sthda.com/english

Edition 1

R Graphics Essentials for Great Data Visualization

+200 Practical Examples You Want to Known

Alboukadel Kassambara

R Graphics Essentials for Great Data Visualization

Alboukadel KASSAMBARA

Published by STHDA (`http://www.sthda.com`), Alboukadel Kassambara

Contact: Alboukadel Kassambara <alboukadel.kassambara@gmail.com>

For general information contact Alboukadel Kassambara <alboukadel.kassambara@gmail.com>.

Contents

Preface

0.1 What you will learn

Data visualization is one of the most important part of data science. Many books and courses present a catalogue of graphics but they don't teach you which charts to use according to the type of the data.

In this book, we start by presenting the key graphic systems and packages available in R, including R base graphs, lattice and ggplot2 plotting systems.

Next, we provide practical examples to create great graphics for the right data using either the ggplot2 package and extensions or the traditional R graphics.

With this book, you 'll learn:

- How to quickly create beautiful graphics using ggplot2 packages
- How to properly customize and annotate the plots
- Type of graphics for visualizing categorical and continuous variables
- How to add automatically p-values to box plots, bar plots and alternatives
- How to add marginal density plots and correlation coefficients to scatter plots
- Key methods for analyzing and visualizing multivariate data
- R functions and packages for plotting time series data
- How to combine multiple plots on one page to create production-quality figures.

0.2 Book website

The website for this book is located at : `http://www.sthda.com/english/`. It contains number of resources.

0.3 Executing the R codes from the PDF

For a single line R code, you can just copy the code from the PDF to the R console.

For a multiple-line R codes, an error is generated, sometimes, when you copy and paste directly the R code from the PDF to the R console. If this happens, a solution is to:

- Paste firstly the code in your R code editor or in your text editor
- Copy the code from your text/code editor to the R console

0.4 Colophon

This book was built with R 3.3.2 and the following packages :

```
##             name    version                         source
## 1       bookdown        0.5      Github:rstudio/bookdown
## 2    changepoint      2.2.2                          CRAN
## 3        cowplot 0.8.0.9000      Github:wilkelab/cowplot
## 4          dplyr      0.7.4                          cran
## 5     factoextra  1.0.5.999  local:kassambara/factoextra
## 6     FactoMineR       1.38                          CRAN
## 7         GGally      1.3.0                          CRAN
## 8     ggcorrplot 0.1.1.9000 Github:kassambara/ggcorrplot
## 9        ggforce      0.1.1     Github:thomasp85/ggforce
## 10     ggformula        0.6                          CRAN
## 11     ggfortify      0.4.1                          CRAN
## 12       ggpmisc     0.2.15                          CRAN
## 13        ggpubr  0.1.5.999     Github:kassambara/ggpubr
## 14       lattice    0.20-34                          CRAN
## 15         readr      1.1.1                          cran
## 16 scatterplot3d     0.3-40                          cran
## 17   strucchange      1.5-1                          CRAN
## 18         tidyr      0.7.2                          cran
```

About the author

Alboukadel Kassambara is a PhD in Bioinformatics and Cancer Biology. He works since many years on genomic data analysis and visualization (read more: http://www.alboukadel.com/).

He has work experiences in statistical and computational methods to identify prognostic and predictive biomarker signatures through integrative analysis of large-scale genomic and clinical data sets.

He created a bioinformatics web-tool named GenomicScape (www.genomicscape.com) which is an easy-to-use web tool for gene expression data analysis and visualization.

He developed also a training website on data science, named STHDA (Statistical Tools for High-throughput Data Analysis, www.sthda.com/english), which contains many tutorials on data analysis and visualization using R software and packages.

He is the author of many popular R packages for:

- multivariate data analysis (**factoextra**, http://www.sthda.com/english/rpkgs/factoextra),
- survival analysis (**survminer**, http://www.sthda.com/english/rpkgs/survminer/),
- correlation analysis (**ggcorrplot**, http://www.sthda.com/english/wiki/ggcorrplot-visualization-of-a-correlation-matrix-using-ggplot2),
- creating publication ready plots in R (**ggpubr**, http://www.sthda.com/english/rpkgs/ggpubr).

Recently, he published three books on data analysis and visualization:

1. Practical Guide to Cluster Analysis in R (https://goo.gl/yhhpXh)
2. Practical Guide To Principal Component Methods in R (https://goo.gl/d4Doz9)

Chapter 1

R Basics for Data Visualization

1.1 Introduction

R is a free and powerful statistical software for analyzing and visualizing data.

In this chapter, you'll learn:

- the basics of R programming for importing and manipulating your data:
  - filtering and ordering rows,
  - renaming and adding columns,
  - computing summary statistics
- R graphics systems and packages for data visualization:
  - R traditional base plots
  - Lattice plotting system that aims to improve on R base graphics
  - ggplot2 package, a powerful and a flexible R package, for producing elegant graphics piece by piece.
  - ggpubr package, which facilitates the creation of beautiful ggplot2-based graphs for researcher with non-advanced programming backgrounds.
  - ggformula package, an extension of ggplot2, based on formula interfaces (much like the lattice interface)

1.2 Install R and RStudio

RStudio is an integrated development environment for R that makes using R easier. R and RStudio can be installed on Windows, MAC OSX and Linux platforms.

1. R can be downloaded and installed from the Comprehensive R Archive Network (CRAN) webpage (`http://cran.r-project.org/`)
2. After installing R software, install also the RStudio software available at: `http://www.rstudio.com/products/RStudio/`.
3. Launch RStudio and start use R inside R studio.

1.3 Install and load required R packages

An R package is a collection of functionalities that extends the capabilities of base R. To use the R code provide in this book, you should install the following R packages:

- `tidyverse` packages, which are a collection of R packages that share the same programming philosophy. These packages include:
  - `readr`: for importing data into R
  - `dplyr`: for data manipulation
  - `ggplot2` and `ggpubr` for data visualization.
- `ggpubr` package, which makes it easy, for beginner, to create publication ready plots.

