

# Seminar 3

Lise Rødland

March 22, 2021

På dette seminaret skal vi gjennom: 1. Repetisjon: om laste inn data 2. Omkoding av variabler 4. Subsetting av datasett 5. Plotting: Intro til ggplot2 6. Ulike typer plott 7. Lagring av plott

## Laste inn data

I dag skal vi bruke et datasett som ligger i en pakke som heter gapminder. For å gjøre det må vi først installere pakken og åpne den.

```
install.packages("gapminder")  
library(gapminder)
```

Nå er datasettet tilgjengelig i R. For å lagre datasettet som et objekt i environment kjører vi denne koden:

```
data <- gapminder
```

Nå er datasettet klart til bruk, men for å øve på å lagre og laste inn datasett skal vi lagre dette datasettet i et Rdata-format ved å bruke funksjonen `save()`. Husk at når man laster inn og lagrer data så vil koden være avhengig av hvilken type dataformat du har og den må tilpasses din mappestruktur. Dersom du ikke jobber i prosjekt så er det første du må gjøre å sette working directory ved å bruke koden `setwd("filbane")`. Deretter bruker du `save()` til å lagre datasettet.

```
save(data, file = "gapminder.Rdata")
```

Du vil nå se datasettet i den mappen du har bedt R om å lagre det i. Før vi laster inn datasettet på nytt så skal vi rydde litt i environment, altså vi skal fjerne noen objekter vi ikke har bruk for lengre for å få bedre oversikt. Det gjør vi ved hjelp av funksjonen `rm()`. `rm()` kan brukes til å fjerne et objekt, flere objekter eller alle objektene fra environment:

```
# Denne koden fjerner alt i environment  
rm(list=ls())  
  
# Denne koden fjerner et objekt  
rm(objektnavn)  
  
# Denne koden fjerner flere objekter  
rm(objektnavn1, objektnavn2)
```

For å fjerne alt så kan du også trykke på ikonet med en feiekost på i environment. Merk at dersom du ikke har lagret objektene på maskinen din så må du kjøre koden på nytt for å få tilbake de objektene du fjerner. Her fjerner jeg dataobjektet mitt før jeg laster det inn på nytt ved hjelp av `load()`:

```
rm(data)

load("../data/gapminder.Rdata")
```

## Omkoding av variabler

Når vi omkoder variabler i et datasett, bør vi opprette en ny variabel. Dersom vi ikke gjør dette, erstatter vi informasjonen i den opprinnelige variabelen. Vi trenger informasjonen i den opprinnelige variabelen for å sjekke at omkodingen har fungert som vi ønsker. Vi kan også gjøre feil som vi ikke kan rette opp uten den opprinnelige variabelen. Først bruker jeg funksjonene `show()`, `head()` og `summary()` for å få litt oversikt over data. Du kan også trykke på datasettet i environment:

```
show(variable.names(gapminder))
```

```
## [1] "country" "continent" "year" "lifeExp" "pop" "gdpPercap"
```

```
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int>  <dbl>    <int>    <dbl>
## 1 Afghanistan Asia      1952   28.8  8425333    779.
## 2 Afghanistan Asia      1957   30.3  9240934    821.
## 3 Afghanistan Asia      1962   32.0 10267083    853.
## 4 Afghanistan Asia      1967   34.0 11537966    836.
## 5 Afghanistan Asia      1972   36.1 13079460    740.
## 6 Afghanistan Asia      1977   38.4 14880372    786.
```

```
summary(gapminder)
```

```
##           country      continent      year      lifeExp
## Afghanistan: 12 Africa :624 Min. :1952 Min. :23.60
## Albania : 12 Americas:300 1st Qu.:1966 1st Qu.:48.20
## Algeria : 12 Asia :396 Median :1980 Median :60.71
## Angola : 12 Europe :360 Mean :1980 Mean :59.47
## Argentina : 12 Oceania : 24 3rd Qu.:1993 3rd Qu.:70.85
## Australia : 12 Max. :2007 Max. :82.60
## (Other) :1632
##      pop      gdpPercap
## Min. :6.001e+04 Min. : 241.2
## 1st Qu.:2.794e+06 1st Qu.: 1202.1
## Median :7.024e+06 Median : 3531.8
## Mean :2.960e+07 Mean : 7215.3
## 3rd Qu.:1.959e+07 3rd Qu.: 9325.5
## Max. :1.319e+09 Max. :113523.1
##
```

## Matematisk omkoding

Først skal vi lære å kode om variablene ved å bruke matematiske omkoding. For å vise dette bruker jeg variabelen `year`. Ved hjelp av `summary()` så finner jeg ut at den laveste verdien på variabelen er året 1952. Det er året observasjonen ble gjort. Jeg vil gjøre dette året til år null, mens de andre verdiene på variabelen skal bli hvor mange år etter år null ble observasjonen gjort.

```
# Sjekker summary for opprinnelige variabel
summary(data$year)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1952   1966   1980    1980   1993    2007
```

```
# spennet er 2007-1952, dvs. 55 år
2007-1952
```

```
## [1] 55
```

```
# Legger til ny variabel i datasettet
data$year_1952 <- data$year - 1952
```

```
# Sjekker summary for ny variabel
summary(data$year_1952)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      0.00   13.75   27.50   27.50   41.25   55.00
```

Så undersøker jeg om omkodingen er blitt gjort riktig ved å bruke funksjonen `table()`. Jeg gjør dette på to måter.

```
# Lager en tabell med opprinnelig og ny variabel:
table(data$year_1952, data$year, useNA = "always")
```

```
##
##      1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007 <NA>
##  0      142    0    0    0    0    0    0    0    0    0    0    0
##  5       0   142    0    0    0    0    0    0    0    0    0    0
## 10       0    0   142    0    0    0    0    0    0    0    0    0
## 15       0    0    0   142    0    0    0    0    0    0    0    0
## 20       0    0    0    0   142    0    0    0    0    0    0    0
## 25       0    0    0    0    0   142    0    0    0    0    0    0
## 30       0    0    0    0    0    0   142    0    0    0    0    0
## 35       0    0    0    0    0    0    0   142    0    0    0    0
## 40       0    0    0    0    0    0    0    0   142    0    0    0
## 45       0    0    0    0    0    0    0    0    0   142    0    0
## 50       0    0    0    0    0    0    0    0    0    0   142    0
## 55       0    0    0    0    0    0    0    0    0    0    0   142
## <NA>     0    0    0    0    0    0    0    0    0    0    0    0
```

```
# Lager en tabell med en logisk test:
table((data$year_1952 + 1952) == data$year, useNA = "always")
```

```
##
## TRUE <NA>
## 1704    0
```

Jeg kan også endre retningen på verdien ved matematisk omkoding. Da ganger jeg variablene med (-1) og plusser på antall verdier variablene har. Nå blir 2007 år null.

```
# Endrer retning på variabelen:
data$year_2007 <- data$year_1952*(-1) + 55
```

```
# Sjekker at det ble riktig med table():
table(data$year_1952, data$year_2007, useNA = "always")
```

```
##
##      0    5    10    15    20    25    30    35    40    45    50    55 <NA>
## 0      0    0    0    0    0    0    0    0    0    0    0 142    0
## 5      0    0    0    0    0    0    0    0    0    0    0 142    0
## 10     0    0    0    0    0    0    0    0    0    0 142    0    0
## 15     0    0    0    0    0    0    0    0    0 142    0    0    0
## 20     0    0    0    0    0    0    0    0 142    0    0    0    0
## 25     0    0    0    0    0    0    0 142    0    0    0    0    0
## 30     0    0    0    0    0    0 142    0    0    0    0    0    0
## 35     0    0    0    0 142    0    0    0    0    0    0    0    0
## 40     0    0    0 142    0    0    0    0    0    0    0    0    0
## 45     0    0 142    0    0    0    0    0    0    0    0    0    0
## 50     0 142    0    0    0    0    0    0    0    0    0    0    0
## 55    142    0    0    0    0    0    0    0    0    0    0    0
## <NA>    0    0    0    0    0    0    0    0    0    0    0    0    0
```

## Omkoding med ifelse()

Den funksjonen jeg bruker mest til omkoding, er `ifelse()`. Funksjonen kan brukes på numeriske og kategoriale variabler. Syntaksen til denne funksjonen kan forklares som følger:

```
data$nyvar <- ifelse(test = my_data$my.variabel == "some logical condition",
                    yes  = "what to return if 'some condition' is TRUE",
                    no   = "what to return if 'some condition' is FALSE")
```

`ifelse()` kan for eksempel brukes til å lage en ny variabel som tar verdien 1 om observasjonen har en verdi som er større enn gjennomsnittet og null ellers. Gjennomsnittet finner du ved å bruke funksjonen `mean()`. Under fyller jeg inn betingelsen (condition), hvilken verdi ny variabel skal dersom betingelsen er oppfylt (yes) og hvilken verdi ny variabel skal ha dersom betingelsen ikke er oppfylt (no). Vanligvis vil vi ikke skrive `test =`, `yes =` og `no =`, men jeg har tatt det med her for å gjøre det lettere å koble de to:

```
data$lifeExp_2 <- ifelse(test = data$lifeExp > mean(data$lifeExp, na.rm = TRUE),
                        yes = 1,
                        no  = 0)
```

Jeg bruker en kombinasjon av en tabell og en logisk test (se forklaringer fra første seminar om logiske tester) for å sjekke at det ble riktig:

```
table(data$lifeExp_2, data$lifeExp > mean(data$lifeExp, na.rm = TRUE), useNA = "always")
```

```
##
##      FALSE TRUE <NA>
## 0      809    0     0
## 1         0 895     0
## <NA>      0    0     0
```

## Omkoding med recode()

Dersom vi vil omkode en variabel med flere variabler så kan vi gjøre det med flere `ifelse()` nøstet i hverandre. Det kan imidlertid bli veldig mye kode. Et alternativ er å kombinere `mutate()` og `recode()`. `mutate()` er en funksjon i tidyverse vi bruker for å lage nye variabler. `recode()` er også en del av tidyverse og den brukes til å endre verdier på variabler. Syntaksen til kombinasjonen av `mutate()` og `recode()` er:

```
mutate(nyvariabel = recode(gammelvariabel,
                           verdi1 = nyverdi1,
                           verdi2 = nyverdi2))
```

I dette eksempelet skal vi lage en ny variabel `continent2` med tallverdier i stedet for kontinentnavn. Etter som funksjonen vi skal bruke ligger i pakken `tidyverse` så må vi aller først laste inn pakken med `library()`

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse_2019.09.23
```

```
## v ggplot2 3.3.2    v purrr   0.3.4
## v tibble  3.0.3    v dplyr  1.0.2
## v tidyr   1.1.2    v stringr 1.4.0
## v readr   1.4.0    v forcats 0.5.0
```

```
## -- Conflicts ----- tidyverse_conflicts_2019.09.23
```

```
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
data <- data %>%
  mutate(continent2 = recode(continent,
                             Africa = 1,
                             Americas = 2,
                             Asia = 3,
                             Europe = 4,
                             Oceania = 5))
```

Vi sjekker med en tabell at det ble riktig:

```
table(data$continent, data$continent2, useNA = "always")
```

```
##
##           1    2    3    4    5 <NA>
## Africa   624    0    0    0    0    0
## Americas  0 300    0    0    0    0
## Asia      0    0 396    0    0    0
## Europe    0    0    0 360    0    0
## Oceania   0    0    0    0  24    0
## <NA>      0    0    0    0    0    0
```

## Subsetting av datasett

Å subsetting betyr å lage et nytt datasett basert på et gammelt. Dette gikk vi gjennom i seminar 2, men jeg repeterer det her. For å subsetting så bruker vi funksjoner i pakken `tidyverse`. Vi bruker pipes angitt ved `%>%` for å binde sammen ulike operasjoner vi vil utføre på et datasett. Dette kalles for piping. Det er en effektiv og ryddig måte å kode på. Derfor kommer jeg ofte til å bruke pipes i seminarene. Pakken installerte vi i andre seminar, men vi må laste den inn for å bruke den:

