

Datorlaboration 8

Josef Wilzén och Måns Magnusson

8 mars 2016

Instruktioner

- Denna laboration ska göras i grupper om **två och två**. Det är viktigt för gruppindelningen att inte ändra grupper.
 - En av ska vara **navigatör** och den andra **programmerar**. Navigatörens ansvar är att ha ett helhetsperspektiv över koden. Byt position var 30:e minut. Båda ska vara engagerade i koden.
 - Det är tillåtet att diskutera med andra grupper, men att plagiera eller skriva kod åt varandra är **inte tillåtet**.
 - Använd inte å, ä eller ö i variabel- eller funktionsnamn.
 - Utgå från laborationsfilen, som går att ladda ned **här**, när du gör inlämningsuppgifterna.
 - Spara denna som labb[no].grupp[no].R , t.ex. labb5_grupp01.R om det är labb 5 och ni tillhör grupp 01. Ta inte med hakparenteser eller stora bokstäver i filnamnet. Denna fil ska laddas upp på LISAM och ska **inte** innehålla något annat än de aktuella funktionerna, namn-, ID- och grupp-variabler och ev. kommentarer. Alltså **inga** andra variabler, funktionsanrop för att testa inlämningsuppgifterna eller anrop till markmyassignment-funktioner. Om ni ska lämna i kompletteringar på del 2, döp då dessa till labb5_grupp01_komp1.R om det är första kompletteringstillfället. Se kurshemsidan för mer information om kompletteringar.
 - Laborationen består av två delar:
 - Datorlaborationen
 - Inlämningsuppgifter
 - I laborationen finns det extrauppgifter markerade med *. Dessa kan hoppas över.
 - Deadline för labben framgår på **LISAM**
-

Innehåll

I Datorlaboration	3
1 Introduktion till ggplot2	4
1.1 Grunden i ggplot2	4
1.1.1 Skapa en ggplot (linje eller scatter)	5
1.1.2 Enklare modifikationer av ett ggplot-objekt	6
1.1.3 Barplot, histogram och boxplot	6
1.1.4 Histogram	7
1.1.5 Boxplot	8
1.2 Grafiska teman/profiler	9
2 Introduktion till spatiala data och visualisering	12
2.1 Förberedelser	12
2.1.1 Rpaket	12
2.1.2 Data	12
2.2 Läsa in shape-filer och spatiala datastrukturer	13
2.2.1 Statistik på karta	14
2.2.2 Kombinera polygoner till större geografiska områden.	15
2.2.3 Skapa delmängder av spatiala objekt	16
2.2.4 * Extraproblem: Statistik över Östergötland	17
2.3 Skapa kartor med ggmap	17
2.4 Visualisera geotaggad data på karta	20
2.5 Kombinera spatiala data med olika projektioner/geografiska referenssystem	23
II Inlämningsuppgifter	26
3 Inlämningsuppgifter	28
3.1 Svenska personnummer	28
3.1.1 Uppgift 1: pnr_format()	29
3.1.2 Uppgift 2: pnr_ctrl()	29
3.1.3 Uppgift 3: pnr_sex()	30
3.1.4 Uppgift 4: pnr_samordn()	30
3.1.5 Uppgift 5: pnr_date()	31
3.1.6 Uppgift 6: pnr_age()	32
3.1.7 Uppgift 7: pnr_info()	32
3.2 Miniprojektet del II	33

Del I

Datorlaboration

Kapitel 1

Introduktion till ggplot2

Paketet `ggplot2` skiljer sig från den grundläggande grafikfunktionaliteten 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 [2, 1] 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 basgrafiksysteem kunde man se grafikfunktionaliteten lite som ett papper vi ritar på. Vi ritar initialt 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.

[Här](#) finns en bra katalog över de flesta graferna i `ggplot2`.

1.1 Grunden i ggplot2

Till skillnad från basgrafen 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 exemplen 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
geom_points	point shape, point size
geom_line	linetype, line size
geom_bar	y min, y max, fill color, outline color

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

1.1.1 Skapa en ggplot (linje eller scatter)

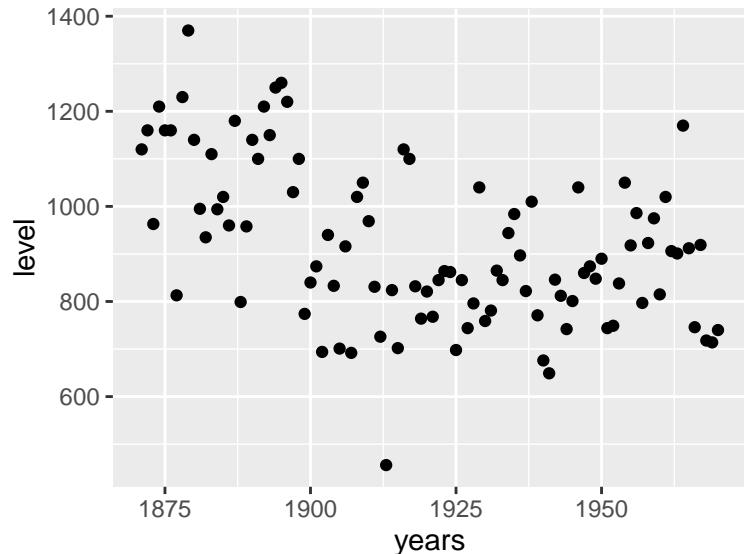
- Vi börjar med att läsa in the datamaterialet Nile.

```
library(ggplot2)

data(Nile)
Nile <- data.frame(level=as.vector(Nile))
Nile$years <- 1871:1970
```

- 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
```



- 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()
```

- 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")
```

1.1.2 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(color="blue", size=4)
```

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 pröva att göra följande förändringar:

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

1.1.3 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`.

- 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()
```

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

```
p + coord_flip()
```

- 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")
```

- 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")
p + scale_fill_manual(values=c("black", "blue", "red"))

```

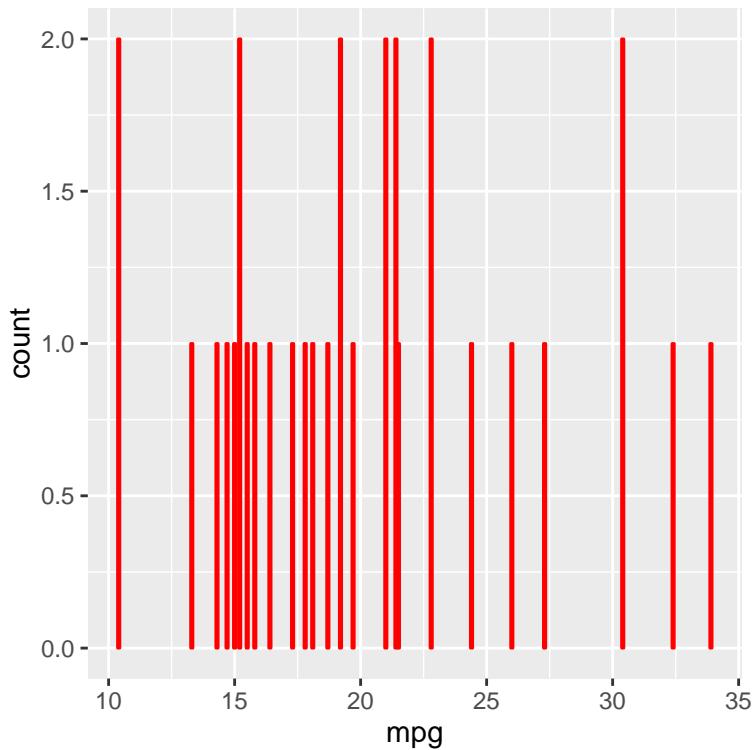
1.1.4 Histogram

- 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

```



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.