1. **Install the tidyverse package**. Installing tidyverse will install automatically readr, dplyr, ggplot2 and more. Type the following code in the R console:

```
install.packages("tidyverse")
```

2. **Install the ggpubr package**.

- We recommend to install the latest developmental version of ggpubr as follow:

```
if(!require(devtools)) install.packages("devtools")
devtools::install_github("kassambara/ggpubr")
```

- If the above R code fails, you can install the latest stable version on CRAN:

```
install.packages("ggpubr")
```

3. **Load required packages**. After installation, you must first load the package for using the functions in the package. The function `library()` is used for this task. An alternative function is `require()`. For example, to load ggplot2 and ggpubr packages, type this:

```
library("ggplot2")
library("ggpubr")
```

Now, we can use R functions, such as *ggscatter*() [in the ggpubr package] for creating a scatter plot.

If you want to learn more about a given function, say ggscatter(), type this in R console: `?ggscatter`.

1.4 Data format

Your data should be in rectangular format, where columns are variables and rows are observations (individuals or samples).

- Column names should be compatible with R naming conventions. Avoid column with blank space and special characters. Good column names: `long_jump` or `long.jump`. Bad column name: `long jump`.

- Avoid beginning column names with a number. Use letter instead. Good column names: `sport_100m` or `x100m`. Bad column name: `100m`.
- Replace missing values by `NA` (for not available)

For example, your data should look like this:

```
  manufacturer model displ year cyl      trans drv
1         audi    a4   1.8 1999   4   auto(l5)   f
2         audi    a4   1.8 1999   4 manual(m5)   f
3         audi    a4   2.0 2008   4 manual(m6)   f
4         audi    a4   2.0 2008   4   auto(av)   f
```

Read more at: Best Practices in Preparing Data Files for Importing into R[1]

1.5 Import your data in R

First, save your data into txt or csv file formats and import it as follow (you will be asked to choose the file):

```
library("readr")

# Reads tab delimited files (.txt tab)
my_data <- read_tsv(file.choose())

# Reads comma (,) delimited files (.csv)
my_data <- read_csv(file.choose())

# Reads semicolon(;) separated files(.csv)
my_data <- read_csv2(file.choose())
```

Read more about how to import data into R at this link: `http://www.sthda.com/english/wiki/importing-data-into-r`

1.6 Demo data sets

R comes with several demo data sets for playing with R functions. The most used R demo data sets include: **USArrests**, **iris** and **mtcars**. To load a demo data set, use the function **data**() as follow. The function `head()` is used to inspect the data.

```
data("iris")   # Loading
head(iris, n = 3)  # Print the first n = 3 rows
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
```

[1] `http://www.sthda.com/english/wiki/best-practices-in-preparing-data-files-for-importing-into-r`

```
## 3          4.7         3.2          1.3         0.2  setosa
```

To learn more about iris data sets, type this:

```
?iris
```

After typing the above R code, you will see the description of `iris` data set: this iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

1.7 Data manipulation

After importing your data in R, you can easily manipulate it using the `dplyr` package (Wickham et al., 2017), which can be installed using the R code: `install.packages("dplyr")`.

After loading dplyr, you can use the following R functions:

- `filter()`: Pick rows (observations/samples) based on their values.
- `distinct()`: Remove duplicate rows.
- `arrange()`: Reorder the rows.
- `select()`: Select columns (variables) by their names.
- `rename()`: Rename columns.
- `mutate()`: Add/create new variables.
- `summarise()`: Compute statistical summaries (e.g., computing the mean or the sum)
- `group_by()`: Operate on subsets of the data set.

Note that, dplyr package allows to use the forward-pipe chaining operator (%>%) for combining multiple operations. For example, x %>% f is equivalent to f(x). Using the pipe (%>%), the output of each operation is passed to the next operation. This makes R programming easy.

We'll show you how these functions work in the different chapters of this book.

1.8 R graphics systems

There are different graphic packages available in R[2] for visualizing your data: 1) R base graphs, 2) Lattice Graphs (Sarkar, 2016) and 3) ggplot2 (Wickham and Chang, 2017).

In this section, we start by providing a quick overview of R base and lattice plots, and then we move to ggplot2 graphic system. The vast majority of plots generated in this book is based on the modern and flexible **ggplot2** R package.

[2] `http://www.sthda.com/english/wiki/data-visualization`

1.8.1 R base graphs

R comes with simple functions to create many types of graphs. For example:

Plot Types	R base function
Scatter plot	plot()
Scatter plot matrix	pairs()
Box plot	boxplot()
Strip chart	stripchart()
Histogram plot	hist()
density plot	density()
Bar plot	barplot()
Line plot	plot() and line()
Pie charts	pie()
Dot charts	dotchart()
Add text to a plot	text()

