

"How to" in **R**

Brice Ozenne

September 4, 2020

This document present ways to perform basic operations in **R**:

- importing data
- data management
- graphical displaying
- modeling
- loops and parallel computing
- generating data through simulation

Contents

1	Packages	5
2	Import/export data	6
2.1	Set the working directory	6
2.2	See which files are present in the current directory	7
2.3	Check that the file we want to import exists:	8
2.4	Display a file before importing it	9
2.5	Import a data from a file (.txt, .csv)	10
2.6	Import data from a specific format (e.g. excel files or outputs from SPSS/SAS)	11
2.7	Export data	12
2.8	Export table	13
2.9	Export graphs	14
3	Data management	15
3.1	Categorize age into groups	15
3.2	Convert list to array	16
3.3	Apply function for each element of a list	17
4	Data management using the <i>data.table</i> package	18
4.1	Introduction	18
4.2	Display a dataset	20
4.3	Extract row(s), i.e. all the variables relative to one or several observations	21
4.3.1	Extract row(s) using row numbers	21
4.3.2	Extract row(s) according to conditions	21
4.4	Extract column(s), i.e. all the observations relative to one or several variables	23
4.4.1	Extract column(s) using column numbers	23
4.4.2	Extract column(s) using column names	24
4.5	Work with categorical variables	25
4.5.1	Convert a numeric/character into a factor	25
4.5.2	Divide a continuous variable into categories	25
4.5.3	Redefine the levels of a factor variable	26
4.6	Extract simple features of a dataset	27
4.6.1	Number of rows and columns	27
4.6.2	Name of the columns	27
4.6.3	Type of the columns	27
4.6.4	Summary statistics by column	27
4.6.5	Number of missing values	28
4.6.6	Mean value of a column	28
4.6.7	Correlation between values of several columns	29
4.7	Performing operations on a group of rows	30
4.7.1	Computing the number of observations per subgroup	30
4.7.2	Computing the mean by subgroup	30
4.7.3	Computing the correlation matrix by subgroup	31
4.8	Sort a dataset according to one or several variables	33

4.9	Change the names of the column in a dataset	35
4.10	Converting a dataset from the wide format to the long format	36
4.10.1	Univariate melt	36
4.10.2	Multivariate melt	38
4.11	Converting a dataset from the long format to the wide format	39
4.11.1	Univariate	39
4.11.2	Multivariate	40
5	Graphical display	41
5.1	Descriptive plots	41
5.1.1	Spaguetti plot	41
5.1.2	Display the mean over time	45
5.1.3	Boxplot + points (non-overlapping)	46
5.2	Diagnostic plots	47
5.2.1	Histogram of the residuals	47
5.3	Customize graphic	48
5.3.1	Modify the legend of a discrete scale (with greek letters)	48
5.3.2	Change the name of the legend	49
5.3.3	Increase the font size	49
5.3.4	Increase size of the legend labels	50
5.3.5	Put the legend at the bottom	50
5.3.6	Number of lines in the legend	51
5.3.7	Default ggplot color palette	51
5.3.8	Color blind palette	51
5.3.9	Rotate x-axis labels	51
5.3.10	Change tick mark labels	52
5.3.11	Combine ggplots	52
5.3.12	Symbols in facet names	53
5.4	Path diagram	54
6	Modeling	55
6.1	Test proportions	55
6.2	Compare proportions between groups (with fixed margins!)	56
6.3	permutation t-test	57
6.4	Testing median	58
6.5	Testing linear hypotheses	60
6.5.1	Separate Wald tests of linear hypotheses	60
6.5.2	Confidence intervals associated with linear hypotheses	62
6.5.3	Joint test of linear hypotheses	62
7	Loops and parallel computations	64
7.1	Apply with progress bar	64
7.2	Parallel computation	65
7.2.1	Detect the number of cores	65
7.2.2	Start a cluster	65
7.2.3	Get the name of each core	65
7.2.4	Export element to cluster	65

7.2.5	Show progress bar (in console)	66
7.2.6	Show progress bar (external)	66
7.2.7	Stop a cluster	66
8	<i>lava</i> package	67
8.1	Generate repeated measurements	67
8.2	Generate data with heteroschadasticity	68
8.3	Generate survival time under non proportional hazard (non-PH)	69
8.4	Generate survival time with delayed treatment effect	71
8.5	Tune optimization parameters	73
9	Miscellaneous	74
9.1	Profile code	74
9.2	Debug	75
9.3	Find all function names from a package	76
9.4	Install development version of R	77
9.5	Install suggested packages	78
9.6	R version	79
9.7	CRAN check rocker	80

1 Packages

The following packages are necessary to run the code suggested in the document:

```
## importing data and data management
library(data.table)

## graphical display
library(ggplot2)
library(ggthemes)
library(abind) # convert list to array

## modeling
library(car)

library(prodlim) # survival analysis
library(survival) # survival analysis

## statistical inference
library(multcomp) # adjust for multiple comparisons
library(exactci) ## ci / p-values for proportions
library(exact2x2) ## compare proportions between groups
library(asht) ## test on the quantile
library(BuyseTest) ## wilcoxon-test with estimated effect size
library(perm) ## permutation tests
library(quantreg) ## quantile regression

## loops and parallel computing
library(pbapply)
library(doSNOW)
library(parallel)

## simulation
library(lava)
```

2 Import/export data

2.1 Set the working directory

The working directory is where **R** will, by default, look for files to import and export data or pictures. The current working directory can be accessed using:

```
getwd()
```

```
[1] "c:/Users/hpl802/AppData/Roaming/R"
```

It can be changed using the function `setwd()`:

```
path <- "c:/Users/hpl802/Documents/GitHub/bozenne.github.io/doc/howTo-R/"  
setwd(path)
```

We can check that the working directory has indeed changed calling again `getwd()`:

```
getwd()
```

```
[1] "c:/Users/hpl802/Documents/GitHub/bozenne.github.io/doc/howTo-R"
```

2.2 See which files are present in the current directory

List all files in the current directory:

```
list.files()
```

```
[1] "#howTo-R.org#"      "figures"      "howTo-R.aux"  "howTo-R.log"
[5] "howTo-R.org"        "howTo-R.org_archive" "howTo-R.pdf"  "howTo-R.tex"
[9] "howTo-R.toc"        "mydata.csv"    "mydata.txt"   "myplot.png"
[13] "Table1.docx"
```

There are many files. To list files in the current directory with a given extension, e.g. `.txt` use:

```
list.files(pattern = ".txt")
```

```
[1] "mydata.txt"
```

There is only one file with a `.txt` extension, it is called `mydata.txt`.

2.3 Check that the file we want to import exists:

Test whether the file exists:

```
file.exists("./mydata.txt")
```

```
[1] TRUE
```


2.4 Display a file before importing it

Display the first three lines of the file we want to import

```
readLines("./mydata.txt")[1:3]
```

```
[1] "Id Age Gender Treatment weight_t1 weight_t2 weight_t3 size_t1 size_t2 size_t3"  
[2] "1 40 Male Yes 50 57 56 50.67 55.88 61.69"  
[3] "2 38 Female No 52 57 63 50.26 55.73 60.37"
```

```
readLines("./mydata.csv")[1:3]
```

```
[1] "Id;Age;Gender;Treatment;weight_t1;weight_t2;weight_t3;size_t1;size_t2;size_t3"  
[2] "1;40;Male;Yes;50;57;56;50,67;55,88;61,69"  
[3] "2;38;Female;No;52;57;63;50,26;55,73;60,37"
```

2.5 Import a data from a file (.txt, .csv)

Import a file and store the dataset into a `data.frame` object:

```
dfW.data <- read.table("./mydata.txt", header = TRUE, na.strings = ".")
```

Import a file and store the dataset into a `data.table` object:

```
dtW.data <- fread("./mydata.txt", header = TRUE, na.strings = ".")
```

In both cases, the argument `na.strings` specifies which character(s) in the dataset stands for missing values. The argument `header=TRUE` indicates that the first line of the dataset contains the name of the columns of the dataset (and not the data of an observation). See `?read.table` or `?fread` for further explanations about the arguments of these functions.

Note: `"./"` stands for current directory, e.g. `"./mydata.txt"` abbreviated in `"mydata.txt"`
 `"../"` stands for parent directory, e.g. `"../mydata.txt"`
 `"/"` stands for root directory, e.g. `"/mydata.txt"`

2.6 Import data from a specific format (e.g. excel files or outputs from SPSS/SAS)

There are many packages that can be used to read excel files, e.g.:

- **readxl** package (no dependency): function `read_excel`, `read_xls`, or `read_xlsx`.
- **xlsx** package: function `read.xlsx`.
- **gdata** package: function `read.xls`.
- **XLConnect** package: function `readWorksheet`.

The **foreign** package enable to read a variety for files, e.g.:

- `read.spss`: read an spss data file.
- `read.ssd`: obtain a data frame from a sas permanent dataset, via `read.xport`.

To load `.rds` files use `readRDS` and to load `.rdata` files use `load`.

2.7 Export data

To export a `data.frame` to a file one can use:

- `write.csv` to export a `.csv` file
- `write.table` to export a `.txt` file
- `readxl::read_excel` to export a `.xlsx` file
- `data.table::fwrite`

```
fwrite(dtW.data, file = "./mydata.csv", sep = ";", dec = ",")  
fwrite(dtW.data, file = "./mydata.txt", sep = " ", dec = ".")
```

To export a single R object (can be anything) use `saveRDS`. To export several R object use `save`. To export the current workspace use `save.image`.

2.8 Export table

```
library(Publish)
myTable1 <- univariateTable(Treatment ~ Age + Gender + weight_t1, data = dtW.data)
```

Export to word:

```
library(officer)
myTable1.doc <- body_add_table(x = read_docx(),
                              value = summary(myTable1))
print(myTable1.doc, target = "./Table1.docx")
```

```
[1] "c:/Users/hpl802/Documents/GitHub/bozenne.github.io/doc/howTo-R/Table1.docx"
```

2.9 Export graphs

The functions `pdf`, `png`, `postscript`, `svg`, `tiff` enables a graph to export to `.pdf`, `.png`, `.eps`, `.svg`, or `.tiff` file:

```
png("myplot.png")
plot(1:10)
dev.off()
```

```
null device
1
```

```
file.exists("myplot.png")
```

```
[1] TRUE
```

For exporting graph generated by **ggplot2**, use `ggsave`.

3 Data management

3.1 Categorize age into groups

```
vec <- dfW.data$weight_t3  
vec
```

```
[1] 56 63 62 60 64 65 66 63 59 64 59 58 63 64 61 64 67 54 57 65 63 60 60 57 66 65 60 53 57 58 58  
[32] 58 59 63 64 58 64 58 59 59 60 59 57 62 61 63 63 63 65 55 59 65 71 64 62 62 64 58 61 61 65 64  
[63] 66 60 58 60 63 57 58 68 59 60 54 61 60 63 61 60 62 61 59 59 65 62 66 58 64 66 62 65 59 63 57  
[94] 62 64 59 63 57 62 59 55 68
```

```
cut(vec, breaks = seq(0,100,5))
```

```
[1] (55,60] (60,65] (60,65] (55,60] (60,65] (60,65] (65,70] (60,65] (55,60] (60,65] (55,60]  
[12] (55,60] (60,65] (60,65] (60,65] (60,65] (65,70] (50,55] (55,60] (60,65] (60,65] (55,60]  
[23] (55,60] (55,60] (65,70] (60,65] (55,60] (50,55] (55,60] (55,60] (55,60] (55,60] (55,60]  
[34] (60,65] (60,65] (55,60] (60,65] (55,60] (55,60] (55,60] (55,60] (55,60] (55,60] (60,65]  
[45] (60,65] (60,65] (60,65] (60,65] (60,65] (50,55] (55,60] (60,65] (70,75] (60,65] (60,65]  
[56] (60,65] (60,65] (55,60] (60,65] (60,65] (60,65] (60,65] (65,70] (55,60] (55,60] (55,60]  
[67] (60,65] (55,60] (55,60] (65,70] (55,60] (55,60] (50,55] (60,65] (55,60] (60,65] (60,65]  
[78] (55,60] (60,65] (60,65] (55,60] (55,60] (60,65] (60,65] (65,70] (55,60] (60,65] (65,70]  
[89] (60,65] (60,65] (55,60] (60,65] (55,60] (60,65] (60,65] (55,60] (60,65] (55,60] (60,65]  
[100] (55,60] (50,55] (65,70]  
20 Levels: (0,5] (5,10] (10,15] (15,20] (20,25] (25,30] (30,35] (35,40] (40,45] (45,50] ... (95,100]
```

3.2 Convert list to array

```
ll <- list(matrix(1,2,2),  
           matrix(3,2,2),  
           matrix(9,2,2))  
do.call(abind, c(ll, list(along = 3)))
```

, , 1

	[,1]	[,2]
[1,]	1	1
[2,]	1	1

, , 2

	[,1]	[,2]
[1,]	3	3
[2,]	3	3

, , 3

	[,1]	[,2]
[1,]	9	9
[2,]	9	9

3.3 Apply function for each element of a list

```
l1 <- list(matrix(1,2,2),  
           matrix(3,2,2),  
           matrix(9,2,2))  
apply(do.call(abind, c(l1, list(along = 3))), 1:2, median)
```

```
      [,1] [,2]  
[1,]    3    3  
[2,]    3    3
```

4 Data management using the *data.table* package

4.1 Introduction

In **R**, data are usually stored in `data.frame` object since compared to matrices, it enables to store in a same object different types of variables (e.g. numeric, categorical, ...). Data management can be performed using the core R function, e.g. using `for` loops or the `apply`, `tapply`, `lapply` functions. However this approach will most often requires many lines of code to get the expected transformation. A faster and safer approach is to functions/packages suited to the structure of longitudinal data.

We present here how to use the *data.table* package to perform the most common operations in data management. The main benefit of using this package are:

- a concise and consistant syntax for performing the most common operations in data management.
- fast and memory efficient implementation (i.e. able to deal with dataset with millions of lines).
- share common features with the SQL terminology.

A concise summary of the features can be found at: <https://s3.amazonaws.com/assets.datacamp.com/img/blog/data+table+cheat+sheet.pdf>

Additional documentation can be found:

- in the documentation of the function `data.table`: type `?data.table` in **R**.
- on the webpage of the package: <https://github.com/Rdatatable/data.table/wiki>.
- in the vignettes of the package: <https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html>.

