

Kursmaterial Introduktion till R - Certifierad Data Scientist

Filip Wästberg & Torbjörn Sjöberg, Solita

2020-11-16

Contents

1 Om dokumentet	5
2 Introduktion till R	7
2.1 Saknade värden (missing values)	8
2.2 R är ett funktionellt programmeringsspråk	10
2.3 Data frames	11
2.4 Paket	12
2.5 tidyverse and friends	12
2.6 Workflow i R	12
2.7 R Markdown	13
2.8 Kodstil: Att skriva kod i R	14
3 Datamanipulering med dplyr	17
3.1 Filter	17
3.2 Arrange	20
3.3 Select	21
3.4 %>% kallad “the pipe”	24
3.5 Mutate	24
3.6 Summarise	28
4 Explorativ dataanalys	33
4.1 Vad är Explorativ Dataanalys?	33
4.2 Datavisualisering	35
4.3 Datasaurus Dozen	37

4.4	Workflow för nytt dataset	88
4.5	Gapminder	89
4.6	Grammar of graphics	91
4.7	Gapminder - fortsättning	155
4.8	EDA är en nyfiken process utan slut!	159
4.9	Länkar till resurser	159
4.10	Övningar Explorativ dataanalys	160
4.11	Fördelningar	163
4.12	Korrelation	170
4.13	Pimp my plot!	171
4.14	Vi kan ändra tema med <code>theme</code>	171
4.15	Andra aesthetics	172
4.16	Diskreta variabler	173
4.17	Facets	173
4.18	Barcharts på aggregerade tabeller	174
4.19	Sortera kategorier	177
4.20	Sortera din bar-chart efter medianpris	177
4.21	Tidsserier	178
4.22	Grammar of Graphics - Artikel om the grammar of graphics som ggplot bygger på	181
4.23	R for Data Science	181
4.24	ggplot2 extensions	182
4.25	Shiny	182
5	Bilaga	183
5.1	Variabeltyper i R	183
5.2	Indexering	191

Chapter 1

Om dokumentet

För att ta del av det här materialet behöver du inte några särskilda förkunskaper. Övningarna och upplägget följer boken *R for Data Science* av Hadley Wickham och Garrett Grolemund som finns gratis. Den boken är ett utmärkt fördjupande komplement till det här materialet.

Chapter 2

Introduktion till R

R är ett programmeringsspråk för dataanalys. Men R sträcker sig långt utöver enkla databearbetningar och statistisk modellering. Tack vare ett aktivt community har det utvecklats en stor mängd paket för att lösa många av de olika uppgifter en dataanalytiker kan tänkas ställas inför.

R kan i sin enklaste form användas som en miniräknare med +, -, / eller *.

Exempelvis:

```
100 + 4
```

```
## [1] 104
```

Eller:

```
4 * 6 - 2
```

```
## [1] 22
```

Beräkningar, eller alla former av manipuleringar kan sparas i så kallade objekt.

Exempelvis kan vi spara en av ovanstående beräkningar i objektet `x` med `<-` som kallas för *the assign operator*.

```
x <- 100 + 4
x
```

```
## [1] 104
```

Pilen `<-` kan även användas på `->`

```
100 + 4 -> x
x
```

```
## [1] 104
```

Du kan spara flera värden i ett objekt genom att omsluta dem med funktionen `c()` och separatera med `,` (`c` står för *combine*). Då kallas objektet för en `vector`.

```
x <- c(4, 100 + 4, 10 * 2)
x
```

```
## [1] 4 104 20
```

Objekt och vektorer är inte begränsade till numeriska värden utan kan även innehålla text.

```
text <- c("hej", "jag", "älskar", "r")
text
```

```
## [1] "hej"     "jag"     "älskar"   "r"
```

Vi kan dock inte blanda text och numeriska värden. Då kommer R att tolka det som text.

```
blandat <- c(1, 5, "hej", 6)
blandat
```

```
## [1] "1"    "5"    "hej"   "6"
```

2.1 Saknade värden (missing values)

Ett vanligt fenomen i data från verkligheten är saknade värden. Saknade värden representeras i R med `NA`. `NA` är inte *noll*. Det är inte heller ett värde. Det är helt enkelt en indikation på att vi inte vet vilket värde som ligger där.

Det här upplevde i alla fall jag som lite förvirrande till en början.

Om vi har en vektor med `NA`:

```
x <- c(4, NA, 2, 50)
```

Och kollar vilka värden som är större än 2.

```
x > 2
```

```
## [1] TRUE NA FALSE TRUE
```

Så ser vi att vår observation med `NA` inte är `TRUE` eller `FALSE` utan helt enkelt `NA`.

Om vi vill kolla vilka värden som är `NA` borde vi kunna göra samma sak:

```
x == NA
```

```
## [1] NA NA NA NA
```

Det här svaret känns kanske inte intuitivt men det här beror på att vi helt enkelt inte vet. Det går inte att utvärdera om `x` är `NA`.

Vi kan illustrera med ett exempel:

```
filips_ålder <- NA
helenas_ålder <- NA
filips_ålder == helenas_ålder
```

```
## [1] NA
```

Är Filips ålder densamma som Helenas ålder för att vi saknar värdet? Nej, det går inte att veta.

Därför använder man i R istället `is.na()` för att undersöka om något är `NA`.

```
is.na(x)
```

```
## [1] FALSE TRUE FALSE FALSE
```

`NA` är ett vanligt argument i funktioner. Ska vi exempelvis beräkna medelvärdet på vår vektor `x` som har `NA` i sig så får vi:

```
mean(x)
```

```
## [1] NA
```

Eftersom vi saknar ett värde kan vi omöjligen veta vad medelvärdet för `x` är. Det saknade värdet kan vara vilken siffra som helst.

Därför kan vi i funktionen `mean()` välja att helt bortse från det saknade värdet:

```
mean(x, na.rm = TRUE)
```

```
## [1] 18.66667
```

2.2 R är ett funktionellt programmeringsspråk

Vi använder oss hela tiden av funktioner i R. Vi kan skriva egna funktioner men oftast använder vi färdiga funktioner från paket med funktioner.

Funktionell programmering är bra för dataanalys. Ofta jobbar vi med data som vi vill manipulera om och om igen och funktioner är ett bra sätt att göra det på. På många sätt liknar R därför Excel som också jobbar mycket med funktioner.

2.2.1 Funktioner

Precis som i Excel finns det flertalet inbyggda statistiska och matematiska funktioner i R:

- `mean()`
- `median()`
- `sd()`
- ...osv

Det finns också matematiska funktioner

- `log()`
- `sin()`
- `cos()`
- ...osv

2.2.2 Dokumentation av funktioner

Funktioner innehåller också dokumentation om hur de fungerar.

För att komma åt dokumentation skriver du ett frågetecken innan funktioner.

```
?mean()
```

2.2.3 Övning

- Använd några av R:s statistiska funktioner på en numerisk vektor

2.3 Data frames

- Det vanligaste formatet i R för dataanalys
- En data.frame är en rektangulär tabell med en eller flera kolumner

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	car
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA
2013	1	1	542	540	2	923	850	33	AA
2013	1	1	544	545	-1	1004	1022	-18	B6
2013	1	1	554	600	-6	812	837	-25	DL
2013	1	1	554	558	-4	740	728	12	UA

Vi kan enkelt skapa våra egna data.frames.

```
data.frame(random_number = rnorm(5))
```

```
##   random_number
## 1 -0.1351667
## 2 -1.4240132
## 3  0.2962874
## 4  1.8660142
## 5  0.7850128
```

En data.frame är ett bra sätt att spara en eller flera vektorer. De måste då vara exakt lika långa.

```
data.frame(x = c(1,2,3),
           y = c("hej", "på", "dig"))
```

```
##   x   y
## 1 1 hej
## 2 2 på
## 3 3 dig
```

Om du sparar vektorerna som två objekt kan du enkelt göra dem till den data.frame.

```
siffror <- c(5,1,2,5)
ord <- c("vad", "var", "det", "där")

data.frame(siffror, ord)
```

```
##  siffror ord
## 1      5 vad
## 2      1 var
## 3      2 det
## 4      5 där
```

2.4 Paket

Vi nämnde tidigare att vi kan spara funktioner i så kallade paket. Paket är precis det. Ett sätt för oss att spara och lätt komma åt funktioner i R. Paket som genomgår viss granskning läggs upp på CRAN (The Comprehensive R Archive Network). Men man kan även skapa egna paket och lägga upp på Github eller spara på sin egen dator.

För att installera ett paket från CRAN använder du funktionen `install.packages("paketnamn")`.

När du sedan laddat ner paketet behöver du ladda det i R genom `library(paketnamn)`.

2.4.1 Övning

Paketet `tidyverse` är nedladdat åt dig. Ladda paketet med `library()`

2.5 tidyverse and friends

`tidyverse` är en samling paket för att lösa olika typer av problem inom dataanalys.

De har en gemensam filosofi: paketen och dess funktioner är i första hand designade för att människor ska använda dem.

Det gör att de av många upplevs som enklare att använda och mer konsekventa än det som kallas *base R*. D.v.s. R utan några paket.

Vi kommer här att fokusera på `tidyverse` eftersom det är ett bra sätt att komma igång med R.

2.6 Workflow i R

I R så jobbar vi med så kallade R-projects. I Rstudio kan du klicka på *file -> New Project -> New Directory -> Create Project* för att skapa ett nytt projekt. Genom att skapa ett projekt vet R alltid var filerna för det projektet är och det blir enklare för dig att jobba.

2.7 R Markdown

R Markdown är ett fil-format för att bygga dynamiska rapporter och dokument i R. R Markdown är skrivet i *markdown*, som är ett enkelt dokumentformat, och innehåller inbäddad R-kod, i form av kodblock. Du kan enkelt infoga ett kodblock i ditt R Markdown genom **ctrl + alt + i** på Windows och **alt + command + i** på Mac.

En R Markdown-fil kan exporteras till HTML, PDF, Word eller Powerpoint samt en rad andra format.

Längst upp i ett R Markdown-dokument finns en **YAML**. I den skriver du titel på dokumentet, författare och output-format. Om du exempelvis vill ha ett word-dokument skriver du:

```
---
title: "Min titel"
author: "Mitt namn"
output: word_document
---
```

Men det vanligaste, och enklaste formatet att arbeta med är **html_document**.

Nedan har du ett exempel ett sådant dokument:

```
---
title: "Min titel"
author: "Mitt namn"
output: html_document
---
```

Det här är ett R Markdown-dokument, här kan vi kombinera skriven text med inbäddad R-kod. Vi kan

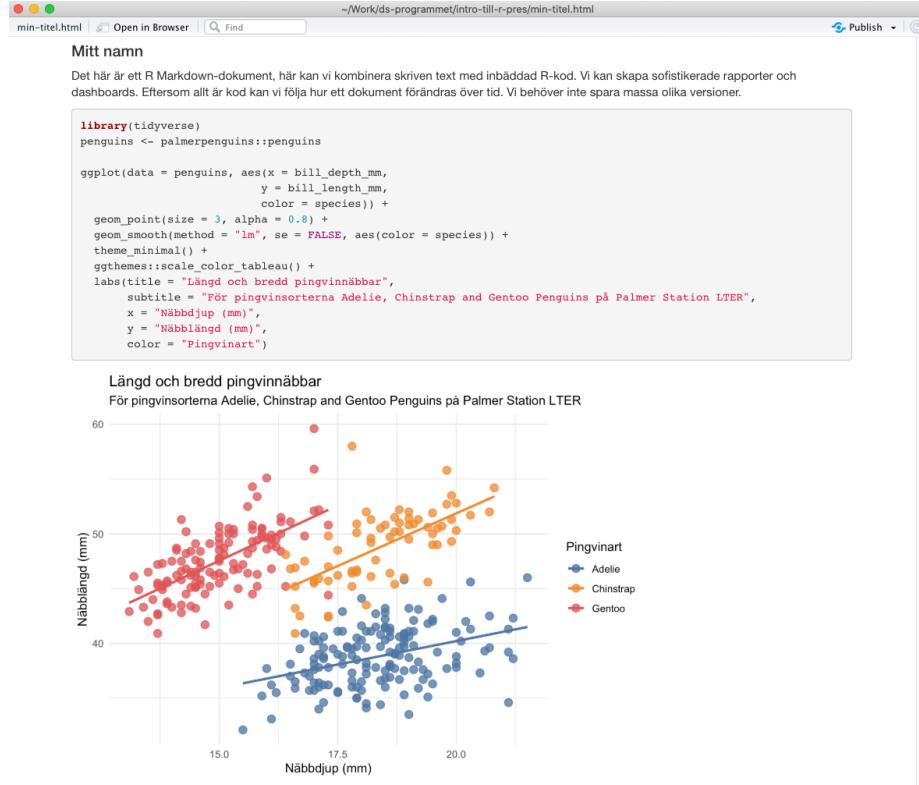
```
```{r, eval=TRUE}
library(tidyverse)
penguins <- palmerpenguins::penguins

ggplot(data = penguins, aes(x = bill_depth_mm,
 y = bill_length_mm,
 color = species)) +
 geom_point(size = 3, alpha = 0.8) +
 geom_smooth(method = "lm", se = FALSE, aes(color = species)) +
 theme_minimal() +
 ggthemes::scale_color_tableau() +
 labs(title = "Längd och bredd pingvinnäbbar",
 subtitle = "För pingvinsorterna Adelie, Chinstrap and Gentoo Penguins på Palmer Station LT")
```

```
x = "Näbbdjup (mm)",
y = "Näbblängd (mm)",
color = "Pingvinart")
```

```

Klickar du sedan på Knit längst upp till vänster i dokumentet, eller kör `knitr::render("filnamn.rmd")` så blir resultatet:



2.8 Kodstil: Att skriva kod i R

När du skriver kod gör du det dels med hänsyn dels till andra men framför allt med hänsyn till ditt framtida jag. Koden ska med andra ord vara enkel att läsa.

Därför kan det vara bra att följa en stilguide.

Jag följer stilguiden från `tidyverse` som säger att:

- Namnge alltid variabler, objekt m.m. med små bokstäver.

Exempelvis är det enklare att läsa:

```
min_egna_funktion <- function(x)
```

I kontrast till:

```
MinEgnaFunktion <- function(x)
```

Vi strävar dessutom efter att skriva kod som vi skriver text, med mellanrum mellan , och ord.

Det här är enklare att läsa:

```
mean(x, na.rm = TRUE)
```

Än det här:

```
mean(x,na.rm=TRUE)
```

När vi sparar filer så försöker vi följa den här syntaxen. Därför ska du inte ha mellanrum i när du sparar filer. `min-r-fil.R` är bra men `min R fil.R` är dåligt.

När vi skriver kod försöker vi dessutom inte att skriva för lång uttryck:

Det här är mycket svårare att läsa:

```
iris %>% group_by(Species) %>% summarise(Sepal.Length = mean(Sepal.Length), Sepal.Width = mean(Sepal.Width))
```

Än det här:

```
iris %>%
  group_by(Species) %>%
  summarise(
    Sepal.Length = mean(Sepal.Length),
    Sepal.Width = mean(Sepal.Width),
    Species = n_distinct(Species)
  )
```


Chapter 3

Datamanipulering med dplyr

Det sägs ofta att en Data Scientist ägnar 80% av sin tid till att manipulera data så att den går att visualisera och modellera. Därför är det klokt att välja en metod och ett paket som underlättar det arbetet för dig.

I den här introduktionen kommer vi att fokusera på att använda paketet `dplyr` för att göra datamanipuleringar. `dplyr` är ett av de mest populära paketen i R och ger dig en bred verktygslåda för att manipulera data. `dplyr` ingår också i ett samlingspaket `tidyverse` och samlar flertalet paket för datamanipulering, visualisering och modellering.

`dplyr` har ett antal *verb* för att göra manipuleringar:

- `filter()` där du väljer observationer baserat på deras värden
- `arrange()` som ändrar ordningen på rader
- `select()` för att välja variabler baserat på deras namn
- `mutate()` för att skapa nya variabler baserat på funktioner
- `summarise()` för att summera många värden till ett värde

Samtliga av dessa verb kan användas i kombination med funktionen `group_by()` som innebär att du utför verben på flera grupper.

Alla verb i `dplyr` är konsekventa. Det första argumentet är din data och i det andra argumentet specificerar du vad du vill göra med din data. Resultatet är alltid en ny data.frame.

3.1 Filter

Med filter kan du enkelt filtrera din data baserat på villkor.

Dessa villkor uttrycks med hjälp av `relationsoperatorer` och `logical operators`.

I R är dessa:

| Relationsoperator | Symbol i R |
|-------------------|------------|
| och (and) | & |
| eller(or) | |
| icke(not) | ! |

| Logical Operators | Symbol i R |
|----------------------|------------|
| lika | == |
| inte lika | != |
| större än eller lika | >= |
| mindre än eller lika | <= |
| större än | > |
| mindre än | < |
| finns i | %in% |

Dessa kan du använda i `filter()`.

PS: Det vanligaste misstaget i början av din R-karriär är att skriva = istället för ==.

Så här använder du operatorerna:

Hitta alla flyg som kom fram 08:30 under februari

```
library(dplyr)
library(nycflights13)
filter(flights, month == 2, arr_time == 830)
```

```
## # A tibble: 9 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>    <dbl>    <int>          <int>
## 1  2013     2     2      656          700     -4       830          839
## 2  2013     2     4      652          600      52       830          759
## 3  2013     2     6      629          630     -1       830          845
## 4  2013     2    13      633          636     -3       830          808
## 5  2013     2    18      717          700      17       830          832
## 6  2013     2    24      557          600     -3       830          837
## 7  2013     2    25      532          540     -8       830          850
## 8  2013     2    26      615          615      0       830          820
## 9  2013     2    28      621          630     -9       830          830
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Hitta alla flygbolag som finns i `c("UA", "DL")` i februari eller mars och som inte var försenade.

```
filter(flights, carrier %in% c("UA", "DL"), month == 2 | month == 3, dep_delay <= 0)

## # A tibble: 10,346 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>          <int>    <dbl>     <int>          <int>
## 1 2013     2     1      520            525     -5       816          820
## 2 2013     2     1      527            530     -3       837          829
## 3 2013     2     1      554            601     -7       920          918
## 4 2013     2     1      558            600     -2       738          759
## 5 2013     2     1      559            600     -1       923          925
## 6 2013     2     1      600            600      0       833          837
## 7 2013     2     1      601            608     -7       703          725
## 8 2013     2     1      601            608     -7       723          755
## 9 2013     2     1      604            610     -6       752          817
## 10 2013    2     1      608            615     -7       837          842
## # ... with 10,336 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

3.1.1 Övning

- Öppna dokumentet `databearbetning-övningar.Rmd`
- Börja med att ladda `tidyverse` och `nycflights13`
- Spara `flights` som en `data.frame` i din `environment`, genom `flights <- flights`

PS. `flights` är en `tibble` som bara är en lite snyggare variant av en `data.frame`.

Nu till övningarna:

Hur många plan...

1. var försenade mer än 1 timme?
2. skulle till Boston ("BOS")
3. lämnade JFK och var försenade
4. lämnade JFK på julafaston
5. var försenade, men inte kom fram försent
6. flög United Airlines (UA) eller American Airlines?

3.2 Arrange

`arrange()` kastar om ordningen på dina rader enligt en av dig vald variabel. Exempelvis kanske vi vill sortera data på försenade avgångar `dep_delay`.

```
arrange(flights, dep_delay)
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
## 1 2013     12     7    2040        2123      -43       40        2352
## 2 2013      2     3    2022        2055      -33      2240        2338
## 3 2013     11    10    1408        1440      -32      1549        1559
## 4 2013      1    11    1900        1930      -30      2233        2243
## 5 2013      1    29    1703        1730      -27      1947        1957
## 6 2013      8     9     729         755      -26      1002        955
## 7 2013     10    23    1907        1932      -25      2143        2143
## 8 2013      3    30    2030        2055      -25      2213        2250
## 9 2013      3     2    1431        1455      -24      1601        1631
## 10 2013     5     5    934         958      -24      1225        1309
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

`arrange()` sorterar per default på sjunkande värde (ascending). Vill du sortera på stigande värde (descending) sätter du `desc()` runt din variabel.

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
## 1 2013     1     9     641        900      1301      1242        1530
## 2 2013     6    15    1432       1935      1137      1607        2120
## 3 2013     1    10    1121       1635      1126      1239        1810
## 4 2013     9    20    1139       1845      1014      1457        2210
## 5 2013     7    22     845       1600      1005      1044        1815
## 6 2013     4    10    1100       1900      960       1342        2211
## 7 2013     3    17    2321       810       911       135         1020
## 8 2013     6    27     959       1900      899       1236        2226
## 9 2013     7    22    2257       759       898       121         1026
## 10 2013    12     5    756       1700      896      1058        2020
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

3.2.1 Övning

Hitta det flyg som...

1. flög längst
2. var mest försenat (när det kom fram)

3.3 Select

Medan `arrange()` kastar om raderna så kastar `select()` om kolumnerna. Men mest används den för att välja ut kolumner av intresse.

```
select(flights, dep_delay, carrier)
```

```
## # A tibble: 336,776 x 2
##   dep_delay carrier
##       <dbl> <chr>
## 1        2 UA
## 2        4 UA
## 3        2 AA
## 4       -1 B6
## 5       -6 DL
## 6       -4 UA
## 7       -5 B6
## 8       -3 EV
## 9       -3 B6
## 10      -2 AA
## # ... with 336,766 more rows
```

Om du av något skäl vill flytta en kolumn till början kan du skriva:

```
select(flights, carrier, everything())
```

```
## # A tibble: 336,776 x 19
##   carrier year month   day dep_time sched_dep_time dep_delay arr_time
##   <chr>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1 UA      2013    1     1     517        515        2     830
## 2 UA      2013    1     1     533        529        4     850
## 3 AA      2013    1     1     542        540        2     923
## 4 B6      2013    1     1     544        545       -1    1004
## 5 DL      2013    1     1     554        600       -6     812
## 6 UA      2013    1     1     554        558       -4     740
```

```
## 7 B6      2013     1     1    555       600      -5     913
## 8 EV      2013     1     1    557       600      -3     709
## 9 B6      2013     1     1    557       600      -3     838
## 10 AA     2013     1     1    558       600      -2     753
## # ... with 336,766 more rows, and 11 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Du kan även välja alla kolumner mellan två kolumner:

```
select(flights, year:day)
```

```
## # A tibble: 336,776 x 3
##   year month   day
##   <int> <int> <int>
## 1 2013     1     1
## 2 2013     1     1
## 3 2013     1     1
## 4 2013     1     1
## 5 2013     1     1
## 6 2013     1     1
## 7 2013     1     1
## 8 2013     1     1
## 9 2013     1     1
## 10 2013    1     1
## # ... with 336,766 more rows
```

Genom att sätta ett minus framför variabelnamnet exkluderar du variabeln.

```
select(flights, -year)
```

```
## # A tibble: 336,776 x 18
##   month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int>    <int>        <int>    <int>      <dbl>      <int>        <int>
## 1     1     1      517          515        2      830        819
## 2     1     1      533          529        4      850        830
## 3     1     1      542          540        2      923        850
## 4     1     1      544          545       -1     1004       1022
## 5     1     1      554          600       -6      812        837
## 6     1     1      554          558       -4      740        728
## 7     1     1      555          600       -5      913        854
## 8     1     1      557          600       -3      709        723
## 9     1     1      557          600       -3      838        846
## 10    1     1      558          600       -2      753        745
```

```
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

3.3.0.1 select() plus hjälpfunktioner

`select()` kommer med ett antal hjälpfunktioner:

- `starts_with("asd")`
- `ends_with("air")`
- `contains("flyg")`
- `matches("asd")`
- `num_range("flyg", 1:10)`

De används för att identifiera kolumner baserat på text.

Exempelvis kan du hitta alla delay-kolumner med `contains`.

```
select(flights, contains("delay"))
```

```
## # A tibble: 336,776 x 2
##       dep_delay arr_delay
##           <dbl>      <dbl>
## 1          2        11
## 2          4        20
## 3          2        33
## 4         -1       -18
## 5         -6       -25
## 6         -4        12
## 7         -5        19
## 8         -3       -14
## 9         -3        -8
## 10        -2         8
## # ... with 336,766 more rows
```

3.3.0.2 rename()

En annan nyttig funktion är `rename()` som kort och gott döper om variabler.

Formeln är `rename(data, ny_variabel = gammal_variabel)`

3.3.1 Övning

1. Som innehåller “dep”
2. Som börjar med “dep”
3. Som är numeriska
4. Döp om `dep_delay` till `försenad_avgång`

3.4 %>% kallad “the pipe”

- Används för att länka ihop flera uttryck i *R*.
- `ctrl + shift + m` (i Windows) och `ctrl + command + m` (Mac)

```
flights %>%
  filter(month == 2, arr_time == 830) %>%
  select(dest, year, month, day, arr_time)
```

```
## # A tibble: 9 x 5
##   dest    year month   day arr_time
##   <chr>  <int> <int> <int>   <int>
## 1 ORD     2013    2     2     830
## 2 DTW     2013    2     4     830
## 3 CLT     2013    2     6     830
## 4 BNA     2013    2    13     830
## 5 BNA     2013    2    18     830
## 6 ATL     2013    2    24     830
## 7 MIA     2013    2    25     830
## 8 CLT     2013    2    26     830
## 9 CLT     2013    2    28     830
```

3.5 Mutate

`mutate()` används för att skapa nya variabler.

Exempelvis kan vi räkna ut hur mycket tid man vunnit om exempelvis flyget landar tidigare än avsett.

```
flights %>%
  mutate(beer_time = dep_delay - arr_delay)
```

```
## # A tibble: 336,776 x 20
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>          <int>     <dbl>    <int>        <int>
## 1 2013     1     1      517         515       2     830        819
## 2 2013     1     1      533         529       4     850        830
## 3 2013     1     1      542         540       2     923        850
## 4 2013     1     1      544         545      -1    1004       1022
## 5 2013     1     1      554         600      -6     812        837
## 6 2013     1     1      554         558      -4     740        728
## 7 2013     1     1      555         600      -5     913        854
## 8 2013     1     1      557         600      -3     709        723
## 9 2013     1     1      557         600      -3     838        846
## 10 2013    1     1      558         600      -2     753        745
## # ... with 336,766 more rows, and 12 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   beer_time <dbl>
```

`mutate()` kan användas för att slå ihop variabler

```
flights %>%
  mutate(date = paste(year, month, day, sep = "-")) %>%
  select(year, month, day, date)
```

```
## # A tibble: 336,776 x 4
##   year month   day date
##   <int> <int> <int> <chr>
## 1 2013     1     1 2013-1-1
## 2 2013     1     1 2013-1-1
## 3 2013     1     1 2013-1-1
## 4 2013     1     1 2013-1-1
## 5 2013     1     1 2013-1-1
## 6 2013     1     1 2013-1-1
## 7 2013     1     1 2013-1-1
## 8 2013     1     1 2013-1-1
## 9 2013     1     1 2013-1-1
## 10 2013    1     1 2013-1-1
## # ... with 336,766 more rows
```

I `mutate()` kan du även använda funktioner såsom `mean()`.

```
flights %>%
  mutate(mean_delay = mean(dep_delay, na.rm = T)) %>%
  select(year, month, day, mean_delay)
```

```
## # A tibble: 336,776 x 4
##   year month   day mean_delay
##   <int> <int> <int>     <dbl>
## 1 2013    1     1      12.6
## 2 2013    1     1      12.6
## 3 2013    1     1      12.6
## 4 2013    1     1      12.6
## 5 2013    1     1      12.6
## 6 2013    1     1      12.6
## 7 2013    1     1      12.6
## 8 2013    1     1      12.6
## 9 2013    1     1      12.6
## 10 2013   1     1      12.6
## # ... with 336,766 more rows
```

3.5.1 if_else()

En vanlig funktion i databearbetning är ifelse-satser.

I R gör du det enklast med funktionen `if_else()` från `dplyr`. Det finns även en inbyggd funktion som heter `ifelse()` som mestadels fungerar bra men den från `dplyr` är något mer stabil.

Du kan använda den i mutate:

```
mutate(flights, försenad = if_else(dep_delay > 5, "försenad", "ej försenad"))
```

```
## # A tibble: 336,776 x 20
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>        <int>     <dbl>     <int>        <int>
## 1 2013    1     1      517          515      2.00      830        819
## 2 2013    1     1      533          529      4.00      850        830
## 3 2013    1     1      542          540      2.00      923        850
## 4 2013    1     1      544          545     -1.00     1004       1022
## 5 2013    1     1      554          600     -6.00      812        837
## 6 2013    1     1      554          558     -4.00      740        728
## 7 2013    1     1      555          600     -5.00      913        854
## 8 2013    1     1      557          600     -3.00      709        723
## 9 2013    1     1      557          600     -3.00      838        846
## 10 2013   1     1      558          600     -2.00      753        745
## # ... with 336,766 more rows, and 12 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   försenad <chr>
```

3.5.2 case_when()

I bland vill man göra flera stycken `if_else()` i samma, exempelvis om man vill dela upp en variabel i flera kategorier beroende på ett logiskt villkor. För att göra det kan du använda funktionen `case_when()`.

```
mutate(flights, försenad_kat = case_when(
  dep_delay < 0 ~ "före tid",
  dep_delay == 0 ~ "i tid",
  dep_delay > 0 ~ "försenad"))

## # A tibble: 336,776 x 20
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1 2013     1     1      517            515       2     830          819
## 2 2013     1     1      533            529       4     850          830
## 3 2013     1     1      542            540       2     923          850
## 4 2013     1     1      544            545      -1    1004         1022
## 5 2013     1     1      554            600      -6     812          837
## 6 2013     1     1      554            558      -4     740          728
## 7 2013     1     1      555            600      -5     913          854
## 8 2013     1     1      557            600      -3     709          723
## 9 2013     1     1      557            600      -3     838          846
## 10 2013    1     1      558            600      -2     753          745
## # ... with 336,766 more rows, and 12 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   försenad_kat <chr>
```

3.5.3 Andra funktioner

I `mutate()` kan du använda de allra flesta funktionerna i R. Här är exempel på några funktioner som kan vara nyttiga i databearbetning:

- Funktioner för att ranka variabler: `rank()`, `min_rank()`, `dense_rank()`, `percent_rank()`
- För att logaritmiska funktioner: `log()`, `log10()`
- För kumulativa beräkningar: `cumsum()`, `cummean()`
- För att bara generera radnummer: `row_number()`
- För att ta observation innan eller efter: `lead()` och `lag()`
- För skapa variabler baserat på värde i andra variabler `if_else()` och `case_when()`

3.5.4 Övningar

1. Skapa en variabel med `dep_delay minus` medelvärdet för `dep_delay`
2. Rangordna flygens distans med `dense_rank()`. Hur hanterar den distanser är lika långa? Vilka fler rankingfunktioner skulle du kunna använda? Se `?dense_rank()`
3. Skapa en variabel som anger om flyget går på våren, hösten, vintern eller sommaren med `case_when()`

3.6 Summarise

Ofta vill man summera variabler för att få ut intressant information. Exempelvis vill vi här kanske veta en rad medelvärden.

```
flights %>%
  summarise(
    mean_air_time = mean(air_time, na.rm = T),
    mean_dep_delay = mean(dep_delay, na.rm = T),
    mean_arr_delay = mean(arr_delay, na.rm = T)
  )

## # A tibble: 1 x 3
##   mean_air_time mean_dep_delay mean_arr_delay
##       <dbl>           <dbl>           <dbl>
## 1         151.          12.6            6.90
```

Men dessa värden är inte så intressanta i sig, utan vi vill kunna göra jämförelse. Då använder vi `group_by()`.

```
flights %>%
  group_by(dest) %>%
  summarise(mean_air_time = mean(air_time, na.rm = T),
            mean_dep_delay = mean(dep_delay, na.rm = T),
            mean_arr_delay = mean(arr_delay, na.rm = T)) %>%
  arrange(desc(mean_arr_delay))

## `summarise()` ungrouping output (override with `.`groups` argument)

## # A tibble: 105 x 4
##   dest   mean_air_time mean_dep_delay mean_arr_delay
##   <chr>      <dbl>           <dbl>           <dbl>
```

```

## 1 CAE          93.2      35.6      41.8
## 2 TUL          178.      34.9      33.7
## 3 OKC          193.      30.6      30.6
## 4 JAC          264.      26.5      28.1
## 5 TYS          97.7      28.5      24.1
## 6 MSN          123.      23.6      20.2
## 7 RIC          54.0      23.6      20.1
## 8 CAK          64.0      20.8      19.7
## 9 DSM          149.      26.2      19.0
## 10 GRR          97       19.5      18.2
## # ... with 95 more rows

```

I `summarise()` kan du använda en rad olika funktioner såsom `sum()` för summeringar, `n()` för antal rader, `median()` för median etc.

```

flights %>%
  group_by(dest) %>%
  summarise(
    n_rows = n(),
    sum_air_time = sum(air_time, na.rm = T),
    median_air_time = median(air_time, na.rm = T)) %>%
  arrange(desc(sum_air_time))

## `summarise()` ungrouping output (override with `.`groups` argument)

## # A tibble: 105 x 4
##   dest  n_rows sum_air_time median_air_time
##   <chr> <int>      <dbl>           <dbl>
## 1 LAX    16174     5258470          328
## 2 SFO    13331     4553684          345
## 3 ORD    17283     1914833          114
## 4 ATL    17215     1901410          112
## 5 MCO    14082     1882381          133
## 6 FLL    12055     1811271          151
## 7 LAS    5997      1795662          301
## 8 MIA    11728     1775327          152
## 9 DFW    8738      1667341          198
## 10 DEN   7266      1623626          225
## # ... with 95 more rows

```

3.6.1 count()

Du kan i `summarise()` räkna antalet observationer med funktionen `n()`.

```
flights %>%
  group_by(carrier) %>%
  summarise(count = n())
```

Men istället för att göra det här kan du använda funktionen `count()`

```
flights %>%
  group_by(carrier) %>%
  count()
```

```
## # A tibble: 16 x 2
## # Groups:   carrier [16]
##   carrier     n
##   <chr>    <int>
## 1 9E        18460
## 2 AA        32729
## 3 AS         714
## 4 B6        54635
## 5 DL        48110
## 6 EV        54173
## 7 F9         685
## 8 FL        3260
## 9 HA         342
## 10 MQ       26397
## 11 OO         32
## 12 UA       58665
## 13 US       20536
## 14 VX        5162
## 15 WN       12275
## 16 YV         601
```

3.6.2 `sample_n()`

Att göra slumpmässiga urval är en vanlig arbetsuppgift för en data scientist. Det gör du enkelt med `sample_n()` från `dplyr`.

Om du vill ta ett slumpmässigt urval om 10 från exempelvis `flights` gör du bara:

```
sample_n(flights, 10)
```

```
## # A tibble: 10 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
```

```

##   <int> <int> <int>   <int>      <int>    <dbl>   <int>   <int>
## 1 2013     1     8    1942     1935      7  2140  2145
## 2 2013    11    20    1800     1801     -1  1946  1944
## 3 2013     2    19    1919     1855      24  2058  2100
## 4 2013     8    26    1550     1550      0  1735  1749
## 5 2013    11    11    2104     2107     -3  2218  2226
## 6 2013    10    24    1618     1615      3  1909  1915
## 7 2013     7    11    1132     1130      2  1333  1327
## 8 2013     1    21    1123     1130     -7  1412  1422
## 9 2013     3    14    1427     1415     12  1632  1631
## 10 2013    6    27   1951     1723     148    NA  2013
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>

```

Om du istället vill ta ett urval som baseras på *procent* kan du använda `sample_frac`, här för 0.1%

```
sample_frac(flights, 0.001)
```

```

## # A tibble: 337 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>   <int>      <int>    <dbl>   <int>      <int>
## 1 2013     5    23    2343     2159     104     52    2326
## 2 2013     9    26    1827     1829     -2    2016    2011
## 3 2013     3     9    1137     1139     -2    1430    1446
## 4 2013    10    31     840      630     130    1114     839
## 5 2013     5    22    1835     1700      95    2006    1819
## 6 2013     8     1    1344     1338      6    1642    1634
## 7 2013     1    11     757      755      2    909     930
## 8 2013     3    26    1730     1730      0    1923    1923
## 9 2013     8     5    1513     1520     -7    1705    1710
## 10 2013    8    25    1729     1735     -6    1918    1929
## # ... with 327 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>

```

3.6.2.1 Övningar

1. Vilken flygplats har högst medelvärde förförseningar från flygplatsen?
2. Vilken flygplats tar emot minst flyg?

3.6.3 Övning - väv ihop allt du lärt dig

1. Vilken månad ska du flyga under för att undvika förseningar?

2. Vilken tid ska du flyga för att undvika förseningar?
3. Hitta det flygbolag som flyger till flest destinationer, rangordna flygbolagen baserat på den informationen.

Chapter 4

Explorativ dataanalys

4.1 Vad är Explorativ Dataanalys?

- En kreativ iterativ process där man ställer frågor/gör hypoteser om sin data och sedan utforskar dem
- Mera konkret kan det beskrivas som
 1. Formulera en hypotes
 2. Transformera din data - om nödvändigt
 3. Visualisera din data (vårt fokus idag)
 4. Modellera - om nödvändigt
 5. Formulera nya hypoteser
- Mindset inte formel!
-

4.1.1 Exempel på basala frågor/hypoteser

Frågeställningar kan vara vaga!

- Vad för data innehåller de olika kolonnerna?
- Hur ser Datans kvalitét ut? e.g missing values, felaktiga tabellvärden, duplicerade värden
- Finns det outliers?
- Medelvärde, spridning
-

4.1.2 Hur gör man i R?

R har basfunktionalitet som kan användas för att snabbt få en numerisk överblick datan t.ex `summary()` och `head()`

```
summary(iris)
```

```
##   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
## Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
## 1st Qu.:5.100  1st Qu.:2.800  1st Qu.:1.600  1st Qu.:0.300
## Median :5.800  Median :3.000  Median :4.350  Median :1.300
## Mean   :5.843  Mean   :3.057  Mean   :3.758  Mean   :1.199
## 3rd Qu.:6.400  3rd Qu.:3.300  3rd Qu.:5.100  3rd Qu.:1.800
## Max.   :7.900  Max.   :4.400  Max.   :6.900  Max.   :2.500
##
## Species
## setosa    :50
## versicolor:50
## virginica :50
##
```

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1       3.5      1.4       0.2  setosa
## 2          4.9       3.0      1.4       0.2  setosa
## 3          4.7       3.2      1.3       0.2  setosa
## 4          4.6       3.1      1.5       0.2  setosa
## 5          5.0       3.6      1.4       0.2  setosa
## 6          5.4       3.9      1.7       0.4  setosa
```

4.1.3 Hur gör man i R?

R har även mer avancerade verktyg: - `dlookr`, ta fram rapport för datakvalitét

- `skimr`, snabb explorativ analys (som vi kommer använda idag)

```
skim(iris)
```

```
— Data Summary —————
      Values
Name          iris
Number of rows 150
Number of columns 5
—————
Column type frequency:
  factor     1
  numeric    4
—————
Group variables   None
—————
— Variable type: factor —————
  skim_variable n_missing complete_rate ordered n_unique top_counts
1 Species           0             1 FALSE            3 set: 50, ver: 50, vir: 50
—————
— Variable type: numeric —————
  skim_variable n_missing complete_rate   mean    sd   p0   p25   p50   p75   p100 hist
1 Sepal.Length      0             1 5.84 0.828  4.3  5.1  5.8  6.4  7.9 
2 Sepal.Width       0             1 3.06 0.436   2   2.8   3   3.3   4.4 
3 Petal.Length      0             1 3.76 1.77    1   1.6  4.35  5.1  6.9 
4 Petal.Width       0             1 1.20 0.762   0.1  0.3  1.3  1.8  2.5 
```

4.2 Datavisualisering

- Grunden i explorativ dataanalys
- Visualisering är grundläggande för att människor ska kunna få en överblick av data

4.2.1 Varför visualisera?

Anscombe quartet - Dataset skapat av Francis Anscombe 1973

```
##   observation set  x     y
## 1           I 10 8.04
## 2           I 10 9.14
## 3           III 10 7.46
## 4           IV  8 6.58
## 5           I   8 6.95
## 6           II  8 8.14
## 7           III 8 6.77
## 8           IV  8 5.76
## 9           I 13 7.58
## 10          II 13 8.74
```

4.2.2 Vad är det första vi gör?

```
## # A tibble: 4 x 5
##   set    x_mean y_mean  x_sd  y_sd
##   <chr>  <dbl>  <dbl> <dbl> <dbl>
## 1 I      9     7.50  3.32  2.03
## 2 II     9     7.50  3.32  2.03
## 3 III    9     7.5   3.32  2.03
## 4 IV     9     7.50  3.32  2.03
```

Ser bra ut!

4.2.3 Steg 2: anpassar en linjär modell

```
anscombe_tidy %>%
  nest(observation, x, y) %>%
  mutate(model = map(data, ~lm(y ~ x, data = .x)),
         tidy = map(model, broom::tidy)) %>%
  unnest(tidy) %>%
  filter(term == "x") %>%
  select(set, estimate, std.error, statistic)
```

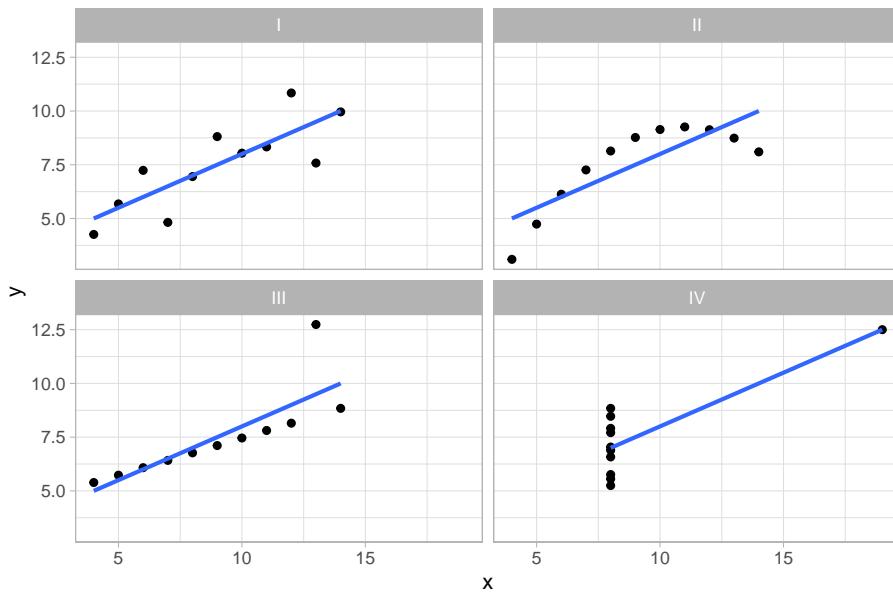
```
## Warning: All elements of `...` must be named.
## Did you want `data = c(observation, x, y)`?
```

```
## # A tibble: 4 x 4
##   set    estimate std.error statistic
##   <chr>     <dbl>     <dbl>     <dbl>
## 1 I       0.500    0.118     4.24
## 2 II      0.5       0.118     4.24
## 3 III     0.500    0.118     4.24
## 4 IV      0.500    0.118     4.24
```

Ser inte ut som att det är någon skillnad mellan grupperna

Men...

```
## `geom_smooth()` using formula 'y ~ x'
```



Source: Anscombe, F. J. (1973). Graphs in Statistical Analysis

Statistiska mått kan gömma mycket av sanningen!

4.3 Datasaurus Dozen

Modern variant av Anscombe - skapat av Alberto Cairo

```
library(datasaurus)
datasaurus_dozen %>%
  group_by(dataset) %>%
  summarise(x_mean = mean(x), y_mean = mean(y),
            x_sd = sd(x), y_sd = sd(y),
            corr = cor(x,y))

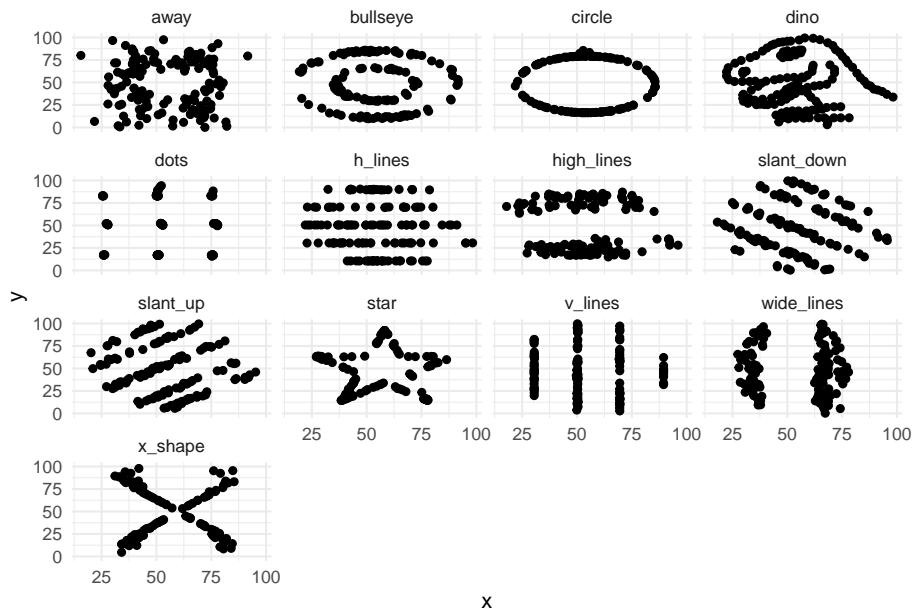
## `summarise()` ungrouping output (override with `^.groups` argument)

## # A tibble: 13 x 6
##   dataset    x_mean   y_mean   x_sd   y_sd   corr
##   <chr>      <dbl>     <dbl>   <dbl>   <dbl>   <dbl>
## 1 away       54.3     47.8    16.8   26.9  -0.0641
## 2 bullseye   54.3     47.8    16.8   26.9  -0.0686
## 3 circle     54.3     47.8    16.8   26.9  -0.0683
## 4 dino       54.3     47.8    16.8   26.9  -0.0645
## 5 dots       54.3     47.8    16.8   26.9  -0.0603
```

```
## 6 h_lines      54.3  47.8 16.8 26.9 -0.0617
## 7 high_lines   54.3  47.8 16.8 26.9 -0.0685
## 8 slant_down   54.3  47.8 16.8 26.9 -0.0690
## 9 slant_up     54.3  47.8 16.8 26.9 -0.0686
## 10 star        54.3  47.8 16.8 26.9 -0.0630
## 11 v_lines      54.3  47.8 16.8 26.9 -0.0694
## 12 wide_lines   54.3  47.8 16.8 26.9 -0.0666
## 13 x_shape      54.3  47.8 16.8 26.9 -0.0656
```

Datasaurus Dozen

```
ggplot(datasaurus_dozen, aes(x=x, y=y))+
  geom_point()+
  theme_minimal() +
  facet_wrap(~dataset)
```

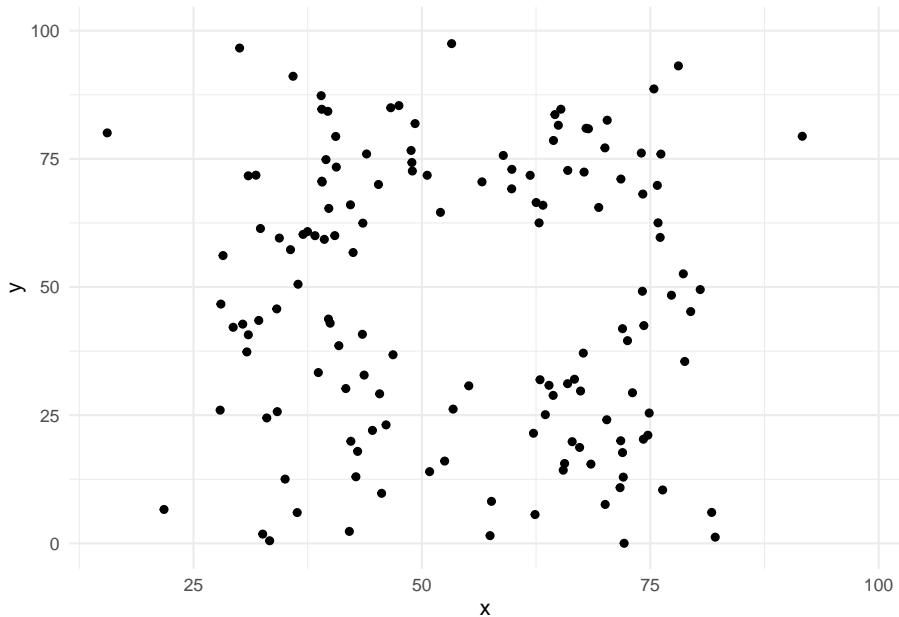
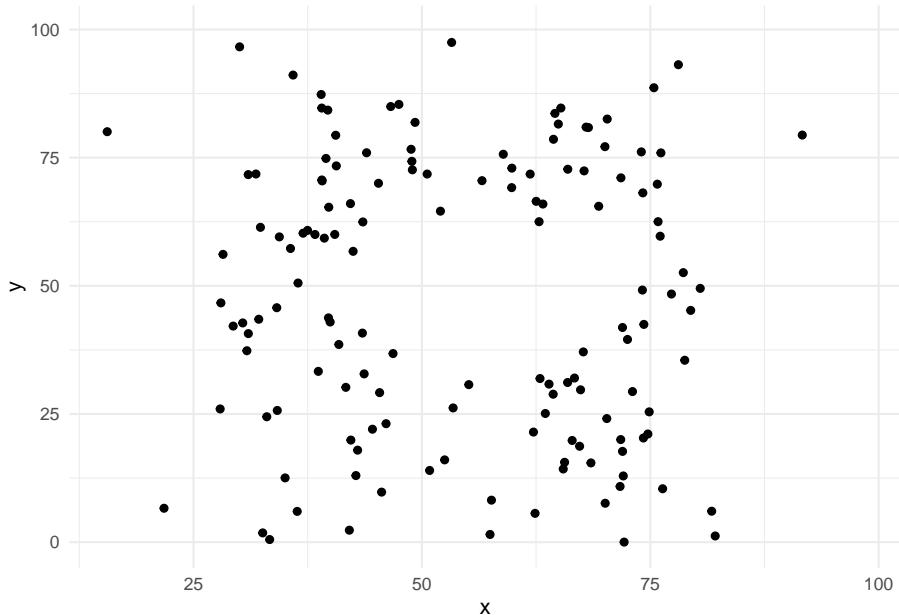


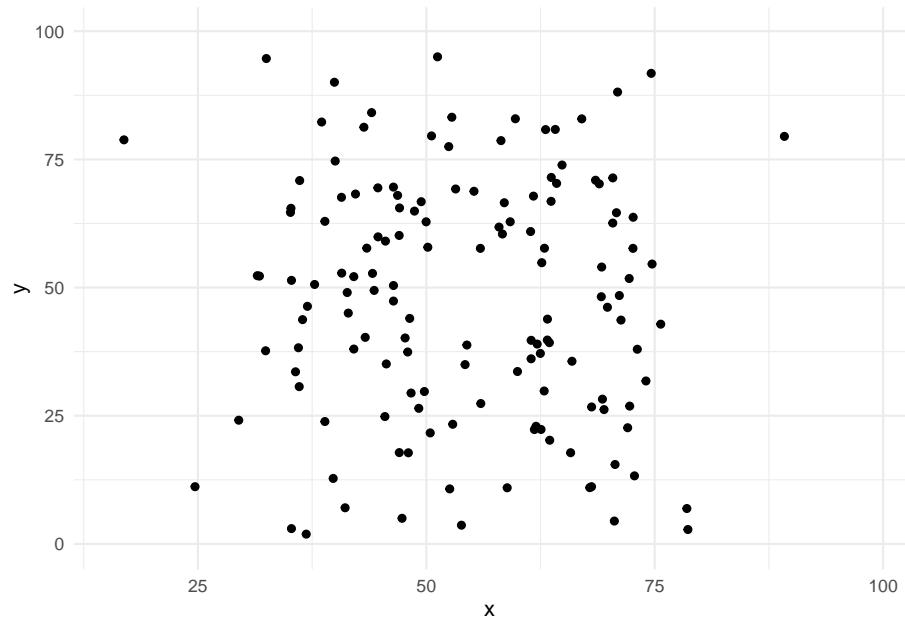
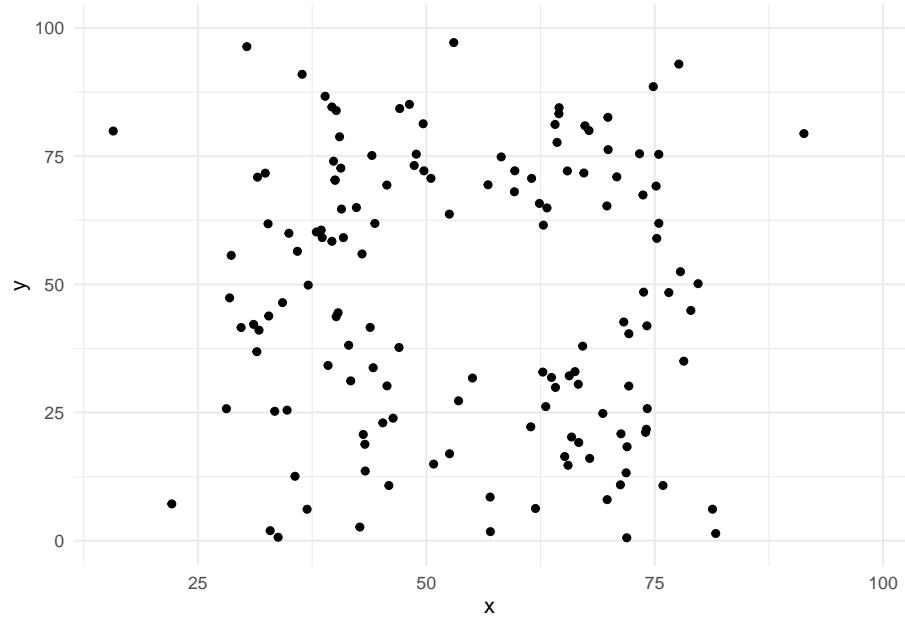
Eller som animering:

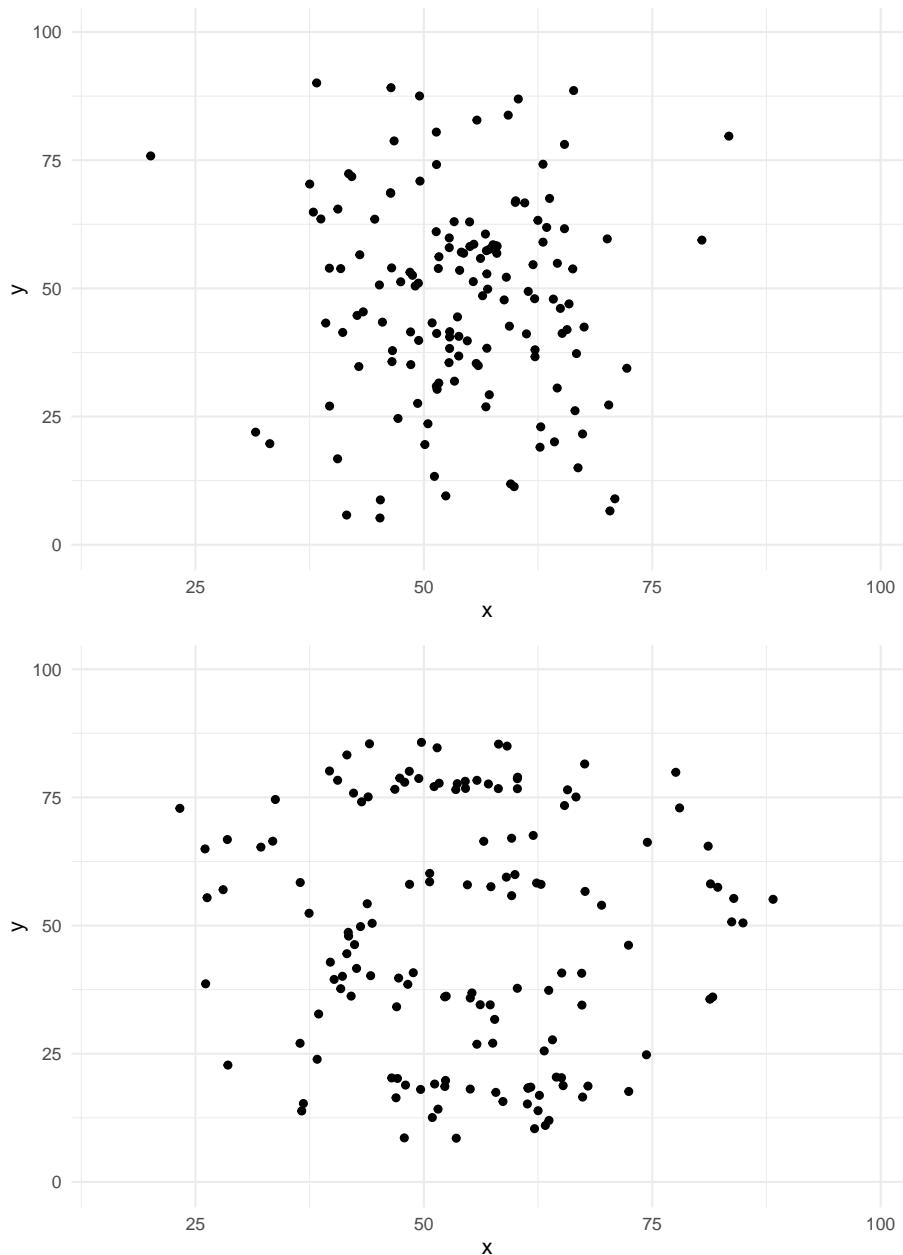
```
library(datasauRus)
library(ggplot2)
library(gganimate)

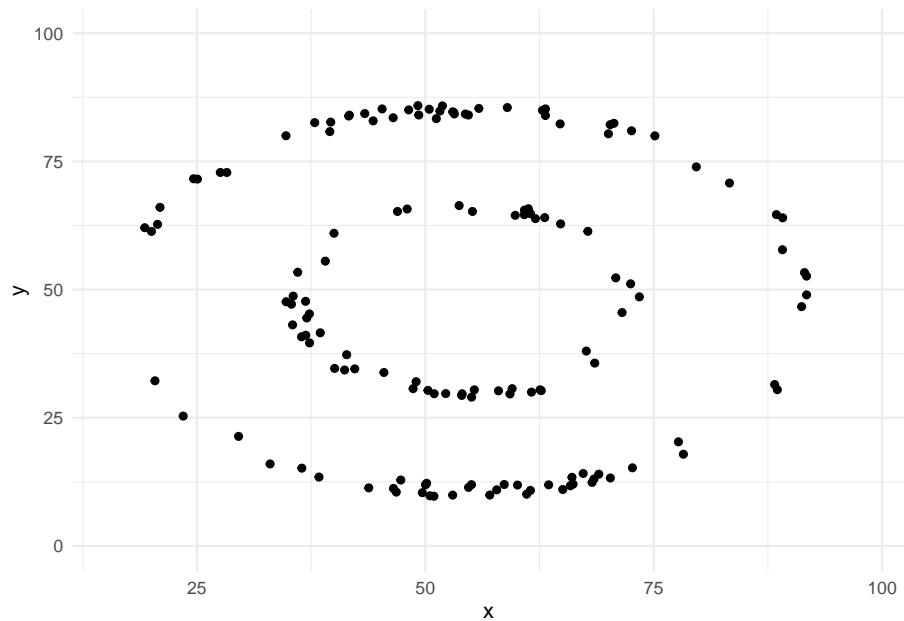
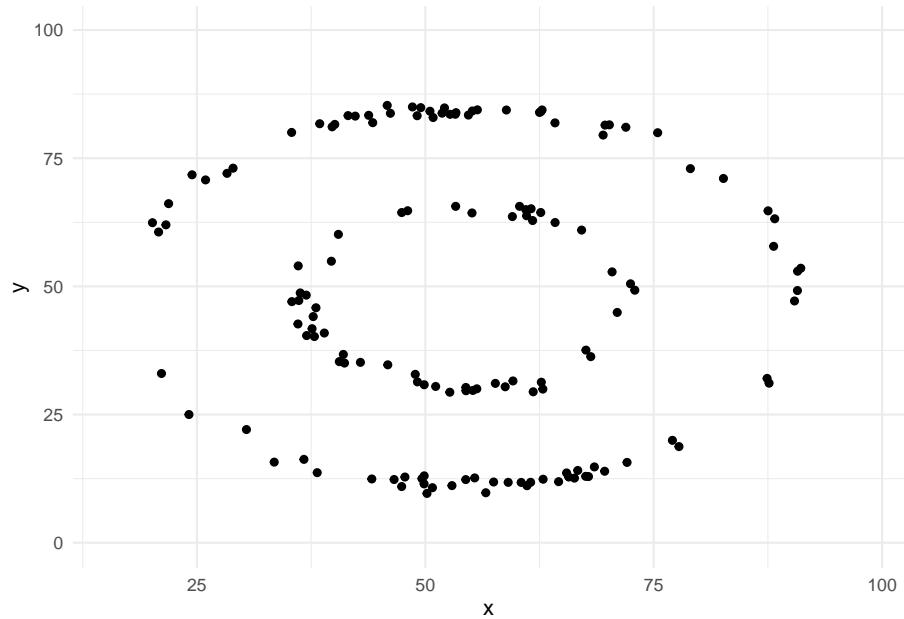
ggplot(datasaurus_dozen, aes(x=x, y=y))+
  geom_point()+
```

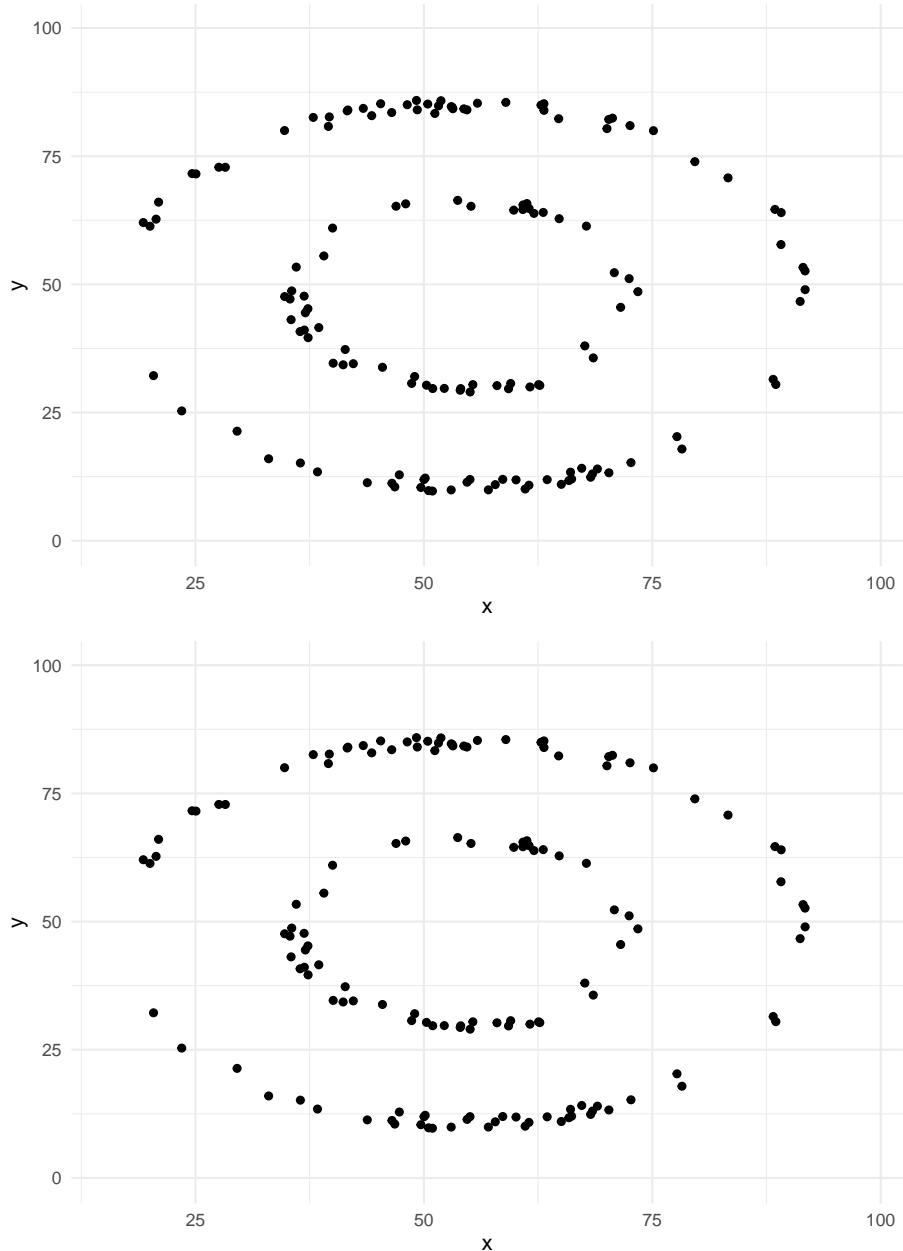
```
theme_minimal() +  
transition_states(dataset, 3, 1) +  
ease_aes('cubic-in-out')
```

