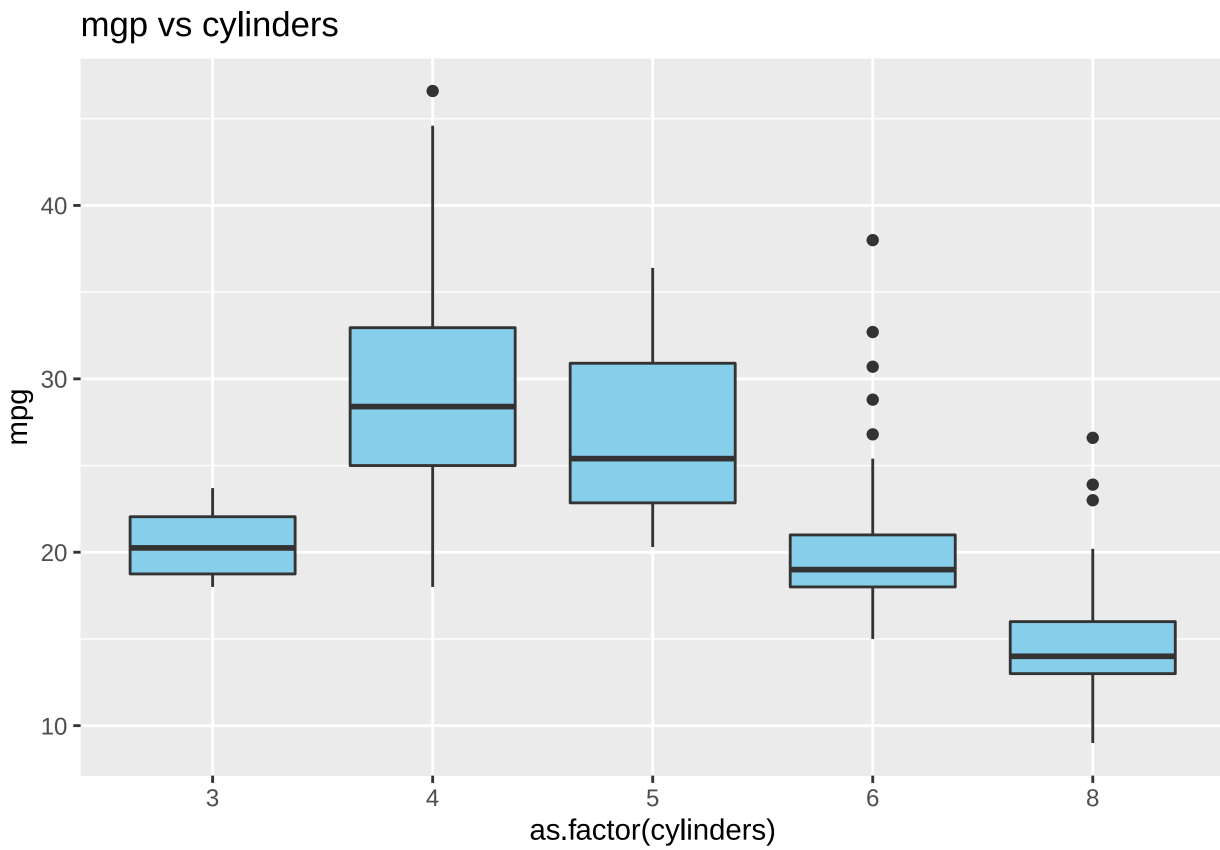
```r
ggpairs(Auto[,quant]) + theme_minimal()
```



