Module 2: Recommended Exercises - Solution

TMA4268 Statistical Learning V2020

Martina Hall, Michail Spitieris, Stefanie Muff, Department of Mathematical Sciences, NTNU

January 17, 2020

Last changes: (10.01.2020: first version)

Problem 1 - 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 specie. Response: specie. Predictors: color, height, leafes ect. Goal: prediction

Problem 2 - 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 3

a) A rigid method will typically have the highest test error.
b) A small (test) variance imply 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 4

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

a)

```r
#a)
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 
```

```
summary(Auto)
```

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

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

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

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

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

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

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

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

```
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 
##     23.44592    194.41199    104.46939   2977.58418     15.54133 
##         year 
##     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 
##     23.44592    194.41199    104.46939   2977.58418     15.54133 
##         year 
##     75.97959
```

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

```
##          mpg displacement   horsepower       weight acceleration
##     7.805007   104.644004    38.491160   849.402560     2.758864
##         year
##     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
##     24.40443    187.24051    100.72152   2935.97152     15.72690
##         year
##     77.14557
```

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

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

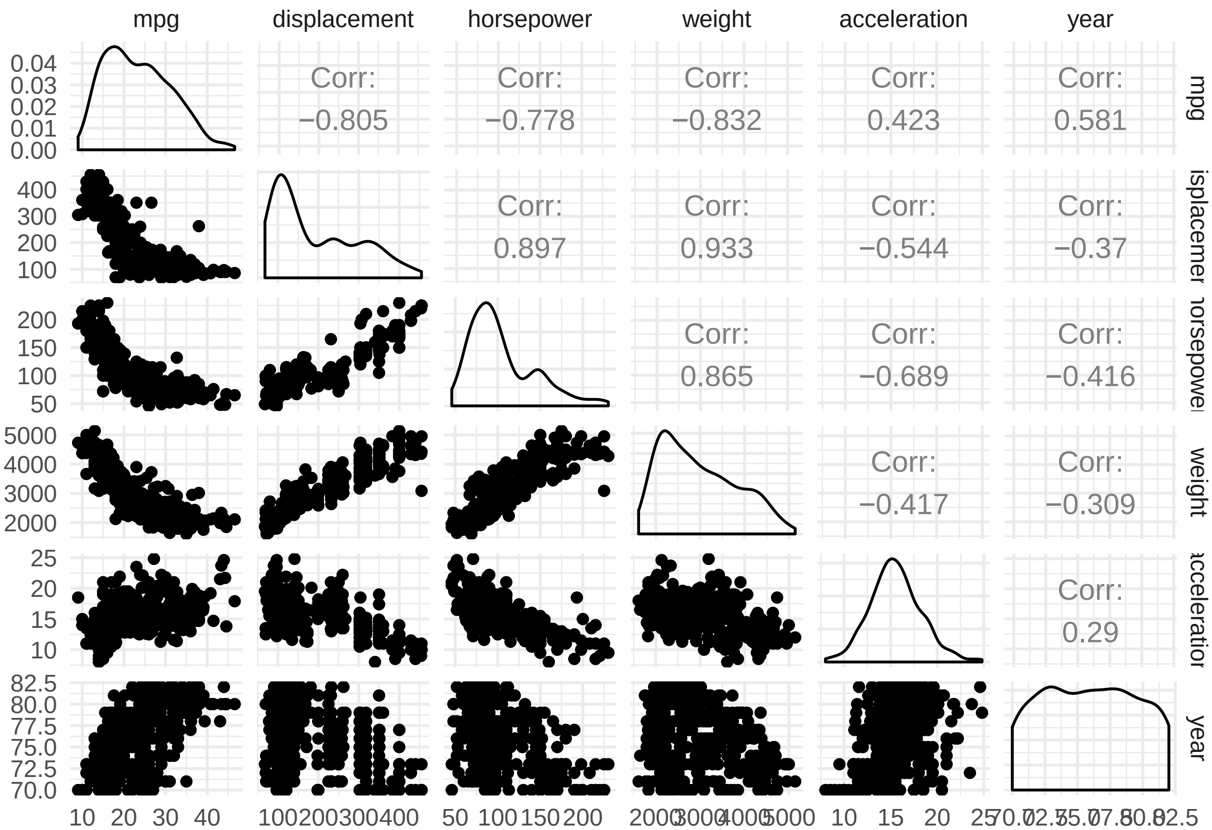
e) Make a scatterplot of the full dataset using the `ggpairs()` function.

```r
library(GGally)
```

