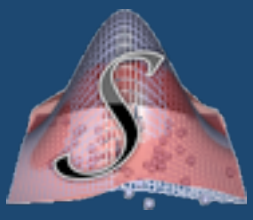


STHDA

Alboukadel Kassambara

ggplot2: The Elements for Elegant Data Visualization in R

First edition

ggplot2: The Elements for Elegant Data Visualization in R

Alboukadel KASSAMBARA

Contents

ggplot2: The Elements for Elegant Data Visualization in R

By Alboukadel Kassambara

Published by STHDA (http://www.sthda.com).

June 2015 : First edition.

Contact : Alboukadel Kassambara alboukadel.kassambara@gmail.com

0.1 Preface

ggplot2 is an R package implemented by **Hadley Wickham** for creating graphs. It's based on the Grammar of Graphics, a concept published by **Leland Wilkinson** in 2005. The official documentation of the package is available here: http://docs.ggplot2.org/current/. The first book on ggplot2 - Ggplot2: Elegant Graphics for Data Analysis (Hadley Wickman) was published in 2009.

ggplot2 has become a popular package for data visualization. In this book, I present the most important functions available in ggplot2 package to quickly and easily generate nice looking graphs.

You will find many examples of R codes and graphics in this document.

Note that, all the analyses in this book were performed using R (ver. 3.1.2) and ggplot2 (ver 1.0.0).

Thanks to Leland Wilkinson for the concept,

Thanks to Hadley Wickham for ggplot2 package and for his great book on ggplot2.

0.2 How this book is organized?

This book contains 3 parts. After giving a brief overview of ggplot2 (in the part 1, chapter 1), quick start guides are provided in chapters 2 - 14 for creating and customizing different types of graphs including box plots, violin plots, dot plots, stripcharts, density plots, histogram plots, scatter plots, bar plots, line plots, error bars, pie charts, qq plots and ECDF.

The last chapter of the part 1 (chapter 15) describes how to save ggplots to a pdf or a png files for presentation.

The part 2 of the book covers how to change graphical parameters including:

- Main title, axis labels and legend titles (chapter 16)
- Position and the appearance of plot legends (chapter 17)
- Manual and automatic coloring (chapter 18)
- Point shapes (chapter 19)
- Line types (chapter 20)
- Adding text annotations to a graph (chapter 21)
- Adding straight lines to a plot: horizontal, vertical and regression lines (chapter 22)
- Axis scales and transformations (chapter 23)
- Axis ticks : customize tick marks and labels (chapter 24)
- Themes and background colors (chapter 25)
- Rotate a graph (chapter 26)
- Facets: split a plot into a matrix of panels (chapter 27)

The part 3 describes some extensions of ggplot2 including:

- Mixing multiple graphs on the same page (chapter 28)
- Plotting a correlation matrix heatmap (chapter 29)
- Plotting survival curves (chapter 30)

Each chapter is organized as an independent quick start guide. This means that, you don't need to read the different chapters in sequence. I just recommend to read firstly the chapter 1, as it gives a quick overview of ggplot2 graphing system.

For each chapter, the covered ggplot2 key functions are generally mentioned at the beginning. The used data are described and many examples of R codes and graphics are provided.

Sometimes, different chapters use the same data. In this case, I decided to repeat the data preparation description in the corresponding chapters. In other words, each chapter is an independent module and this gives the possibility to the user to read only the chapter of interest.

0.3 How to execute the R codes provided in this book?

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

Part I

Quick start guide to create graphics with ggplot2

Chapter 1

Brief introduction to ggplot2

Key functions : **qplot()** and **ggplot()**.

1.1 What's ggplot2?

ggplot2 is a powerful **R package**, implemented by **Hadley Wickham**, for producing beautiful graphs. The **gg** in ggplot2 stands for **Grammar of Graphics**, a graphic concept which describes plots by using a "grammar".

Two main functions are available in **ggplot2** package: a **qplot()** and **ggplot()** functions.

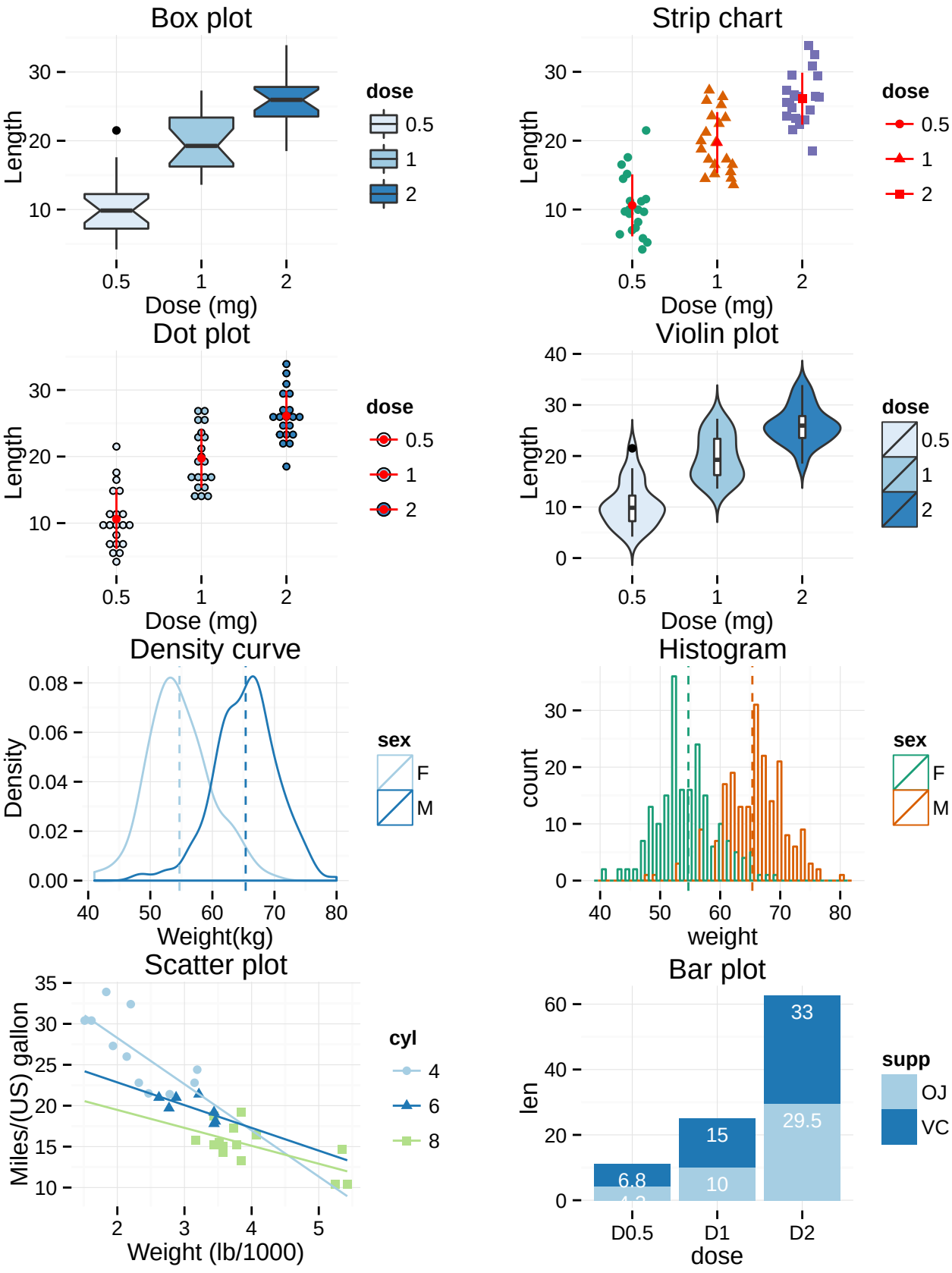
- **qplot()** is a quick plot function which is easy to use for simple plots.
- The **ggplot()** function uses the powerful grammar of graphics to build plot piece by piece.

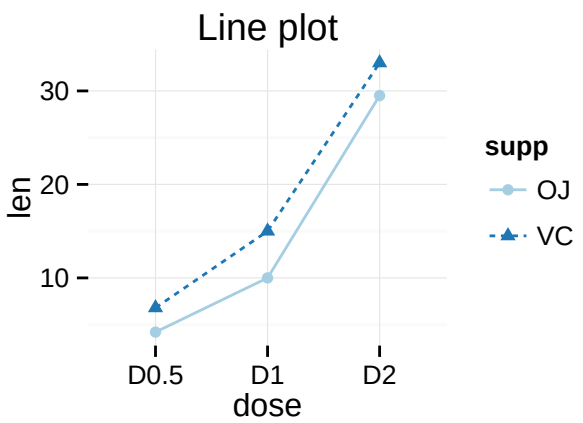
According to ggplot2 concept, a plot can be divide in different fundamental parts:
Plot = data + Aesthetics + Geometry.

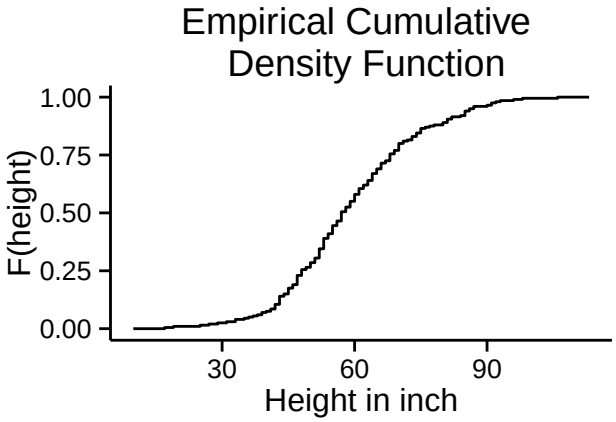
- **data** is a data frame
- **Aesthetics** is used to indicate x and y variables. It can also be used to control the **color**, the **size** or the **shape** of a point, ...
- **Geometry** corresponds to the type of graphics (**histogram**, **box plot**, **line plot**, **density plot**, **dot plot**, ...)

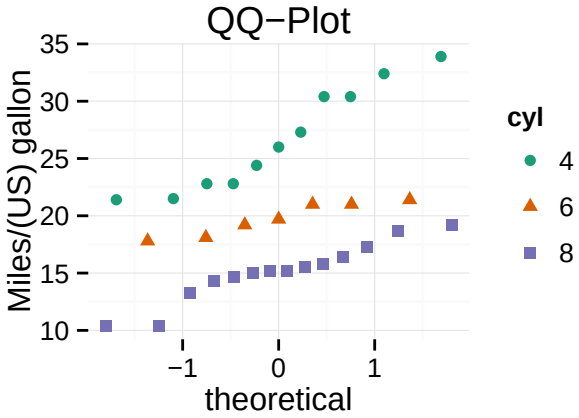
This document describes how to create and customize different types of graphs using ggplot2. Many examples of code and graphics are provided.

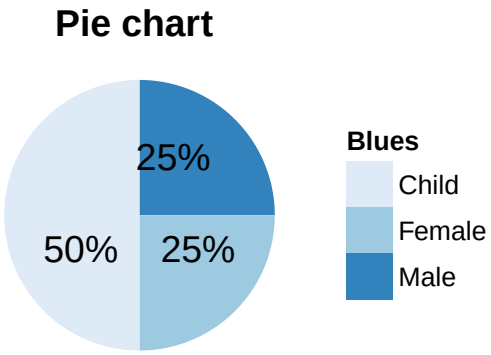
Some examples of graphs, described in this book, are shown below:

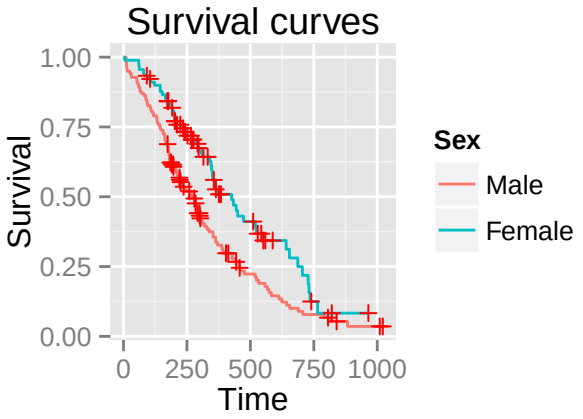

Box plot
Length
Dose (mg)
dose
0.5
1
2
Strip chart
Dot plot
Violin plot
Density curve
Density
Weight(kg)
sex
F
M
Histogram
count
weight
Scatter plot
Miles/(US) gallon
Weight (lb/1000)
cyl
4
6
8
Bar plot
len
dose
D0.5
D1
D2
supp
OJ
VC
33
29.5
15
10
6.8

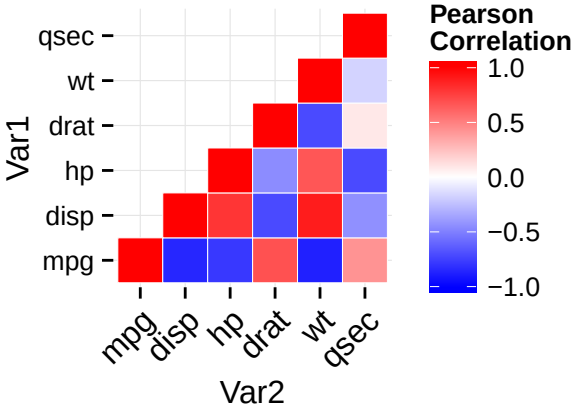

Line plot
30
20
10
len
D0.5
D1
D2
dose
supp
OJ
VC


Empirical Cumulative
Density Function
1.00
0.75
0.50
0.25
0.00
F(height)
30
60
90
Height in inch


QQ−Plot
35
30
25
20
15
10
Miles/(US) gallon
−1
0
1
theoretical
cyl
4
6
8


Pie chart
25%
50%
25%
Blues
Child
Female
Male


Survival curves
1.00
0.75
0.50
0.25
0.00
Survival
0
250
500
750
1000
Time
Sex
Male
Female


qsec
wt
drat
hp
disp
mpg
Var1
mpg
disp
hp
drat
wt
qsec
Var2
Pearson
Correlation
1.0
0.5
0.0
−0.5
−1.0

1.2 Install and load ggplot2 package

Use the R code below:

```
# Installation
install.packages('ggplot2')

# Loading
library(ggplot2)
```

1.3 Data format

The data must be a data frame (columns are variables and rows are observations).

The data set **mtcars** is used in the examples below:

```
data(mtcars)
df <- mtcars[, c("mpg", "cyl", "wt")]
head(df)
```

```
##                    mpg cyl    wt
## Mazda RX4         21.0   6 2.620
## Mazda RX4 Wag     21.0   6 2.875
## Datsun 710        22.8   4 2.320
## Hornet 4 Drive    21.4   6 3.215
## Hornet Sportabout 18.7   8 3.440
## Valiant           18.1   6 3.460
```

mtcars : Motor Trend Car Road Tests.
Description: The data comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973 - 74 models).

Three of the variables are used in this section:

- [, 1] mpg Miles/(US) gallon
- [, 2] cyl Number of cylinders
- [, 3] wt Weight (lb/1000)

1.4 Quick plot : qplot() function

The function **qplot()** is very similar to the standard **plot()** function from the R base package. It can be used to create and combine easily different types of plots. However, it remains less flexible than the function **ggplot()**.

This section provides a brief introduction to **qplot()**. Concerning the function **ggplot()**, many articles are available in the next chapters for creating and customizing different plots.

1.4.1 Usage

A simplified format of **qplot()** is :

```
qplot(x, y = NULL, data, geom = "auto",
      xlim = c(NA, NA), ylim = c(NA, NA))
```

- **x** : x values
- **y** : y values (optional)
- **data** : data frame to be used (optional).
- **geom** : character vector specifying the geom to use. Defaults to "point" if x and y are specified, and "histogram" if only x is specified.
- **xlim**, **ylim**: x and y axis limits

Other arguments, including *main*, *xlab*, *ylab* and *log*, can be used also:

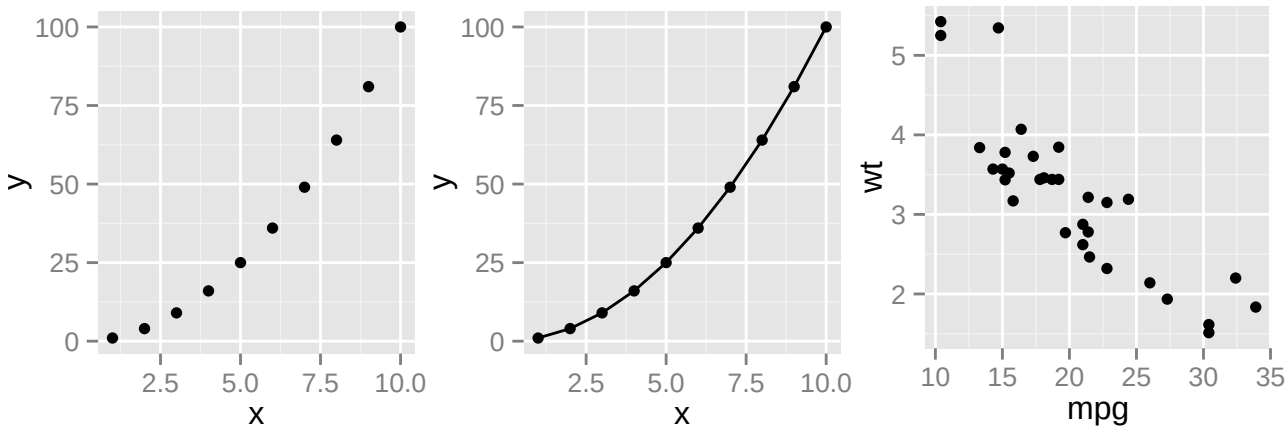
- **main**: plot title
- **xlab, ylab**: x and y axis labels
- **log**: which variables to log transform. Allowed values are "x", "y" or "xy"

1.4.2 Scatter plots

1.4.2.1 Basic scatter plots

The plot can be created using data from either numeric vectors (*x and y*) or a data frame (*mtcars*):

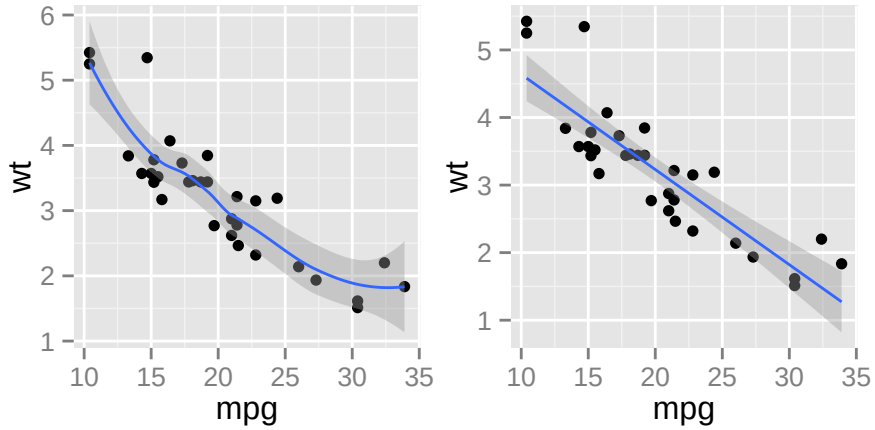
```
# Use data from numeric vectors
x <- 1:10; y = x*x

# Basic plot
qplot(x,y)

# Combine points and line
qplot(x, y, geom = c("point", "line"))

# Use data from a data frame
qplot(mpg, wt, data = mtcars)
```



1.4.2.2 Scatter plots with linear fits

The option **smooth** is used to add a **smoothed line**. The confidence interval (level = 95% by default), around the smoothed line, is also displayed:

```
# Smoothing
qplot(mpg, wt, data = mtcars, geom = c("point", "smooth"))

# Regression line
qplot(mpg, wt, data = mtcars, geom = c("point", "smooth"),
      method = "lm")
```



To draw a regression line the argument **method = "lm"** is used in combination with **geom = "smoth".**

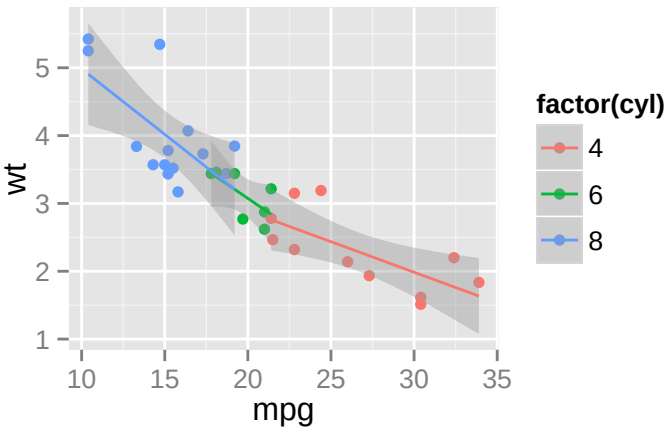
The allowed values for the argument **method** includes:

- **method = "loess"**: This is the default value for small number of observations. It computes a smooth local regression. You can read more about **loess** using the R code **?loess**.
- **method ="lm"**: It fits a **linear model**.

1.4.2.3 Linear fits by groups

The argument **color** is used to tell **R** that we want to color the points by groups:

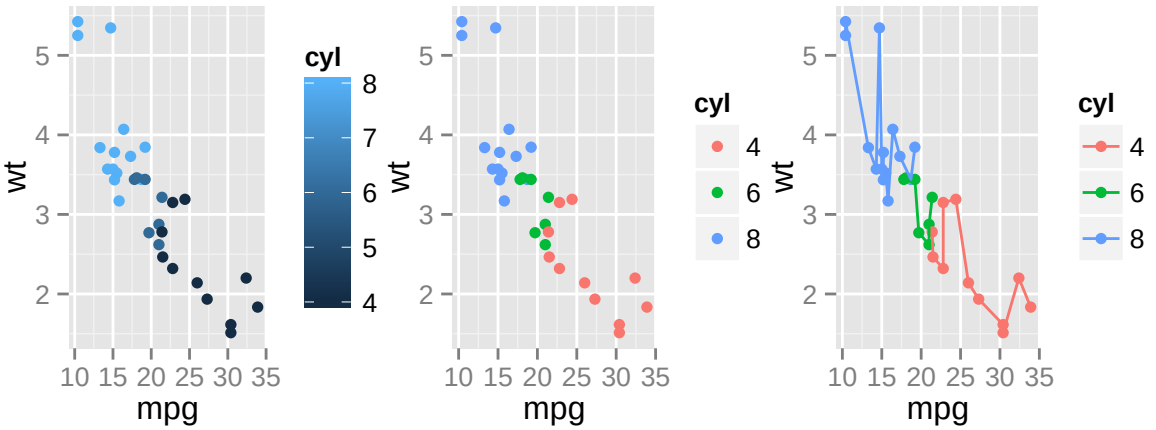
```
# Linear fits by group
qplot(mpg, wt, data = mtcars, color = factor(cyl),
      geom=c("point", "smooth"),
      method="lm")
```



1.4.2.4 Change scatter plot colors

Points can be colored according to the values of a continuous or a discrete variable. The argument **color** is used.

```
# Change the color by a continuous numeric variable
qplot(mpg, wt, data = mtcars, color = cyl)

# Change the color by groups (factor)
df <- mtcars
df[,'cyl'] <- as.factor(df[,'cyl'])
qplot(mpg, wt, data = df, color = cyl)

# Add lines
qplot(mpg, wt, data = df, color = cyl,
      geom=c("point", "line"))
```



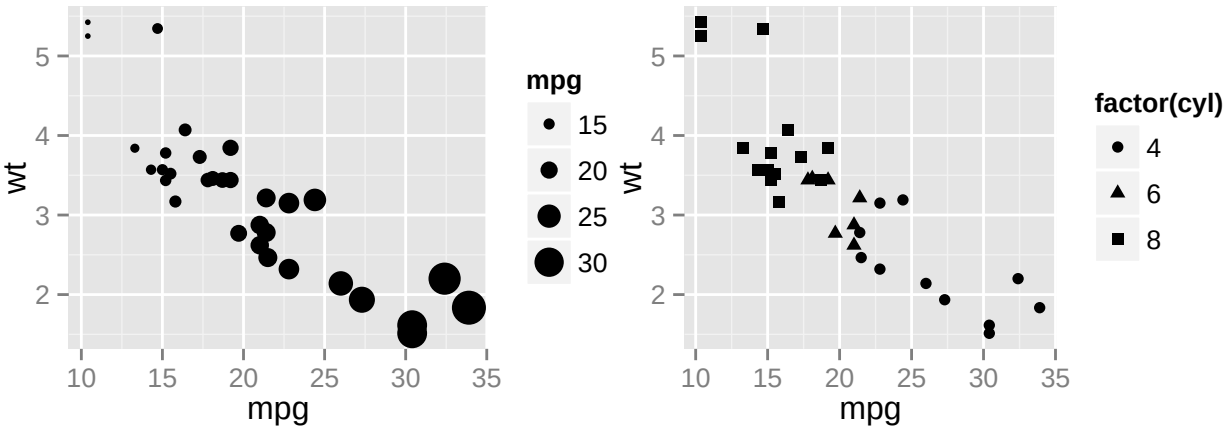
Note that you can also use the following R code to generate the second plot :

```
qplot(mpg, wt, data=df, colour= factor(cyl))
```

1.4.2.5 Change the shape and the size of points

Like color, the **shape** and the **size** of points can be controlled by a continuous or a discrete variable.

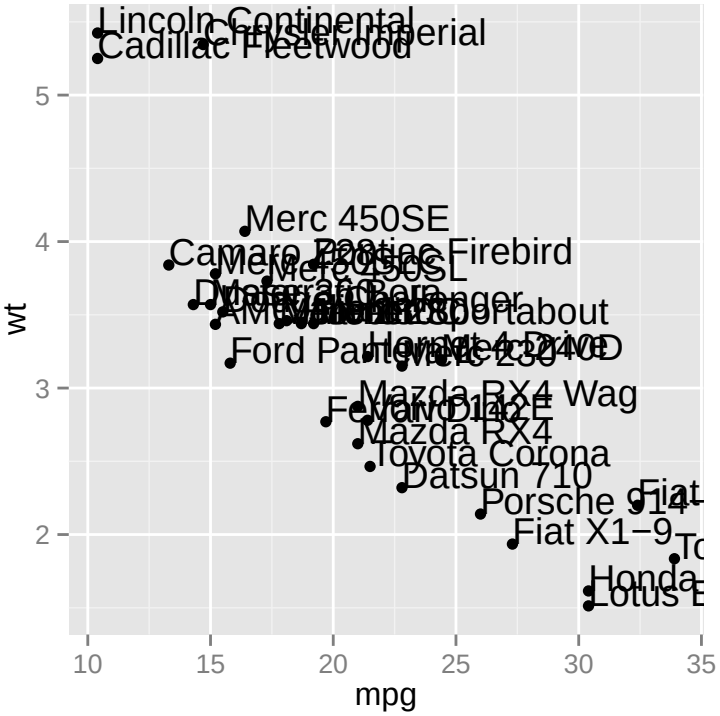
```
# Change the size of points according to
  # the values of a continuous variable
qplot(mpg, wt, data = mtcars, size = mpg)

# Change point shapes by groups
qplot(mpg, wt, data = mtcars, shape = factor(cyl))
```



1.4.3 Scatter plots with texts

To add a text the argument **geom = "text"** is used. The argument **label** is used to specify the text for each points. The parameters *hjuts* and *vjust* are the horizontal and the vertical justification of the texts, respectively.

```
qplot(mpg, wt, data = mtcars, label = rownames(mtcars),
      geom = c("point", "text"),
      hjust = 0, vjust = 0)
```

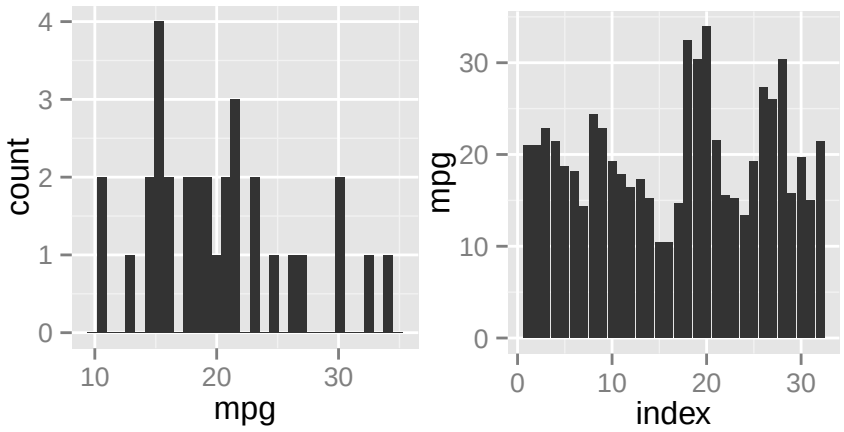


1.4.4 Bar plots

It's possible to draw a **bar plot** using the argument **geom = "bar"**.