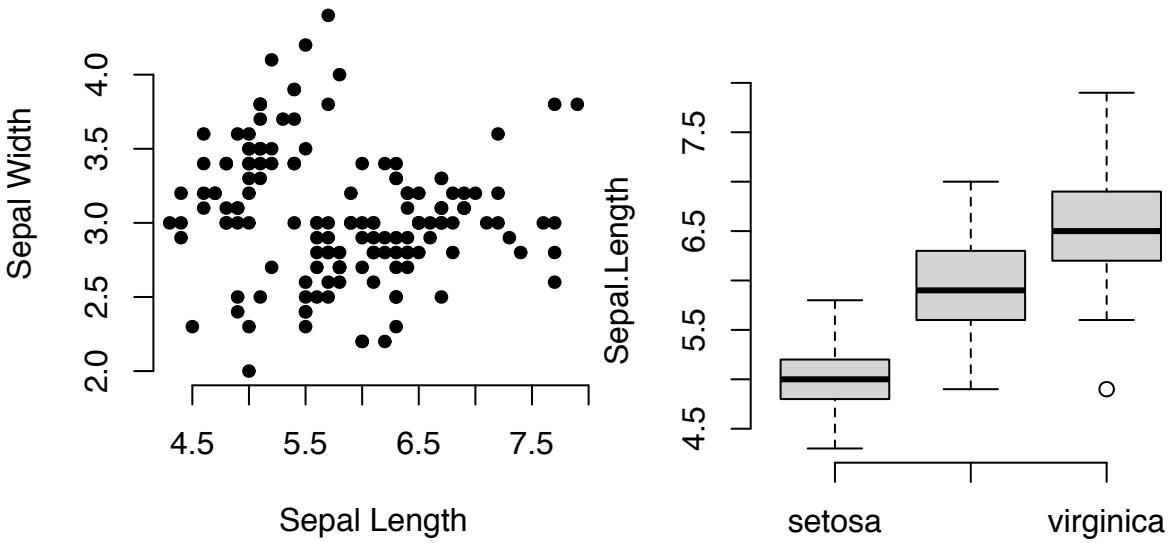
In the most cases, you can use the following arguments to customize the plot:

- `pch`: change point shapes. Allowed values comprise number from 1 to 25.
- `cex`: change point size. Example: `cex = 0.8`.
- `col`: change point color. Example: col = "blue".
- `frame`: logical value. `frame = FALSE` removes the plot panel border frame.
- `main`, `xlab`, `ylab`. Specify the main title and the x/y axis labels -, respectively
- `las`: For a vertical x axis text, use `las = 2`.

In the following R code, we'll use the iris data set to create a:

- (1) Scatter plot of Sepal.Length (on x-axis) and Sepal.Width (on y-axis).
- (2) Box plot of Sepal.length (y-axis) by Species (x-axis)

```
# (1) Create a scatter lot
plot(
  x = iris$Sepal.Length, y = iris$Sepal.Width,
  pch = 19, cex = 0.8, frame = FALSE,
  xlab = "Sepal Length",ylab = "Sepal Width"
  )

# (2) Create a box plot
boxplot(Sepal.Length ~ Species, data = iris,
        ylab = "Sepal.Length",
        frame = FALSE, col = "lightgray")
```



Read more examples at: R base Graphics on STHDA, `http://www.sthda.com/english/wiki/r-base-graphs`

1.8.2 Lattice graphics

The **lattice** R package provides a plotting system that aims to improve on R base graphs. After installing the package, whith the R command `install.packages("lattice")`, you can test the following functions.

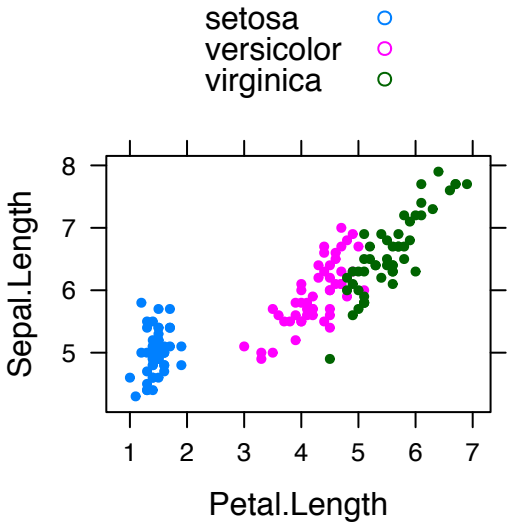
- Main functions in the lattice package:

Plot types	Lattice functions
Scatter plot	xyplot()
Scatter plot matrix	splom()
3D scatter plot	cloud()
Box plot	bwplot()
strip plots (1-D scatter plots)	stripplot()
Dot plot	dotplot()
Bar chart	barchart()
Histogram	histogram()
Density plot	densityplot()
Theoretical quantile plot	qqmath()
Two-sample quantile plot	qq()
3D contour plot of surfaces	contourplot()
False color level plot of surfaces	levelplot()
Parallel coordinates plot	parallel()
3D wireframe graph	wireframe()

The lattice package uses formula interface. For example, in lattice terminology, the formula y ~ x | group, means that we want to plot the y variable according to the x variable, splitting the plot into multiple panels by the variable group.

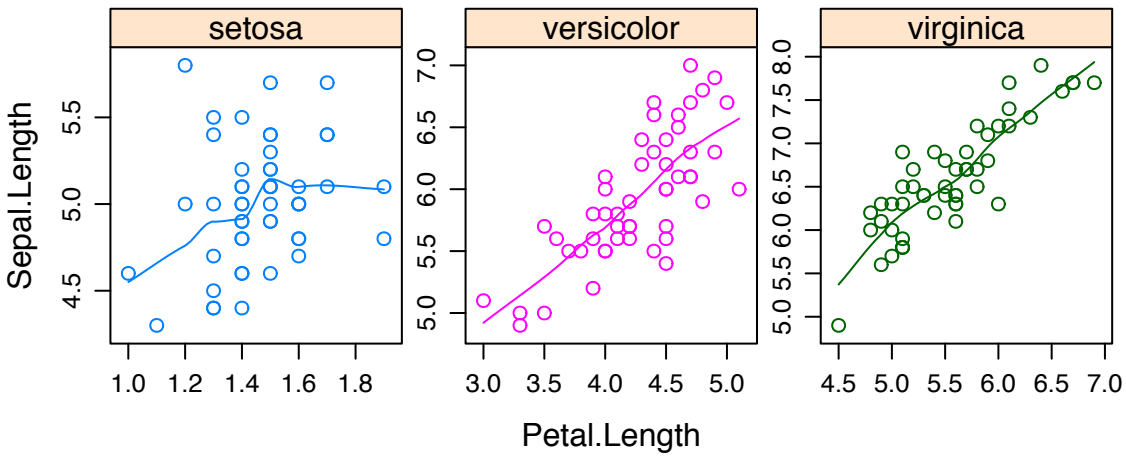
- **Create a basic scatter plot of y by x**. Syntax: `y ~ x`. Change the color by groups and use `auto.key = TRUE` to show legends:

```
library("lattice")
xyplot(
  Sepal.Length ~ Petal.Length, group = Species,
  data = iris, auto.key = TRUE, pch = 19, cex = 0.5
  )
```



