

Datorlaboration 6

Måns Magnusson

VT 2015

Instruktioner

- **Allmänt**

Vid tidigare laborationer har vi använt SAS för att dra urval, studier av teoretiska egenskaper hos estimatorer och allokerat urval. Nu ska vi fokusera på att med R göra analyser, bortfallhantering och estimation då vi fått in data från en surveyundersökning.

- **Datamaterial**

Vilket datamaterial som ska användas framgår av respektive uppgift. Allt datamaterial finns att tillgå [här](#) om inte annat anges. För att ladda ned datan, klicka på den datafil du vill ladda ned och klicka sedan på "Raw" med högra musknappen och klicka "Spara länk som..." .

- **Hjälpmaterial**

Behöver ni hjälp med att använda R-paketet survey finns, utöver dokumentationen, extra material här. Det finns också en bok *Complex surveys : a guide to analysis using R* som behandlar analyser med surveypaketet i R.

- Det är tillåtet att diskutera med andra men att plagiera andra grupper är **inte tillåtet**.
 - Utgå från mallen för laborationsrapporter som går att ladda ned som [LyX](#) eller [PDF](#). Samtliga labbrapporter ska lämnas in **i PDF-format via LISAM**.
 - Deadline för labben framgår på [kurshemsidan](#).
 - **Laborationsrapport**
Rapporten ska innehålla den kod ni kört, eventuella resultat samt svara på de frågor som finns i laborationen.
-

Innehåll

1 Förberedelser	3
1.1 Läsa in paket som ska användas	3
1.2 Ladda ned och läs in Survey 2010	3
1.3 Ladda ned och läs in agpop.dat	4
1.4 Kartfiler	4
2 Grafik	4
2.1 Kategoriska variabler	5
2.2 Scatterplots i surveyer	5
2.3 Surveyer och ggplot2	8
3 Surveyer och kartor	8
3.1 Skapa kartor med ggmap	12
3.2 Visualisera geotaggad data på karta	15
3.3 Kombinera spatiala data med olika projektioner/geografiska referenssystem	18
Referenser	21
4 Bilaga: Repetition av grunderna i ggplot2	22
4.1 Grunden i ggplot2	22
4.2 Skapa en ggplot (linje eller scatter)	22
4.3 Enklare modifikationer av ett ggplot-objekt	23
4.4 Barplot, histogram och boxplot	24
4.5 Histogram	25
4.6 Boxplot	26
4.7 Grafiska teman/profiler	26

1 Förberedelser

1.1 Läsa in paket som ska användas

Som ett första steg behöver vi installera en hel del paket som kan hantera och arbeta med spatiala datastrukturer. Vi kommer därför behöva den grundläggande funktionaliteten i följande R-paket:

Paket	Innehåll
<code>maptools</code>	Verktyg för att använda och läsa in spatiala data
<code>rgeos</code>	Kopplingar från R till GEOS (Geometry engine open source)
<code>rgdal</code>	Verktyg för att konvertera mellan projektioner
<code>ggmap</code>	Spatial visualisering med google maps och opens street map.
<code>sp</code>	Klasser och metoder för spatiala datastrukturer

När det gäller paketen `rgeos` och `rgdal` bygger dessa paket dessutom på andra implementationer i språk som Java och C++. Det kan göra det något mer krångligt att installera dessa paket i vissa fall. För att installera dessa paket på Mac OS kan jag rekommendera [\[denna\]](#) och [\[denna\]](#) guide för installation.

- Läs dessutom in följande paket i R:

```
library(survey)
library(rmeta)
library(hexbin)
library(ggplot2)
library(pxweb)
```

Om det inte går att läsa in paketet (ex. du har en egen dator) behöver du installera paketet först. Det kräver internetanslutning och då använder du exempelvis följande kod.

```
install.packages("survey")
```

1.2 Ladda ned och läs in Survey 2010

- Vi har fått tillgång till det datamaterial som ligger till grund för boken *Den svenska väljaren* Hagevi (2011). Ett mindre urval av variablene i studien har sparats som en R-fil. Ladda ned filen från kurshemsidan och läs in den i R med följande funktion

```
load("svy2010.Rdata")
```

- Det går också att ladda in filen direkt från webben med `repmis`-paketet:

```
library(repmis)
data_path <- "https://raw.github.com/MansMeg/KursSvyMeth/master/Labs/DataFiles/svy2010.Rdata"
source_data(data_path)

## Downloading data from: https://raw.github.com/MansMeg/KursSvyMeth/master/Labs/DataFiles/svy2010.Rdata
##
## SHA-1 hash of the downloaded data file is:
## d11fe835c8306f4746b9f950bc128985049815e2

## [1] "svy2010"
```

- Du ska nu ha läst in en fil med 1613 observationer och 70 variabler. Information om respektive variabler finns i dokumentet **KodbokSurvey2010.pdf** som finns på samma ställe som datamaterialet, dock finns inte alla variabler med i datasettet.
- Vi ska nu skapa två kontinuerliga variabler i datasettet `fathAge` och `mothAge` på följande sätt:

```
svy2010$fathAge <- svy2010$FR37 - svy2010$FR40_1  
svy2010$fathAge[abs(svy2010$fathAge) > 100 | svy2010$fathAge < 0] <- NA  
svy2010$mothAge <- svy2010$FR37 - svy2010$FR40_2  
svy2010$mothAge[abs(svy2010$mothAge) > 100 | svy2010$mothAge < 0] <- NA
```

1.3 Ladda ned och läs in agpop.dat

- Ladda ned och läs in `agpop.dat` i R, vilket kan göras med följande kod:

```
agpop<-read.table("agpop.dat",header=TRUE,sep=",")
```

- I detta fall kan vi självklart använda `repmis`-paketet.

```
agpop <- source_data(url = "https://raw.github.com/MansMeg/KursSvyMeth/master/Labs/DataFiles/agpop.dat"  
## Downloading data from: https://raw.github.com/MansMeg/KursSvyMeth/master/Labs/DataFiles/agpop.dat  
##  
## SHA-1 hash of the downloaded data file is:  
## fdd78ace764a7b61254f073564ab50968cdd9375
```

- Glöm inte bort att ta bort bortfallet för variablerna `ACRES92` och `ACRES87` i `agpop`. För att ta bort saknade värden kan följande kod användas:

```
agpop<-agpop[agpop$ACRES92>0 & agpop$ACRES87>0,]
```

1.4 Kartfiler

I denna laboration behöver vi också ladda ned spatiala data vi ska använda oss av. Detta är exempel på svenska data över Sveriges läns- och kommunindelning i form av shape-fil. Vi laddar ned dessa shapefiler med paketet `downloader` (filen finns också på kurshemsidan för manuell nedladdning).

```
library(downloader)  
# Counties  
lan_remote <- "https://raw.github.com/MansMeg/KursSvyMeth/master/Labs/DataFiles/Lan_SCB.zip"  
lan_local <- "Lan_SCB.zip"  
download(lan_remote, destfile = lan_local)  
unzip(lan_local)  
  
# Municipalities  
kommuner_remote <- "https://raw.github.com/MansMeg/KursSvyMeth/master/Labs/DataFiles/Kommun_SCB.zip"  
kommuner_local <- "Kommun_SCB.zip"  
download(kommuner_remote, destfile = kommuner_local)  
unzip(kommuner_local)
```

2 Grafik

Vi ska nu skapa grafik baserat på surveydata. Eftersom vikter är något som påverkar våra variabler och vår grafik väljer vi därfor att dels använda en stratifierad urvalsdesign som exempel och dels data från Survey 2010. Visualisering är en konstform så försök göra din grafik så tilltalade som möjligt. Använd gärna färger.

Börja med att skapa de två surveyobjekten vi ska använda, dels baserat på `agpop` och dels baserat på Survey 2010.