Note: the **wide format** denote a format where each line corresponds to a different individuals. Repeated measurements of the same quantity (e.g. weight) for a given individual are stored in different columns (e.g. `weight_t1`, `weight_t2`).

```
head(dtW.data)
```

	Id	Age	Gender	Treatment	weight_t1	weight_t2	weight_t3	size_t1	size_t2	size_t3
1:	1	40	Male	Yes	50	57	56	50.67	55.88	61.69
2:	2	38	Female	No	52	57	63	50.26	55.73	60.37
3:	3	41	Male	No	47	54	62	46.61	50.89	56.52
4:	4	41	Female	Yes	48	55	60	45.95	53.10	59.82
5:	5	42	Female	Yes	52	56	64	52.86	58.41	63.79
6:	6	38	Male	Yes	52	59	65	49.37	57.91	64.45

The **long** format denote a format where the same individual may appear on different lines but a given quantity is only stored in one column. In case of repeated measurement, an additional column encodes at which repetition the measurement was obtained (e.g. **time**):

```
head(dtL.data)
```

	Id	Gender	Treatment	Age	time	weight	size
1:	1	Male	Yes	40	1	50	50.67
2:	2	Female	No	38	1	52	50.26
3:	3	Male	No	41	1	47	46.61
4:	4	Female	Yes	41	1	48	45.95
5:	5	Female	Yes	42	1	52	52.86
6:	6	Male	Yes	38	1	52	49.37

4.2 Display a dataset

Using the print method:

```
print(dtW.data) # equivalent to just dtW.data
```

```
      Id Age Gender Treatment weight_t1 weight_t2 weight_t3 size_t1 size_t2 size_t3
1:     1  40   Male      Yes        50        57        56  50.67  55.88  61.69
2:     2  38  Female      No         52        57        63  50.26  55.73  60.37
3:     3  41   Male      No         47        54        62  46.61  50.89  56.52
4:     4  41  Female      Yes         48        55        60  45.95  53.10  59.82
5:     5  42  Female      Yes         52        56        64  52.86  58.41  63.79
---
98:   98  39   Male      No         53        59        57  49.51  53.80  61.13
99:   99  42  Female      Yes         51        57        62  47.60  56.55  59.47
100: 100  40  Female      No         53        55        59  50.06  54.90  61.89
101: 101  38  Female      No         48        58        55  49.51  54.01  62.32
102: 102  39  Female      No         52        58        68  47.35  56.08  59.49
```

To print more lines use the argument `topn`:

```
print(dtW.data, topn = 6)
```

```
      Id Age Gender Treatment weight_t1 weight_t2 weight_t3 size_t1 size_t2 size_t3
1:     1  40   Male      Yes        50        57        56  50.67  55.88  61.69
2:     2  38  Female      No         52        57        63  50.26  55.73  60.37
3:     3  41   Male      No         47        54        62  46.61  50.89  56.52
4:     4  41  Female      Yes         48        55        60  45.95  53.10  59.82
5:     5  42  Female      Yes         52        56        64  52.86  58.41  63.79
6:     6  38   Male      Yes         52        59        65  49.37  57.91  64.45
---
97:   97  39   Male      No         50        60        63  51.72  57.86  61.06
98:   98  39   Male      No         53        59        57  49.51  53.80  61.13
99:   99  42  Female      Yes         51        57        62  47.60  56.55  59.47
100: 100  40  Female      No         53        55        59  50.06  54.90  61.89
101: 101  38  Female      No         48        58        55  49.51  54.01  62.32
102: 102  39  Female      No         52        58        68  47.35  56.08  59.49
```

4.3 Extract row(s), i.e. all the variables relative to one or several observations

4.3.1 Extract row(s) using row numbers

Extract the third line:

```
dtW.data[3]
```

```
Id Age Gender Treatment weight_t1 weight_t2 weight_t3 size_t1 size_t2 size_t3
1:  3  41   Male       No         47         54         62  46.61  50.89  56.52
```

Extract line one to four:

```
dtW.data[1:4]
```

```
Id Age Gender Treatment weight_t1 weight_t2 weight_t3 size_t1 size_t2 size_t3
1:  1  40   Male       Yes         50         57         56  50.67  55.88  61.69
2:  2  38 Female       No         52         57         63  50.26  55.73  60.37
3:  3  41   Male       No         47         54         62  46.61  50.89  56.52
4:  4  41 Female       Yes         48         55         60  45.95  53.10  59.82
```

Extract line one, three, and five:

```
dtW.data[c(1,3,5)]
```

```
Id Age Gender Treatment weight_t1 weight_t2 weight_t3 size_t1 size_t2 size_t3
1:  1  40   Male       Yes         50         57         56  50.67  55.88  61.69
2:  3  41   Male       No         47         54         62  46.61  50.89  56.52
3:  5  42 Female       Yes         52         56         64  52.86  58.41  63.79
```

4.3.2 Extract row(s) according to conditions

Extract lines corresponding to the observations with Id equals to 1:

```
dtW.data[Id == 1]
```

```
Id Age Gender Treatment weight_t1 weight_t2 weight_t3 size_t1 size_t2 size_t3
1:  1  40   Male       Yes         50         57         56  50.67  55.88  61.69
```

Extract lines corresponding to the males:

```
newdata <- dtW.data[Gender == "Male"]
head(newdata)
```

	Id	Age	Gender	Treatment	weight_t1	weight_t2	weight_t3	size_t1	size_t2	size_t3
1:	1	40	Male	Yes	50	57	56	50.67	55.88	61.69
2:	3	41	Male	No	47	54	62	46.61	50.89	56.52
3:	6	38	Male	Yes	52	59	65	49.37	57.91	64.45
4:	9	42	Male	Yes	46	52	59	49.53	52.84	60.54
5:	11	42	Male	No	55	58	59	50.03	55.09	60.94
6:	12	41	Male	Yes	50	52	58	48.66	52.73	55.86

Extract lines corresponding to the males whose age is inferior or equal to 38:

```
dtW.data[Gender == "Male" & Age <= 38]
```

	Id	Age	Gender	Treatment	weight_t1	weight_t2	weight_t3	size_t1	size_t2	size_t3
1:	6	38	Male	Yes	52	59	65	49.37	57.91	64.45
2:	41	37	Male	No	53	55	60	47.59	53.75	57.00
3:	76	38	Male	No	53	57	63	48.10	54.82	55.29
4:	91	38	Male	No	51	55	59	52.05	57.01	59.53

Extract lines corresponding to observations where Age is inferior or equal to 37, or greater or equal to 43 :

```
dtW.data[Age <= 37 | Age >= 43]
```

	Id	Age	Gender	Treatment	weight_t1	weight_t2	weight_t3	size_t1	size_t2	size_t3
1:	10	43	Female	Yes	52	57	64	53.22	57.25	62.94
2:	41	37	Male	No	53	55	60	47.59	53.75	57.00
3:	45	43	Female	Yes	48	51	61	49.88	54.41	56.18
4:	73	43	Male	Yes	46	53	54	48.44	52.74	60.93

4.4 Extract column(s), i.e. all the observations relative to one or several variables

4.4.1 Extract column(s) using column numbers

Extract the third column:

```
dtW.data[, 3, with = FALSE]
```

```
      Gender
1:   Male
2: Female
3:   Male
4: Female
5: Female
---
98:   Male
99: Female
100: Female
101: Female
102: Female
```

Alternatively:

```
dtW.data[[3]]
```

```
[1] "Male"  "Female" "Male"  "Female" "Female" "Male"  "Female" "Female" "Male"  "Female"
[11] "Male"  "Male"  "Female" "Female" "Female" "Female" "Female" "Female" "Male"  "Female"
[21] "Male"  "Male"  "Female" "Male"  "Female" "Male"  "Male"  "Male"  "Female" "Female"
[31] "Male"  "Male"  "Male"  "Male"  "Female" "Female" "Female" "Female" "Male"  "Male"
[41] "Male"  "Female" "Female" "Female" "Female" "Female" "Female" "Female" "Male"  "Male"
[51] "Female" "Male"  "Male"  "Male"  "Female" "Female" "Male"  "Male"  "Female" "Male"
[61] "Female" "Male"  "Male"  "Male"  "Female" "Male"  "Female" "Male"  "Male"  "Male"
[71] "Female" "Female" "Male"  "Female" "Female" "Male"  "Female" "Female" "Female" "Female"
[81] "Male"  "Male"  "Female" "Female" "Male"  "Female" "Female" "Female" "Female" "Female"
[91] "Male"  "Male"  "Male"  "Male"  "Male"  "Male"  "Male"  "Male"  "Female" "Female"
[101] "Female" "Female"
```

Extract column one, three, and five:

```
dtW.data[, c(1,3,5), with = FALSE]
```

```
      Id Gender weight_t1
1:    1  Male        50
2:    2 Female        52
3:    3  Male        47
4:    4 Female        48
5:    5 Female        52
---
98:   98  Male        53
```

```

99: 99 Female      51
100: 100 Female    53
101: 101 Female    48
102: 102 Female    52

```

4.4.2 Extract column(s) using column names

Extract one column, e.g. Id:

```
dtW.data[, Id] # similar to dtW.data[, "Id", with=FALSE]
```

```

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
[47] 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
[70] 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92
[93] 93 94 95 96 97 98 99 100 101 102

```

Extract several columns, e.g. Id and Age:

```

dtW.data[, .(Id, Age)]
# similar to dtW.data[, c("Id", "Age"), with = FALSE]
# similar to dtW.data[, .SD, .SDcols = c("Id", "Age")]

```

```

      Id Age
1:    1  40
2:    2  38
3:    3  41
4:    4  41
5:    5  42
---
98:  98  39
99:  99  42
100: 100  40
101: 101  38
102: 102  39

```


4.5 Work with categorical variables

4.5.1 Convert a numeric/character into a factor

```
class(dtW.data[,Gender])
```

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

```
dtW.data[, Gender := as.factor(Gender)]  
class(dtW.data[,Gender])
```

```
[1] "factor"
```

```
class(dtW.data[,Id])
```

```
[1] "integer"
```

```
dtW.data[, Id := as.factor(Id)]  
class(dtW.data[,Id])
```

```
[1] "factor"
```

4.5.2 Divide a continuous variable into categories

```
dtW.data[, AgeCategory := cut(Age, breaks = c(0,38,40,42,100))]  
dtW.data[,.(Age, AgeCategory)]
```

```
   Age AgeCategory  
1:  40    (38,40]  
2:  38     (0,38]  
3:  41    (40,42]  
4:  41    (40,42]  
5:  42    (40,42]  
---  
98:  39    (38,40]  
99:  42    (40,42]  
100: 40    (38,40]  
101: 38     (0,38]  
102: 39    (38,40]
```

Alternatively:

```
dtW.data[, AgeCategory0 := findInterval(Age, vec = c(0,38,40,42,100))]  
dtW.data[,.(Age, AgeCategory0)]
```

	Age	AgeCategory0
1:	40	3
2:	38	2
3:	41	3
4:	41	3
5:	42	4

98:	39	2
99:	42	4
100:	40	3
101:	38	2
102:	39	2

The arguments `rightmost` and `left.open` can be used to decide what to do with the values equaling the breaks (i.e. one of the value of the argument `vec`). But it is often easier to modify `vec` such that no value equals the breaks, e.g. using `c(0,38,40,42,100)-1e12`.

4.5.3 Redefine the levels of a factor variable

```
dtW.data[,AgeCategory0 := factor(AgeCategory0,
                                levels = 1:4,
                                labels = c("[0-37)", "[38-39)", "[40-41)", "[42-100)"))]
dtW.data[,.(Age, AgeCategory0, AgeCategory)]
```

	Age	AgeCategory0	AgeCategory
1:	40	[40-41)	(38,40]
2:	38	[38-39)	(0,38]
3:	41	[40-41)	(40,42]
4:	41	[40-41)	(40,42]
5:	42	[42-100)	(40,42]

98:	39	[38-39)	(38,40]
99:	42	[42-100)	(40,42]
100:	40	[40-41)	(38,40]
101:	38	[38-39)	(0,38]
102:	39	[38-39)	(38,40]

4.6 Extract simple features of a dataset

4.6.1 Number of rows and columns

```
dim(dtW.data)
```

```
[1] 102 12
```

The dataset has 102 rows and 7 columns.

4.6.2 Name of the columns

```
names(dtW.data)
```

```
[1] "Id"           "Age"          "Gender"        "Treatment"     "weight_t1"     "weight_t2"
[7] "weight_t3"    "size_t1"      "size_t2"      "size_t3"      "AgeCategory"   "AgeCategory0"
```

4.6.3 Type of the columns

```
str(dtW.data)
```

```
Classes 'data.table' and 'data.frame':      102 obs. of  12 variables:
 $ Id          : Factor w/ 102 levels "1","2","3","4",...: 1 2 3 4 5 6 7 8 9 10 ...
 $ Age         : num  40 38 41 41 42 38 42 40 42 43 ...
 $ Gender      : Factor w/ 2 levels "Female","Male": 2 1 2 1 1 2 1 1 2 1 ...
 $ Treatment   : chr   "Yes" "No" "No" "Yes" ...
 $ weight_t1   : num   50 52 47 48 52 52 52 51 46 52 ...
 $ weight_t2   : int    57 57 54 55 56 59 63 52 52 57 ...
 $ weight_t3   : int    56 63 62 60 64 65 66 63 59 64 ...
 $ size_t1     : num   50.7 50.3 46.6 46 52.9 ...
 $ size_t2     : num   55.9 55.7 50.9 53.1 58.4 ...
 $ size_t3     : num   61.7 60.4 56.5 59.8 63.8 ...
 $ AgeCategory : Factor w/ 4 levels "(0,38)","(38,40)",...: 2 1 3 3 3 1 3 2 3 4 ...
 $ AgeCategory0: Factor w/ 4 levels "[0-37)","[38-39)",...: 3 2 3 3 4 2 4 3 4 4 ...
 - attr(*, ".internal.selfref")=<externalptr>
 - attr(*, "index")= int
```

The column **Gender** contains a factor variable with two levels "Yes" and "No".

The column **Id** contains integers while the columns **weight_t3** contains numeric numbers.

4.6.4 Summary statistics by column

```
summary(dtW.data)
```

The column **Gender** contains 48 **Male** and 54 **Female**. The median value of **Age** is 40.

