R-anvisningar till *Grundläggande statistik*

2022-07-26

Contents

Chapter 1

Introduktion

Detta dokument är en kort introduktion till R för en kurs i grundläggande statistik.

Chapter 2

Installation

2.1 Inledning

För att köra R-kod på sin dator krävs en installation av programspråket R. För att effektivt arbeta i R används ofta en utvecklingsmiljö (ett tilläggsprogram som på flera sätt förenklar arbetet) och här ges anvisningar till den vanligaste utvecklingsmiljön för R, som är RStudio. För att komma ingång måste man alltså installera R och RStudio.

2.2 Installation av R

Programspråket R kan laddas ner från https://www.r-project.org/ med följande steg:

1. Klicka på *CRAN* längst upp till vänster.
2. Klicka på den översta länken under 0-Cloud.
3. Välj en nedladdning beroende på operativsystem.
4. För Windows, välj *base*. För macOS, välj den senaste tillgängliga versionen.
5. Installera R från den nedladdade filen. Installation sker som för andra nedladdade program.

2.3 Installation av RStudio

RStudio kan laddas ner från https://www.rstudio.com/ med följande steg:

1. Klicka på *Download* uppe till höger.
2. Scrolla nedåt och välj *Download* under *RStudio Desktop*.
3. Klicka på nedladdningsknappen.

4. Installera RStudio från den nedladdade filen. Installation sker som för andra nedladdade program.

2.4 Gränssnittet i RStudio

När man nu öppnar RStudio ser man att fönstret är uppdelat i fyra delar och att varje del består av en eller flera flikar. De viktigaste är i nuläget

- *Console* där kod körs och resultat skrivs ut,
- *Environment* där man ser skapade objekt,
- *History* där man ser tidigare körd kod,
- *Plots* där man ser skapade grafer, och
- *Help* där man ser hjälpsidor för funktioner.

Ofta skriver man inte sin kod direkt i konsollen, utan i ett separat *skript* - en vanlig textfil som innehåller den kod man vill köra. Genom att organisera sin kod i ett skript kan man lätt strukturera och dokumentera sitt arbete. I RStudio kan man öppna ett nytt skript genom att gå till *File > New File > R Script* eller genom att klicka *Ctrl + Shift + N*. Ett tomt skript öppnar sig då i det övre vänstra delfönstret. Om man skriver

```
a <- 5
```

i skriptet och trycker *Ctrl + Enter* bör man se att koden i skriptet körs i konsollen. Om man tittar i fliken *Environment* ska man också se att det nu skapats ett objekt *a*.

2.5 Paket i R

En av de stora styrkorna med R är att språket kan byggas ut av dess användare. De här tilläggen kan sedan samlas i paket (*packages*) och delas med andra. Rs officiella bibliotek för paket kallas för *CRAN* (*Comprehensive R Archive Network*) och består av mer än 18 000 uppladdade paket som innehåller allt från fritt tillgänglig data till avancerade statistiska modeller.

För att använda ett specifikt paket måste det först installeras. Om man vet namnet på paketet man vill installera kan man köra

```
install.packages("tidyverse")
```

I det här fallet installeras paketet `tidyverse`, vilket innehåller funktioner för hantering av data.

I RStudio kan man också installera paket från *Packages*-fliken.

Paket måste också laddas för varje ny session. Innan man kan använda innehållet i ett paket måste man därför köra

```
library(tidyverse)
```

Chapter 3

Datorövning 1

Datorövning 1 handlar om grunderna till R. Efter övningen ska vi kunna

- Starta RStudio och orientera oss i gränssnittet,
- Installera och ladda tilläggspaket (*Packages*)
- Definera objekt och tillämpa funktioner i R,
- Importera data från en online-källa,
- Transformera en tabell med data genom att välja kolumner, filtrera rader och summera per grupp,
- Skapa grafer med `ggplot2`.

3.1 Uppstart och orientering

För att arbeta i R måste vi installera språket R och ett gränssnitt för att arbeta i R, vanligen *RStudio*. Titta på kapitlet *Installation* om programmen inte är installerade på ditt system.

Starta RStudio, till exempel genom att gå till Startmenyn och söka på RStudio eller genom att dubbelklicka på en fil som öppnas i RStudio. Gränssnittet i RStudio är uppdelat i fyra delar och varje del består av en eller flera flikar. De viktigaste är i nuläget

- *Console* där kod körs och resultat skrivs ut,
- *Environment* där man ser skapade objekt,
- *History* där man ser tidigare körd kod,
- *Plots* där man ser skapade grafer, och
- *Help* där man ser hjälpsidor för funktioner.

Uppgift 3.1 (Help-fliken)**.** Hitta fliken *Help*, klicka på husikonen under fliken. Finns det en länk med *RStudio Cheat Sheets*? Följ den länken för att hitta guider till R som kan bli nyttiga längre från. För nu, gå tillbaka till RStudio.

Ofta skriver man inte sin kod direkt i konsollen, utan i ett separat *skript* - en vanlig textfil som innehåller den kod man vill köra. Genom att organisera sin kod i ett skript kan man lätt strukturera och dokumentera sitt arbete. I RStudio kan man öppna ett nytt skript genom att gå till *File > New File > R Script* eller genom att klicka *Ctrl + Shift + N*. Ett tomt skript öppnar sig då i det övre vänstra delfönstret. Om du läser det här i RStudio, genom att ha laddat ner .R-filen, läster du just nu ett skript.

Uppgift 3.2 (Ett första skript)**.** Öppna ett nytt skript genom File-menyn eller genom *Ctrl + Shift + N*. Skriv

```
a <- 5
```

i skriptet och tryck *Ctrl + Enter*. Titta i flikarna *Console* och *Environment*. Har något hänt? Du bör se att koden i skriptet körts i konsollen och att ett nytt objekt `a` ligger i *Environment*.

3.2 *Packages* från CRAN

En av de stora styrkorna med R är att språket kan byggas ut av dess användare. De här tilläggen kan sedan samlas i paket (*packages*) och delas med andra. Rs officiella bibliotek för paket kallas för *CRAN* (*Comprehensive R Archive Network*) och består av mer än 18 000 uppladdade paket som innehåller allt från fritt tillgänglig data till avancerade statistiska modeller.

För att använda ett specifikt paket måste det först installeras. Om man vet namnet på paketet man vill installera kan man köra

```
install.packages("tidyverse")
```

I det här fallet installeras paketet `tidyverse`, vilket innehåller funktioner för hantering av data.

I RStudio kan man också installera paket från *Packages*-fliken.

Uppgift 3.3 (Installera tidyverse-paketet)**.** Kör raden ovan för att installera `tidyverse`. Du kan antingen köra raden genom att skriva den i *Console* eller genom att skriva i ett skript och köra därifrån genom *Ctrl + Enter*.

Uppgift 3.4 (Installera gapminder-paketet)**.** Paketet `gapminder` innehåller lite intressant data vi kommer använda senare. Installera paketet `gapminder` genom att fylla i och köra raden nedan.

```
install.packages("___")
```

Paket måste också laddas för varje ny session. Innan man kan använda innehållet i ett paket måste man därför köra

```
library(tidyverse)
```

Uppgift 3.5 (Ladda gapminder-paketet)**.** Ladda paketet `gapminder` genom att fylla i och köra raden nedan.

```
library(___)
```

Uppgift 3.6 (Paket som inte finns)**.** Vad händer om man försöker installera ett paket som inte finns på *CRAN* ? Testa till exempel

```
install.packages("ThisIsNotTheNameOfAnyPackage")
```

och

```
library(ThisIsNotTheNameOfAnyPackage)
```

3.3 Objekt och funktioner

Ett *objekt* i R är en namngiven informationsmängd. Objekt kan se ut på många olika sätt - under kursens gång används objekt som består av insamlad data (konstruerade som vektorer eller tabeller), objekt som är statistiska modeller, och flera andra former. I R skapar man objekt med *assign*-pilen `<-` (mindre än och bindestreck).

