Computer Lab 4

Emil K Svensson

2 December 2016

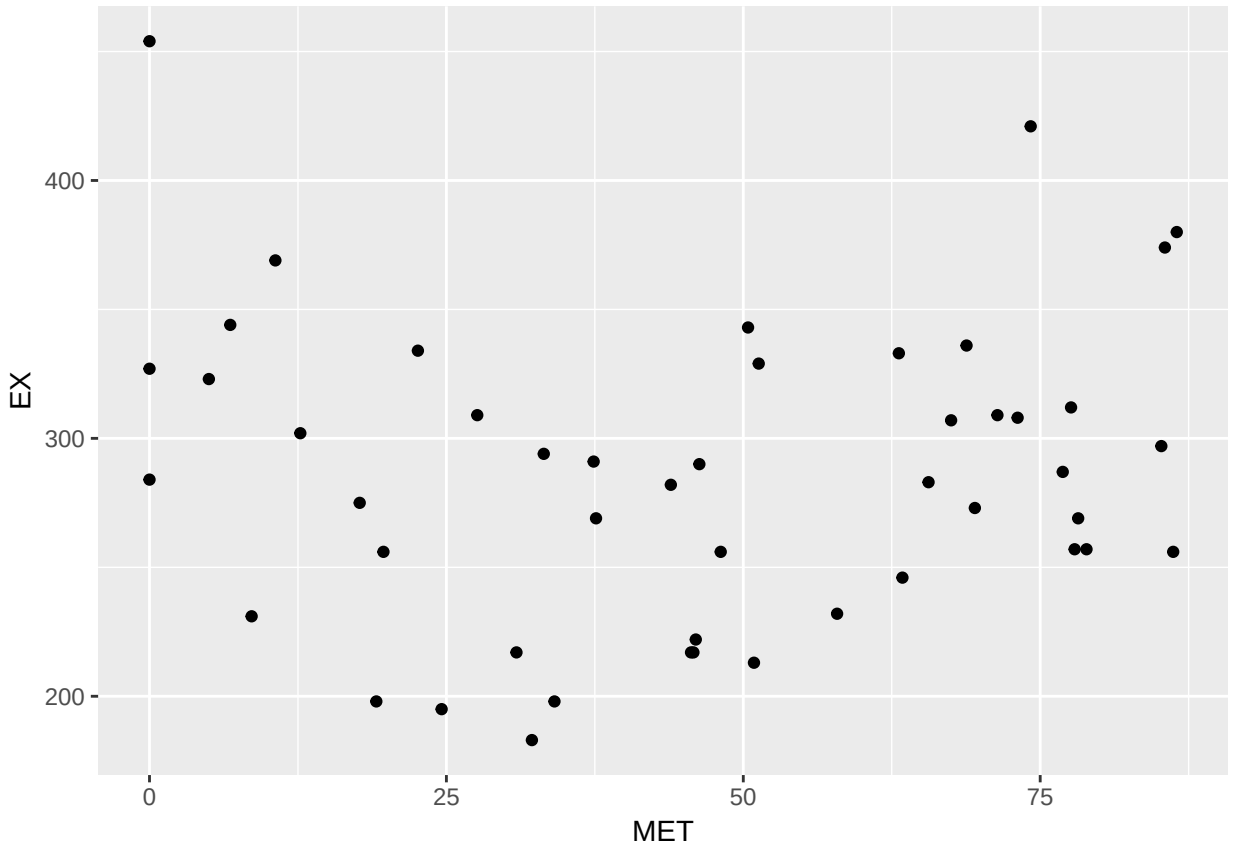
Assignment 1

1.1

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

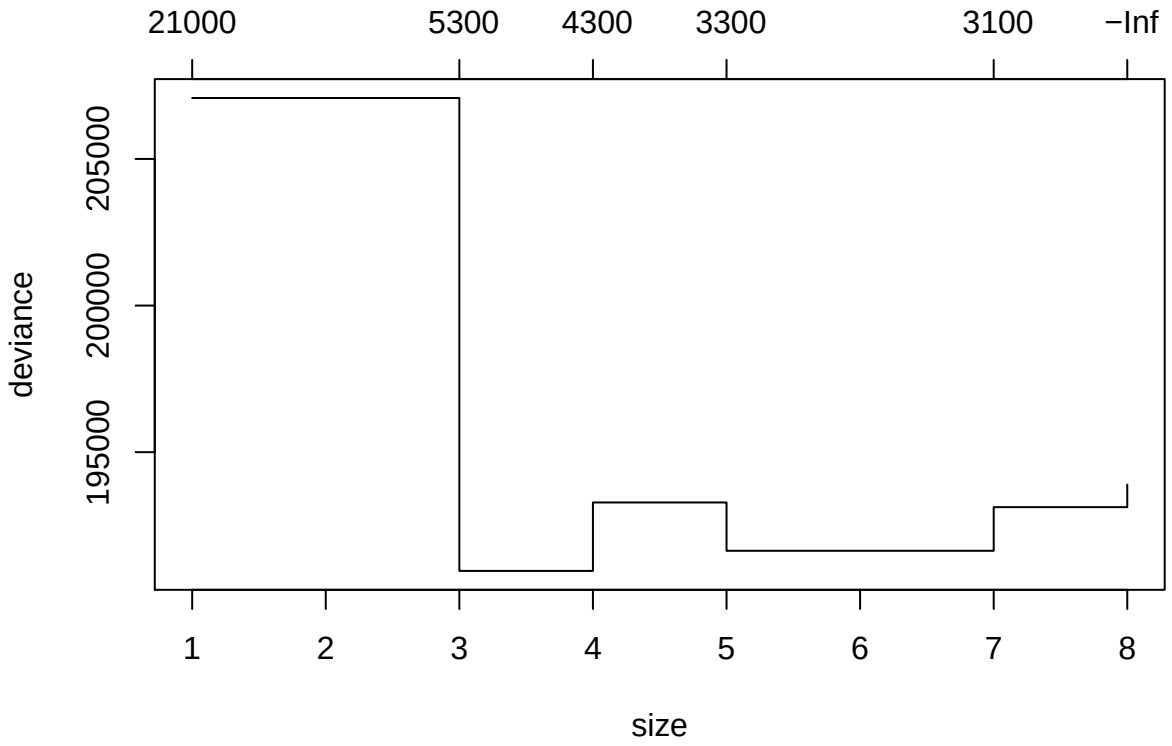
```
library(tree)
library(boot)
library(grid)
library(fastICA)
library(gridExtra)


State <- read.csv2("State.csv")
State <- State[order(State$MET),]

a11 <- ggplot(data = State, aes(x = MET,y = EX)) + geom_point()
plot(a11)
```



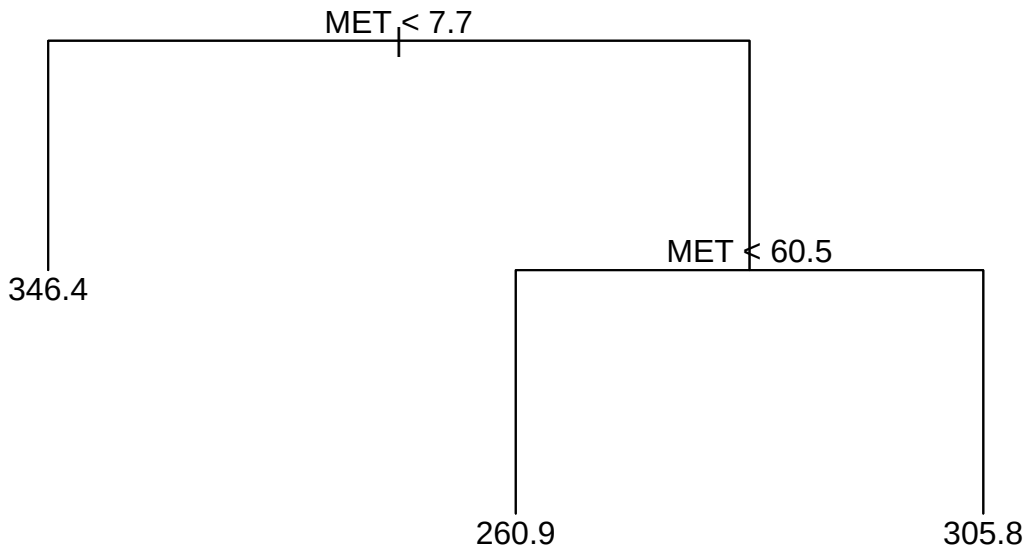
Some sort of linear model with polynomial functions and an dummy variable might be efficient at solving explaning this kind of pattern.