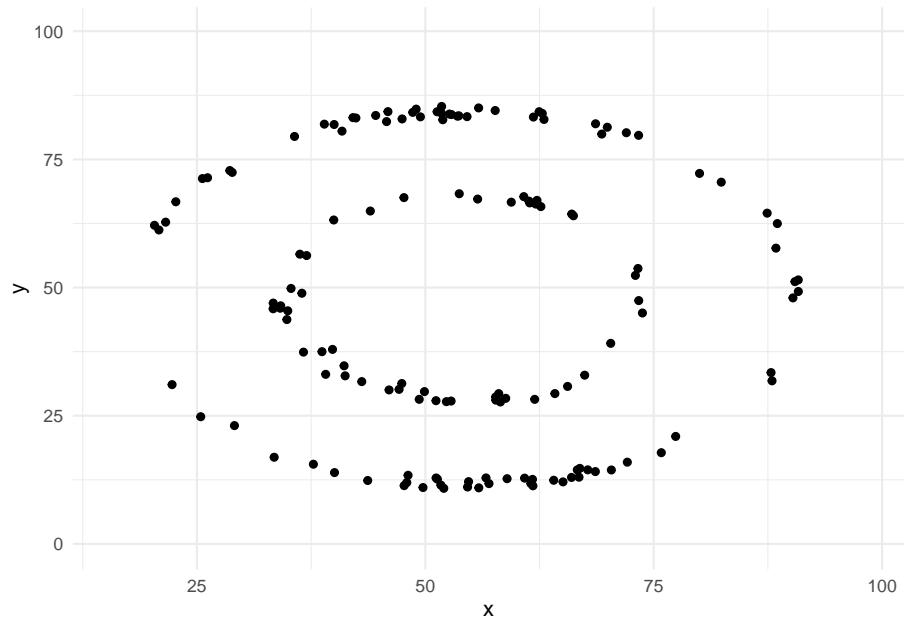
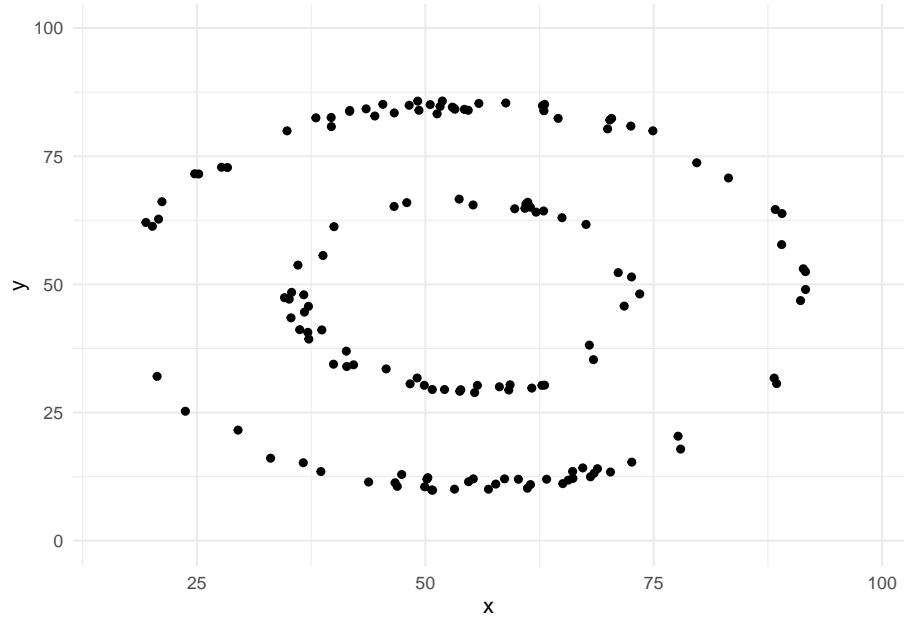


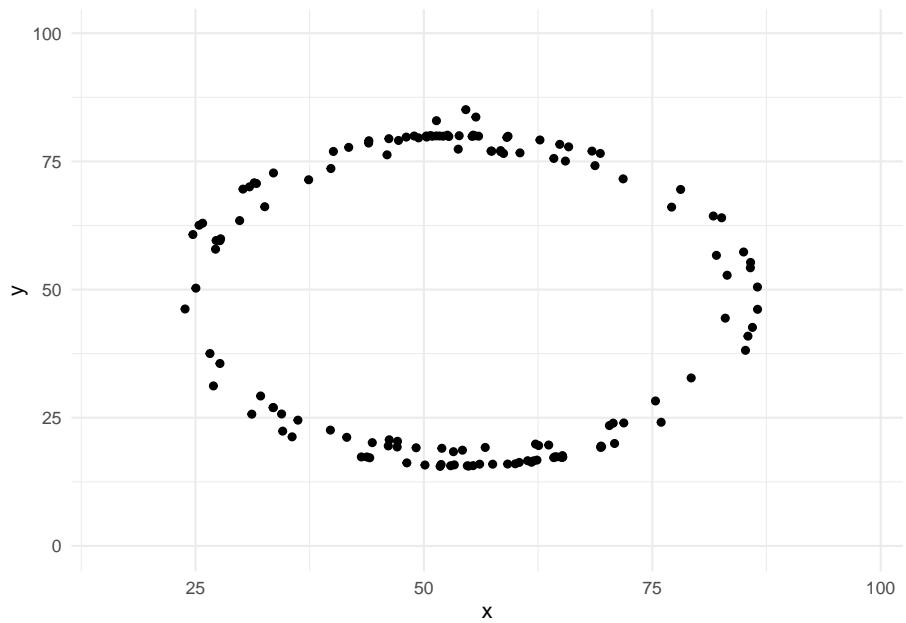
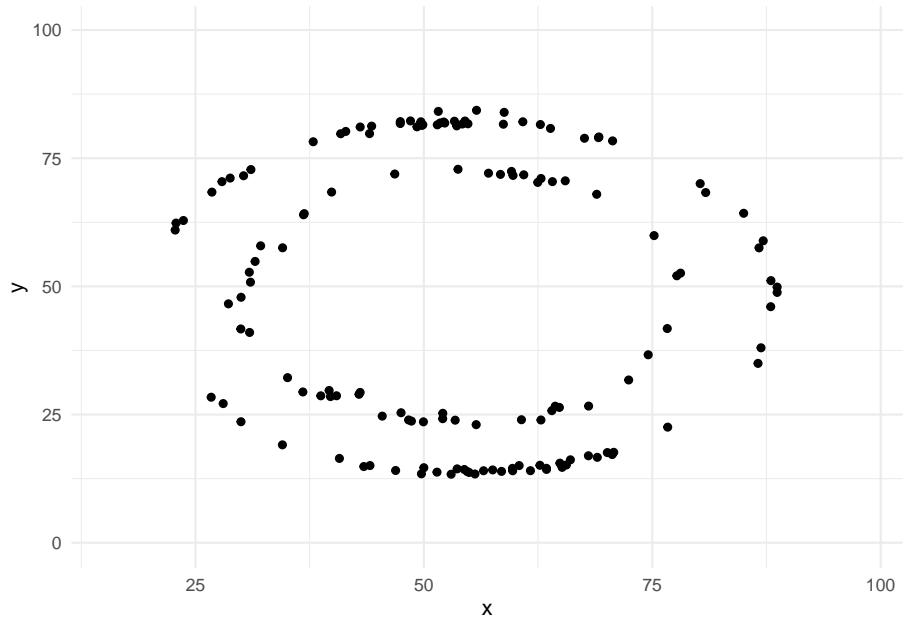


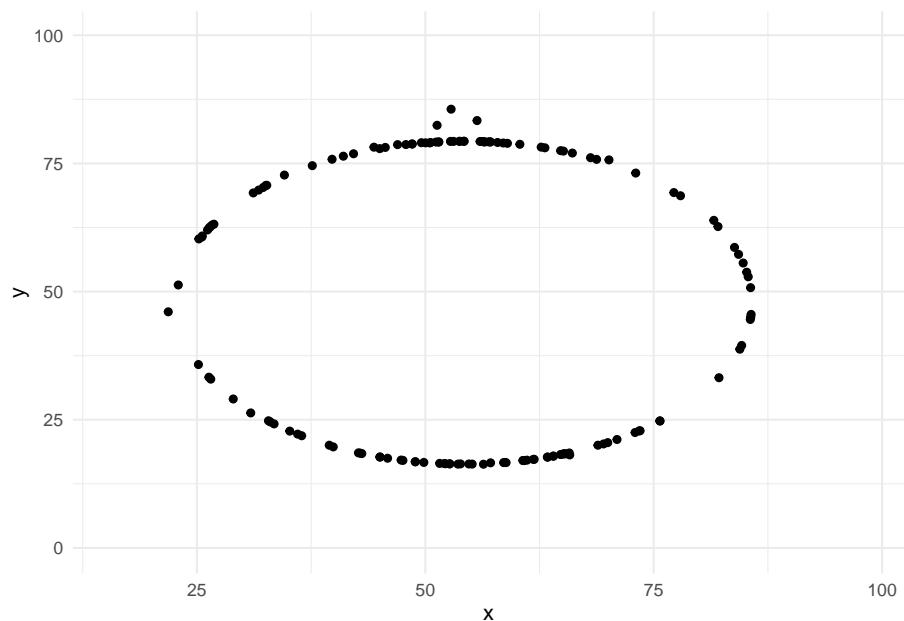
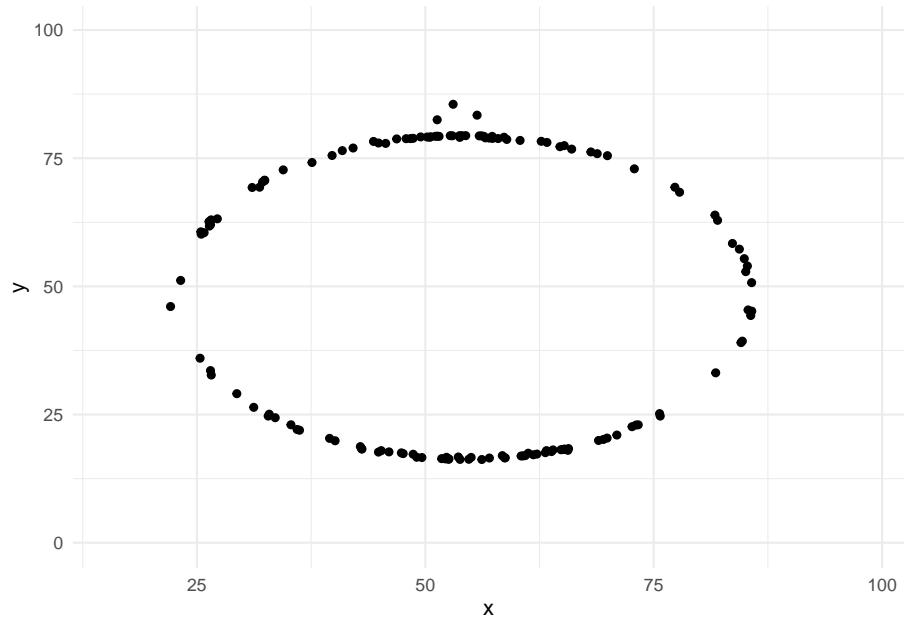


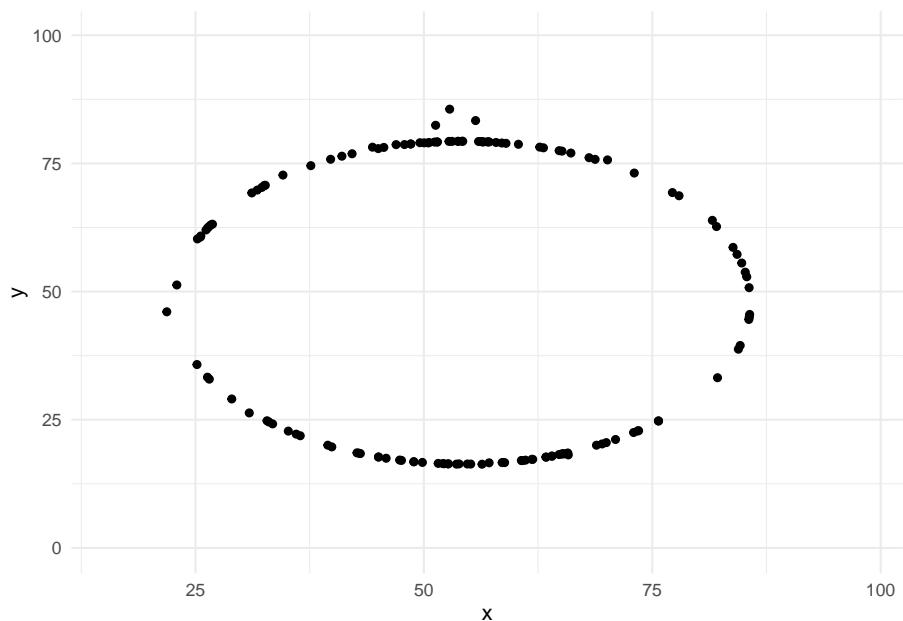
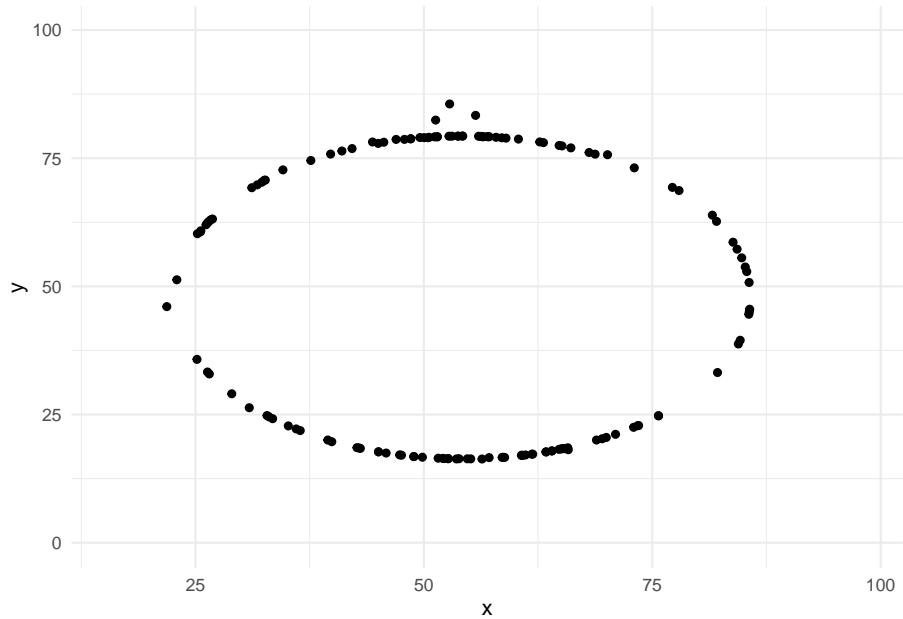


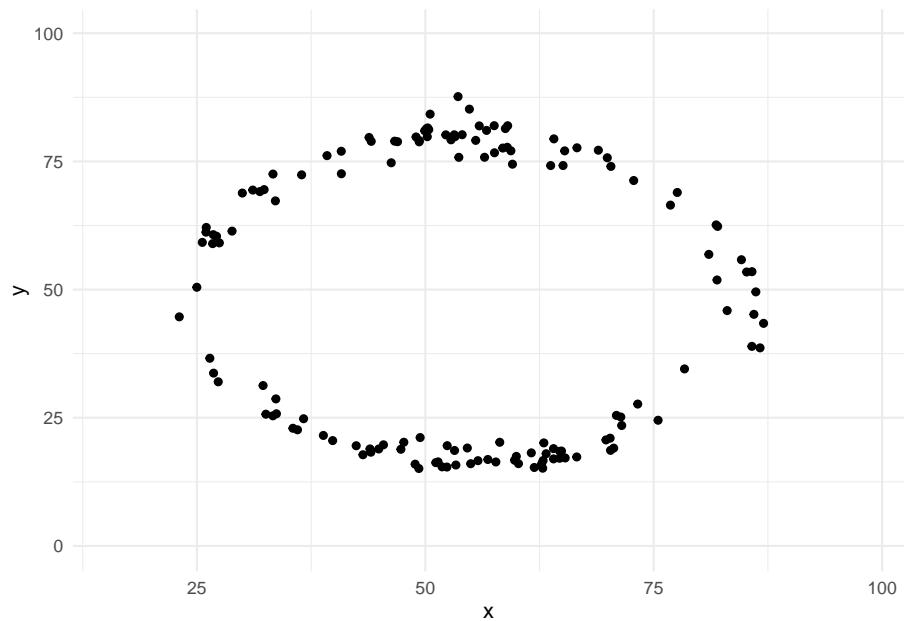
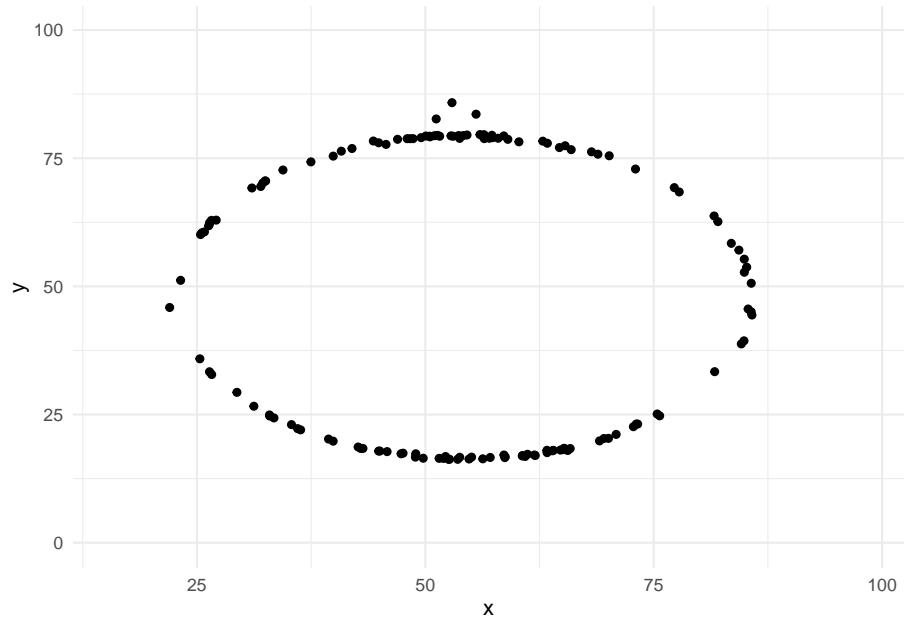


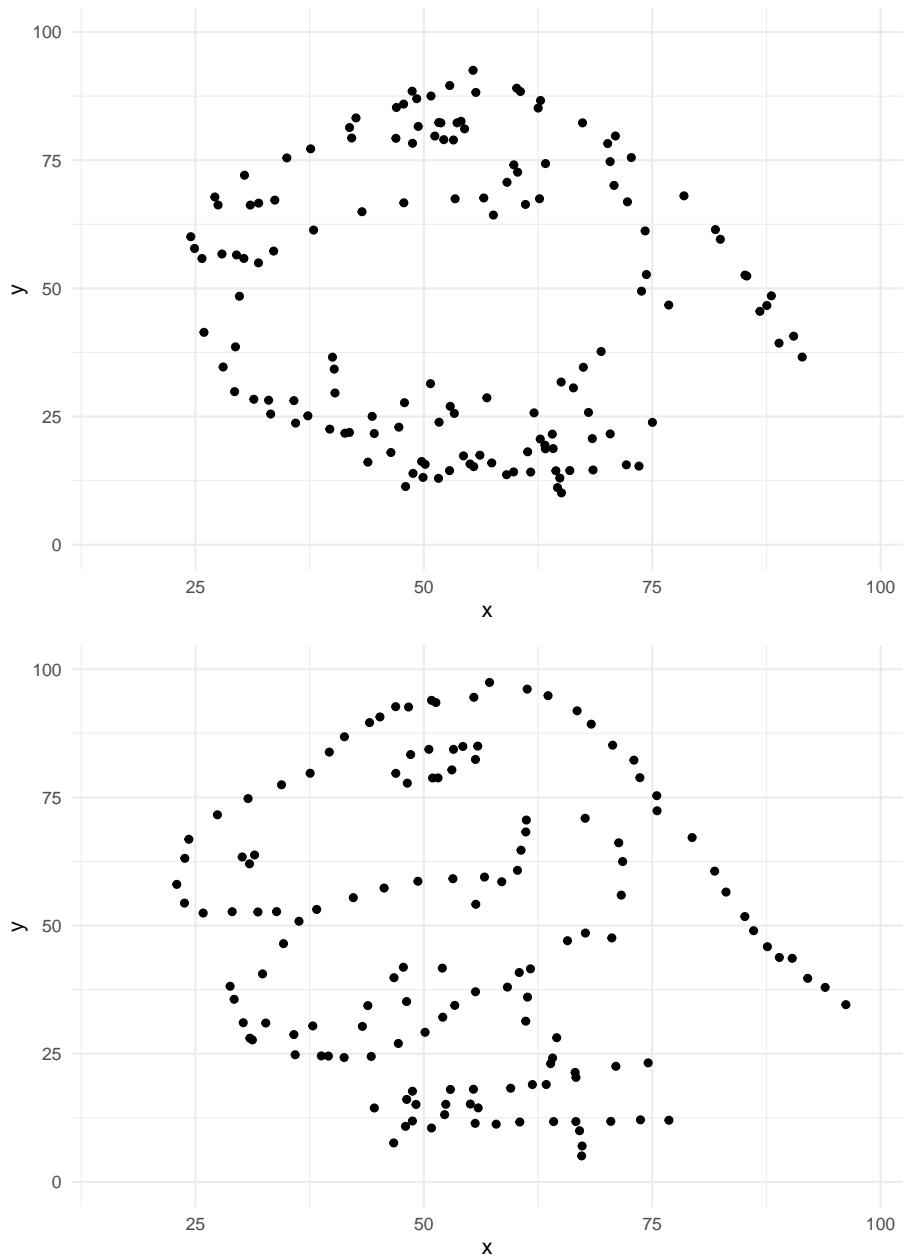


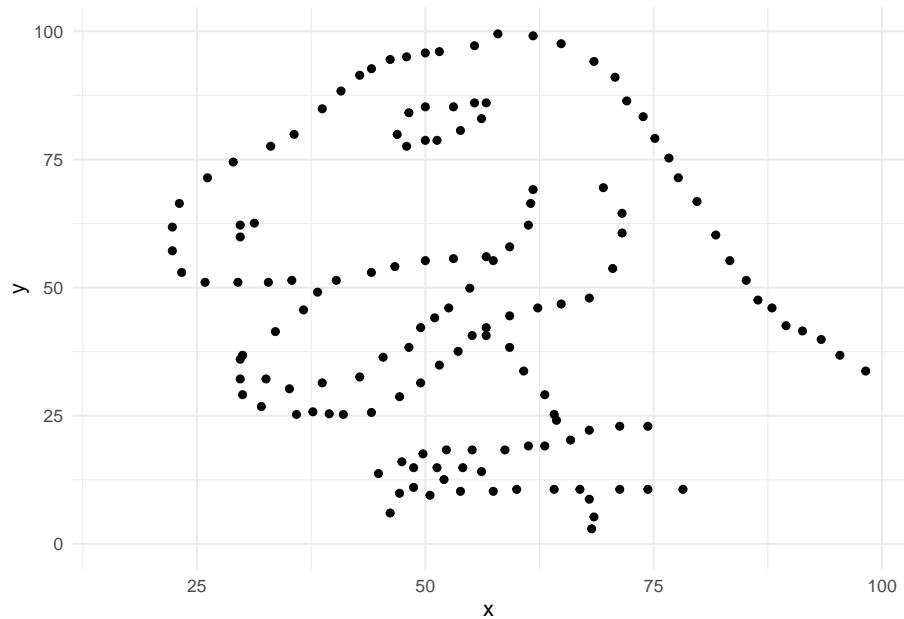
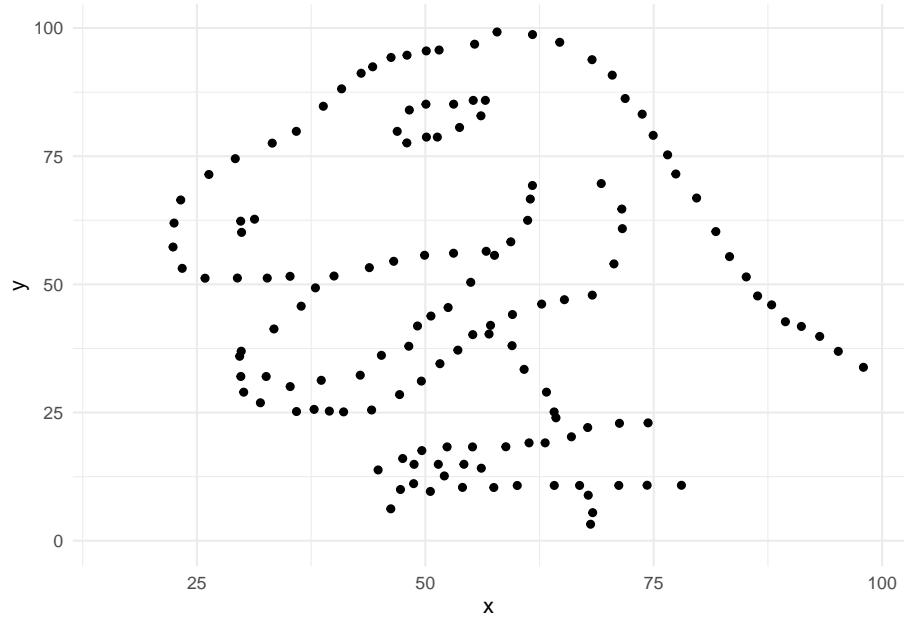


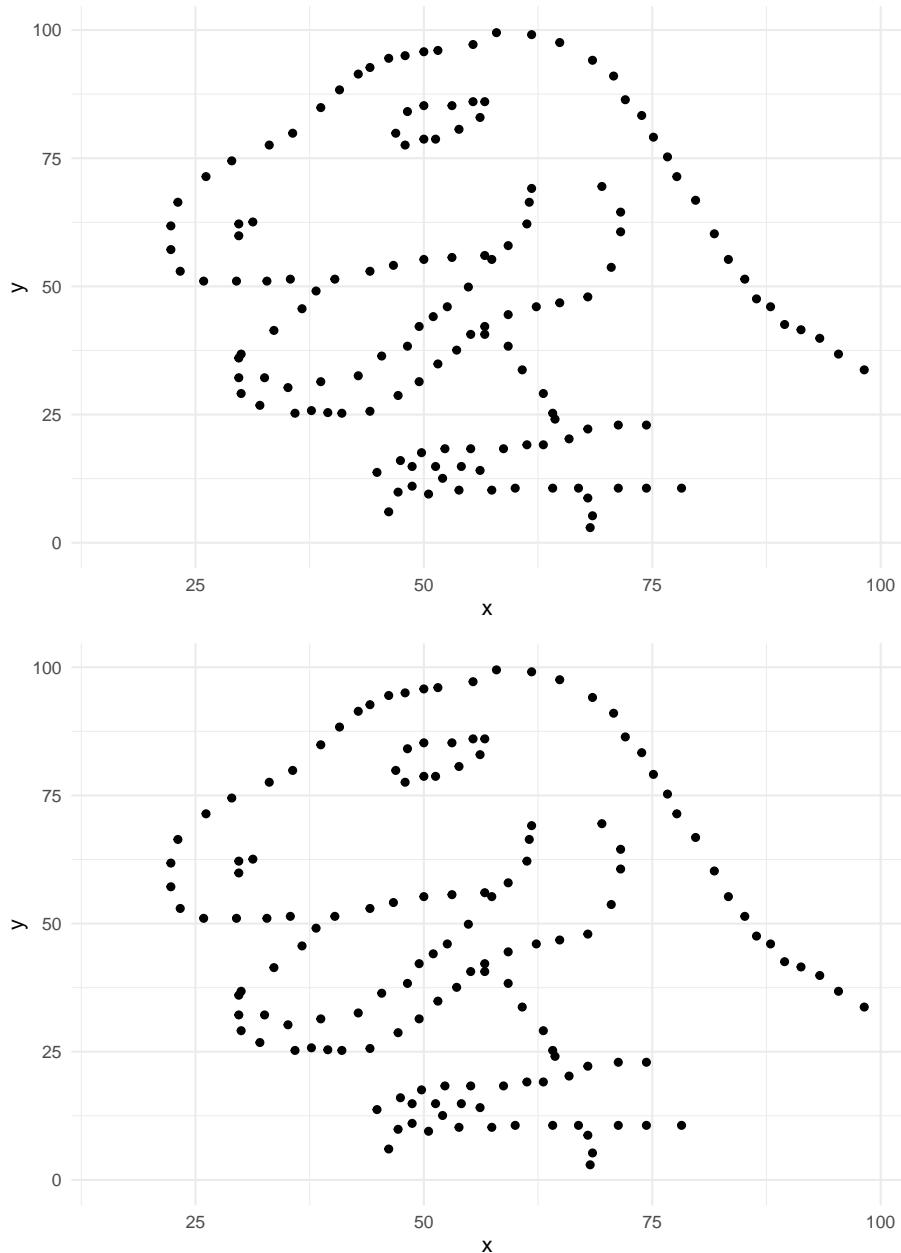


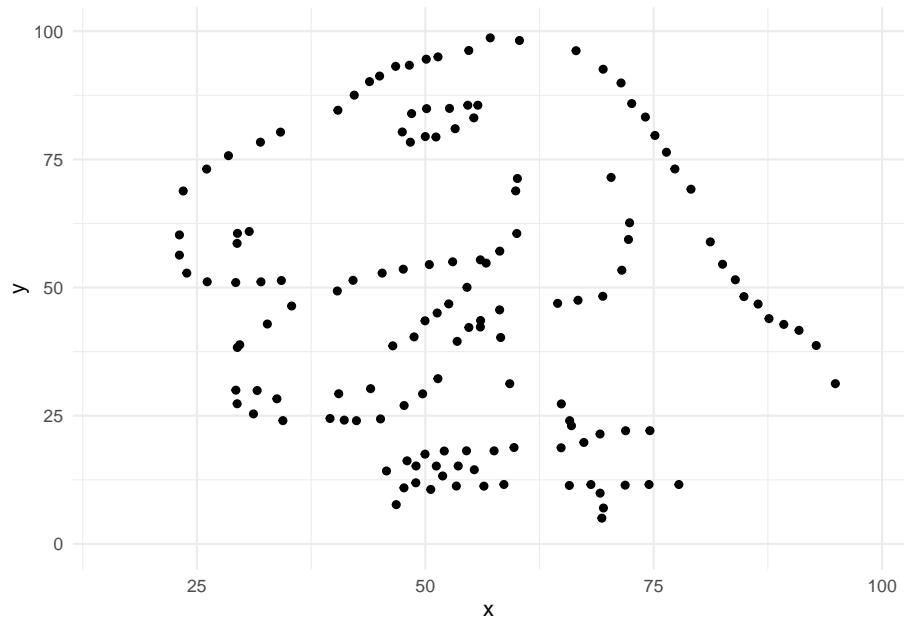
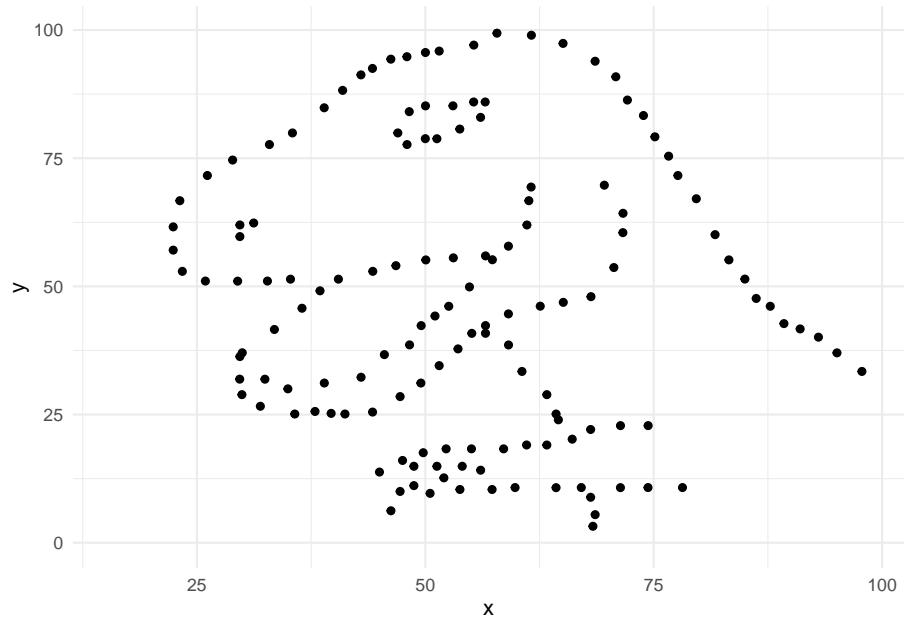


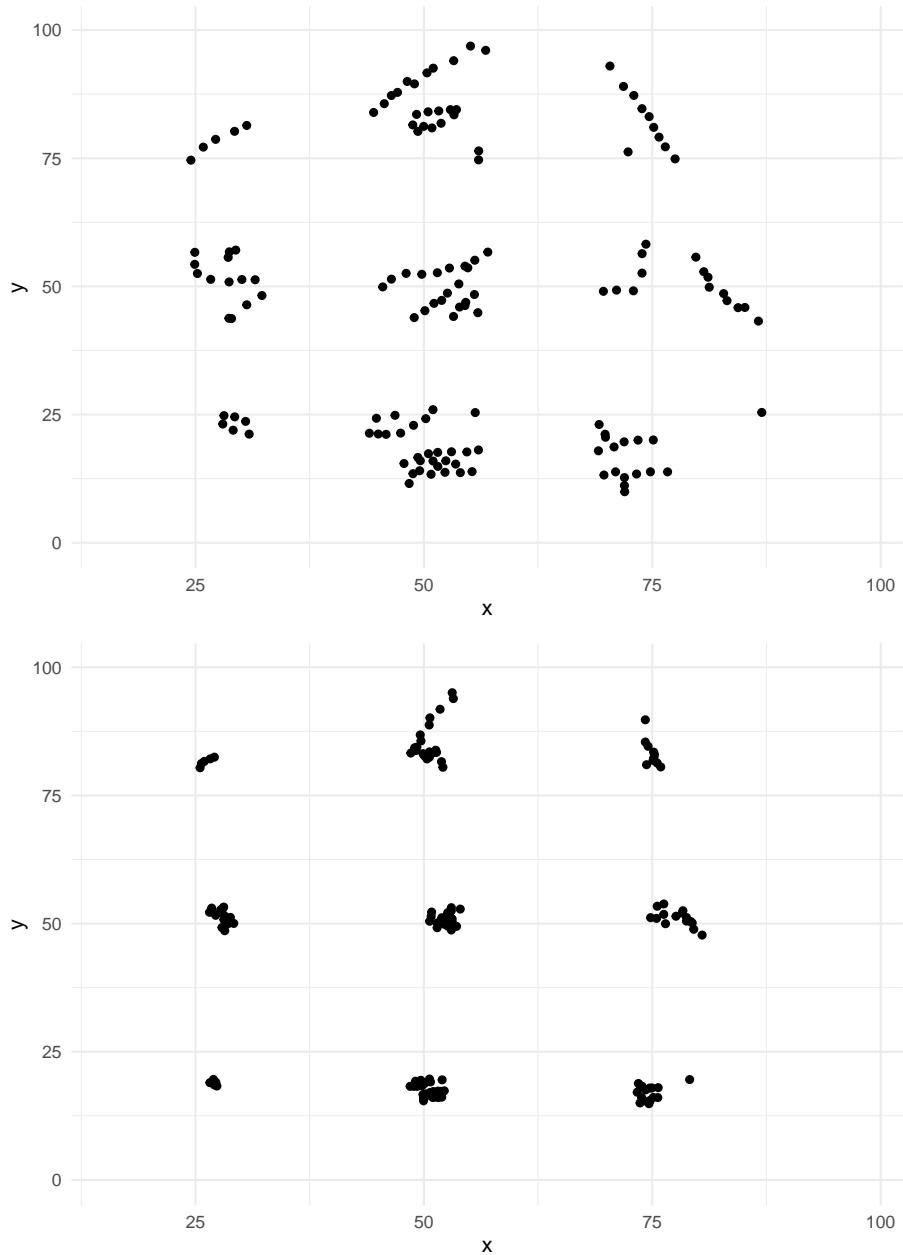


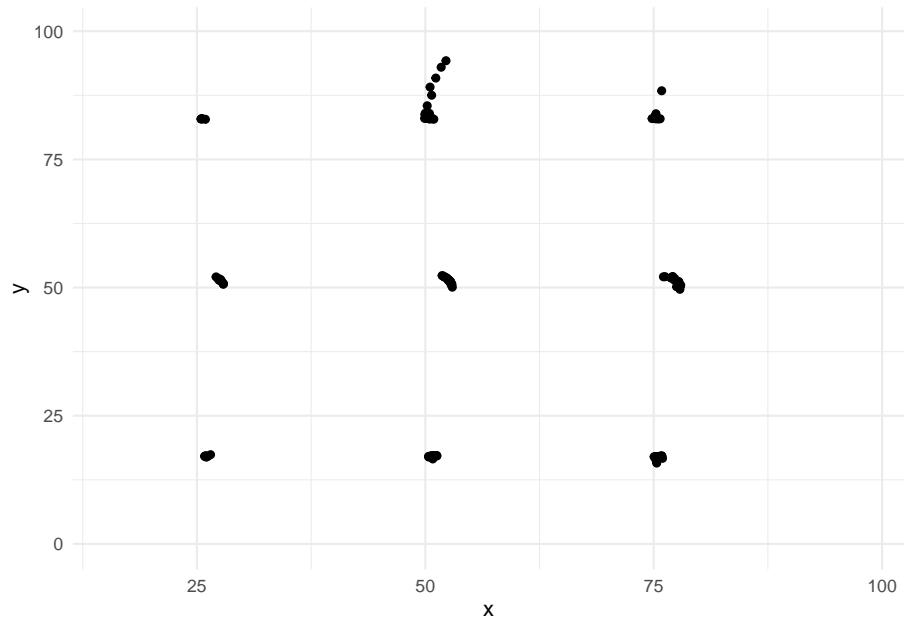
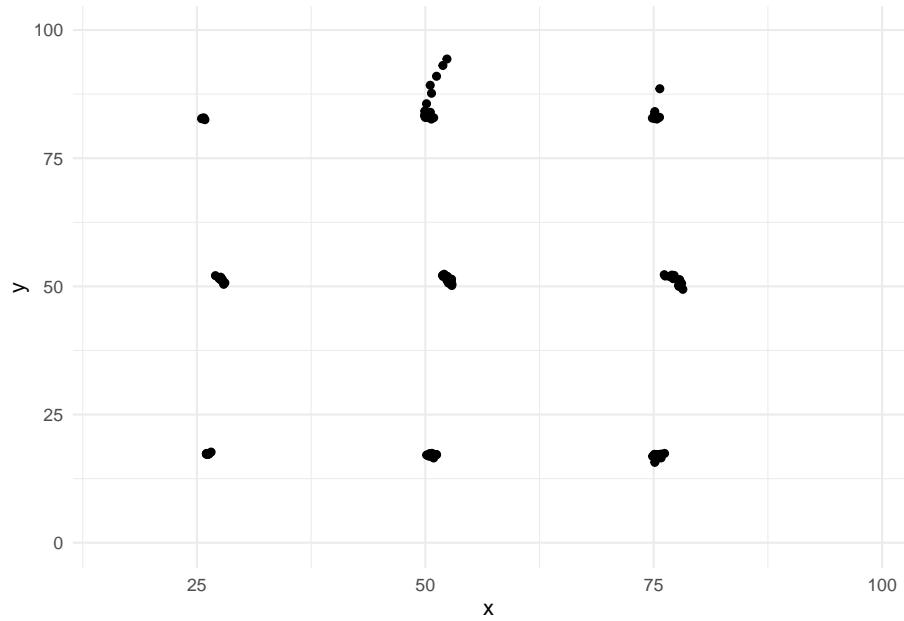


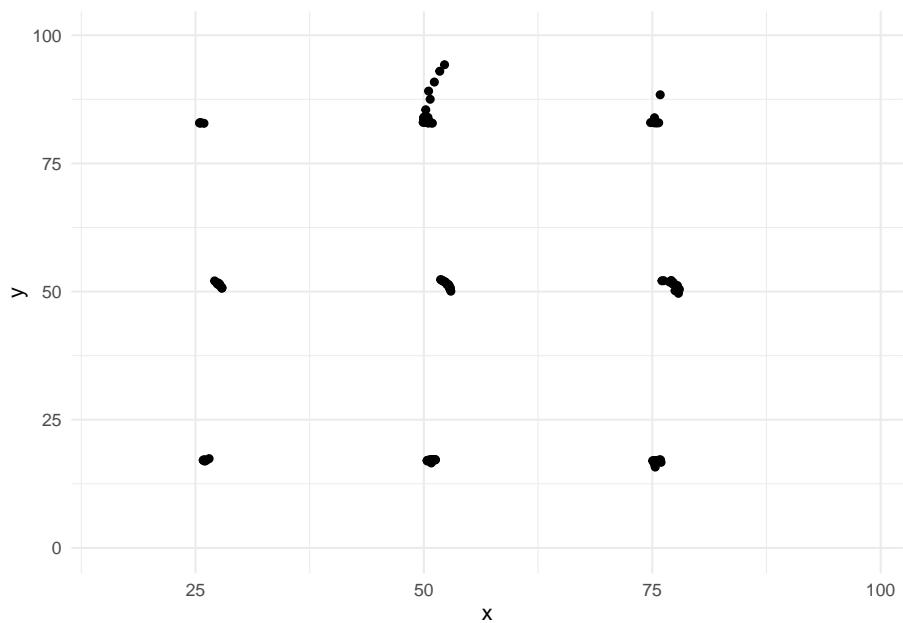
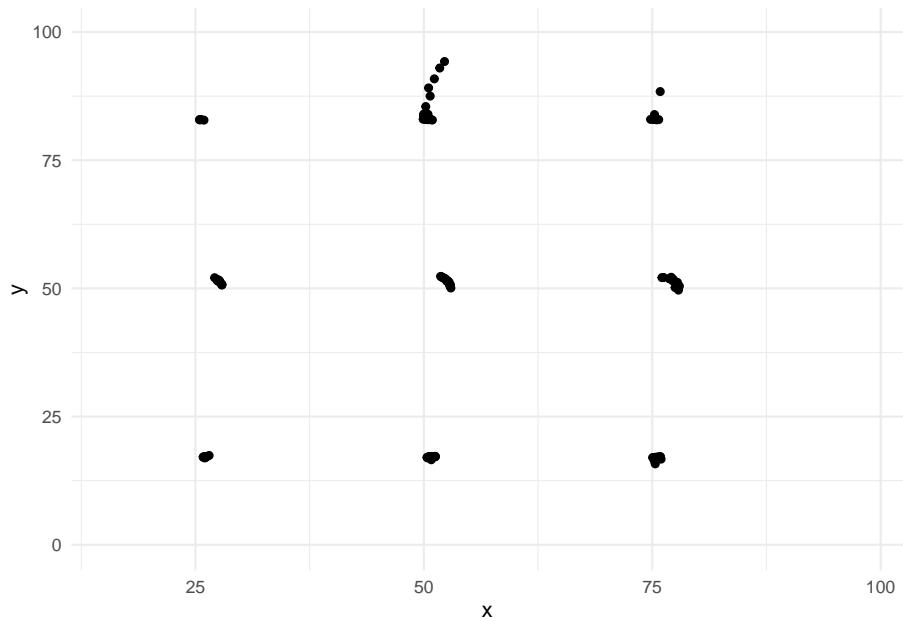


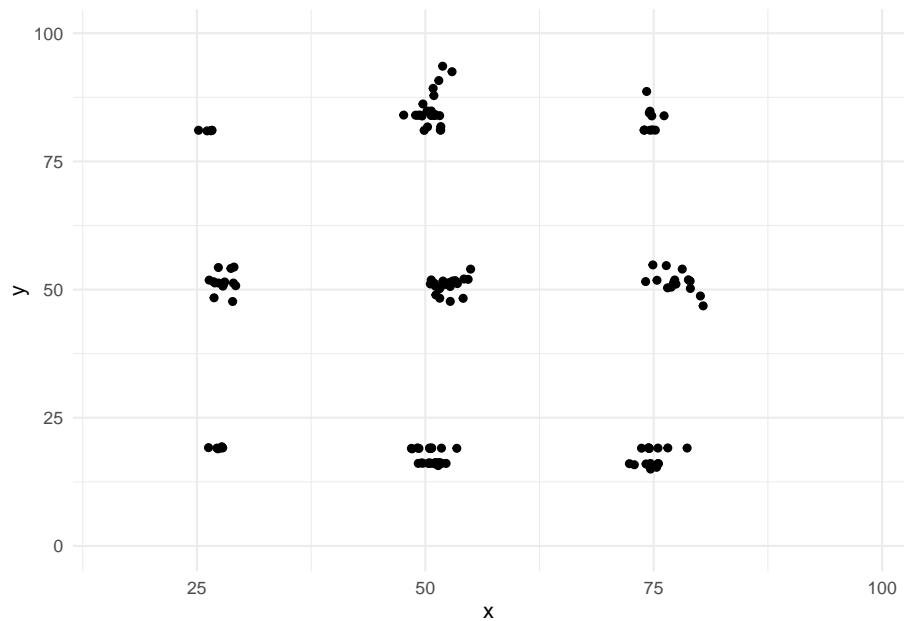
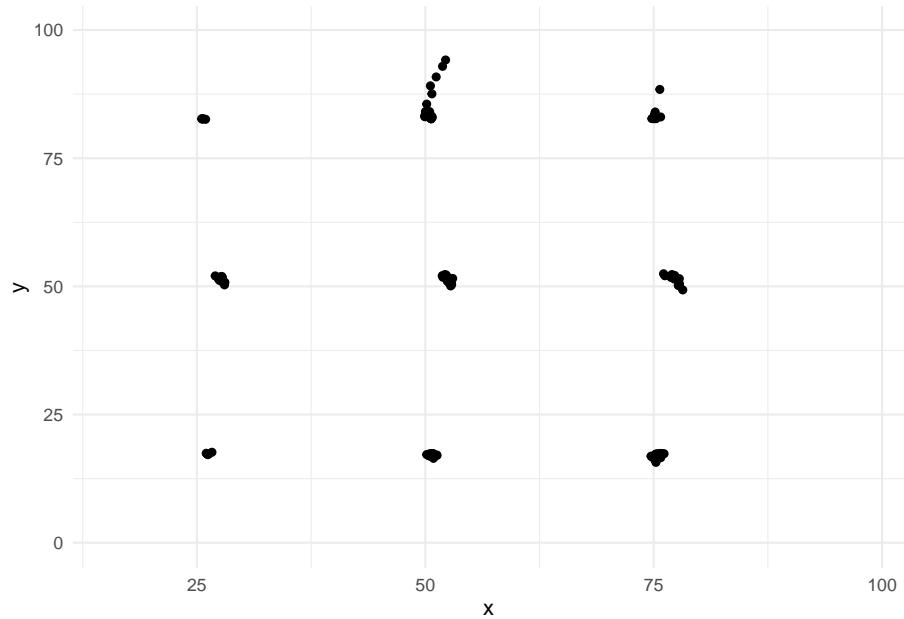


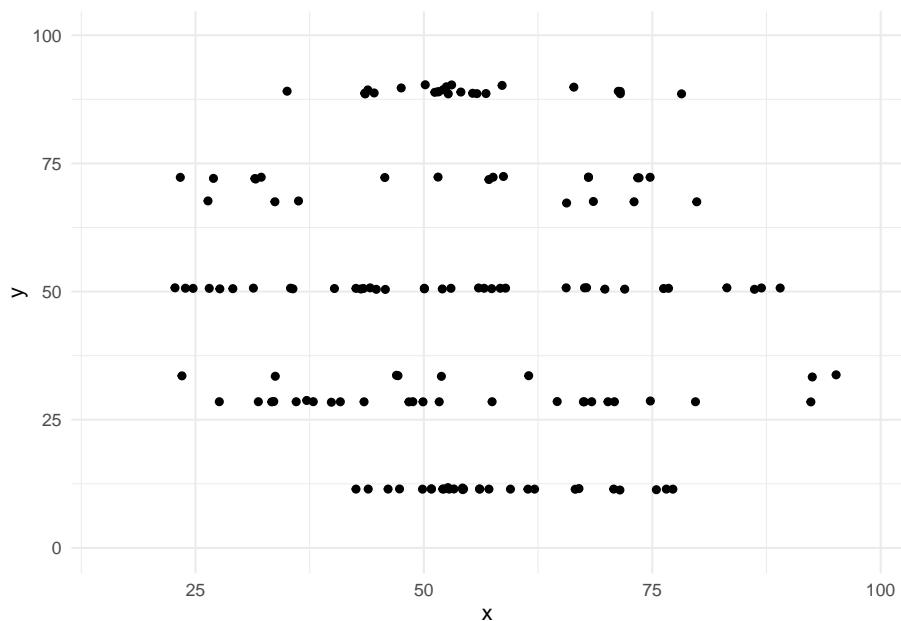
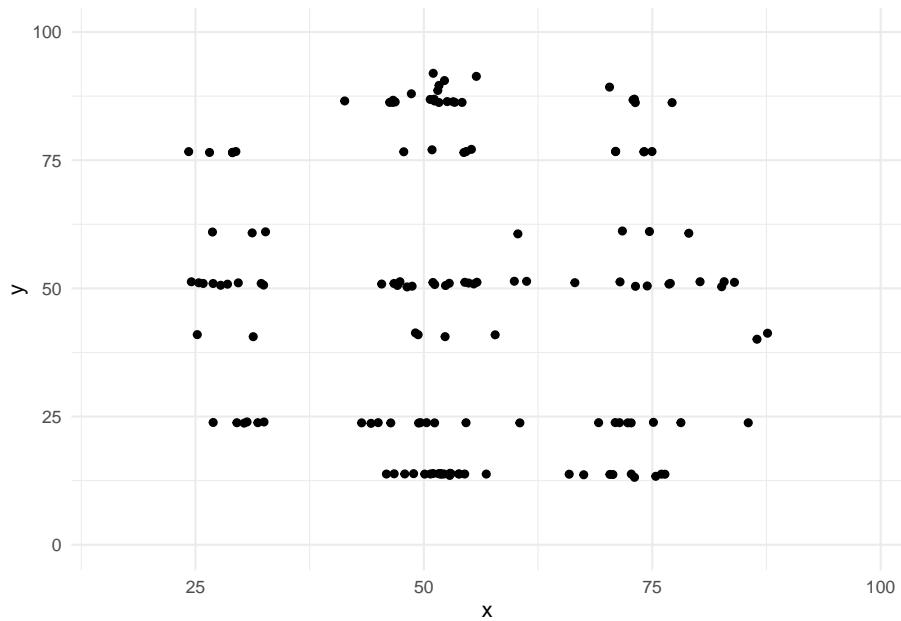


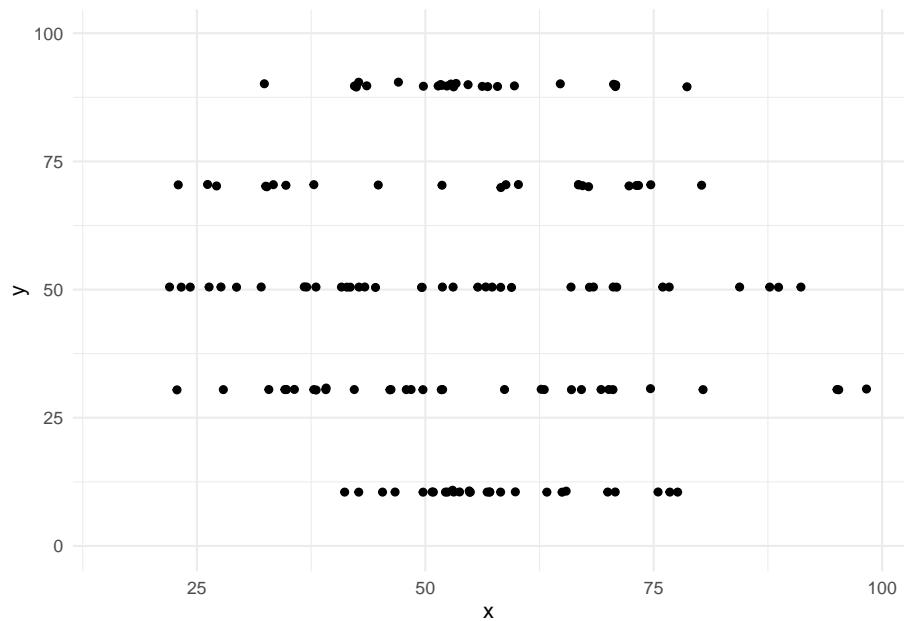
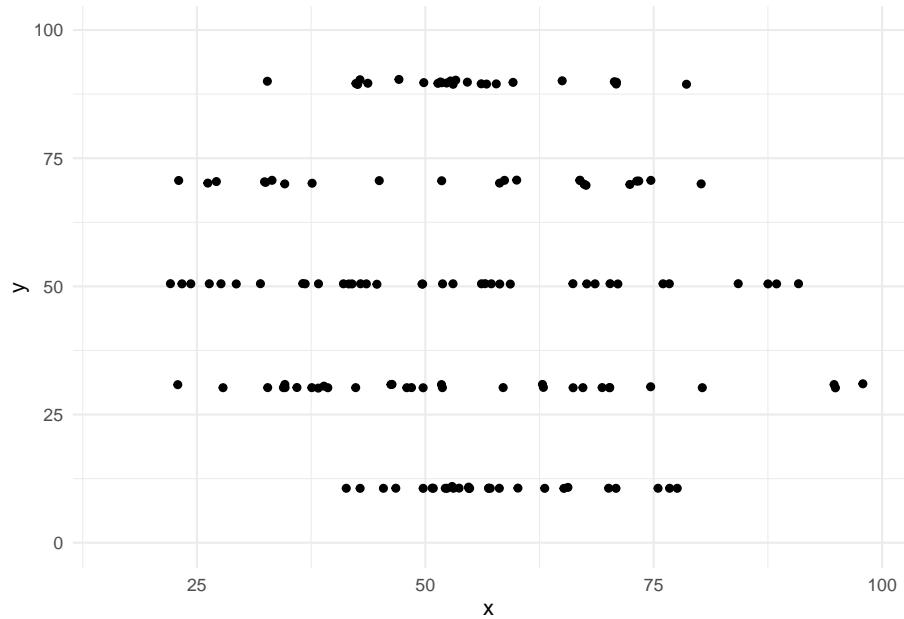


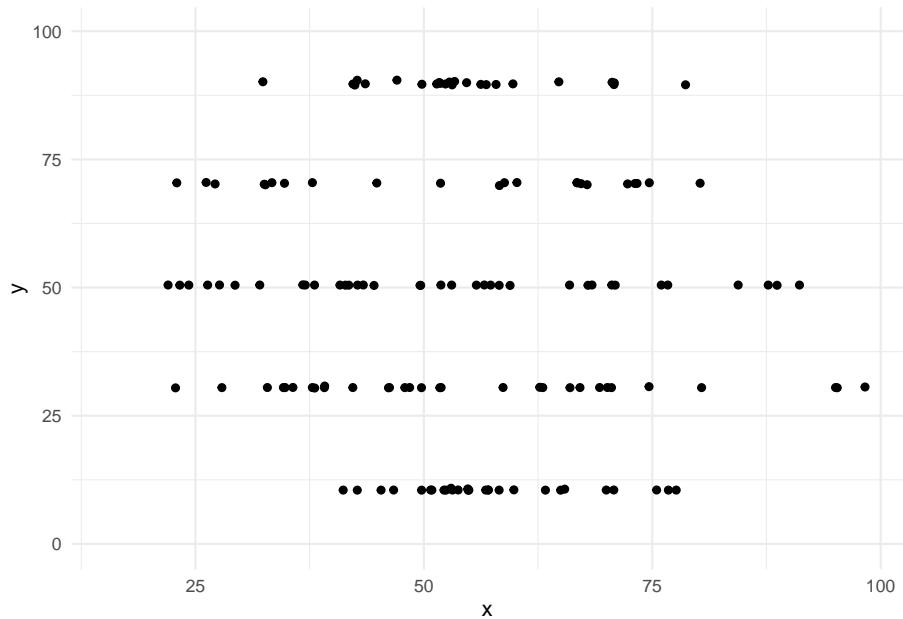
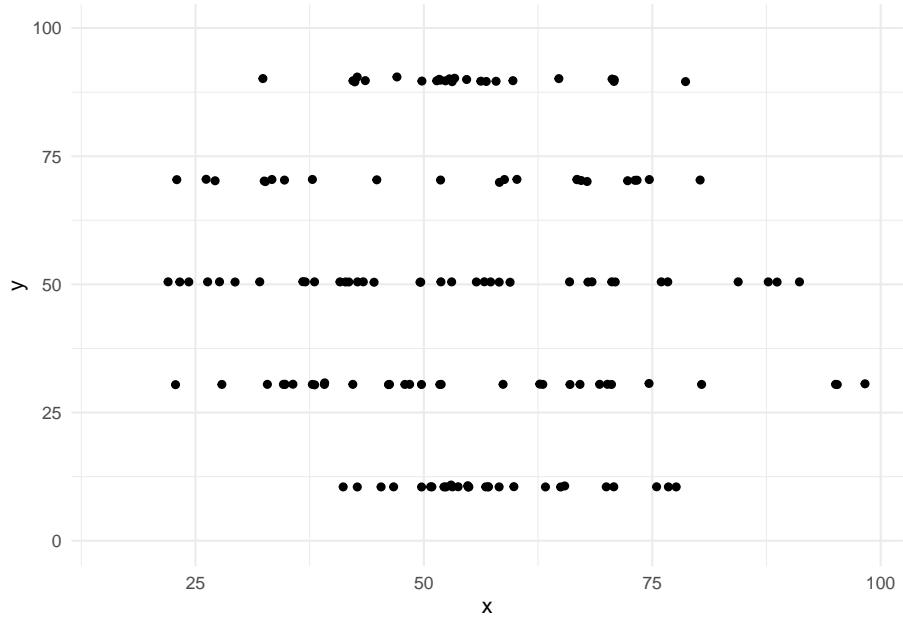


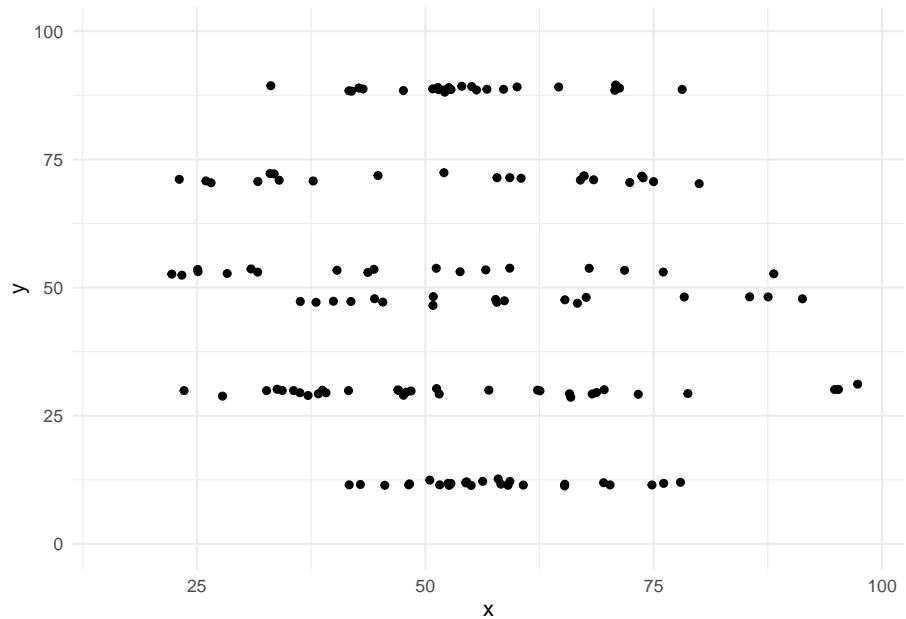
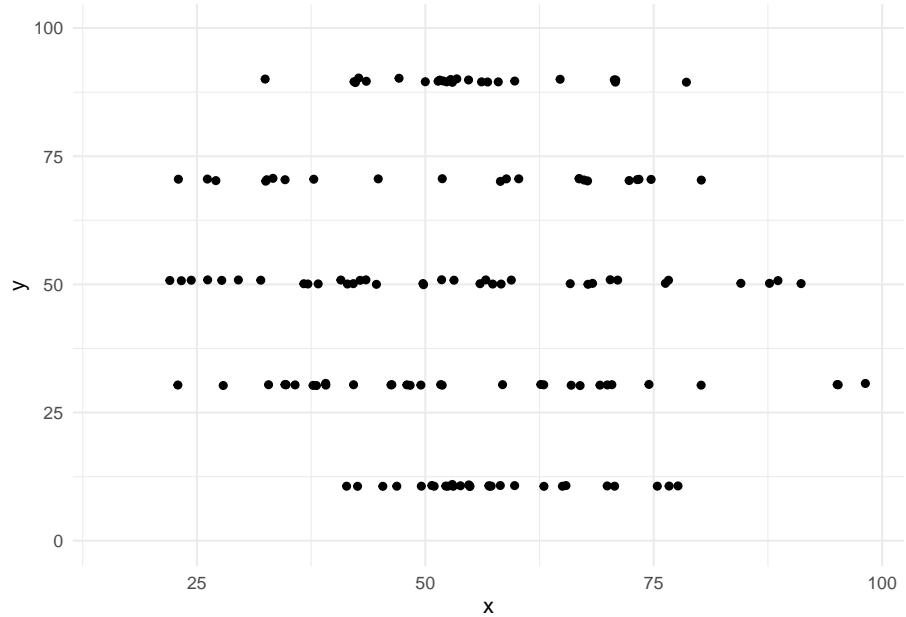