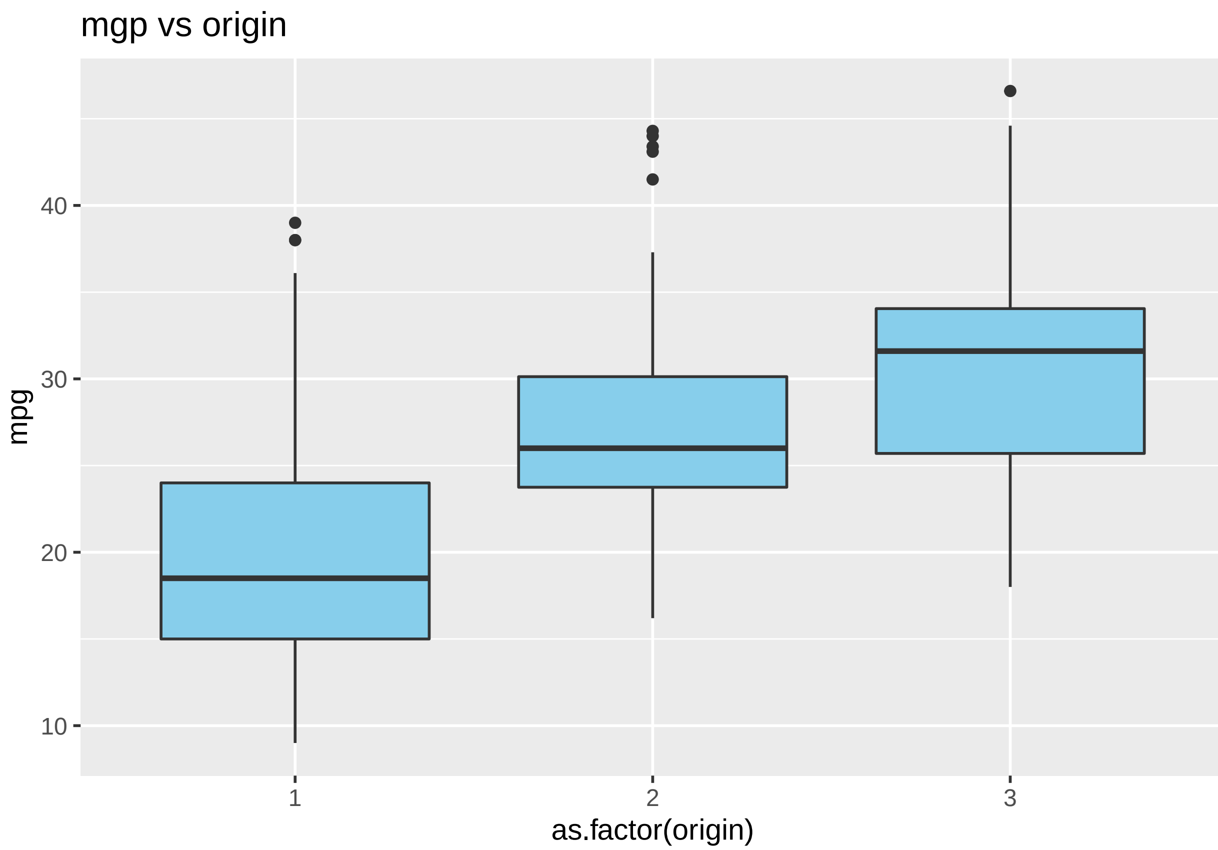
We see that there seems to be strong relationships (based on curve trends and correlation) between the pairs: `mpg` and `displacement`, `mpg` and `horsepower`, `mpg` and `weight`, `displacement` and `horsepower`, `displacement` and `weight`, `horsepower` and `weght`, and `horsepower` and `acceleration`.

f) Wish to predict gas milage based on the other variables. From the scatterplot we see that `displacement`, `horsepower` and `weight` could be good choises for prediction of `mpg`. Check if the qualitative predictors could also be good choises by plotting them against `mpg`.

```
ggplot(Auto, aes(as.factor(cylinders), mpg)) +
  geom_boxplot(fill="skyblue") +
  labs(title = "mgp vs cylinders")
```




```
ggplot(Auto, aes(as.factor(origin), mpg)) +
  geom_boxplot(fill="skyblue") +
  labs(title = "mgp vs origin")
```



From these plots we see that both `cylinders` and `origin` could be good choices for prediction of `mgp`, because the miles per gallon (mpg) seems to depend on these two variables.

g) To find the correlation of the given variables, we need the covariance of these variable as well as the standard deviations, which are both available in the covariance matrix. Remember the the variance of each variable is given in the diagonal of the covariance matrix.

```
covMat = cov(Auto[,quant])
covMat[1,2]/(sqrt(covMat[1,1])*sqrt(covMat[2,2]))
```

```
## [1] -0.8051269
```

```
covMat[1,3]/(sqrt(covMat[1,1])*sqrt(covMat[3,3]))
```

```
## [1] -0.7784268
```

```
covMat[1,4]/(sqrt(covMat[1,1])*sqrt(covMat[4,4]))
```

```
## [1] -0.8322442
```

```
cor(Auto[,quant])
```

```
##                    mpg displacement horsepower     weight acceleration
## mpg          1.0000000   -0.8051269 -0.7784268 -0.8322442    0.4233285
```

```
## displacement -0.8051269    1.0000000  0.8972570  0.9329944   -0.5438005
## horsepower   -0.7784268    0.8972570  1.0000000  0.8645377   -0.6891955
## weight       -0.8322442    0.9329944  0.8645377  1.0000000   -0.4168392
## acceleration  0.4233285   -0.5438005 -0.6891955 -0.4168392    1.0000000
## year          0.5805410   -0.3698552 -0.4163615 -0.3091199    0.2903161
##                    year
## mpg           0.5805410
## displacement -0.3698552
## horsepower   -0.4163615
## weight       -0.3091199
## acceleration  0.2903161
## year          1.0000000
```

We see that the obtained correlations coincide with the given elements in the correlation matrix.

Problem 4

a) Simulate values from the four multivariate distributions using `mvnorm()`.

