RHUL Psychology Statistical modelling notebook

Matteo Lisi

2022-06-29

Contents

Chapter 1

About

This online book is created and maintained by Matteo Lisi and is meant to be a shared resource for staff and students at the Department of Psychology of Royal Holloway, University of London. It will contain a miscellanous set of tutorial, examples, case studies, workshops materials and any other useful material related to data analysis and modelling. These will be added and revised over time, based on the most common questions and requests that I receive.

This is a work in progress and may contain imprecisions and typos. If you spot any please let me know at matteo.lisi [at] rhul.ac.uk. The materials that will be included builds upon and draw from existing literature on statistics and modelling. I will endeavor to properly cite existing books and papers; but if any author feels that I have not given them fair acknowledgement, please let me know and I will make amend.

Chapter 2

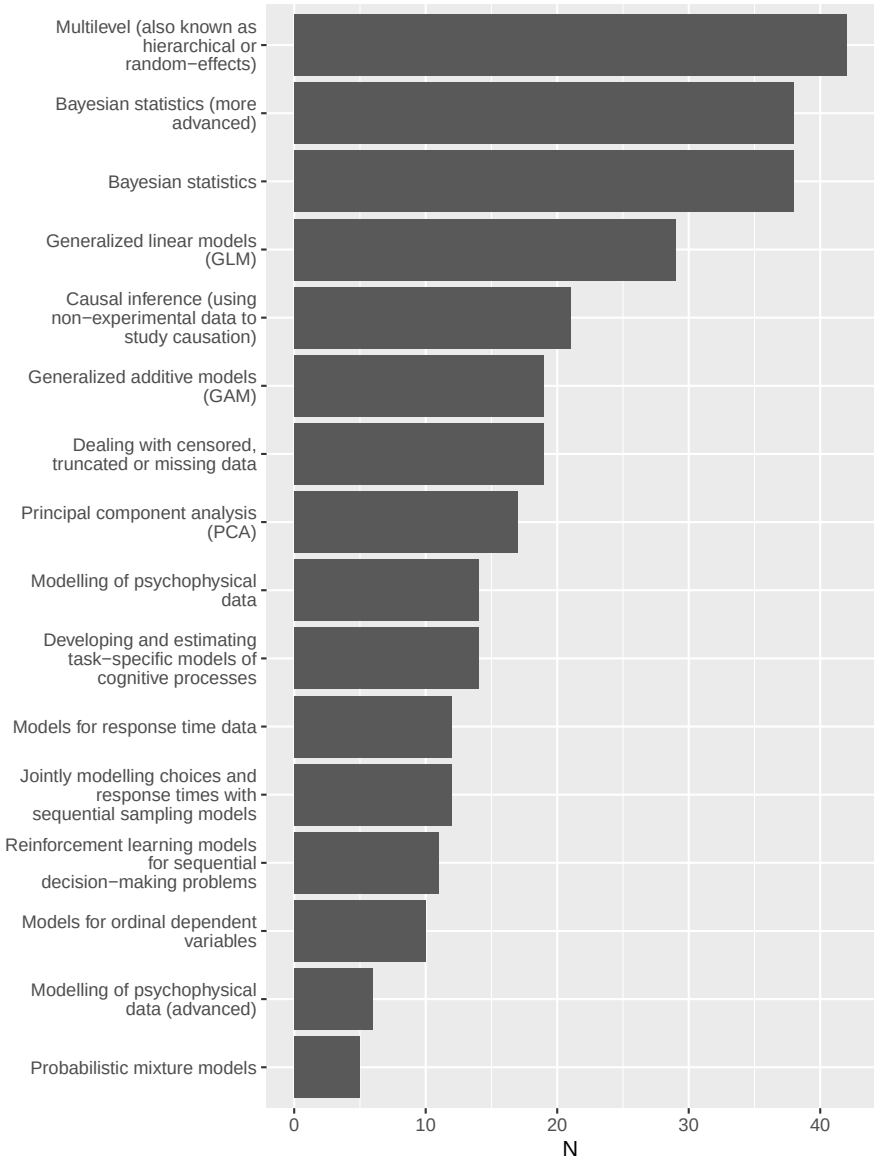
Departmental survey about statistical methods

I used an anonymous survey to ask colleagues some questions about which topics may be more interesting or useful in their research.

2.1 March 2022

2.1.1 Question 1

In the first question people indicated topics of interests. The winner are multi-level models, followed closely by Bayesian statistics.



There were some additional suggestions.

```
#> [1] "power analyses using Shiny apps"
#> [2] "agent-based models"
#> [3] "this may be covered in the above, but approaches to analysing experience sampli
#> [4] "Methods for longitudinal analyses"
#> [5] "Network modelling"
#> [6] "Neural networks, Markov processes"
```

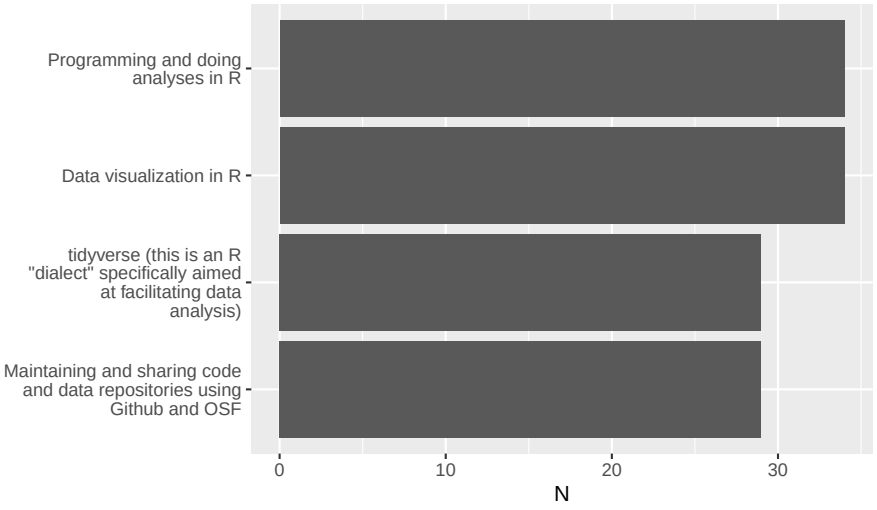
```
#> [7] "Random forests and related"
#> [8] "causal modelling using regression models - path models etc"
#> [9] "prediction modelling"
```

A few other topics were mentioned in the comment section:

- Shiny apps
- Network modelling
- Longitudinal analyses
- Random forests
- Neural network

2.1.2 Question 2

Here people indicated their interest for topics related to data analysis.

