Module 2: Recommended Exercises

alexaoh

17.01.2021

Problem 1

a)

Describe a real-life application in which classification might be useful. Identify the response and the predictors. Is the goal inference or prediction?

A real-life application of classification might be, for example, in deciding whether a special type of diet might lead to heart disease. In this case, the predictors would be the different types of diets and the response would be whether or not each person in the study develops cardiovascular disease or not. The goal might be both prediction and inference: One might want to predict whether a person has high risk of heart disease based on his/her diet, or one might want to learn which types of diets are most dangerous in the cardiovascular sense.

b)

Describe a real-life application in which regression might be useful. Identify the response and the predictors. Is the goal inference or prediction?

A real-life application of regression might be, for example, to predict how the standings in a football league will be at the end of the season. In this case, the predictors would be standings in the previous seasons, historical trends, stats of newly transferred players in each team, earlier results between each teams or others. The response would be the placements of each team in the league. The goal of this application is prediction.

Problem 2

Take a look at Figure 2.9 in the course book (p.31).

a)

Will a flexible or rigid method typically have the highest test error?

Somewhere in between a very flexible and a very rigid model will often be the sweet spot. Both a flexible and a rigid model will typically have high test error. Which of these is highest depends on the distribution of the data. However, between the models chosen in the figure, it is apparent that the more rigid method (linear regression) has the highest test error, but this is specific to this example.

b)

Does a small variance imply that the data has been under- or overfit?

A large variance could imply that the data has been overfit, because more flexible statistical methods have higher variance. This is the result of the flexible method following the observations very closely, which leads to a high variance, since the estimated function $\hat{f}$ will change a lot if the observations change. This overfitting is observed with increasing flexibility in the figure, because the mean squared error increases for the test data, despite the decrease of the mean squared error for the training data. One can say that the flexible model

tries too hard to find patterns in the data, and consequently picks up patterns that are not to be found in reality (these are caused by random chance and not by properties of the unknown function f).

c)

Relate the problem of over- and underfitting to the bias-variance trade-off.

The bias-variance trade-off says that, in order to minimize the expected test error, one needs to select statistical models that achieve low variance and low bias. A very flexible method will achieve low bias, but high variance, while a very rigid method will achieve low variance, but high bias. When the data is overfit, the variance becomes too large, despite the fact that the bias is small. In this case, the variance is "overpowering" the decrease in bias, which means that the expected test error increases. Similarly, when the data is underfit, the variance is low but the bias is large. In this case, the bias is too large compared to the low variance, and the expected test error increases. This is why a model which has the "right amount" of flexibility often is the best way to go when the goal is to minimize the expected test error.

Problem 3 – Exercise 2.4.9 from ISL textbook (modified)

This exercise involves the `Auto` dataset from the `ISLR` library. Load the data into your R session by running the following commands:

```
library(ISLR)
data(Auto)
```

a)

View the data. What are the dimensions of the data? Which predictors are quantitative and which are qualitative?

```
dim(Auto) # Dimensions.
```

```
#> [1] 392   9
```

```
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
```

```
sapply(Auto, class) # Makes it more obvious which predictors are qualitative and quantitative.
```

```
#>          mpg    cylinders displacement   horsepower       weight acceleration
#>    "numeric"    "numeric"    "numeric"    "numeric"    "numeric"    "numeric"
#>         year       origin         name
#>    "numeric"    "numeric"     "factor"
```

```r
str(Auto) # Could also be used (easier).
```

```
#> '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 1
```

All predictors are quantitative, except for 'name', 'origin' and 'cylinders', which are qualitative.

b)

What is the range (min, max) of each quantitative predictor? Hint: use the ***range()*** *function. For more advanced users, check out* ***sapply()****.*

```r
# Remember to import dplyr in setup for the first variant to work.
# Two different methods of removing the categorical variable.
quant <- Auto %>% dplyr::select(-c(name, origin, cylinders)) # Using dplyr.
quant2 <- Auto[, -c(2, 8, 9)] # Using regular R.
identical(quant, quant2) # Sidenote: Shows that the two methods give the same result.
```

```
#> [1] TRUE
```

```r
sapply(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)

What is the mean and standard deviation of each quantitative predictor?

```r
sapply(quant, mean) # Mean.
```

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

```r
sapply(quant, sd) # Standard deviation.
```

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

d)

Now, make a new dataset called ***ReducedAuto*** *where you remove the 10th through 85th observations. What is the range, mean and standard deviation of the quantitative predictors in this reduced set?*

```r
ReducedAuto <- Auto[-c(10:85), ]
dim(ReducedAuto) # The rows have been removed.
```