- **Multiple panel plots by groups**. Syntax: `y ~ x | group`.

```
xyplot(
  Sepal.Length ~ Petal.Length | Species,
  layout = c(3, 1),                  # panel with ncol = 3 and nrow = 1
  group = Species, data = iris,
  type = c("p", "smooth"),           # Show points and smoothed line
  scales = "free"                    # Make panels axis scales independent
  )
```



Read more examples at: Lattice Graphics on STHDA[a]

[a] http://www.sthda.com/english/wiki/lattice-graphs

1.8.3 ggplot2 graphics

GGPlot2 is a powerful and a flexible R package, implemented by Hadley Wickham, for producing elegant graphics piece by piece. The **gg** in ggplot2 means *Grammar of Graphics*, a graphic concept which describes plots by using a "grammar". According to the ggplot2 concept, a plot can be divided into different fundamental parts: **Plot = data + Aesthetics + Geometry**

- **data**: a data frame
- **aesthetics**: used to indicate the **x** and **y** variables. It can be also used to control the **color**, the **size** and the **shape** of points, etc….
- **geometry**: corresponds to the type of graphics (histogram, box plot, line plot, ….)

The ggplot2 syntax might seem opaque for beginners, but once you understand the basics, you can create and customize any kind of plots you want.

Note that, to reduce this opacity, we recently created an R package, named **ggpubr** (ggplot2 Based Publication Ready Plots), for making ggplot simpler for students and researchers with non-advanced programming backgrounds. We'll present ggpubr in the next section.

After installing and loading the ggplot2 package, you can use the following key functions:

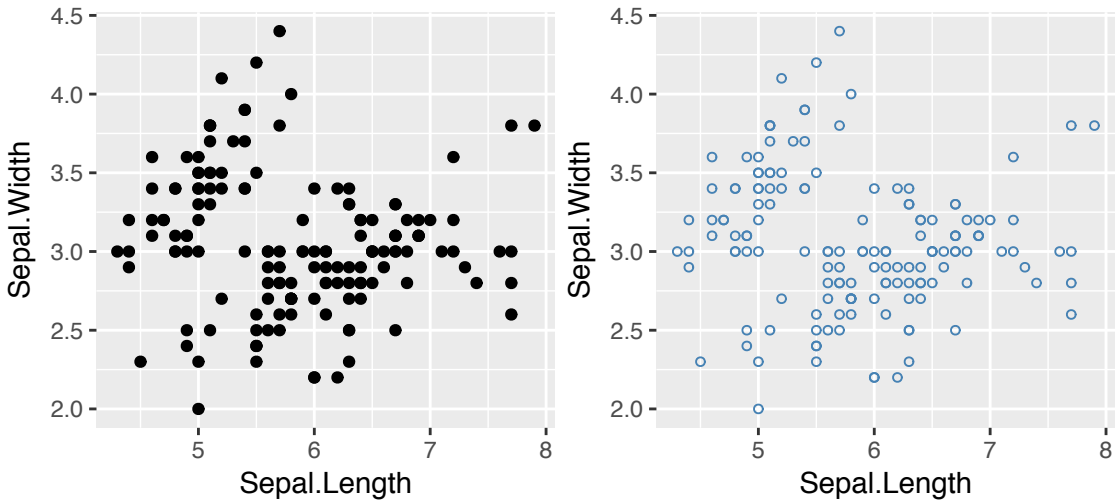
Plot types	GGPlot2 functions
Initialize a ggplot	ggplot()
Scatter plot	geom_point()
Box plot	geom_boxplot()
Violin plot	geom_violin()
strip chart	geom_jitter()
Dot plot	geom_dotplot()
Bar chart	geom_bar()
Line plot	geom_line()
Histogram	geom_histogram()
Density plot	geom_density()
Error bars	geom_errorbar()
QQ plot	stat_qq()
ECDF plot	stat_ecdf()
Title and axis labels	labs()

The main function in the ggplot2 package is `ggplot()`, which can be used to initialize the plotting system with data and x/y variables.

For example, the following R code takes the `iris` data set to initialize the ggplot and then a layer (`geom_point()`) is added onto the ggplot to create a scatter plot of `x = Sepal.Length` by `y = Sepal.Width`:

```
library(ggplot2)
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()
```

```
# Change point size, color and shape
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(size = 1.2, color = "steelblue", shape = 21)
```

