Projekt 1

Filip Axelsson & Julia Holmgren

2021-02-02

Uppgift 1

Det första som görs är att datasetet laddas ner och undersöks. a) Datasetet innehåller 506 obseravtioner och 14 variabler varav *medv* är en responsvariabel. Ett utdrag på data visualiseras i *Tabell 1*.

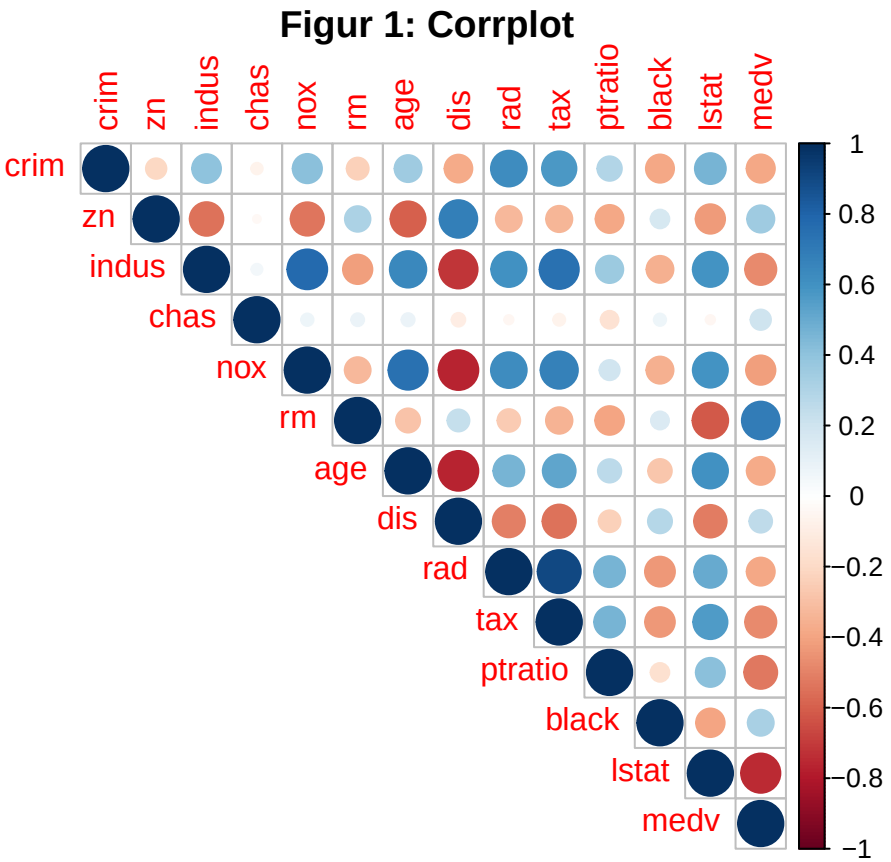
```
### Förbered data
data(Boston)
# glimpse(Boston)
set.seed(123) # Sätt ett seed
data <- Boston
index <- sample(1:nrow(data), round(0.75 * nrow(data))) # Fördelrar in 75/25 split, train och test.
train <- data[index, ]
test <- data[-index, ]

data %>%
  head(5) %>%
  knitr::kable(
    caption = "Tabell 1: Utdrag från data"
  )
```

Table 1: Tabell 1: Utdrag från data

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

b) Skapar en correlationsplot för att undersöka vilka variabler som korrelarar med varandra.

```
### Heatmap
corr_matrix <- cor(train)
corrplot(corr_matrix, type = "upper", title = "Figur 1: Corrplot",  mar=c(0,0,1,0))
```