Total number

[1] 0

Number of missing values by variable:

Id	Age	Gender	Treatment	weight_t1	weight_t2	weight_t3
0	0	0	0	0	0	0
size_t1	size_t2	size_t3	AgeCategory	AgeCategory0		
0	0	0	0	0		

Number of missing values by observation:

[illegible]

First extract the values from a column:

28

Then compute the mean:

```
mean(vec.tempo)
```

```
[1] 40.26471
```

Alternatively:

```
dtW.data[,mean(Age)]
```

```
[1] 40.26471
```

4.6.7 Correlation between values of several columns

First extract the columns:

```
dt.tempo <- dtW.data[,.(weight_t1,weight_t2,weight_t3)]
```

Then compute the correlation:

```
cor(dt.tempo)
```

```
      weight_t1 weight_t2 weight_t3
weight_t1 1.0000000 0.1882809 0.3179175
weight_t2 0.1882809 1.0000000 0.2374259
weight_t3 0.3179175 0.2374259 1.0000000
```

Alternatively:

```
dtW.data[,cor(cbind(weight_t1,weight_t2,weight_t3))]
```

```
      weight_t1 weight_t2 weight_t3
weight_t1 1.0000000 0.1882809 0.3179175
weight_t2 0.1882809 1.0000000 0.2374259
weight_t3 0.3179175 0.2374259 1.0000000
```

4.7 Performing operations on a group of rows

4.7.1 Computing the number of observations per subgroup

Compute the number of observation per gender:

```
dtW.data[, .N, by = "Gender"]
```

```
Gender  N
1:  Male 48
2: Female 54
```

Alternatively:

```
dtW.data[, NROW(.SD), by = "Gender"]
```

```
Gender V1
1:  Male 48
2: Female 54
```

4.7.2 Computing the mean by subgroup

Compute the mean weight at time 1 by gender:

```
dtW.data[, mean(weight_t1), by = "Gender"]
```

```
Gender      V1
1:  Male 50.45833
2: Female 51.24074
```

Alternative display:

```
dtW.data[, .(mean = mean(weight_t1)), by = "Gender"]
```

```
Gender      mean
1:  Male 50.45833
2: Female 51.24074
```

Compute the mean weight at time 1 to 3 by gender:

```
dtW.data[, .(mean_t1 = mean(weight_t1),
              mean_t2 = mean(weight_t2),
              mean_t3 = mean(weight_t3)),
           by = "Gender"]
```

```
Gender mean_t1 mean_t2 mean_t3
1:  Male 50.45833 55.81250 60.64583
2: Female 51.24074 56.72222 61.68519
```

Compute the mean weight at time 1 to 3 by gender and treatment group:

```
dtW.data[, .(mean_t1 = mean(weight_t1),
             mean_t2 = mean(weight_t2),
             mean_t3 = mean(weight_t3)),
           by = c("Gender", "Treatment")]
```

```
Gender Treatment mean_t1 mean_t2 mean_t3
1:   Male       Yes 50.42857 55.09524 60.23810
2: Female       No  51.65517 56.93103 61.75862
3:   Male       No  50.48148 56.37037 60.96296
4: Female       Yes 50.76000 56.48000 61.60000
```

4.7.3 Computing the correlation matrix by subgroup

We create a matrix containing the variables of interest, compute the correlation matrix and print it.

```
null.result <- dtW.data[, print(cor(cbind(weight_t1, weight_t2, weight_t3))),
                          by = "Gender"]
```

```
           weight_t1 weight_t2 weight_t3
weight_t1 1.0000000 0.2867753 0.2886667
weight_t2 0.2867753 1.0000000 0.2740567
weight_t3 0.2886667 0.2740567 1.0000000
           weight_t1 weight_t2 weight_t3
weight_t1 1.00000000 0.03214955 0.3148578
weight_t2 0.03214955 1.00000000 0.1551156
weight_t3 0.31485784 0.15511561 1.0000000
```

If we want to store the correlation matrix we need to wrap it into `.()` to keep the matrix format:

```
result <- dtW.data[, .(cor = .(cor(cbind(weight_t1, weight_t2, weight_t3)))),
                    by = "Gender"]
result[, cor]
```

```
[[1]]
           weight_t1 weight_t2 weight_t3
weight_t1 1.0000000 0.2867753 0.2886667
weight_t2 0.2867753 1.0000000 0.2740567
weight_t3 0.2886667 0.2740567 1.0000000

[[2]]
           weight_t1 weight_t2 weight_t3
weight_t1 1.00000000 0.03214955 0.3148578
weight_t2 0.03214955 1.00000000 0.1551156
weight_t3 0.31485784 0.15511561 1.0000000
```

Alternatively:

```
null.result <- dtW.data[, print(cor(.SD)),  
                           .SDcols = c("weight_t1","weight_t2","weight_t3"),  
                           by = "Gender"]
```

```
      weight_t1 weight_t2 weight_t3  
weight_t1 1.0000000 0.2867753 0.2886667  
weight_t2 0.2867753 1.0000000 0.2740567  
weight_t3 0.2886667 0.2740567 1.0000000  
      weight_t1 weight_t2 weight_t3  
weight_t1 1.00000000 0.03214955 0.3148578  
weight_t2 0.03214955 1.00000000 0.1551156  
weight_t3 0.31485784 0.15511561 1.0000000
```


4.8 Sort a dataset according to one or several variables

Sort the dataset according to Age:

```
setkeyv(dtW.data, c("Age"))
dtW.data
```

	Id	Age	Gender	Treatment	weight_t1	weight_t2	weight_t3	size_t1	size_t2	size_t3	AgeCategory
1:	41	37	Male	No	53	55	60	47.59	53.75	57.00	(0,38]
2:	2	38	Female	No	52	57	63	50.26	55.73	60.37	(0,38]
3:	6	38	Male	Yes	52	59	65	49.37	57.91	64.45	(0,38]
4:	46	38	Female	No	53	57	63	49.27	61.45	66.59	(0,38]
5:	48	38	Female	No	52	57	63	54.27	57.71	65.63	(0,38]

98:	95	42	Male	Yes	51	55	64	51.05	56.48	60.30	(40,42]
99:	99	42	Female	Yes	51	57	62	47.60	56.55	59.47	(40,42]
100:	10	43	Female	Yes	52	57	64	53.22	57.25	62.94	(42,100]
101:	45	43	Female	Yes	48	51	61	49.88	54.41	56.18	(42,100]
102:	73	43	Male	Yes	46	53	54	48.44	52.74	60.93	(42,100]
AgeCategory0											
1:	[0-37)										
2:	[38-39)										
3:	[38-39)										
4:	[38-39)										
5:	[38-39)										

98:	[42-100)										
99:	[42-100)										
100:	[42-100)										
101:	[42-100)										
102:	[42-100)										

Sort the dataset according to Age and then weight_t1:

```
setkeyv(dtW.data, cols = c("Age", "weight_t1"))
dtW.data
```

	Id	Age	Gender	Treatment	weight_t1	weight_t2	weight_t3	size_t1	size_t2	size_t3	AgeCategory
1:	41	37	Male	No	53	55	60	47.59	53.75	57.00	(0,38]
2:	101	38	Female	No	48	58	55	49.51	54.01	62.32	(0,38]
3:	59	38	Female	Yes	49	60	61	51.08	53.77	60.75	(0,38]
4:	91	38	Male	No	51	55	59	52.05	57.01	59.53	(0,38]
5:	2	38	Female	No	52	57	63	50.26	55.73	60.37	(0,38]

98:	11	42	Male	No	55	58	59	50.03	55.09	60.94	(40,42]
99:	54	42	Male	Yes	57	60	64	58.75	57.57	63.98	(40,42]
100:	73	43	Male	Yes	46	53	54	48.44	52.74	60.93	(42,100]
101:	45	43	Female	Yes	48	51	61	49.88	54.41	56.18	(42,100]
102:	10	43	Female	Yes	52	57	64	53.22	57.25	62.94	(42,100]
AgeCategory0											

1:	[0-37)
2:	[38-39)
3:	[38-39)
4:	[38-39)
5:	[38-39)

98:	[42-100)
99:	[42-100)
100:	[42-100)
101:	[42-100)
102:	[42-100)

4.9 Change the names of the column in a dataset

Use a small dataset

```
dt.simple <- dtW.data[,.(Age,Gender,Id,Treatment)]  
head(dt.simple)
```

	Age	Gender	Id	Treatment
1:	37	Male	41	No
2:	38	Female	101	No
3:	38	Female	59	Yes
4:	38	Male	91	No
5:	38	Female	2	No
6:	38	Male	6	Yes

Change all names:

```
setnames(dt.simple, c("AgeXX","GenderYY","IdZZ","Treat"))  
head(dt.simple)
```

	AgeXX	GenderYY	IdZZ	Treat
1:	37	Male	41	No
2:	38	Female	101	No
3:	38	Female	59	Yes
4:	38	Male	91	No
5:	38	Female	2	No
6:	38	Male	6	Yes

Change one or several names (less memory efficient):

```
names(dt.simple)[1:2] <- c("Age","Gender")  
head(dt.simple)
```

	Age	Gender	IdZZ	Treat
1:	37	Male	41	No
2:	38	Female	101	No
3:	38	Female	59	Yes
4:	38	Male	91	No
5:	38	Female	2	No
6:	38	Male	6	Yes

4.10 Converting a dataset from the wide format to the long format

4.10.1 Univariate melt

Data in the wide format:

```
head(dtW.data)
```

	Id	Age	Gender	Treatment	weight_t1	weight_t2	weight_t3	size_t1	size_t2	size_t3	AgeCategory
1:	41	37	Male	No	53	55	60	47.59	53.75	57.00	(0,38]
2:	101	38	Female	No	48	58	55	49.51	54.01	62.32	(0,38]
3:	59	38	Female	Yes	49	60	61	51.08	53.77	60.75	(0,38]
4:	91	38	Male	No	51	55	59	52.05	57.01	59.53	(0,38]
5:	2	38	Female	No	52	57	63	50.26	55.73	60.37	(0,38]
6:	6	38	Male	Yes	52	59	65	49.37	57.91	64.45	(0,38]

AgeCategory0

1:	[0-37)
2:	[38-39)
3:	[38-39)
4:	[38-39)
5:	[38-39)
6:	[38-39)

The conversion can be done naming explicitly the columns or using **patterns**:

```
dtL.data <- melt(dtW.data, id.vars = c("Id", "Gender", "Treatment", "Age"),
  measure=c("weight_t1", "weight_t2", "weight_t3"),
  variable.name = "time", value.name = "weight")

dtL.data.bis <- melt(dtW.data, id.vars = c("Id", "Gender", "Treatment", "Age"),
  measure=patterns("weight_t"),
  variable.name = "time", value.name = "weight")

identical(dtL.data, dtL.data.bis)
```

Warning message:

```
In melt.data.table(dtW.data, id.vars = c("Id", "Gender", "Treatment",
'measure.vars' [weight_t1, weight_t2, weight_t3] are not all of the same type. By order of hierarchy, the mol
Warning message:
In melt.data.table(dtW.data, id.vars = c("Id", "Gender", "Treatment",
'measure.vars' [weight_t1, weight_t2, weight_t3] are not all of the same type. By order of hierarchy, the mol
[1] TRUE
```

Arguments (see ?melt.data.table for more details):

- **id.vars**: name of the column(s) that are kept constant over the repetitions
- **measure.vars**: name of the columns to be melted in a single one (i.e. repeated measurements).

Data in the long format:

```
head(dtL.data)
```

	Id	Gender	Treatment	Age	time	weight
1:	41	Male	No	37	weight_t1	53
2:	101	Female	No	38	weight_t1	48
3:	59	Female	Yes	38	weight_t1	49
4:	91	Male	No	38	weight_t1	51
5:	2	Female	No	38	weight_t1	52
6:	6	Male	Yes	38	weight_t1	52

Reorder the data by Id and time:

```
setkeyv(dtL.data, c("Id", "time"))  
head(dtL.data)
```

	Id	Gender	Treatment	Age	time	weight
1:	1	Male	Yes	40	weight_t1	50
2:	1	Male	Yes	40	weight_t2	57
3:	1	Male	Yes	40	weight_t3	56
4:	2	Female	No	38	weight_t1	52
5:	2	Female	No	38	weight_t2	57
6:	2	Female	No	38	weight_t3	63

4.10.2 Multivariate melt

Use a list of vectors each containing a vector with the columns to be melted:

```
dtL.data <- melt(dtW.data, id.vars = c("Id","Gender","Treatment","Age"),
  measure=list(c("weight_t1","weight_t2","weight_t3"),
    c("size_t1","size_t2","size_t3")),
  variable.name = "time", value.name = c("weight","size"))

dtL.data.bis <- melt(dtW.data, id.vars = c("Id","Gender","Treatment","Age"),
  measure=patterns("weight_t","size_t"),
  variable.name = "time", value.name = c("weight","size"))

identical(dtL.data,dtL.data.bis)
```

[1] TRUE

dtL.data

```
   Id Gender Treatment Age time weight  size
1:  41   Male       No  37    1    53 47.59
2: 101 Female       No  38    1    48 49.51
3:  59 Female      Yes  38    1    49 51.08
4:  91   Male       No  38    1    51 52.05
5:   2 Female       No  38    1    52 50.26
---
302: 11   Male       No  42    3    59 60.94
303: 54   Male      Yes  42    3    64 63.98
304: 73   Male      Yes  43    3    54 60.93
305: 45 Female      Yes  43    3    61 56.18
306: 10 Female      Yes  43    3    64 62.94
```

4.11 Converting a dataset from the long format to the wide format

4.11.1 Univariate

Data in the long format:

```
head(dtL.data)
```

	Id	Gender	Treatment	Age	time	weight	size
1:	41	Male	No	37	1	53	47.59
2:	101	Female	No	38	1	48	49.51
3:	59	Female	Yes	38	1	49	51.08
4:	91	Male	No	38	1	51	52.05
5:	2	Female	No	38	1	52	50.26
6:	6	Male	Yes	38	1	52	49.37

The conversion can be done using a formula:

- left side: variables that do not vary
- right side: variable indexing the repetition whose values will be used to name the new columns.

```
dtW.data <- dcast(dtL.data, value.var = c("weight"),  
                  formula = Id + Gender + Treatment + Age ~ time)
```

Data in the wide format:

```
setnames(dtW.data, old = c("1","2","3"), new = paste0("weight_t",1:3))  
dtW.data
```

	Id	Gender	Treatment	Age	weight_t1	weight_t2	weight_t3
1:	1	Male	Yes	40	50	57	56
2:	2	Female	No	38	52	57	63
3:	3	Male	No	41	47	54	62
4:	4	Female	Yes	41	48	55	60
5:	5	Female	Yes	42	52	56	64

98:	98	Male	No	39	53	59	57
99:	99	Female	Yes	42	51	57	62
100:	100	Female	No	40	53	55	59
101:	101	Female	No	38	48	58	55
102:	102	Female	No	39	52	58	68

4.11.2 Multivariate

Same as before but with several elements in the argument `value.var`. Note that the repetition index (here `time`) must be the same for both variables:

```
dtW.data <- dcast(dtL.data, value.var = c("weight", "size"),
                  formula = Id + Gender + Treatment + Age ~ time)
```