I ett tidigare exempel fanns raden

```
a <- 5
```

Här skapas ett objekt med namnet `a` som innehåller informationen `5`. *Assign*-pilen pekar alltså på det namn man vill ge objektet och pekar från objektets innehåll.

Ett lite mer komplicerat exempel på ett objekt ges av

```
b <- c(3, 1, 4, 1, 5, 9)
```

Här skapas ett objekt `b` som innehåller en *serie* numeriska värden (en *vektor*).

Uppgift 3.7 (Skapa en vektor)**.** Skapa ett objekt med namnet `new_vector` som innehåller värden 5, 7 och 10 genom att fylla i följande rad.

```
new_vector <- c(_, _, _)
```

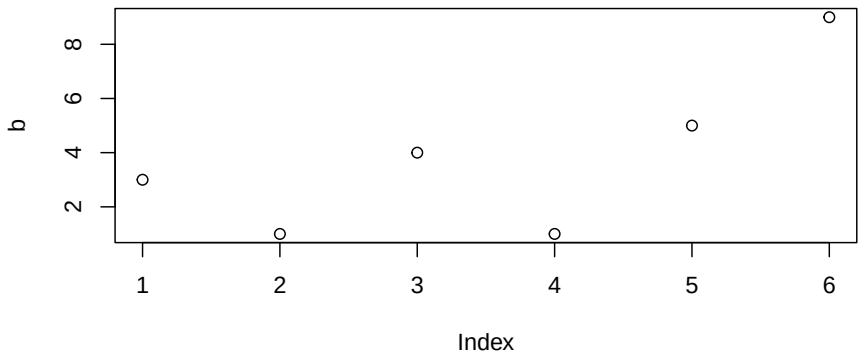
Objekt kan manipuleras genom att tillämpa *funktioner*. En funktion tar någon ingående data och ger något utgående resultat. Funktioner anges genom att skriva funktionens namn följt av ingående data inom parenteser, och resultatet kan antingen skrivas ut i konsollen eller sparas som ett nytt objekt. En grundinstallation av R innehåller en mängd färdiga funktioner, t.ex.

```
sum(b)
```

```
## [1] 23
```

vilket ger summan av värdena i vektorn `b`,

```
plot(b)
```



som ger en simpel graf, och

```
sqrt(b)
```

```
## [1] 1.732051 1.000000 2.000000 1.000000 2.236068 3.000000
```

som beräknar kvadratroten för varje element i vektorn.

Uppgift 3.8 (Summera vektorn)**.** Fyll i och kör följande rad för att beräkna summan av vektorn `new_vector`

```
sum(___)
```

Vid konstruktionen av vektorn användes också en grundläggande funktion - funktionen `c` som tar en serie värden och skapar en sammanhängande vektor av värden.

Alla R-funktioner har en tillhörande hjälpfil som kan plockas fram genom att skriva frågetecken följt av funktionsnamnet, t.ex. `?sum`. Från hjälpfilen får man att `sum` tar numeriska vektorer som ingående värde och beräknar summan. Man kan styra funktionens beteende genom att sätta ett argument `na.rm` (vilket här styr hur funktionen hanterar saknade värden). Som illustration kan man titta på

```
b <- c(3, 1, 4, 1, 5, 9, NA)  # Lägger till ett saknat värde
sum(b)                        # na.rm = FALSE är grundinställning
```

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

```
sum(b, na.rm = TRUE)          # na.rm sätts till TRUE
```

```
## [1] 23
```

Det första försöket `sum(b)` ger utfallet `NA`, men om man sätter `na.rm = TRUE` beräknas summan efter att det saknade värdet plockats bort. Notera också att skript kan kommenteras med `#`.

3.4 Sekvenser av funktioner

Ofta vill man genomföra flera operationer på ett objekt. Man behöver då genomföra en sekvens av funktioner. Säg till exempel att man har värdena

$$(-4, -2, -1, 1, 2, 4)$$

och vill ta absolutvärde (vilket gör negativa tal till motsvarande positiva tal) och sedan summera. Den typen av sekvenser kan genomföras på ett par olika sätt. Ett första sätt är att spara resultatet i varje steg och sedan använda utfallet i nästa steg:

```
c <- c(-4, -2, -1, 1, 2, 4)
c_absolute <- abs(c)
sum(c_absolute)
```

```
## [1] 14
```

Här skapas ett objekt `c` som innehåller en vektor där några tal är negativa. I nästa rad används `abs` för att skapa absolutvärden. Slutligen summeras absolutvärdena med `sum`. Notera att det är möjligt att skapa ett objekt med namnet `c` trots att det redan är namnet på en funktion - R förstår ur sammanhanget om objektet eller funktionen ska användas.

Uppgift 3.9 (Kvadrat, summa och roten ur)**.** Fyll i och kör följande rader för att ta varje värde i `new_vector` i kvadrat, *sedan* summera, och sedan ta roten ur.

```
new_vector_squared <- new_vector^2 # Ta kvadraten av varje värde
new_vector_squared_sum <- sum(___) # Summera vektorn med kvadrater
sqrt(___)                          # Ta kvadratroten ur summan
```

Ett alternativ är att skriva en senare funktion *runt* en tidigare funktion. Det fungerar för att R utvärderar funktioner inifrån-ut. Med samma exempel som tidigare får man

```
sum(abs(c(-4, -2, -1, 1, 2, 4)))
```

medan beräkningen i övningen blir

```
sqrt(sum(new_vector^2))
```

Den här typen av skrivning kan spara plats men blir snabbt svårläst.

Ett sista alternativ är att använda en så kallad *pipe* (namnet kommer från att en sekvens funktioner kallas en *pipeline*). En pipe skrivs `%>%` och tar utfallet av en funktion till vänster och sänder till en funktion till höger. Språkligt kan pipen utläsas *och sen*. Funktionen kan laddas genom att ladda paketet `tidyverse`. Med samma exempel som tidigare kan vi skriva

```
library(tidyverse)

c(-4, -2, -1, 1, 2, 4) %>%  # Skapa en datamängd och sen
  abs() %>%                 # ta absolutvärden, och sen
  sum()                     # beräkna summan.
```

```
## [1] 14
```

Uppgift 3.10 (Kvadrat, summa och rot med pipe)**.** Fyll i de saknade funktionerna och kör följande rader för att ta varje värde i `new_vector` i kvadrat, *sedan* summera, och sedan ta roten ur, denna gång genom att länka funktionerna med en pipe `%>%`.

```
new_vector^2 %>%        # Ta kvadraterna av new_vector, och sen
  ___() %>%             # beräkna summan, och sen
  ____()                # Ta kvadratroten (med sqrt())
```

3.5 Datainskrivning och dataimport från web

3.5.1 Inskrivning av data

Det första praktiska steget i en statistisk analys är att importera data. I R kan det göras genom att direkt skriva in sin data och spara som ett nytt objekt, men ett bättre och vanligare sätt är att importera sin data från en extern fil eller databas.