1.2

```
## Har eg. ingen koll alls på det här.
state.tree <- tree(EX ~ MET, data = State, minsize = 8 , split = "deviance")

#plot(state.tree)

set.seed(12345)
plot(cv.tree(state.tree))
```

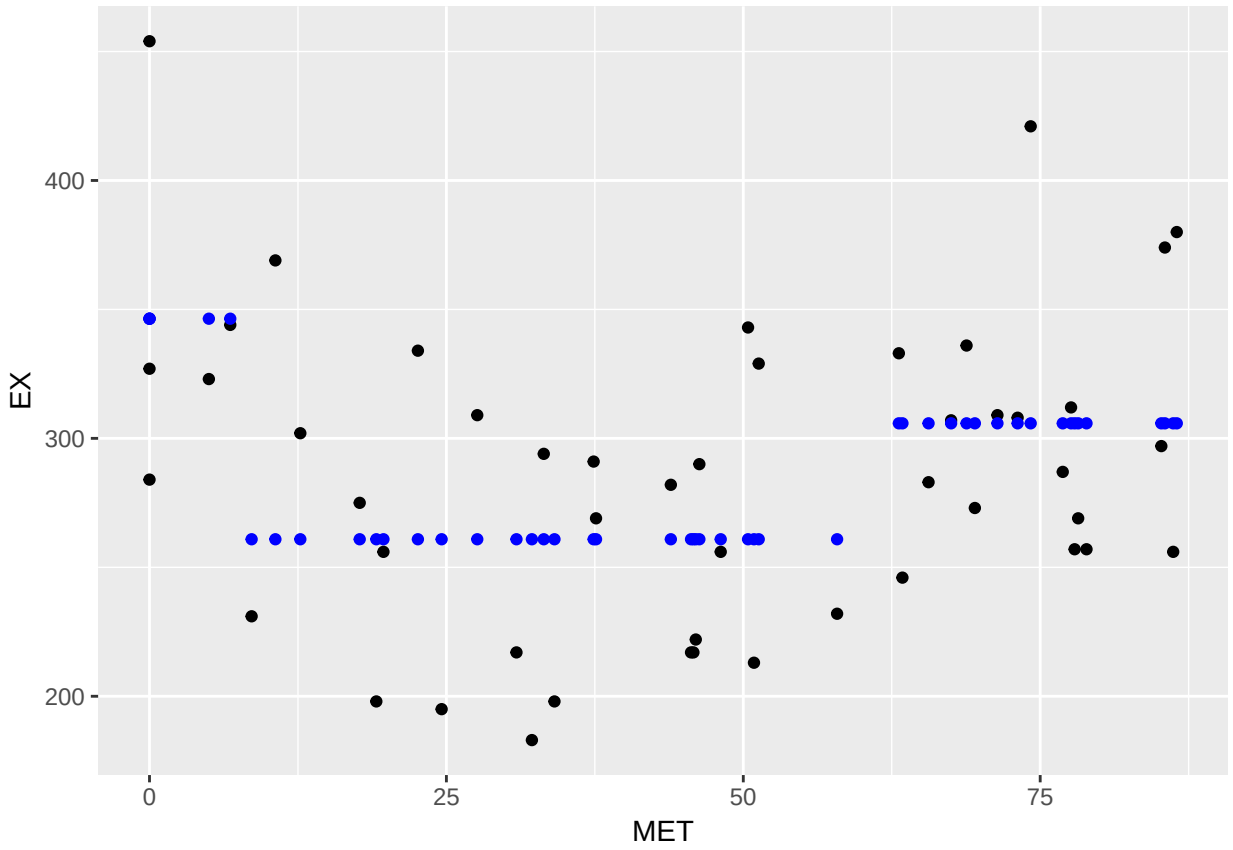



```
# three is the optimal tree.

state.prune <- prune.tree(state.tree, best=3)

plot(state.prune)
text(state.prune)
```



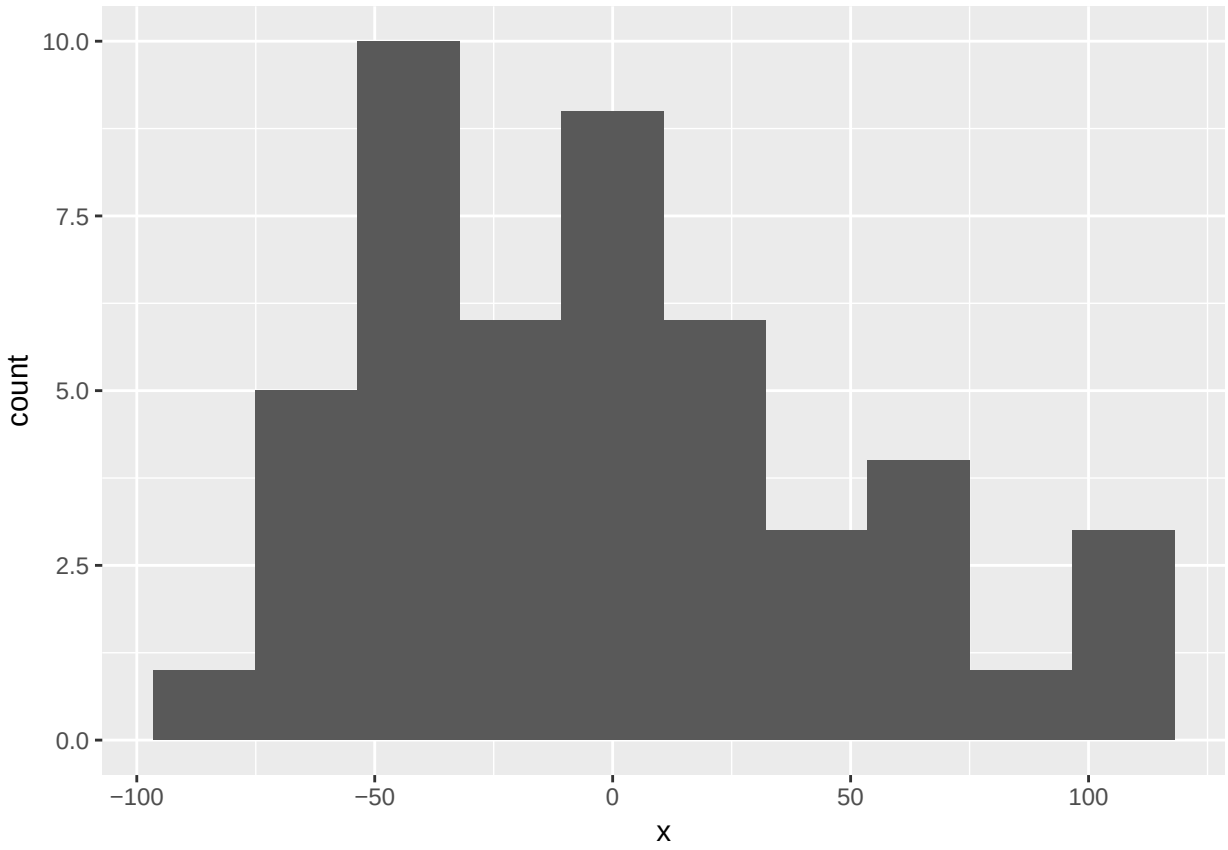
The optimal tree contained two splits and had in total three leafs.

```
a12 <- a11 + geom_point( y = predict(state.prune), col = "blue" )
plot(a12)
```



The fitted values seem to decently match the observed values, although there is quite large errors since the model only split. For values larger than 55 the fit seems the worst.

```
ggplot(data = data.frame(x = resid(state.prune))) +
  geom_histogram(aes(x = x), bins  = 10)
```



The residuals seem to have a longer left tail. But there are few observations in the data and one could easily imagine that if more data was observed the histogram should assume a more bell-shaped curve.

