Data vizualisation with **R** (session 4)

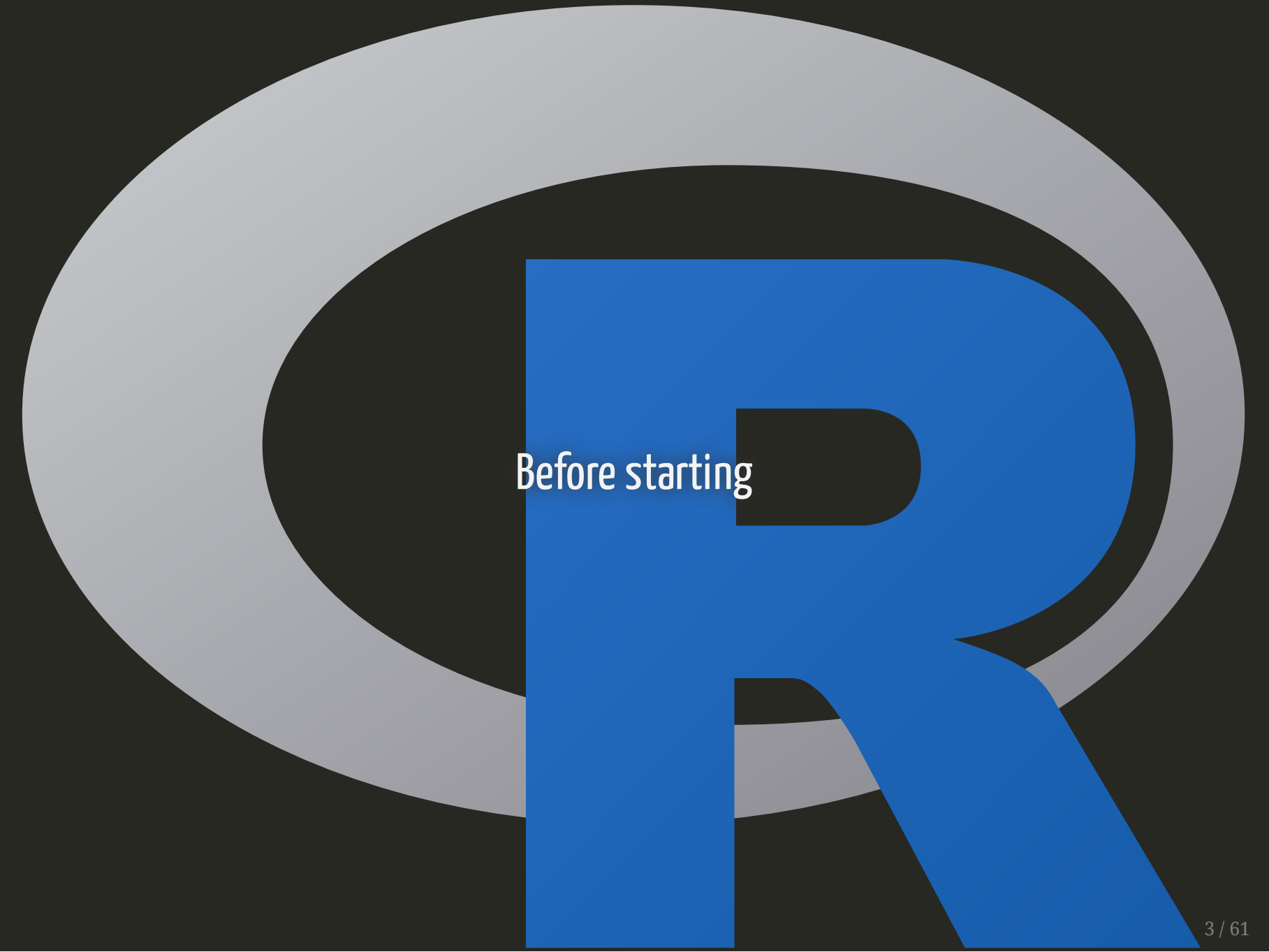
M2 Statistics and Econometrics

Thibault Laurent

Toulouse School of Economics, CNRS

Last update: 2023-10-04

Table of contents


Before starting

Packages and software versions

This document has been compiled under this version: R version 4.3.1 (2023-06-16). Install the following packages and load them:

```
install.packages(c("broom", "gapminder", "ggcorrplot", "ggiraph", "ggpol", "ggridges", "ggtern", "kableExtra'
```

```
require("gapminder") # use data from https://www.gapminder.org/
require("ggcorrplot") # coorelation matrix with ggplot2 style
require("ggiraph") # interactive plot
require("ggridges")    # multi density plot
require("ggpol")       # ggplot2 extensions
require("ggtern")      # ternary diagram
require("kableExtra") # include nice Table in Markdown document
require("plotly")      # interactive graphics
require("RColorBrewer") # color palettes
require("stargazer")    # inlude latex/html regression table
require("ggplot2")      # data visualization
require("visreg")     # visualization of regression models
require("shiny")     # shiny App
require("survival")   # survival data analysis
require("survminer") # visualization of survival data analysis
require("vcd")        # visualization of categorical data
```


1. Nice tools for writting a report

Include tables in a **R** Markdown document

- **R** Markdown is very useful for writing reports which contain text, codes and results of codes.
- Use function `kable()` in `knitr` package permits to print a table

````
```{r, results = 'asis'}
knitr::kable(head(iris[1:5, ]))
```
````

```
kableExtra::kbl(head(iris[1:5, ]))
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

Another example from kableExtra

```
vs_dt <- iris[1:5, ]
vs_dt[1:4] <- lapply(vs_dt[1:4], function(x) {
    cell_spec(x, bold = T, color = spec_color(x, end = 0.9),
              font_size = spec_font_size(x))
})
vs_dt[5] <- cell_spec(vs_dt[[5]], color = "white", bold = T,
    background = spec_color(1:5, end = 0.9, option = "A", direction = -1))
kbl(vs_dt, escape = F, align = "c") %>% kable_classic("striped", full_width = F)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa

More informations here.

Print the result of a linear regression model

One can use function `kbl()` after applying function `tidy()` on the `lm` class of object

```
lm(Sepal.Length ~ Species, data = iris) %>%
  broom::tidy() %>%
  kbl()
```

term	estimate	std.error	statistic	p.value
(Intercept)	5.006	0.0728022	68.761639	0
Speciesversicolor	0.930	0.1029579	9.032819	0
Speciesvirginica	1.582	0.1029579	15.365506	0

More informations here.

Compare different regression models

`stargazer()` from `stargazer` permits to print several regression table

```
mod_1  <- lm(Sepal.Length ~ Petal.Length, data = ir
mod_2  <- lm(Sepal.Length ~ Petal.Width, data = iri
stargazer::stargazer(mod_1, mod_2, type = "html",
  title = "Regression results", header = F)
```