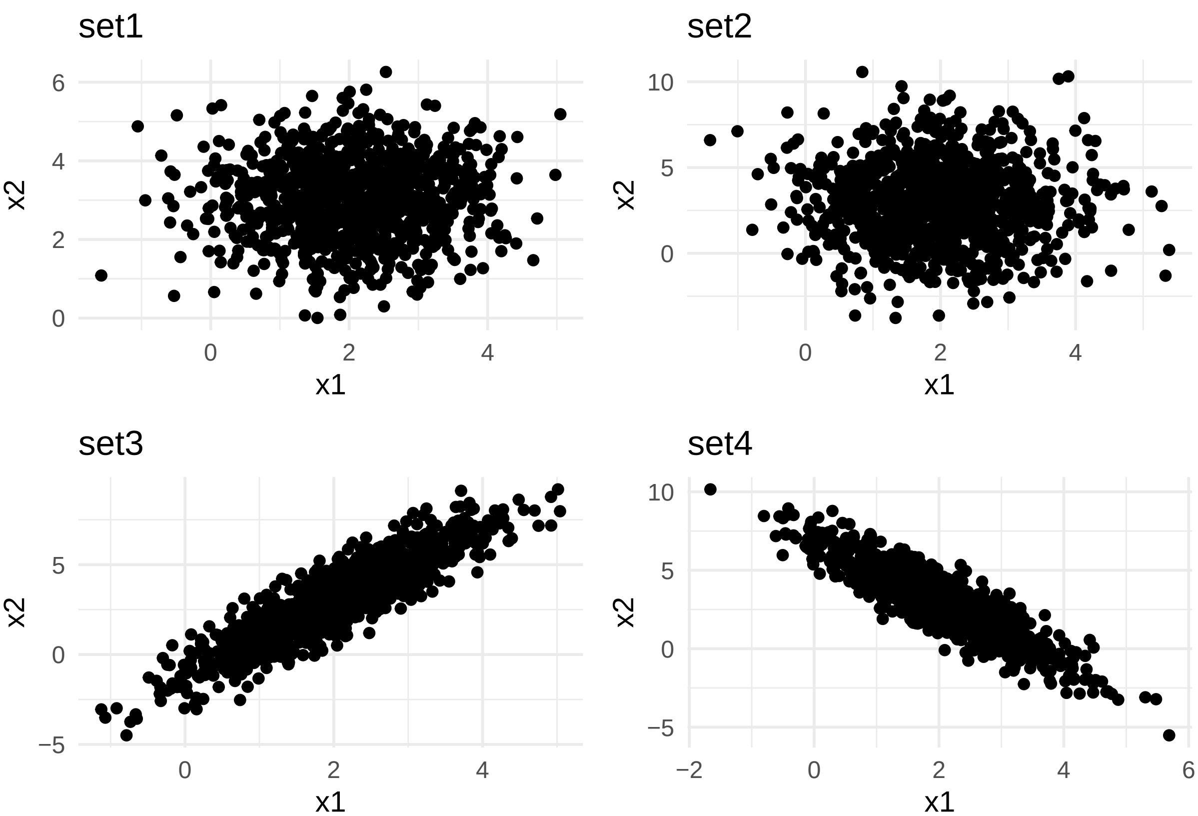
```r
# simulate 1000 values from the multivariate normal distribution with mean = c(2,3) and cov(1,0,0,1)
library(MASS)
set1 = as.data.frame(mvrnorm(n = 1000, mu=c(2,3), Sigma = matrix(c(1,0,0,1), ncol=2)))
colnames(set1) = c("x1", "x2")

set2 = as.data.frame(mvrnorm(n = 1000, mu=c(2,3), Sigma = matrix(c(1,0,0,5), ncol=2)))
colnames(set2) = c("x1", "x2")

set3 = as.data.frame(mvrnorm(n = 1000, mu=c(2,3), Sigma = matrix(c(1,2,2,5), ncol=2)))
colnames(set3) = c("x1", "x2")

set4 = as.data.frame(mvrnorm(n = 1000, mu=c(2,3), Sigma = matrix(c(1,-2,-2,5), ncol=2)))
colnames(set4) = c("x1", "x2")
```

b) Plot the simulated distribtions

```r
#install.packages("patchwork")
library(patchwork)
```

```
##
## Attaching package: 'patchwork'

## The following object is masked from 'package:MASS':
##
##     area
```

```r
p1 = ggplot(set1, aes(x1,x2)) + geom_point() + labs(title = "set1") + theme_minimal()
p2 = ggplot(set2, aes(x1,x2)) + geom_point() + labs(title = "set2") + theme_minimal()
p3 = ggplot(set3, aes(x1,x2)) + geom_point() + labs(title = "set3") + theme_minimal()
p4 = ggplot(set4, aes(x1,x2)) + geom_point() + labs(title = "set4") + theme_minimal()
(p1 + p2) / (p3 + p4)
```