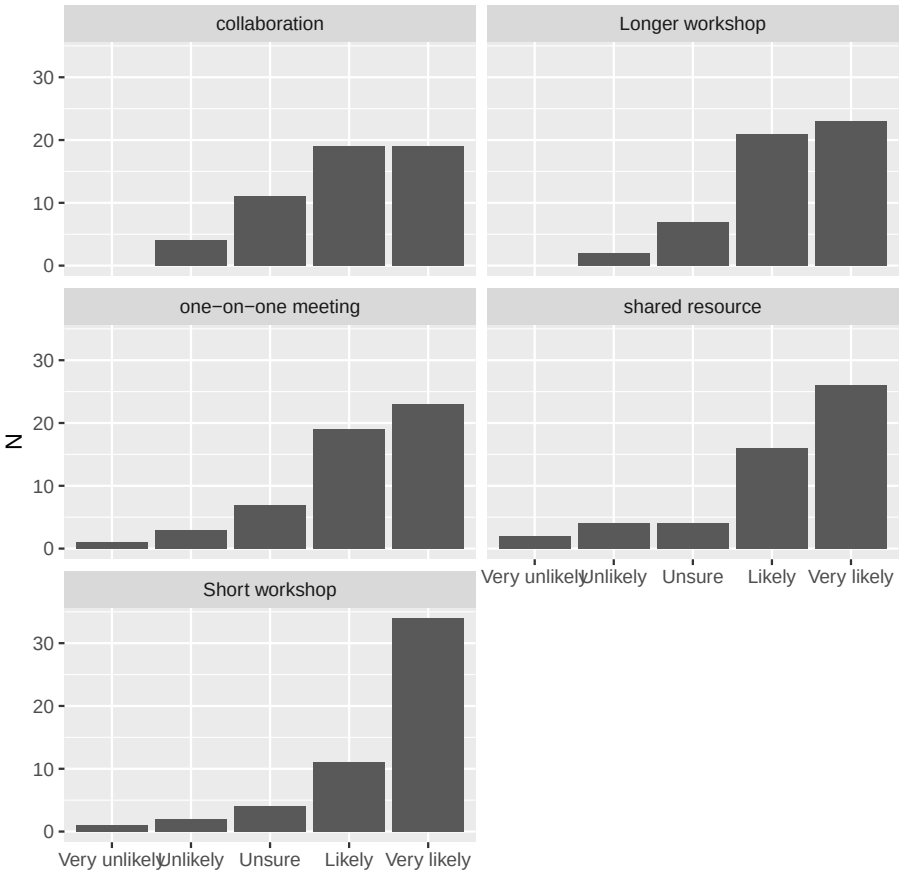


Other things mentioned in the comments were:

- SPM
- Docker
- Python

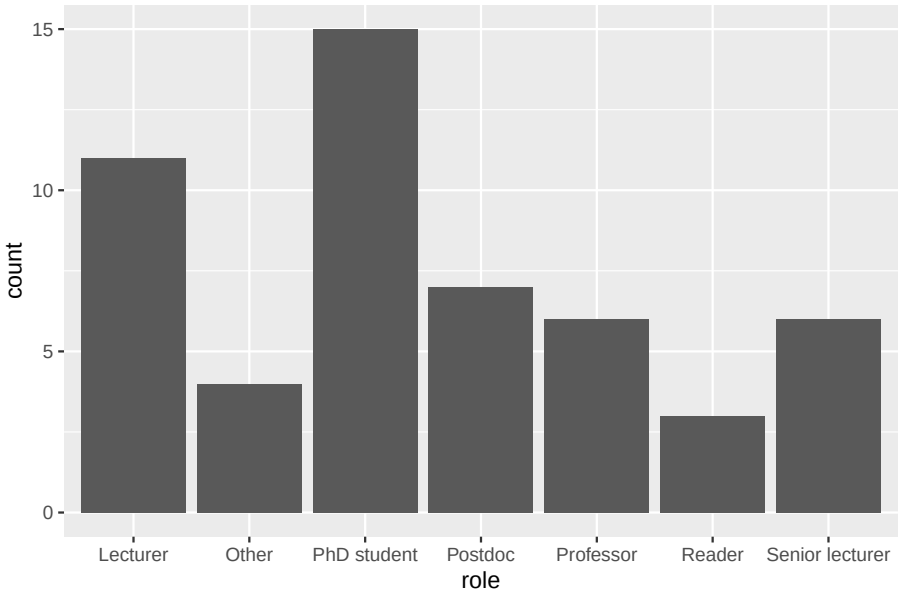
2.1.3 Question 4

This question was about likelihood of using different formats of support



2.1.4 Respondents' status

The final questions asked about the status / career level.


15
10
5
0
count
Lecturer
Other
PhD student
Postdoc
Professor
Reader
Senior lecturer
role

Chapter 3

Introduction to R

Most of the practical statistical tutorials and recipes in this book use the software R, so this section provides some introduction to R for the uninitiated.

3.1 Installing R

The base R system can be downloaded at the following link, which provides installers for both Windows, Mac and Linux:

https://cran.rstudio.com/

In addition to the base R system, it is useful to have also R-studio, which is an IDE (Integrated Development Environment) for R, and provides both an editor, a graphical interface and much more. It can be downloaded from:

https://www.rstudio.com/products/rstudio/download/

3.2 First steps

R is a programming language and free software environment for statistical computing and graphics. It is an *interpreted language*, which means that to give

instructions to the computer you do not have to compile it first in machine language, everything is done 'on the fly' through a command line interpreter, e.g. if you type `2+2` in the command line R, the computer will reply with the answer (try this on your computer):

```
2+2
#> [1] 4
```

Typically the normal workflow involve writing and saving a series of instructions in a *script* file (usually saved with the `.R` extension), which can be executed (either step by step or all at once). Since all steps of the analyes are documented in the script, this makes them transparent and reproducible.

In an R script you can use the `#` sign to add comments, so that you and others can understand what the R code is about. Commented lines are ignored by R, so they will not influence your result. See the next example:

```
# calculate 3 + 4
3 + 4
#> [1] 7
```

3.2.1 Arithmetic with R

In its most basic form, R can be used as a simple calculator. Consider the following arithmetic operators:

- Addition: `+`
- Subtraction: `-`
- Multiplication: `*`
- Division: `/`
- Exponentiation: `^`
- Modulo: `%%`

The last two might need some explaining:

The `^` operator raises the number to its left to the power of the number to its right: for example `3^2` is 9.

The modulo returns the remainder of the division of the number to the left by the number on its right, for example 5 modulo 3 (or `5 %% 3`) is 2.

3.2.2 Variable assignment

A basic concept in programming (statistical or not) is called a *variable*.

A variable allows you to store a value (e.g. 2) or an object (e.g. a function description) in R. You can then later use this variable's name to easily access the value or the object that is stored within this variable.

You can assign a value `2` to a variable `my_var` with the command

```
my_var <- 2
```

Note that you would have obtained the same result using:

```
2 -> my_var
```

that is, the *assignment operator* works in both directions `<-` and `->`.

The variable can then be used in any computation, for example:

```
my_var + 2
#> [1] 4
```

3.2.3 Basic data types in R

Variables can be of many types, not just numerical values. For example, they can contain *text* values (e.g. a string of characters). Arithmetic operators such as `+` do no work with these. If you tried to apply them characters R will give you an error message.

```
# Assign a value to the variable apples
apples <- 5

# Assign a text value
oranges <- "six"

#
apples + oranges
#> Error in apples + oranges: non-numeric argument to binary operator
```

In fact R works with numerous data types, and some of these are not numerical (so they can't be added, subtracted, etc.). Some of the most basic types to get started are:

- Decimal values like 4.5 are called numerics.
- Natural numbers like 4 are called integers. Integers are also numerics.
- Boolean values (`TRUE` or `FALSE`, abbreviated `T` and `F`) are called logical[1].
- Text (or string) values are called characters.

3.2.4 Vectors and other data types

Additionally, the simple data types listed above can be combined in more complex 'objects' that can comprise several values. For example, we can obtain a *vector* by concatenating values using the function `c()`. This can be applied both on numerical or character data types, e.g.