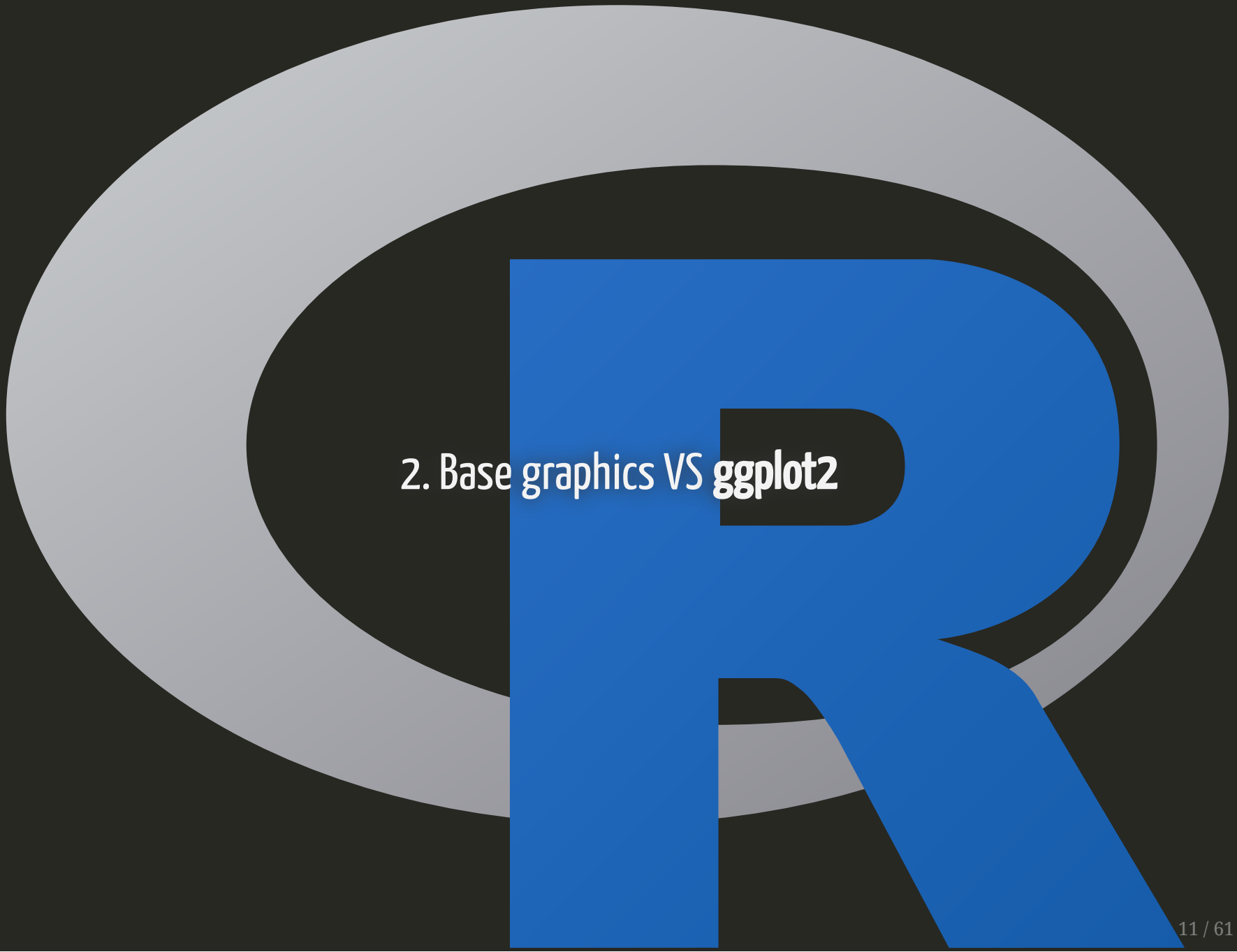
Regression results

	Dependent variable:	
	Sepal.Length	
	(1)	(2)
Petal.Length	0.409***	
	(0.019)	
Petal.Width		0.889***
		(0.051)
Constant	4.307***	4.778***
	(0.078)	(0.073)
Observations	150	150
R^2	0.760	0.669
Adjusted R^2	0.758	0.667
Residual Std. Error (df = 148)	0.407	0.478
F Statistic (df = 1; 148)	468.550***	299.167***
Note:	*p<0.1; **p<0.05; ***p<0.01	

👩‍🎓 Training

Exercise 4.1

- Insert in a Markdown document the correlation matrix of the numeric variables of the `iris` data.
- Insert the table of results of a regression analysis of the `iris` data


2. Base graphics VS ggplot2

Base graphics VS **ggplot2**

There are two main approaches for doing graphics

I. Base functions programming.

II. Use package `ggplot2` which uses its own syntax.

Comparisons of the two approaches:

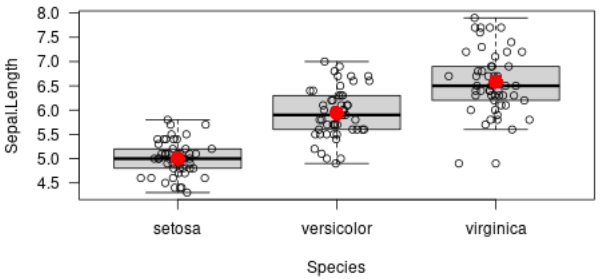
- Most of the time, the two approaches are not complementary.
- Objective of this section: obtain the same graphics by using the two approaches.
- Some references:
  - Comparing ggplot2 and R Base Graphics
  - Graphics with R (in French)
  - Univariate Graphs
- Package `esquisse` allows to obtain `ggplot2` codes using an interface

Remark: actually, there is a third approach by using `lattice` package (see this link for more informations), but we will not present this solution

Base graphics syntax

1. (optionnal): use `par()` function for defining the margin, the size of the labels, etc.
2. Open a device with a first-level plot function like `plot()`, `hist()`, `barplot()`, `boxplot()`, etc.
3. Customize the plot by using functions like `points()`, `lines()`, `text()`, `legend()`

```
op <- par(oma = c(1, 1, 0, 1), las = 1)
boxplot(Sepal.Length ~ Species, data = iris)
points(as.numeric(iris$Species) + rnorm(150, 0, 0.1), iris$Sepal.Length)
points(c(1, 2, 3), tapply(iris$Sepal.Length, iris$Species, mean),
 col = "red", pch = 16, cex = 2)
```

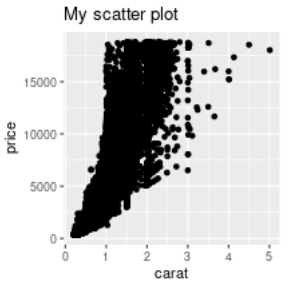



```
par(op)
```

ggplot2 syntax

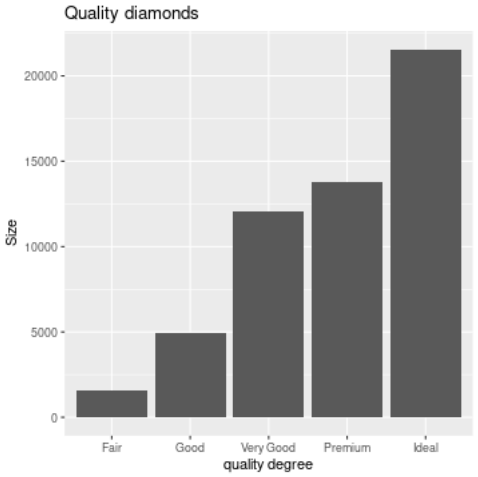
1. use `ggplot()` function for specifying:
   - data frame containing the data to be plotted
   - the mapping of the variables to visual properties of the graph within the `aes()` function.
2. Add the operator + followed by the geometric objects `Geoms` (points, lines, bars, etc.) that can be placed on a graph using functions that look `geom_xxx()`
3. Add the operator + followed by the `scales` control which define how variables are mapped to the visual characteristics of the plot. Use functions which look `scale_xxx()`
4. Add the operator + followed by the `facets` which reproduce a graph for each level of a given variable. Use functions that look `facet_xxx()`

```
data("diamonds")
ggplot(diamonds,
       aes(x = carat,
           y = price)) +
  geom_point() +
  ggtitle("My scatter plot")
```



Barplot with **ggplot2**

```
ggplot(diamonds) +
  aes(x = cut) +
  geom_bar(stat = "count") +
  xlab("quality degree") +
  ylab("Size") +
  ggtitle("Quality diamonds")
```



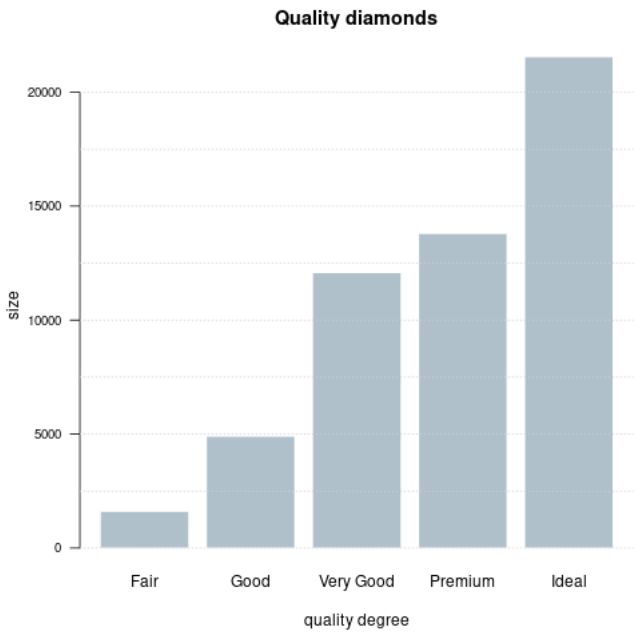
Barplot with graphic base

We first need to define the contingency table (`table` object)

```
tab_cut <- table(diamonds$cut)
```

We then use `barplot()` function and `abline()` to get the horizontal lines like with `ggplot2`

```
par(las = 1)
barplot(tab_cut,
    col = "#AFC0CB",
    border = FALSE,
    main = "Quality diamonds",
    xlab = "quality degree",
    ylab = "size",
    cex.axis = 0.8)
abline(h = seq(0, 20000, by = 2500),
  col = "lightgray", lty = "dotted")
```