```
#> [1] 316   9
```

```r
quant.ReducedAuto <- ReducedAuto %>% dplyr::select(-c(name, origin, cylinders))
sapply(quant.ReducedAuto, range) # 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(quant.ReducedAuto, mean) # Mean.
```

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

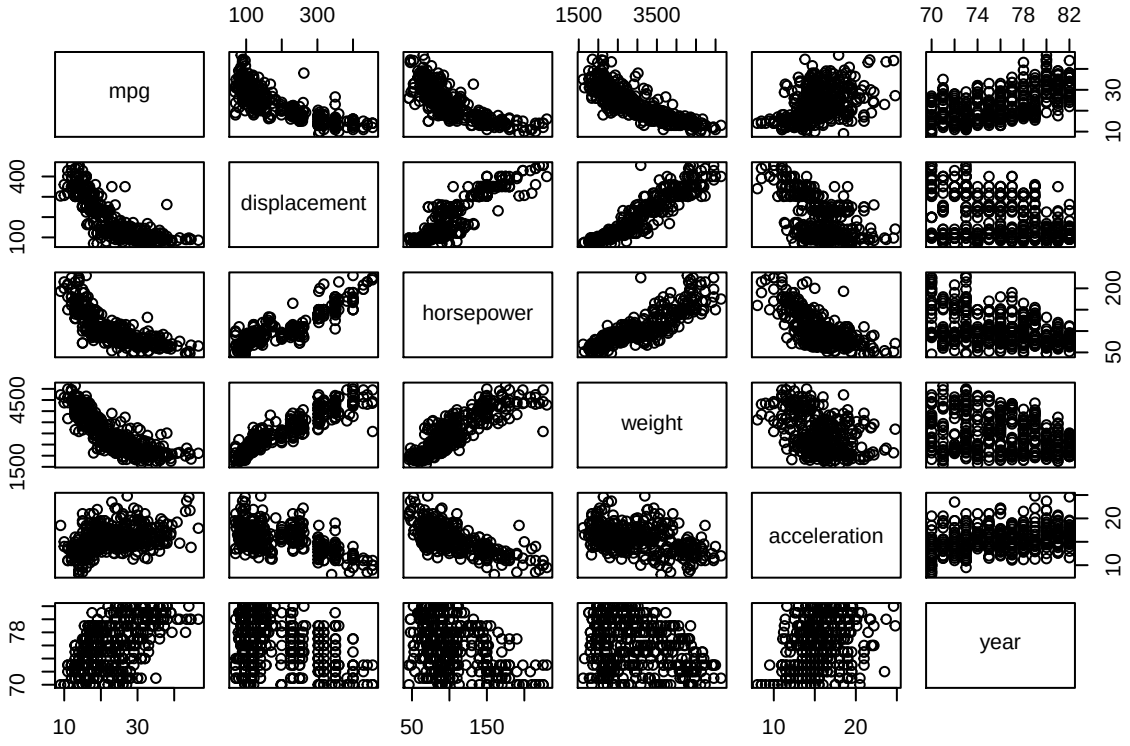
```r
sapply(quant.ReducedAuto, sd) # Standard deviation.
```

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

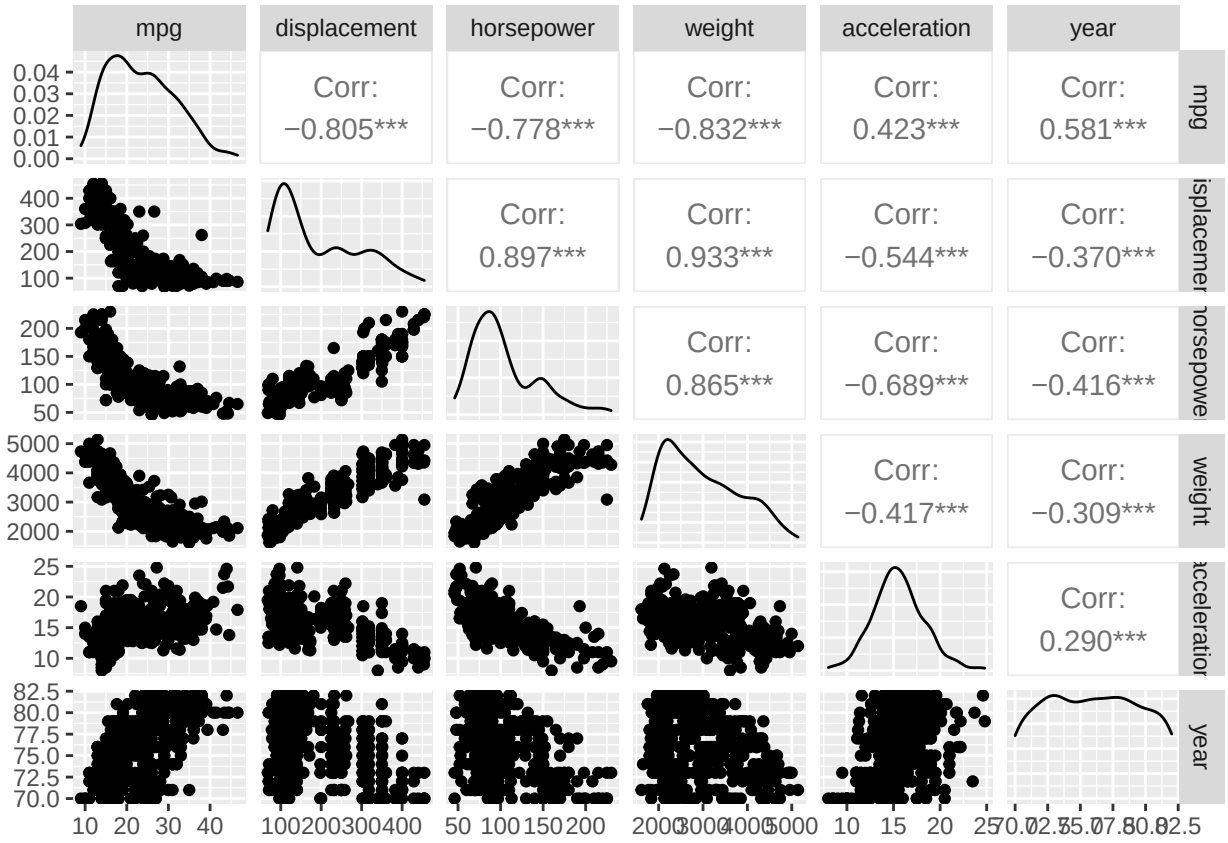
e)

Using the full dataset, investigate the quantitative predictors graphically using a scatterplot. Do you see any strong relationships between the predictors? Hint: try out the `ggpairs()` *function from the* `GGally` *package.*