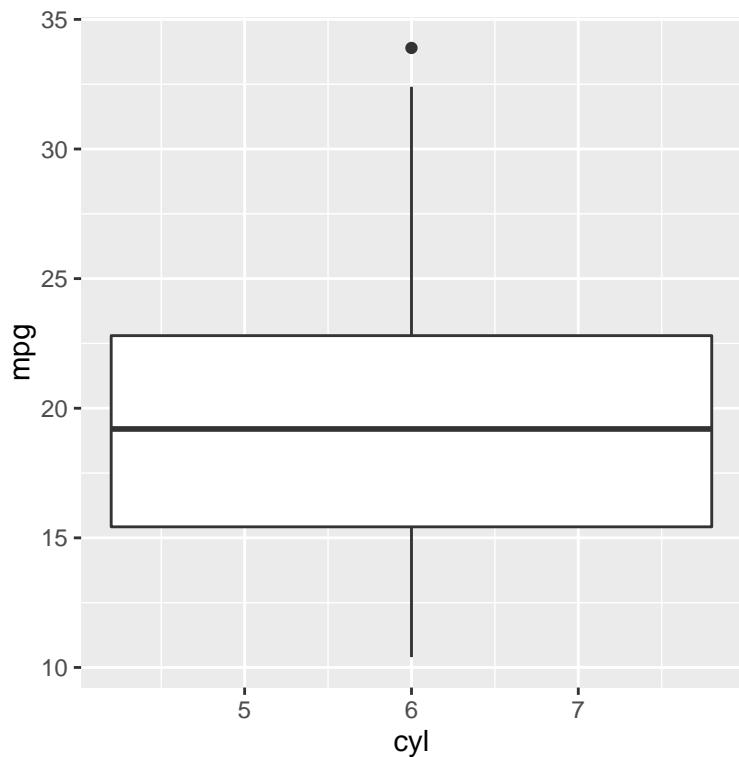
```
p <- ggplot(data=mtcars) + aes(x=mpg)
p <- p + geom_histogram(fill="darkblue", colour="red", binwidth=10)
p
```

1.1.5 Boxplot

1. 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

Warning: Continuous x aesthetic -- did you forget aes(group=...)?
```



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

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

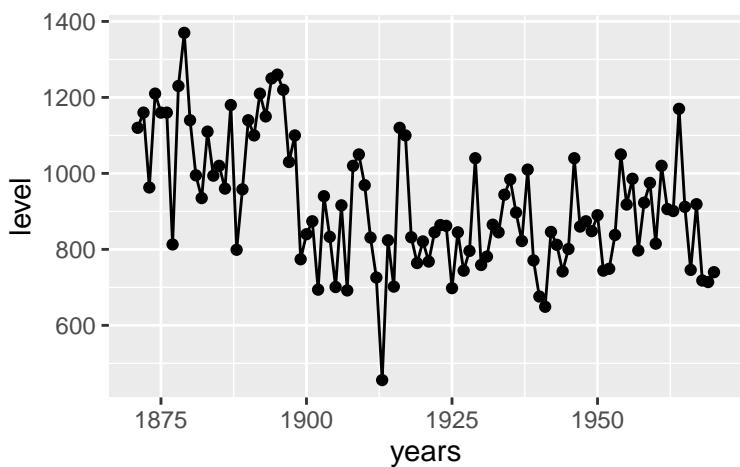
1.2 Grafiska teman/profiler

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`.

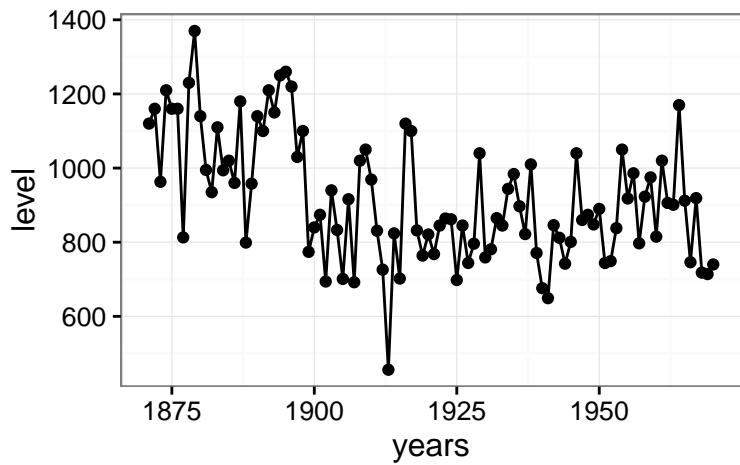
Den stora fördelen är att har vi väl skapat ett tema (vilket kan ta lite tid) kan temat läggas till mycket enkelt till samtliga grafer. Detta underlättar kopplingen mellan exempelvis grafiska profiler och de grafer som produceras, vilket gör att `ggplot2` är mycket populärt i företag och organisationer.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

1. 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) + geom_line() + geom_point()
p
```



```
p <- p + theme_bw()
p
```

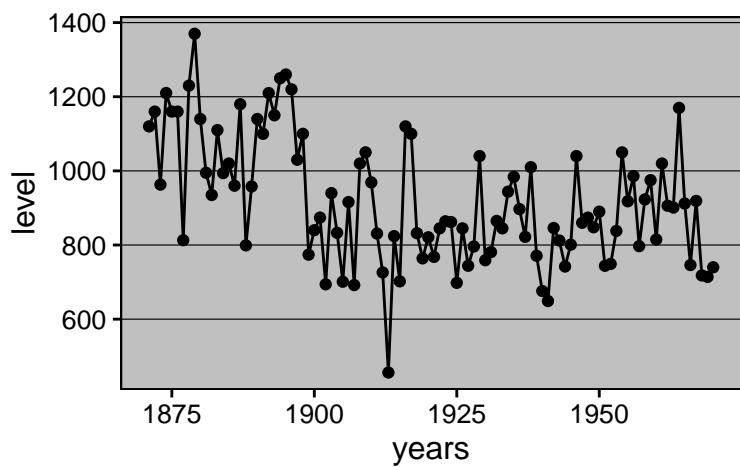


2. Pröva på liknande sätt följande teman: `theme_grey()`, `theme_classic()`.
3. Ett tema i `ggplot2` är bara en funktion, så det är enkelt att titta på hur temat ser ut och sedan utgå från ett befintligt tema för att anpassa det till det utseende vi själva vill ha. Sedan kan detta tema enkelt spridas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

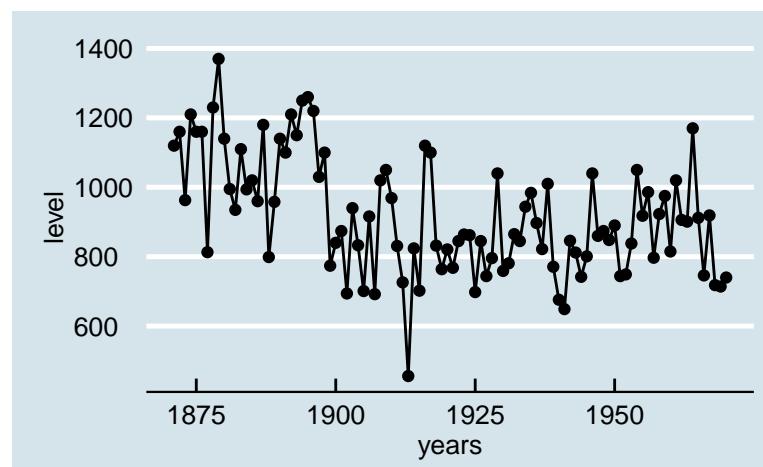
```
theme_bw
```

4. Att ändra den grafiska profilen innebär då bara att ändra denna temafunktion (även om det kan innebära en del jobb).
5. Det finns också ett separat R-paket med ett antal vanliga teman kallat `ggthemes`. De olika teman som är installerade i `ggthemes` framgår [**här**].
6. Med dessa går det enkelt att skapa olika färgsättningar för samma graf. I `ggthemes`-paketet finns också färgpaletter som passar bra för personer som är färgblinda.