```
library(tidyverse)
```

Den første funksjonen `select()` brukes til å velge ut hvilke variabler du vil ha med i det nye datasettet:

```
data_pop <- data %>%
  select(pop)

data_sub <- data %>%
  select(pop, country, year)
```

Du kan også bruke funksjonen `filter()` til å velge ut spesielle observasjoner som tilfredsstiller de kravene du setter. Under har jeg brukt funksjonen til å bare ta med observasjonene med verdien 1954 på variabelen `year` og som har en verdi høyere end gjennomsnittet på variablene `pop`:

```
data_1952 <- data %>%
  filter(year == 1952)

data_1952pop <- data %>%
  filter(year == 1952,
         pop > mean(pop))
```

Nå har du fått fire nye objekter i environment; `data_pop`, `data_sub`, `data_1952` og `data_1952pop`. Legg merke til at disse har ulikt antall observasjoner (obs.) og variabler (variables).

Vi skal ikke bruke disse videre så vi fjerner dem med `rm()` for å rydde litt opp:

```
rm(data_pop, data_sub, data_1952, data_1952pop)
```

## Intro til ggplot2

Det finnes flere ulike måter å plote på i R. Pakken som vi skal bruke heter **ggplot2**, og er en del av **tidyverse**-universet. Når vi laster inn **tidyverse** så er altså **ggplot2** en del av pakka.

Det første vi skal gjøre nå er å bli litt kjent med hvordan vi bygger figurer med ggplot2. Kort fortalt må vi sette sammen flere argumenter, eller deler med informasjon, for at ggplot skal vite hva slags figur vi ønsker å lage. Når jeg går gjennom dette nå så kommer jeg gradvis til å legge på flere lag i plottet. Når dere koder selv så trenger dere ikke kjøre hver kodebit separat, alt kan kjøres samlet på én gang, slik det ser ut i den siste koden vi lager. Nå deler jeg det opp kun for å forklare.

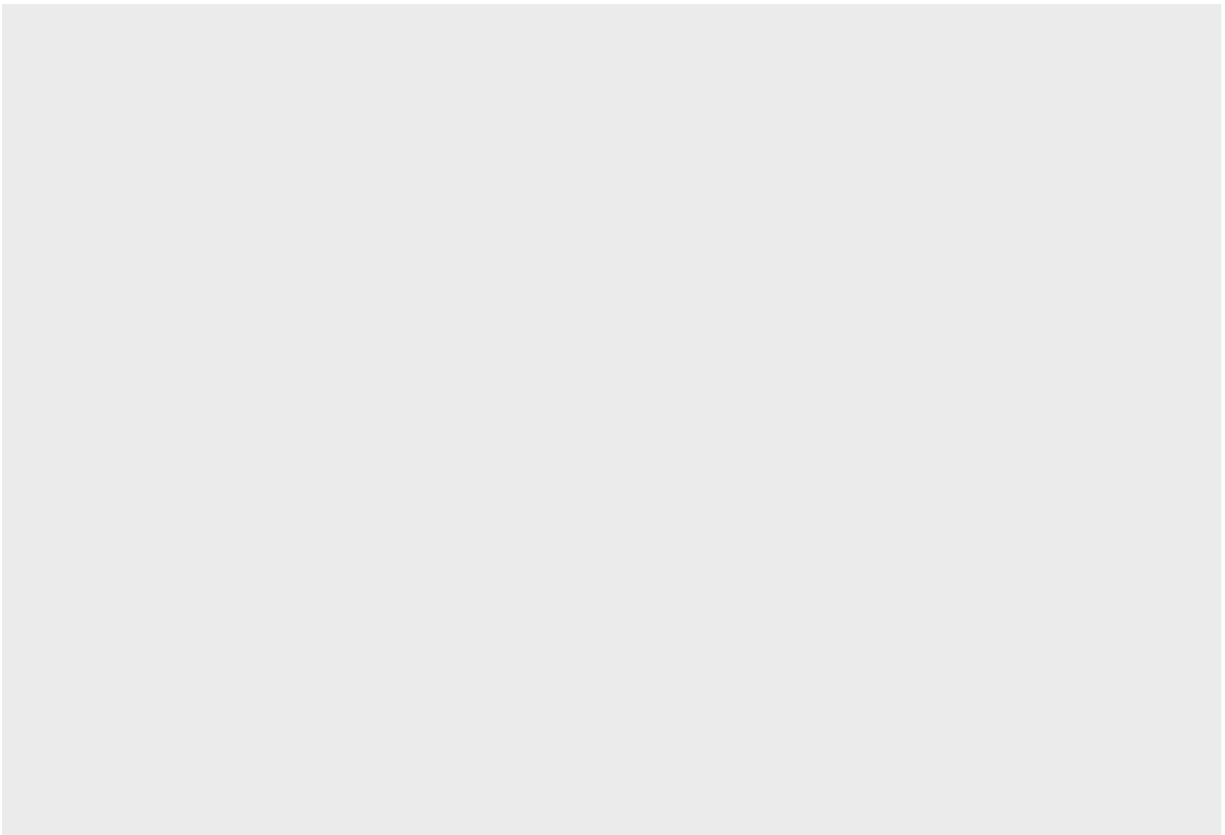
PS: Dette trenger dere ikke tenke så mye på akkurat nå, men til en annen gang så er det lurt å huske på at ggplot2 vil ha dataene i et “tidy” format, dvs. at hver rad (vannrett) er en observasjon, og hver kolonne er ulike variabler (loddrett). Dette kan dere lese mer om i kapittel 3 i Healy-boka på pensum.

Vi kommer til å følge de fem stegene presentert på side 60 i Healy: 1. Fortell **ggplot()** hvor du vil hente data fra. 2. Fortell **ggplot()** hvilken sammenheng du vil plote. 3. Fortell **ggplot()** hvordan du vil fremstille sammenhengen. 4. Legg til **geoms\_** etter behov en etter en. 5. Bruk funksjoner til å justere skala, etiketter, tittel o.l..

### Steg 1: Fortell ggplot() hvor vi vil hente data fra

Steg 1 er å fortelle ggplot hvor den skal hente data fra. Det gjør vi ved å spesifisere **data =** argumentet i funksjonen **ggplot()**. I vårt tilfelle heter datasettet **data** og vi skriver:

```
ggplot(data = data)
```

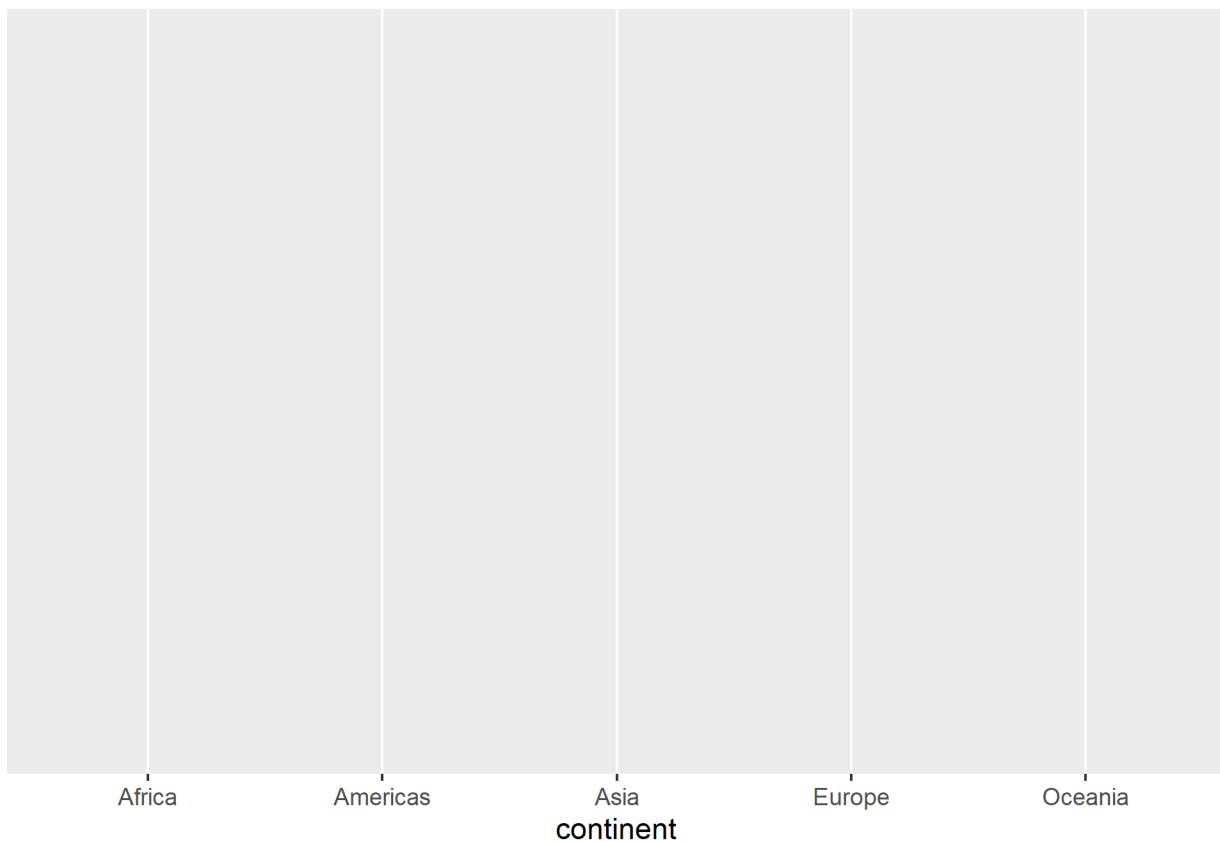


Hva skjer når vi kjører koden nå? Vi får et helt tomt plott. Mer info må til!

## Steg 2: Fortell ggplot hvilken sammenheng vi vil plote

Det neste vi gjør er å fortelle ggplot hva vi vil måle på x- og y-aksen. Nå skal vi starte med et stolpediagram, der vi vil se hvor mange observasjoner det er i datasettet fordelt på de fem kontinentene. Da trenger vi bare å gi ggplot beskjed om at vi vil ha `continent` på x-aksen. Y-aksen vår blir automatisk til antall (`count`). Senere i dette seminaret skal vi se på et eksempel hvor vi gir informasjon om begge aksene. Inne i `ggplot()` hvor vi har fortalt hvilket datasett vi vil bruke, så legger vi til en `mapping`. `aes()` står for aesthetics, og det vi gjør ved bruk av `aes()` er å fortelle ggplot hvilke variabler vi vil visualisere. Vi forteller ggplot at vi vil at variabelen `continent` skal vises på x-aksen.