```r
library(GGally)
pairs(quant) # Regular pairs plot.
```




```r
ggpairs(quant)
```



Based on the scatter plots, some relationships between the quantitative predictors seem to be stronger than others. It looks like the following pairs of predictors are highly correlated

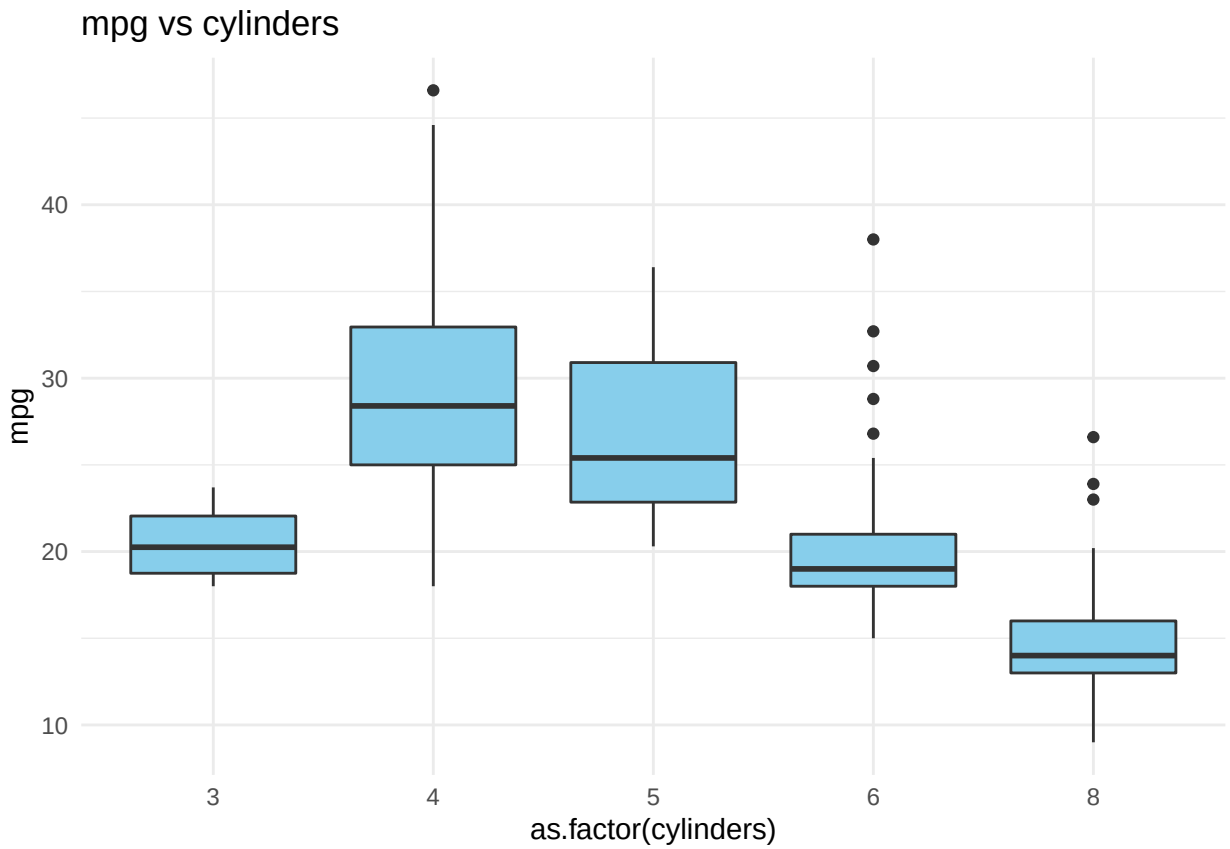
- Displacement and weight
- Horsepower and weight
- Displacement and horsepower
- Mpg and displacement
- Mpg and weight
- Mpg and horsepower

f)

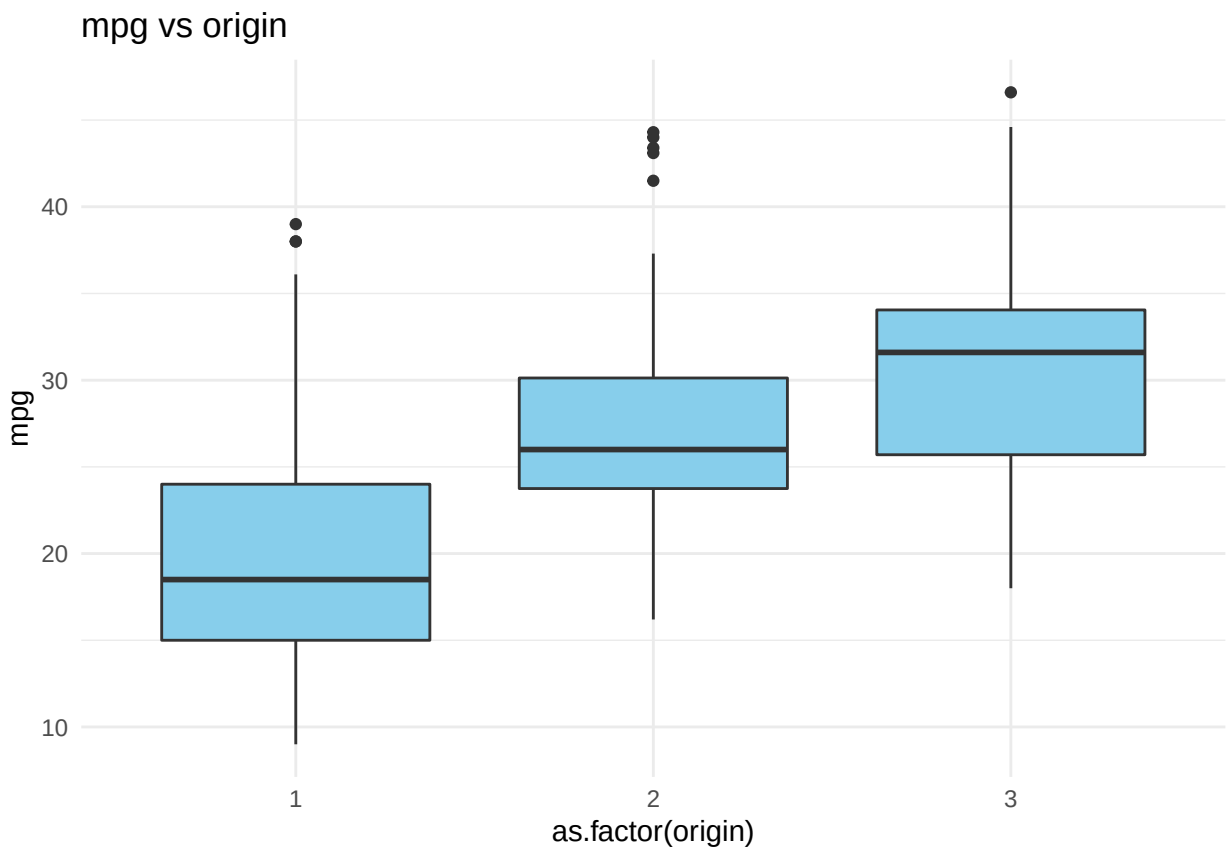
Suppose we wish to predict gas milage (`mpg`) on the basis of the other variables (both quantitative and qualitative). Make some plots showing the relationships between `mpg` and the qualitative predictors (hint: `geom_boxplot()`). Which predictors would you consider helpful when predicting `mpg`?

Checking the qualitative predictors 'cylinders', 'origin' and 'name'.

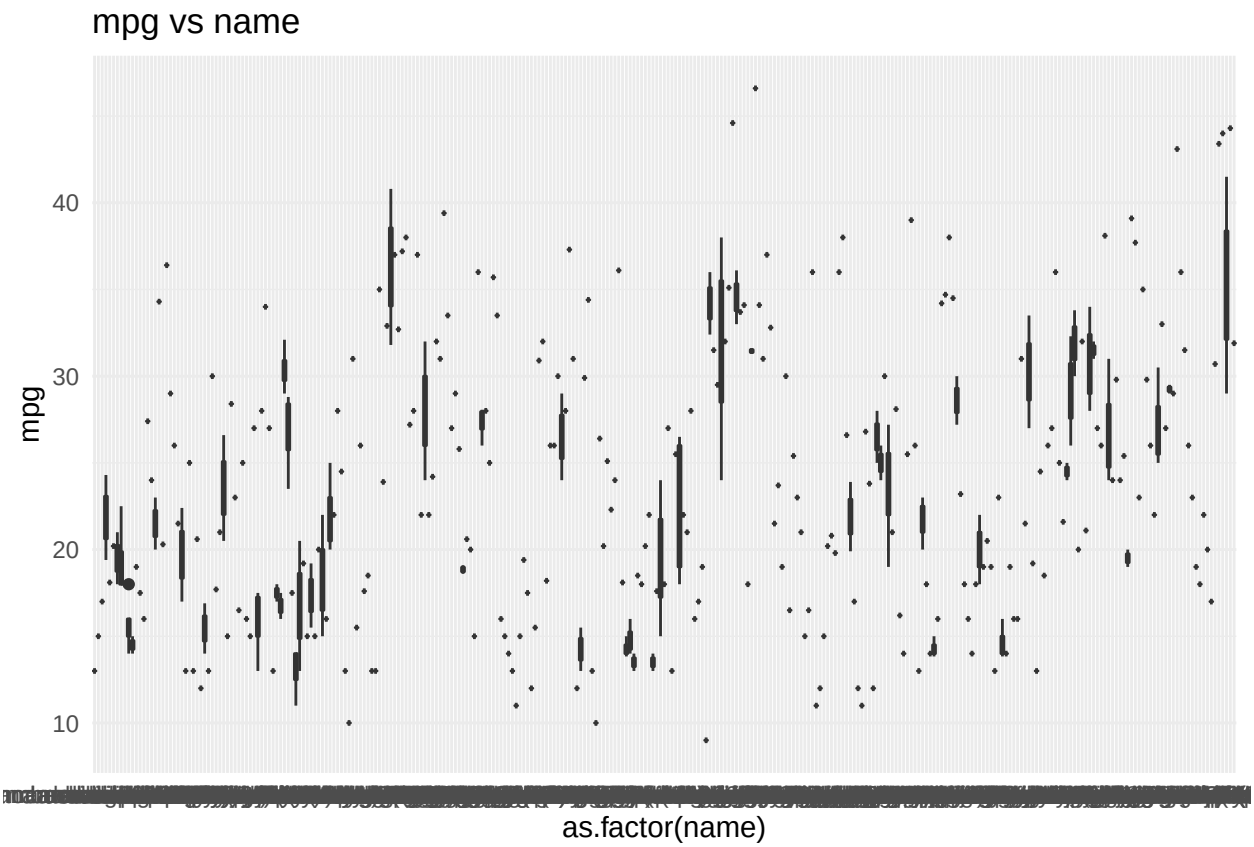
```
ggplot(Auto, aes(x = as.factor(cylinders), y = mpg)) +
   geom_boxplot(fill = "skyblue") +
   labs(title = "mpg vs cylinders") +
   theme_minimal()
```




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