Plotting a time serie

We create a vector of numeric values:

```
serie <- c(161.31, 154.00, 161.94, 160.23, 173.20, 170.21,
   163.97, 161.70, 144.91, 145.31, 140.50, 139.58, 135.60,
   124.40, 132.24, 150.51, 146.56, 153.00, 151.78, 160.65,
   158.32, 158.06, 153.50, 161.95, 167.00, 175.00, 180.48,
   173.82, 160.05, 152.80, 153.58, 145.00, 142.98, 145.35)
```

We create a vector of type `Date`:

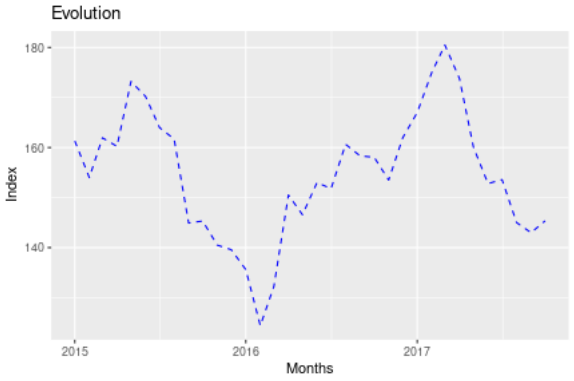
```
date_serie <- seq(as.Date("2015/1/1"), by = "month", length.out = 34)
```

For using `ggplot2`, user needs to create a `data.frame` (or `tibble`) object (which is not the case with the **R** base code which allows the use of vectors in most of the functions):

```
serie_df <- data.frame(date_serie = date_serie, serie = serie)
```

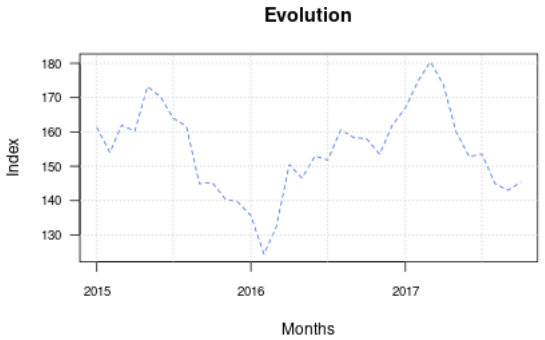
Plot a time serie with **ggplot2**

```
ggplot(serie_df) +
  aes(x = date_serie, y = serie) +
  geom_line(linetype = 2, colour = "blue")  +
  xlab("Months") +
  ylab("Index") +
  ggtitle("Evolution")
```



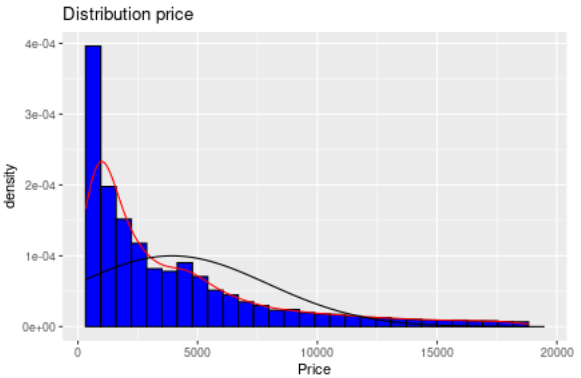
Plot a time serie with R base code

```
par(las = 1)
plot(serie ~ date_serie, data = serie_df, type = "l", col = "royalblue", lty = 2,
  main = "Evolution", xlab = "Months", ylab = "Index", cex.axis = 0.8)
abline(h = seq(130, 180, by = 10), v = date_serie[seq(1, 32, 6)], col = "lightgray", lty = "dotted")
```



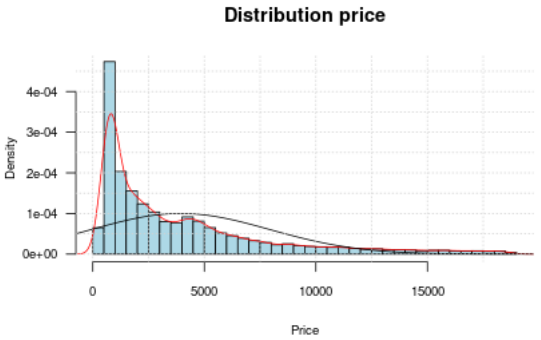
Histogram and density plot with **ggplot2**

```
ggplot(diamonds) +
 aes(x = price) +
  geom_histogram(aes(y = after_stat(density)), fill = "blue", colour = "black", bins = 30) +
  geom_density(colour = "red", adjust = 2) +
  stat_function(fun = dnorm, args = c(mean = mean(diamonds$price), sd = sd(diamonds$price))) +
  xlab("Price") +
  ggtitle("Distribution price")
```



Histogram and density plot with with **R** base code

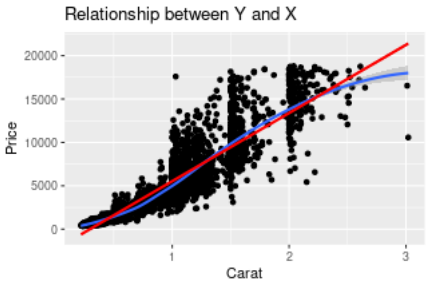
```
par(las = 1, cex.axis = 0.8, cex.lab = 0.8)
hist(diamonds$price, freq = F, col = "lightblue", nclass = 30, xlab = "Price",  main = "Distribution price")
lines(density(diamonds$price),col = "red")
x_seq <- seq(-1000, 20000, by = 100)
lines(x_seq, dnorm(x_seq, mean(diamonds$price), sd(diamonds$price)))
abline(h = seq(0, 0.0005, by = 0.00005), v = seq(0, 20000, by = 2500), col = "lightgray", lty = "dotted")
```



Scatter plot with **ggplot2**

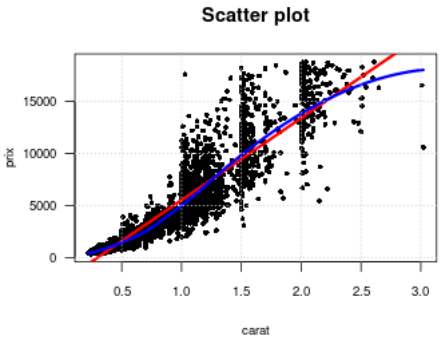
We select first a sample of the observations:

```
set.seed(123) # fix a seed
diam_ech <- diamonds[sample(nrow(diamonds), 5000, replace = F), ]
ggplot(diam_ech) +
  aes(x = carat, y = price) +
  geom_point() +
  geom_smooth(method = "loess") +  # Non parametric regression
  geom_smooth(method = "lm", col = "red") + # Linear regression
  xlab("Carat") +
  ylab("Price") +
  ggtitle("Relationship between Y and X")
```



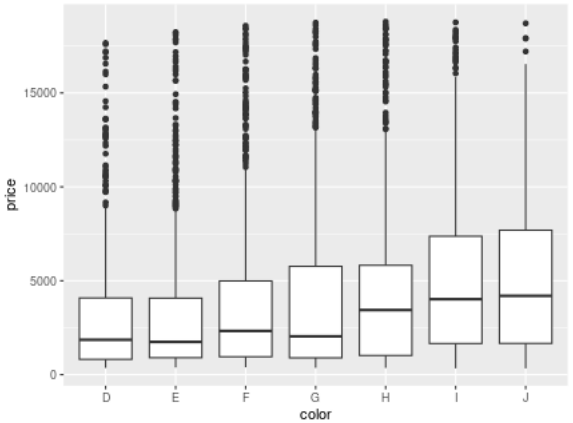
Scatter plot with **R** base code

```
par(las = 1, cex.axis = 0.8, cex.lab = 0.8)
plot(price ~ carat, data = diam_ech, pch = 16, cex = 0.7, xlab = "carat",
   ylab= "prix", main = "Scatter plot")
abline(lm(price ~ carat, data = diam_ech), col = "red", lwd = 3)
# values to predict
x_carat <- seq(0, 4.5, 0.01)
lines(x_carat, predict(loess(price ~ carat, data = diam_ech), data.frame(carat = x_carat)),
      col = "blue", lwd = 3)
abline(h = seq(0, 20000, by = 5000), v = seq(0, 4, by = 0.5), col = "lightgray", lty = "dotted")
```