Data in the wide format:

```
dtW.data
```

```
      Id Gender Treatment Age weight_1 weight_2 weight_3 size_1 size_2 size_3
1:    1  Male      Yes  40         50         57         56  50.67  55.88  61.69
2:    2 Female     No  38         52         57         63  50.26  55.73  60.37
3:    3  Male     No  41         47         54         62  46.61  50.89  56.52
4:    4 Female     Yes  41         48         55         60  45.95  53.10  59.82
5:    5 Female     Yes  42         52         56         64  52.86  58.41  63.79
---
98:  98  Male     No  39         53         59         57  49.51  53.80  61.13
99:  99 Female     Yes  42         51         57         62  47.60  56.55  59.47
100: 100 Female     No  40         53         55         59  50.06  54.90  61.89
101: 101 Female     No  38         48         58         55  49.51  54.01  62.32
102: 102 Female     No  39         52         58         68  47.35  56.08  59.49
```


5 Graphical display

5.1 Descriptive plots

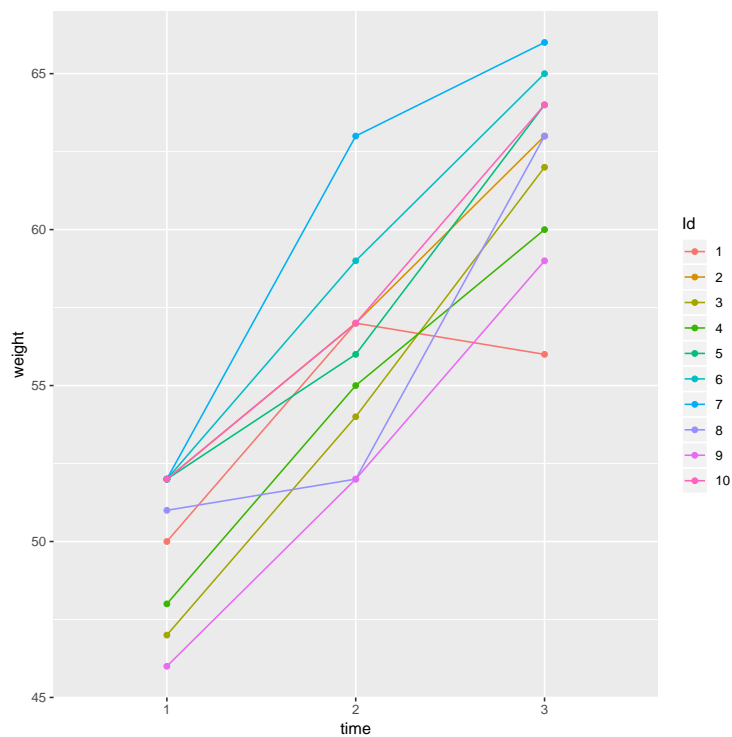
```
head(dtL.data)
```

	Id	Gender	Treatment	Age	time	weight	size
1:	1	Male	Yes	40	1	50	50.67
2:	2	Female	No	38	1	52	50.26
3:	3	Male	No	41	1	47	46.61
4:	4	Female	Yes	41	1	48	45.95
5:	5	Female	Yes	42	1	52	52.86
6:	6	Male	Yes	38	1	52	49.37

5.1.1 Spaguetti plot

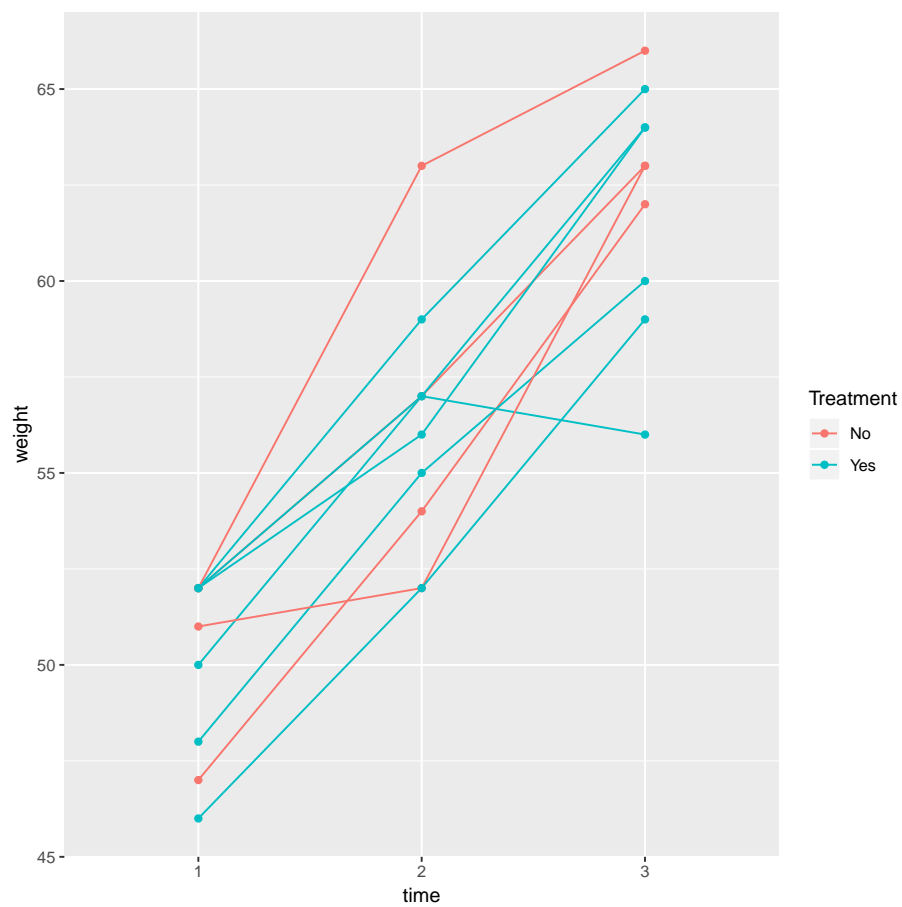
1. color by individual (first ten individuals)

```
gg.spaguettii1 <- ggplot(dtL.data[Id %in% 1:10],  
  aes(x = time, y = weight, color = Id, group = Id))  
gg.spaguettii1 <- gg.spaguettii1 + geom_line() + geom_point()  
gg.spaguettii1
```



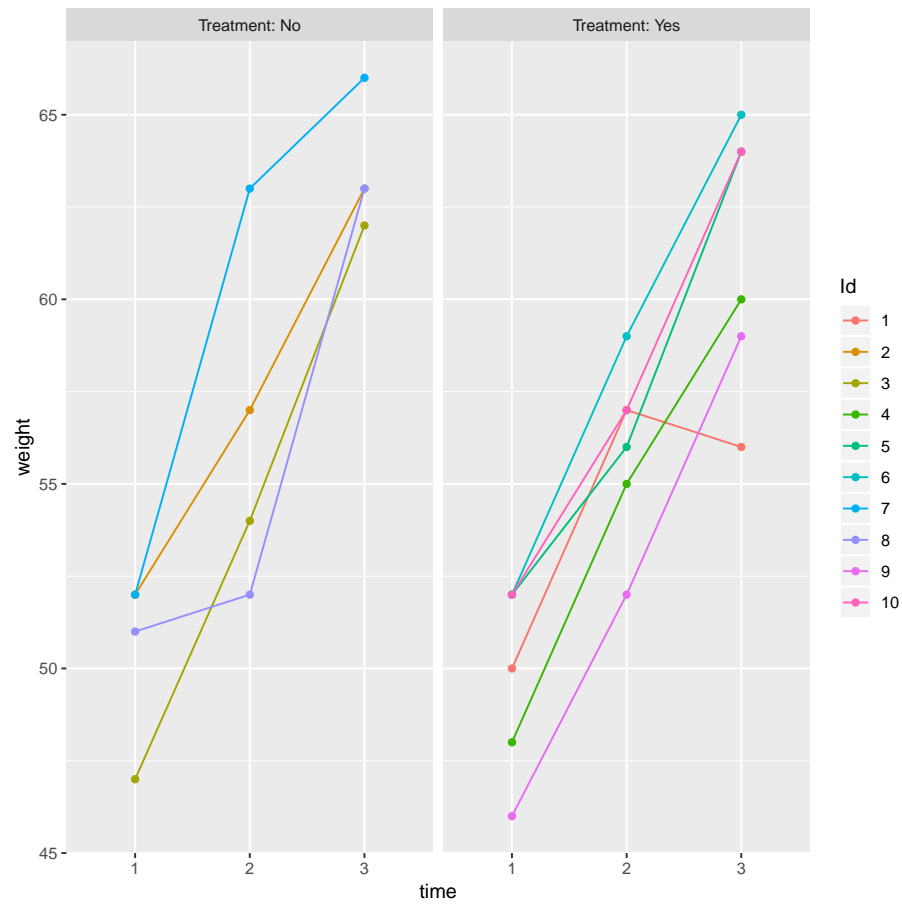
2. color by treatment group (first ten individuals)

```
gg.spaguetti2 <- ggplot(dtL.data[Id %in% 1:10],  
  aes(x = time, y = weight, color = Treatment, group = Id))  
gg.spaguetti2 <- gg.spaguetti2 + geom_line() + geom_point()  
gg.spaguetti2
```



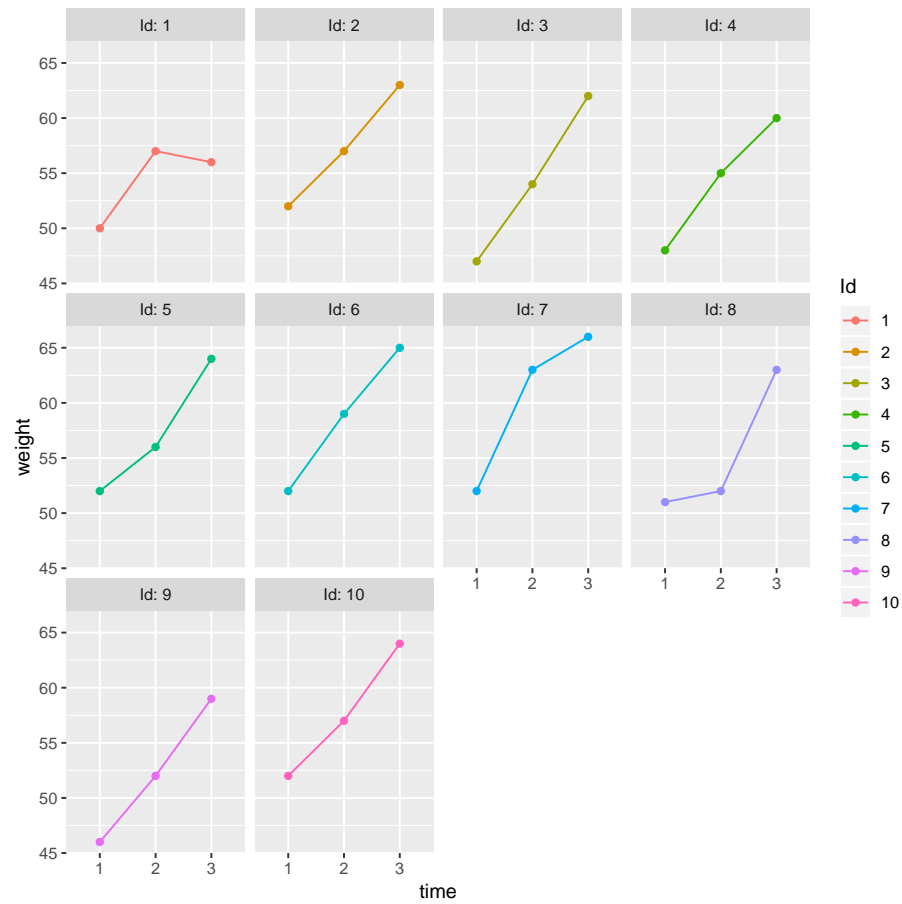
3. pannel for each treatment group (first ten individuals)

```
gg.spaguetti3 <- ggplot(dtL.data[Id %in% 1:10],
  aes(x = time, y = weight, color = Id, group = Id))
gg.spaguetti3 <- gg.spaguetti3 + geom_line() + geom_point()
gg.spaguetti3 <- gg.spaguetti3 + facet_wrap(~ Treatment, labeller = label_both)
gg.spaguetti3
```



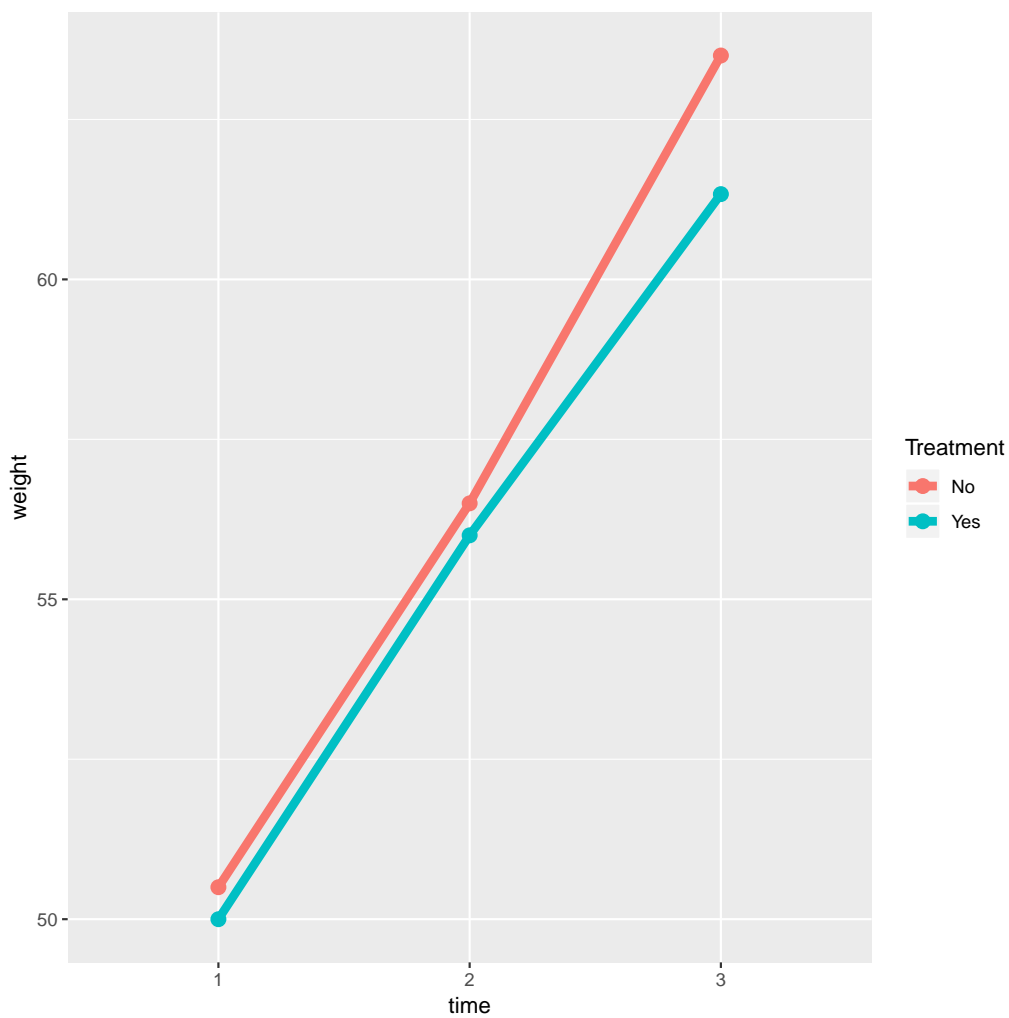
4. individual spaghetti plot (first ten individuals)

```
gg.spaguetti4 <- ggplot(dtL.data[Id %in% 1:10],
  aes(x = time, y = weight, color = Id, group = Id))
gg.spaguetti4 <- gg.spaguetti4 + geom_line() + geom_point()
gg.spaguetti4 <- gg.spaguetti4 + facet_wrap(~ Id, labeller = label_both)
gg.spaguetti4
```