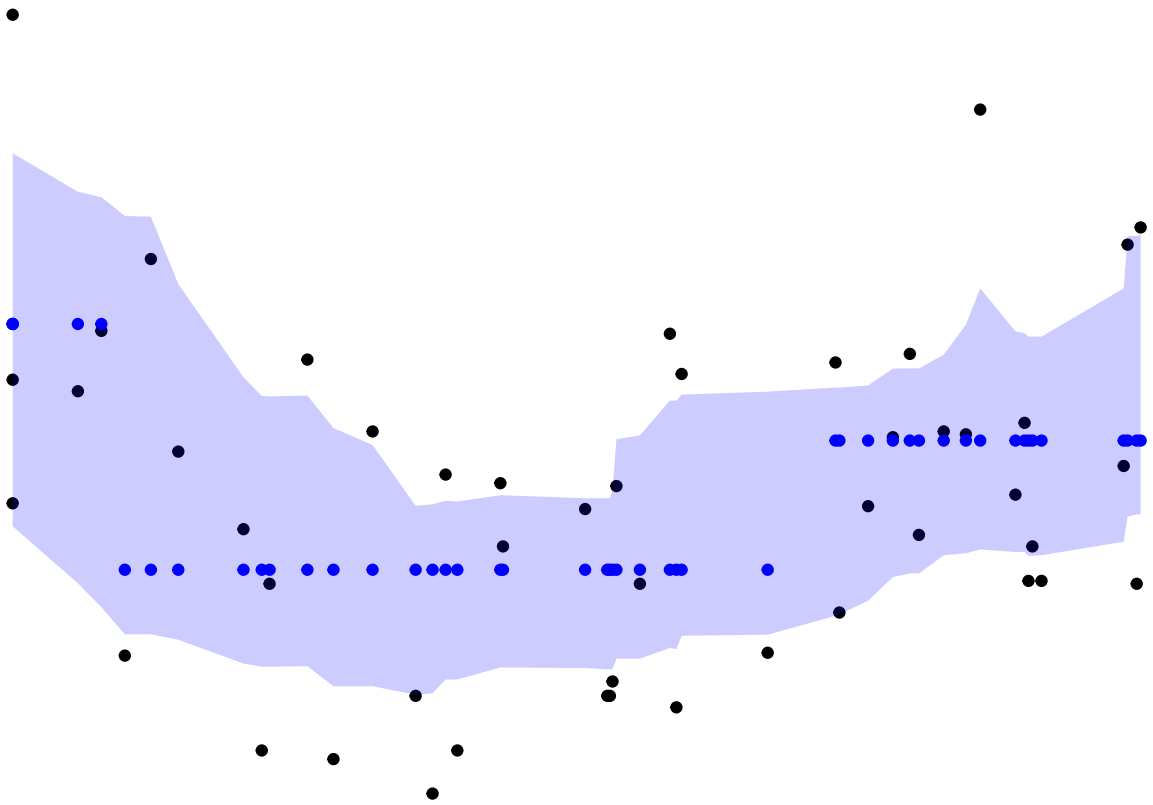
1.3

```
tree.fun <- function(data, ind){

  data <- data[ind,] #shuffle procedure
  trio <- tree(EX ~ MET, data = data,  #fitting
               control =
                 tree.control( minsize = 8, nobs = nrow(data))
               )
  trio <- prune.tree(trio, best = 3)
  return( predict(trio, newdata = State) )

}

set.seed(12345)
tree.boot <- boot(State, tree.fun, R = 1000)
```

```
## Warning in prune.tree(trio, best = 3): best is bigger than tree size
```

```
#Creates a data.frame used for plotting.
State.plot <- data.frame(lower =envelope(tree.boot)$point[2,])
State.plot$upper <-  envelope(tree.boot)$point[1,]
State.plot$predicted<-predict(state.prune)
```

```
State.plot$EX <- State$EX
State.plot$MET <- State$MET

ggplot(data = State.plot) +
  geom_point(aes(x = MET, y = EX)) +
  geom_point(aes(x = MET, y = predicted), col = "blue") +
  geom_ribbon(aes(x = MET ,ymin = lower, ymax = upper), alpha = 0.2, fill = "blue") + theme_void()
```

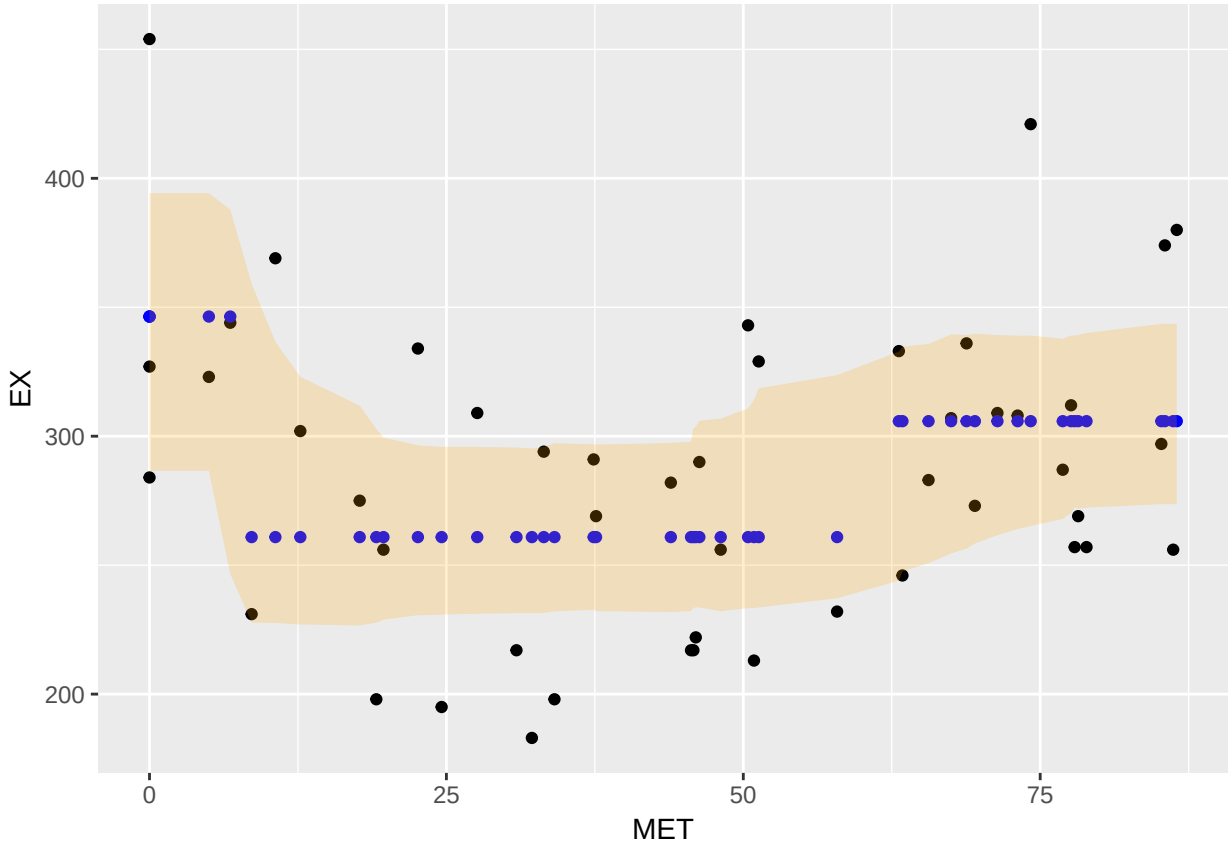


Yes the model fitted in 1.2 seem resonable as all fitted values are inside the confidence-band. The confidence band seem to vary in size over the data and is a bit jumpy.

