Øvelse 2

Statistiske fordelinger, lineær regression, løkker og egne funktioner

Underviser: Troels Ankjær
Hjælpelærer: Nicklas Larsen

Introduktion til programmering i R
4 timers kursus i R
Oktober 2019

VIDEN DER STYRKER

Start

Øvelse 2 består af 3 større opgaver,

Formateringen i opgaverne er som følger:

Formatering:	Eksempel:
R kode	funktioner **dnorm** eller grafer **plot**
Indhold i datasæt og parametre	*huspriser* eller *x*

1. Opgave - Statistiske fordelinger

R har en masse indbyggede statistiske fordelinger, som kan bruges til at lave sandsynlighedsregning. For hver fordeling er der nogle beregningsmuligheder.

- 'd' - tæthedsfunktion (density)
- 'p' - kumuleret tæthedsfunktion (sandsynlighed for $\leq x$) (destribution function)
- 'r' - simuler tilfældige tal fra fordelingen (random generation)

1.1 Normalfordelingen

Hvis man bruger 'd' finder man sandsynligheden for et specifikt udfald. Vi har en normalfordeling med et gennemsnit på 7 og der er en standardafvigelse på 5. Hvad er sandsynligheden for at få præcis 10? Det beregnes ved

dnorm(10,**mean** = 7, **sd** = 5),

Svaret skulle gerne være 6.6%. Find nu sandsynligheden for at få præcis 7, hvis gennemsnittet falder til 4 og standardafvigelse stiger til 6.

Denne gang vil vi gerne se på hvad sandsynligheden for at få over 10 med et gennemsnit på 7 og en standardafvigelse på 5. Ved 'p' finder den sandsynligheden op til udfaldet.

pnorm(10,mean = 7, sd = 5)

Det blev dog et stort tal og virker utroligt hvis det er sandsynligheden for at få over 10, men det vi fandt var sandsynligheden for at få under 10. Derfor skal beregningen trækkes fra 1.

1-**pnorm**(10,mean = 7, sd = 5)

Find nu sandsynligheden for at for en scorer på over 7, hvis gennemsnittet falder til 4 og standardafvigelse stiger til 6.

Den sidste mulighed vi gennemgår er at lave et tilfældigt (random) datasæt som passer til en fordeling. Vi kan derved lave et datasæt som simulerer en normalfordeling. Hvis vi siger at vi gerne vil have 500 værdier med et gennemsnittet på 7 og standardafvigelse på 5.

rnorm(n = 500,mean = 7, sd = 5)

Prøv og plot data. Når i har plottet det, så prøve og kør både **rnorm** og plottet endnu en gang. Hver gang i kører den vil den lave et nyt datasæt.
Lav et nyt datasæt med 200 med et gennemsnittet på 4 og standardafvigelse på 6. Gem det i et parameter *K* og plot det.

1.2 Binomialfordeling

Man bruger binomialfordelingen, når man har et forsøg, der kun har to udfald: succes og fiasko. Man gentager forsøget et antal gange. Dette antal kaldes antalsparameteren og betegnes med 'n'. Desuden skal der være en fast sandsynlighed for at der bliver succes. Denne kaldes sandsynlighedsparameteren og betegnes med p.

Vi tager udgangspunkt i plat og krone. Vi vil gerne finde ud af hvad sandsynligheden er for at få krone 26 eller færre gange ud af 51 kast. Vi går ud fra at sandsynligheden for plat og krone er lige.

pbinom(26,**size**=51,**prob**=0.5)

Hvad er sandsynligheden for at du får krone præcis 26 gange ud af 51 kast? Hvad er sandsynligheden for at du får krone over 45 gange ud af 51 kast? Lav det nu om til en snyde mønt, hvor der er større sandsynlighed for plat end krone. Sandsynligheden for at få plat er 75% per kast. Beregn sandsynligheden for at få 9 krone på 14 kast med den nye mønt.

1.3 One Sample t-Test

Det er en parametrisk test, der bruges til at teste om gennemsnittet af en prøve fra en normal fordeling med rimelighed kunne være en bestemt værdi. For at lave en one sample t-Test skal vi lave et datasæt det gøres ved:

t1<−**rnorm**(50,**mean**=10,**sd**=0.5)

For at lave testen skrives:

```
t.test(t1, mu=10)
```

Hvis p-værdien ikke mindre end signifikansniveauet på 0.05, vil det ikke være muligt og afvise nullhypotesen at middelværdien = 10. Prøv og ændre mean i t1 til 1, hvad kan man nu sige ud fra p-værdien?

1.4 Two Sample t-Test

Two Sample t-Test bruges til at sammenligne middelværdien af 2 prøver. t-Test forudsætter, at de prøver, der testes, trækkes fra en normal fordeling. Start med at lave to arrays:

```
x <− c(0.85, 0.80, 1.89, 1.04, 1.45,0.34, 1.91, 1.64, 0.73, 1.46)
    y <− c(1.15, 0.88, 0.90, 0.434, 1.54)

t.test(1:10, y = c(7:20))
```

Hvis p-værdien er mindre end signifikansniveauet på 0.05, vil det være muligt og afvise nullhypotesen at middelværdien ikke er ens.

