Datorlaboration 1

Statistik och dataanalys 2 (ST1201), HT2023

Introduktion

Laborationen ska genomföras som en *Quarto-notebook*. En stor fördel med notebook-formatet är att det låter er skapa eget kursmaterial genom att kombinera kod med text som beskriver vad koden gör. Att skriva sitt egna kursmaterial och att med egna ord förklara hur saker fungerar är ett av dom bästa sätten att lära sig.

Dom första tre laborationerna matchar mot dom tre delarna i inlämningsuppgiften. När du är klar med dagens laboration är du alltså redo att börja med del 1 på inlämningsuppgiften.

- Det är helt OK att ni samarbetar under labben, men skriv din egen labbrapport! Det är viktigt att faktiskt skriva koden själv.
- Om du fastnar, testa att se om du kan hitta en lösning i dokumentationen. För att se dokumentationen för en viss funktion skriver du ett frågetecken följt av funktionens namn i konsolen, exempelvis `?lm`. Funkar inte det, testa google eller chatGPT, och funkar inte det, fråga labbansvarig. Vi finns där för att svar på dina frågor, med det är viktigt att du tränar på att lösa problem själv.

Innehåll

I den här laboration kommer du att

- lära dig grundläggande data wrangling med `dplyr`,
- komma igång med datavisualisering med `ggplot2`,
- lära dig hur man estimerar och visualiserar en linjär regressionsmodell med interaktionseffekter.

Del 1 - Data wrangling med `dplyr` och datavisualisering med `ggplot2`

I den här delen kommer du lära dig följande.

- Vad en `tibble` är och hur den skiljer sig från en `data.frame`.
- Hur du kan välja ut specifika observationer (rader) från ett dataset med `filter()`.
- Hur du kan välja ut specifika variabler (kolumner) från ett dataset med `select()`.
- Hur du kan skapa nya kolumner baserat på värden i gamla kolumner med `mutate()`.
- Hur du använder `%>%`-operatorn i `dplyr` till att kombinera olika funktioner.

1a - att komma igång med själva rapportdokumentet

Ditt första uppdrag är att skapa en quarto notebook att dokumentera labben i.

- ☐ Börja med att skapa ett tomt quarto-dokument. Du gör detta enklast genom att först skapa ett nytt *Quarto document* i R-studio, klicka över till `Source` och sen radera allt utom preamble (det som står mellan dom streckade linjerna).
- ☐ Ändra `title` från *Untitled* till *Laboration 1.* Klicka på *Render* så ska ett htlm-dokument skapas som endast innehåller texten *Laboration 1.*

Det är en bra vana att lägga alla sina `library()`-anrop i en chunk i början av sitt skript. Detta gör det lätt att direkt se vilka paket som används, samt gör det lättare att gömma alla `library()`-anrop i själva rapportdokumentet (exempelvis htlm-dokumentet).

- ☐ Skapa en kodchunk som innehåller koden `library(tidyverse)`. Klicka på *Render* för att uppdatera ditt html-dokument. Om du har gjort rätt kommer ditt html-dokument nu innehålla `library(tidyverse)` samt en utskrift som innehåller en lista på paket som laddats in, samt ett antal konflikter.

Information att ett paket laddats in är sällan något vi vill ha med i vår slutrapport. Exempelvis ska det inte se ut så här i inlämningsuppgiften. För att kontrollera vilken information som visas i html-dokumentet använder vi oss av *chunk options* (chunkinställning). En chunkinställning är en rad inne i en kodchunk som börjar med `#|`. Chunkinställningen `#| echo: false` säger till R att vi endast vill se *resultatet* från chunken och inte själva koden.

- ☐ Lägg till `#| echo: false` i kodchunken du använder för `library(tidyverse)`. Rendera och verifiera att `library(tidyverse)` inte längre syns i html-dokumentet.

Vissa paket skriver ut meddelanden och varningar när dom laddas in, vilket inte ser så snyggt ut i en rapport. För att förhindra att denna information skrivs ut behöver vi chunkinställningen `#| output: false`.