1.3

```
myrng <- function(data, model) {
  data1<-data.frame(EX = data$EX, MET = data$MET)
  n<-nrow(data1)
  data1$EX <- rnorm(n,predict(model, newdata=data1),sd(resid(model)))
  return(data1)
}

para.fun <- function(data1){

  trio <- tree(EX ~ MET, data = data1,
               control =
                 tree.control( minsize = 8, nobs = nrow(data1)))
  trio <- prune.tree(trio, best = 3)
    #predict values for all Area values from the original data
```

```
  priceP <- predict(trio,newdata=data1)
  return(priceP)
}

set.seed(12345)
tree.boot.para <- boot(State, statistic = para.fun, R=1000,
            mle=state.prune ,ran.gen = myrng, sim="parametric")

State.plot$lower2 <-  envelope(tree.boot.para)$point[2,]
State.plot$upper2 <-  envelope(tree.boot.para)$point[1,]

ggplot(data = State.plot) +
  geom_point(aes(x = MET, y = EX)) +
  geom_point(aes(x = MET, y = predicted), col = "blue") +
  geom_ribbon(aes(x = MET ,ymin = lower2, ymax = upper2), alpha = 0.2, fill = "orange")
```



The parametric bootstrap confidenceband is alot smoother and thinner than the non-parametric version and should be prefered over the non-parametric in this case.

```
para.fun.p <- function(data1){

  trio <- tree(EX ~ MET, data = data1,
               control =
                 tree.control( minsize = 8, nobs = nrow(State)))
  trio <- prune.tree(trio, best = 3)
  #predict values for all Area values from the original data
  priceP <- predict(trio,newdata=State)
```

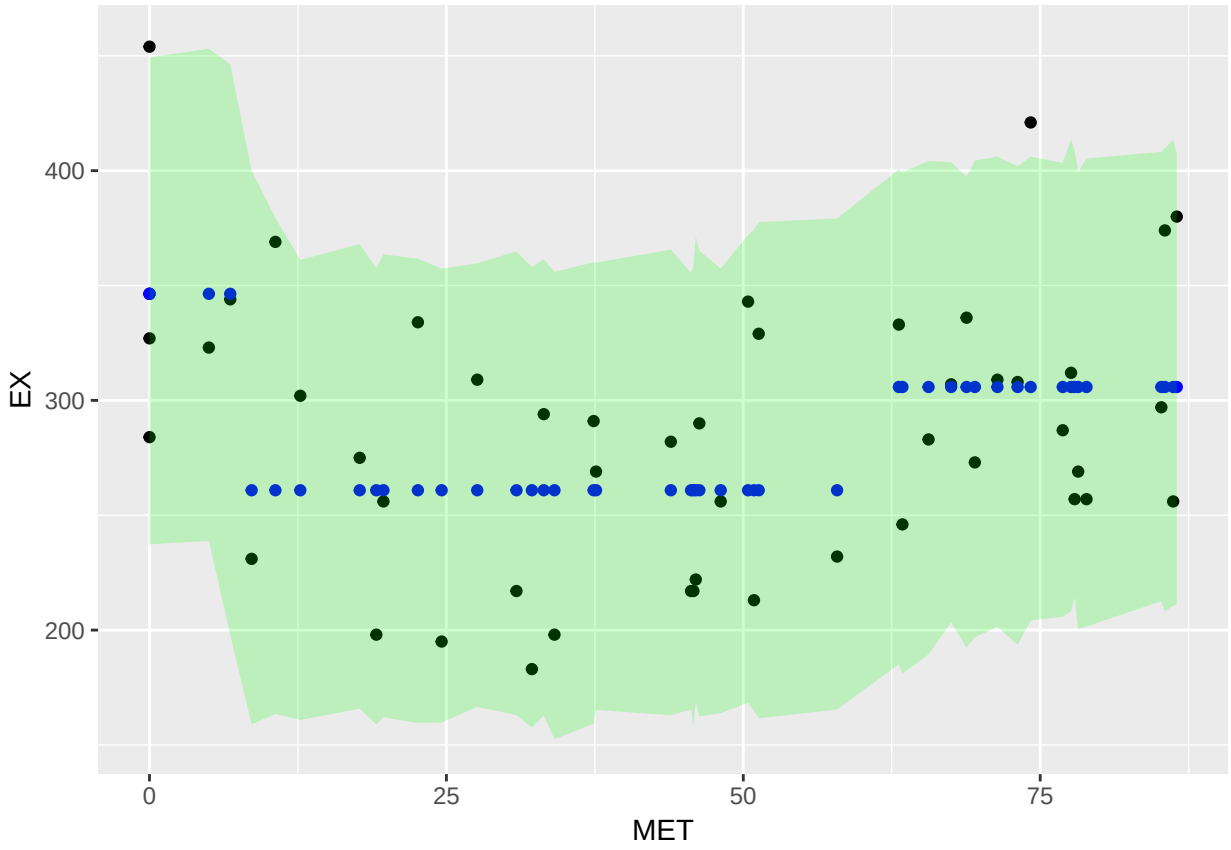
```
  priceP <- rnorm(n = nrow(data1), mean = priceP, sd = sd(resid(trio)))
  return(priceP)
}

set.seed(12345)
tree.boot.para.p <- boot(State, statistic = para.fun.p, R=1000,
                         mle=state.prune ,ran.gen = myrng, sim="parametric")

State.plot$lower3 <-  envelope(tree.boot.para.p)$point[2,]
```

```
## Warning in envelope(tree.boot.para.p): unable to achieve requested overall
## error rate
```

```
State.plot$upper3 <-  envelope(tree.boot.para.p)$point[1,]
```

```
## Warning in envelope(tree.boot.para.p): unable to achieve requested overall
## error rate
```

```
ggplot(data = State.plot) +
  geom_point(aes(x = MET, y = EX)) +
  geom_point(aes(x = MET, y = predicted), col = "blue") +
  geom_ribbon(aes(x = MET ,ymin = lower3, ymax = upper3), alpha = 0.2,fill ="green")
```



The prediction band seems to cover all but two observations, which is about 5 % of the data. This is resonable since it is a 95 % prediction band.

1.5

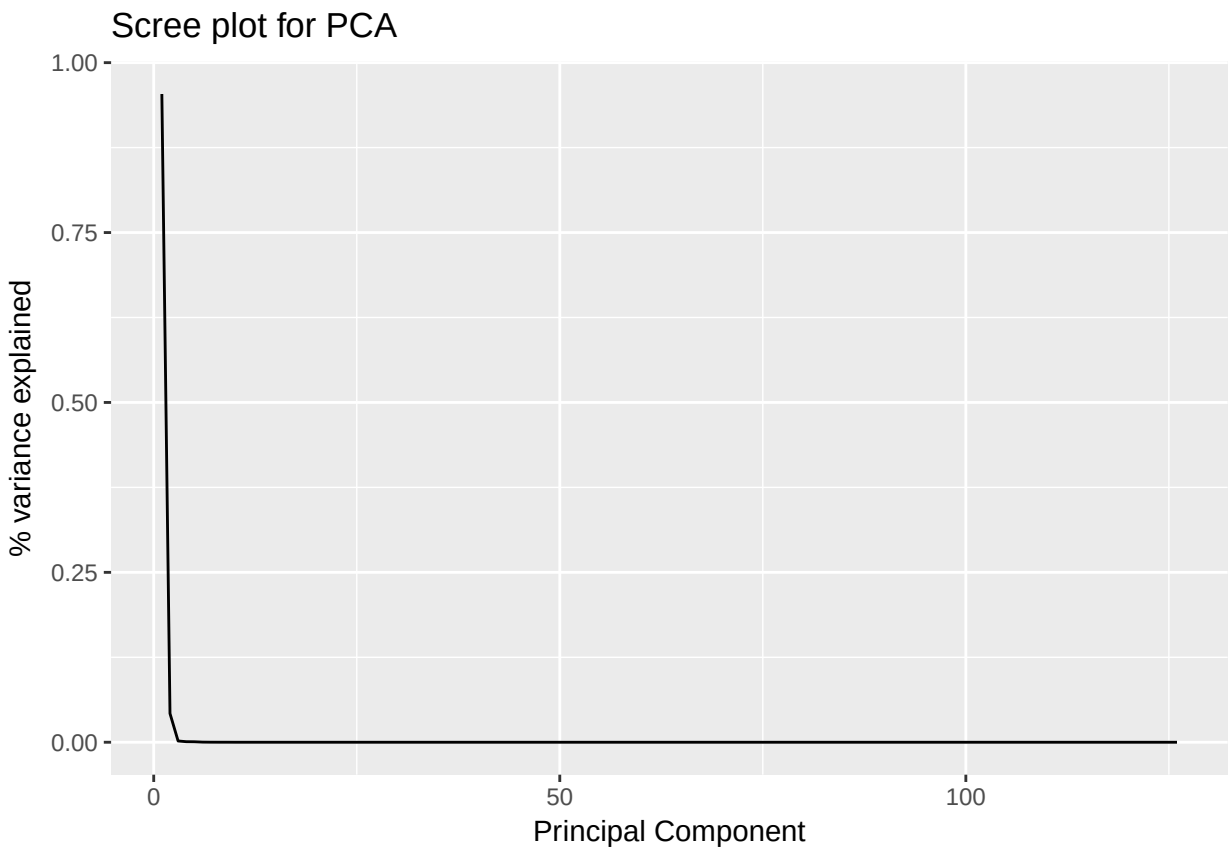
In this case i think it is resonable to use the parametric one even though it might be a bit of a stretch to say it normaly distributed but as seen in this example it works.