[1] Note that you can add or multiply logical Boolean values: internally `FALSE` are treated as zeroes, and `TRUE` as ones.

```
some_numbers <- c(4,87,10, 0.5, -6)
some_numbers
#> [1]  4.0 87.0 10.0  0.5 -6.0

my_modules <- c("PS115", "PS509", "PS300", "PS938", "PS9457")
my_modules
#> [1] "PS115"  "PS509"  "PS300"  "PS938"  "PS9457"
```

There are some special handy functions to create specific types of vectors, such as *sequences* (using the function `seq()` or the operator `:`)

```
x <- seq(from = -10, to = 10, by = 2)
x
#>  [1] -10  -8  -6  -4  -2   0   2   4   6   8  10

y <- seq(-0, 1, 0.1)
y
#>  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

z <- 1:5
z
#> [1] 1 2 3 4 5
```

Another useful type of vector can be obtained by *repetition* of elements, and this can be numerical, character, or even applied to other vectors

```
rep(3, 5)
#> [1] 3 3 3 3 3

x <- 1:3
rep(x, 4)
#>  [1] 1 2 3 1 2 3 1 2 3 1 2 3

rep(c("leo the cat", "daisy the dog"), 2)
#> [1] "leo the cat"   "daisy the dog" "leo the cat"
#> [4] "daisy the dog"
```

We can combine vectors of different types into a *data frame*, one of the most useful ways of storing data in R. Let's say we have 3 vectors:

```
# create a numeric vector
a <- c(0, NA, 2:4)  # NA means not available

# create a character vector
b <-  c("PS115", "PS509", "PS300", "PS938", "PS9457")

# create a logical vector
```

```
c <- c(TRUE, FALSE, TRUE, FALSE, FALSE)  # must all be caps!
```

we can combine them into a data.frame using:

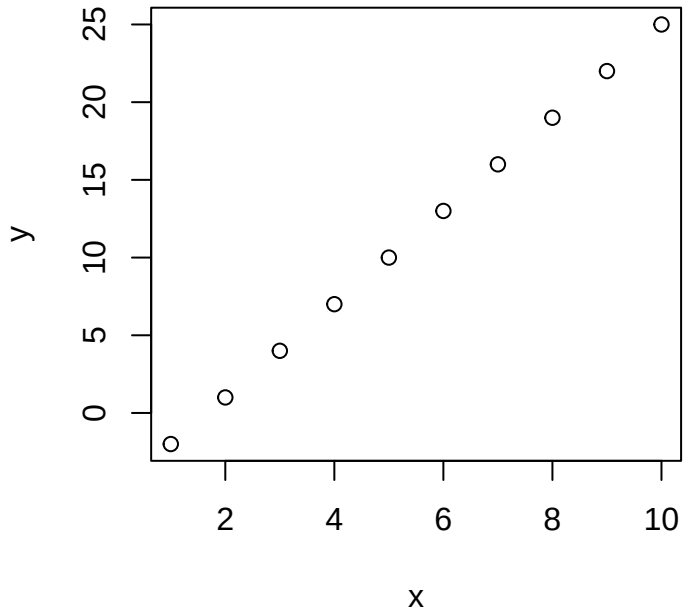
```
# create a data frame with the vectors a, b,and c that we just created
my_dataframe <- data.frame(a,b,c)

# we could also change the column names (currently they are a, b, c)
colnames(my_dataframe) <- c("some_numbers", "my_modules", "logical_values")

# now let's have a look at it
my_dataframe
#>   some_numbers my_modules logical_values
#> 1            0      PS115           TRUE
#> 2           NA      PS509          FALSE
#> 3            2      PS300           TRUE
#> 4            3      PS938          FALSE
#> 5            4     PS9457          FALSE
```

Although note that in most cases we would probably import a dataframe from an external data file, for example using the functions `read.table` or `read.csv`.

3.2.5 Basic plotting in R

We can create plots using the function `plot()`. For example:

```
x = 1:10
y = 3*x - 5
plot(x, y)
```

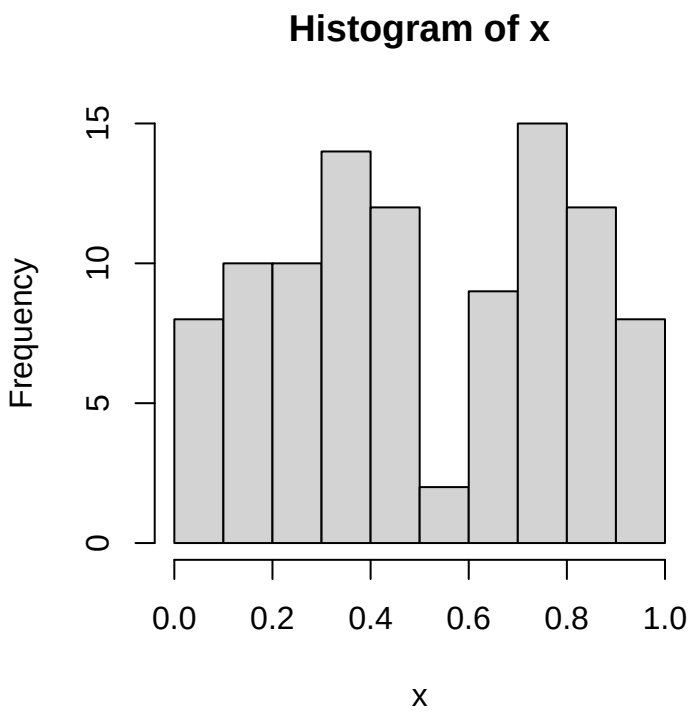


3.2.6 Other operations

3.2.6.1 Random number generation

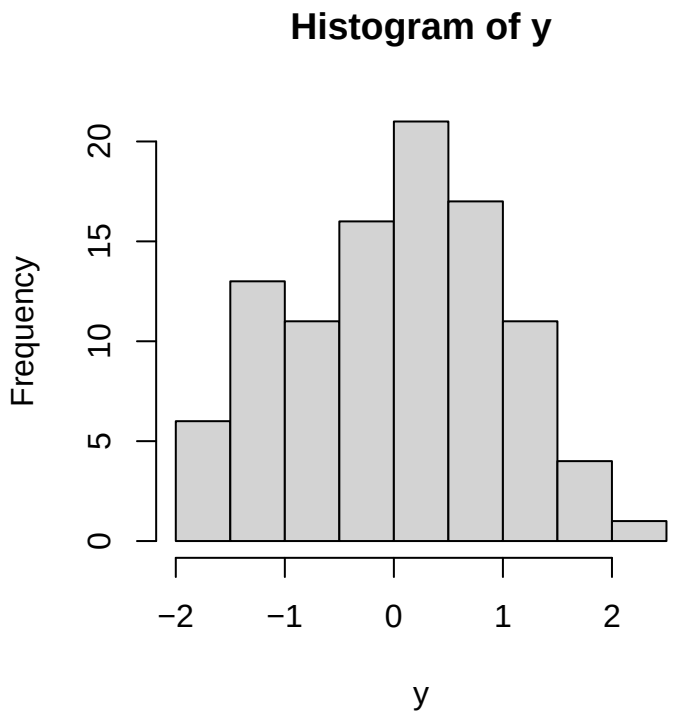
Generate uniformly distributed random numbers (function `runif()`)

```
x <- runif(100, min = 0, max = 1)
hist(x)
```



Generate numbers from a normal distribution

```
y <- rnorm(100, mean = 0, sd = 1)
hist(y)
```



3.2.7 Getting help

R has a lot of functions, and extra packages that can provides even more. It may seem a bit overwhelming, but it is very easy to get help about how to use a function: just type in a question mark, followed by the name of the function. For example, to see the help of the function we used above to generate the histogram, type

```
?hist
```

3.3 Resources for learning R

There is plenty of resources on the web to learn R. I will recommend a couple that I think are particularly well-done and useful:

- Software Carpentry tutorials on R for Reproducible Scientific Analysis
- The free book Learning Statistics with R by Danielle Navarro

Chapter 4

Linear models

This section will provide some worked examples of how to do analyses in R.

4.1 Simple linear regression

In this example[1] we will see how to import data into R and perform a simple linear regression analysis.

According to the standard big-bang model, the universe expands uniformly and locally, according to Hubble's law(Hubble, 1929)

$$\text{velocity} = \beta \times \text{distance}$$

where velocity and distance are the relative velocity and distance of a galaxy, respectively; and β is "Hubble's constant"[2]. Note that this is a simple linear equation, in which β indicate how much the variable velocity changes for each unitary increase in the variable distance.

According to this model β^{-1} gives the approximate age of the universe, but β is unknown and must somehow be estimated from observations of velocity and distance, made for a variety of galaxies at different distances from us. Luckily we have available data from the Hubble Space Telescope. Velocities are assessed by measuring the Doppler effect red shift in the spectrum of light that we receive from the Galaxies. Distance is estimated more indirectly, by using the discovery that in certain class of stars (Cepheids), which display fluctuations in diameter and temperature over a stable period, there is a systematic relationship between the period and their luminosity.