```
ggplot(Auto, aes(x = as.factor(name), y = mpg)) +
   geom_boxplot(fill = "skyblue") +
   labs(title = "mpg vs name") +
   theme_minimal()
```



Based on the scatter plots (and the correlation coefficients) in task e) I would guess that **weight**, **displacement**, and **horsepower** (quantitative), as well as **cylinders** and **origin** (qualitative), could be considered helpful when trying to predict **mpg**.

g)

The correlation of two variables X and Y are defined as

$$\text{cor}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}.$$

Both the correlation matrix and covariance matrix are easily assessed in R with the `cor()` *and* `cov()` *functions. Use only the covariance matrix to find the correlation between* `mpg` *and* `displacement`*,* `mpg` *and* `horsepower`*, and* `mpg` *and* `weight`*. Do your results coincide with the correlation matrix you find using* `cor(Auto[,quant])`*?*

```
quantile <-  c(1,3,4,5,6,7)
covMat <-  cov(Auto[,quantile]) # Covariance matrix of the given quantiles.
knitr::kable(covMat)
```

	mpg	displacement	horsepower	weight	acceleration	year
mpg	60.918142	-657.5852	-233.85793	-5517.4407	9.115514	16.691477
displacement	-657.585207	10950.3676	3614.03374	82929.1001	-156.994435	-142.572133
horsepower	-233.857926	3614.0337	1481.56939	28265.6202	-73.186967	-59.036432
weight	-5517.440704	82929.1001	28265.62023	721484.7090	-976.815253	-967.228457
acceleration	9.115514	-156.9944	-73.18697	-976.8153	7.611331	2.950462
year	16.691477	-142.5721	-59.03643	-967.2285	2.950462	13.569915

```r
# Mpg and displacement.
cor.mpg.disp <- covMat[1,2]/sqrt(covMat[1,1]*covMat[2,2])
cor.mpg.disp
```

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

```r
# Mpg and horsepower.
cor.mpg.horse <- covMat[1,3]/sqrt(covMat[1,1]*covMat[3,3])
cor.mpg.horse
```

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

```r
# Mpg and weight.
cor.mpg.weight <- covMat[1,4]/sqrt(covMat[1,1]*covMat[4,4])
cor.mpg.weight
```

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

```r
# Correlation matrix.
corMat <- cor(Auto[, quantile])
knitr::kable(corMat)
```

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

```r
# Check if they coincide.
identical(cor.mpg.disp, corMat[1,2])
```

```
#> [1] TRUE
```

```r
identical(cor.mpg.horse, corMat[1,3])
```

```
#> [1] TRUE
```

```r
identical(cor.mpg.weight, corMat[1,4])
```

```
#> [1] TRUE
```

Problem 4 – Multivariate normal distribution

The pdf of a multivariate normal distribution is on the form

$$f(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|} \exp\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\},$$

where $\mathbf{x}$ is a random vector of size $p \times 1$, $\boldsymbol{\mu}$ is the mean vector of size $p \times 1$ and $\boldsymbol{\Sigma}$ is the covariance matrix of size $p \times p$.

a)

Use the `mvrnorm()` function from the `MASS` library to simulate 1000 values from multivariate normal distributions with

i)

$$\boldsymbol{\mu} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

```
samples.1 <- mvrnorm(n = 1000, mu = c(2, 3), Sigma = matrix(c(1, 0, 0, 1), nrow = 2))
```

ii)

$$\boldsymbol{\mu} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 5 \end{pmatrix},$$

```
samples.2 <- mvrnorm(n = 1000, mu = c(2, 3), Sigma = matrix(c(1, 0, 0, 5), nrow = 2))
```

iii)

$$\boldsymbol{\mu} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 2 \\ 2 & 5 \end{pmatrix},$$

```
samples.3 <- mvrnorm(n = 1000, mu = c(2, 3), Sigma = matrix(c(1, 2, 2, 5), nrow = 2))
```

iv)

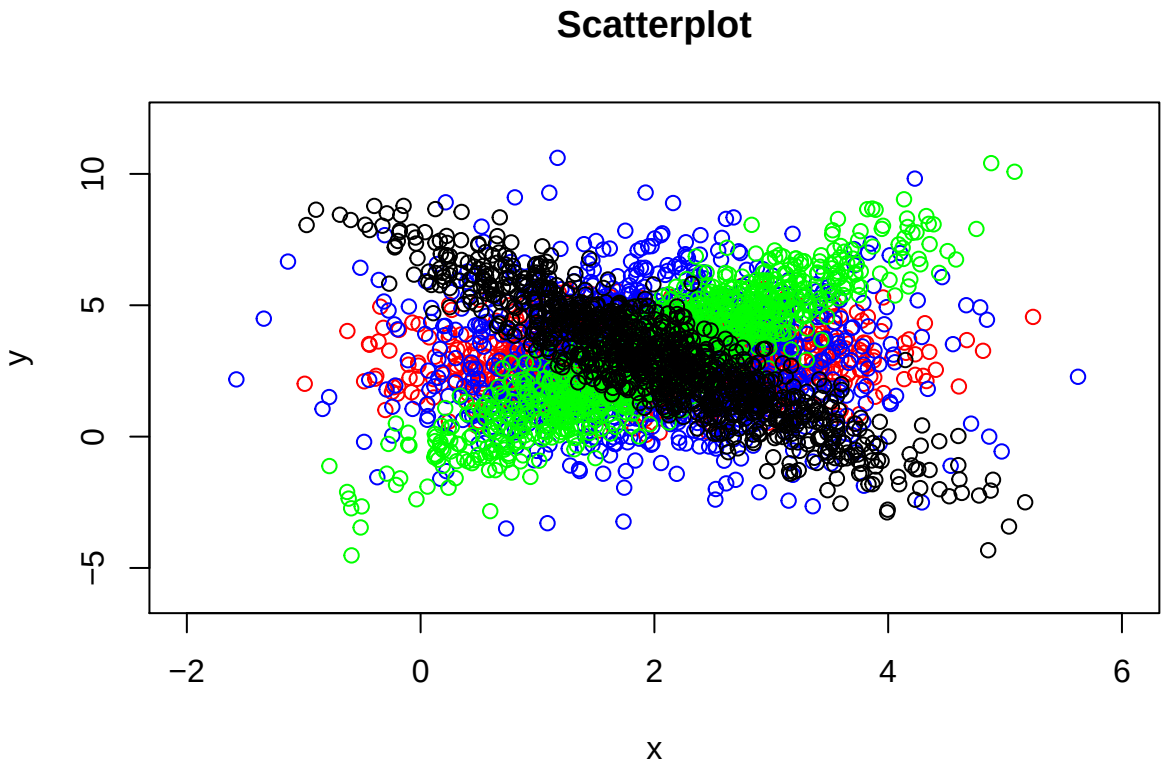
$$\boldsymbol{\mu} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & -2 \\ -2 & 5 \end{pmatrix}.$$

```
samples.4 <- mvrnorm(n = 1000, mu = c(2, 3), Sigma = matrix(c(1, -2, -2, 5), nrow = 2))
```

b)

Make a scatterplot of the four sets of simulated datasets. Can you see which plot belongs to which distribution?