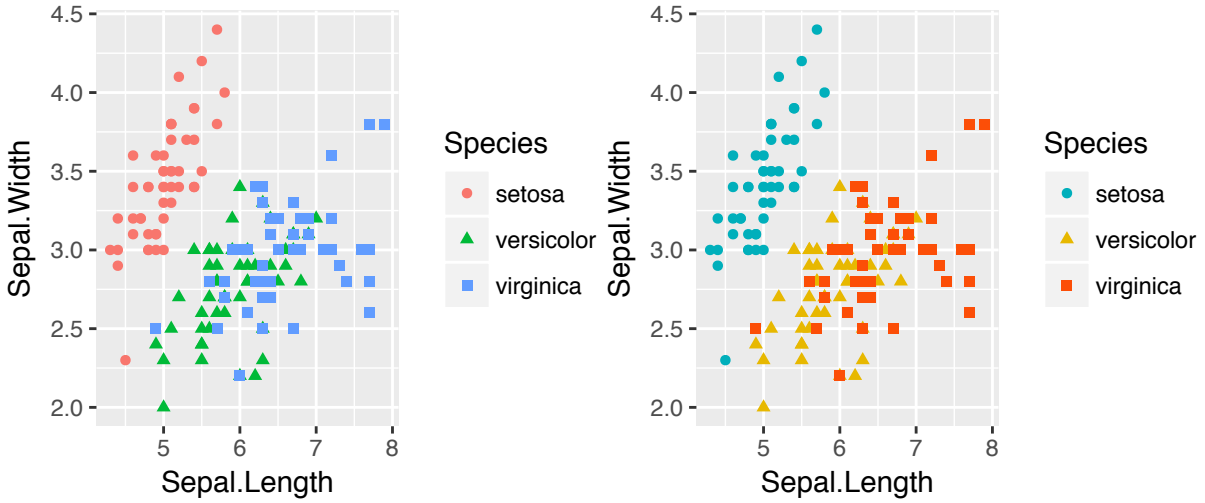


Note that, in the code above, the shape of points is specified as number. To display the different point shape available in R, type this:

```
ggpubr::show_point_shapes()
```

It's also possible to control points shape and color by a grouping variable (here, `Species`). For example, in the code below, we map points color and shape to the `Species` grouping variable.

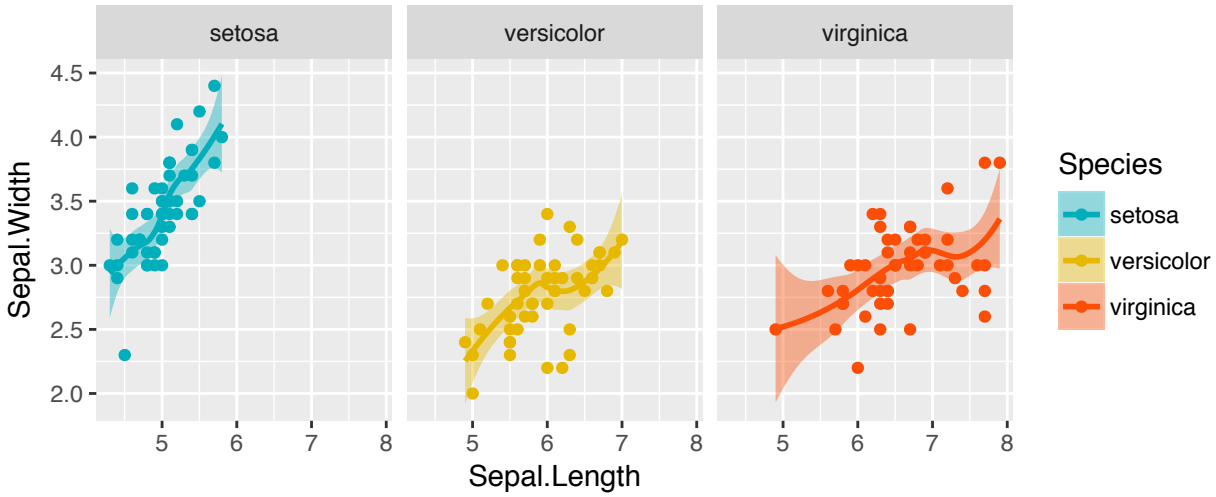
```
# Control points color by groups
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(color = Species, shape = Species))

# Change the default color manually.
# Use the scale_color_manual() function
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(color = Species, shape = Species))+
  scale_color_manual(values = c("#00AFBB", "#E7B800", "#FC4E07"))
```



You can also split the plot into multiple panels according to a grouping variable. R function: `facet_wrap()`. Another interesting feature of ggplot2, is the possibility to combine multiple layers on the same plot. For example, with the following R code, we'll:

- Add points with `geom_point()`, colored by groups.
- Add the fitted smoothed regression line using `geom_smooth()`. By default the function `geom_smooth()` add the regression line and the confidence area. You can control the line color and confidence area fill color by groups.
- Facet the plot into multiple panels by groups
- Change color and fill manually using the function `scale_color_manual()` and `scale_fill_manual()`

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(color = Species))+
  geom_smooth(aes(color = Species, fill = Species))+
  facet_wrap(~Species, ncol = 3, nrow = 1)+
  scale_color_manual(values = c("#00AFBB", "#E7B800", "#FC4E07"))+
  scale_fill_manual(values = c("#00AFBB", "#E7B800", "#FC4E07"))
```



Note that, the default theme of ggplots is `theme_gray()` (or `theme_grey()`), which is theme with grey background and white grid lines. More themes are available for professional presentations or publications. These include: `theme_bw()`, `theme_classic()` and

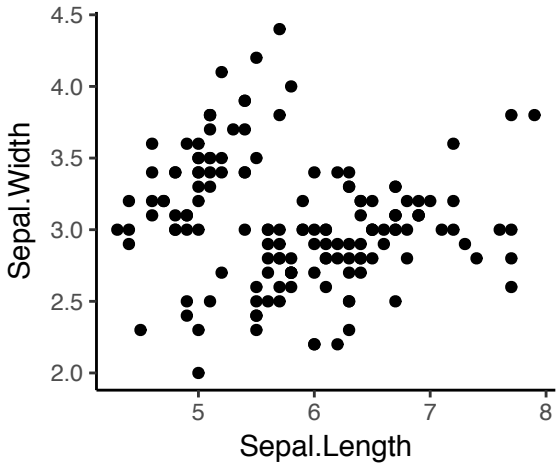
`theme_minimal()`.

To change the theme of a given ggplot (p), use this: `p + theme_classic()`. To change the default theme to `theme_classic()` for all the future ggplots during your entire R session, type the following R code:

```
theme_set(
  theme_classic()
)
```

Now you can create ggplots with `theme_classic()` as default theme:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()
```

