Introduction to

Part II

by
Knut Helge Jensen

Today's main tasks

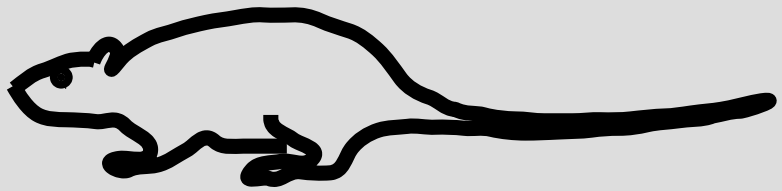
1) Dataset manipulation

2) How to think when you want to find a model for your data

Dataset manipulation

Dataset: Body mass (g) of *Rattus norvegicus* depending on sex.

```
#Add dataset directly with R syntax:
rats.df <- read.table(header=T, text="
Sex Body.mass
female 240.62
female 235.55
female 249.91
female 258.42
female 272.92
male 361.29
male 353.17
male 347.15
male 353.11
male 347.75
")
```



How to add a variable to a dataset?

```
Liver.mass <- c(9.63, 9.51, 10.06, 10.14, 10.92,
14.3, 13.93, 13.7, 13.87, 14.01)

rats.df$Liver.mass <- Liver.mass
```

When you want to put a variable inside a dataset; here the variable we have called **`Liver.mass`**, you first write the name of the dataset, then a dollar sign, followed by the name you give to the new variable.

On the right hand side of the assignment symbol (**`<-`**), you describe what to put into the dataset. In the above example we put a variable object called **`Liver.mass`** into the dataset. In this example, the variable was created before we put it into the dataset but it is also possible to do it directly:

```
rats.df$Liver.mass <- c(9.63, 9.51, 10.06, 10.14,
10.92, 14.3, 13.93, 13.7, 13.87, 14.01)
```

…add variable continued

You may of course add a variable based on calculations from other variables in the dataset, just like you would do calculations in a spreadsheet. It is easier in R since you don't copy formulas over many cells. Below is an example of how to calculate individual liver mass measured in percent of total body mass and create a new variable of the results:

```
attach(rats.df)*
rats.df$LM.percent <- Liver.mass*100/Body.mass
```

Another example with the use of the logical function **`ifelse`**: Create a variable that describes when liver mass is greater than or equal to 4% of total body mass (given the value 1) or below 4% (given the value 0):

```
attach(rats.df)*
rats.df$LM4 <- ifelse(LM.percent >= 4, 1,0)
```

* The attach command is explained after next slide.

Let's have a look at the dataset after the three variables have been added:

```
> rats.df
      Sex Body.mass Liver.mass LM.percent LM4
1  female    240.62       9.63   4.002161   1
2  female    235.55       9.51   4.037359   1
3  female    249.91      10.06   4.025449   1
4  female    258.42      10.14   3.923845   0
5  female    272.92      10.92   4.001173   1
6    male    361.29      14.30   3.958039   0
7    male    353.17      13.93   3.944276   0
8    male    347.15      13.70   3.946421   0
9    male    353.11      13.87   3.927954   0
10   male    347.75      14.01   4.028756   1
```

The attach command

In an unattached data frame, a call for one of its variables is done like this: **<name-of-data-frame>$<name-of-vector>**

Thus, to call for Liver.mass from the rats.df dataset:
rats.df$Liver.mass

You may use the attach command to make the variables within a data frame available in the search path:

attach(name-of-data-frame)

For the rats.df dataset: **attach(rats.df)**

After attaching the dataset to the search path, you may call for the same variable directly by typing: **Liver.mass**

…attach command continued

If I did not use the attach command on the **`rats.df`** dataset, the calculations of percent liver mass would have to be coded as follows:

```
rats.df$Liver.mass*100/rats.df$Body.mass
```

However, I would generally recommend not to use attach. The two most common pitfalls when relying on attach to perform direct calls to variables are:

1) You may have a variable placed outside a data frame with the same name as a variable placed inside a data frame. If not specified, R will by default use the one outside.

2) If you forget to update the search path, i.e. not redo the attach command after you have done some changes to a dataset, R will use the dataset that is placed in the search path, i.e. as it was before the changes.

How to get subsets of your data?