I ett tidigare exempel användes funktionen `c` för att skapa en vektor av data. Ofta ordnas flera vektorer i en tabell där varje kolumn är en vektor och varje rad en observation av någon enhet. En datatabell (en `data.frame` i R) skapas genom funktionen `data.frame()` följt av namngivna vektorer. Exempeldata kan skrivas in genom föjande.

```
dat <- data.frame(Vecka = c(7,7,7,7,7,7,11,11,11,11,11,11),
                  Behandling = c("A","A","A","B","B","B","A","A","A","B","B","B"),
                  Vikt = c(232,161,148,368,218,257,1633,2213,972,2560,2430,855),
                  N = c(2.63,2.90,2.99,3.54,3.30,2.85,1.53,1.90,NA,2.58,NA,NA))

dat
```

```
##    Vecka Behandling Vikt    N
## 1      7          A  232 2.63
## 2      7          A  161 2.90
## 3      7          A  148 2.99
## 4      7          B  368 3.54
## 5      7          B  218 3.30
## 6      7          B  257 2.85
## 7     11          A 1633 1.53
## 8     11          A 2213 1.90
## 9     11          A  972   NA
## 10    11          B 2560 2.58
## 11    11          B 2430   NA
## 12    11          B  855   NA
```

Radbrytningar och blanksteg är oviktiga i R, och används bara för läsbarhet här. Saknade värden skrivs in som `NA` för *not available*. Notera att alla kolumner inte behöver vara av samma datatyp men att värden inom en kolumn måste vara det. Här är *Behandling* text medan övriga kolumner är numeriska.

Uppgift 3.11 (Alea iacta est)**.** Kasta din tärning tio gånger och skriv in resultatet i en datatabell i R med hjälp av grundkoden nedan. Om du saknar en tärning, fråga lämplig person om du kan få en. Behåll tärningen, den behövs till nästa datorövning (och närhelst man står inför ett avgörande livsbeslut).

```
dat_dice <- data.frame(Kast = c(1,2,3,4,5,6,7,8,9,10),
                       Utfall = c(_,_,_,_,_,_,_,_,_,_))
dat_dice
```

3.5.2 Import från en extern fil

Inskrivning av värden är ofta tidskrävande och kan lätt leda till misstag. Det är därför mycket vanligare att data läses in från en extern fil. Det finns en mängd funktioner för dataimport och det exakta valet av funktion beror på vilken typ av fil datan är sparad i. Ett vanligt filformat är .csv (*comma separated values*). Här importerar vi en fil med data från Spotify.

```
dat <- read_csv("https://raw.githubusercontent.com/adamflr/ST0060/master/Data/Spotify_data.csv")
dat
```

```
## # A tibble: 76,622 x 24
##    artist_name album_name    track_number track_name album_type album_release_d~
##    <chr>       <chr>                <dbl> <chr>      <chr>      <chr>
##  1 100 gecs    1000 gecs an~            1 money mac~ album      2020-07-10
##  2 100 gecs    1000 gecs an~            2 ringtone ~ album      2020-07-10
##  3 100 gecs    1000 gecs an~            3 745 stick~ album      2020-07-10
##  4 100 gecs    1000 gecs an~            4 gec 2 Ü (~ album      2020-07-10
##  5 100 gecs    1000 gecs an~            5 hand crus~ album      2020-07-10
##  6 100 gecs    1000 gecs an~            6 800db clo~ album      2020-07-10
```

```
##  7 100 gecs    1000 gecs an~            7 stupid ho~ album      2020-07-10
##  8 100 gecs    1000 gecs an~            8 ringtone ~ album      2020-07-10
##  9 100 gecs    1000 gecs an~            9 xXXi_wud_~ album      2020-07-10
## 10 100 gecs    1000 gecs an~           10 745 stick~ album      2020-07-10
## # ... with 76,612 more rows, and 18 more variables: album_release_year <dbl>,
## #   danceability <dbl>, energy <dbl>, key <dbl>, loudness <dbl>, mode <dbl>,
## #   speechiness <dbl>, acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, time_signature <dbl>,
## #   explicit <lgl>, type <chr>, key_name <chr>, mode_name <chr>, key_mode <chr>
```

Om importen fungerat kan man skriva ut tabellens översta rader genom att köra objektets namn.

För att snabbt se vilka artister datan täcker kan man köra

```
unique(dat$artist_name)
```

där `unique` är en funktion som tar bort alla dubletter och `dat$artist_name` används för att plocka ut kolumnen `artist_name` ur tabellen `dat`.

3.6 Transformera en tabell med `select`, `filter`, `mutate` och `summarise`

3.6.1 Urval med `select` och `filter`

En vanlig operation på en tabell är att göra ett urval - antingen ett urval av rader (t.ex. en viss artist), vilket kallas *filtrering* eller ett urval av variabler (t.ex. artist och albumnamn), vilket kallas *selektion*. Det finns flera olika sätt att göra ett urval i R. Det traditionella sättet är att använda index inom hakparenteser (t.ex. `dat[4, 2]` för fjärde raden, andra kolumnen) eller dollartecken för specifika kolumner (t.ex. `dat$artist_name` för artistnamn). Här fokuseras dock på hur det kan göras med funktionerna `filter` och `select` från paketet `tidyverse`.

För att filtrera på ett givet land kan använda pipe-funktionen från datan till en filter-funktion, t.ex.

```
dat %>%                           # Ta spotify-datan och sen
  filter(artist_name == "Robyn") # filtrera för en specifik artist
```

```
## # A tibble: 1,142 x 24
##    artist_name album_name track_number track_name    album_type album_release_d~
##    <chr>       <chr>              <dbl> <chr>         <chr>      <chr>
##  1 Robyn       Honey                  1 Missing U     album      2018-10-26
##  2 Robyn       Honey                  2 Human Being   album      2018-10-26
##  3 Robyn       Honey                  3 Because It's~ album      2018-10-26
##  4 Robyn       Honey                  4 Baby Forgive~ album      2018-10-26
##  5 Robyn       Honey                  5 Send To Robi~ album      2018-10-26
##  6 Robyn       Honey                  6 Honey         album      2018-10-26
```

```
##  7 Robyn       Honey                 7 Between The ~ album      2018-10-26
##  8 Robyn       Honey                 8 Beach2k20     album      2018-10-26
##  9 Robyn       Honey                 9 Ever Again    album      2018-10-26
## 10 Robyn       Honey                 1 Missing U     album      2018-10-26
## # ... with 1,132 more rows, and 18 more variables: album_release_year <dbl>,
## #   danceability <dbl>, energy <dbl>, key <dbl>, loudness <dbl>, mode <dbl>,
## #   speechiness <dbl>, acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, time_signature <dbl>,
## #   explicit <lgl>, type <chr>, key_name <chr>, mode_name <chr>, key_mode <chr>
```

Inom filter-funktionen anges ett logisk villkor `country == "Sweden"` och utfallet är de rader där villkoret är sant. Notera de dubbla likhetstecknen - de måste användas för ett logisk villkor eftersom enkelt likhetstecken används för att skapa objekt och sätta funktionsargument.