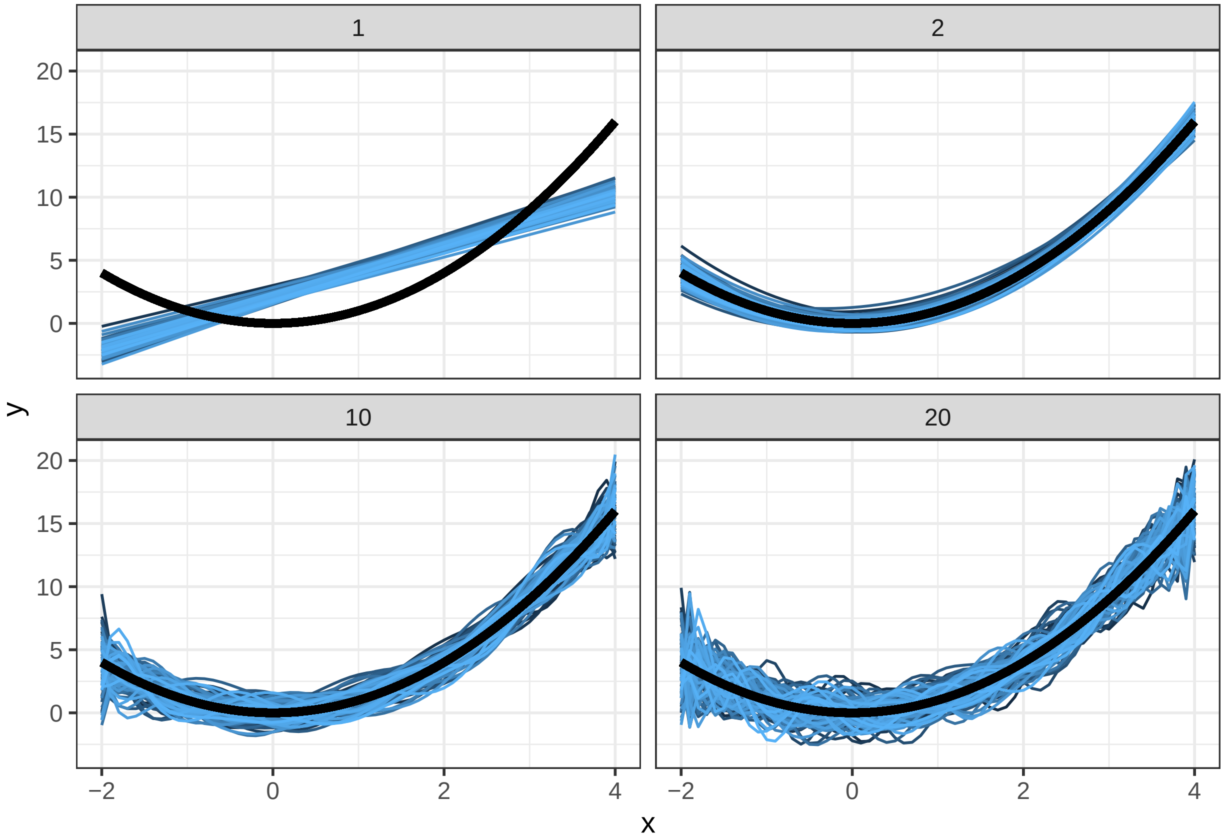


Problem 5

a) We sample from the model $y = x^2 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 2^2)$ and $x \in \{-2, -1.9, -1.8, ..., 3.8, 3.9, 4\}$. This means that $y \sim \mathcal{N}(x^2, 2^2)$. A total of 100 samples from this model are generated for each of the 61 x's. See comments in code for further explanations.

```
set.seed(2) # to reproduce

M <- 100 # repeated samplings, x fixed
nord <- 20 # order of polynomials

#------

x <- seq(from = -2, to = 4, by = 0.1) #We make a sequence of 61 points, x. These are the points for whi

truefunc <- function(x) {
  return(x^2) #The true f(x)=x^2.
}
true_y <- truefunc(x) #We find f(x) for each element in vector x.
error <- matrix(rnorm(length(x) * M, mean = 0, sd = 2),
                nrow = M,
                byrow = TRUE) #Noise (epsilon) is sampled from a normal distribution and stored in this
ymat <- matrix(rep(true_y, M), byrow = T, nrow = M) + error  #The 100 samples or the observations are s

#------
```

```
predictions_list <- lapply(1:nord, matrix, data = NA, nrow = M, ncol = ncol(ymat))
for(i in 1:nord){
  for(j in 1:M){
    predictions_list[[i]][j, ] <- predict(lm(ymat[j,] ~ poly(x, i, raw = TRUE))) #Based on the response
  }
}

# Plotting -----

library(tidyverse) # The tidyverse contains ggplot2, as well as tidyr and dplyr,
# which we can use for dataframe manipulation.

list_of_matrices_with_deg_id <- lapply(1:nord, function(poly_degree) cbind(predictions_list[[poly_degre
# Now predictions_list is a list with 20 entries, where each entry is a matrix
# with 100 rows, where each row is the predicted polynomial of that degree.
# We also have a column for the simulation number, and a column for polynomial degree.

# Extract each matrix and bind them to one large matrix
stacked_matrices <-  NULL
for (i in 1:nord) {
  stacked_matrices <-
    rbind(stacked_matrices, list_of_matrices_with_deg_id[[i]])
}
stacked_matrices_df <- as.data.frame(stacked_matrices)

# Convert from wide to long (because that is the best format for ggplot2)
long_predictions_df <- pivot_longer(stacked_matrices_df, !c(simulation_num, poly_degree), values_to = "

# Now we can use ggplot2!
# We just want to plot for degrees 1, 2, 10 and 20.

plotting_df <- cbind(long_predictions_df, x = x) %>% # adding the x-vector to the dataframe
  filter(poly_degree %in% c(1, 2, 10, 20)) # Select only the predictions using degree 1, 2, 10 or 20

ggplot(plotting_df, aes(x = x, y = y, group = simulation_num)) +
  geom_line(aes(color = simulation_num)) +
  geom_line(aes(x = x, y = x^2), size = 1.5) +
  facet_wrap(~ poly_degree) +
  theme_bw() +
  theme(legend.position = "none")
```