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

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

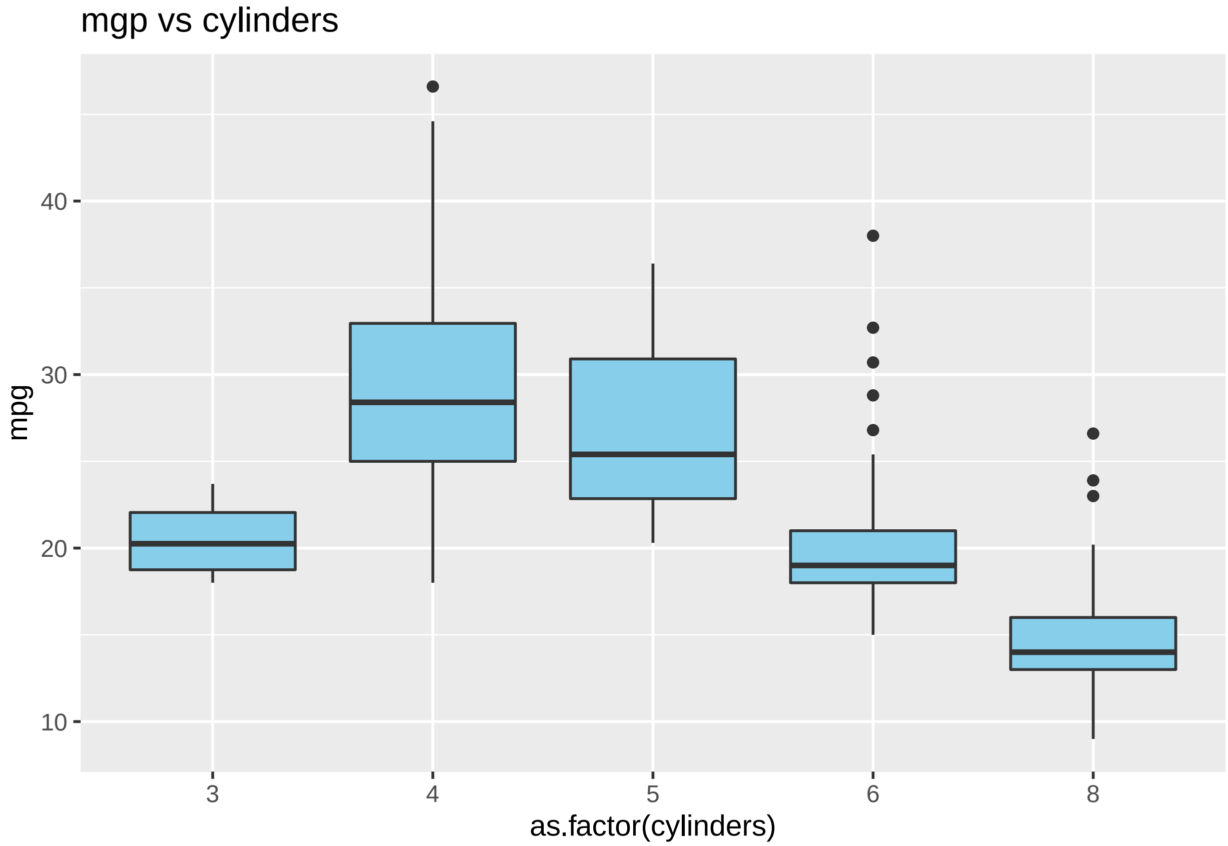
```r
ggpairs(Auto[,quant]) + theme_minimal()
```



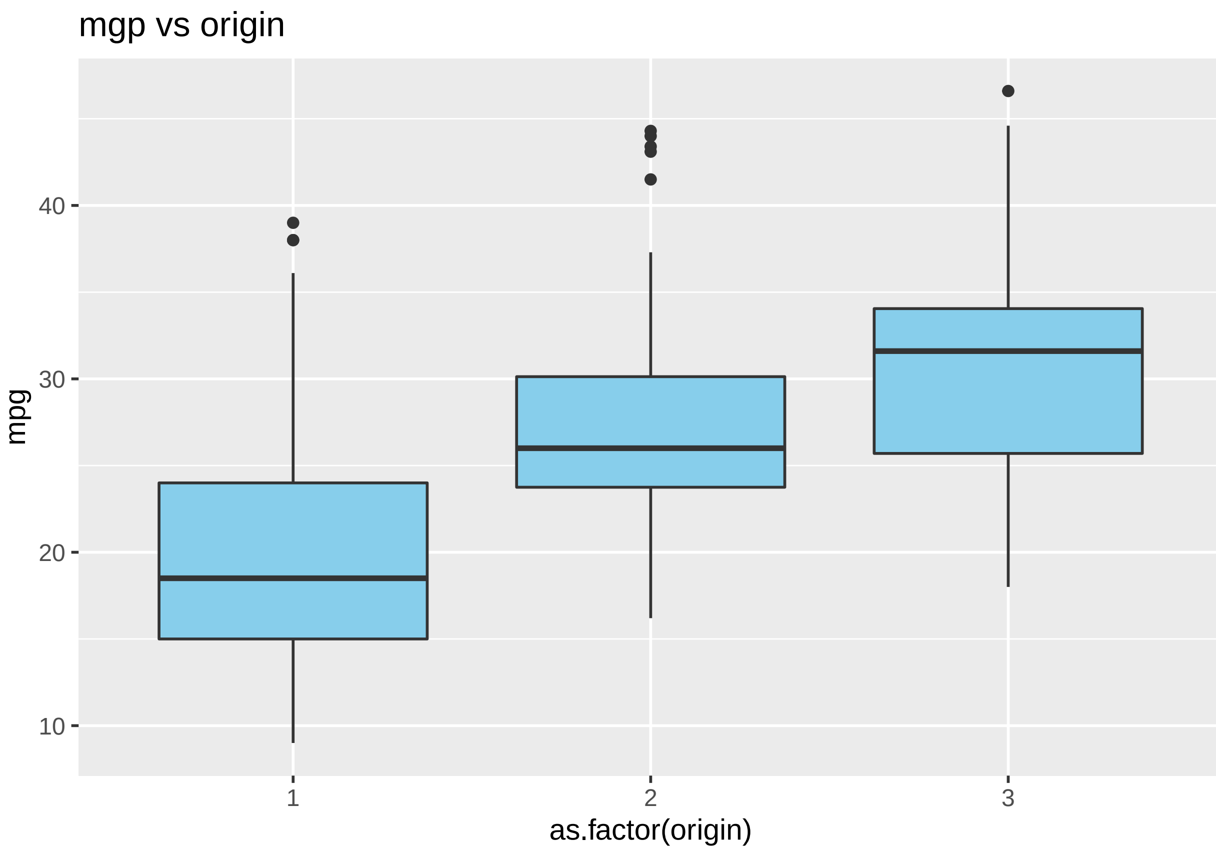
We see that there seems to be strong relationships (based on curve trends and correlation) between the pairs: `mpg` and `displacement`, `mpg` and `horsepower`, `mpg` and `weight`, `displacement` and `horsepower`, `displacement` and `weight`, `horsepower` and `weght`, and `horsepower` and `acceleration`.