Dra ett (Neymanallokerat) stratifierat urval från `agpop` och skapa ett surveyobjekt på samma sätt som gjordes i laboration D4. Jag döper mitt objekt till `agSTRAT`.

Skapa sedan ett surveyobjekt baserat på datamaterialet i Survey 2010, jag kommer kalla detta objekt `svy2010design`.

2.1 Kategoriska variabler

- a) Vi börjar med att producera enklare grafik baserat på surveyresultaten. Denna grafik är inte specifik för surveyundersökningar, men nu skapar vi grafer baserat på våra skattningar av populationen istället på hur data ser ut i vårt urval, d.v.s. vi beaktar vikterna i vår grafik.

```
bys <- svyby(formula=~FARMS92, by=~REGION, design=agSTRAT, FUN=svytotals)
barplot(bys)
dotchart(bys)
```

Vi kan också inkludera både antalet observationer och vår osäkerhet med en så kallad `forestplot()` från `rmeta`-paketet.

```
labelMatrix <- matrix(rownames(bys), nrow=4)
forestplot(labeltext=labelMatrix,
            mean = coef(bys),
            lower= coef(bys) - 1.96 * SE(bys),
            upper = coef(bys) + 1.96 * SE(bys))
```

- b) Vi ska nu gå in på de mer specifika funktioner som finns för surveydata i surveypaketet. Exempelvis kan vi använda `svyboxplot()` för att skapa boxplots för populationen vi vill estimera. Funktionen estimerar kvantilerna i populationen och skapar en boxplot baserat på dessa estimat.

```
svyboxplot(ACRES92 ~ REGION, design=agSTRAT)
```

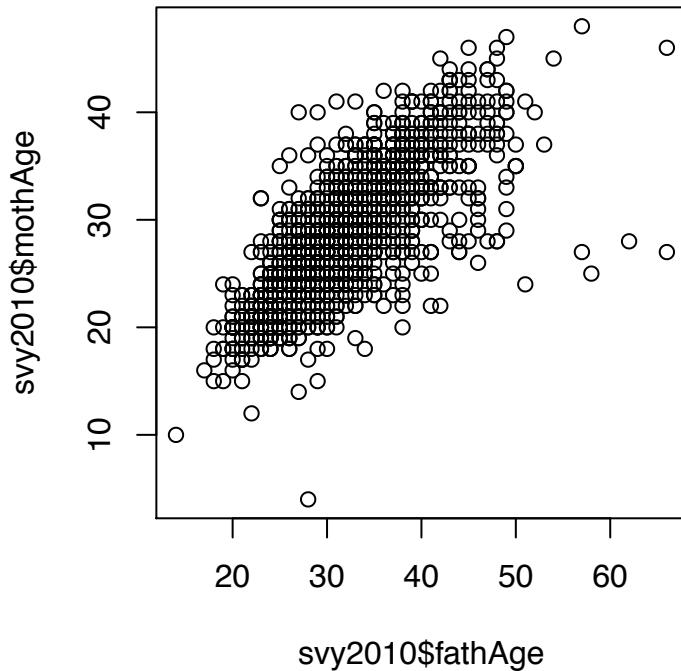
På ett liknande sätt fungerar funktionen `svyhist()` som skapar ett histogram baserat på de viktade observationerna.

```
svyhist(~ ACRES92, design=agSTRAT)
```

2.2 Scatterplots i surveyer

- a) Som nämnts tidigare är ett (delikat) problem i surveyer ofta att antalet observationer är mycket stort. Det gör att scatterplots ofta kan bli helt överläckta med datapunkter vilket gör det svårt att se resultaten. Nedan är ett exempel på scatterplot där det är mycket svårt att se fördelningen då punkterna ligger ”ovanpå” varandra.

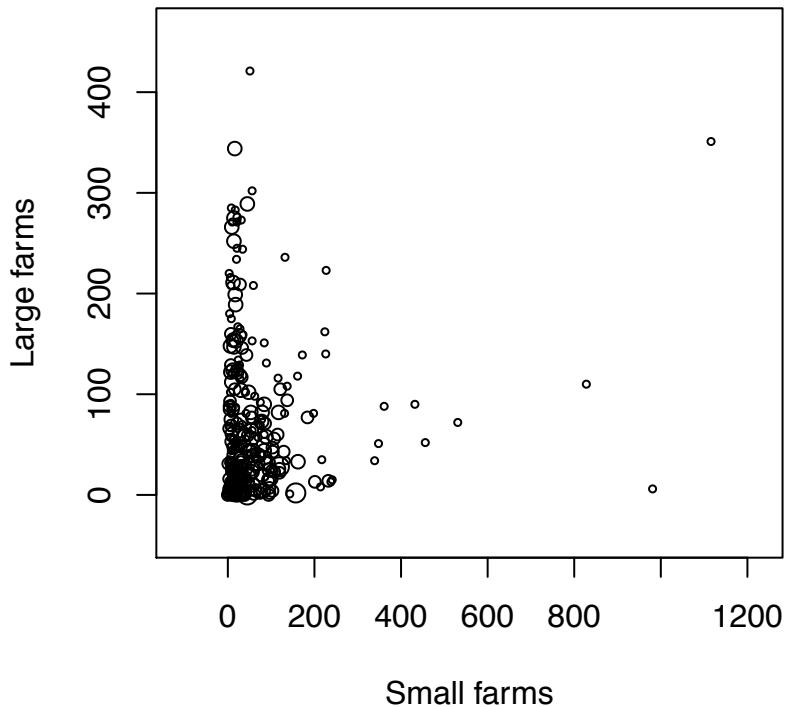
```
plot(svy2010$fathAge, svy2010$mothAge)
```



Vi ska nu pröva olika former av alternativ för surveydata.

Dessa plottar kan visas på olika sätt. Dels kan vikterna tas med i beaktande och explicit användas i en scatterplot. Detta görs med funktionen `svyplot()` på följande sätt. Nedan skapas en “bubbleplot” men där storleken på bublorna är baserade på vikterna i datamaterialet.

```
svyplot(LARGEF92~SMALLF92, design=agSTRAT, style="bubble", xlab="Small farms", ylab="Large farms")
```



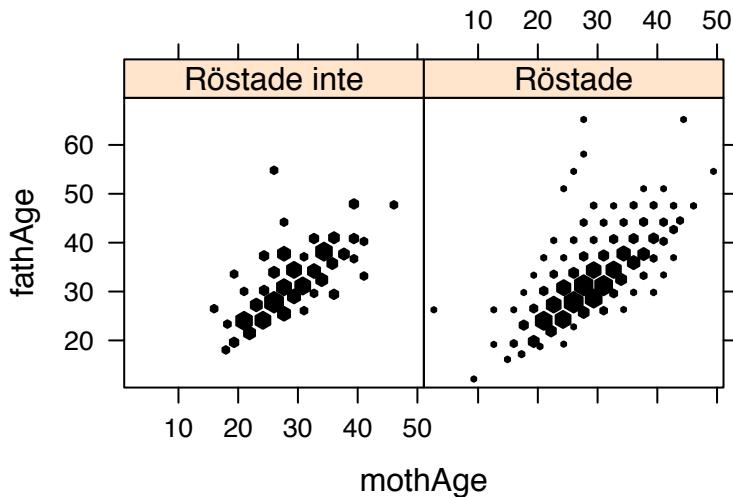
Nedan är exempel på andra metoder för att producera plottar för surveydata (eller generellt stora data).

```
svyplot(fathAge~mothAge, design = svy2010design, style="transparent", pch=19, alpha=c(0,.1))
svyplot(fathAge~mothAge, design = svy2010design, style="hex")
svyplot(fathAge~mothAge, design = svy2010design, style="grayhex")
```

Skapa plottarna ovan och inkludera dem i din rapport. Gör dem så snygga du kan (se hjälpen) med labels på x- och y-axeln. Vilka plottar föredrar du och varför?

b) Till sist ska vi pröva att skapa så kallade “conditioning plots” d.v.s. scatterplots för olika grupper. Dessa plots påminner om funktionerna ovan, dock med tillägget | som indikerar för vilken kategorisk variabel ska plottarna delas upp. Nedan är ett exempel.

```
svycoplot(fathAge~mothAge | Valdeltagande, design = svy2010design, style="hex")
## Loading required package: lattice
```



Välj ut en variabel och skapa en så snygg conditioning plot du kan. Använd hjälpen för att se hur kan styra olika delar.