The upper left plot shows 100 predictions when we assume that y is a linear function of x, the upper right plot hows 100 predictions when we assume that y is function of poynomials up to x^2, the lower left plot shows 100 predictions when we assume y is a function of polynomials up to x^{10} and the lower right plot shows 100 predictions when assuming y is a function of polynomials up to x^{20}.

b) Run the code attached and consider the following plots:

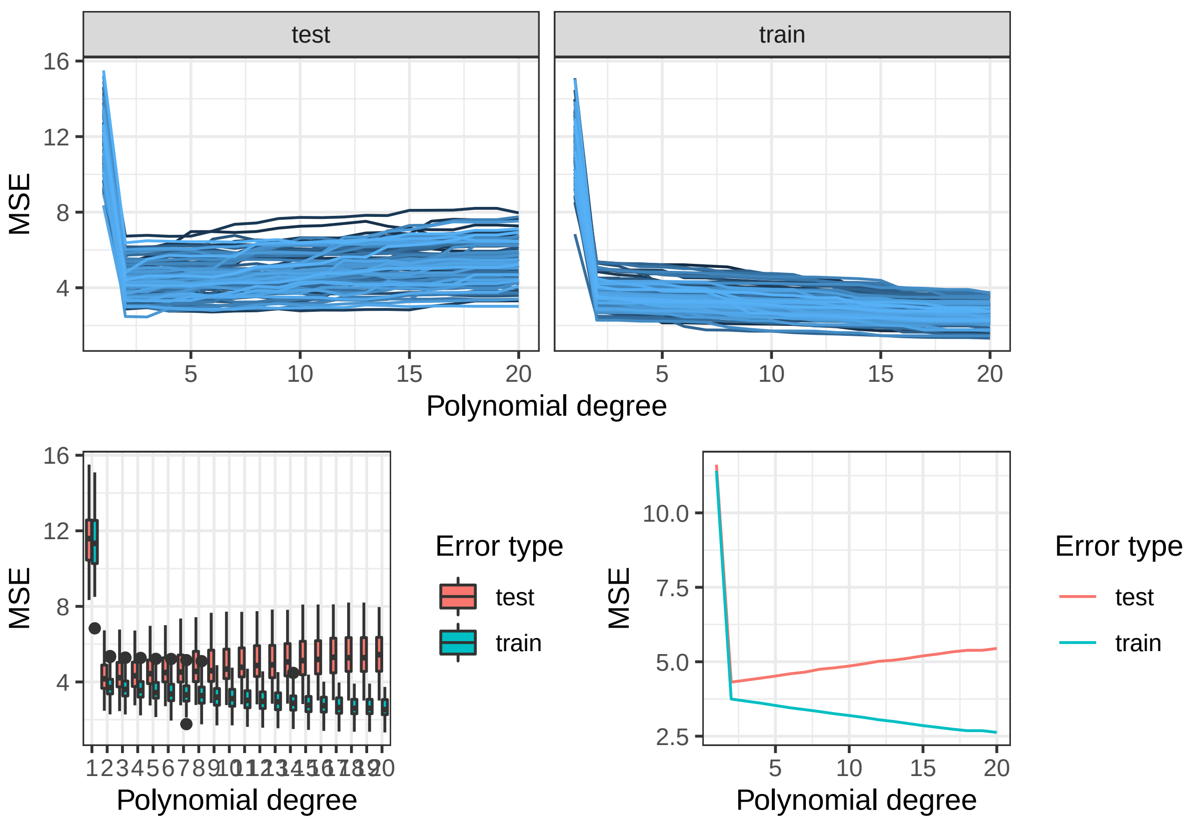
```
set.seed(2) # to reproduce
M <- 100 # repeated samplings,x fixed but new errors
nord <- 20

x <- seq(from = -2, to = 4, by = 0.1)
truefunc <- function(x){
  return(x^2)
}
true_y <- truefunc(x)
error <- matrix(rnorm(length(x)*M, mean = 0, sd = 2), nrow = M, byrow = TRUE)
testerror <- matrix(rnorm(length(x)*M, mean = 0, sd = 2), nrow = M, byrow = TRUE)
ymat <- matrix(rep(true_y, M), byrow = T, nrow = M) + error
testymat <- matrix(rep(true_y, M), byrow=T, nrow=M) + testerror

predictions_list <- lapply(1:nord, matrix, data = NA, nrow = M, ncol = ncol(ymat))
for(i in 1:nord){
  for(j in 1:M){
    predictions_list[[i]][j, ] <- predict(lm(ymat[j,] ~ poly(x, i, raw = TRUE)))
  }
}
```

```
trainMSE <- lapply(1:nord, function(poly_degree) rowMeans((predictions_list[[poly_degree]] - ymat)^2))
testMSE <- lapply(1:nord, function(poly_degree) rowMeans((predictions_list[[poly_degree]] - testymat)^2

# Plotting -----
library(tidyverse) # The tidyverse contains ggplot2, as well as tidyr and dplyr,
# which we can use for dataframe manipulation.

# Convert each matrix in the list form wide to long (because that is the best format for ggplot2)
list_train_MSE <- lapply(1:nord, function(poly_degree) cbind(error = trainMSE[[poly_degree]],
                                                              poly_degree,
                                                              error_type = "train",
                                                              simulation_num = 1:M))
list_test_MSE <- lapply(1:nord, function(poly_degree) cbind(error = testMSE[[poly_degree]],
                                                             poly_degree,
                                                             error_type = "test",