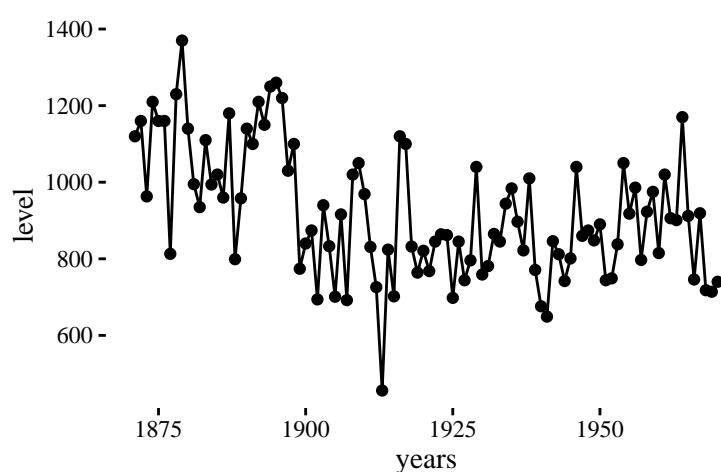
```
library(ggthemes)
p + theme_excel()
```



```
p + theme_economist()
```



```
p + theme_tufte()
```



Kapitel 2

Introduktion till spatiala data och visualisering

Spatiala data , geografiska data eller geotaggad data är data som innehåller information om lokalitet eller plats. Det är en datastruktur som blir allt mer vanligt i och med att allt mer data digitaliseras. Det kan vara data om positioner (från exempelvis mobiltelefoner eller twittermeddelanden), det kan vara kartor för att skapa geografiska referenser till exempelvis data på kommunal nivå och det kan vara eller det kan vara spatial visualisering där exempelvis områden med större brottslighet visualiseras.

Samlingsnamnet för denna typ av analyser är spatiala data eller spatial analys. En mycket bra hemsida med fördjupningar finns [\[här\]](#).

Tidigare har denna typ av analys och databearbetning varit reserverade för personer med särskilt kunnande i geografiska informationsystem eller GIS, men på senare tid har R seglat upp som en allt större konkurrens när det gäller spatial visualisering och analys.

I detta avsnitt kommer vi göra en kortare djupdykning i detta fält. Djupdykningen kommer vara långt ifrån fullständig, men ge en känsla för att det ofta inte behöver vara särskilt svårt att skapa kartvisualiseringar.

2.1 Förberedelser

2.1.1 Rpaket

Som ett första steg behöver vi installera en hel del paket som kan hantera och arbeta med spatiala datastrukturer. Vi kommer därför gå igenom den mest 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.

2.1.2 Data

I denna laboration behöver vi också ladda ned den data vi ska använda oss av. Detta är exempel på svenska data över sveriges kommunindelning i form av shape-fil. Vi laddar ned dessa shapefiler med paketet `downloader`.

```
library(downloader)
# Download and read data
komuner_remote <- "https://raw.githubusercontent.com/MansMeg/KursSvyMeth/master/Labs/DataFiles/Kommun_SCB.zip"
```

```

komuner_local <- "Kommun_SCB.zip"
download(komuner_remote, destfile = komuner_local)
unzip(komuner_local)

```

2.2 Läsa in shape-filer och spatiala datastrukturer

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.

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

```

library(rgdal)

Loading required package: methods
Loading required package: sp
rgdal: version: 1.1-3, (SVN revision 594)
Geospatial Data Abstraction Library extensions to R successfully loaded
Loaded GDAL runtime: GDAL 2.0.1, released 2015/09/15
Path to GDAL shared files: C:/Users/Florence/Documents/R/win-library/3.2/rgdal/gdal
GDAL does not use iconv for recoding strings.
Loaded PROJ.4 runtime: Rel. 4.9.1, 04 March 2015, [PJ_VERSION: 491]
Path to PROJ.4 shared files: C:/Users/Florence/Documents/R/win-library/3.2/rgdal/proj
Linking to sp version: 1.2-2

library(rgeos)

rgeos version: 0.3-17, (SVN revision 520)
GEOS runtime version: 3.5.0-CAPI-1.9.0 r4084
Linking to sp version: 1.2-2
Polygon checking: TRUE

library(maptools)

Checking rgeos availability: TRUE

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

```

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

```

class(swe_municip)
summary(swe_municip)

```

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

```
plot(swe_municip)
```



4. Nu har vi läst in och studerat vår första shapefil!

2.2.1 Statistik på karta

Nästa steg är att integrera statistik med vår karta, för att på detta sätt visualisera kommunala data. Med Koden nedan laddar jag ned kommunala data från SCB (antalet mjölkkor per kommun) och städar datat något.

```
library(pxweb)

pxweb: R tools for PX-WEB API.
Copyright (C) 2014 Mans Magnusson, Leo Lahti, Love Hansson
https://github.com/ropengov/pxweb
```

```
my_data <-
  get_pxweb_data(url = "http://api.scb.se/0V0104/v1/doris/sv/ssd/J0/J00103/HusdjurK",
                 dims = list(Region = c('*'),
                             Djurslag = c('05'),
                             ContentsCode = c('J00103K2'),
                             Tid = c('2007')),
                 clean = TRUE)
my_data <- my_data[, c(1,5)]
my_data[, 1] <- substr(as.character(my_data$region), 1, 4)
my_data[is.na(my_data[, 2]), 2] <- 0
```

1. När det gäller spatiala data 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:

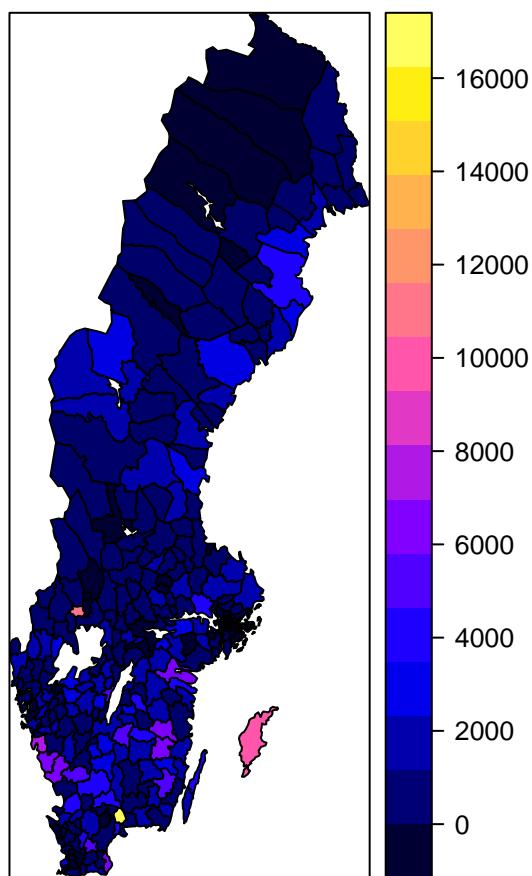
```
head(swe_municip@data)
```

2. 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_municip@data <- merge(swe_municip@data, my_data, by.x="KNKOD", by.y="region")
```

3. 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_municip, "values")
```