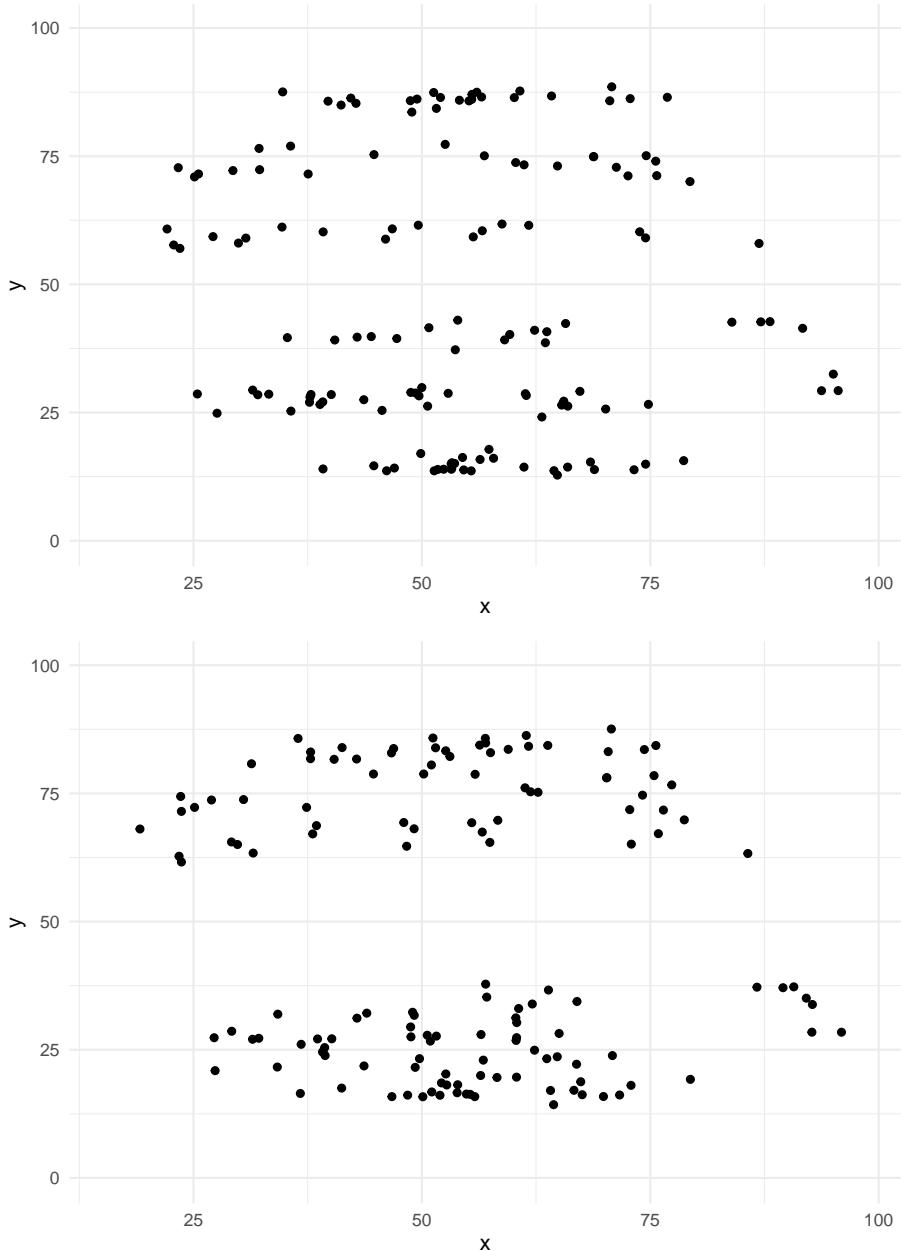


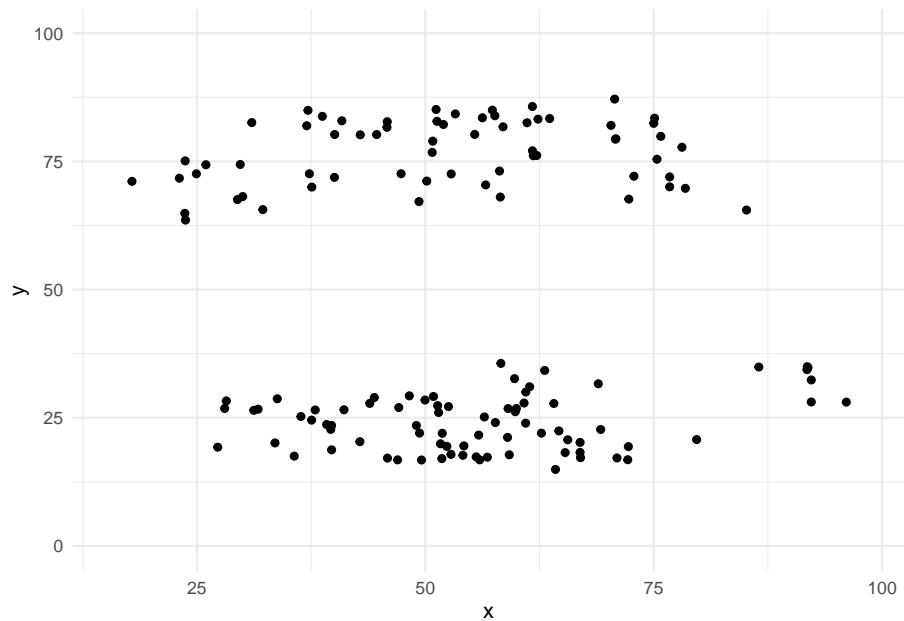
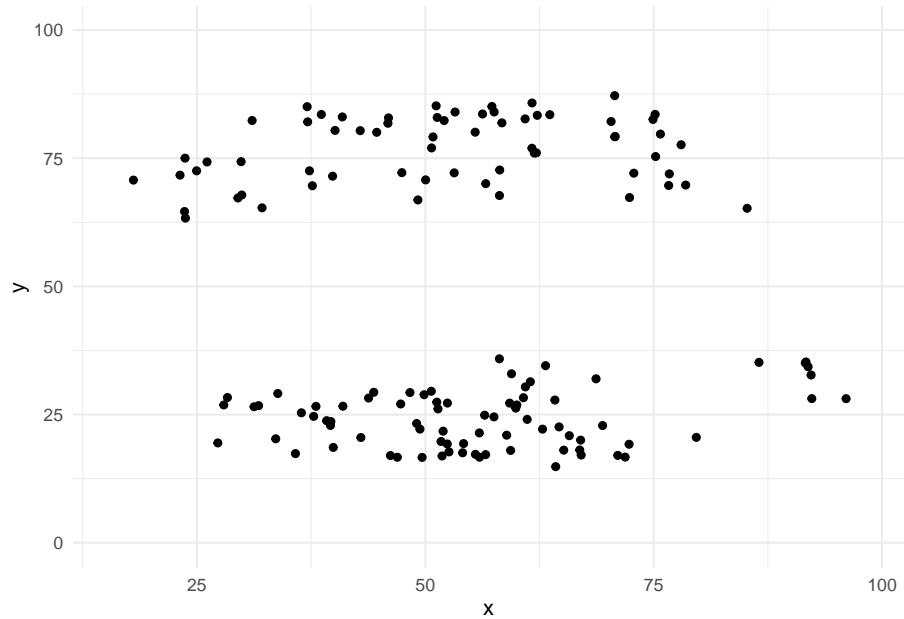


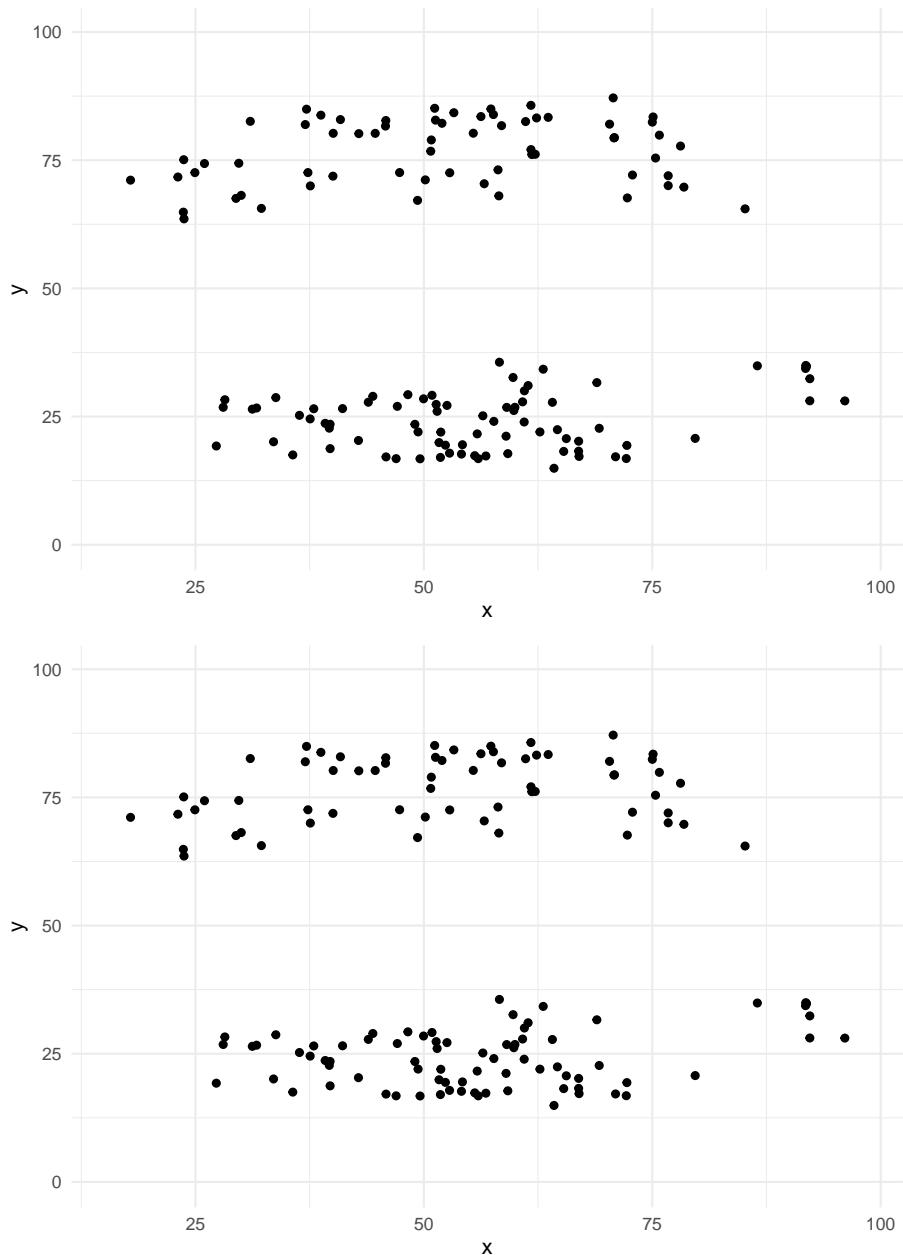


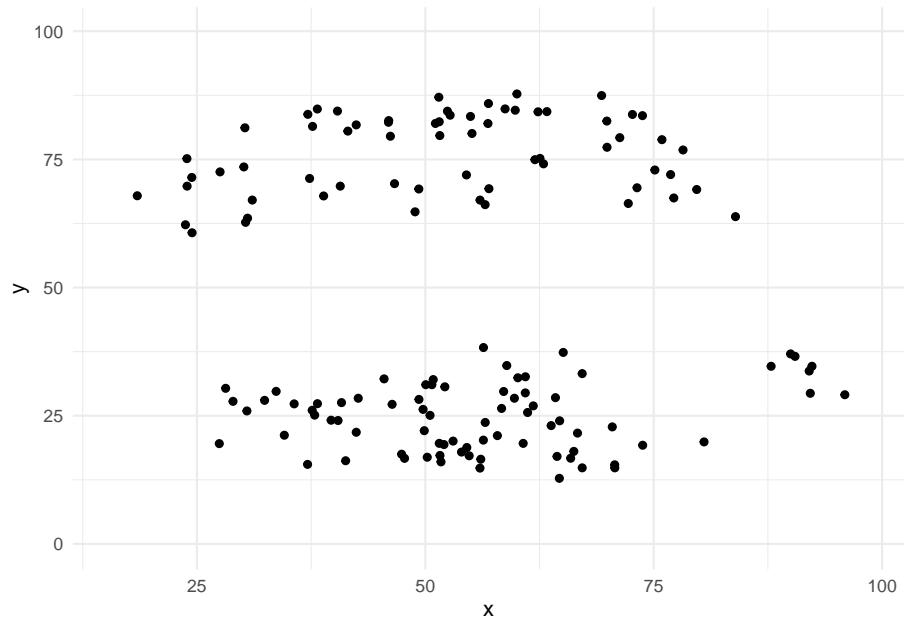
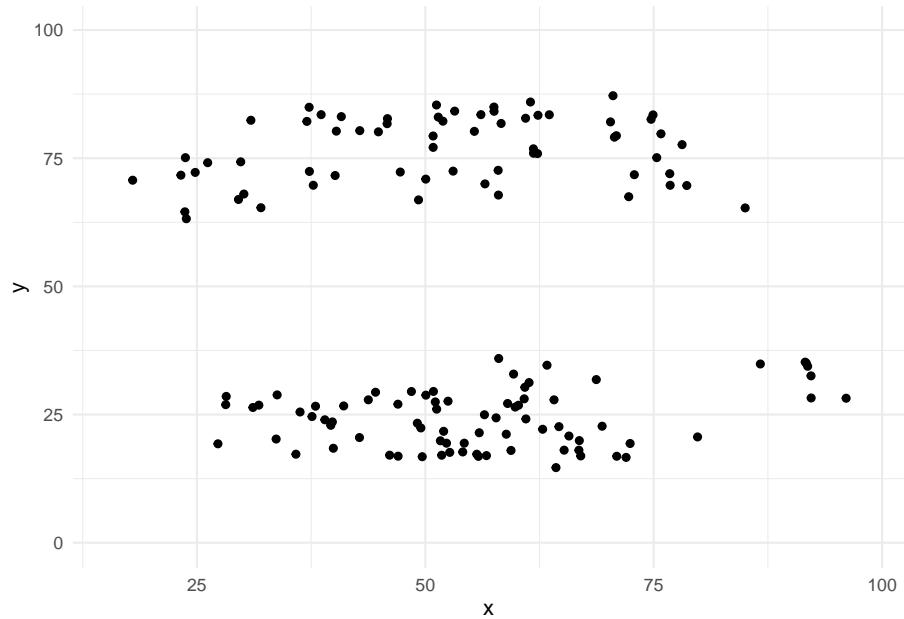


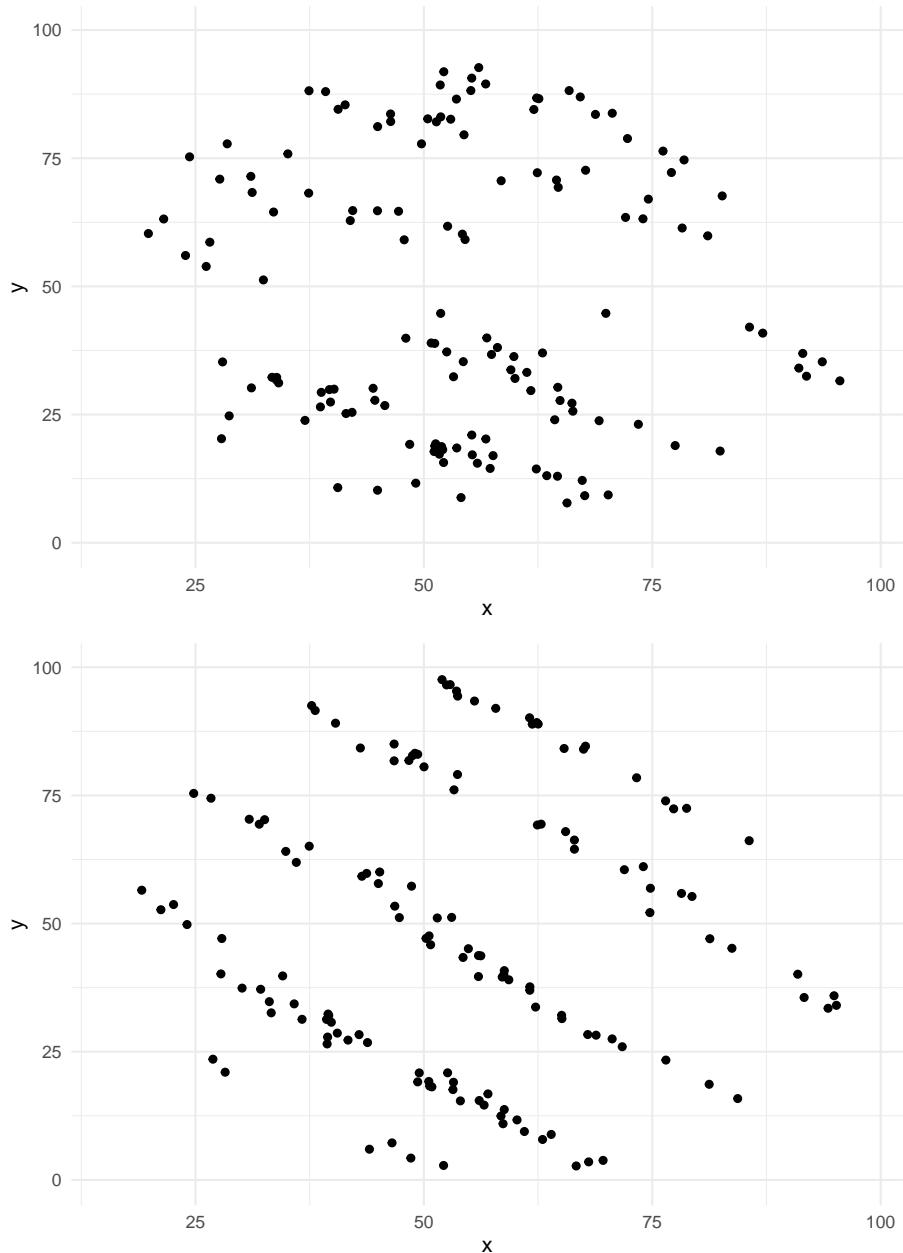


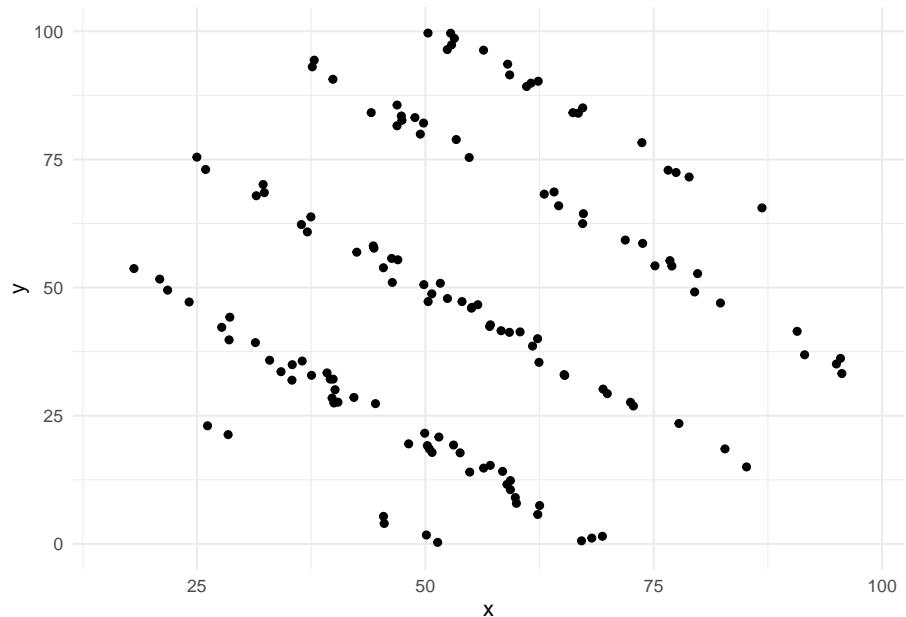
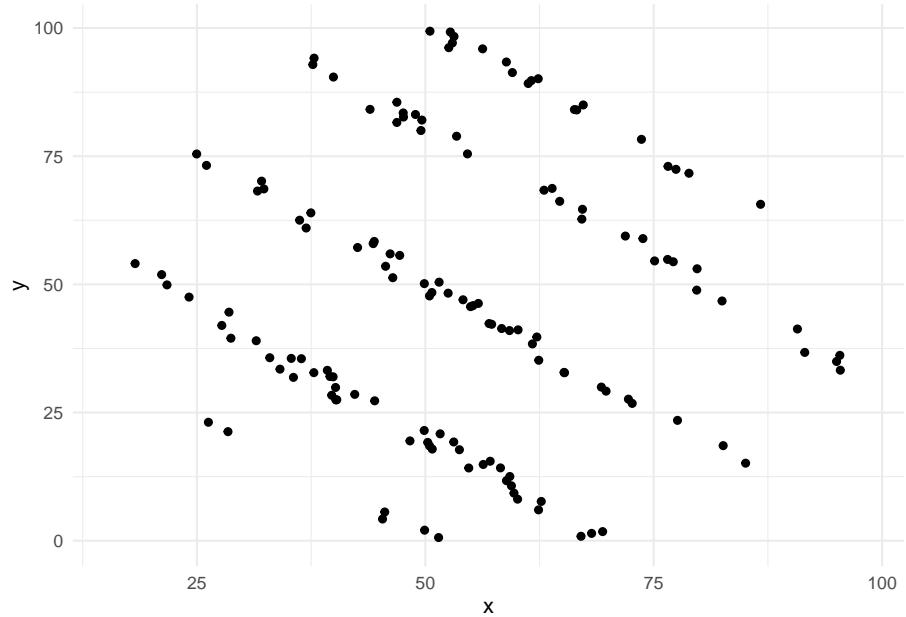


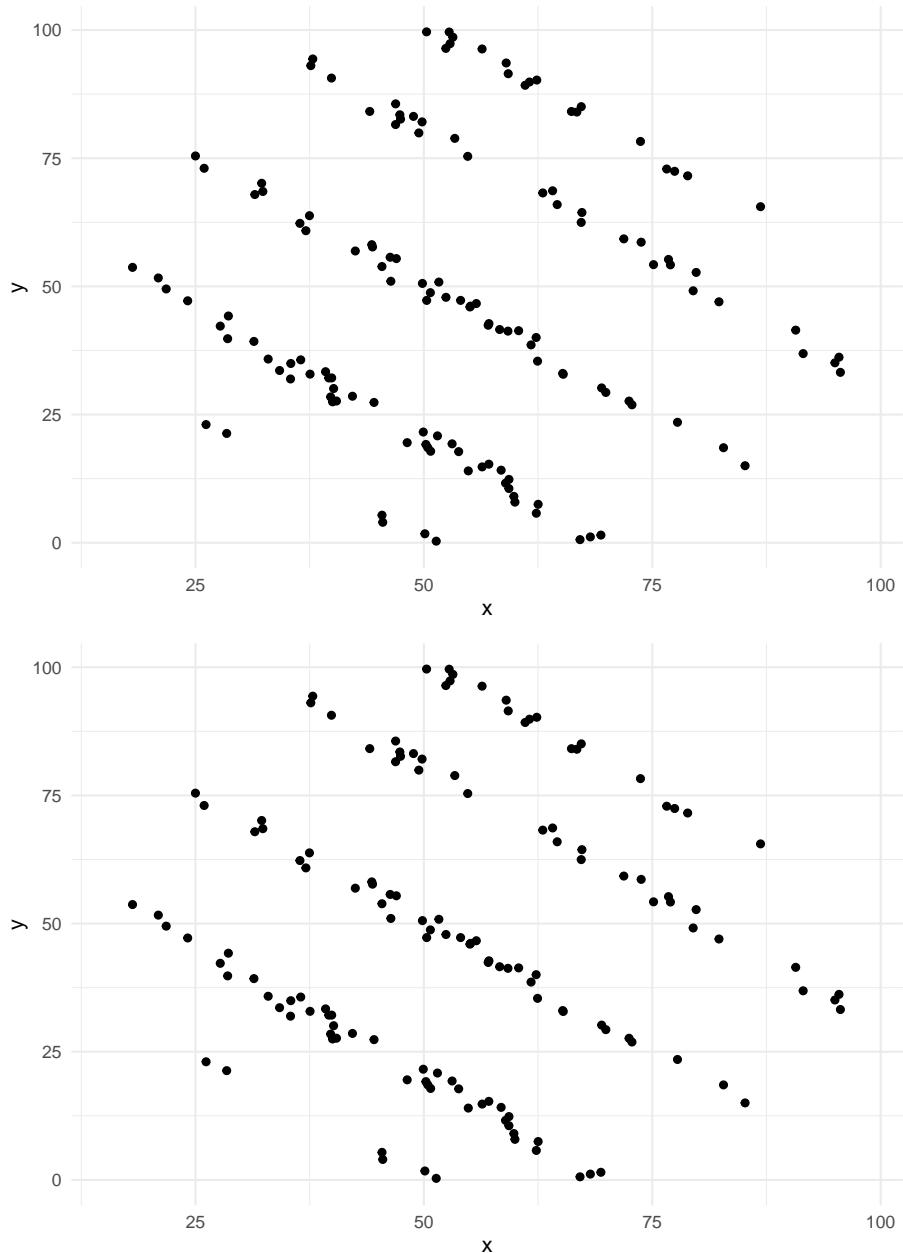


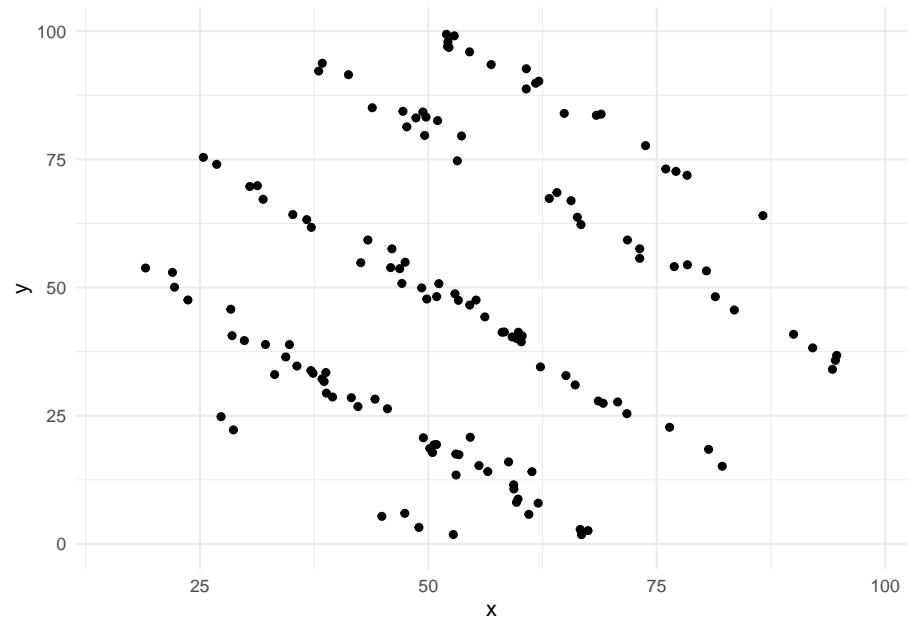
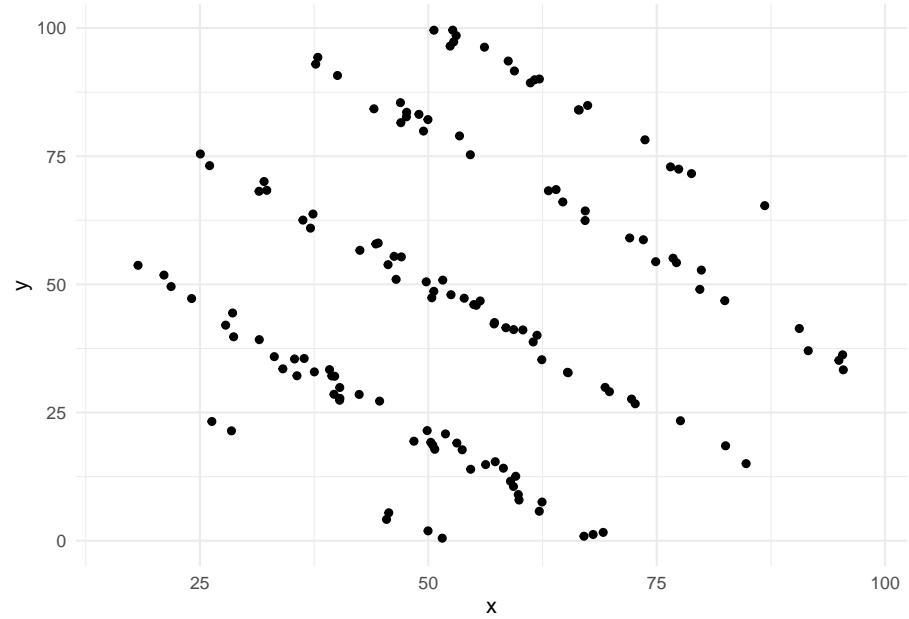


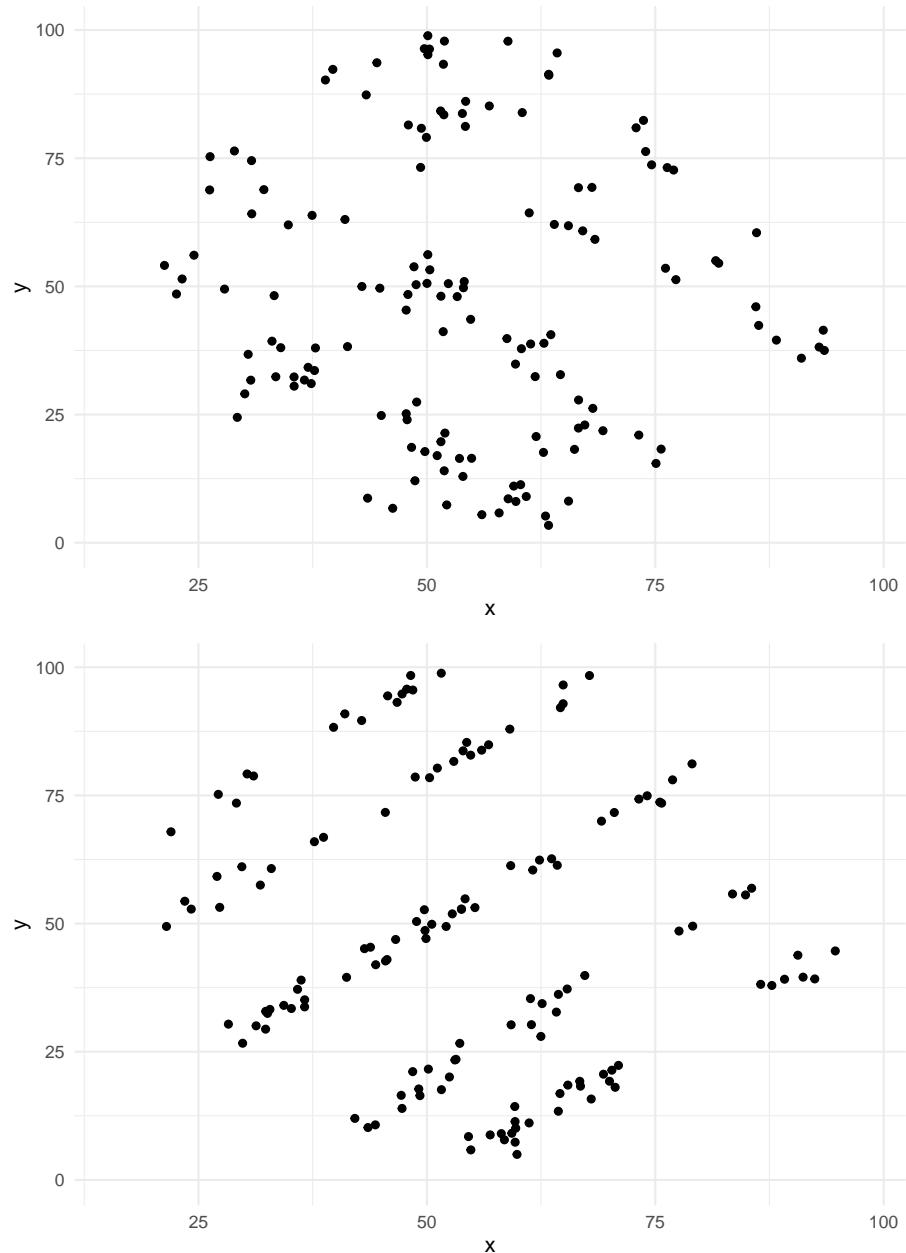


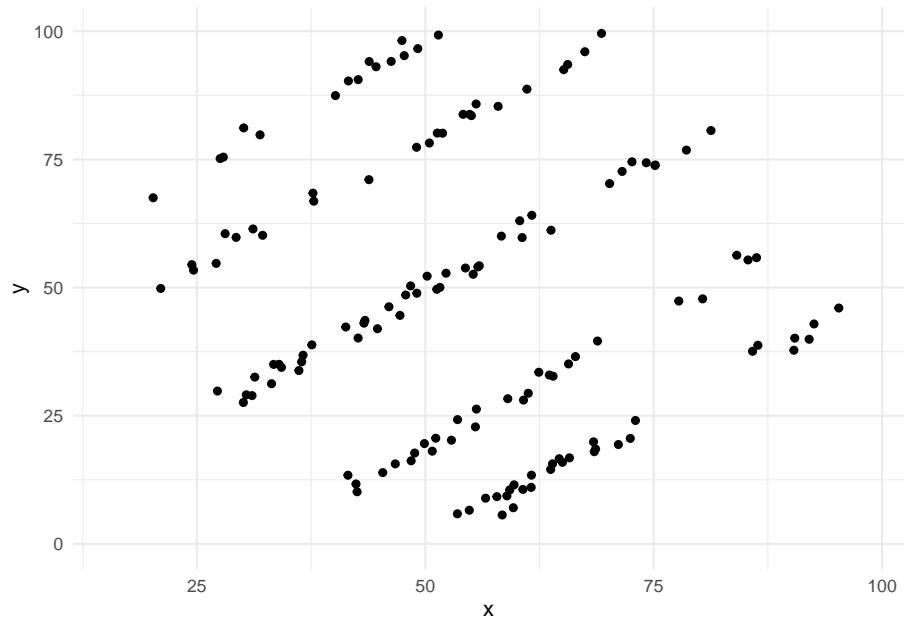
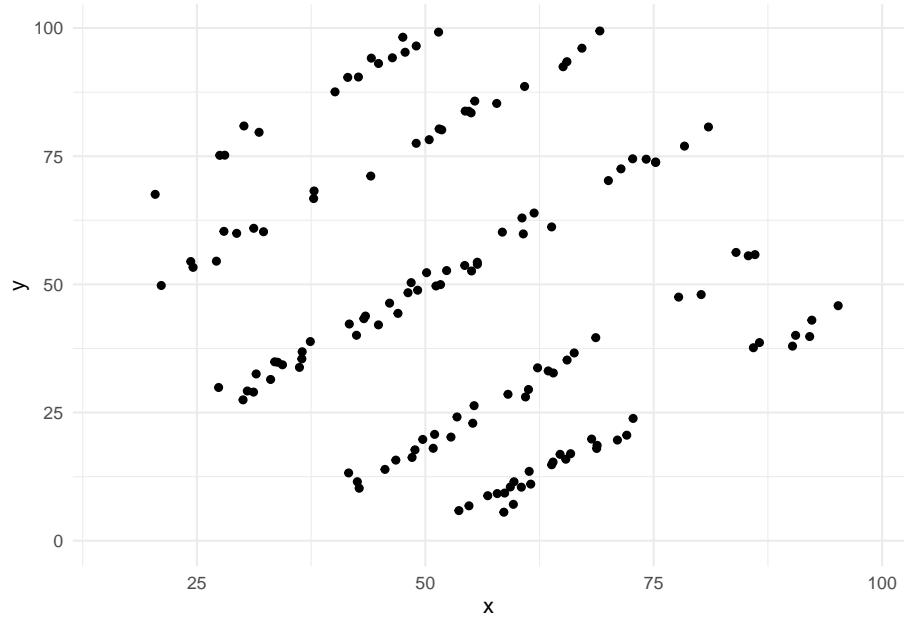


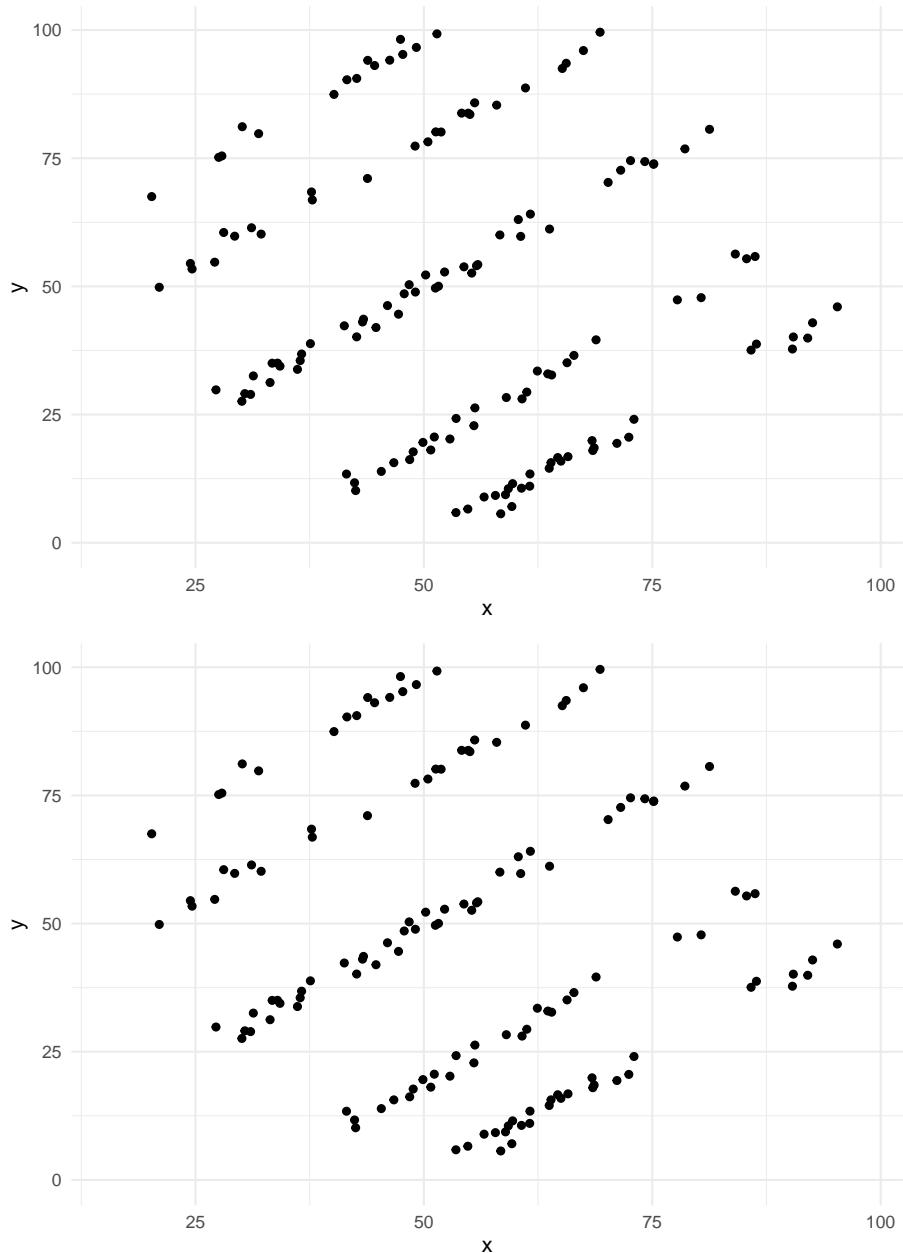


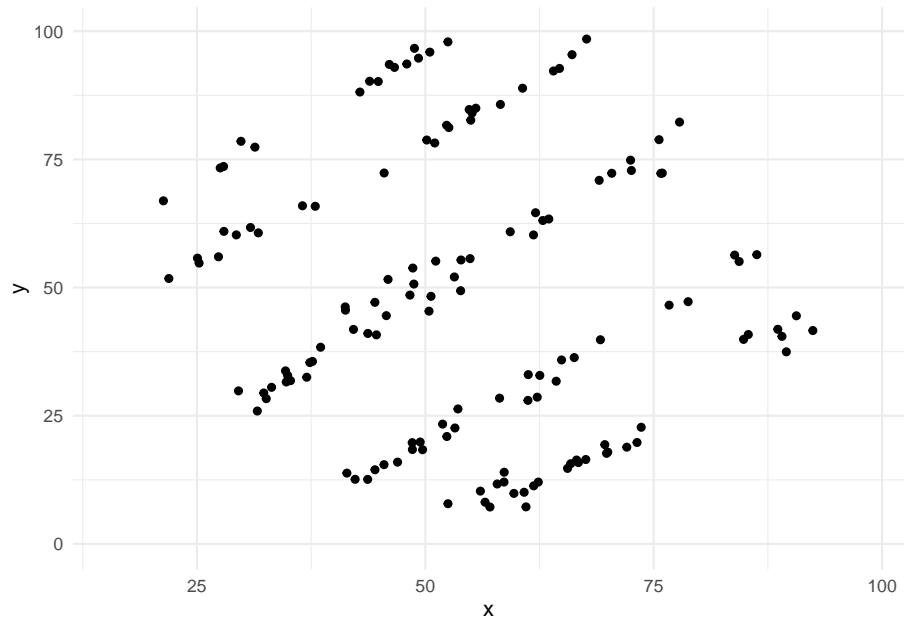
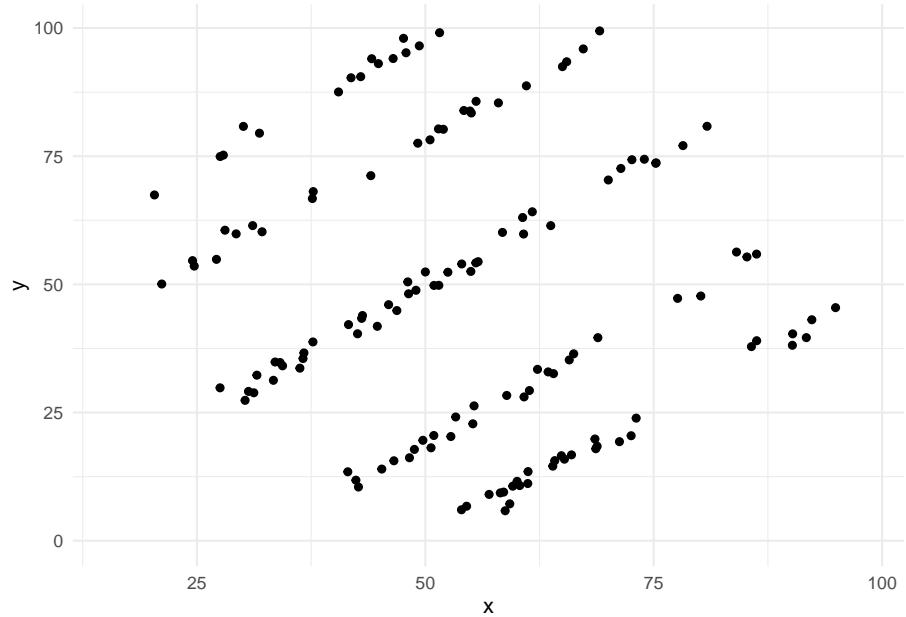


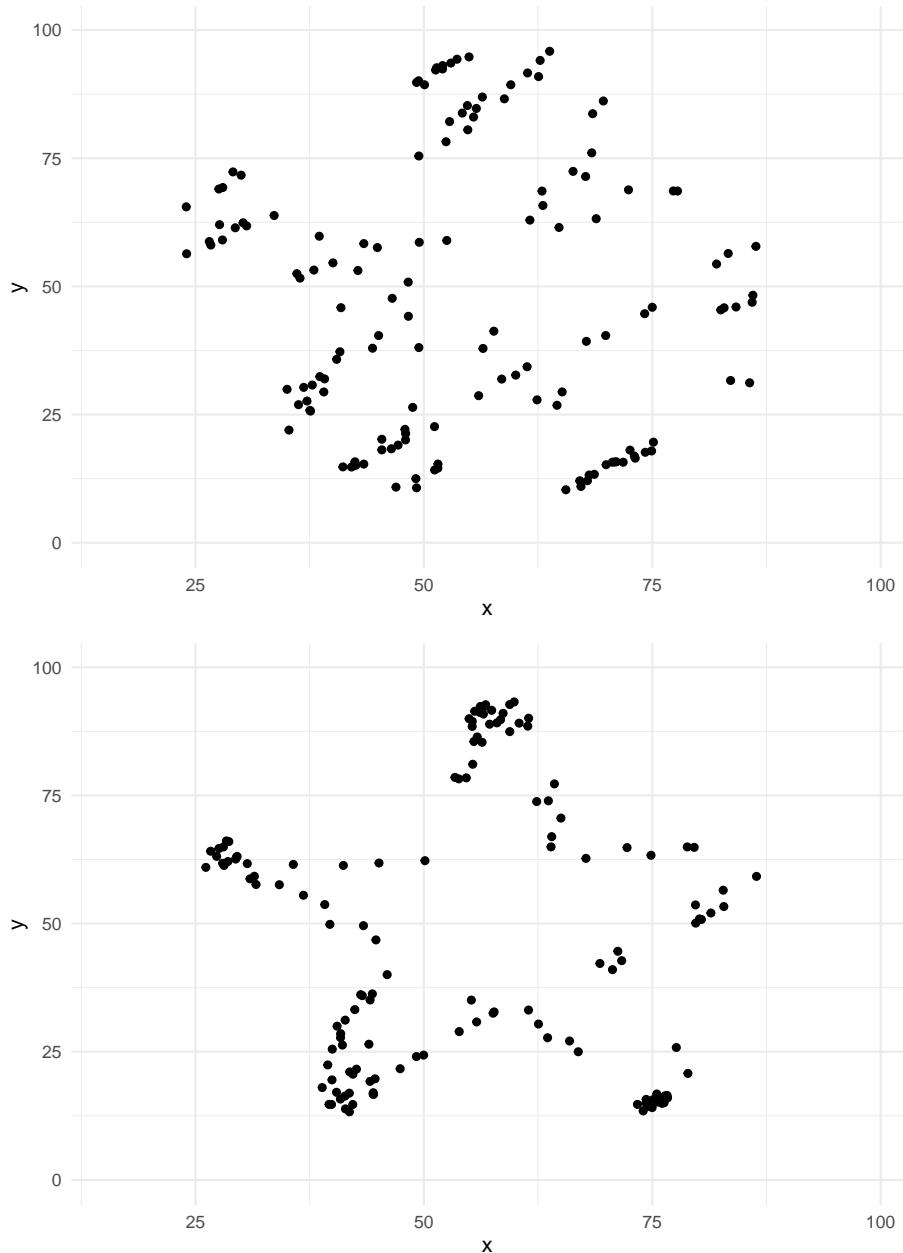


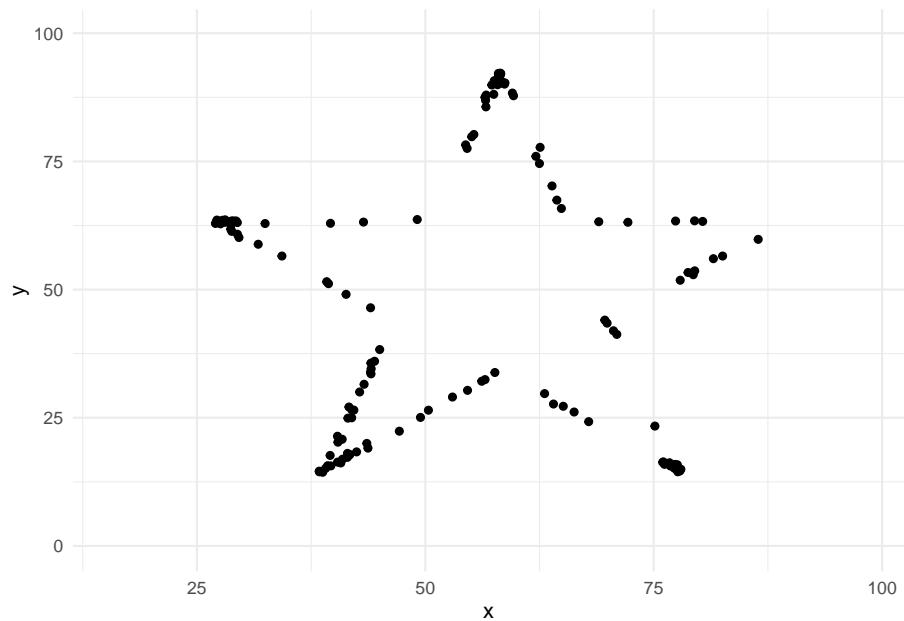
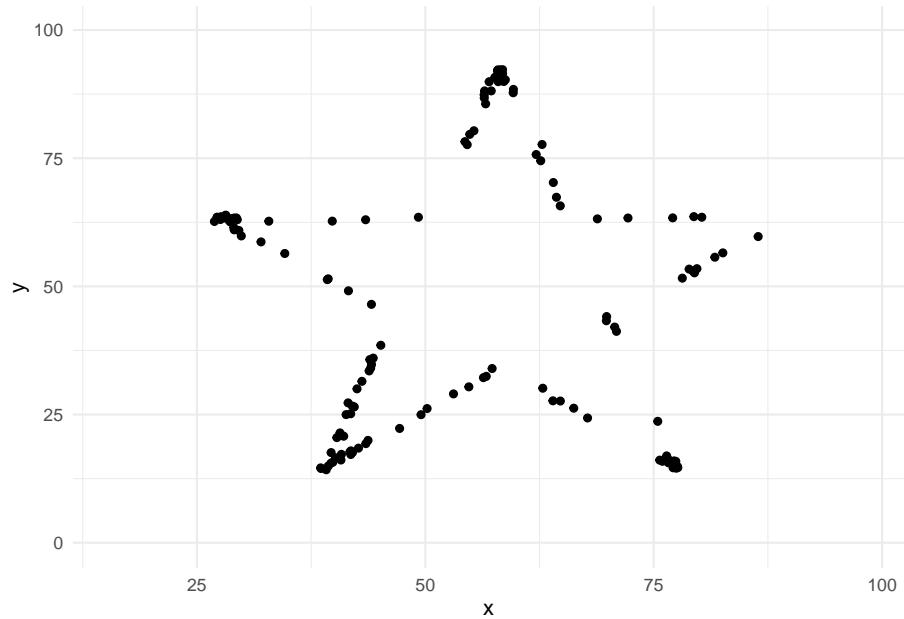


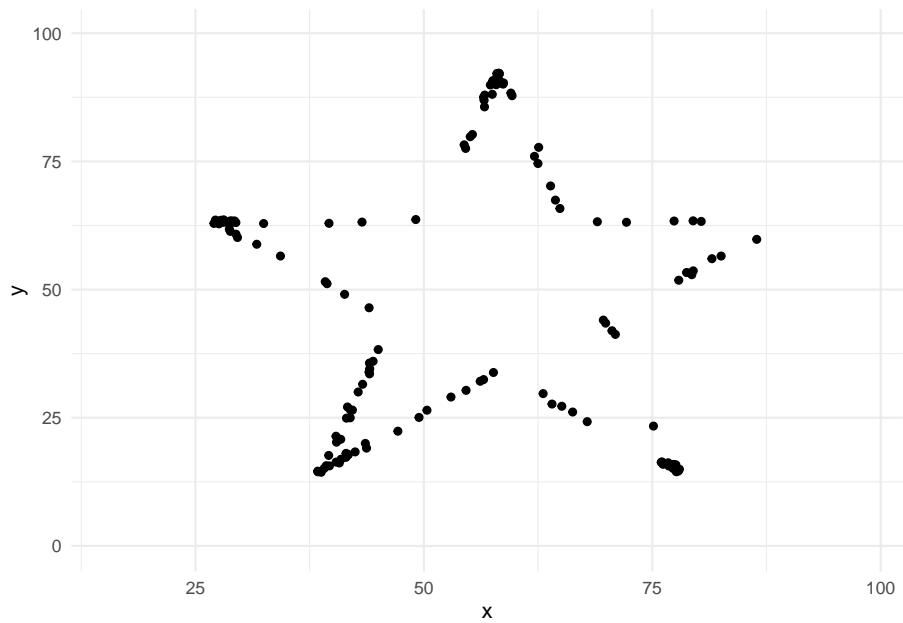
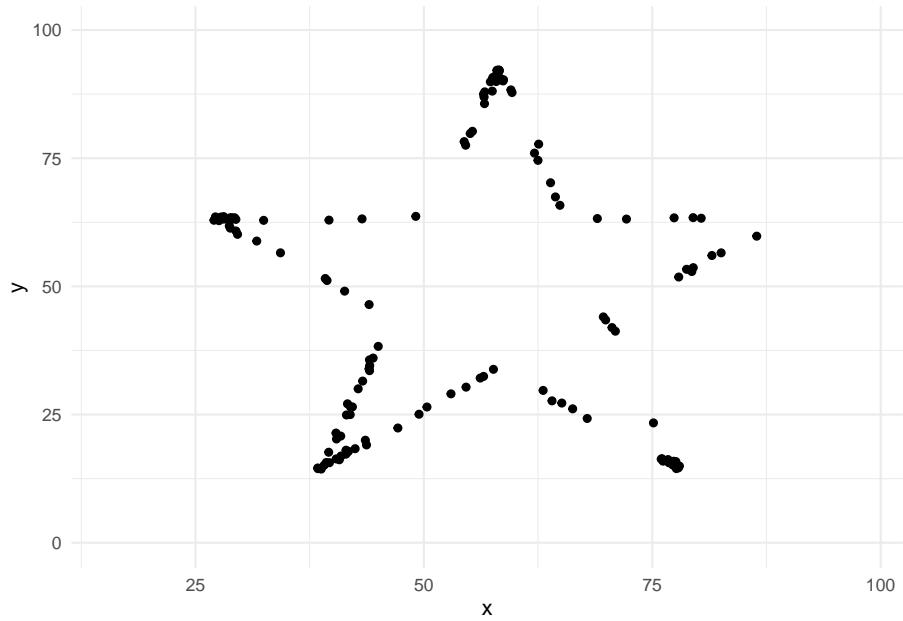


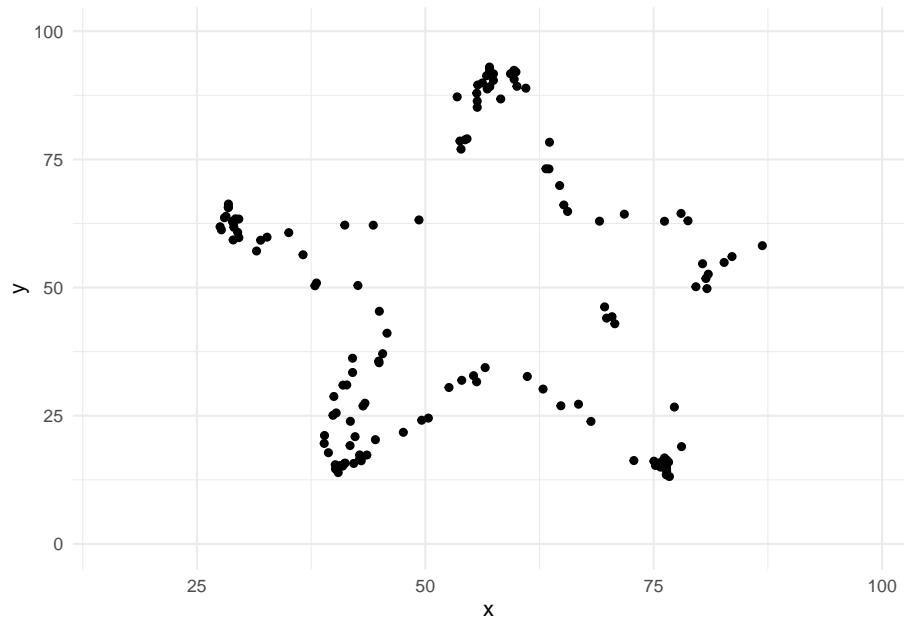
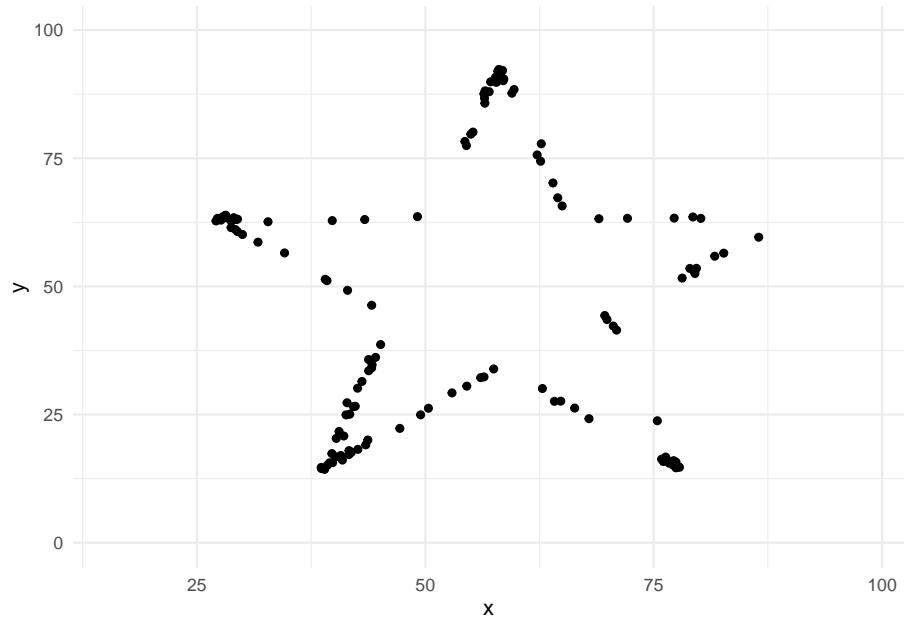


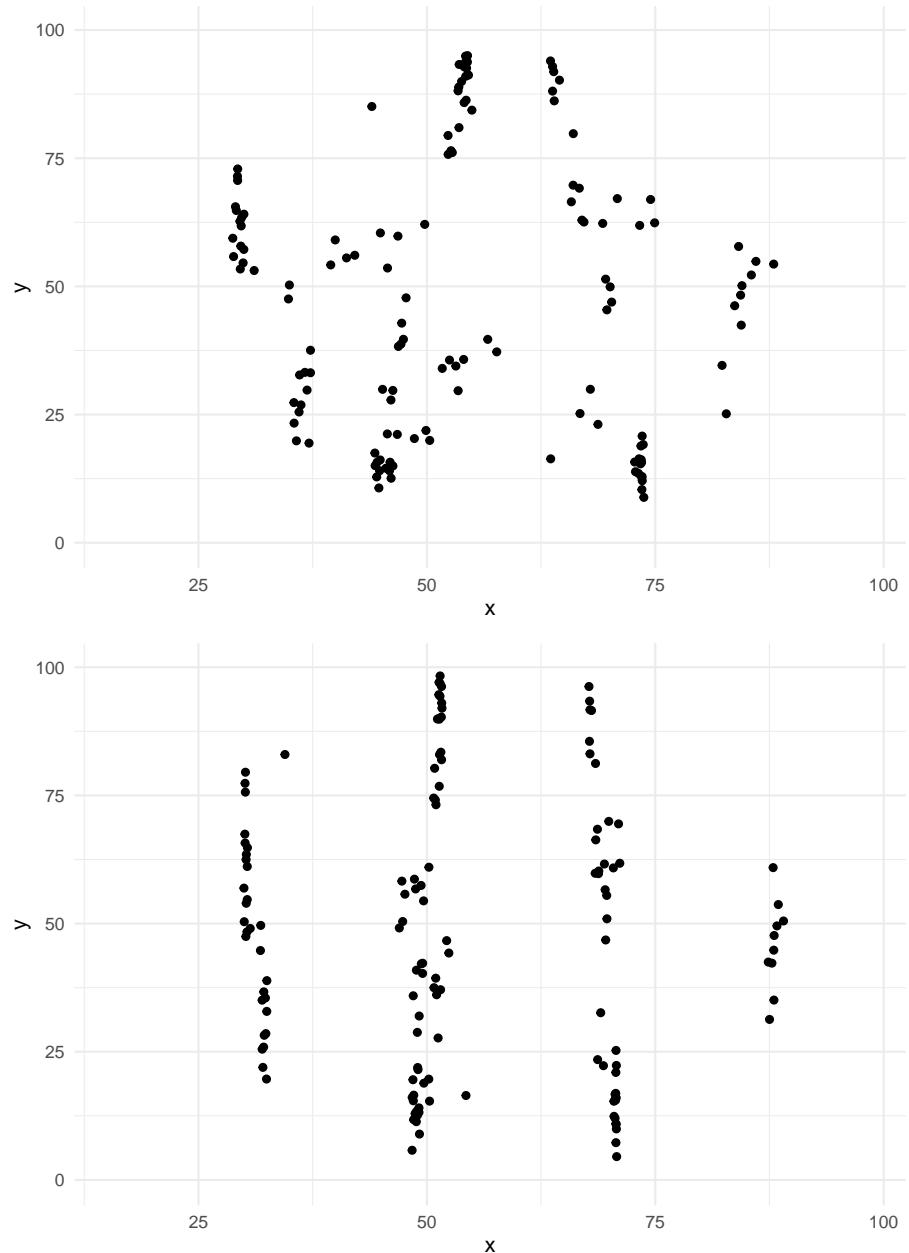


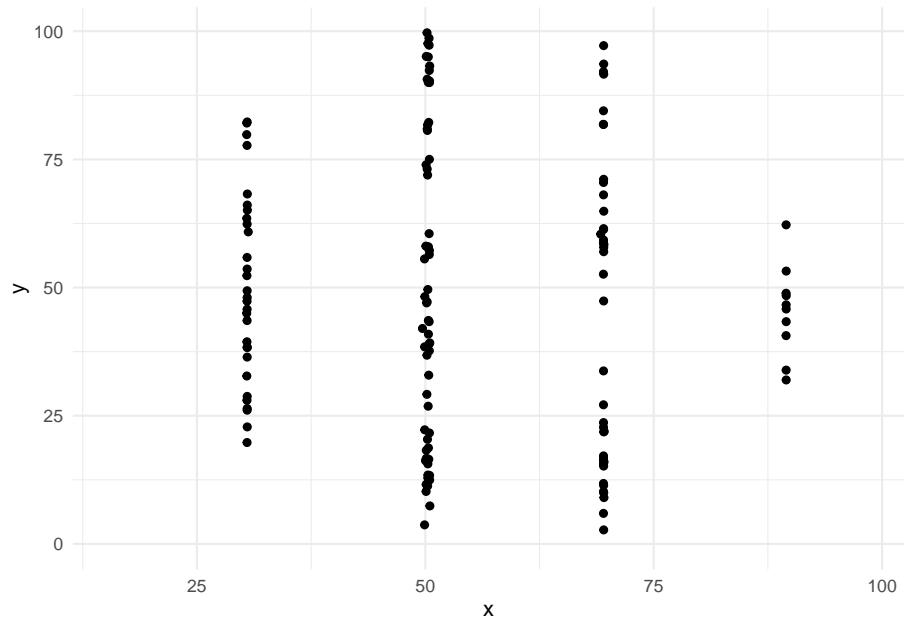
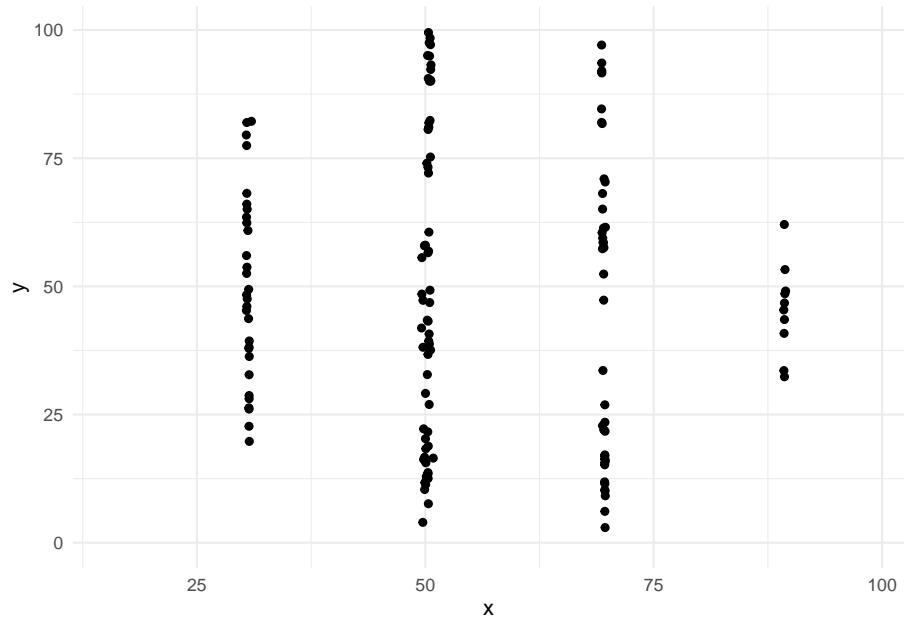