                                                             simulation_num = 1:M))

# Now predictions_list is a list with 20 entries, where each entry is a matrix
# with 100 rows, where each row is the predicted polynomial of that degree.

stacked_train <- NULL
for (i in 1:nord) {
  stacked_train <-
    rbind(stacked_train, list_train_MSE[[i]])
}
stacked_test <- NULL
for (i in 1:nord) {
  stacked_test <-
    rbind(stacked_test, list_test_MSE[[i]])
}

stacked_errors_df <- as.data.frame(rbind(stacked_train, stacked_test))
# This is already on long format.
stacked_errors_df$error <- as.numeric(stacked_errors_df$error)
stacked_errors_df$simulation_num <- as.integer(stacked_errors_df$simulation_num)
stacked_errors_df$poly_degree <- as.integer(stacked_errors_df$poly_degree)

p.all_lines <- ggplot(data = stacked_errors_df, aes(x = poly_degree, y = error, group = simulation_num)
  geom_line(aes(color = simulation_num)) +
  facet_wrap(~ error_type) +
  xlab("Polynomial degree") +
  ylab("MSE") +
  theme_bw() +
  theme(legend.position = "none")

p.bars <- ggplot(stacked_errors_df, aes(x = as.factor(poly_degree), y = error)) +
  geom_boxplot(aes(fill = error_type)) +
  scale_fill_discrete(name = "Error type") +
  xlab("Polynomial degree") +
  ylab("MSE") +
  theme_bw()

# Here we find the average test error and training error across the repeated simulations.
```

```r
# The symbol "%>%" is called a pipe, and comes from the tidyverse packages,
# which provide convenient functions for working with data frames.
means_across_simulations <- stacked_errors_df %>%
  group_by(error_type, poly_degree) %>%
  summarise(mean = mean(error))
```

```
## `summarise()` has grouped output by 'error_type'. You can override using the `.groups` argument.
```

```r
p.means <- ggplot(means_across_simulations, aes(x = poly_degree, y = mean)) +
  geom_line(aes(color = error_type)) +
  scale_color_discrete(name = "Error type") +
  xlab("Polynomial degree") +
  ylab("MSE") +
  theme_bw()

library(patchwork) # The library patchwork is the best way of combining ggplot2 objects.
# You could also use the function ggarrange from the ggpubr package.

p.all_lines / (p.bars + p.means)
```