Assignment 2

2.1

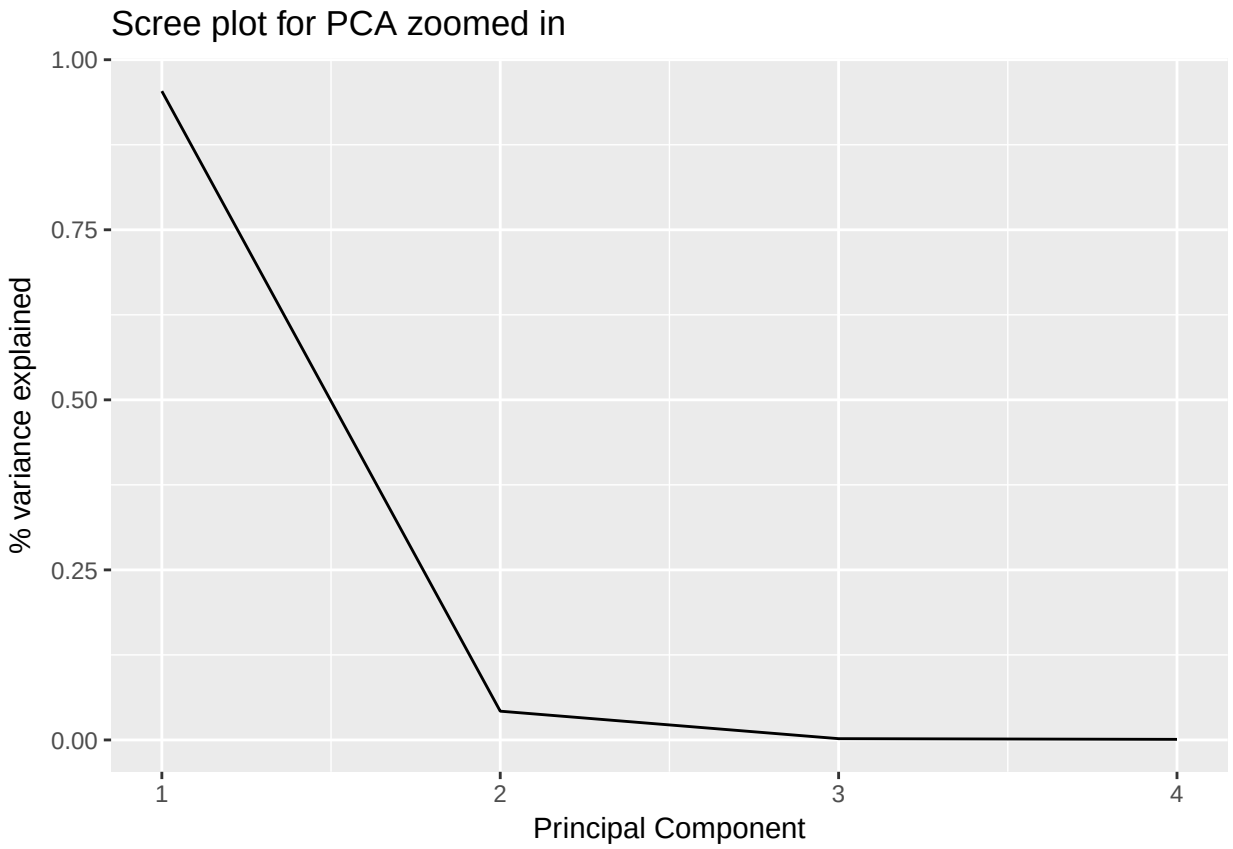
Instead of using a barplot I use a line-plot since it more clearly shows the variation explained, it is also common practise in PCA.

```
NIR<-read.csv2("NIRspectra.csv")

pca.nir <- prcomp(NIR[,-127], scale = TRUE)

pca.varexplaind<- pca.nir$sdev^2/sum(pca.nir$sdev^2)

pca.eigens <-  pca.nir$sdev^2

ggplot() + geom_line(aes(x = 1:length(pca.varexplaind), y = pca.varexplaind))  +
  labs(title="Scree plot for PCA",x="Principal Component", y = "% variance explained")
```



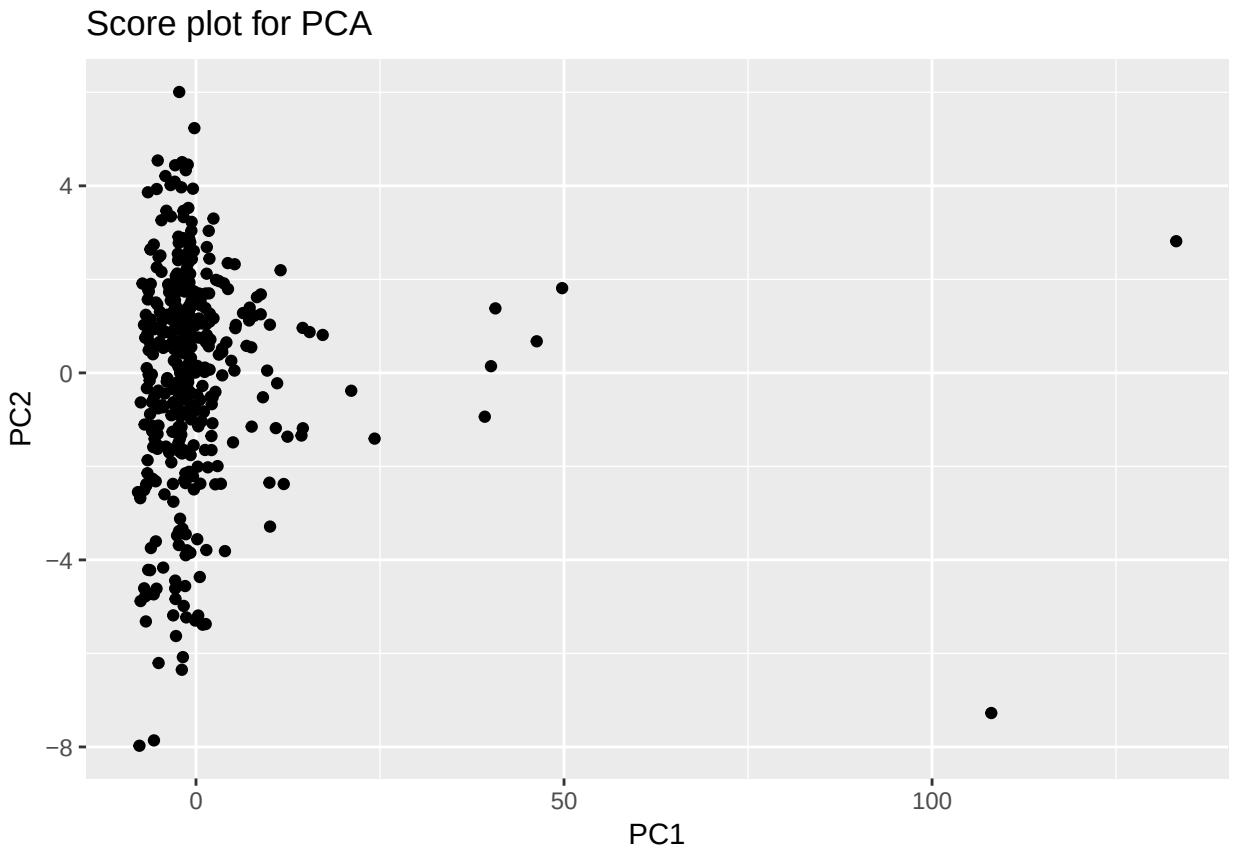
There is a quick drop off in the number of PCA-components that explain the variance. The elbow shape is a good way to determine which should be kept, in this case only PC1 is needed. But to get to 99 % of the variance exp PC2 is also needed.

```
ggplot() + geom_line(aes(x = 1:4, y = pca.varexplaind[1:4])) +
  labs(title="Scree plot for PCA zoomed in",x="Principal Component", y = "% variance explained")
```



A closer look at the drop of for the 4 first PCA components. Here the elbow shape is more visible as we have less variables plotted.