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`.

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




```
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.

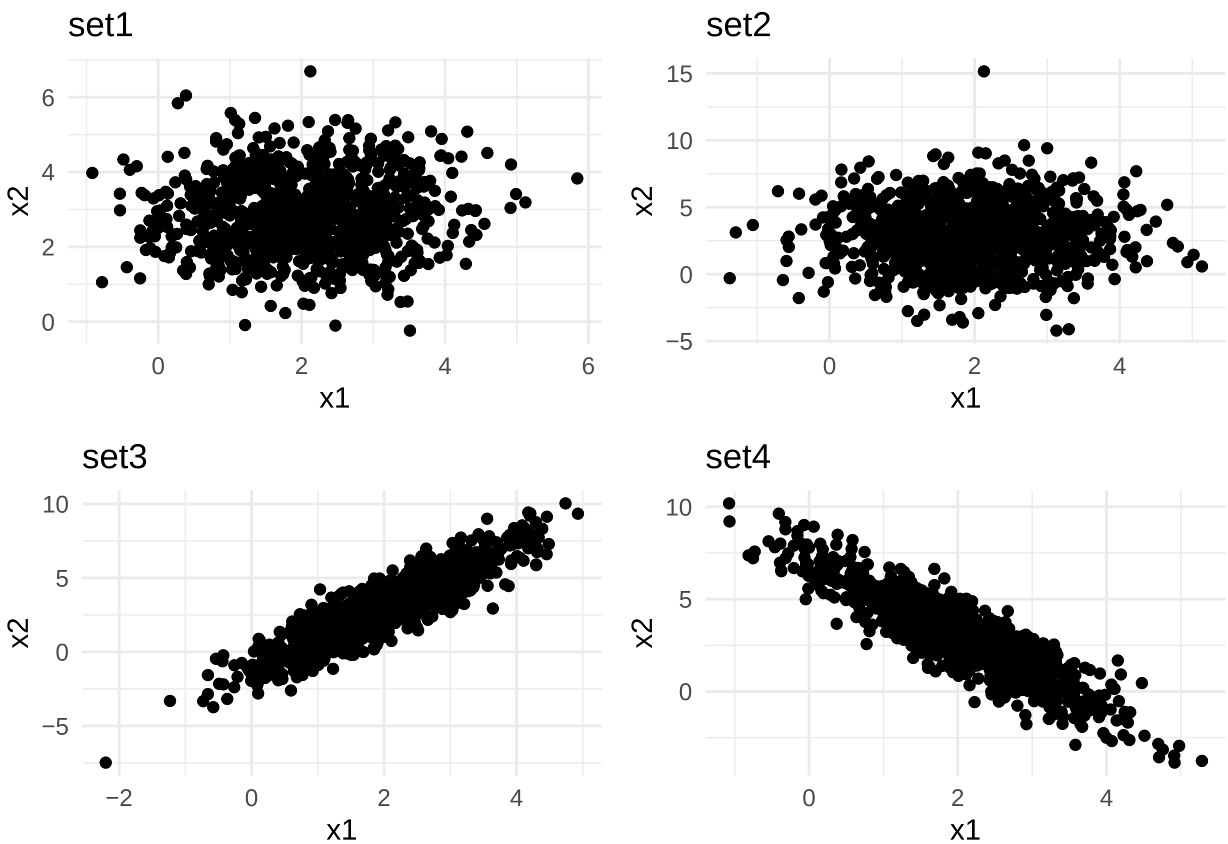
Problem 5

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("gridExtra")
library(gridExtra)
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()
grid.arrange(p1,p2,p3,p4, ncol=2)
```