2.3 Surveyer och ggplot2

Den grafik ovan som kan produceras med surveypaketet är väl genomtänkt, men knappast särskilt vacker i sitt utförande. Vill vi istället skapa grafik för surveyer med `ggplot2` krävs något mer arbete. I slutet av denna laboration finns en repetition av hur vi skapar grafik med `ggplot2`.

Det första steget vi behöver ta är att vi behöver plocka ut surveyvikterna och surveydatan för att skapa en `data.frame` vi kan använda med `ggplot2`.

```
my_df <- cbind(agSTRAT$variables, data.frame(w=weights(agSTRAT)))
```

Nu har vis kapat ett dataset med vikter. Vi kan nu skapa grafik på vanligt sätt i `ggplot2`, men vi anger argumentet `weight` i `aesthetic` för `ggplot2`. Pröva koden nedan.

```
ggplot(data=my_df) + geom_point(aes(x = LARGEF92, y = SMALLF92, size=w, color=REGION))

ggplot(data=my_df) +
  stat_bin2d(aes(x = LARGEF92, y = SMALLF92, weight=w),
             size = .5, bins = 20, alpha = 0.4)

ggplot(data=my_df) + aes(x = SMALLF92, weight=w) + geom_histogram()

ggplot(data=my_df) + aes(y = SMALLF92, x=REGION, weight=w) + geom_boxplot()

ggplot(data=my_df) + aes(x = REGION, weight=w) + geom_bar()
```

Använd nu `ggplot2` och visualisera datamaterialet Survey 2010 med `ggplots` motsvarigheter till grafiken i 2.1 och 2.2 ovan.

3 Surveyer och kartor

- a) I surveyer och liknande undersökningar är det inte sällsynt att det finns ett intresse av producera kartor av det material som samlas in. Vi kommer därför gå igenom den mest grundläggande funktionalitet för att skapa kartor.

För att skapa kartor behöver vi en så kallad shapefil. Dessa innehåller geografisk data i vektoriserat format vilket vi kan läsa in och modifiera i R. Vektoriserade geografiska data innebär att vår data representeras som punkter, vektorer eller polygoner. Mer information om shapefiler finns [[här](#)] och om vektoriserad GIS-information finns [[här](#)].

Shapefiler som är aktuella för just era problem beror på detaljeringsgrad, men församlingsgränser eller kommunindelningsgränser av intresse finns på de flesta kommuner och länsstyrelser. Ofta finns informationen att tillgå gratis som shapefiler. Exempelvis tillhandahåller valmyndigheten samtliga valdistrikts i form av en shapefil.

Läs in shapefilen med hjälp av `readShapePoly()` på följande sätt i R.

```
library(rgdal)

## Loading required package: sp
## rgdal: version: 0.8-16, (SVN revision 498)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 1.11.0, released 2014/04/16
## Path to GDAL shared files: /Users/manma97/Library/R/3.1/library/rgdal/gdal
## Loaded PROJ.4 runtime: Rel. 4.8.0, 6 March 2012, [PJ_VERSION: 480]
## Path to PROJ.4 shared files: /Users/manma97/Library/R/3.1/library/rgdal/proj

library(rgeos)

## rgeos version: 0.3-4, (SVN revision 438)
## GEOS runtime version: 3.4.2-CAPI-1.8.2 r3921
## Polygon checking: TRUE

library(maptools)

## Checking rgeos availability: TRUE

swe_county <- readShapePoly("Lan_SCB/Lan_SCB_07.shp")
```

Vi har nu läst in shapefilen i R. Vi kan få en snabb beskrivning av våra geografiska data med `summary()`

```
class(swe_county)
summary(swe_county)
```

Som ett första steg kan vi också skapa en snabb grafisk plot baserat på vårt spatiala objekt.

```
plot(swe_county)
```



Nu har vi läst in och studerat vår första shapefil!

b) Vi ska nu skapa statistik på karta baserat på surveyestimat. Det första steget är att skapa den variabel vi vill beskriva med kartan. Jag vill analysera andelen som vill skära ned på den offentliga sektorn i olika områden. Först skapar jag variabeln nedan, observera att vårt dataset ligger i designobjektet och har namnet **variables**.

```
svy2010design$variables$minskaOff <- as.numeric(svy2010$FR28_1 %in% c("Ganska bra", "Mycket bra"))
```

Nästa steg är att skapa skattningar för respektive län, detta görs med **svyby()**.

```
minskaOffEst <- svyby(~minskaOff, by=~LKF, design=svy2010design, svymean)
```

Precis som surveyobjektet har en egen **data.frame** i surveyobjektet finns vårt dataset (kopplade till varje polygon - i detta fall kommun) som en egen **data.frame** i vårt spatiala objekt. Vi kan komma åt datat med **@** på följande sätt:

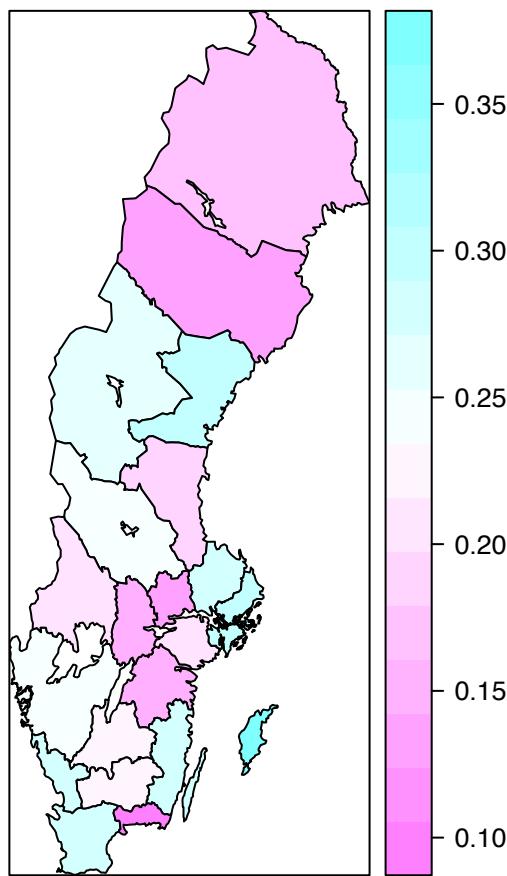
```
head(swe_county@data)
```

Denna **data.frame** kan vi hantera precis som vanliga **data.frames**. Vill vi lägga på data använder vi därför **merge()** och lägger till våra data om kor till respektive kommun.

```
swe_county@data <- merge(swe_county@data, minskaOffEst, by.x="LNKOD", by.y="LKF")
```

Nu har vi lagt till våra estimat och kan använda **spplot()** för att skapa en karta baserad på vår statistik. Pröva följande kod.

```
spplot(swe_county, "minskaOff")
```



Vi har nu gjort en karta över andelen som vill minska den offentliga sektorn i olika landsting.

Välj nu en annan fråga i enkäten och gör en så snygg karta som möjligt. Gör också en mindre analys kring de geografiska skillnaderna ni ser.

c) Som ni kanske upptäckt är ett problem med estimaten att de är olika särskra för olika län. Exempelvis i Stockholms län finns många observationer medan på Gotland är antalet observationer mycket mindre.