2.2.2 Kombinera polygoner till större geografiska områden.

Nu har vi arbetat med svenska kommuner. Vill vi arbeta med län går detta att göra genom att läsa in en separat shapefil för län. Vi kan dock kombinera polygoner till större områden direkt i R. I detta fall är det län, men det går att kombinera områden på ett godtyckligt sätt.

1. Vi börjar att skapa en variabel som heter `LN` som indikerar vilket län respektive kommun ligger i. Tack vara kommunkoderna är det möjligt att se detta direkt. De två första positionerna anger län.
2. Nästa steg blir att slå samman de kommunala polygonerna för att bilda länspolygoner. För detta använder vi `unionSpatialPolygons()`. I detta fall verkar det som några kommuner hamnat i fel

län. Detta beror på att länskoden för vissa kommuner blivit felaktig.

```
swe_county <- unionSpatialPolygons(SpP = swe_municip, IDs = swe_municip@data$LN)
```

3. Vi kan sedan studera resultatet med `plot()` och `summary()`.

```
plot(swe_county)
summary(swe_county)
```

4. När vi lägger ihop olika polygoner försinner vårt datamaterial i vårt spatiala objekt. För att lägga till nya data behöver vi därför göra detta ”manuellt”. Vi börjar med att aggregera upp våra data till länsnivå.

```
aggr_data <- aggregate(x=swe_municip@data$values, by=list(swe_municip@data$LN), FUN=sum)
names(aggr_data) <- c("LN", "kossor")
```

5. Nästa steg är att lägga till dessa nya aggregerade data till vårt nya spatiala objekt. För att detta ska gå måste rad-id i vår data.frame överens med polygon-id i det spatiala objektet. Därför börjar vi med att lägga till radnamn och därefter använder vi `SpatialPolygonsDataFrame()` för att kombinera data på länsnivå.

```
rownames(aggr_data) <- aggr_data$LN
swe_county <- SpatialPolygonsDataFrame(Sr=swe_county, data=aggr_data)
```

6. Nu kan vi återigen skapa en karta med lite större områden där antalet kor framgår mer tydligt. (Men på grund av kommunkodsproblem så har några kommuner hamnat fel.)

```
spplot(swe_county, "kossor")
```

2.2.3 Skapa delmängder av spatiala objekt

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. Vi utgår än en gång från våra kommuner som spatialt objekt.

1. 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[swe_municip@data[['LN']]=="01",]
```

2. Vi kan nu studera dels hur vårt spatiala objekt ser ut med `plot()` samt få en bild av var kossorna bor i Stockholms län med `spplot()`.

2.2.4 * Extraproblem: Statistik över Östergötland

Vi ska nu skapa en karta över Östergötland för en godtycklig statistisk storhet. Ta reda på vilken länskod Östergötland har. Gå in på SCB och hämta ned data på kommunnivå för Östergötlands län. Du kan välja vilken statistik du vill (ex. valresultat finns på kommunal nivå) och skapa en karta över denna variabel för Östergötland.

För att snabbt ladda ned data till R kan du använda paketet `pxweb`.

```
library(pxweb)
my_data <- interactive_pxweb(api="scb", lang="sv")
```

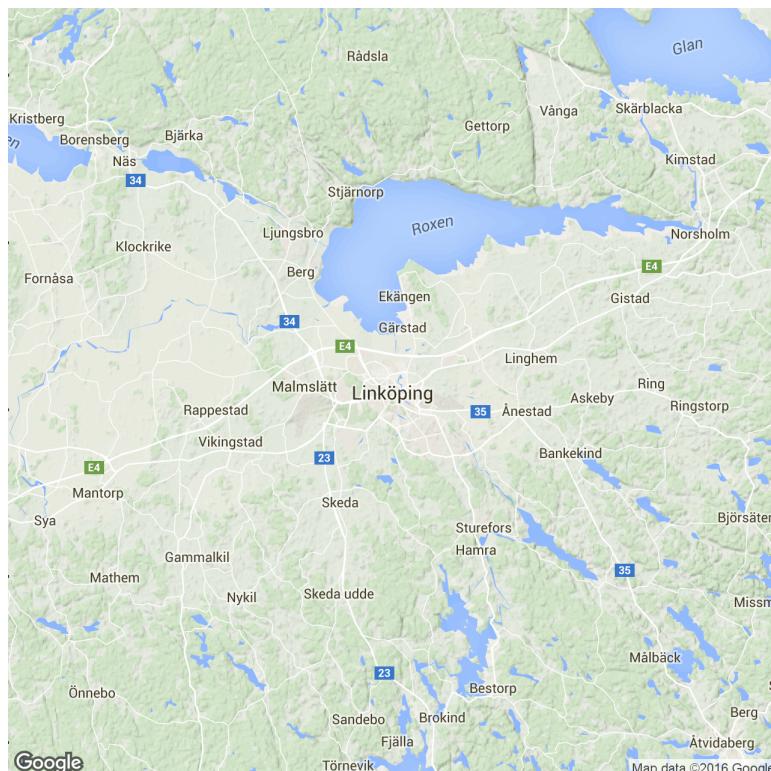
2.3 Skapa kartor med `ggmap`

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 är det 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.

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

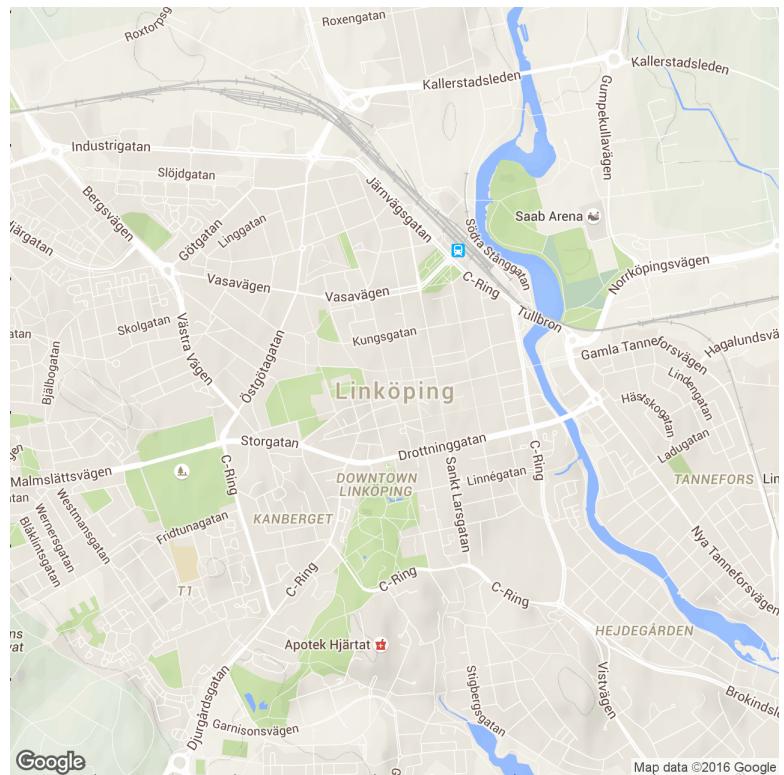
```
library(ggmap)
qmap("Linkoping")
```

Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=Linkoping&zoom=10&size=640x640
Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Linkoping&sensor=false



2. 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 användandet av kartan. Framöver kommer denna extra information döljas. Vill vi zomma in eller ut använder vi argumentet `zoom`.

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



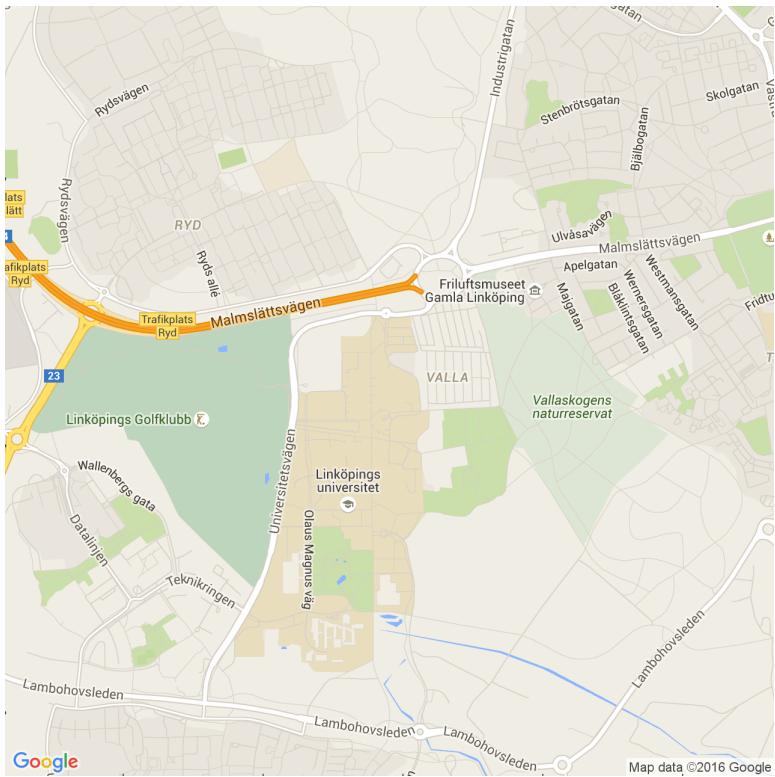
3. Vi kan också använda Google Maps för att få ut koordinater för enskilda platser med `geocode()`.

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

lon	lat
15.579	58.402

4. 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")
```