You may use square brackets to get subsets of your dataset:

General syntax:
`dataset.df[i,j]`
where i refer to row number and j refers to column number

Sex	Body.mass	Liver.mass	LM.percent	LM4
female	240.62	9.63	4.002161	1
female	235.55	9.51	4.037359	1
female	249.91	10.06	4.025449	1
female	258.42	10.14	3.923845	0
female	272.92	10.92	4.001173	1
male	361.29	14.30	3.958039	0
male	353.17	13.93	3.944276	0
male	347.15	13.70	3.946421	0
male	353.11	13.87	3.927954	0
male	347.75	14.01	4.028756	1

To extract row 1 from rats.df:
`rats.df[1,]`

```
     Sex Body.mass Liver.mass LM.percent LM4
1 female    240.62       9.63   4.002161   1
```

To extract column 2:
`rats.df[,2]`

```
[1] 240.62 235.55 249.91 258.42 272.92 361.29 353.17 347.15 353.11 347.75
```

...subsets continued

To extract row 4 from column 2:

```
rats.df[4,2]
[1] 258.42
```

To extract row 4 from columns 1, 2, and 3:

```
rats.df[4,1:3]
     Sex Body.mass Liver.mass
4 female    258.42      10.14
```

To extract row 4 from columns 1, 3, and 5:

```
rats.df[4,c(1,3,5)]
     Sex Liver.mass LM4
4 female      10.14   0
```

	Sex	Body.mass	Liver.mass	LM.percent	LM4
	female	240.62	9.63	4.002161	1
	female	235.55	9.51	4.037359	1
	female	249.91	10.06	4.025449	1
	female	258.42	10.14	3.923845	0
	female	272.92	10.92	4.001173	1
	male	361.29	14.30	3.958039	0
	male	353.17	13.93	3.944276	0
	male	347.15	13.70	3.946421	0
	male	353.11	13.87	3.927954	0
	male	347.75	14.01	4.028756	1

...subsets continued

Instead of using column numbers, you may use the column names:

Exactly the same example as the previous; extracting row 4 from columns 1, 3, and 5, but by using column names:

Sex	Body.mass	Liver.mass	LM.percent	LM4
female	240.62	9.63	4.002161	1
female	235.55	9.51	4.037359	1
female	249.91	10.06	4.025449	1
female	258.42	10.14	3.923845	0
female	272.92	10.92	4.001173	1
male	361.29	14.30	3.958039	0
male	353.17	13.93	3.944276	0
male	347.15	13.70	3.946421	0
male	353.11	13.87	3.927954	0
male	347.75	14.01	4.028756	1

```
rats.df[4,c("Sex","Liver.mass","LM4")]
```

```
     Sex Liver.mass LM4
4 female      10.14   0
```

…subsets continued

For creating subsets, it is probably more useful to to create a new dataset which is a subset of another. It is recommended to use the **`subset`** function for this purpose:

Example: You want to create a subset of rats.df
that only contains females:

```
fem.rats.df <- subset(rats.df, Sex=="female")
fem.rats.df
```

```
     Sex Body.mass Liver.mass LM.percent LM4
1 female    240.62       9.63   4.002161   1
2 female    235.55       9.51   4.037359   1
3 female    249.91      10.06   4.025449   1
4 female    258.42      10.14   3.923845   0
5 female    272.92      10.92   4.001173   1
```

…subsets continued

Subsets based on values from two variables.

Example: you want to extract the females but only when Liver.mass is equal to or above four percent (here by using the LM4 variable):

```
fem.bigL.df <- subset(rats.df, Sex=="female" & LM4==1)
fem.bigL.df
```

```
     Sex Body.mass Liver.mass LM.percent LM4
1 female    240.62       9.63   4.002161   1
2 female    235.55       9.51   4.037359   1
3 female    249.91      10.06   4.025449   1
5 female    272.92      10.92   4.001173   1
```

…subsets continued

Subsets based on two different values within the same variable and in addition a value of a second variable.

Example: you want to extract rats that have Body.mass above 300 or less than 240, but only if LM4 is 1, i.e. if liver mass is equal to or larger than 4% of Body.mass.

```
rats2.df <- subset(rats.df, (Body.mass>300 | Body.mass<240) & LM4==1)
rats2.df
```

```
       Sex Body.mass Liver.mass LM.percent LM4
2   female     235.55       9.51    4.037359   1
10    male     347.75      14.01    4.028756   1
```

Note! The pipe symbol (**|**) means **or** in the syntax. Many "non-programmers" tend to use **&** instead of **|** here, but remember to think logically: You can't have a variable that is both larger than 400 **and** less than 241, but it can be larger than **or** less than. Do also note the brackets: Use brackets around multiple subsettings from the same variable when it is followed by more subsettings from other variable(s)!

…subsets continued

Some useful R syntax symbols connected to subsetting etc.:

`x == y`	x is equal to y
`x != y`	x is unequal to y
`x < y`	x is less than y
`x <= y`	x is less than or equal to y
`x > y`	x is greater than y
`x >= y`	x is greater than or equal to y

How to restructure a dataset?

Assume you have noted down growth of males and females of a species as shown below, where t0, t1, t2 etc. represent body mass (g) measured every 10th day.

You want to restructure the dataset so that body mass becomes a single variable and with an additional variable describing time in days from first measure. In other words: Restructure from horizontal to vertical orientation of the data.