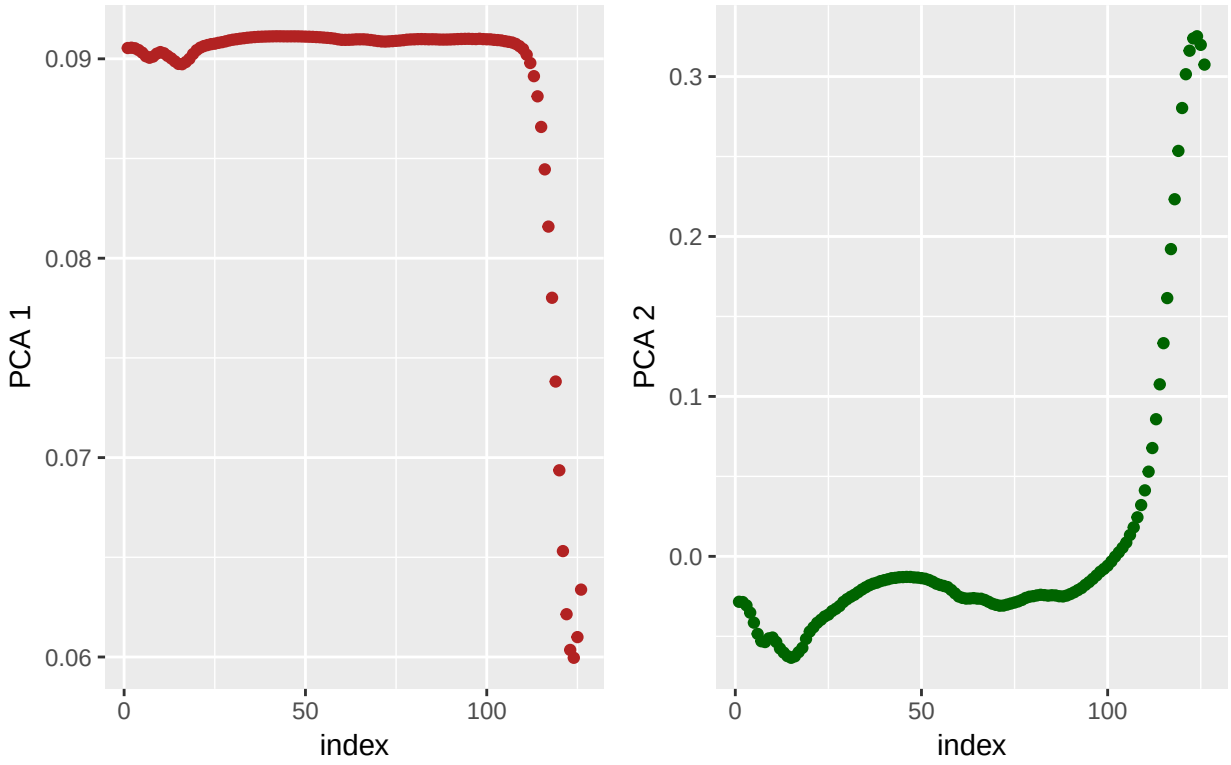
```
scores<-data.frame(pca.nir$x[,1:2])
ggplot(data = scores) + geom_point(aes(x = PC1, y = PC2))  +
  labs(title="Score plot for PCA",x="PC1", y = "PC2")
```



There seem to be two extreme outliers in the PC1 which have scores over 100, these are no outliers in the PC2 scores.

2.2

```
eigenvectors <- as.data.frame(pca.nir$rotation[,1:2])
colnames(eigenvectors) <- paste0("pca",1:2)

pP<-ggplot(data = eigenvectors) + geom_point(aes(x = 1:126, y = pca1), col = "firebrick") +
  labs(x= "index", y = "PCA 1", title = "")

pP2<-ggplot(data = eigenvectors) + geom_point(aes(x = 1:126, y = pca2), col = "darkgreen") +
  labs(x= "index", y = "PCA 2", title = "")

gRid<-grid.arrange(pP,pP2,ncol = 2, top ="PCA Trace plots ")
grid.newpage()
grid.draw(gRid)
```



The trace plots shows that the first principal component gives a lot of weight to around the 110 first variables observations and after there is a quick drop of in what the last 17 variables explain. In the second PC the situation is revesed and the last 17 variables are given more weight.

2.3

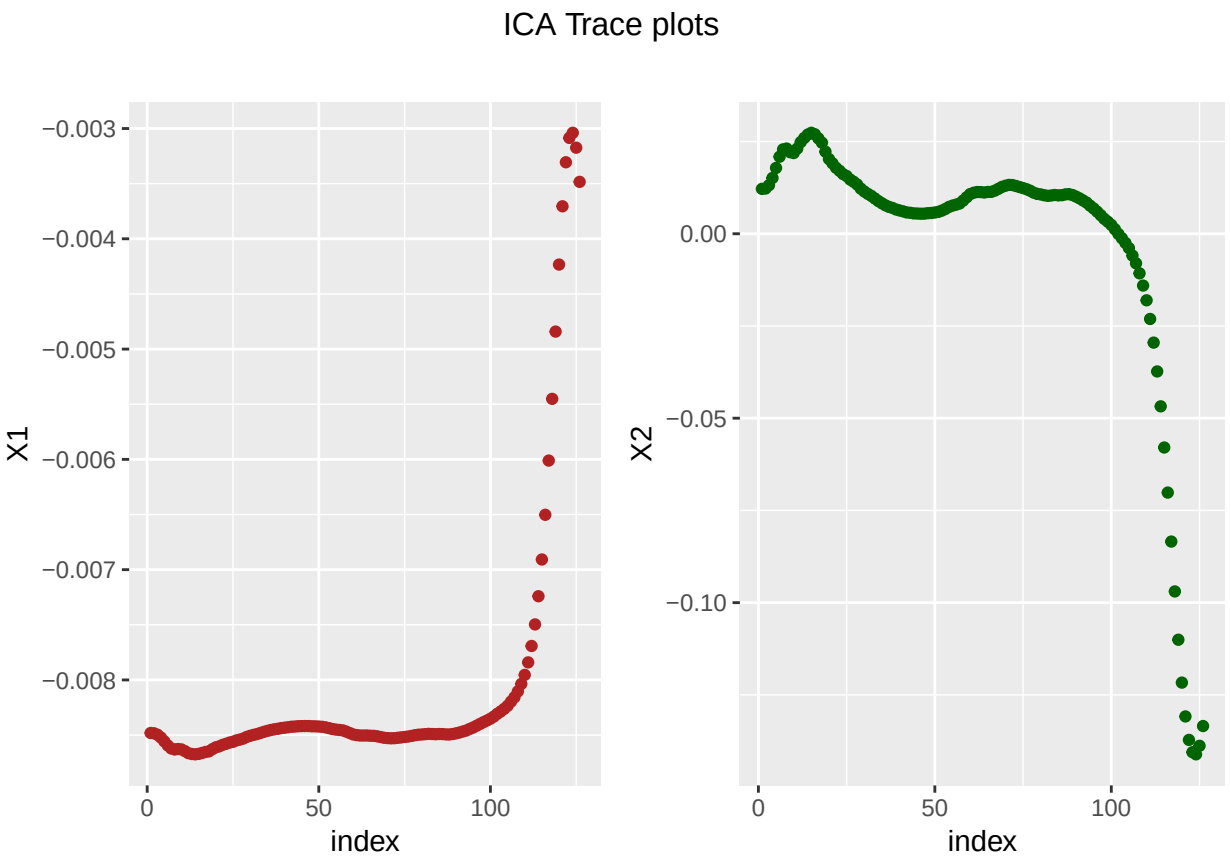
a)

```
set.seed(12345)
slowCoop<- fastICA(scale(NIR[,-127]), n.comp = 2, alg.typ = "parallel", fun = "logcosh")

Wprim <- data.frame(slowCoop$K %*% slowCoop$W)
slowCoop$W
```

```
##           [,1]       [,2]
## [1,] 0.9998508  0.0172753
## [2,] 0.0172753 -0.9998508
```

```
pX1 <- ggplot(data = Wprim) + geom_point(aes(x = 1:126, y = X1), col = "firebrick") +
  labs(x= "index", y = "X1", title = "")

pX2 <- ggplot(data = Wprim) + geom_point(aes(x = 1:126, y = X2), col = "darkgreen") +
  labs(x= "index", y = "X2", title = "")

gRid<-grid.arrange(pX1,pX2,ncol = 2, top ="ICA Trace plots ")
grid.newpage()
```

```
grid.draw(gRid)
```