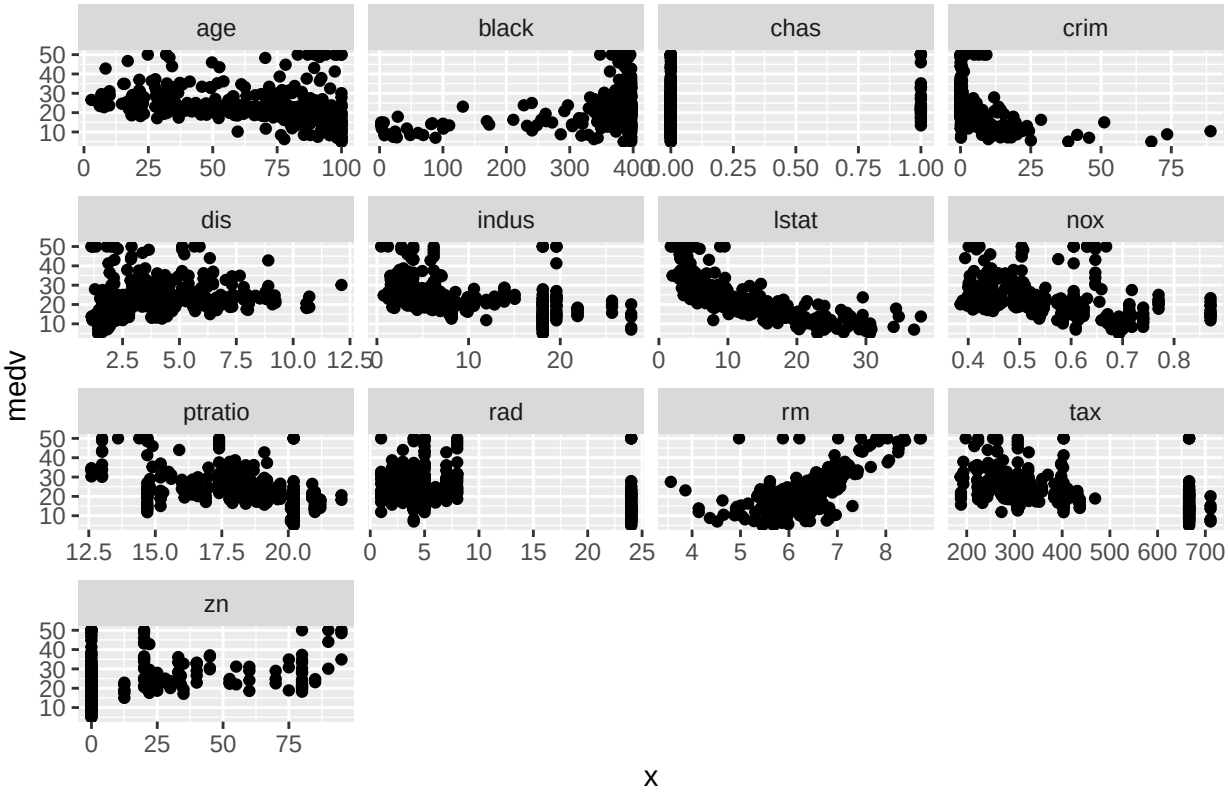


Enligt *Figur 1* kan det avläsas att variablerna *indus* och *nox* har mest korrelation med de andra förklaringsvariaberna. Det går även att avläsa att *chas* är den variabel med lägst korrelation. Variablerna *crim*, *ptratio* och *black* har en låg korrelation.

Härnäst plotas punktploter för att undersöka om förklarade vaiablerna har något speciellt sanband med responsvariaben *medv*.

```
### Pairwise plots
train %>%
  pivot_longer(-medv, values_to = "x") %>%
  ggplot(aes(x = x, y = medv)) +
  geom_point() +
  # stat_smooth() +
  facet_wrap(~name, scales = "free_x") +
  ggtitle("Figur 2: Parvisa punkt-plotter")
```

Figur 2: Parvisa punkt−plotter



Genom att analysera *Figur 2* kan det observera att *lstat*, *rm* och *crim* har de tydligaste sambanden med responsvariabeln *medv*.

c) Koden nedanför delar upp datasetet i train-set och test-set, det görs en 75/25-split.

```
index <- sample(1:nrow(data), round(0.75 * nrow(data))) # Fördelrar in 75/25 split, train och test.
train <- data[index, ]
test <- data[-index, ]
```

Uppgift 2

a)
Genom att antag en datamängd $D = \{(x_i, y_i), i = 1, 2, \ldots, n\}$ där $x_i = (x_{i1}, \ldots, x_{id}) \in \mathbb{R}^d$ och $y_i \in \mathbb{R}$. Antag även att $y = f_\theta(x_i) + \epsilon_i$, där ϵ_i är obereonde och normalfördelad, $\epsilon_i \sim N(0, \sigma^2)$. $f_\theta(x_i) = \alpha + \beta_1 x_{i1} + \ldots + \beta_d x_{id}$ kallas linjär regression. Detta leder till att vi nu vill välja den fördelning som bäst passar in på data, detta kan göras med hjälp utav maximum-likelihood principen, $(\hat{\theta}, \hat{\sigma}^2) = \underset{(\theta,\sigma^2)}{\text{argmax}} = \mathbb{P}(y_1, \ldots, y_n | \theta, \sigma^2)$ dock är det lättare att minimiera log-likelihooden.