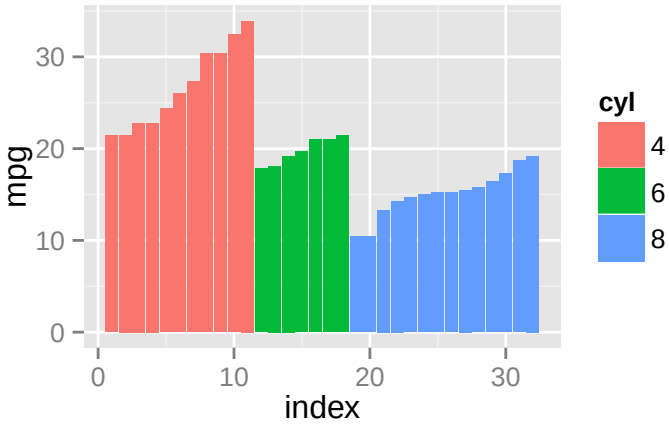
If you want **y** to represent counts of cases, use **stat = "bin"** and don't map a variable to y. If you want **y** to represent values in the data, use **stat = "identity"**.

```
# y represents the count of cases
qplot(mpg, data = mtcars, geom = "bar")

# y represents values in the data
index <- 1:nrow(mtcars)
qplot(index, mpg, data = mtcars,
      geom = "bar", stat = "identity")
```



Change bar plot fill color. In the R code below, the data are sorted for each group before plotting.

```
# Order the data by cyl and then by mpg values
df <- mtcars[order(mtcars[, "cyl"], mtcars[, "mpg"]),]
df[,'cyl'] <- as.factor(df[,'cyl'])
index <- 1:nrow(df)

# Change fill color by group (cyl)
qplot(index, mpg, data = df,
      geom = "bar", stat = "identity", fill = cyl)
```



1.4.5 Box plots, dot plots and violin plots

PlantGrowth data set is used in the following example :

```
head(PlantGrowth)
```

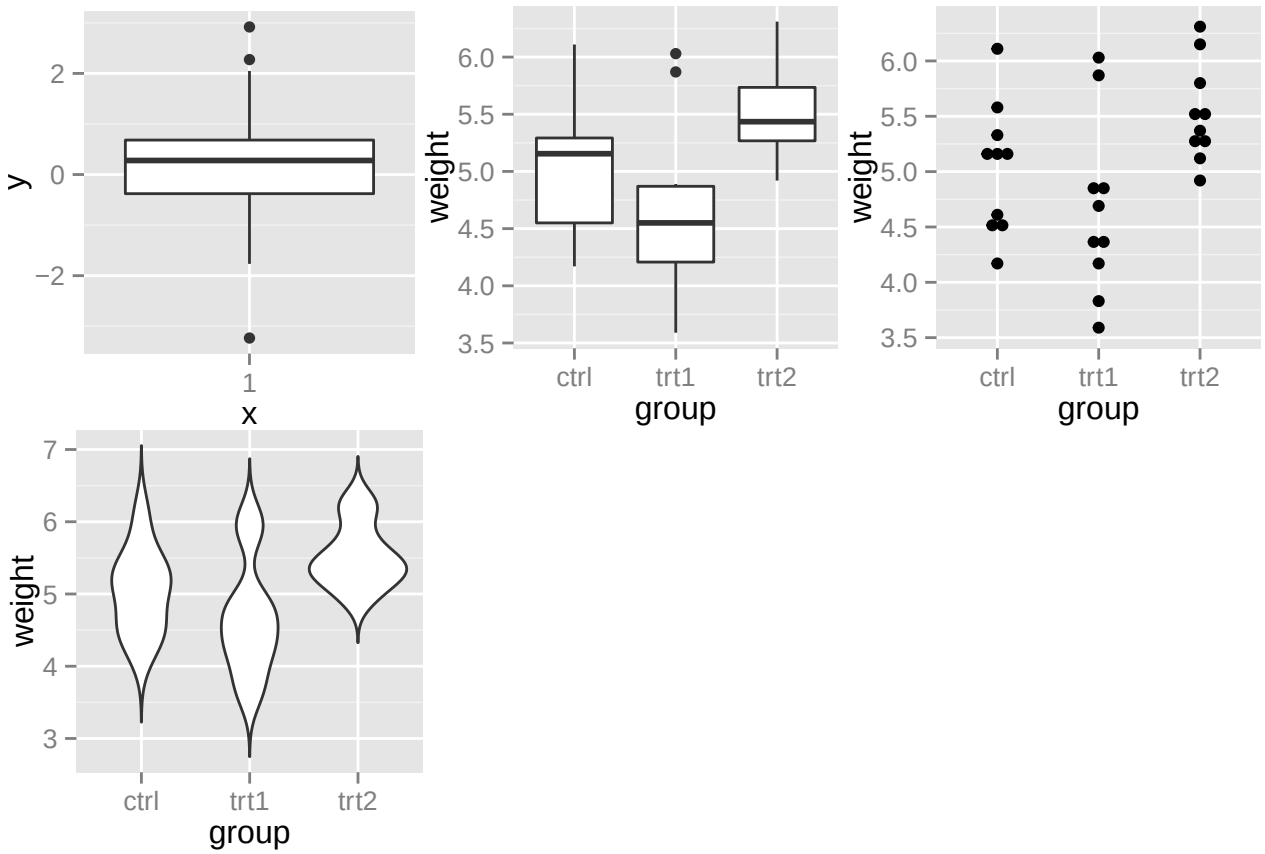
```
##   weight group
## 1   4.17  ctrl
```

```
## 2   5.58  ctrl
## 3   5.18  ctrl
## 4   6.11  ctrl
## 5   4.50  ctrl
## 6   4.61  ctrl
```

The geometries below can be used:

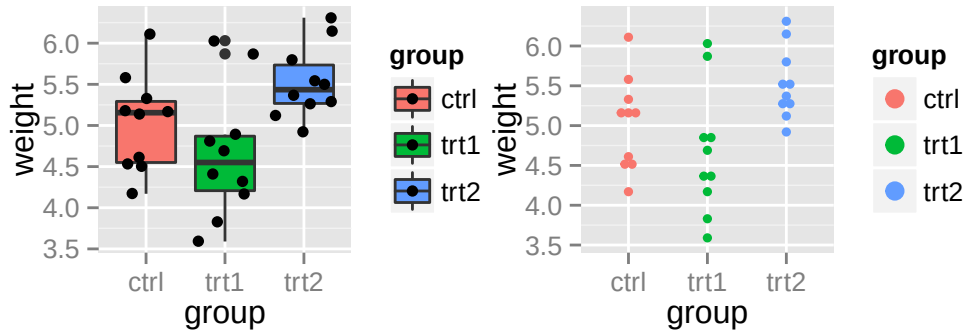
- **geom = "boxplot"**: draws a box plot
- **geom = "dotplot"**: draws a dot plot. The supplementary arguments *stackdir = "center"* and *binaxis = "y"* are also used.
- **geom = "violin"**: draws a violin plot. The argument **trim** is set to FALSE

```r
# Basic box plot from a numeric vector
x <- "1"
y <- rnorm(100)
qplot(x, y, geom = "boxplot")

# Basic box plot from data frame
qplot(group, weight, data = PlantGrowth,
      geom = "boxplot")

# Dot plot
qplot(group, weight, data = PlantGrowth,
      geom = "dotplot",
      stackdir = "center", binaxis = "y")

# Violin plot
qplot(group, weight, data = PlantGrowth,
      geom = "violin", trim = FALSE)
```



Change the color by groups:

```
# Box plot from a data frame
# Add jitter and change fill color by group
qplot(group, weight, data = PlantGrowth,
      geom=c("boxplot", "jitter"), fill = group)

# Dot plot
qplot(group, weight, data = PlantGrowth,
      geom = "dotplot", stackdir = "center", binaxis = "y",
      color = group, fill = group)
```



1.4.6 Histogram and density plots

Histogram and **density** plots are used to display the distribution of data.

1.4.6.1 Data

The R code below generates some data containing the weights by sex (M for male; F for female):

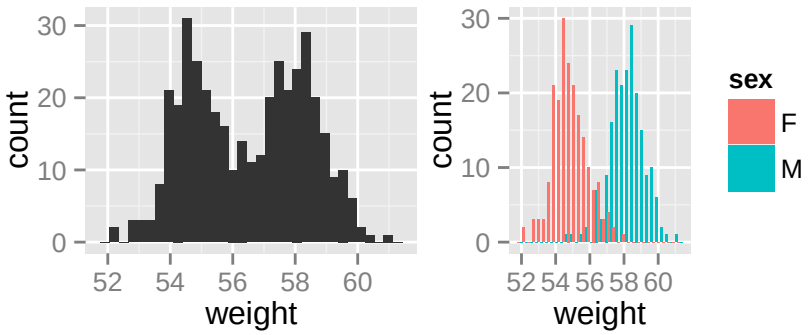
```
set.seed(1234)
mydata = data.frame(
        sex = factor(rep(c("F", "M"), each=200)),
        weight = c(rnorm(200, 55), rnorm(200, 58)))
head(mydata)
```

```
##   sex   weight
## 1   F 53.79293
## 2   F 55.27743
## 3   F 56.08444
## 4   F 52.65430
## 5   F 55.42912
## 6   F 55.50606
```

1.4.6.2 Histogram plots

The argument **geom = "histogram"** is used.

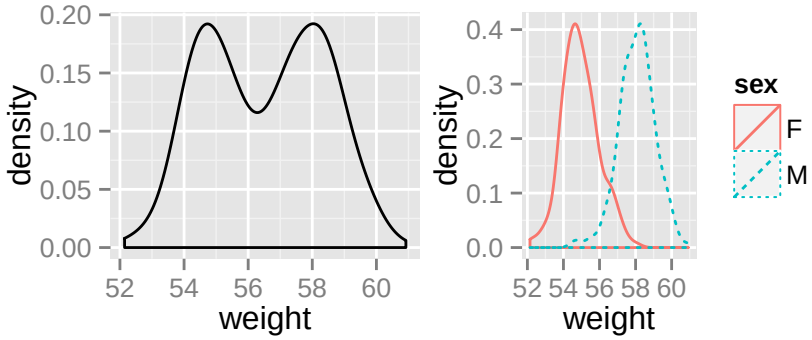
```
# Basic histogram
qplot(weight, data = mydata, geom = "histogram")

# Change histogram fill color by group (sex)
qplot(weight, data = mydata, geom = "histogram",
    fill = sex, position = "dodge")
```



1.4.6.3 Density plots

The argument **geom = "density"** is used:

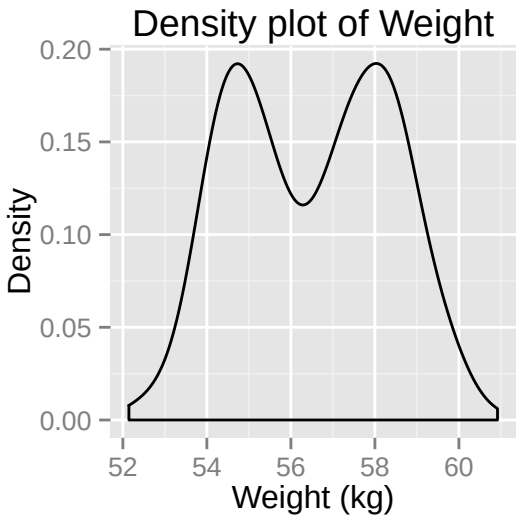
```
# Basic density plot
qplot(weight, data = mydata, geom = "density")

# Change density plot line color by group (sex)
# change line type
qplot(weight, data = mydata, geom = "density",
    color = sex, linetype = sex)
```



1.4.7 Main titles and axis labels

The main title and axis labels can be added to the plot as follow:

```
qplot(weight, data = mydata, geom = "density",
      xlab = "Weight (kg)", ylab = "Density",
      main = "Density plot of Weight")
```



1.5 Introduction to ggplot() function

As mentioned above, there are two main functions in **ggplot2** package for generating graphics:

- The quick and easy-to-use function: **qplot()**
- The more powerful and flexible function to build the plot piece by piece: **ggplot()**

This section describes briefly how to use the function **ggplot()**.

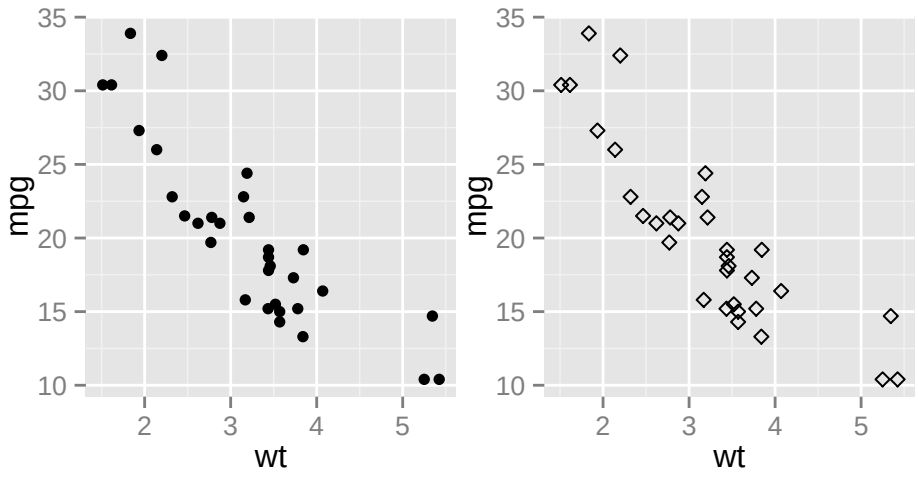
Recall that, the concept of **ggplot** divides a plot in different fundamental parts: **plot = data + Aesthetics + geometry**

- **data**: a data frame. Columns are variables
- **Aesthetics** is used to specify the x and y variables. It can also be used to control the **color**, the **size** or the **shape** of points, ...

- **Geometry** corresponds to the type of graphics (**histogram**, **boxplot**, **line**, **density**, **dotplot**, **bar**, ...)

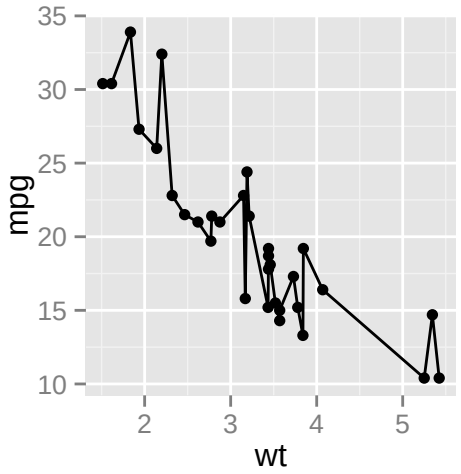
To demonstrate how the function **ggplot()** works, we'll draw a **scatter plot**:

```
# Basic scatter plot
ggplot(data = mtcars, aes(x = wt, y = mpg)) +
  geom_point()

# Change the point size, and shape
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(size = 2, shape = 23)
```



In ggplot2 terminology, the function **geom_point()** is called a layer. You can combine multiple layers as follow:

```
ggplot(data = mtcars, aes(x = wt, y = mpg)) +
  geom_point() + # to draw points
  geom_line() # to draw a line
```

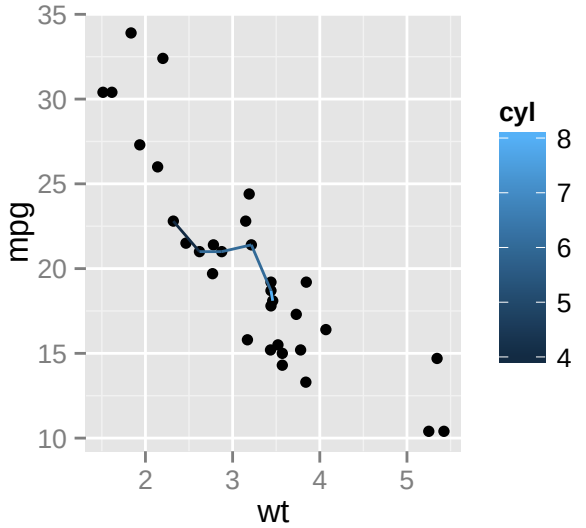


In the R code above, the two layers, **geom_point()** and **geom_line()**, use the same data and the same aesthetic mapping provided in the main function **ggplot**.

Note that, it's possible to use different data and mapping for different layers. For example in the R code below:

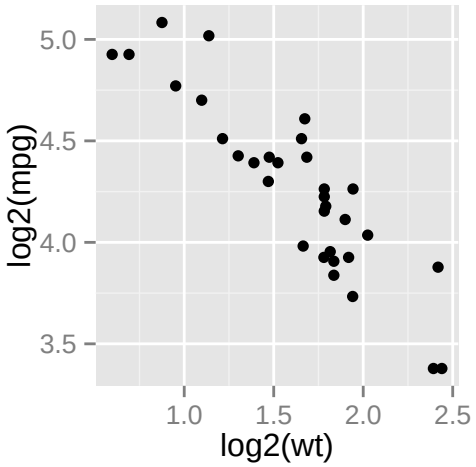
- The entire *mtcars* data is used by the layer *geom_point()*
- A subset of *mtcars* data is used by the layer *geom_line()*. The line is colored according to the values of the continuous variable *cyl*.

```
ggplot(data = mtcars, aes(x = wt, y = mpg)) +
  geom_point() + # to draw points
  geom_line(data = head(mtcars), aes(color = cyl))
```



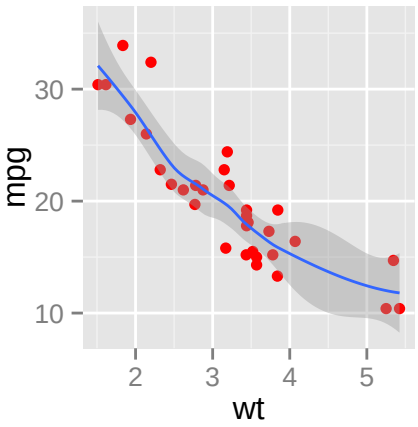
It's also possible to do simple calculations in the function **aes()**:

```
# Log2 transformation in the aes()
ggplot(data = mtcars, aes(x = log2(wt), y = log2(mpg))) +
  geom_point()
```



The function **aes_string()** can be also used for aesthetic mappings from a string objects. An example is shown below:

```
ggplot(data = mtcars, aes_string(x = "wt", y = "mpg")) +
  geom_point(color = "red") +
  geom_smooth()
```

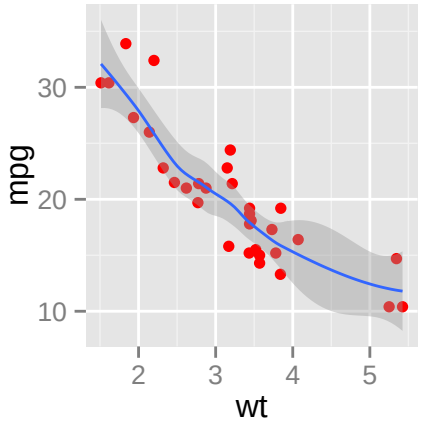


Note that, the function **aes_string()** is particularly useful when writing in functions that create a ggplot (see the R function below).

```
#+++++++++++++++
# Helper function for creating a scatter plot
# +++++++++++++++++++++
# data: data frame
# xName: the name of the column containing the x variable
# yName: the name of the column containing the y variable
ggpoints <- function (data, xName, yName){
  p <- ggplot(data = data, aes_string(xName, yName)) +
      geom_point(color = "red") +
      geom_smooth()

  return(p)
}
```

Create a scatter plot using the helper function **ggpoints()**:

```
ggpoints(mtcars, xName ="wt", yName = "mpg")
```



Note that, the function ggplot() is comprehensively described in the next chapters.

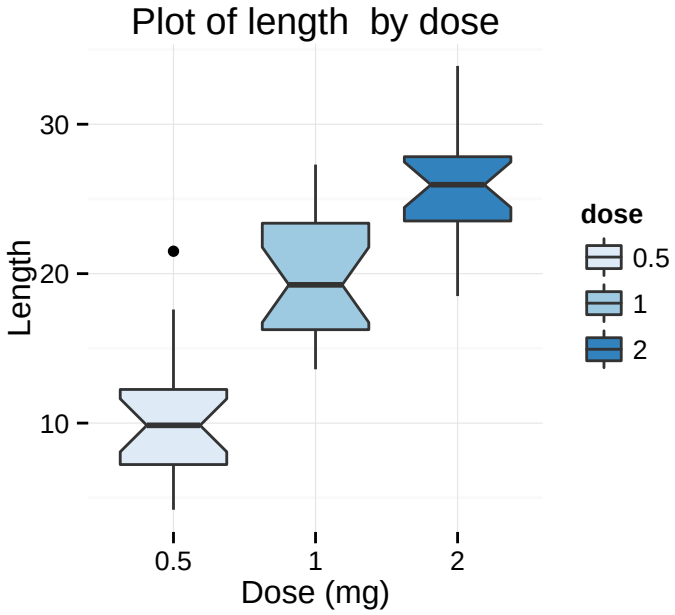
Chapter 2

Box plots

The function **geom_boxplot()** is used to create a **box plot** using **ggplot2**. A simplified format is :

```
geom_boxplot(outlier.colour = "black", outlier.shape = 16,
             outlier.size = 2, notch = FALSE)
```

- **outlier.colour**, **outlier.shape**, **outlier.size**: The color, the shape and the size for outlying points
- **notch**: logical value. If TRUE, makes a **notched box plot**. The notch displays a confidence interval around the median which is normally based on the median +/- 1.58*IQR/sqrt(n). Notches are used to compare groups; if the notches of two boxes do not overlap, this is a strong evidence that the medians differ.



Key functions: **geom_boxplot()**, **stat_boxplot()** and **stat_summary()**.

2.1 Prepare the data

ToothGrowth data is used :

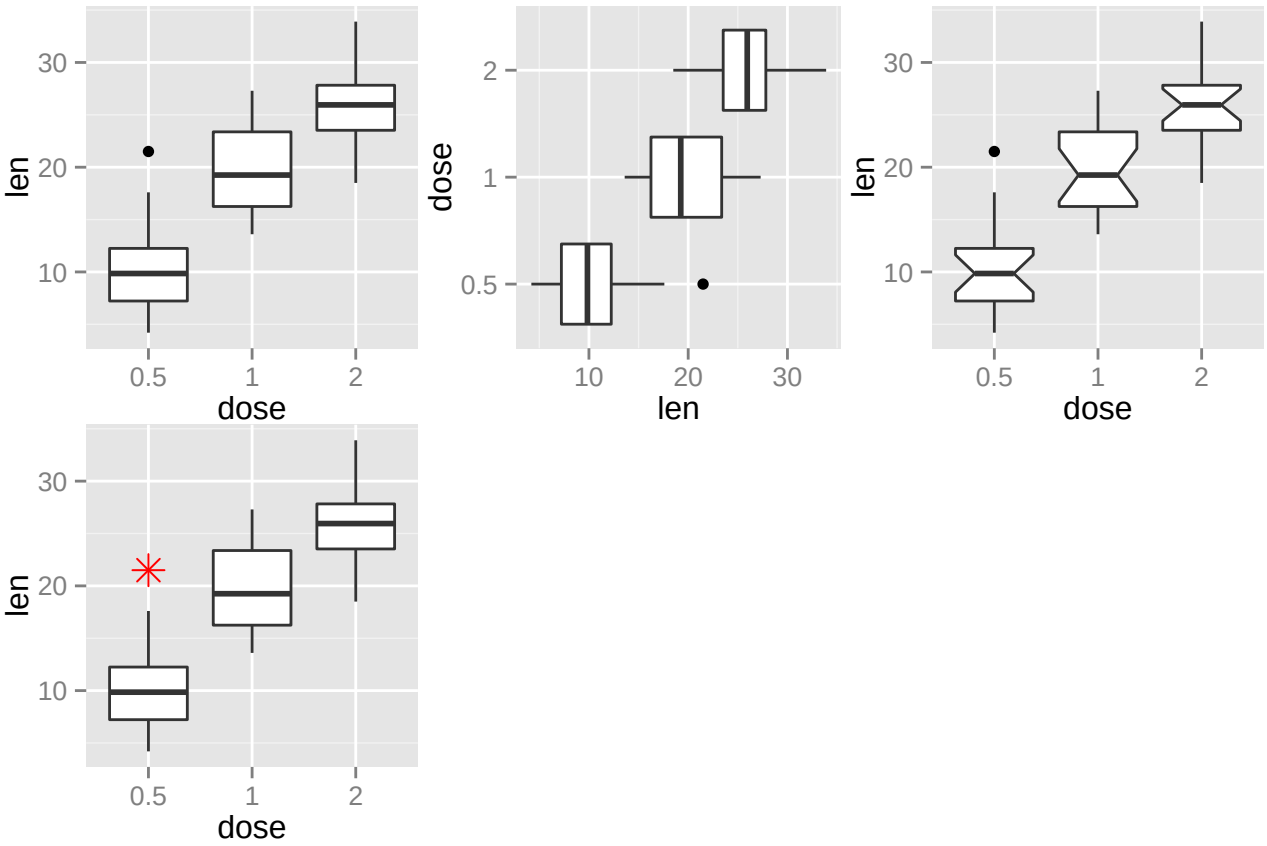
```
# Convert the variable dose from a numeric to a factor variable
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that the variable "dose" is converted as a factor variable using the above R script.

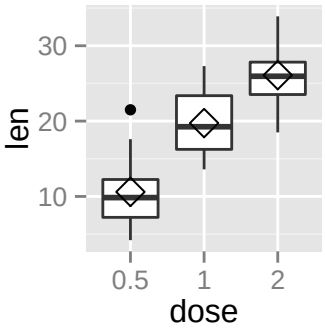
2.2 Basic box plots

```
library(ggplot2)
# Basic box plot
p <- ggplot(ToothGrowth, aes(x = dose, y = len)) +
  geom_boxplot()
p

# Rotate the box plot
p + coord_flip()

# Notched box plot
ggplot(ToothGrowth, aes(x = dose, y = len)) +
  geom_boxplot(notch = TRUE)

# Change outlier, color, shape and size
ggplot(ToothGrowth, aes(x = dose, y = len)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 8,
               outlier.size = 4)
```



The function **stat_summary()** can be used to add mean points to a box plot :

```
# Box plot with mean points
p + stat_summary(fun.y = mean, geom = "point", shape = 23, size = 4)
```



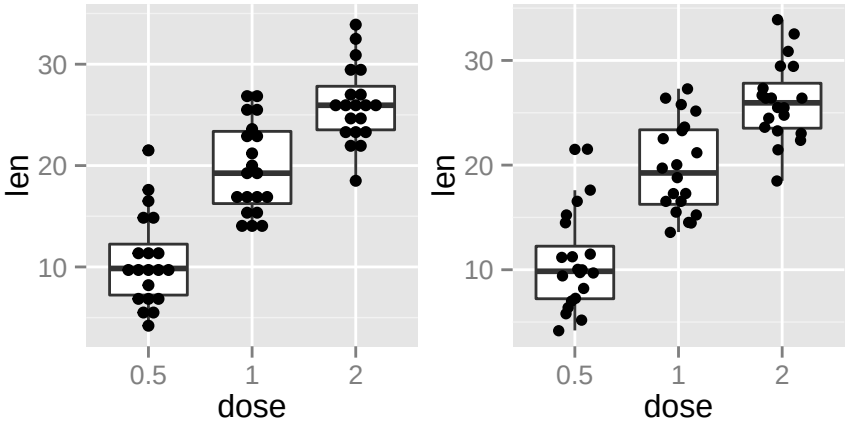
Choose which items to display :

```
p + scale_x_discrete(limits=c("0.5", "2"))
```

2.3 Box plot with dots

Dots (or points) can be added to a box plot using the functions **geom_dotplot()** or **geom_jitter()** :

```
# Box plot with dot plot
p + geom_dotplot(binaxis = 'y', stackdir = 'center', dotsize = 1)