5. 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")
```

6. 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=58.40208,15.579011&zoom=14&size=640x640&maptype=toner&language=sv-SE&sensor=false
Map from URL : http://tile.stamen.com/toner/14/8899/4898.png
Map from URL : http://tile.stamen.com/toner/14/8900/4898.png
Map from URL : http://tile.stamen.com/toner/14/8901/4898.png
Map from URL : http://tile.stamen.com/toner/14/8902/4898.png
Map from URL : http://tile.stamen.com/toner/14/8899/4899.png
Map from URL : http://tile.stamen.com/toner/14/8900/4899.png
Map from URL : http://tile.stamen.com/toner/14/8901/4899.png
Map from URL : http://tile.stamen.com/toner/14/8902/4899.png
Map from URL : http://tile.stamen.com/toner/14/8899/4900.png
Map from URL : http://tile.stamen.com/toner/14/8900/4900.png
Map from URL : http://tile.stamen.com/toner/14/8901/4900.png
Map from URL : http://tile.stamen.com/toner/14/8902/4900.png
Map from URL : http://tile.stamen.com/toner/14/8899/4901.png
Map from URL : http://tile.stamen.com/toner/14/8900/4901.png
Map from URL : http://tile.stamen.com/toner/14/8901/4901.png
Map from URL : http://tile.stamen.com/toner/14/8902/4901.png
```



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

2.4 Visualisera geotaggad data på karta

Att visualisera data på kartor följer i princip samma grundstruktur för att visualisera data med två kontinuerliga variabler (longitud och latitud). Detta framgår tydligt om vi använder följande kod.

```
ggmap(linkpg_uni, extent = "normal", zoom=12)
```

- Med följande kod slumpas ett antal datapunkter ut kring Linköping.

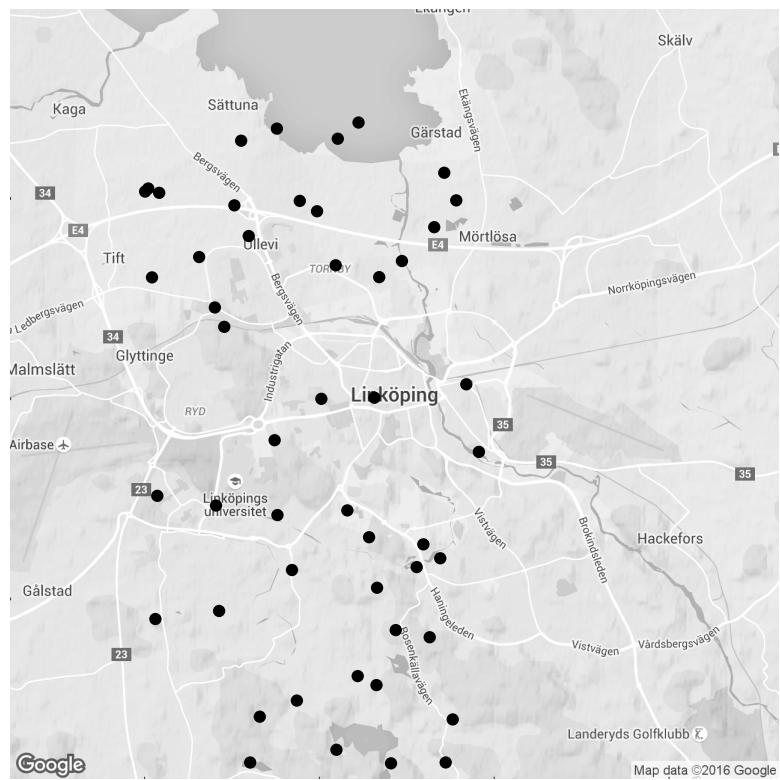
```
set.seed(1984)
linkpg_map <- qmap("Linkoping", color="bw", zoom =12)
df <-
  data.frame(
    lon = jitter(rep(15.6, 50), amount = .15),
    lat = jitter(rep(58.4, 50), amount = .152),
    event = sample(c("Kometnedslag", "Olycka", "Polis"), size = 50, replace = TRUE)
  )
```

```
Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=Linkoping&zoom=12&size=640x640
Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Linkoping&sensor=false
```