Sex	t0	t1	t2	t3	t4	t5
female	78.37	118.35	158.21	198.83	242.38	283.22
female	77.1	113.86	146.18	180.68	225.5	265.59
female	82.14	128.97	163.06	201.12	246.68	286.84
female	78.28	113.94	151.37	193.99	234.13	273.76
female	81.76	121.07	163.56	198.67	238.96	280.02
male	79.6	133.81	195.27	257.04	315.25	375.01
male	83.28	138.46	192.7	253.38	312.57	372.16
male	81.43	138.03	195.3	260.05	320.2	380.85
male	74.45	134.38	197.77	255.44	316.09	375.13
male	81.56	137.72	196.81	261.6	319.23	378.53

…restructure dataset continued

```
#Import of dataset:
growth.df <- read.table("http://folk.uib.no/nzlkj/data/rats2.txt",
header=T)

#Loading required library:
library(tidyr)

#The restructuring syntax:
growth2.df <- gather(data=growth.df, key=Time,
factor_key=T, value=Body.mass, t0:t5)
```

The syntax of the **gather** function explained:

data=	Name of dataset to be restructured.
key=	Name of column to become index variable with levels representing each variable that is combined.
factor_key=T	To store key as a factorial variable.
value=	Name of variable representing combined variables.

…restructure dataset continued

The result of the restructuring:

Sex	Time	Body.mass
female	t0	78.37
female	t0	77.10
female	t0	82.14
female	t0	78.28
female	t0	81.76
male	t0	79.60
male	t0	83.28
male	t0	81.43
male	t0	74.45
male	t0	81.56
female	t1	118.35
female	t1	113.86
etc.	etc.	etc.

How to make a variable from **Time** that represents actual distance in time?

Since the **gather** function uses header names of the variables to be combined (here **t0, t1** etc.) when creating the **key=** variable (here **Time**), the **key=** variable becomes categorical. You may therefore want a new variable for time that represents actual distance in time between observations. Since body mass measures were repeated every 10th day, you may create the new variable by the following syntax:

```
Day.list <- c(0,10,20,30,40,50)
growth2.df$Day <- Day.list[growth2.df$Time]
```

Syntax explained:
The first line creates an index variable that represents days of measurements. The second line performs indexing based on this index variable. It starts with the level of **Time** that would appear first when put in descending order (t0) and replace it with the first number in **Day.list** (0), then the next level of **Time** (t1) get the next number of **Day.list** (10) etc.

How to make an aggregated version of a dataset?

Example: You want a new dataset from growth2.df that contains the same variables but only with mean values of Body.mass for each day, and sex. This is easily done by the **aggregate** function:

```
mean.df <- aggregate(Body.mass~Day+Time+Sex,
data=growth2.df, mean)
```

	Day	Time	Sex	Body.mass
1	0	t0	female	79.530
2	10	t1	female	119.238
3	20	t2	female	156.476
4	30	t3	female	194.658
5	40	t4	female	237.530
6	50	t5	female	277.886
7	0	t0	male	80.064
8	10	t1	male	136.480
9	20	t2	male	195.570
10	30	t3	male	257.502
11	40	t4	male	316.668
12	50	t5	male	376.336

How to make your own functions?

The general structure of a function that you write for R is:

```
functionname <- function(arg1, arg2, ... )
{
statements
return(object)
}
```

The object returned can be any data type or graphics.

…make your own functions continued

A simple example: You are doing so many calculations of sphere volumes that you want to make a function for it:

```
volume.sphere <- function(radius)
{
if (missing(radius))
stop("Need to specify radius of cylinder for calculations.")
a <- 4/3*pi*radius^3
return(a)
}
```

Using the function on a single sphere radius value:

```
volume.sphere(15)
```

Using the function on a variable containing radii of three spheres:

```
r <- c(5,10,15)
volume.sphere(r)
```

How to find a correct type of model for your data?

Need the answer to the following three questions

1) What is the hypothesis?

2) What is the underlying distribution of your response variable and what is the predictor(s)?

3) What type of study design do you have?

1) What is the hypothesis?

To define a clear hypothesis is always needed before you think about any analysis. If you don't know what you want to test there is of course no point in searching for a suitable test.

Hypothesis example:
H_0: The body mass (g) of *Rattus norvegicus* do not depend on sex.

2) What is the underlying distribution of the response variable and what is the predictor(s)?

In statistical models, the response* variable is a function of the predictor* variable(s). Thus, the difference between the two types of variables is tightly linked to the hypothesis.

$$\text{Body mass}_i = \beta_0 + \beta_1 \text{Sex}_i + e_i$$

*Another term for response variable is *dependent variable* Normally we want to explain its variability by one or more predictor variables.

*Other terms for predictor is *independent variable* or *explanatory variable*.