[1] Taken from Simon Wood's book on GAM(Wood, 2017).
[2] Note the Hubble "constant" is a constant only in space, not in time

We can load a dataset of measurements from the Hubble Space Telescope in R using the following code

```
d <- read.table(file="https://raw.githubusercontent.com/mattelisi/RHUL-stats/main/data/
                header=T)
```

`read.table` is a generic function to import dataset in text files (e.g. .csv files) into R. We use the argument `header=T` to specify that the first line of the dataset gives the names of the columns. Note that the argument `file` here is a URL, but it could be also a path to a file in our local folder. To see the help of this function, and what other arguments and features are available type `?read.table` in the R command line.

We can use the command `str()` to examine what we have imported

```
str(d)
#> 'data.frame':    24 obs. of  3 variables:
#>  $ Galaxy  : chr  "NGC0300" "NGC0925" "NGC1326A" "NGC1365" ...
#>  $ velocity: int  133 664 1794 1594 1473 278 714 882 80 772 ...
#>  $ distance: num  2 9.16 16.14 17.95 21.88 ...
```

This tells us that our data frame has 3 variables:

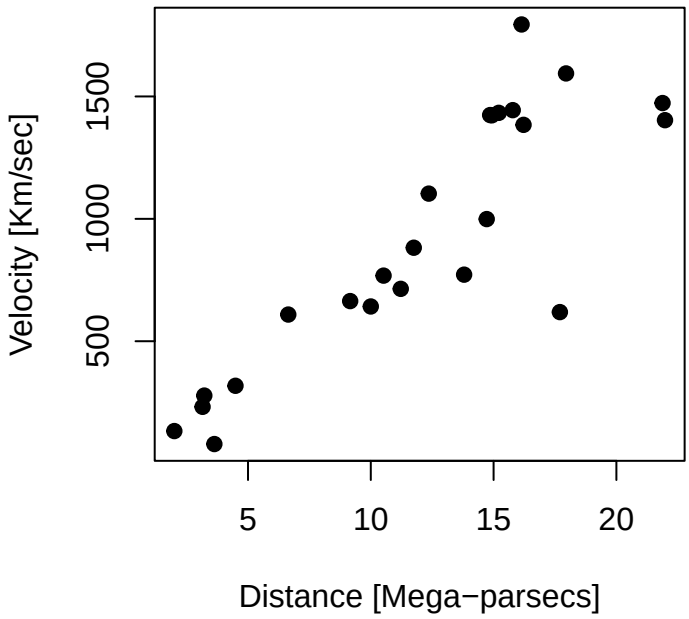
- `Galaxy`, the 'names' of the galaxies in the dataset
- `velocity`, their relative velocity in Km/sec
- `distance`, their distance expressed in Mega-parsecs[3]

We can plot[4] them using the following code:

```
plot(d$distance, # indicate which variable on X axis
     d$velocity, # indicate which variable on Y axis
     xlab="Distance [Mega-parsecs]",
     ylab="Velocity [Km/sec]",
     pch=19) # set the type of point
```

[3] 1Mega-parsec $= 3.09 \times 10^{19}$Km

[4] See `?plot` for more info about how to customize plots in R.



It is clear, from the figure, that the observed data do not follow Hubble's law exactly, but given the how these measurements were obtained (there is uncertainty about the true values of the distance and velocities) it would be surprising if they did. Given the apparent variability, what can be inferred from these data? In particular:

1. what value of β is most consistent with the data?
2. what range of β values is consistent with the data?

In order to make inferences we make some assumptions about the nature of the measurement noise. Specifically, we assume that measurements errors are well-characterized by a Gaussian distribution. This result in the following model:

$$y = \beta x + \epsilon$$
$$\epsilon \sim \mathcal{N}\left(0, \sigma_\epsilon^2\right)$$

which is essentially a linear regression but without the intercept: that is, whereas normally a linear regression model include an additive term that is not multiplied with the predictor (as in $y = \beta_0 + \beta_1 x + \epsilon$), which gives the expected value of the dependent variable when all predictors are set to zero, in this case the theory tells us we can assume the intercept (the term β_0) is zero and we can

ignore it.

We can fit the model with the function `lm` in R. Note that to tell R that I don't want to fit the intercept, I include in the formula the term `0 +` - this essentially tells R that the intercept term is set to zero[5]

```
hub.m <- lm(velocity ~ 0 + distance, d)
summary(hub.m)
#>
#> Call:
#> lm(formula = velocity ~ 0 + distance, data = d)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -736.5 -132.5  -19.0  172.2  558.0
#>
#> Coefficients:
#>          Estimate Std. Error t value Pr(>|t|)
#> distance   76.581      3.965   19.32 1.03e-15 ***
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 258.9 on 23 degrees of freedom
#> Multiple R-squared:  0.9419, Adjusted R-squared:  0.9394
#> F-statistic: 373.1 on 1 and 23 DF,  p-value: 1.032e-15
```

So, based on this data, **our estimate of the Hubble constant is 76.58 with a standard error of 3.96.** The standard error - which is the standard deviation of the sampling distribution of our estimates - gives an ideas of the range of values that is compatible with our data and could be used to compute a confidence intervals (roughly, we would expect that the 'true' values of the parameters lies in the interval defined by $\pm$ 2 standard errors 95% of the times).

So, how old is the universe?

The Hubble constant estimate have units of $\frac{\text{Km/sec}}{\text{Mega-parsecs}}$. A Mega-parsecs is 3.09×10^{19}Km, so we divide our estimate of $\widehat{\beta}$ by this amount. The reciprocal of $\widehat{\beta}$ then gives the approximate age of the universe (in seconds). In R we can calculate it (in years) as follow

```
# transform in Km
hubble.const <- coef(hub.m)/(3.09 * 10^(19))

# invert to get age in seconds
age <- 1/hubble.const
```

[5] A similar results would have been obtained using the notation `velocity ~ -1 + distance`.

```r
# use unname() to avoid carrying over
# the label "distance" from the model
age <- unname(age)

# transform age in years
age <- age/(60^2 * 24 * 365)

# age in billion years
age/10^9
#> [1] 12.79469
```

giving an estimate of about 13 billion years.

Chapter 5

Models for count data

This section will provide some examples of models that can deal with *count* data. Typically, count data occurs when the dependent variable is the counted number of occurrences of an event - for example the number of patients arriving in an emergency department (A&E) in a given time of the day - e.g. between 10:00 and 11:00. In this case, the dependent variable (the number of patients) has several characteristics that make it unsuitable for analysis with standard linear models such as linear regression: their distribution is discrete, composed only of non-negative integers, and is often positively skewed, with many observations having a value of 0.

Another characteristic is that the variance of the observations (e.g. the variance of the number of counts across observations within the same condition) increases with their expected value (e.g. the average number of counts for that condition)[1].

5.1 Poisson model

The simplest model that account for the characteristics mentioned above is a generalized linear model that assume a dependent variable with a Poisson distribution. The Poisson distribution has a single free parameter, usually notated with λ, which gives both the expected value (the mean) and the variance of the count variable. In fact it assumes that the variance has the same value as the mean.

Formally, a Poisson model is usually formulated as follow: let $y = (y_1, ..., y_n)$ be the dependent variable, consisting of counts (non-negative integers) and x the independent variable (predictor). Then

[1] This makes sense if you think that when the expected number of counts is very low, say ≈ 1, there cannot be many observations with very high counts - otherwise their average wouldn't be as low (recall that counts are strictly non-negative). In other words, the variance must be low when the average is also low.

$$y_i \sim \text{Poisson}\,(\lambda_i)$$

where[2]

$$\log(\lambda_i) = \beta_0 + \beta_1 x_i$$

or alternatively

$$\lambda_i = e^{\beta_0 + \beta_1 x_i}$$

Which suggest that we can use the exponential function (in R the function `exp()`) to calculate the predicted values of the mean and variance of the dependent variable for any values of the predictors.