$$log(\mathbb{P}(y_1, \ldots, y_n | \theta, \sigma^2)) = \sum_{i=1}^{n} log(\mathbb{P}(y_i | \theta, \sigma^2)) = \ldots = -\frac{n}{2} log(2\pi) - \frac{n}{2} log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$$

$$\Leftrightarrow$$

$$\hat{\theta} = \underset{\theta}{\text{argmin}} = \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$$

b) Genom nedanstående kod, anpassas en linjär regression där samtliga variabeler anvnäds för att förklara repsonsvaribeln *medv*.

```r
### Linjär regression
lm.fit <- glm(medv ~ ., data = Boston)
summary(lm.fit)
```

```
##
## Call:
## glm(formula = medv ~ ., data = Boston)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -15.595   -2.730   -0.518    1.777   26.199
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
## chas         2.687e+00  8.616e-01   3.118 0.001925 **
## nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
## rm           3.810e+00  4.179e-01   9.116  < 2e-16 ***
## age          6.922e-04  1.321e-02   0.052 0.958229
## dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
## tax         -1.233e-02  3.760e-03  -3.280 0.001112 **
## ptratio     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
## black        9.312e-03  2.686e-03   3.467 0.000573 ***
## lstat       -5.248e-01  5.072e-02 -10.347  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 22.51785)
##
##     Null deviance: 42716  on 505  degrees of freedom
## Residual deviance: 11079  on 492  degrees of freedom
## AIC: 3027.6
##
## Number of Fisher Scoring iterations: 2
```

```r
pred.lm <- predict(lm.fit, test)
```

c)

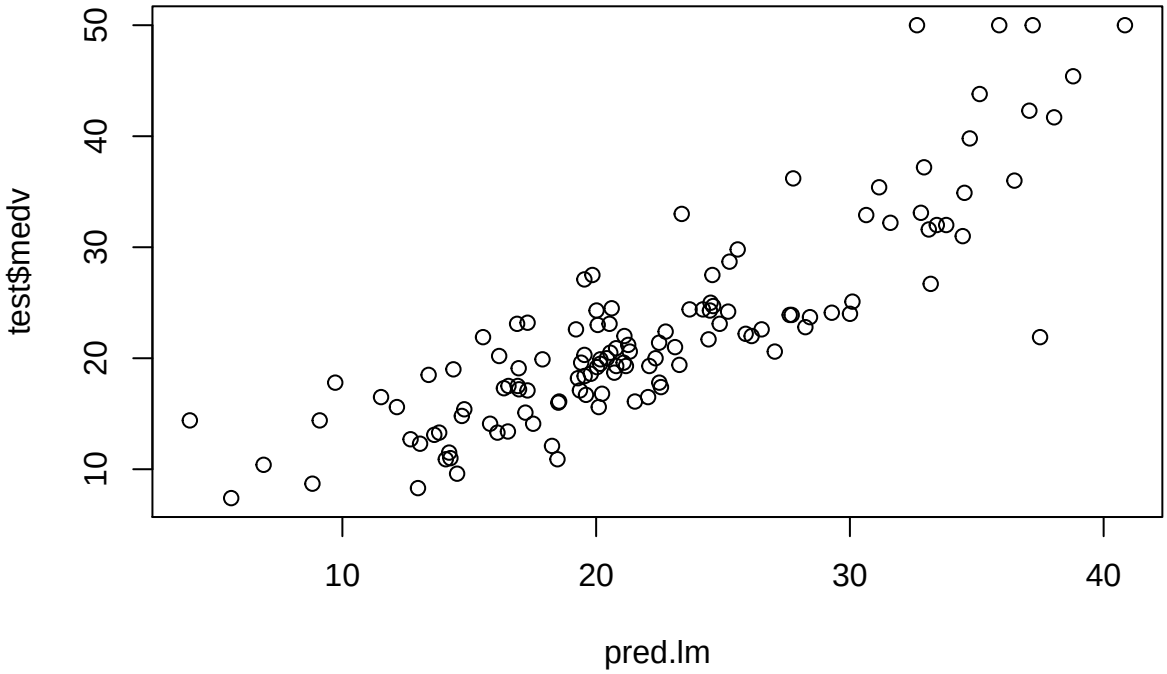
Genom att använda sig av lossfunktionen MSE, där de stora felen dominerar optimeringen kan modeller jämföras och en eventuell overfitting kan undivkas. I detta fall med hjälp av koden nedanför, generades en på $MSE = 22.1962131$

```r
### Funktion för medelkvardratfelet
mse <- function(actual, predicted) {
  mean((actual - predicted)^2)
}
  first_model <- mse(actual = test$medv, predicted = pred.lm)
```