5.1.2 Display the mean over time

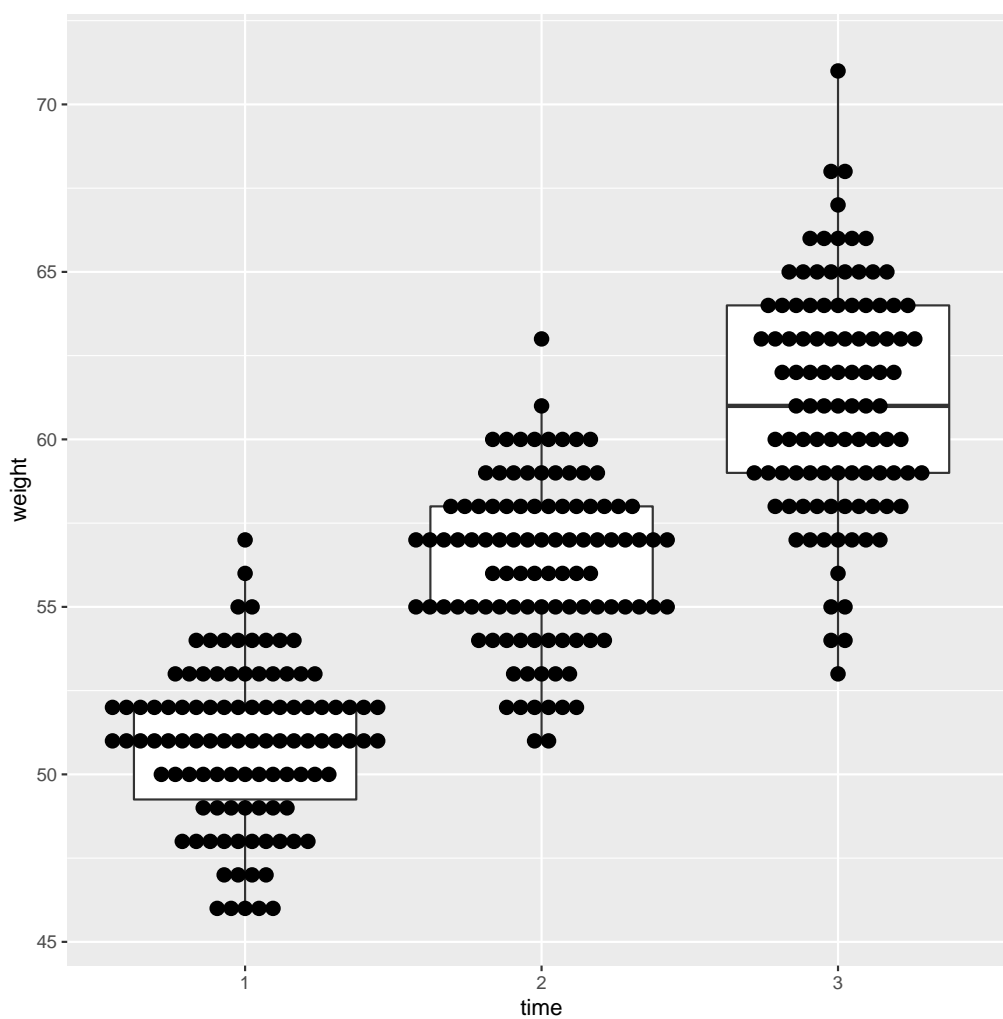
```
gg.mean <- ggplot(dtL.data[Id %in% 1:10], aes(x = time, y = weight))
gg.mean <- gg.mean + stat_summary(aes(group = Treatment, color = Treatment),
                                geom = "line", fun.y = mean, size = 2)
gg.mean <- gg.mean + stat_summary(aes(group = Treatment, color = Treatment),
                                geom = "point", fun.y = mean, size = 3)
```



5.1.3 Boxplot + points (non-overlapping)

```
gg.hist <- ggplot(dtL.data, aes(x = time, y = weight))  
gg.hist <- gg.hist + geom_boxplot()  
gg.hist <- gg.hist + geom_dotplot(binaxis = "y", stackdir = "center", dotsize = 0.5)  
gg.hist
```

`'stat_bindot()'` using `'bins = 30'`. Pick better value with `'binwidth'`.



5.2 Diagnostic plots

Consider the linear model:

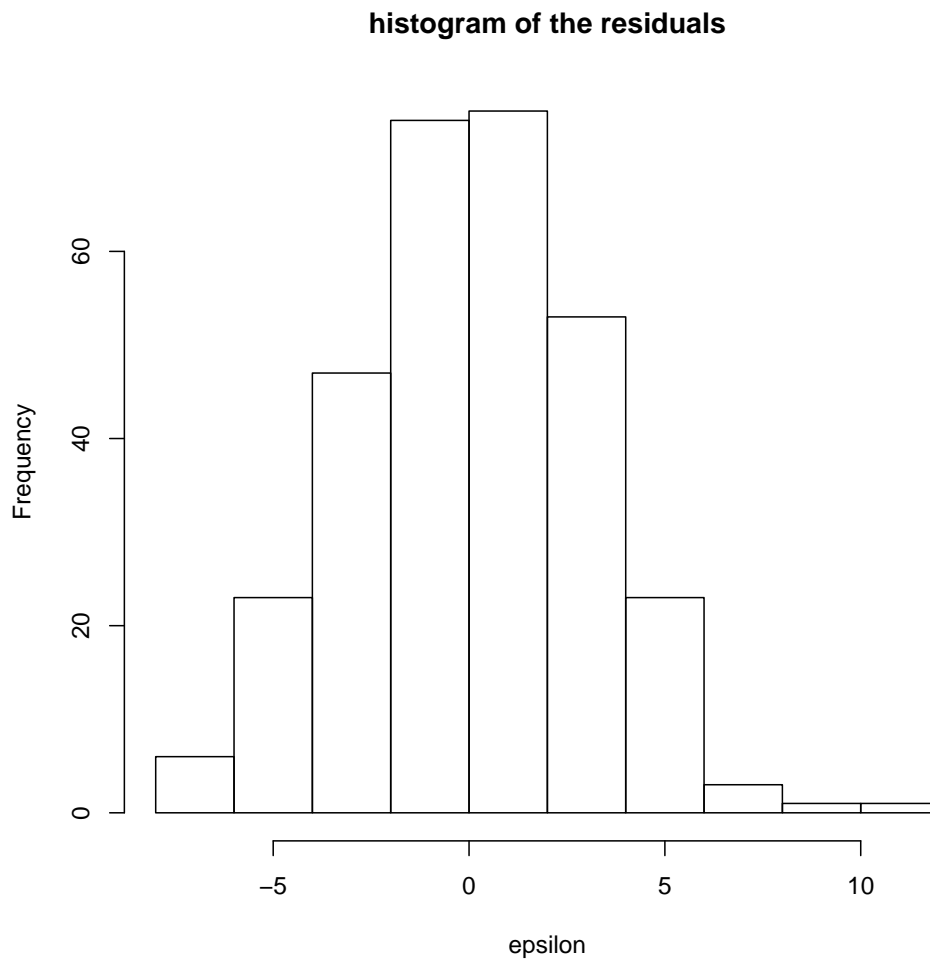
```
e.lm <- lm(weight ~ Age + Treatment + size,  
            data = dtL.data)
```

5.2.1 Histogram of the residuals

Extract the residuals:

```
epsilon <- residuals(e.lm, type = "response")
```

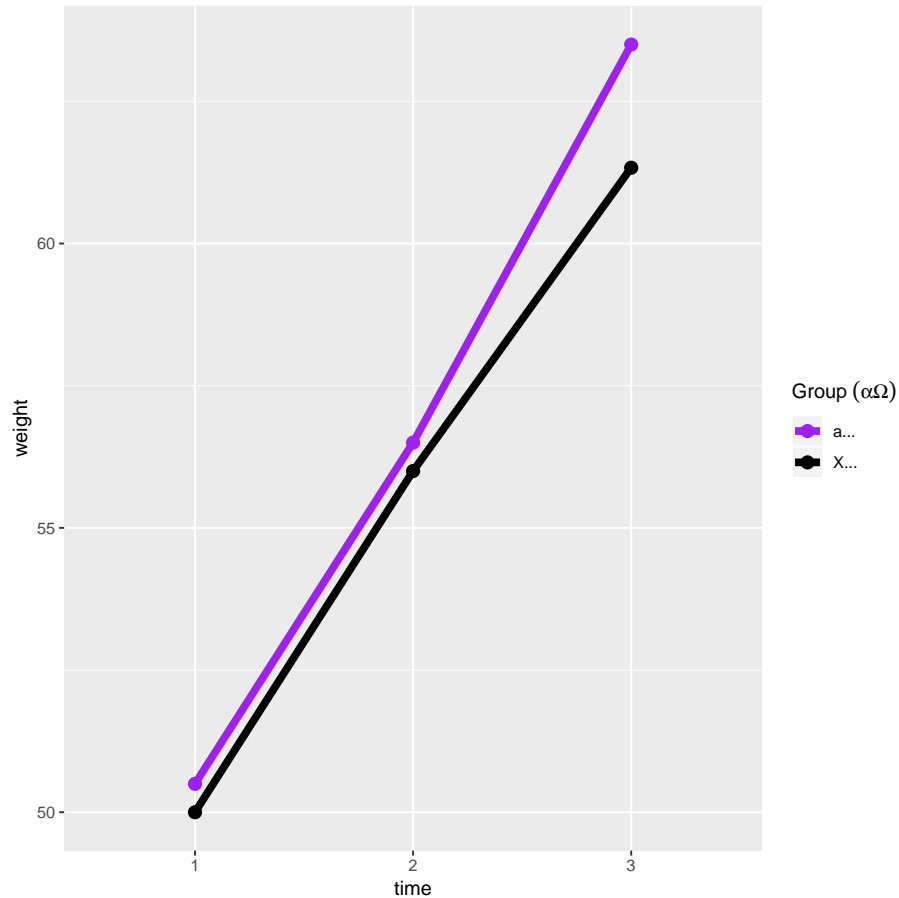
Display the histogram



5.3 Customize graphic

5.3.1 Modify the legend of a discrete scale (with greek letters)

```
gg.mean2 <- gg.mean + scale_colour_manual(name = expression("Group"~(alpha*Omega)),  
  labels = c("\u03b1\u2090", "X\u1D30"),  
  values = c("No" = "purple",  
    "Yes" = "black"))
```



See also:

- https://en.wikipedia.org/wiki/List_of_Unicode_characters
- https://en.wikipedia.org/wiki/Unicode_subscripts_and_superscripts
- <https://stackoverflow.com/questions/5293715/how-to-use-greek-symbols-in-ggplot2>

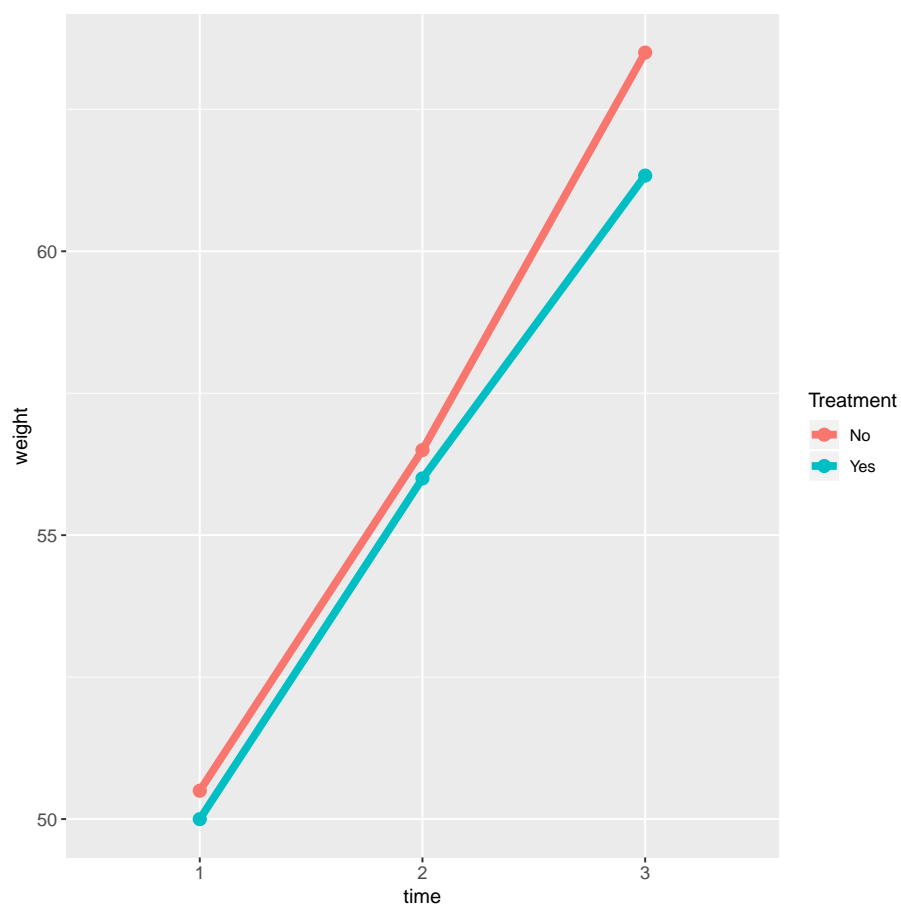
5.3.2 Change the name of the legend

```
gg.mean3 <- gg.mean2 + labs(colour="xyz")
```

5.3.3 Increase the font size

All text:

```
gg.mean3 <- gg.mean + theme(text = element_text(size=10))
```



Only x axis labels:

```
gg.mean3 <- gg.mean + theme(axis.text = element_text(size=10))
```

Only axis title:

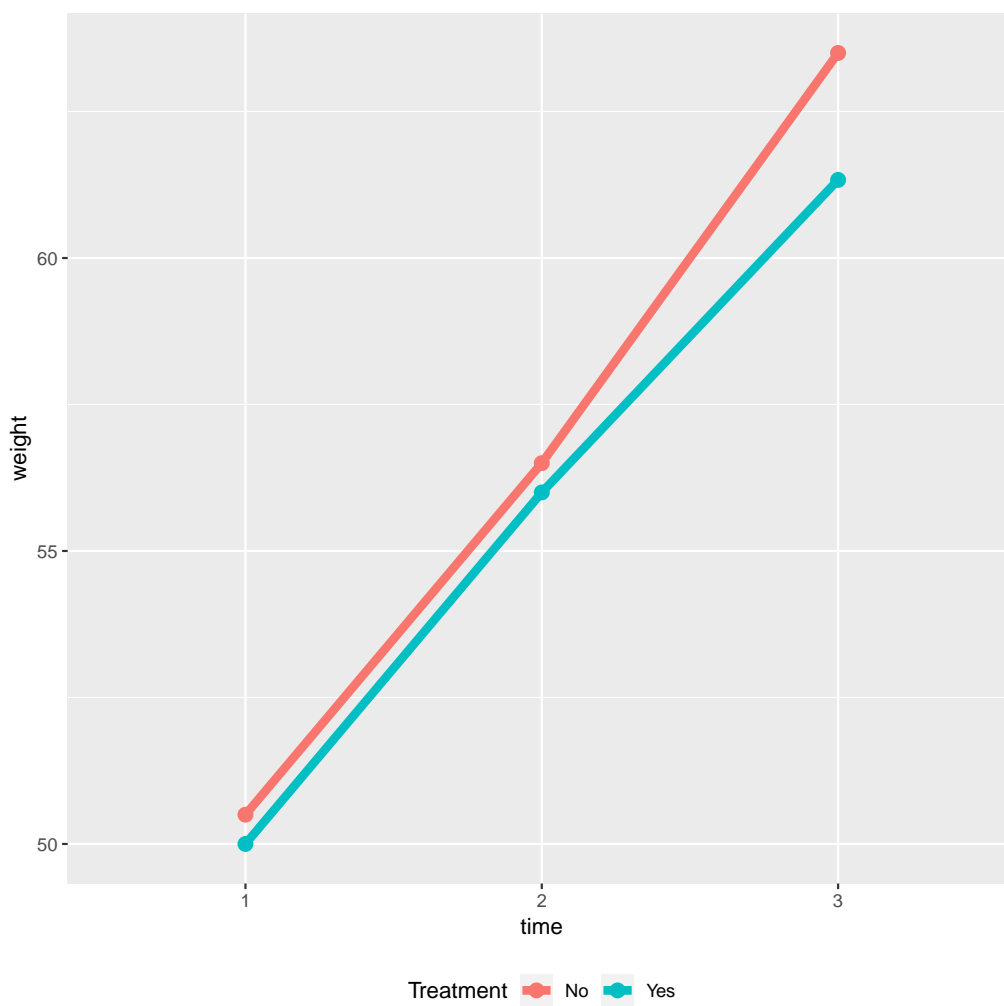
```
gg.mean3 <- gg.mean + theme(axis.title = element_text(size=10))
```

5.3.4 Increase size of the legend labels

```
gg.mean + theme(axis.title = element_text(size=10), legend.key.size = unit(3, "line"))
```

5.3.5 Put the legend at the bottom

```
gg.mean4 <- gg.mean + theme(legend.position="bottom")
```



5.3.6 Number of lines in the legend

```
gg.mean + guides(color = guide_legend(nrow = 2, byrow = TRUE))
```

5.3.7 Default ggplot color palette

```
gg_color_hue <- function(n) {  
  hues = seq(15, 375, length = n + 1)  
  hcl(h = hues, l = 65, c = 100)[1:n]  
}
```

5.3.8 Color blind palette

```
ggthemes::colorblind_pal()(8) ## also consider scale_color_colorblind
```

```
[1] "#000000" "#E69F00" "#56B4E9" "#009E73" "#F0E442" "#0072B2" "#D55E00" "#CC79A7"
```

5.3.9 Rotate x-axis labels

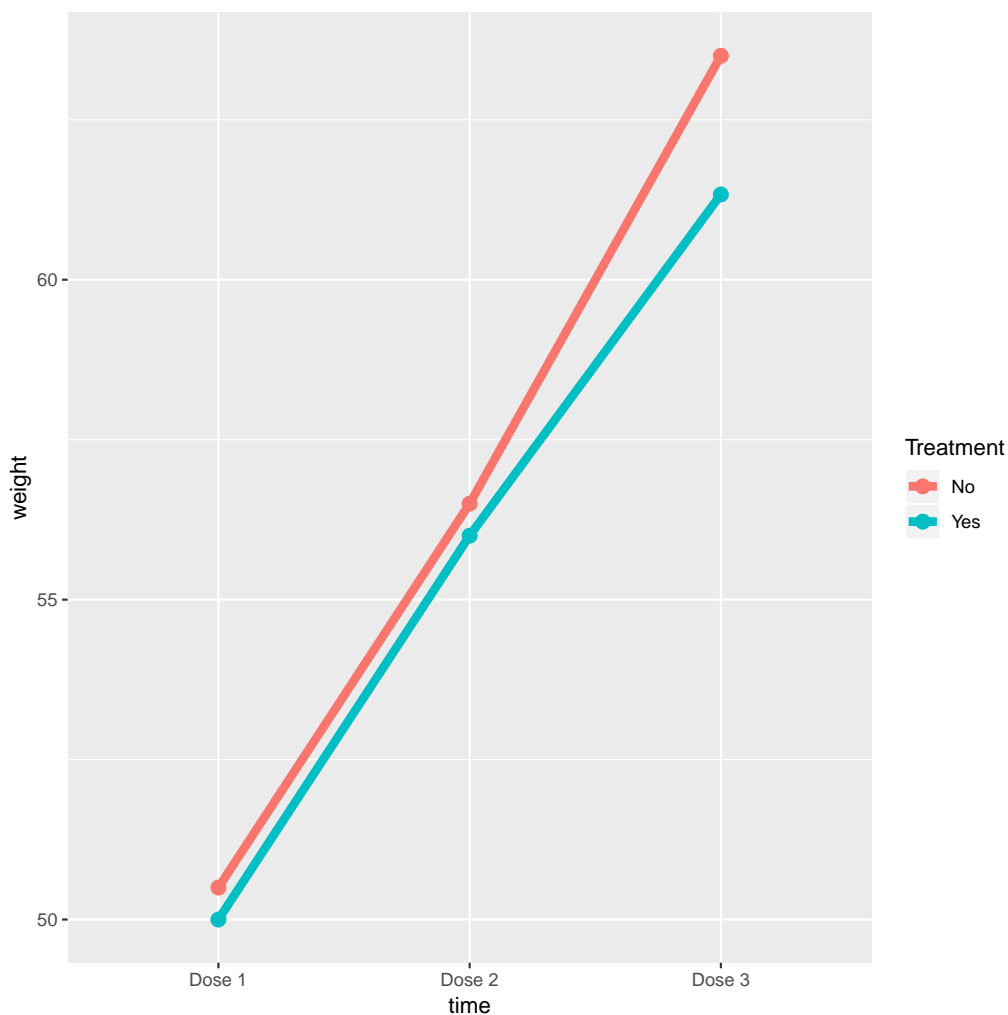
```
theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

List of 1

```
$ axis.text.x:List of 11  
 ..$ family      : NULL  
 ..$ face        : NULL  
 ..$ colour      : NULL  
 ..$ size        : NULL  
 ..$ hjust       : num 1  
 ..$ vjust       : NULL  
 ..$ angle       : num 90  
 ..$ lineheight  : NULL  
 ..$ margin      : NULL  
 ..$ debug       : NULL  
 ..$ inherit.blank: logi FALSE  
 ..- attr(*, "class")= chr [1:2] "element_text" "element"  
- attr(*, "class")= chr [1:2] "theme" "gg"  
- attr(*, "complete")= logi FALSE  
- attr(*, "validate")= logi TRUE
```

5.3.10 Change tick mark labels

```
gg.mean5 <- gg.mean + scale_x_discrete(breaks=c("1","2","3"),  
                                         labels=c("Dose 1", "Dose 2", "Dose 3"))
```



5.3.11 Combine ggplots

(from <https://stackoverflow.com/questions/13649473/add-a-common-legend-for-combined-ggplots>)

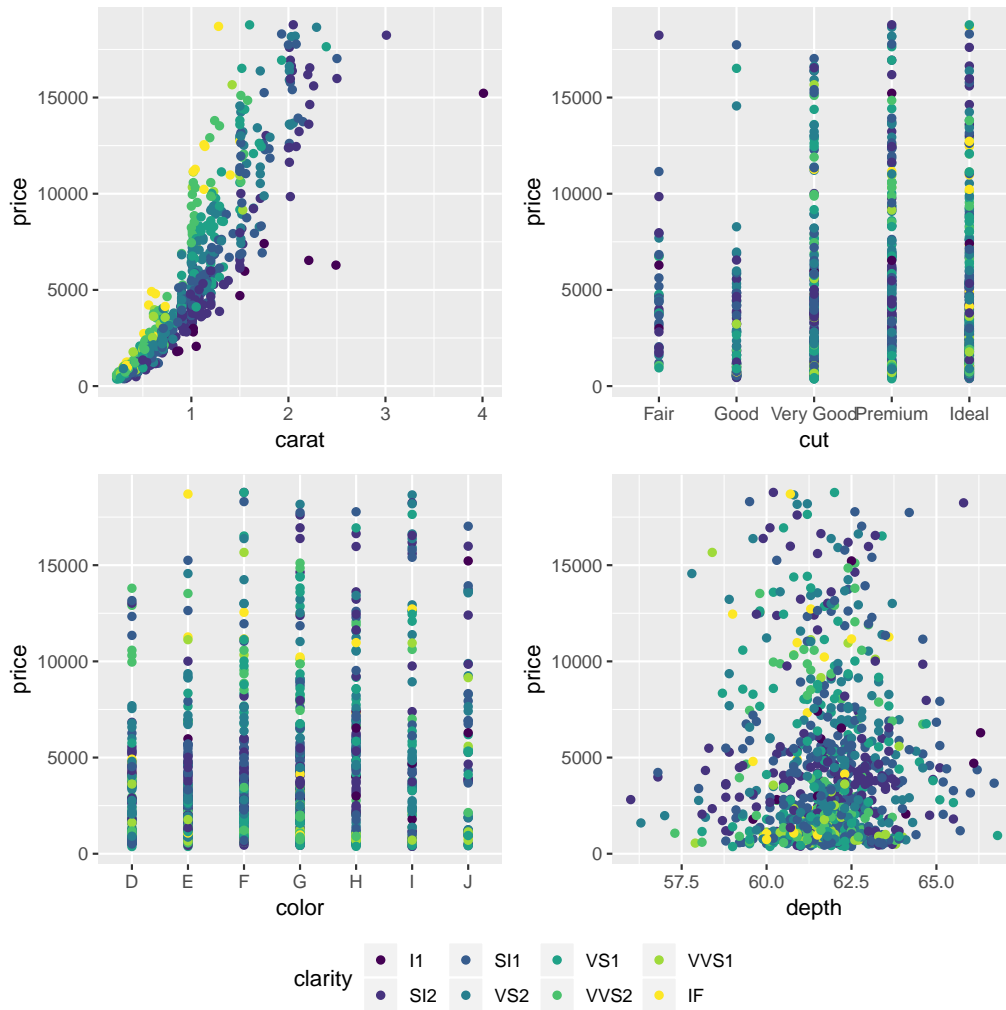
```
library(ggpubr)  
  
dsamp <- diamonds[sample(nrow(diamonds), 1000), ]  
p1 <- qplot(carat, price, data = dsamp, colour = clarity)
```

```

p2 <- qplot(cut, price, data = dsamp, colour = clarity)
p3 <- qplot(color, price, data = dsamp, colour = clarity)
p4 <- qplot(depth, price, data = dsamp, colour = clarity)

out <- ggarrange(p1, p2, p3, p4, ncol=2, nrow=2, common.legend = TRUE, legend="bottom")

```



5.3.12 Symbols in facet names

5.4 Path diagram

Using lava:

```
m <- lvm(Y~E+X1+X2+M,M~E,E~X2)
```

```
plot(m, plot.engine="rgraphviz")
```

Dynamic graph:

```
plot(m, plot.engine="visnetwork")
```

6 Modeling

6.1 Test proportions

```
binom.exact(c(15,4), p = 0.5) ## 15 success, 4 failures
```

```
Exact two-sided binomial test (central method)
```

```
data: c(15, 4)
number of successes = 15, number of trials = 19, p-value = 0.01921
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.5443469 0.9394755
sample estimates:
probability of success
 0.7894737
```

6.2 Compare proportions between groups (with fixed margins!)

Data:

```
tab <- rbind(c(8,3),
             c(5,12))
colnames(tab) <- c("worse","better")
rownames(tab) <- c("Dalteparin","Placebo")
tab <- tab[2:1,]
tab
```

	worse	better
Placebo	5	12
Dalteparin	8	3

Unpaired: (fisher test)

```
fisher.exact(tab)
```

Two-sided Fisher's Exact Test (usual method using minimum likelihood)

```
data: tab
p-value = 0.05103
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.0272 1.0631
sample estimates:
odds ratio
 0.168551
```

Paired: (mc-nemar test)

```
mcnemar.exact(tab)
```

Exact McNemar test (with central confidence intervals)

```
data: tab
b = 12, c = 8, p-value = 0.5034
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.5638258 4.2303974
sample estimates:
odds ratio
 1.5
```


6.3 permutation t-test

Data:

```
set.seed(10)
X <- rlnorm(10, meanlog = 2, sdlog = 0.5)
Y <- rlnorm(10, meanlog = 1.8, sdlog = 0.5)
```

Approximation based on asymptotic result:

```
permTS(x = X, y = Y, method = "pclt")
```

Permutation Test using Asymptotic Approximation

```
data: X and Y
Z = -1.5476, p-value = 0.1217
alternative hypothesis: true mean X - mean Y is not equal to 0
sample estimates:
mean X - mean Y
-1.533514
```

Approximation based on simulations:

```
permTS(x = X, y = Y, method = "exact.mc")
```

Exact Permutation Test Estimated by Monte Carlo

```
data: X and Y
p-value = 0.112
alternative hypothesis: true mean X - mean Y is not equal to 0
sample estimates:
mean X - mean Y
-1.533514
```

```
p-value estimated from 999 Monte Carlo replications
99 percent confidence interval on p-value:
0.07625212 0.15272627
```

Exact:

```
permTS(x = X, y = Y, method = "exact.ce")
```

Exact Permutation Test (complete enumeration)

```
data: X and Y
p-value = 0.1238
alternative hypothesis: true mean X - mean Y is not equal to 0
sample estimates:
mean X - mean Y
-1.533514
```

6.4 Testing median

Data:

```
set.seed(10)
X <- rlnorm(100, meanlog = 2, sdlog = 0.5) - 6.5
```

Median test

```
quantileTest(X)
```

Exact Test/Confidence Interval for Median

```
data: X
quantile for prob = 0.5, pAG = 0.18410, pAL = 0.86437, pc = 0.36820, p-value = 0.3682
alternative hypothesis: true median is not equal to 0
95 percent confidence interval:
 -0.3701565  1.4997902
sample estimates:
 median
0.2082777
```

```
df <- data.frame(value=X)
e <- rq(value~1, tau = 0.5, data = df)
summary(e, se = "nid")
```

Warning message:

In rq.fit.br(x, y, tau = tau, ...) : Solution may be nonunique

Call: rq(formula = value ~ 1, tau = 0.5, data = df)

tau: [1] 0.5

Coefficients:

	Value	Std. Error	t value	Pr(> t)
(Intercept)	0.20213	0.49381	0.40932	0.68319

Other quantiles

```
e2 <- rq(value~1, tau = c(0.25,0.5,0.75), data = df)
summary(e2, se = "nid")
```

Warning messages:

1: In rq.fit.br(x, y, tau = tau, ...) : Solution may be nonunique
2: In rq.fit.br(x, y, tau = tau, ...) : Solution may be nonunique
3: In rq.fit.br(x, y, tau = tau, ...) : Solution may be nonunique

Call: rq(formula = value ~ 1, tau = c(0.25, 0.5, 0.75), data = df)

```
tau: [1] 0.25
```

```
Coefficients:
```

	Value	Std. Error	t value	Pr(> t)
(Intercept)	-1.61744	0.37283	-4.33828	0.00003

```
Call: rq(formula = value ~ 1, tau = c(0.25, 0.5, 0.75), data = df)
```

```
tau: [1] 0.5
```

```
Coefficients:
```

	Value	Std. Error	t value	Pr(> t)
(Intercept)	0.20213	0.49381	0.40932	0.68319

```
Call: rq(formula = value ~ 1, tau = c(0.25, 0.5, 0.75), data = df)
```

```
tau: [1] 0.75
```

```
Coefficients:
```

	Value	Std. Error	t value	Pr(> t)
(Intercept)	3.43848	0.68607	5.01186	0.00000

6.5 Testing linear hypotheses

Consider the linear model:

```
e.lm <- lm(weight ~ Age + Treatment + size,
           data = dtL.data)
summary(e.lm)$coef
```

```
              Estimate Std. Error    t value    Pr(>|t|)
(Intercept)  13.11292977  5.84498969   2.2434479 2.559263e-02
Age          -0.05479836  0.13849481  -0.3956709 6.926272e-01
TreatmentYes -0.65247721  0.36126020  -1.8061143 7.189597e-02
size         0.81718969  0.03513376  23.2593869 2.743182e-69
```

To test linear hypotheses we first need to define them using a contrast matrix:

```
name.coef <- names(coef(e.lm))
n.coef <- length(name.coef)

C <- matrix(0,nrow = 3, ncol = n.coef,
           dimnames = list (c("Age","2 Treatment","All"), name.coef))
C["Age","Age"] <- 1
C["2 Treatment","TreatmentYes"] <- 2
C["All",-1] <- 1
C
```

```
              (Intercept) Age TreatmentYes size
Age              0      1              0      0
2 Treatment      0      0              2      0
All              0      1              1      1
```

6.5.1 Separate Wald tests of linear hypotheses

No adjustment for multiple comparison:

```
summary(glht(e.lm, linfct = C), test = univariate())
```

Simultaneous Tests for General Linear Hypotheses

Fit: `lm(formula = weight ~ Age + Treatment + size, data = dtL.data)`

Linear Hypotheses:

```
              Estimate Std. Error t value Pr(>|t|)
Age == 0      -0.0548     0.1385  -0.396   0.6926
2 Treatment == 0 -1.3050     0.7225  -1.806   0.0719 .
All == 0       0.1099     0.3513   0.313   0.7546
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Univariate p values reported)
```

Adjustment using bonferroni:

```
summary(glht(e.lm, linfct = C), test = adjusted("bonferroni"))
```

Simultaneous Tests for General Linear Hypotheses

```
Fit: lm(formula = weight ~ Age + Treatment + size, data = dtL.data)
```

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t)
Age == 0	-0.0548	0.1385	-0.396	1.000
2 Treatment == 0	-1.3050	0.7225	-1.806	0.216
All == 0	0.1099	0.3513	0.313	1.000

(Adjusted p values reported -- bonferroni method)

Adjustment using the max statistic:

```
summary(glht(e.lm, linfct = C), test = adjusted("single-step"))
```

Simultaneous Tests for General Linear Hypotheses

```
Fit: lm(formula = weight ~ Age + Treatment + size, data = dtL.data)
```

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t)
Age == 0	-0.0548	0.1385	-0.396	0.916
2 Treatment == 0	-1.3050	0.7225	-1.806	0.157
All == 0	0.1099	0.3513	0.313	0.948

(Adjusted p values reported -- single-step method)

Alternative syntax (without contrast matrix):

```
summary(glht(e.lm,
  linfct = c("Age = 0",
             "2*TreatmentYes = 0",
             "Age + TreatmentYes + size = 0")),
  test = adjusted("single-step"))
```

Simultaneous Tests for General Linear Hypotheses

```
Fit: lm(formula = weight ~ Age + Treatment + size, data = dtL.data)
```

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t)
Age == 0	-0.0548	0.1385	-0.396	0.916
2 * TreatmentYes == 0	-1.3050	0.7225	-1.806	0.157
Age + TreatmentYes + size == 0	0.1099	0.3513	0.313	0.948

(Adjusted p values reported -- single-step method)

6.5.2 Confidence intervals associated with linear hypotheses

With no adjustment for multiplicity:

```
confint(glht(e.lm, linfct = C), calpha = univariate_calpha())
```

Simultaneous Confidence Intervals

```
Fit: lm(formula = weight ~ Age + Treatment + size, data = dtL.data)
```

```
Quantile = 1.9679  
95% confidence level
```

Linear Hypotheses:

	Estimate	lwr	upr
Age == 0	-0.0548	-0.3273	0.2177
2 Treatment == 0	-1.3050	-2.7268	0.1169
All == 0	0.1099	-0.5815	0.8013

With adjustment for multiplicity:

```
confint(glht(e.lm, linfct = C), calpha = adjusted_calpha())
```

Simultaneous Confidence Intervals

```
Fit: lm(formula = weight ~ Age + Treatment + size, data = dtL.data)
```

```
Quantile = 2.314  
95% family-wise confidence level
```

Linear Hypotheses:

	Estimate	lwr	upr
Age == 0	-0.0548	-0.3753	0.2657
2 Treatment == 0	-1.3050	-2.9769	0.3670
All == 0	0.1099	-0.7031	0.9229

6.5.3 Joint test of linear hypotheses

One can use the `Ftest()` or `Chisqtest()` to obtain a joint test:

```
summary(glht(e.lm,  
  linfct = c("Age = 0",  
             "2*TreatmentYes = 0",  
             "Age + TreatmentYes + size = 0")),  
  test = Ftest())
```

General Linear Hypotheses

Linear Hypotheses:

	Estimate
Age == 0	-0.0548
2 * TreatmentYes == 0	-1.3050
Age + TreatmentYes + size == 0	0.1099

Global Test:

	F	DF1	DF2	Pr(>F)
1	181.2	3	302	3.349e-67

The same can be obtained using the `linearHypothesis` method from the `car` package:

```
linearHypothesis(e.lm, hypothesis.matrix = C, rhs = c(0,0,0))
```

Linear hypothesis test

Hypothesis:

```
Age = 0
2 TreatmentYes = 0
Age + TreatmentYes + size = 0
```

Model 1: restricted model

Model 2: weight ~ Age + Treatment + size

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	305	7748.5				
2	302	2767.2	3	4981.3	181.21	< 2.2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

7 Loops and parallel computations

7.1 Apply with progress bar

```
ls.res <- pbapply::pblapply(1:5, FUN = rnorm)
```

```
| 0 % ~calculating  
|+++++++| 20% ~00s  
|+++++++| 40% ~00s  
|+++++++| 60% ~00s  
|+++++++| 80% ~00s  
|+++++++| 100% elapsed = 00s
```


7.2 Parallel computation

7.2.1 Detect the number of cores

```
cores <- parallel::detectCores()
cores
```

```
[1] 4
```

7.2.2 Start a cluster

```
cpus <- 2

cl <- snow::makeSOCKcluster(cpus)
doSNOW::registerDoSNOW(cl)
```

7.2.3 Get the name of each core

```
cpus.name <- unlist(parallel::clusterCall(cl = cl, function(x){
  myName <- paste(Sys.info()[['nodename']], Sys.getpid(), sep='-')
  return(myName)
}))
cpus.name
```

```
[1] "SUND31034-5800" "SUND31034-5992"
```

7.2.4 Export element to cluster

```
parallel::clusterExport(cl, varlist = "cpus.name")

parallel::clusterCall(cl = cl, function(x){
  indexCPU <- which(cpus.name == paste(Sys.info()[['nodename']], Sys.getpid(), sep='-')
  indexCPU
})
```

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] 2
```

7.2.5 Show progress bar (in console)

```
n.sim <- 20

pb <- txtProgressBar(max = n.sim, style=3)
opts <- list(progress = function(n) setTxtProgressBar(pb, n))

ls.res <- foreach::"%dopar%"(
  foreach::foreach(i=1:n.sim, .options.snow=opts), {
    Sys.sleep(0.1)
  })
```

7.2.6 Show progress bar (external)

```
n.sim <- 20
parallel::clusterExport(cl, varlist = "n.sim")

ls.res <- foreach::"%dopar%"(
  foreach::foreach(iCpus=1:cpus), {
    pb <- tcltk::tkProgressBar(paste0("CPU ",iCpus), min = 0,
      max = n.sim, initial = 0)

    for(iSim in 1:n.sim){
      Sys.sleep(0.1)
      tcltk::setTkProgressBar(pb = pb, value = iSim,
        label = paste(iSim," over ",n.
sim," iterations done") )
    }

    close(pb)
  })
```

7.2.7 Stop a cluster

```
parallel::stopCluster(cl)
```

8 *lava* package

8.1 Generate repeated measurements

Model: Simulation:

```
set.seed(10)
dfW.data <- sim(m, n = 102, latent = FALSE)
```

Display simulated data:

```
head(dfW.data)
```

	weight_t1	Gender	Treatment	weight_t2	weight_t3	size_t1	size_t2	size_t3	Age	Id
1	49.59633	Male	Yes	56.62904	55.58780	50.66805	55.88362	61.69410	39.54546	1
2	52.35484	Female	No	56.68563	63.21026	50.26003	55.72930	60.36953	37.70748	2
3	46.53011	Male	No	54.36636	62.05018	46.61315	50.89281	56.52237	40.80342	3
4	48.48417	Female	Yes	54.79413	59.72995	45.95248	53.09941	59.82107	40.94933	4
5	52.17022	Female	Yes	55.71550	64.21010	52.86341	58.40516	63.79082	42.06512	5
6	52.18837	Male	Yes	58.86797	64.51316	49.36853	57.90530	64.45437	37.68392	6

Modify simulated data

```
dtW.data <- as.data.table(dfW.data)
dtW.data[,paste0("weight_t",1:3) := lapply(.SD,round),
          .SDcols = paste0("weight_t",1:3)]
dtW.data[,paste0("size_t",1:3) := lapply(.SD,round, digit = 2),
          .SDcols = paste0("size_t",1:3)]
dtW.data[,Age := round(Age)]

setcolororder(dtW.data, c("Id","Age","Gender","Treatment",
                           paste0("weight_t",1:3),paste0("size_t",1:3)))
head(dtW.data)
```

	Id	Age	Gender	Treatment	weight_t1	weight_t2	weight_t3	size_t1	size_t2	size_t3
1:	1	40	Male	Yes	50	57	56	50.67	55.88	61.69
2:	2	38	Female	No	52	57	63	50.26	55.73	60.37
3:	3	41	Male	No	47	54	62	46.61	50.89	56.52
4:	4	41	Female	Yes	48	55	60	45.95	53.10	59.82
5:	5	42	Female	Yes	52	56	64	52.86	58.41	63.79
6:	6	38	Male	Yes	52	59	65	49.37	57.91	64.45

Export data:

```
fwrite(dtW.data, file = "./mydata.csv", sep = ";", dec = ",")
fwrite(dtW.data, file = "./mydata.txt", sep = " ", dec = ".")
```

8.2 Generate data with heteroschadasticity

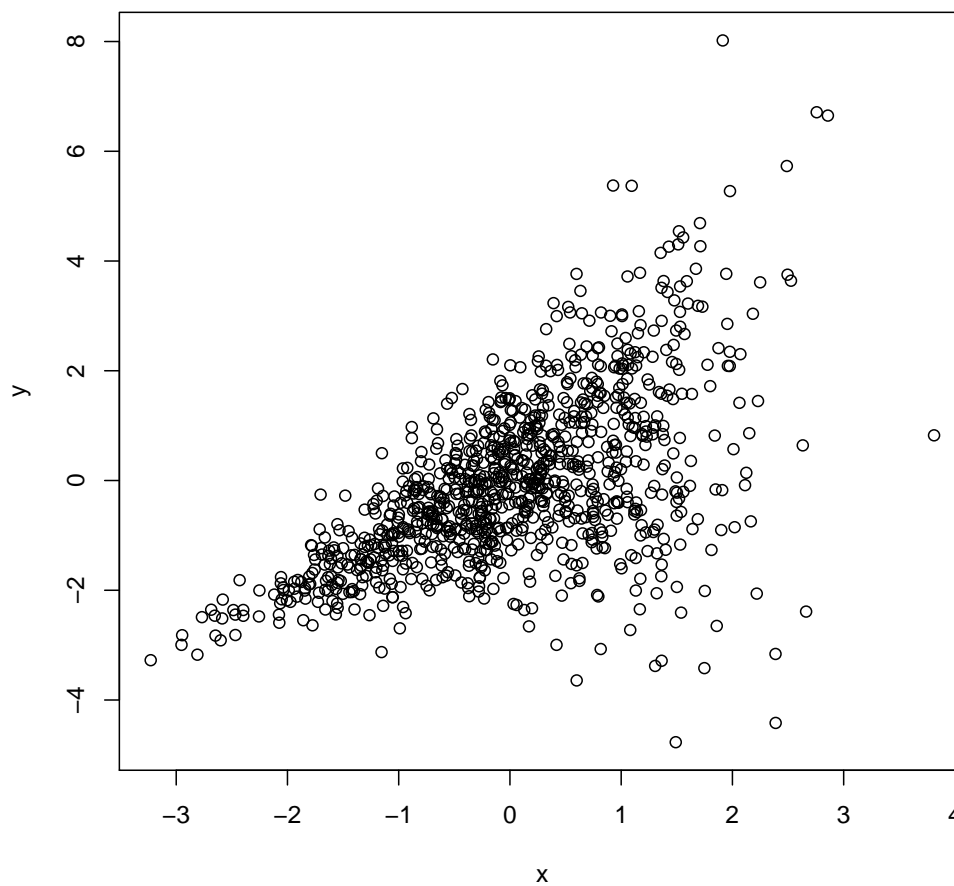
Model:

```
mSim <- lvm(y[m:v]~x)
constrain(mSim, v ~ x + a + b) <- function(x){ x[,2] + x[,3] * exp(x[,1]) }
parameter(mSim, start = c(0,1)) <- ~ a + b
```