Det som då kan vara av intresse är att kombinera län till större områden. Det finns i Survey 2000 så kallade riksområden som vi kan använda.

Lägg till variabeln `romr` i det spatiala objektets dataset på liknande sätt som ovan. Efter att det är gjort kan vi använda funktionen `unionSpatialPolygons` för att lägga samman länsområdena (polygoner) till större områden.

```
swe_county@data$romr <- 0
swe_county@data$romr[swe_county@data$LNKOD %in% c("01")] <- 1
swe_county@data$romr[swe_county@data$LNKOD %in% c("05", "04", "18", "19", "03")] <- 2
swe_county@data$romr[swe_county@data$LNKOD %in% c("06", "07", "08", "09")] <- 3
swe_county@data$romr[swe_county@data$LNKOD %in% c("12", "10")] <- 4
swe_county@data$romr[swe_county@data$LNKOD %in% c("13", "14")] <- 5
swe_county@data$romr[swe_county@data$LNKOD %in% c("17", "20", "21")] <- 6
swe_county@data$romr[swe_county@data$LNKOD %in% c("23", "22")] <- 7
swe_county@data$romr[swe_county@data$LNKOD %in% c("24", "25")] <- 8
```

```
swe_romr <- unionSpatialPolygons(SpP = swe_county, IDs = swe_county@data$romr)
```

På detta sätt får vi ett nytt spatialt objekt med de nya geografiska områdena:

När vi lägger ihop olika polygoner i en karta kommer objektet att reduceras till ett `SpatialPolygons`-objekt istället för ett objekt av klassen `SpatialPolygonsDataFrame` (som krävs för att visualisera da-

ta). Vi behöver därför aggregera upp data på den nya indelningsnivån och skapa ett objekt av klassen `SpatialPolygonsDataFrame`.

```
aggr_data <- aggregate(x=swe_county$data$BEF05, by=list(swe_county$data$romr), FUN=sum)
names(aggr_data) <- c("romr", "BEF05")
swe_romr <- SpatialPolygonsDataFrame(Sr=swe_romr, data=aggr_data)
```

Med detta objekt kan vi sedan visualisera data med de nya områdena med hjälp av följande kod:

```
spplot(swe_romr, "BEF05")
```

Använd dessa nya större områden för att igen skapa en karta baserad på dessa områden med statistik för den variabel du använt. Använd `svyby()` för att skatta din variabel i olika riksområden.

Gör en så snygg karta du kan. Skiljer sig slutsatsen från din tidigare slutsats på länsnivå?

- d) Som ett sista steg ska vi plocka ut endast delar av en karta för att skapa kartor över mindre delområden. Börja med att läsa in shapefilen som innehåller kommungränser.

```
swe_municip <- readShapePoly("Kommun_SCB/Kommun_SCB_07.shp")
```

Som ett sista steg ska vi plocka ut endast delar av en karta för att skapa kartor över mindre delområden. Detta blir extra intressant när vi visualisera data på kartor för lokala surveyer (exempelvis i Östergötland).

Vi ska nu istället plocka ut en delmängd av dessa kommuner. Än en gång använder vi länskoderna vi skapat. För att plocka ut ett visst antal kommuner gör vi precis som för att indexera rader i en data.frame i R. För att välja ut länet "01" (Stockholms län) används följande kod.

```
spatial_sthlm <- swe_municip[substr(swe_municip$KNKOD, 1, 2)=="01", ]
```

Studera hur vårt det spatiala objekt ser ut med `plot()`.

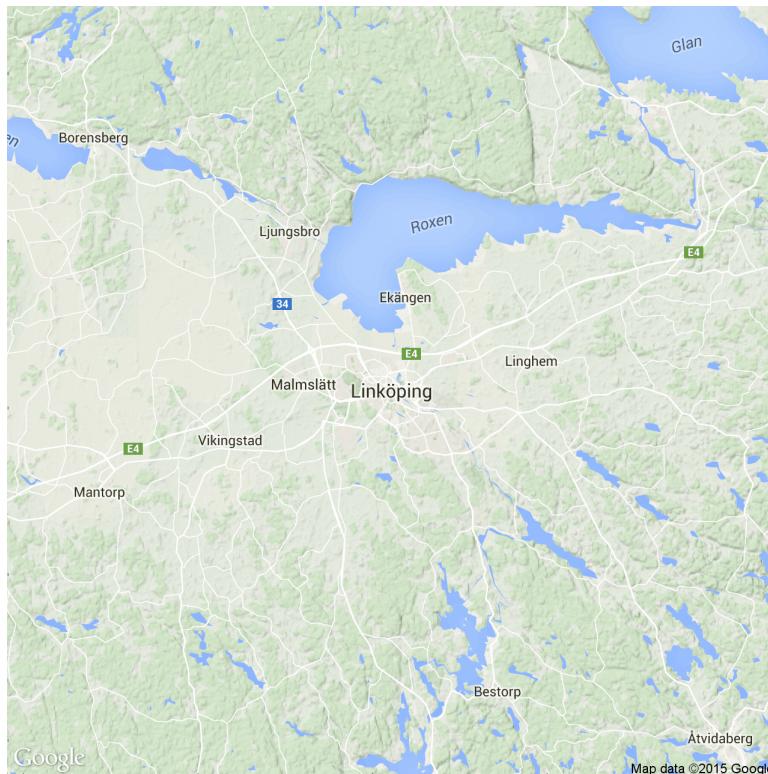
3.1 Skapa kartor med ggmap

Vi har hittills arbetat med shapefiler som innehåller spatiala data. Inte sällan vill vi också sätta våra spatiala data i relation till kartor. Särskilt om vi har data som är geotaggade, d.v.s. innehåller koordinater (ex. adress) kan det vara av intresse att visualisera dessa på en karta. För att använda kartor i R är det enklaste att använda oss av de kartor som finns öppet tillgängliga på webben, som Google Maps, Open Street Map m.fl. Med paketet `ggmap` kan vi enkelt läsa in en godtycklig kvadratisk karta.

Vi börjar med att läsa in paketet `ggmap` och skapar en första karta över Linköping på följande sätt.

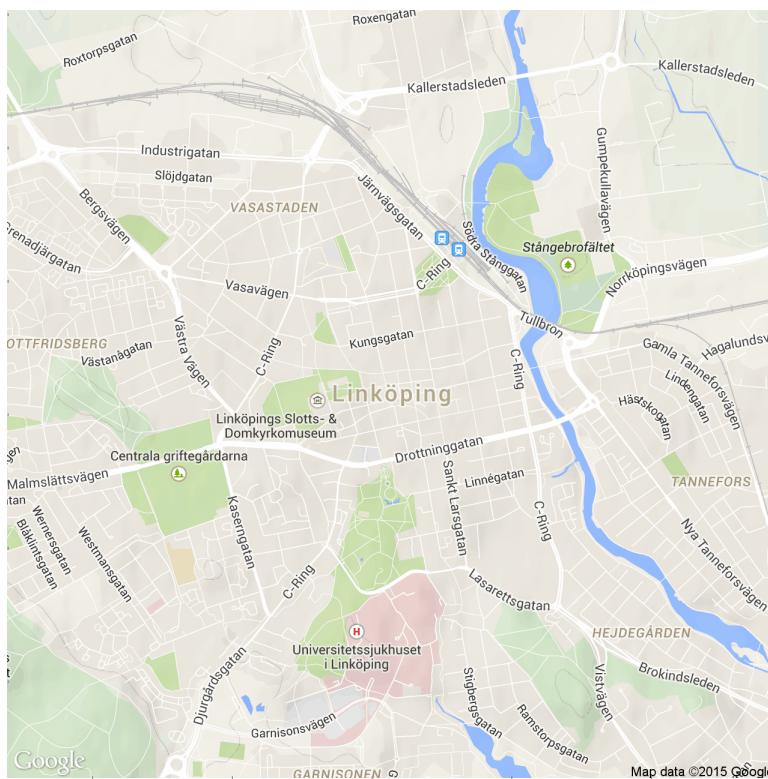
```
library(ggmap)
qmap("Linkoping")

## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=Linkoping&zoom=10&size=%20640
## Google Maps API Terms of Service : http://developers.google.com/maps/terms
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Linkoping&sensor=false
## Google Maps API Terms of Service : http://developers.google.com/maps/terms
```