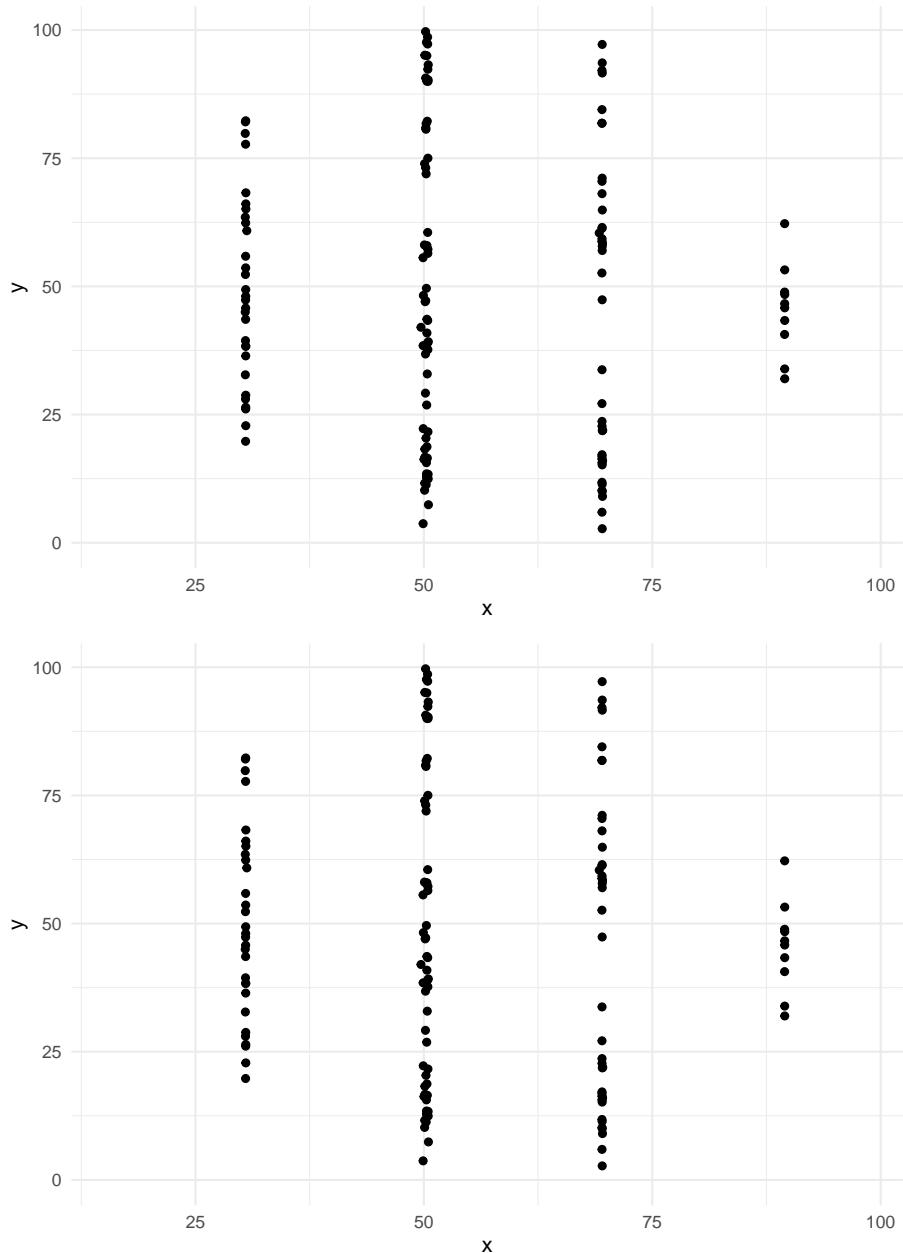


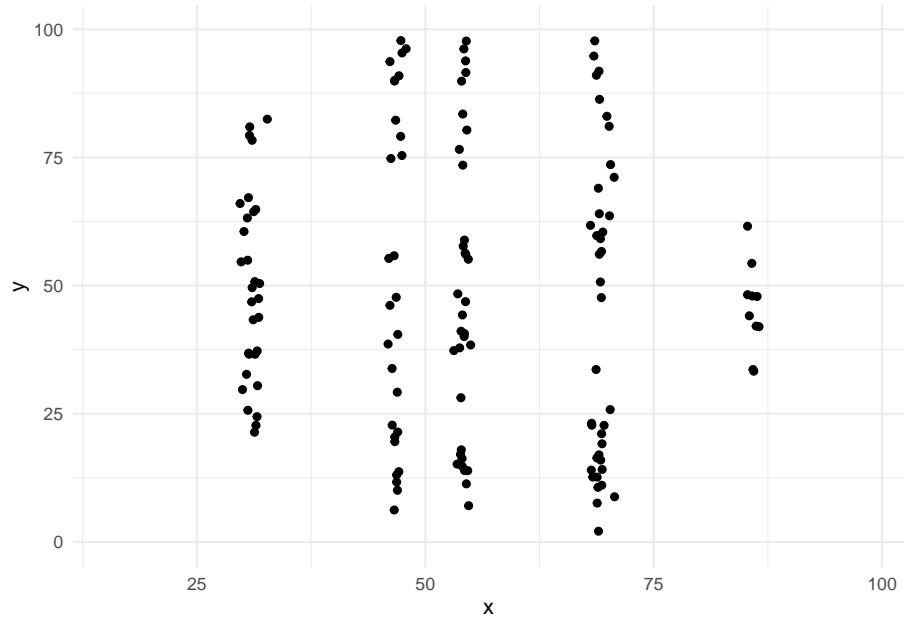
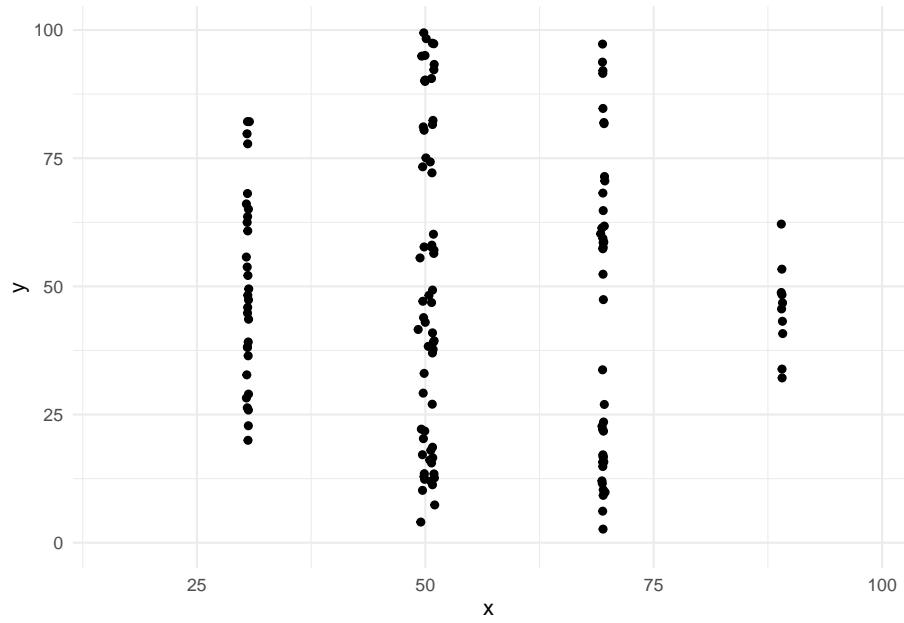


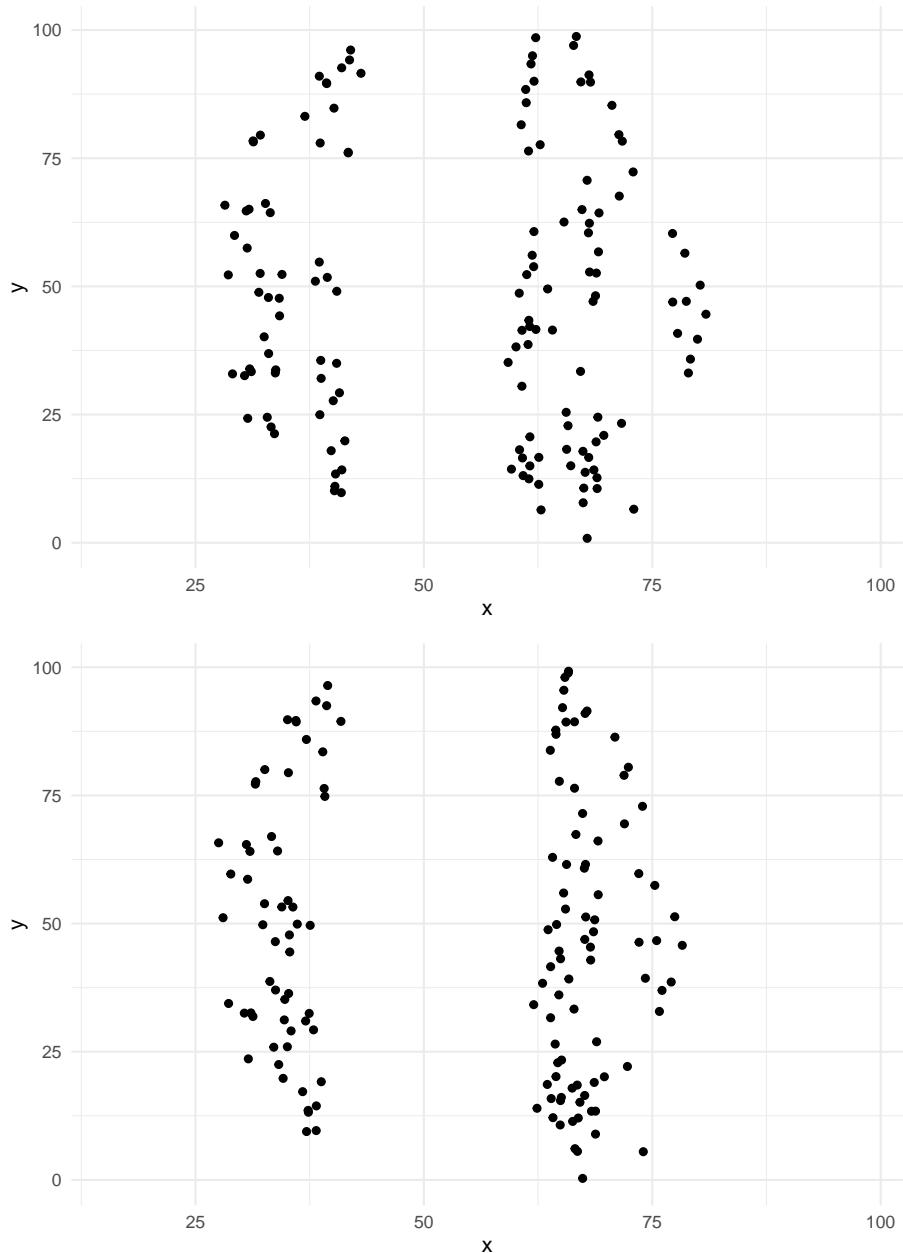


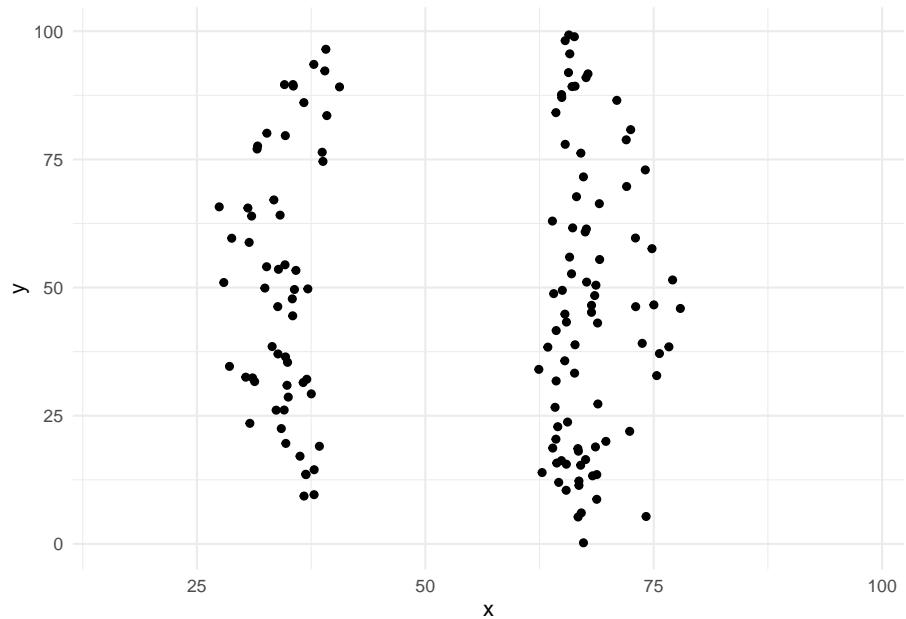
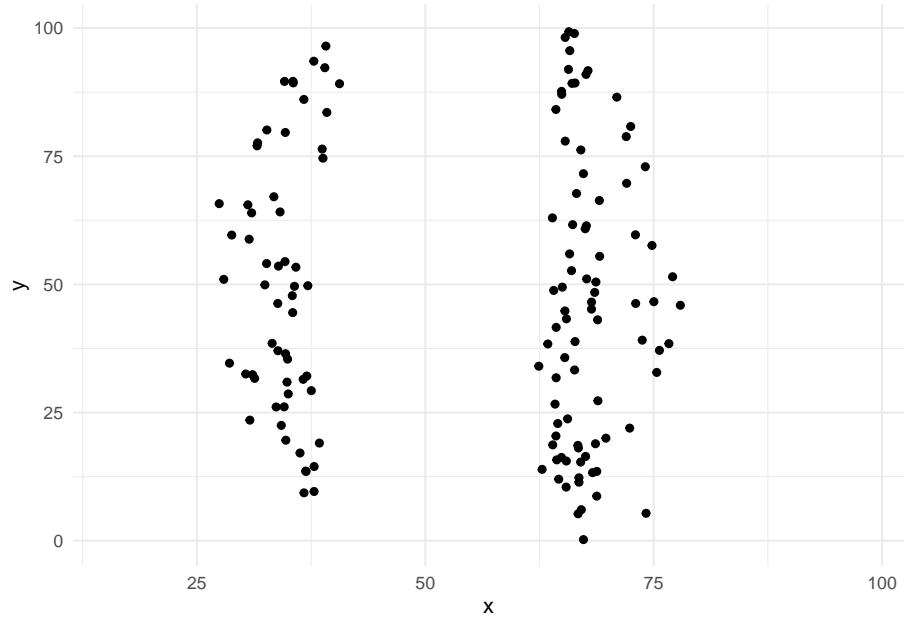


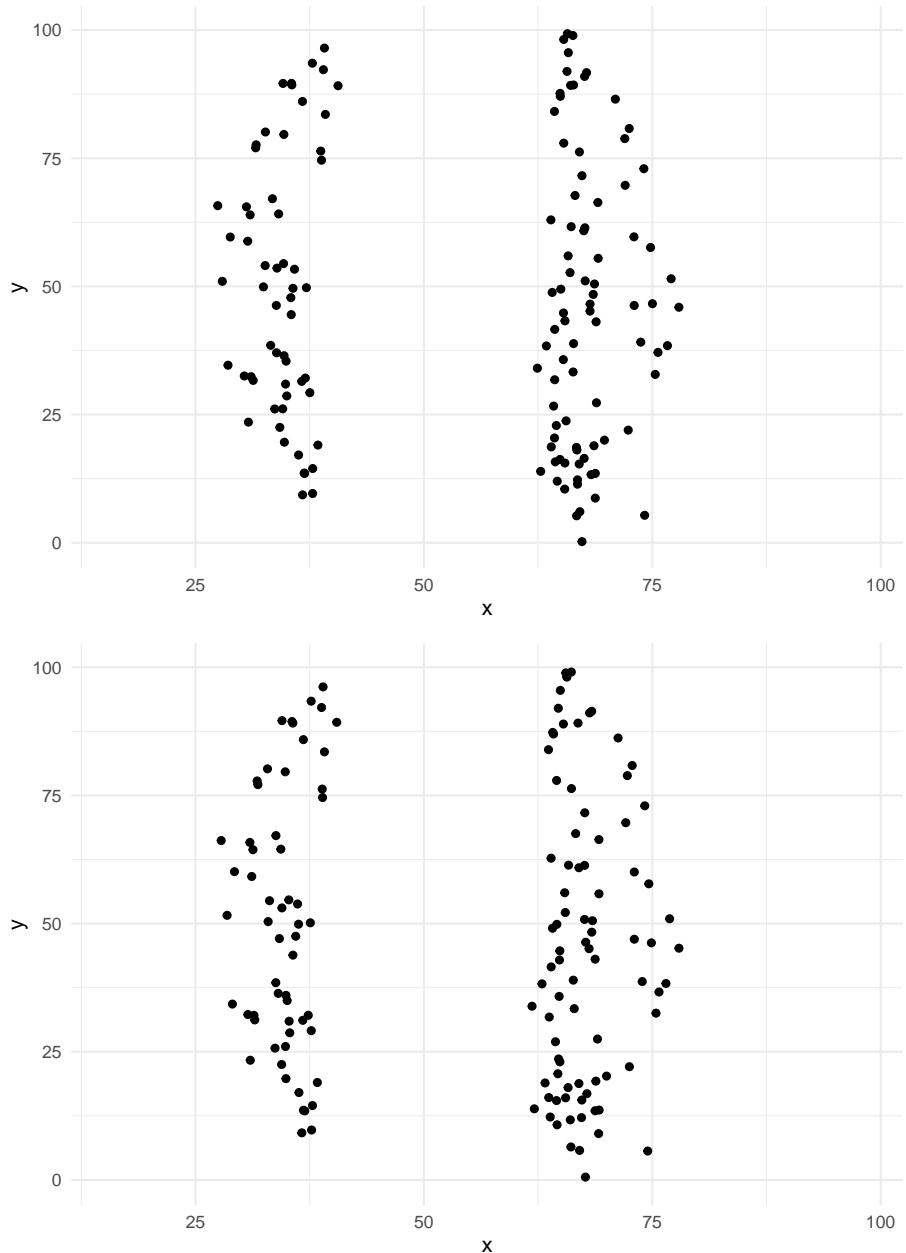


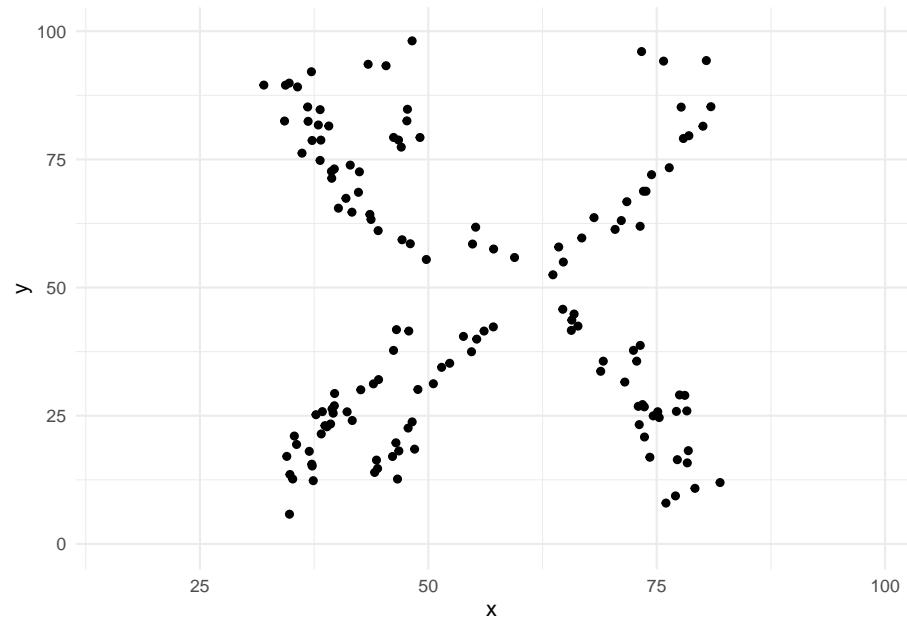
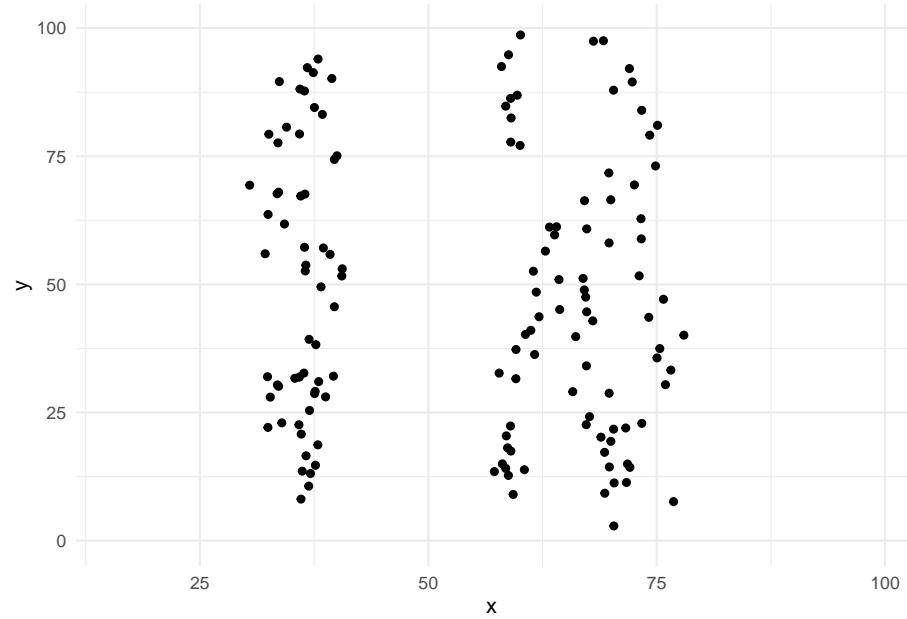


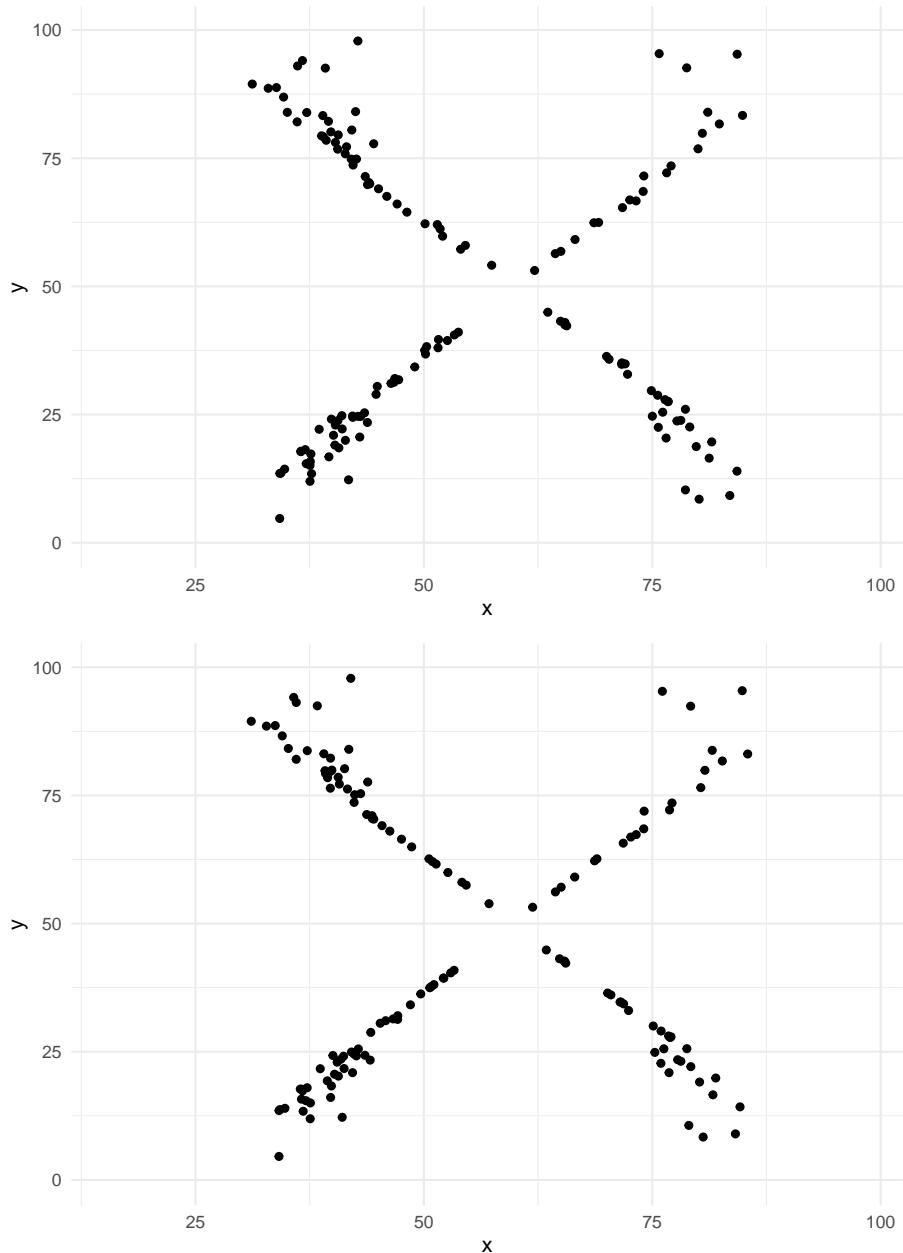


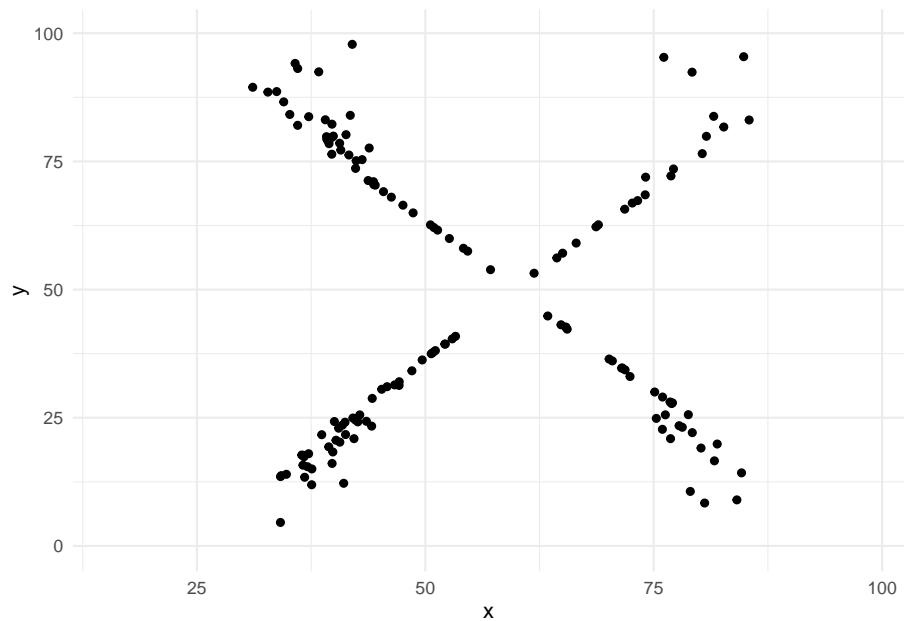
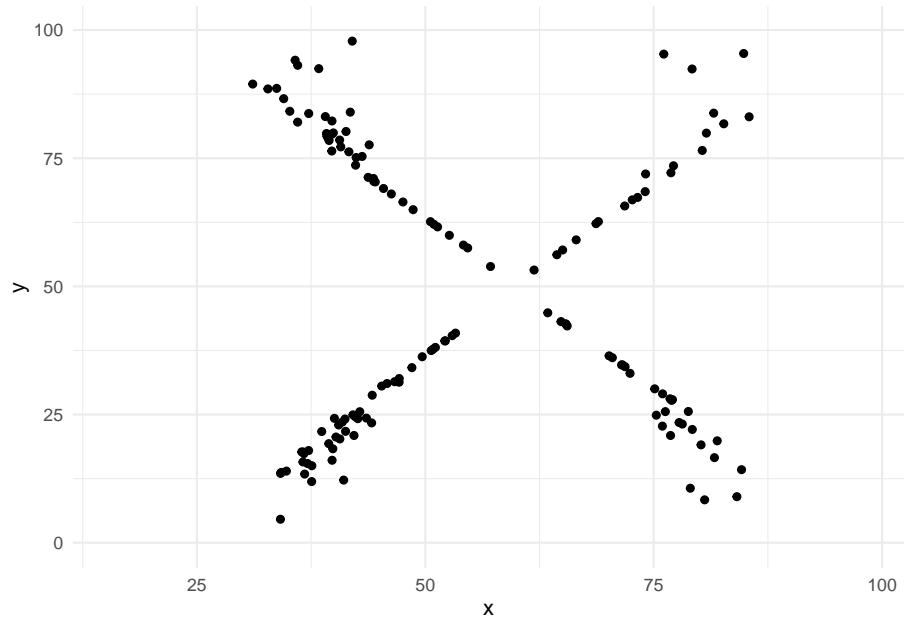


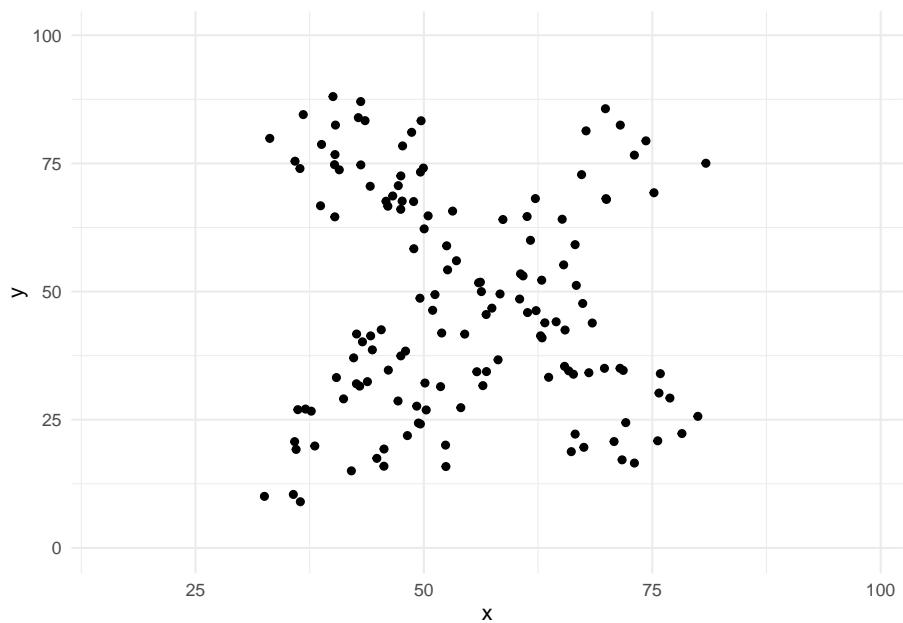
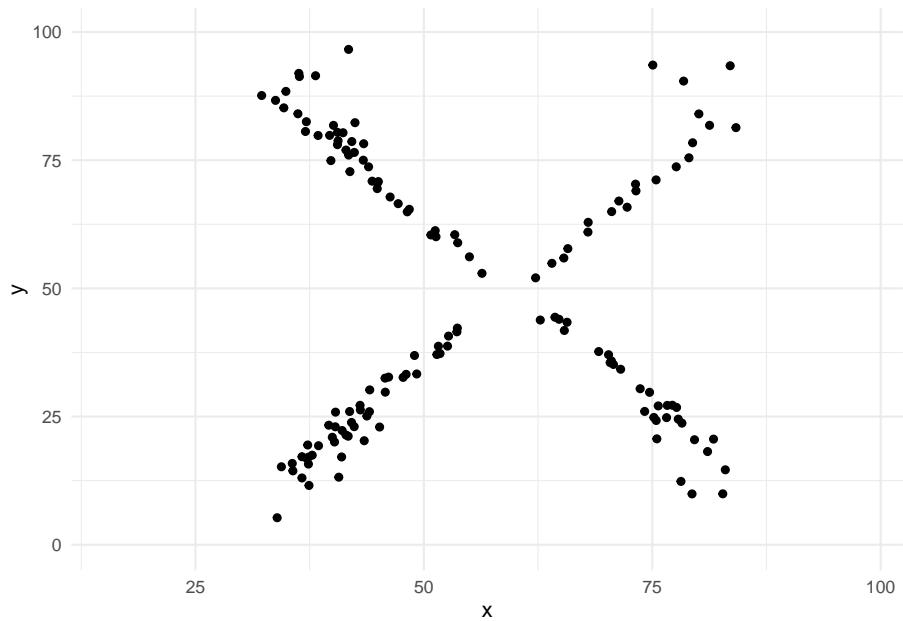


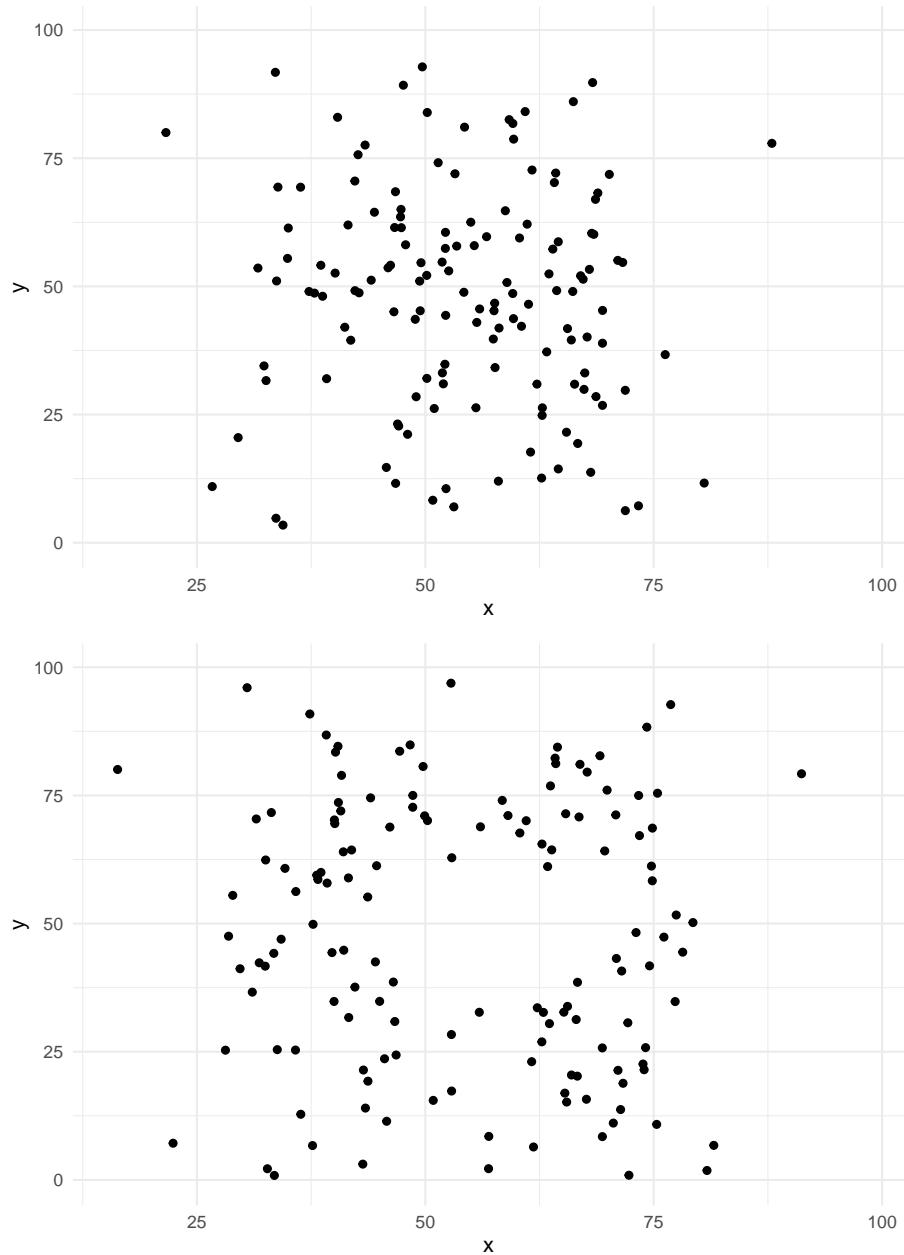












4.4 Workflow för nytt dataset

- Börja *alltid* med visualisering, särskilt om du ska göra modellerings

- explorativ visualisering är en iterativ process
- Börjar övergripande sedan mera specialiserade beroende på hur dina hypoteser/frågor utvecklas

Gör *alltid* explorativ analys med ett programmeringsspråk

- Det gör att din analys finns dokumenterad
- Du kan dela med den till andra
- Den är reproducerbar och kan granskas

Saker du bör undvika: BI-verktyg för explorativ analys och Excel.

4.5 Gapminder

- Vi har fått ett dataset `gapminder` och egentligen inga andra uppgifter och vi vill förstå det
- Vi börjar med att snabbt kolla på datan med `head()`

```
library(gapminder)
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.
```

- Undersöker med hjälp av `summary()`/`skim()`

```
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
##
```

```
library(skimr)  
skim(gapminder)
```

```

-- Data Summary -----
Name          Values
Name          gapminder
Number of rows 1704
Number of columns 6

-----
Column type frequency:
  factor      2
  numeric     4

-----
Group variables   None

-- Variable type: factor --
skim_variable n_missing complete_rate ordered n_unique top_counts
1 country        0            1 FALSE          142 Afg: 12, Alb: 12, Alg: 12, Ang: 12
2 continent       0            1 FALSE           5 Afr: 624, Asi: 396, Eur: 360, Ame: 300

-- Variable type: numeric --
skim_variable n_missing complete_rate    mean      sd    p0    p25    p50    p75    p100 hist
1 year           0            1    1980.    17.3  1952    1966.    1980.    1993.    2007
2 lifeExp         0            1      59.5    12.9   23.6    48.2    60.7    70.8    82.6
3 pop             0            1 29601212. 106157897. 60011 2793664 7023596. 19585222. 1318683096
4 gdpPercap       0            1    7215.    9857.   241.    1202.    3532.    9325.   113523.

```

4.5.1 Datapunkter per land?

```
gapminder %>% filter(country == "Sweden") %>% arrange(year)
```

```
## # A tibble: 12 x 6
##   country continent year lifeExp     pop gdpPercap
##   <fct>    <fct>   <int>   <dbl>   <int>   <dbl>
## 1 Sweden    Europe    1952    71.9  7124673   8528.
## 2 Sweden    Europe    1957    72.5  7363802   9912.
## 3 Sweden    Europe    1962    73.4  7561588  12329.
## 4 Sweden    Europe    1967    74.2  7867931  15258.
## 5 Sweden    Europe    1972    74.7  8122293  17832.
```

```
##  6 Sweden Europe    1977    75.4 8251648    18856.
##  7 Sweden Europe    1982    76.4 8325260    20667.
##  8 Sweden Europe    1987    77.2 8421403    23587.
##  9 Sweden Europe    1992    78.2 8718867    23880.
## 10 Sweden Europe    1997    79.4 8897619    25267.
## 11 Sweden Europe    2002    80.0 8954175    29342.
## 12 Sweden Europe    2007    80.9 9031088    33860.
```

4.5.1.1 Naturlig fråga: Hur påverkar ekonomisk utveckling den förväntande livslängden i ett land?

För att undersöka den här frågan så vill vi göra en visualisering med hjälp av `ggplot2`

Då behöver vi först

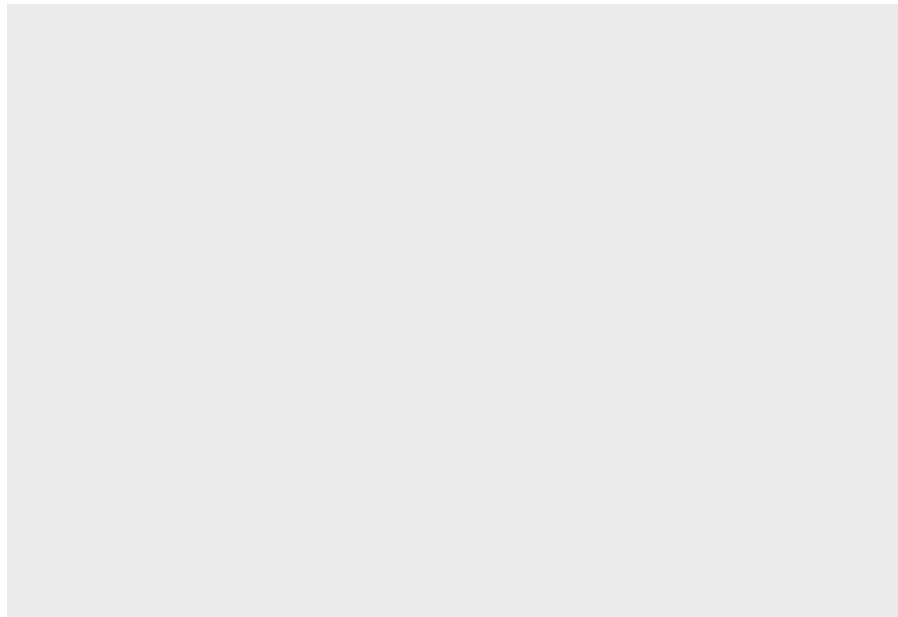
4.6 Grammar of graphics

4.6.1 1. Data

För att minska mängden data kollar vi bara på ett år

```
data <- gapminder %>%
  filter(year == 1972)

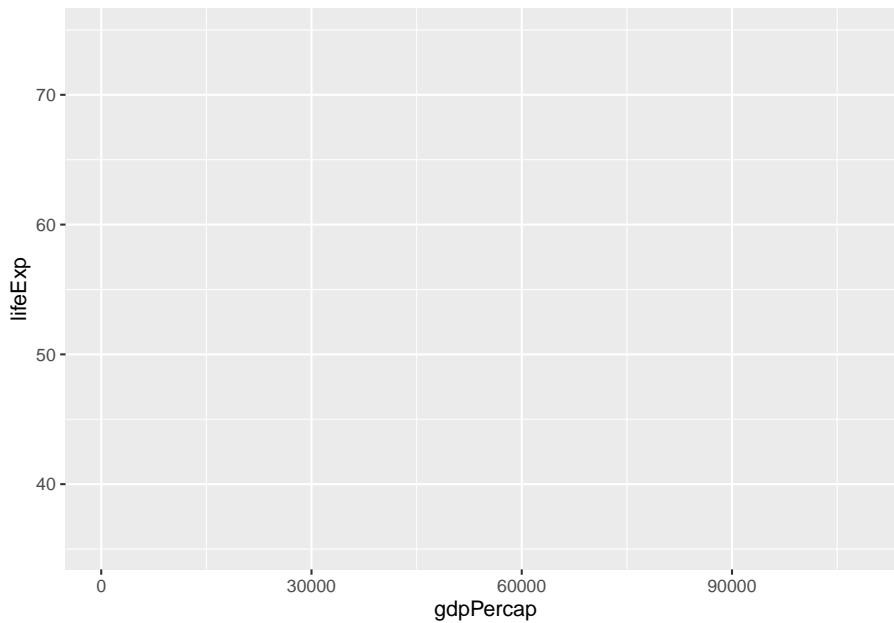
ggplot(data)
```



4.6.2 2. Aesthetics

- Vi behöver mappa data till visualiseringen. Vi mappar data till `aesthetics` i visualiseringen. En `aesthetic` kan vara exempelvis `x`-axeln eller `y`-axeln.
- För att svara på vår fråga kan vi exempelvis mappa `gdpPercap` till `x`-axeln och `lifeExp` till `y`-axeln

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



4.6.3 3. Geometriska objekt

Vi behöver geometriska objekt som representerar data, exempelvis punkter, linjer eller staplar.

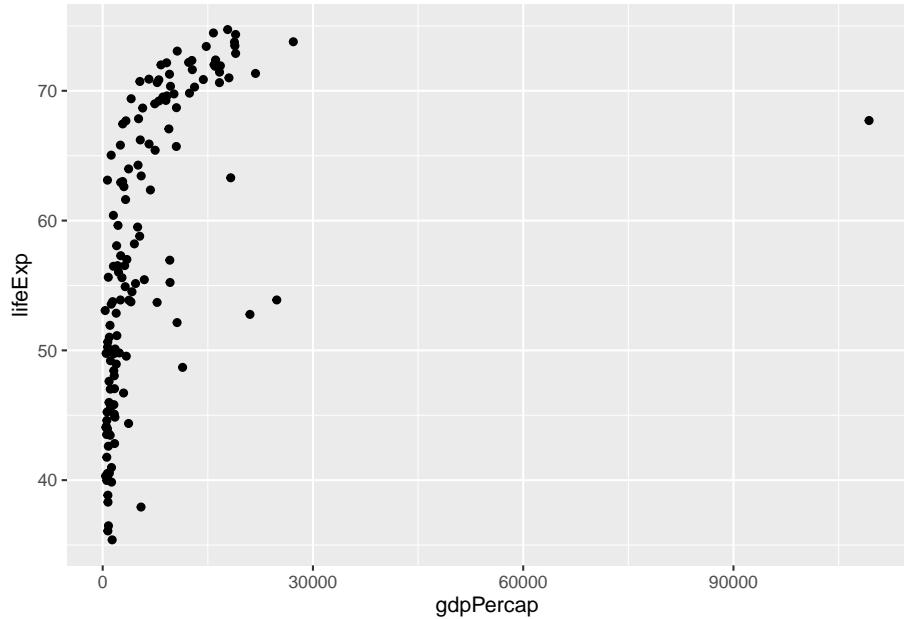
I ggplot2 kallas dessa för `geoms`, exempelvis:

`geom_point()`: punkter,

`geom_line()`: linjer,

`geom_bar()`: staplar

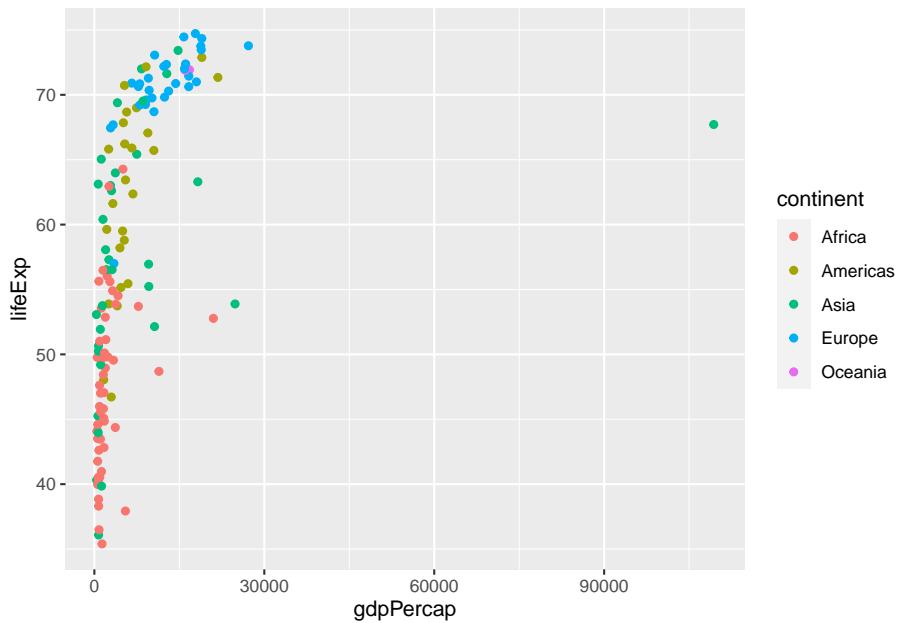
```
ggplot(data = data, aes(x = gdpPerCap, y = lifeExp)) +  
  geom_point()
```



Det verkar finnas ett samband, kan vi förstå mera?

Mer `aesthetics`

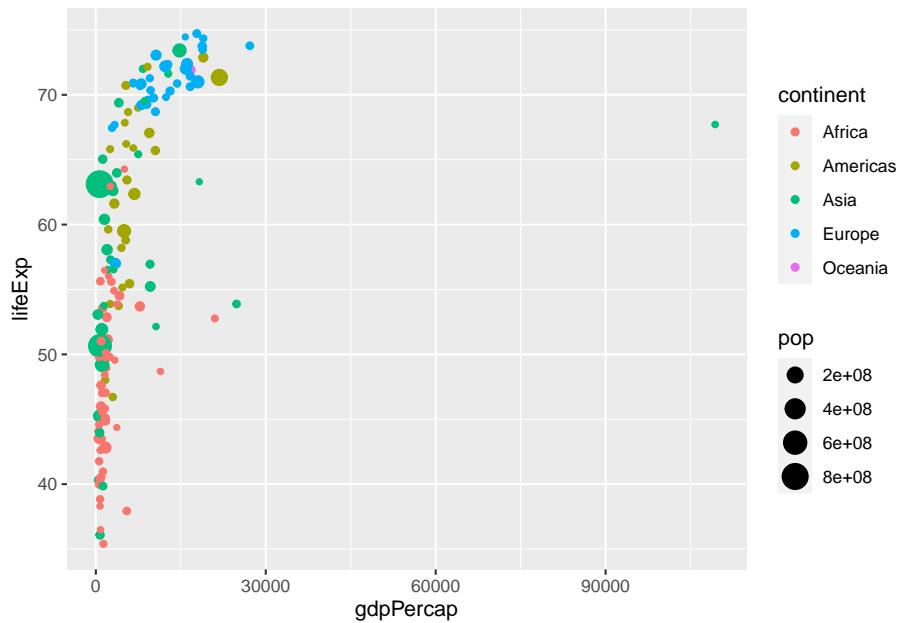
```
p <- ggplot(data = data,
             aes(x = gdpPercap,
                 y = lifeExp,
                 color = continent)) +
  geom_point()
```



4.6.3.1 Storlek ~ invånare

ännu mer aesthetics

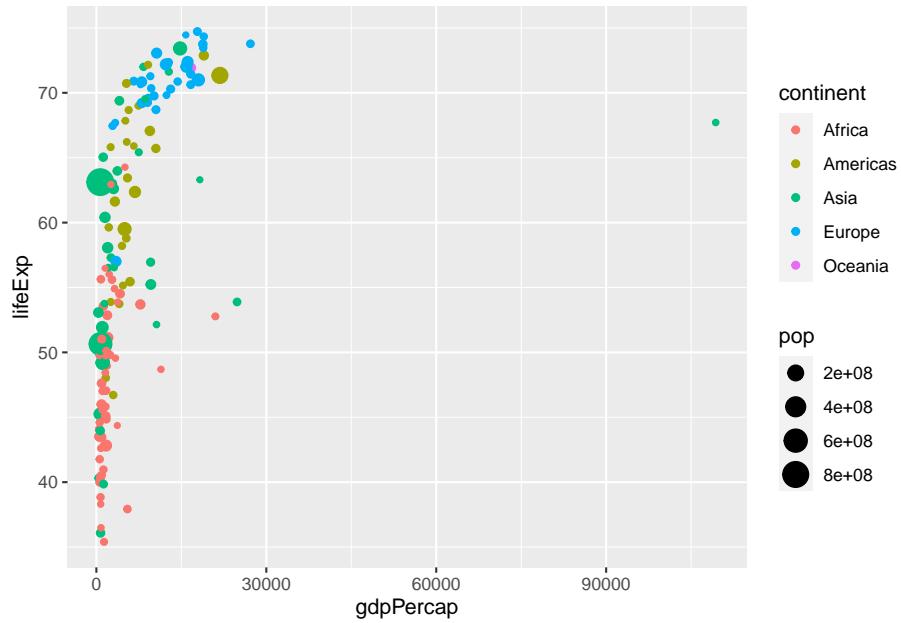
```
ggplot(data = data,
       aes(x = gdpPercap,
           y = lifeExp,
           color = continent,
           size = pop)) +
  geom_point()
```



4.6.4 4. Skala

Vilken skala ska axlarna i grafen ha?

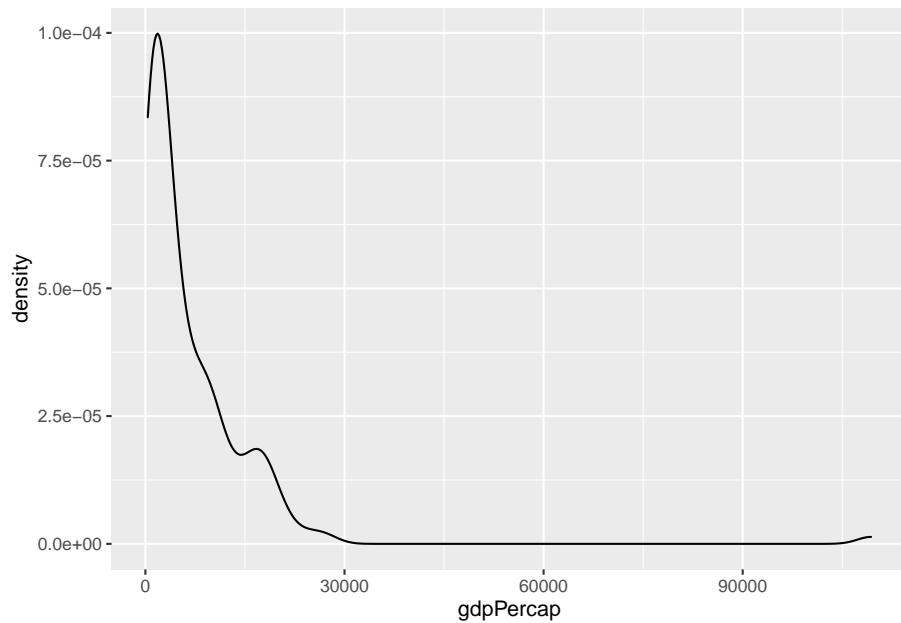
```
ggplot(data = data, aes(x = gdpPerCap, y = lifeExp,
                        size = pop, color = continent)) +
  geom_point() +
  scale_y_continuous() + ## Default value
  scale_x_continuous()
```



4.6.5 GDP per capita är skevt fördelad

Vanligt i variabler som motsvarar pengar

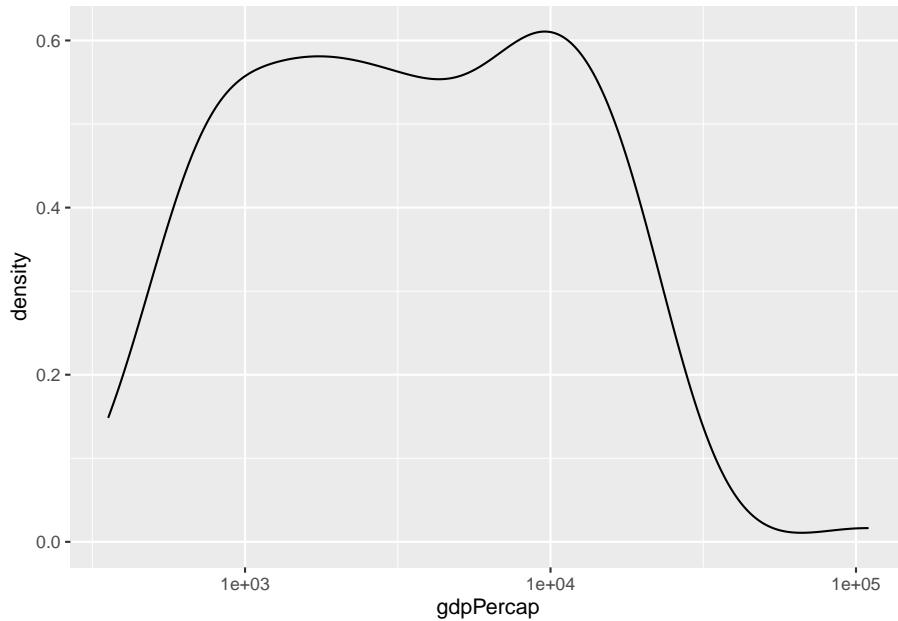
```
ggplot(data, aes(x = gdpPerCap)) +
  geom_density()
```



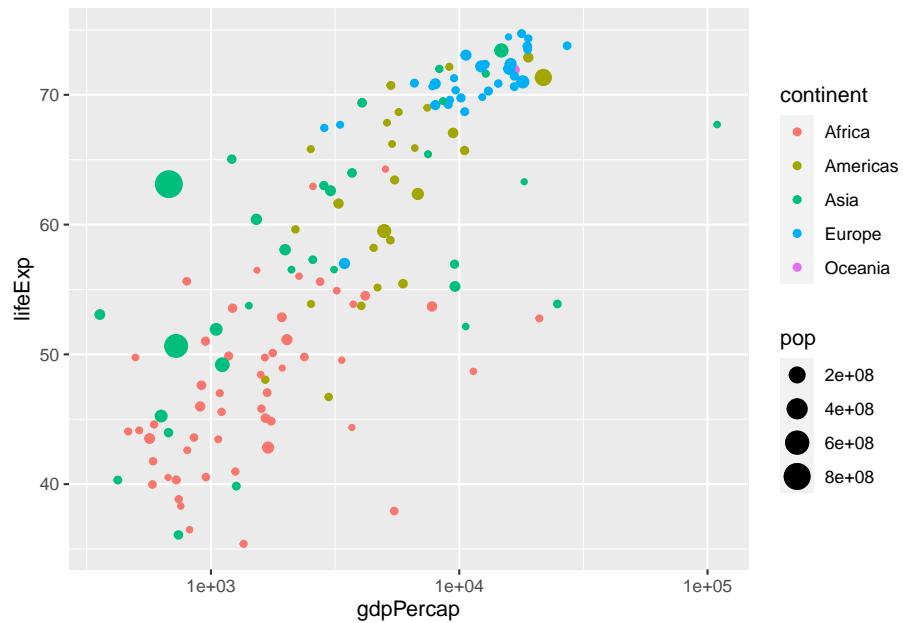
4.6.5.1 Logaritmisk skala

Genom att ta logaritmen kan vi justera för den skeva fördelningen

```
ggplot(data, aes(x = gdpPerCap)) +  
  geom_density() +  
  scale_x_log10() ##
```



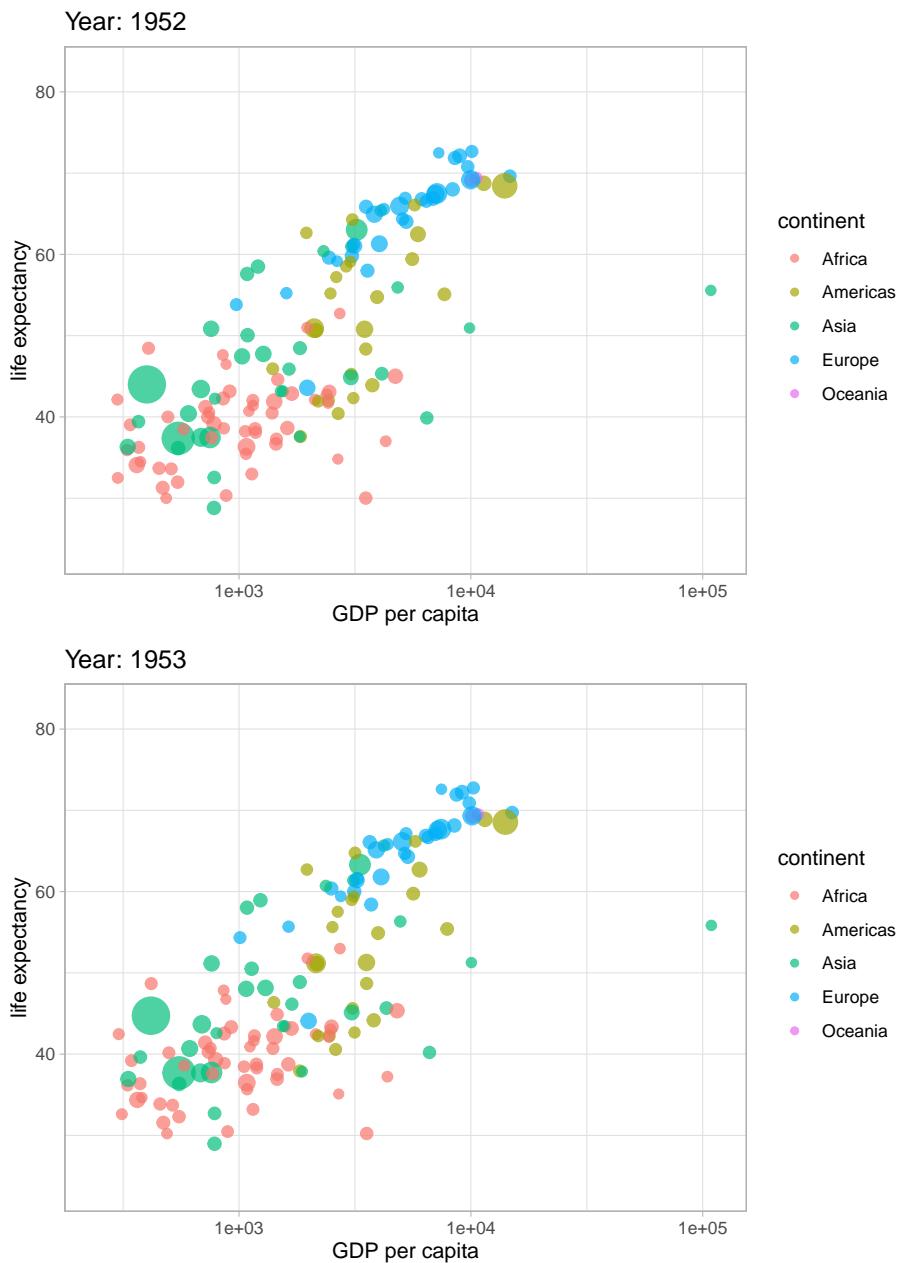
```
ggplot(data = data, aes(x = gdpPercap,
                        y = lifeExp,
                        size = pop,
                        color = continent)) +
  geom_point() +
  scale_x_log10()
```