Problem 6

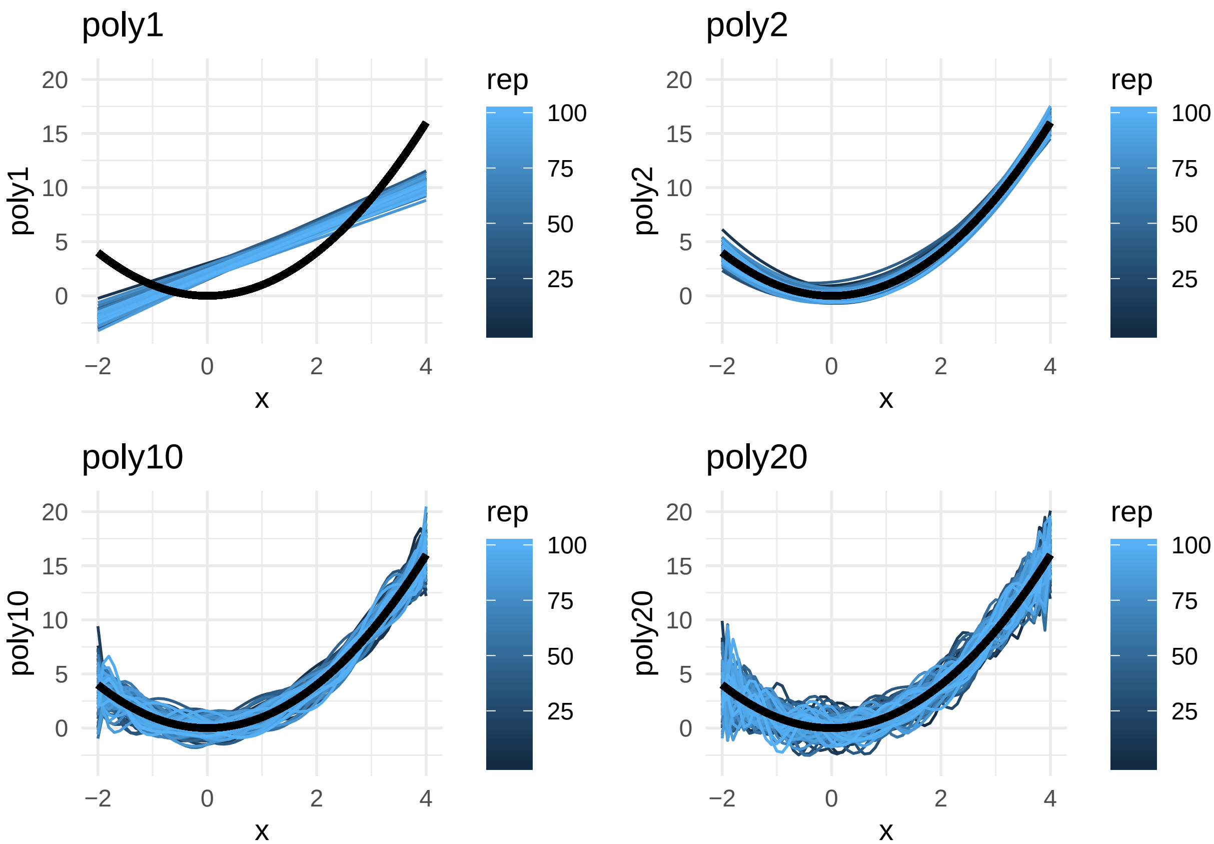
a)

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.

```
library(ggplot2)
library(ggpubr)
set.seed(2) # to reproduce

M=100 # repeated samplings, x fixed
nord=20 # order of polynoms


x = seq(-2, 4, 0.1) #We make a sequence of 61 points, x. These are the points for which we evaluate the
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 n
ymat = matrix(rep(true_y,M),byrow=T,nrow=M) + error #The 100 samples or the observations are stored in
```

```r
predarray=array(NA,dim=c(M,length(x),nord))
for (i in 1:M){
  for (j in 1:nord){
    predarray[i,,j]=predict(lm(ymat[i,]~poly(x, j,raw=TRUE)))
    #Based on the response y_i and the x_i's, we fit a polynomial model of degre 1,...,20. This means t
  }
}
# M matrices of size length(x) times nord
# first, only look at variablity in the M fits and plot M curves where we had 1.

# for plotting need to stack the matrices underneath eachother and make new variable "rep"
stackmat=NULL
for (i in 1:M) stackmat=rbind(stackmat,cbind(x,rep(i,length(x)),predarray[i,,]))
#dim(stackmat)
colnames(stackmat)=c("x","rep",paste("poly",1:20,sep=""))
sdf=as.data.frame(stackmat) #NB have poly1-20 now - but first only use 1,2,20
# to add true curve using stat_function - easiest solution
true_x=x
yrange=range(apply(sdf,2,range)[,3:22])
p1=ggplot(data=sdf,aes(x=x,y=poly1,group=rep,colour=rep))+scale_y_continuous(limits=yrange)+geom_line()
p1=p1+stat_function(fun=truefunc,lwd=1.3,colour="black")+ggtitle("poly1")+theme_minimal()
p2=ggplot(data=sdf,aes(x=x,y=poly2,group=rep,colour=rep))+scale_y_continuous(limits=yrange)+geom_line()
p2=p2+stat_function(fun=truefunc,lwd=1.3,colour="black")+ggtitle("poly2")+theme_minimal()
p10=ggplot(data=sdf,aes(x=x,y=poly10,group=rep,colour=rep))+scale_y_continuous(limits=yrange)+geom_line
p10=p10+stat_function(fun=truefunc,lwd=1.3,colour="black")+ggtitle("poly10")+theme_minimal()
p20=ggplot(data=sdf,aes(x=x,y=poly20,group=rep,colour=rep))+scale_y_continuous(limits=yrange)+geom_line
p20=p20+stat_function(fun=truefunc,lwd=1.3,colour="black")+ggtitle("poly20")+theme_minimal()
ggarrange(p1,p2,p10,p20)
```