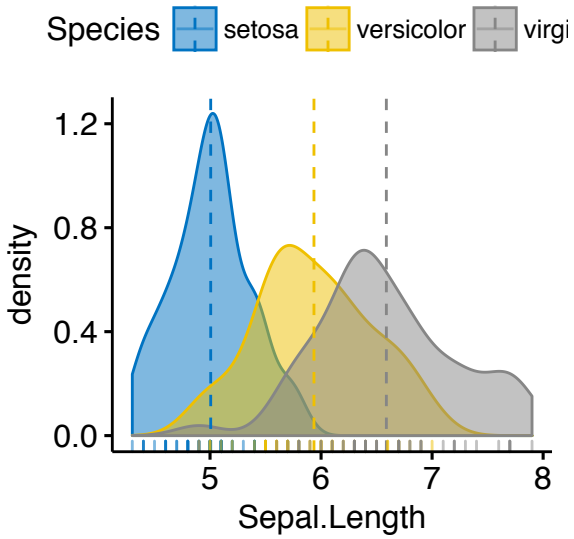


1.8.4 ggpubr for publication ready plots

The **ggpubr** R package facilitates the creation of beautiful ggplot2-based graphs for researcher with non-advanced programming backgrounds (Kassambara, 2017).

For example, to create the density distribution of "Sepal.Length", colored by groups ("Species"), type this:

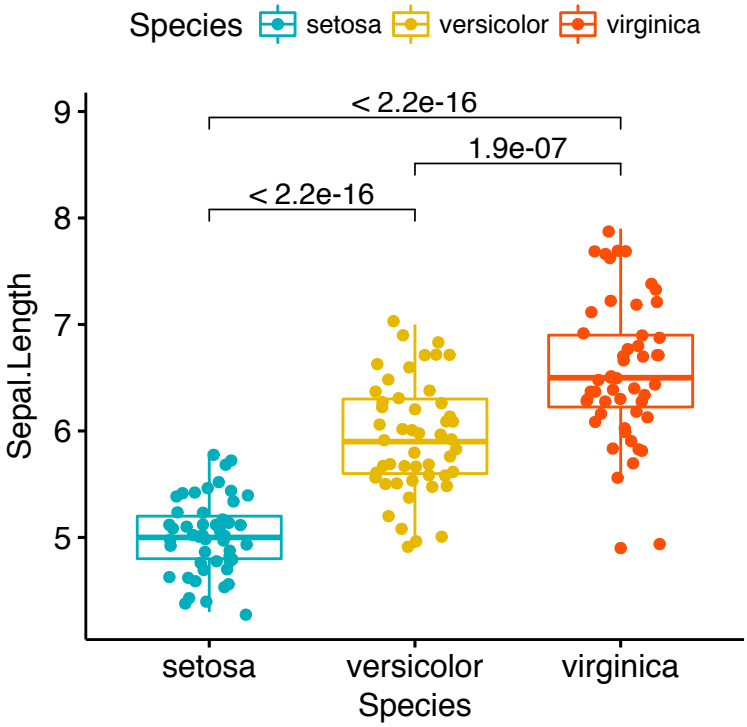
```
library(ggpubr)
# Density plot with mean lines and marginal rug
ggdensity(iris, x = "Sepal.Length",
   add = "mean", rug = TRUE,              # Add mean line and marginal rugs
   color = "Species", fill = "Species",   # Color by groups
   palette = "jco")                       # use jco journal color palette
```



Note that the argument `palette` can take also a custom color palette. For example `palette= c("#00AFBB", "#E7B800", "#FC4E07")`.

- Create a box plot with p-values comparing groups:

```
# Groups that we want to compare
my_comparisons <- list(
  c("setosa", "versicolor"), c("versicolor", "virginica"),
  c("setosa", "virginica")
)

# Create the box plot. Change colors by groups: Species
# Add jitter points and change the shape by groups
ggboxplot(
  iris, x = "Species", y = "Sepal.Length",
  color = "Species", palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  add = "jitter"
  )+
  stat_compare_means(comparisons = my_comparisons, method = "t.test")
```



Learn more on STHDA at: ggpubr: Publication Ready Plots[a]

[a] http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/

1.9 Export R graphics

You can export R graphics to many file formats, including: PDF, PostScript, SVG vector files, Windows MetaFile (WMF), PNG, TIFF, JPEG, etc.

The standard procedure to save any graphics from R is as follow:

1. **Open a graphic device** using one of the following functions:

- pdf("r-graphics.pdf"),
- postscript("r-graphics.ps"),
- svg("r-graphics.svg"),
- png("r-graphics.png"),
- tiff("r-graphics.tiff"),
- jpeg("r-graphics.jpg"),
- win.metafile("r-graphics.wmf"),
- and so on.

Additional arguments indicating the width and the height (in inches) of the graphics region can be also specified in the mentioned function.

2. **Create a plot**

3. **Close the graphic device** using the function `dev.off()`

For example, you can export R base plots to a pdf file as follow:

```
pdf("r-base-plot.pdf")
# Plot 1 --> in the first page of PDF
plot(x = iris$Sepal.Length, y = iris$Sepal.Width)
# Plot 2 ---> in the second page of the PDF
hist(iris$Sepal.Length)
dev.off()
```

To export ggplot2 graphs, the R code looks like this:

```
# Create some plots
library(ggplot2)
myplot1 <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point()
myplot2 <- ggplot(iris, aes(Species, Sepal.Length)) +
  geom_boxplot()

# Print plots to a pdf file
pdf("ggplot.pdf")
print(myplot1)     # Plot 1 --> in the first page of PDF
print(myplot2)     # Plot 2 ---> in the second page of the PDF
dev.off()
```

Note that for a ggplot, you can also use the following functions to export the graphic:

- `ggsave()` [in ggplot2]. Makes it easy to save a ggplot. It guesses the type of graphics device from the file extension.
- `ggexport()` [in ggpubr]. Makes it easy to arrange and export multiple ggplots at once.