Uppgift 3.12 (Filtrera för artist)**.** Vad måste ändras i koden för att istället plocka ut rader där artisten är Esperanza Spalding? Hur många rader har det urvalet?

```
dat %>%                           # Ta spotify-datan och sen
  filter(artist_name == "Robyn") # filtrera för en specifik artist
```

Om man vill välja flera artister kan man använda funktionen `%in%` på ett liknande sätt.

```
dat %>%
  filter(artist_name %in% c("Robyn", "Esperanza Spalding"))
```

och om man vill ha mer än ett villkor kan man rada dem i filter-funktionen eller ha flera filter-steg:

```
dat %>%
  filter(artist_name %in% c("Robyn", "Esperanza Spalding"),
         key_name == "D#")
```

alternativt

```
dat %>%
  filter(artist_name %in% c("Robyn", "Esperanza Spalding")) %>%
  filter(key_name == "D#")
```

För att se fler eller färre rader kan man använda en pipe `%>%` till funktionen `print`. Följande skriver ut fem rader

```
dat %>%
  filter(artist_name %in% c("Robyn", "Esperanza Spalding")) %>%
  filter(key_name == "D#") %>%
  print(n = 5)
```

```
## # A tibble: 21 x 24
```

```
##   artist_name     album_name track_number track_name album_type album_release_d~
##   <chr>           <chr>              <dbl> <chr>      <chr>      <chr>
## 1 Esperanza Spal~ SONGWRIGH~             7 Formwela 7 album      2021-09-24
## 2 Esperanza Spal~ 12 Little~             2 To Tide U~ album      2019-05-10
## 3 Esperanza Spal~ 12 Little~             3 'Til the ~ album      2019-05-10
## 4 Esperanza Spal~ Emily's D~             7 Ebony And~ album      2016-01-01
## 5 Esperanza Spal~ Emily's D~             7 Ebony And~ album      2016-01-01
## # ... with 16 more rows, and 18 more variables: album_release_year <dbl>,
## #   danceability <dbl>, energy <dbl>, key <dbl>, loudness <dbl>, mode <dbl>,
## #   speechiness <dbl>, acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, time_signature <dbl>,
## #   explicit <lgl>, type <chr>, key_name <chr>, mode_name <chr>, key_mode <chr>
```

Om man istället vill göra ett urval av kolumner kan man använda `select`. Som argument anges de kolumner man vill välja, t.ex.

```
dat %>%
  select(artist_name, album_name)
```

```
## # A tibble: 76,622 x 2
##    artist_name album_name
##    <chr>       <chr>
##  1 100 gecs    1000 gecs and The Tree of Clues
##  2 100 gecs    1000 gecs and The Tree of Clues
##  3 100 gecs    1000 gecs and The Tree of Clues
##  4 100 gecs    1000 gecs and The Tree of Clues
##  5 100 gecs    1000 gecs and The Tree of Clues
##  6 100 gecs    1000 gecs and The Tree of Clues
##  7 100 gecs    1000 gecs and The Tree of Clues
##  8 100 gecs    1000 gecs and The Tree of Clues
##  9 100 gecs    1000 gecs and The Tree of Clues
## 10 100 gecs    1000 gecs and The Tree of Clues
## # ... with 76,612 more rows
```

Som avslutning ges ett lite mer komplicerat exempel på ett urval av artist, år och spår för spår med ett tempo över 180 bpm släppta under 2015.

```
dat %>%                                                  # Ta datan och sen
  filter(album_release_year == 2015,                     # filtrera för rader där år är 2
         tempo > 180) %>%                                # tempot över 160, och sen
  select(artist_name, album_release_year, track_name) # selektera på artist, år och sp
```

```
## # A tibble: 122 x 3
##    artist_name            album_release_year track_name
##    <chr>                                <dbl> <chr>
##  1 A Sunny Day In Glasgow                2015 The Strange Presents of Idols (Mer~
##  2 A$AP Rocky                            2015 Lord Pretty Flacko Jodye 2 (LPFJ2)
##  3 A$AP Rocky                            2015 Lord Pretty Flacko Jodye 2 (LPFJ2)
```

```
##  4 Anderson .Paak                        2015 Off the Ground
##  5 Björk                                 2015 Notget - Live
##  6 Björk                                 2015 Notget
##  7 Björk                                 2015 Notget
##  8 Björk                                 2015 Notget
##  9 Blood Orange                          2015 Sandra's Smile
## 10 Burna Boy                             2015 Soke
## # ... with 112 more rows
```

Uppgift 3.13 (Snabba spår)**.** Funktionen `arrange` sorterar data efter en angiven kolumn. Följande stycke ger oss Björks snabbaste spår.

```
dat %>%                                                    # Ta datan, och sen
  filter(artist_name == "Björk") %>%                       # filtrera för rader där artist är Björ
  select(artist_name, album_name, track_name, tempo) %>%   # välj kolumner med artist, album, spår
  arrange(-tempo)                                          # ordna efter tempo (minus för fallande
```

Gör lämpliga ändringar för att hitta Kate Bushs snabbaste spår. Gör ytterligare ändringar för att hitta Daft Punks långsammaste spår.

3.6.2 Ändra och skapa nya kolumner med `mutate`

Variabler kan omräknas och nya variabler kan skapas med `mutate`-funktionen. I spotify-datan finns tempo som slag per minut. Om man vill ha slag per sekund kan man skapa en nya kolumn och beräkna den som tempo delat på 60.

```
dat <- dat %>%
  mutate(beats_per_second = tempo / 60)
```

Den inledande delen med `dat <-` gör så att utfallet av beräkningen sparas i objektet `dat`. Vi kan skriva ut objektet och se resultatet av beräkningen:

```
dat %>% select(tempo, beats_per_second)
```

```
## # A tibble: 76,622 x 2
##    tempo beats_per_second
##    <dbl>            <dbl>
##  1  132.             2.20
##  2  107.             1.78
##  3  128.             2.14
##  4  113.             1.89
##  5  174.             2.89
##  6  142.             2.36
##  7  166.             2.77
##  8  125.             2.08
##  9  170.             2.83
## 10  150.             2.50
## # ... with 76,612 more rows
```

Uppgift 3.14 (Glädje per dansbarhet)**.** Följande stycke beräknar kvoten av kolumnerna valence och danceability i en ny kolumn 'valence_to_danceability.

```
dat %>%                                                    # Ta datan, och sen
  mutate(valence_to_danceability = valence / danceability) %>% # beräkna valence delat
  select(valence, danceability, valence_to_danceability)       # välj relevanta kolumn
```

Gör lämpliga ändringar för att multiplicera `tempo` och `time signature`.

3.6.3 Summera kolumner med `group_by` och `summarise`

För att presentera insamlad data på ett tolkningsbart sätt används sammanfattande mått såsom summor, medelvärden, medianer och standardavvikelser. Den typen av beräkningar kan göras som ett nytt steg i en pipe med hjälp av funktionen `summarise`. Om man kombinerar `summarise` med funktionen `group_by` kan man dessutom summera efter en indelning given av en annan variabel. En beräkning av genomsnittligt tempo per år kan till exempel ges av

```
dat %>%                                        # Ta datan, och sen
  group_by(album_release_year) %>%             # gruppera efter år, och sen
  summarise(Medeltempo = mean(tempo))          # beräkna medelvärde av tempo
```

```
## # A tibble: 63 x 2
##    album_release_year Medeltempo
##                 <dbl>      <dbl>
##  1               1960      103.
##  2               1961      104.
##  3               1962      121.
##  4               1963      116.
##  5               1964      122.
##  6               1965      114.
##  7               1966      121.
##  8               1967      118.
##  9               1968       95.0
## 10               1969      116.
## # ... with 53 more rows
```

I det sista steget skapas en variabel *Medeltempo* som ges av medelvärde av den ursprungliga variabeln *tempo*. Här använder vi också funktionen `mean` för att beräkna medelvärdet.