# Box plot with jittered points
# 0.2 : degree of jitter in x direction
p + geom_jitter(shape = 16, position = position_jitter(0.2))
```

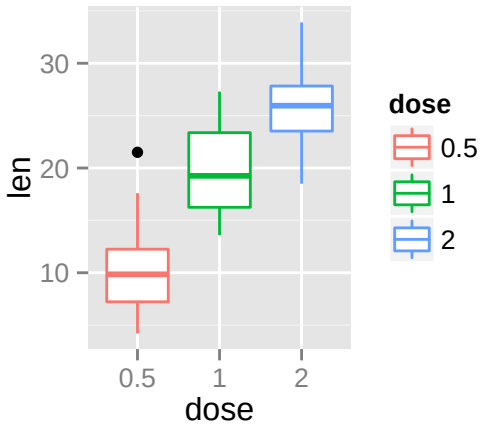


2.4 Change box plot colors by groups

2.4.1 Change box plot line colors

Box plot line colors can be automatically controlled by the levels of the variable *dose* :

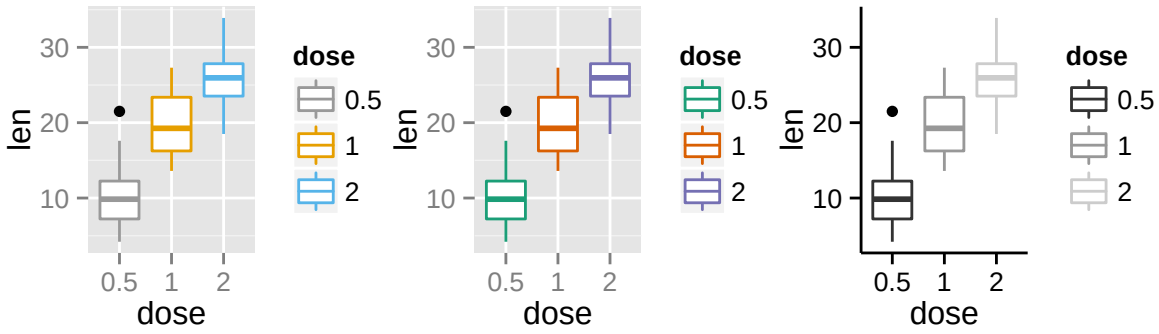
```
# Change box plot line colors by groups
p<-ggplot(ToothGrowth, aes(x=dose, y=len, color=dose)) +
  geom_boxplot()
p
```



It is also possible to *change manually box plot line colors* using the functions :

- *scale_color_manual()* : to use custom colors
- *scale_color_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_color_grey()* : to use grey color palettes

```
# Use custom color palettes
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
p+scale_color_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey() + theme_classic()
```

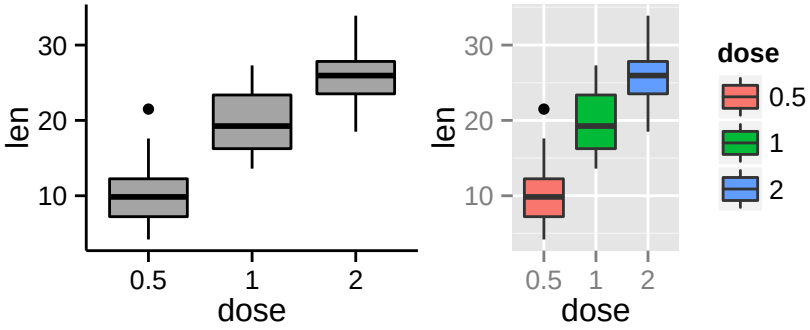


Read more on ggplot2 colors: Chapter 18

2.4.2 Change box plot fill colors

In the R code below, box plot fill colors are automatically controlled by the levels of *dose* :

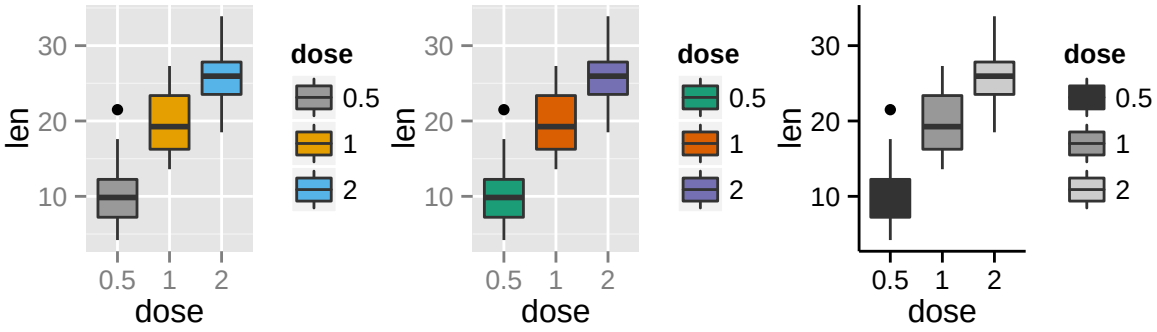
```
# Use single color
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_boxplot(fill='#A4A4A4', color="black")+
  theme_classic()

# Change box plot colors by groups
p<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_boxplot()
p
```



It is also possible to change manually box plot fill colors using the functions :

- *scale_fill_manual()* : to use custom colors
- *scale_fill_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_fill_grey()* : to use grey color palettes

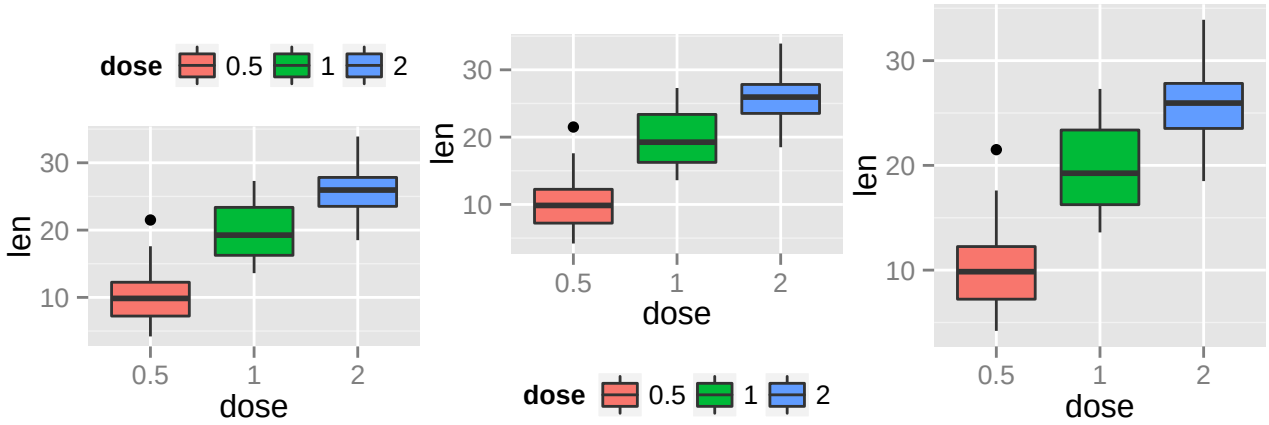
```
# Use custom color palettes
p+scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# use brewer color palettes
p+scale_fill_brewer(palette="Dark2")

# Use grey scale
p + scale_fill_grey() + theme_classic()
```



Read more on ggplot2 colors: Chapter 18

2.5 Change the legend position

```
p + theme(legend.position="top")

p + theme(legend.position="bottom")

p + theme(legend.position="none") # Remove legend
```



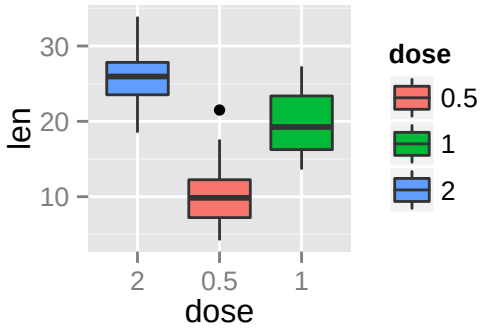
The allowed values for the arguments **legend.position** are : "left","top", "right", "bottom".

Read more on ggplot legend : Chapter 17

2.6 Change the order of items in the legend

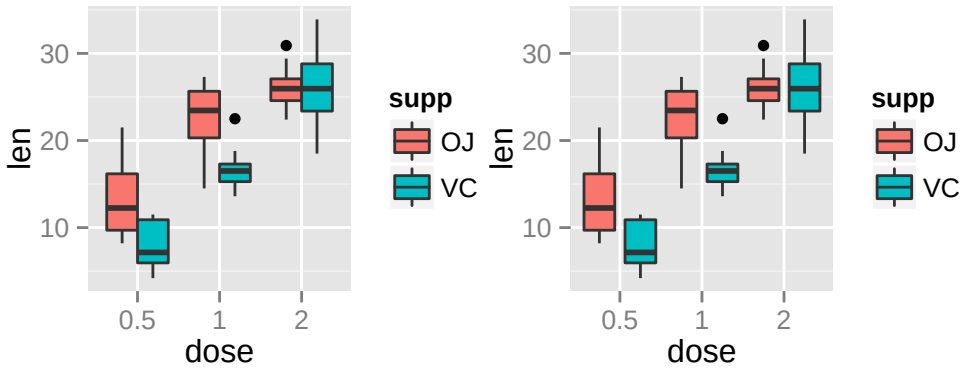
The function **scale_x_discrete** can be used to change the order of items from c("0.5", "1", "2") to c("2", "0.5", "1"):

```
p + scale_x_discrete(limits=c("2", "0.5", "1"))
```



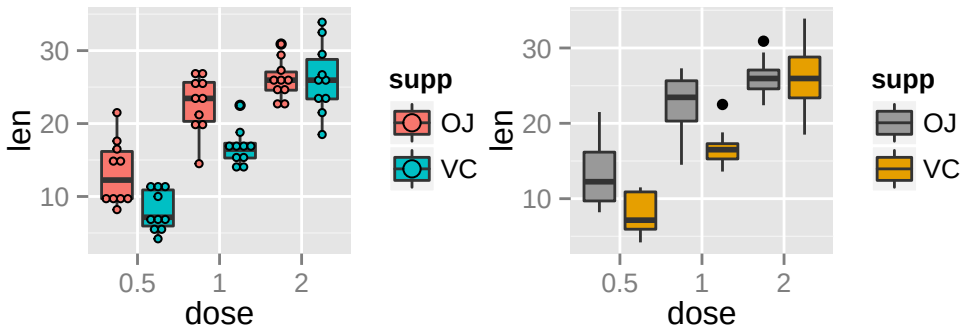
2.7 Box plot with multiple groups

```
# Change box plot colors by groups
ggplot(ToothGrowth, aes(x=dose, y=len, fill=supp)) +
  geom_boxplot()

# Change the position
p<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=supp)) +
  geom_boxplot(position=position_dodge(1))
p
```



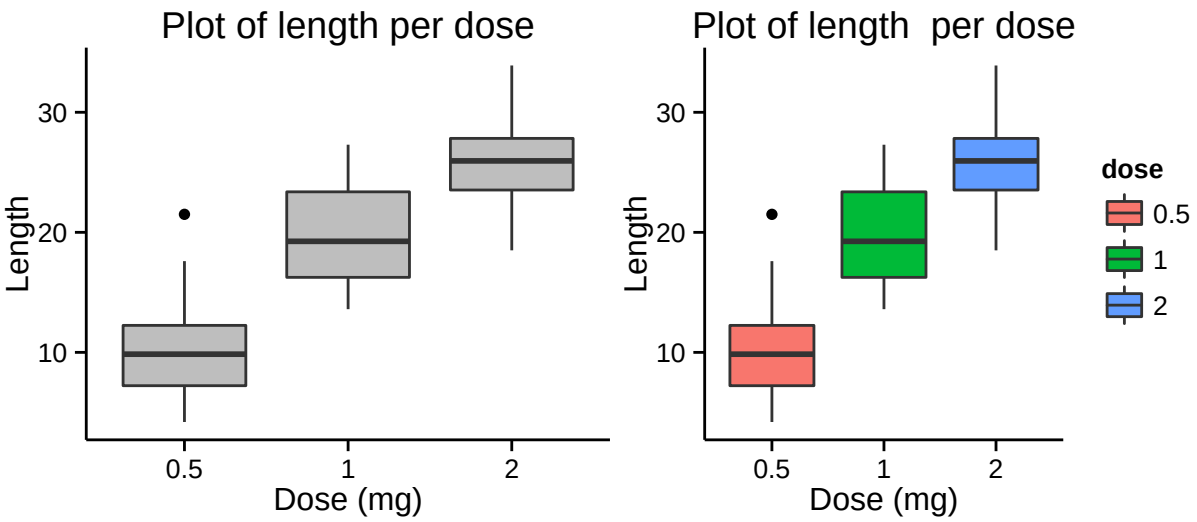
Change box plot colors and add dots :

```
# Add dots
p + geom_dotplot(binaxis='y', stackdir='center',
                 position=position_dodge(1))

# Change colors
p+scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))
```



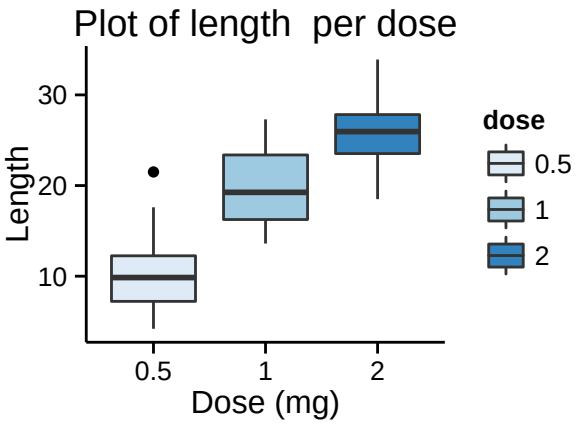
2.8 Customized box plots

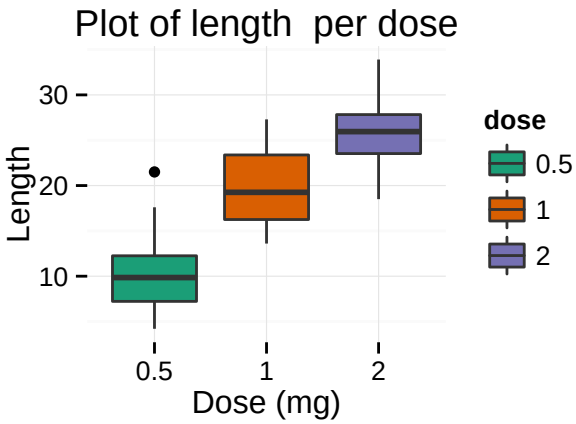
```
# Basic box plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_boxplot(fill="gray")+
  labs(title="Plot of length per dose",x="Dose (mg)", y = "Length")+
  theme_classic()

# Change  automatically color by groups
bp <- ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_boxplot()+
  labs(title="Plot of length  per dose",x="Dose (mg)", y = "Length")

bp + theme_classic()
```

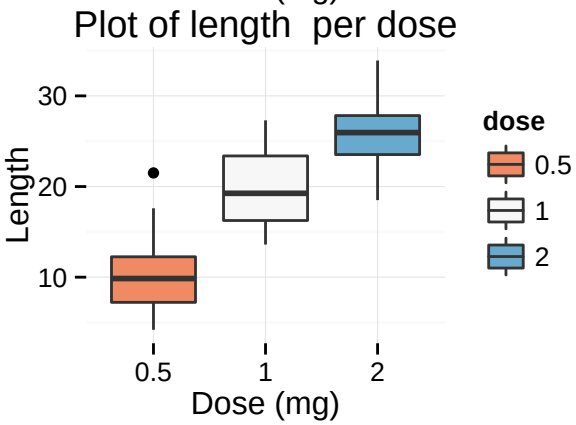


Change fill colors manually :

```
# Continuous colors
bp + scale_fill_brewer(palette="Blues") + theme_classic()

# Discrete colors
bp + scale_fill_brewer(palette="Dark2") + theme_minimal()

# Gradient colors
bp + scale_fill_brewer(palette="RdBu") + theme_minimal()
```







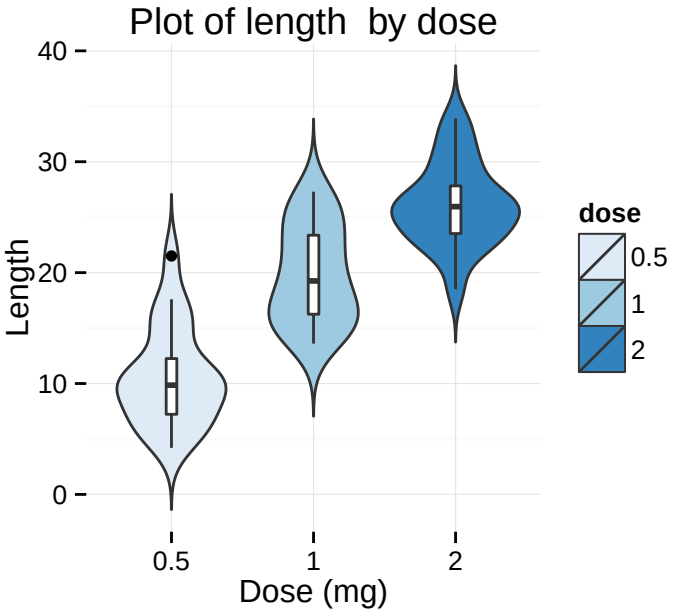
Read more on ggplot2 colors here : Chapter 18

Chapter 3

Violin plots

Violin plots are similar to **box plots** (Chapter 2), except that they also show the kernel probability density of the data at different values. Typically, violin plots will include a marker for the median of the data and a box indicating the interquartile range, as in standard box plots.

The function **geom_violin()** is used to produce a violin plot.



Key functions: **geom_violin()**, **stat_ydensity()**.

3.1 Prepare the data

ToothGrowth data is used :

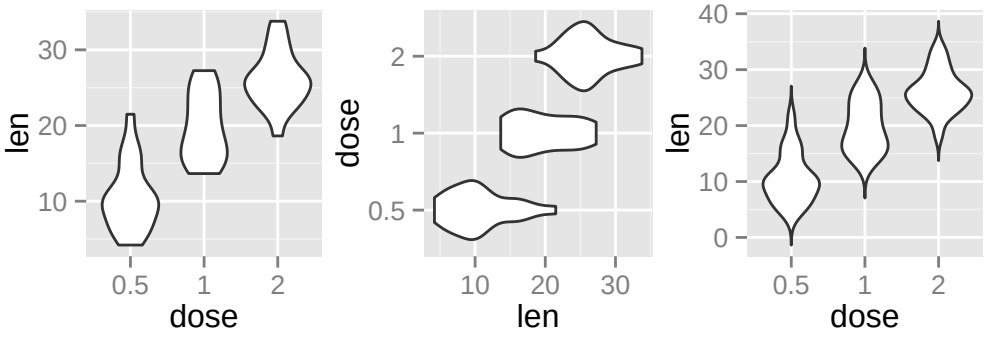
```
# Convert the variable dose from a numeric to a factor variable
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that the variable "dose" is converted as a factor variable using the above R script.

3.2 Basic violin plots

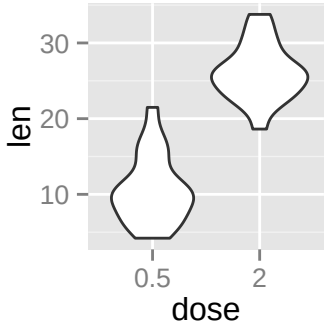
```
library(ggplot2)
# Basic violin plot
p <- ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_violin()
p

# Rotate the violin plot
p + coord_flip()

# Set trim argument to FALSE
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_violin(trim=FALSE)
```



Note that by default *trim = TRUE*. In this case, the tails of the violins are trimmed. If *FALSE*, the tails are not trimmed.

Choose which items to display :

```
p + scale_x_discrete(limits=c("0.5", "2"))
```

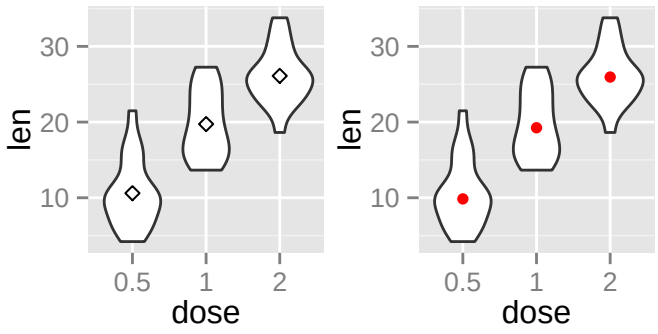


3.3 Add summary statistics on a violin plot

The function **stat_summary()** can be used to add mean/median points and more on a violin plot.

3.3.1 Add mean and median points

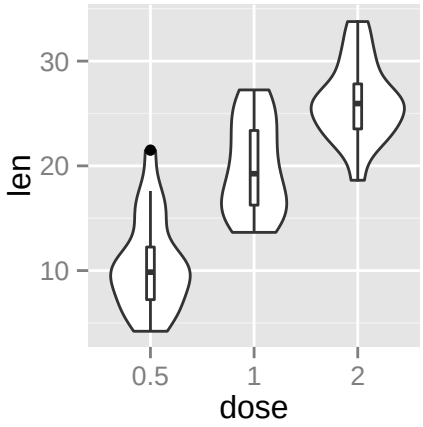
```
# Violin plot with mean points
p + stat_summary(fun.y=mean, geom="point", shape=23, size=2)

# Violin plot with median points
p + stat_summary(fun.y=median, geom="point", size=2, color="red")
```



3.3.2 Add median and quartiles

A solution is to use the function **geom_boxplot** :

```
p + geom_boxplot(width=0.1)
```



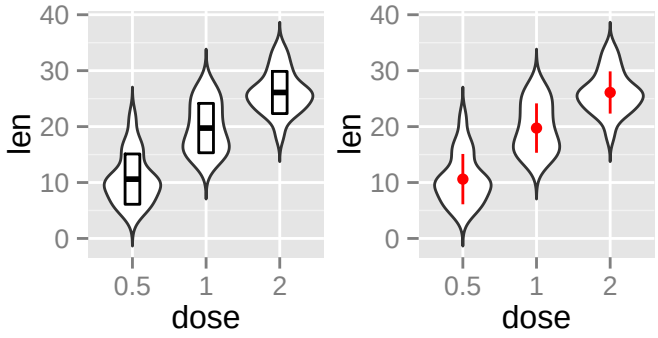
3.3.3 Add mean and standard deviation

The function **mean_sdl** is used. *mean_sdl* computes the *mean* plus or minus a *constant* times the *standard deviation*.

In the R code below, the constant is specified using the argument *mult* (mult = 1). By default mult = 2.

The mean +/- SD can be added as a *crossbar* or a *pointrange* :

```
p <- ggplot(ToothGrowth, aes(x=dose, y=len)) +
    geom_violin(trim=FALSE)

p + stat_summary(fun.data="mean_sdl", mult=1,
                 geom="crossbar", width=0.2 )

p + stat_summary(fun.data=mean_sdl, mult=1,
                 geom="pointrange", color="red")
```

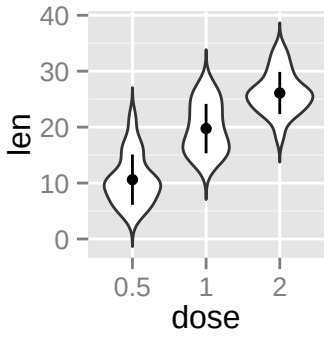


Note that, you can also define a custom function to produce summary statistics as follow :

```
# Function to produce summary statistics (mean and +/- sd)
data_summary <- function(x) {
   m <- mean(x)
   ymin <- m-sd(x)
   ymax <- m+sd(x)
   return(c(y=m,ymin=ymin,ymax=ymax))
}
```

Use a custom summary function :

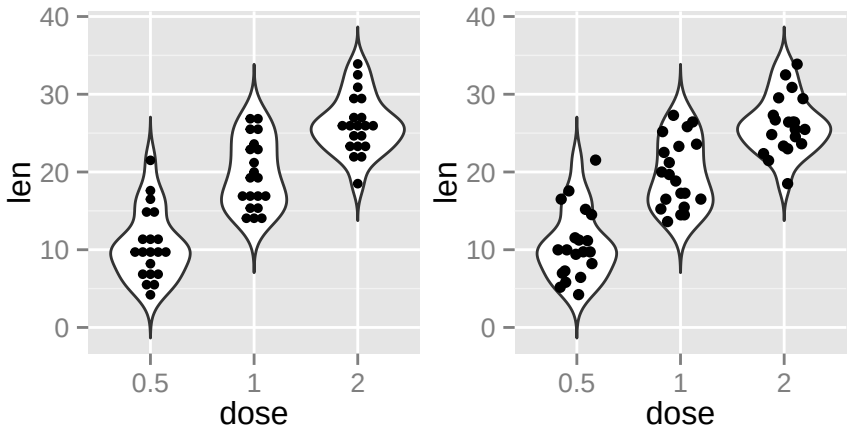
```
p + stat_summary(fun.data=data_summary)
```



3.4 Violin plot with dots

Dots (or points) can be added to a violin plot using the functions **geom_dotplot()** or **geom_jitter()** :

```
# violin plot with dot plot
p + geom_dotplot(binaxis='y', stackdir='center', dotsize=1)

# violin plot with jittered points
# 0.2 : degree of jitter in x direction
p + geom_jitter(shape=16, position=position_jitter(0.2))
```

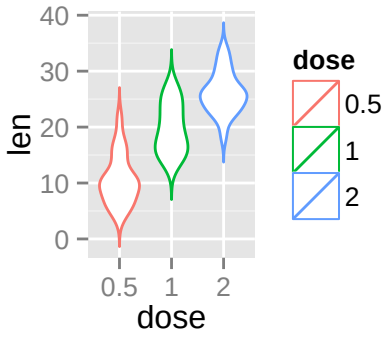


3.5 Change violin plot colors by groups

3.5.1 Change violin plot line colors

Violin plot line colors can be automatically controlled by the levels of *dose* :

```
# Change violin plot line colors by groups
p<-ggplot(ToothGrowth, aes(x=dose, y=len, color=dose)) +
  geom_violin(trim=FALSE)
p
```

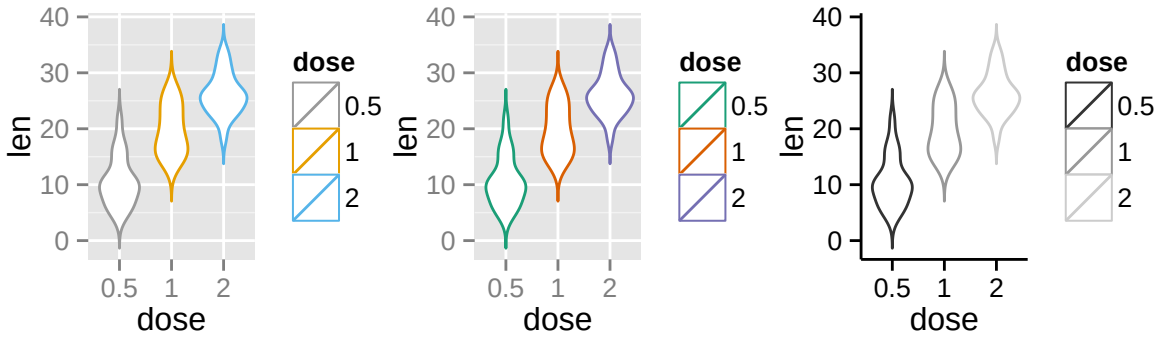


It is also possible to *change manually violin plot line colors* using the functions :

- *scale_color_manual()* : to use custom colors
- *scale_color_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_color_grey()* : to use grey color palettes

```
# Use custom color palettes
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
```

```
p+scale_color_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey() + theme_classic()
```

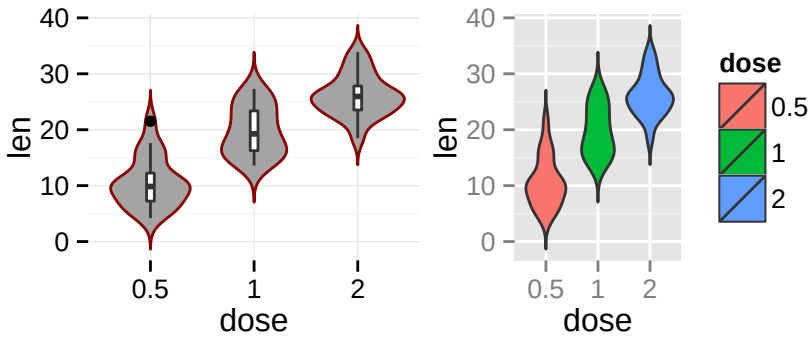


Read more on ggplot2 colors: Chapter 18

3.5.2 Change violin plot fill colors

In the R code below, the fill colors of the violin plot are automatically controlled by the levels of *dose* :

```
# Use single color
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_violin(trim=FALSE, fill='#A4A4A4', color="darkred")+
  geom_boxplot(width=0.1) + theme_minimal()

# Change violin plot colors by groups
p<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_violin(trim=FALSE)
p
```

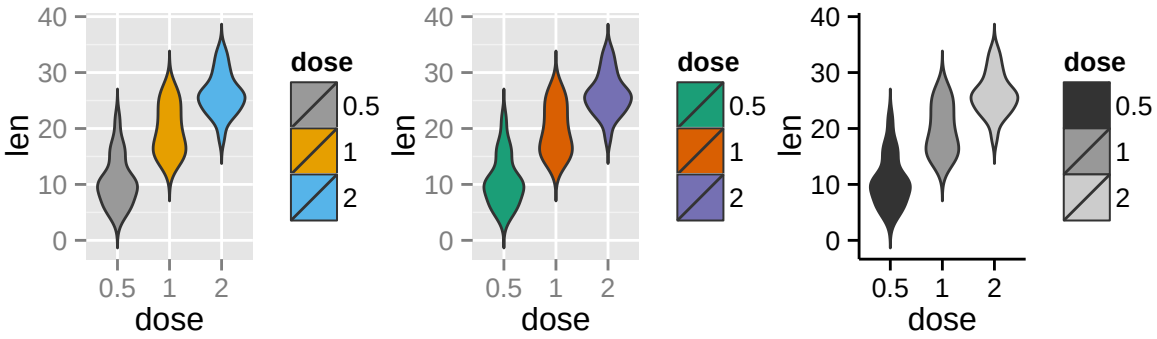


It is also possible to *change manually violin plot colors* using the functions :

- *scale_fill_manual()* : to use custom colors

- *scale_fill_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_fill_grey()* : to use grey color palettes

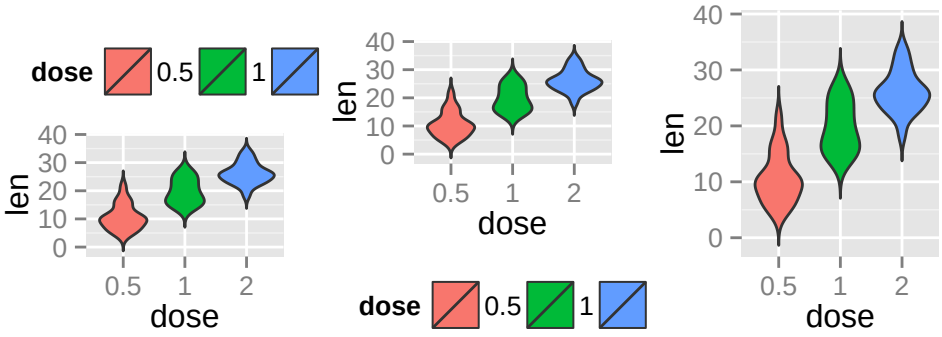
```
# Use custom color palettes
p+scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
p+scale_fill_brewer(palette="Dark2")

# Use grey scale
p + scale_fill_grey() + theme_classic()
```



Read more on ggplot2 colors: Chapter 18

3.6 Change the legend position

```
p + theme(legend.position="top")

p + theme(legend.position="bottom")

p + theme(legend.position="none") # Remove legend
```



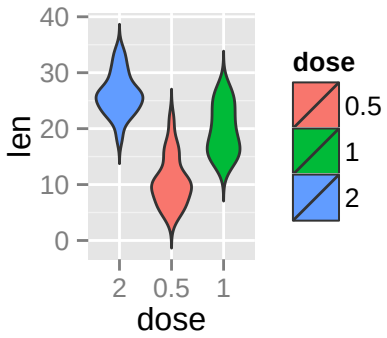
The allowed values for the arguments **legend.position** are : "left","top", "right", "bottom".