1.5 Kolmogorov-Smirnov test

Kolmogorov-Smirnov-testen bruges til at kontrollere, om 2 prøver følger den samme fordeling.

Vi prøver først og se om hvordan det ser ud med to forskelige fordelinger.

```
x <− rnorm(1000)
    y <− runif(1000)
```

For at lave testen skrives:

```
ks.test(x, y)
```

Hvis p-værdi ¡0.05 (signifikansniveau), afviser vi nulhypotesen, at de trækkes fra samme fordeling. Med andre ord betyder $p < 0.05$ at x og y er fra forskellige distributioner Prøv nu istedet

```
x <− runif(1000)
    y <− runif(1000)
    ks.test(x, y)
```

Hvordan ser p værdien ud nu? Prøv og se med andre fordelinger om de giver samme resultat?

2. Opgave - ANOVA

En kemiker ønsker at teste virkningen af fire kemiske agenser på styrken af en bestemt type klud. Fordi der kan være variabilitet fra en bolt til en anden, beslutter kemikeren at bruge et randomiseret blokdesign, med bolte som betragtes som blokke. Hun vælger fem bolte og anvender alle fire kemikalier i tilfældig rækkefølge til hver bolt.
Ud fra oplysningerne ved vi at de fire kemikalier er a 4 og de fem bolte er b. Det vil sige at a er 4 og b er 5. Ud fra det kan vi få N. Derfor kan det skrives således

$a <- 4$
$b <- 5$
$N <-$ a*b

Observationerne er oplyst som følgende.

dat $<-$ **data.frame***(Chemical=factor(rep(1:a,each=b)),Bolt=factor(rep(1:b,times=a)),*
strength=c(73,68,74,71,67, 73,67,75,72,70, 75,68,78,73,68, 73,71,75,75,69))

Udfra dette kan vi lave en ANOVA analyse. Det gøres ved at skrive.

aovmodel$<-$ **lm**(**strength** $\sim$ **Chemical + Bolt,data=dat**)

anova(*aovmodel*)

Da p værdien er højere end 5% for Bolt kan vi ikke afvise H0 hypotesen.

3. Opgave - Lineær regression

Start med lav en lineær regression over boligpriserne over tiden fra *data*. Dette gøres vha. funktionen **lm**. Boligpriserne og tiden er dem som vi har gemt i *p* og *a*. Der skal der bare skrives

fx $<-$ **lm**($p \sim a$)

eller skriv *fx* $<-$ **lm**(*huspriser*$\sim$*aar*) hvis du har brugt **attach**(*data*) i dit dokument. Skriv *fx* og kør linjen. Nu kommer de estimerede koefficienter frem for regressionen (skæring og hældning). Ved at skrive **summary**(*fx*) kommer der et overblik af regressionen. Her vises bl.a. standard fejl og R^2, som fortæller hvor godt regressionen passer datapunkterne. I dette tilfælde er en lineær regression ikke en særlig god model til at beskrive huspriserne over tid.

Du kan plotte regressionen sammen med dataplotted sådan,

plot(*a*,*p*)

abline(*fx*)

Lav en regression over *aar* og *cpi* (definer den som *gx*). Er det en god model? Lav et plot over den.

3.1 Opgave - Residual plot

For rigtigt at kunne se, hvor god en model er til at beskrive data, kan man kigge på residualerne. Skriv først **par**(**mfrow** = **c**(2,2)) og kør linjen. Derefter skriv **plot**(*fx*). Nu kommer der 4 plots op som fortæller hvor residualerne er i forhold til modellen.

Lav et residual plot over modellen *gx* fra forgående opgave.

3.2 Opgave - Multivariat analyse

Den første opgave var en simpel lineær regression. Nu vil vi lave en model som indeholder flere variable end en enkelt. Dette gøres ved samme funktion før, der skal bare tilføjes nogen flere variable som det kan ses under her:

tx <− **lm**(*huspriser* ∼ *cpi*+*indk*+*ci6*,**data=** data)

Prøv modellen efter med **summary**(*tx*). Her kan p værdien ses på de forskellige variable. Den med den højeste er ci6. Den er også højere end de tilladte 5% og derved ikke signifikant. Derfor fjerner vi nu ci6.

tx1 <− **lm**(*huspriser* ∼ *cpi*+*indk*,**data=** data)

Prøv modellen efter med **summary**(*tx1*). Nu kan det ses at begge variable er signifikante. Ud fra de tre variable så var den bedst mulige model vi kunne lave tx1. Prøv og lav residual plot over modellen *tx1*. Prøv og lav den bedst mulige model over huspriser med variablene **skat**,**led**, **indk** og **veast**.