Som framgår ovan så får vi dels upp en karta över Linköping (på en relativt hög upplösning) och dels information om att vi använt Google Maps samt information om vilka villkor som gäller för användadet av kartan. Framöver kommer denna extra information döljas. Vill vi zomma in eller ut använder vi argumentet **zoom**.

```
qmap("Linkoping", zoom=14)
```



Vi kan också använda Google Maps för att få ut koordinater för enskilda platser med **geocode()**. Av hänsyn till de personer som deltar i en undersökning vill vi sällan skriva ut exakta positioner för de

som svarat. Om vi använder postadresser kan vi dock visualisera surveyer på kartor utan att avslöja de riktiga adresserna för de som svarat.

```
linkpg_uni <- unlist(geocode("Linkopings Universitet"))
linkpg_uni
geocode("11732 Stockholm")

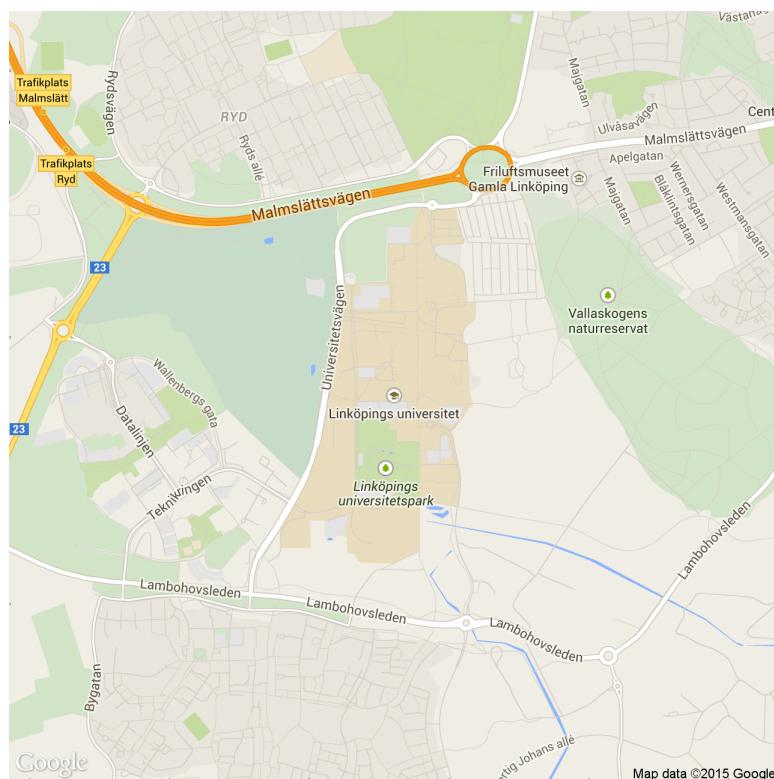
##   lon   lat
## 15.58 58.40

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=11732+Stockholm&sensor=false
## Google Maps API Terms of Service : http://developers.google.com/maps/terms

##   lon   lat
## 1 18.04 59.32
```

Då `qmap()` använder sig av koordinater kan vi också ange koordinater för att skapa kartor. Med argumentet `maptype` van vi välja mellan google maps kartor "roadmap", "satellite", "hybrid", "terrain".

```
qmap(location=linkpg_uni, zoom=14, maptype="roadmap")
```



Vi kan också välja svartvita kartor med argumentet `color`. Precis som i `ggplot2` kan vi spara våra kartobjekt för att skriva ut dem senare.

```
karta <- qmap(linkpg_uni, zoom = 14, maptype="satellite", color="bw")
```

Det finns fyra olika kartsystem som går att använda från `ggmap`. Vill vi använda något annat kartsystem använder vi `source`.

```
qmap(linkpg_uni, zoom = 14, source = "stamen", maptype = "toner")
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=58.397836,15.576007&zoom=14&size=640x480&t=h&t=m
## Google Maps API Terms of Service : http://developers.google.com/maps/terms
```



Nu har vi skapat kartor i R. Nästa steg är att kombinera kartor och spatiala data.

3.2 Visualisera geotaggad data på karta

Vi har tidigare sett hur vi kan få ut longitud och latitud med `geocode()`. Har vi väl denna geotaggade information är det enkelt att visualisera materialet på en karta. Att visualisera data på kartor följer i princip samma grundstruktur för att visualisera data med två kontinuerliga variabler (longitude och latitude). Detta framgår tydligt om vi använder följande kod.

```
ggmap(paris, extent = "normal")
```

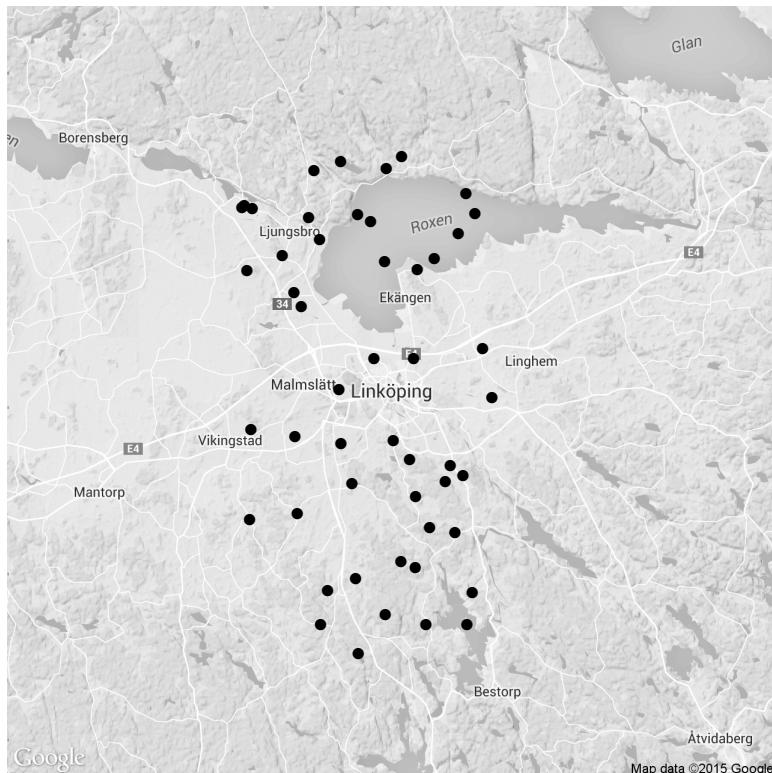
Med följande kod slumpas ett antal datapunkter ut kring Linköping.

```
set.seed(1984)
linkpg_map <- qmap("Linkoping", color="bw")
df <-
  data.frame(
    lon = jitter(rep(15.6, 50), amount = .15),
    lat = jitter(rep(58.4, 50), amount = .152),
    event = sample(c("Kometnedslag", "Olycka", "Polis"), size = 50, replace = TRUE)
  )
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=Linkoping&zoom=10&size=%20640x480&t=h&t=m
## Google Maps API Terms of Service : http://developers.google.com/maps/terms
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Linkoping&sensor=false
## Google Maps API Terms of Service : http://developers.google.com/maps/terms
```