See also the following blog post to save high-resolution ggplots[a]

[a] http://www.sthda.com/english/wiki/saving-high-resolution-ggplots-how-to-preserve-semi-transparency

Chapter 2

Plot One Variable

2.1 Introduction

To visualize one variable, the type of graphs to be used depends on the type of the variable:

- For **categorical variable** or grouping variables. You can visualize the count of categories using a bar plot or using a pie chart to show the proportion of each category.
- For **continuous variable**, you can visualize the distribution of the variable using density plots, histograms and alternatives.

In this R graphics tutorial, you'll learn how to:

- Visualize a categorical variable using bar plots, dot charts and pie charts
- Visualize the distribution of a continuous variable using:
  - density and histogram plots,
  - other alternatives, such as frequency polygon, area plots, dot plots, box plots, Empirical cumulative distribution function (ECDF) and Quantile-quantile plot (QQ plots).
  - Density ridgeline plots, which are useful for visualizing changes in distributions, of a continuous variable, over time or space.
  - Bar plot and modern alternatives, including lollipop charts and cleveland's dot plots.

2.2 Prerequisites

Load required packages and set the theme function `theme_pubr()` [in ggpubr] as the default theme:

```
library(ggplot2)
library(ggpubr)
theme_set(theme_pubr())
```

2.3 One categorical variable

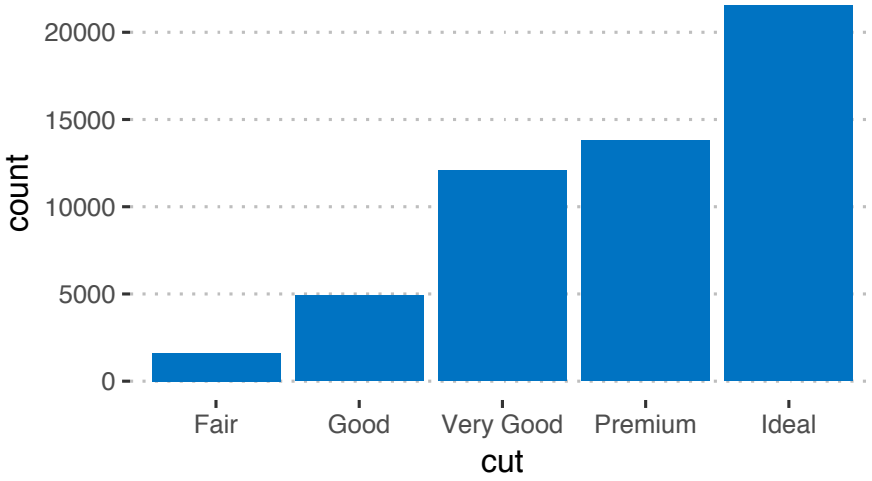
2.3.1 Bar plot of counts

- Plot types: Bar plot of the count of group levels
- Key function: `geom_bar()`
- Key arguments: `alpha`, `color`, `fill`, `linetype` and `size`

Demo data set: `diamonds` [in ggplot2]. Contains the prices and other attributes of almost 54000 diamonds. The column `cut` contains the quality of the diamonds cut (Fair, Good, Very Good, Premium, Ideal).

The R code below creates a bar plot visualizing the number of elements in each category of diamonds cut.

```
ggplot(diamonds, aes(cut)) +
  geom_bar(fill = "#0073C2FF") +
  theme_pubclean()
```



Compute the frequency of each category and add the labels on the bar plot:

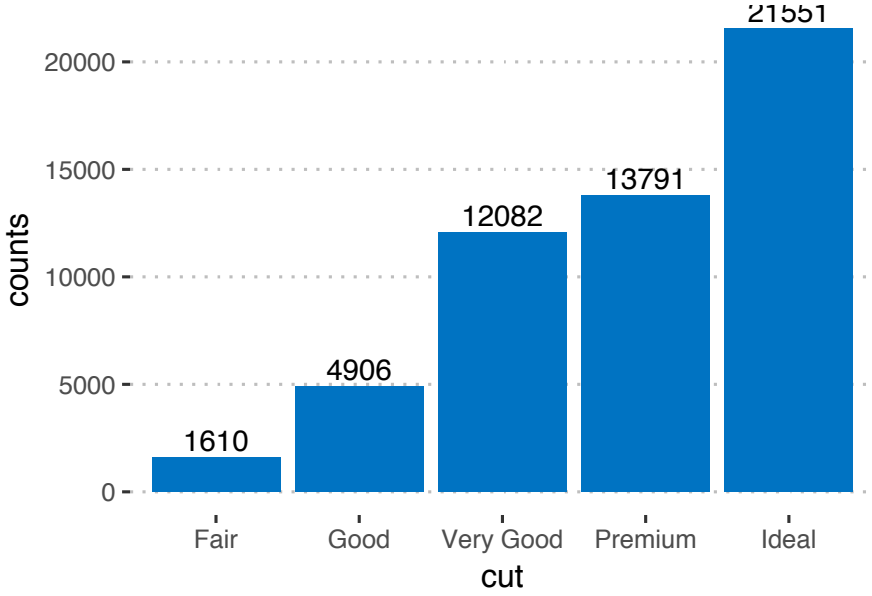
- `dplyr` package used to summarise the data
- `geom_bar()` with option `stat = "identity"` is used to create the bar plot of the summary output as it is.
- `geom_text()` used to add text labels. Adjust the position of the labels by using `hjust` (horizontal justification) and `vjust` (vertical justification). Values should be in [0, 1].

```
# Compute the frequency
library(dplyr)
df <- diamonds %>%
  group_by(cut) %>%
  summarise(counts = n())
df
```

```
## # A tibble: 5 x 2
##         cut counts
##       <ord>  <int>
```

```
## 1      Fair   1610
## 2      Good   4906
## 3 Very Good  12082
## 4   Premium  13791
## 5     Ideal  21551
```

```
# Create the bar plot. Use theme_pubclean() [in ggpubr]
ggplot(df, aes(x = cut, y = counts)) +
  geom_bar(fill = "#0073C2FF", stat = "identity") +
  geom_text(aes(label = counts), vjust = -0.3) +
  theme_pubclean()
```



2.3.2 Pie charts

Pie chart is just a stacked bar chart in polar coordinates.