```
plot(NULL, NULL, main = "Scatterplot", xlim = c(-2, 6), ylim = c(-6, 12), xlab = "x", ylab = "y")
points(samples.1, col = "red")
points(samples.2, col = "blue")
points(samples.3, col = "green")
points(samples.4)
```



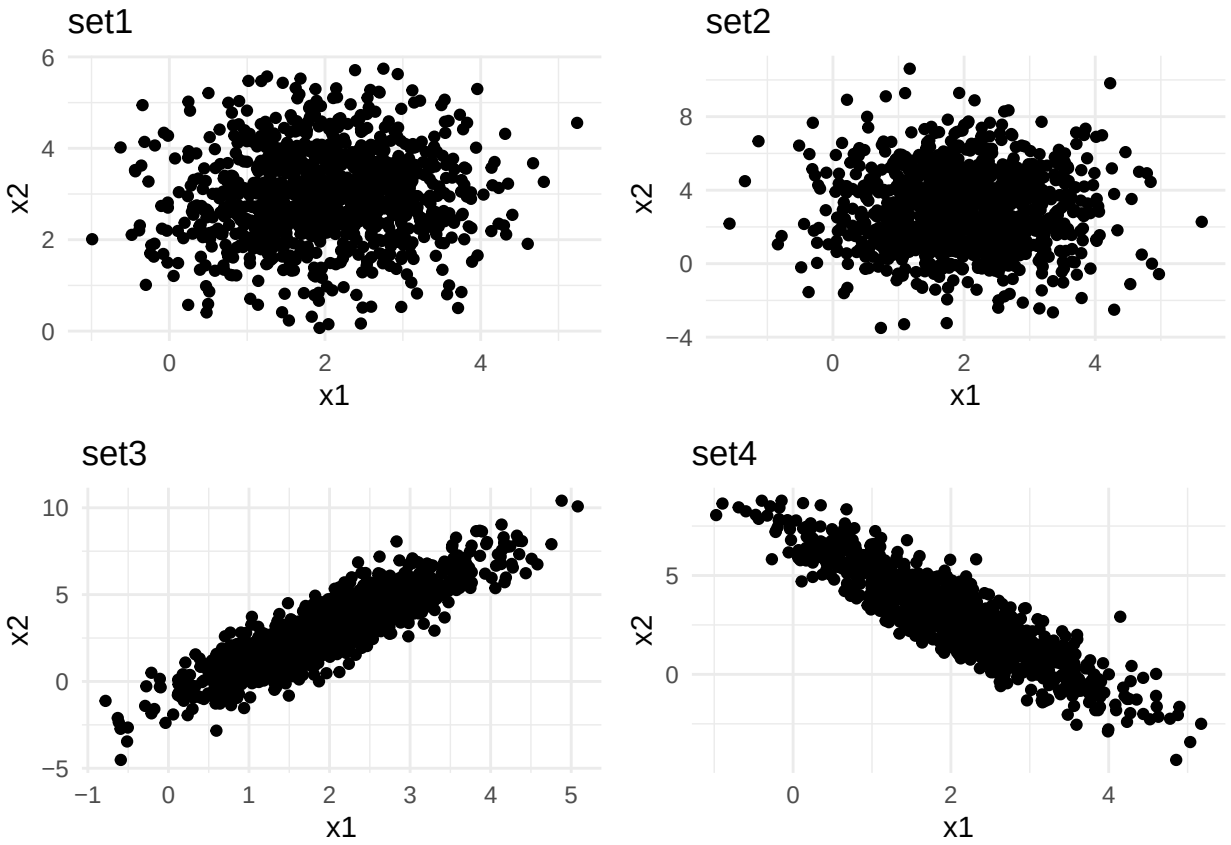

```
# Could have plotted like in LF (way better, since each scatter plot is placed in a grid).
library(gridExtra)
set1 <- as.data.frame(samples.1)
colnames(set1) = c("x1", "x2")

set2 <- as.data.frame(samples.2)
colnames(set2) = c("x1", "x2")

set3 <- as.data.frame(samples.3)
colnames(set3) = c("x1", "x2")

set4 <- as.data.frame(samples.4)
colnames(set4) = c("x1", "x2")

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)
```



It is apparent that the green dots belong to the distribution in iii), since these points are clearly correlated, which is not the case for the two first distributions. Furthermore, the blue points correspond to distribution ii), since the variance is larger in y. The red points then correspond to i), which is a "standard Gaussian" around (2, 3). The black points show distribution iv) since the correlation is negative, and the variance in y is the same as for iii).

Problem 5 – Theory and practice: training and test MSE; bias-variance

We will now look closely into the simulations and calculations performed for the training error (`trainMSE`), test error (`testMSE`), and the bias-variance trade-off in lecture 1 of module 2.

Below, the code to run the simulation is included. The data is simulated according to the following specifications:

- *True function $f(x) = x^2$ with normal noise $\varepsilon \sim N(0, 2^2)$.*
- *$x = -2.0, -1.9, ..., 4.0$ (grid with 61 values).*
- *Parametric models are fitted (polynomials of degree 1 to degree 20).*
- *M=100 simulations.*

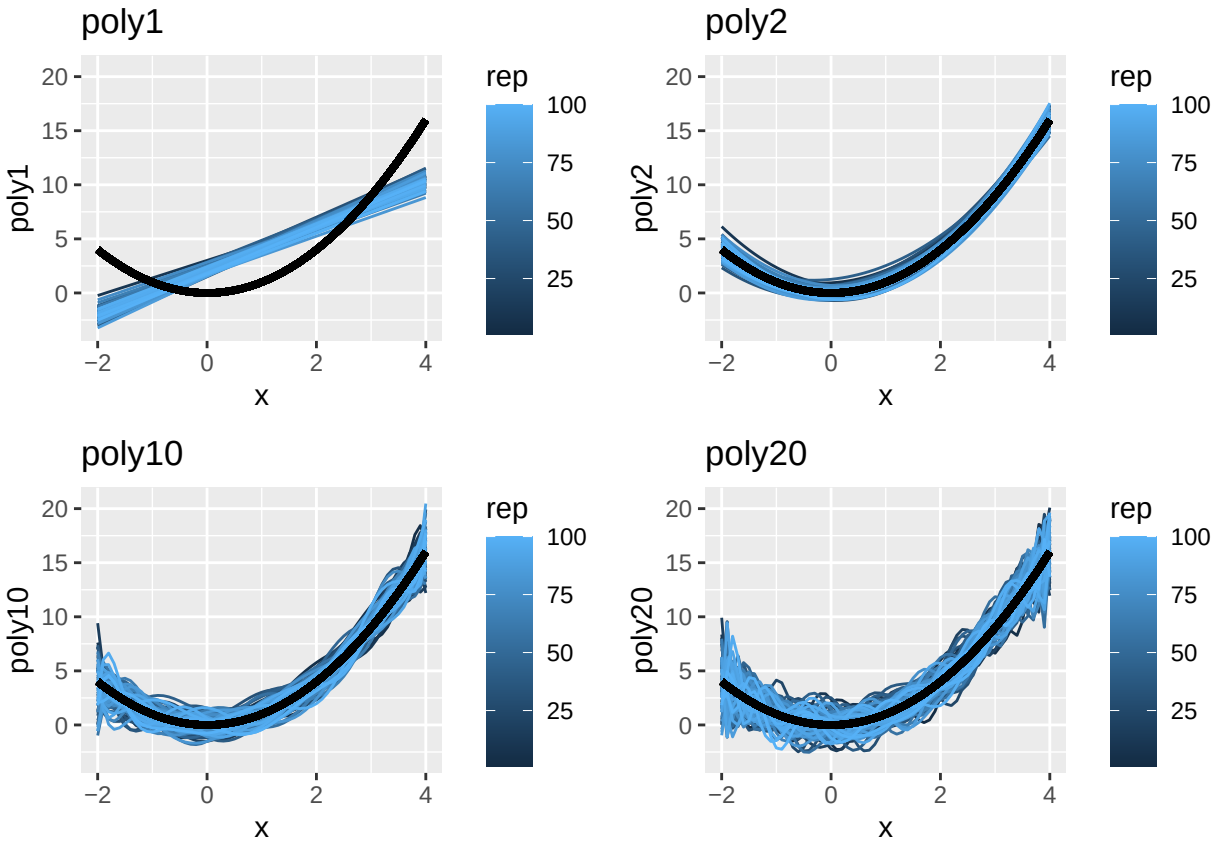
a) Problem set-up

Look at the code below, copy it and run it yourself. Explain roughly what is done (you do not need to understand the code in detail).

We will learn more about the `lm` function in Module 3 - now just think of this as fitting a polynomial regression and then `predict` gives the fitted curve in our grid points. `predarray` is just a way to save M simulations of 61 gridpoints in x and 20 polynomial models.