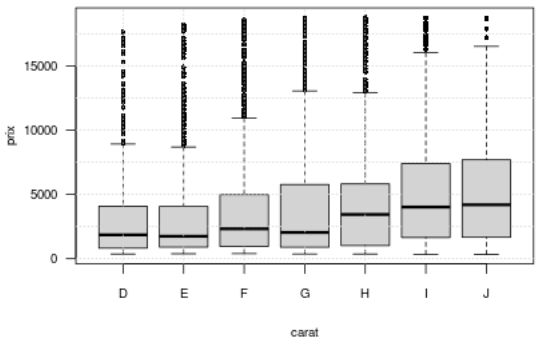
Parallel boxplot with **ggplot2**

```
ggplot(diam_ech) +
  aes(x = color, y = price) +
  geom_boxplot()
```



Parallel boxplot with **R** base code

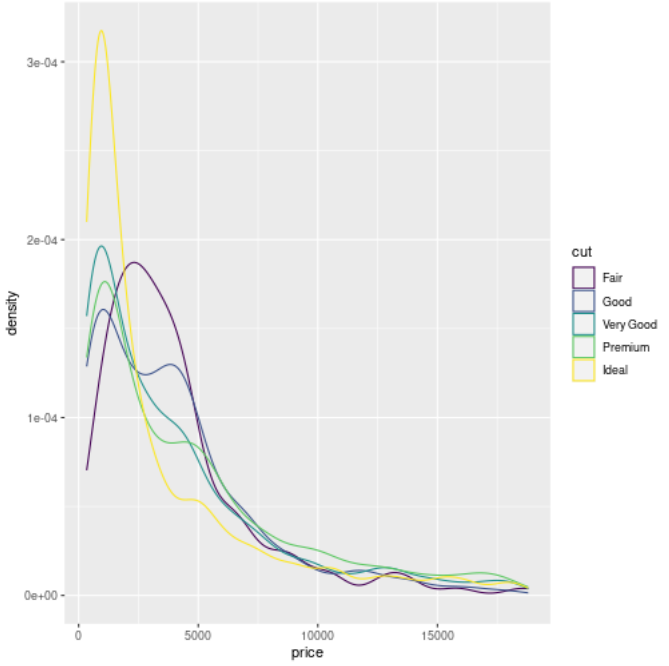
```
par(las = 1, cex.axis = 0.8, cex.lab = 0.8)
boxplot(price ~ color, data = diam_ech, pch = 16, cex = 0.7, xlab = "carat", ylab= "prix")
abline(h = seq(0, 20000, by = 2500), v = seq(0, 5, by = 1), col = "lightgray", lty = "dotted")
```



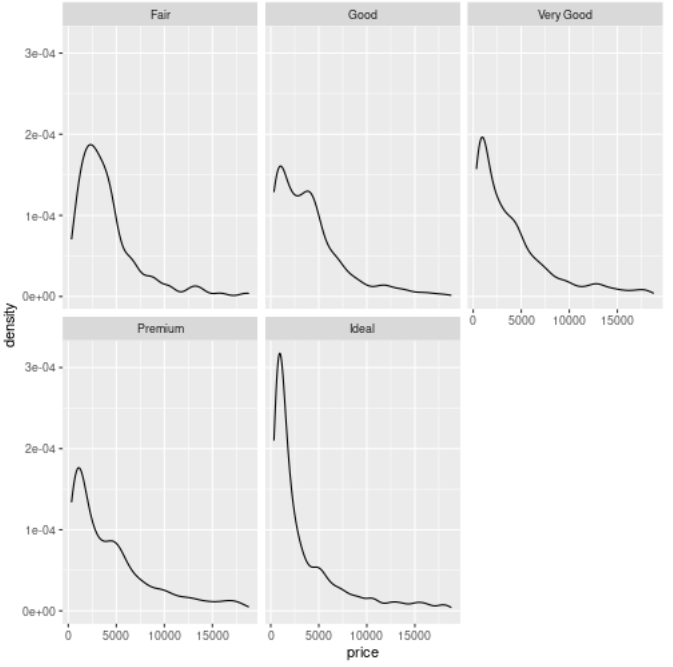
What is conditionnal graphics?

Objective: it consists in representing the distributions on sub-populations (for example, a population can be splitted with respect to a qualitative variable). It can be done differently:

Represent all the sub-distributions in the same graphic



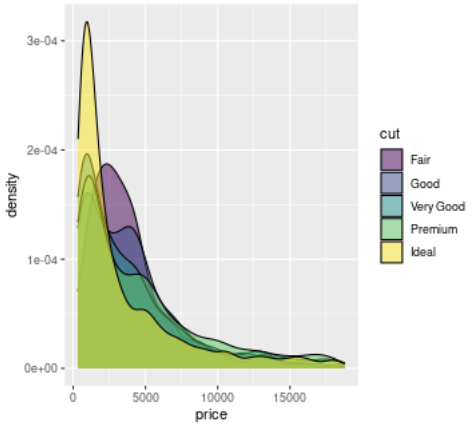
Repeat the same graphic for each sub-sample (can be called `facets`).



Conditionnal density plot with **ggplot2**

It is very easy to plot conditionnal graphics with **ggplot2**. Note that these 3 rows of code represent actually a lot of **R** base codes. Initial graphical parameters are already chosen.

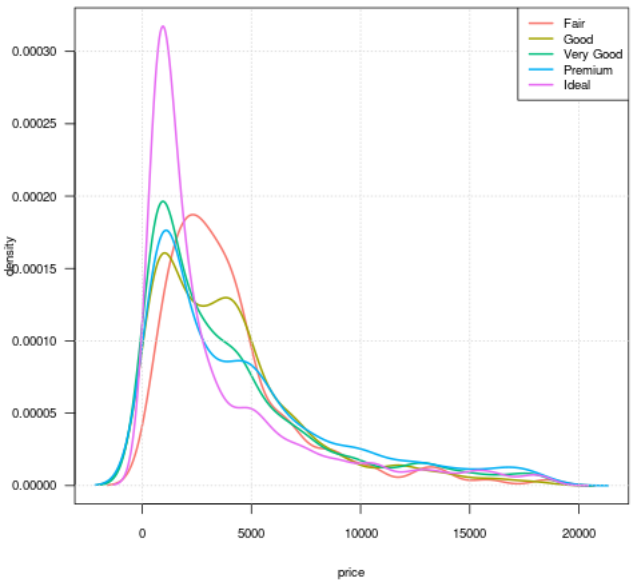
```
ggplot(diam_ech) +
  aes(x = price, fill = cut) +
  geom_density(alpha = 0.5)
```



Conditionnal density plot with **R** base code

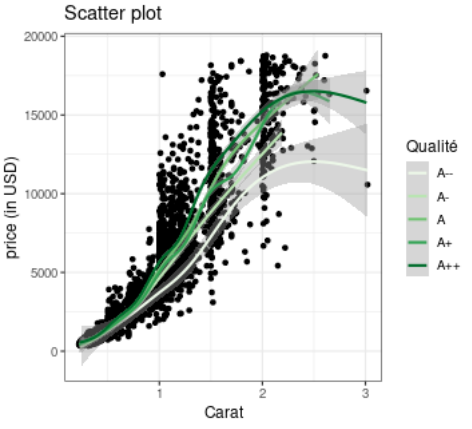
It can be boring to make conditional grahics with **R** base code but ... it is still possible to make some elegant graphics and maybe easier to change optional parameters.

```
list_price <- split(diam_ech$price, diam_ech$cut)
list_density <- lapply(list_price, density)
par(las = 1, cex.axis = 0.8, cex.lab = 0.8)
plot(range(unlist(lapply(
  list_density, function(l) range(l$x)))),
  range(unlist(lapply(list_density,
                function(l) range(l$y)))),
  type = "n", xlab = "price", ylab = "density")
col_pal <- c("#F8766D", "#A3A500", "#00BF7D", "#00E
dont_print <- mapply(lines, list_density, col = col
abline(h = seq(0, 4*10^(-4), by = 10^(-4)),
       v = seq(0, 25000, by = 5000),
       col = "lightgray", lty = "dotted")
legend("topright", legend = names(list_density),
       col = col_pal, lwd = 2, cex = 0.8)
```



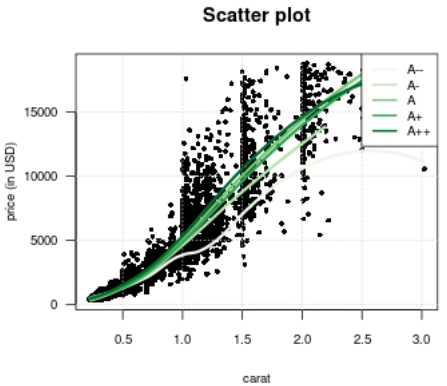
Conditionnal scatter plot with **ggplot2**

```
ggplot(diam_ech) +
  aes(x = carat, y = price) +
  geom_point() +
  geom_smooth(aes(colour = cut)) +
  theme_bw() +
  xlab("Carat") +
  ylab("price (in USD)") +
  ggtitle("Scatter plot") +
  scale_colour_brewer(name = "Qualité",
  labels = c("A--", "A-", "A", "A+", "A++"),
  palette = "Greens")
```