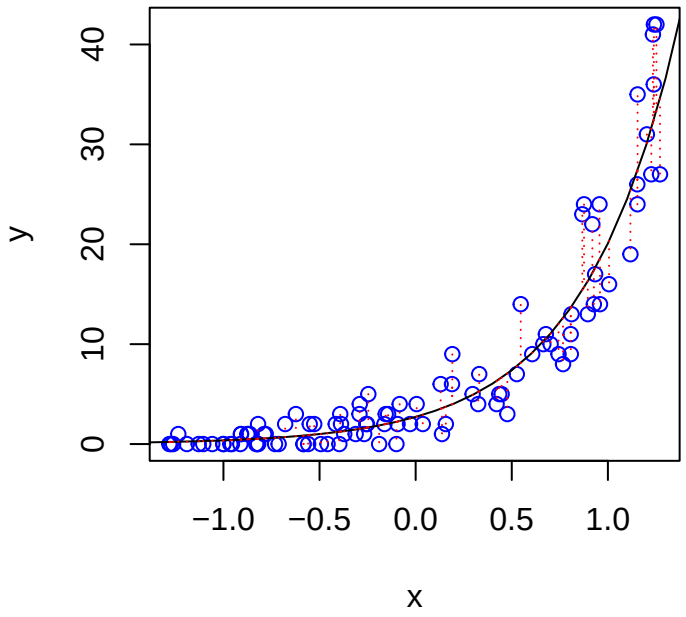
However, in practice, data often do not conform to the constraint of having identical mean and variance. Often the observed variance of the count is higher than what predicted according to the Poisson model (we say in this case that the data is *over-dispersed*).

How does over-dispersion look like?

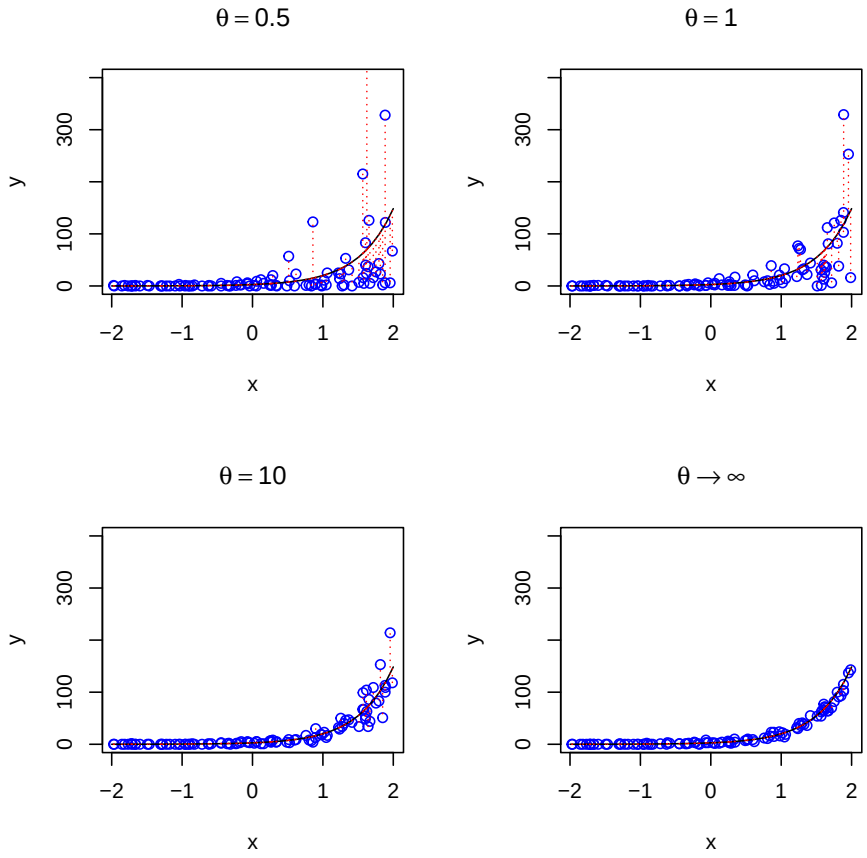
The type of data expected under a Poisson model is illustrated in the figure below, which shows 100 datapoints simulated from the model $y_i \sim \text{Poisson}\,(e^{1+2x_i})$. The vertical deviations of the datapoints from the line are consistent with the property of the Poisson distribution that the variance of hte count has the same value as their expected value, formally, $Var(y) = \mathbb{E}(y)$).

```
set.seed(2)
n <- 100
x <- runif(n, -1.3, 1.3)
a <- 1
b <- 2
linpred <- a + b*x # linear predictor part
y <- rpois(n, exp(linpred)) # simulate Poisson observations

plot(x,y,col="blue") # plot
x_ <- seq(-2,2,0.1)
lines(x_, exp(a+b*x_))
segments(x,exp(a+b*x),x,y, lwd=1,lty=3, col="red")
```

[2]This is the most common formulation, and is sometime referred to as log 'link' function, to indicate the fact that we have a function (the logarithm function) that 'link' the linear part of the model (the linear combination of the independent variables, here $\beta_0 + \beta_1 x_i$) with the parameter of the distribution of the dependent variable (here the Poisson rate parameter λ). This is a common component to all generalized linear models, for example for the logistic regression we have a 'logit' link function - the quantile function of the standard logistic distribution - that link the linear predictor part with the parameter p of the binomial distribution.



We can adjust the code above to simulate the same data with some degree of over-dispersion, using a negative binomial distribution, for different values of the precision parameter theta (θ), which regulate the degree of overdispersion. Importantly, these datapoints are simulated assuming the same function fo the average (expected) number of counts (same also as the previous figure), they just differe in the amount of overdispersion relative to a Poissone model. Note that for value arbitrarily large of the precision parameter $\theta \to \infty$ (bottom-right panel) the negative binomial converges to the Poisson.



5.2 Negative binomial model

As mentioned above, data are often overdispersed relative to the Poisson (that is, their variance is larger than the mean). This is an issue because when the data are overdispersed then our results may be largely influences by few extreme datapoints. Moreover, we will have the wrong estimated about the variability of the data. To account for overdispersion we can use the negative binomial which can be seen as a generalization of the Poisson model[3].

The negative binomial is very similar to the Poisson, but includes an additional

[3]The name *negative binomial* comes from a sampling procedure that give rise to this distribution: basically the negative binomial gives the probability of the number of successes in a sequence of independent and identically distributed Bernoulli trials before a specified (non-random) number of failures occur (this pre-fixed number of failures is a parameter of the negative binomial under an alternative parametrization of this distribution).

precision (or "reciprocal dispersion") parameter which I referred to as θ[4]. Essentially, whereas for the Poisson we had that $Var(y) = \mathbb{E}(y)$, now we have that

$$Var(y) = \mathbb{E}(y) + \frac{\mathbb{E}(y)^2}{\theta}$$

5.3 Examples

5.3.1 Anchoring and alcohol units

To see how this would work in practice, I use negative binomial to analyse a dataset that is based on an experiment run by Ryan McKay and his MSc students[5].

The goal of the study was to use the anchoring effect to:

> ...to see if we could attenuate under-reporting of alcohol consumption (which is a medical problem). Participants in a high-anchor condition were asked "do you drink more or less than 40 units of alcohol a week", and were then asked to estimate exactly how many units they'd consumed. Those in a low anchor condition were initially asked "do you drink more or less than 4 units of alcohol a week" before giving their precise estimate, and those in a control condition just gave their precise estimate.