Om man vill summera flera variabler kan man ange flera beräkning inom `summarise`, t.ex.

```
dat %>%                                      # Ta datan, och sen
  group_by(album_release_year) %>%           # gruppera efter år, och sen
  summarise(Medeltempo = mean(tempo),        # beräkna medelvärde av tempo
            Medelvalence = mean(valence), # beräkna medelvärde av valence
```

```
            antal_spår = n()) %>%          # beräkna antalet spår, och sen
  arrange(-Medelvalence)                   # ordna efter medelvalence i sjunkande ordning
```

```
## # A tibble: 63 x 4
##    album_release_year Medeltempo Medelvalence antal_spår
##                 <dbl>      <dbl>        <dbl>      <int>
##  1               1960      103.         0.816         29
##  2               1961      104.         0.752         12
##  3               1968       95.0        0.738         21
##  4               1981      123.         0.671         17
##  5               1965      114.         0.642        147
##  6               1972      118.         0.641         64
##  7               1976      123.         0.626         65
##  8               1982      114.         0.624         50
##  9               1975      123.         0.622         84
## 10               1962      121.         0.613         13
## # ... with 53 more rows
```

Uppgift 3.15 (Gladast artist)**.** Vad ska ändras i stycket ovan för att beräkna medelvalence per artist istället för per år?

Vad måste ändras i stycket nedan för att se vilka artister som påverkar värdet för 1960?

```
dat %>%
  filter(album_release_year == 1976) %>%
  count(artist_name, album_name, album_release_year)
```

3.7 Grafer med ggplot2

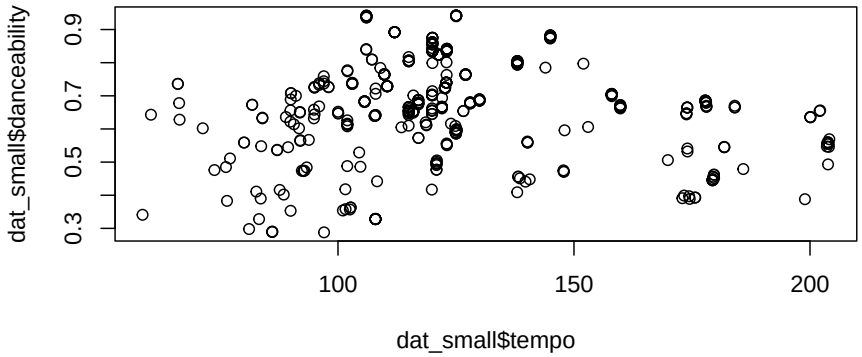
Vi kan nu börja titta på grafer. Eftersom datan är ganska stor och grafer lätt blir oöversiktliga, börjar vi med att skapa en lite mindre datamängd.

```
dat_small <- dat %>%
  filter(artist_name == "Robyn", album_type == "album")
```

Uppgift 3.16 (Valfritt artistbyte)**.** Om du vill titta data på något annan artist, gör lämplig ändring i stycket ovan. Kom ihåg att man skriva ut artister i datan med `unique(dat$artist_name)`.

R har en mängd grundläggande funktioner för grafer. Ett enkelt spridningsdiagram kan till exempel skapas med

```
plot(x = dat_small$tempo, y = dat_small$danceability)
```



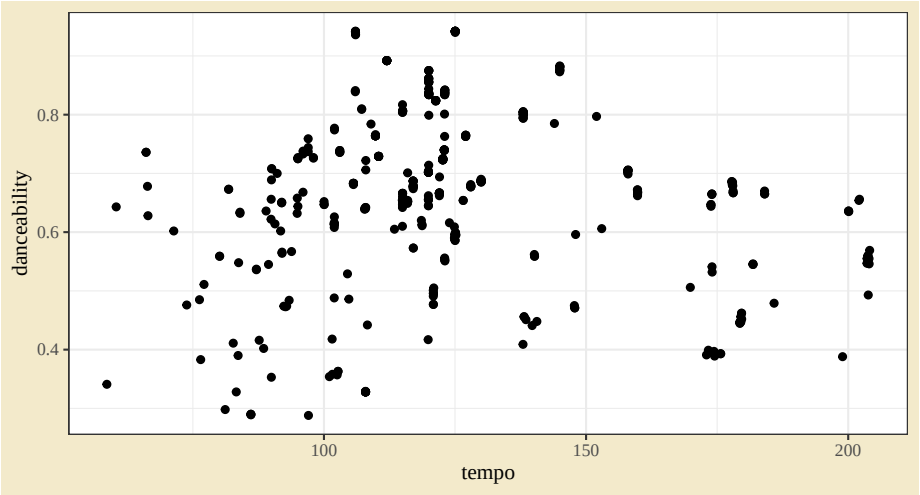
Tecknet `$` används här för att välja en kolumn i en tabell.

För mer avancerade grafer används dock ofta funktioner ur Rs paketbibliotek. Här illustreras det mest populära - `ggplot2`. I `ggplot2` byggs grafer upp med tre grundläggande byggstenar:

- *data*, informationen man vill visualisera,
- *aestethics*, en koppling mellan data och visuella element såsom grafens axlar, objekts storlek och färg,
- *geometries*, de geometriska former som visas i grafen.

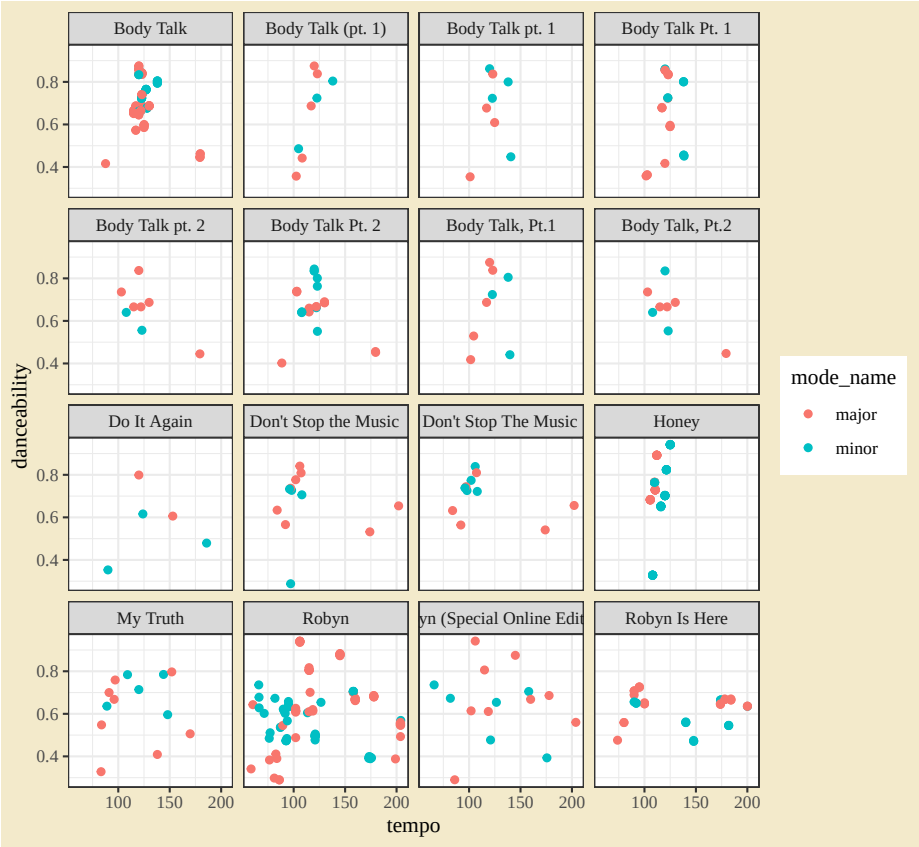
En graf skrivs med en startfunktion `ggplot` som anger namnet på datan och grafens *aestethics*, och därefter sätts geometriska element genom funktioner som börjar med `geom_`. Ett spridningsdiagram kan t.ex. skapas med `geom_point`.

```
ggplot(dat_small, aes(x = tempo, y = danceability)) +
  geom_point()
```