- ☐ Lägg till `#| output: false` i din kodchunk. Rendera och verifiera att ditt html-dokument åter bara innehåller texten *Laboration 1.*

Du kommer klara dig långt på `#| echo: false` och `#| output: false`. En annan chunkinställning som kan vara bra att känna till är `#| eval: false` som hindrar att koden i en chunk körs över huvud taget. Detta är bra om du har inkomplett kod i en chunk, men vill kunna rendera din notebook utan att få en massa felmeddelanden.

När du gör inlämningen förväntas du följa instruktionerna ovan och ha alla `library()`-anrop i en kodchunk i början av er noteboook (efter preamble) med `#| echo: false` och `#| output: false`.

1b - läs in data

Du kommer under labben använda två paket som inte finns på CRAN: `SUdatasets` och `sda123`. Dessa paket går inte att installera med `install.packages()`, utan du behöver använda koden nedan.

```
library(remotes) # Paketet `remotes` behöver vara installerat, finns på CRAN
install_github("StatisticsSU/sda123") # Det här behöver du bara göra en gång
install_github("StatisticsSU/SUdatasets") # Det här behöver du bara göra en gång
```

- ☐ Kontrollera om `sda123` och `SUdatasets` är installerade. Om dom *inte* är det ska du installera dom. Lägg sedan till `library(sda123)` och `library(SUdatasets)` i paketchunken du skapade i 1a.

`library(SUdatasets)` ger dig tillgång till ett antal olika dataset, men du kommer inte att se något av dom under `Environment` innan du interagerat med dom. Börja med att ladda in datasetet `galton` som ett `tibble` i ett objekt med namnet `df`. Undersök sedan datasetet genom att skriva dess namn, som nedan.

```
df <- tibble(galton)
df
```

Du kommer att se att utrskriften ser lite annorlunda ut än du är van vid när du undersöker en `data.frame`. En `tibble` är tidyverse motsvarighet till en `data.frame`, och visar alltid bara ett rimligt antal rader. Vi kan också direkt se vilka typer variablerna har!

- ☐ Skriv `?galton` i konsolen för att läsa om datasetet.

1c - deskriptiv analys

Eftersom att datasetet samlades in i England så är alla längder angivna i tum (1 tum = 2.54 cm). Detta är lite förvirrande för en svensk publik, och du behöver därför först skapa nya längdvariabler med längder angivna i centimeter. För att göra detta ska du använda `mutate()`. För att skapa en version av `Height` i cm kan du använda

```
df %>% mutate(height_cm = Height * 2.54)
```

För att använda `mutate()` skriver du först namnet på det dataset du vill utgå ifrån. Du skriver sen *pipe*-operatorn, `%>%`, följt av `mutate(height_cm = Height * 2.54)` för att säga till R att skapa en ny variabel som antar samma värde som `Height`, fast multiplicerat med 2.54. Koden ovan har ett problem. Den sparar inte vårat nya dataset (det som skrivits ut) någonstans! För att spara datasetet måste vi inleda med namnet på objektet vi vill spara i följt av `<-`, som vanligt.

Om vi vill lägga till en variabel vill vi ofta skriva över det gamla datasetet, vilket vi gör genom att skriva

```
df <- df %>% mutate(height_cm = Height * 2.54)
```

- ☐ Lägg till en tre nya kolumner till `df`: `mom_cm` och `dad_cm` som anger föräldrarnas längd i cm, samt `parents_avg` som visar medelvärdet av mamman och pappans längd (angivet i centimeter).

Du ska nu beskriva fördelningen för variabeln `height_cm` med ett histogram. För att göra detta ska du använda paketet `ggplot2`. `ggplot2` fungerar på ett lite annorlunda sätt än datavisualiseringen i base R och kan kännas förvirrande i början. `ggplot2` är döpt efter boken *The Grammar of Graphics*, och är ett mycket kraftfullt verktyg som låter dig skapa fantastiska datavisualiseringar. `ggplot2` är ett eget paket, men det läses in automatiskt när du använder `library(tidyverse)`, så du behöver in lägga till ett separat anrop.