In R we begin by loading the data

```
library(tidyverse)
#> -- Attaching packages ------------------- tidyverse 1.3.1 --
#> v ggplot2 3.3.6     v purrr   0.3.4
#> v tibble  3.1.7     v dplyr   1.0.9
#> v tidyr   1.2.0     v stringr 1.4.0
#> v readr   2.1.2     v forcats 0.5.1
#> -- Conflicts ---------------------- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
d <- read_csv("../data/nb_units.csv", show_col_types = F) # data availabel in the data folder of
d
#> # A tibble: 930 x 3
#>    gender condition units
#>    <chr>  <chr>     <dbl>
#>  1 Male   high         10
#>  2 Male   low           0
#>  3 Female high          1
```

[4] I choose 'theta' for consistency with the R output but note that this is sometime referred to as ϕ in the literature ¯_()_/¯

[5] Note that this is not the actual dataset that they collected but just some data that I simulated based on their research idea.

```
#>  4 Female high          2
#>  5 Male   high          0
#>  6 Female low           2
#>  7 Male   control      23
#>  8 Female low           2
#>  9 Female high          1
#> 10 Male   control       4
#> # ... with 920 more rows
```

We can calculate the mean and variance of the number of units reported in each conditions. This reveal that the variance across participants in the number of units reported is many times higher than the mean in the number of reported units.

```
d %>%
  group_by(condition) %>%
  summarise(Mean = mean(units),
            Variance = var(units)) %>%
  knitr::kable(digits=2,
               caption="Mean and Variance of weekly units of alcohol reported.")
```

Mean and Variance of weekly units of alcohol reported.

condition

Mean

Variance

control

7.53

138.35

high

11.97

363.69

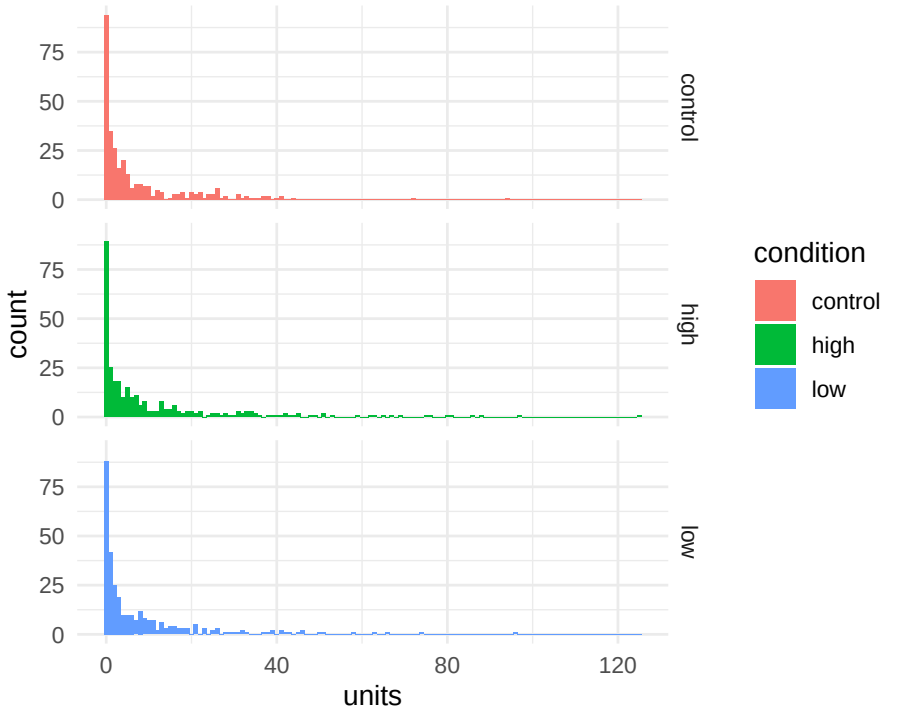
low

8.12

177.05

We can use ggplot2 library to visualize the distributions of reported units in each condition. We can see tha the distribution are skewed and contains many 0, which would make them unsuitable for an analysis with a linear regression model.

```
d %>%
  ggplot(aes(x=units, fill=condition)) +
  geom_histogram(binwidth=1)+
  facet_grid(condition~.) +
  theme_minimal()
```



To estimate the negative-binomial model, we can use the function `glm.nb()` from available in the `MASS` package. Our predictor condition is categorical with 3 levels and therefore it is coded internally as a set of 2 dummy variables. We can see how the contrast is set using

```
d$condition <- factor(d$condition) # tell R that this is a categorical factor
contrasts(d$condition)
#>         high low
#> control    0   0
#> high       1   0
#> low        0   1
```

This indicate that `control` is our baseline condition and the model will have 2 coefficients that code for the difference in the `high` and `low` anchoring condition relative to the control one.

Note also that for this analysis the variable `units` must contain only integer values - if participants reported non-integer values (e.g. a bottle of lager is about 1.7 units) we could divide everything by the minimum common denominator so that we end up with integer values.

The following command can be used to estimate the model and examine the results

```
library(MASS)
#>
#> Attaching package: 'MASS'
#> The following object is masked from 'package:dplyr':
#>
#>     select
nb01 <- glm.nb(units ~ condition, data = d)
summary(nb01)
#>
#> Call:
#> glm.nb(formula = units ~ condition, data = d, init.theta = 0.3852966632,
#>     link = log)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -1.6347  -1.5261  -0.5620   0.1311   2.5895
#>
#> Coefficients:
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    2.01856    0.09396  21.482  < 2e-16 ***
#> conditionhigh  0.46365    0.13218   3.508 0.000452 ***
#> conditionlow   0.07564    0.13255   0.571 0.568251
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for Negative Binomial(0.3853) family taken to be 1)
#>
#>     Null deviance: 1040.0  on 929  degrees of freedom
#> Residual deviance: 1025.2  on 927  degrees of freedom
#> AIC: 5656.8
#>
#> Number of Fisher Scoring iterations: 1
#>
#>
#>               Theta:  0.3853
#>           Std. Err.:  0.0196
#>
```

```
#>  2 x log-likelihood:  -5648.7830
```

We can see from output that the condition `high` anchoring elicited reports with higher number of alcohol units than the `control` condition.

We can use the model to make a more precise statement about the size of the difference. We can use the value of the coefficients to calculate the predicted values of counts. The exact values of the coefficients can be accessed from the fitted model using the `$` operator

```
nb01$coefficients
#>   (Intercept) conditionhigh  conditionlow
#>    2.01856406    0.46365080    0.07563937
```

The values are combined together according to the dummy variables coding for the condition and represents the linear predictor part of the model:

$$\lambda_i = \exp\left(\beta_0 + \beta_1 \times D_{\text{high}} + \beta_2 \times D_{\text{low}}\right)$$

where I have used the notation D_{high} and D_{low} to indicate the two dummy variable, whose value is 1 for observation in the `high` and `low` conditions, respectively, and zero otherwise.

β_0 is a common notation for the intercept parameter - in this case it gives the expected number of alcohol units in the control condition (because for observations in the control condition we have that $D_{\text{high}} = D_{\text{low}} = 0$). Thus our model predict an average number of counts in the control condition of

```
exp(nb01$coefficients["(Intercept)"]) # equivalent to exp(nb01$coefficients[1])
#> (Intercept)
#>    7.527508
```

(Compare this value with the table above).

Furthermore, our models tells us also that the number of reported alcohol units increase multiplicatively in the `high` condition by a factor of

```
exp(nb01$coefficients["conditionhigh"])
#> conditionhigh
#>      1.589868
```

In fact the predicted number of counts in the `high` condition can be derived from the model as

```
exp(nb01$coefficients["(Intercept)"])  * exp(nb01$coefficients["conditionhigh"])
#> (Intercept)
#>    11.96774
```

or equivalently

```
exp(nb01$coefficients["(Intercept)"] + nb01$coefficients["conditionhigh"])
#> (Intercept)
#>    11.96774
```

Finally, note that we can use the `sjPlot` library to prepare a fancy version of the model output, and we can see that the multiplicative factor that describe the increase in reported units is called here an *incidence ratio*[6].

```
library(sjPlot)
#> Install package "strengejacke" from GitHub (`devtools::install_github("strengejacke/
tab_model(nb01)
```

units

Predictors

Incidence Rate Ratios

CI

p

(Intercept)

7.53

6.29 – 9.10

<0.001

condition [high]

1.59

1.23 – 2.06

<0.001

condition [low]

1.08

0.83 – 1.40

0.568

Observations

930

R2 Nagelkerke

0.023

[6] Although honestly I am not sure how common is this terminology

5.3.1.1 Adding predictors

The dataset include also information about the gender of the participants. We may hypothesize that male participants drink more than female ones[7]. Does taking this into account improve the accuracy of our modelling?