Grafen kan byggas ut genom att sätta *aestethics* för färg och storlek. Man kan också dela en graf i småfönster med `facet_wrap` och styra grafens utseende genom att sätta ett tema såsom `theme_bw`.

```
ggplot(dat_small, aes(x = tempo, y = danceability, color = mode_name)) +
  geom_point() +
  facet_wrap(~ album_name)
```



Uppgift 3.17 (Dur och moll)**.** Vad ska ändras i stycket nedan för att skapa en graf med dur/moll (`mode_name`) på x-axeln, valens (`valence`) på y-axeln och skilda småfönster för olika år (`album_release_year`)?

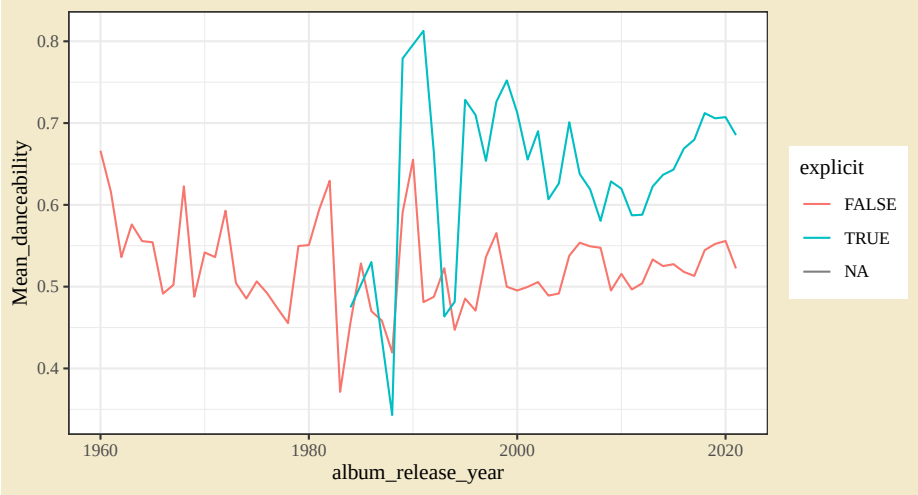
```
ggplot(dat_small, aes(x = mode_name, y = valence, color = album_name)) +
  geom_point() +
  facet_wrap(~ album_release_year)
```

Har spår i dur (*major*) högre valens?

Andra graftyper kan skapas med andra `geom_`-funktioner. För ett linjediagram används `geom_line`. De observationer som ska ge en specifik linje anges med `group` i `aes`-funktionen. Låt oss beräkna medeldansbarhet över tid, uppdelat efter markeringen för *explicit* (alltså om spåret är barnvänligt eller inte).

```
dat_mean_over_time <- dat %>%
  group_by(album_release_year, explicit) %>%
  summarise(Mean_danceability = mean(danceability))

ggplot(dat_mean_over_time, aes(x = album_release_year, y = Mean_danceability, color = e
```

```
  geom_line()
```

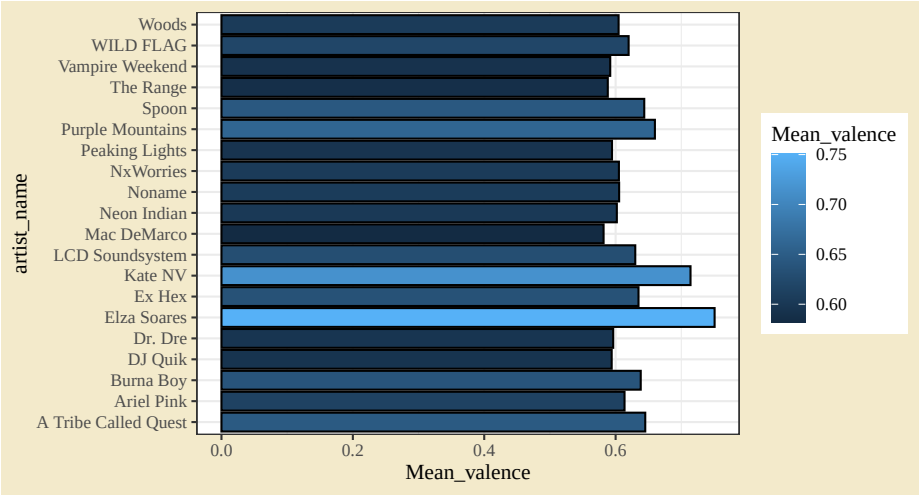


Här ger färgen uppdelningen i explicit och icke-explicit (`TRUE` för explicit). Det finns också spår som saknat värde för explicit och därför blir `NA` här. Kan vi utifrån grafen säga att barnvänlig musik är mer eller mindre dansbar än icke-barnvänlig?

Uppgift 3.18 (Explicit glädje)**.** Vad ska ändras i stycket ovan för att för en graf över medelvalens (`valence`) istället för dansbarhet? Är barnvänlig musik gladare eller ledsnare är icke-barnvänlig?

Stapeldiagram ges av `geom_col` (`col` för *column*). Man kan också använda `geom_bar` om man bara vill räkna antal rader per någon kategori. Följande beräknar valens per artist, ordnar efter valens, väljer ut de tjugo högsta, och plottar i ett (liggande) stapeldiagram.

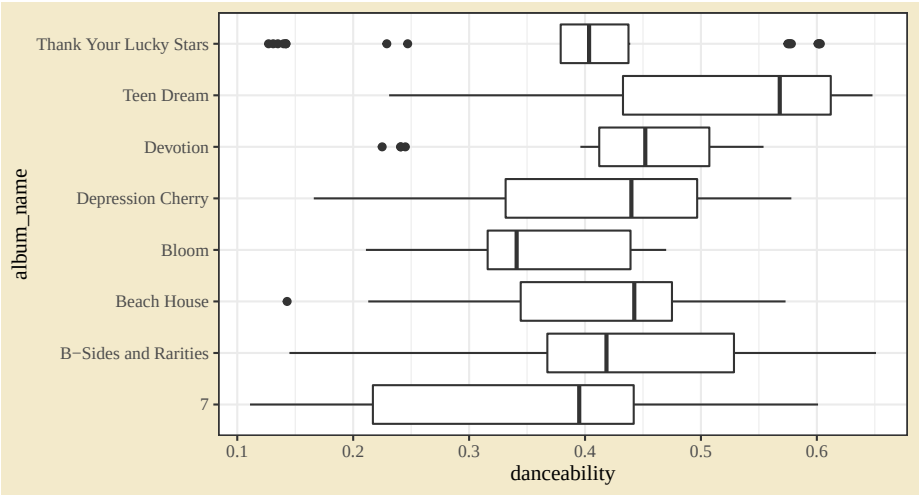
```
dat %>%                                                          # Ta datan, och sen
  group_by(artist_name) %>%                                      # gruppera efter artist,
  summarise(Mean_valence = mean(valence)) %>%                    # ta medelvärdet av vale
  arrange(-Mean_valence) %>%                                     # ordna efter medelvaler
  slice(1:20) %>%                                                # ta ut de tjugo första
  ggplot(aes(x = Mean_valence, y = artist_name, fill = Mean_valence)) +  # starta en ggplot där
  geom_col(color = "black")                                      # skapa en geometri av
```



Argumentet `fill` styr färgen för ytor (här staplarnas ytor) medan `color` i `geom_col()` styr kanten runt varje stapel.