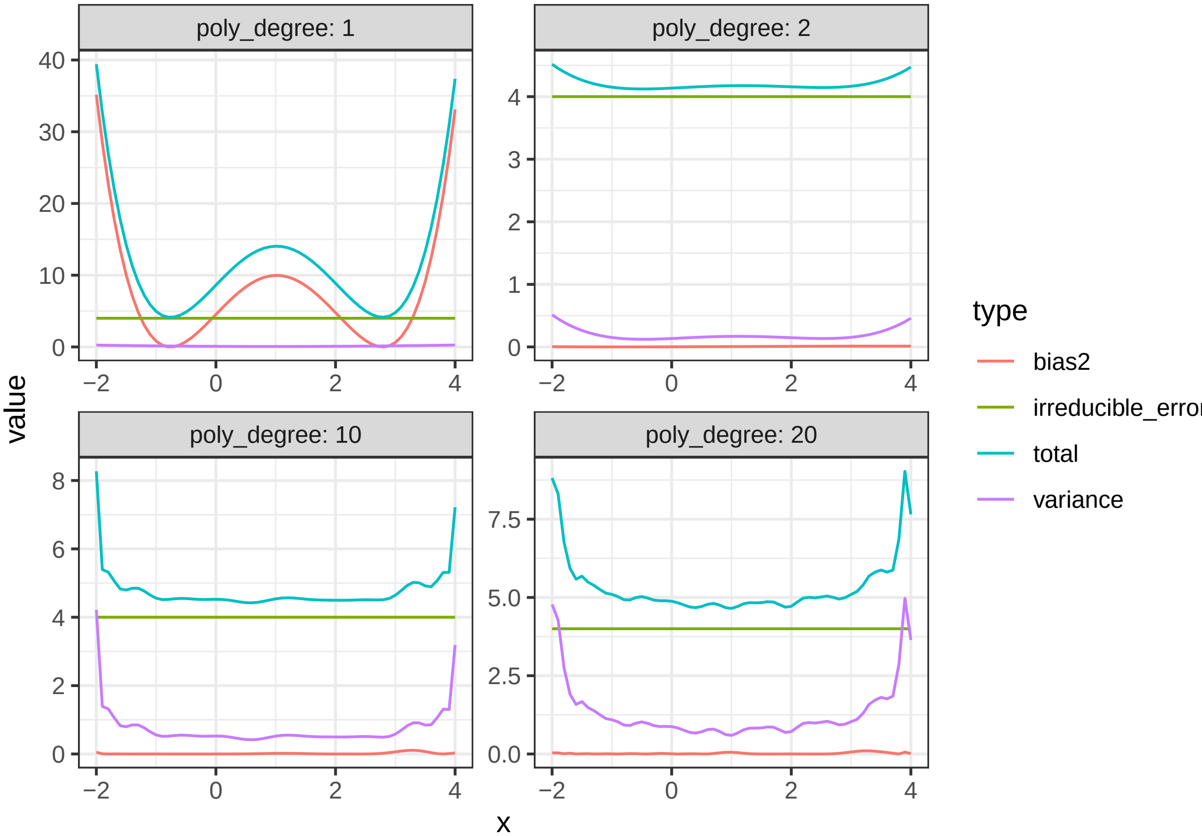


The plots show that the test MSE in general is larger than the train MSE. This is reasonable. The fitted model is fitted based on the training set. Thus, the error will be smaller for the train data than for the test data. Furthermore, the plots show that the difference between the MSE for the test set and the training set increases when the degree of the polynomial increases. When the degree of the polynomial increases, we get a more flexible model. The fitted curve will try to pass through the training data if possible, which typically leads to an overfitted model that performs bad for test data.

- We observe that poly 2 gives the smallest mean testMSE, while poly 20 gives the smallest trainMSE. Based on these plots, we would choose poly 2 for prediction of a new value of y, as the testMSE tells us more about how the model performs on data not used to train the model.

c) Run the code and consider the following plots:

```
meanmat <- matrix(ncol = length(x), nrow = nord)
varmat <- matrix(ncol = length(x), nrow = nord)
for (j in 1:nord){
  meanmat[j,] <- apply(predictions_list[[j]], 2, mean) # we now take the mean over the M simulations -
  varmat[j,] <- apply(predictions_list[[j]], 2, var)
}

# nord times length(x)
bias2mat <- (meanmat - matrix(rep(true_y, nord), byrow = TRUE, nrow = nord))^2 #here the truth is final

df <- data.frame(x = rep(x, each = nord), poly_degree = rep(1:nord, length(x)),
                 bias2 = c(bias2mat), variance = c(varmat),
                 irreducible_error = rep(4, prod(dim(varmat)))) #irr is just 1

df$total <- df$bias2 + df$variance + df$irreducible_error

df_long <- pivot_longer(df, cols = !c(x, poly_degree), names_to = "type")

df_select_poly <- filter(df_long, poly_degree %in% c(1, 2, 10, 20))

ggplot(df_select_poly, aes(x = x, y = value, group = type)) +
  geom_line(aes(color = type)) +
  facet_wrap(~poly_degree, scales = "free", labeller = label_both) +
  theme_bw()
```