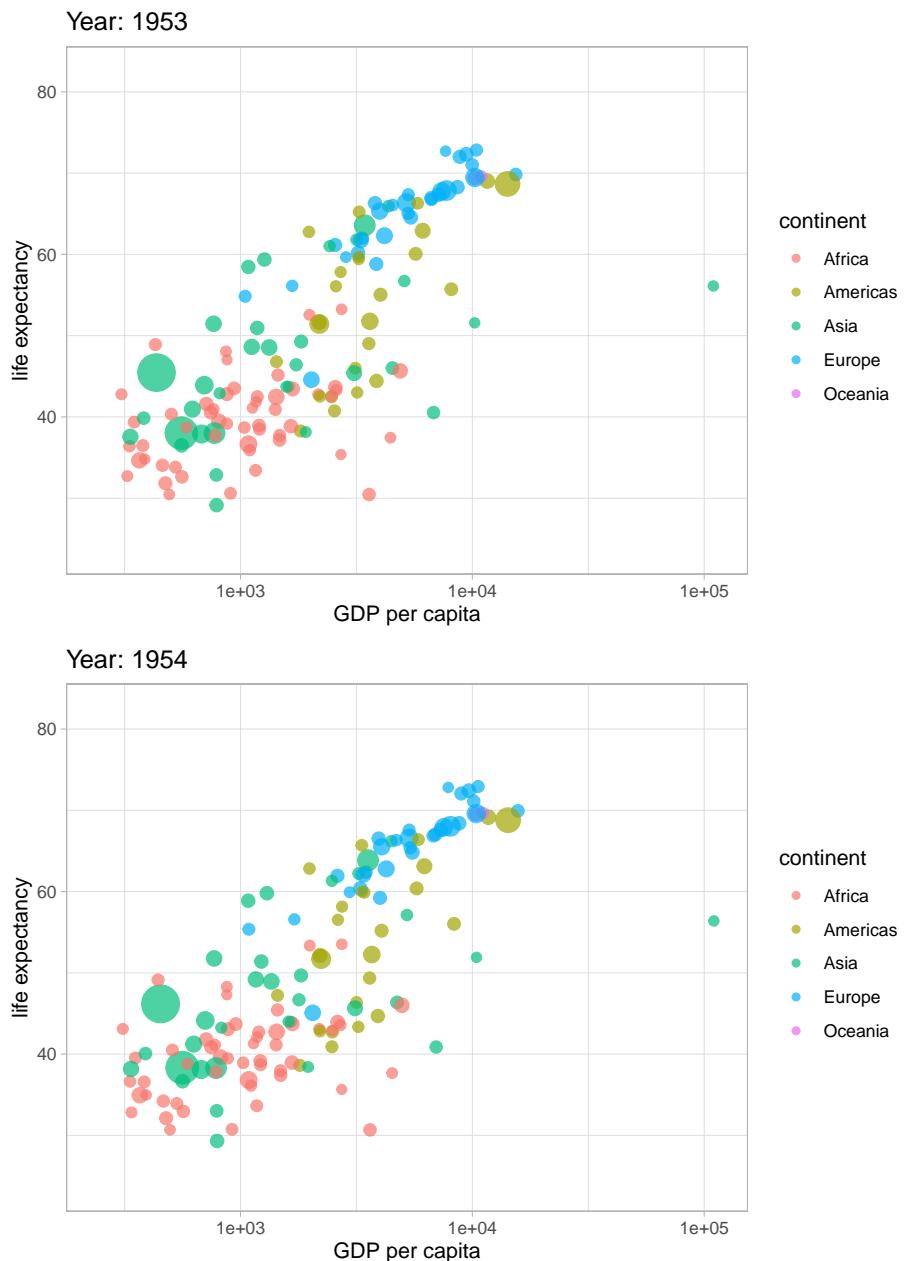


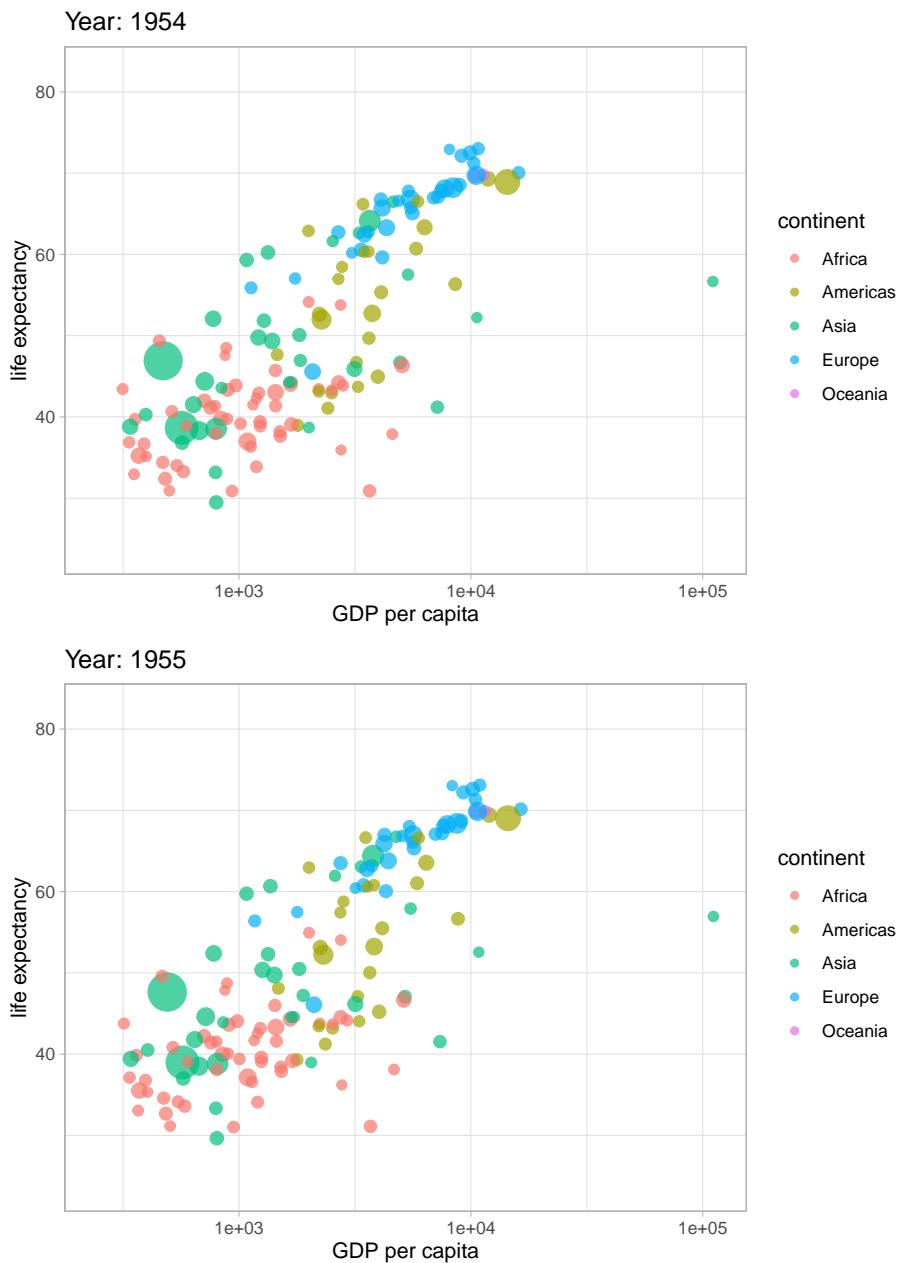
#<4 Hans Roslings berömda visualisering

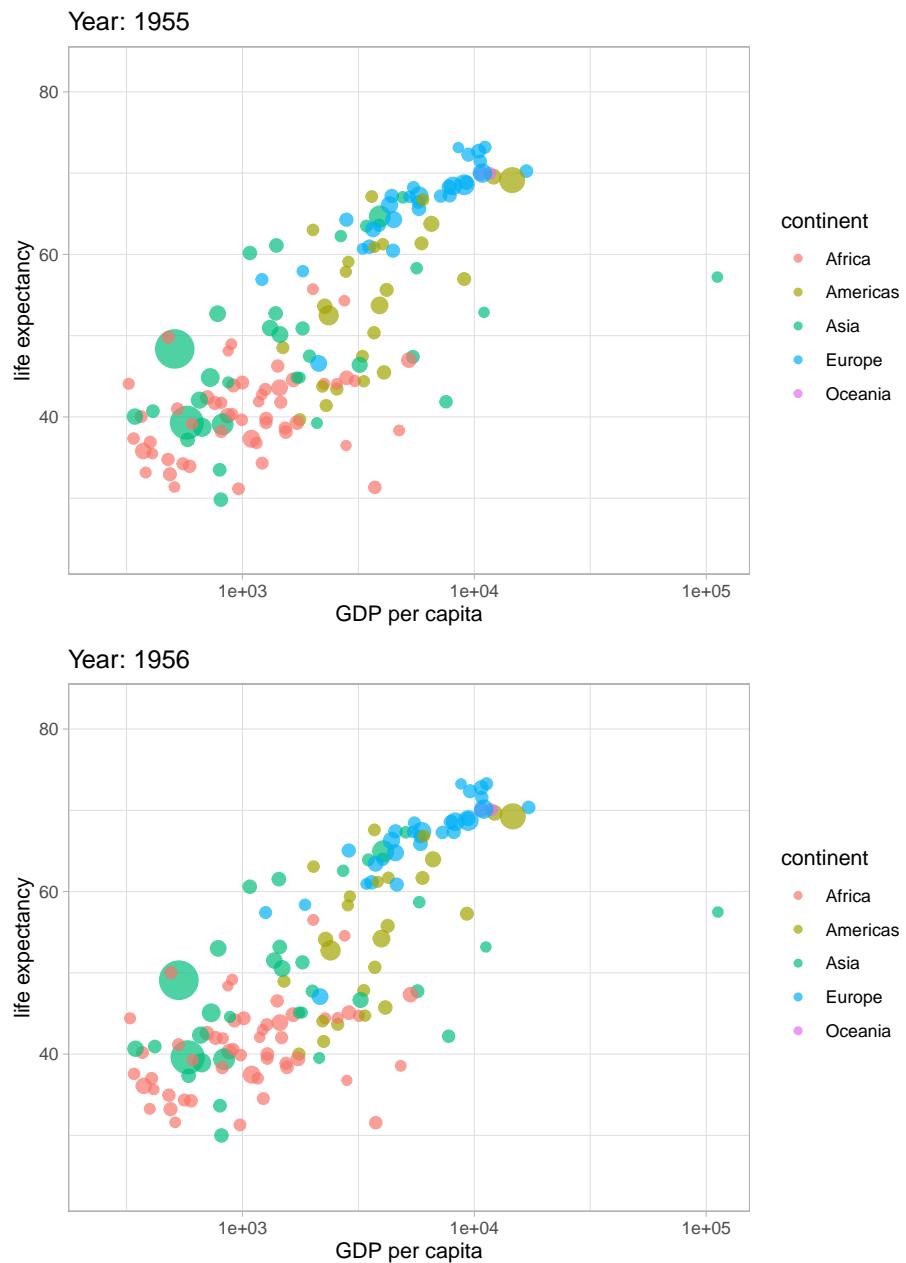
```
library(gganimate)
library(gapminder)

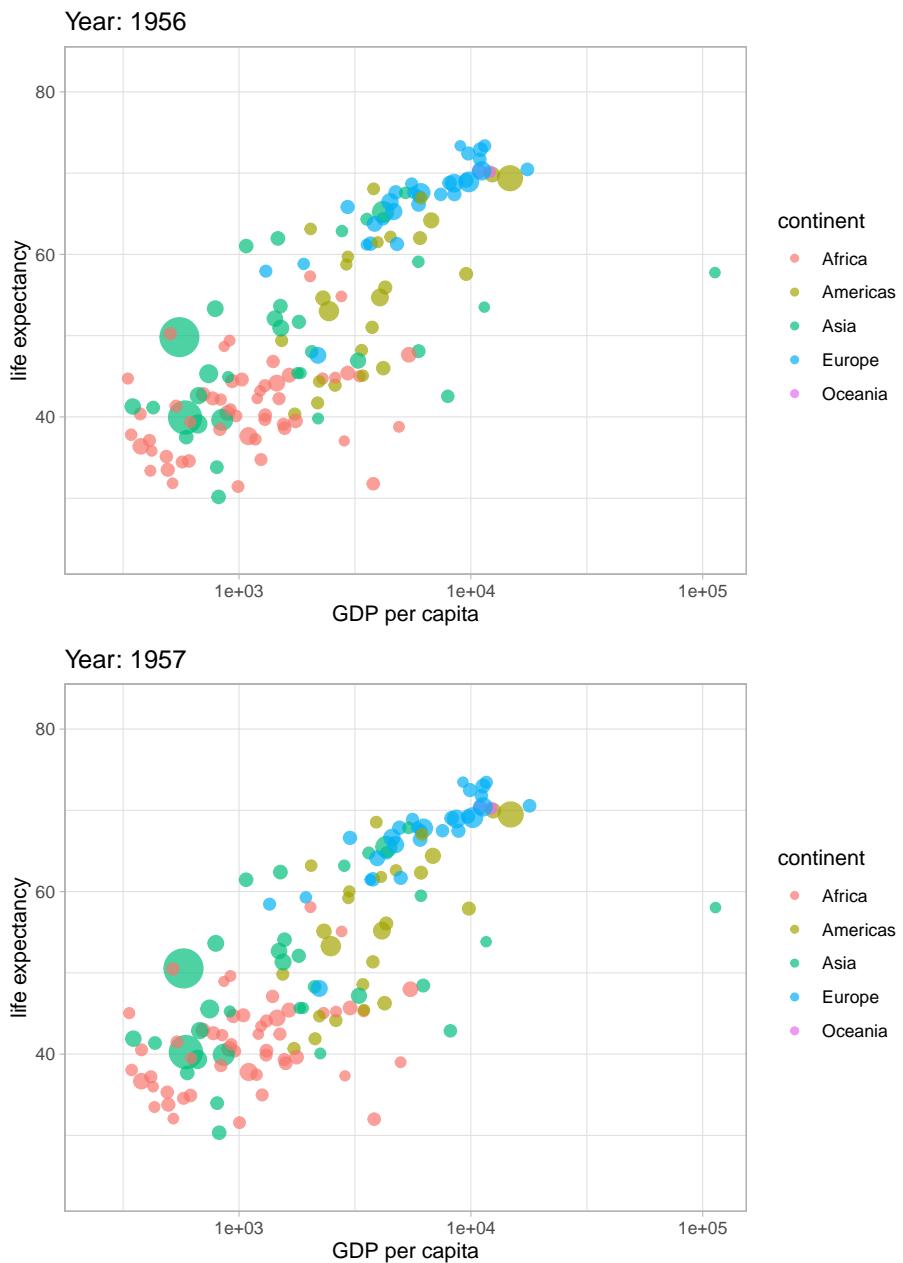
ggplot(gapminder, aes(gdpPercap, lifeExp, size = pop, colour = continent)) +
  geom_point(alpha = 0.7) +
  #scale_colour_manual(values = country_colors) +
  scale_size(range = c(2, 12)) +
  scale_x_log10() +
  guides(color= guide_legend(), size=FALSE) +
  theme_light() +
  labs(title = 'Year: {frame_time}',
       x = 'GDP per capita',
       y = 'life expectancy') +
  transition_time(year) +
  ease_aes('linear')
```