Read more on ggplot legends : Chapter 17

3.7 Change the order of items in the legend

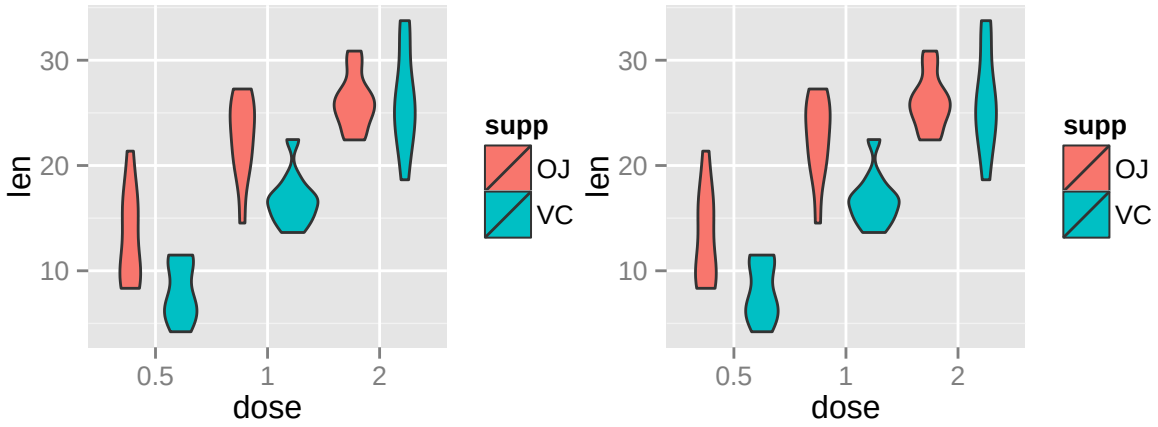
The function **scale_x_discrete** can be used to change the order of items to "2", "0.5", "1" :

```
p + scale_x_discrete(limits=c("2", "0.5", "1"))
```



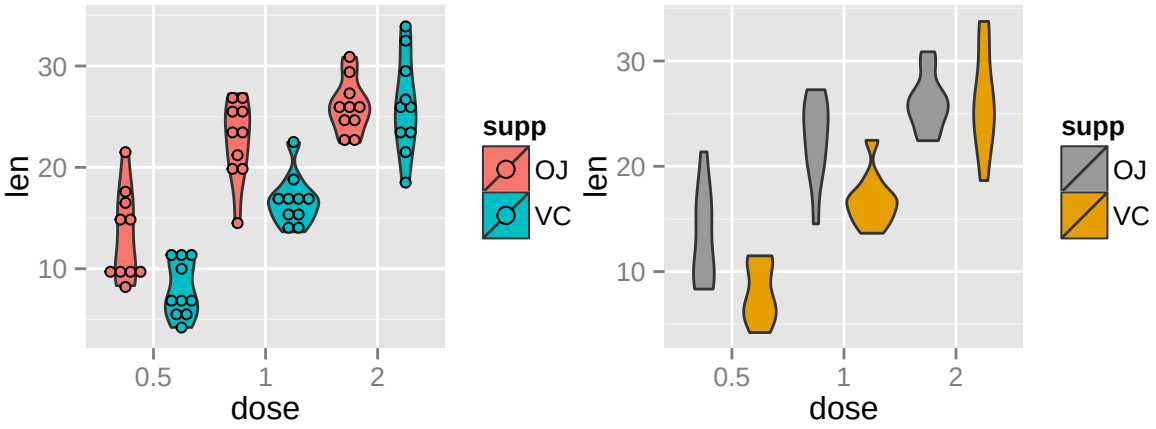
3.8 Violin plot with multiple groups

```
# Change violin plot colors by groups
ggplot(ToothGrowth, aes(x=dose, y=len, fill=supp)) +
  geom_violin()

# Change the position
p<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=supp)) +
  geom_violin(position=position_dodge(1))
p
```



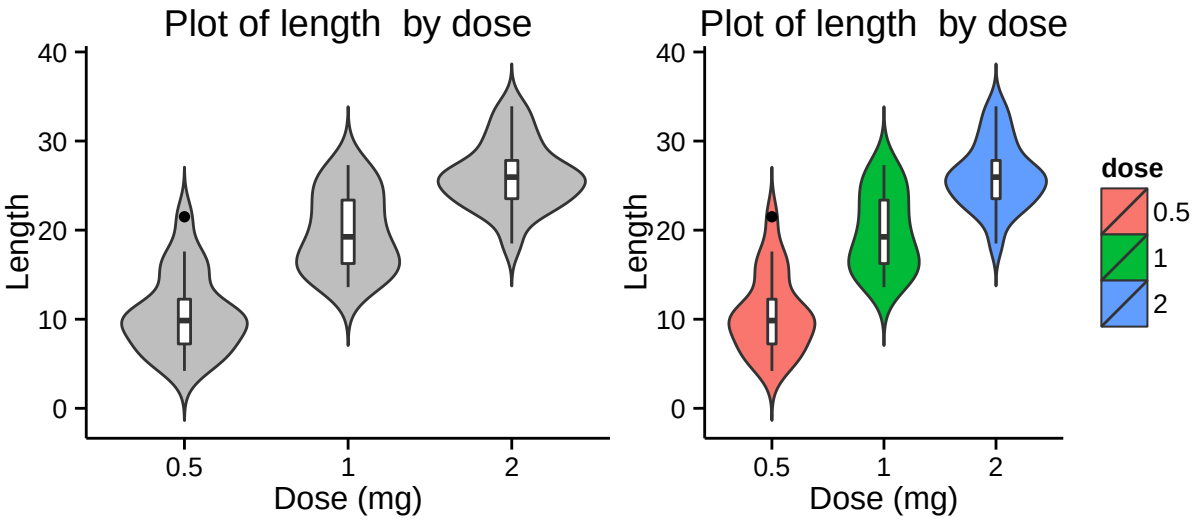
Change violin plot colors and add dots :

```
# Add dots
p + geom_dotplot(binaxis='y', stackdir='center',
                 position=position_dodge(1))

# Change colors
p+scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))
```



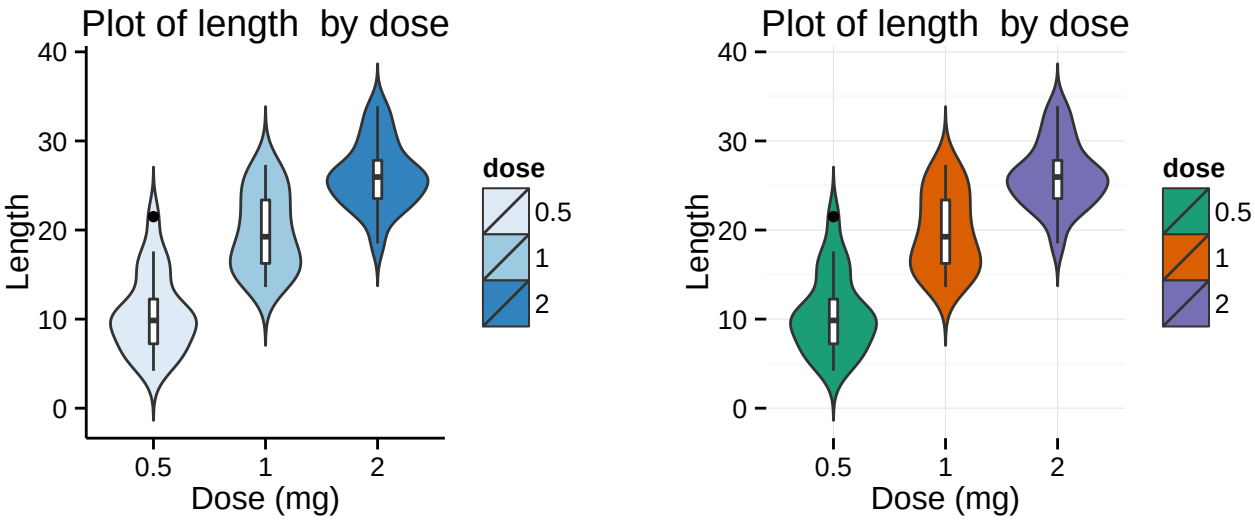
3.9 Customized violin plots

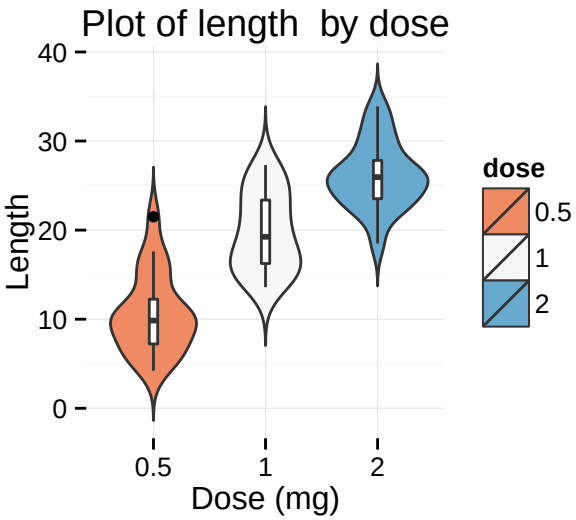
```
# Basic violin plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_violin(trim=FALSE, fill="gray")+
  labs(title="Plot of length  by dose",x="Dose (mg)", y = "Length")+
  geom_boxplot(width=0.1)+
  theme_classic()

# Change color by groups
dp <- ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_violin(trim=FALSE)+
  geom_boxplot(width=0.1, fill="white")+
  labs(title="Plot of length  by dose",x="Dose (mg)", y = "Length")

dp + theme_classic()
```



Change fill colors manually :

```
# Continuous colors
dp + scale_fill_brewer(palette="Blues") + theme_classic()

# Discrete colors
dp + scale_fill_brewer(palette="Dark2") + theme_minimal()

# Gradient colors
dp + scale_fill_brewer(palette="RdBu") + theme_minimal()
```



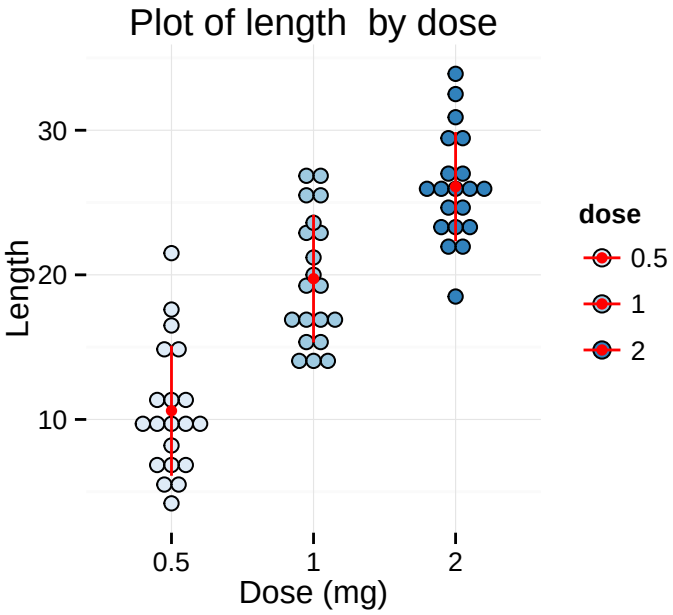


Read more on ggplot2 colors: Chapter 18

Chapter 4

Dot plots

The function **geom_dotplot()** is used to create a **dot plot** using **ggplot2**.



Key functions: **geom_dotplot()**, **stat_bindot()**

4.1 Prepare the data

ToothGrowth data is used :

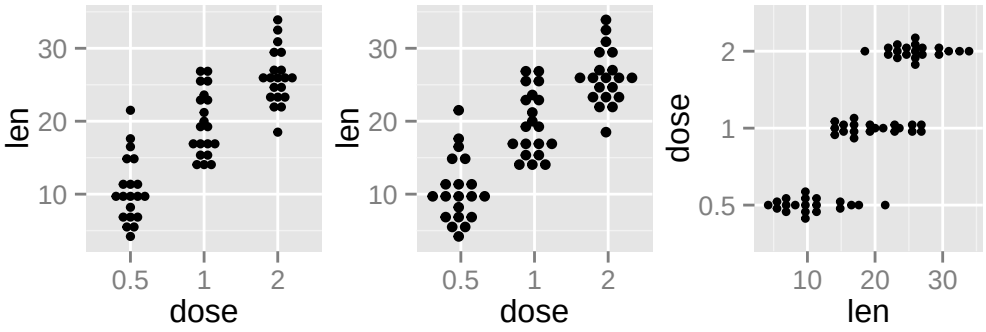
```
# Convert the variable dose from a numeric to a factor variable
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that the variable "dose" is converted as a factor variable using the above R script.

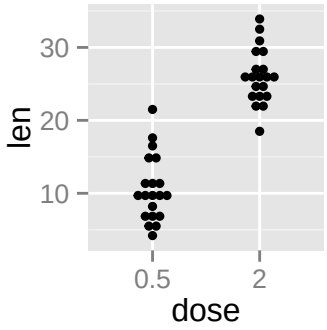
4.2 Basic dot plots

```
library(ggplot2)
# Basic dot plot
p<-ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_dotplot(binaxis='y', stackdir='center')
p

# Change dotsize and stack ratio
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_dotplot(binaxis='y', stackdir='center',
               stackratio=1.5, dotsize=1.2)

# Rotate the dot plot
p + coord_flip()
```



Choose which items to display :

```
p + scale_x_discrete(limits=c("0.5", "2"))
```

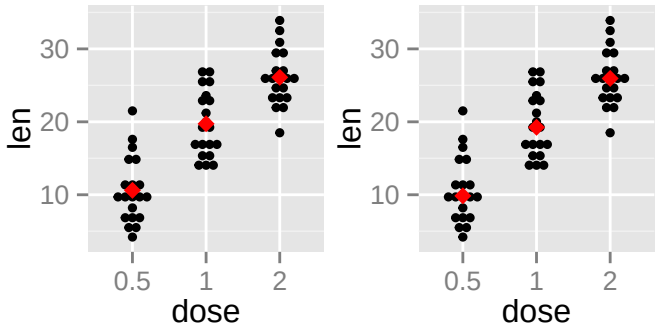


4.3 Add summary statistics on a dot plot

The function **stat_summary()** can be used to add mean/median points and more to a dot plot.

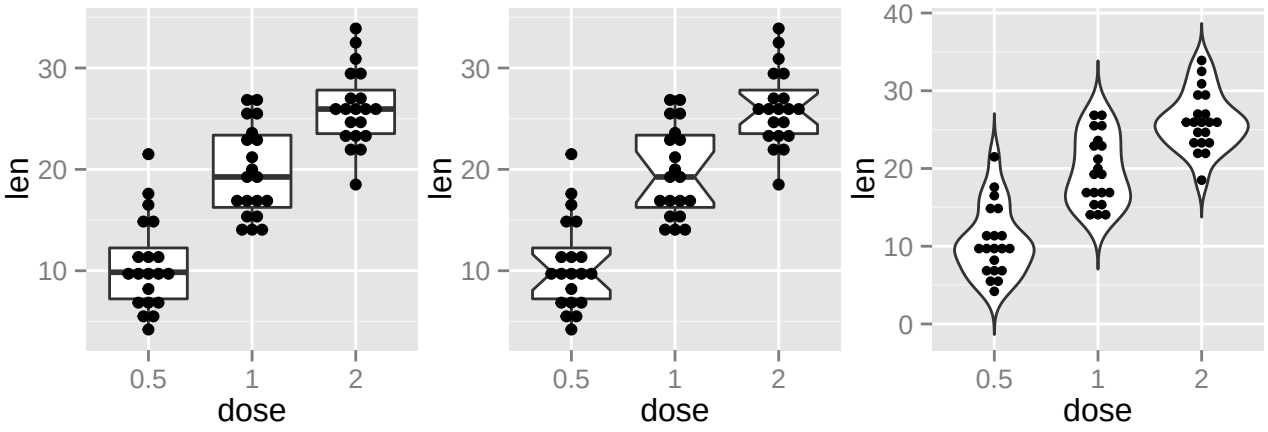
4.3.1 Add mean and median points

```
# dot plot with mean points
p + stat_summary(fun.y=mean, geom="point", shape=18,
                 size=3, color="red")

# dot plot with median points
p + stat_summary(fun.y=median, geom="point", shape=18,
                 size=3, color="red")
```



4.3.2 Dot plot with box plot and violin plot

```
# Add basic box plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_boxplot()+
  geom_dotplot(binaxis='y', stackdir='center')

# Add notched box plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_boxplot(notch = TRUE)+
  geom_dotplot(binaxis='y', stackdir='center')

# Add violin plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_violin(trim = FALSE)+
  geom_dotplot(binaxis='y', stackdir='center')
```



Read more on box plot : Chapter 2

Read more on violin plot : Chapter 3

4.3.3 Add mean and standard deviation

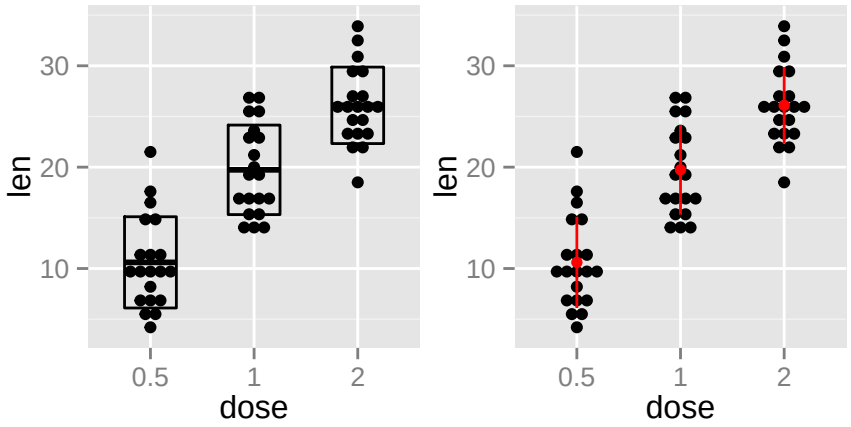
The function **mean_sdl** is used. *mean_sdl* computes the *mean* plus or minus a *constant* times the *standard deviation*.

In the R code below, the constant is specified using the argument *mult* (mult = 1). By default mult = 2.

The mean +/- SD can be added as a *crossbar* or a *pointrange* :

```
p <- ggplot(ToothGrowth, aes(x=dose, y=len)) +
    geom_dotplot(binaxis='y', stackdir='center')

p + stat_summary(fun.data="mean_sdl", mult=1,
```

```
                  geom="crossbar", width=0.5)

p + stat_summary(fun.data=mean_sdl, mult=1,
                  geom="pointrange", color="red")
```

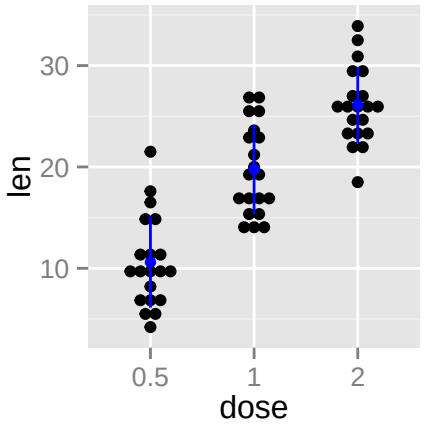


Note that, you can also define a custom function to produce summary statistics as follow.

```
# Function to produce summary statistics (mean and +/- sd)
data_summary <- function(x) {
   m <- mean(x)
   ymin <- m-sd(x)
   ymax <- m+sd(x)
   return(c(y=m,ymin=ymin,ymax=ymax))
}
```

Use a custom summary function :

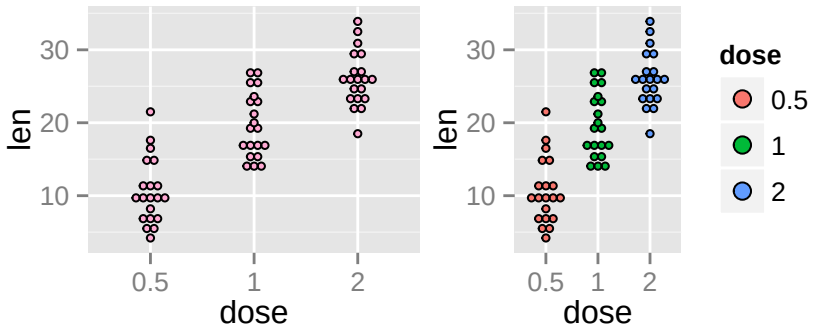
```
p + stat_summary(fun.data=data_summary, color="blue")
```



4.4 Change dot plot colors by groups

In the R code below, the fill colors of the dot plot are automatically controlled by the levels of *dose* :

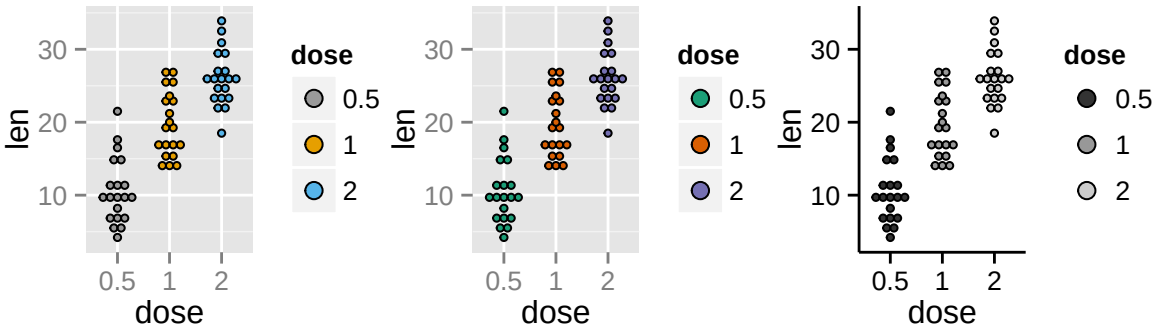
```
# Use single fill color
ggplot(ToothGrowth, aes(x=dose, y=len)) +
    geom_dotplot(binaxis='y', stackdir='center', fill="#FFAAD4")

# Change dot plot colors by groups
p<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_dotplot(binaxis='y', stackdir='center')
p
```



It is also possible to *change manually dot plot colors* using the functions :

- *scale_fill_manual()* : to use custom colors
- *scale_fill_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_fill_grey()* : to use grey color palettes

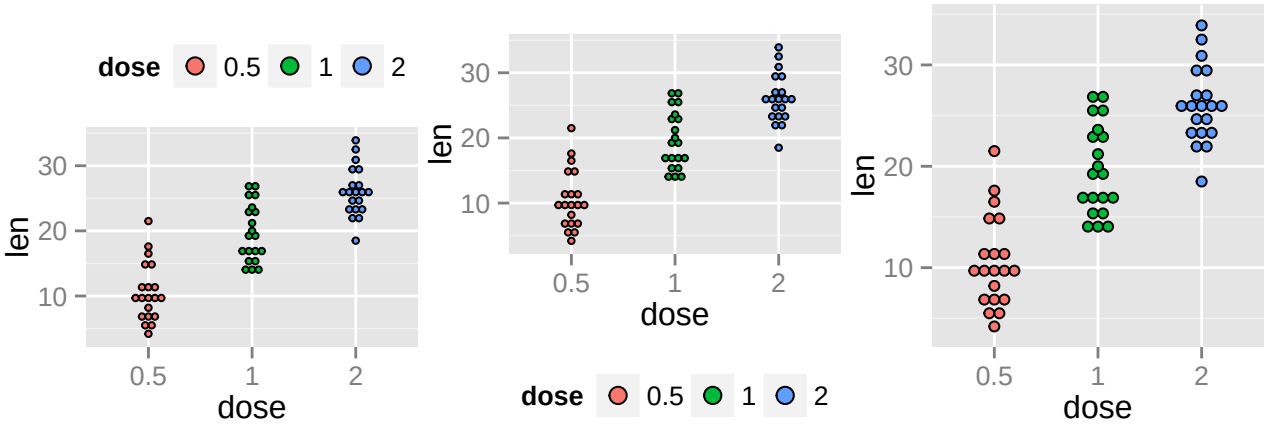
```
# Use custom color palettes
p+scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
p+scale_fill_brewer(palette="Dark2")

# Use grey scale
p + scale_fill_grey() + theme_classic()
```



Read more on ggplot2 colors here : Chapter 18

4.5 Change the legend position

```
p + theme(legend.position="top")

p + theme(legend.position="bottom")

p + theme(legend.position="none") # Remove legend
```



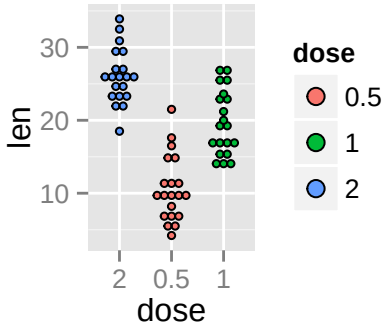
The allowed values for the arguments **legend.position** are : "left","top", "right", "bottom".

Read more on ggplot legends : Chapter 17

4.6 Change the order of items in the legend

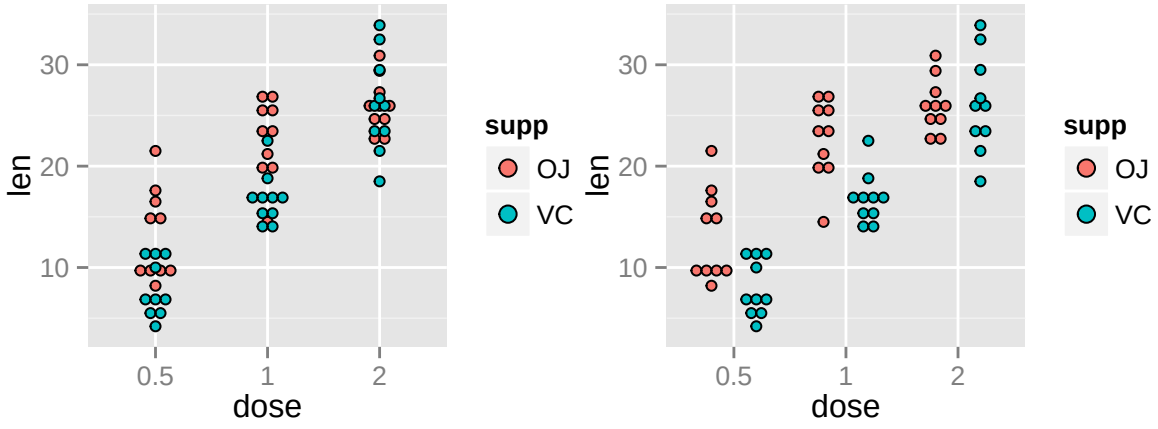
The function **scale_x_discrete** can be used to change the order of items to "2", "0.5", "1" :