First,

- Arrange the grouping variable (`cut`) in descending order. This important to compute the y coordinates of labels.
- compute the proportion (counts/total) of each category
- compute the position of the text labels as the cumulative sum of the proportion. To put the labels in the center of pies, we'll use `cumsum(prop) - 0.5*prop` as label position.

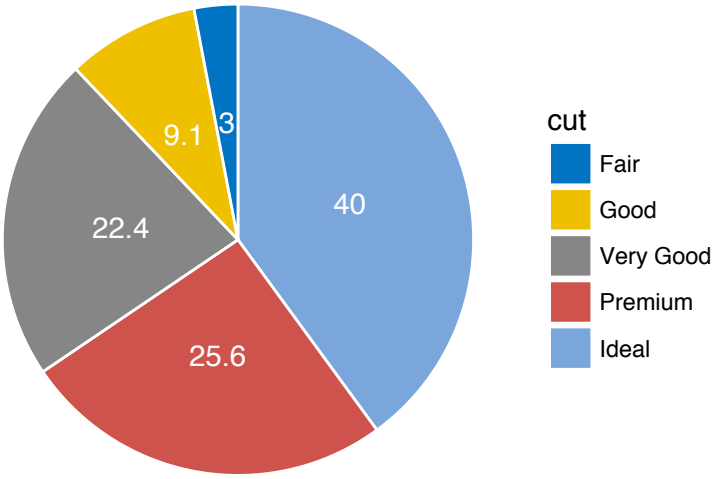
```
df <- df %>%
  arrange(desc(cut)) %>%
  mutate(prop = round(counts*100/sum(counts), 1),
         lab.ypos = cumsum(prop) - 0.5*prop)
head(df, 4)
```

```
## # A tibble: 4 x 4
##         cut counts  prop lab.ypos
```

```
##       <ord>  <int> <dbl>    <dbl>
## 1     Ideal  21551  40.0     20.0
## 2   Premium  13791  25.6     52.8
## 3 Very Good  12082  22.4     76.8
## 4      Good   4906   9.1     92.5
```

- Create the pie charts using ggplot2 verbs. Key function: `coord_polar()`.

```
ggplot(df, aes(x = "", y = prop, fill = cut)) +
  geom_bar(width = 1, stat = "identity", color = "white") +
  geom_text(aes(y = lab.ypos, label = prop), color = "white")+
  coord_polar("y", start = 0)+
  ggpubr::fill_palette("jco")+
  theme_void()
```



- Alternative solution to easily create a pie chart: use the function `ggpie()`[in ggpubr]:

```
ggpie(
  df, x = "prop", label = "prop",
  lab.pos = "in", lab.font = list(color = "white"),
  fill = "cut", color = "white",
  palette = "jco"
)
```

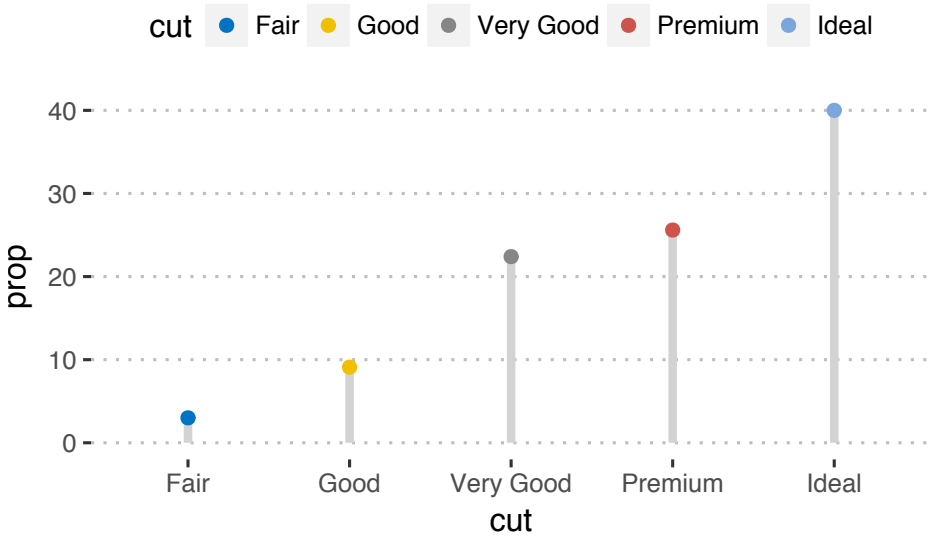
2.3.3 Dot charts

Dot chart is an alternative to bar plots. Key functions:

- `geom_linerange()`:Creates line segments from x to ymax
- `geom_point()`: adds dots
- `ggpubr::color_palette()`: changes color palette.

```
ggplot(df, aes(cut, prop)) +
  geom_linerange(
    aes(x = cut, ymin = 0, ymax = prop),
    color = "lightgray", size = 1.5
```

```
    )+
  geom_point(aes(color = cut), size = 2)+
  ggpubr::color_palette("jco")+
  theme_pubclean()
```



Easy alternative to create a dot chart. Use `ggdotchart()` [ggpubr]:

```
ggdotchart(
  df, x = "cut", y = "prop",
  color = "cut", size = 3,        # Points color and size
  add = "segment",                # Add line segments
  add.params = list(size = 2),
  palette = "jco",
  ggtheme = theme_pubclean()
)
```

2.4 One continuous variable

Different types of graphs can be used to visualize the distribution of a continuous variable, including: density and histogram plots.

2.4.1 Data format

Create some data (`wdata`) containing the weights by sex (M for male; F for female):

```
set.seed(1234)
wdata = data.frame(
        sex = factor(rep(c("F", "M"), each=200)),
        weight = c(rnorm(200, 55), rnorm(200, 58))
        )

head(wdata, 4)
```