Vi kan skapa ett histogram med `ggplot2` för en variabel med namnet `variabel_1` från datasetet `min_df` genom att skriva

```
ggplot(data = min_df, aes(x = variabel_1)) +
  geom_histogram()
```

Kodbiten på den första raden finns alltid med när man gör en figur med `ggplot2`. Det första argumentet, `data`, anger vilket dataset det är vi ska använda när vi plottar. Den andra delen, `aes()`, berättar för `ggplot2` vilket variabler i datasetet som ska användas till vad. I det här exemplet säger jag att variabeln `variabel_1` ska kopplas till x-axeln.

- ☐ Gör ett histogram för variabeln `height_cm` i ditt dataset genom att modifiera koden ovan.

Som du ser gnäller R lite på dig, och varnar att du inte har valt antal bins. Ett histogram står och faller på att ett rimligt antal bins används, och det är alltid en bra idé att experimentera lite och se vad som passar bäst.

- ☐ Ändra antalet bins genom att ange argumentet `binwidth` i `geom_histogram()`, alltså `geom_histogram(binwidth = )`. `binwidth` anger hur breda histogrammets staplar ska vara. Testa med värdena 0.5, 2, och 15. Vilket värde hade R valt ut i första försöket?

Du ska nu skapa ett histogram för `height_cm` uppdelat på `gender`. Detta är mycket lättare att göra med `ggplot` än med base R! Vi lägger helt enkelt till `fill = Gender` inne i `aes()` efter `x = height_cm` (`fill` är färgen som staplarna är fylld med, därav namnet). Tänk på att dom två argumenten måste ha ett komma mellan sig!

På samma sätt som vi "kopplade ihop" variabeln `Height` med x-axeln, har vi nu "kopplat ihop" variabeln `Gender` med färg, och R kommer att rita ett separat histogram, med olika färg, för män och kvinnor! Vi får till och med en legend utskriven!

- ☐ Skapa ett histogram med olika färg för män och kvinnor genom att ersätta `aes(x = height)` med `aes(x = height, fill = Gender)`.

Om du vill ändra vilken färg som kopplas till vilken stapel kan du lägga till `+ scale_fill_manual(values = c("green", "purple"))`, efter `geom_histogram()`. Det är bara att experimentera med färgnamn!

```
ggplot(data = mitt_data, aes(x = variabel_1, fill = variabel_2)) +
  geom_histogram() +
  scale_fill_manual(values = c("green", "purple"))
```

När vi tittar på histogrammet ser ser vi att staplarna står *på* varandra, vilket gör det svårt att se vad som pågår. Vi kan använda argumentet `alpha` tillsammans med `postition` för att fixa till detta. `alpha` bestämmer hur "genomskinlig" en färg ska vara, och anges som ett argument till `geom_histogram()`. `position = "identity"` säger att varje stapel ska börja från botten, alltså att dom inte ska staplas på varandra.

```
ggplot(data = mitt_data, aes(x = variabel_1, fill = variabel_2)) +
  geom_histogram(alpha = 0.5, position = "identity")
```

- ☐ Lägg till `alpha = 0.5, position = "identity"` innuti `geom_histogram()`. Testa några olika värden på `alpha` (mellan 0 och 1) och se vad som händer!
- ☐ Skapa, fortfarande med `ggplot2`, en boxplot som visar variabeln `height_cm` uppdelat på `Gender`. Det här har vi inte gått igenom på föreläsningen så du kommer behöva googla!

1d - subsetta data

Du ska nu genomföra några varianter på dom analyser som vi gick igenom på föreläsningarna. Som ett första steg ska du skapa ett nytt dataset som endast innehåller kvinnorna från `galton`. Till din hjälp har du funktionen `filter()` från `dplyr`. Funktionen `filter()` heter som den gör eftersom att den låter oss *filtrera* data.

Använd följande kod för att skapa ett subset av data som endast innehåller kvinnor.

```
df_f <- df %>% filter(Gender == "F")
```

En fördel med `dplyr` syntax är att vi kan göra flera olika steg "i ett" genom att skapa en kedja med pipe-operatorer. Vi hade alltså kunnat kombinera stegen ovan där vi räknade om alla längder med steget där vi valde ut alla kvinnor genom att kombinera flera pipe-operatorer ungefär så här (koden är lämnad inkomplett, fixa till den om du vill!)

```
df_f <- df %>%
  mutate(
    height_cm = Height * 2.54,
    mom_cm = ,
    dad_cm =,
    parents_avg = ) %>%
  filter(Gender == "F")
```

Denna sekvens kan du läsa som: ta datasetet `df`, lägg till tre kolumner med längder i cm, lägg sedan till en kolumn med medelvärdet av mamman och pappans längd, välj sedan ut alla kvinnor.