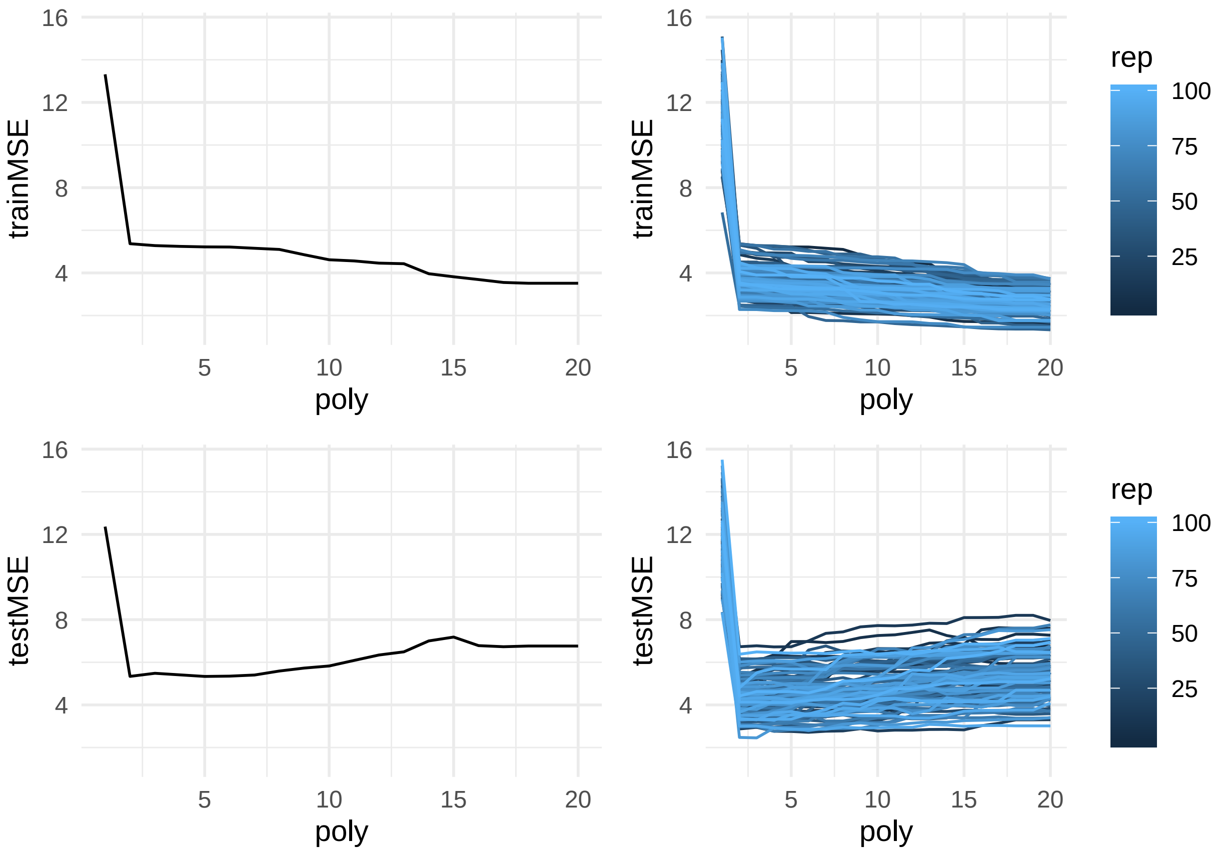
The upper left plot shows 100 predictions when we assume that y is a linear function of x, the upper right plot hows 100 predictions when we assume that y is function of poynomials up to x^2, the lower left plot shows 100 predictions when we assume y is a function of polynomials up to x^{10} and the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

b)

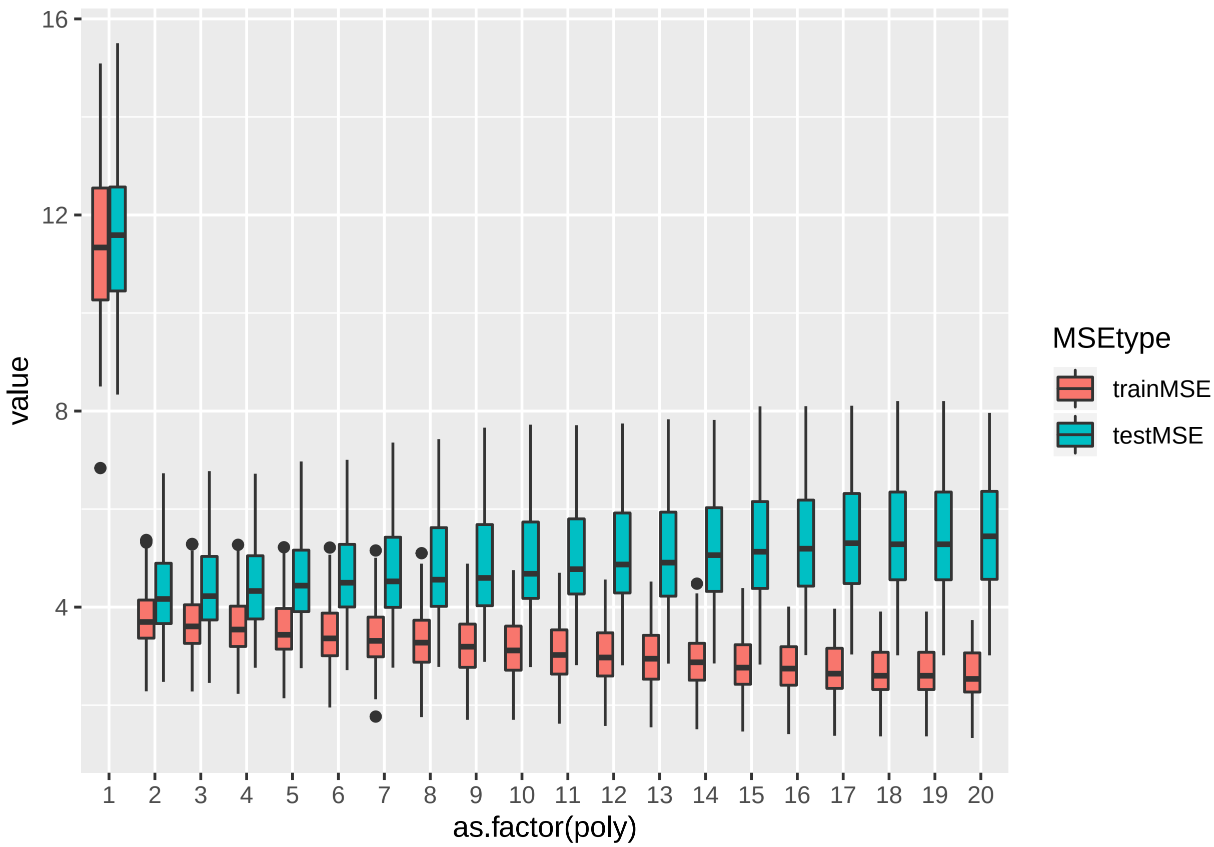
Run the code attached and consider the following plots:

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

M=100 # repeated samplings,x fixed but new errors
nord=20
x = seq(-2, 4, 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

predarray=array(NA,dim=c(M,length(x),nord))
for (i in 1:M)
{
  for (j in 1:nord)
  {
```

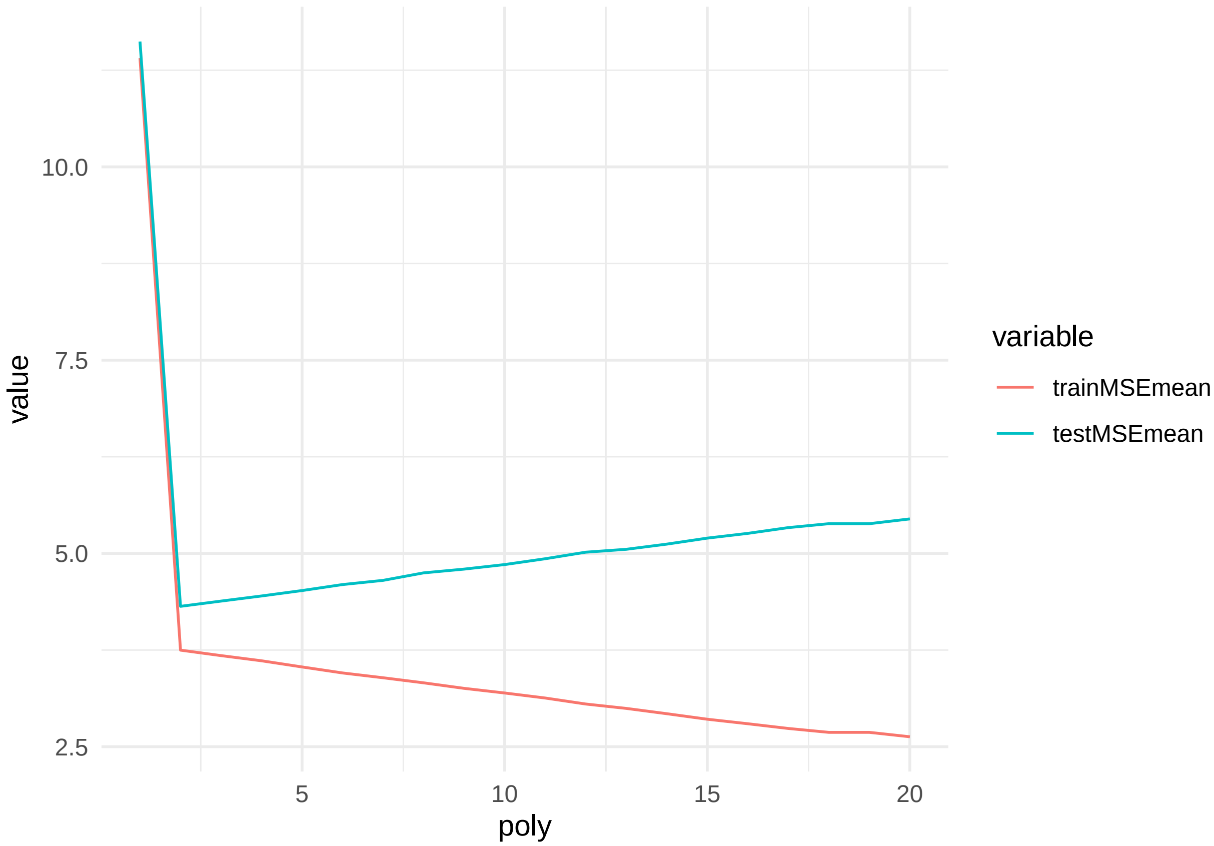
```
    predarray[i,,j]=predict(lm(ymat[i,]~poly(x, j,raw=TRUE)))
  }
}
trainMSE=matrix(ncol=nord,nrow=M)
for (i in 1:M) trainMSE[i,]=apply((predarray[i,,]-ymat[i,])^2,2,mean)
testMSE=matrix(ncol=nord,nrow=M)
for (i in 1:M) testMSE[i,]=apply((predarray[i,,]-testymat[i,])^2,2,mean)

library(ggplot2)
library(ggpubr)