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

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

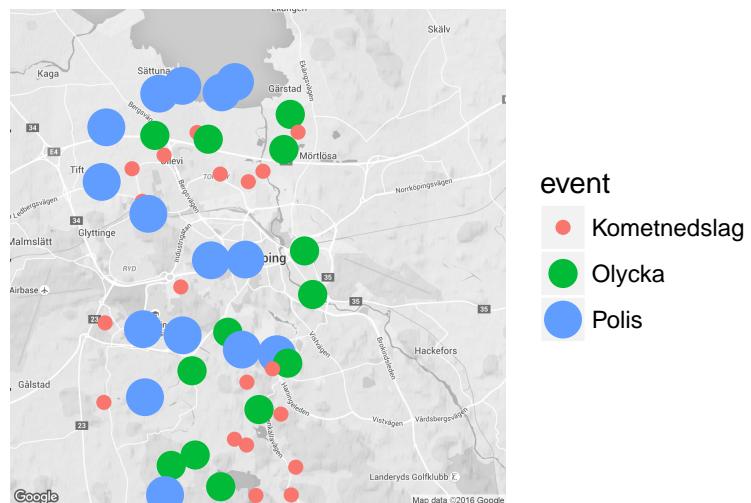
Warning: Removed 1 rows containing missing values (geom_point).
```



3. 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)

Warning: Removed 1 rows containing missing values (geom_point).
```

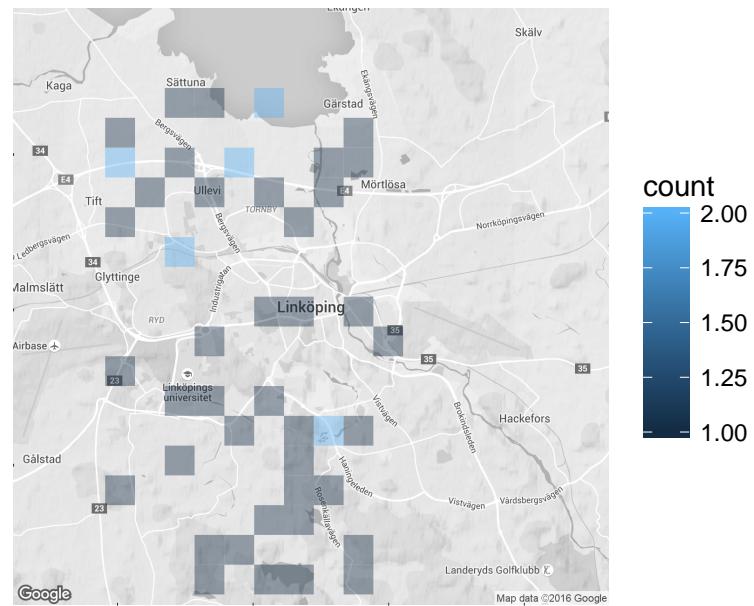


4. Vill vi visualisera frekvenser efter två kontinuerliga variabler finns dels `stat_bin2d` och `stat_density2d`

tillgängligt i ggplot2.

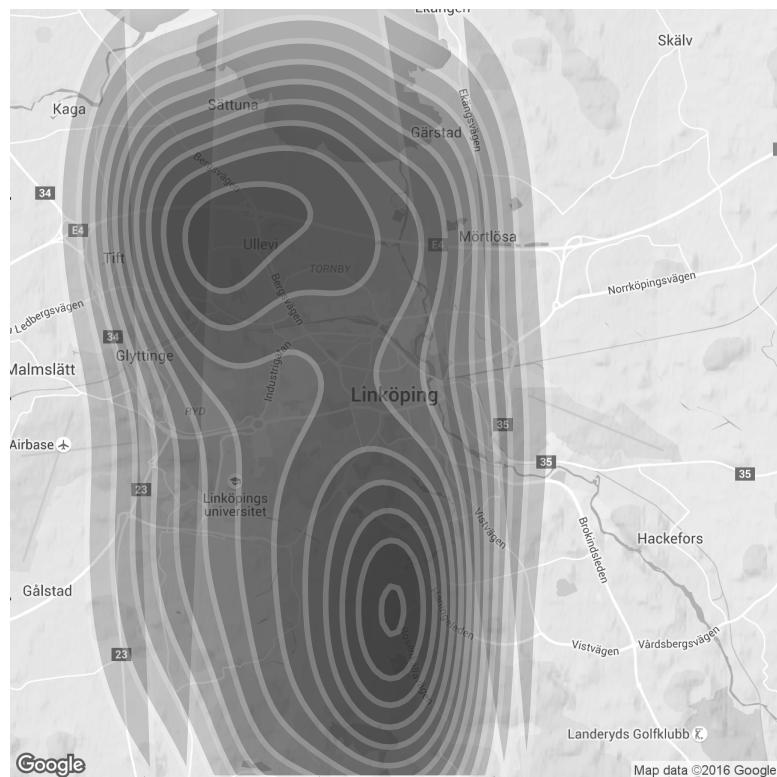
```
linkpg_map +  
  stat_bin2d(aes(x = lon, y = lat),  
             size = .5, bins = 20, alpha = 0.4, data = df)
```

```
Warning: Removed 1 rows containing non-finite values (stat_bin2d).
```



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

```
Warning: Removed 1 rows containing non-finite values (stat_density2d).
```



2.5 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 koordinatssystem. 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.

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

- 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=640x620
Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Linkoping&sensor=false

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.

Litteraturförteckning

- [1] Leland Wilkinson. *The grammar of graphics*. Springer, 2012.
- [2] Leland Wilkinson, D Wills, D Rope, A Norton, and R Dubbs. *The grammar of graphics*. Springer, 2006.

Del II

Inlämningsuppgifter

Tips!

Inlämningsuppgifterna innebär att konstruera funktioner. Ofta är det bra att bryta ned programmeringsuppgifter i färre små steg och testa att det fungerar i varje steg.

1. Lös uppgiften med vanlig kod direkt i R-Studio (precis som i datorlaborationen ovan) utan att skapa en funktion.
2. Testa att du får samma resultat som testexemplen.
3. Implementera koden du skrivit i 1. ovan som en funktion.
4. Testa att du får samma resultat som i testexemplen, nu med funktionen.

Automatisk återkoppling med `markmyassignment`

Som ett komplement för att snabbt kunna få återkoppling på de olika arbetsuppgifterna finns paketet `markmyassignment`. Med detta är det möjligt att direkt få återkoppling på uppgifterna i laborationen, oavsett dator. Dock krävs internetanalsutning.

Information om hur du installerar och använder `markmyassignment` för att få direkt återkoppling på dina laborationer finns att tillgå [här](#).

Samma information finns också i R och går att läsa genom att först installera `markmyassignment`.

```
install.packages("markmyassignment")
```

Om du ska installera ett paket i PC-pularana så behöver du ange följande:

```
install.packages("markmyassignment", lib = "sökväg till en mapp i din hemkatalog")
```

Tänk på att i sökvägar till mappar/filer i R i Windowssystem så används ‘‘\\’’, tex ‘‘C:\\Users\\Josef’’.

Därefter går det att läsa information om hur du använder `markmyassignment` med följande kommando i R:

```
vignette("markmyassignment")
```

Det går även att komma åt vignetten [här](#). Till sist går det att komma åt hjälppfilerna och dokumentationen i `markmyassignment` på följande sätt:

```
help(package = "markmyassignment")
```

Lycka till!