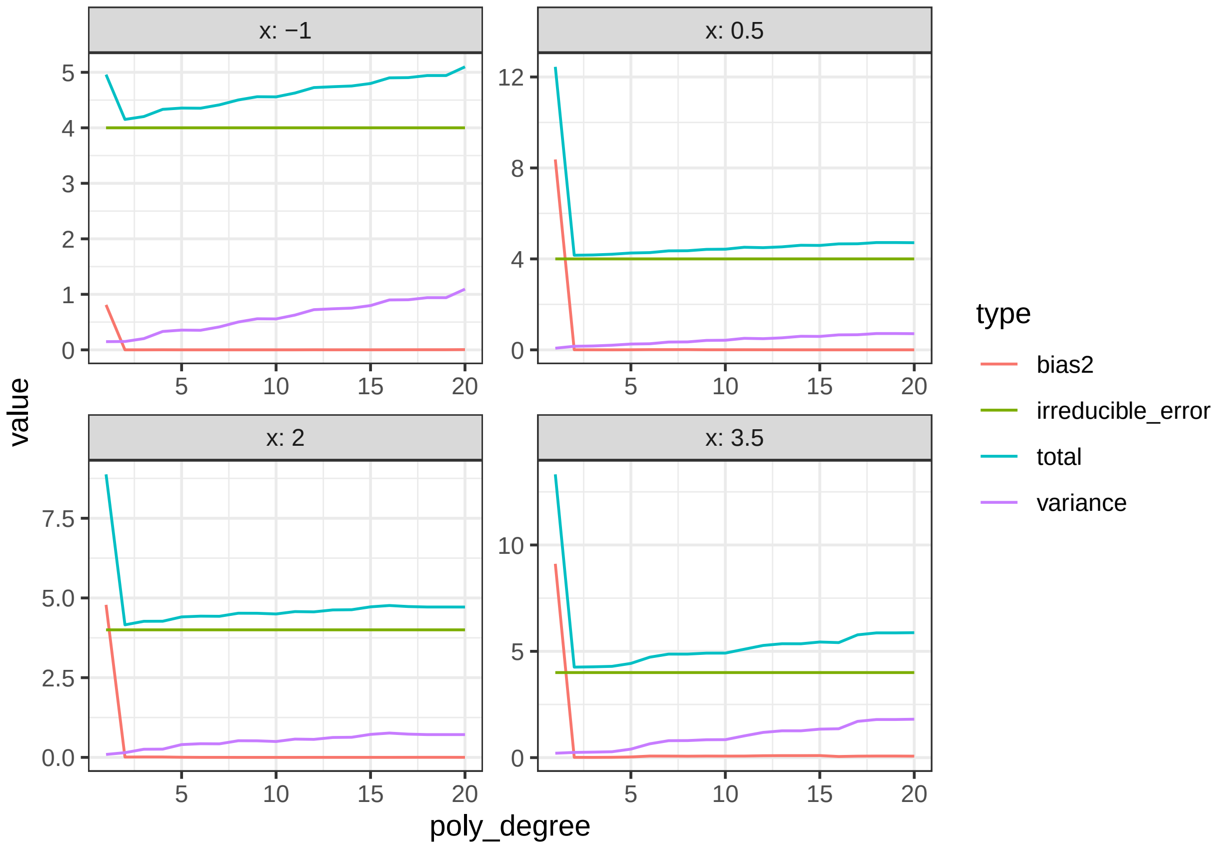


We see that the irriducible error remains constant with the complexity of the model and the variance (green) increases with the complexity. A linear model gives variance close to zero, while a polynomial of degree 20 gives variance close to 1 (larger at the borders). A more complex model is more flexible as it can turn up and down and change direction fast. This leads to larger variance. (Look at the plot in 2a, there is a larger variety of curves you can make when the degree is 20 compared to if the degree is 1.).

Further, we see that the bias is large for poly1, the linear model. The linear model is quite rigid, so if the true underlying model is non-linear, we typically get large deviations between the fitted line and the training data. If we study the first plot, it seems like the fitted line goes through the training data in $x = -1$ and $x = 3$ as the bias is close to zero here (this is confirmed by looking at the upper left plot in 2a).

The polynomial models with degree larger than one lead to lower bias. Recall that this is the training bias: The polynomial models will try to pass through the training points if possible, and when the degree of the polynomial is large they are able to do so because they have large flexibility compared to a linear model.

```r
df_select_x <- filter(df_long, x %in% c(-1, 0.5, 2, 3.5))

ggplot(df_select_x, aes(x = poly_degree, y = value, group = type)) +
  geom_line(aes(color = type)) +
  facet_wrap(~x, scales = "free", labeller = label_both) +
  theme_bw()
```



Compare to Figures in 2.12 on page 36 in ISL (our textbook).

d) To change $f(x)$, replace

```
truefunc=function(x) return(x^2)
```

by for example

```
truefunc=function(x) return(x^4)
```

or

```
truefunc=function(x) return(exp(2*x))
```

and rerun the code. Study the results.

If you want to set the variance to 1 for example, set $sd = 1$ in these parts of the code in 2a and 2b:

```
rnorm(length(x)*M, mean=0, sd=1)
```

Also change the following part in 2c:

```
df <- data.frame(x = rep(x, each = nord), poly_degree = rep(1:nord, length(x)),
                 bias2 = c(bias2mat), variance = c(varmat),
                 irreducible_error = rep(4, prod(dim(varmat)))) #irr is just 1
```

to get correct plots of the irreducible error. Here, $rep(4, prod(dim(varmat)))$ is replaced by $rep(1, prod(dim(varmat)))$.

R packages

If you want to look at the .Rmd file and `knit` it, you need to first install the following packages (only once).

```
install.packages("ggplot2")
install.packages("gamlss.data")
install.packages("tidyverse")
install.packages("GGally")
install.packages("Matrix")
install.packages("patchwork")
```