Sex	Body.mass
female	240.62
female	235.55
female	249.91
female	258.42
female	272.92
male	361.29
male	353.17
male	347.15
male	353.11
male	347.75

To know the properties about your response and predictor(s) are key to choose a proper statistical model

Response variable

- continuous?
- count?
- proportion?
- binary?
- time-event?

We care about this mainly because statistical models differ in assumptions about the underlying distribution of the response variable.

Predictor variable(s)

- categorical?
- continuous?
- both? (when you have more than one variable)

We care about this mainly because of how we should interpret the output of our models.

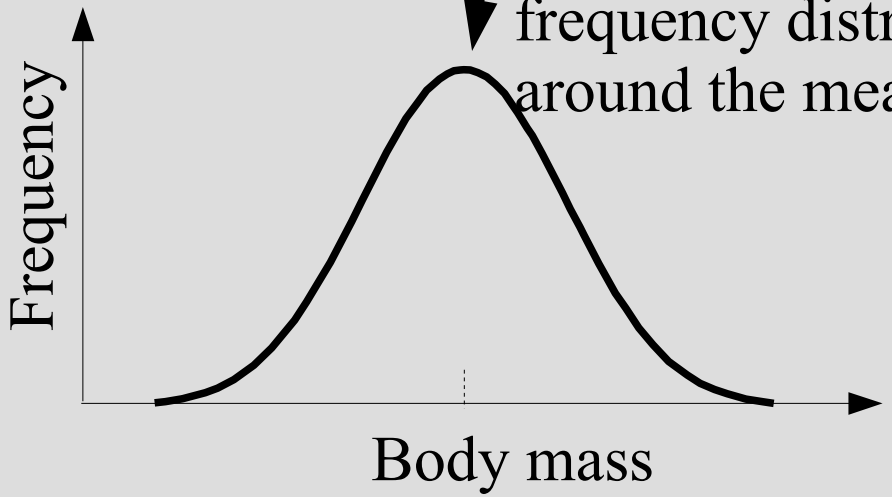
When to assume that the underlying distribution of the response variable follows a **normal** distribution? (also called Gaussian distribution after Johann Carl Friedrich Gauss, 1777-1855)

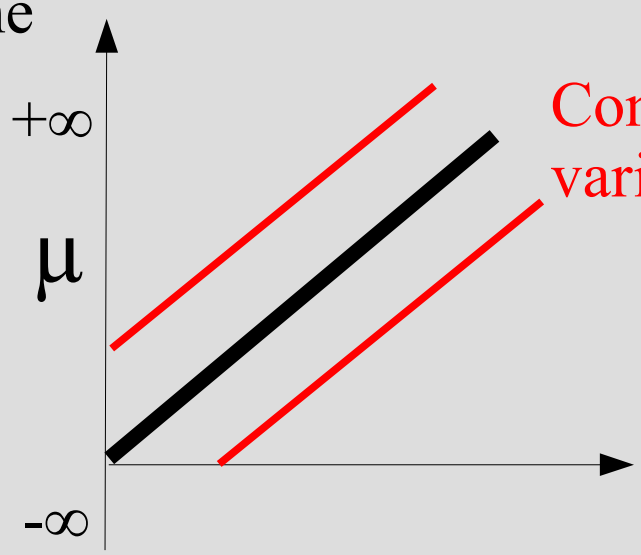
- Typically when the response variable represents continuous data like body mass, blood pressure, body temperature, errors in measurements etc.

Gauss's portrait published in Astronomische Nachrichten 1828

Example: A random sample from a population of rats, with measurements of individual body mass.

Gives a characteristic bell-shaped curve for the frequency distribution around the mean.



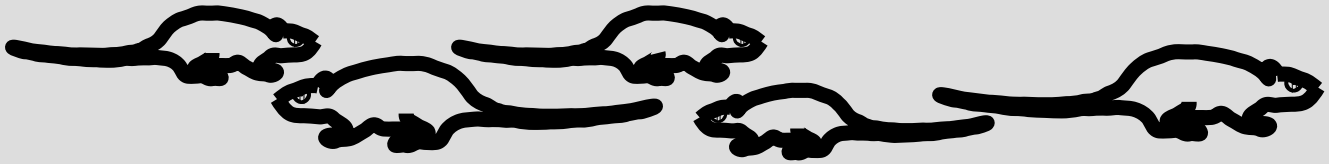


Constant variance

When do we assume that the underlying distribution of the response variable follows a **poisson** distribution?