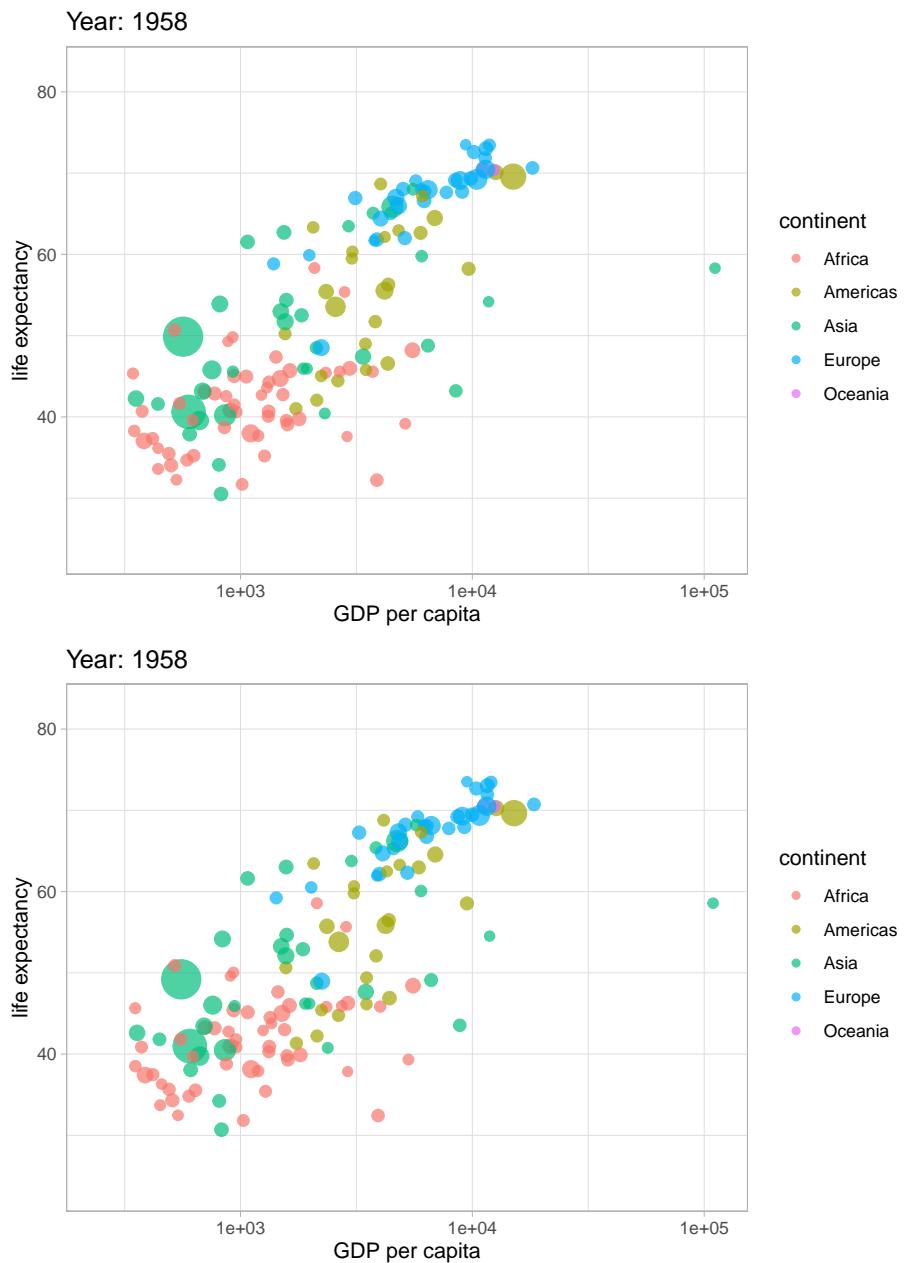


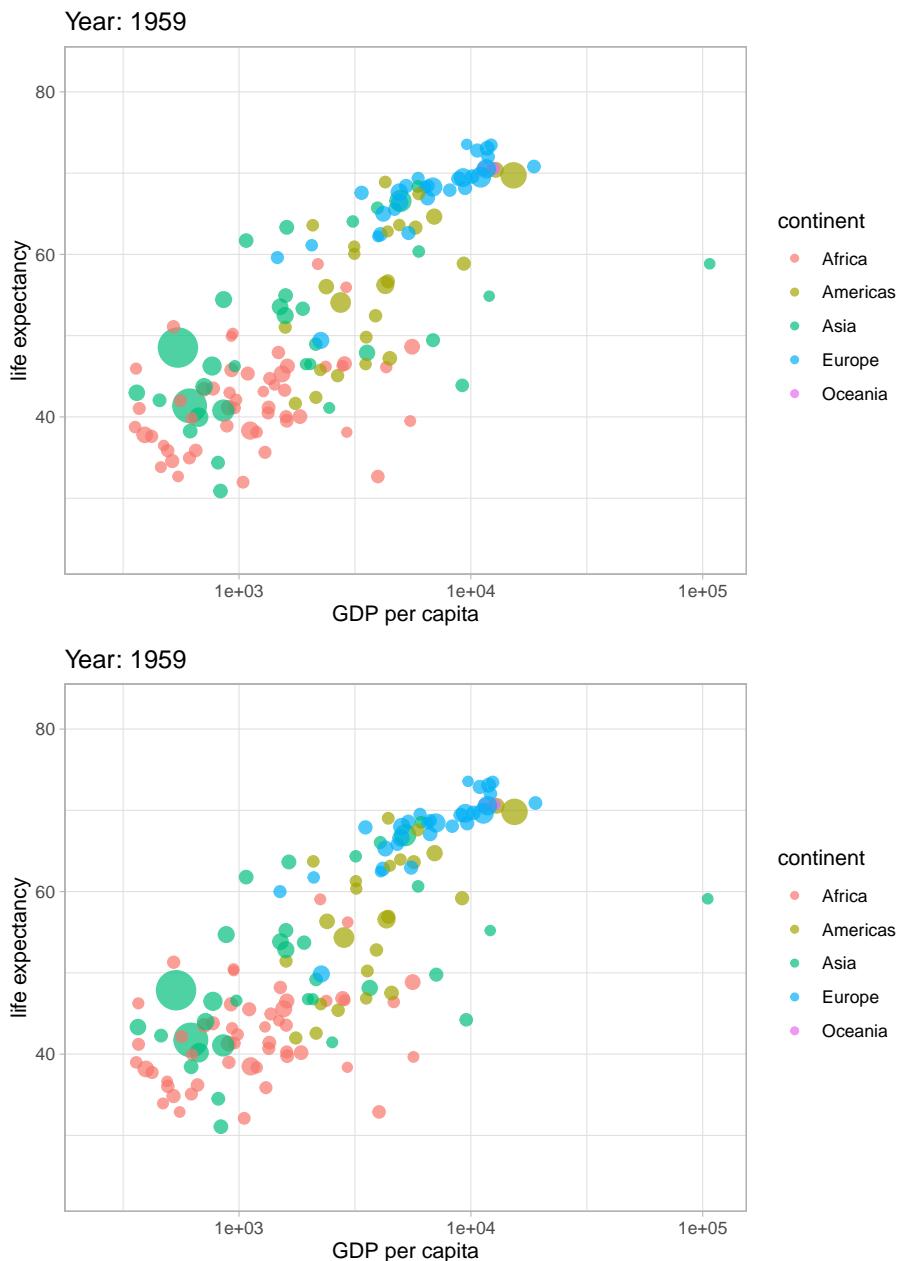


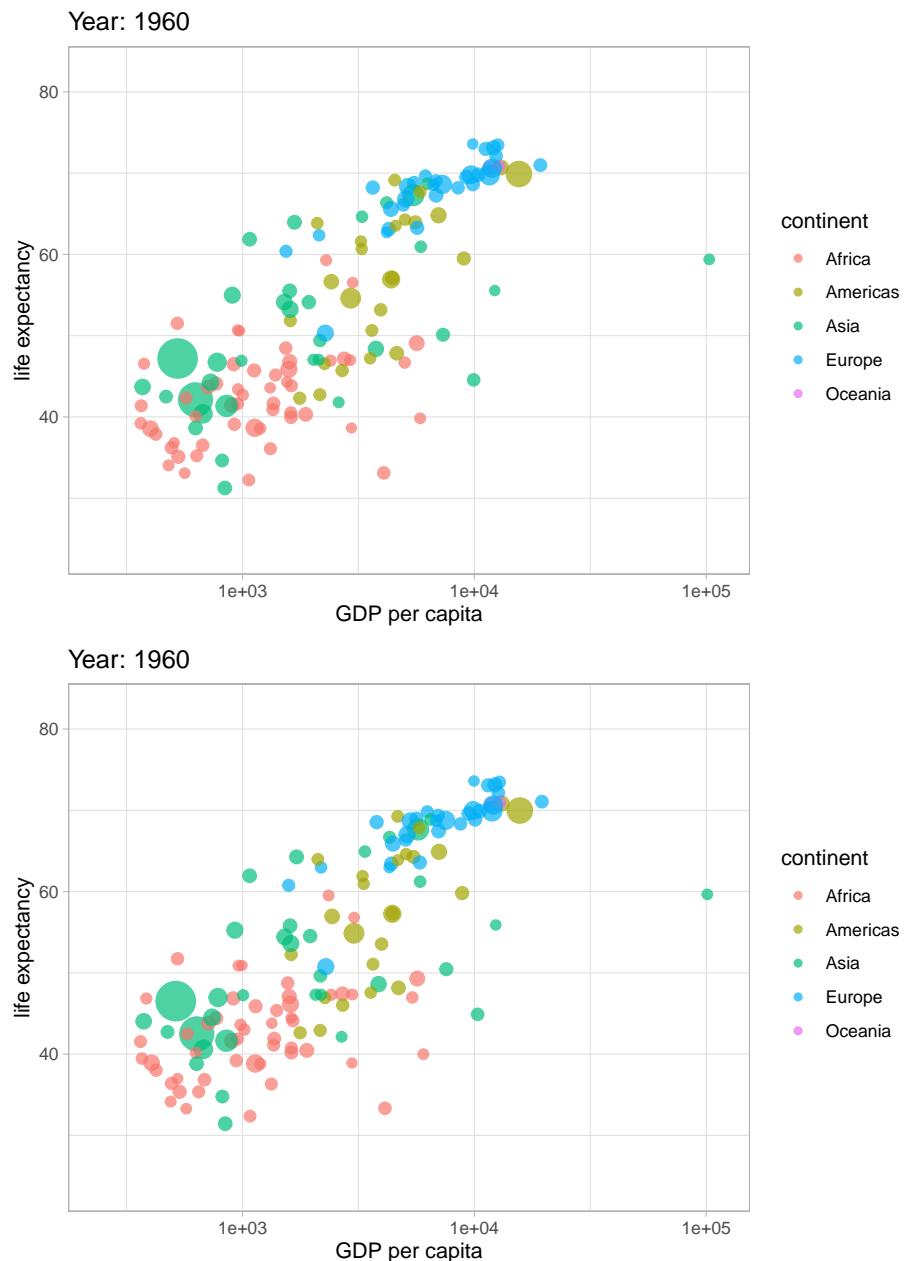


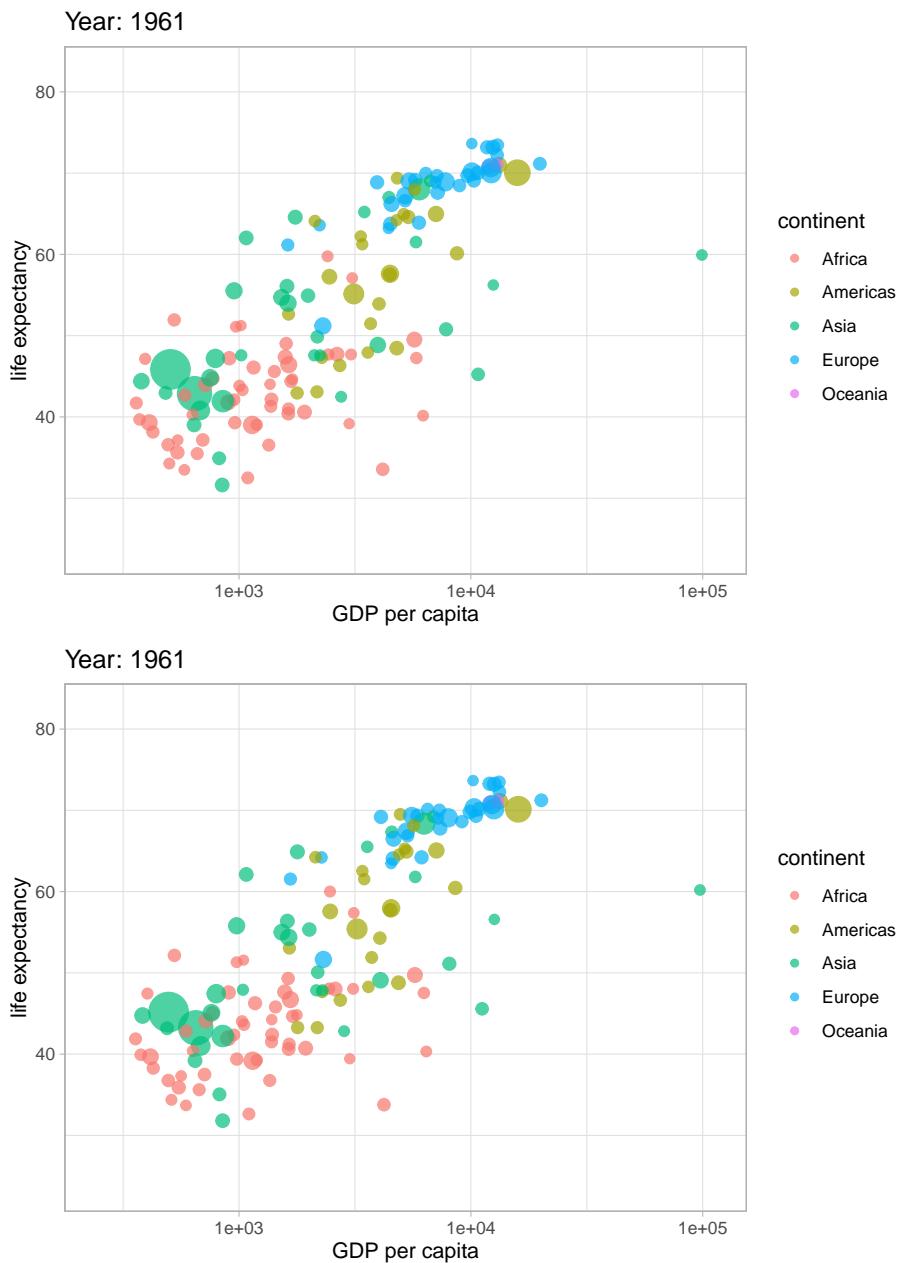


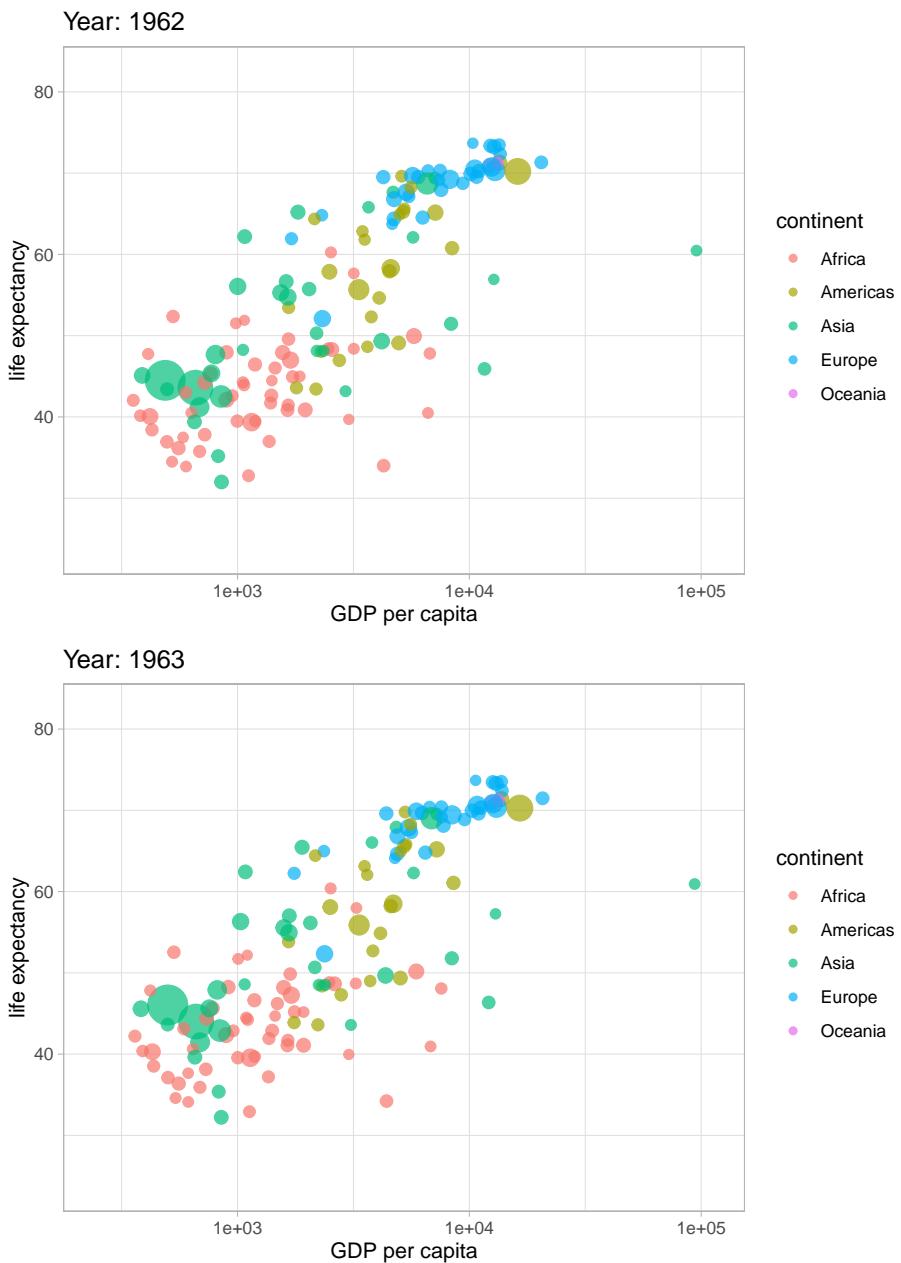


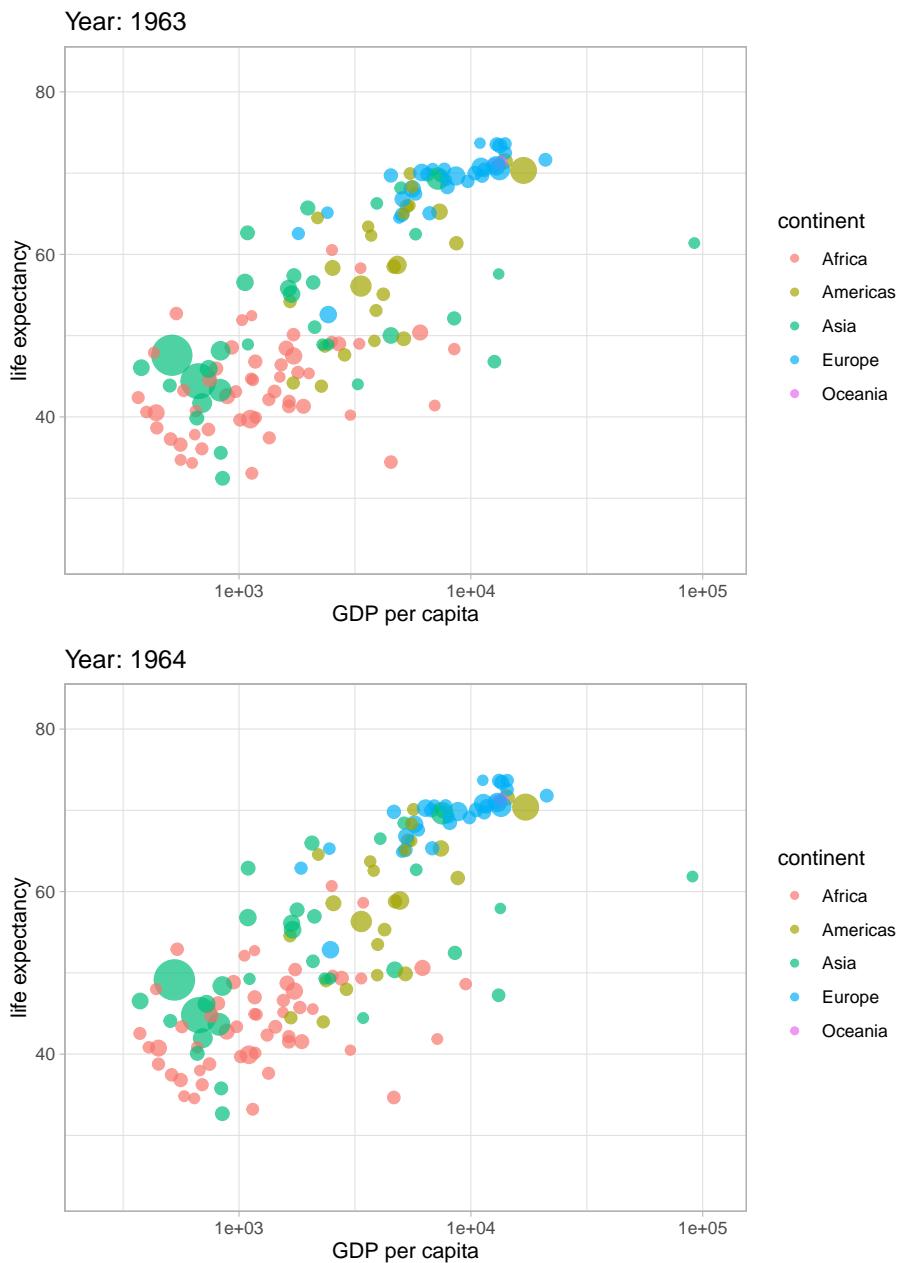


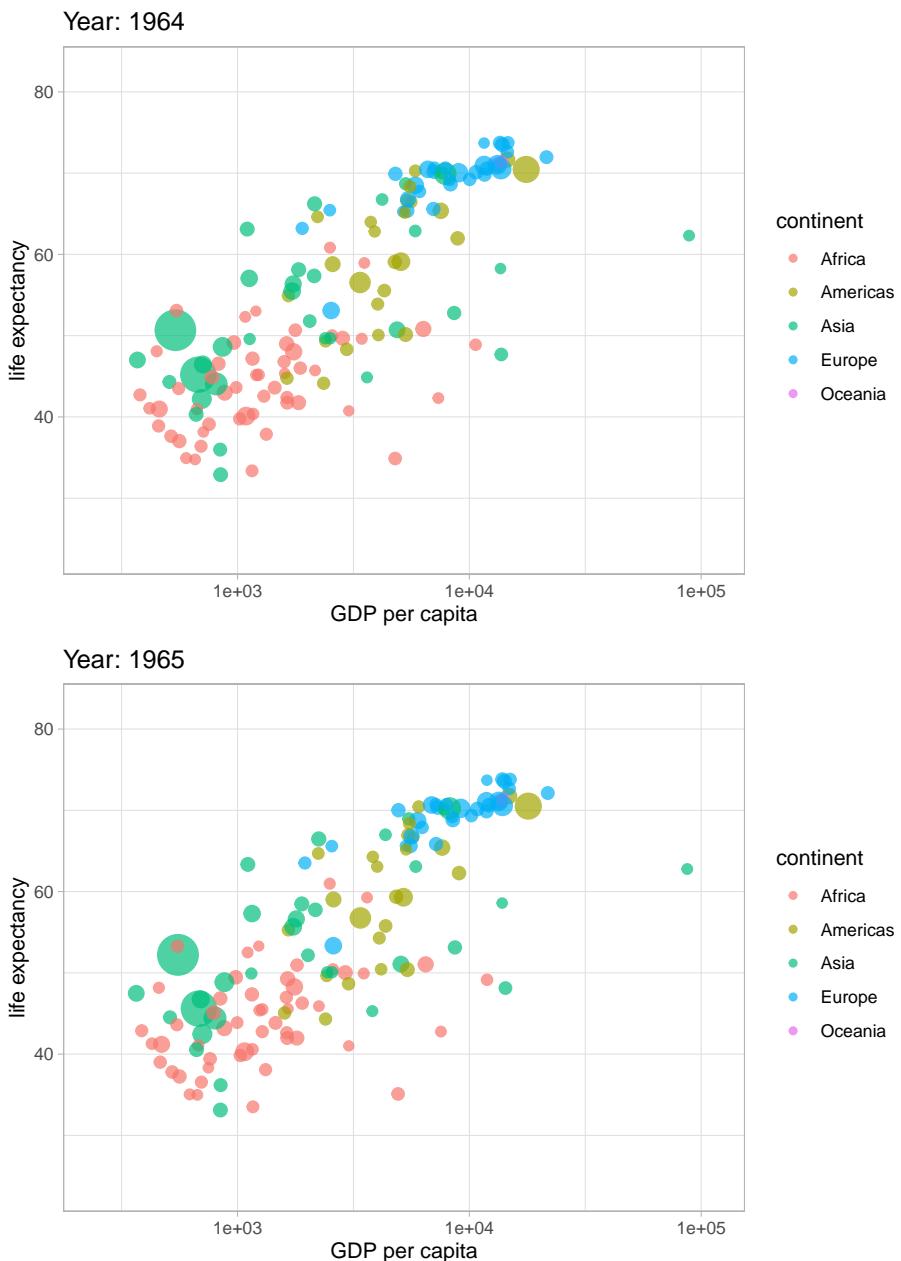


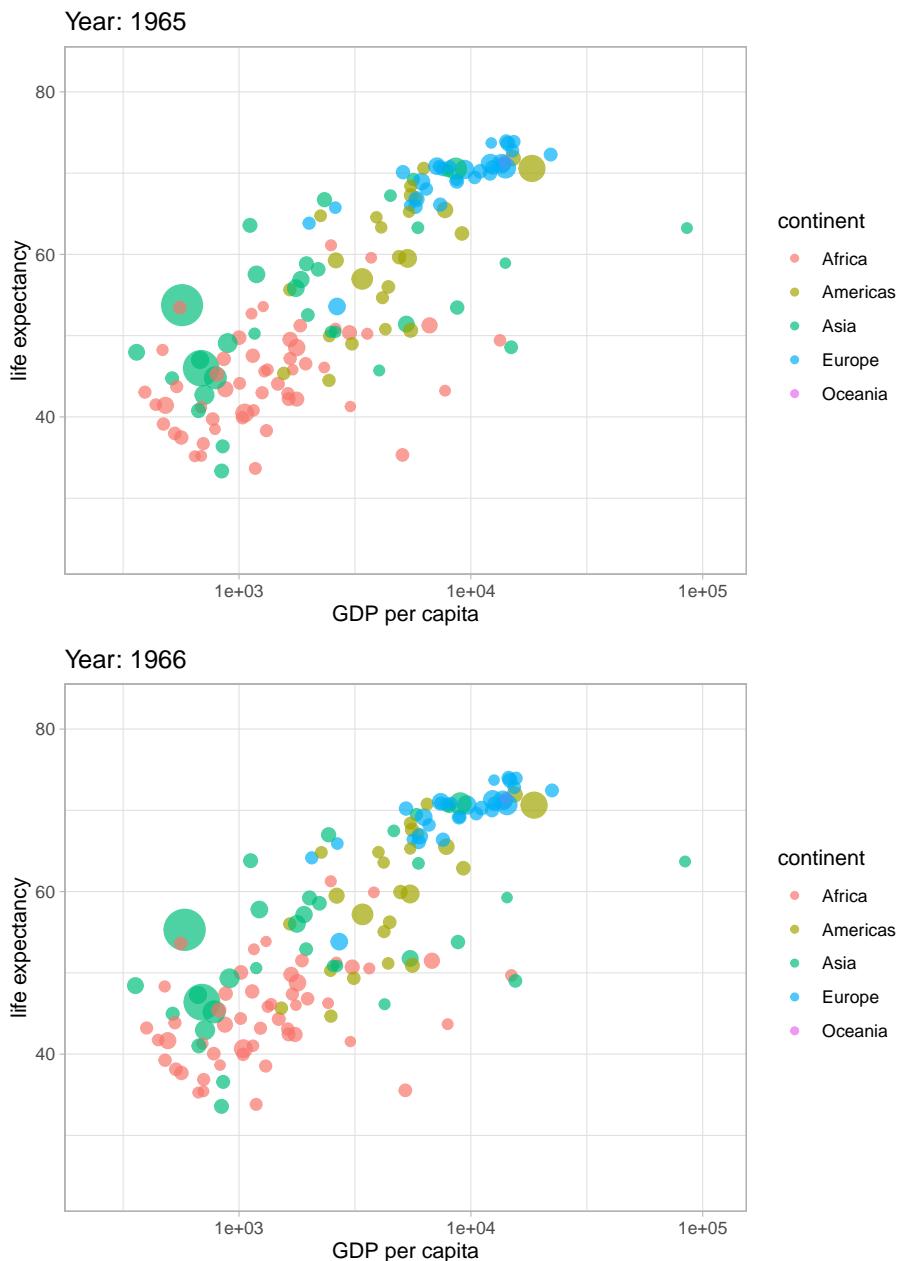


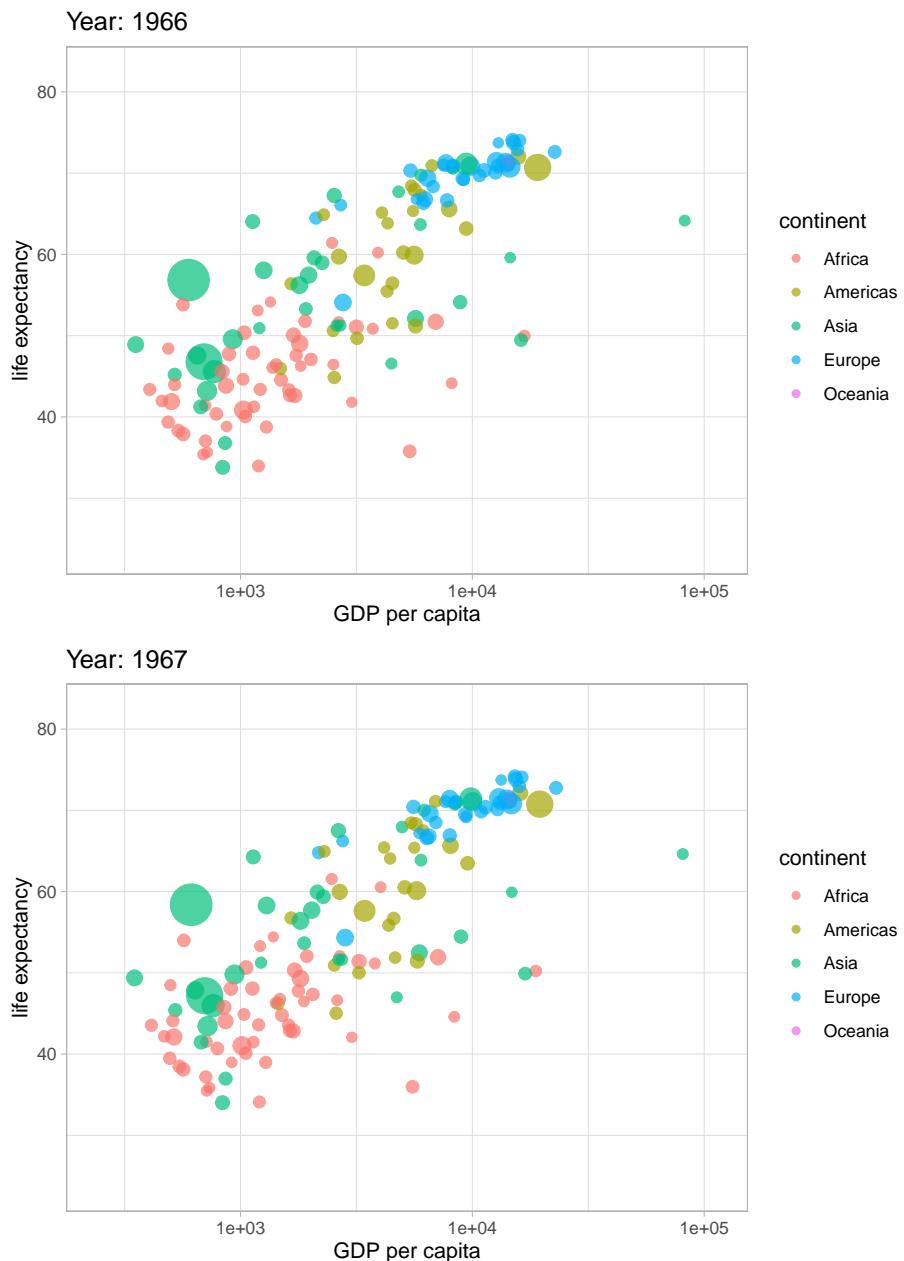


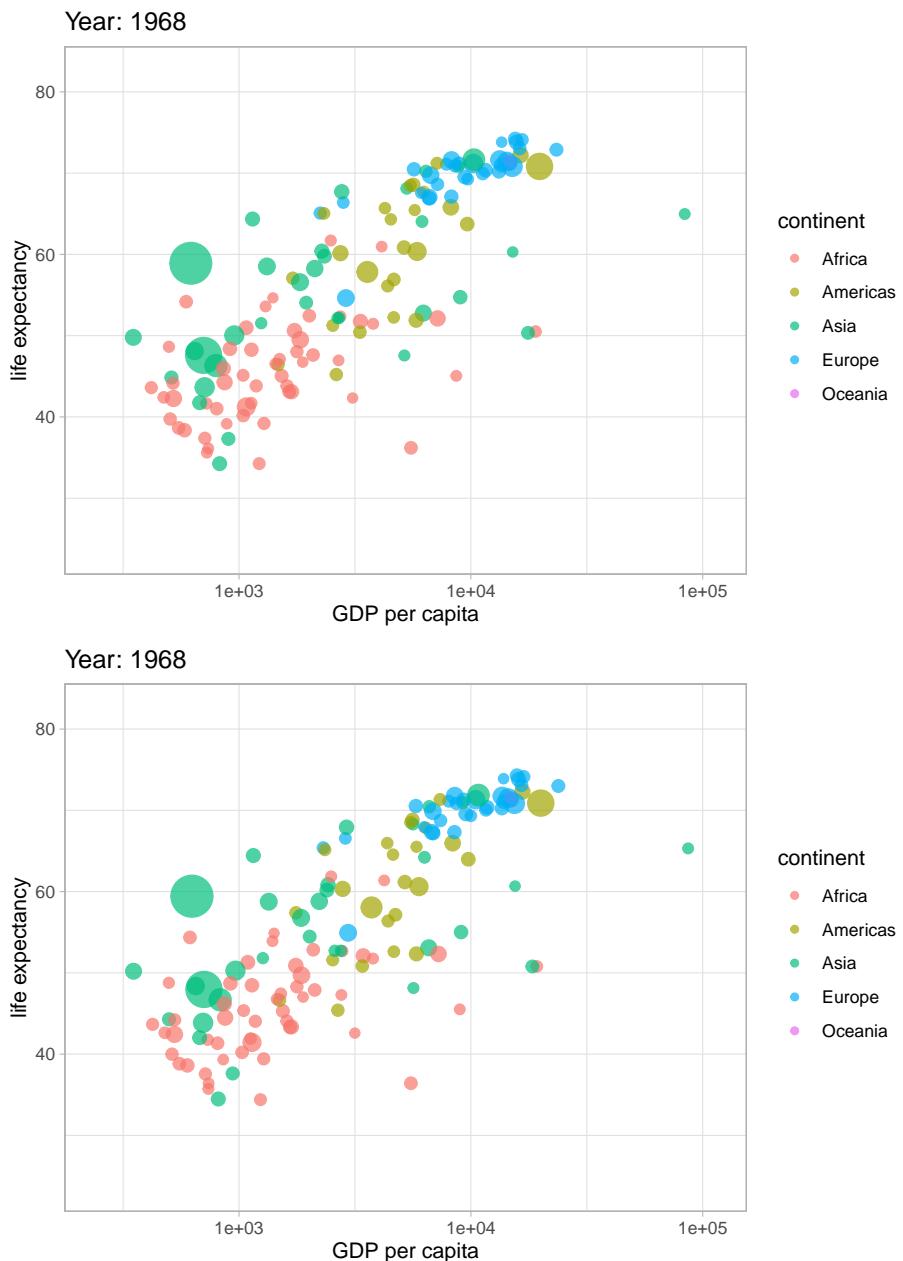


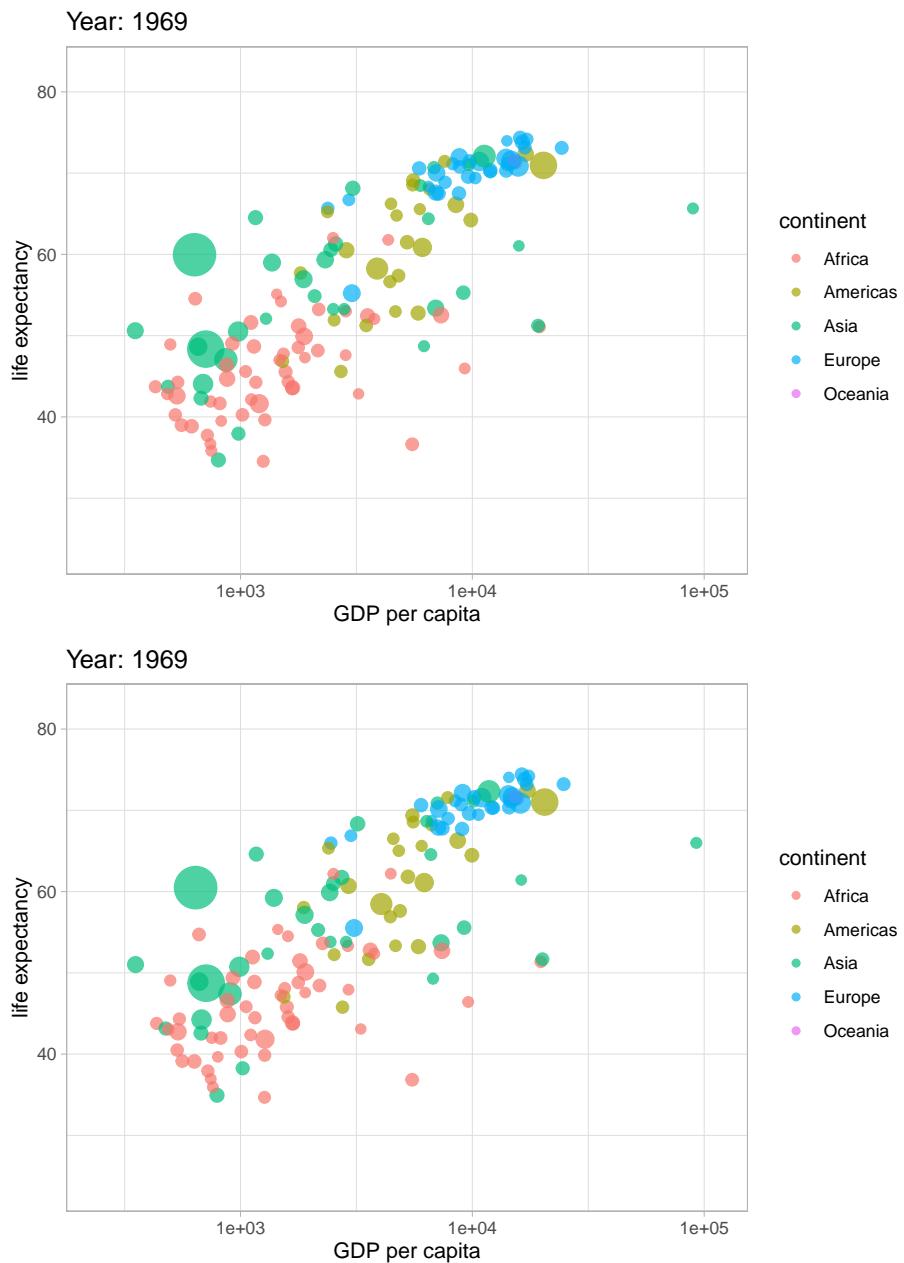


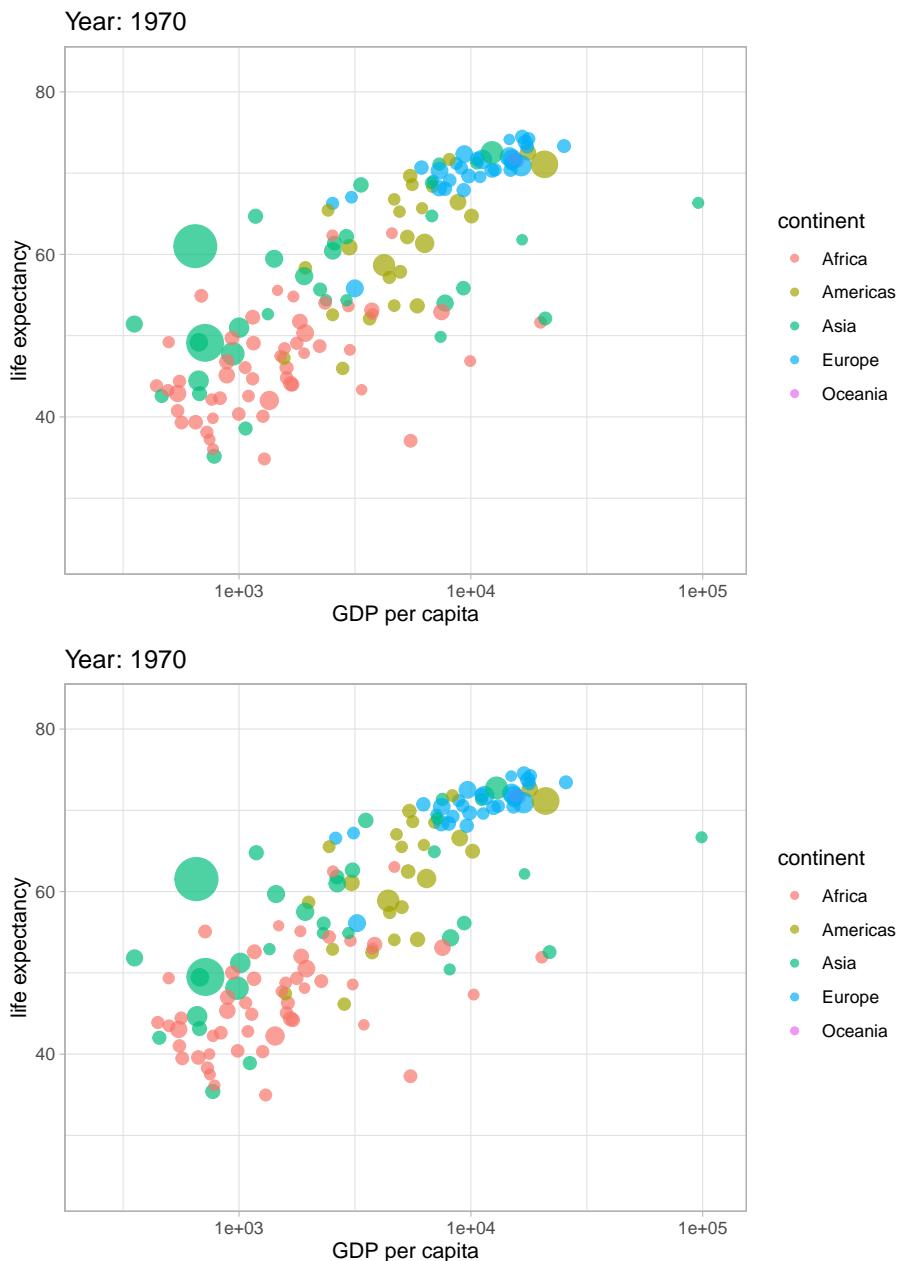


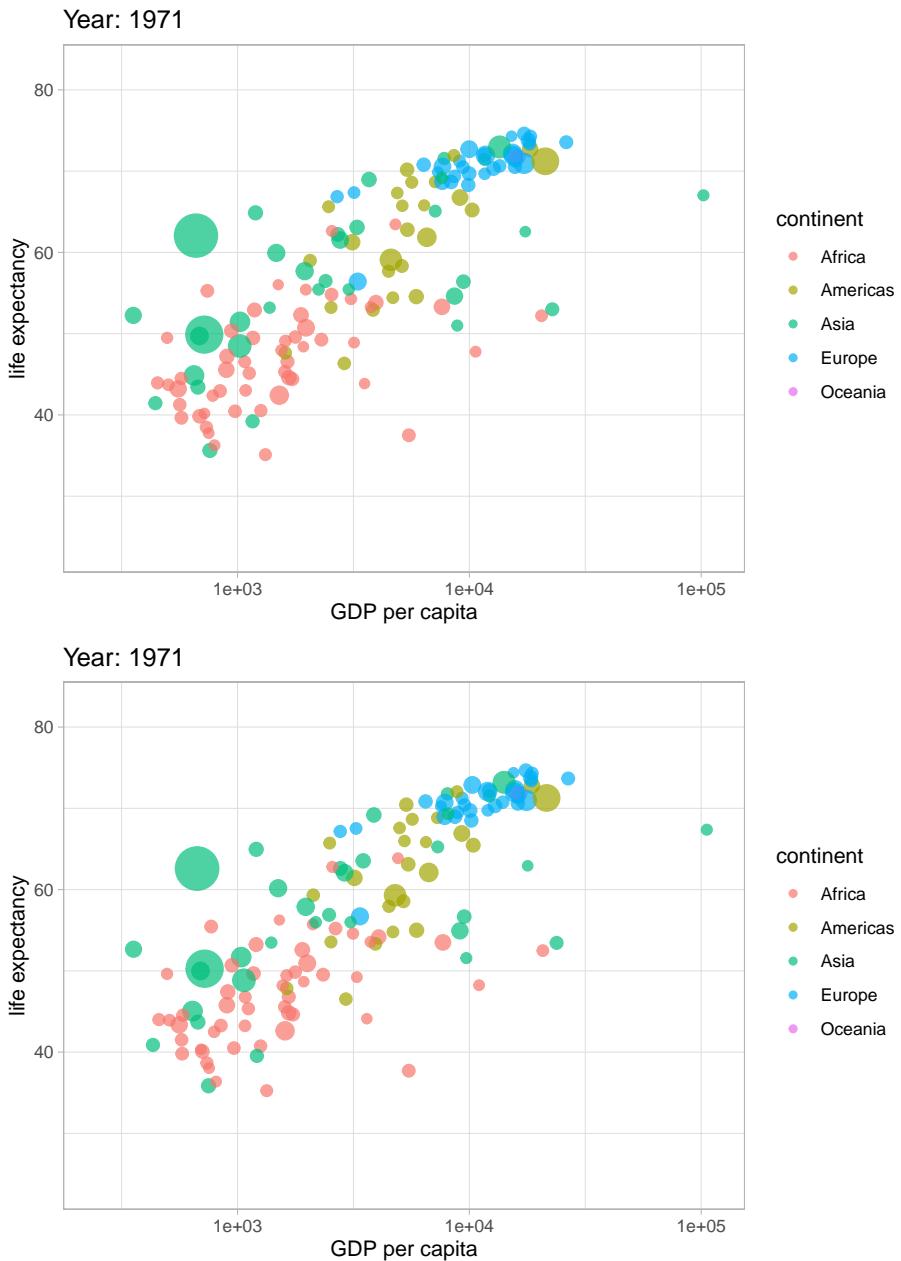


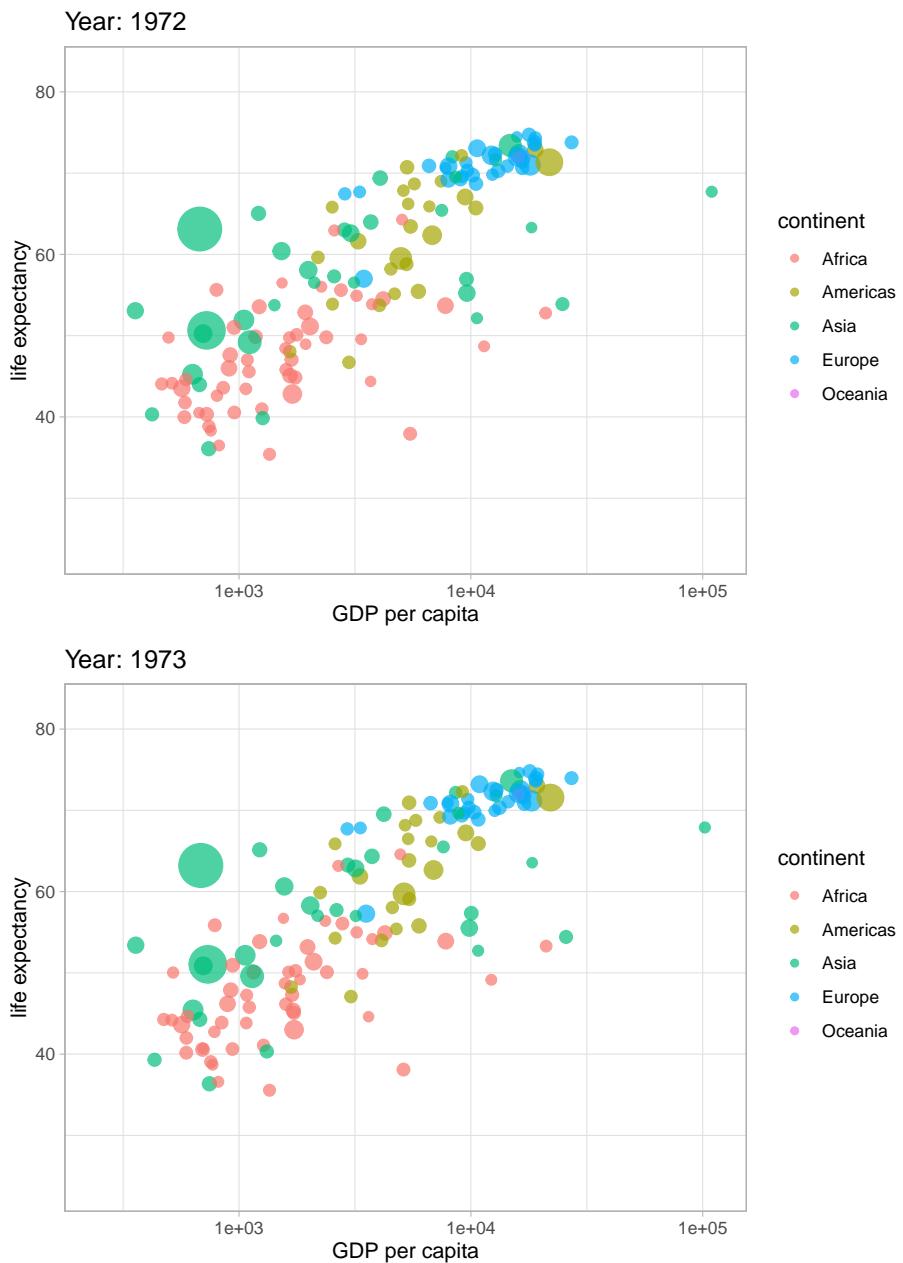


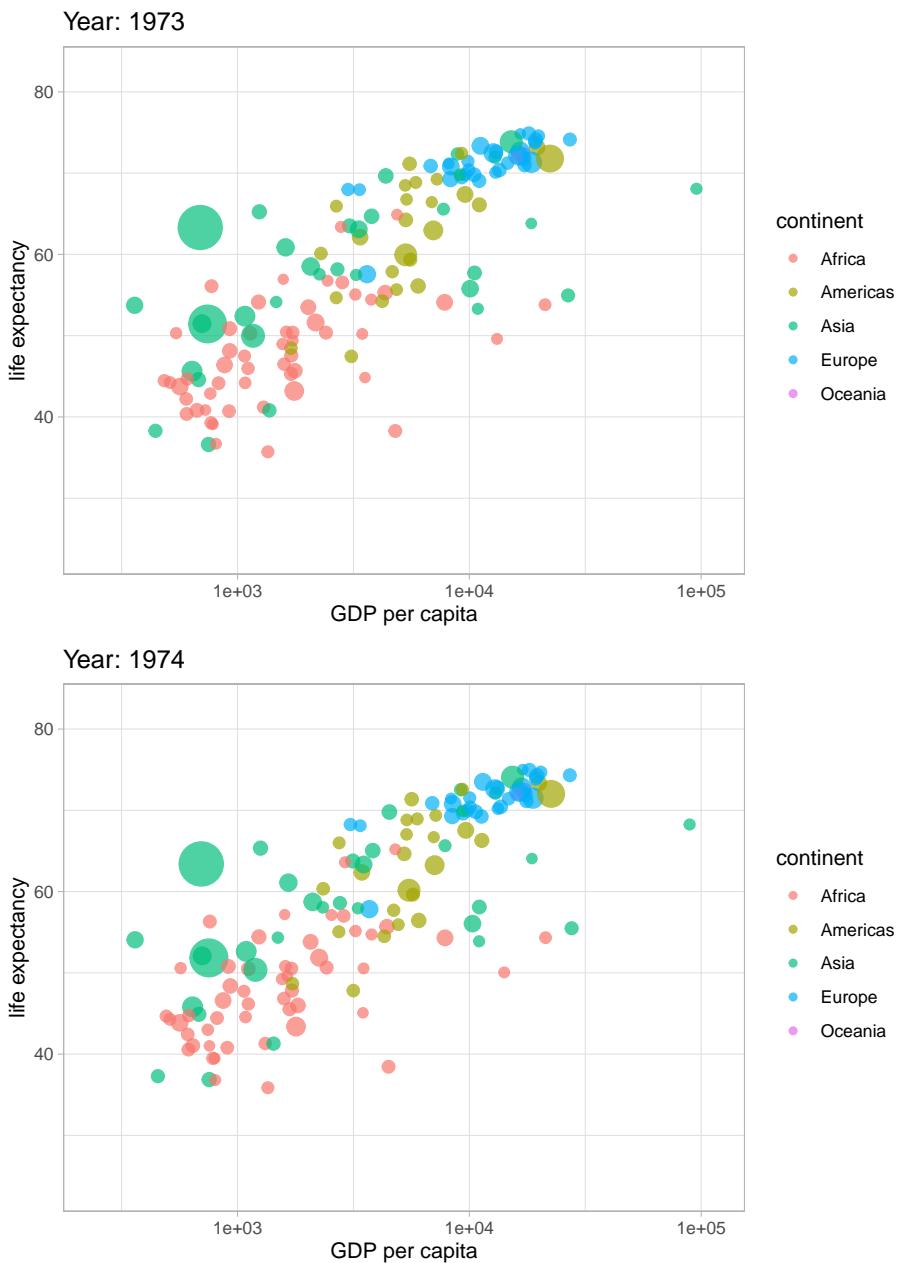


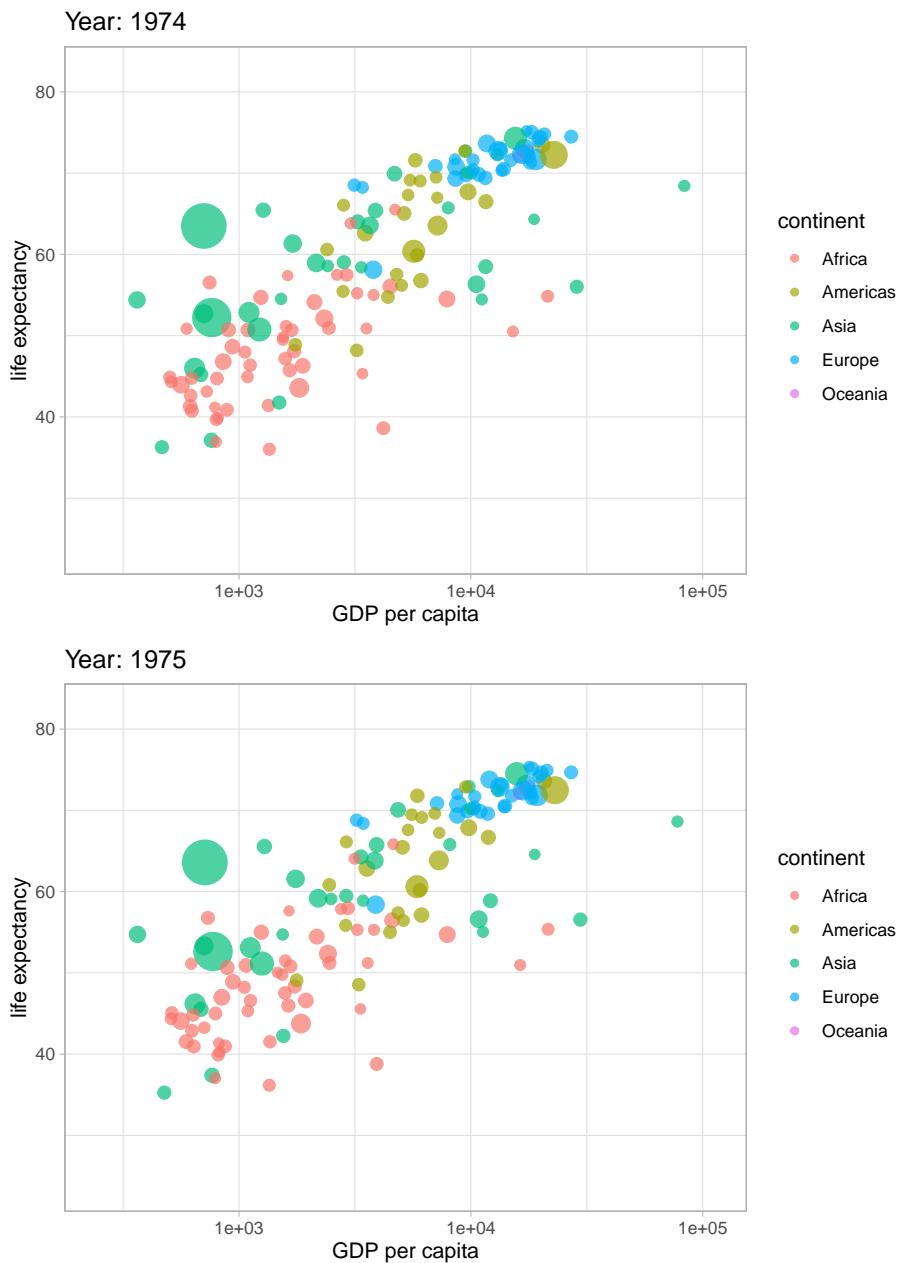


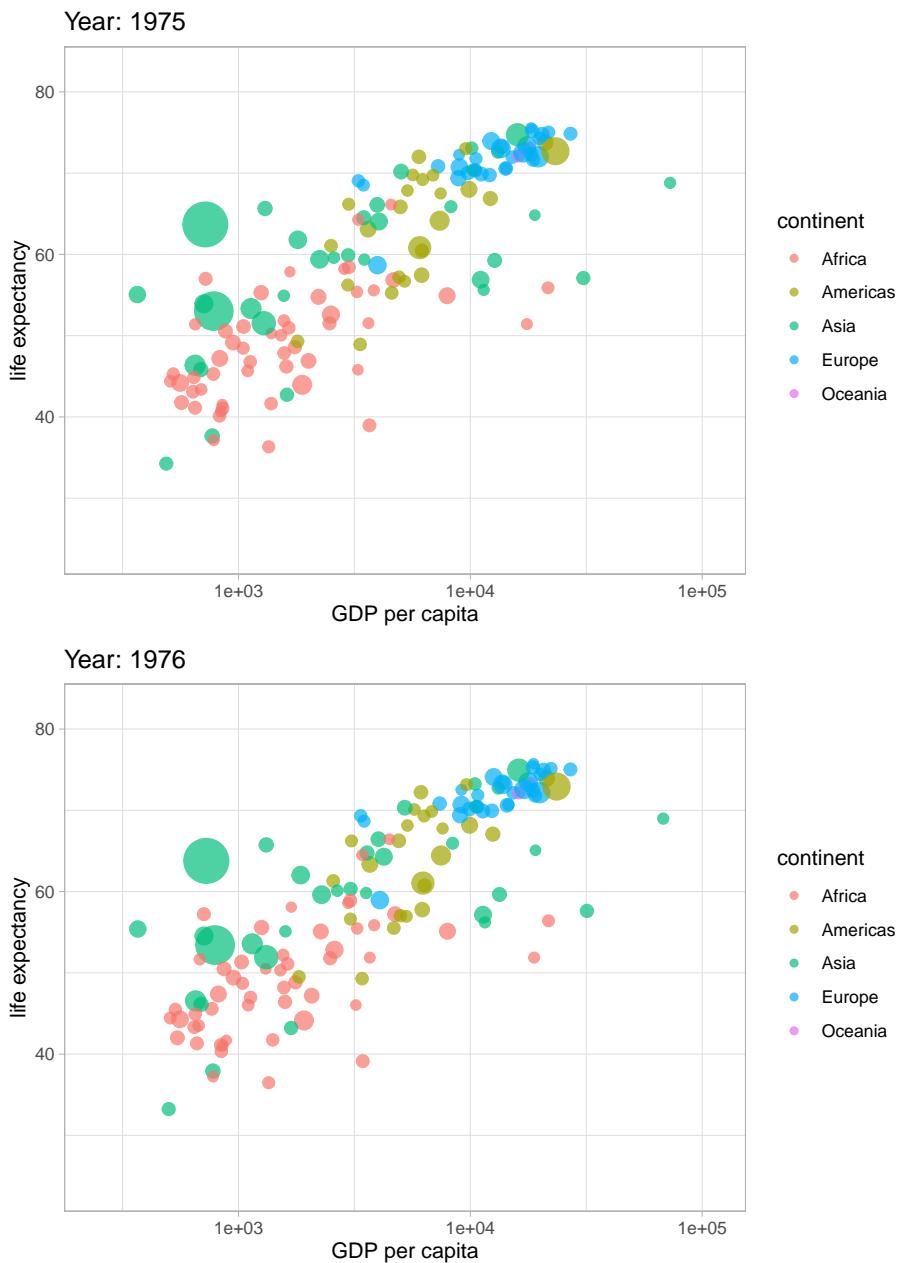


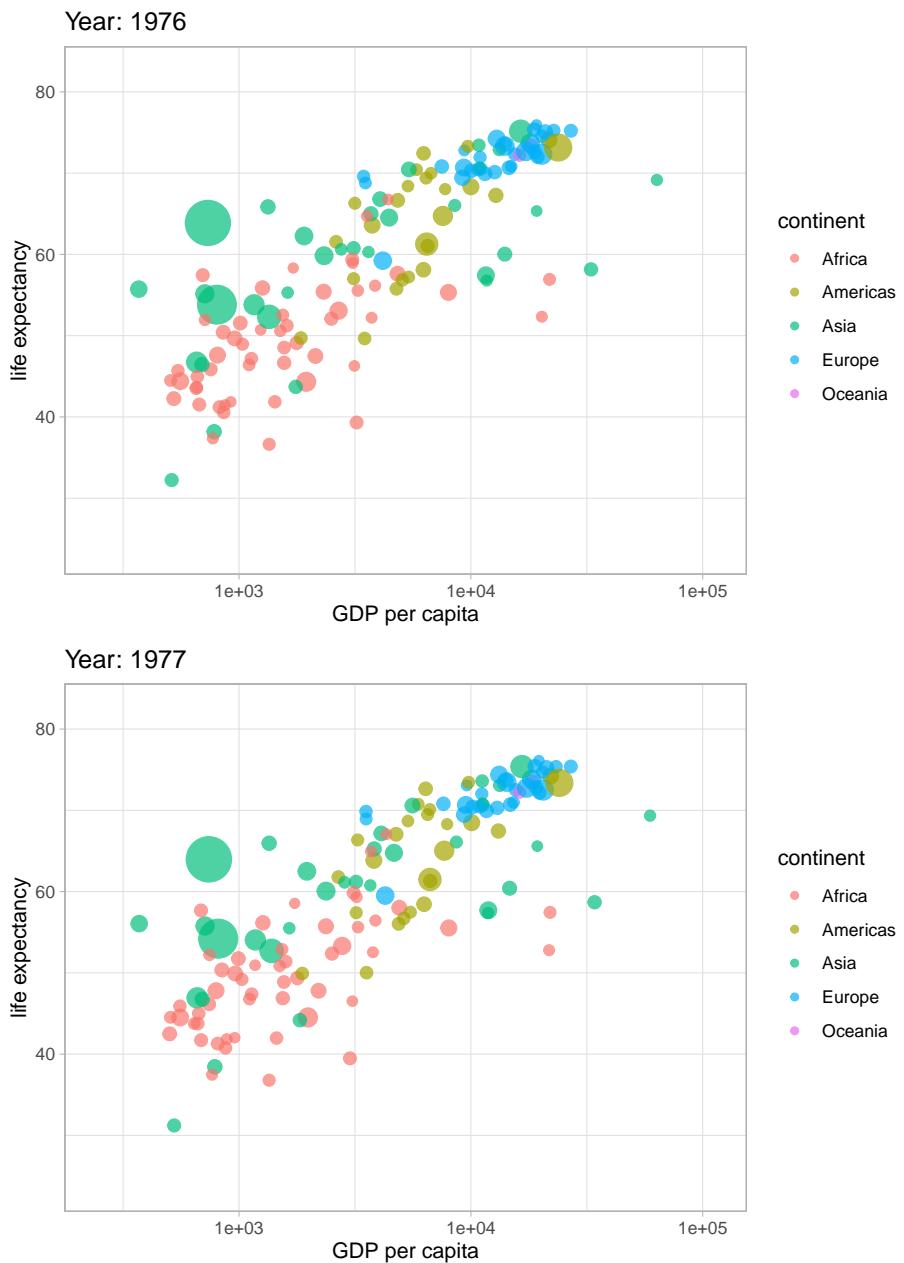


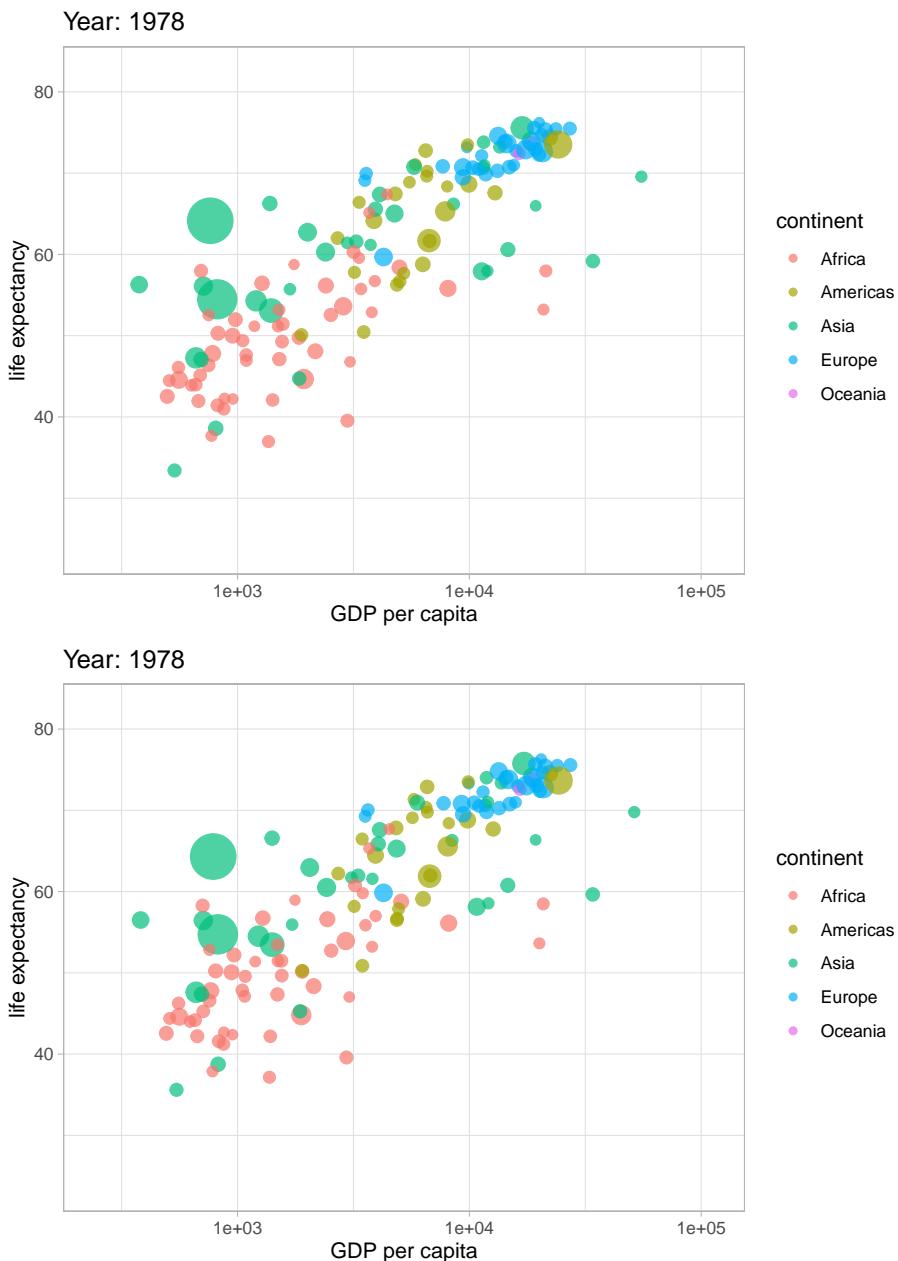


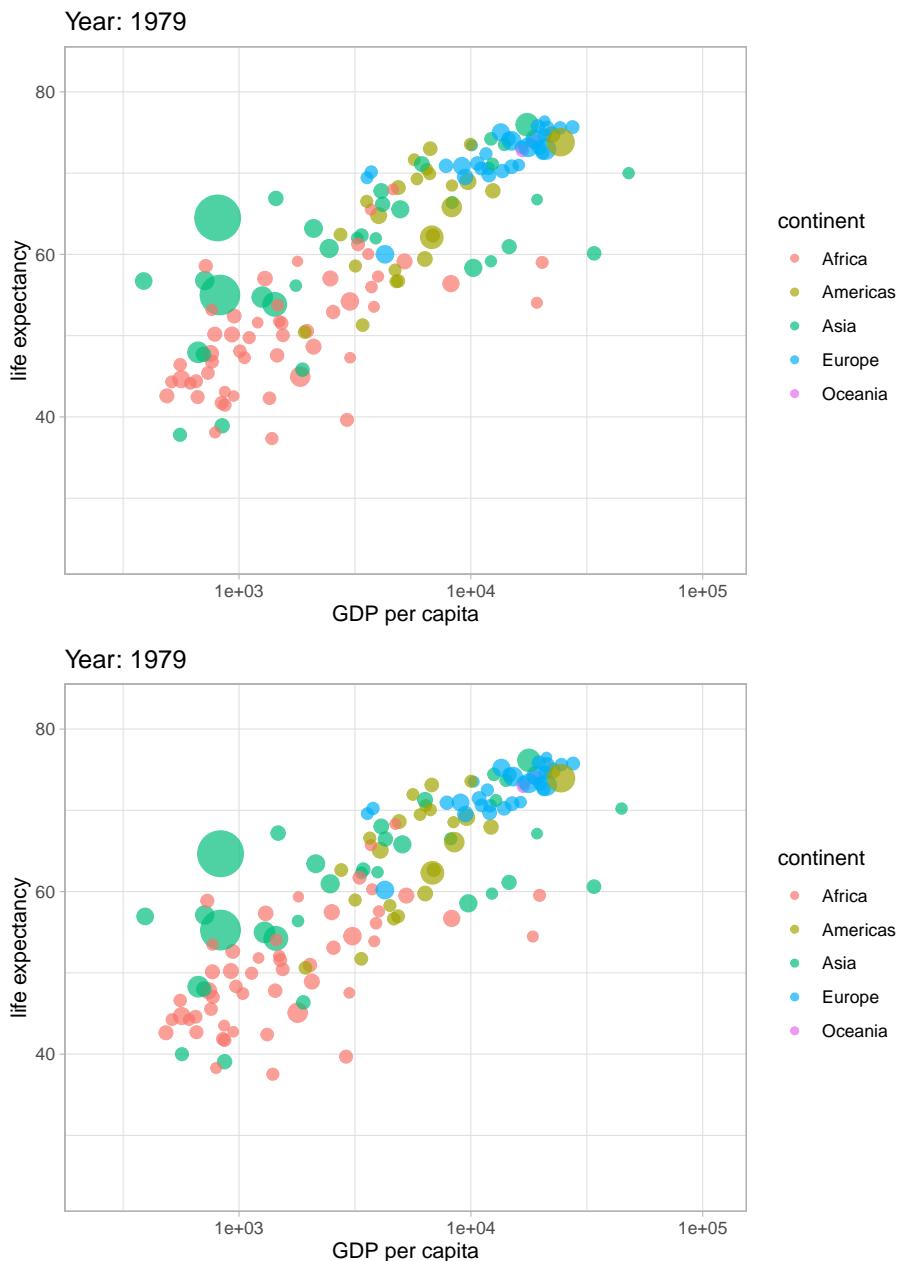


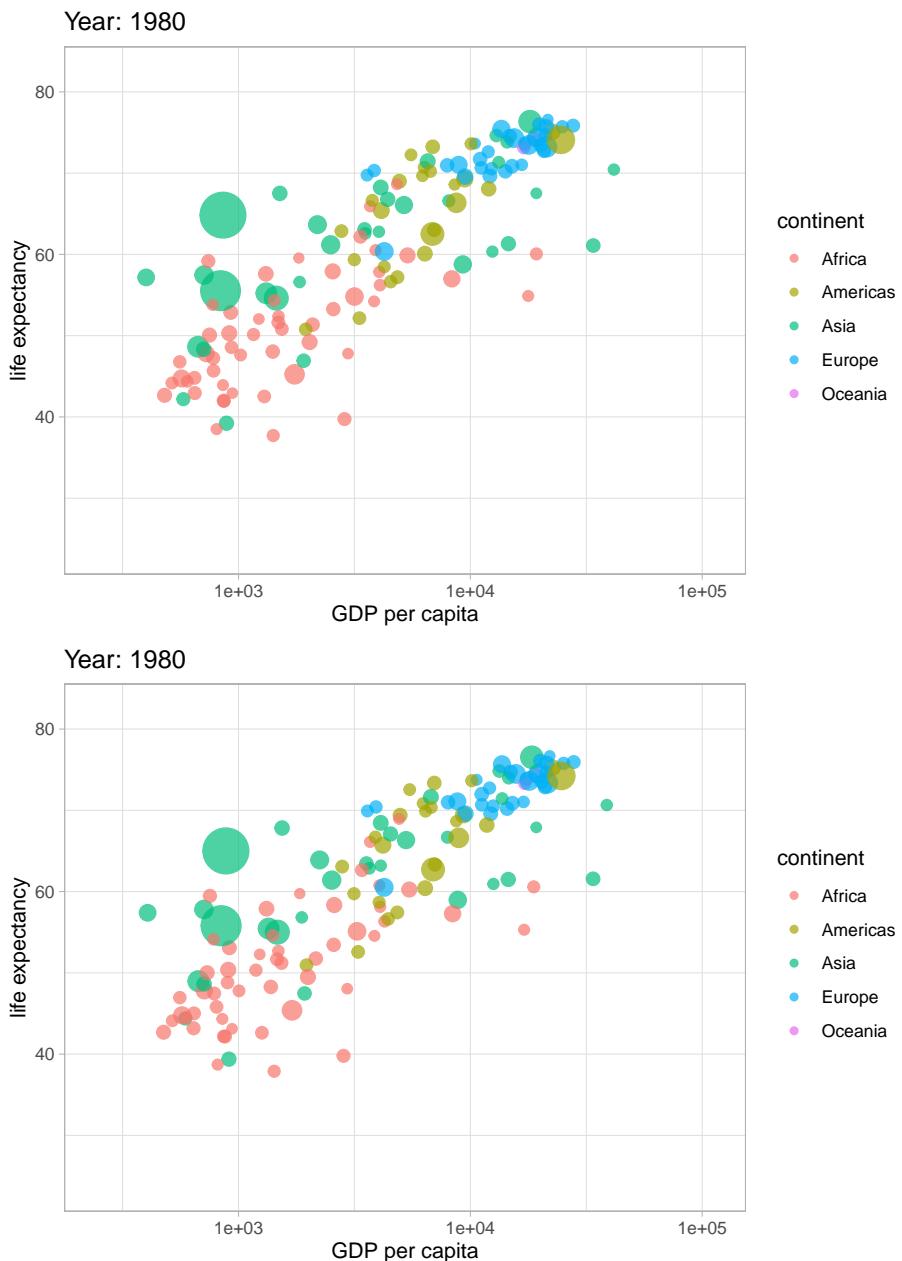


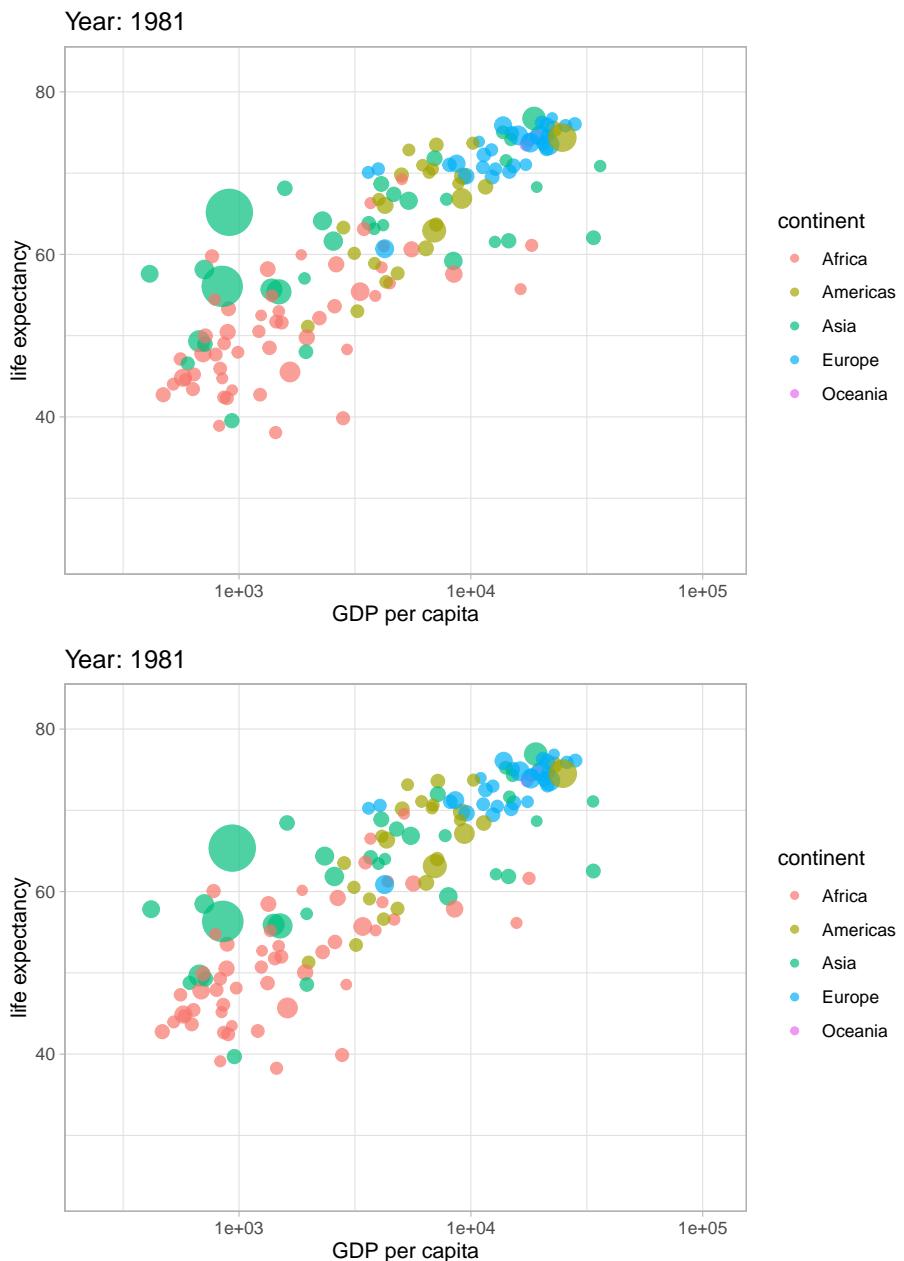


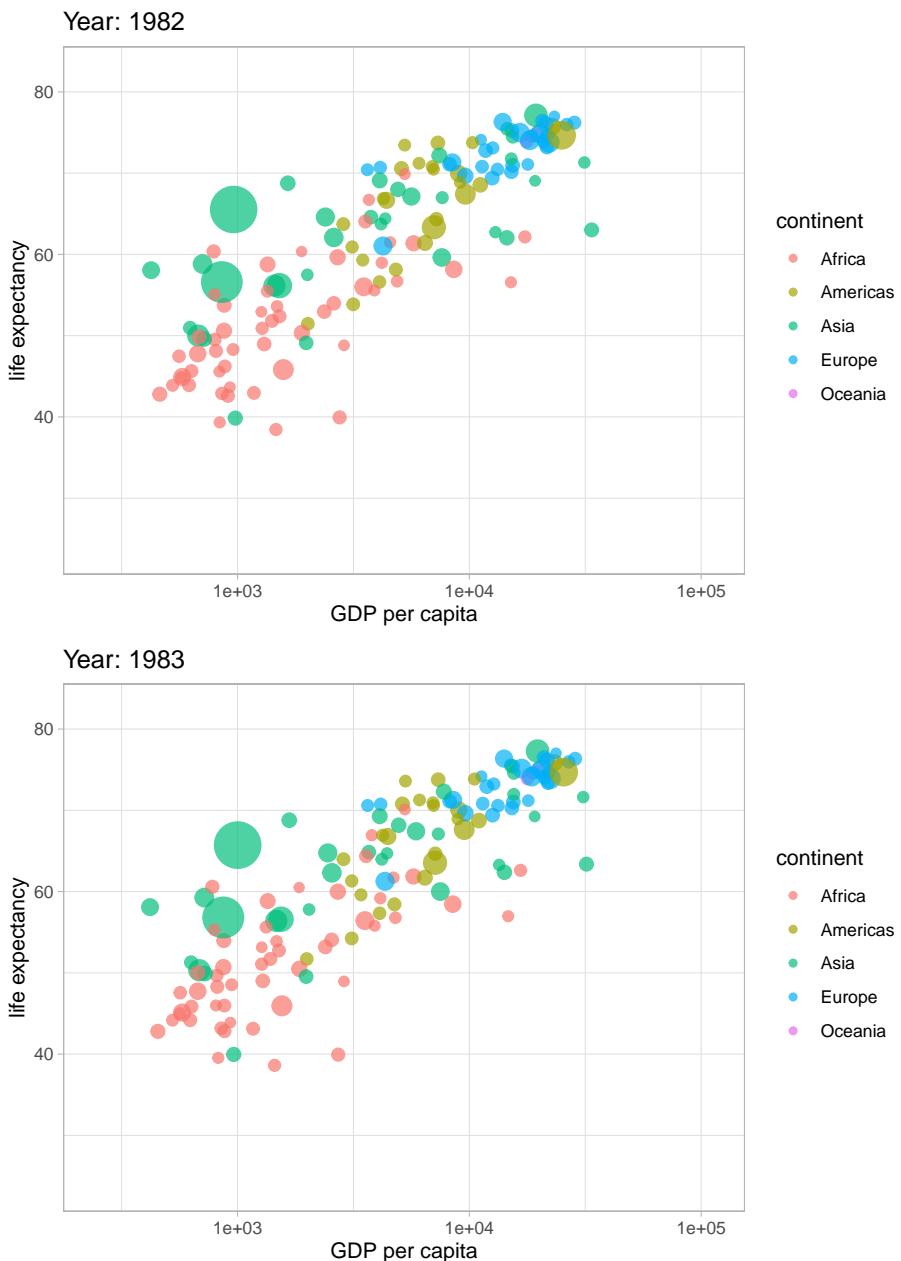


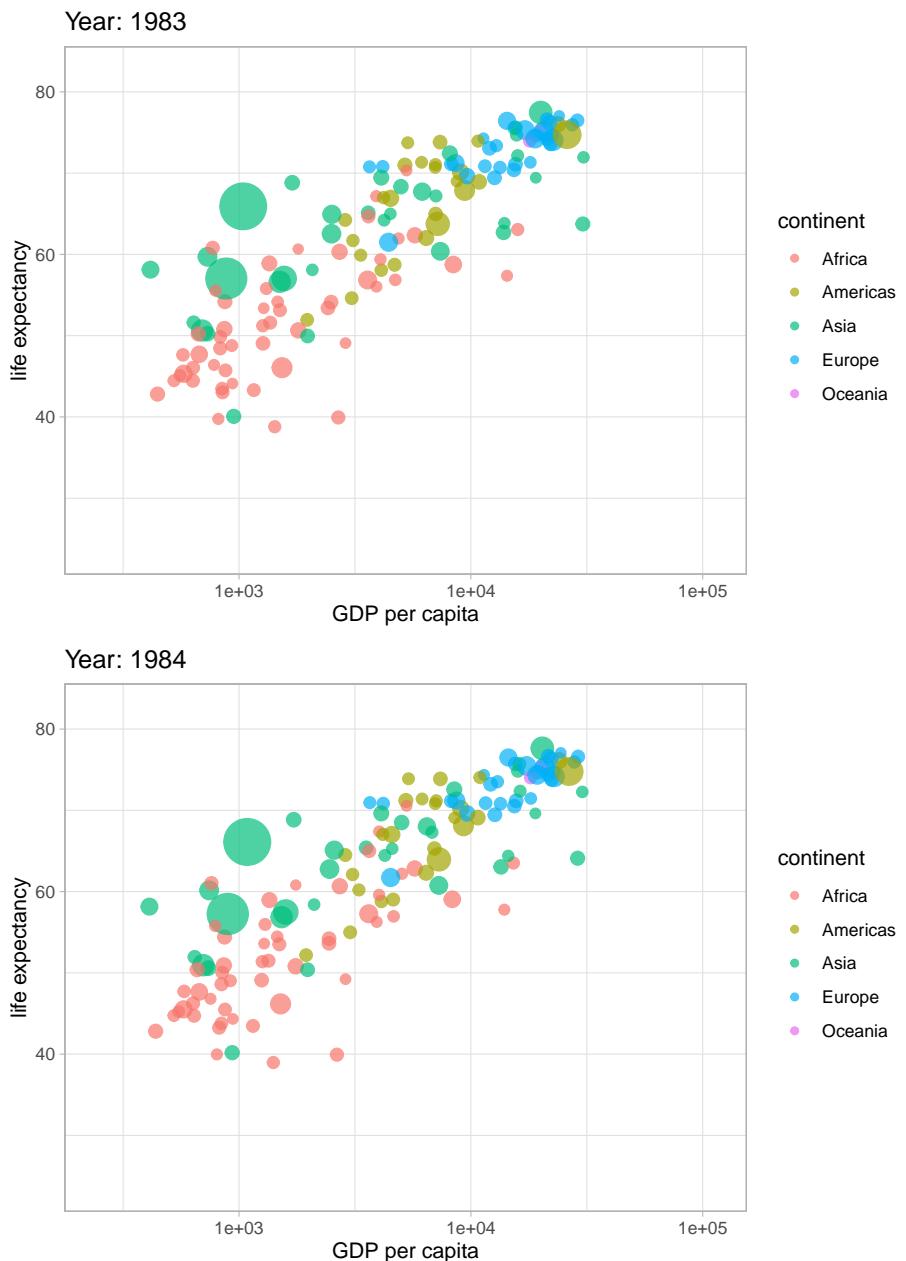


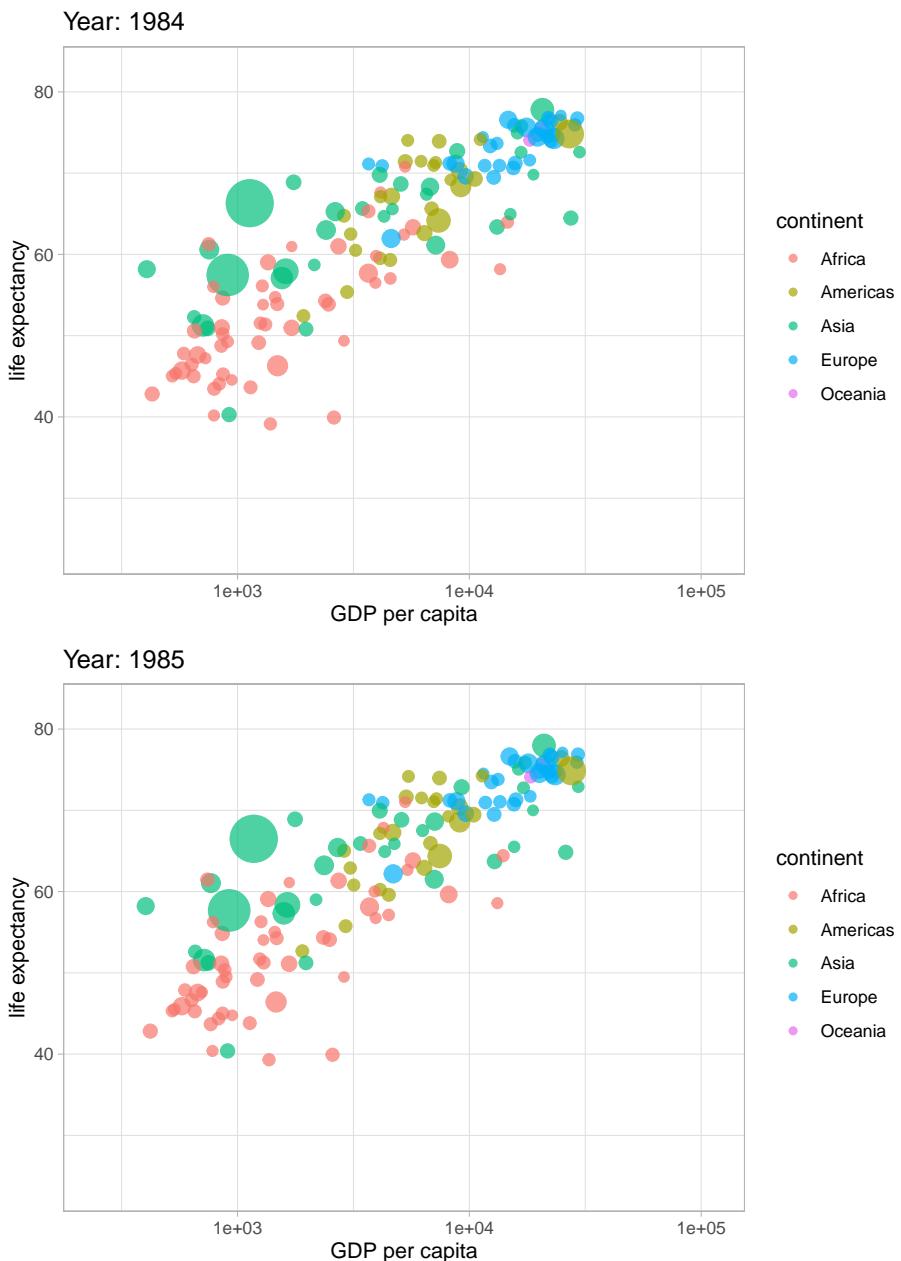


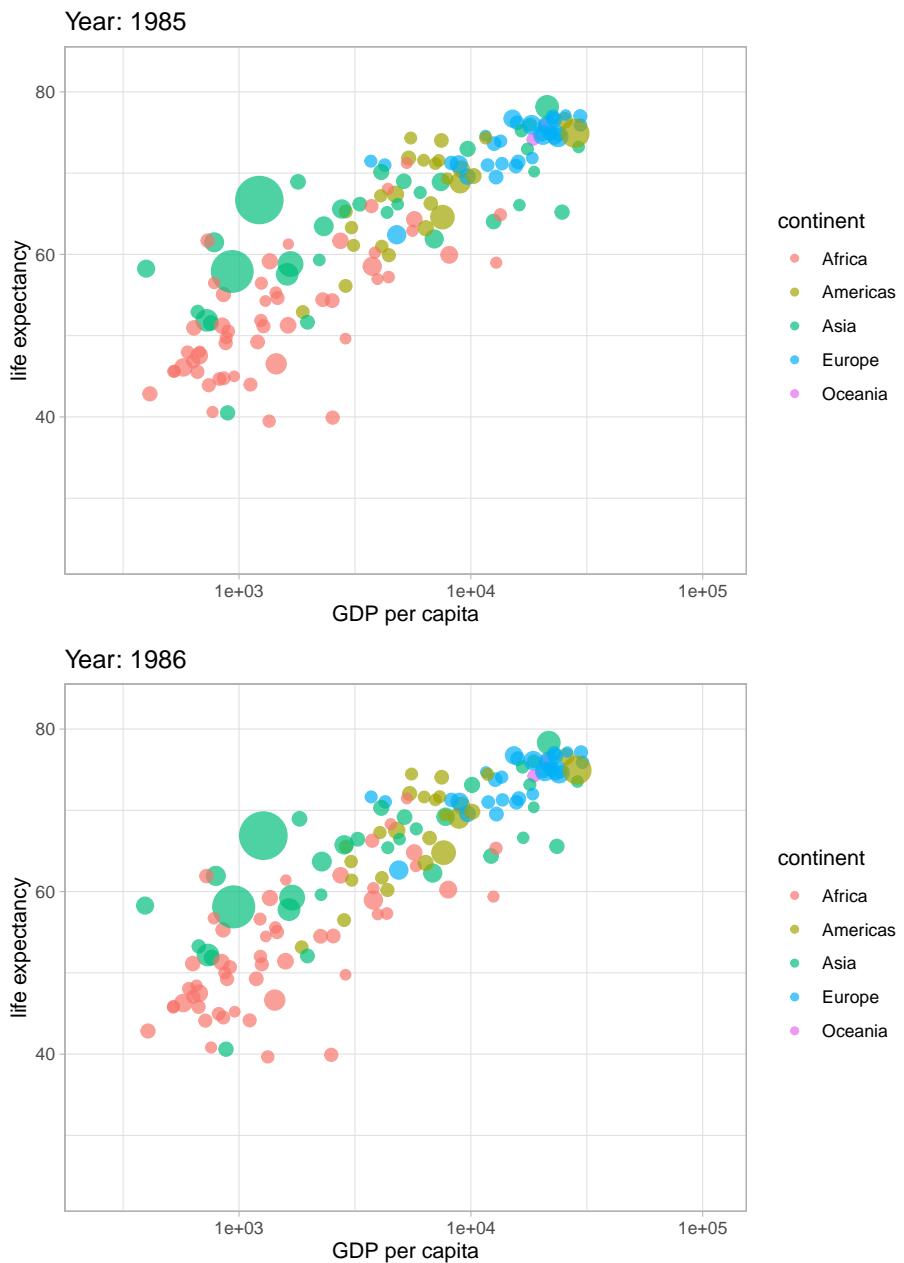


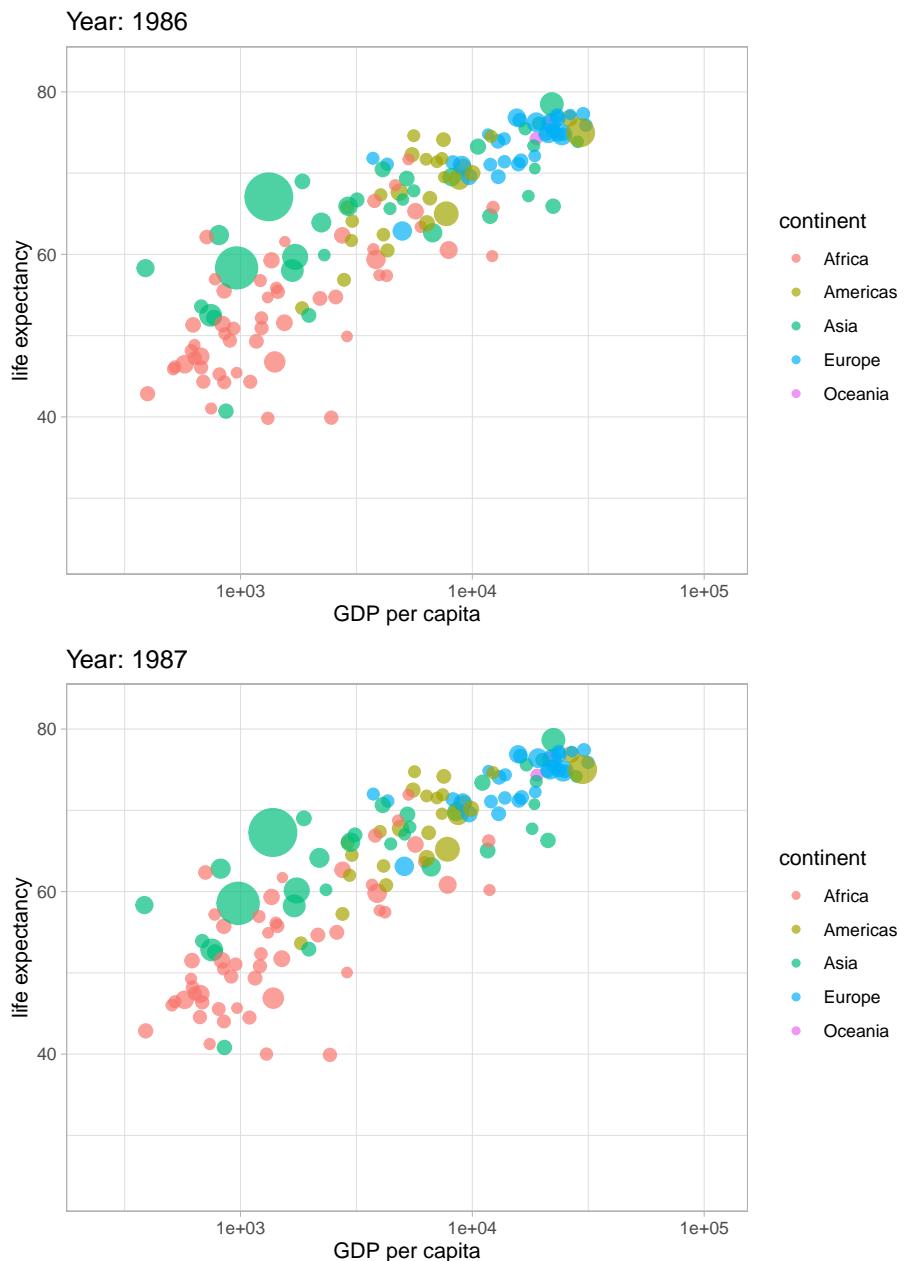


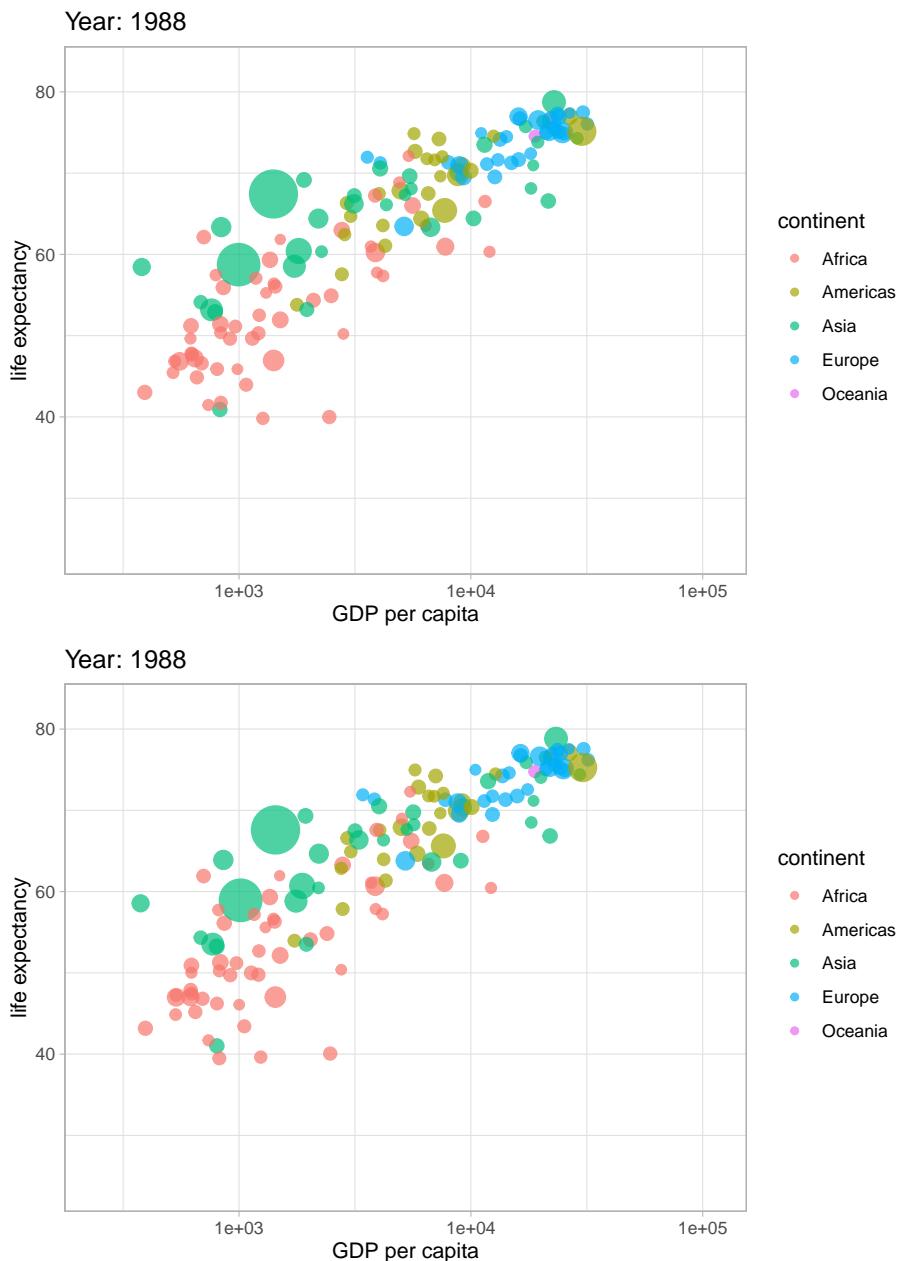


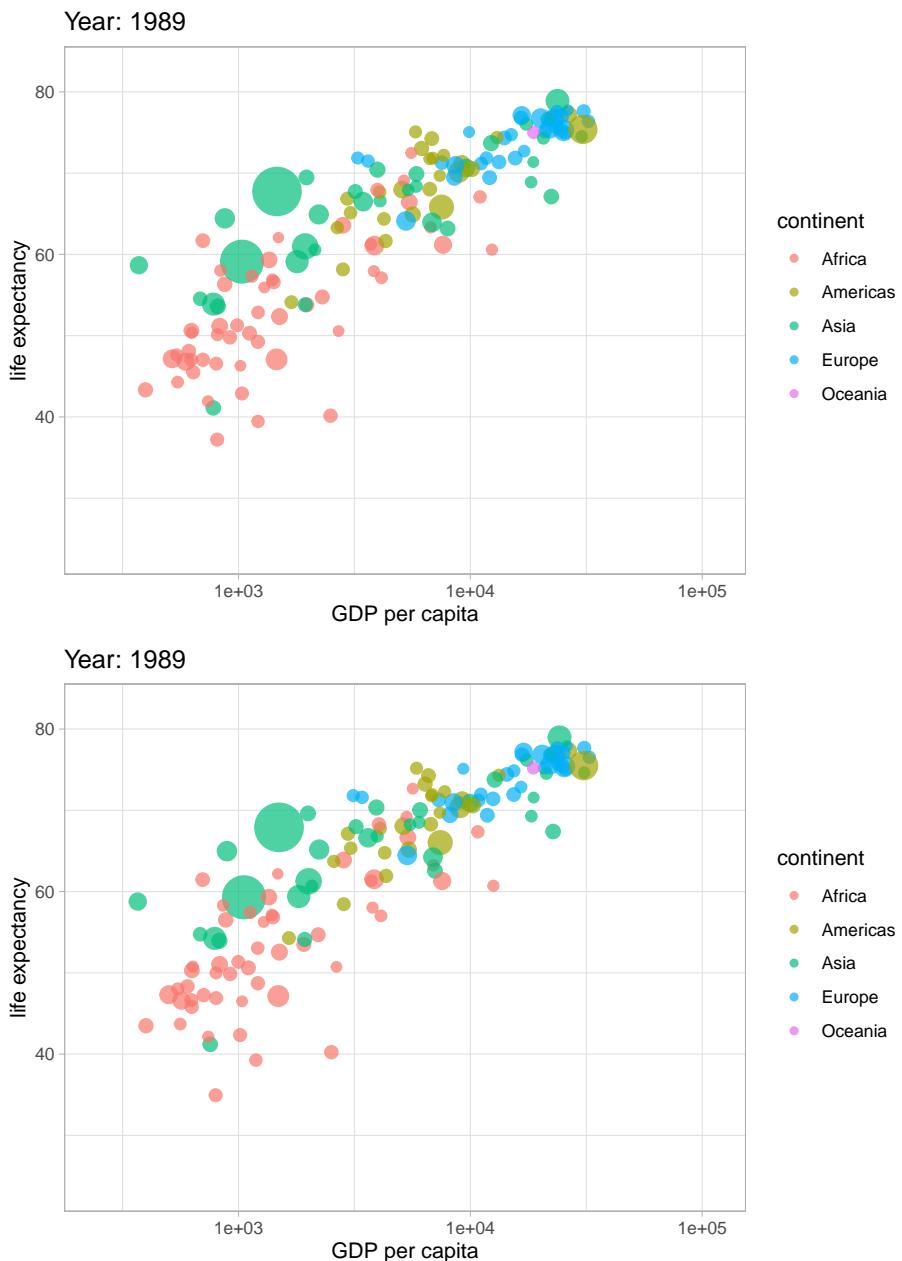


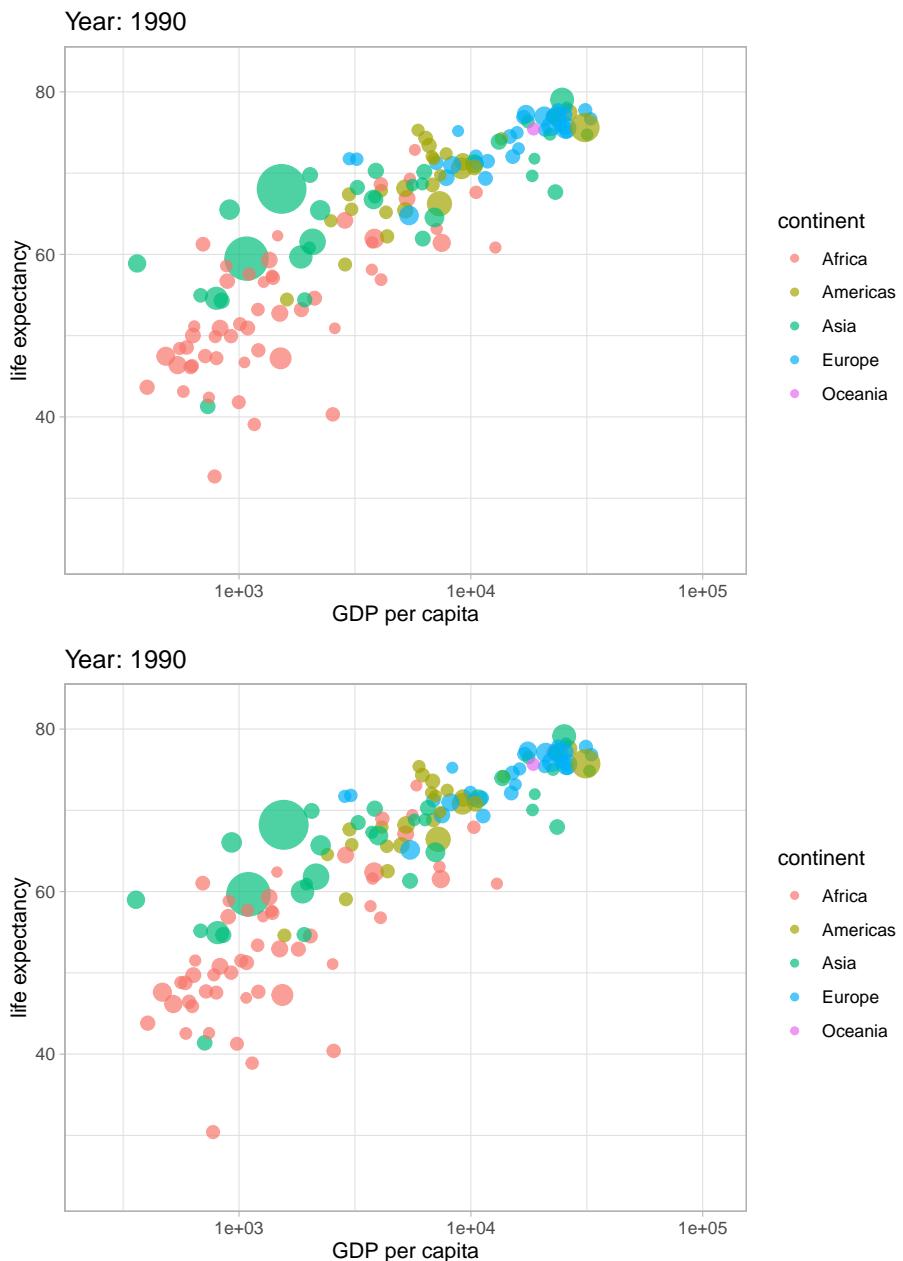


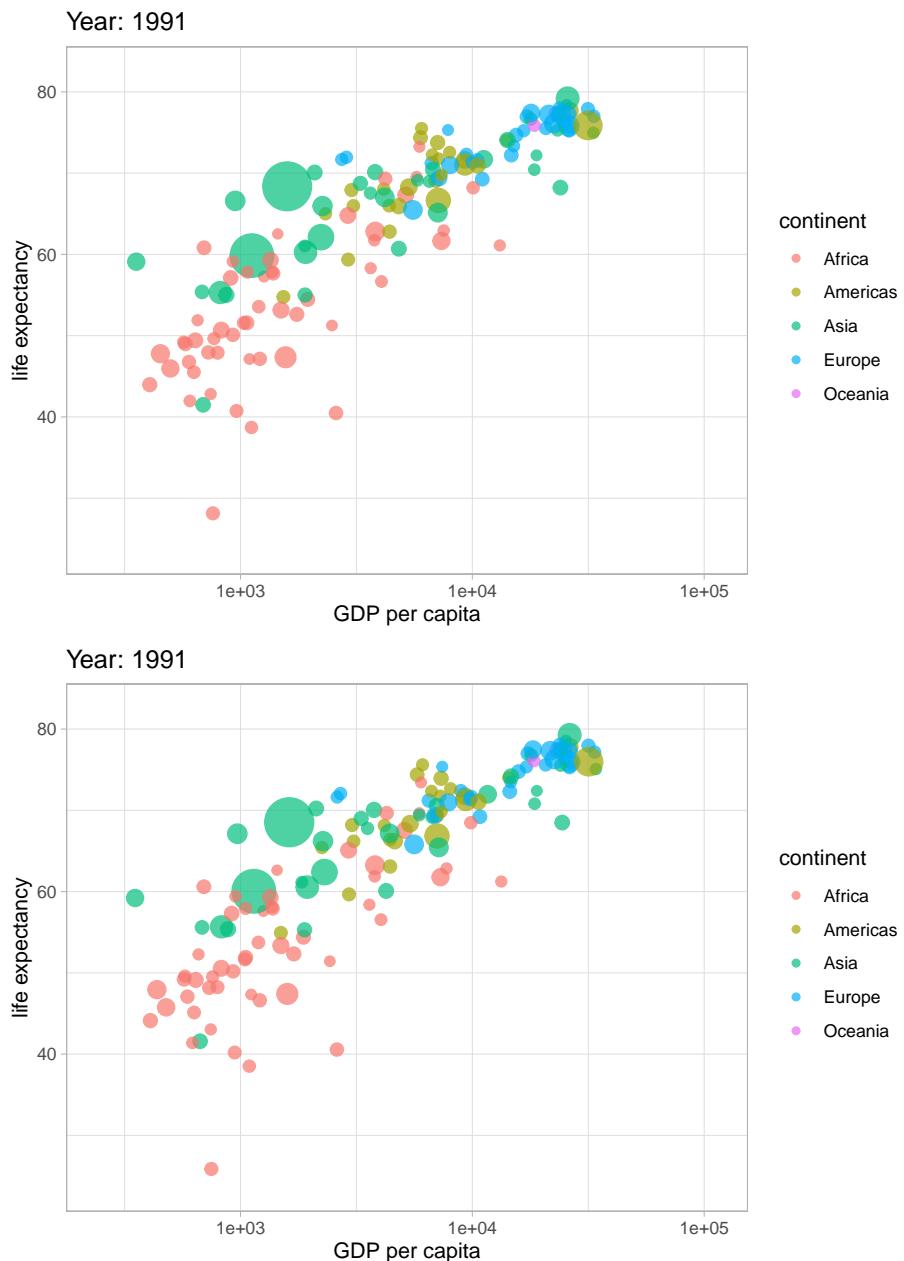


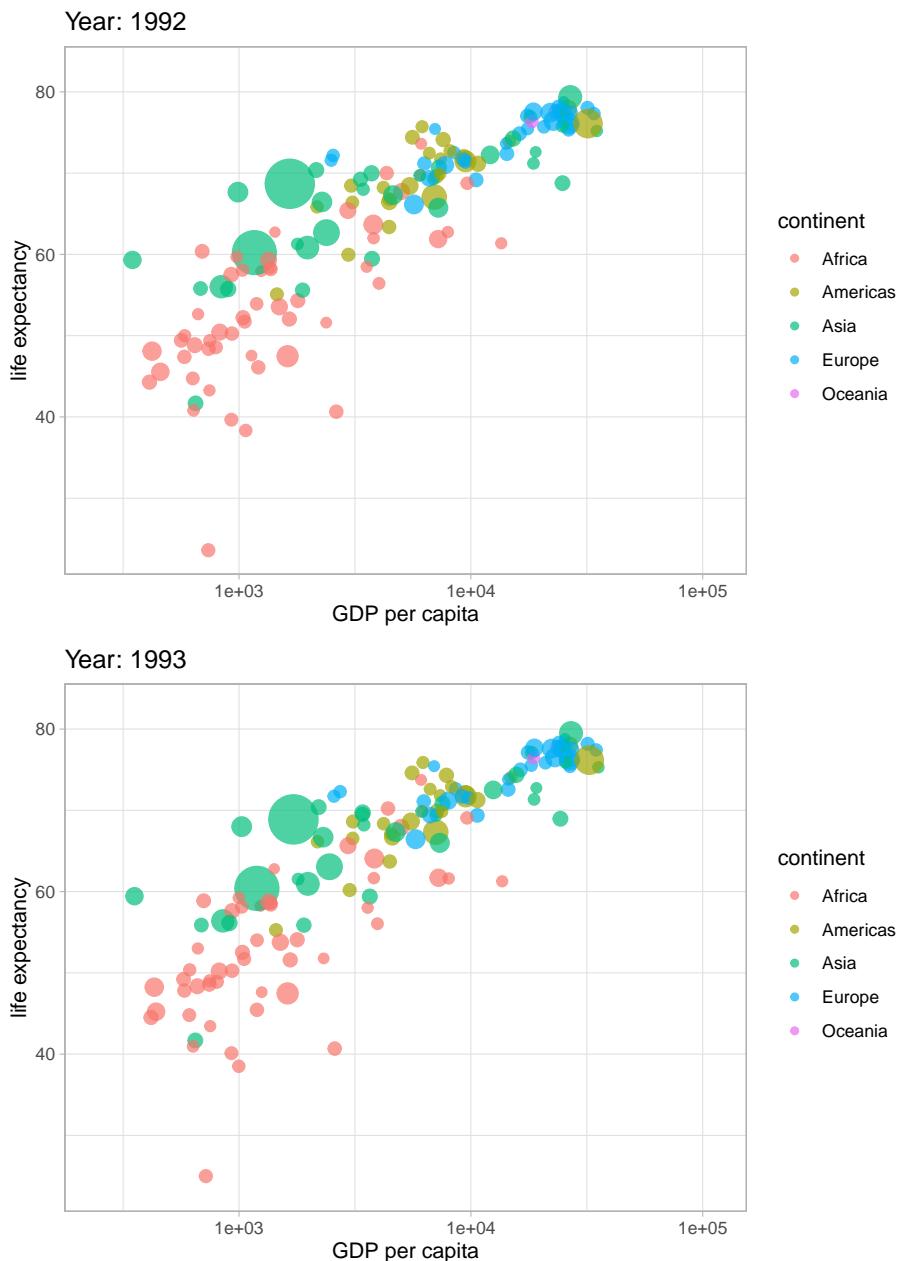


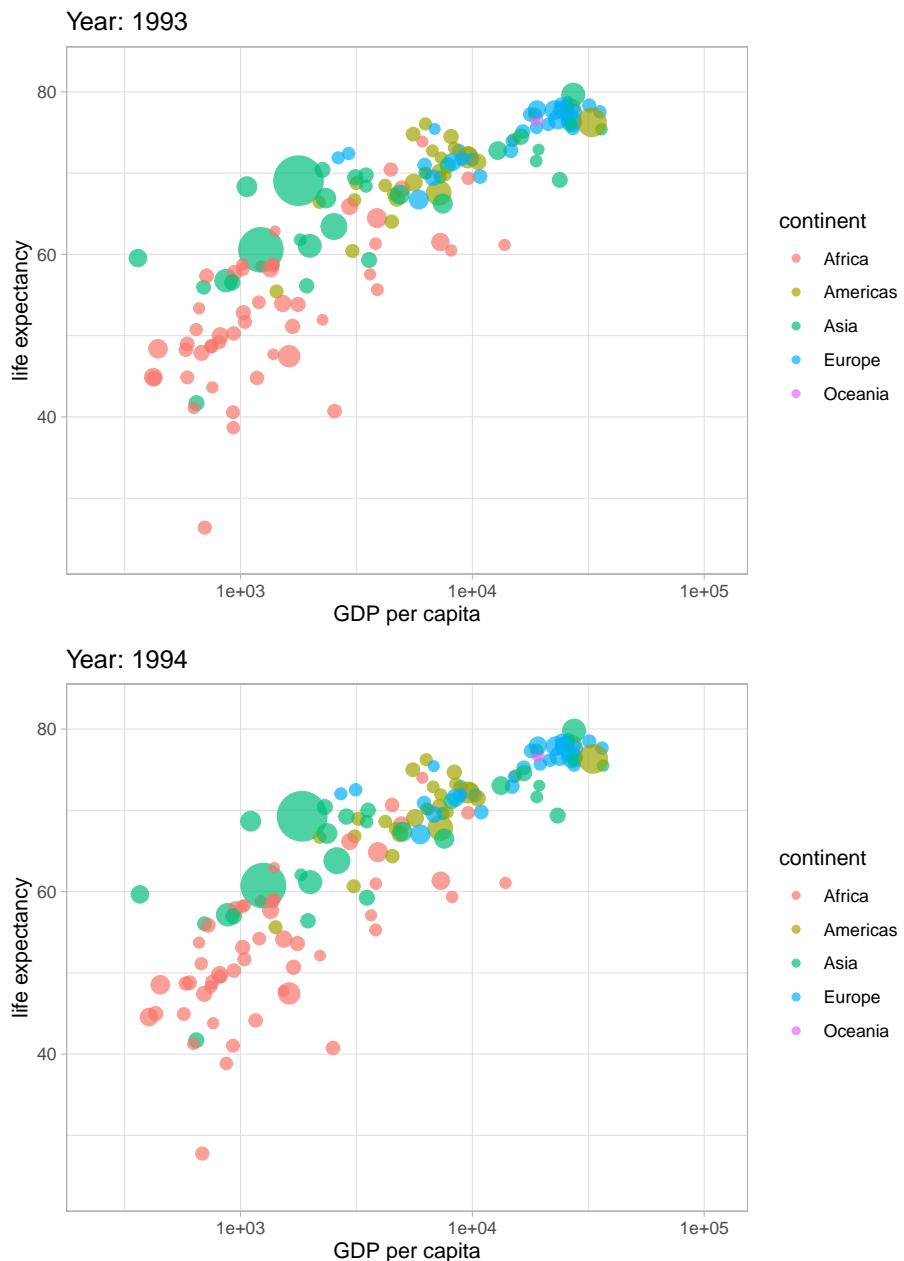


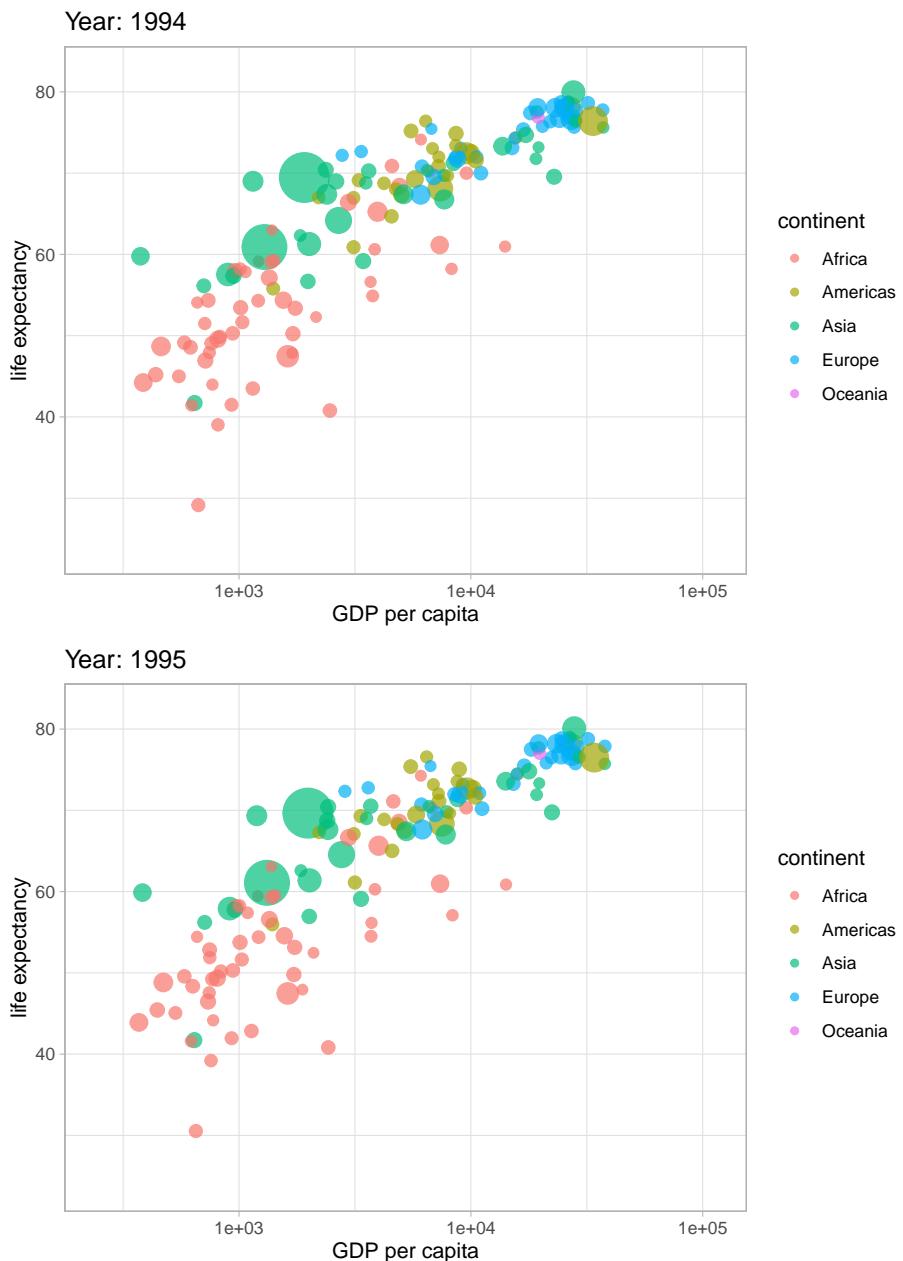


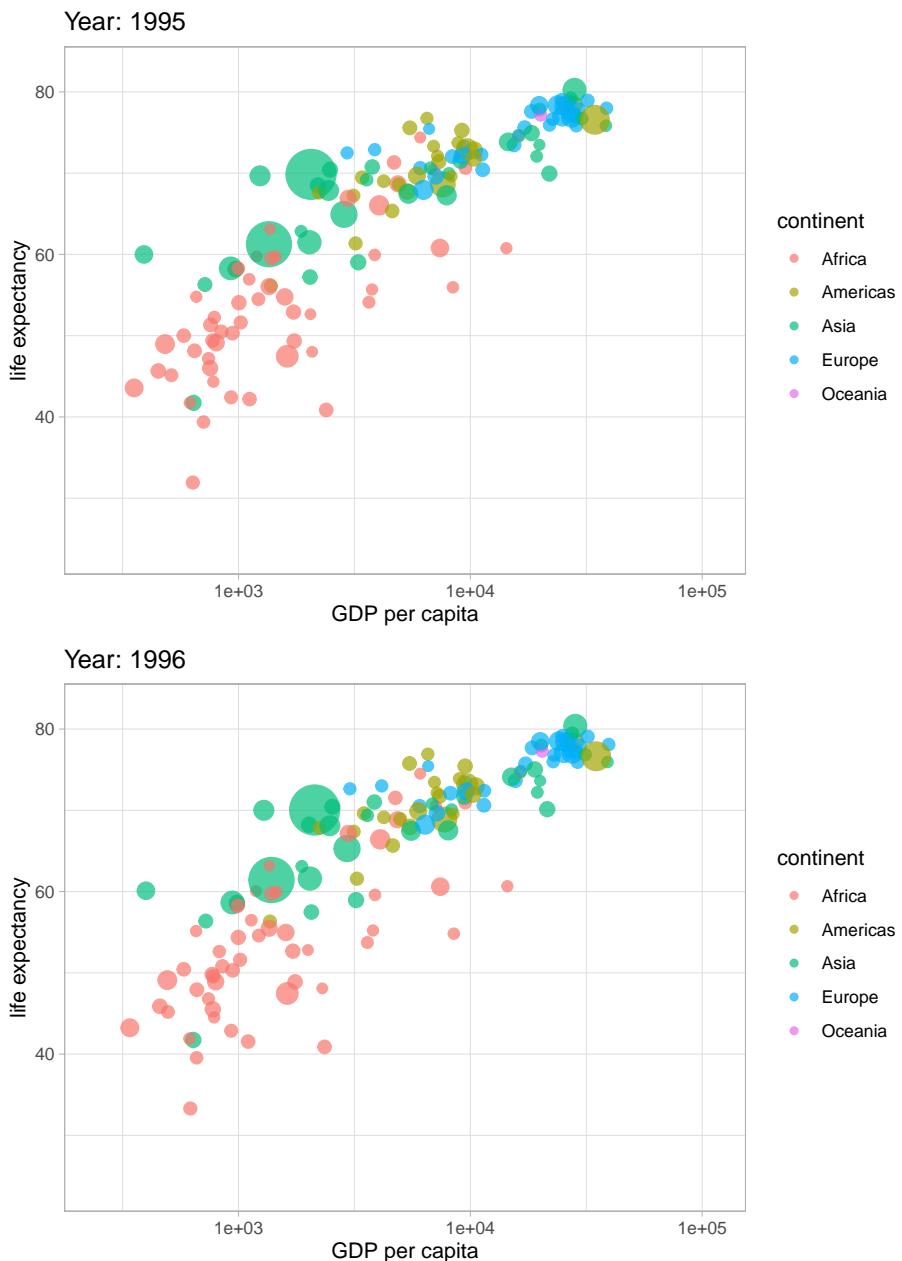


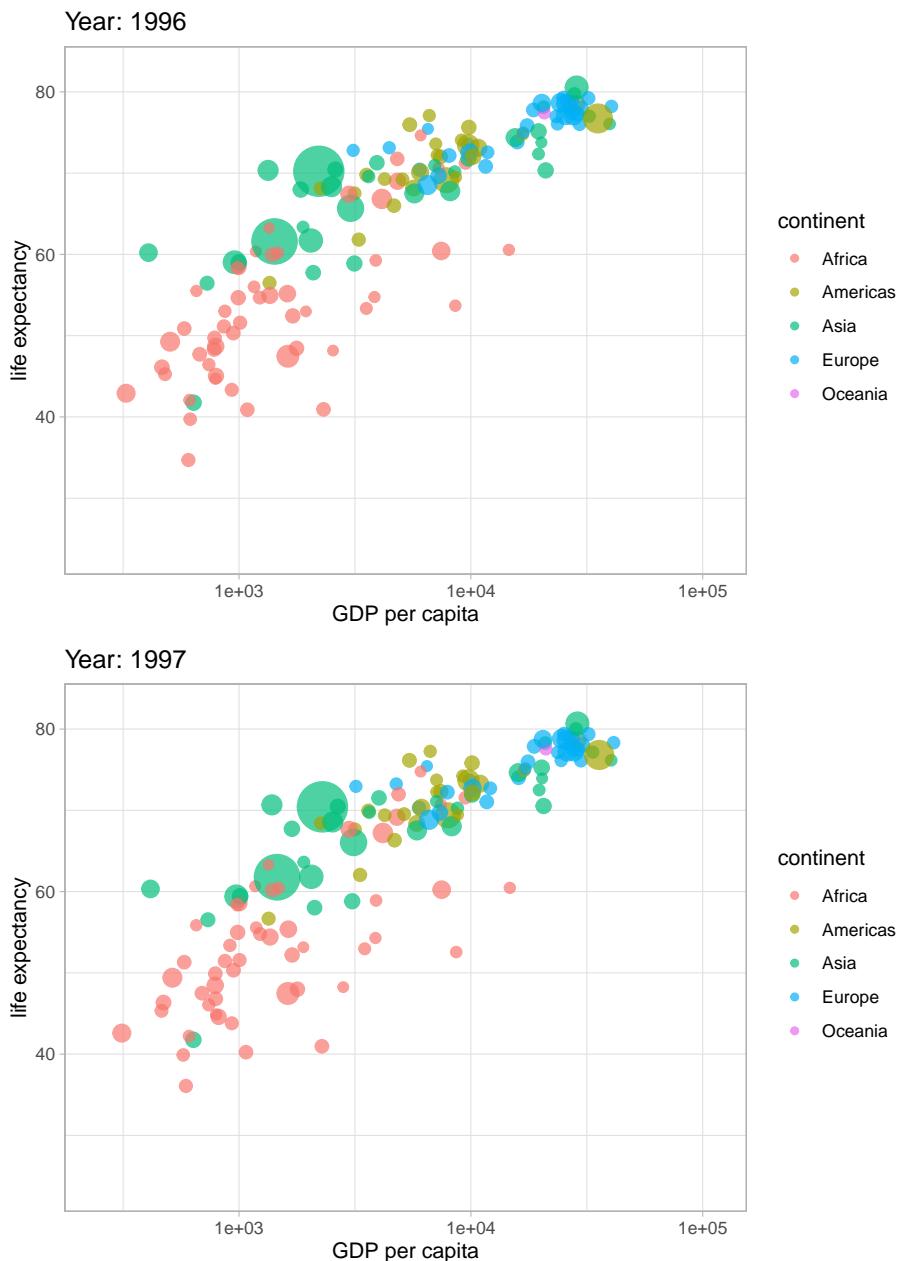


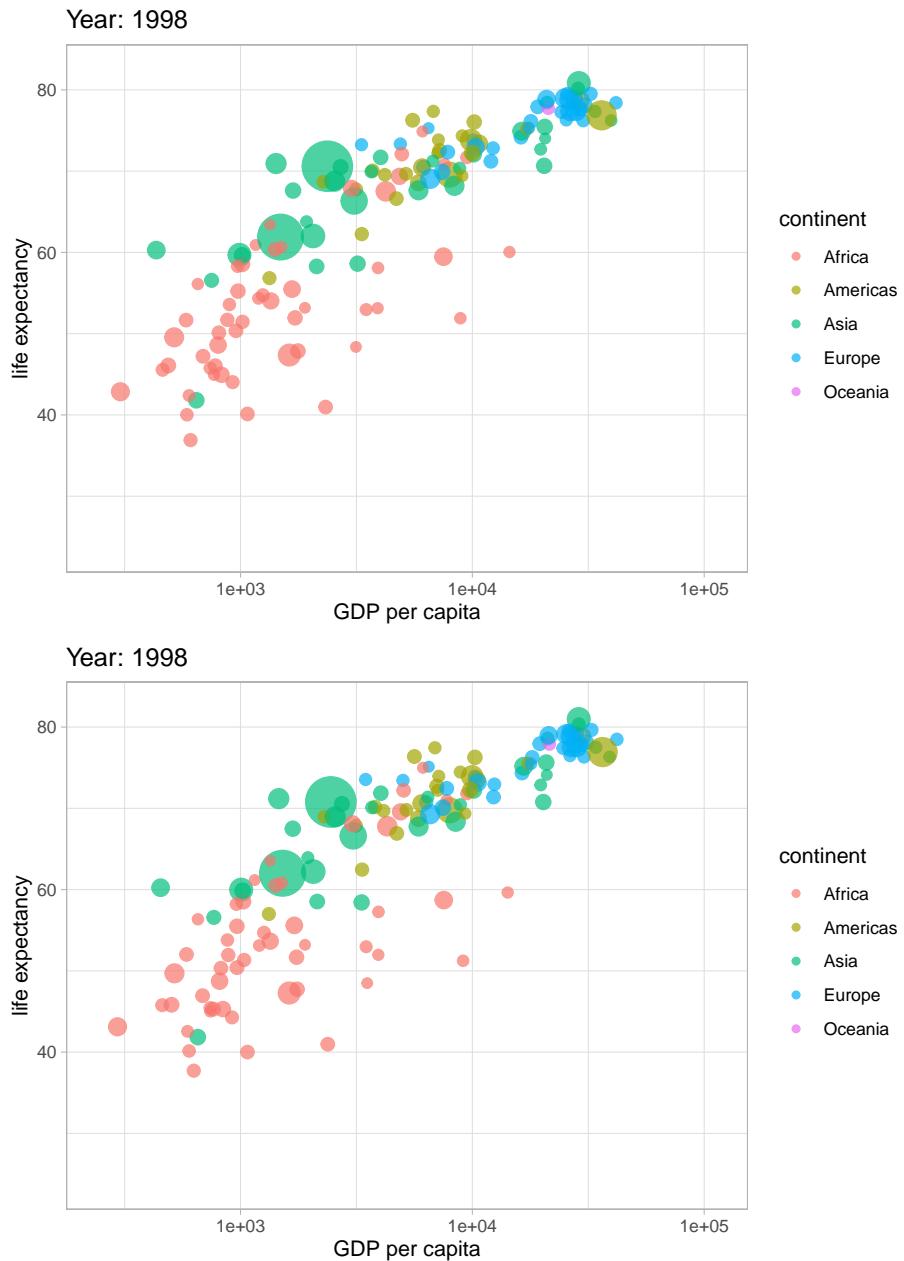


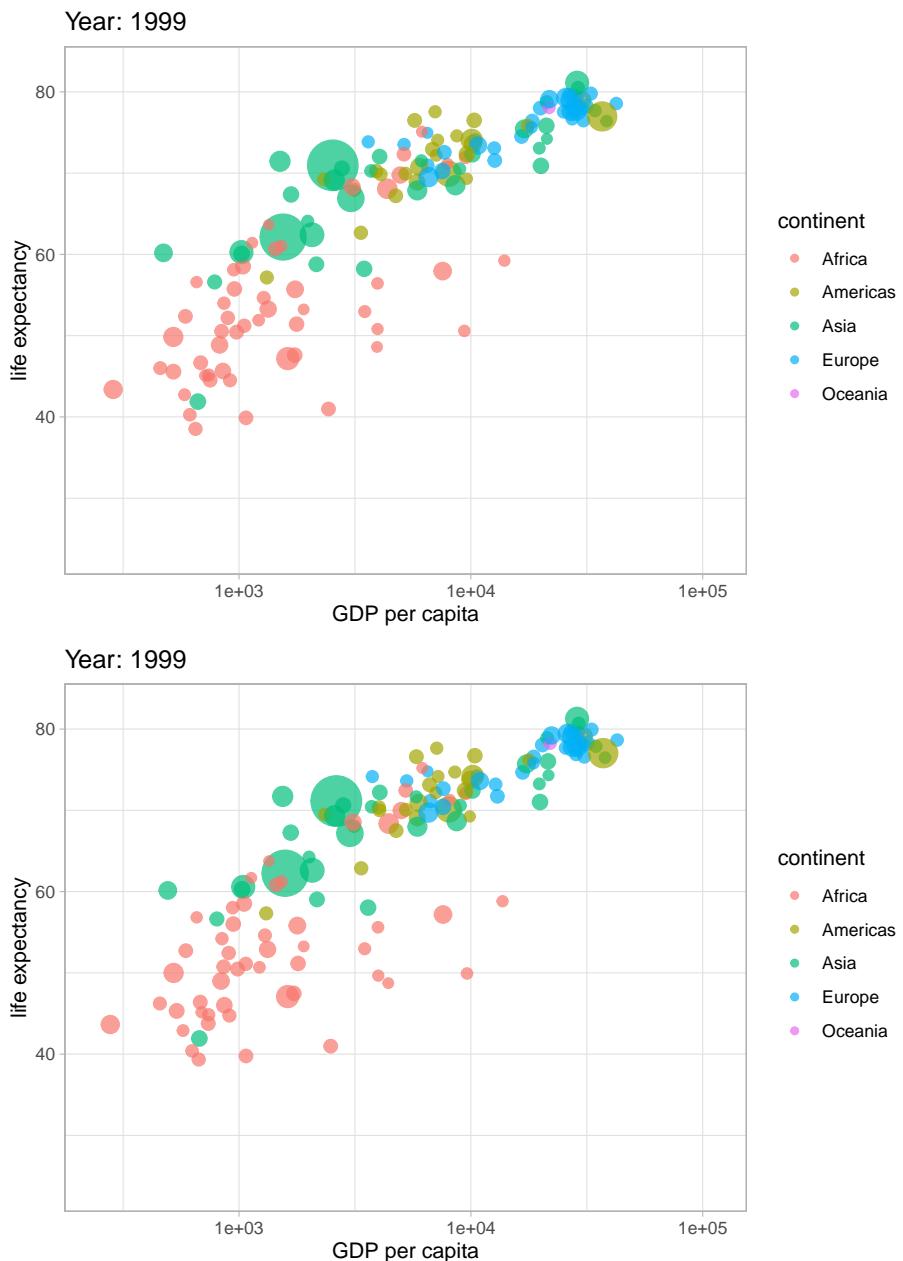


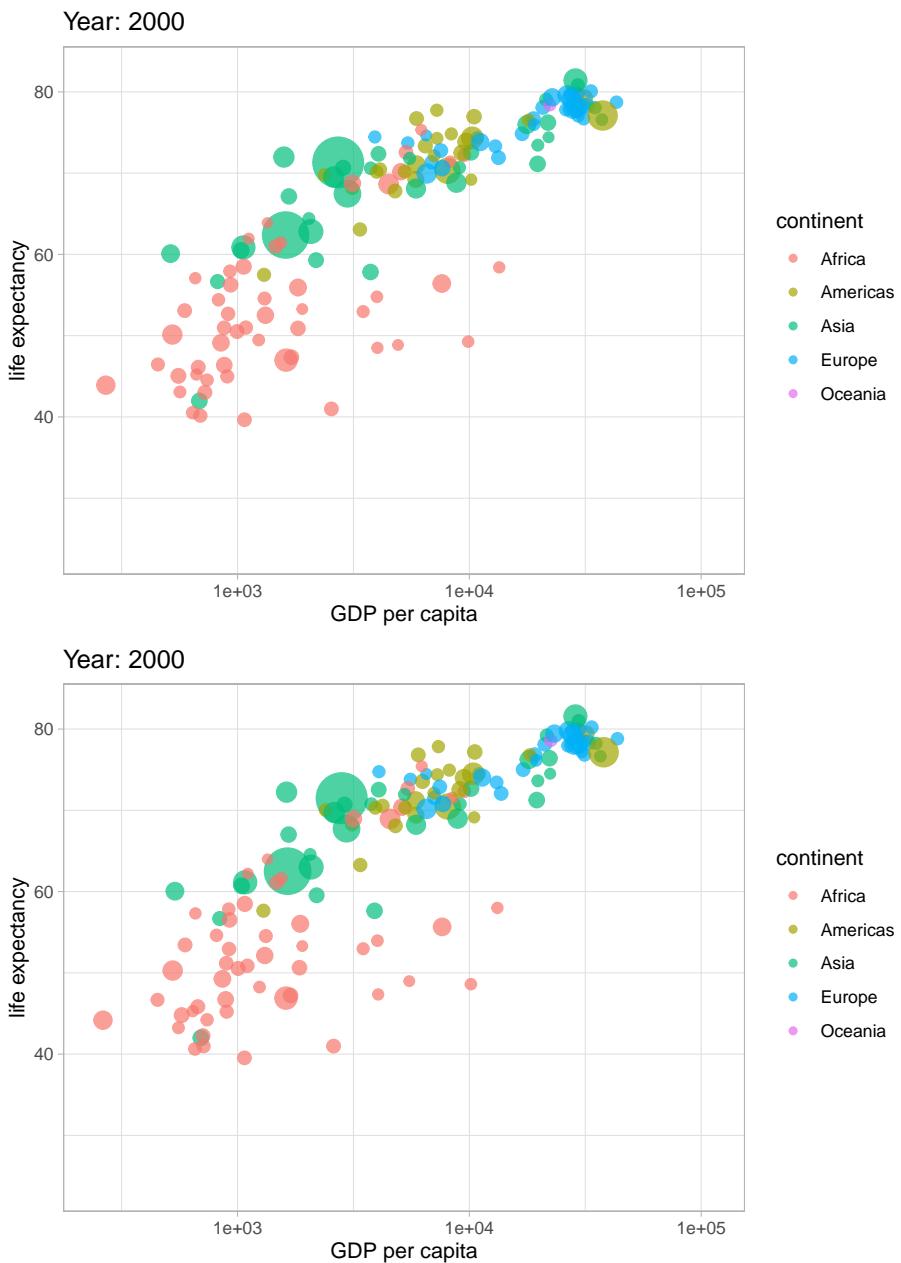


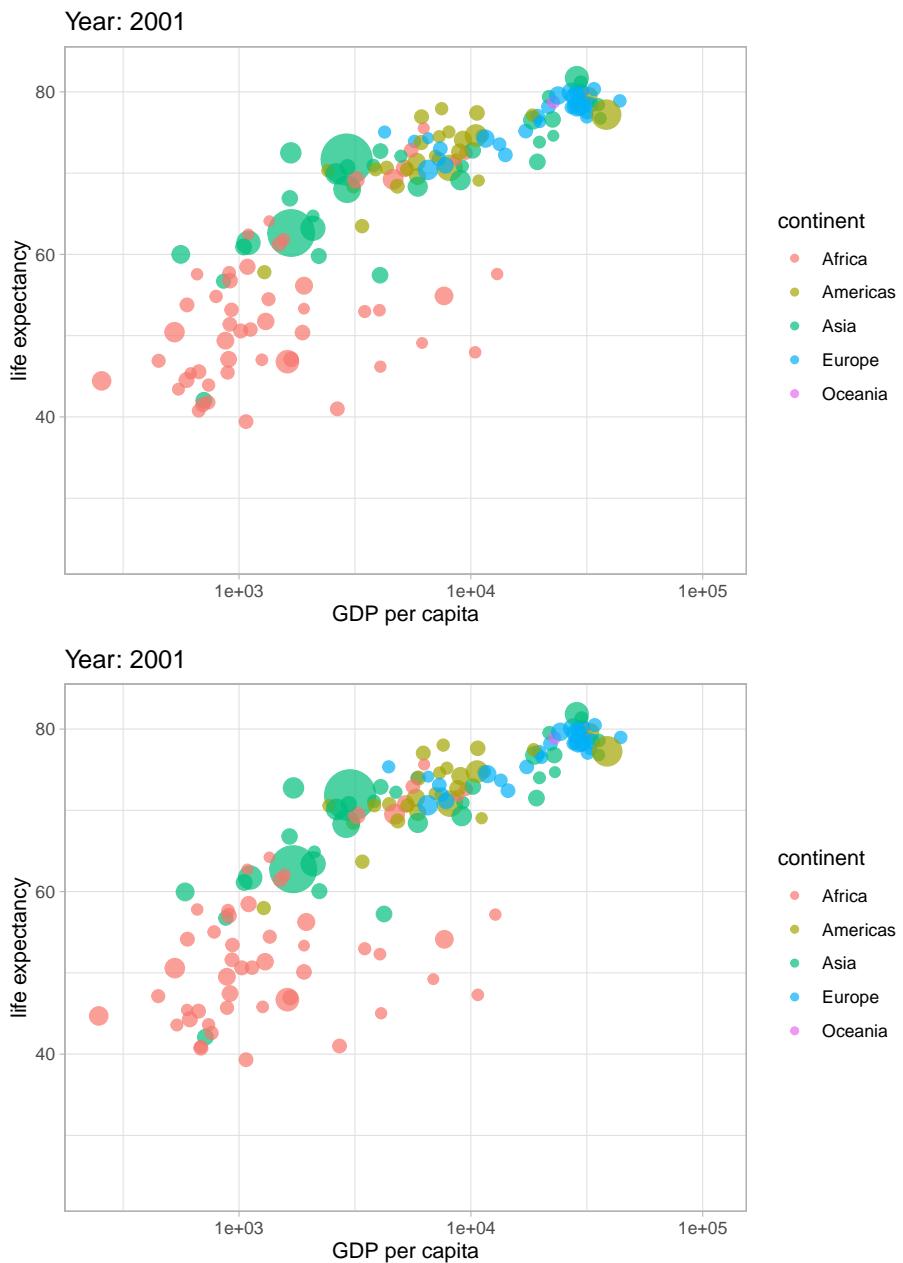


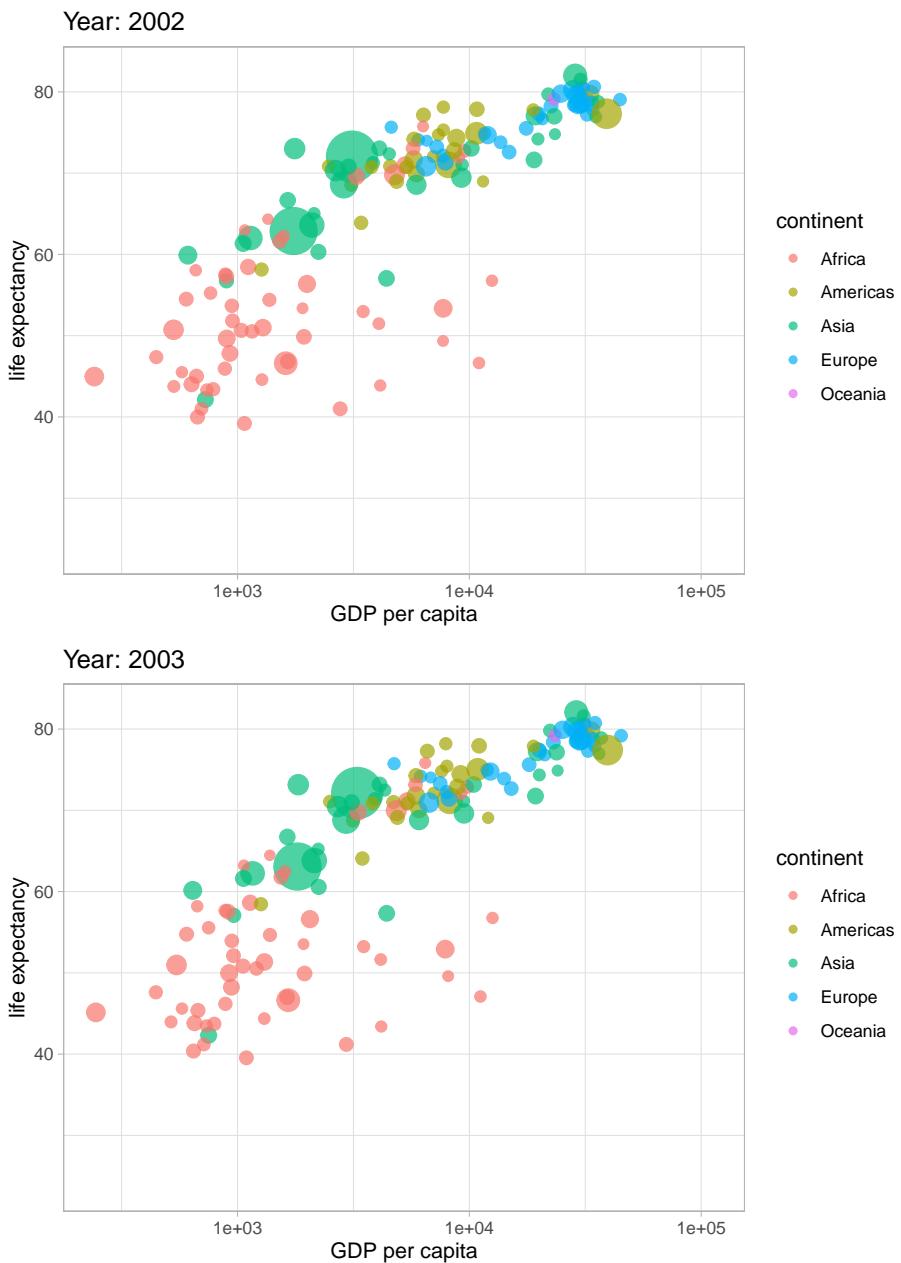


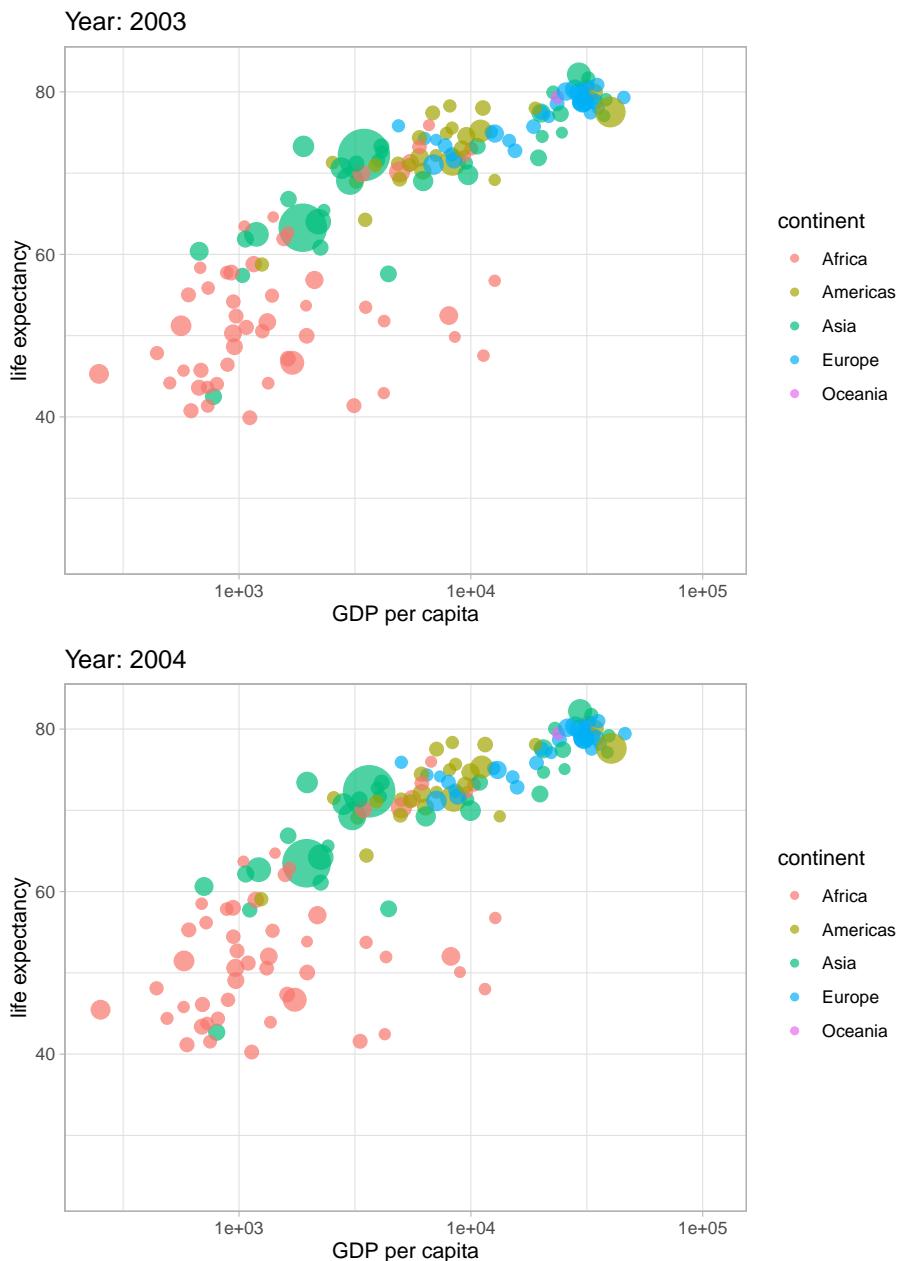


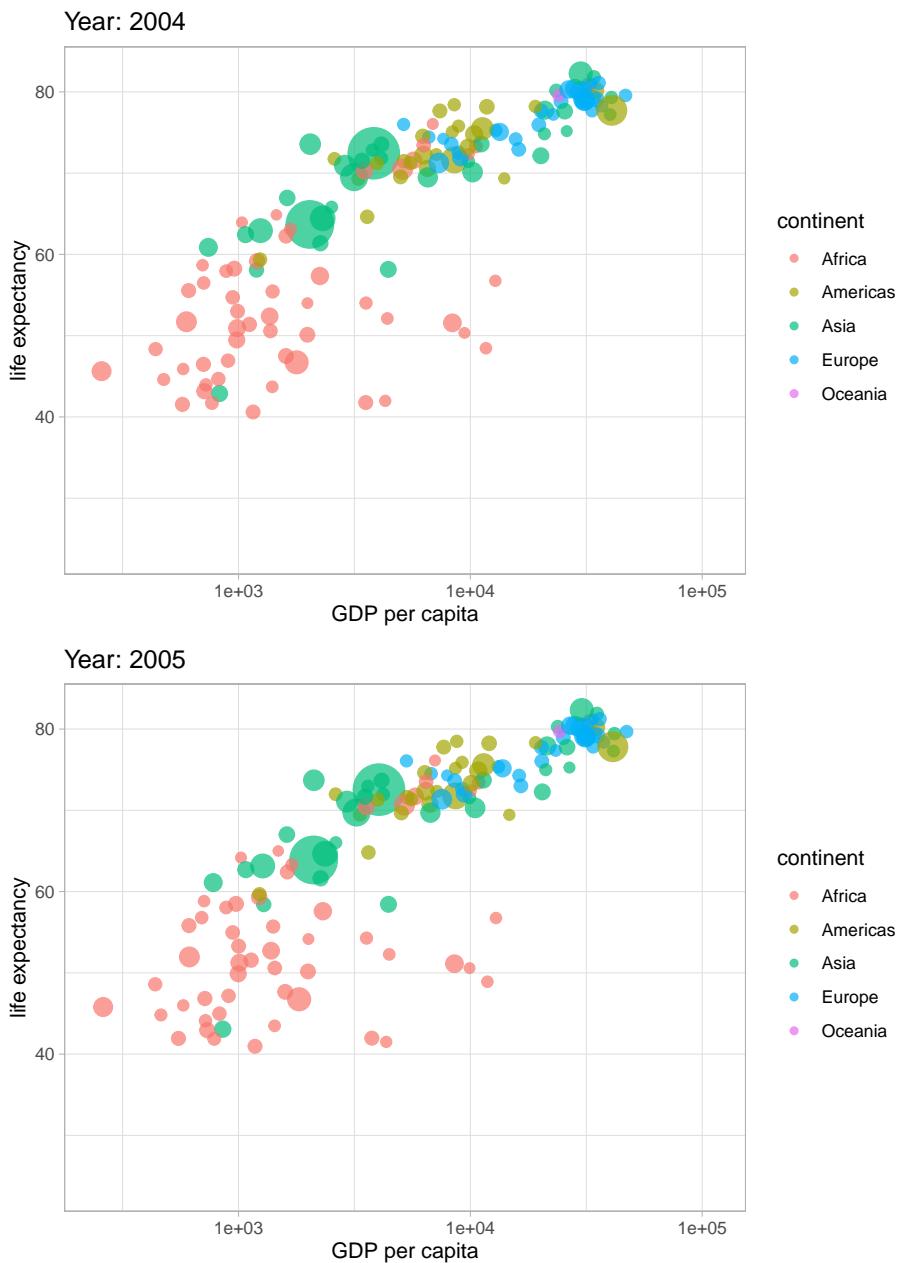


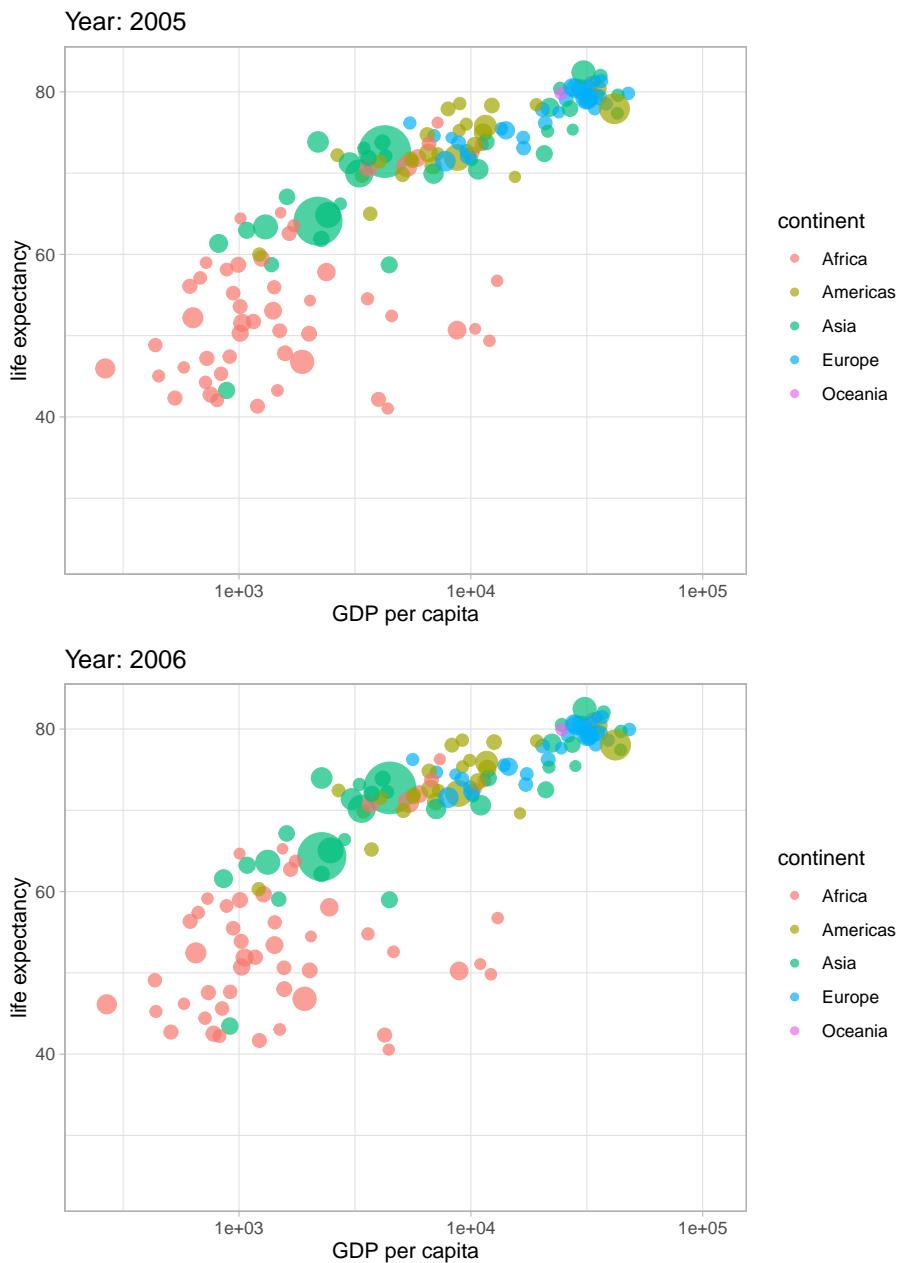


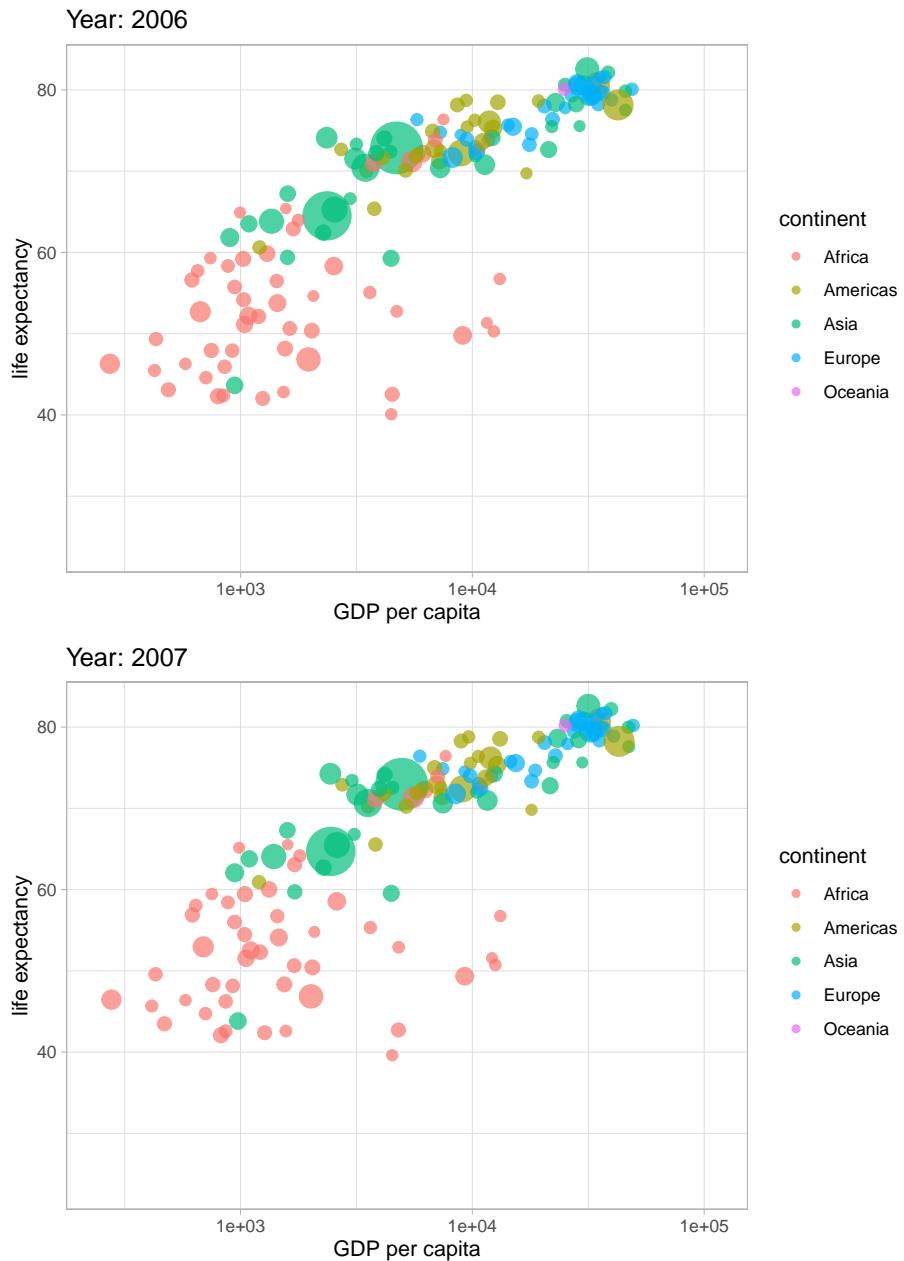












4.6.6 5. Statistiska beräkningar

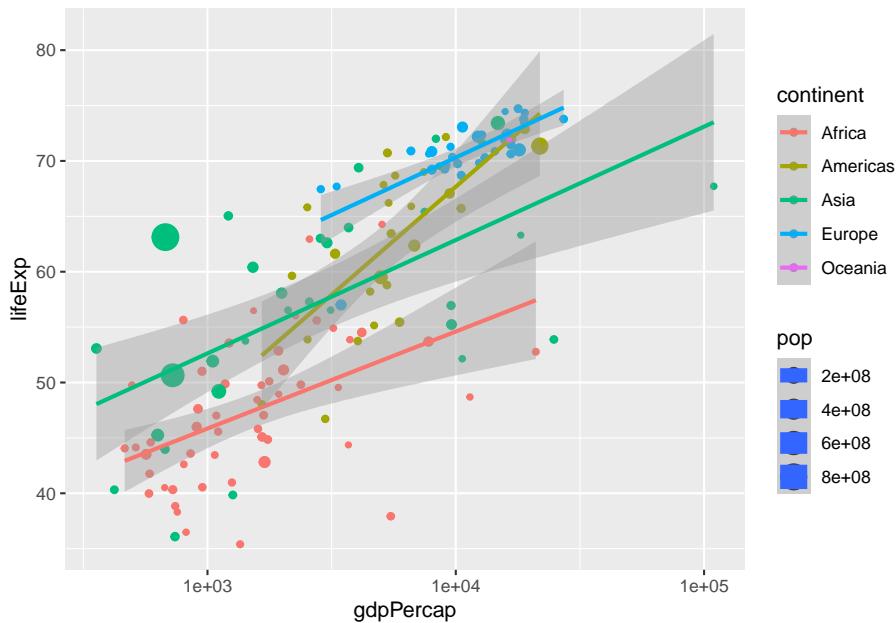
För att kvantifiera sammanbandet mera kan vi också lägga till statistiska beräkningar till grafen

```
ggplot(data = data, aes(x = gdpPercap,
                        y = lifeExp,
                        size = pop,
                        color = continent)) +
  geom_point() +
  scale_x_log10() +
  stat_smooth(method = "lm")

## `geom_smooth()` using formula 'y ~ x'

## Warning in qt((1 - level)/2, df): NaNs produced

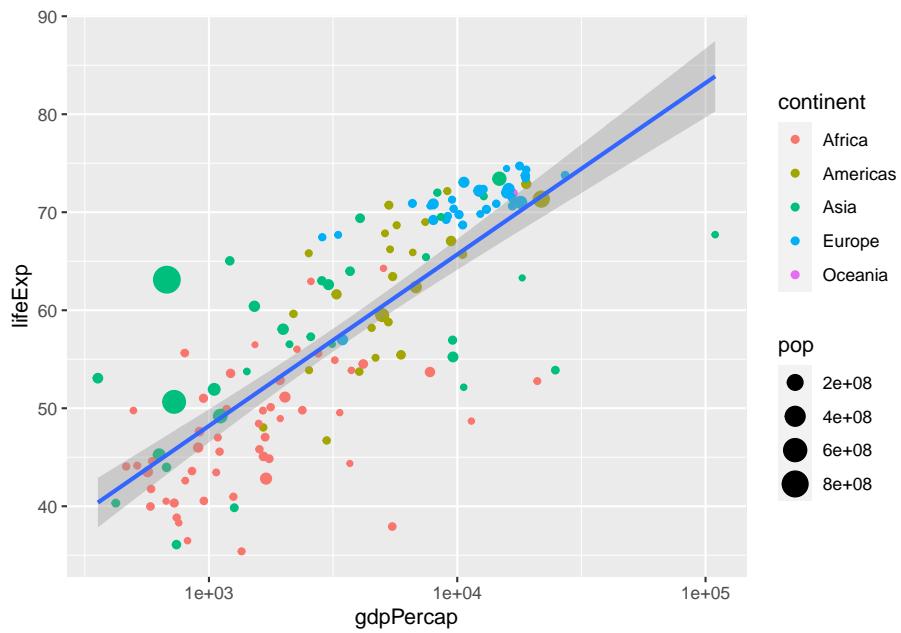
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf
```



- Nu beräknas statistik per grupp. Det är eftersom att vi specificerat i vår `aes()` i `ggplot()`.
- Vi kan flytta `aes()` till våra geometriska objekt om vi vill beräkna statistik för hela gruppen.

```
ggplot(data = data, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(aes(size = pop, color = continent)) +
  scale_x_log10() +
  stat_smooth(method = "lm")