```
ggplot(data = data, mapping = aes(x = continent))
```



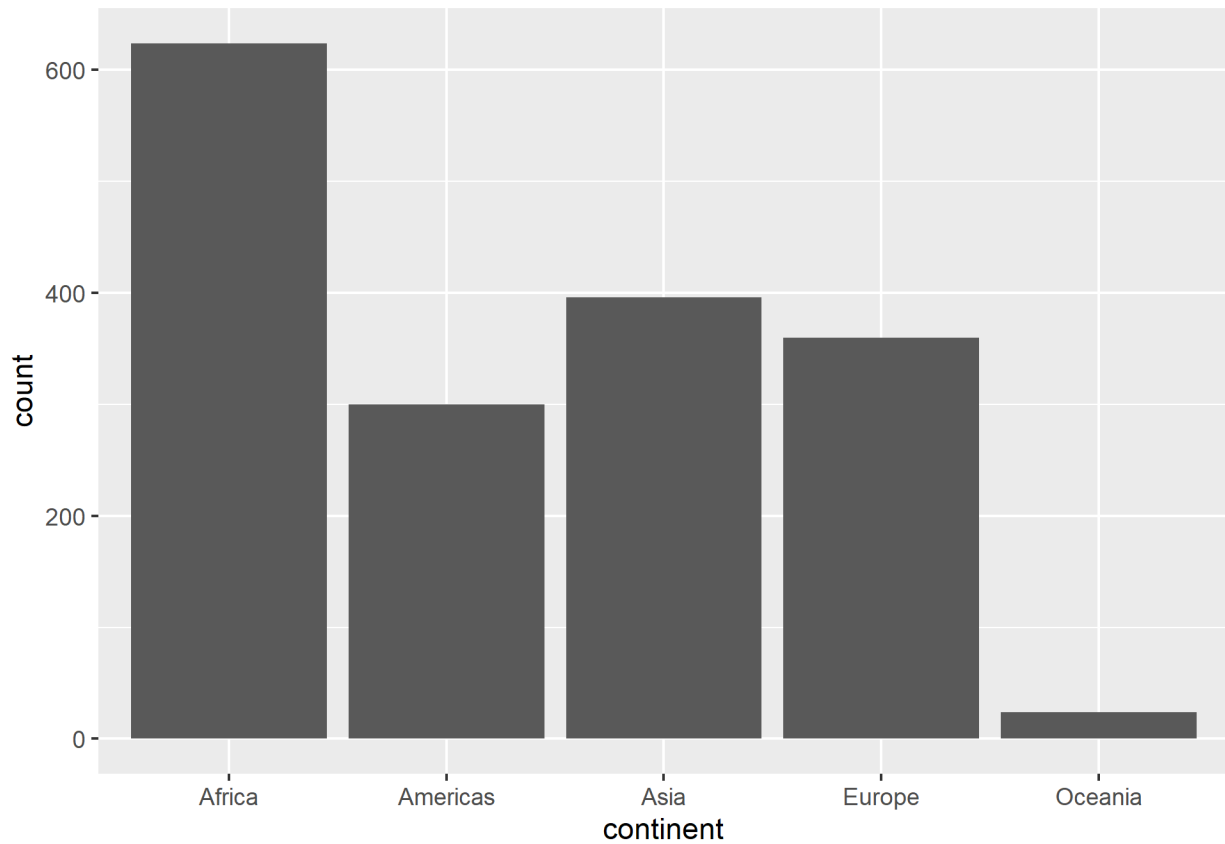
Det vi skriver i `aes()` kobler variabler til ting du ser i plottet. Det kan for eksempel være x- og y-verdier (`x = variabelnavn`, `y = variabelnavn`), farger (`color = variabelnavn`), former (`shape = variabelnavn`) eller størrelse (`size = variabelnavn`).

## Steg 3: Fortell ggplot hvordan du vil illustrere sammenhengen

Når vi kjører koden, så ser vi at kontinentene er på plass, men vi får likevel ikke noe særlig informasjon fra plottet. Hva mangler? Vi må skrive hva slags plott vi vil lage. Vi vil lage et stolpediagram, og velger `geom_bar()`. Hvis du prøver å skrive inn `geom_` vil du se at du får en hel rekke forslag til ulike plott. Flere av disse skal vi komme tilbake til.



```
ggplot(data = data, aes(x=continent)) +  
  geom_bar()
```



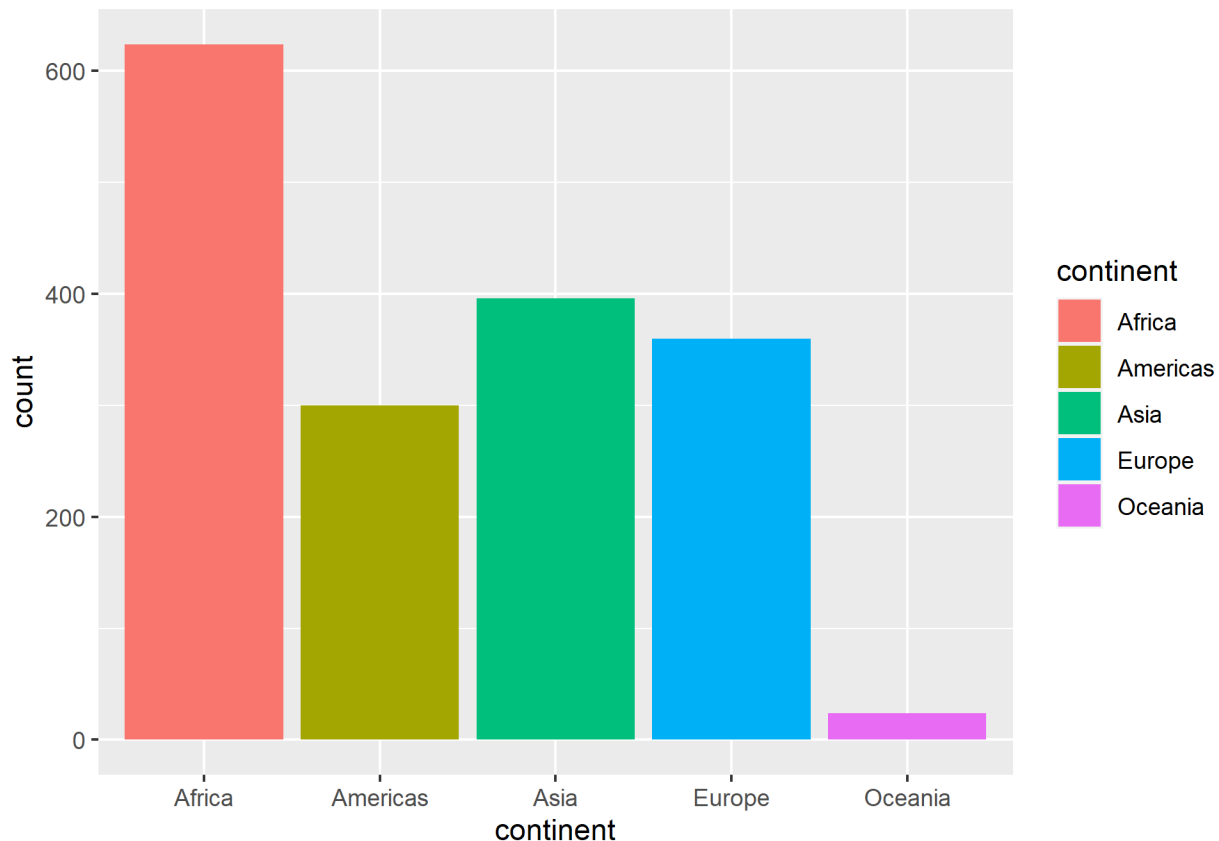
Da har vi fått opp et plott som viser observasjonene i datasettet fordelt på kontinenter. Vi kan dobbeltsjekke at dette ser riktig ut ved å lage et summary av variabelen.

```
summary(data$continent)
```

```
##   Africa Americas   Asia  Europe Oceania  
##     624     300    396   360    24
```

Det ser fint ut. Vi kan også legge inn andre typer av aesthetics i mappingen. Her er det flere muligheter, som dere vil bli kjent med etter hvert. Én av dem er å bruke ulike farger for å gi informasjon. Hvis vi vil at hvert kontinent skal få sin egen farge, så kan vi gjøre det slik:

```
ggplot(data = data, aes(x = continent, fill = continent)) +  
  geom_bar()
```



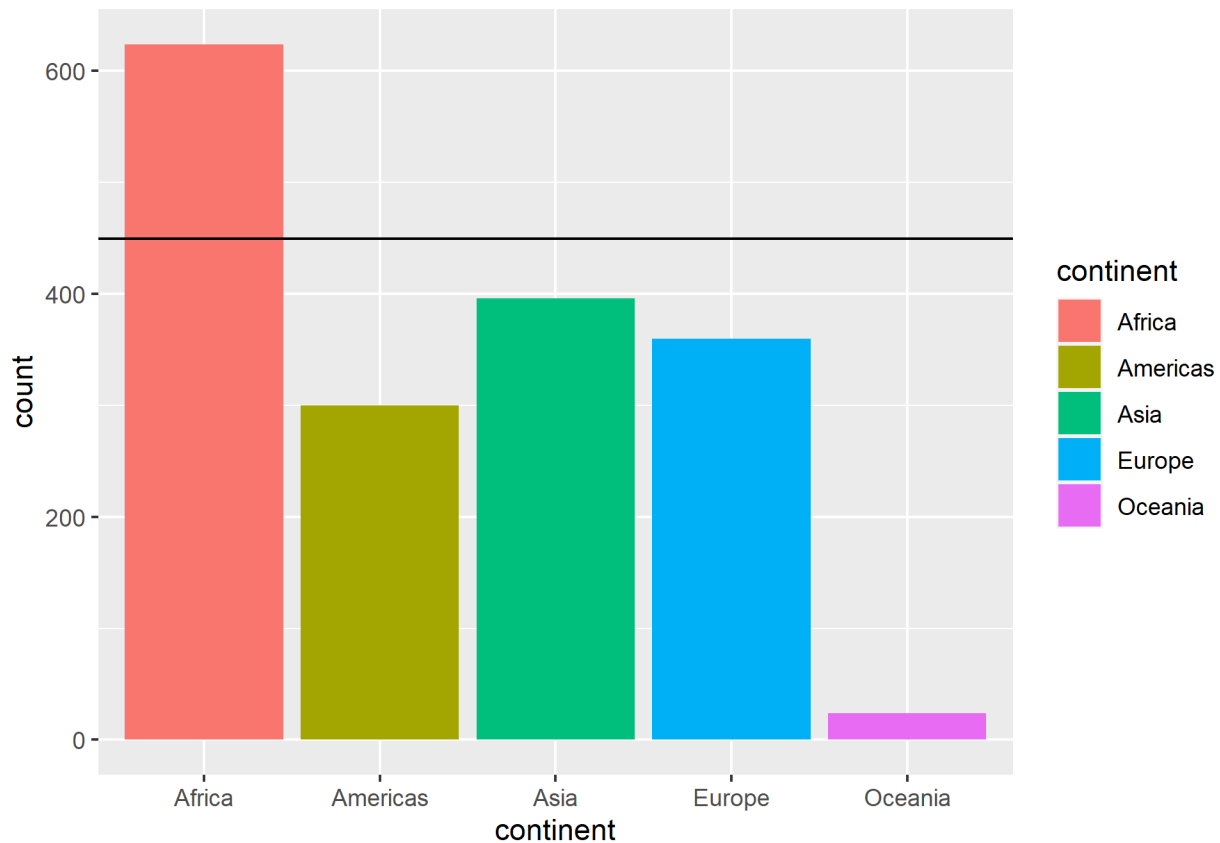
#### Steg 4: Legg til `geom_` etter behov en etter en.

Vi kan legge til flere `geom_`-elementer i plottet vårt ved å bruke `+`. Jeg kan for eksempel legge til en linje som indikerer gjennomsnittlig antall observasjoner per land. Da legger vi først til en variabel som angir antall observasjoner per land i datasettet. Dette gjør vi ved hjelp av funksjonen `add_count()`:

```
data <- data %>%
  add_count(continent)
```

Vi kan se i datasettet vårt at vi har fått en ny variabel `n`. Denne angir antall observasjoner per kontinent i datasettet vårt. For å legge til en linje som indikerer gjennomsnittlig antall observasjoner bruker vi `geom_hline()` (h for horisontal linje). I `aes()` gir vi ggplot informasjon om hvor denne linjen skal krysse y-aksen:

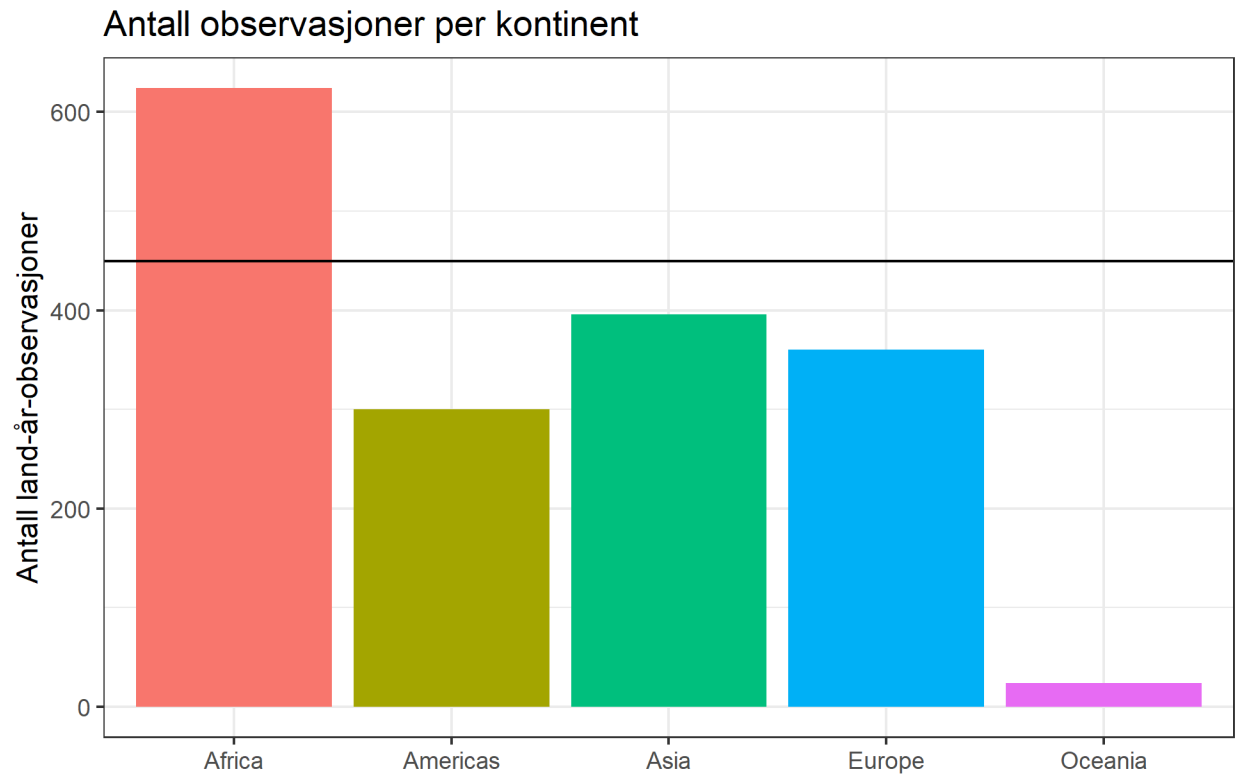
```
ggplot(data = data, aes(x = continent, fill = continent)) +
  geom_bar() +
  geom_hline(aes(yintercept = mean(n)))
```



### Steg 5: Bruk funksjoner til å justere skala, etiketter, tittel o.l..

Vi kan også endre teksten på x- og y-aksen, og gi plottet en tittel ved å legge til `labs()`, fjerne boksen med informasjon om farger og navn på høyre side ved hjelp av `theme(legend.position = "none")` og gjøre bakgrunnen litt penere med `theme_bw()`:

```
ggplot(data, aes(x = continent, fill = continent)) +
  geom_bar() +
  geom_hline(aes(yintercept = mean(n))) +
  labs(x = "",
       y = "Antall land-år-observasjoner",
       title = "Antall observasjoner per kontinent",
       caption = "Gjennomsnittlig antall observasjoner er gitt ved den horisontale linjen") +
  theme_bw() +
  theme(legend.position = "none")
```



Gjennomsnittlig antall observasjoner er gitt ved den horisontale linjen

Da har vi gått gjennom det helt grunnleggende i ggplot. Dette er en pakke med veldig mange ulike muligheter, og man blir bedre kjent med de for hver gang man bruker den. Jeg anbefaler dere å prøve dere fram, og sjekke ut noen av ressursene som jeg linker til i bunnen av dokumentet. Kjør gjerne deler av plottet for å se hva de ulike linjene gjør, google dere frem osv.

## Ulike typer plott

Nå skal vi lage ulike typer av plott ved bruk av ggplot. Hva slags plott du vil lage vil avhenge av hva slags variabler det er du vil undersøke og målenivået på disse variablene. For eksempel er det ikke så nyttig å lage et stolpediagram hvor du har en kontinuerlig variabel på x-aksen (for eksempel forventet levealder, som er oppgitt i desimaltall). Da vil en stolpe for hver unike verdi gi veldig, veldig mange stolper.

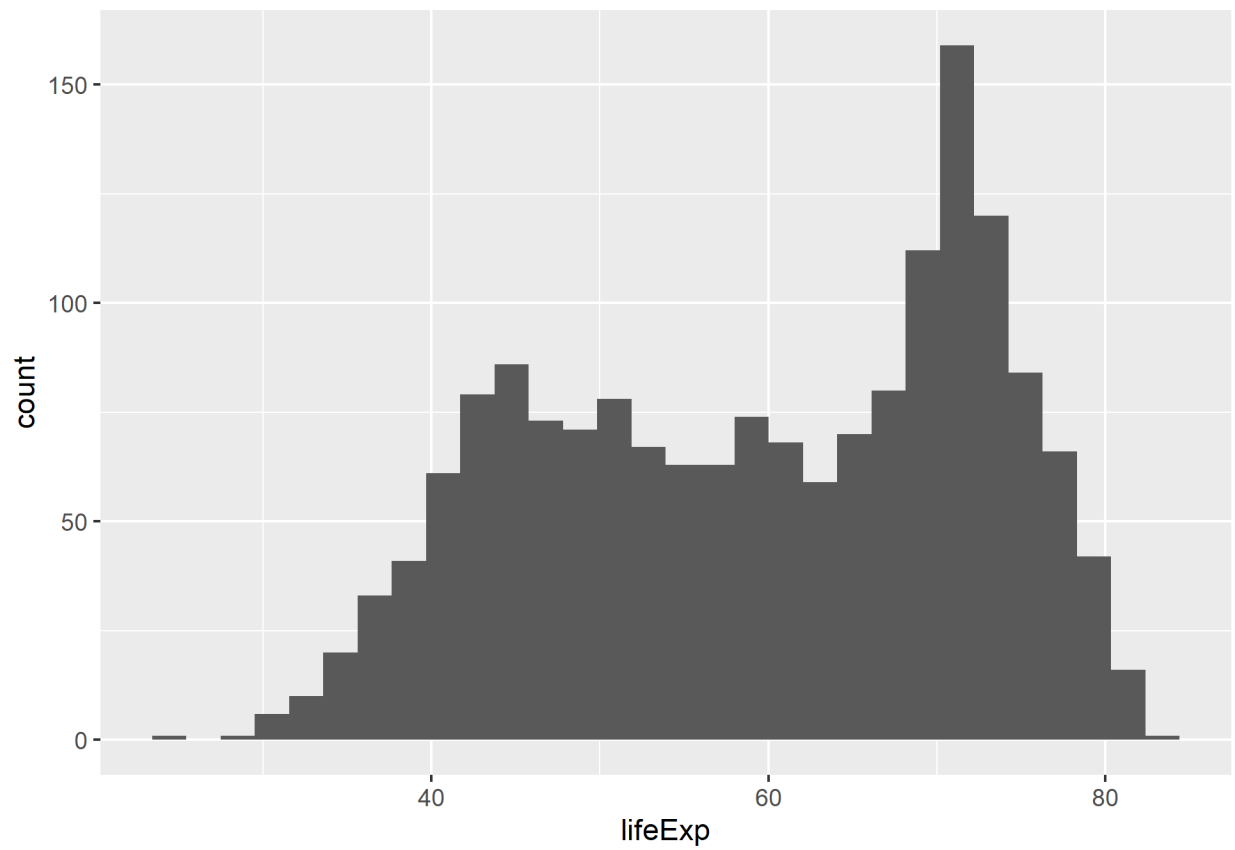
La oss i første omgang se på litt ulike varianter av plot for variabelen `lifeExp` som måler forventet levealder. Da lagrer vi først informasjon om data og verdier i et objekt så vi slipper å skrive det hver gang:

```
plot_le <- ggplot(data, aes(x = lifeExp))
```

## Histogrammer

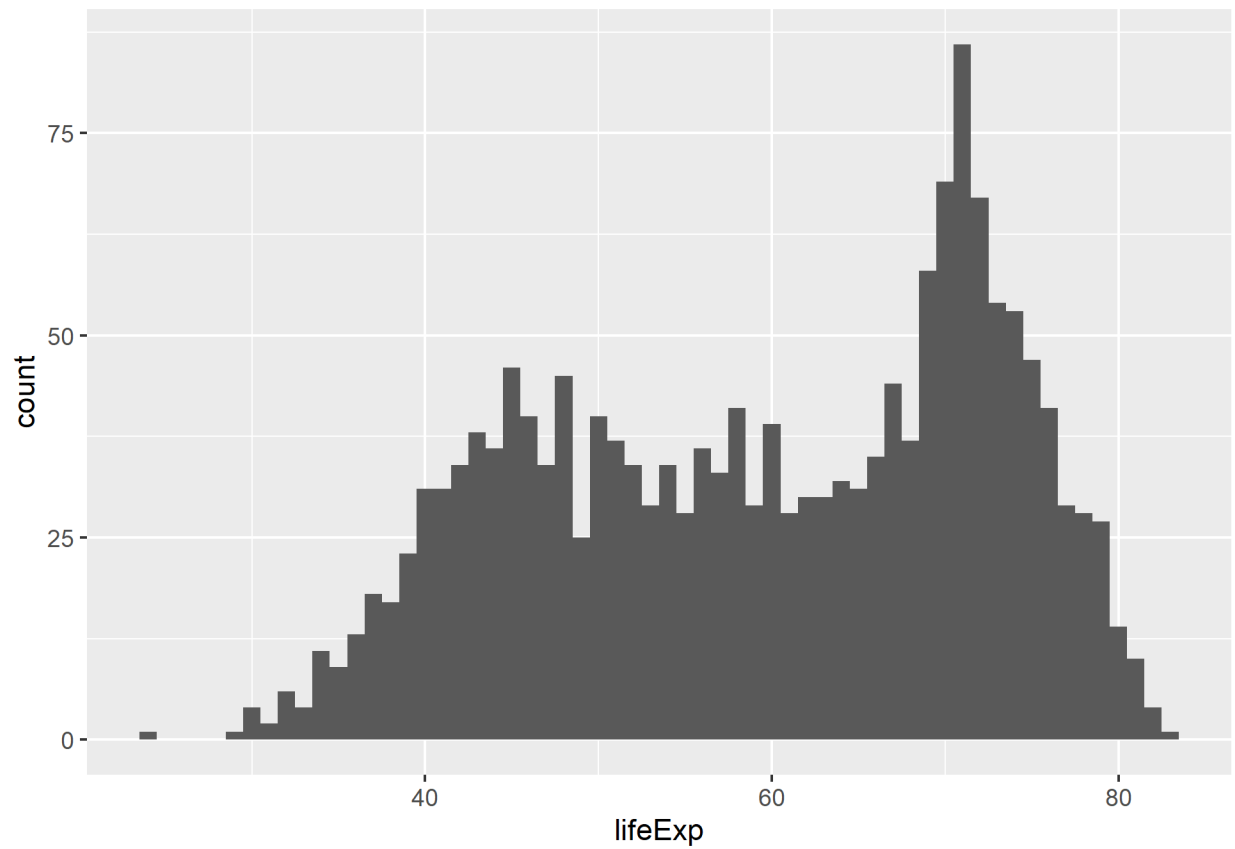
Når vi undersøker ulike plot kan vi skrive `plot_le +` i stedet for `ggplot(data, aes(lifeExp)) +` fordi sistnevnte informasjon er lagret i førstnevnte objekt. Dette sparer oss for en del kode. For en kontinuerlig variabel så kan det være mer nyttig med et histogram, som lager stolper for intervaller:

```
plot_le +  
  geom_histogram()
```



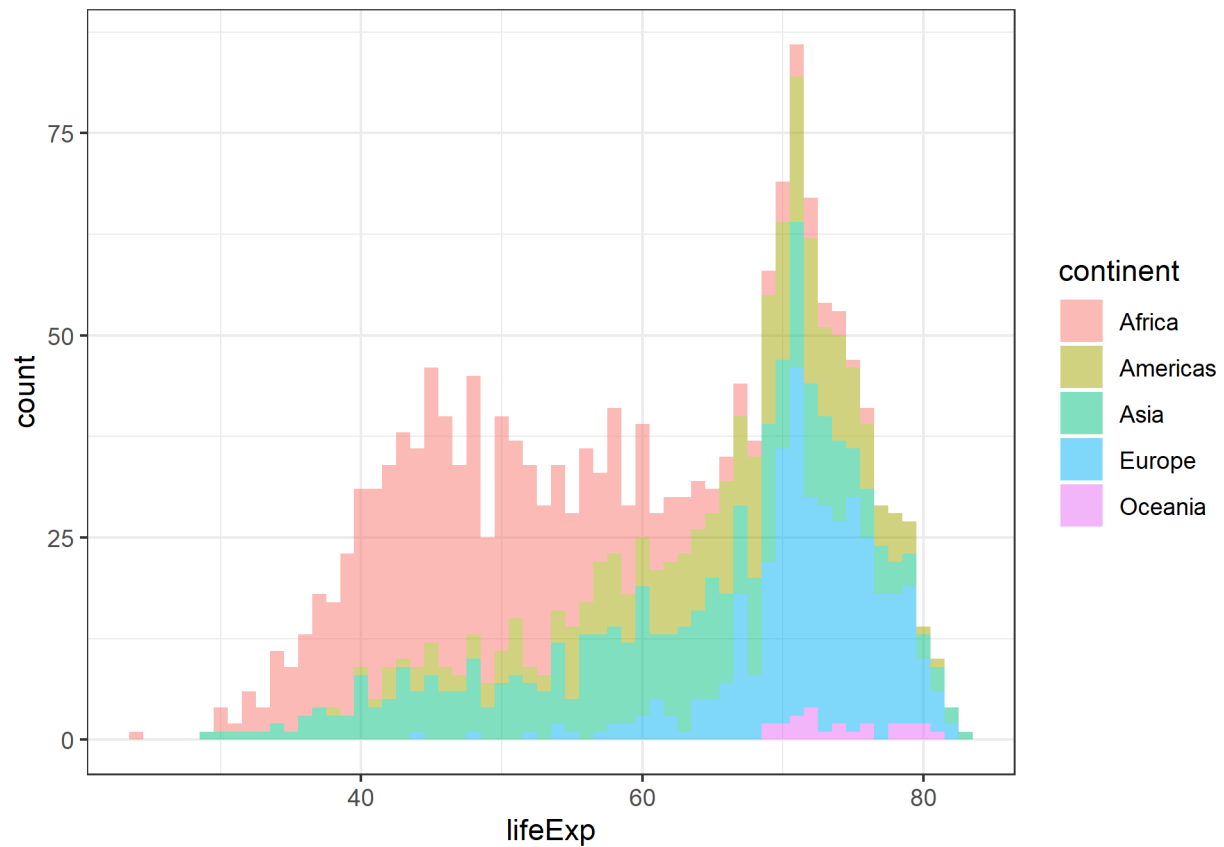
Hvis vi ikke sier noe annet, så kommer ggplot til å velge hvor stort hvert intervall skal være. Vi kan sette hver stolpe til å være ett år bred ved å skrive inn `binwidth = 1`:

```
plot_le +  
  geom_histogram(binwidth = 1)
```



Hvis vi vil se hvordan dette fordeler seg på hvert kontinent, så kan vi legge inn fill:

```
plot_le +  
  geom_histogram(aes(fill=continent),  
                 binwidth = 1,  
                 alpha = 0.5) +  
  theme_bw()
```

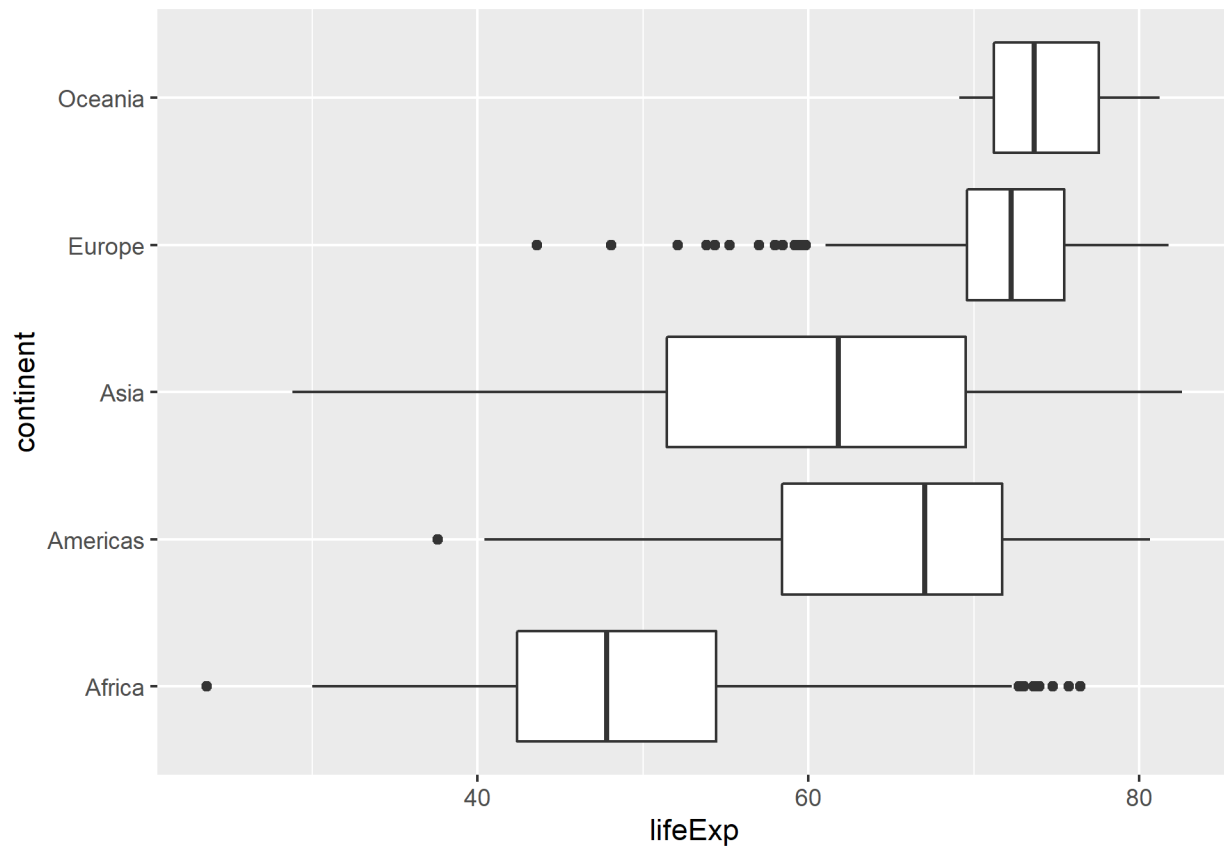


Dette gir oss en viss oversikt over hvordan forventet levealder varierer mellom kontinentene, men hvis vi vil se nærmere på dette så kan et boxplot for forventet levealder per kontinent være et bedre alternativ.

## Boxplot

for å lage et boxplot så spesifiserer vi y-aksen ved hjelp av `aes(y = continent)` i `geom_boxplot`:

```
plot_le +  
  geom_boxplot(aes(y = continent))
```

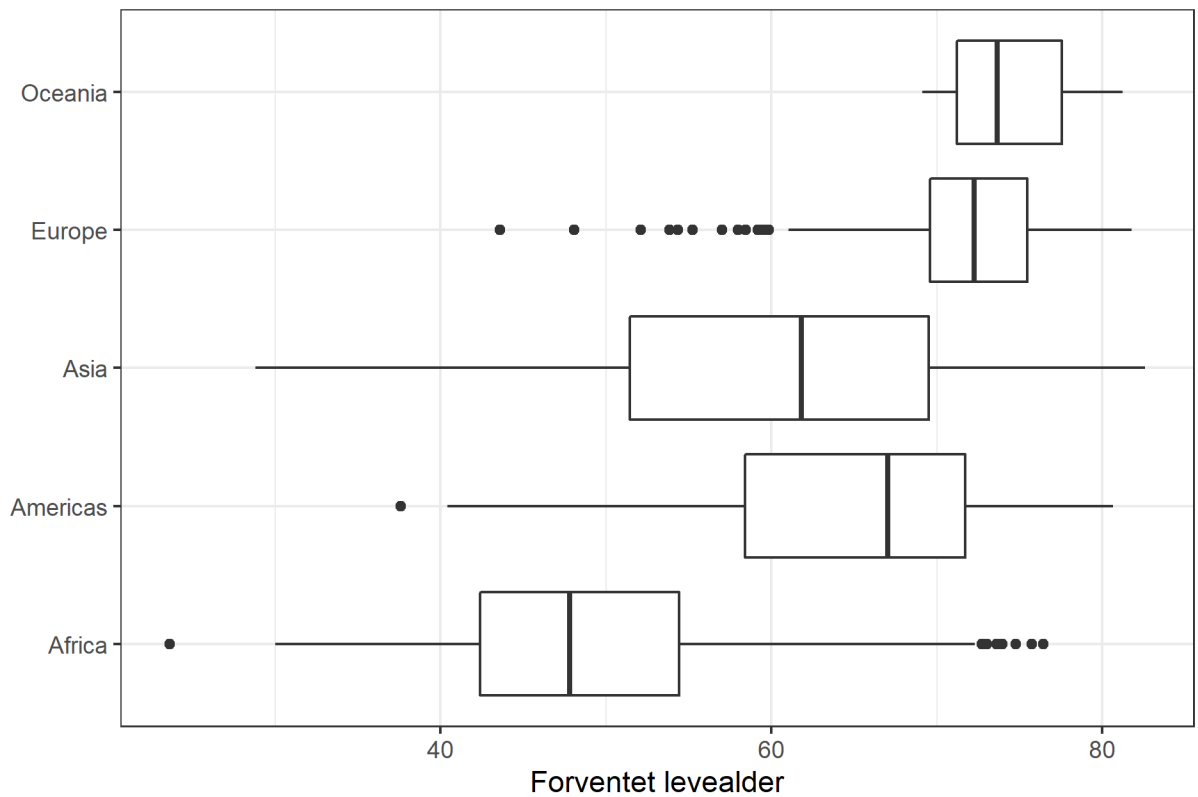


Her er et eksempel på hvor mye finere plottet kan bli med litt flere kodelinjer:

```
plot_le +
  geom_boxplot(aes(y = continent)) +
  theme_bw() +
  labs(title = "Boxplot for forventet levealder per kontinent",
        x = "Forventet levealder",
        y = "") +
  theme(legend.position = "none")
```



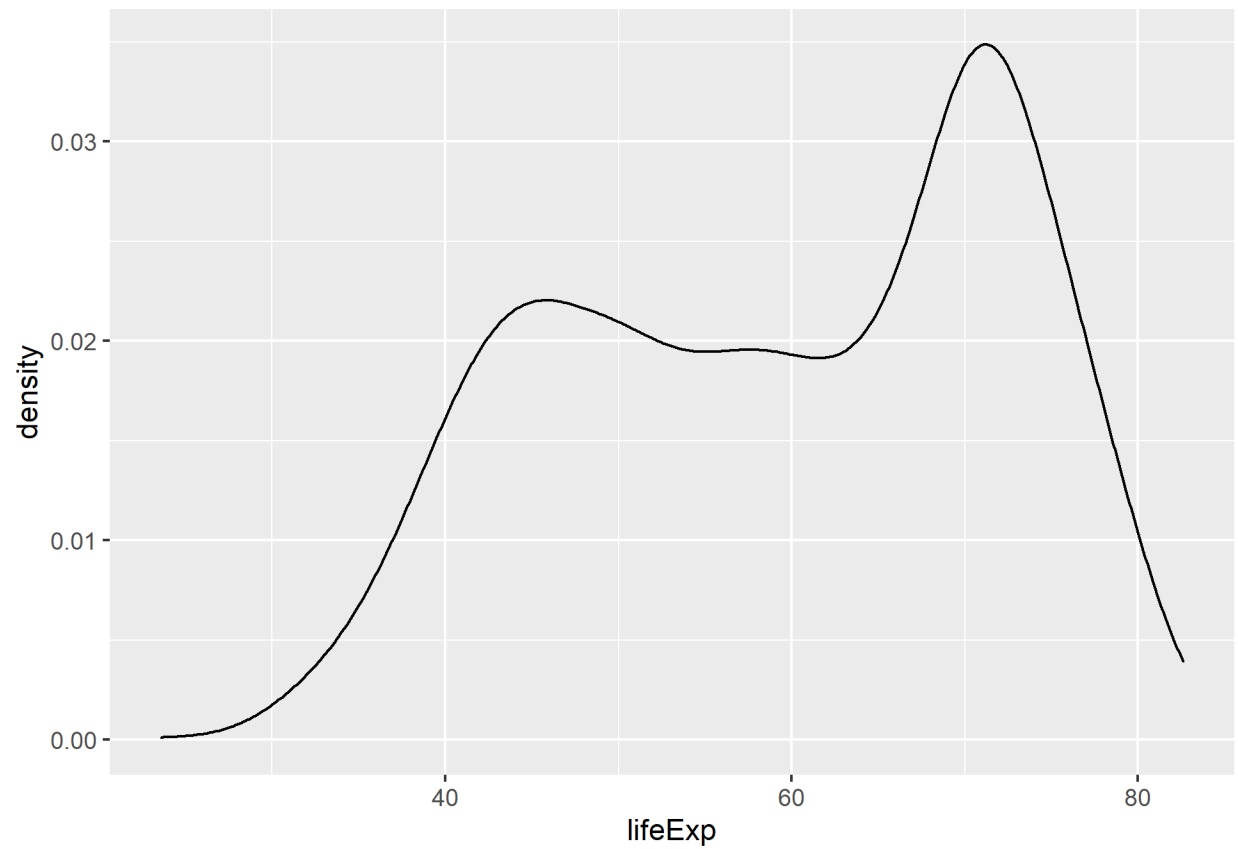
Boxplot for forventet levealder per kontinent



## Density plot/tetthetsplot

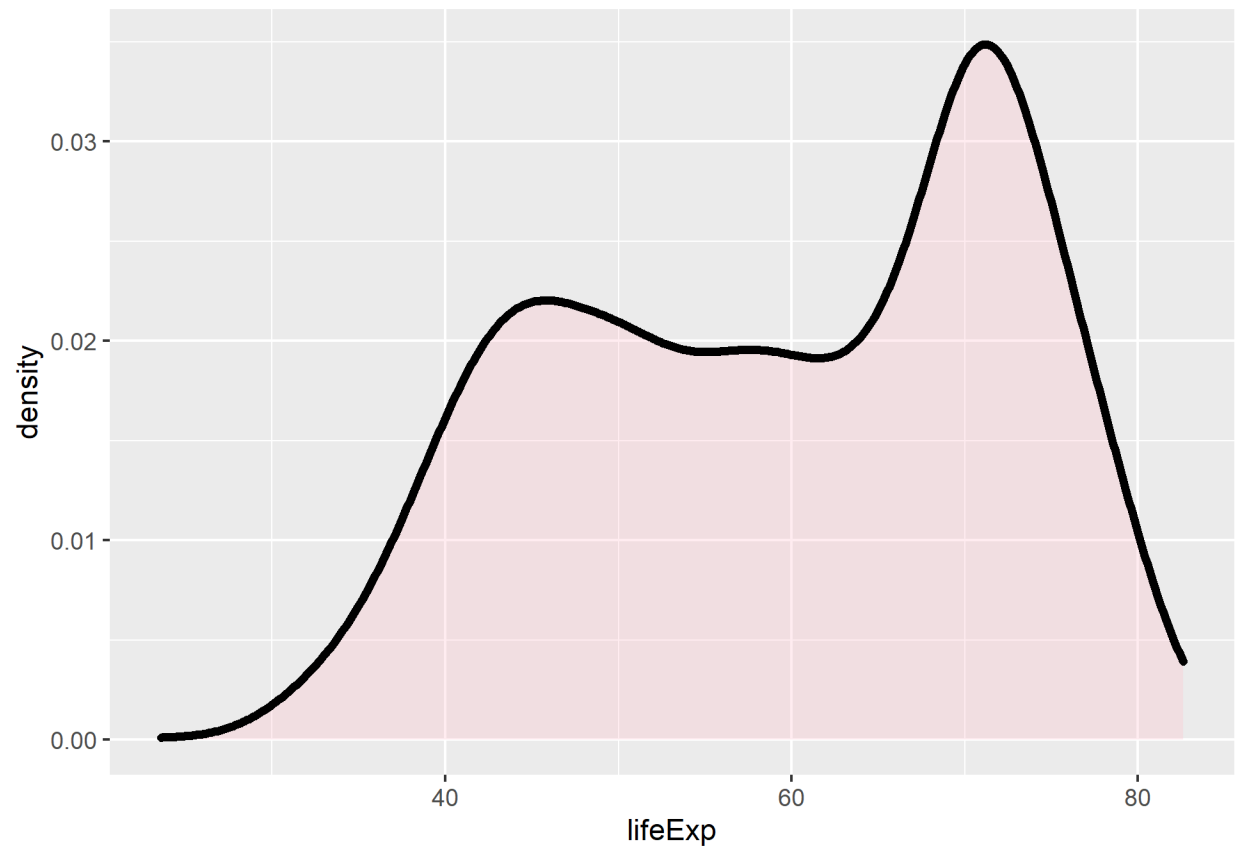
En annen metode kan være å bruke et density plot/tetthetsplott:

```
plot_le +  
  geom_density()
```



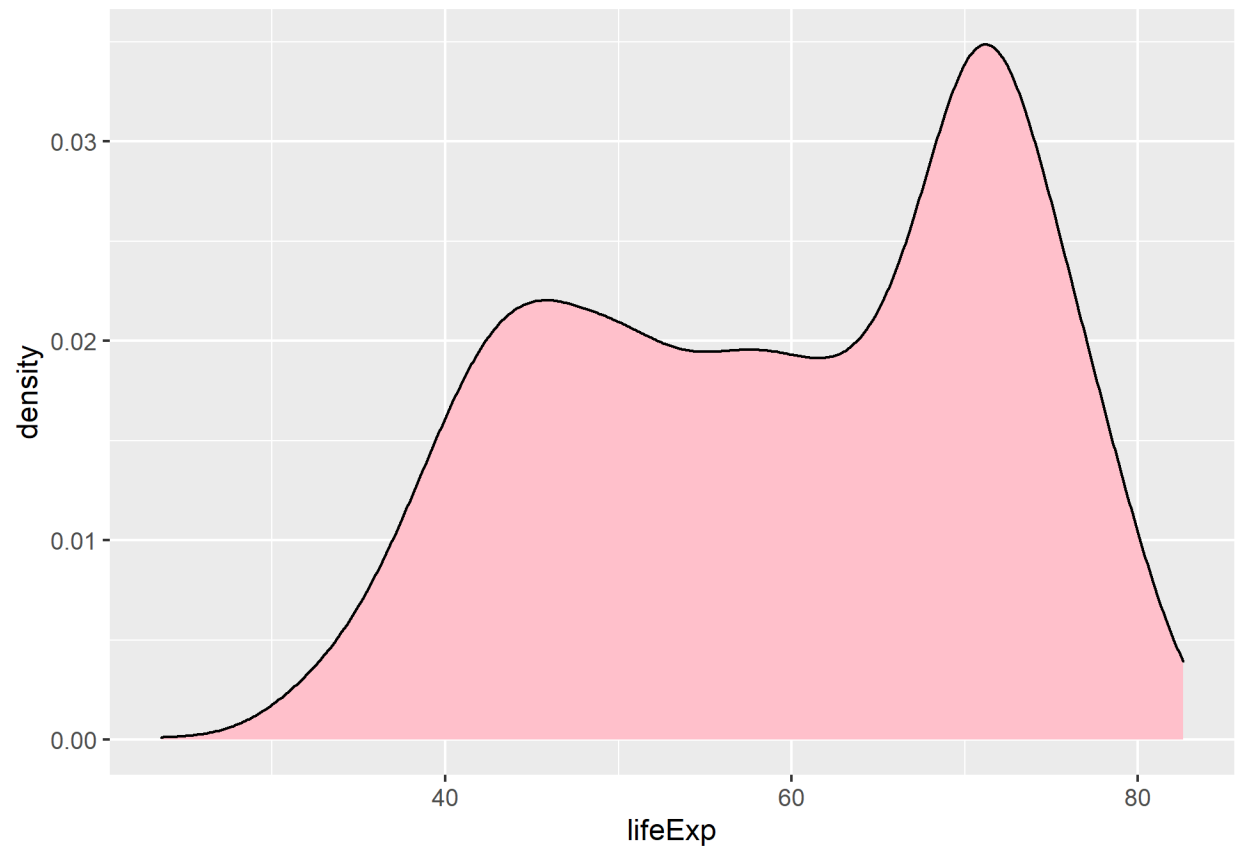
Inne i parentesen til plottypen kan vi legge til spesifikasjoner om tykkelse på linja, farge osv.