# format suitable for plotting
stackmat=NULL
for (i in 1:M) stackmat=rbind(stackmat,cbind(rep(i,nord),1:nord,trainMSE[i,],testMSE[i,]))
colnames(stackmat)=c("rep","poly","trainMSE","testMSE")
sdf=as.data.frame(stackmat)
yrange=range(sdf[,3:4])
p1=ggplot(data=sdf[1:nord,],aes(x=poly,y=trainMSE))+scale_y_continuous(limits=yrange)+geom_line()+theme_
pall= ggplot(data=sdf,aes(x=poly,group=rep,y=trainMSE,colour=rep))+scale_y_continuous(limits=yrange)+ge
testp1=ggplot(data=sdf[1:nord,],aes(x=poly,y=testMSE))+scale_y_continuous(limits=yrange)+geom_line()+th
testpall= ggplot(data=sdf,aes(x=poly,group=rep,y=testMSE,colour=rep))+scale_y_continuous(limits=yrange)
ggarrange(p1,pall,testp1,testpall)
```




```
library(reshape2)
df=melt(sdf,id=c("poly","rep"))[,-2]
colnames(df)[2]="MSEtype"
ggplot(data=df,aes(x=as.factor(poly),y=value))+geom_boxplot(aes(fill=MSEtype))
```




```
trainMSEmean=apply(trainMSE,2,mean)
testMSEmean=apply(testMSE,2,mean)
meandf=melt(data.frame(cbind("poly"=1:nord,trainMSEmean,testMSEmean)),id="poly")
ggplot(data=meandf,aes(x=poly,y=value,colour=variable))+geom_line()+theme_minimal()
```



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.

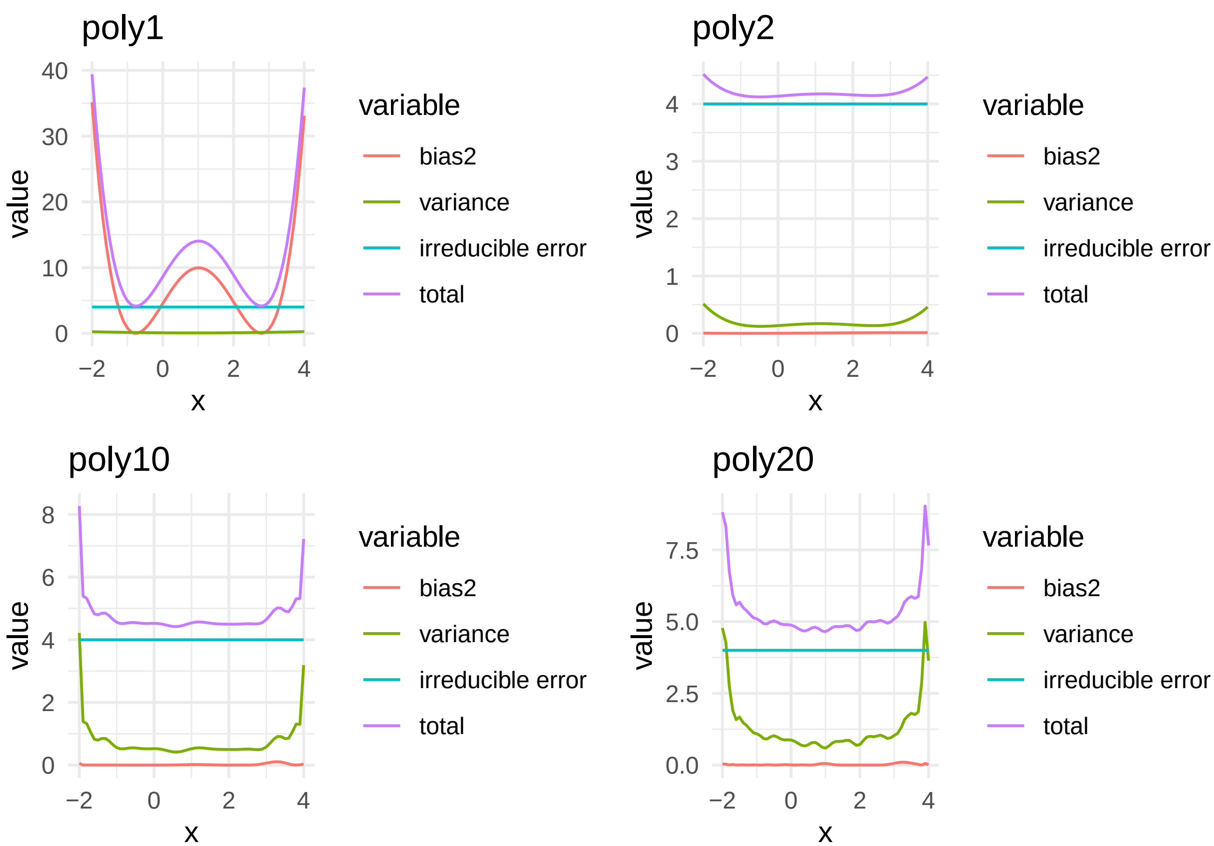
- We observe that poly 2 gives the smallest mean testMSE, while poly 20 gives the smallest trainMSE. Based on these plots, we would choose poly 2 for prediction of a new value of y, as the testMSE tells us more about how the model performs on data not used to train the model.

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(predarray[,,j],2,mean) # we now take the mean over the M simulations - to mimic E a
  varmat[j,]=apply(predarray[,,j],2,var)
}
# nord times length(x)
bias2mat=(meanmat-matrix(rep(true_y,nord),byrow=TRUE,nrow=nord))^2 #here the truth is

df=data.frame(rep(x,each=nord),rep(1:nord,length(x)),c(bias2mat),c(varmat),rep(4,prod(dim(varmat)))) #i
colnames(df)=c("x","poly","bias2","variance","irreducible error") #suitable for plotting
```

```
df$total=df$bias2+df$variance+df$`irreducible error`
hdf=melt(df,id=c("x","poly"))
hdf1=hdf[hdf$poly==1,]
hdf2=hdf[hdf$poly==2,]
hdf10=hdf[hdf$poly==10,]
hdf20=hdf[hdf$poly==20,]

p1=ggplot(data=hdf1,aes(x=x,y=value,colour=variable))+geom_line()+ggtitle("poly1")+theme_minimal()
p2=ggplot(data=hdf2,aes(x=x,y=value,colour=variable))+geom_line()+ggtitle("poly2")+theme_minimal()
p10=ggplot(data=hdf10,aes(x=x,y=value,colour=variable))+geom_line()+ggtitle("poly10")+theme_minimal()
p20=ggplot(data=hdf20,aes(x=x,y=value,colour=variable))+geom_line()+ggtitle("poly20")+theme_minimal()
ggarrange(p1,p2,p10,p20)
```