Man kan styra grafiken i en `ggplot` genom funktionen `theme()`. Det är ett ganska komplicerat ämne, men låt oss titta på några grunder. Vi börjar med att skapa en enkel graf: en boxplot över dansbarhet per album för bandet *Beach House* (som spelade på Malmöfestivalen 2009).

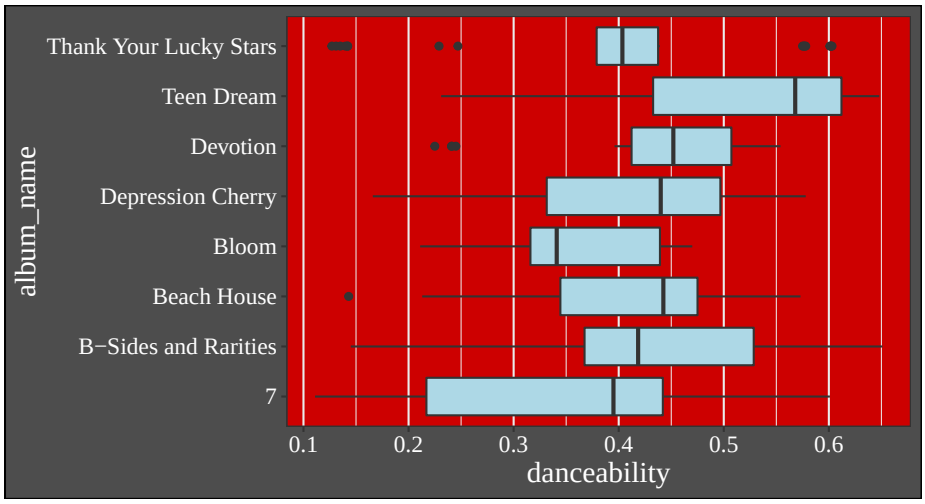
```
dat_small <- dat %>% filter(artist_name == "Beach House", album_type == "album")

ggplot(dat_small, aes(danceability, album_name)) +
  geom_boxplot()
```



Vi kan ändra utseendet på grafen genom argument inom geometrier och med funktionen `theme()`. I `theme()` sätter man de specifika egenskaper man

vill ändra genom att tillskriva dem ett *element*. Valet av element beror på typen av grafiskt objekt - text sätts t.ex. med `element_text()` och ytor med `element_rect()` (för *rectangle*). Vi ger ett exempel med ändrad bakgrund, rutmönster, och teckenstorlek.

```
ggplot(dat_small, aes(danceability, album_name)) +
  geom_boxplot(fill = "lightblue") +
  theme(panel.background = element_rect(fill = "red3"),
        text = element_text(size = 15, color = "white", family = "serif"),
        axis.text = element_text(color = "white"),
        plot.background = element_rect(fill = "grey30", color = "black"),
        panel.grid.major.y = element_blank())
```



Uppgift 3.19 (Temaval 1)**.** Ändra färgvalen i grafen ovan för att skapa snyggast möjliga graf. Funktionen `colors()` ger de färger som finns tillängliga i R. Man kan också använda hex-koden för färger, t.ex. `fill = "#ffdd00"`.

Uppgift 3.20 (Temaval 2)**.** Ändra färgvalen i grafen ovan för att skapa fulast möjliga graf. Visa de två graferna för någon annan och se om de kan säga vilken som är vilken.

3.8 Bonus: interaktiva grafer med `plotly`

Låt oss ta en titt på `plotly`, ett av flera R-paket som gör det möjligt att skapa interaktiva grafer. Vi börjar med att installera och ladda paketet.

```
# install.packages("plotly")
library(plotly)
```

Paketet innehåller en smidig funktion `ggplotly()` för att göra en interaktiv graf från en `ggplot`. Vi börjar med att filtrera datan för en specifik artist och

albumtyp. I samma pipe skapar vi en ny kolumn decade, som beräknar årtiondet utifrån året. Den exakta beräkning är inte så viktig, men ta gärna en titt och se om du förstår vad som delarna gör.

```
dat_small <- dat %>%
  filter(artist_name == "David Bowie", album_type == "album") %>%
  mutate(Decade = floor(album_release_year / 10) * 10)
```

(Säg t.ex. att vi har året 1979. Att dela med 10 ger 197.9. Funktionen `floor` avrundar nedåt till 197. Multiplikationen med 10 ger 1970.)

Vi kan nu konstruera en graf med `ggplot()`. Låt oss ha dansbarhet på x-axeln och valens på y-axeln. Geomet `geom_point()` ger ett spridningdiagram och `facet_wrap(~ Decade)` delar i småfönster efter årtionde. Slutligen tar `theme(legend.position = "none")` bort *legenden* - guiden som anger vilken färg som är vilket album.

Notera att vi sparar grafen som ett objekt `g`. För att se grafen kör vi objektnamnet.

```
g <- ggplot(dat_small, aes(danceability, valence, color = album_name, text = track_name
  geom_point() +
  facet_wrap(~ Decade) +            # Skapar småfönster per årtionde
  theme(legend.position = "none")   # Tar bort legenden (kopplingen mellan färg och al
g
```

När vi har en färdig ggplot kan `ggplotly()` ge en interaktiv version av samma graf.

```
ggplotly(g)
```

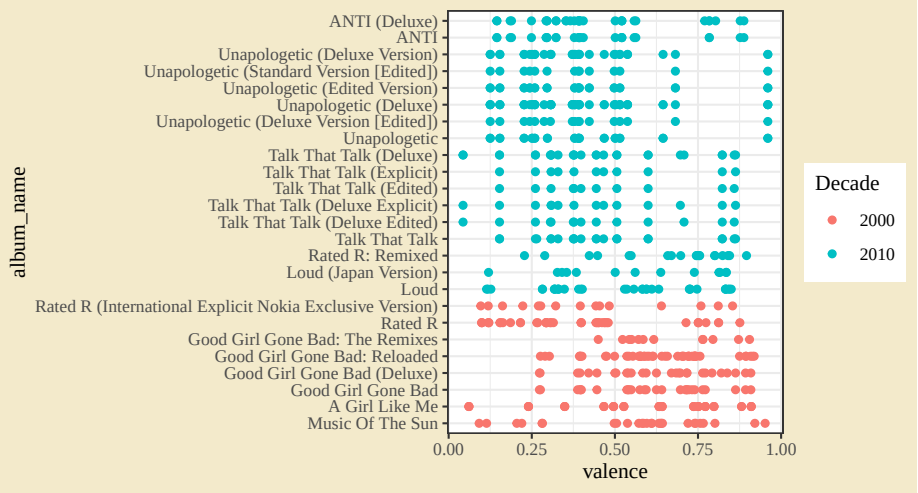
Uppgift 3.21 (Interaktiv graf med annan artist)**.** Gör lämpliga ändringar i stycket nedan för att skapa en interaktiv graf med en annan artist och med tempo på x-axeln och dansbarhet på y-axeln. Kom ihåg att du kan se tillgängliga artister med raden `unique(dat$artist_name)`.

```
dat_small <- dat %>%
  filter(artist_name == "David Bowie", album_type == "album") %>%
  mutate(Decade = floor(album_release_year / 10) * 10)

g <- ggplot(dat_small, aes(danceability, valence, color = album_name, text = track_name
  geom_point() +
  facet_wrap(~ Decade) +            # Skapar småfönster per årtionde
  theme(legend.position = "none")   # Tar bort legenden (kopplingen mellan färg och al
g

ggplotly(g)
```