Anledningen till att koden ovan byter rad så många gånger är för att den ska vara lättare att läsa. När vi skriver en lång sekvens av pipe-operatorer så är det lätt hänt att vi får väldigt långa rader vilket blir svårt att läsa. Försök undvika kodrader som är längre än 80 tecken.

1f - Plotta sambandet mellan föräldrars och deras döttrars längd med ett spridningsdiagram (scatterplot)

Du kommer snart att genomföra en enkel linjär regression, men först ska du undersöka hur data ser ut med ett spridningsdiagram. När du skapar ett spridningsdiagram med `ggplot2` så kommer koden att se ut så här

```
ggplot(mitt_data, aes(x = variabel_1, y = variabel_2)) +
  geom_point()
```

Precis som med histogrammet så börjar vi med `ggplot()`-delen. Det första argumentet anger namnet på datasetet, och inom `aes()` ska vi ange hur dom olika variablerna i datasetet ska representeras i vår figur. I det här fallet säger vi att `variabel_1` ska mappas till x-axeln och `variabel_2` till y-axeln. Efter att vi bestämt vilken variabel som ska kopplas till vad behöver vi ange *vad* det är vi ska rita. `geom_point()` säger till R att vi vill rita *punkter*.

- ☐ Gör ett spridningsdiagram som visar sambandet mellan `height_cm` (på y-axeln) och `parents_avg` (på x-axeln) genom att anpassa koden ovan.

Del 2 - Linjär regression

2a - Enkel linjär regression

Du ska nu skatta en linjär regressionsmodell som undersöker sambandet mellan kvinnors längd (responsvariabel) och deras föräldrars genomsnittliga längd (prediktor) med `lm()`-funktionen.

```
min_modell <- lm(responsvariabel ~ forklarande_variabel, mitt_data)
```

☐ Anpassa koden ovan till ditt dataset och namnet på dina variabler.

`lm()`-funktionens första argument använder *formelsyntax*. Det som står till vänster om `~` är *responsvariabeln* och det som står till höger är *prediktorn*. Om flera prediktorer ska vara med behöver du separerar dom med `+`. `lm()` inkluderar per automatik ett intercept. För att stoppa R från att inkludera ett intercept behöver vi lägga till `0 +` till höger om `~`, som i exemplet nedan

```
modell_utan_intercept <- lm(responsvariabel ~ 0 + prediktor, mitt_data)
```

☐ Använd funktionen `reg_summary()` för att undersöka om `parents_avg` är signifikant på 1 % signifikansknivå.
☐ Hur stor är förklaringsgraden, `R2`?

En fördel med enkel linjär regression är att vi kan visulisera regressionslinjen på ett enkelt sätt. `ggplot2` gör det extra enkelt för oss. För att lägga till en linjär regressionsmodell till en plot kan vi använda `geom_smooth()`. `geom_smooth()` lägger till en "utjämnad" linje till vår figur, och den kan plotta olika typer av linjer (inte bara linjär regression). För att lägga till en linjär regressionslinje så använder vi `geom_smooth(method = "lm")`, där "lm" står för linear model.

```
ggplot(mitt_data, aes(x = variabel_1, y = variabel_2)) +
  geom_point() +
  geom_smooth(method = "lm")
```

Du kan skriva `?geom_smooth()` i konsolen för att se vilka olika argument `geom_smooth()` tar. Några av dom viktigaste beskrivs nedan.

- `formula =` låter dig välja exakt vilken regressionslinje som ska ritas. Om du inte anger någon formel så kommer en regressionslinje ritas baserat på vad dina x- och y-variabler är.