```
library(ggplot2)
library(ggpubr)
```

```r
set.seed(2) # to reproduce
M=100 # repeated samplings, x fixed
nord=20 # order of polynoms
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)
ymat = matrix(rep(true_y,M),byrow=T,nrow=M) + error
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)))
  }
}
# 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")
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")
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")
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")
ggarrange(p1,p2,p10,p20)
```



What do you observe in the produced plot? Which polynomial fits the best to the true curve?

First of all, it is obvious that poly1 is very far away from the true curve. Furthermore, poly2 is fitting the black true function the best, based on the smaller variance/spread of the fitting curves and the expected value of all the repeated samples. All in all, poly1 is underfit, while poly10 and (especially) poly20 are overfit, since the variance is larger.

b) Train and test MSE

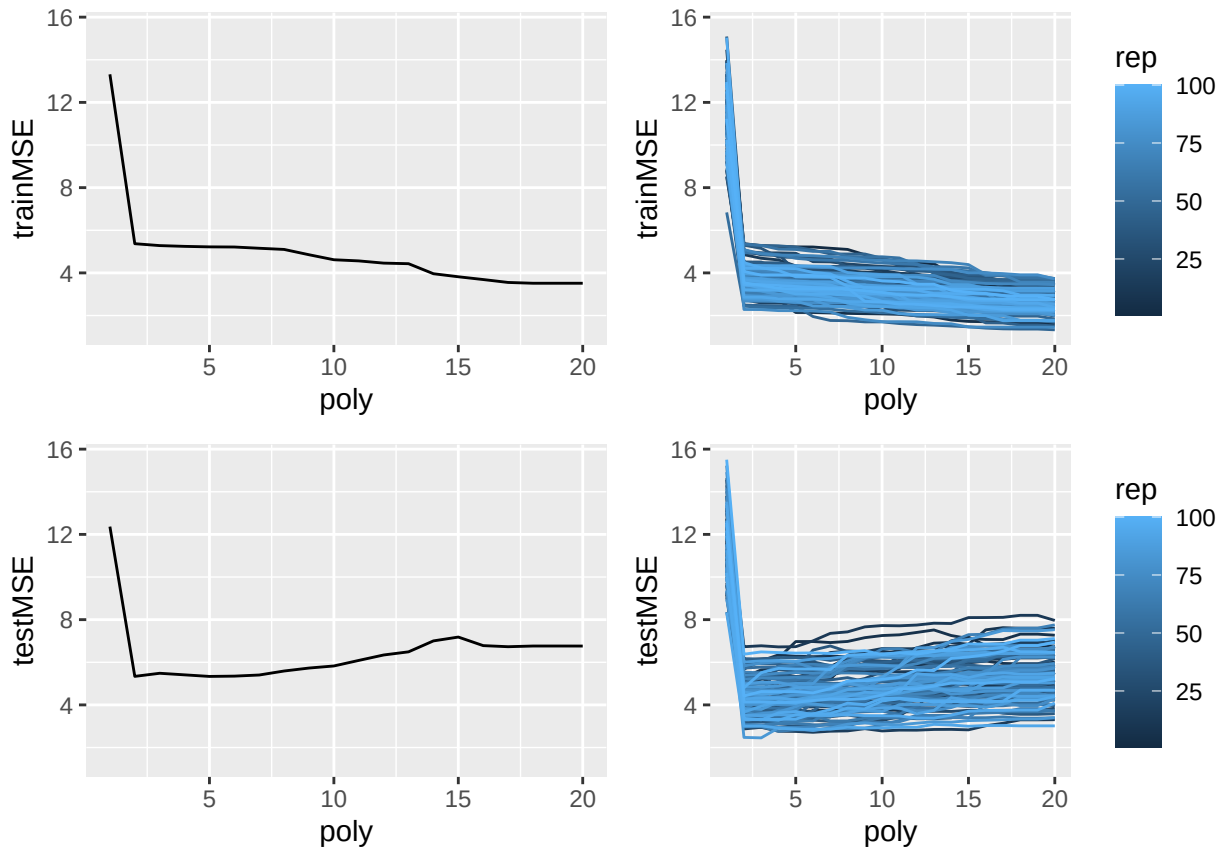
First we produce predictions at each grid point based on our training data (`x` and `ymat`). Then we draw new observations to calculate test MSE, see `testymat`. Observe how `trainMSE` and `testMSE` are calculated, and then run the code

```
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^3)
}
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)
testMSE=matrix(ncol=nord,nrow=M)
for (i in 1:M){
  trainMSE[i,]=apply((predarray[i,,]-ymat[i,])^2,2,mean)
  testMSE[i,]=apply((predarray[i,,]-testymat[i,])^2,2,mean)
}
```

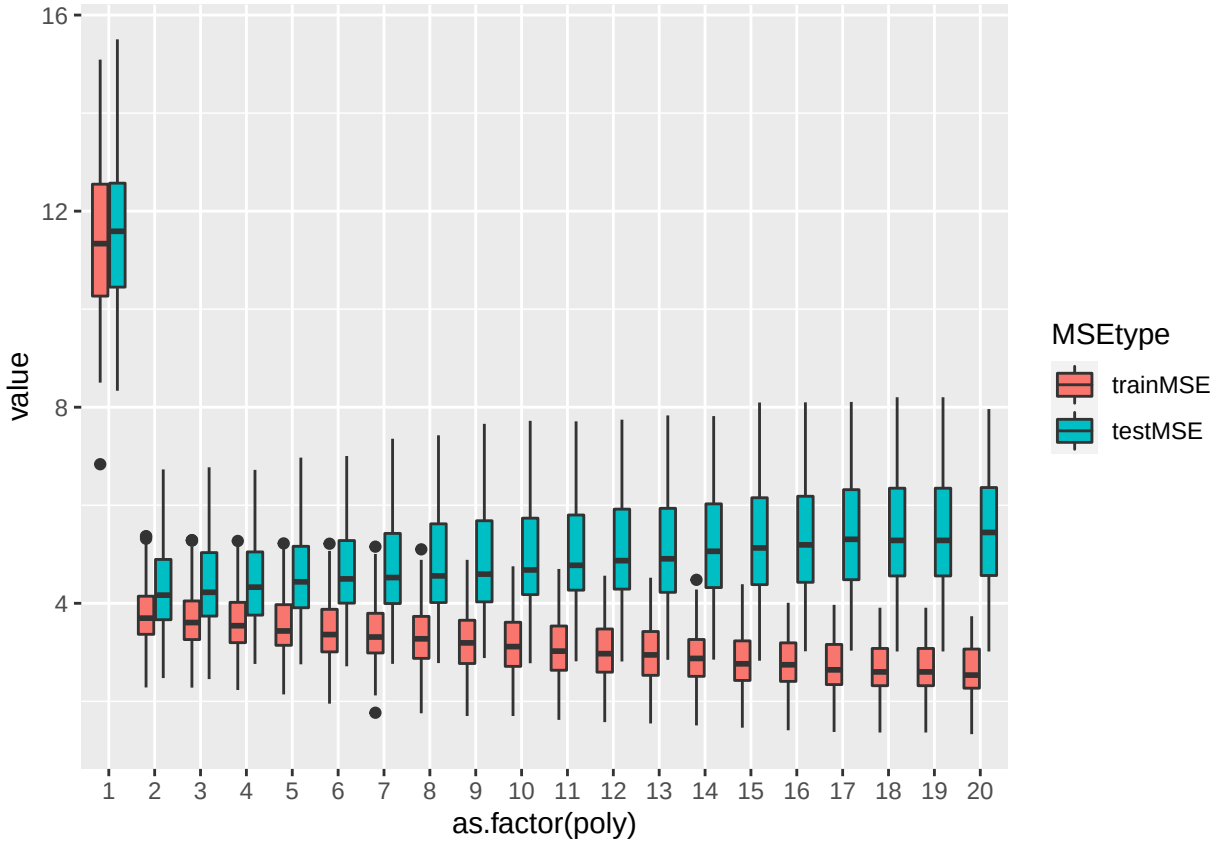
Next, we plot – first for one train + test data set, then for 99 more.