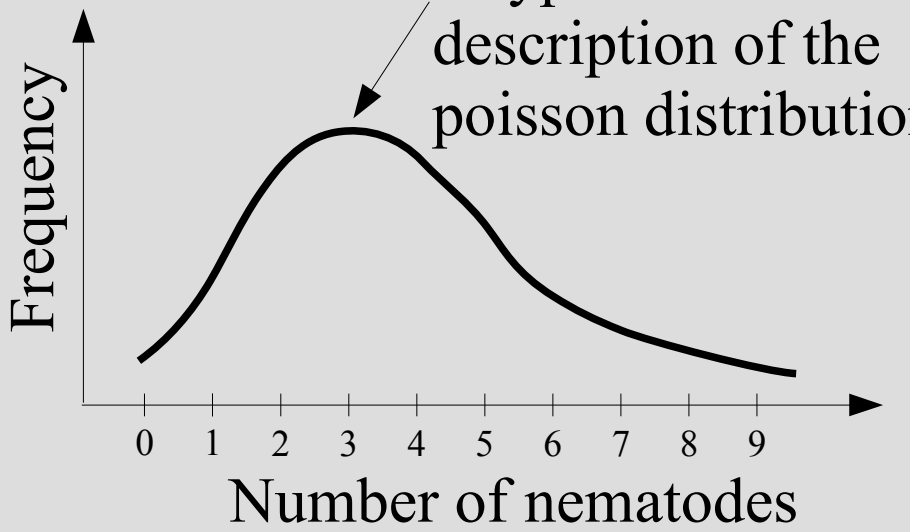
- Typically when the response variable represents count data.

Example: Counts of the number of nematodes in each individual rat:

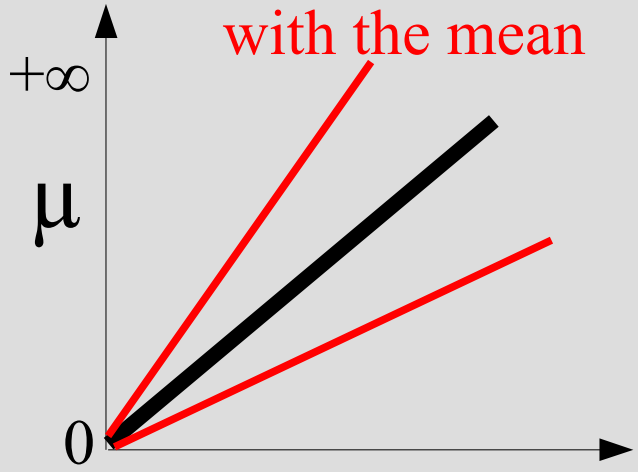


Siméon Denis Poisson (1781–1840)

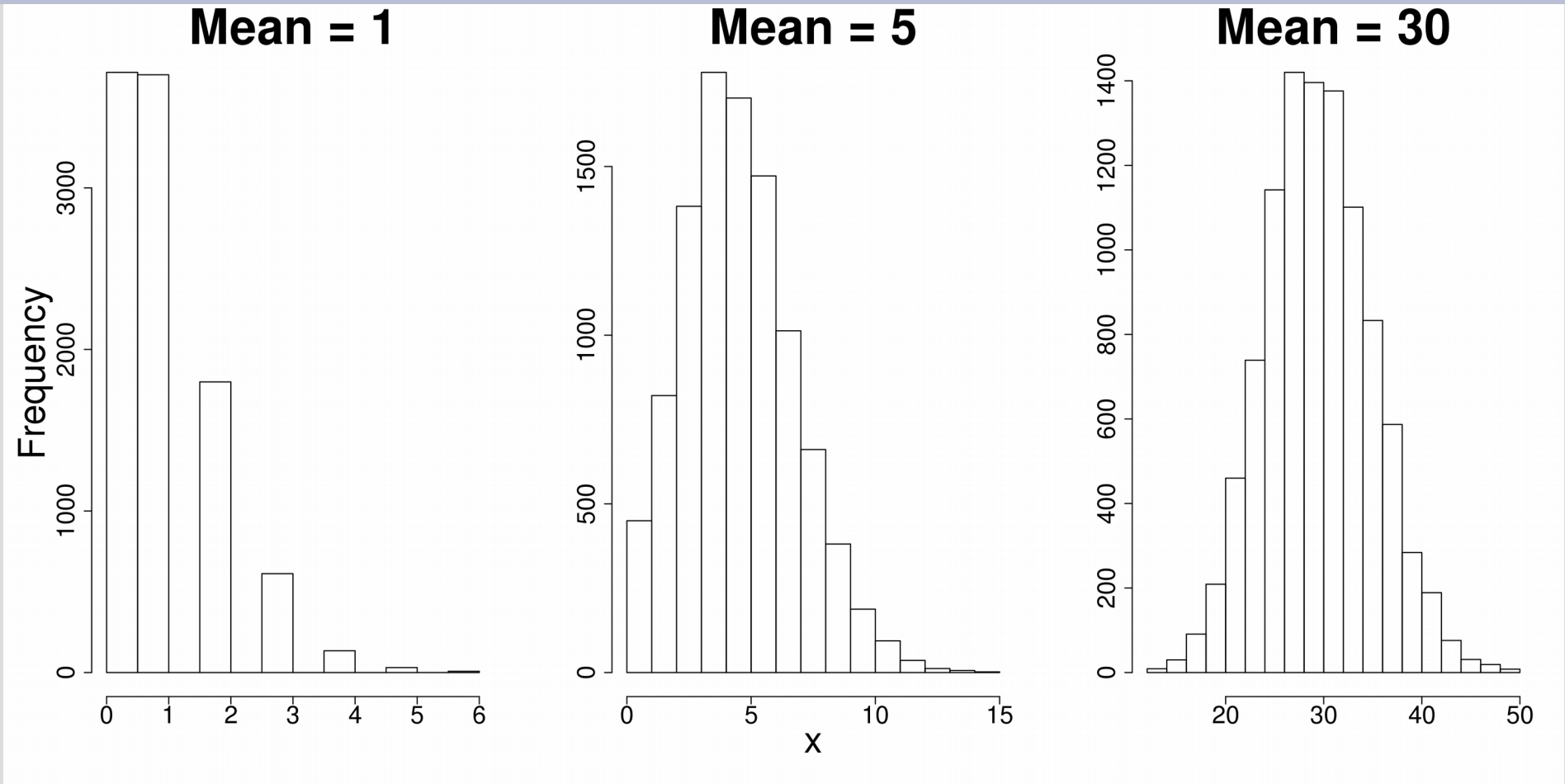
A typical text book description of the poisson distribution.