- `se =` låter dig bestämma om ett *konfidensintervall* ska ritas runt regressionslinjen. Standardvärdet för detta argument är `TRUE`, så om du bara vill visa själva regressionslinjen och inte intervallet behöver du skriva `se = FALSE`.
- `level =` anger vilket konfidensintervall som ska visas. Standardvärdet är 0.95 för ett 95 %-igt konfidensinterval, men du kan använda vilken konfidensgrad du vill.

☐ Skapa ett spridningsdiagram som visar sambandet mellan `Height` och `parents_avg` tillsammans med en regressionslinje.

☐ Ändra så att konfidensintervallet som visas är ett 90 % konfidensintervall.

☐ Täcker konfidensintervallet dom flesta punkterna? Borde det göra det?

2b - Linjär regression med interaktionseffekter

Du ska nu gå tillbaks till orginaldatasetet, dvs det som innehåller både män och kvinnor, och jobba med modellen

$\texttt{height\_cm} = \alpha + \beta_1 \texttt{Gender} + \beta_2 \texttt{parents\_avg} + \beta_3 \texttt{Gender} \cdot \texttt{parents\_avg} + \varepsilon$ (1)

Detta är en multipel linjär regression som innehåller

- En numerisk förklarande variabel (`parents_avg`)
- En förklarande variabel som är en dummy (`Gender`)
- En *interaktionseffekt* mellan `parents_avg` och `Gender`

För att specificera en interaktionseffekt när vi använder `lm()` så skriver vi `:` mellan namnen på dom variabler vi vill interagera. Alltså, om vi har två variabler `var_1` och `var_2` skriver vi

```
lm(utafallsvariabel ~ var_1 + var_2 + var_1:var_2, data = mitt_data)
```

☐ Skatta modellen i (1).
☐ Är β_3 signifikant skild från 0 på 5 % signifikansnivå? Hur tolkar du detta?
☐ Har `R2` ökat? Verkar det rimligt?

En enkel linjär regressionsmodell kan representeras grafiskt som en rak linje. En linjär regression med en numerisk- och en dummyvariabel kan representeras grafiskt som *två* raka linjer. En linjär regression med en numerisk variabel, en dummyvariabel och en interaktionseffekt kan representeras grafiskt som två raka linjer med *olika lutning*. Du ska nu skapa en figur som visualiserar din nya regressionsmodell.

☐ Använd `ggplot2` för att skapa ett spridningsdiagram, precis som du gjorde i 1f. Denna gång ska du dock lägga till `col = Gender` i `aes()`. Om du har gjort rätt så kommer du få en figur där punkterna har olika färger!

☐ Använd `geom_smooth(method = "lm")` för att lägga till regressionslinjer till din figur. Notera hur `ggplot` direkt skapar två separata linjer!

2c - Residualanalys

Innan vi kan lita på våra slutsatser—exempelvis signifikanstestet—behöver vi försäkra oss om att modellen är korrekt specificerad. Vi gör detta med en *residualanalys*. I en komplett residualanalys ska vi undersöka att

- Residualerna är *normalfördelade*
- Residualerna är *oberoende*
- Residualerna är homoskedastiska, dvs har *konstant varians*

☐ Använd funktionen `reg_residuals()` från `sda123` för att undersöka om antagandena i din modell verkar uppfylla.

2d - Prediktion

Regressionsanalys kan användas till två olika saker: att förstå hur olika variabler hänger ihop, och till prediktion. Prediktion gör lättast med `predict()` funktionen.

```
predict(
  object = min_modell,
  newdata = data.frame(var_1 = , var_2 = ),
  interval = c("prediction")
)
```

- Argumentet `object` är modellen du vill använda för att göra din prediktion, i det här fallet namnet på din regressionsmodell.
- Argumentet `newdata` anger värdena på dom förklarande variablerna för observationen du vill predicera och behöver anges som en `data.frame`. För regressionsmodellen med interaktionseffekt är dom två variablerna `Gender` och `parents_avg`.
- `interval` anger vilket typ av intervall som ska beräknas. Det kan anta värdet `"prediction"` eller `"confidence"` (om du inte anger något så kommer du inte få något interall).
- Level anger konfidensgraden för intervallen, och har 0.95 som standardvärde.

☐ Skapa ett prediktionsintervall för en ny observation. Välj själv värden på prediktorerna.