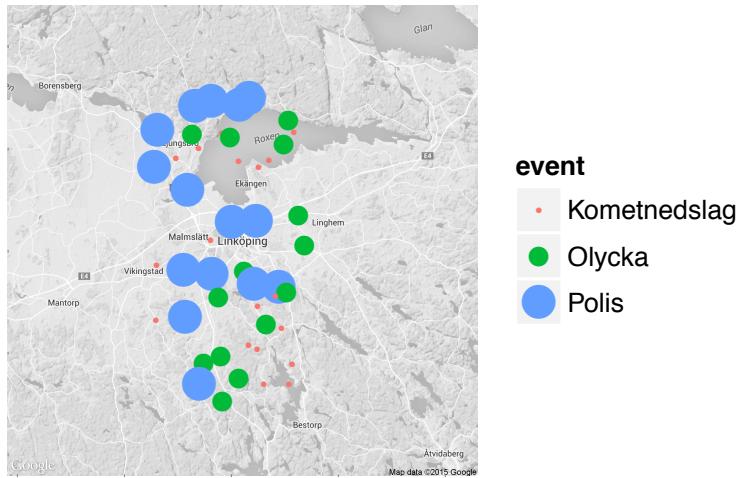
Vi kan använda samtliga funktioner i ggplot2. Nedan är motsvarigheten till en scatterplot.

```
linkpg_map +  
  geom_point(aes(x = lon, y = lat), data = df)
```



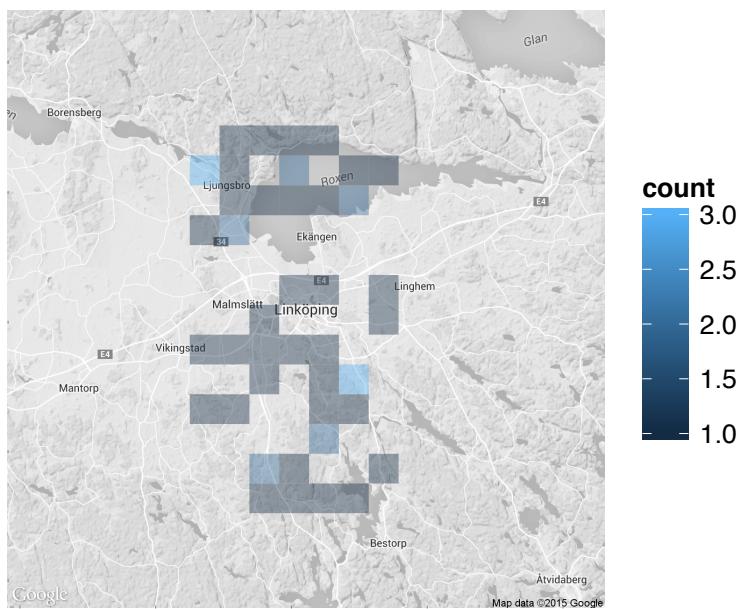
Vill vi visualisera olika kategorier kan vi använde `aesthetics` i ggplot2.

```
linkpg_map +  
  geom_point(aes(x = lon, y = lat, colour = event, size = event),  
             data = df)
```

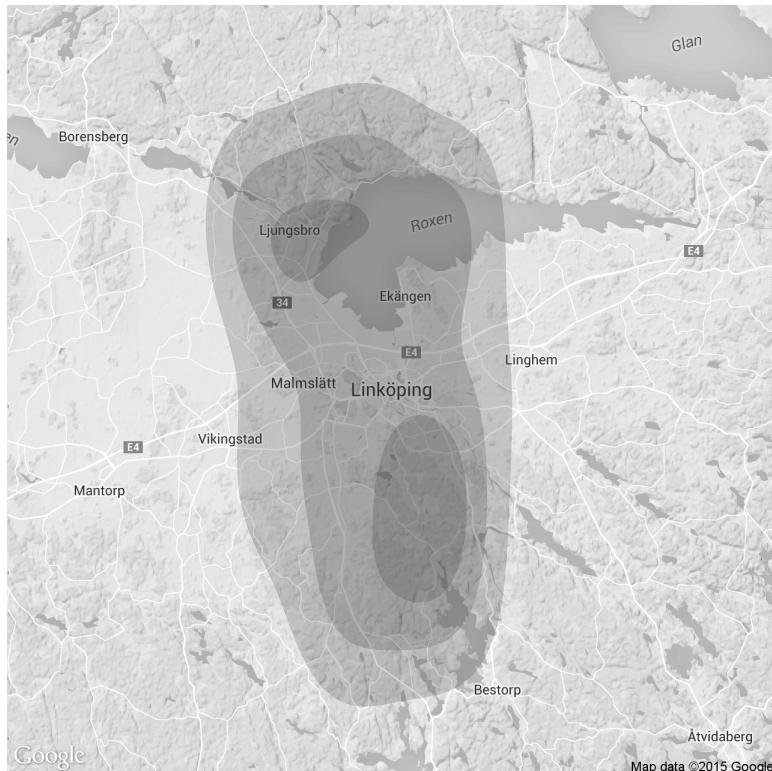


Vill vi visualisera frekvenser efter två kontinuerliga variabler finns dels `stat_bin2d` och `stat_density2d` tillgängligt i `ggplot2`.

```
linkpg_map +
  stat_bin2d(aes(x = lon, y = lat),
             size = .5, bins = 20, alpha = 0.4, data = df)
```



```
linkpg_map +
  stat_density2d(aes(x = lon, y = lat),
                 alpha = 0.2, size = 1, bins = 4,
                 data = df, geom = "polygon")
```



Då det i grunden är samma princip för att visualisera data på karta som att visualisera scatterplots kan vi använda oss av vikter i `ggplot2` som vi gjorde tidigare. Pröva att tilldela de påhittade punkterna i df olika vikter och visualisera de nya viktade punkterna på kartan i en plot.

3.3 Kombinera spatiala data med olika projektioner/geografiska referenssystem

När vi har många olika typer av data vi vill kombinera på samma karta måste de ha samma projektion och samma koordinatsystem. En bra och kort introduktion till kartprojektioner hittar du [\[här\]](#).

Alla data vi vill kombinera behöver vi känna till vilket geografiskt referenssystem som används. Exempelvis Google Maps använder sig av formatet EPSG 4326 (vilket vi kallar longitud och latitud). För att kolla upp vilket EPSG-nummer ett format kan vi använda <http://spatialreference.org/>.

Vi ska nu pröva att visualisera våra kommungränser vi läst in i shapefilen ovan på Google Maps.

- Det första steget vi måste ta är att vi måste konvertera våra kommunkartor till EPSG 4326 för att visualisera dem med Google Maps. Vi börjar med att tillskriva vår shapefil det EPSG-format filen har. (Filens användar SWEREF 99 TM - slå upp det på <http://spatialreference.org/> så framgår EPSG-koden).

```
proj4string(swe_municip) <- CRS("+init=epsg:3006")
```

- Vi har nu tillskrivit vårt spatiala objekt ett geografiskt referenssystem. Nästa steg är att konvertera den till longitud och latitud. För att konvertera mellan geografiska referenssystem behövs rgdal-paketet. Vi vet att EPSG-koden för detta är 4326 och använder `spTransform()` för att göra konverteringen på följande sätt.

```
swe_municip_4326 <- spTransform(swe_municip, CRS("+init=epsg:4326"))
```

3. Nu har vi konverterat vår shapefil till rätt format och kan visualisera våra data på en karta. För att `ggplot2` ska kunna läsa ett spatialt polygonobjekt behöver vi först göra om det spatala objektet till en `data.frame` som `ggplot` kan använda med

```
data <- fortify(swe_municip_4326)

## Regions defined for each Polygons

linkpg_large <- qmap("Linkoping", zoom = 6, maptype = "terrain")

## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=Linkoping&zoom=6&size=%2000x%2000&maptype=terrain&language=en&sensor=false
## Google Maps API Terms of Service : http://developers.google.com/maps/terms
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Linkoping&size=600x600
## Google Maps API Terms of Service : http://developers.google.com/maps/terms

linkpg_large + geom_polygon(aes(x = long, y = lat, group = group),
                           data = data, colour = "white", fill = "blue",
                           alpha = .2, size = .3)
```



4. På detta sätt kan vi visualisera de flesta spatiala datatyper på ett enkelt och snabbt sätt.

Referenser

- Hagevi, M., 2011. Den svenska väljaren, 1st Edition. Boréa, Umeå.
- Lumley, T., 2010. Complex surveys : a guide to analysis using R. Wiley-Blackwell, Oxford.