Variance increases with the mean



Note! For poisson distributed data, the shape varies with the mean



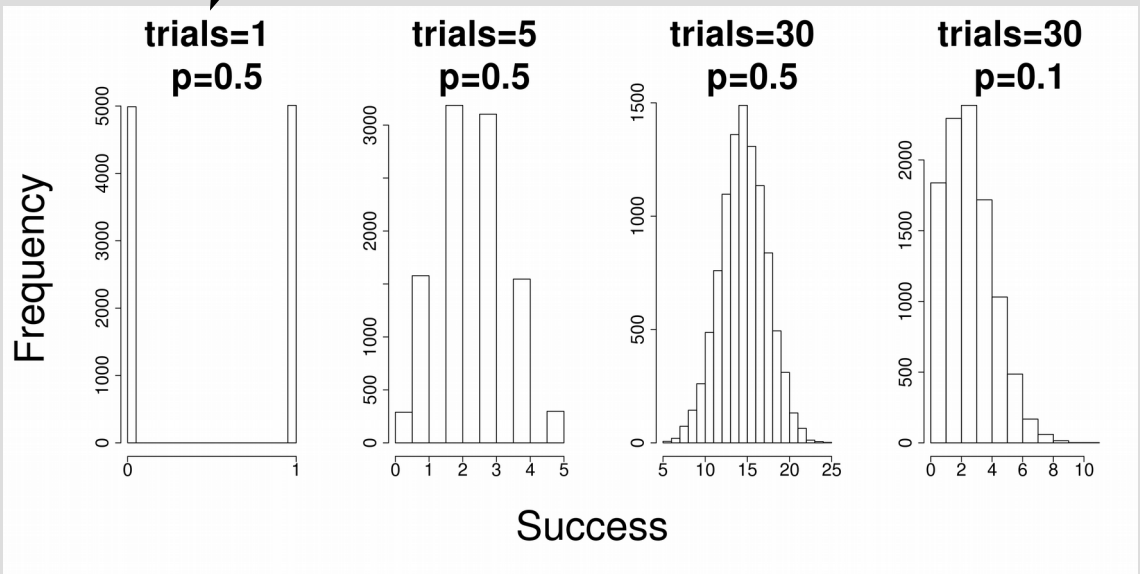
It may look normal at mean=30, but the main concern should be the variance which is non-constant for poisson distributed data, i.e. groups with different mean have different variance around the mean.

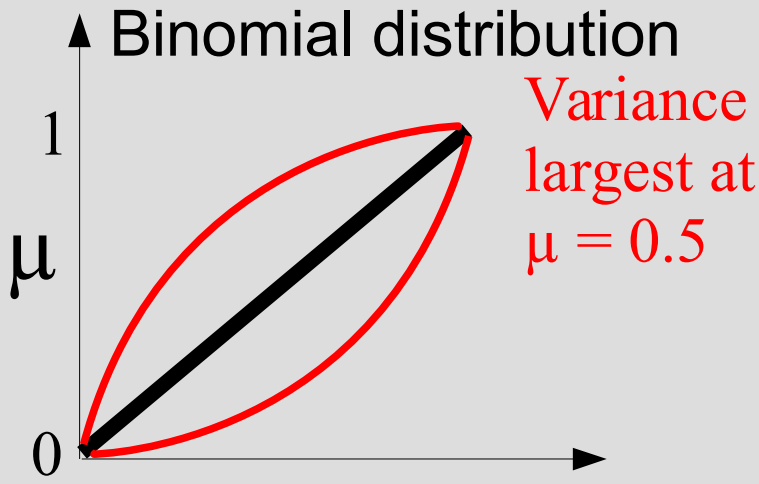
When do we assume that the underlying distribution of the response variable follows a **binomial** distribution?