d) Genom att plotta de predikterade värderna mot de observarande kan det enkelt analyseras att ett linjär samband finns, vilket även är något som bör föreligga om modellen är korret. Detta kan visualiseras i *Figur 3* nedanför.

```r
### Plot Prediktion vs observerade vä
plot(pred.lm, test$medv, main = "Figur 3: Prediktion vs observerade ")
```

Figur 3: Prediktion vs observerade



e)

```
##
## Call:
## glm(formula = medv ~ . - indus - age, data = Boston)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -15.5984   -2.7386   -0.5046    1.7273   26.2373
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  36.341145   5.067492   7.171 2.73e-12 ***
## crim         -0.108413   0.032779  -3.307 0.001010 **
## zn            0.045845   0.013523   3.390 0.000754 ***
## chas          2.718716   0.854240   3.183 0.001551 **
## nox         -17.376023   3.535243  -4.915 1.21e-06 ***
## rm            3.801579   0.406316   9.356  < 2e-16 ***
## dis          -1.492711   0.185731  -8.037 6.84e-15 ***
## rad           0.299608   0.063402   4.726 3.00e-06 ***
## tax          -0.011778   0.003372  -3.493 0.000521 ***
## ptratio      -0.946525   0.129066  -7.334 9.24e-13 ***
## black         0.009291   0.002674   3.475 0.000557 ***
## lstat        -0.522553   0.047424 -11.019  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for gaussian family taken to be 22.43191)
##
##     Null deviance: 42716  on 505  degrees of freedom
## Residual deviance: 11081  on 494  degrees of freedom
## AIC: 3023.7
##
## Number of Fisher Scoring iterations: 2
```

Genom att använda sig av backward-elimination, det vill säga att utgå från en modell där samtliga variabler ingår i modellen och sedan eliminera en variabel i taget. Utval av variabel sker genom att ta de som inte är signifikanta, detta resulterade i att variabelerna *age* och *indus* elimninerades, vilket gav en ny modell än den som beksrev i fråga a). Genom att ha eliminerat två variabler fås en ny MSE vilket resulterade i följande värde: 22.2236268. Detta värde skiljer sig inte mycket ifrån första modellen, dock har modelkomplixiteten minskat vilket är något att föredra.

```
lm.fit2 <- glm(medv ~ .-indus-age, data = Boston)
summary(lm.fit2)
pred.lm2 <- predict(lm.fit2, test)
second_model <- mse(actual = test$medv, predicted = pred.lm2)
plot(pred.lm2, test$medv)
```

Uppgift 3

a) *K-närmaste granne (knn) regression* är en oparametrisk algoritm som tar in träningsdata $D = (x_i, y_i)$ och försöker minimera funktionen $f(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$ där $N_k(x) =$ de k "närmaste" i D pukterna till x. Det vill säga till exempel Manhattan avståndet $d(x, y) = ||x - y|| = \sum_{i=1}^{n} |x_i - y_i|$. (Det går också att använda andra avståndsfunktioner som exempel Euclidean- och Miukowsi-avståndsfunktioner.) Det är vanligt att använda sig att Cross-validation för att hitta det K som minimerar funktionen.