4 Bilaga: Repetition av grunderna i ggplot2

Paketet `ggplot2` skiljer sig från den grundläggande grafifikfunktionaliteten som finns implementerat i R. Paketet bygger på vad som brukar kallas “The grammar of graphics” (därav `gg` i `ggplot2`) och är ett försök till ett formellt språk för att uttrycka hur en visualisering ska se ut. Mer teori bakom denna grammatik går att finna i “The grammar of graphics” och är grunden bakom exempelvis SPSS grafiksystem. Genom att ha en grundläggande förståelse för denna grammatik kan vi enkelt och snabbt skapa mycket komplicerade visualiseringar.

I R:s basgrafiksyste kunde man se grafifikfunktionaliteten lite som ett papper vi ritar på. Vi ritar initialet upp vår graf och kan sedan lägga till/rita ”ovanpå” det befintliga pappret. `ggplot` är annorlunda. Med `ggplot` skapar vi ett grafikobjekt och vi kan lägga till bit för bit av grafen för att när vi sedan är klar med vår graf visualisera den. Det gör det enklare att bygga upp komplicerade grafer utan att behöva använda särskilt mycket kod.

4.1 Grunden i ggplot2

Till skillnad från basgrafiken utgår `ggplot` alltid från en `data.frame`. Baserat på denna `data.frame` skapas sedan grafen med två huvudsakliga komponenter:

- `aes` (aesthetic) som handlar om utseendet på grafen, färger, former m.m.
- `geom` (geometrics) som beskriver vilken typ av graf vi vill ha (bar, line, points)

Vi lägger sedan till dessa komponenter till vår graf och `data.frame`.

När det gäller de olika geometriska argumenten, d.v.s. de olika typer av grafer som går att skapa, finns det ett mycket stor antal vi kan använda oss av. Några exempel är:

geom	Beskrivning
<code>geom_point</code>	Scatterplot
<code>geom_line</code>	Line graph
<code>geom_bar</code>	Barplot
<code>geom_boxplot</code>	Boxplot
<code>geom_histogram</code>	Histogram

Exakt hur dessa geometriska figurer ska se ut styrs sedan med `aes`. Nedan finns några exempel:

aes	Beskrivning
<code>x</code>	x-axel
<code>y</code>	y-axel
<code>size</code>	storlek
<code>col</code>	färg
<code>shape</code>	form

De enskilda geometriska figurerna kan i sin tur ha ett antal olika aesthetics. Nedan finns lite exempel.

geom	Specifika aesthetics
<code>geom_points</code>	point <code>shape</code> , point <code>size</code>
<code>geom_line</code>	<code>linetype</code> , <code>line size</code>
<code>geom_bar</code>	<code>y min</code> , <code>y max</code> , <code>fill color</code> , <code>outline color</code>

Med dessa verktyg har vi en grund för att bygga upp ett mycket stort antal visualiseringar.

4.2 Skapa en ggplot (linje eller scatter)

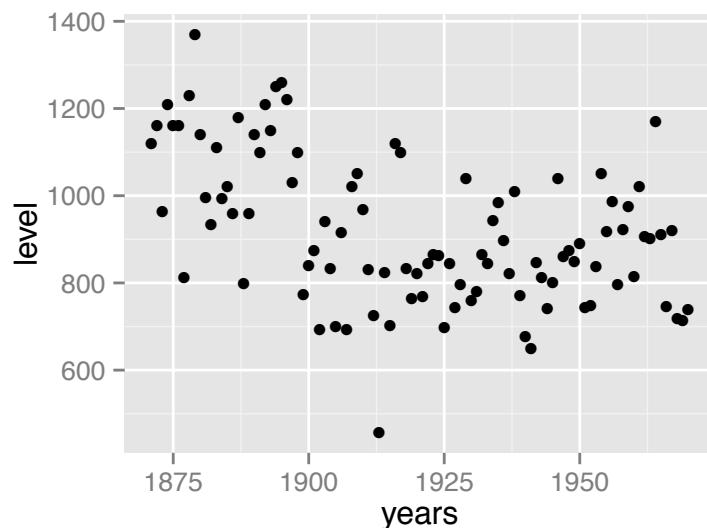
1. Vi börjar med att läsa in the datamaterialet `Nile`.

```
library(ggplot2)
data(Nile)
Nile <- as.data.frame(Nile)
colnames(Nile) <- "level"
Nile$years <- 1871:1970
```

2. För att skapa en `ggplot` börjar vi med att skapa grunden för plotten med funktionen `ggplot()`. Nedan är ett exempel på att skapa en `ggplot` med `Nile`, sedan lägger vi till att `x` ska utgöras av variabeln `years` och `level`. Sedan lägger vi till att plotten ska utgöras av punkter. Vi sparar grafen som variabeln `p`. För att skapa grafen tittar vi bara på `p`:

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_point()
p

## Don't know how to automatically pick scale for object of type ts. Defaulting to
continuous
```



3. Vill vi ändra till en linjegraf (vilket känns bättre) här byter vi bara ut geometrin:

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_line()
```

4. Vill vi lägga till både punkter och linjer i samma graf kan vi bara ta `p` och lägga till punkter. Här blir det tydligt hur vi i `ggplot` lägger till lager på lager och sedan producerar en visualisering:

```
p <- p + geom_point()
```

5. På samma sätt kan vi också lägga till rubriker och axelettiketter:

```
p <- p + xlab("Years") + ylab("Water level") + ggtitle("Nile series")
Nile$period <- "- 1900" Nile$period[Nile$years >= 1900] <- "1900 - 1945" Nile$period[Nile$years
```

4.3 Enklare modifikationer av ett `ggplot`-objekt

1. Vill vi ändra färg och form på olika delar i en graf behöver vi ange exakt var dessa förändringar ska ske.

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_line(color="red", size=3) + geom_point(col
```

2. Om vi nu vill förtydliga vissa delar av grafen med olika färger eller använder vi `aes` i den del av grafen vi vill ändra. Först ska vi skapa en ny faktorvariabel vi vill visualisera.

```
Nile$period <- "- 1900"
Nile$period[Nile$years >= 1900] <- "1900 - 1945"
Nile$period[Nile$years > 1945] <- "1945 + "
Nile$period <- as.factor(Nile$period)
```

3. Vill vi nu exempelvis lyfta in visualiseringen i linjerna måste vi lägga `aes` där.

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_line(aes(color=period)) + geom_point()
```

4. Vill vi istället modifiera punkterna lägger vi till det i `geom_point()`.

```
p <- ggplot(data=Nile) + aes(x=years, y=level) + geom_line() + geom_point(aes(color=period))
```

5. Vill vi lägga det i hela grafen kan vi lägga till färgen i den huvudsakliga styrningen av aesthetics i grafen.

```
p <- ggplot(data=Nile) + aes(x=years, y=level, color=period) + geom_line() + geom_point()
```

6. Baserat på graferna ovan prova att göra följande förändringar:

- (a) Ändra typ av linje i grafen [**Tips!** `linetype`]
- (b) Ändra typ av punkter i grafen [**Tips!** `shape`]
- (c) Gör punkterna transparaenta [**Tips!** `alpha`]

4.4 Barplot, histogram och boxplot

För att pröva dessa diagram använder vi oss av datamaterialet mtcars. Vi börjar med att läsa in datamaterialet mtcars. För att få mer information om detta datamaterial, använd `?mtcars`. Vi gör också om

```
data(mtcars)
mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$gear <- as.factor(mtcars$gear)
```

Till skillnad från basgrafen använder vi inte olika funktioner för olika plottar utan vi använder bara olika `geoms`.