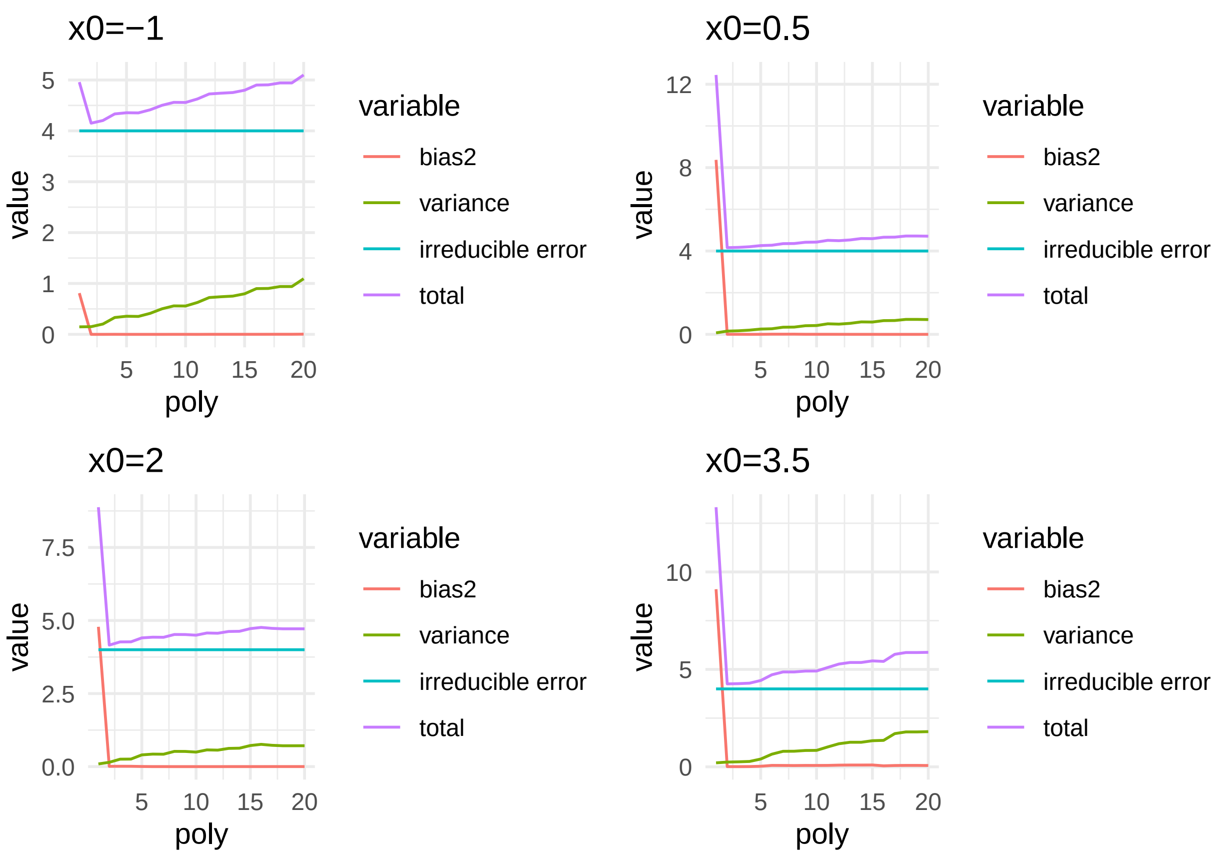


We see that the irridcible error remains constant with the complexity of the model and the variance (green) increases with the complexity. A linear model gives variance close to zero, while a polynomial of degree 20 gives variance close to 1 (larger at the borders). A more complex model is more flexible 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.

```
hdfatxa=hdf[hdf$x==-1,]
hdfatxb=hdf[hdf$x==0.5,]
hdfatxc=hdf[hdf$x==2,]
hdfatxd=hdf[hdf$x==3.5,]
pa=ggplot(data=hdfatxa,aes(x=poly,y=value,colour=variable))+geom_line()+ggtitle("x0=-1")+theme_minimal(
pb=ggplot(data=hdfatxb,aes(x=poly,y=value,colour=variable))+geom_line()+ggtitle("x0=0.5")+theme_minimal
pc=ggplot(data=hdfatxc,aes(x=poly,y=value,colour=variable))+geom_line()+ggtitle("x0=2")+theme_minimal()
pd=ggplot(data=hdfatxd,aes(x=poly,y=value,colour=variable))+geom_line()+ggtitle("x0=3.5")+theme_minimal
ggarrange(pa,pb,pc,pd)
```



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(rep(x,each=nord),rep(1:nord,length(x)),c(bias2mat),c(varmat),rep(1,prod(dim(varmat)))) #i
```

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("ggpubr")
```