3.3 Opgave - Forudsigelse

Nu er der laves en model med flere variable. Denne model kan i bruge til at forudsige den næste værdi. I starter med at lave noget nyt data, hvor i sætter de værdi ind som i gerne vil bruge i modellen. De værdier vi har valgt er følgende:

$newdata <- data.frame(cpi = 300, indk = 180)$

Her efter tager vi disse værdier og sætter ind i predict funktionen sammen med modellen vi lavede.

$pred <-$ **predict**$(tx1, new_data, interval = "prediction")$

I pred har vi nu hvilken værdi huspriserne ville have hvis cpi og indk ændrede sig. lwr og upr er den nedre og øvre usikkerheden.

4. Opgave - Løkker

Løkker er en metode hvor man får R til at lave den samme beregning flere gange uden at man skal kører linjen flere gange. Altså køre i ring. Løkker bliver også brugt i mange andre programmeringssprog.

For-løkker

Vi vil starte med at lave en for-løkke. Den bedste danske oversættelse er nok en 'fra-til'-løkke.

I en for-løkke beder vi løkken om at kører fra $n = n_1$ til $n = n_2$ og hver gang skrifter n værdi så den hæves med 1. Hvis $n_1 = 3$ og $n_2 = 5$, så vil n første gang være 3, anden "runde" være 4, og tredje "runde" være 5, hvorefter den vil stoppe løkken.

Start med at skrive at skrive **for**(n in 1:6). Derved sætter vi n til at starte ved 1 og slutte når den er 6. Skift linje og skriv { og skrift linje igen. Skriv nu $z<-(n+3)*4$. Nu laver vi en beregning som vi gemmer i z. Skrift linje igen og skriv til sidst }. Det betyder at løkken er slut. Det kommer til at se sådan her ud:

for(n in 1:6)
{
$z<-(n+3)*4$
}

Kør alle linjerne. Hvordan ser z ud? Det ser ud til at z kun har gemt den sidste beregning. Det er fordi vi ikke har bedt den om at gemme alle værdier i en vektor. Lav en array med seks nuller med **rep** funktionen fra opgaven om arrays i øvelse 1. For at få den til at gemme flere værdier i z så i skrive $z[n]$. Det skulle nu se sådan her ud.

z=**rep**(0,**times**=6)

for(n in 1:6)
{
$z[n]<-(n+3)*4$
}

Definer k til at være rækken (1,3,5). Lav en for-løkke som går gennem k (n in k). Hvor der bliver beregnet $n*(n-1)+24-n$. Alle værdierne skal gemmes i parameteret h (som skal have fire elementer). Tænk over at n ikke kan bruges som placering i h, så her skal du definere en ny parameter. Prøv selv og ellers spørg om hjælp.

While-løkker

Den næste løkke er en while-løkke, som direkte oversættes til 'imens'-løkke. Her går løkken så længe (imens) en værdien er inden for en hvis grænse. Det kunne fx være at n er mindre end 10.

Start med at skrive n=1. Denne gang bliver vi nødt til at fortælle hvad n er da det er den værdi vi bruger til at stoppe while-løkken med (sæt n til 1). Skriv nu **while**($n<10$). Nu

starter n med at være 1 og while-løkken går imens n er mindre end 10. Nu kan vi skrive de beregninger vi gerne vil have den til at lave. Skift linje og skriv $z[n] <- (n+3)*4$. Skift linje og skrive $n <- n+1$. Det er vigtigt at n bliver større ellers vil løkken kører uendeligt (hvilket vi ikke har tid til i et 4 timers kursus). Hver gang løkken kører en "runde" bliver n altså 1 større. Skift linje igen og skriv }. Så det kommer til at se sådan ud:

```
n=1

while(n< 10)
    {
    z[n]<- (n+3)*4
    n<- n+1
    }
```

Prøv at lave en while-løkke som kører så længe $t < 20$. Hvor der bliver beregnet $t < -n*(n-1)+2-n$ og n stiger 1 hver gang. Alle værdierne skal gemmes i parameteren h.

5. Opgave - Funktioner

Dette er en kort intro til at lave sine egne funktioner i R. Vi har allerede kigget på nogen af de indbyggede som **min**, **max** og **rnorm**.

Det er opbygget så man starter med at lave et navn på funktionen, som man skal kalde for at kører den. Derefter hvilke nogle argumenter den skal have for at kunne kører. Det næste er hvad den skal gøre med argumenterne og hvordan den skal returnere det.

```
navn.på.funktion <- function(argument1, argument2)
    {
    statements
    return(element)
    }
```

Vi starter med at lave en meget simpelt funktion ved navn fun1. Vi vil gerne lave en funktion som kan beregne $y < -3*x-4$. Den skal så give os y.

```
fun1 <- function(x)
    {
    y < -3*x–4
    return(y)
    }
```

For at finde svaret når x er 10 skrives **fun1**(10). Lave en funktion **fun2** som beregner $z < -x^2 - 2x + 8$. Hvad giver det hvis x er 6?

Lav nu en funktion **fun3** som både regner $y < -3 * x - 4$ og $z < -x^2 - 2x + 8$. Den skal returnere y og z. Hvad er y og z når x er 23?