```

`geom_smooth()` using formula 'y ~ x'



4.6.7 6. Facets

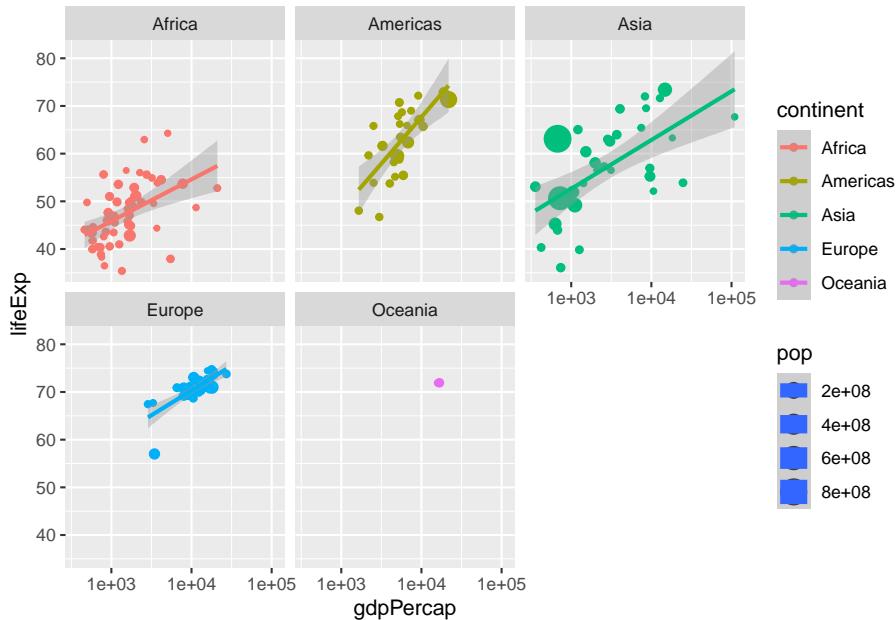
Vi kan också dela upp grafen i flera visualiseringar

```
ggplot(data = data, aes(x = gdpPerCap,
                        y = lifeExp,
                        size = pop,
                        color = continent)) +
  geom_point() +
  scale_x_log10() +
  stat_smooth(method = "lm") +
  facet_wrap(~continent)
```

`geom_smooth()` using formula 'y ~ x'

```
## Warning in qt((1 - level)/2, df): NaNs produced

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf
```



Vi kan också specificera skalorna för varje subplot

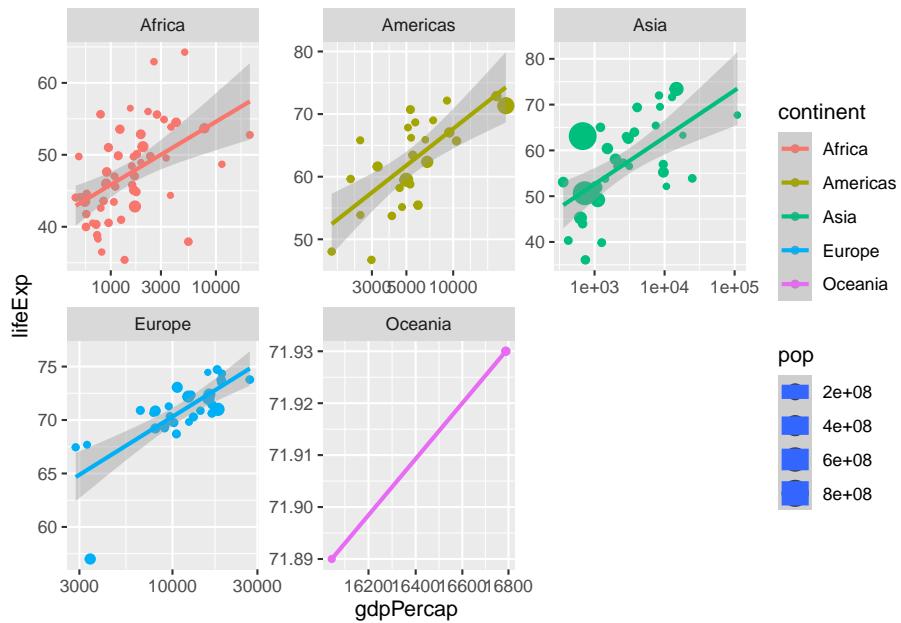
```
ggplot(data = data, aes(x = gdpPercap,
                        y = lifeExp,
                        size = pop,
                        color = continent)) +
```

```
geom_point() +
scale_x_log10() +
stat_smooth(method = "lm") +
facet_wrap(~continent, scales = "free")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

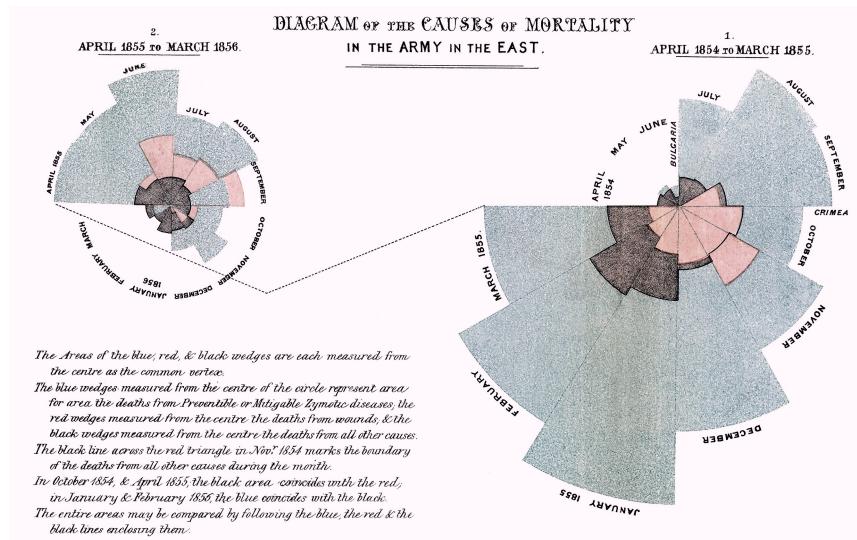
```
## Warning in qt((1 - level)/2, df): NaNs produced
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf
```



4.6.8 7. Koordinatsystem

- Ett sista lager vi skulle kunna använda för att ändra vår graf
- Exempelvis Kartesiskt eller Polärt
- Polärt för exempelvis cirkeldiagram



4.6.9 Grammar of graphics

R Paketet som har använts heter `ggplot2` och det bygger på en variant av *The Grammar of Graphics* och består av de 7 beståndsdelar vi precis gått igenom

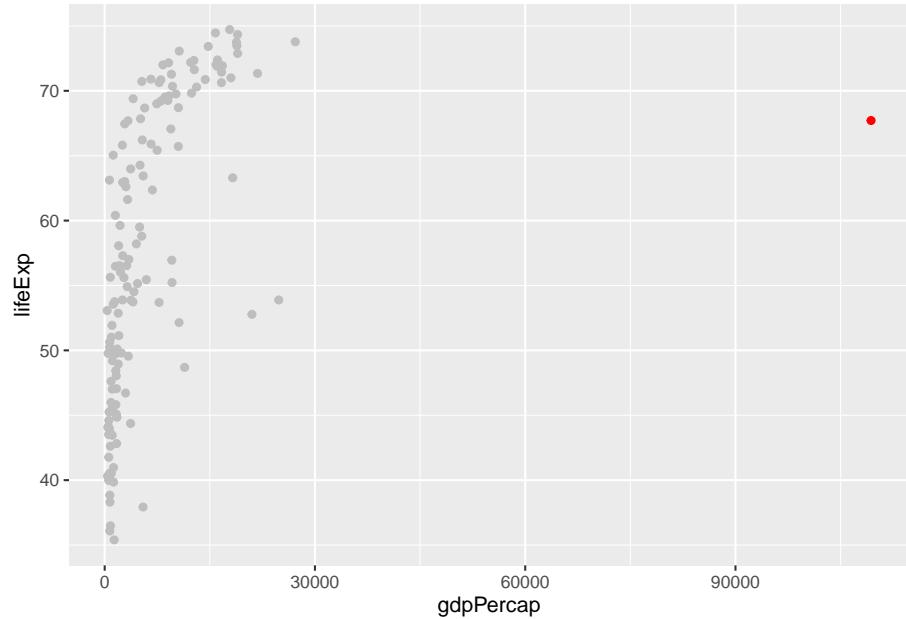
.footnote[Wickham, H (2010), “A layered Grammar of Graphics”, Journal of Computational and Graphical Statistics, vol. 19, no. 1, pp. 3–28,]

1. Data
2. Aesthetics
3. Geometric Objects
4. Scale
5. Statistics
6. Facets
7. Coordinate systems
- (8.) Labels, titlar, legends

Som kan manipuleras för att skapa de visualiseringar vi vill ha

4.7 Gapminder - fortsättning

Vi såg att det fanns en outlier i `gdpPercap` per `lifeExp`



Vilket land är det?

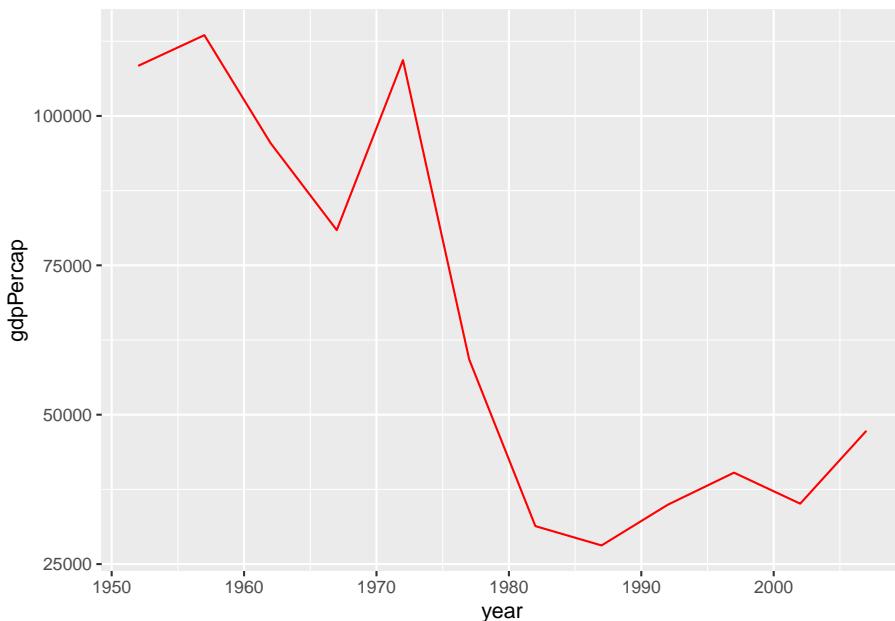
```
data %>% arrange(desc(gdpPercap))
```

```
## # A tibble: 142 x 6
##   country     continent year lifeExp      pop gdpPercap
##   <fct>       <fct>    <int>   <dbl>    <int>     <dbl>
## 1 Kuwait      Asia      1972    67.7    841934  109348.
## 2 Switzerland Europe    1972    73.8    6401400  27195.
## 3 Saudi Arabia Asia      1972    53.9    6472756  24837.
## 4 United States Americas 1972    71.3    209896000 21806.
## 5 Libya       Africa    1972    52.8    2183877  21011.
## 6 Canada      Americas  1972    72.9    22284500  18971.
## 7 Norway      Europe    1972    74.3    3933004  18965.
## 8 Denmark     Europe    1972    73.5    4991596  18866.
## 9 Netherlands Europe    1972    73.8    13329874  18795.
## 10 Bahrain    Asia      1972    63.3    230800   18269.
## # ... with 132 more rows
```

Är det en utveckling som håller i sig?