```
plot_le +  
  geom_density(size=1.5, fill="pink", alpha=0.3)
```



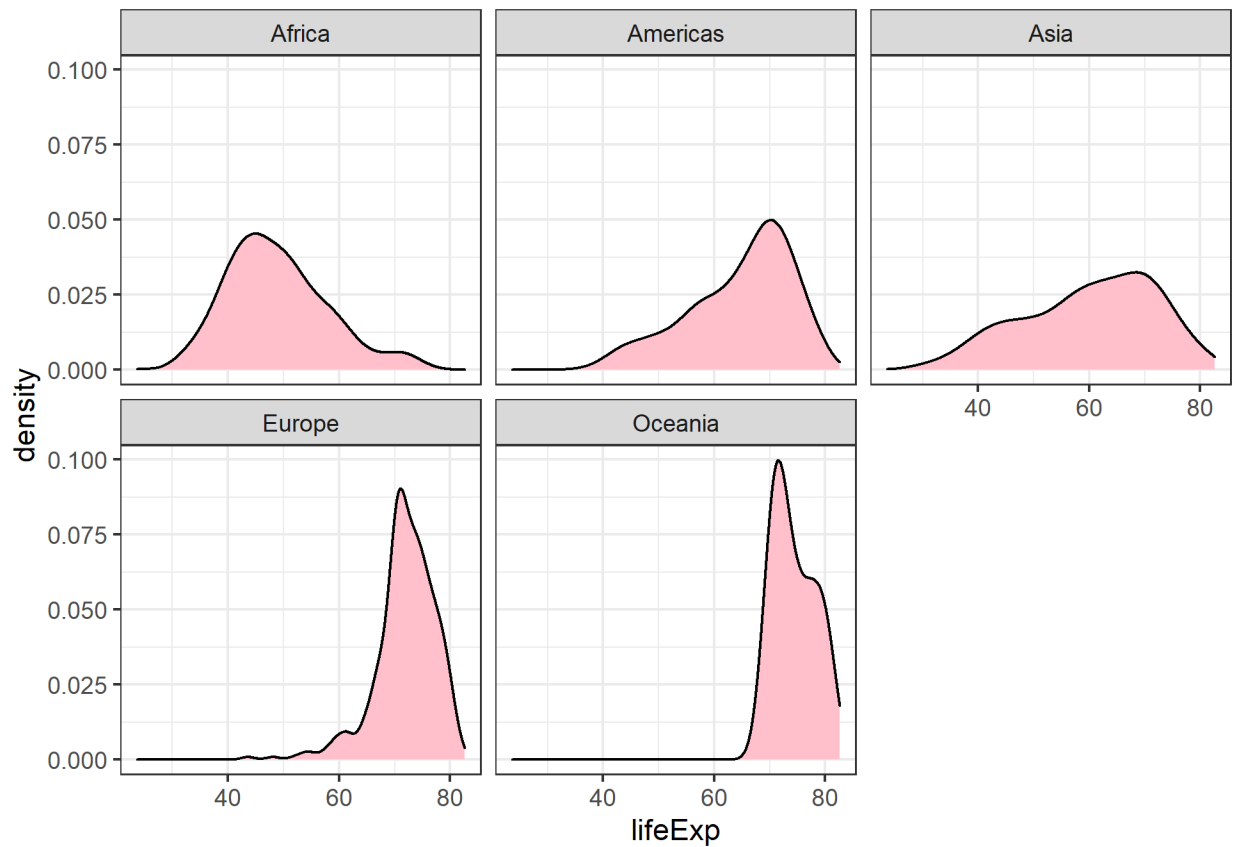
Hva skjer hvis vi endrer 1) alpha til 1 og 2) size til 0.5?

```
plot_le +  
  geom_density(size=0.5, fill="pink", alpha=1)
```



Hvis du ønsker å separere plottene, slik at de ulike kontinentene ikke ligger over hverandre, men får hvert sitt plott, så kan du legge til en `facet_wrap()`:

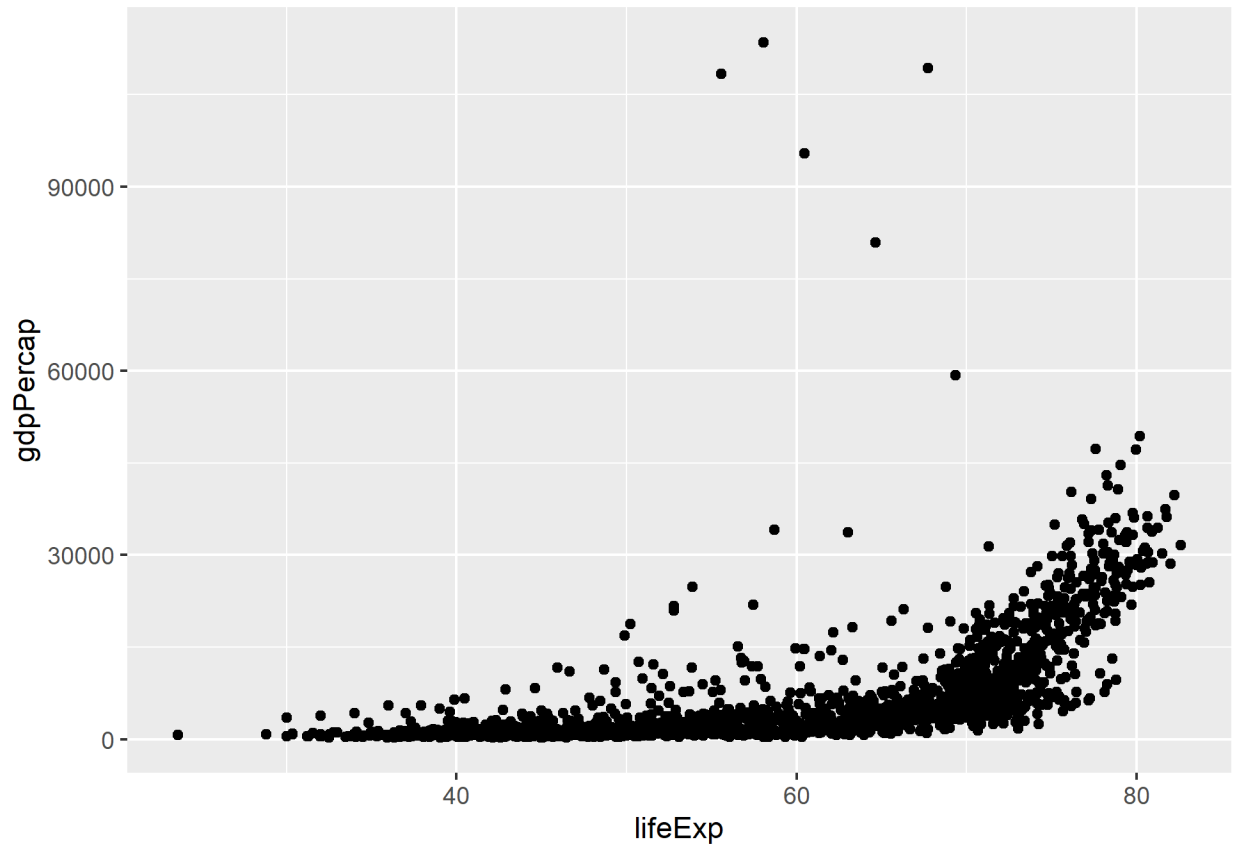
```
plot_le +  
  geom_density(size=0.5, fill="pink", alpha=1) +  
  facet_wrap(vars(continent)) +  
  theme_bw()
```



## Spredningsplot

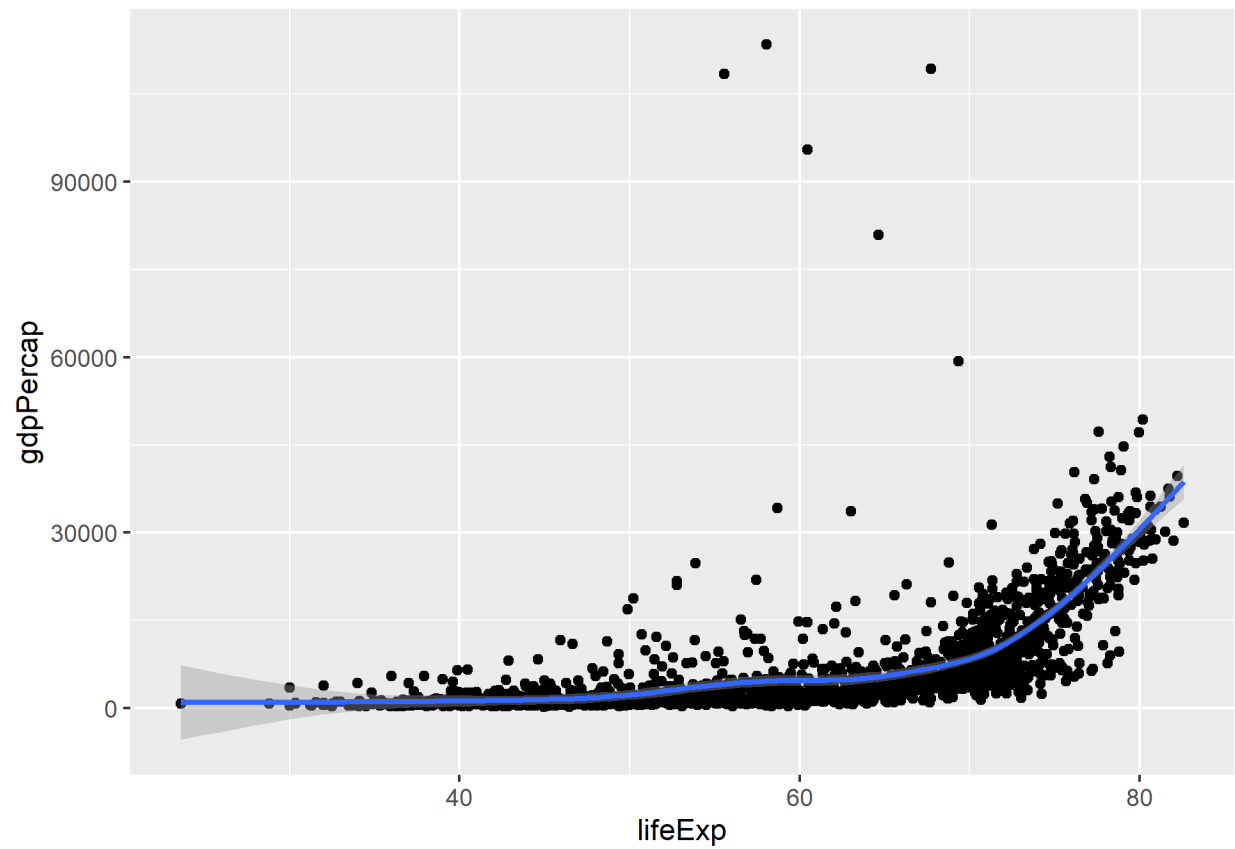
Med et spredningsplot (scatterplot) kan vi vise sammenhengen mellom to kontinuerlige variabler. Her må vi gi ggplot informasjon om hvilken variabel som skal mappes på x-aksen og på y-aksen. For å lage scatterplottet bruker vi `geom_point()`.

