Data visualization with R

R course,
Master 2 Statistics and Econometrics, TSE
Formation OMP

Thibault LAURENT
thibault.laurent@tse-fr.eu

Toulouse School of Economics, CNRS

October 24, 2019
Packages and software versions

```r

> devtools::install_github('hadley/ggplot2')

> require("gapminder")
> require("ggcorrplot")
> require("ggridges")
> require("ggpol")
> require("plotly")
> require("RColorBrewer")
> require("stargazer")
> require("ggplot2")
> require("visreg")
> require("survival")
> require("survminer")
> require("vcd")
> sessionInfo()

R version 3.6.1 (2019-07-05)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.6 LTS
How presenting reports and results?

- Use R Markdown for writing reports
- Use function `kable()` in `knitr` package for printing table

```r
\%
\{r, results = 'asis'}
knitr::kable(head(iris[, 2:4]))
\%
```

<table>
<thead>
<tr>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
</tr>
<tr>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
</tr>
<tr>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
</tr>
</tbody>
</table>
How presenting regression results?

- Use function `stargazer()` in `stargazer` package for printing regression table
```
```{r, results = 'asis'}
output <- lm(Sepal.Length ~ Species, data = iris)
stargazer::stargazer(output, type = "html",
                     title = "Résultat de régression", header = F)
```

**Table:** Résultat de régression

<table>
<thead>
<tr>
<th></th>
<th>Dependent variable: Sepal.Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speciesversicolor</td>
<td>0.930*** (0.103)</td>
</tr>
<tr>
<td>Speciesvirginica</td>
<td>1.582*** (0.103)</td>
</tr>
<tr>
<td>Constant</td>
<td>5.006*** (0.073)</td>
</tr>
</tbody>
</table>

Observations 150
R² 0.619
Adjusted R² 0.614
Residual Std. Error 0.515 (df = 147)
1 Introduction
2 Base Graphics VS ggplot2
3 Original statistical graphics
4 Interactive Graphics
Two main options for coding graphics

1. Use base functions like `plot()`, `hist()`, `barplot()`, `boxplot()`, etc., usually followed by functions like `lines()`, `text()`, `legend()` for customizing the plots. Use also `par()` function for defining the margin, the size of the labels, etc.

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

- Most of the time, the two approaches are not complementary.
- Objective of this section: obtain the same graphics by using the two approaches.

**Bibliography:**

- [https://flowingdata.com/2016/03/22/comparing-ggplot2-and-r-base-graphics/](https://flowingdata.com/2016/03/22/comparing-ggplot2-and-r-base-graphics/)
- [https://rkabacoff.github.io/datavis/Univariate.html](https://rkabacoff.github.io/datavis/Univariate.html)
Basic graphics syntax

1. (optional): 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()`
Basic graphics syntax: example

```r
> op <- par(oma = c(1, 1, 0, 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. + *Geoms* are the geometric objects (points, lines, bars, etc.) that can be placed on a graph using functions that start with `geom_`

3. + *scales*: Scales control how variables are mapped to the visual characteristics of the plot. Use functions which start with `scale_`

4. + *facets*: Facets reproduce a graph for each level a given variable. Use functions that start with `facet_`
ggplot2 syntax: example

```r
> data("diamonds")
> ggplot(diamonds,
    aes(x = carat,
        y = price)) +
  geom_point() +
  ggtitle("My scatter plot")
```
> ggplot(diamonds) +
  aes(x = cut) +
  geom_bar(stat = "count") +
  xlab("Degré de qualité") +
  ylab("Effectifs") +
  ggtitle("Quality diamonds")
Build a table object:

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

Graphics function:

```r
> par(las = 1)
> barplot(tab_cut,
  col = "#AFC0CB",
  border = FALSE,
  main = "Quality diamonds",
  xlab = "Degré de qualité",
  ylab = "Effectifs",
  cex.axis = 0.8)
> abline(h = seq(0, 20000, by = 2500),
  col = "lightgray", lty = "dotted")
```
Create a vector of numeric values:

```r
> 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)
```

Create a vector of type Date:

```r
> 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:

```r
> serie_df = data.frame(date_serie = date_serie, serie = serie)
```
Time serie: ggplot2

```r
> ggplot(serie_df) +
  aes(x = date_serie,
      y = serie) +
  geom_line(linetype = 2,
            colour = "blue") +
  xlab("Mois observés") +
  ylab("Indice boursier") +
  ggtitle("Evolution")
```
Time serie: graphic base

```r
> par(las = 1)
> plot(serie ~ date_serie,
      data = serie_df,
      type = "l",
      col = "royalblue",
      lty = 2,
      main = "Evolution",
      xlab = "Mois observés",
      ylab = "Indice boursier",
      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: ggplot2

```r
> ggplot(diamonds) +
  aes(x = price) +
  geom_histogram(aes(
    y = ..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("Prix observé") +
  ggtitle("Distribution price")
```
Histogram and density plot: Graphics base

```r
> par(las = 1, cex.axis = 0.8, cex.lab = 0.8)
> hist(diamonds$price, freq = F, 
col = "lightblue", nclass = 30, 
lab = "Prix du diamant", 
main = "Distribution price")
> lines(density(diamonds$price), 
col = "red")
> x_seq <- seq(-1000, 20000, by = 100)
> lines(x_seq, dnorm(x_seq, 
mean(diamonds$price), 
std(diamonds$price)))
> abline(h = seq(0, 0.0005, by = 0.00005), 
v = seq(0, 20000, by = 2500), 
col = "lightgray", lty = "dotted")
```
We select first a sample of the observations:

```r
> set.seed(123) # on fixe une graine aléatoire
> diam_ech <- diamonds[sample(nrow(diamonds), 5000, replace = F),]

> ggplot(diam_ech) +
  aes(x = carat, y = price) +
  geom_point() +
  geom_smooth(method = "loess") +
  geom_smooth(method = "lm",
             col = "red") +
  xlab("Carat") +
  ylab("Prix observé") +
  ggtitle("Lien entre deux variables quantis")
```
> 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: ggplot2

> ggplot(diam_ech) +
  aes(x = color, y = price) +
  geom_boxplot()
> 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"
Objective: adding the information related to a qualitative variable. Two options:

1. Use the same graphic by using options `fill=` or `colors=`. Example: plotting several regression lines with different colors in the same graphic with respect to the levels of one factor.

2. Repeat the same graphic as many time as the number of levels by using the functions `facets`.

Conditionnall graphics
Conditionnal density plot: ggplot2

\[
\text{ggplot}(\text{diam}_\text{ech}) + \\
\text{aes}(x = \text{price}, \text{fill} = \text{cut}) + \\
\text{geom}_\text{density}(\text{alpha} = 0.5)
\]

Remark: 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.
Conditionnal density plot: base graphics

```r
> 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 = "prix",
      ylab = "densité")
> col_pal <- c("#F8766D", "#A3A500",
              "#00BF7D", "#00B0F6", "#E76BF3")
> dont_print <- mapply(lines, list_density,
                       col = col_pal, lwd = 2)
> 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)
```
```
> 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 scatterplot: Graphic base

```r
> 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 = "price (in USD)", main = "Scatter plot")
> abline(h = seq(0, 20000, by = 5000),
  v = seq(0, 4, by = 0.5),
  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), list_loess,
  col = col_pal, lwd = 3)
> legend("topright", legend = c("A--", "A-", "A", "A+", "A++"),
  col = col_pal, lwd = 2, cex = 0.8)
```
Conditionnal parallel boxplot: ggplot2

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

<table>
<thead>
<tr>
<th>color</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fair</td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td></td>
</tr>
<tr>
<td>Very Good</td>
<td></td>
</tr>
<tr>
<td>Premium</td>
<td></td>
</tr>
<tr>
<td>Ideal</td>
<td></td>
</tr>
</tbody>
</table>
# choix des couleurs :

```r
> col_pal <- c("#F8766D", "#A3A500", "#00BF7D", "#00B0F6", "#E76BF3")
> 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 = "color",
    ylab = "price", at = c(1:5, 7:11,
    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 = "lightgray", lty = "dotted")
> legend(45, 15000, legend = c("A--", "A-", "A", "A+", "A++"),
    fill = col_pal)
```
Facets with ggplot2

> ggplot(diam_ech) +
aes(x = price) +
geom_density() +
facet_wrap(~ cut)
1 Introduction

2 Base Graphics VS ggplot2

3 Original statistical graphics

4 Interactive Graphics
Violin plots

```r
> 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

```r
> 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 plots

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

Compute the mean and standard deviation of price with respect to the variable color and cut

```r
> 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))
```
> ggplot(plotdata, aes(x = cut, y = mean, group=color, color=color)) + geom_point(size = 3) + geom_line(size = 1) + geom_errorbar(aes(ymin = mean - se, ymax = mean + se), width = .1)
Compare the 2007 life expectancy for Asian country using the gapminder dataset.

```r
> 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

```r
time_chart <- data.frame(
  year = rep(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()
```

![Area chart graphic](image-url)
Correlation plot

```r
> r <- cor(iris[, 1:4], use = "complete.obs")
> ggcorrplot(r,
  hc.order = TRUE,
  type = "lower",
  lab = TRUE)
```
The `visreg()` function takes (1) the model and (2) the variable of interest and plots the conditional relationship, controlling for the other variables.


```r
> res_lm <- lm(Sepal.Length ~ Sepal.Width + Petal.Width + Species, data = iris)
> visreg(res_lm, "Sepal.Width", gg = TRUE)
```
Example on a qualitative variable:

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

![Graph showing linear regression with species as a qualitative variable]
Logistic regression

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

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

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

> 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

> tab <- xtabs(~cut + color, diamonds)

> mosaic(tab,
    shade = TRUE,
    legend = TRUE)
Interactive graphics with respect to library plotly.js (in JavaScript)

More informations here: https://plot.ly/r/

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

Example with an interactive boxplot:

```r
> p <- plot_ly(diamonds, x = ~price, color = ~cut, type = "box")
> p
```
Example with an interactive scatterplot:

```r
> p <- plot_ly(diamonds, x = ~carat, y = ~price,
   type = "scatter",
   mode = "markers",
   hoverinfo = 'text',
   text = ~paste('Carat: ', carat,
                 'Price: ', price,
                 'Clarity: ', diamonds$clarity),
   color = ~carat)
> p
```
plotly with ggplot2 style

- Use function `ggplotly()`. Example:

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

- Other packages for interactive graphics:
  - ggvis: see https://ggvis.rstudio.com/
  - rCharts: see https://ramnathv.github.io/rCharts/
Short introduction to shiny

- **shiny**: Interactive web page
- Examples of shiny app: http://shiny.rstudio.com/gallery/
- 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/
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 in the 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.
Description of the options in the 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.