- Typically when the response variable represents binary data like presence/absence or proportions.

Example: Checking the nematode infection status; infected (1) or not infected (0) for each individual rat.

Jacob Bernoulli (1654-1705)





3) What type of study design do you have?

You should care about this mainly to evaluate if you have some clustering in your data that creates some dependency between observations.

Ordinary least squares models (anova and linear regression) and generalized linear models (poisson regression and logistic regression) assumes observations to be independent.

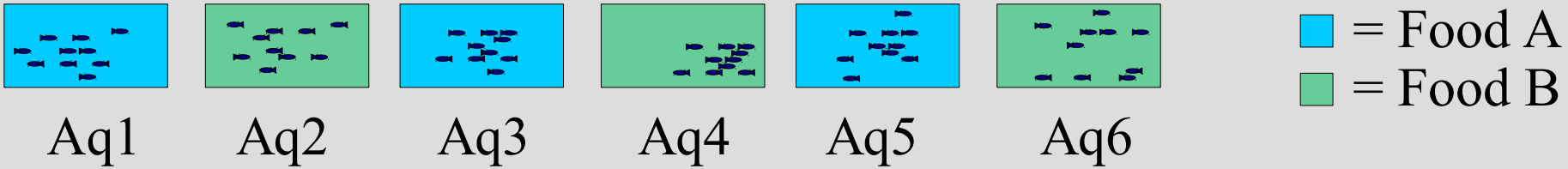
If you have some kind of clustering of your data, you should consider a type of mixed effects model instead (linear mixed-effects model or generalized linear mixed-effects model).

Explaining clustering

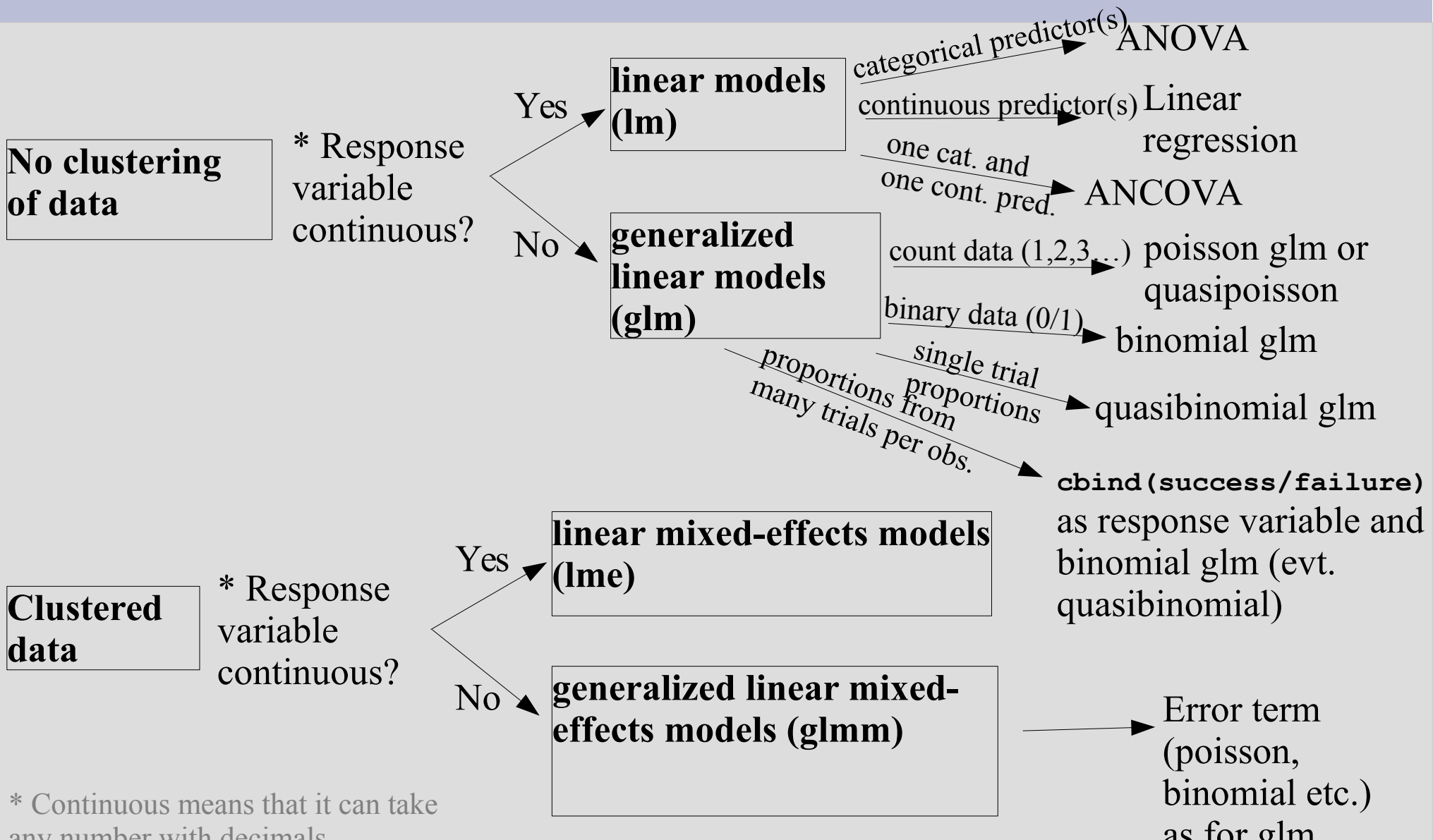
Clustered data, i.e. dependent observations.