Kapitel 3

Inlämningsuppgifter

För att använda `markmyassignment` i denna laboration ange:

```
library(markmyassignment)

Loading required package: yaml
Loading required package: testthat
Loading required package: httr

lab_path <-
"https://raw.githubusercontent.com/STIMALiU/KursRprgm/master/Labs/Tests/d8.yml"
suppressWarnings(set_assignment(lab_path))

Assignment set:
D8 : Statistisk programmering med R: Lab 8
```

3.1 Svenska personnummer

I Sverige har samtliga medborgare personnummer som de behåller livet ut och som används för identifikation. Personnummret består av tre delar, födelsedatum, födelsenummer och en kontrollsiffra. Som standard anges personnummer på följande sätt AAAAMDDNNNK där AAAA är födelseåret, MM födelsemånaden, DD födelsedagen, NNN födelsenumret och K kontrollsiffran.

Kontrollsiffran beräknas baserat på de övriga siffrorna i personnummret vilket gör att det är möjligt att kontrollera om ett personnummer är korrekt eller inte. Det är också möjligt att utifrån ett personnummer beräkna ålder och kön (samt för vissa även födelseort, men det spelar ingen roll i denna uppgift).

Detaljerna om för hur kön och kontrollsiffran beräknas finns i Skatteverkets broschyr SKV 704 [PDF]. Läs igenom denna broschyr innan du gör uppgiften nedan.

Exempel på personnummer som kan användas för att testa dina funktioner finns dels i broschyren från Skatteverket och dels på Wikipedia (sökord: "Personnummer i Sverige"). Du kan självklart även testa med ditt eget personnummer om du vill.

Syftet med denna uppgifter att skapa flera mindre funktioner och sedan kombinera ihop dessa funktioner till en större mer komplex funktion.

Det vi vill ha i slutändan är en funktion som tar en vektor med personnummer på olika format. Funktionen ska sedan returnera ett dataset med den information som finns i personnummret (med undantag för födelselän). Vi tar det dock i flera steg, med flera olika funktioner som utför olika steg. De stegen vi kommer göra är:

1. Skapa en funktion för att konvertera personnummer till ett standardformat som vi kan arbeta med vidare.
2. Skapa en funktion för att kontrollera kontrollsiffran i ett personnummer.
3. Skapa en funktion för att kontrollera om personnummret är ett samordningsnummer.
4. Skapa en funktion för att ta fram uppgift om kön från ett personnummer.

5. Kontrollera/skapa ett datum att beräkna ålder från.
6. Skapa en funktion för att ta fram uppgifter om ålder från ett personnummer och ett givet datum.
7. Skapa en funktion som sätter samman funktionerna ovan till en funktion, som tar en vektor av personnummer som input och returnerar ett dataset med personnummer och övrig information.

Följande funktioner kommer vara mycket användbara i denna uppgift: `str_c()`/`paste()`, `str_sub/substr()` och `Sys.Date()`. Kolla upp dessa funktioner innan du sätter igång.

3.1.1 Uppgift 1: pnr_format()

Personnummer förekommer i många olika format i vanliga dataanalyser. De format funktionen ska kunna hantera är ÅÄMMDD-NNNK, ÅÄMMDDNNNK och AAAÄMMDDNNNK. Vi hoppar över personnummer på formen ÅÄMMDD+NNNK. Detta innebär att vi antar att alla personnummer är yngre än 100 år gamla. I R kan dessutom personnummer förekomma både som numeriska faktorvariabler och som textvariabler. Vår funktion ska klara samtliga dessa fall.

Funktionen ska heta `pnr_format()` med argumentet `pnr` och ska kunna ta ett personnummer på ett godtyckligt format och returnera personnummret som ett textelement med följande format: AAAÄMMDDNNNK

Ett förslag på de steg som kan ingå är:

1. Konvertera numeriska och faktorvariabler till text.
2. Använd en villkorssats för att hantera de tre olika formaten ovan [**Tips!** `str_length()`/`nchar()`]

Här är textexempel på hur funktionen ska fungera:

```
pnr <- "640823-3234"
pnr_format(pnr)

[1] "196408233234"

pnr <- 1311310324
pnr_format(pnr)

[1] "201311310324"

pnr <- "198112189876"
pnr_format(pnr)

[1] "198112189876"
```

3.1.2 Uppgift 2: pnr_ctrl()

Nästa steg i funktionen är att kontrollera om ett personnummer är korrekt eller inte. För att beräkna en kontrollsiffra används den så kallade Luhn-algoritmen, mer information finns [**här**]. Vi ska skapa en funktion som använder Luhn-algoritmen för att testa om ett personnummer är korrekt eller inte. Fördelen nu är att vi vet exakt på vilket format personnummren kommer att vara eftersom vi kommer använda funktionen `pnr_format()` innan vi anropar `pnr_ctrl()`.

Funktionen ska ta argumentet `pnr` och returnera TRUE eller FALSE beroende på om personnummret är korrekt eller inte.

Ett förslag på hur funktionen kan implementeras är följande:

1. Dela upp personnummret så respektive siffra blir ett eget element. [**Tips!** `str_split()`/`strsplt()` och `unlist()`]
2. Konvertera de uppdelade siffrorna till ett numeriskt format.
3. Den vektor av de enskilda siffrorna i personnummret kan nu användas i Luhn - algoritmen. Det enklaste sättet är att multiplicera personnummrets vektor med en beräkningsvektor av 0:or 1:or och 2:or på det sätt som beräkningen specificeras av Luhn-algoritmen.

Obs! Skatteverkets beräkning görs inte på hela personnummret som returnerades av `pnrFormat()`, de delar som inte ska räknas kan sättas till 0 i beräkningsvektorn.

4. Nästa steg är att summera alla värden i vektorn ovan. Tänk på att tal större än 9 ska räknas som summan av tiotalssiffran och entalssiffran. [Tips! %% och %/ %]
5. Summera värdena på vektorn som beräknades i 4 ovan. Plocka ut entalssiffran och dra denna entalssiffra från 10. Du har nu räknat ut kontrollsiffran. Puh!
6. Testa om den uträknade kontrollsiffran är samma som kontrollsiffran i personnummret.

Här är ett testexempel på hur funktionen ska fungera:

```
pnr <- "196408233234"
pnr_ctrl(pnr)

[1] TRUE

pnr <- "190101010101"
pnr_ctrl(pnr)

[1] FALSE

pnr <- "198112189876"
pnr_ctrl(pnr)

[1] TRUE

pnr <- "190303030303"
pnr_ctrl(pnr)

[1] FALSE
```

3.1.3 Uppgift 3: pnr.sex()

I denna uppgift ska vi från ett personnummer räkna ut det juridiska könet. Som framgår i skattebroschyren ska detta räknas ut genom att undersöka om den näst sista siffran i personnummret är jämt (kvinn) eller udda (man). Detta är vad som definierar en persons juridiska kön.

Skapa nu en funktion du kallar `pnr.sex()` med argumentet `pnr`. Denna funktion ska ta ett personnummer och returnera en persons kön som ett textelement, M för man och K för kvinna.

Ett förslag på hur funktionen kan implementeras är följande:

1. Plocka ut den näst sista siffran i personnumret.
2. Konvertera denna siffra till numeriskt format och testa om siffran är jämn (returnera K) eller udda (returnera M)

Här är testexempel på hur funktionen ska fungera:

```
pnr <- "196408233234"
pnr_sex(pnr)

[1] "M"

pnr <- "190202020202"
pnr_sex(pnr)

[1] "K"
```

3.1.4 Uppgift 4: pnr.samordn()

Vissa personer som inte är svenska medborgare kan få ett svenska samordningsnummer som fungerar på samma sätt som personnummer. Då får man ett så kallad samordningsnummer. Den enda skillnaden är att att talet 60 har lagts till personnummrets födelsedatum (d.v.s. DD i ÅÅÅÅMMDD).