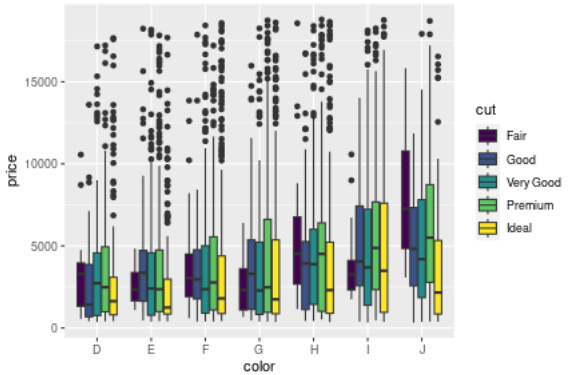
Conditionnal scatter plot with **R** base code

```
par(las = 1, cex.axis = 0.8, cex.lab = 0.8)
plot(price ~ carat, data = diam_ech, pch = 16, cex
 ylab = "price (in USD)", main = "Scatter plot")
abline(h = seq(0, 20000, by = 5000), v = seq(0, 4,
       col = "lightgray", lty = "dotted")
list_df <- split(diam_ech, diam_ech$cut)
x_carat <- seq(0, 4.5, 0.01)
list_loess <- lapply(list_df,
  function(obj) predict(loess(price ~ carat,
    data = obj), data.frame(carat = x_carat)))
require("RColorBrewer")
col_pal <- brewer.pal(length(list_price), "Greens")
dont_print <- mapply(lines, rep(list(x_carat), 5),
       col = col_pal, lwd = 3)
legend("topright", legend = c("A--", "A-", "A", "A+
       col = col_pal, lwd = 2, cex = 0.8)
```



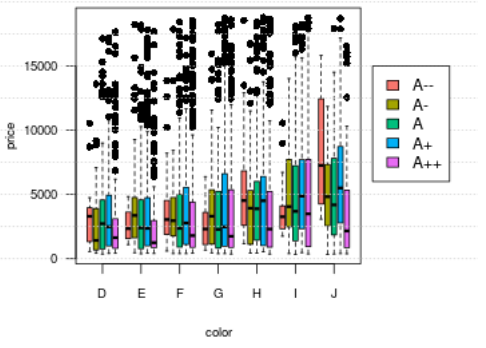
Conditionnal parallel boxplot with **ggplot2**

```
ggplot(diam_ech) +
  aes(x = color, y = price, fill = cut) +
  geom_boxplot()
```



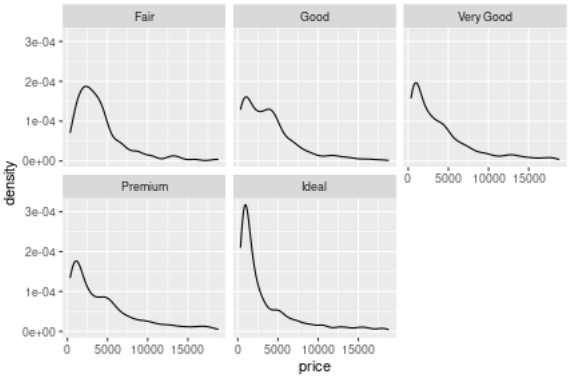
Conditionnal parallel boxplot with **R** base code

```
col_pal <- c("#F8766D", "#A3A500", "#00BF7D", "#00E
par(las = 1, cex.axis = 0.8, cex.lab = 0.8,
    xpd = T, mar = par()$mar + c(0, 0, 0, 4))
boxplot(price ~ cut + color, data = diam_ech, xlab
 ylab = "price", at = c(1:5, 7:11, 13:17, 19:23, 25
 col = rep(col_pal, 7), pch = 16, xaxt = "n")
axis(1, at = c(3, 9, 15, 21, 27, 33, 39),
     labels = c("D", "E", "F", "G", "H", "I", "J"))
abline(h = seq(0, 20000, by = 2500), col = "lightgr
legend(45, 15000, legend = c("A--", "A-", "A", "A+'
       fill = col_pal)
```



Facets with **ggplot2**

```
ggplot(diam_ech) +
  aes(x = price) +
  geom_density() +
  facet_wrap(~ cut)
```



👩‍🎓 Training

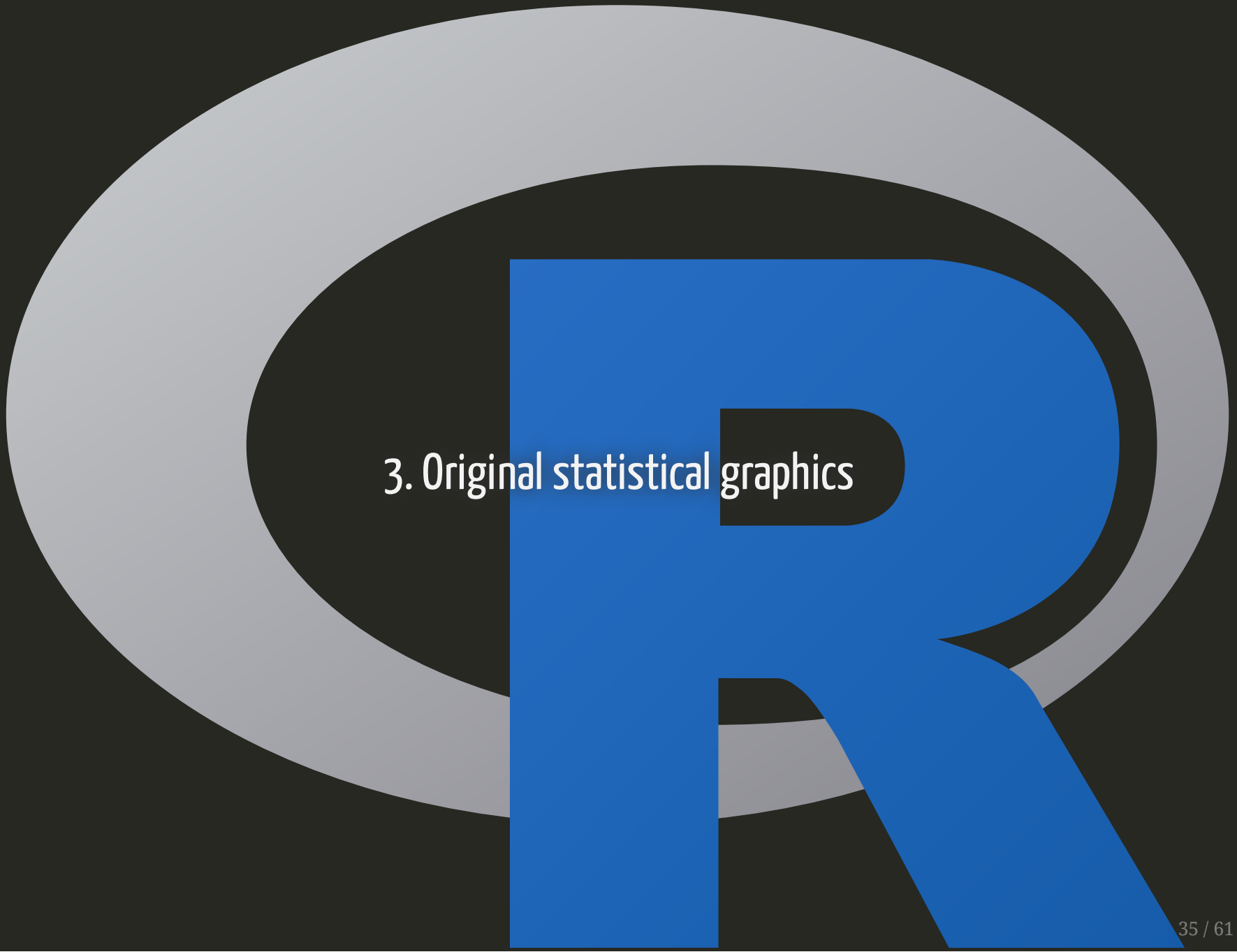
Exercise 4.2

- Find the code which allows to obtain with **ggplot2** this figure:

```
op <- par(oma = c(1, 1, 0, 1), las = 1)
boxplot(Sepal.Length ~ Species, data = iris)
points(as.numeric(iris$Species) + rnorm(150, 0, 0.1), iris$Sepal.Length)
points(c(1, 2, 3), tapply(iris$Sepal.Length, iris$Species, mean),
 col = "red", pch = 16, cex = 2)
par(op)
```