Simulation:

```
set.seed(10)
n <- 1e3
df.tempo <- sim(mSim, n = n)
```

Display:



8.3 Generate survival time under non proportional hazard (non-PH)

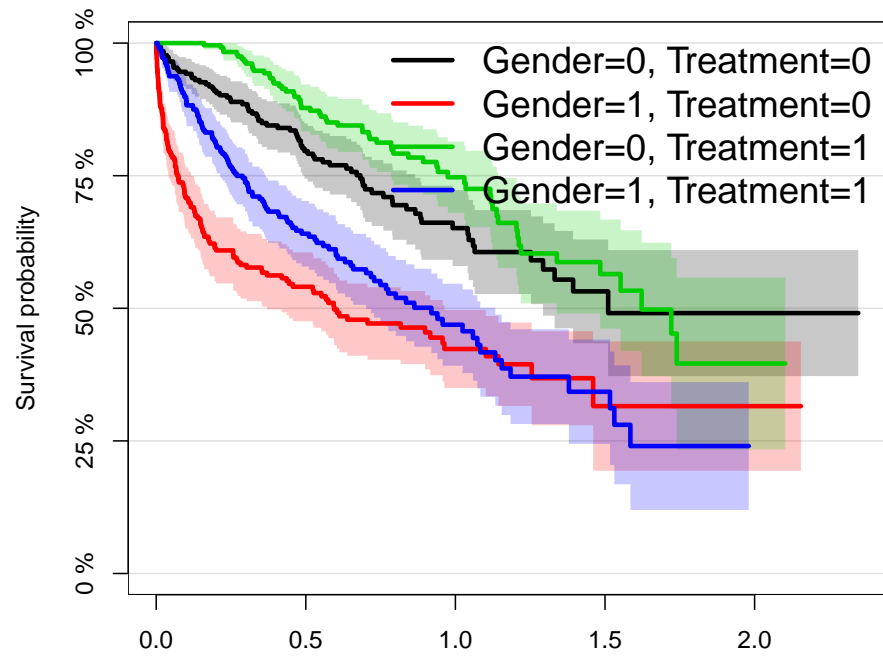
Model:

```
mSim <- lvm()
regression(mSim) <- eventtime ~ Gender + Age
regression(mSim) <- s ~ exp(0.6 * Treatment - 0.5 * Gender)
distribution(mSim, ~ Treatment + Gender) <- binomial.lvm()
distribution(mSim, ~ cens) <- coxWeibull.lvm(scale = 1)
distribution(mSim, ~ eventtime) <- coxWeibull.lvm(scale = 0.3, shape = ~ s)
eventTime(mSim) <- time ~ min(eventtime = 1, cens = 0)
```

Simulation:

```
set.seed(10)
n <- 1e3
df.tempo <- sim(mSim, n = n)
```

Display:



Gender					Time						
0, Treatment:	259	216	173	107	73	45	24	10	7	2	1
1, Treatment:	236	139	99	68	43	21	8	5	1	1	0
0, Treatment:	249	234	174	128	81	49	31	14	5	0	0
1, Treatment:	256	190	120	79	46	25	12	5	1	0	0

8.4 Generate survival time with delayed treatment effect

Generative model with non-PH group effect but no Age effect:

```
rates1 <- c(0.25,0.5,0.1); cuts <- c(0,3,5)
rates2 <- c(0.25,0.1,0.1); cuts <- c(0,3,5)
lasttime <- 20

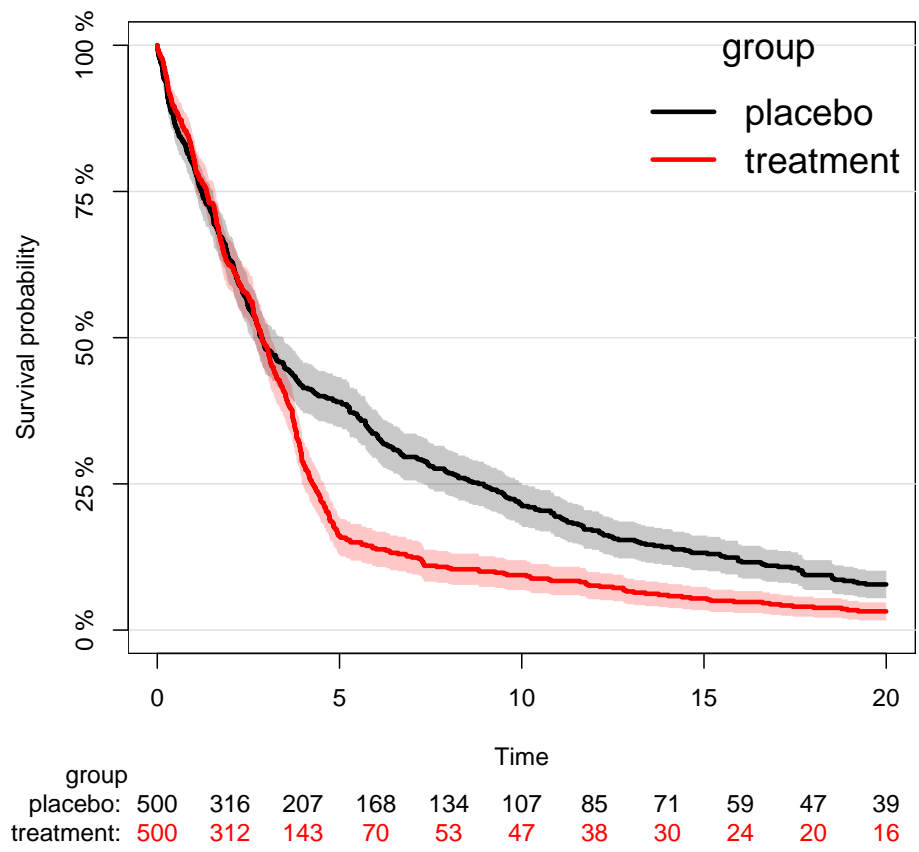
m1 <- lvm(Age[50:5]~1)
m2 <- lvm(Age[50:5]~1)
distribution(m1,~eventtime) <- coxExponential.lvm(rate=rates1,timecut=cuts)
distribution(m2,~eventtime) <- coxExponential.lvm(rate=rates2,timecut=cuts)
transform(m1,status~eventtime) <- function(x){as.numeric(x[,1]<= lasttime)}
transform(m2,status~eventtime) <- function(x){as.numeric(x[,1]<= lasttime)}
transform(m1,time~eventtime) <- function(x){pmin(lasttime,x[,1])}
transform(m2,time~eventtime) <- function(x){pmin(lasttime,x[,1])}
latent(m1) <- ~eventtime
latent(m2) <- ~eventtime
```

Simulate data:

```
set.seed(12)
n <- 500
d1 <- as.data.table(sim(m1,n,latent=FALSE))
d2 <- as.data.table(sim(m2,n,latent=FALSE))
dt.data <- rbind(cbind(d1,group="treatment"),cbind(d2,group="placebo"))
dt.data
```

```
      Age status      time      group
1: 46.68935     1  3.8755119 treatment
2: 53.52666     1  3.2816799 treatment
3: 47.86065     1  0.8515517 treatment
4: 47.94281     1 10.1313180 treatment
5: 45.53314     1  2.6198951 treatment
---
996: 46.47948     1  2.1560011 placebo
997: 52.78256     1  6.6831242 placebo
998: 45.10627     1  6.0589065 placebo
999: 49.24545     1 12.5248064 placebo
1000: 49.08839     1  1.9096902 placebo
```

Display survival curves by group:



8.5 Tune optimization parameters

```
library(lava)
set.seed(10)
dd <- sim(lvm(Y~X1+X2+X3), 100)
ee <- estimate(lvm(Y~X1+X2+X3+eta), data = dd, control = list(trace = 1, iter.max =
  200))
```

```
0:    260.69531: -0.00392152  0.00000  0.00000  0.00000  0.00000  1.89517  1.19723  0.900000
1:    232.00124: -0.0200212 -0.0305115  0.473562  0.497905  0.592152  1.80796  1.17166  0.808180
2:    202.02739: -0.0397653 -0.0662080  0.941194  0.998447  1.03587  1.47069  1.05604  0.430261
3:    189.85429: -0.0476622 -0.0778219  0.942488  0.999840  1.03679  1.27746  0.902950  0.0991273
4:    188.92755: -0.0488980 -0.0794005  0.942488  0.999840  1.03679  1.26451  0.851417  0.0150308
5:    188.92730: -0.0489067 -0.0794116  0.942488  0.999840  1.03679  1.26448  0.850633  0.0137765
6:    188.92730: -0.0489067 -0.0794116  0.942488  0.999840  1.03679  1.26448  0.850633  0.0137765
Warning messages:
1: In estimate.lvm(lvm(Y ~ X1 + X2 + X3 + eta), data = dd, control = list(trace = 1)) :
  Lack of convergence. Increase number of iteration or change starting values.
2: In sqrt(diag(asVar)) : NaNs produced
```

9 Miscellaneous

9.1 Profile code

```
library(lava)
m <- lvm(Y ~ X + G)
FUN <- function(n){
  d <- lava::sim(m, n = n)
  estimate(m,d)
}
profvis::profvis(FUN(n = 500))
```

#+RESULTS[<2019-06-27 to 09:37> a0d5077301cabedce939985d9ce7fb7eb9072578]:

9.2 Debug

To not show too many lines before debug:

```
options(deparse.max.lines = 200)
```

To show at which line in the program an error occurred:

```
options(error = function() revTraceback(max.lines = 5))
```

9.3 Find all function names from a package

```
r <- unclass(lsf.str(envir = asNamespace("lava"), all = T))  
r[grep("coef", r)]
```

```
[1] "coef.CrossValidated"  "coef.effects"        "coef.estimate"       "coef.estimate.list"  
[5] "coef.lvm"             "coef.lvm.mixture"    "coef.lvmfit"         "coef.multigroup"  
[9] "coef.multigroupfit"   "coef.multinomial"    "coef.ordreg"         "coef.pcor"  
[13] "coef.summary.estimate" "coef.summary.lvmfit"  "coef.twostageCV"     "coef.zibreg"  
[17] "describecoef"         "excoef"              "stdcoef"
```

9.4 Install development version of R

<https://cran.r-project.org/bin/windows/base/rdevel.html>

9.5 Install suggested packages

```
char.package <- utils::packageDescription("butils", fields = "Suggests")
vec.package <- unlist(strsplit(gsub("[[:blank:]]", "", charPackage), split = ","))
install.packages(vec.package)
```

9.6 R version

```
sessionInfo()
```

R version 3.5.1 (2018-07-02)

Platform: x86_64-w64-mingw32/x64 (64-bit)

Running under: Windows 7 x64 (build 7601) Service Pack 1

Matrix products: default

locale:

[1] LC_COLLATE=Danish_Denmark.1252 LC_CTYPE=Danish_Denmark.1252 LC_MONETARY=Danish_Denmark.1252
[4] LC_NUMERIC=C LC_TIME=Danish_Denmark.1252

attached base packages:

[1] parallel stats graphics grDevices utils datasets methods base

other attached packages:

[1] ggpubr_0.2 magrittr_1.5 officer_0.3.2 Publish_2018.04.17 lava_1.6.5
[6] doSNOW_1.0.16 snow_0.4-3 iterators_1.0.10 foreach_1.4.4 pbapply_1.3-4
[11] multcomp_1.4-8 TH.data_1.0-9 MASS_7.3-50 mvtnorm_1.0-8 survival_2.44-1.1
[16] prodlim_2018.04.18 car_3.0-2 carData_3.0-2 ggplot2_3.1.0 data.table_1.12.0

loaded via a namespace (and not attached):

[1] Rcpp_1.0.1 lattice_0.20-35 visNetwork_2.0.4 zoo_1.8-4 assertthat_0.2.0
[6] digest_0.6.17 R6_2.3.0 cellranger_1.1.0 plyr_1.8.4 pillar_1.3.1
[11] rlang_0.3.1 lazyeval_0.2.1 curl_3.2 readxl_1.1.0 uuid_0.1-2
[16] Matrix_1.2-14 labeling_0.3 splines_3.5.1 stringr_1.3.1 foreign_0.8-70
[21] htmlwidgets_1.3 munsell_0.5.0 compiler_3.5.1 pkgconfig_2.0.2 base64enc_0.1-3
[26] htmltools_0.3.6 tidyselect_0.2.5 gridExtra_2.3 tibble_2.0.1 rio_0.5.10
[31] codetools_0.2-15 viridisLite_0.3.0 crayon_1.3.4 dplyr_0.7.8 withr_2.1.2
[36] grid_3.5.1 jsonlite_1.5 gtable_0.2.0 scales_1.0.0 zip_1.0.0
[41] stringi_1.2.4 ggthemes_4.0.1 bindrcpp_0.2.2 xml2_1.2.0 sandwich_2.5-0
[46] cowplot_0.9.3 openxlsx_4.1.0 tools_3.5.1 forcats_0.3.0 glue_1.3.0
[51] purrr_0.3.0 hms_0.4.2 yaml_2.2.0 abind_1.4-5 colorspace_1.3-2
[56] bindr_0.1.1 haven_1.1.2

9.7 CRAN check rocker

<https://www.brodieg.com/2018/04/06/adventures-in-r-and-compiled-code/>

```
docker run -rm -ti -v $(pwd):/mydir wch1/r-debug RDvalgrind -e "install.packages('/mydir/fansi0.2.1.tar.gz')"  
RDvalgrind -d valgrind # and run tests  
RDcsan  
wget -O - https://github.com/bozenne/BuyseTest/tarball/master | tar xz
```