Exempelvis en person som är född 19640823 får följande första siffror i personnumret: 19640883.

Skapa en funktion du kallar `pnr.samordn()` med argumentet `pnr` som tar ett personnummer på formatet genererat av funktionen `pnr.format()` och returnerar `TRUE` om det är ett samordningsnummer och `FALSE` annars.

Ett förslag på hur funktionen kan implementeras är följande:

1. Plocka ut födelsedatumet ur personnummret.
2. Konvertera datumet till ett numeriskt värde och prova om detta värde är större än 60.

Här är testexempel på hur funktionen ska fungera:

```
pnr <- "196408232324"  
pnr.samordn(pnr)
```

```
[1] FALSE
```

```
pnr <- "198112789876"  
pnr.samordn(pnr)
```

```
[1] TRUE
```

```
pnr <- "198112189876"  
pnr.samordn(pnr)
```

```
[1] FALSE
```

3.1.5 Uppgift 5: `pnr.date()`

Vi ska i denna funktion skapa ett datum för att senare beräkna åldern för olika individer. Skapa en funktion du kallar `pnr.date()` som tar argumentet `pnr` och `date`. Argumentet `date` ska ha följande textformat: `ÅÅÅÅ-MM-DD` och som default ska `NA` anges. Om datumet inte är på detta format ska funktionen stoppas med `stop()` och returnera följande felmeddelande:

```
Incorrect date format: Correct format should be YYYY-MM-DD.
```

Om inget datum anges av användaren ska den 31 december under föregående år returneras som datum.
Tips: lubridate

Ett förslag på hur funktionen kan implementeras är följande:

1. Ange ett defaultvärde för argumentet `date` som inte är aktuellt, ex. `NA`.
2. Om `date` har defaultvärdet, sätt datumvärdet den 31 december föregående år. [Tips! `is.na()`, `Sys.Date()` och `paste()`]
3. Testa om datumformatet är korrekt. Gör följande kontroller [Tips! `all()`]:
 - (a) Testa först om `date` är av typ `character`
 - (b) Är `ÅÅÅÅ`, `MM` och `DD` siffror. Detta kan kontrolleras genom att konvertera till numeriskt värde.
Är det då inte siffror blir värdet `NA`. [Tips! `is.na()`]
 - (c) Är `MM` större än 0 och mindre än 13.
 - (d) Är `DD` större än 0 och mindre än 32.
4. Om datumformatet är är inkorrekt stoppa funktionen och returnera felmeddelandet ovan.
5. Annars, returnera datumet på korrekt format.

Här är testexempel på hur funktionen ska fungera:

```
pnr.date("2010-10-10")
```

```
[1] "2010-10-10"
```

```

pnr_date()
[1] "2015-12-31"

pnr_date("Hejbaberiba")
Incorrect date format: Correct format should be YYYY-MM-DD.

pnr_date(TRUE)
Incorrect date format: Correct format should be YYYY-MM-DD.

pnr_date(1)
Incorrect date format: Correct format should be YYYY-MM-DD.

```

3.1.6 Uppgift 6: pnr_age()

Sist ska vi baserat på dels ett personnummer och dels ett datum beräkna åldern för personen vid detta datum. Skapa en funktion du kallar `pnr_age()` tar argumentet `pnr` och argumentet `date`. Argumentet `date` ska ha följande textformat: AAAA-MM-DD. Du kan utgå från att formatet är på detta sätt då `pnrDate()` kommer anropas innan denna funktion.

Ett förslag på hur funktionen kan implementeras är följande:

1. Räkna ut skillnaden i hela år mellan datumets årtal personnummrets årtal. D.v.s hur gammal personen är den 31 december.
Tips! Funktioner från `lubridate`, `as.Date()` och `as.character()` och `as.numeric()`.
2. Pröva om månad och dag är större (senare) för `pnr` än för `date`. Om så är fallet dra av ett år från årtalsberäkningen ovan (personen har ännu inte fyllt år) och returnera åldern vid det givna datumet.
3. Alt: Använd lämpliga funktioner från `lubridate`.

Här är testexempel på hur funktionen ska fungera:

```

pnr <- "196408233234"
pnr_age(pnr, date = "2010-10-10")

[1] 46

pnr <- "198112189876"
pnr_age(pnr, date = "2014-12-31")

[1] 33

```

3.1.7 Uppgift 7: pnr_info()

Nu har vi skapat ett antal funktioner för att beräkna olika delar av personnummret. Funktionen ska heta `pnr_info()` och ta argumentet `pnr` samt Nu ska vi sätta ihop dessa funktioner till en enda funktion som baserat på en vektor av personnummer returnerar en `data.frame` följande variabler:

1. `pnr`: personnummret i textformat,
2. `correct`: information om personnummret är korrekt,
3. `samordn`: om personnummret är ett samordningsnummer
4. `sex`: kön och
5. `age`: ålder i år

Det ska också vara möjligt att skicka vidare datum till funktionen `pnr_date()`, men om inget skickas med ska defaultvärdet i `pnr_date()` användas.

Funktionen ska dessutom generera följande meddelande med `message()`:

`The age has been calculated at [DATUM].`

Ett förslag på hur funktionen kan implementeras är följande:

1. Kontrollera/skapa ett korrekt datum för att beräkna ålder med `pnr_date()`.
2. Formatera om alla personnummer i inputvektorn till standardformatet med `pnr_format()`.
3. Räkna ut vilka personnummer som är korrekta personnummer med `pnr_ctrl()`.
4. Använd `pnr_samordn()` för att skapa en vektor över vilka personnummer som är samordningsnummer.
5. Använd `pnr_sex()` för att räkna ut könet för respektive personnummer.
6. Dra av 6 från första siffran i födelsedatumet för de personnummer som är samordningsnummer. Du kan antingen göra detta direkt i `pnr_info()` eller skapa en till egen funktion som gör just detta. Syftet med detta är att inte samordningsnummer ska ställa till problem när vi räknar ut åldern med `pnr_age()`.
7. Använd `pnr_age()` för att beräkna åldern för respektive personnummer.
8. Skriv ut meddelandet ovan med `message()`.
9. Sätt samman dessa resultat till den `data.frame` som ska returneras.

Här är testexempel på hur funktionen ska fungera:

```
pnr <- c("196408233234", "640883-3234", "198112189876", "700308-1242")
pnr_info(pnr)
```

`The age has been calculated at 2015-12-31.`

```
  pnr correct samordn sex age
1 196408233234    TRUE   FALSE   M  51
2 196408833234   FALSE    TRUE   M  51
3 198112189876    TRUE   FALSE   M  34
4 197003081242    TRUE   FALSE   K  45
```

```
pnr_info(pnr, date = "2000-06-01")
```

`The age has been calculated at 2000-06-01.`

```
  pnr correct samordn sex age
1 196408233234    TRUE   FALSE   M  35
2 196408833234   FALSE    TRUE   M  35
3 198112189876    TRUE   FALSE   M  18
4 197003081242    TRUE   FALSE   K  30
```

3.2 Miniprojektet del II

Den sista delen av denna laboration är att genomföra miniprojektet del II. Se kurshemsidan för detaljer.

Nu är du klar!