- Find the code in **R** base code which allows to obtain this figure:

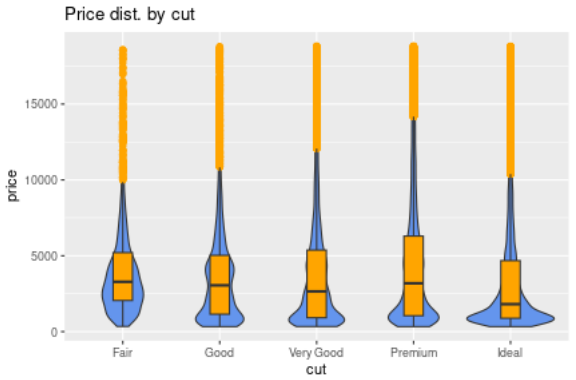
```
data("diamonds")
ggplot(diamonds,
       aes(x = carat,
           y = price)) +
  geom_point() +
  ggtitle("My scatter plot")
```


3. Original statistical graphics

Violin plots

It is the combination of density plot (in blue) and boxplot

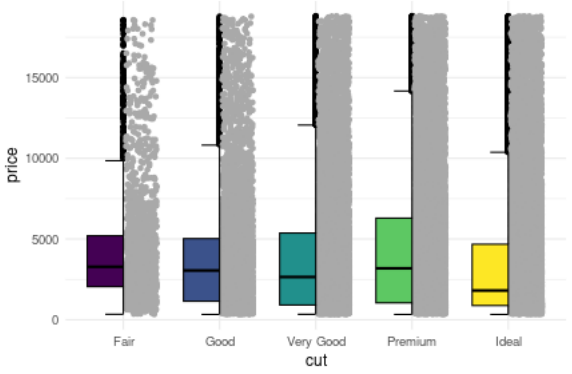
```
ggplot(diamonds, aes(x = cut, y = price)) +
  geom_violin(fill = "cornflowerblue") +
  geom_boxplot(width = .2, fill = "orange",
  outlier.color = "orange", outlier.size = 2) +
  labs(title = "Price dist. by cut")
```



Combining jitter and boxplot

It is the combination of boxplot and jitter

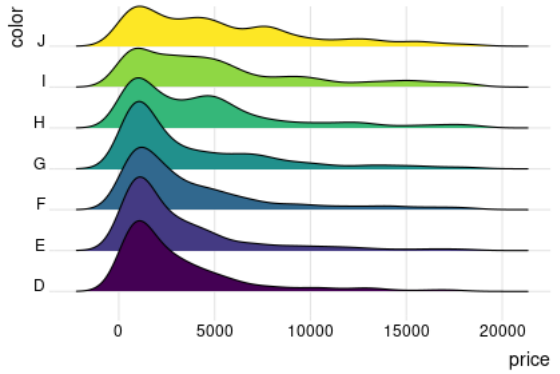
```
ggplot(diamonds, aes(x = cut, y = price, fill = cut)) +
  geom_boxjitter(color = "black", jitter.color = "darkgrey", errorbar.draw = TRUE) +
  theme_minimal() +
  theme(legend.position = "none")
```



Ridgeline plot

It represents the density plot on sub-sample

```
ggplot(diam_ech) +
  aes(x = price, y = color, fill = color) +
  geom_density_ridges() +
  theme_ridges() +
  labs("Price by levels color") +
  theme(legend.position = "none")
```



Comparing the means of sub-sample

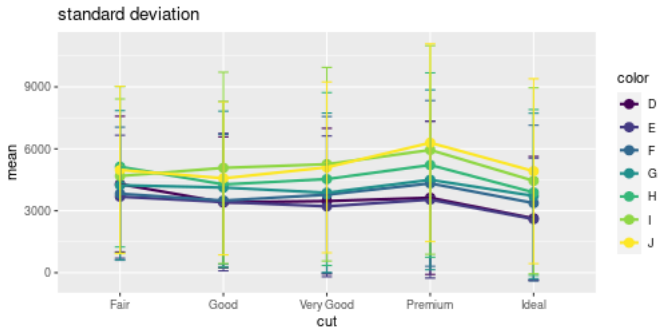
We first compute in a `data.frame` the mean, the standard deviation, the standard error and the confidence interval of a numeric variable with respect to 2 categorical variables

```
library(dplyr)
plotdata <- diamonds %>%
  group_by(color, cut) %>%
  summarize(n = n(),
         mean = mean(price),
         sd = sd(price),
         se = sd / sqrt(n),
         ci = qt(0.975, df = n - 1) * sd / sqrt(n))
```

Represent error bars (1)

⚠️ we get different results with respect to the choosen criteria. Here the SD:

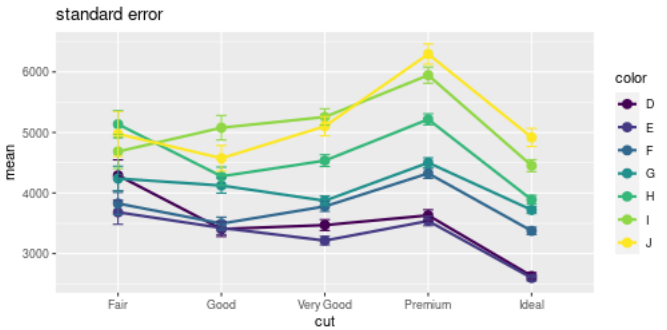
```
ggplot(plotdata) +
 aes(x = cut, y = mean, group = color, color = color) +
  geom_point(size = 3) +
  geom_line(linewidth = 1) +
  geom_errorbar(aes(ymin  = mean - sd, ymax = mean + sd), width = .1) +
  labs(title = "standard deviation")
```



Represent error bars (2)

⚠️ we get different results with respect to the choosen criteria. Here the SE:

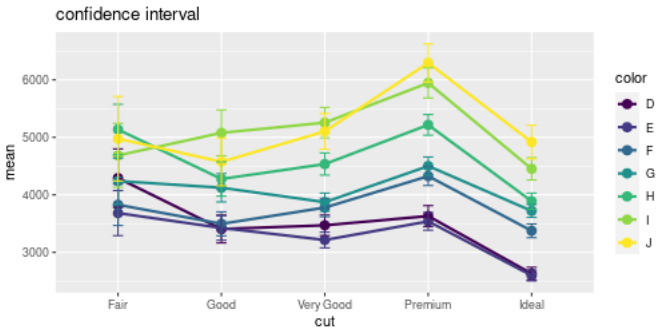
```
ggplot(plotdata) +
 aes(x = cut, y = mean, group = color, color = color) +
  geom_point(size = 3) +
  geom_line(linewidth = 1) +
  geom_errorbar(aes(ymin  = mean - se, ymax = mean + se), width = .1)  +
  labs(title = "standard error")
```



Represent error bars (3)

⚠️ we get different results with respect to the choosen criteria. Here the CI:

```
ggplot(plotdata) +
 aes(x = cut, y = mean, group = color, color = color) +
  geom_point(size = 3) +
  geom_line(linewidth = 1) +
  geom_errorbar(aes(ymin  = mean - ci, ymax = mean + ci), width = .1)  +
  labs(title = "confidence interval")
```

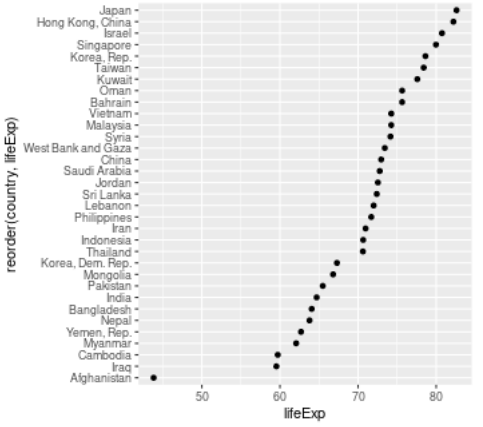


Cleveland dot chart

Compare the 2007 life expectancy for Asian country using the gapminder dataset.

```
data(gapminder, package = "gapminder")

library(dplyr)
plotdata <- gapminder %>%
  filter(continent == "Asia" & year == 2007)
ggplot(plotdata) +
  aes(x = lifeExp, y = reorder(country, lifeExp)) +
  geom_point()
```