```
p + scale_x_discrete(limits=c("2", "0.5", "1"))
```



4.7 Dot plot with multiple groups

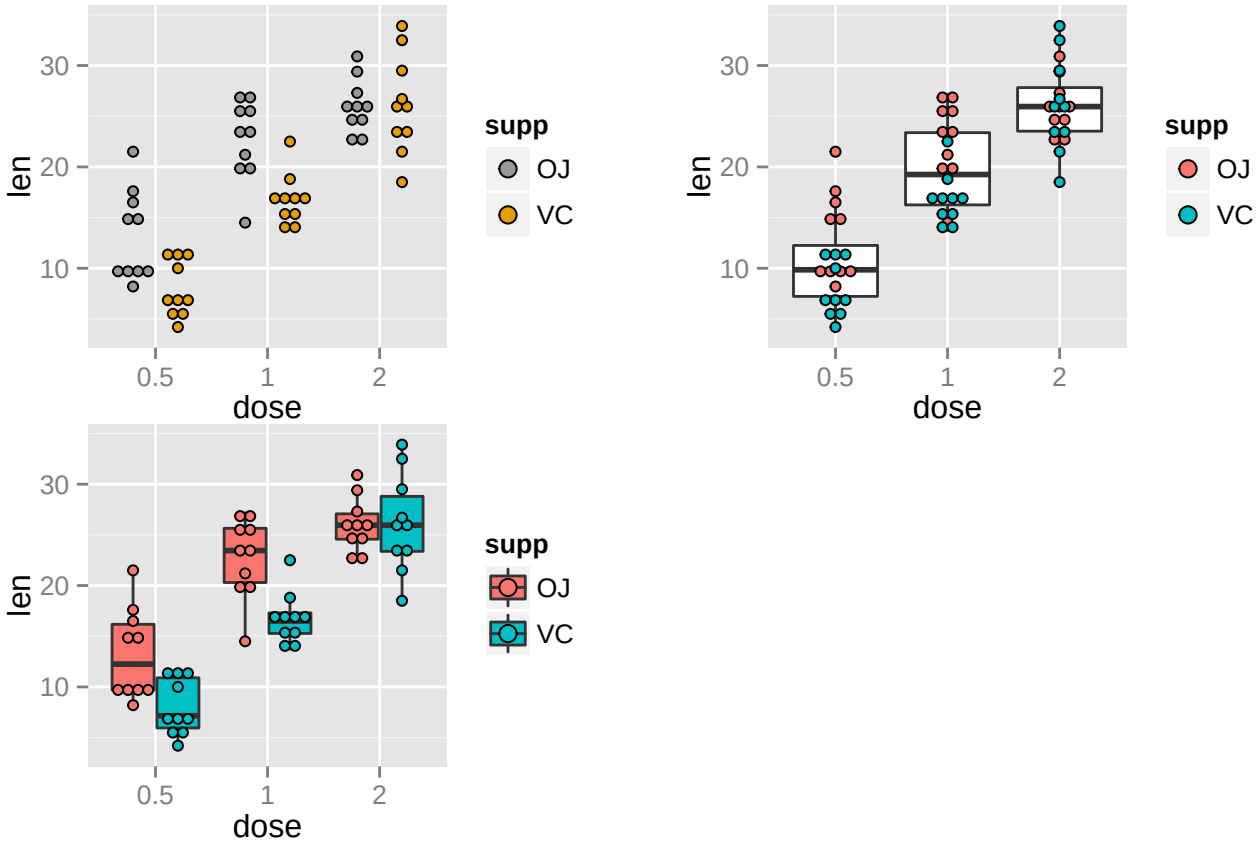
```
# Change dot plot colors by groups
ggplot(ToothGrowth, aes(x=dose, y=len, fill=supp)) +
  geom_dotplot(binaxis='y', stackdir='center')

# Change the position : interval between dot plot of the same group
p<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=supp)) +
  geom_dotplot(binaxis='y', stackdir='center',
               position=position_dodge(0.8))
p
```



Change dot plot colors and add box plots :

```
# Change colors
p+scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Add box plots
```

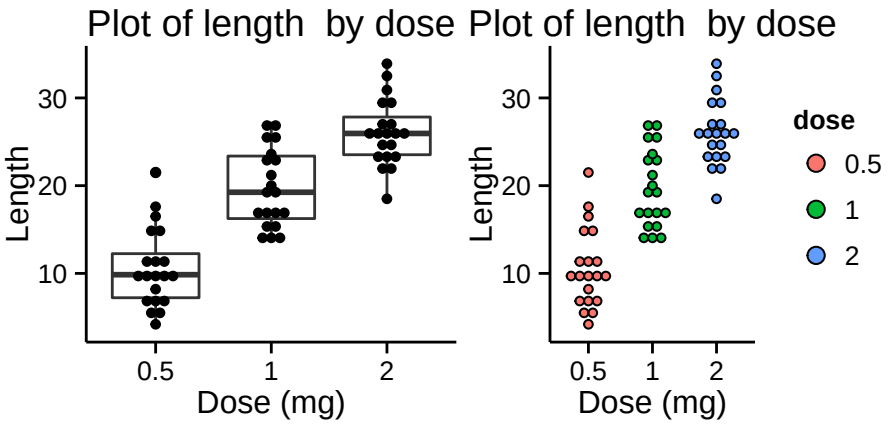
```
ggplot(ToothGrowth, aes(x=dose, y=len, fill=supp)) +
  geom_boxplot(fill="white")+
  geom_dotplot(binaxis='y', stackdir='center')

# Change the position
ggplot(ToothGrowth, aes(x=dose, y=len, fill=supp)) +
  geom_boxplot(position=position_dodge(0.8))+
  geom_dotplot(binaxis='y', stackdir='center',
               position=position_dodge(0.8))
```



4.8 Customized dot plots

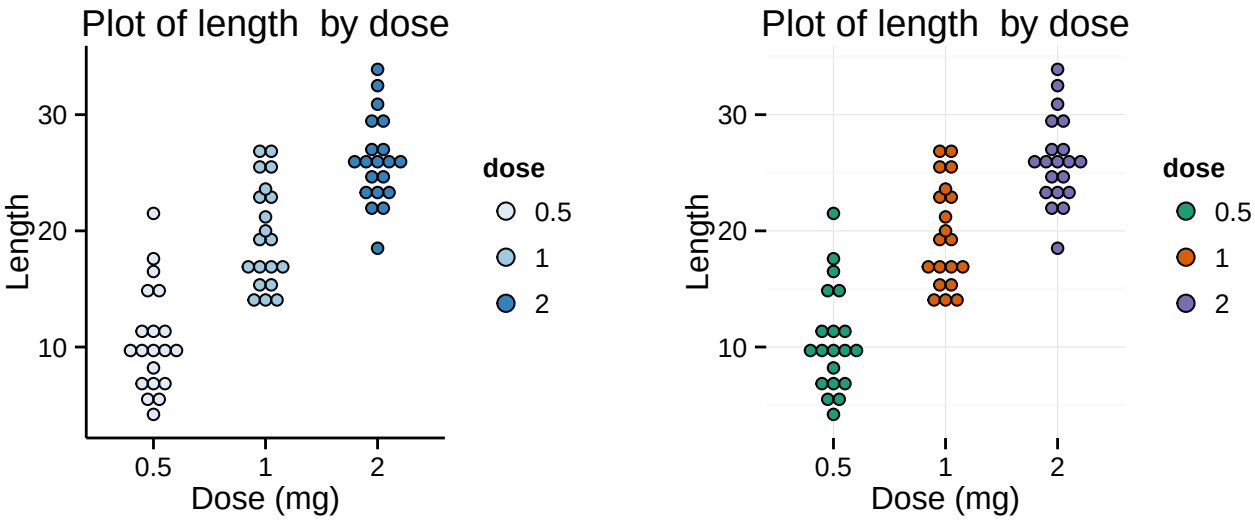
```
# Basic dot plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_boxplot()+
  geom_dotplot(binaxis='y', stackdir='center')+
  labs(title="Plot of length  by dose",x="Dose (mg)", y = "Length")+
  theme_classic()
```

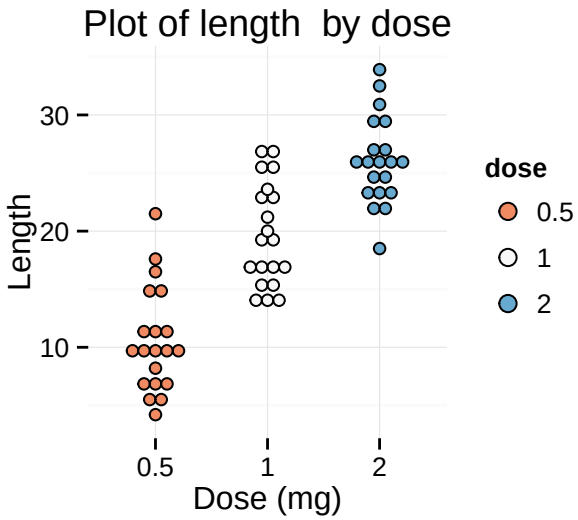
```
# Change color by groups
dp <-ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_dotplot(binaxis='y', stackdir='center')+
  labs(title="Plot of length  by dose",x="Dose (mg)", y = "Length")

dp + theme_classic()
```



Change fill colors manually :

```
# Continuous colors
dp + scale_fill_brewer(palette="Blues") + theme_classic()

# Discrete colors
dp + scale_fill_brewer(palette="Dark2") + theme_minimal()

# Gradient colors
dp + scale_fill_brewer(palette="RdBu") + theme_minimal()
```





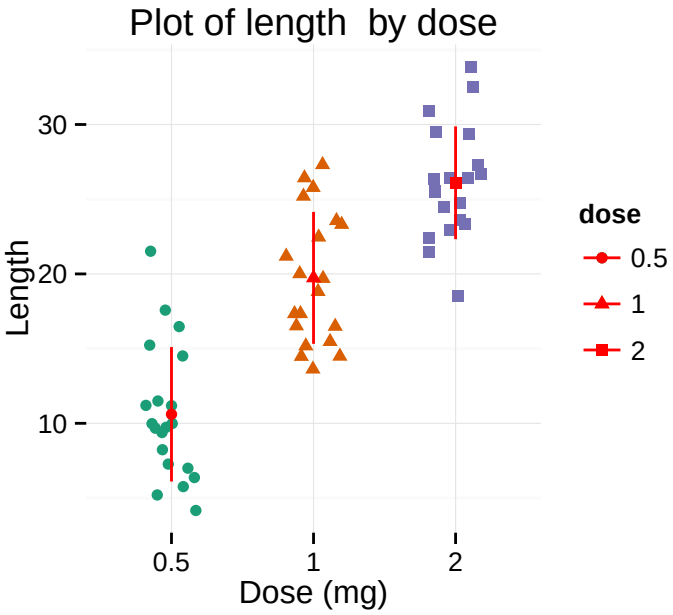
Read more on ggplot2 colors here : Chapter 18

Chapter 5

Stripcharts

Stripcharts are also known as one dimensional scatter plots. These plots are suitable compared to box plots when sample sizes are small.

The function **geom_jitter()** is used.



Key functions: **geom_jitter()**.

5.1 Prepare the data

ToothGrowth data is used :

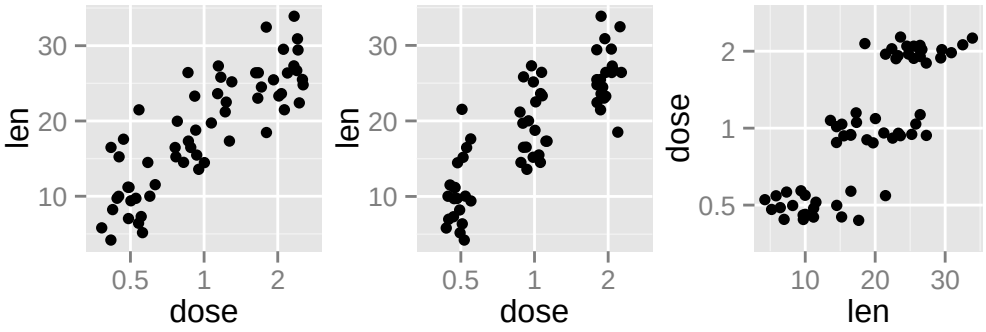
```
# Convert the variable dose from a numeric to a factor variable
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that the variable "dose" is converted as a factor variable using the above R script.

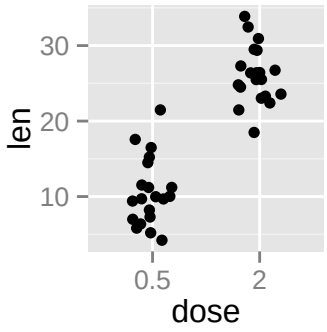
5.2 Basic stripcharts

```
library(ggplot2)
# Basic stripchart
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_jitter()

# Change the position
# 0.2 : degree of jitter in x direction
p<-ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_jitter(position=position_jitter(0.2))
p

# Rotate the stripchart
p + coord_flip()
```



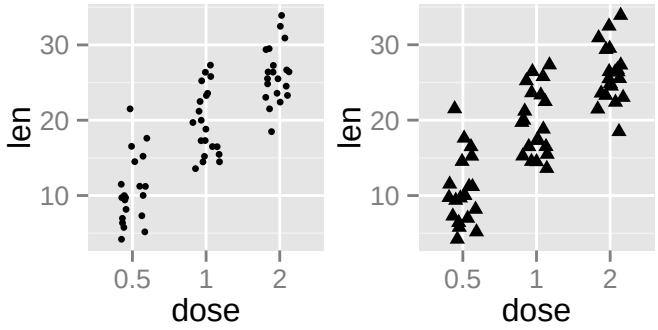
Choose which items to display :

```
p + scale_x_discrete(limits=c("0.5", "2"))
```



Change point shapes and size :

```
# Change point size
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_jitter(position=position_jitter(0.2), cex=1.2)

# Change shape
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_jitter(position=position_jitter(0.2), shape=17)
```



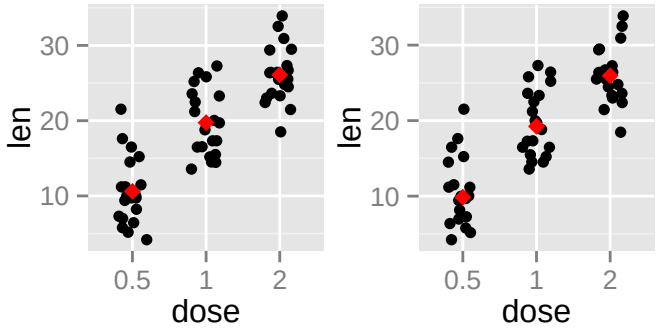
Read more on point shapes: Chapter 19

5.3 Add summary statistics on a stripchart

The function **stat_summary()** can be used to add mean/median points and more to a stripchart.

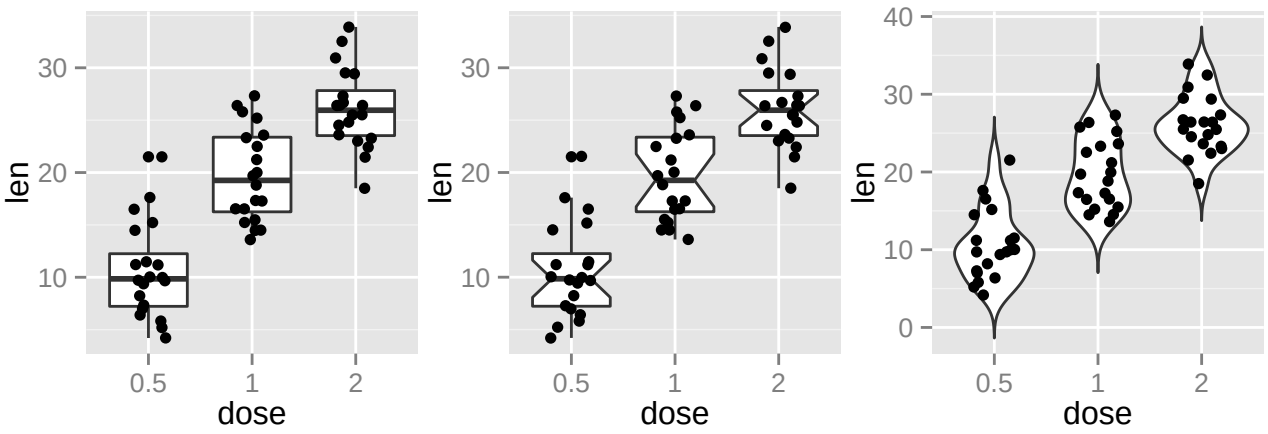
5.3.1 Add mean and median points

```
# Stripchart with mean points
p + stat_summary(fun.y=mean, geom="point", shape=18,
                 size=3, color="red")

# Stripchart with median points
p + stat_summary(fun.y=median, geom="point", shape=18,
                 size=3, color="red")
```



5.3.2 Stripchart with box blot and violin plot

```
# Add basic box plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_boxplot()+
  geom_jitter(position=position_jitter(0.2))

# Add notched box plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_boxplot(notch = TRUE)+
  geom_jitter(position=position_jitter(0.2))

# Add violin plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_violin(trim = FALSE)+
  geom_jitter(position=position_jitter(0.2))
```



Read more on box plot : Chapter 2

Read more on violin plot : Chapter 3

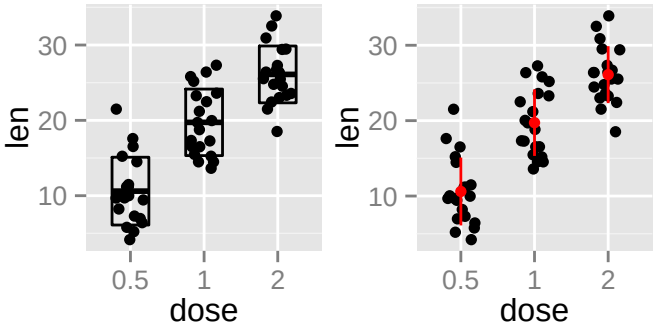
5.3.3 Add mean and standard deviation

The function **mean_sdl** is used. *mean_sdl* computes the *mean* plus or minus a *constant* times the *standard deviation*.

In the R code below, the constant is specified using the argument *mult* (mult = 1). By default mult = 2.

The mean +/- SD can be added as a *crossbar* or a *pointrange* :

```
p <- ggplot(ToothGrowth, aes(x=dose, y=len)) +
    geom_jitter(position=position_jitter(0.2))

p + stat_summary(fun.data="mean_sdl", mult=1,
                 geom="crossbar", width=0.5)

p + stat_summary(fun.data=mean_sdl, mult=1,
                 geom="pointrange", color="red")
```

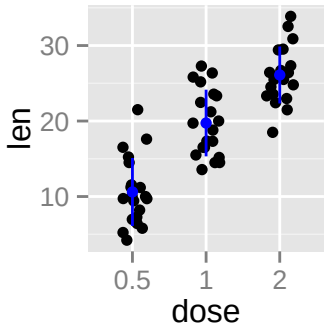


Note that, you can also define a custom function to produce summary statistics as follow

```
# Function to produce summary statistics (mean and +/- sd)
data_summary <- function(x) {
   m <- mean(x)
   ymin <- m-sd(x)
   ymax <- m+sd(x)
   return(c(y=m,ymin=ymin,ymax=ymax))
}
```

Use a custom summary function :

```
p + stat_summary(fun.data=data_summary, color="blue")
```

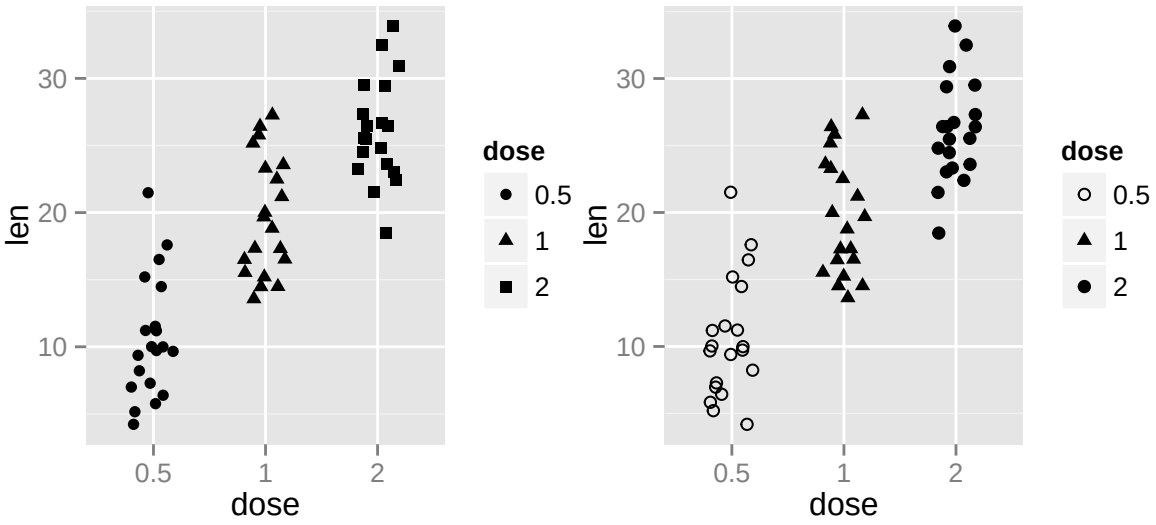


5.4 Change point shapes by groups

In the R code below, point shapes are controlled automatically by the variable *dose*.

You can also set point shapes manually using the function **scale_shape_manual()**

```
# Change point shapes by groups
p<-ggplot(ToothGrowth, aes(x=dose, y=len, shape=dose)) +
    geom_jitter(position=position_jitter(0.2))
p

# Change point shapes manually
p + scale_shape_manual(values=c(1,17,19))
```

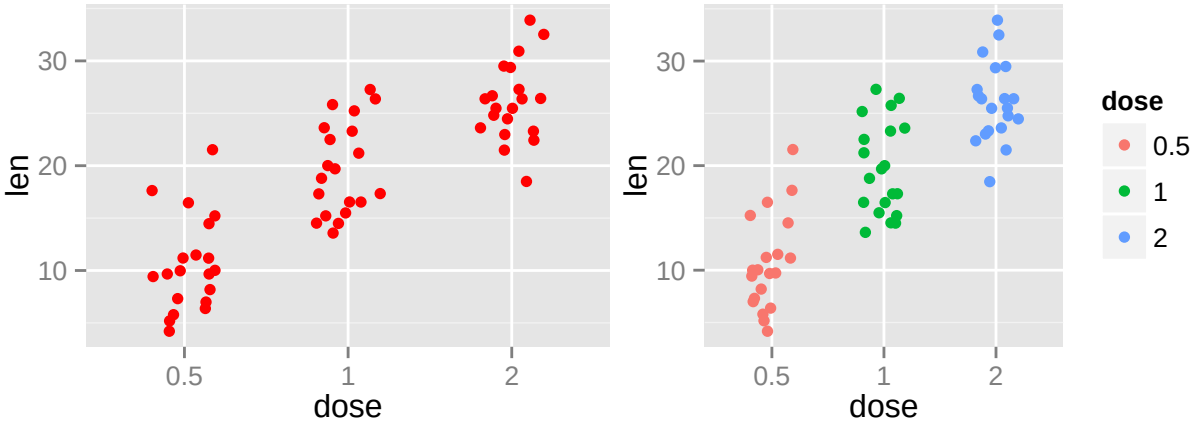


Read more on point shapes : Chapter 19

5.5 Change stripchart colors by groups

In the R code below, point colors of the stripchart are automatically controlled by the levels of *dose* :

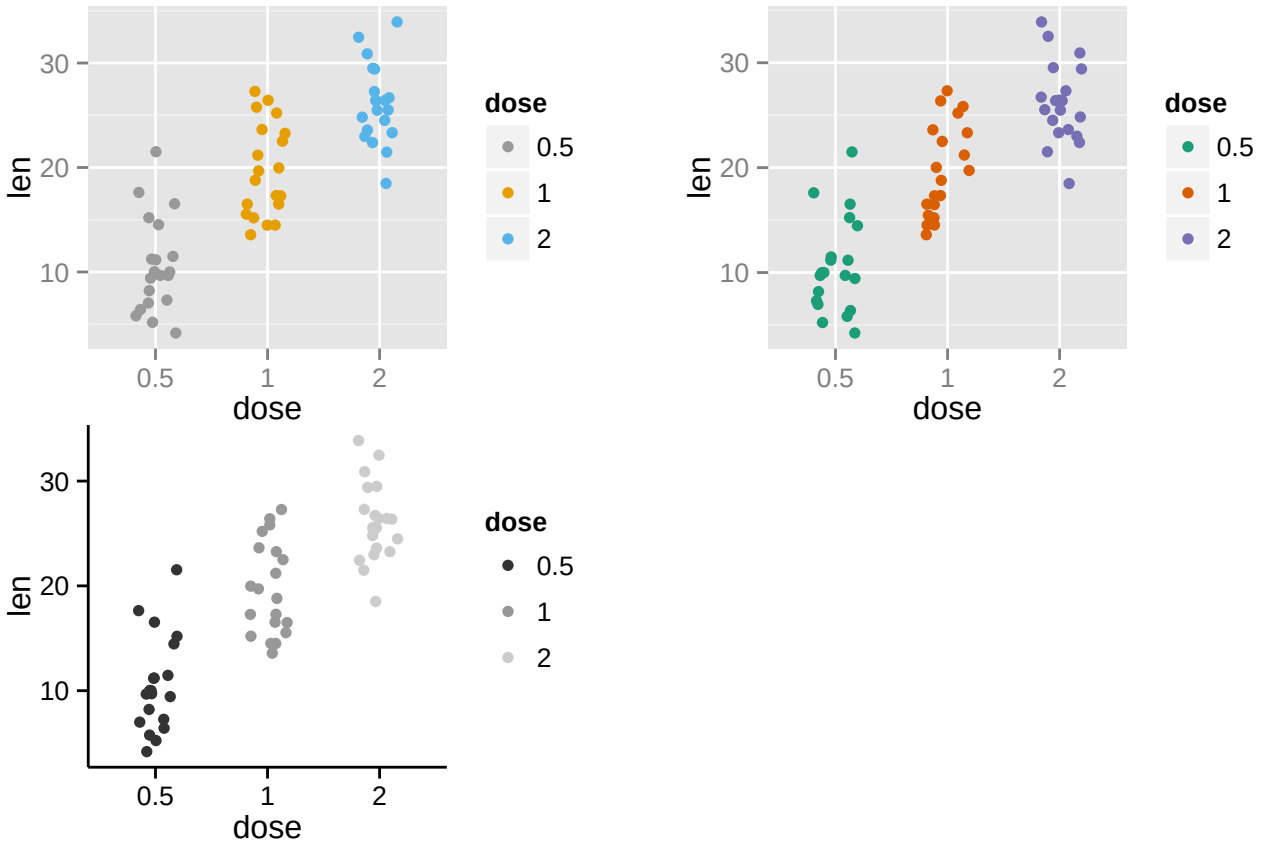
```
# Use single color
ggplot(ToothGrowth, aes(x=dose, y=len)) +
    geom_jitter(position=position_jitter(0.2), color="red")

# Change stripchart colors by groups
p<-ggplot(ToothGrowth, aes(x=dose, y=len, color=dose)) +
  geom_jitter(position=position_jitter(0.2))
p
```



It is also possible to *change manually stripchart colors* using the functions :

- *scale_color_manual()* : to use custom colors
- *scale_color_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_color_grey()* : to use grey color palettes

```
# Use custom color palettes
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
p+scale_color_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey() + theme_classic()
```

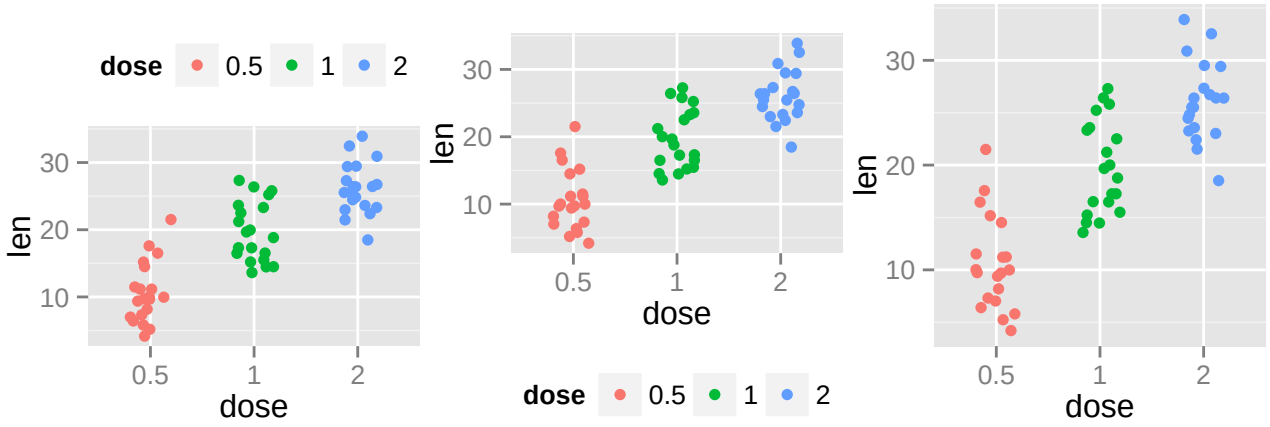


Read more on ggplot2 colors here: Chapter 18

5.6 Change the legend position

```
p + theme(legend.position="top")
```

```
p + theme(legend.position="bottom")

p + theme(legend.position="none")# Remove legend
```



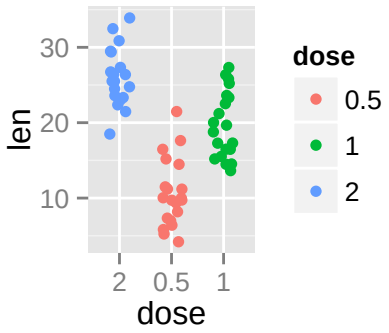
The allowed values for the arguments **legend.position** are : "left","top", "right", "bottom".