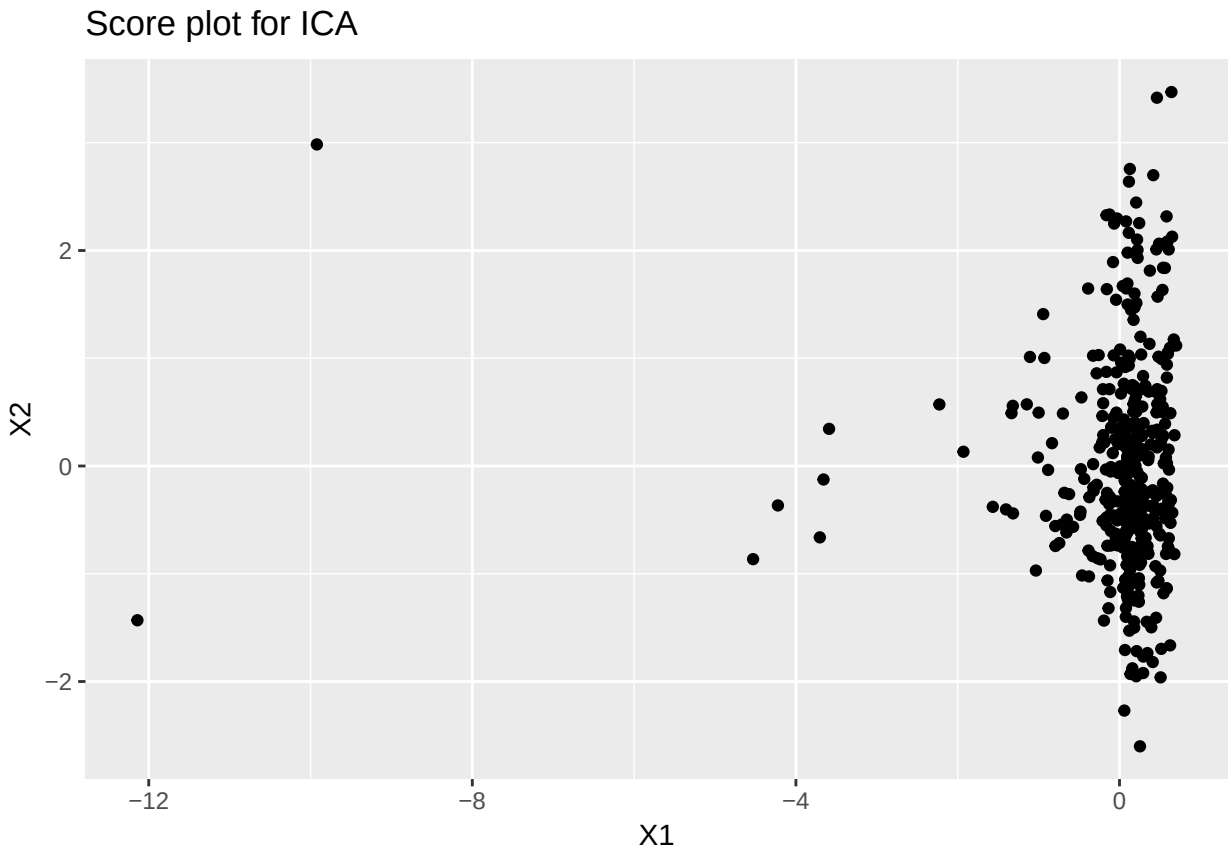


The W' is the ICA loadings so its the equivalent of the eigenvectors in the PCA.

These Looks similar to the PCA trace-plots but flipped. This is because ICA and PCA has different vectors that try to explain the dataset. The PCA tries to maximise the variance with its vectors and there for draws the vectors in a different way than the ICA. In the IC2 (X2) there are some loadings that are zero and therefor these variables are explained by the other variables.

b)

```
S <- data.frame(slowCoop$S[,1:2])
ggplot(data = S) + geom_point(aes(x = X1 , y = X2)) +
  labs(title="Score plot for ICA")
```



The results seem to be very similar to the PCA-plot, but here the first component is negative instead of positive which gives it a mirrored result.

2.4

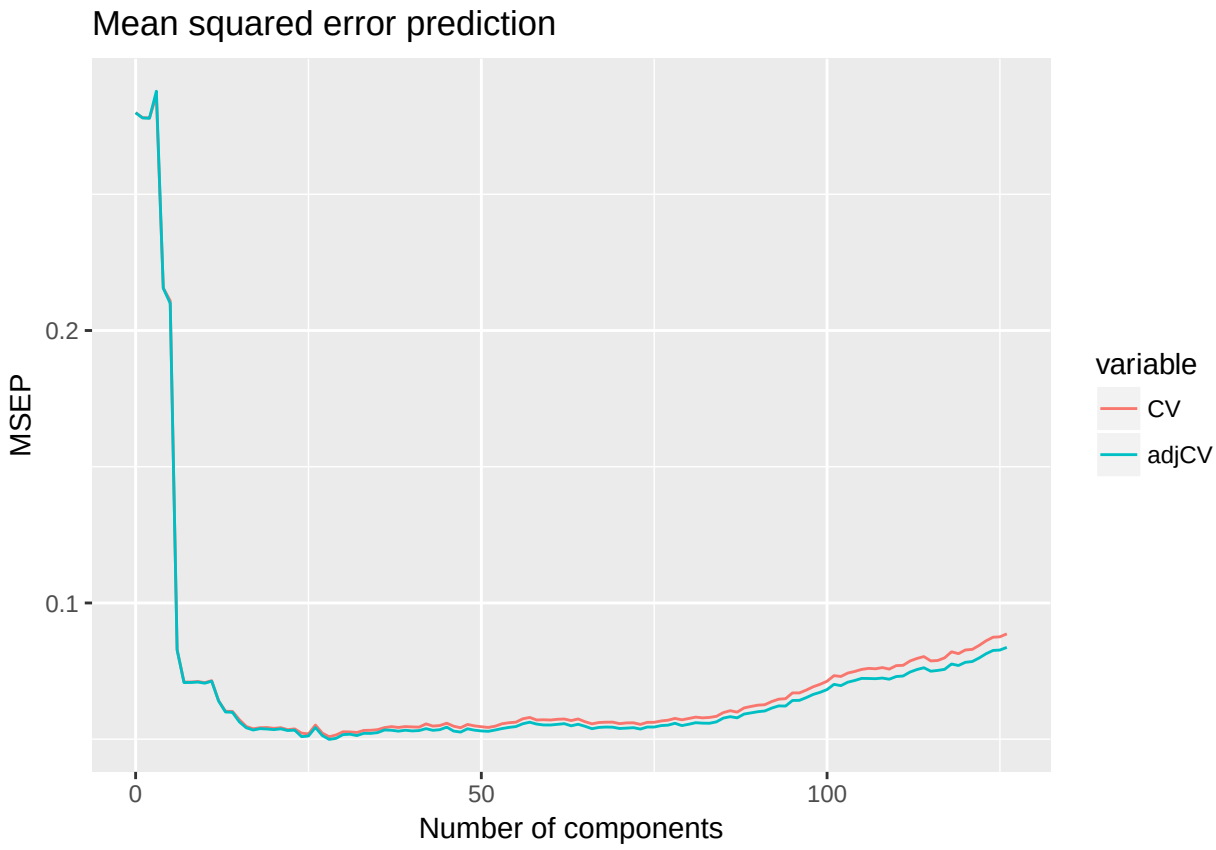
```
library(pls)
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
set.seed(12345)
mypcr <- pcr(Viscosity~.,data = NIR, scale = TRUE)
set.seed(12345)
cv.mypcr<- crossval(mypcr)

#str(mypcr)
#plot(MSEP(cv.mypcr))

cvMSEP<- MSEP(cv.mypcr)

msep<-data.frame(t(matrix(cvMSEP$val,nrow = 2)))
msep$nocomp <- cvMSEP$comps
colnames(msep) <- c("CV","adjCV","nocomp")
```

```
library(reshape2)
smelt<-melt(msep,id = "nocomp")

ggplot(smelt,aes(x = nocomp, y = value, col=variable)) + geom_line() +
  labs(title="Mean squared error prediction",x="Number of components", y = "MSEP")
```



It's resonable to select either 7 or around 17 components to have in the model as after that the number of components added does not contribute to a lower MSEP.