Uppgift 3.22 (Interaktiv graf med andra axlar)**.** Vi fortsätter med ett nytt

exempel, nu med ett spridningsdiagram med album på y-axeln och valens på x-axeln.

```
dat_small <- dat %>%
  filter(artist_name == "Rihanna", album_type == "album") %>%
  mutate(Decade = floor(album_release_year / 10) * 10,
         Decade = as.character(Decade),
         album_name = reorder(album_name, album_release_year)) # Funktionen reorder() ordnar en v

g <- ggplot(dat_small, aes(valence, album_name, color = Decade, text = track_name)) +
  geom_point()
g
```



Ändra gärna artist på lämpligt ställe. Vad måste läggas till för en interaktiv version av samma graf?

Hemsidan https://plotly.com/r/ innehåller fler exempel för den som är intresserad.

3.9 Bonus: Warming stripes

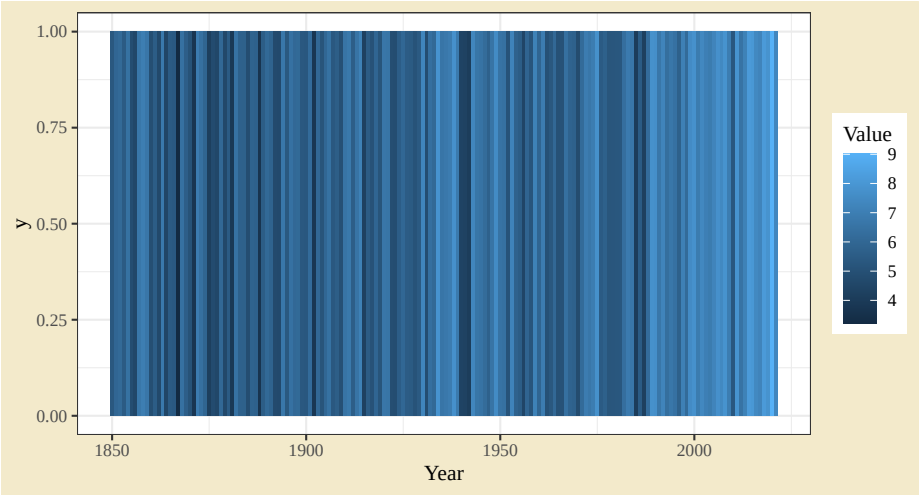
Warming stripes har sedan de först introducerades av Ed Hawking 2018 blivit en vanlig illustration av temperaturökning. I en warming stripe-graf anges varje år av en stapel och stapels färg ges av ett temperaturmått, vanligen årets medeltemperatur. I ggplot-terminologi har vi geometrier (staplar eller kolumner) med en x-position som ges av år och en ifylld färg som ges av temperatur.

För att göra en graf behöver vi data över temperaturer. Följande rad hämtar temperaturdata från Stockholm. Källa: https://miljobarometern.stockholm.se/klimat/klimat-och-vaderstatistik/medeltemperatur/

```
dat_temp <- read_csv("Data/Temperatur, Stockholm.csv")
```

Vi skapar nu en graf, som alltså ska ha en x-axeln given av år (`Year`) och ifylld färg som ges av temperatur (`Value`). Vi sätter höjden till 1.

```
ggplot(dat_temp, aes(x = Year, y = 1, fill = Value)) +
  geom_col()
```



Uppgift 3.23 (Staplar utan mellanrum)**.** Ett första problem är att staplarna inte fyller ytan. Man kan styra staplars bredd med argumentet `width`, t.ex.

```
ggplot(dat_temp, aes(x = Year, y = 1, fill = Value)) +
  geom_col(width = 0.1)
```

Hitta ett värde för `width` som ger staplar utan mellanrum.

Uppgift 3.24 (Färgval)**.** Ett andra problem är att ggplots grundval för färger är från svart till blått. För klassiska warming stripes vill vi ha en skala från blått till rött. Färgerna i en skala ändras med särskilda `scale_()`-funktioner. En färgskala för ifylld färg kan sättas med `scale_fill_gradientn()`, till exempel

```
ggplot(dat_temp, aes(x = Year, y = 1, fill = Value)) +
  geom_col(width = 0.1) +
  scale_fill_gradientn(colours = c("darkgreen", "blue", "white", "yellow", "purple"))
```

Välj färger som ger en naturlig skala från blått till rött. Funktionen `colors()` ger valbara färger i R. Några möjliga val kan vara `darkblue`, `blue`, `white`, `red`, `salmon`, `darkred`, `steelblue` och `skyblue`.

Uppgift 3.25 (Enkel graf)**.** Slutligen brukar warming stripes presenteras med så lite kringinformation som möjligt. I ggplot kan grafelement tas bort med `theme()`. Här är som exempel en graf utan y-axel, tickmärken och legend.

```
ggplot(dat_temp, aes(x = Year, y = 1, fill = Value)) +
  theme(axis.title = element_blank(),
        legend.position = "none",
        plot.background = element_blank(),
        panel.background = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks = element_blank())
```

Använd temat från exemplet för att skapa en enklare version av de warming stripes från föregående uppgift.

3.10 Valfria hemuppgifter

Uppgift 3.26 (Installera R)**.** Gå till https://www.r-project.org/ och installera R på ditt hemma-system. Instruktioner finns i R-anvisningarnas kapitel *Installation*.

Uppgift 3.27 (Installera RStudio)**.** Gå till https://www.rstudio.com/ och installera RStudio på ditt hemma-system. Instruktioner finns i R-anvisningarnas kapitel *Installation*.

Uppgift 3.28 (Öppna RStudio)**.** Öppna skriptet till datorövning 1 i RStudio. Notera om något ser annorlunda ut mot hur det såg ut i datorsal.

Uppgift 3.29 (Cheat sheets)**.** I början av datorövning 1 plockade vi upp en hemsida med *cheat sheets* - korta instruktioner och guider till specifika R-paket. Vi kan hitta några av dem på https://www.rstudio.com/resources/cheatsheets/. Ladda ner guiden till `ggplot2`. Hur många olika `theme_()`-funktioner (som t.ex. `theme_bw()`) beskrivs i guiden? Vad gör en `theme_`-funktion?

Uppgift 3.30 (Timothée Chalamet)**.** Hitta klippet där skådespelaren Timothée Chalamet sjunger en sång om statistik. Uppskatta spårets dansbarhet och valens på en skala från noll till ett. Fundera på om det är rimligt att sätta en siffra på dansbarhet och valens.