Read more on ggplot legends: Chapter 17

5.7 Change the order of items in the legend

The function **scale_x_discrete** can be used to change the order of items to "2", "0.5", "1" :

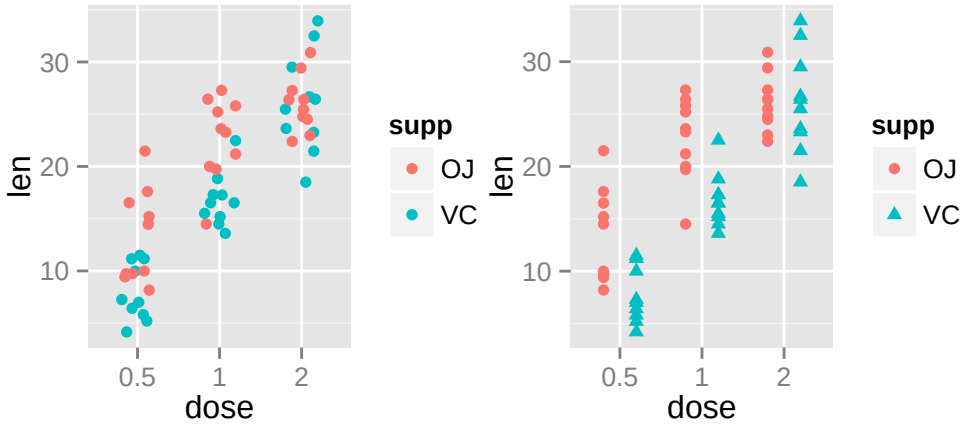
```
p + scale_x_discrete(limits=c("2", "0.5", "1"))
```



5.8 Stripchart with multiple groups

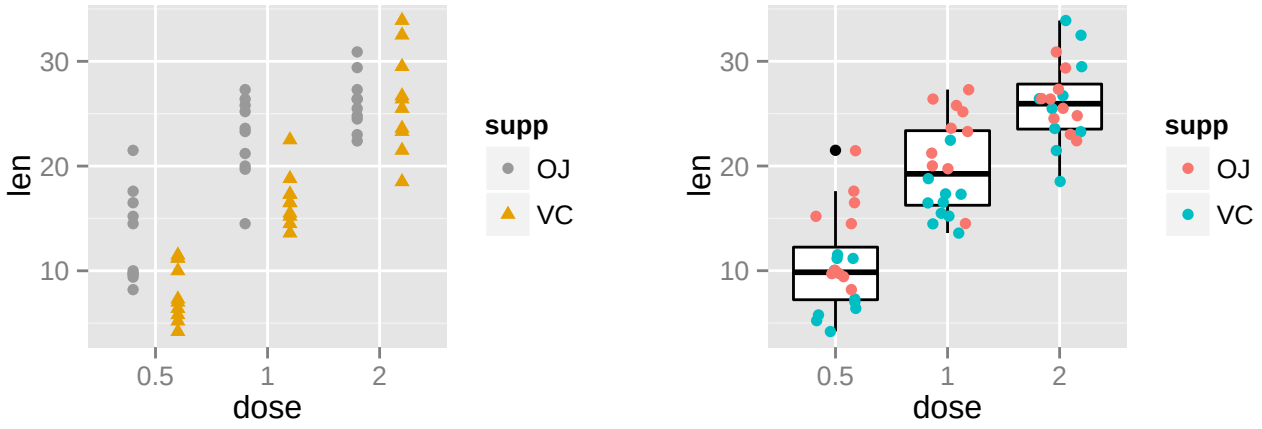
```
# Change stripchart colors by groups
ggplot(ToothGrowth, aes(x=dose, y=len, color=supp)) +
  geom_jitter(position=position_jitter(0.2))
```

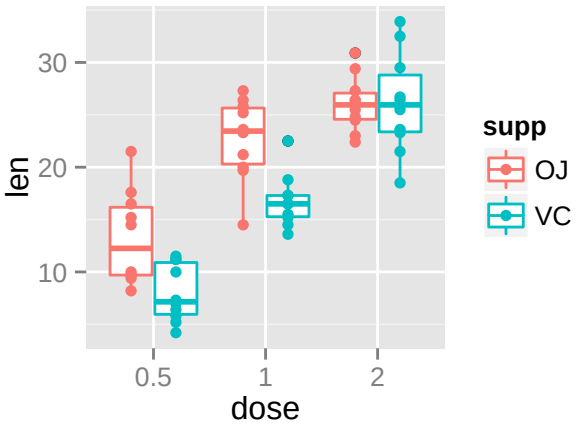
```
# Change the position : interval between stripchart of the same group
p<-ggplot(ToothGrowth, aes(x=dose, y=len, color=supp, shape=supp)) +
  geom_jitter(position=position_dodge(0.8))
p
```



Change stripchart colors and add box plots :

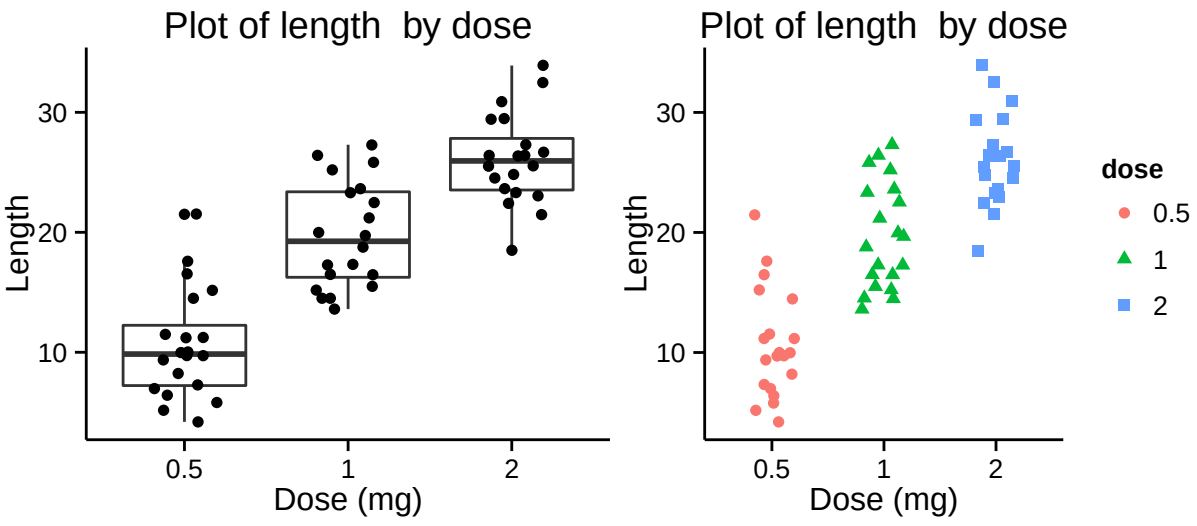
```
# Change colors
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Add box plots
ggplot(ToothGrowth, aes(x=dose, y=len, color=supp)) +
  geom_boxplot(color="black")+
  geom_jitter(position=position_jitter(0.2))

# Change the position
ggplot(ToothGrowth, aes(x=dose, y=len, color=supp)) +
  geom_boxplot(position=position_dodge(0.8))+
  geom_jitter(position=position_dodge(0.8))
```





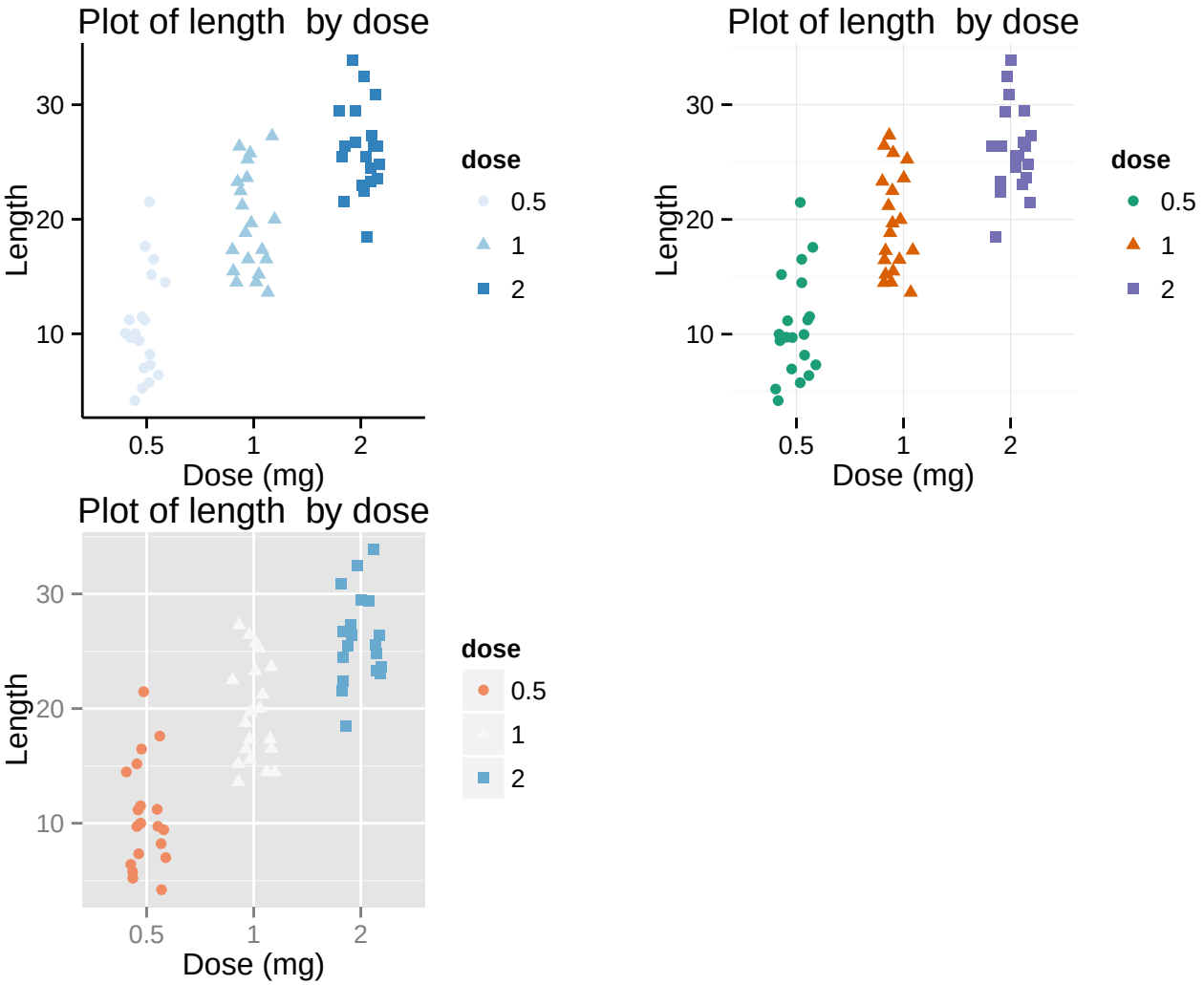
5.9 Customized stripcharts

```
# Basic stripchart
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_boxplot()+
  geom_jitter(position=position_jitter(0.2))+
  labs(title="Plot of length  by dose",x="Dose (mg)", y = "Length")+
  theme_classic()

# Change color/shape by groups
p <- ggplot(ToothGrowth, aes(x=dose, y=len, color=dose, shape=dose)) +
  geom_jitter(position=position_jitter(0.2))+
  labs(title="Plot of length  by dose",x="Dose (mg)", y = "Length")

p + theme_classic()
```



Change colors manually :

```
# Continuous colors
p + scale_color_brewer(palette="Blues") + theme_classic()

# Discrete colors
p + scale_color_brewer(palette="Dark2") + theme_minimal()

# Gradient colors
p + scale_color_brewer(palette="RdBu")
```

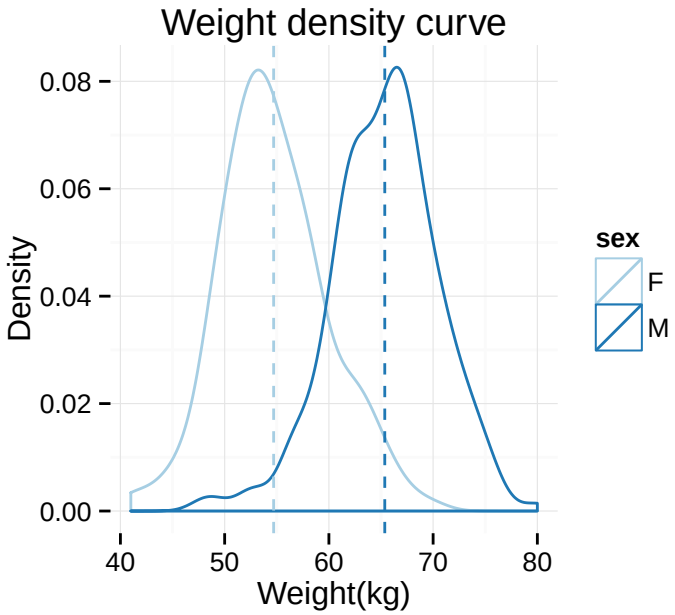


Read more on ggplot2 colors here : Chapter 18

Chapter 6

Density plots

The function **geom_density()** is used to create a **density plot**. You can also add a line for the mean using the function **geom_vline()** [Chapter 22].



Key functions: **geom_density()**, **stat_density()**.

6.1 Prepare the data

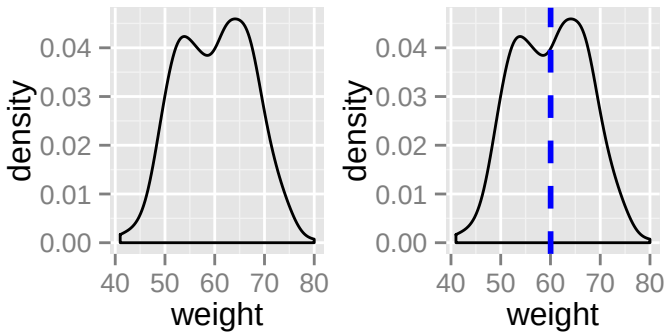
This data will be used for the examples below :

```
set.seed(1234)

df <- data.frame(
  sex=factor(rep(c("F", "M"), each=200)),
  weight=round(c(rnorm(200, mean=55, sd=5),
                 rnorm(200, mean=65, sd=5))))

head(df)
```

```
##   sex weight
## 1   F     49
## 2   F     56
## 3   F     60
## 4   F     43
## 5   F     57
## 6   F     58
```

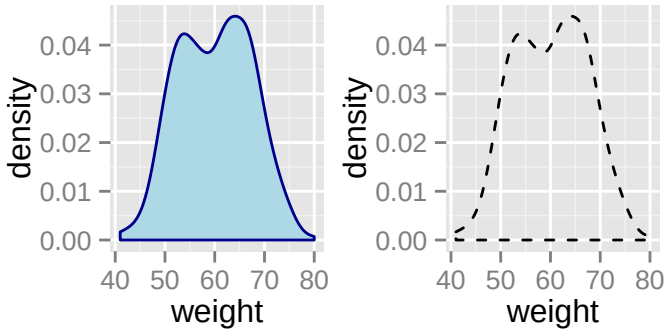
6.2 Basic density plots

```
library(ggplot2)
# Basic density
p <- ggplot(df, aes(x=weight)) +
  geom_density()
p

# Add mean line
p+ geom_vline(aes(xintercept=mean(weight)),
            color="blue", linetype="dashed", size=1)
```



6.3 Change density plot line types and colors

```
# Change line color and fill color
ggplot(df, aes(x=weight))+
  geom_density(color="darkblue", fill="lightblue")

# Change line type
ggplot(df, aes(x=weight))+
  geom_density(linetype="dashed")
```



Read more on ggplot2 line types: Chapter 20

6.4 Change density plot colors by groups

6.4.1 Calculate the mean of each group :

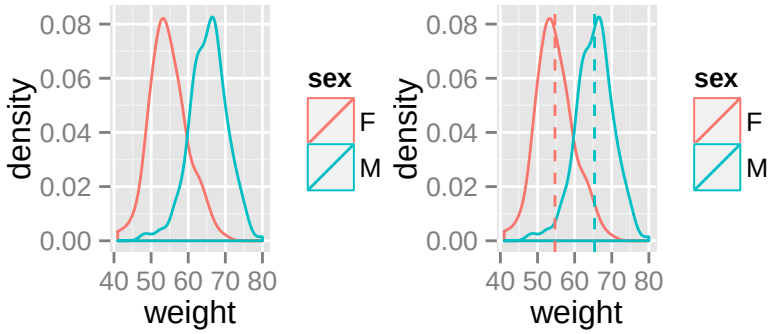
```
library(plyr)
mu <- ddply(df, "sex", summarise, grp.mean=mean(weight))
head(mu)
```

```
##   sex grp.mean
## 1   F    54.70
## 2   M    65.36
```

6.4.2 Change line colors

Density plot line colors can be automatically controlled by the levels of *sex* :

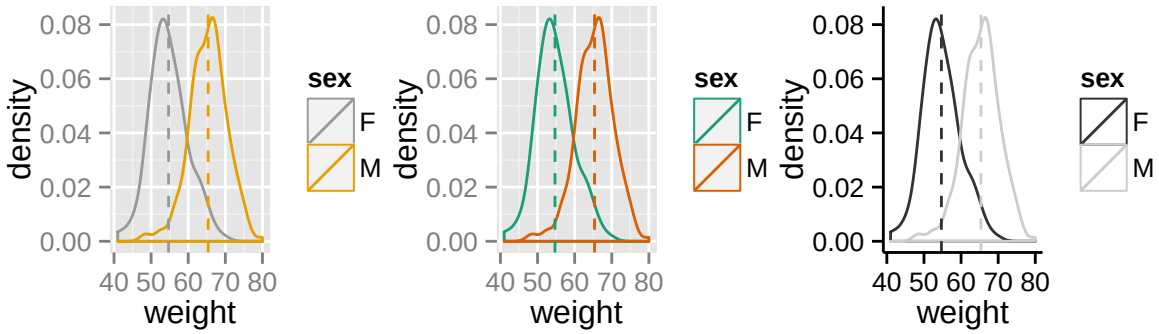
```
# Change density plot line colors by groups
ggplot(df, aes(x=weight, color=sex)) +
  geom_density()

# Add mean lines
p<-ggplot(df, aes(x=weight, color=sex)) +
  geom_density()+
  geom_vline(data=mu, aes(xintercept=grp.mean, color=sex),
             linetype="dashed")
p
```



It is also possible to *change manually density plot line colors* using the functions :

- *scale_color_manual()* : to use custom colors
- *scale_color_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_color_grey()* : to use grey color palettes

```
# Use custom color palettes
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
p+scale_color_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey() + theme_classic()
```

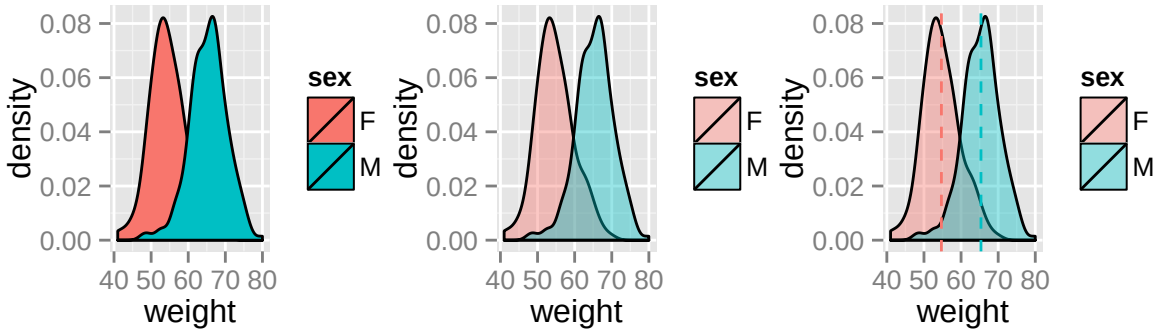


Read more on ggplot2 colors here: Chapter 18

6.4.3 Change fill colors

Density plot fill colors can be automatically controlled by the levels of *sex* :

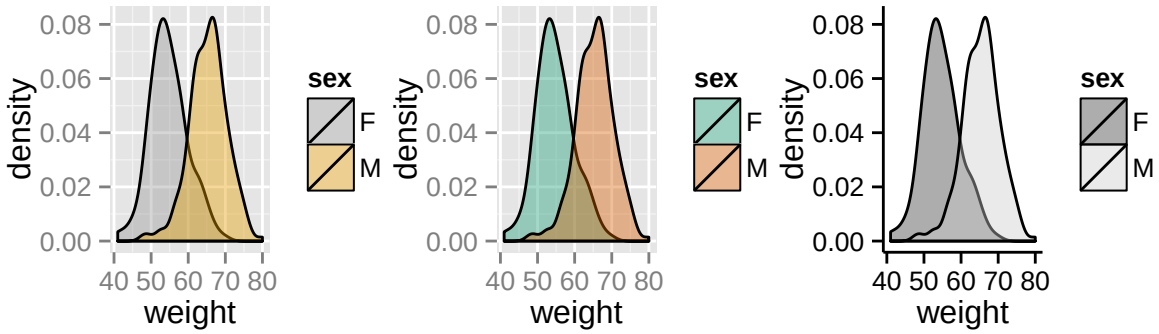
```
# Change density plot fill colors by groups
ggplot(df, aes(x=weight, fill=sex)) +
  geom_density()

# Use semi-transparent fill
p<-ggplot(df, aes(x=weight, fill=sex)) +
  geom_density(alpha=0.4)
p

# Add mean lines
p+geom_vline(data=mu, aes(xintercept=grp.mean, color=sex),
             linetype="dashed")
```



It is also possible to change manually density plot fill colors using the functions :

- *scale_fill_manual()* : to use custom colors
- *scale_fill_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_fill_grey()* : to use grey color palettes

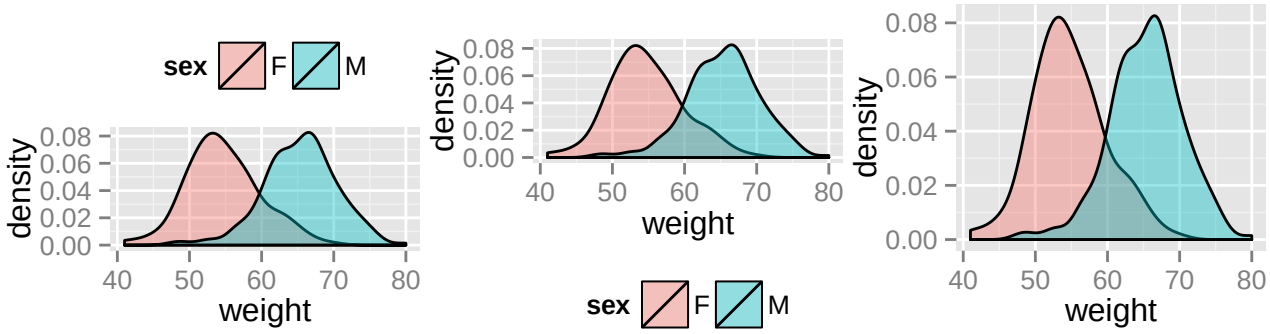
```
# Use custom color palettes
p+scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# use brewer color palettes
p+scale_fill_brewer(palette="Dark2")

# Use grey scale
p + scale_fill_grey() + theme_classic()
```



Read more on ggplot2 colors here: Chapter 18

6.5 Change the legend position

```
p + theme(legend.position="top")

p + theme(legend.position="bottom")

p + theme(legend.position="none") # Remove legend
```



The allowed values for the arguments **legend.position** are : "left","top", "right", "bottom".