Code

```
knitr::opts_chunk$set(echo = TRUE)

library(ggplot2)
library(tree)
library(boot)
library(grid)
library(fastICA)
library(gridExtra)


State <- read.csv2("State.csv")
State <- State[order(State$MET),]

a11 <- ggplot(data = State, aes(x = MET,y = EX)) + geom_point()
plot(a11)
## Har eg. ingen koll alls på det här.
state.tree <- tree(EX ~ MET, data = State, minsize = 8 , split = "deviance")

#plot(state.tree)

set.seed(12345)
plot(cv.tree(state.tree))
# three is the optimal tree.

state.prune <- prune.tree(state.tree, best=3)

plot(state.prune)
text(state.prune)




a12 <- a11 + geom_point( y = predict(state.prune), col = "blue" )
plot(a12)
ggplot(data = data.frame(x = resid(state.prune))) +
  geom_histogram(aes(x = x), bins  = 10)


tree.fun <- function(data, ind){

  data <- data[ind,] #shuffle procedure
  trio <- tree(EX ~ MET, data = data,  #fitting
               control =
                 tree.control( minsize = 8, nobs = nrow(data))
               )
  trio <- prune.tree(trio, best = 3)
  return( predict(trio, newdata = State) )

}

set.seed(12345)
```

```
tree.boot <- boot(State, tree.fun, R = 1000)


#Creates a data.frame used for plotting.
State.plot <- data.frame(lower =envelope(tree.boot)$point[2,])
State.plot$upper <-  envelope(tree.boot)$point[1,]
State.plot$predicted<-predict(state.prune)
State.plot$EX <- State$EX
State.plot$MET <- State$MET

ggplot(data = State.plot) +
  geom_point(aes(x = MET, y = EX)) +
  geom_point(aes(x = MET, y = predicted), col = "blue") +
  geom_ribbon(aes(x = MET ,ymin = lower, ymax = upper), alpha = 0.2, fill = "blue") + theme_void()


myrng <- function(data, model) {
  data1<-data.frame(EX = data$EX, MET = data$MET)
  n<-nrow(data1)
  data1$EX <- rnorm(n,predict(model, newdata=data1),sd(resid(model)))
  return(data1)
}

para.fun <- function(data1){

  trio <- tree(EX ~ MET, data = data1,
               control =
                 tree.control( minsize = 8, nobs = nrow(data1)))
  trio <- prune.tree(trio, best = 3)
    #predict values for all Area values from the original data
  priceP <- predict(trio,newdata=data1)
  return(priceP)
}

set.seed(12345)
tree.boot.para <- boot(State, statistic = para.fun, R=1000,
            mle=state.prune ,ran.gen = myrng, sim="parametric")

State.plot$lower2 <-  envelope(tree.boot.para)$point[2,]
State.plot$upper2 <-  envelope(tree.boot.para)$point[1,]

ggplot(data = State.plot) +
  geom_point(aes(x = MET, y = EX)) +
  geom_point(aes(x = MET, y = predicted), col = "blue") +
  geom_ribbon(aes(x = MET ,ymin = lower2, ymax = upper2), alpha = 0.2, fill = "orange")



para.fun.p <- function(data1){

  trio <- tree(EX ~ MET, data = data1,
               control =
                 tree.control( minsize = 8, nobs = nrow(State)))
```

```
  trio <- prune.tree(trio, best = 3)
  #predict values for all Area values from the original data
  priceP <- predict(trio,newdata=State)
  priceP <- rnorm(n = nrow(data1), mean = priceP, sd = sd(resid(trio)))
  return(priceP)
}

set.seed(12345)
tree.boot.para.p <- boot(State, statistic = para.fun.p, R=1000,
                         mle=state.prune ,ran.gen = myrng, sim="parametric")

State.plot$lower3 <-  envelope(tree.boot.para.p)$point[2,]
State.plot$upper3 <-  envelope(tree.boot.para.p)$point[1,]

ggplot(data = State.plot) +
  geom_point(aes(x = MET, y = EX)) +
  geom_point(aes(x = MET, y = predicted), col = "blue") +
  geom_ribbon(aes(x = MET ,ymin = lower3, ymax = upper3), alpha = 0.2,fill ="green")



NIR<-read.csv2("NIRspectra.csv")

pca.nir <- prcomp(NIR[,-127], scale = TRUE)

pca.varexplaind<- pca.nir$sdev^2/sum(pca.nir$sdev^2)

pca.eigens <-  pca.nir$sdev^2

ggplot() + geom_line(aes(x = 1:length(pca.varexplaind), y = pca.varexplaind))  +
  labs(title="Scree plot for PCA",x="Principal Component", y = "% variance explained")


ggplot() + geom_line(aes(x = 1:4, y = pca.varexplaind[1:4])) +
  labs(title="Scree plot for PCA zoomed in",x="Principal Component", y = "% variance explained")


scores<-data.frame(pca.nir$x[,1:2])
ggplot(data = scores) + geom_point(aes(x = PC1, y = PC2))  +
  labs(title="Score plot for PCA",x="PC1", y = "PC2")


eigenvectors <- as.data.frame(pca.nir$rotation[,1:2])
colnames(eigenvectors) <- paste0("pca",1:2)

pP<-ggplot(data = eigenvectors) + geom_point(aes(x = 1:126, y = pca1), col = "firebrick") +
  labs(x= "index", y = "PCA 1", title = "")

pP2<-ggplot(data = eigenvectors) + geom_point(aes(x = 1:126, y = pca2), col = "darkgreen") +
  labs(x= "index", y = "PCA 2", title = "")

gRid<-grid.arrange(pP,pP2,ncol = 2, top ="PCA Trace plots ")
grid.newpage()
```

```r
grid.draw(gRid)

set.seed(12345)
slowCoop<- fastICA(scale(NIR[,-127]), n.comp = 2, alg.typ = "parallel", fun = "logcosh")

Wprim <- data.frame(slowCoop$K %*% slowCoop$W)
slowCoop$W


pX1 <- ggplot(data = Wprim) + geom_point(aes(x = 1:126, y = X1), col = "firebrick") +
  labs(x= "index", y = "X1", title = "")

pX2 <- ggplot(data = Wprim) + geom_point(aes(x = 1:126, y = X2), col = "darkgreen") +
  labs(x= "index", y = "X2", title = "")

gRid<-grid.arrange(pX1,pX2,ncol = 2, top ="ICA Trace plots ")
grid.newpage()
grid.draw(gRid)

S <- data.frame(slowCoop$S[,1:2])
ggplot(data = S) + geom_point(aes(x = X1 , y = X2)) +
  labs(title="Score plot for ICA")
library(pls)
set.seed(12345)
mypcr <- pcr(Viscosity~.,data = NIR, scale = TRUE)
set.seed(12345)
cv.mypcr<- crossval(mypcr)

#str(mypcr)
#plot(MSEP(cv.mypcr))

cvMSEP<- MSEP(cv.mypcr)

msep<-data.frame(t(matrix(cvMSEP$val,nrow = 2)))
msep$nocomp <- cvMSEP$comps
colnames(msep) <- c("CV","adjCV","nocomp")

library(reshape2)
smelt<-melt(msep,id = "nocomp")

ggplot(smelt,aes(x = nocomp, y = value, col=variable)) + geom_line() +
  labs(title="Mean squared error prediction",x="Number of components", y = "MSEP")
```