1. Vill vi exempelvis skapa ett stapeldiagram anger vi bara en axel och ett annat geom, men i övrigt är det inge större skillnad mot en linjegraf:

```
p <- ggplot(data=mtcars) + aes(x=cyl) + geom_bar()
```

2. Vi kan också enkelt lägga till funktionen `coord_flip()` för att skapa ett liggande stapeldiagram istället för ett stående.

```
p + coord_flip()
```

3. Skillnaden ligger i att det finns lite andra aesthetics för stapeldiagram än för övriga diagram som `fill`.

```
p <- ggplot(data=mtcars) + aes(x=cyl) + geom_bar(fill="darkblue", colour="red")
```

4. För att skapa stapeldiagram med flera grupper behöver vi dels lägga till en till variabel som indikerar att vi vill ha ex. olika färger för olika grupper samt ange hur dessa diagram ska se ut. Pröva exemplen nedan:

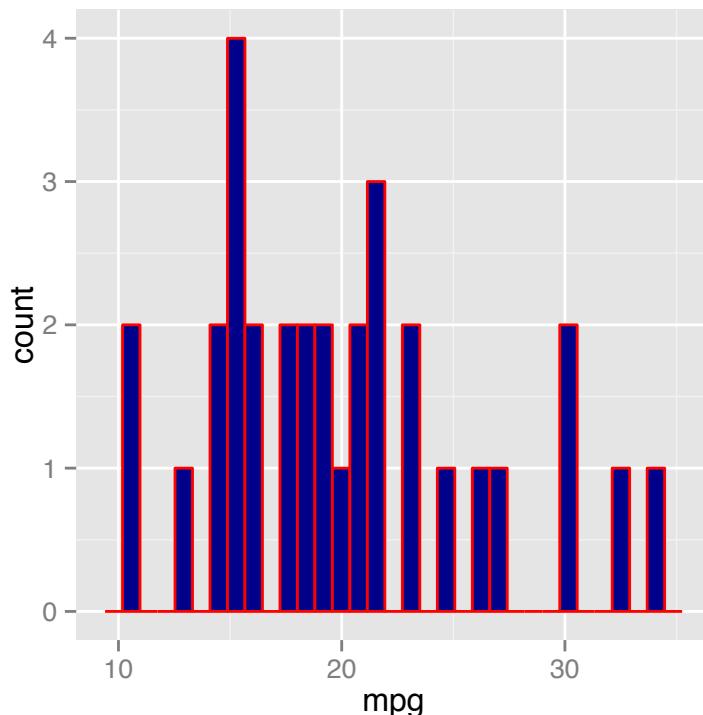
```
p <- ggplot(data=mtcars) + aes(x=cyl, fill=gear) + geom_bar(position="stack")
p <- ggplot(data=mtcars) + aes(x=cyl, fill=gear) + geom_bar(position="dodge")
p + scale_fill_discrete(name="Testa\nDetta")
```

4.5 Histogram

1. Den egentliga skillnaden mellan ett stapeldiagram och ett histogram är bara huruvida variabeln är kontinuerlig eller inte. Detta gör att för att skapa ett histogram gör vi på exakt samma sätt, men vi använder oss av en kontinuerlig variabel:

```
p <- ggplot(data=mtcars) + aes(x=mpg) + geom_bar(fill="darkblue", colour="red")
p

## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



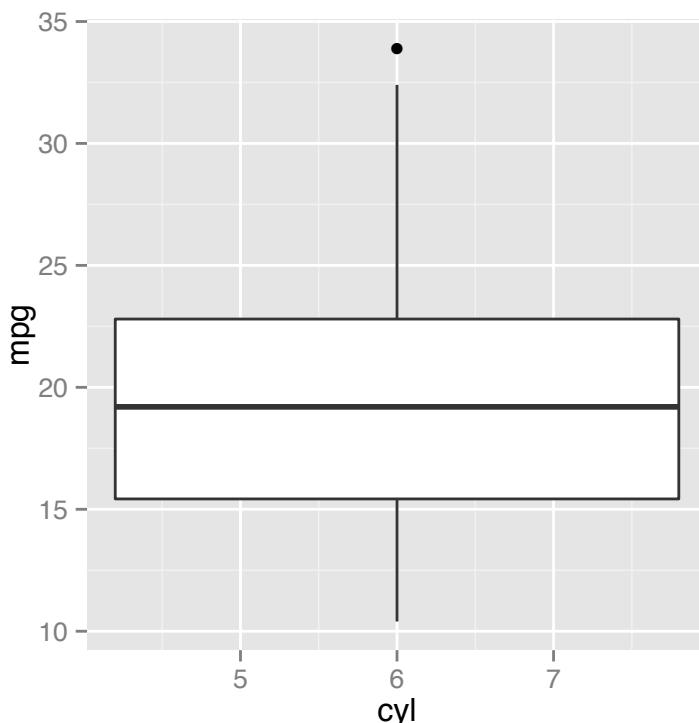
2. Sättet ovan är ett sätt att skapa ett histogram. Vi kan också använda den specialgjorda geometriska funktionen `geom_histogram()` om vi vill kunna hantera histogram enklare.

```
p <- ggplot(data=mtcars) + aes(x=mpg) + geom_histogram(fill="darkblue", colour="red", binwidth=10)
```

4.6 Boxplot

- Boxplottar är egentligen en kombination av kontinuerliga variabler. Precis som tidigare inleder vi skapa en `ggplot` med ett datamaterial och definierar vilka variabler vi vill använda.

```
p <- ggplot(data=mtcars) + aes(x=cyl, y=mpg) + geom_boxplot()
p
```



- Vill vi sedan göra förändringar kan vi lägga till det till och från.

```
p + coord_flip() + xlab("X") + ggtitle("Hejsan")
```

4.7 Grafiska teman/profiler

En av de stora fördelarna med att `ggplot` skiljer ut själva plotten från utseendet är att det är enkelt att skapa strukturer för olika delar av en graf som vi vill använda flera gånger. En av de bästa exemplen på detta är teman i `ggplot`. Ett tema är en uppsättning med inställningar för en grafisk profil som vi vill använda i `ggplot2`.

Den stora fördelen är att har vi väl skapat ett tema (vilket kan ta lite tid) kan temat läggas till mycket enkelt till samtliga grafer. Detta underlättar kopplingen mellan exempelvis grafiska profiler och de grafer som produceras, vilket gör att `ggplot2` är mycket populärt i företag och organisationer. Ett exempel på rapport som använder `ggplot2` genomgående är Pensionsmyndighetens [Orange rapport].

- Med `ggplot2` kommer en del teman förinstallerade och precis som allt annat i R är det enkelt att bara lägga till den grafiska profilen efter att vi skapat en graf.

```
p <- ggplot(data=Nile) + aes(x=years, y=level, color=period) + geom_line() + geom_point()  
p <- p + theme_bw()
```

2. Pröva på liknande sätt följande teman: `theme_grey()`, `theme_classic()`.
3. Ett tema i `ggplot2` är bara en funktion, så det är enkelt att titta på hur temat ser ut och sedan utgå från ett befintligt tema för att anpassa det till det utseende vi själva vill ha. Sedan kan detta tema enkelt spridas till alla som arbetar med visualisering med `ggplot`.

```
theme_bw
```

4. Att ändra den grafiska profilen innebär då bara att ändra denna temafunktion (även om det kan innebära en del jobb).