```
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()
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()
testpall= ggplot(data=sdf,aes(x=poly,group=rep,y=testMSE,colour=rep))+scale_y_continuous(limits=yrange)
ggarrange(p1,pall,testp1,testpall)
```

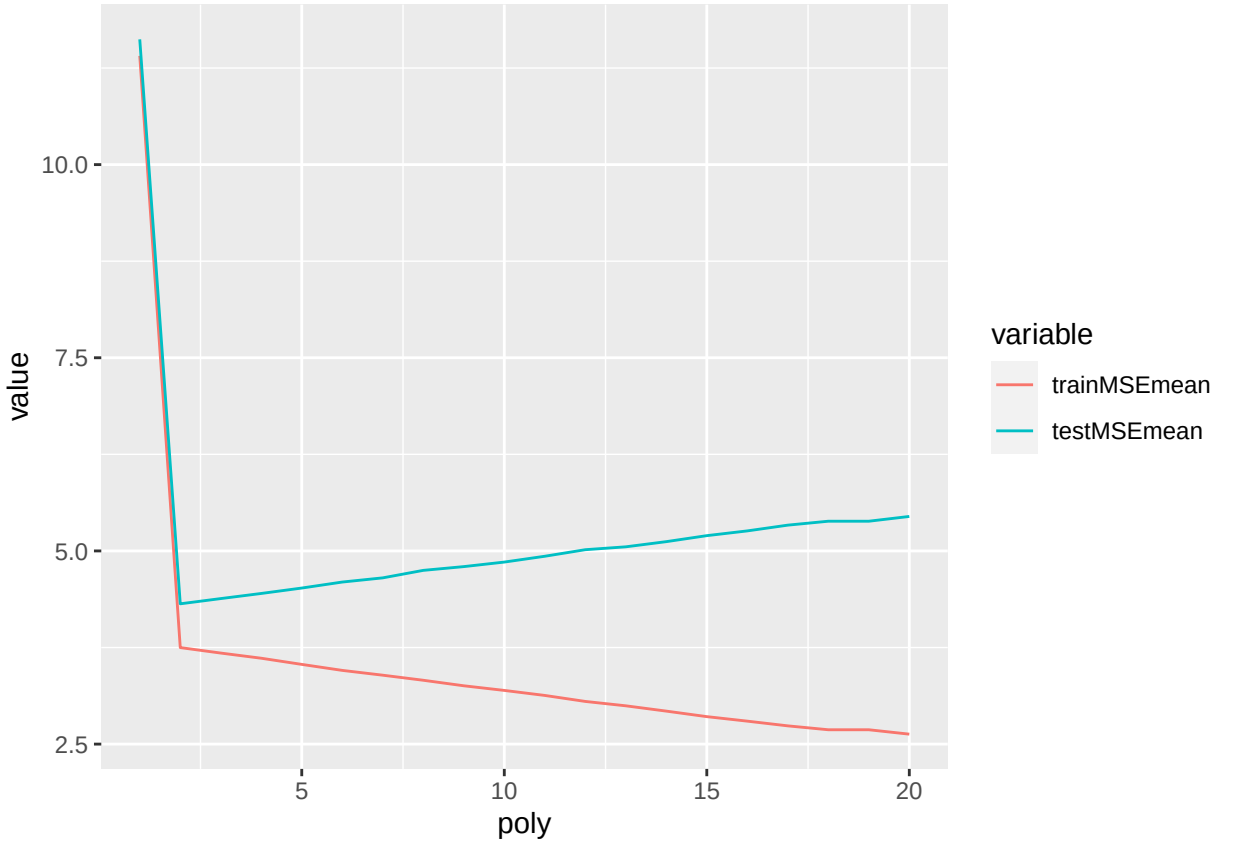


More plots now: first boxplot and then mean for train and test MSE:

```
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()
```



- *Which value of the polynomial gives the smallest mean testMSE?* Answer: The value of the polynomial which gives the smallest mean is poly2, as is seen from the last plot above.
- *Which gives the smallest mean trainMSE?* Answer: poly20 (and larger if a higher number of degrees of freedom had been added to the simulations) gives the smallest mean trainMSE. The trainMSE is decreasing with increasing poly-number, until it hits a random error (variance) in the model.
- *Which would you use to predict a new value of y?* Answer: Since poly2 gives the smallest mean testMSE, I would use this to predict a new value of y.

c) Bias and variance - we use the truth!

Finally, we want to see how the expected quadratic loss can be decomposed into

- *irreducible error: $Var(\varepsilon) = 4$*
- *squared bias: difference between mean of estimated parametric model chosen and the true underlying curve (`truefunc`)*
- *variance: variance of the estimated parametric model*

Notice that the test data is not used – only predicted values in each x grid point.

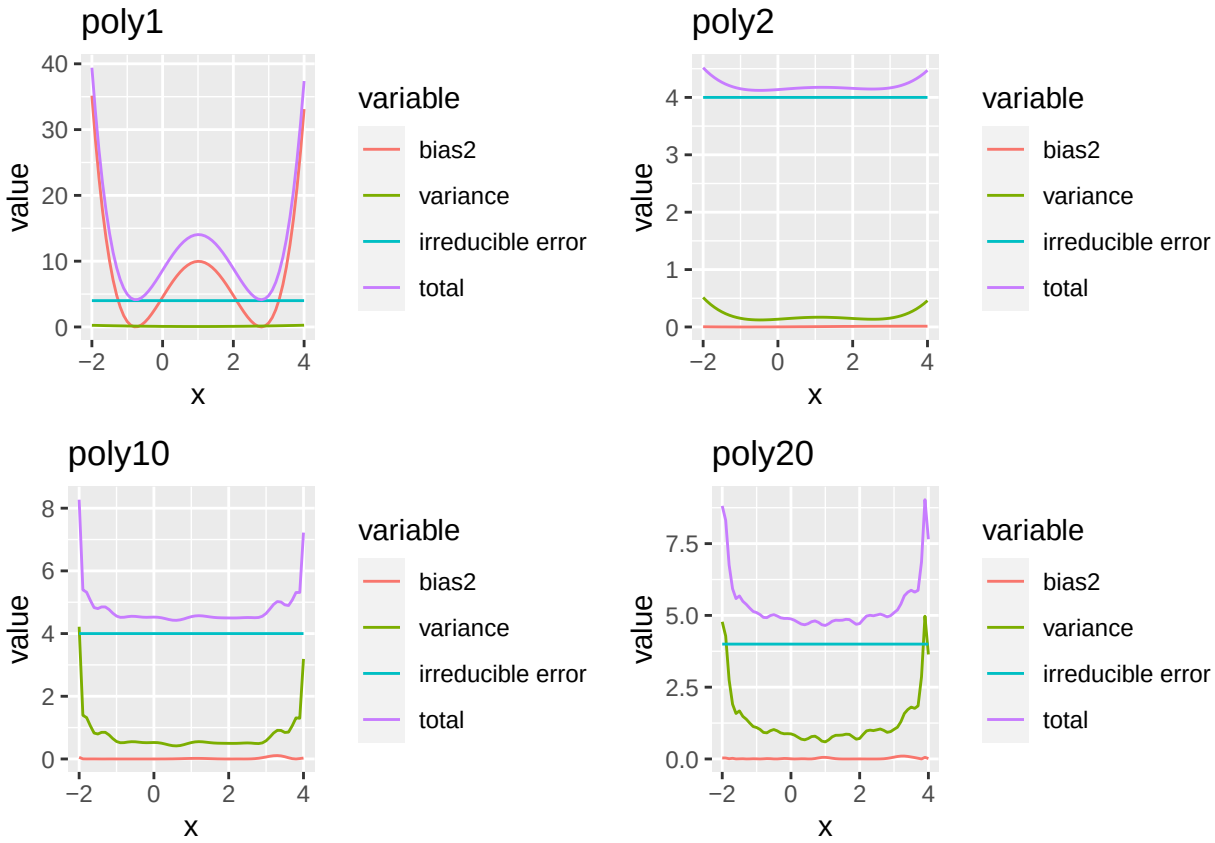
Study and run the code. Explain the plots produced.