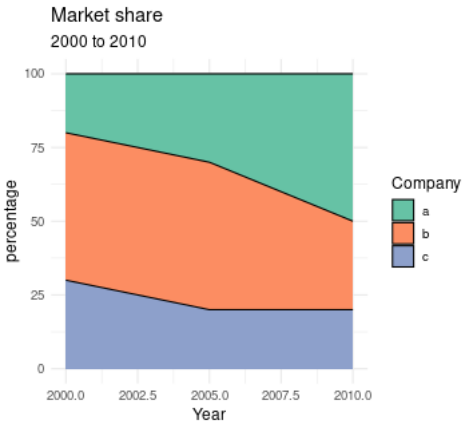


Area chart

Compare the shares of different companies across the time

```
time_chart <- data.frame(
  year = rep(as.Date(c(2000, 2005, 2010)), each =3),
  market_share = c(20, 50, 30,
                   30, 50, 20,
                   50, 30, 20),
  comp = rep(c("a", "b", "c"), 3)
)

ggplot(time_chart) +
  aes(x = year, y = market_share, fill = comp) +
  geom_area(color = "black") +
  labs(title = "Market share",
       subtitle = "2000 to 2010",
       x = "Year",
       y = "percentage",
       fill = "Company") +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()
```



Ternary diagram for compositional data

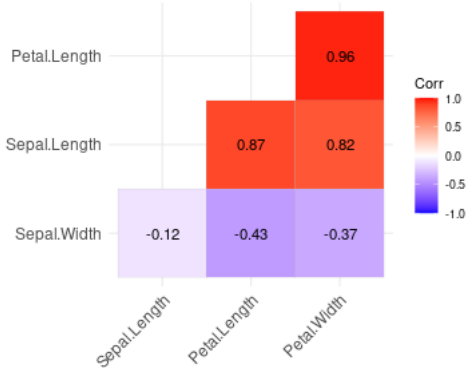
Compare the shares of different companies across the time

```
library(tidyverse)
time_chart_wide <- pivot_wider(time_chart, values_f
    names_from = comp)

library("ggtern")
ggtern(data = time_chart_wide,
    mapping = aes(x = a, y = b, z = c)) +
    geom_point(size = 1.5)
```

Correlation plot

```
r <- cor(iris[, 1:4], use = "complete.obs")
ggcorrplot(r, hc.order = TRUE, type = "lower", lab = TRUE)
```

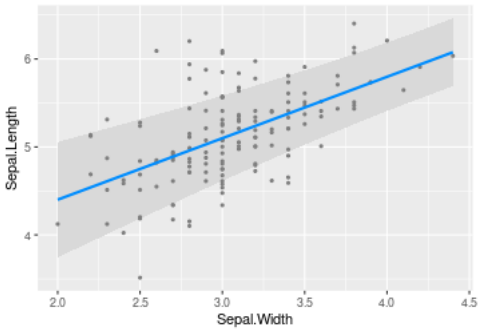


Linear regression (1)

The `visreg()` function takes (1) the model and (2) the variable of interest and plots the conditional relationship, controlling for the other variables (see this link for more informations).

Example on a numeric variable:

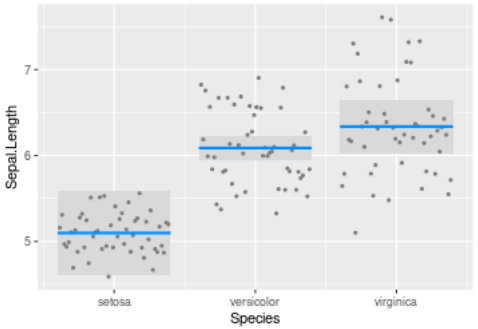
```
res_lm <- lm(Sepal.Length ~ Sepal.Width + Petal.Width + Species, data = iris)
visreg(res_lm, "Sepal.Width", gg = TRUE)
```



Linear regression (2)

Example on a categorical variable:

```
visreg(res_lm, "Species", gg = TRUE)
```



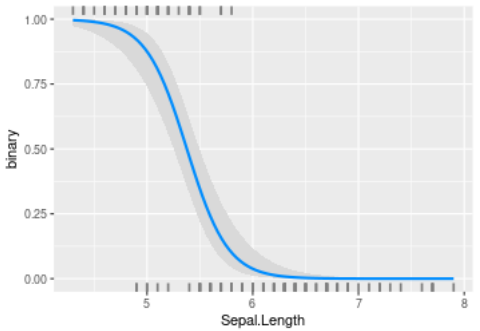
Logistic regression

Model first:

```
iris$binary <- factor(ifelse(iris$Species == "setosa", 1, 0))
res_glm <- glm(binary ~  Sepal.Length, family = binomial(link = "logit"), data = iris)
```

Plot the probability function with respect to one explanatory variable:

```
visreg(res_glm, "Sepal.Length", gg = TRUE, scale="response")
```



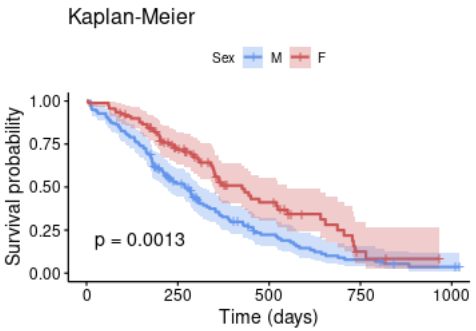
Survival plot

Model first:

```
sfit <- survfit(Surv(time, status) ~  sex, data = lung)
```

Plot the survival plot:

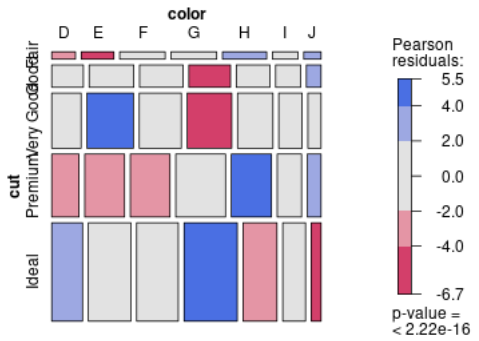
```
ggsurvplot(sfit, conf.int = TRUE, pval = TRUE, legend.labs = c("M", "F"), legend.title = "Sex",
  palette = c("cornflowerblue", "indianred3"),title = "Kaplan-Meier", xlab = "Time (days)")
```



Mosaic plot

The mosaic plot permits to appreciate the link between two categorical variables:

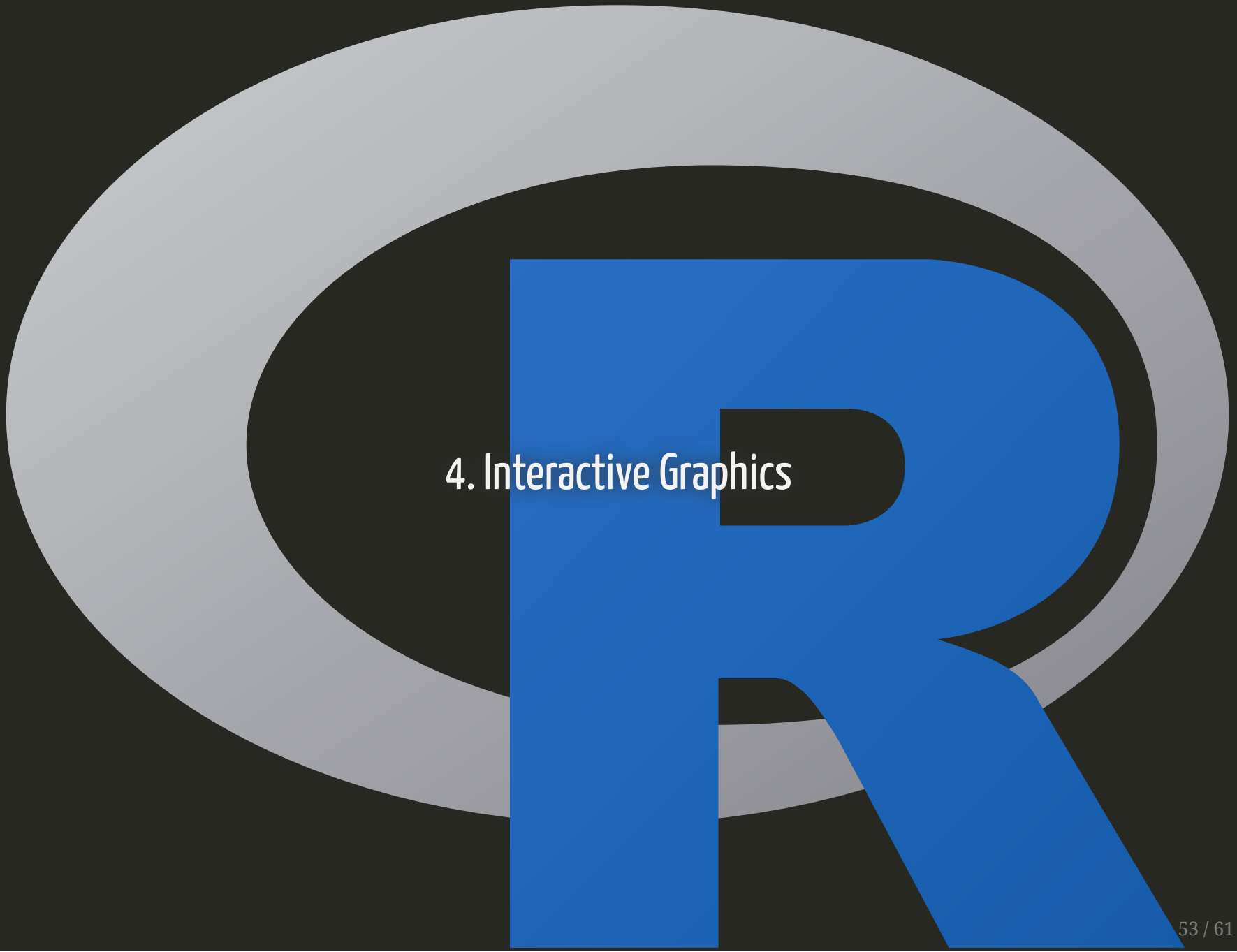
```
tab <- xtabs(~cut + color, diamonds)
mosaic(tab, shade = TRUE, legend = TRUE)
```