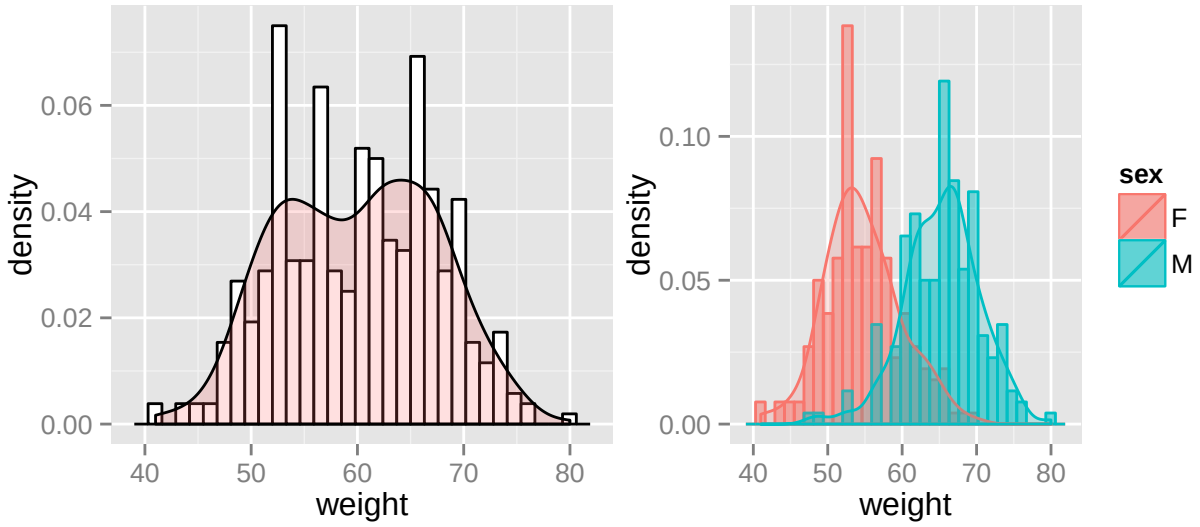
Read more on ggplot legends : Chapter 17

6.6 Combine histogram and density plots

- The histogram is plotted with density instead of count values on y-axis
- Overlay with transparent density plot

```
# Histogram with density plot
ggplot(df, aes(x=weight)) +
 geom_histogram(aes(y=..density..), colour="black", fill="white")+
 geom_density(alpha=.2, fill="#FF6666")
```

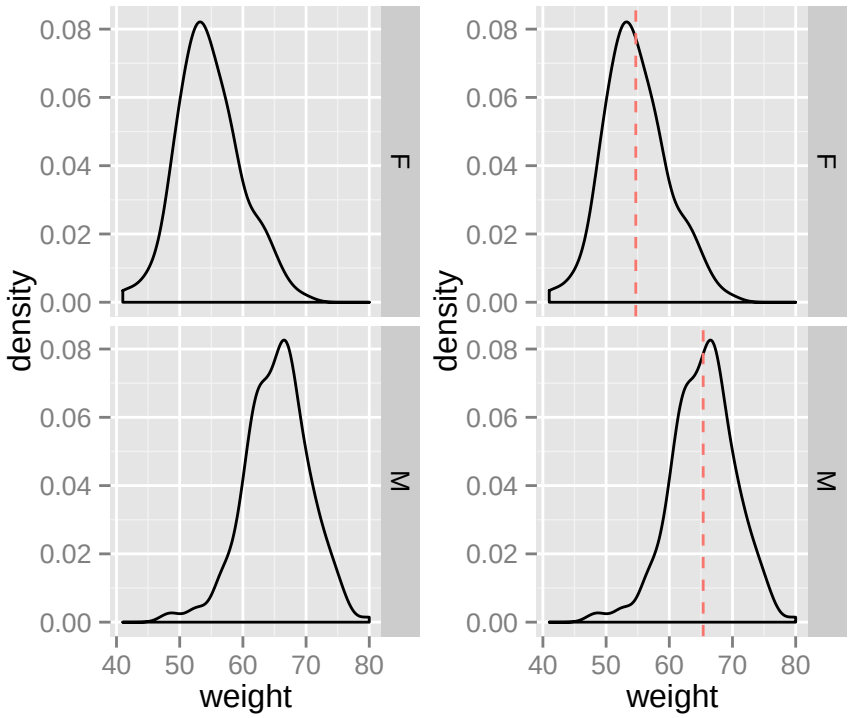
```
# Color by groups
ggplot(df, aes(x=weight, color=sex, fill=sex)) +
 geom_histogram(aes(y=..density..), alpha=0.5,
                position="identity")+
 geom_density(alpha=.2)
```



6.7 Use facets

Split the plot in multiple panels:

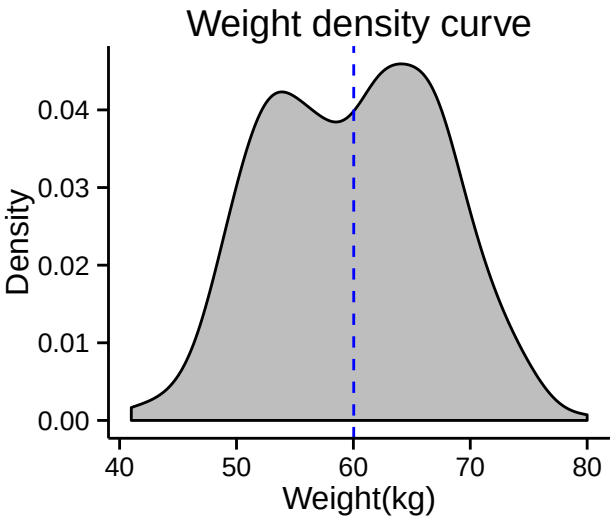
```
p<-ggplot(df, aes(x=weight))+
  geom_density()+facet_grid(sex ~ .)
p

# Add mean lines
p+geom_vline(data=mu, aes(xintercept=grp.mean, color="red"),
             linetype="dashed")
```

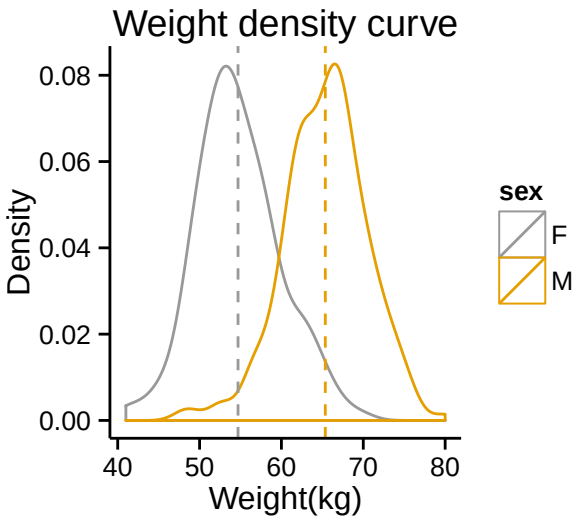


Read more on facets: Chapter 27

6.8 Customized density plots

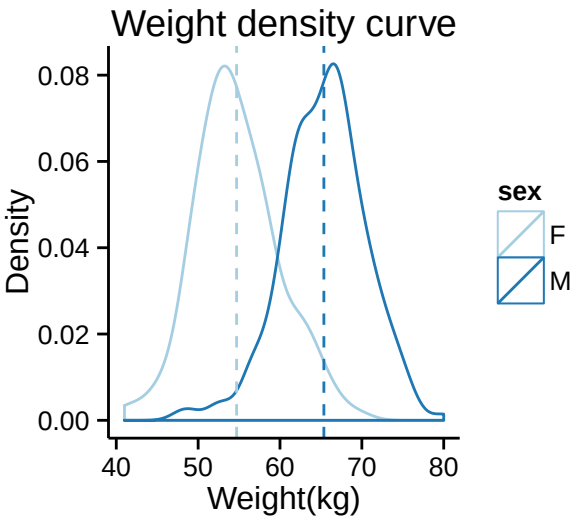
```
# Basic density
ggplot(df, aes(x=weight, fill=sex)) +
  geom_density(fill="gray")+
  geom_vline(aes(xintercept=mean(weight)), color="blue",
             linetype="dashed")+
  labs(title="Weight density curve",x="Weight(kg)", y = "Density")+
  theme_classic()

# Change line colors by groups
p<- ggplot(df, aes(x=weight, color=sex)) +
  geom_density()+
  geom_vline(data=mu, aes(xintercept=grp.mean, color=sex),
             linetype="dashed")+
  labs(title="Weight density curve",x="Weight(kg)", y = "Density")

p + scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))+
  theme_classic()
```

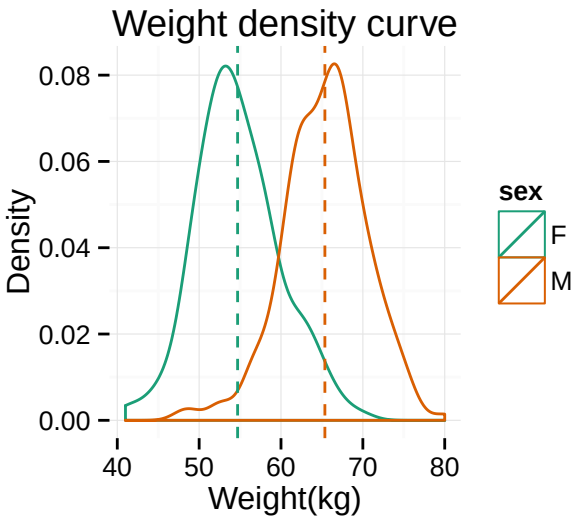


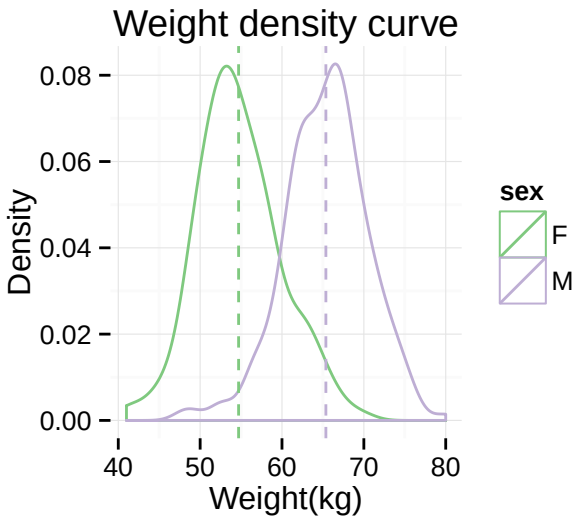


Change line colors manually :

```
# Continuous colors
p + scale_color_brewer(palette="Paired") + theme_classic()

# Discrete colors
p + scale_color_brewer(palette="Dark2") + theme_minimal()

# Gradient colors
p + scale_color_brewer(palette="Accent") + theme_minimal()
```





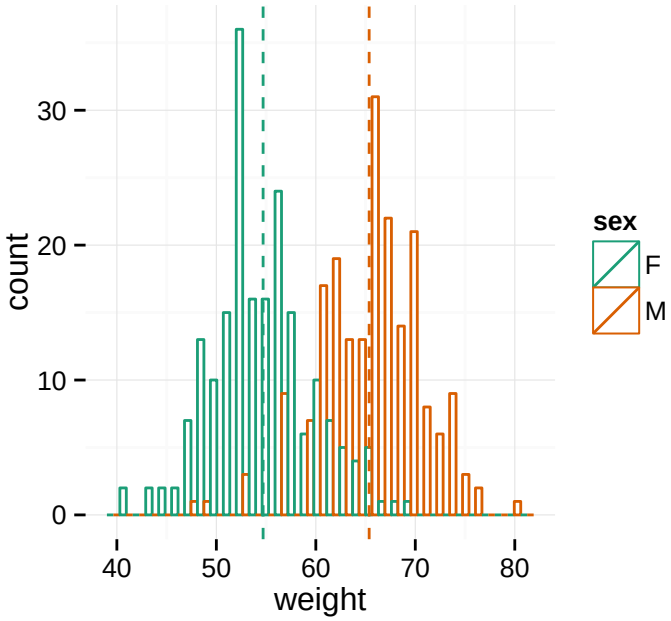


Read more on ggplot2 colors here : Chapter 18

Chapter 7

Histogram plots

The function **geom_histogram()** is used to create a **histogram plot**. You can also add a line for the mean using the function **geom_vline()** [Chapter 22].



Key functions: **geom_histogram()**, **stat_bin()**

7.1 Prepare the data

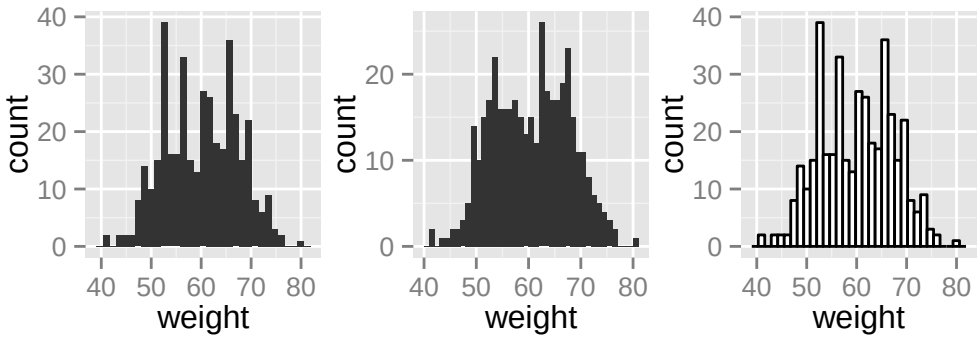
The data below will be used :

```
set.seed(1234)

df <- data.frame(
  sex=factor(rep(c("F", "M"), each=200)),
  weight=round(c(rnorm(200, mean=55, sd=5), rnorm(200, mean=65, sd=5)))
  )

head(df)
```

```
##   sex weight
## 1   F     49
## 2   F     56
## 3   F     60
## 4   F     43
## 5   F     57
## 6   F     58
```

7.2 Basic histogram plots

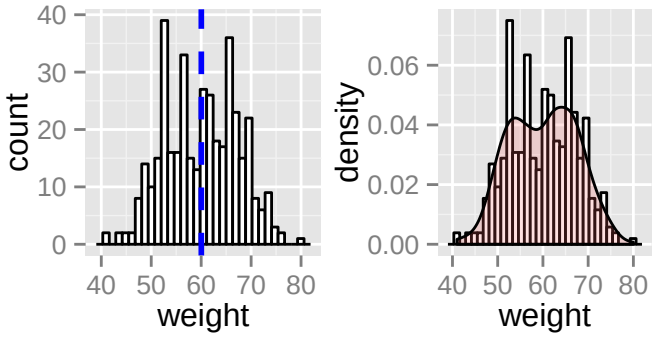
```
library(ggplot2)
# Basic histogram
ggplot(df, aes(x=weight)) + geom_histogram()

# Change the width of bins
ggplot(df, aes(x=weight)) +
  geom_histogram(binwidth=1)

# Change colors
p<-ggplot(df, aes(x=weight)) +
  geom_histogram(color="black", fill="white")
p
```



7.3 Add mean line and density plot on the histogram

- The histogram is plotted with density instead of count on y-axis
- Overlay with transparent density plot. The value of *alpha* controls the level of transparency

```
# Add mean line
p+ geom_vline(aes(xintercept=mean(weight)),
              color="blue", linetype="dashed", size=1)

# Histogram with density plot
ggplot(df, aes(x=weight)) +
 geom_histogram(aes(y=..density..), colour="black", fill="white")+
 geom_density(alpha=.2, fill="#FF6666")
```

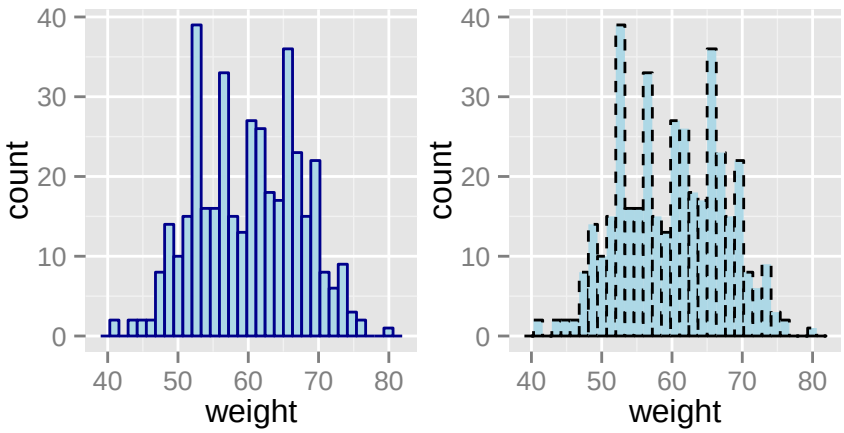


Read more on ggplot2 line types: Chapter 20

7.4 Change histogram plot line types and colors

```
# Change line color and fill color
ggplot(df, aes(x=weight))+
```

```
  geom_histogram(color="darkblue", fill="lightblue")

# Change line type
ggplot(df, aes(x=weight))+
  geom_histogram(color="black", fill="lightblue",
                 linetype="dashed")
```



7.5 Change histogram plot colors by groups

7.5.1 Calculate the mean of each group :

The package **plyr** is used to calculate the average weight of each group :

```
library(plyr)
mu <- ddply(df, "sex", summarise, grp.mean=mean(weight))
head(mu)
```

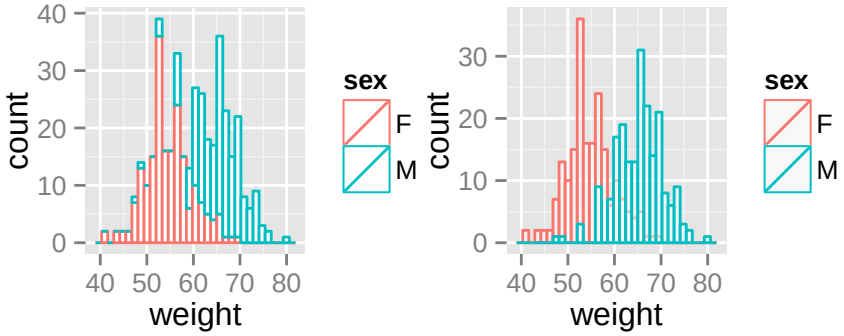
```
##   sex grp.mean
## 1   F    54.70
## 2   M    65.36
```

7.5.2 Change line colors

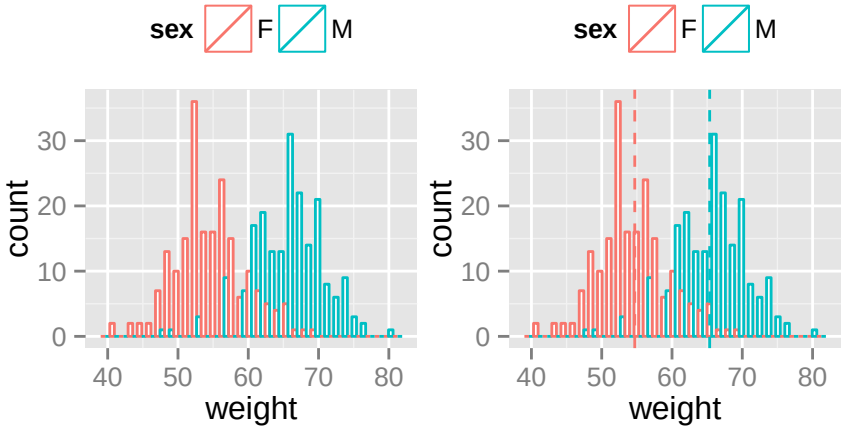
Histogram plot line colors can be automatically controlled by the levels of the variable *sex*.

Note that, you can change the position adjustment to use for overlapping points on the layer. Possible values for the argument **position** are "identity", "stack", "dodge". Default value is "stack".

```
# Change histogram plot line colors by groups
ggplot(df, aes(x=weight, color=sex)) +
  geom_histogram(fill="white")

# Overlaid histograms
ggplot(df, aes(x=weight, color=sex)) +
  geom_histogram(fill="white", alpha=0.5, position="identity")
```



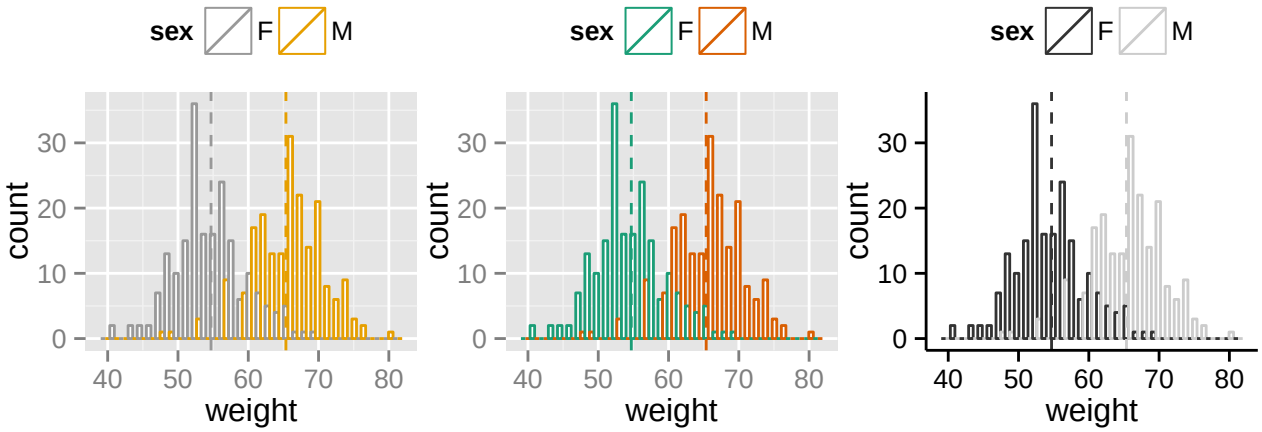

```
# Interleaved histograms
ggplot(df, aes(x=weight, color=sex)) +
  geom_histogram(fill="white", position="dodge")+
  theme(legend.position="top")

# Add mean lines
p<-ggplot(df, aes(x=weight, color=sex)) +
  geom_histogram(fill="white", position="dodge")+
  geom_vline(data=mu, aes(xintercept=grp.mean, color=sex),
             linetype="dashed")+
  theme(legend.position="top")
p
```



It is also possible to *change manually histogram plot line colors* using the functions :

- *scale_color_manual()* : to use custom colors
- *scale_color_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_color_grey()* : to use grey color palettes

```
# Use custom color palettes
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
p+scale_color_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey() + theme_classic() +
  theme(legend.position="top")
```

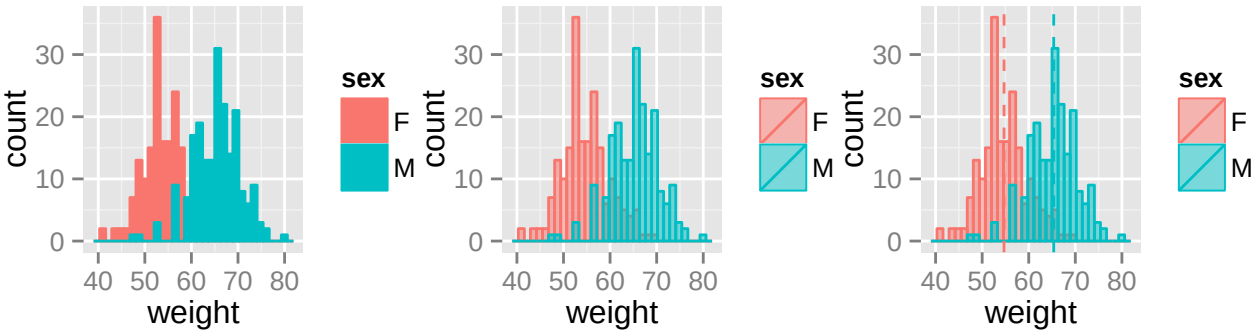


Read more on ggplot2 colors here : Chapter 18

7.5.3 Change fill colors

Histogram plot fill colors can be automatically controlled by the levels of *sex* :

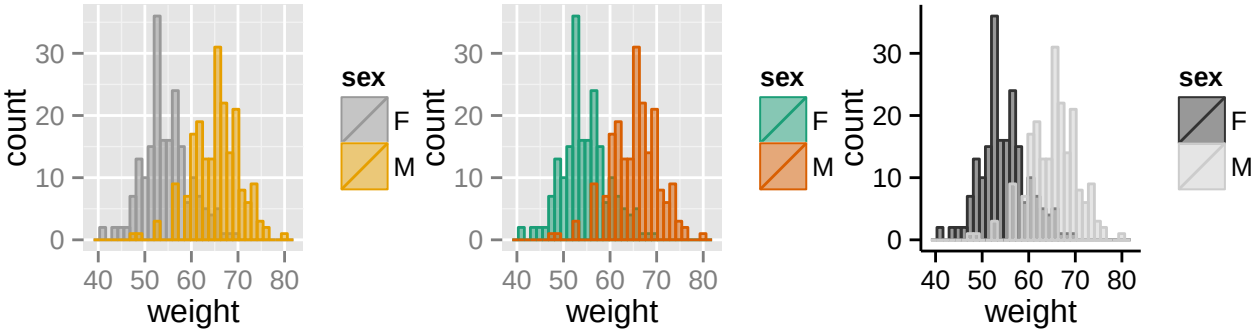
```
# Change histogram plot fill colors by groups
ggplot(df, aes(x=weight, fill=sex, color=sex)) +
  geom_histogram(position="identity")

# Use semi-transparent fill
p<-ggplot(df, aes(x=weight, fill=sex, color=sex)) +
  geom_histogram(position="identity", alpha=0.5)
p

# Add mean lines
p+geom_vline(data=mu, aes(xintercept=grp.mean, color=sex),
             linetype="dashed")
```



It is also possible to change manually histogram plot fill colors using the functions :

- *scale_fill_manual()* : to use custom colors
- *scale_fill_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_fill_grey()* : to use grey color palettes

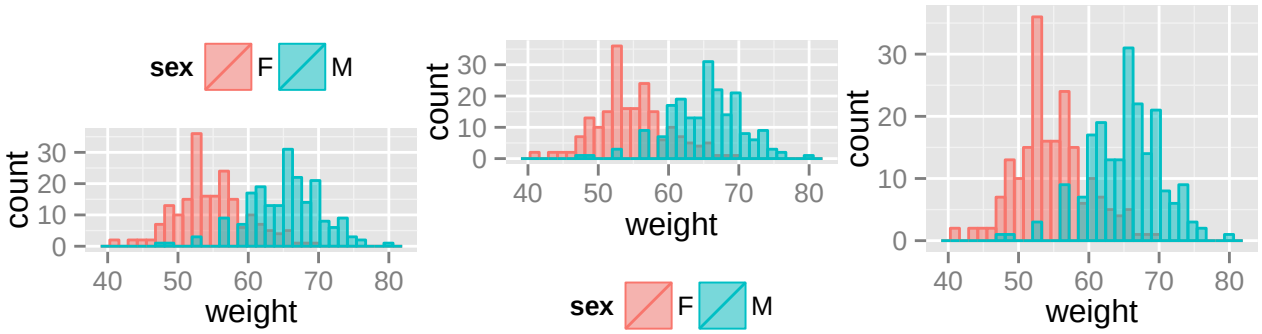
```
# Use custom color palettes
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))+
  scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# use brewer color palettes
p+scale_color_brewer(palette="Dark2")+
  scale_fill_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey()+scale_fill_grey() +
  theme_classic()
```



Read more on ggplot2 colors here: Chapter 18

7.6 Change the legend position

```
p + theme(legend.position="top")

p + theme(legend.position="bottom")

# Remove legend
p + theme(legend.position="none")
```



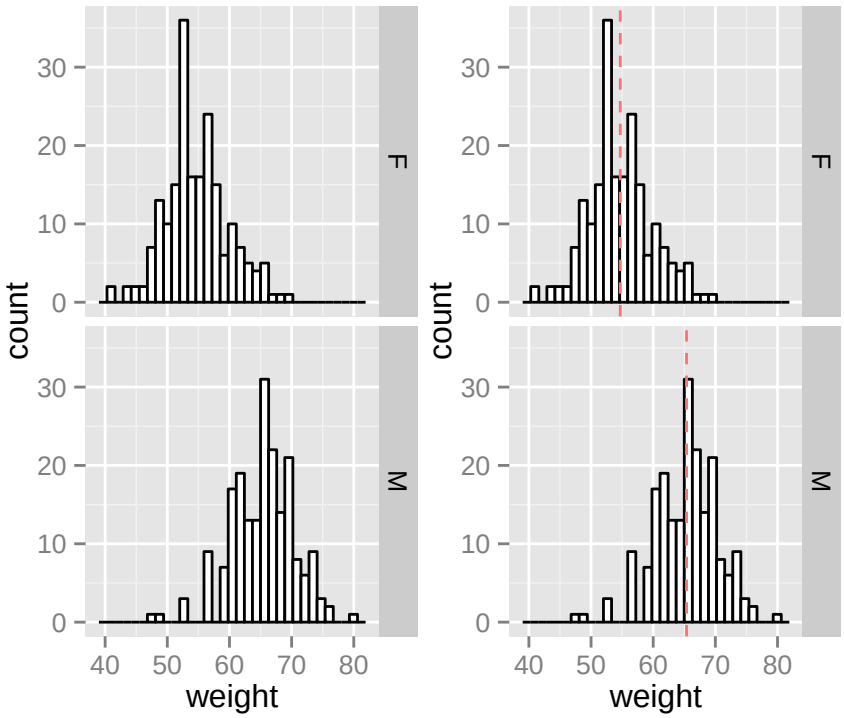
The allowed values for the arguments **legend.position** are : "left","top", "right", "bottom".