Examples of clustered data is nested designs (the nesting factor is a cluster), repeated measurements (each individual is a cluster), line transects (each transect is a cluster) etc.

An example of a clustered design:
H_0: Food A and B gives the same fat content of *Salmo salar.*
Design: Six aquaria is set up with 10 fish in each. Half of the aquaria are given Food A and the other half are given Food B. Observations are clustered in aquaria. The 10 fish within an aquarium are dependent observations.



Statistical key for common models



* Continuous means that it can take any number with decimals.

Some examples with datasets: Performing tests with R

Two sample t-test:

H_0: The body mass in *Rattus norvegicus* is the same for males and females.

```
#Data import:
rats.df <-
read.table("http://folk.uib.no/nzlkj/data/rats.txt",
header=T, sep="", dec=".")

#Test:
t.test(Body.mass~Sex, data=rats.df)
```

Note! Never do any statistical tests before you get to know your data through plotting. Plotting is not done here since it is not the scope of this lecture.

One-way ANalysis Of VAriance (ANOVA):

H_0: The time spent solving a mathematical problem is the same for professional chess, squash and tennis players.

```
#Data import:
solvtime.df <-
read.table("http://folk.uib.no/nzlkj/data/solving.time.txt",
header=T, sep=",", dec=".")

#Model:
fit.lm <-lm(Time~Player, data=solvtime.df)

#Model outputs:
anova(fit.lm)
summary(fit.lm)
```

For one-way anova, given that the test came out significant, it is common to perform an unplanned multiple comparisons test to figure out which groups that differ.

Here is an example for the above model:

```
#Loading required library (need to be installed)
library(multcomp)

#Performing a Tukey HSD multiple comarisons test:
mc <- glht(fit.lm, linfct = mcp(Player="Tukey"),
data=solvtime.df)
summary(mc)
```

Note! With R, you can even use the glht function to perform multiple comparisons on lme, glm and glmm.

Simple linear regression:

H_0: The stopping distance of a car does not depend on its speed.

```
#Data import:
stop.df <-
read.table("http://folk.uib.no/nzlkj/data/stop.txt",
header=T, sep="", dec=".")

#Model:
fit2.lm <-lm(Stop.distance~Speed, data=stop.df)

#Model outputs:
summary(fit2.lm)
```

ANalysis of COVAriance (ANCOVA):

H_0: The effect of age on time needed to klimb a rope is the same for males and females.

```
#Data import:
klimb.df <-
read.table("http://folk.uib.no/nzlkj/data/klimb.txt",
header=T, sep="\t", dec=".")

#Model:
fit3.lm <- lm(Time~Sex*Age, data=klimb.df)

#Model outputs:
anova(fit3.lm)
summary(fit3.lm)
```

Generalized linear models for count data (poisson regression):

H_0: The amount of bacteria, measured as colony forming units (CFU), in Møllendal river is the same above and below pipeline A.

```
#Data import:
tcb.df <- read.table("http://folk.uib.no/nzlkj/data/TCB.txt",
header=T, sep=",", dec=".")

#Model:
fit4.glm <- glm(CFU~Site, family="poisson",
data=tcb.df)

#Model outputs:
anova(fit4.glm, test="Chi"); summary(fit4.glm)
```

Note! if overdispersed data or data with decimals due to division with volume of sample unit, then quasipoisson and test="F")

Generalized linear models for binary data (binary logistic regression):

H_0: The occurrence of a species does not depend on temperature.

```
#Data import:
occ.df <-
read.table("http://folk.uib.no/nzlkj/data/occur.txt",
header=T, sep=",", dec=".")

#Model:
fit5.glm <- glm(Occurrence~Temp, family="binomial",
data=occ.df)

#Model outputs:
anova(fit5.glm, test="Chi"); summary(fit5.glm)
```