b)

```
### K-närmaste granne regression
x_train <- train %>% dplyr::select(-medv)
x_test <- test %>% dplyr::select(-medv)
y_train <- train %>% dplyr::select(medv)
y_test <- test %>% dplyr::select(medv)



pred.knn <- knn.reg(train = x_train,
                    test = x_test,
                    y = y_train,
                    k = 10)$pred

k_10 <- mse(y_test$medv,pred.knn)
```

vid k = 10 ges ett MSE på 45.522123

c) I nästa koddel så utförs KNN för olika k som presenteras i tabell nedan med MSE för KNN.

```
MSE <- c()
for (i in 3:15) {
  predicted <- knn.reg(train = x_train,
                    test = x_test,
                    y = y_train,
                    k = i)$pred
```

```
  MSE <- append(MSE,mse(y_test$medv,predicted))

}
k <- c(3:15)

knitr::kable(cbind(k,MSE),
             caption = "Tabell 2: Visar olika modeller för KNN och respektive MSE")
```

Table 2: Tabell 2: Visar olika modeller för KNN och respektive MSE

k	MSE
3	41.64101
4	41.13232
5	42.47663
6	43.96065
7	44.28149
8	42.99609
9	44.17853
10	45.52212
11	45.62820
12	45.75783
13	45.12316
14	46.54363
15	46.48968

Utifrån *Tabell 2* kan det analyseras att när $k = 4$ ger det lägsta värdet av MSE, vilket även leder till att detta blir det mest lämpliga värdet på k.

d) När k är litet så är bias stort och variansen liten då k ökar så minskar bias och variansen ökar. modell-komplexiteten är stor då man har mycket data och många regioner så när k är litet är modell-komplexiteten stor. Ett optimalt k är ett som ger låg bias och låg varians. Vid låga k har man lågt träningsfel men även ett högt testfel. Det finns inte direkt någon förbestämd metod för att hitta det bästa värder för k, därav kan det vara lämpligt som har gjorts i detta fall ta fram en mängd olika värden (helst slumpmässiga) och jämföra de emellan varandra, som skrivits tidigare ett litet k kan leda till ostabila beslutsnivåer. I detta fall skulle vi säga att $k = 4$ är det mest optimala då det generade lägst MSE.

Uppgift 4

a) Ett neuralt nätverk är en algoritm som försöker efterlikna ett biologisk neuronnät. Genom att försöka hitta mönster i data. Detta görs genom ett nätverk som består av noder som bildar lager och i mellan lagerna finns det vikter.
Det första lagret kallas inputlagret där sätts data in. Sista lagret kallas för output de lagren imellan kallas för hidden layers. Vikterna och parametrana för noderna ändras stegvis för att optimera en lossfunktion som i detta fall är MSE.
Detta kan även defineras med hjälp matematik och görs genom följande:
Ett neutralt närverk med parametrar $\theta = \{(w^i, b^i), i = 1, 2 \dots l+1\}$ är en icke-linjär funktion f_θ som uppfyller:

- $f_\theta(x) = f^{l+1}(f'(\dots(f'(x)\dots)))$
- $h^0 = x, h^i = f^i(h^{i-1}) = g^i(w^i h^i + b^i)$ där g^i är en funktion som appliceras komponentvis på vektorn $w^i h^i + b^i$
- g^i är en activation function $\Rightarrow g^i$ tillhör en grupp funktioner.

b) Koden nedan tar in alla 13 variabler i Boston datan och anpassar en regression med hjälp av två hidden layers första med fem neuroner och andra med 3 neuroner.

```
#### Neuralt nätverk
f <- as.formula("medv ~ .")
nn <- neuralnet(formula = f,
                data = train,
                hidden = c(5,3))
pred.nn <- predict(nn, test)
nn_mse <-mse(test$medv,pred.nn)
```

Det nuerala nätverket gav en prediktion med MSE på 73.6533203 på testdata.

c) I nästa kodchunk så gör vi samma sak som innan fast skalar om boston-datan genom att normalisera den.

```
# Normalisera test data
f <- as.formula("medv ~ .")
train_scale <- scale(x_train) %>%
                as.data.frame() %>%
                dplyr::mutate(medv = y_train$medv)

test_scale <- scale(x_test,
                    center = colMeans(x_train),
                    scale = apply(x_train, 2, sd)) %>%
                as.data.frame() %>%
                dplyr::mutate(medv = y_test$medv)


nn_scale <-  neuralnet(formula = f,
                data = train_scale,
                hidden = c(5,3),
                threshold = 0.5)



pred.nn_scale <- predict(nn_scale, test_scale)

nn_scale_mse <- mse(test_scale$medv,pred.nn_scale)
```