```
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 finally used!
```

Plotting the polys as a function of x:

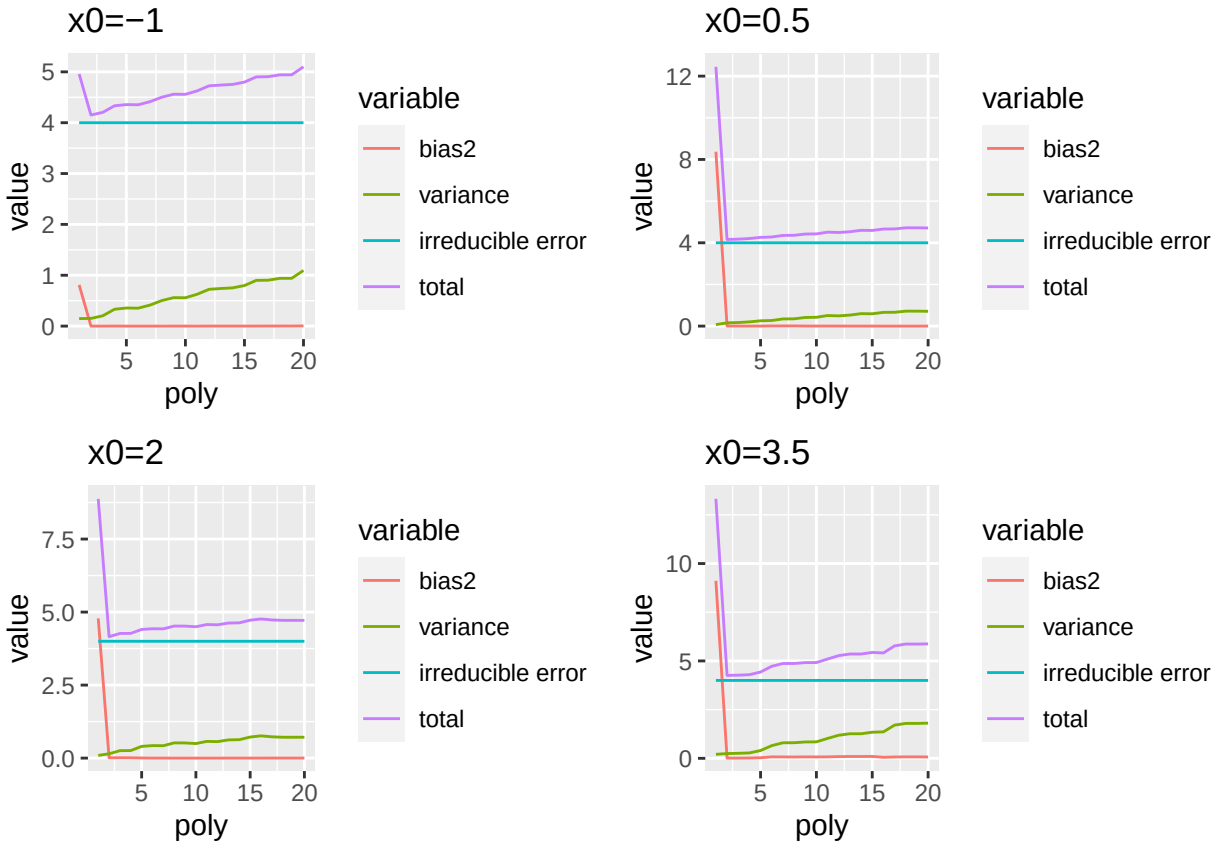
```
df=data.frame(rep(x,each=nord),rep(1:nord,length(x)),c(bias2mat),c(varmat),rep(4,prod(dim(varmat)))) #i
colnames(df)=c("x","poly","bias2","variance","irreducible error") #suitable for plotting
df$total=df$bias2+df$variance+df$`irreducible error`
hdf=melt(df,id=c("x","poly"))
hdf1=hdf[hdf$poly==1,]
hdf2=hdf[hdf$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")
p2=ggplot(data=hdf2,aes(x=x,y=value,colour=variable))+geom_line()+ggtitle("poly2")
p10=ggplot(data=hdf10,aes(x=x,y=value,colour=variable))+geom_line()+ggtitle("poly10")
p20=ggplot(data=hdf20,aes(x=x,y=value,colour=variable))+geom_line()+ggtitle("poly20")
ggarrange(p1,p2,p10,p20)
```



Now plotting effect of more complex model at 4 chosen values of x, compare to Figures in 2.12 on page 36 in ISL (our textbook).

```
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")
```

```
pb=ggplot(data=hdfatxb,aes(x=poly,y=value,colour=variable))+geom_line()+ggtitle("x0=0.5")
pc=ggplot(data=hdfatxc,aes(x=poly,y=value,colour=variable))+geom_line()+ggtitle("x0=2")
pd=ggplot(data=hdfatxd,aes(x=poly,y=value,colour=variable))+geom_line()+ggtitle("x0=3.5")
ggarrange(pa,pb,pc,pd)
```



Study the final plot you produced: when the flexibility increases (poly increase), what happens with i) the squared bias, Answer: The bias decreases. *ii) the variance,* Answer: The variance increases. *iii) the irreducible error?*
Answer: The irreducible error is constant (hence: irreducible).

d) Repeat a-c

Try to change the true function ***truefunc*** *to something else - maybe order 3? What does this do the the plots produced? Maybe you then also want to plot poly3?*

Also try to change the standard deviation of the noise added to the curve (now it is sd=2). What happens if you change this to sd=1 or sd=3?

Or, change to the true function that is not a polynomial?

Acknowledgements

We thank Mette Langaas and her PhD students (in particular Julia Debik) from 2018 and 2019 for building up the original version of this exercise sheet.