To test this, we can estimate an additional model with also gender as predictor. We can compare this to the previous one using a *likelihood-ratio test*. This is based on a theorem which states that the difference in log-likelihood[8] between nested models is (asymptotically) distributed according to a Chi-squared distribution, therefore allowing the calculation of a p-value. In R this can be done using the function `anova()`.

First, let's fit an additional model with the extra predictor `gender`

```
nb02 <- glm.nb(units ~ condition + gender, data = d)
summary(nb02)
#>
#> Call:
#> glm.nb(formula = units ~ condition + gender, data = d, init.theta = 0.4212084665,
#>     link = log)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -1.7705  -1.4368  -0.5506   0.1926   2.3480
#>
#> Coefficients:
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept)   1.495941   0.105452  14.186  < 2e-16 ***
#> conditionhigh 0.426116   0.127159   3.351 0.000805 ***
#> conditionlow  0.006701   0.127720   0.052 0.958156
#> genderMale    0.909721   0.103969   8.750  < 2e-16 ***
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for Negative Binomial(0.4212) family taken to be 1)
#>
#>     Null deviance: 1115.5  on 929  degrees of freedom
#> Residual deviance: 1025.4  on 926  degrees of freedom
#> AIC: 5587.9
```

[7] For the sake of the example we use only 2 gender categories, but in a real study we should be mindful to include more options for non-binary / third gender participants, as McKay's students did in the real study.

[8] In this model the parameters are estimated via maximum likelihood, which amounts to choosing the values of the parameters that maximize the probability (likelihood) of the data under the model. Thus when we refer to the log-likelihood of a model we indicate the logarithm of the maximized value of the likelihood function.

```
#>
#> Number of Fisher Scoring iterations: 1
#>
#>
#>               Theta:  0.4212
#>           Std. Err.:  0.0219
#>
#>  2 x log-likelihood:  -5577.9060
```

This indicate that indeed male participants report on average

```
exp(nb02$coefficients["genderMale"])
#> genderMale
#>    2.48363
```

times more units of alcohol per week than females.

We can already see that the difference due to gender is significant, but nevertheless let's compare them using a likelihood ratio test.

```
anova(nb01, nb02)
#> Likelihood ratio tests of Negative Binomial Models
#>
#> Response: units
#>                 Model     theta Resid. df    2 x log-lik.
#> 1           condition 0.3852967       927       -5648.783
#> 2 condition + gender 0.4212085       926       -5577.906
#>     Test    df LR stat. Pr(Chi)
#> 1
#> 2 1 vs 2     1 70.87633       0
```

Here the value of the likelihood ratio statistic is NA, 70.88 and under the null hypothesis that any improvement of model fit obtained after adding gender as predictor is due to chance is distributed as a Chi-square with 1 degree of freedom.

5.3.1.2 Plotting model fit

It's not straightforward to visualize the model fit to the data - the code below give one possibility:

```
# here I make a new data matric for claculating the prediction of the model
nd <- expand.grid(condition=unique(d$condition),
                  units = 0:max(d$units),
                  KEEP.OUT.ATTRS = F)

# use the predict() function to calculate the predicted counts for each condition
nd$predicted_units <- predict(nb01, newdata=nd, type="response")
```

```
# here I use the dnbinom() function - which gives the probability density of
# the negative binomial - to calculate the probability of the observations under the model
nd$pred_density <- dnbinom(nd$units, mu=nd$predicted_units, size=nb01$theta)

# finally take all together and plot
d %>%
  ggplot(aes(x=units, fill=condition)) +
  geom_histogram(aes(y=..density..),binwidth=1, color="white")+
  geom_line(data=nd, aes(x=units, y=pred_density),size=1) +
  facet_grid(condition~.) +
  theme_minimal() +
  coord_cartesian(xlim=c(0,40))
```

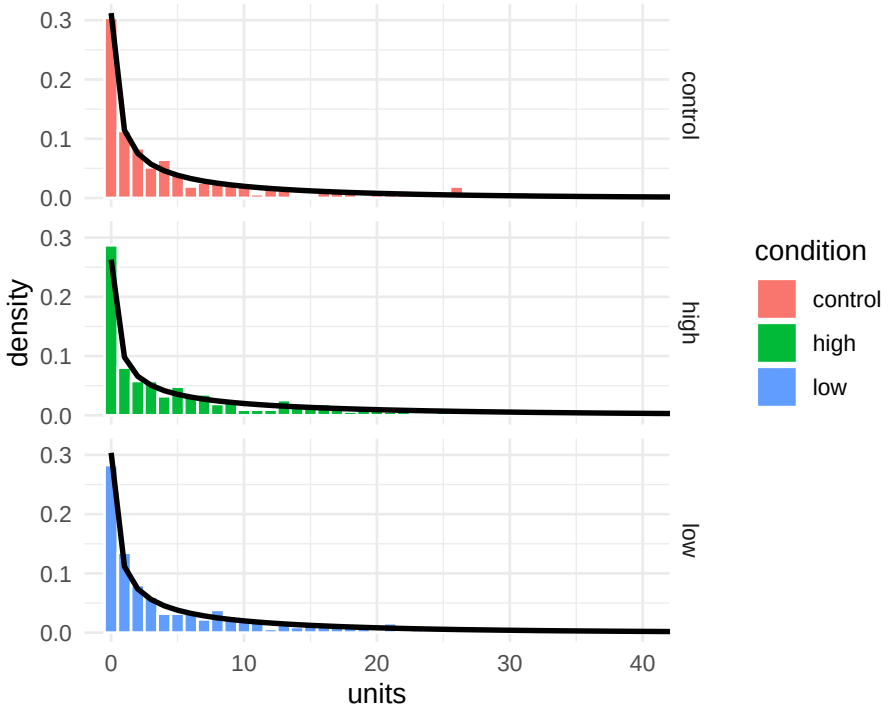
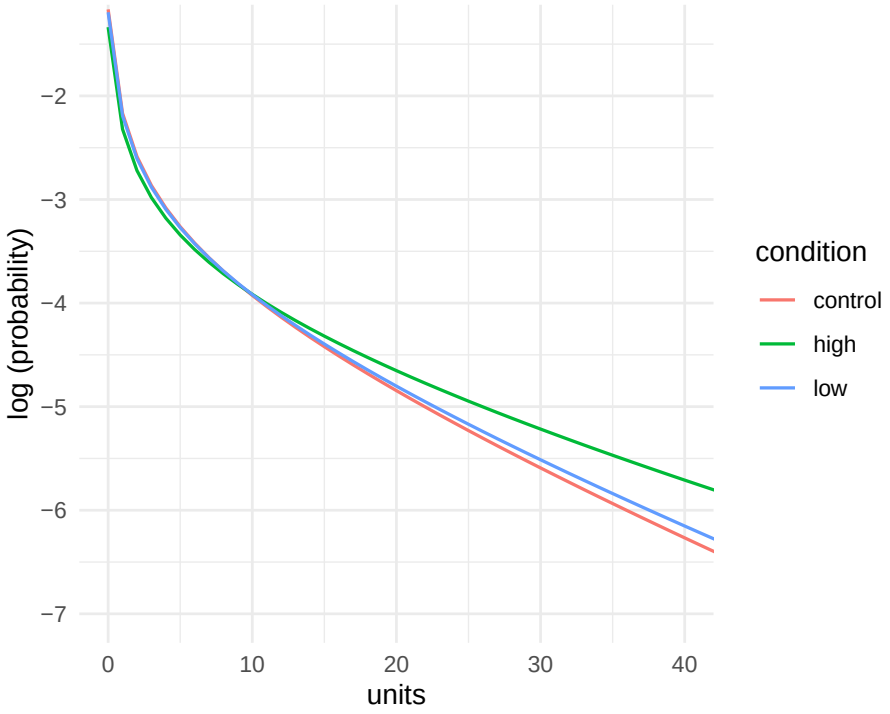