I detta fall fick vi ett MSE på 23.4501902 som är mycket lägre än när datan inte var normaliserad. d) Härnäst plotar vi de neurala nätverket.

```
plot(nn_scale)
```

I *figur 4* ser vi hur alla variabler "skickar" signlar till alla neuroner där i sin tur skickar vidare signalen om den är stark nog. Vi ser det vill säga i svart ser vi varje lager och dess vikt i varje anslutning, i blått visas varje bias-term som är adderad för respektive steg. Det kan även noteras att detta är något som är lite mer svårtolkat jämfört med linjär regression. Nätverket brukar oftas kallas för "black box" eftersom det inte går att säga så mycket om anpassningen utifrån plotten.

e) I denna del försöker vi hitta ett neuralt nätverk som presterar bättre än de innan. Detta görs genom att skapa en for-loop som först går igenom alla möjliga kombinationer med 1-2 hidden layers där det är upp till 10 neuroner i respektive hidden layer. Vid fler hidden layers och samma max antal på neuroner skulle kräva en enorm kraft på datorn då antalet *stepmax* skulle behövas ökas och det skulle även ta tid. Vi ser resultatet över de 5 bästa modeller i *Tabell 3* , den modell som gav lägst MSE är med 2 hidden layers och 9 neuroner i första lagret respektive 2 det andra. Det kan vara värt att noteras att *threshold* har öktat från 0.5 till 3 detta på grund att det annars inte konvergerade. Denna kod körs inte på grund av att det kräver så mycket tid,

vilket är en nackdel vid träning av neurala nätverk.

```
# Skapar en funktion som kan kontrollera alla kombinationer av hidden layers och neuroner
combination_nn <- function(l,n){
  model <- c()
  mse_value <- c()
  for(k in 1:l){
    comb <- permutations(n,k) #skapar olika kombinationer med l hidden layers och 1-10 neuroner
    for(i in 1:nrow(comb)){
    layer <- comb[i,]
    nn_loop <- neuralnet(formula = f,
                data = train_scale,
                hidden = layer,
                threshold = 3) # anger threshold = 3 pga att det annars inte konvergerar.
    pred.nn_loop <- predict(nn_loop, test_scale)
    mse_value <-append(mse_value,c(mse(test_scale$medv,pred.nn_loop)))
    if(length(layer) >= 2){
    model <- append(model,paste(as.character(layer[1]),",",as.character(layer[2])))
    }
    else{
      model <- append(model, as.character(layer))
      }
    }
  }
return(as_tibble(cbind(model,mse_value)))
}

set.seed(1997)
combination_nn(2,10) %>%
  mutate(mse_value = as.numeric(mse_value)) %>%
  arrange(mse_value) %>%
  head(5) %>%
  knitr::kable(
    col.names = c("Modell", "MSE"),
    caption = "Tabell 3: De 5 bästa modeller för det neurala nätverket med lägst MSE"
  )
```

Table 3: Tabell 3: De 5 bästa modeller för det neurala nätverket med lägst MSE

Modell	MSE
9 , 2	12.51111
8	13.06585
2 , 9	13.25603
3 , 2	13.36203
7 , 3	13.61811

Uppgift 5

I denna raport så har tre metoder använvänds för regession. Den första var linjärregression som gav ett relativt lågt mse på 22 och väldigt tolkningsbar modell.
Därefter testades metoden KNN som endast beror på x värderna. Denna metod gav sämre MSE på 41. KNN är en icke-paramtisk model vilket är en skillnad mot linjär regression är en parametrisk. KNN är även en väldigt långsam metod mätt i tid, då det behövs hålla reda på all träningsdata och hitta "neighbors".
Den sista metod som testades var neurala nätverk vilket var den metod med bäst MSE på endast 10. Dock så är NN svårtolkad och tar längre tid att träna. Jämfört med linjär regression så är denna icke linjär i parametern θ samt icke linjärt beroende i y på x kan modelleras indirekt av modell. Medan i linjär regression måste detta göras explicit.
Detta gör att linjär regression är den metod som föredras att användas i denna rapport då den både är enkel att tolka och använda, men även gav ett relativt lågt MSE. Det ska inte glömmas bort att detta är en ganska enkel model, vilket gör att det finns risk för underfit.