```
kuwait_data <- gapminder %>%
  filter(country == "Kuwait")
```

```
kuwait_data %>%
  ggplot(aes(x=year, y=gdpPercap)) +
  geom_line(color='red') + scale_color_identity()
```



I jämförelse med utvecklingen i resten av toppländerna 1972?

```
top_countries <- data %>% top_n(11, gdpPercap) %>%
  filter( country!= 'Kuwait') %>%
  pull(country)
top_countries

## [1] Bahrain      Canada       Denmark      Germany      Libya
## [6] Netherlands  Norway      Saudi Arabia Switzerland United States
## 142 Levels: Afghanistan Albania Algeria Angola Argentina Australia ... Zimbabwe

top_mean <- gapminder %>%
  filter(country %in% top_countries) %>%
  group_by(year) %>%
  summarise(gdp_mean = mean(gdpPercap))

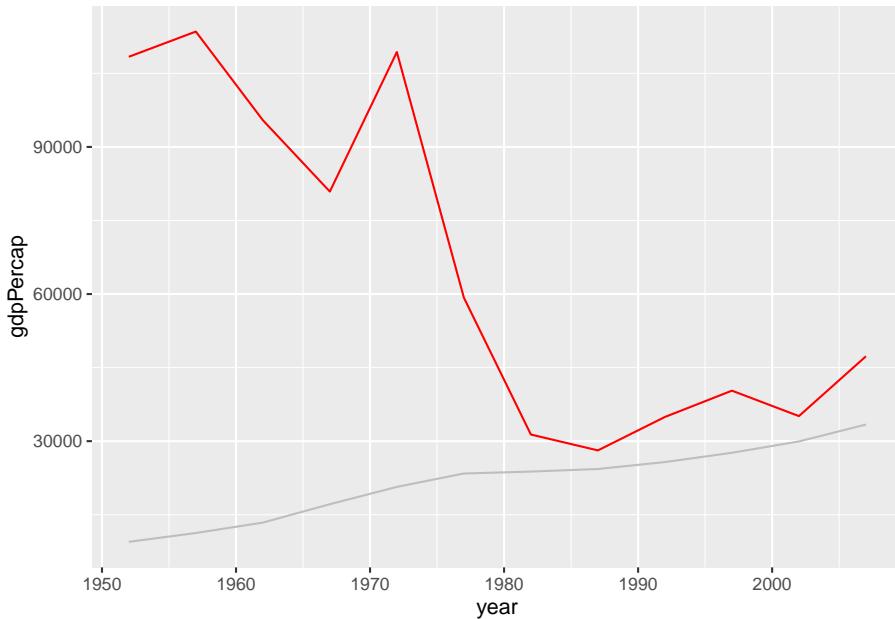
## `summarise()` ungrouping output (override with `.`groups` argument)
```

```
top_mean
```

```
## # A tibble: 12 x 2
##       year gdp_mean
##   <int>    <dbl>
## 1 1952     9468.
## 2 1957    11271.
## 3 1962    13393.
## 4 1967    17146.
## 5 1972    20673.
## 6 1977    23406.
## 7 1982    23799.
## 8 1987    24323.
## 9 1992    25740.
## 10 1997   27633.
## 11 2002   29947.
## 12 2007   33389.
```

```
data_added_mean <- kuwait_data %>% add_column(
  "mean" = top_mean$gdp_mean)

data_added_mean %>%
  ggplot(aes(x = year)) +
  geom_line(aes(y = gdpPercap), color = 'red') +
  geom_line(aes(y = mean), color = 'gray') +
  scale_color_identity()
```



4.8 EDA är en nyfiken process utan slut!

Det finns i princip alltid mera saker man skulle kunna undersöka som:

- Hur ser utvecklingen ut om vi istället för att undersöka BNP per kapita, undersöker totala BNP per land?
- Om vi utesluter Kuwait som en outlier 1972?
- Om vi inför en annan gruppering?
- etc..

4.8.1 Det ni ska ta med er idag är inte de specifika sakerna vi har undersökt, utan ett mindset!

4.9 Länkar till resurser

- R for Data Science - fantastisk bok som har bra material om allting Data Science
- Visual Vocabulary - Financial Times referensmaterial för visualiseringar

- ggplot2 - dokumentation för ggplot2
- kaggle - Hemsida med AI/ML tävlingar och källa till gratis dataset
- stack overflow - Forum för programmeringsfrågor, där svar på nästan alla problem man stöter på finns

4.10 Övningar Explorativ dataanalyse

Datan vi använder idag kommer från Hemnet!

```
library(tidyverse)      # Innehåller många mindre paket bl.a ggplot2
library(scales)        # Färghanteringspaket
library(gapminder)     # Datasetet gapminder
library(hrbrthemes)    # Teman till ggplot2

hem <- read_csv("https://raw.githubusercontent.com/Ferrologic/ds-program-block2/master/hemnet.csv")

## # A tibble: 99,838 x 15
##   sold_date   final_price square_m_price type area city   sq_m rooms   fee
##   <date>           <dbl>            <dbl> <chr> <chr> <dbl> <dbl> <dbl>
## 1 2019-01-11     2800000          56000 bost~ järv~ solna   50    2  3087
## 2 2019-01-11     1280000          18003 bost~ östr~ malmö  711    3  4346
## 3 2019-01-11     7250000          76316 frit~ sote~ sote~  95    6    NA
## 4 2019-01-11     2130000          38727 bost~ väll~ stoc~  55    2  3838
## 5 2019-01-11     950000          11176 fril~ cent~ kil    85    3    NA
## 6 2019-01-11     2195000          39909 bost~ edsä~ soll~  55    2  3208
## 7 2019-01-11     2900000          30851 par/~ bran~ hani~  94    3    NA
## 8 2019-01-11     4200000          46154 bost~ minn~ stoc~  91    4  4641
## 9 2019-01-11     2825000          49045 bost~ erik~ göte~  576    2  2875
## 10 2019-01-11    1995000          29556 bost~ stoc~ stoc~  675    2  5218
## # ... with 99,828 more rows, and 6 more variables: add_area_sqm <dbl>,
## #   perc_change <dbl>, list_price <dbl>, price_diff <dbl>, month <chr>,
## #   weekday <chr>
```

Vi kan börja med att använda `skim()` från `skimr` på vårt dataset.

```
library(___)
skim(___)
```

```
## Error: <text>:1:9: unexpected input
## 1: library(_
##           ^
```

Det var mycket information. Men vi kan försöka plocka ut några delar.

Hur många rader och kolumner har datasetet?

Vad för typ av data är kolumnen `area`?

I vilken kolumn saknas det flest värden?

Men vad innehåller egentligen kolumnerna `area`, `type`? För att svara på det så kan vi titta på några rader av dataset med R:s inbyggda funktion `head()`

```
head(___)
```

```
## Error: <text>:1:6: unexpected input
## 1: head(_
##           ^
```

Säg att det företag du jobbar på endast är intresserad av Stockholm, Malmö och Göteborg då börjar vi med att endast titta på de försäljningar som gjorts där:

Om du känner att du hellre vill undersöka några andra städer så är det bara att skriva in dem istället, men försäkra dig om att städerna finns med genom att exempelvis använda `hem %>% filter(city == 'din stad')` eller `hem[hem$city == 'din stad',]`.

```
hem_storstad <- hem %>% filter(city %in% c('stockholm', 'malmö', ___))
```

```
## Error: <text>:1:66: unexpected input
## 1: hem_storstad <- hem %>% filter(city %in% c('stockholm', 'malmö', _  
##           ^
```

Hur har de slutgiltiga priserna i storstäderna sett ut?

```
ggplot(___,  
       aes(x = ___)) +  
       geom_density()
```

```
## Error: <text>:1:8: unexpected input
## 1: ggplot(_
##           ^
```

Är det nån skillnad mellan städerna? Det kan vi undersöka med hjälp av färg och en ridgeplot:

```
library(ggrridges)
ggplot(_,
       aes(x = ___,
            y = city,
            color = ___,
            fill = city)) + geom_density_ridges()
```

```
## Error: <text>:2:8: unexpected input
## 1: library(ggrridges)
## 2: ggplot(_
## ^
```

Vill vi göra en lättare jämförelse mellan städerna kan vi använda `geom_density()` istället och färg. Notera användandet av `alpha` och jämför med det tidigare kommandot. Det är också värt att notera användandet av parametern `alpha`, som styr hur genomskinliga de olika densiteterna är.

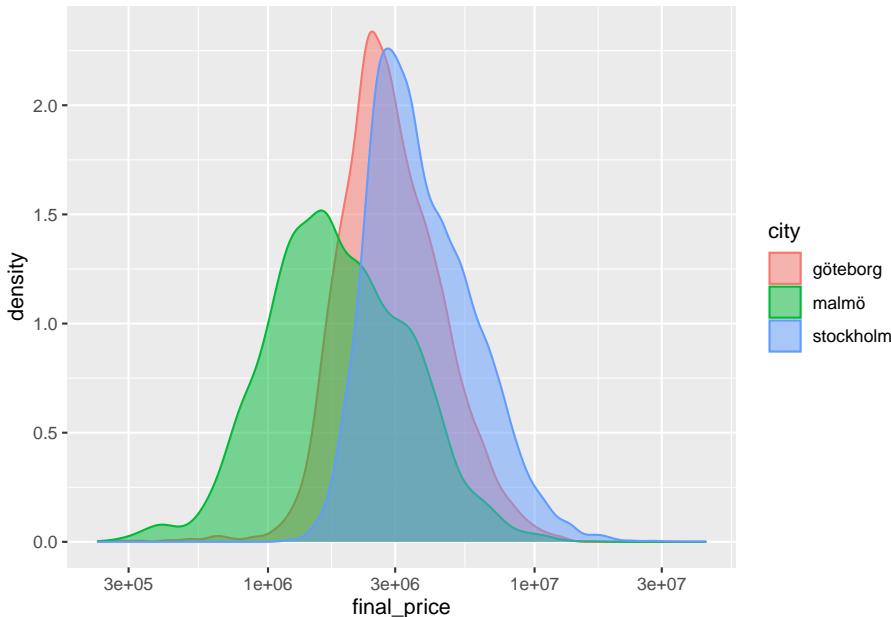
```
___ %>% ggplot(aes(x = ___,
                      color = city,
                      fill = city)) +
  geom_density(alpha=0.5)
```

```
## Error: <text>:1:1: unexpected input
## 1: _^
## ^
```

Som i `gapminder`-exemplet från presentationen så är fördelningen för final price skev. Ingen så kan vi använda en logaritmisk skala på x axeln `scale_x_log10()` för att motverka detta.

```
hem_storstad <- hem %>% filter(city %in% c('stockholm', 'malmö', "göteborg"))

hem_storstad %>% ggplot(aes(x = final_price,
                               color = city, fill = city)) +
  geom_density(alpha = 0.5) +
  scale_x_log10()
```



Vad kan vi se för skillnader mellan fördelningen av slutpriser mellan de städerna? Vilken stad är dyrast? Vad är det vanligaste priset ungefär? Är det någon stad som är mer olik de andra?

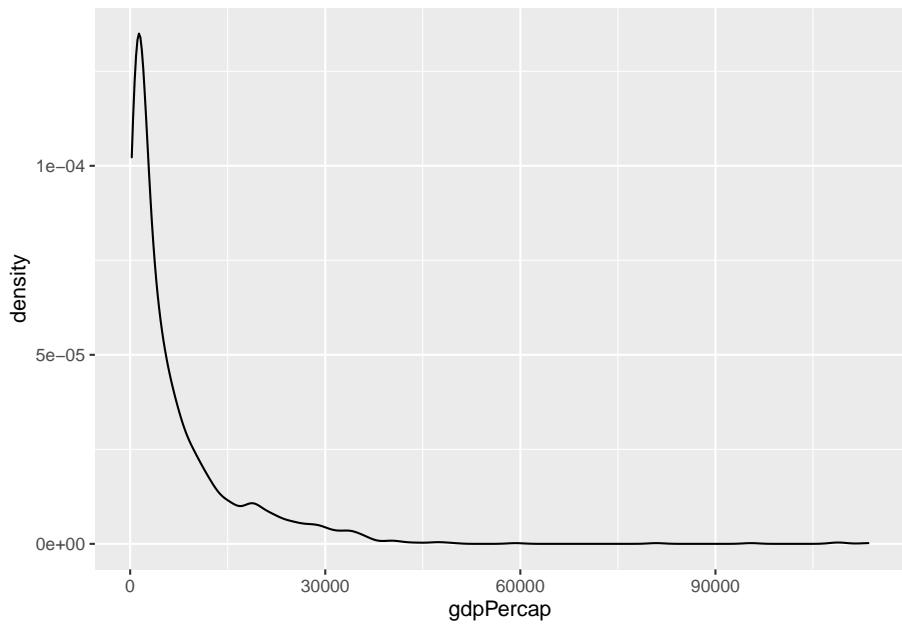
Det går självklart att fortsätta att ställa mera frågor och undersöka dem, men nu kommer resten av övningarna att fokusera på att introducera flera sorters visualiseringar!

4.11 Fördelningar

I de flesta fall är vi intresserade av fördelningar. De vanligaste typerna av visualiseringar för fördelningar är dessa:

Densitetsdiagram:

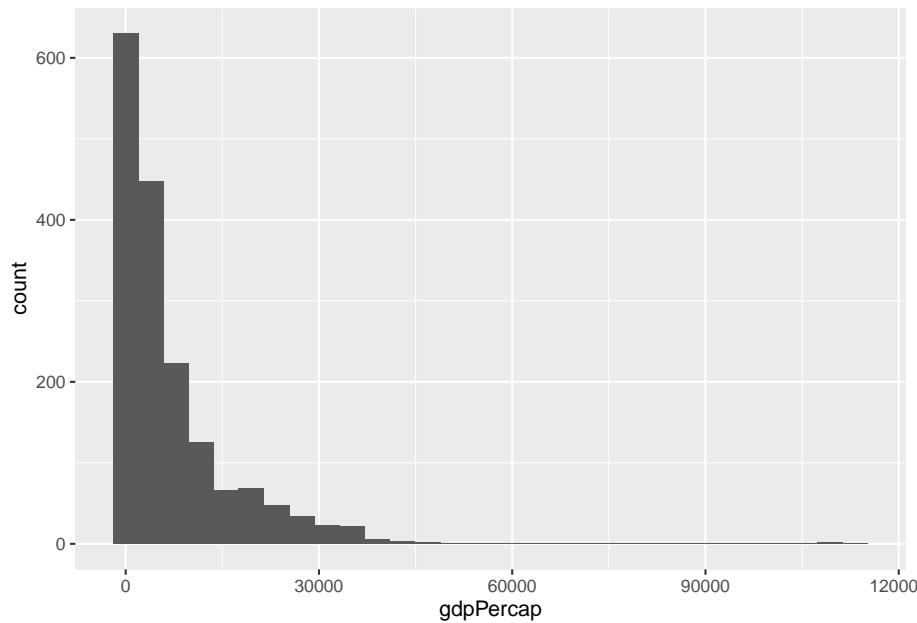
```
library(gapminder)
ggplot(gapminder, aes(x = gdpPercap)) +
  geom_density()
```



Histogram:

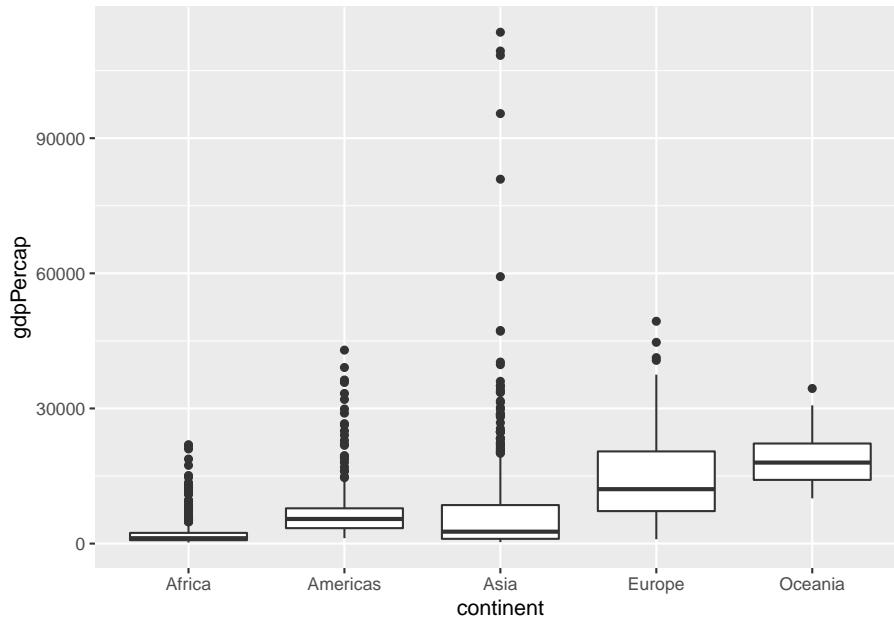
```
ggplot(gapminder, aes(x = gdpPerCap)) +  
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



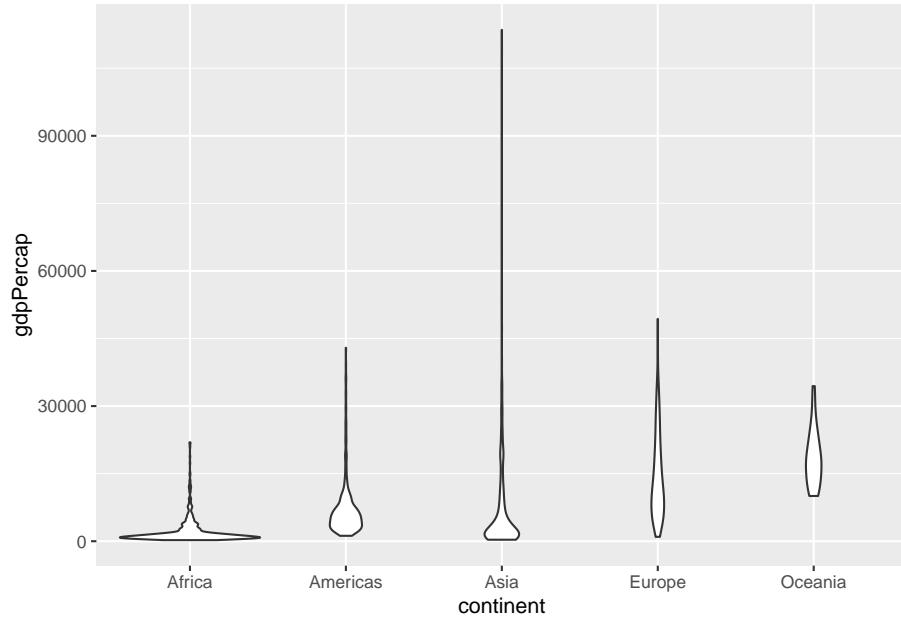
Boxplots:

```
ggplot(gapminder, aes(x = continent, y = gdpPerCap)) +  
  geom_boxplot()
```



Fioldiagram:

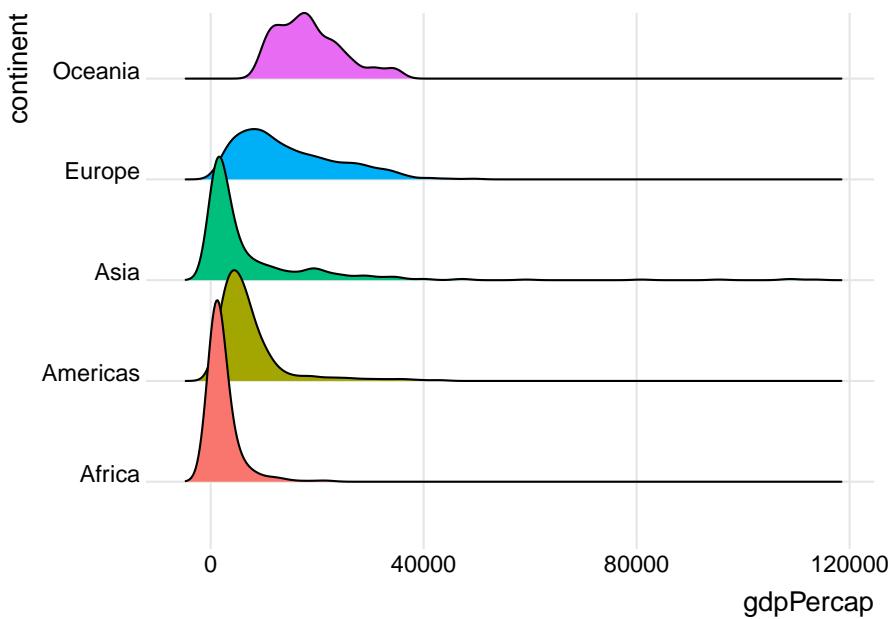
```
ggplot(gapminder, aes(x = continent, y = gdpPercap)) +
  geom_violin()
```



Ridgediagram:

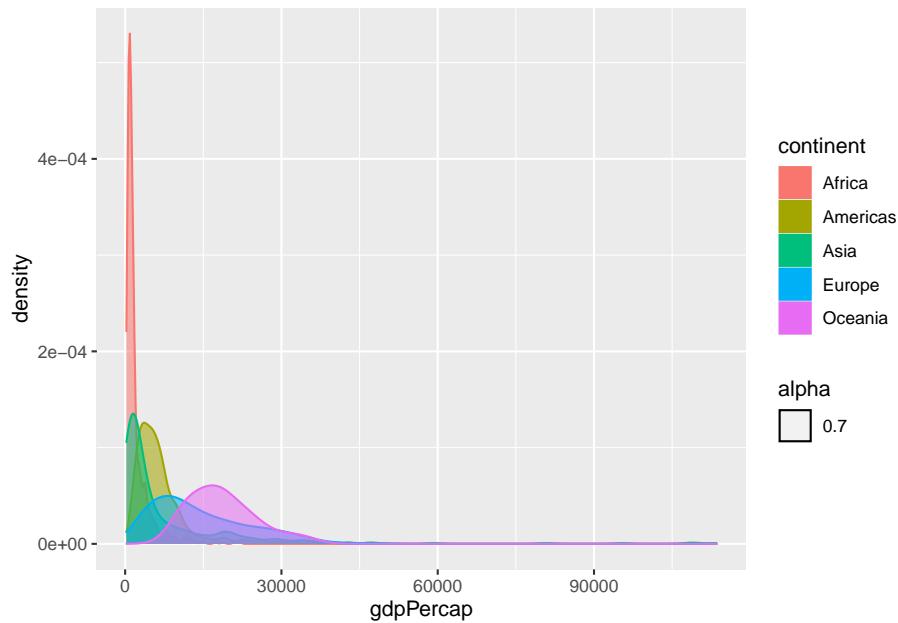
```
library(ggridges)
ggplot(gapminder, aes(x = gdpPercap, y = continent, fill = continent)) +
  geom_density_ridges() +
  theme_ridges() +
  theme(legend.position = "none")
```

```
## Picking joint bandwidth of 1650
```



För att direkt jämföra grupperna går även ett densitetsdiagram med färg att använda

```
ggplot(gapminder, aes(x = gdpPerCap, color = continent)) +  
  geom_density(aes(fill = continent, alpha = 0.7))
```



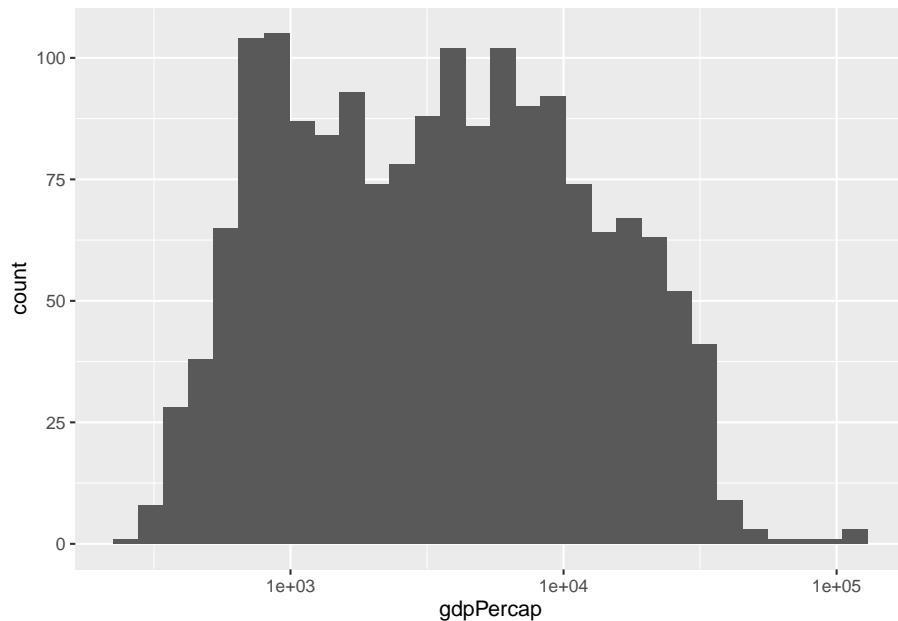
4.11.1 Skalar och fördelningar

I verkligheten är data sällan normalt fördelad. Med den underliggande statistiska processen kan ofta vara det. Därför kan det vara klokt, om data är skevt fördelat, att testa att använda en annan skala på data. Det finns flera olika sätt att transformera data till en annan skala men ett av de vanligaste är att ta logaritmen.

Nedan skalar vi x-axeln med `log10()`, som är den logaritmen vi oftast använder i “verkligheten”.

```
ggplot(gapminder, aes(x = gdpPercap)) +
  geom_histogram() +
  scale_x_log10()

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



4.11.2 Övningar - Fördelning

Nu är det din tur att prova på de här graferna på hemnetdataen `hem`. Hur ser fördelningen ut för `final_price`?

```
ggplot(aes(_)) +
  geom_  
  
## Error: <text>:1:12: unexpected input
## 1: ggplot(aes(_  
##  
##
```

Hur ser fördelningen ut för respektive `type`? Nedan använder du dig av en ridge-plot.

```
library(ggridges)
ggplot(hem, aes(x = final_price, y = _, fill = type)) +
  geom_density_ridges() +
  theme_ridges() +
  theme(legend.position = _) +
  scale_x_continuous(labels = scales::number)
```

```
## Error: <text>:2:38: unexpected input
## 1: library(gggridges)
## 2: ggplot(hem, aes(x = final_price, y = _
##
```

Boxplot är en annan värdefull visualiseringstyp. Den kalkylerar även outliers, som är de svarta prickarna. Outliers är här definierade som $Q3 + 1.5 \cdot IQR$, d.v.s. den tredje kvartilen + 1.5 gånger avståndet mellan kvartil 1 och kvartil 3.

- Visualisera slutpris per bostadstyp
- Notera att vi använder `scales` i `scale_x_continuous()` för att få snyggare formatering på x-axlen. `scales` har också formateringsalternativ för procent, kr/\$/€ med mera.

```
ggplot(hem, aes(x = final_price, y = _, fill = _)) +
  geom_boxplot() +
  theme(legend.position = "none") +
  scale_x_continuous(labels = scales::number_format(suffix = "kr"))
```

```
## Error: <text>:1:38: unexpected input
## 1: ggplot(hem, aes(x = final_price, y = _
##
```

4.12 Korrelation

Visualisera relationen mellan `list_price` och `final_price`.

- Vad är dina `aesthetics`?
- Vilken `geom_` använder du?
- Passar skalan?
- Finns det någon passande statistic för att visuellt beskriva relationen?

```
ggplot(_, aes(x = _, y = _)) +
  geom_ +
  stat_
```

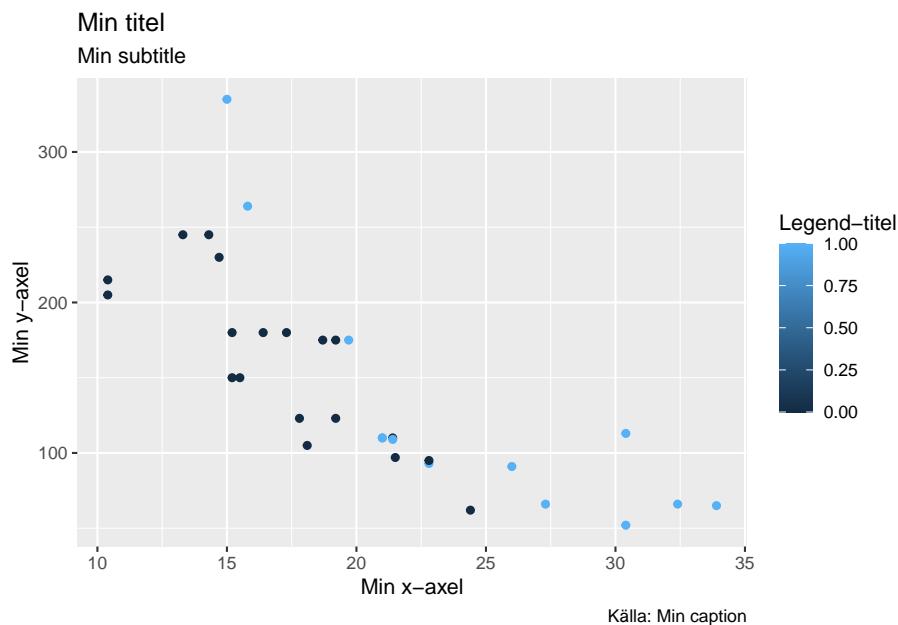
```
## Error: <text>:1:8: unexpected input
## 1: ggplot(_
##
```

4.13 Pimp my plot!

Vi kan addera titlar och data-etiketter med `labs()`.

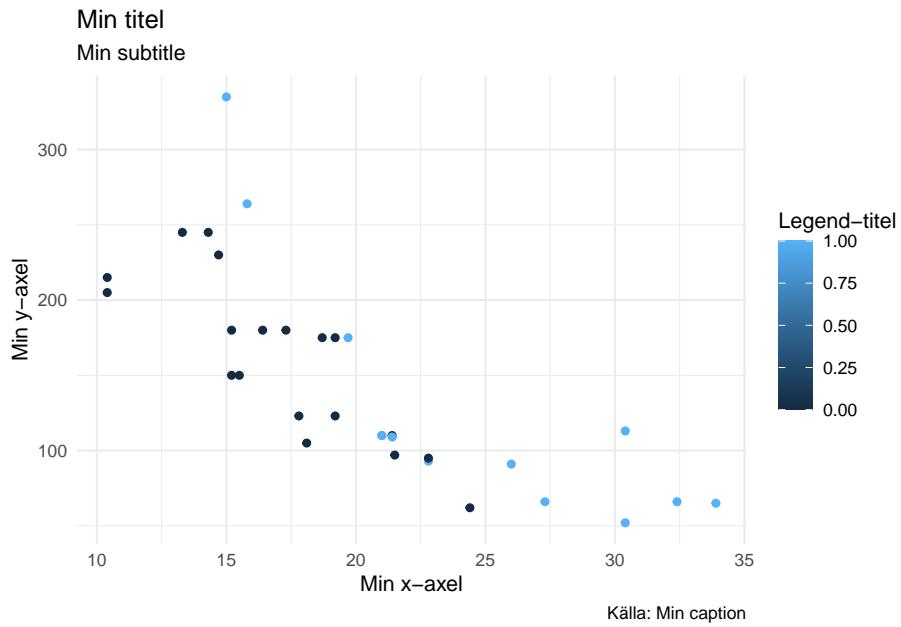
```
p <- ggplot(mtcars, aes(mpg, hp, color = am)) +
  geom_point() +
  labs(title = "Min titel",
       x = "Min x-axel",
       y = "Min y-axel",
       subtitle = "Min subtitle",
       caption = "Källa: Min caption",
       color = "Legend-titel")
```

p



4.14 Vi kan ändra tema med `theme_...`

```
p + theme_minimal()
```



Utgå från relationen mellan slutpris och list pris. Ändra labs och lägg till ett tema.

OBS Det finns en uppsjö av olika teman. Exempelvis i paketen `hrbrthemes`, `ggthemes`, `ggthemr` m.m. Du kan också skapa dina egna teman exempelvis med företagsfont och färger. Testa gärna exempelvis `theme_economist()` och `theme_solarized_2()`

```
library(ggthemes)
ggplot(hem, aes(x = list_price, y = final_price)) +
  geom_point() +
  stat_smooth() +
  labs(title = "",
       x = "",
       y = "",
       subtitle = "",
       caption = "") +
  theme_
```

```
## Error in eval(expr, envir, enclos): object 'theme_' not found
```

4.15 Andra aesthetics

- Utgå från din visualisering av relationen mellan utropspris och slutpris

- Mappa vår aesthetic `color` till en numerisk variabel, exempelvis `fee`.
- Vad händer om vår data har stora outliers? (testa att mappa `color` till `sq_m`)

```
ggplot(hem, aes(x = list_price,
                 y = final_price,
                 color = ___)) +
  geom_point() +
  stat_smooth()

## Error: <text>:3:25: unexpected input
## 2:                 y = final_price,
## 3:                 color =
##
```

4.16 Diskreta variabler

- Utgå från din visualisering av relationen mellan utropspris och slutpris
- Mappa vår aesthetic `color` till en kategorisk variabel, notera att kategoriska variabler med fler än 30 kategorier kommer att bli svåra att visualisera. Se exempelvis `city` och `area`, som du därför bör undvika.
- Om det är en kategorisk variabel som är numerisk behöver du ändra den med `as.factor()`

```
ggplot(hem, aes(x = list_price,
                 y = final_price,
                 color = ...)) +
  geom_point() +
  stat_smooth()
```

```
## Error in ggplot.default(hem, aes(x = list_price, y = final_price, color = ...)): '...' used in
```

4.17 Facets

- Visualisera fördelningen per bostadstyp med sub-plots istället
- Hur hanterar du x-axeln? Finns det något sätt att göra den snyggare?
- För att hantera långa labels i `facet_wrap()` kan du använda funktionen `label_beller` och `label_wrap_gen(10)`.

```
ggplot(hem, aes(final_price)) +
  geom_histogram() +
  scale_x_log10() +
  facet_wrap(~___, labeller(type = label_wrap_gen(10)), scales = "free")

## Error: <text>:4:15: unexpected input
## 3:   scale_x_log10() +
## 4:   facet_wrap(~_
##
```

4.18 Barcharts på aggregerade tabeller

Inte sällan har vi aggregerade tabeller som vi vill visualisera.

```
gap_continent <- gapminder %>%
  group_by(continent) %>%
  summarise(mean_gdp_per_cap = mean(gdpPercap))

## `summarise()` ungrouping output (override with ` `.groups` argument)

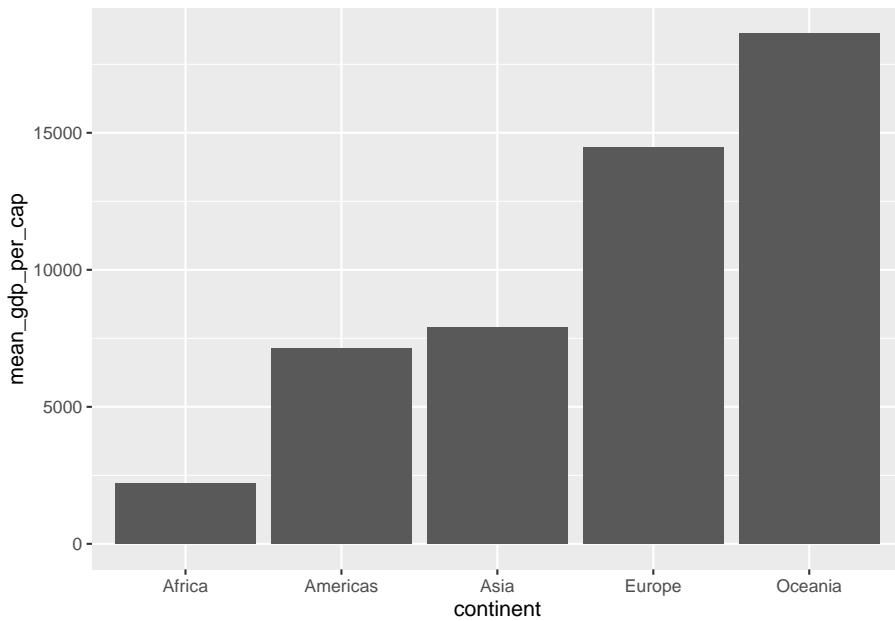
gap_continent

## # A tibble: 5 x 2
##   continent mean_gdp_per_cap
##   <fct>          <dbl>
## 1 Africa           2194.
## 2 Americas         7136.
## 3 Asia             7902.
## 4 Europe           14469.
## 5 Oceania          18622.
```

4.18.1 geom_col()

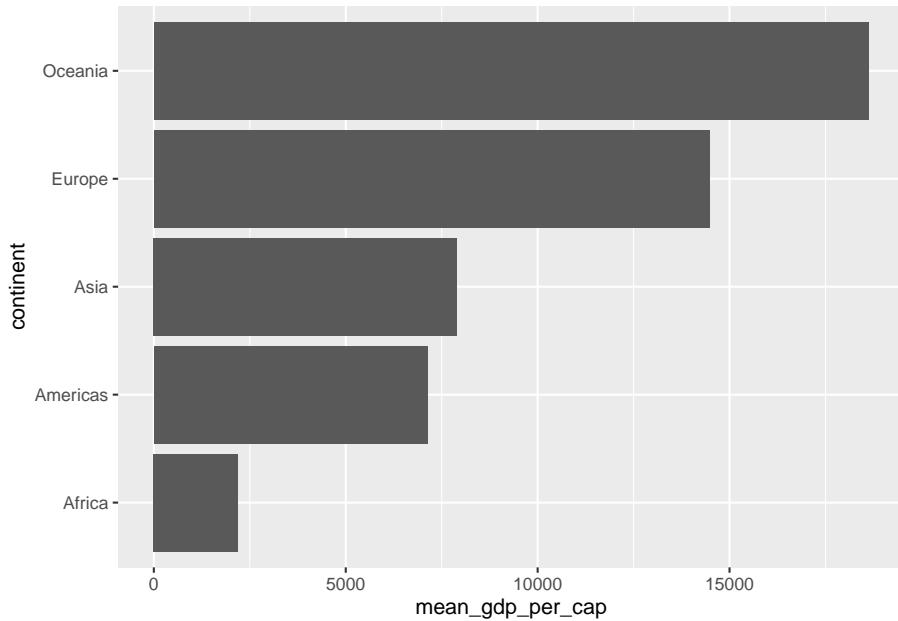
- `geom_bar()` används när vi vill veta antalet rader per kategori
- `geom_col()` används på färdiga tabeller
- Sorterar efter bokstavsordning på kategorier

```
gap_continent %>%
  ggplot(aes(x = continent, y = mean_gdp_per_cap)) +
  geom_col()
```



Ibland har vi många kategorier, då kan vi bara byta plats på x och y

```
gap_continent %>%
  ggplot(aes(y = continent, x = mean_gdp_per_cap)) +
  geom_col()
```



4.18.2 Hur fördelar sig kvadratmeterpris per bostadstyp?

- Ta fram en aggregerad tabell med genomsnittligt kvadratmeterpris per bostadstyp
- Vilken funktion använder du för att kalkylera genomsnitt? Ha fördelnin-
gen i åtanke
- Visualisera med `geom_col()`

```
tbl_type <- hem %>%
  group_by(type) %>%
  summarise(median_sq_m_price = median(square_m_price, na.rm = T))

## `summarise()` ungrouping output (override with `~.groups` argument)

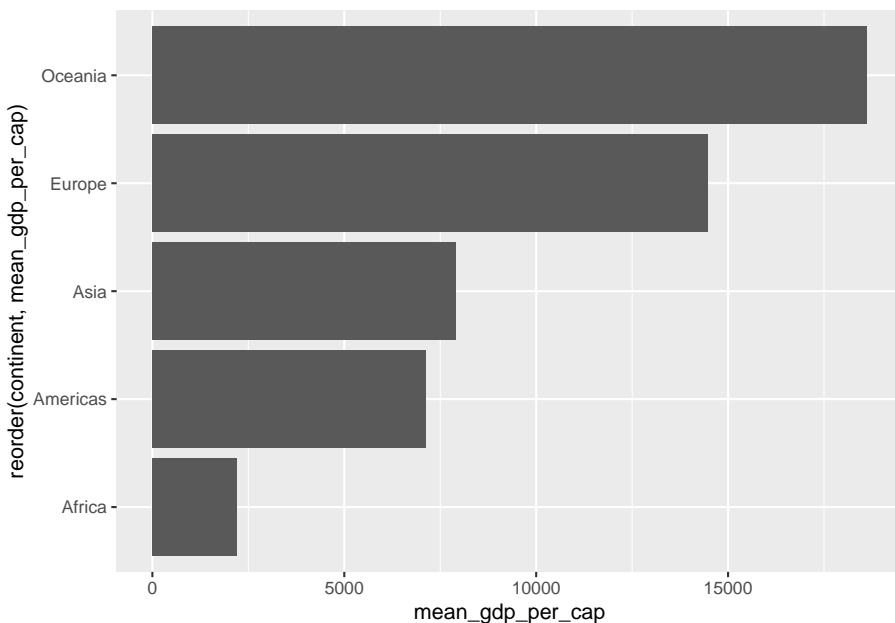
ggplot(tbl_type, aes(y = type, x = _)) +
  geom_col()

## Error: <text>:1:36: unexpected input
## 1: ggplot(tbl_type, aes(y = type, x = _
##
```

4.19 Sortera kategorier

- Ofta vill vi sortera en kategori vi visualiseringar i fallande eller stigande ordning
- Då kan vi bara använda `reorder(x, X = sum)`

```
gap_continent %>%
  ggplot(aes(y = reorder(continent, mean_gdp_per_cap), x = mean_gdp_per_cap)) +
  geom_col()
```



4.20 Sortera din bar-chart efter medianpris

```
ggplot(tbl_type,
       aes(y = reorder(type, __),
           x = median_sq_m_price)) +
  geom_col()

## Error: <text>:2:35: unexpected input
## 1: ggplot(tbl_type,
## 2:           aes(y = reorder(type, _
```

4.21 Tidsserier

Vi har någon form av tidsserie som vi vill visualisera. Oftast har vi då en tidskomponent, exempelvis dag eller månad. I vår hemnet-data har vi datum.

- Kalkylera genomsnittligt försäljningspris per dag och visualisera
- Vilken geom använder du?
- Vilken skala använder du?

```
hem %>%
  group_by(sold_date) %>%
  summarise(median_sq_m_price = median(square_m_price, na.rm = T)) %>%
  ggplot(aes(_, y = median_sq_m_price)) +
  geom_()
```

```
## Error: <text>:4:14: unexpected input
## 3:   summarise(median_sq_m_price = median(square_m_price, na.rm = T)) %>%
## 4:     ggplot(aes(_
##
```

————— OM DU HAR TID ÖVER —————— Kolla på den här listan och gör det som verkar mest intressant!

1. Fri uppgift som går ut på att undersöka ett annat dataset själv och utforska det och se vad du hittar.
2. Lite mera om tidsserier och hur du kan ändra granulariteten på dina tidsserier
3. Testa på enkel interaktivitet i dina grafer
4. Läsa på från en rekommenderad lista
5. Undersöka själv. Du kan självklart fortsätta att undersöka Hemnet datan! Men om du inte vill det så är lite exempel på dataset som du kan använda: **iris** - inbyggt dataset i r som innehåller mätdata om ett par olika blomarter. **iris** finns alltid tillgängligt i r och är bara att kalla på:

mtcars - inbyggt dataset i r som innehåller mätdata om 32 bilmodeller från 1974. **mtcars** finns alltid tillgängligt på samma sätt som **iris**

Pokemon - Dataset som innehåller statistik om alla Pokemon genom tiderna. Hämtat från Kaggle. Laddas med `pokemon <- read_csv('data/pokemon.csv')`

Temp - Globala temperaturer per land och datum från 1743 till 2013 mätt en gång i månaden. Laddas med `temp <- read_csv('data/GlobalTemp.csv')`

Covid-19 - Dataset hämtat från Our World In Data (<https://ourworldindata.org/coronavirus-source-data>), 15/11/2020, som får sin data från the European Center for Disease Control. Innehåller daglig data från hela världen. Det här datasetet är större och svårare att hantera än de andra. Datasetet laddas med `cov <- read_csv('data/covid19-ECDC.csv')`

```
## Börja undersöka här!
# iris
# mtcars
# pokemon <- read_csv('data/pokemon.csv')
# temp <- read_csv('data/GlobalTemp.csv')
# cov <- read_csv('data/covid19-ECDC.csv')
```

2. Mer stabila tidsserier

- Inte sällan vill vi ha lite mindre granulära tidsserier än det vi hade tidigare, exempelvis per vecka eller månad
- Vi kan då använda antigen `lubridate` för att få fram exempelvis `vecka` och `år`
- Eller `tsibble` för att få fram `yearweek` eller `yearmonth`.
- Ladda `tsibble` och `lubridate`
- Skapa en kolumn som representera vecka
- En som representerar år
- En som representerar år-vecka med `tsibble::yearweek()`
- Visualisera per år-vecka
- Visualisera per vecka och sätt `color = år`

```
library(lubridate)
library(tsibble)

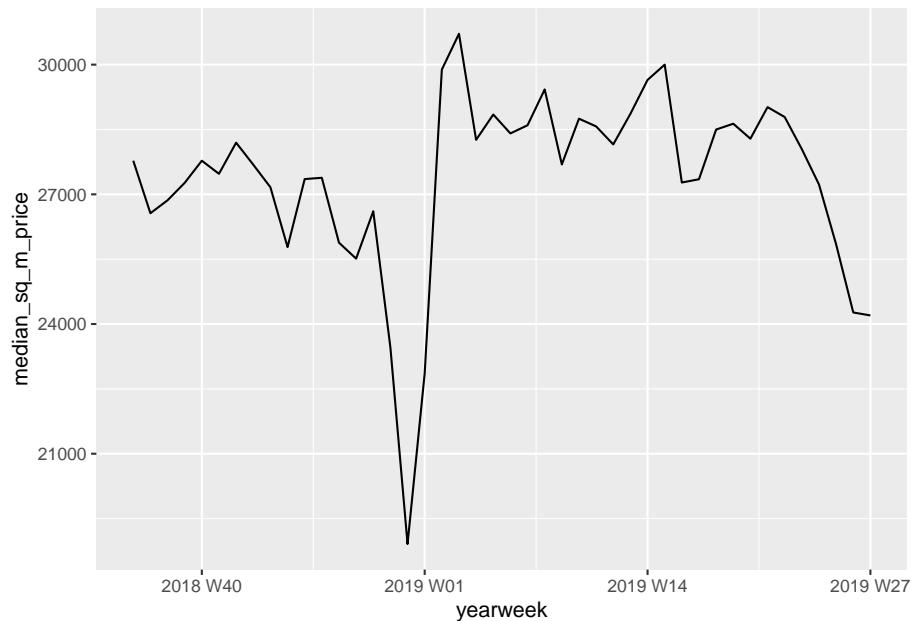
weekly_sales <- hem %>%
  mutate(
    year = as.factor(year(sold_date)),
    week = week(sold_date),
    yearweek = yearweek(sold_date)
  )

p <- weekly_sales %>%
  group_by(yearweek) %>%
```

```
summarise(median_sq_m_price = median(square_m_price, na.rm = T)) %>%
  ggplot(aes(yearweek, median_sq_m_price)) +
  geom_line()
```

```
## `summarise()` ungrouping output (override with `.`groups` argument)
```

P



3. Interaktivitet

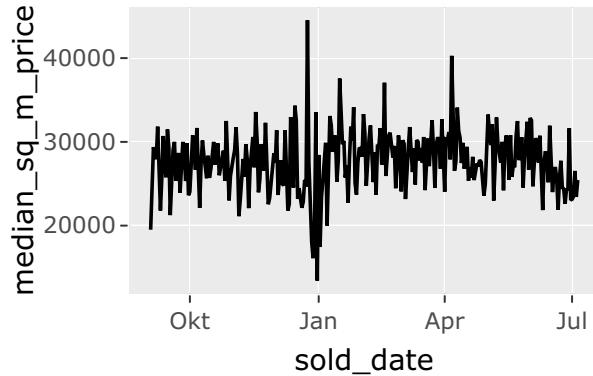
Med paketet `plotly` kan du ta ett `ggplot2`-objekt och göra om det till en interaktiv graf, som kräver en hemsida.

```
library(plotly)
plot <- hem %>%
  group_by(sold_date) %>%
  summarise(median_sq_m_price = median(square_m_price, na.rm = T)) %>%
  ggplot(aes(sold_date, y = median_sq_m_price)) +
  geom_line()

## `summarise()` ungrouping output (override with `.`groups` argument)
```

4.22. GRAMMAR OF GRAPHICS - ARTIKEL OM THE GRAMMAR OF GRAPHICS SOM GGPLOT BYGGER PÅ

```
ggplotly(plot)
```



4. Läslista

4.22 Grammar of Graphics - Artikel om the grammar of graphics som ggplot bygger på

- <https://towardsdatascience.com/a-comprehensive-guide-to-the-grammar-of-graphics-for-effective-visualization-of-multi-dimensional-1f92b4ed4149>

4.23 R for Data Science

- Både kapitel 3 - Data Visualisation och Kapitel 7 - Exploratory Data Analysis är väldigt relevanta

4.24 ggplot2 extensions

- Det finns ett flertal extensions till ggplot
- Gå in och kolla på några på <https://exts.ggplot2.tidyverse.org/gallery/>

4.25 Shiny

- Ett ramverk för att bygga skräddarsydda applikationer i R
- Har likheter med traditionella BI-verktyg men är Open Source och därmed kostnadsfritt
- Är mer flexibelt än ett traditionellt BI-verktyg - allt du kan göra i HTML kan du göra i Shiny.
- Ett otroligt kraftig verktyg för en Data Scientist - automatisera mera
- Innebär att vi kan paketiera analyser som applikationer som användare kan agera med
- Exempelvis ta fram en prognosmodell och låta användare testa att själva ändra paramterar i den, exempelvis vilken typ av säsongsvariation
- <https://shiny.rstudio.com/gallery/>
- <https://shiny.rstudio.com/tutorial/>

Chapter 5

Bilaga

5.1 Variabeltyper i R

De variabeltyper som finns i R är:

| Beskrivning | Synonymer | typeof() | Exempel |
|----------------|------------------------------|-----------|--------------|
| Heltal | int, numeric | integer | -2, 0, 1 |
| Reella tal | real, double, float, numeric | double | 1.03, 0.22 |
| Komplexa tal | cplx | complex | 1+2i |
| Logiska värden | boolean, bool, logi | logical | TRUE, FALSE |
| Faktorer | factors | fctr | “Women” |
| Datum | dates | date | “2017-02-04” |
| Text | string, char | character | “Hello R!” |

`flights` är en så kallad `data.frame` som egentligen är en samling vektorer. `data.frame` är den vanligaste data-typen i R för att arbeta med data.

Om du är nyfiken på att undersöka variabeltypen, eller klassen, på ett objekt eller en kolumn kan du använda funktionen `class()`

```
class(x)
```

```
## [1] "numeric"
```

```
class(text)
```

```
## [1] "character"
```

5.1.1 Vektorer

Vektorer är ett objekt i R som innehåller en eller flera element. En vektor kan endast innehålla en typ av variabel. Det vanligaste sättet att skapa en vektor är genom `c()`.

```
# Numerisk vektor
numvec <- c(1,4,7)

# Textvektor
charvec <- c("Kalle", "Anka")
```

5.1.2 Faktorvariabel

En annan typ av variabel är den såkallade faktorer, *factor*. Faktorer är kategoriska variabler som har olika nivåer som representerar fakta. Kön eller åldersgrupp är exempel på faktorer.

```
# minText är en textvektor
min_text <- c("A", "B", "A", "A")

# Konvertera minText till faktorvariabel med as.factor
min_faktor <- as.factor(min_text)

min_faktor

## [1] A B A A
## Levels: A B

# För att undersöka klassen av vektorn minFaktor
class(min_faktor)

## [1] "factor"

# För att undersöka vilka unika värden finns på minFaktor
levels(min_faktor)

## [1] "A" "B"
```

5.1.3 Logik

Logiska värdena förekommer ofta i programmeringsvärlden. De logiska värdena i R är: **TRUE**, **FALSE** och **NA**.

5.1.3.1 Logiska operatorer

Logiska operatorer är de operatorer som bara kan användas på logiska vektorer. Det kallas även för *boolsk algebra*.

De viktigaste operatorerna i R visas i tabellen nedan.

| Operator | Symbol i R |
|-----------|------------|
| och (and) | & |
| eller(or) | |
| icke(not) | ! |

5.1.3.2 Relationsoperatorer

Man vill ofta jämföra olika vektorer för att sedan indexera en datamängd. Detta görs med hjälp av relationsoperatorer som beskrivs nedan.

1. Använder relationsoperatorer för att jämföra olika variabler.
2. Relationsoperatorer returnerar logiska värde.
3. Logiska värdena används sedan för indexering.

De vanligaste relationsoperatorer i R är följande:

| Relationsoperatorer | Symbol i R |
|----------------------|------------|
| lika | == |
| inte lika | != |
| större än eller lika | >= |
| mindre än eller lika | <= |
| större än | > |
| mindre än | < |
| finns i | %in% |

5.1.4 Matriser

Matriser är också en av de vanligaste datastruktur i R. Den har två dimensioner, rader och kolumner. Likt som vektorer så kan matriser bara innehålla en typ av element. Det innebär att en matris kan endast vara en textmatris, numerisk matris eller logisk matris.

Matriserna kan skapas enkelt i R med funktionen **matrix()**.

```
min_matris <- matrix(data=c(1,2,3,4), nrow=2)
min_matris

##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

5.1.5 Data.frame

`data.frame` är en tabell som innehåller flera olika typer av variabler. Det är egentligen en samling av vektorer, därför kan en `data.frame` innehålla olika typer av vektorer som numeriska, logiska, textsträngar och faktorer.

För att skapa ett dataset i R används funktionen `data.frame()`

```
df <- data.frame(num = 1:3, text = rep("Text", 3), logi=c(TRUE, TRUE, FALSE))
df

##   num text  logi
## 1   1 Text  TRUE
## 2   2 Text  TRUE
## 3   3 Text FALSE
```

Det finns ett antal förinstallerade data frames i R som du kan använda, det är även vanligt med paket vars enda syfte är att innehålla data.

För att kolla upp vilka data frames som finns förinstallerade kan du använda funktionen `data()`.

```
data()

# För att läsa in en data.frame i Global Environment så skriv namn av data inom parentes
data(iris)
iris
```

5.1.5.1 \$

Även om `dplyr` är att föredra för datamanipuleringar är det bra att veta att det går att göra samma manipuleringar i R men med en annan syntax. Då använder man i princip endast de inbyggda paketen (ofta kallade för `base`) som laddas mer R.

En viktig operatör i base-R är `$` (extract-operator) som extraherar värden från objekt baserat på namn.

Exempelvis

```
mtcars$cyl
```

```
## [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

Med \$ kan du också skapa nya variabler.

```
mtcars$ny_variabel <- mtcars$mpg / mtcars$cyl
```

Eller döpa om variabler

```
mtcars$mpg_med_konstigt_namn <- mtcars$mpg
```

5.1.5.2 colnames()

Om du vill ha dina kolumnnamn som en vektor kan du extrahera dessa med `colnames()`.

```
colnames(mtcars)
```

```
## [1] "mpg"                 "cyl"                "disp"
## [4] "hp"                  "drat"               "wt"
## [7] "qsec"               "vs"                 "am"
## [10] "gear"                "carb"               "ny_variabel"
## [13] "mpg_med_konstigt_namn"
```

I bland kanske du har en lista på variabelnamn som du vill att data ska ha.

```
kolumnnamn <- c("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K")
```

Då kan du applicera `colnames()` med din vektor på din `data.frame`.

```
colnames(mtcars) <- kolumnnamn
mtcars
```

| | A | B | C | D | E | F | G | H | I | J | K | NA | NA |
|----------------------|------|---|-------|-----|------|-------|-------|---|---|---|---|----------|------|
| ## Mazda RX4 | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.620 | 16.46 | 0 | 1 | 4 | 4 | 3.500000 | 21.0 |
| ## Mazda RX4 Wag | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | 4 | 4 | 3.500000 | 21.0 |
| ## Datsun 710 | 22.8 | 4 | 108.0 | 93 | 3.85 | 2.320 | 18.61 | 1 | 1 | 4 | 1 | 5.700000 | 22.8 |
| ## Hornet 4 Drive | 21.4 | 6 | 258.0 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 | 3.566667 | 21.4 |
| ## Hornet Sportabout | 18.7 | 8 | 360.0 | 175 | 3.15 | 3.440 | 17.02 | 0 | 0 | 3 | 2 | 2.337500 | 18.7 |
| ## Valiant | 18.1 | 6 | 225.0 | 105 | 2.76 | 3.460 | 20.22 | 1 | 0 | 3 | 1 | 3.016667 | 18.1 |

```

## Duster 360      14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4 1.787500 14.3
## Merc 240D      24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2 6.100000 24.4
## Merc 230       22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2 5.700000 22.8
## Merc 280       19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4 3.200000 19.2
## Merc 280C      17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4 2.966667 17.8
## Merc 450SE     16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3 2.050000 16.4
## Merc 450SL      17.3 8 275.8 180 3.07 3.730 17.60 0 0 3 3 2.162500 17.3
## Merc 450SLC    15.2 8 275.8 180 3.07 3.780 18.00 0 0 3 3 1.900000 15.2
## Cadillac Fleetwood 10.4 8 472.0 205 2.93 5.250 17.98 0 0 3 4 1.300000 10.4
## Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82 0 0 3 4 1.300000 10.4
## Chrysler Imperial 14.7 8 440.0 230 3.23 5.345 17.42 0 0 3 4 1.837500 14.7
## Fiat 128        32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1 8.100000 32.4
## Honda Civic     30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2 7.600000 30.4
## Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1 8.475000 33.9
## Toyota Corona   21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1 5.375000 21.5
## Dodge Challenger 15.5 8 318.0 150 2.76 3.520 16.87 0 0 3 2 1.937500 15.5
## AMC Javelin     15.2 8 304.0 150 3.15 3.435 17.30 0 0 3 2 1.900000 15.2
## Camaro Z28      13.3 8 350.0 245 3.73 3.840 15.41 0 0 3 4 1.662500 13.3
## Pontiac Firebird 19.2 8 400.0 175 3.08 3.845 17.05 0 0 3 2 2.400000 19.2
## Fiat X1-9        27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1 6.825000 27.3
## Porsche 914-2    26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2 6.500000 26.0
## Lotus Europa     30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2 7.600000 30.4
## Ford Pantera L   15.8 8 351.0 264 4.22 3.170 14.50 0 1 5 4 1.975000 15.8
## Ferrari Dino     19.7 6 145.0 175 3.62 2.770 15.50 0 1 5 6 3.283333 19.7
## Maserati Bora    15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8 1.875000 15.0
## Volvo 142E       21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2 5.350000 21.4

```

5.1.6 Listor

Listor är mer generell datastruktur i R. Man kan jämföra lista med en “låda” som innehåller olika datastrukturer i R. En lista kan innehålla vektorer, data.frame och matriser, i olika längd och storlek, samtidigt.

För att skapa en lista används funktionen `list()`

```

# skapar vektorer a, b, c
a <- c("hej", "Kalle", "hejdå")
b <- c(TRUE, FALSE, FALSE, TRUE)
c <- 1:8

x <- list(a,b,c)
x

```

```

## [[1]]
## [1] "hej"    "Kalle"  "hejdå"

```

```
##  
## [[2]]  
## [1] TRUE FALSE FALSE TRUE  
##  
## [[3]]  
## [1] 1 2 3 4 5 6 7 8
```

Ett mycket vanligt exempel på en lista är modellobjekt, dvs objektet man skapar när man anpassar en modell, till exempel en regressionsmodell. Hela resultatet från modellanpassningen ligger på en lista. Se nedan.

```
# Skapar en data.frame  
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)  
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)  
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))  
weight <- c(ctl, trt)  
  
# Linjär regression.  
lm <- lm(weight ~ group)  
  
is.list(lm)  
  
## [1] TRUE  
  
# Vilka element eller objekt som finns i listan lm?  
attributes(lm)  
  
## $names  
## [1] "coefficients"   "residuals"      "effects"       "rank"  
## [5] "fitted.values"  "assign"        "qr"           "df.residual"  
## [9] "contrasts"       "xlevels"       "call"          "terms"  
## [13] "model"  
##  
## $class  
## [1] "lm"
```

lm är en lista som innehåller bland annat regressions koefficienter, residualer, anpassade värden, mm.

5.1.7 Datum

I R uttrycks datum som antal dagar från 1970-01-01. Datum som kommer innan det skrivs ut som negativa värde.

```

minaDatum <- as.Date(c("2018-08-14", "1970-01-01"))

minaDatum[1] - minaDatum[2]

## Time difference of 17757 days

as.Date(17757, origin = "1970-01-01")

## [1] "2018-08-14"

```

Sys.Date() ger dagens datum. **date()** returnerar dagens veckodag, datum och tid.

5.1.7.1 Olika datumformat

| Symbol i R | Betydelse | Exempel |
|------------|--------------------|---------|
| %d dag | dagnummer | 01-31 |
| %a | förkortad veckodag | Mon |
| %A ful | full veckodag | Monday |
| %m mån | månad nummer | 00-12 |
| %b | förkortad månad | Jan |
| %B ful | full månad | January |
| %y | 2 siffrig år | 18 |
| %Y 4 s | 4 siffrig år | 2018 |

5.1.7.2 Formatera datum

```

idag <- Sys.Date()

format_idag <- format(idag, format = "%B %d %Y")
format_idag

## [1] "November 16 2020"

format(minaDatum[2], format="%m/%d/%y")

## [1] "01/01/70"

```

5.2 Indexering

Ibland är man bara intresserad av en kolumn eller enstaka värde. För att indexera eller plocka ut specifikt element ur ett objekt kan man använda sig av hackparentes [] eller dollar tecken \$ och detta görs på olika sätt. ### Genom att ange positions nummer

```
#Vektor
testScore <- c(7, 15, 20, 20, 23, 1.2)
testScore[3] #tredje element
```

```
## [1] 20
```

```
testScore[-1] #alla utom den första
```

```
## [1] 15.0 20.0 20.0 23.0 1.2
```

Kör koden nedan för att se resultatet.

```
#Data.frame.
data(iris)
iris[, 1] # Väljer kolumn nummer 1
iris[1:5, c(1,2)] # Rad 1 till 5 på Kolumn 1 och 2
```

5.2.1 Genom att begränsa värden som man vill plocka ut

```
# Vektor
testScore[testScore < 10] # alla som är mindre än 10
```

```
## [1] 7.0 1.2
```

```
testScore[testScore == 20] # alla som är lika med 20
```

```
## [1] 20 20
```

Kör koden nedan för att se resultatet.

```
# Data.frame
head(iris[iris[,1] > 7,]) # alla observationer där kolumn nummer 1 är större än 7. Funktionen he
```

5.2.2 Genom att ange namn av elementet eller kolumnen

Det går att namnge element i en vektor. Koder nedan visar hur man ger namn till varje element i en vektor. Efter att ha namngott vektorn kan du nu plocka ut värde med hjälp av namn.

```
# Sätter namn på respektive element i testScore
names(testScore) <- c("Zaida", "Anna", "Helena", "Per", "Filip", "Annelie")
testScore

##   Zaida    Anna  Helena     Per    Filip Annelie
##     7.0    15.0    20.0    20.0    23.0     1.2

testScore["Per"]

## Per
## 20

testScore["Filip"]

## Filip
## 23

# Kolumnen i en data.frame oftast har namn. Objekt som har 2 eller mer dimensioner kan
colnames(iris)

## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```