Figure 5.1: The black line represent the predicted probability of the data (note that I clipped the x-axis at 40)

Admittedly the probability of the data under the model (the black lines) looks quite similar across the three panels, however, the model does assign higher probability to higher count values in the `high` condition compared to the other ones - we can see this by putting them together in the same panel, and by plotting the logarithm of the probability instead of the probability itself. These

changes in the probability of the data may not seems large when looked at in this way, but they amount to quite substantial changes in the average number of counts - recall that in the `high` condition participants reported on average nearly 1.6 times the number of alcohol units than in the control condition.

```
nd %>%
  ggplot(aes(x=units, y=log(pred_density), color=condition))+
  geom_line(size=0.6)+
  theme_minimal() +
  coord_cartesian(xlim=c(0,40),ylim=c(-7,-1.4))+
  labs(y="log (probability)")
```



Chapter 6

Meta-analyses

For running meta-analyses, we recommend the `metafor` package for R (see link 1, link 2).

A comprehensive, hands-on guide on how to use this package is provided in the book by Harrer and colleagues (Harrer et al., 2021), freely available at this link.

An alternative to the `metafor` package is to Bayesian multilevel modelling (also discussed in the book by Harrer and colleagues). A more technical discussion of Bayesian multilevel modelling for meta-analyes is provided in this paper by Williams, Rast and Bürkner (Williams et al., 2018).

Chapter 7

Missing data

7.1 Types of missing data

Following the work of Rubin(RUBIN, 1976), missing data are typically grouped in 3 categories:

- Missing completely at random (**MCAR**). This assumes that the probability of being missing is the same for all cases; this implies that the mechanisms that causes missingness is not related in any way to the data. For example, say, there's a known unpredictable error on the server side that prevented recording some responses for some respondents to a survey. As the missingness is entirely independent on the respondents' characteristics, this would be MCAR. When the data are MCAR we can ignore a lot of the complexities and just do a *complete-case* analysis (that is, simply exclude incomplete observations from the dataset). A part from possible loss of information, doing a complete case analysis should not introduce bias in the results. In practice, however, it is difficult to establish whether the data are truly MCAR. Ideally, to argue that data are MCAR, one should have a good idea of the mechanisms that caused missingness (more on this below). Formally, data is MCAR if

  $$\Pr(R = 0|Y_{\text{obs}}, Y_{\text{mis}}, \psi) = \Pr(R = 0|\psi)$$

  where R is an indicator variable that is set to 0 for missing data and 1 otherwise; $Y_{\text{obs}}, Y_{\text{mis}}$ indicate observed and missing data, respectively; and ψ is simply a parameter that determine the overall (fixed) probability of being missing.

- Missing at random (**MAR**). A less strong assumption about missingness is that it is systematically related to the observed but not the unobserved data. For example, data are MAR if in a study male respondents are less likely to complete a survey on depression severity than female respondents

- that is, the probability of reaching the end of the survey is related to their sex (fully observed) but not the severity of their symptoms. Formally, data is MAR if

$$\Pr(R = 0|Y_{\text{obs}}, Y_{\text{mis}}, \psi) = \Pr(R = 0|Y_{\text{obs}}, \psi)$$

When data are missing at random (MAR) the results of complete case analyses may be biased and a common approach to deal with this is to use imputation. Stef van Buuren has a freely available online book on this topic(van Buuren, 2018). Among other things, it illustrates how to do multiple imputation in R with examples.

- Missing not at random (**MNAR**). This means that the probability of being missing varies for reasons that are unknown to us, and may depends on the missing values themselves. Formally this means that $\Pr(R = 0|Y_{\text{obs}}, Y_{\text{mis}}, \psi)$ does not simplify in any way. This case is the most hard to handle: a complete case analyses may or may not be biased, but there is no way of knowing it and we may have to find more information about what caused missingness.

7.2 Deciding whether the data are MCAR

As MCAR is the only scenario in which it is safe to do a complete case analysis, it would seem useful to have way to test this assumption. Some approaches have been proposed to test whether the data are MCAR, but they are not widely used and it's not clear how useful they are in practice. For example one could run a logistic regression with "missingness" as dependent variable (e.g. an indicator variable set to 1 if data is missing and 0 otherwise), and all other variables as predictors - if the data are MCAR then none of the predictors should predict missingness. A popular alternative, implemented in several software packages is Little's test(Little, 1988).

Technically, these approaches can help determine whether the missingness depends on some observed variables (that is, if they are MAR), but strictly speaking cannot exclude missingness due to unobserved variables (MNAR scenario). Nevertheless, if one has good reasons to believe that the data are MCAR, and want to add some statistical test that corroborate this assumption, these could be reasonable tests to do. However, it remains important to also discuss openly possible reasons and mechanisms of missingness, and explain why we deem it a priori plausible that the data are MCAR. In fact, *statistical tests alone cannot tell whether data are missing completely at random.* The terms MCAR, MAR and MNAR refers to the *causal* mechanisms that is responsible for missing data and, strictly speaking, causal claims cannot be decided uniquely on the basis of a simple statistical test. If the data "pass" the test it would provide some additional support to the assumption that they are MCAR, but in and of itself the test alone does not fully satisfy the assumptions of MCAR. To see why note that MCAR (as formally defined above) assumes also that there should be no relationship between the missingness on a particular variable and the values of

that same variable: but since this is a question about what is missing from the data, it cannot be tested with any quantitative analysis of the available data. Finally, it should be added that as these are null-hypothesis significance test, a failure to reject the null hypothesis does not, in and of itself, provide evidence for the null hypothesis (that the data are MCAR). It may be also that we don't have enough power to reliably detect the pattern in the missingness.

7.3 Causal analysis and Bayesian imputation

The best and most principled approach to deal with missingness (at least in my opinion) is to think hard about the causal mechanisms that may determine missingness, and use our assumption about the causal mechanisms to perform a full Bayesian imputation (that is , treating the missing data as parameter and estimating them).

I plan to create and include here a worked example of how to do this; in the meantime interested readers are referred to Chapter 15 (in particular section 15.2) of the excellent book by Richard McElreath *Statistical Rethinking*(McElreath, 2020) which present a very accessible worked example of how to do this in R.

Bibliography

Harrer, M., Cuijpers, P., A, F. T., and Ebert, D. D. (2021). *Doing Meta-Analysis With R: A Hands-On Guide*. Chapman & Hall/CRC Press, Boca Raton, FL and London, 1st edition.

Hubble, E. (1929). A relation between distance and radial velocity among extra-galactic nebulae. *Proceedings of the National Academy of Sciences*, 15(3):168–173.

Little, R. J. A. (1988). A test of missing completely at random for multivariate data with missing values. *Journal of the American Statistical Association*, 83(404):1198–1202.

McElreath, R. (2020). *Statistical Rethinking, A Course in R and Stan*. Chapman & Hall/CRC Press, 2nd edition.

RUBIN, D. B. (1976). Inference and missing data. *Biometrika*, 63(3):581–592.

van Buuren, S. (2018). *Flexible imputation of missing data*. Chapman & Hall/CRC Press, 2nd edition.

Williams, D. R., Rast, P., and Bürkner, P. C. (2018). Bayesian meta-analysis with weakly informative prior distributions.

Wood, S. (2017). *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC Press, 2nd edition.