```
ggplot(data,
  aes(x=lifeExp, y=gdpPercap)) +
  geom_point()
```



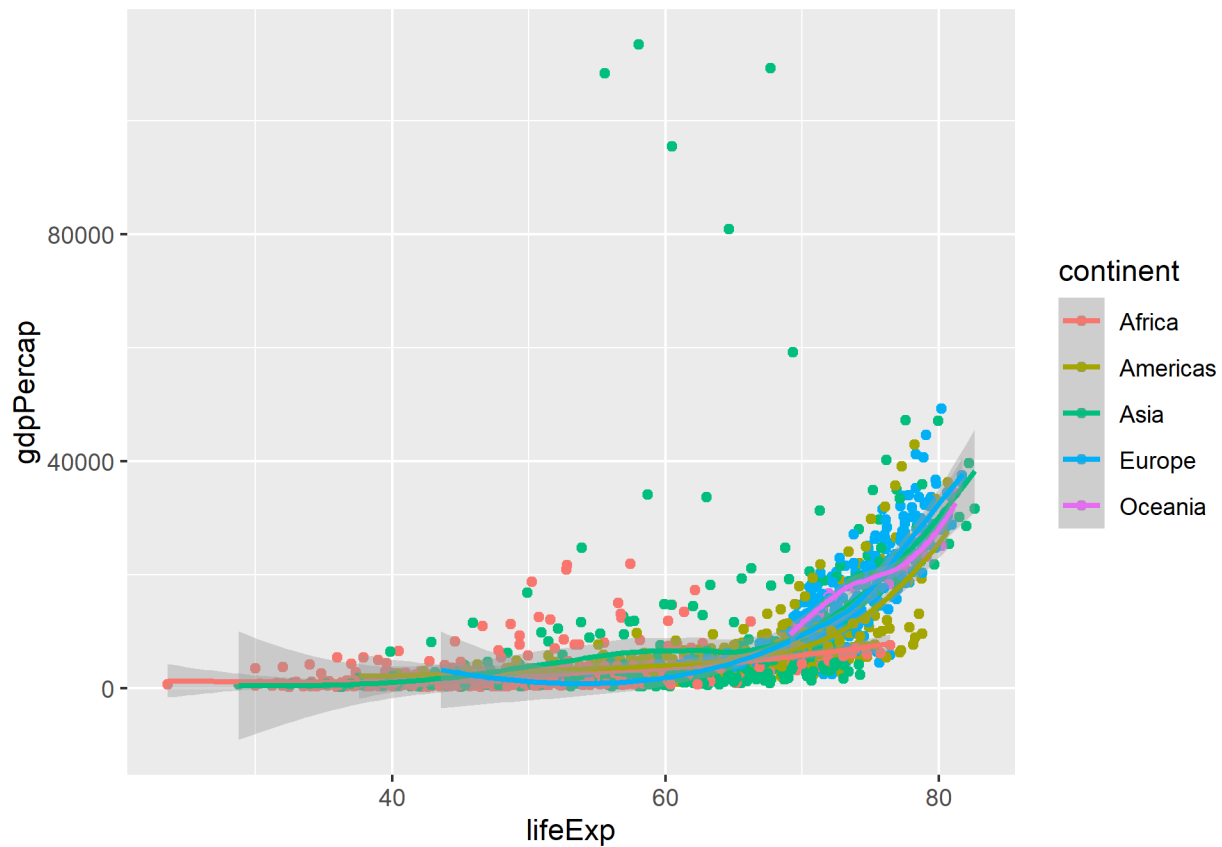
Vi kan legge til en linje som viser gjennomsnittet i observasjonene ved å bruke `geom_smooth`:

```
ggplot(data, aes(x=lifeExp, y=gdpPercap)) +  
  geom_point() +  
  geom_smooth()
```



Skille kontinentene fra hverandre ved bruk av ulike farger:

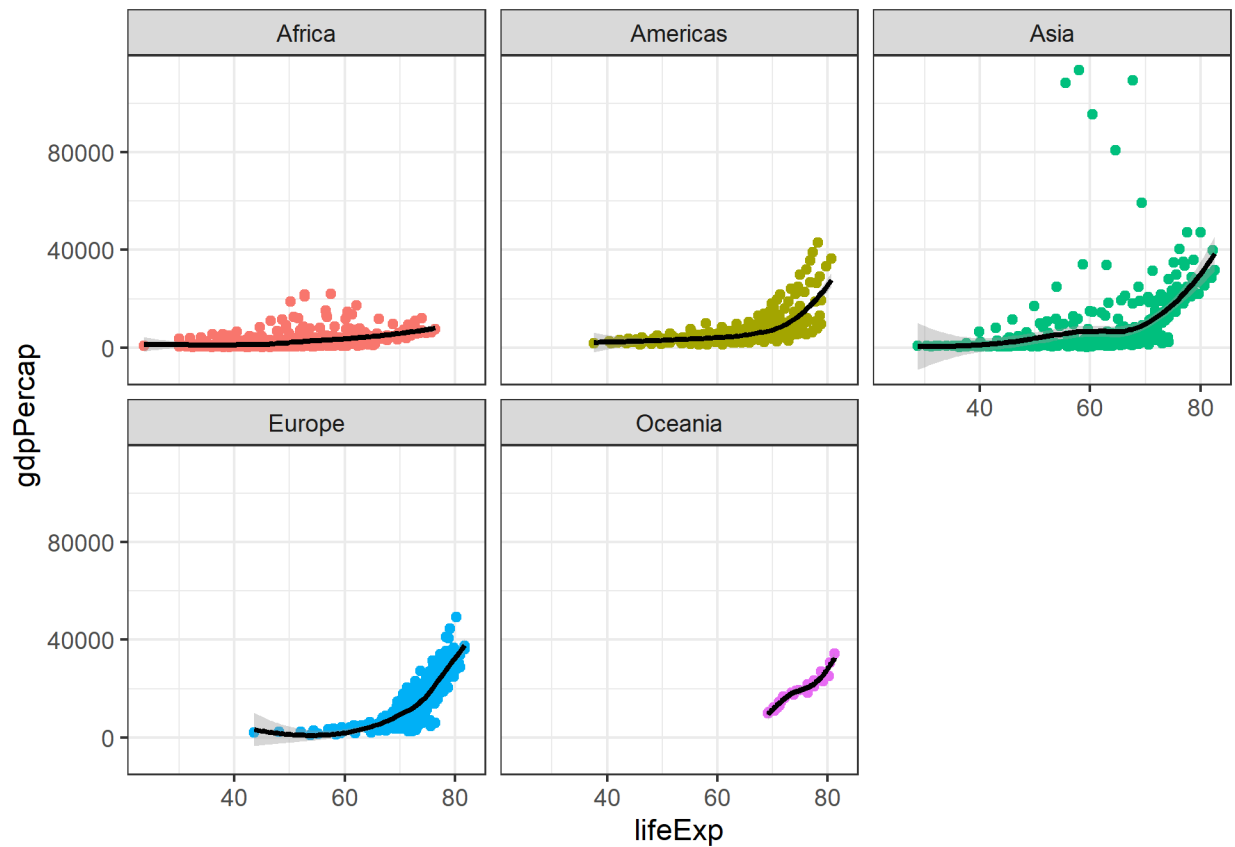
```
ggplot(data,
  aes(x=lifeExp, y=gdpPercap, col=continent)) +
  geom_point() +
  geom_smooth()
```



Vi kan legge inn en `facet_wrap()` for at det skal se litt ryddigere ut, og samtidig gi `geom_smooth()` en farge som vises bedre.

```
ggplot(data,
  aes(x = lifeExp, y = gdpPercap, col = continent)) +
  geom_point() +
  geom_smooth(colour="black") +
  facet_wrap(~continent) +
  theme_bw() +
  theme(legend.position = "none")
```

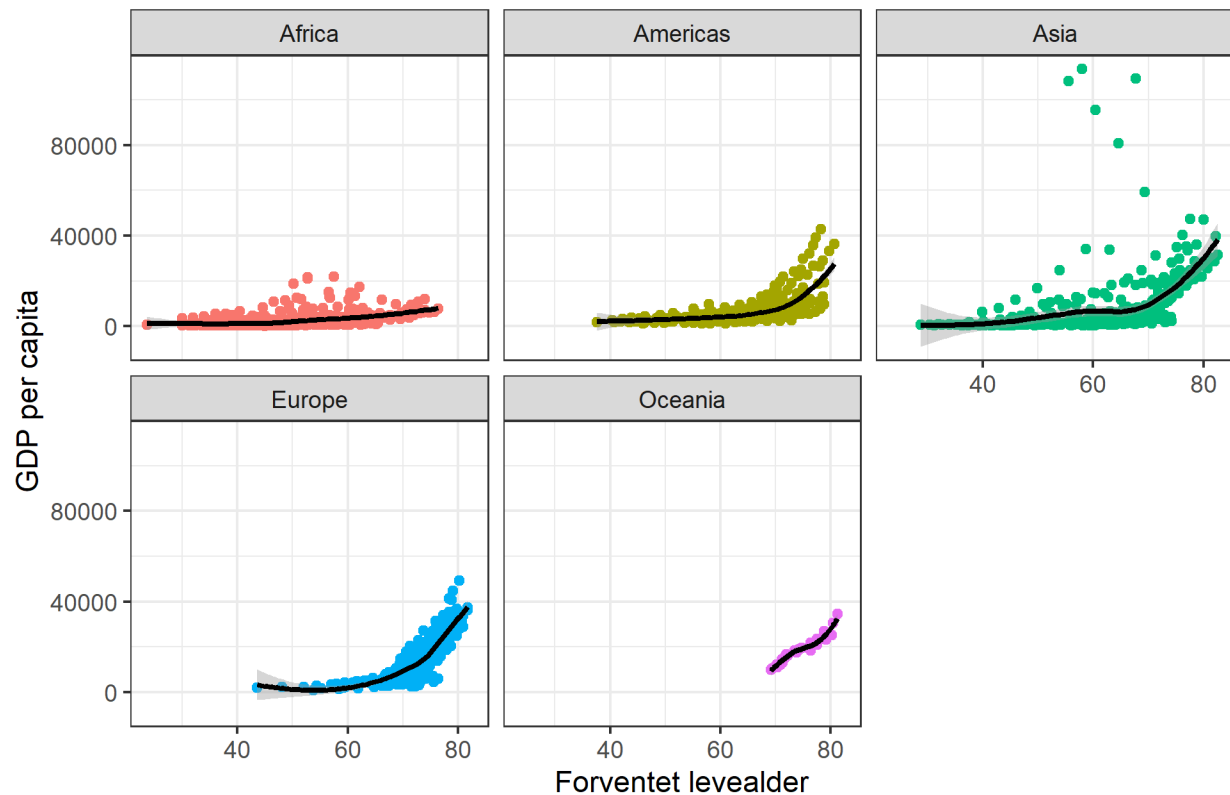




Her kan vi repetere hvordan vi legger til nye titler til x- og y-aksen, samt tittel for hele plottet:

```
ggplot(data,
  aes(x = lifeExp, y = gdpPercap, col = continent)) +
  geom_point() +
  geom_smooth(colour="black") +
  facet_wrap(vars(continent)) +
  labs(x = "Forventet levealder",
    y = "GDP per capita",
    title = "Et plot med Gapminderdata") +
  theme_bw() +
  theme(legend.position = "none")
```

## Et plot med Gapminderdata



## Lagring av plott

Du kan lagre plottene dine i working directory ved å bruke `ggsave()`. For å lagre det siste plottet du har kjørt i R, gjør følgende: Bruk `ggsave()`, gi plottet et navn, og spesifiser filtypen du vil lagre som:

```
# For png-format:
ggsave(filename = "gdplevealder.png")

# For pdf-format:
ggsave(filename = "gdplevealder.pdf")

# For jpeg-format:
ggsave(filename = "gdplevealder.jpeg")

# Du trenger egentlig ikke skrive "filename" så lenge du bruker ggsave:
ggsave("gdplevealder.png")
```

For å lagre et plott som er et objekt gjør du følgende:

```
ggsave("navnpåplott.filtipe", plot = objektnavn)

# For eksempel:
ggsave("histogram_gapminder.png", plot = gm_h)
```

Ressurser for ggplot2:

<https://ggplot2.tidyverse.org/>

<https://socviz.co/makeplot.html#makeplot>

<http://euclid.psych.yorku.ca/www/psy6135/tutorials/gapminder.html>

<https://www.tidytuesday.com/>

<https://twitter.com/hashtag/tidytuesday>