👩‍🎓 Training

Exercise 4.3

- On the `lung` data used previously, make a mosaic plot between `status` and `sex` variable.
- On the `lung` data, make a ridge plot of variable `age` with respect to `status`.
- Make a correlation plot of variables `ph.karno`, `pat.karno`, `meal.cal`, `wt.loss` in the `lung` data.


4. Interactive Graphics

plotly package (1)

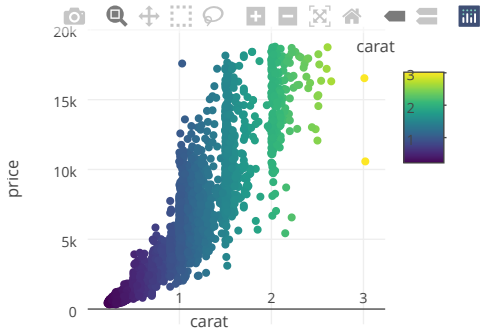
- Interactive graphics with respect to library **plotly.js** (in **JavaScript**)
- More informations here
- Syntax:
  - 1st argument: the `data.frame`
  - argument `x=` gives the name of the x variable; argument `y=` gives the name of the y variable;
  - argument `color=` gives the name of the conditionnal variable
  - argument `type=` gives the type of graphic

```
plot_ly(diam_ech, x = ~price, color = ~cut, type = "box")
```

plotly package (2)

Another example with a scatterplot:

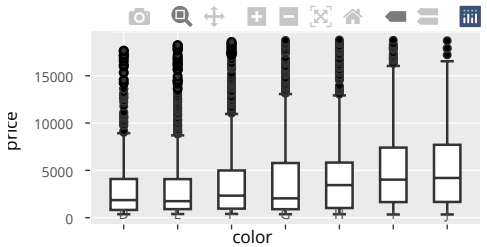
```
plot_ly(diam_ech, x = ~carat, y = ~price, type = "scatter",
    mode = "markers", hoverinfo = 'text',
    text = ~paste('Carat: ', carat, '\n Price: ', price, '\n Clarity: ', diam_ech$clarity),
    color = ~carat)
```



plotly with **ggplot2** style

Use function `ggplotly()` on a **ggplot2** syntax. Example:

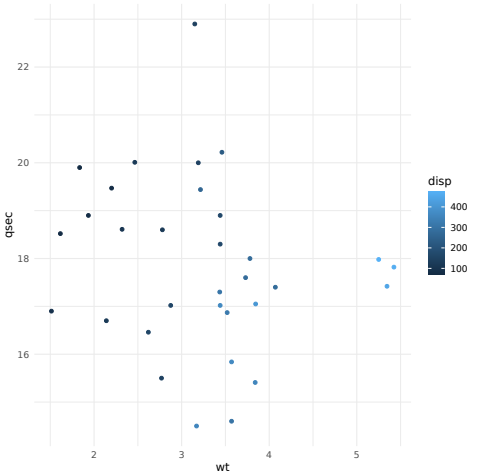
```
p <- ggplot(diam_ech) +
  aes(x = color, y = price) +
  geom_boxplot()
ggplotly(p)
```



ggiraph with ggplot2 style

A new library that allows to keep the aesthetics than origin **ggplot2**

```
library(ggiraph)
data <- mtcars
data$carname <- row.names(data)
gg_point = ggplot(data = data) +
     geom_point_interactive(aes(x = wt, y = qsec, color = disp,
     tooltip = carname, data_id = carname)) +
   theme_minimal()
girafe(ggobj = gg_point, options = list(opts_sizing(rescale = TRUE, width = .3)))
```

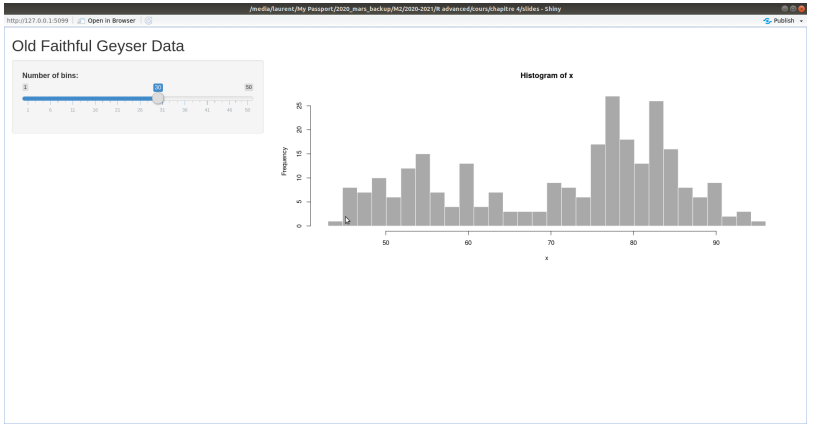


Other packages for interactive graphics: **ggvis**(see this link) and **rCharts** (see this link)

Short introduction to **Shiny**

What is shiny ?

- **Shiny**: Interactive web page
- Examples of shiny app: http://shiny.rstudio.com/gallery/
- More informations here
- Create a new shiny App: File - New File - Shiny Web App...
- Possibility to create web pages. **RStudio** proposes to host a couple of App for free. See https://www.rstudio.com/products/shiny/shiny-server/ for more informations



The two files

```
library(shiny)
# Define UI for application that draws a histogram
ui <- fluidPage(
  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                   min = 1,
                   max = 50,
                   value = 30)
       ),
  # Show a plot of the generated distribution
  mainPanel(
         plotOutput("distPlot")
      )
    )
)
```

```
# Define server logic required to draw a histogram
server <- function(input, output) {

  output$distPlot <- renderPlot({
  # generate bins based on input$bins from ui.R
  x    <- faithful[, 2]
  bins <- seq(min(x), max(x), length.out = input$bi

  # draw the histogram with the specified number of
  hist(x, breaks = bins, col = 'darkgray', border =
 })
}
```

To run the application:

```
shinyApp(ui = ui, server = server)
```

Description of the two files

- *ui.R*: indicates how the screen should be organized. For example, on the left, we print the title, the ruler, etc. that the user can eventually modify. On the right, we decide to print the graphic.
- *server.R*: contains the codes which permits to plot a graphic by using the parameters defined by users in the interface.
- When these files are opened in **RStudio**, it is then possible to run the App by clicking on the button Run App.

Description of the options

ui.R file

- *textInput()*: entering a string,
- *numericInput()*: entering a numeric,
- *selectInput()*: Create a select list input control,
- *sliderInput()*: Slider Input Widget,
- *radioButtons()*: Create radio buttons
- *fileInput()*: File Upload Control.

server.R file

- For plotting a graphic: use *renderPlot()* in **server.R** and *plotOutput()* in **ui.R**.
- For printing text: use *renderPrint()* in **server.R** and *textOutput()* in **ui.R**.
- For printing data table: use *renderDataTable()* in **server.R** and *dataTableOutput()* in **ui.R**.
- For printing images: use *renderImage()* in **server.R** and *imageOutput()* in **ui.R**.