Read more on ggplot legends: Chapter 17

7.7 Use facets

Split the plot into multiple panels :

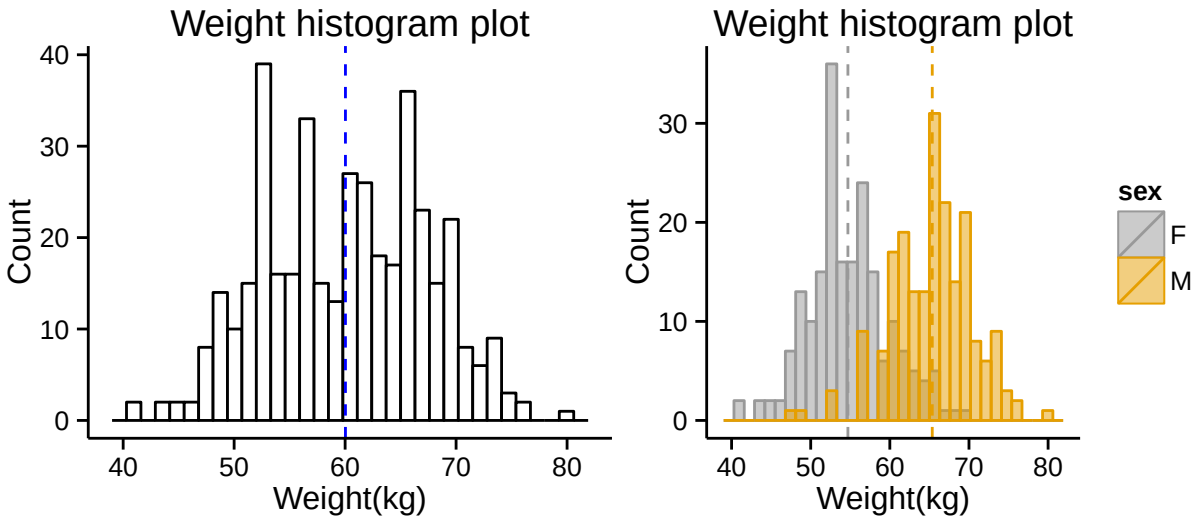
```
p<-ggplot(df, aes(x=weight))+
  geom_histogram(color="black", fill="white")+
  facet_grid(sex ~ .)
p

# Add mean lines
p+geom_vline(data=mu, aes(xintercept=grp.mean, color="red"),
             linetype="dashed")
```



Read more on facets: Chapter **??**

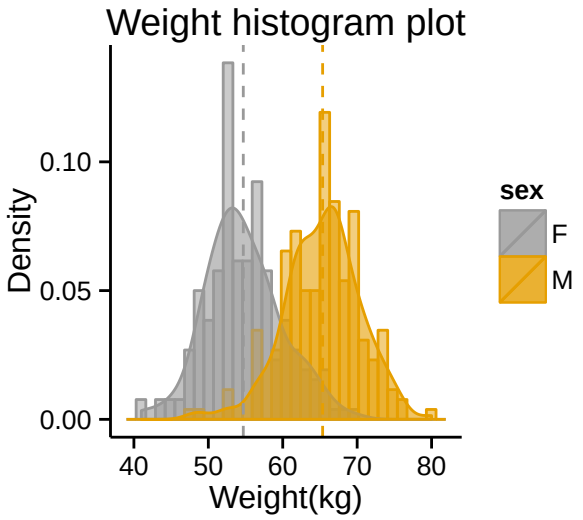
7.8 Customized histogram plots

```
# Basic histogram
ggplot(df, aes(x=weight, fill=sex)) +
  geom_histogram(fill="white", color="black")+
  geom_vline(aes(xintercept=mean(weight)), color="blue",
             linetype="dashed")+
  labs(title="Weight histogram plot",x="Weight(kg)", y = "Count")+
  theme_classic()

# Change line colors by groups
ggplot(df, aes(x=weight, color=sex, fill=sex)) +
  geom_histogram(position="identity", alpha=0.5)+
  geom_vline(data=mu, aes(xintercept=grp.mean, color=sex),
             linetype="dashed")+
  scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))+
  scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))+
  labs(title="Weight histogram plot",x="Weight(kg)", y = "Count")+
  theme_classic()
```



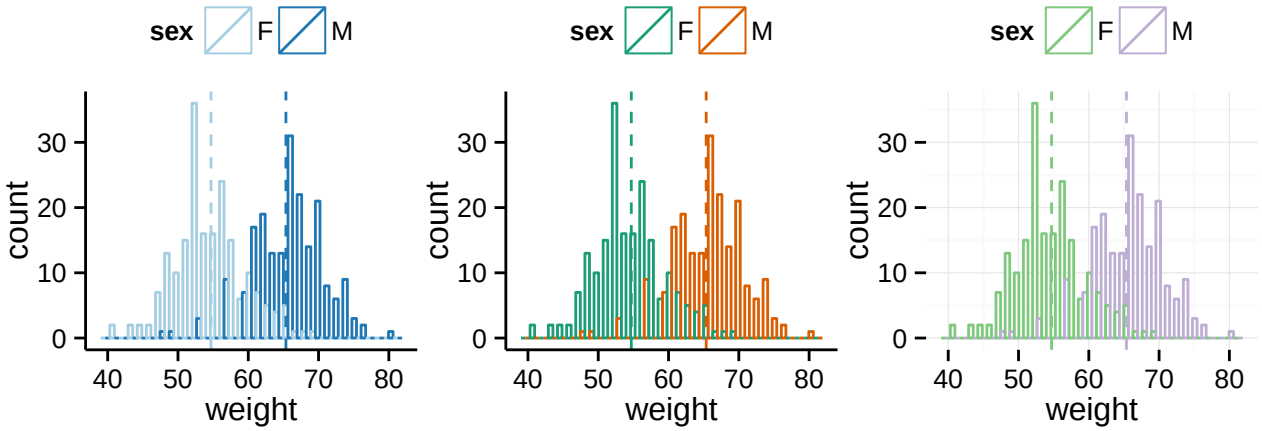
Combine histogram and density plots :

```
# Change line colors by groups
ggplot(df, aes(x=weight, color=sex, fill=sex)) +
geom_histogram(aes(y=..density..), position="identity", alpha=0.5)+
geom_density(alpha=0.6)+
geom_vline(data=mu, aes(xintercept=grp.mean, color=sex),
           linetype="dashed")+
scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))+
scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))+
labs(title="Weight histogram plot",x="Weight(kg)", y = "Density")+
theme_classic()
```



Change line colors manually :

```
p<-ggplot(df, aes(x=weight, color=sex)) +
  geom_histogram(fill="white", position="dodge")+
  geom_vline(data=mu, aes(xintercept=grp.mean, color=sex),
             linetype="dashed")

# Continuous colors
p + scale_color_brewer(palette="Paired") +
  theme_classic()+theme(legend.position="top")

# Discrete colors
p + scale_color_brewer(palette="Dark2") +
  theme_minimal()+theme_classic()+theme(legend.position="top")

# Gradient colors
p + scale_color_brewer(palette="Accent") +
  theme_minimal()+theme(legend.position="top")
```

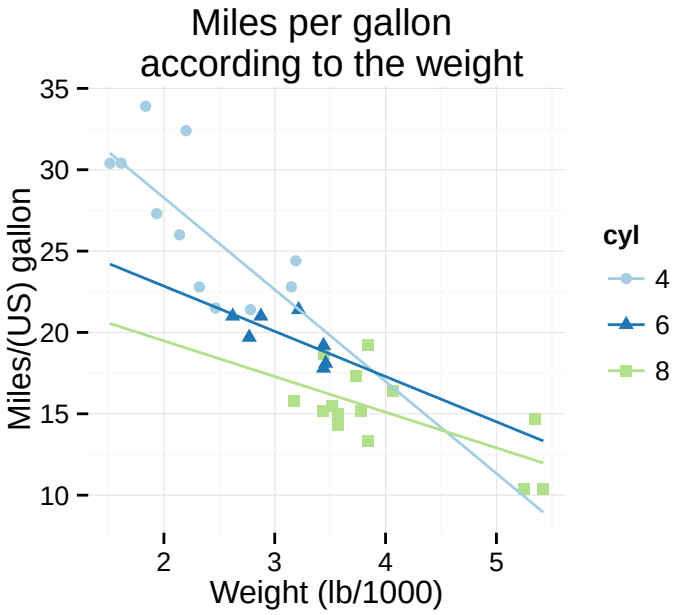


Read more on ggplot2 colors here : Chapter 18

Chapter 8

Scatter plots

The function **geom_point()** is used to create a **scatter plot** using **ggplot2** package.



Key functions:

- geom_point()
- geom_smooth(), stat_smooth()
- geom_rug()
- geom_density2d(), stat_density2d()
- stat_bin2d(), geom_bin2d(), stat_bin2d(), stat_summary2d()
- geom_hex() (see stat_binhex), stat_summary_hex()

8.1 Prepare the data

mtcars data is used in the examples below.

```
# Convert cyl column from a numeric to a factor variable
mtcars$cyl <- as.factor(mtcars$cyl)
head(mtcars)
```

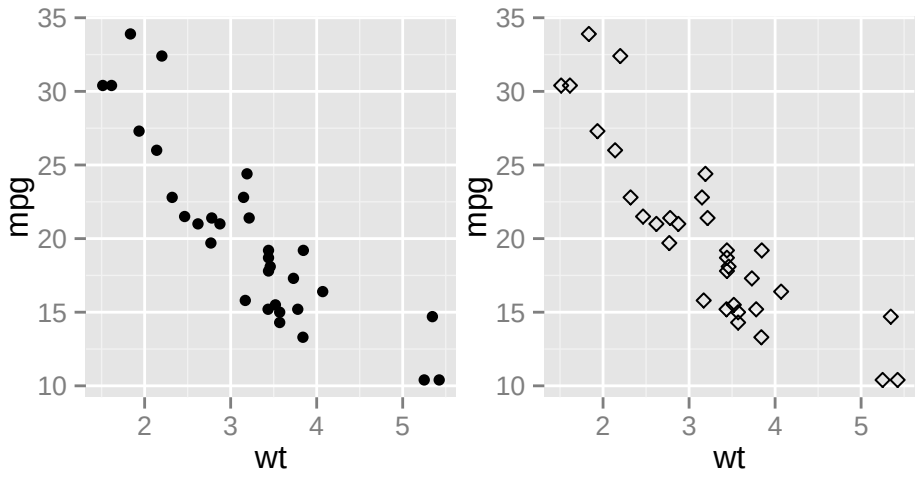
```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

8.2 Basic scatter plots

Simple scatter plots are created using the R code below. The color, the size and the shape of points can be changed using the function **geom_point()** as follow :

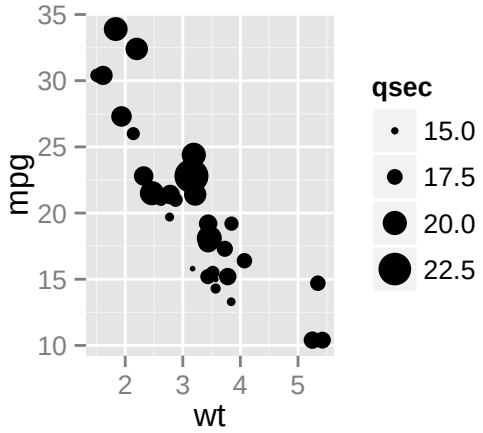
```
geom_point(size, color, shape)
```

```
library(ggplot2)

# Basic scatter plot
ggplot(mtcars, aes(x=wt, y=mpg)) + geom_point()

# Change the point size, and shape
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(size=2, shape=23)
```



Note that, the size of the points can be controlled by the values of a continuous variable as in the example below.

```
# Change the point size
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(aes(size=qsec))
```

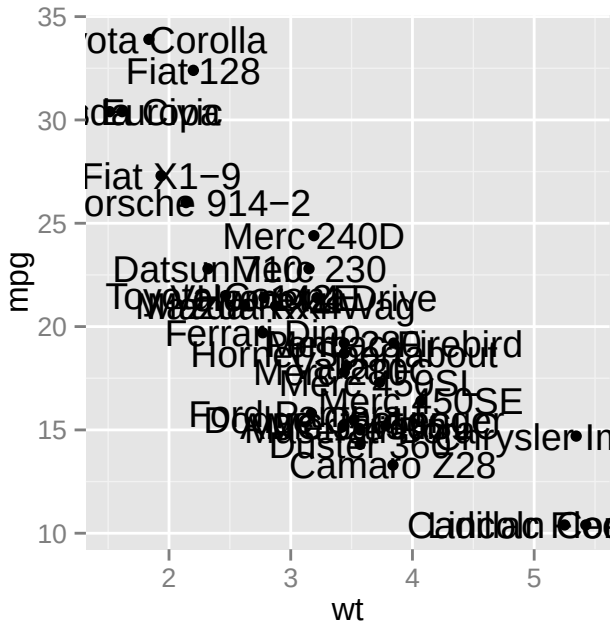


Read more on point shapes: Chapter 19

8.3 Label points in the scatter plot

The function **geom_text()** can be used :

```
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point() +
  geom_text(label=rownames(mtcars))
```



Read more on text annotations : Chapter 21

8.3.1 Add regression lines

The functions below can be used to add regression lines to a scatter plot :

- **geom_smooth()** and **stat_smooth()**
- **geom_abline()**

geom_abline() is described here: Chapter 22.

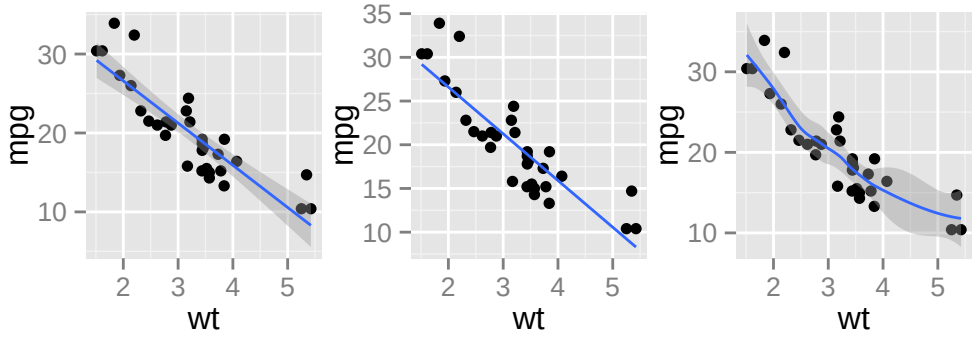
Only the function **geom_smooth()** is covered in this section.

A simplified format is :

```
geom_smooth(method="auto", se=TRUE, fullrange=FALSE, level=0.95)
```

- **method** : smoothing method to be used. Possible values are lm, glm, gam, loess, rlm.
- **se** : logical value. If TRUE, confidence interval is displayed around smooth.
- **fullrange** : logical value. If TRUE, the fit spans the full range of the plot
- **level** : level of confidence interval to use. Default value is 0.95

```
# Add the regression line
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth(method=lm)

# Remove the confidence interval
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth(method=lm, se=FALSE)

# Loess method
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth()
```

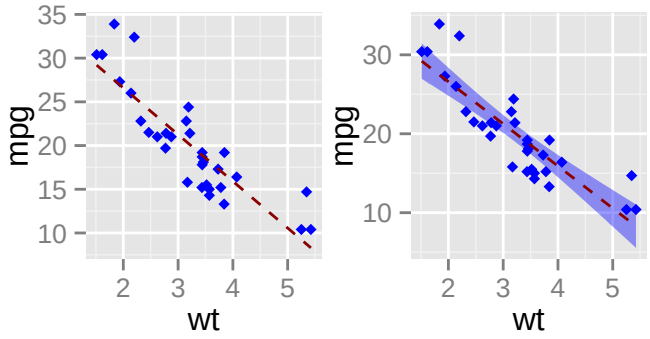


8.3.2 Change the appearance of points and lines

This section describes how to change :

- the color and the shape of points
- the line type and color of the regression line
- the fill color of the confidence interval

```
# Change the point colors and shapes
# Change the line type and color
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(shape=18, color="blue")+
  geom_smooth(method=lm, se=FALSE, linetype="dashed",
              color="darkred")

# Change the confidence interval fill color
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(shape=18, color="blue")+
  geom_smooth(method=lm,  linetype="dashed",
              color="darkred", fill="blue")
```



Note that a transparent color is used, by default, for the confidence band. This can be changed by using the argument *alpha* : *geom_smooth(fill="blue", alpha=1)*

Read more on point shapes : Chapter 19

Read more on line types : Chapter 20

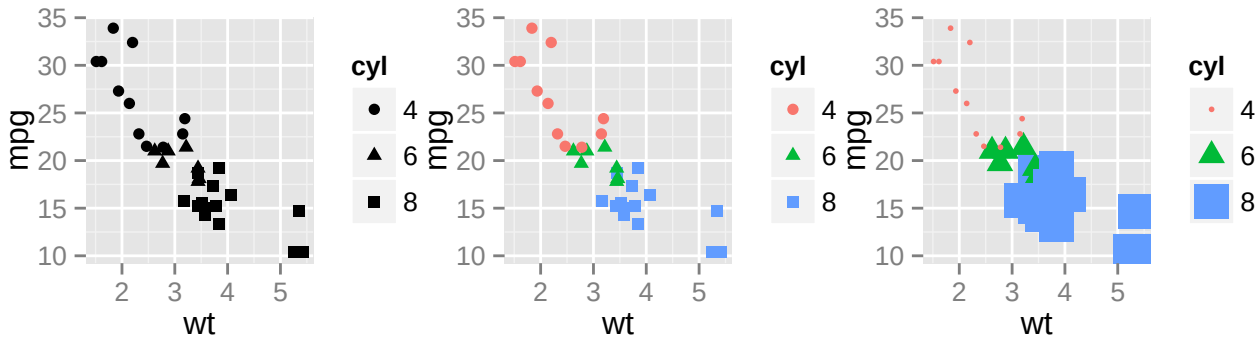
8.4 Scatter plots with multiple groups

This section describes how to change point colors and shapes automatically and manually.

8.4.1 Change the point color/shape/size automatically

In the R code below, point shapes, colors and sizes are controlled by the levels of the factor variable *cyl* :

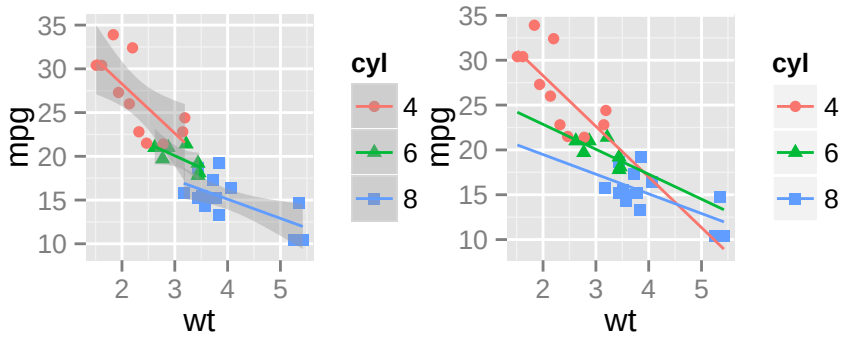
```
# Change point shapes by the levels of cyl
ggplot(mtcars, aes(x=wt, y=mpg, shape=cyl)) +
  geom_point()

# Change point shapes and colors
ggplot(mtcars, aes(x=wt, y=mpg, shape=cyl, color=cyl)) +
  geom_point()

# Change point shapes, colors and sizes
ggplot(mtcars, aes(x=wt, y=mpg, shape=cyl, color=cyl, size=cyl)) +
  geom_point()
```



8.4.2 Add regression lines

Regression lines can be added as follow :

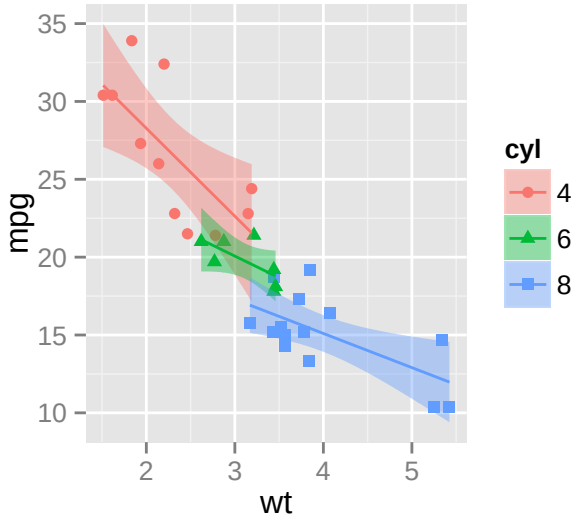
```
# Add regression lines
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
  geom_smooth(method=lm)

# Remove confidence intervals
# Extend the regression lines
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)
```



Note that, you can also change the line type of the regression lines by using the aesthetic *linetype = cyl*.

The fill color of confidence bands can be changed as follow :

```
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
  geom_smooth(method=lm, aes(fill=cyl))
```



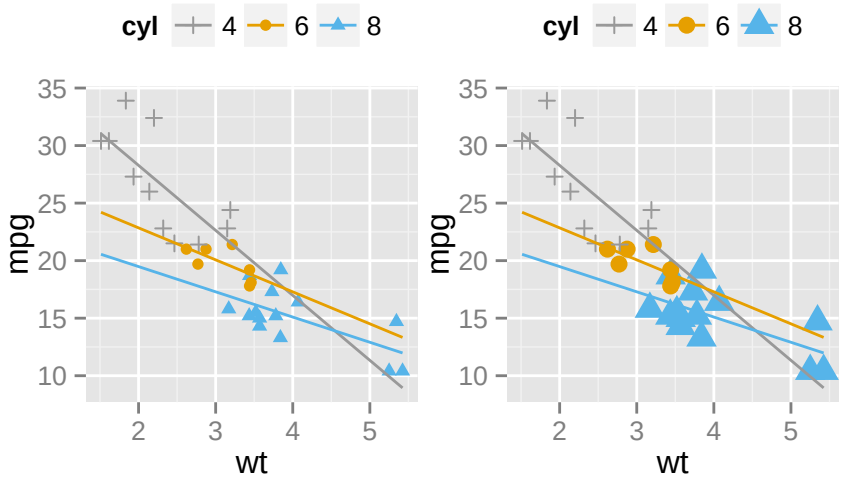
8.4.3 Change the point color/shape/size manually

The functions below are used :

- *scale_shape_manual()* for point shapes
- *scale_color_manual()* for point colors
- *scale_size_manual()* for point sizes

```
# Change point shapes and colors manually
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
```

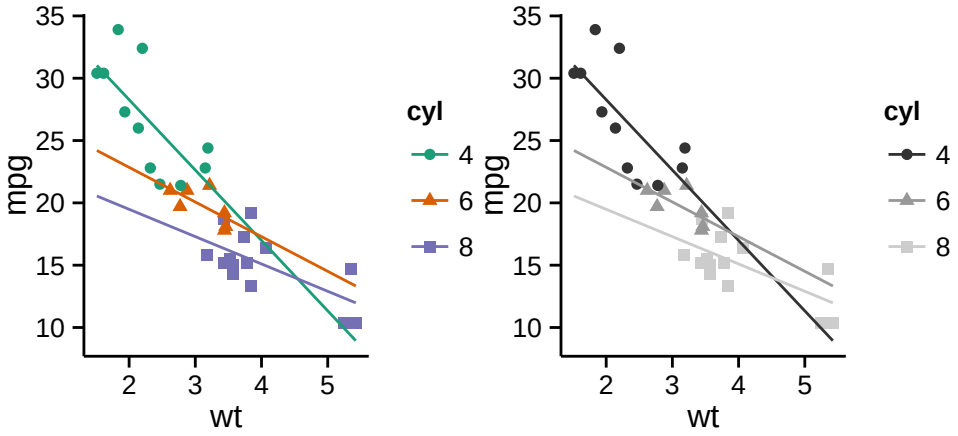
```
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)+
  scale_shape_manual(values=c(3, 16, 17))+
  scale_color_manual(values=c('#999999','#E69F00', '#56B4E9'))+
  theme(legend.position="top")

# Change the point sizes manually
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl))+
  geom_point(aes(size=cyl)) +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)+
  scale_shape_manual(values=c(3, 16, 17))+
  scale_color_manual(values=c('#999999','#E69F00', '#56B4E9'))+
  scale_size_manual(values=c(2,3,4))+
  theme(legend.position="top")
```



It is also possible to *change manually point and line colors* using the functions :

- *scale_color_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_color_grey()* : to use grey color palettes

```
p <- ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)+
  theme_classic()

# Use brewer color palettes
p+scale_color_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey()
```



Read more on ggplot2 colors here : Chapter 18

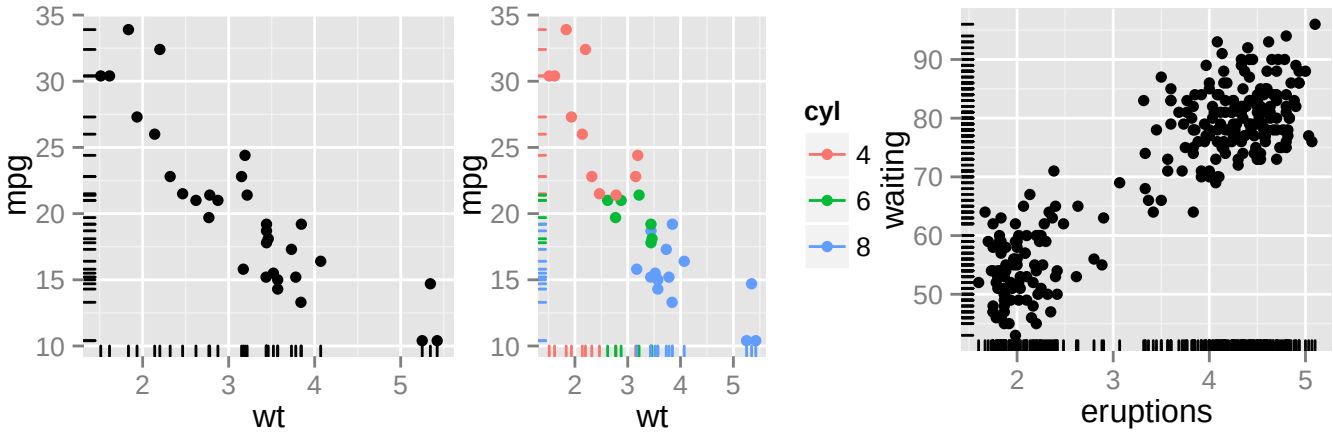
8.5 Add marginal rugs to a scatter plot

The function **geom_rug()** can be used :

```
geom_rug(sides ="bl")
```

Sides : a string that controls which sides of the plot the rugs appear on. Allowed value is a string containing any of "trbl", for top, right, bottom, and left.

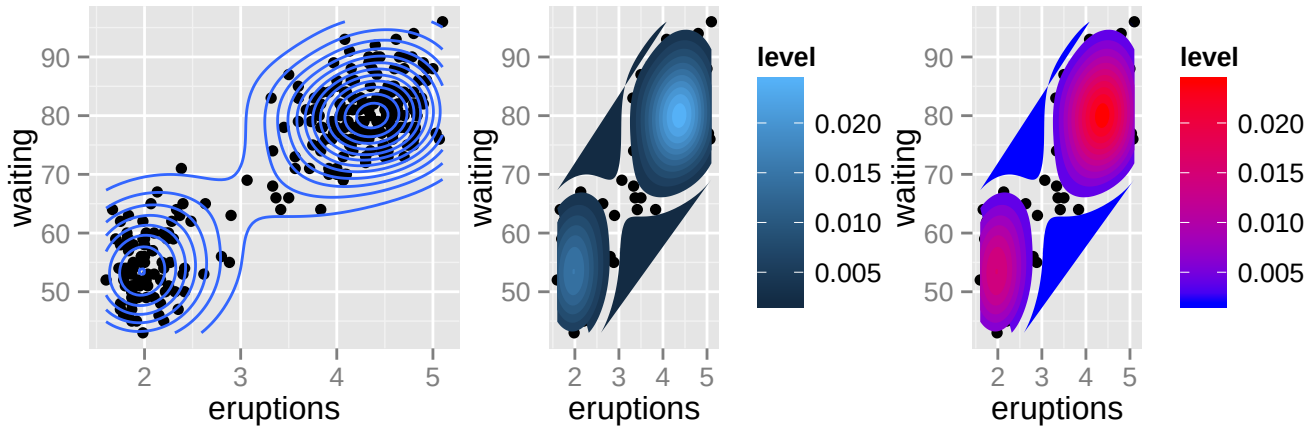
```
# Add marginal rugs
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point() + geom_rug()

# Change colors
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl)) +
  geom_point() + geom_rug()

# Add marginal rugs using faithful data
ggplot(faithful, aes(x=eruptions, y=waiting)) +
  geom_point() + geom_rug()
```



8.6 Scatter plots with the 2d density estimation

The functions **geom_density2d()** or **stat_density2d()** can be used :

```
# Scatter plot with the 2d density estimation
sp <- ggplot(faithful, aes(x=eruptions, y=waiting)) +
  geom_point()
sp + geom_density2d()

# Gradient color
sp + stat_density2d(aes(fill = ..level..), geom="polygon")

# Change the gradient color
sp + stat_density2d(aes(fill = ..level..), geom="polygon")+
  scale_fill_gradient(low="blue", high="red")
```



Read more on ggplot2 colors here : Chapter 18

8.7 Scatter plots with rectangular bins

The number of observations is counted in each bins and displayed using any of the functions below :

- **geom_bin2d()** for adding a heatmap of 2d bin counts
- **stat_bin2d()** for counting the number of observation in rectangular bins
- **stat_summary2d()** to apply function for 2D rectangular bins

The simplified formats of these functions are :

```
plot + geom_bin2d(...)

plot+stat_bin2d(geom=NULL, bins=30)

plot + stat_summary2d(geom = NULL, bins = 30, fun = mean)
```

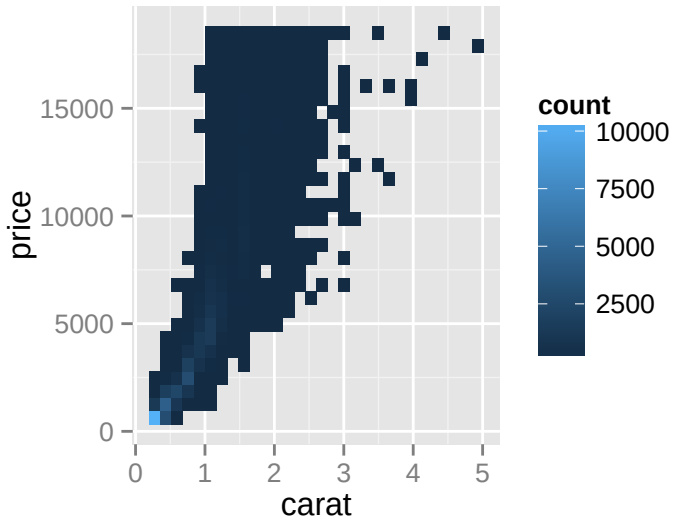
- **geom** : geometrical object to display the data
- **bins** : Number of bins in both vertical and horizontal directions. The default value is 30
- **fun** : function for summary

The data sets *diamonds* from ggplot2 package is used :

```
head(diamonds)
```

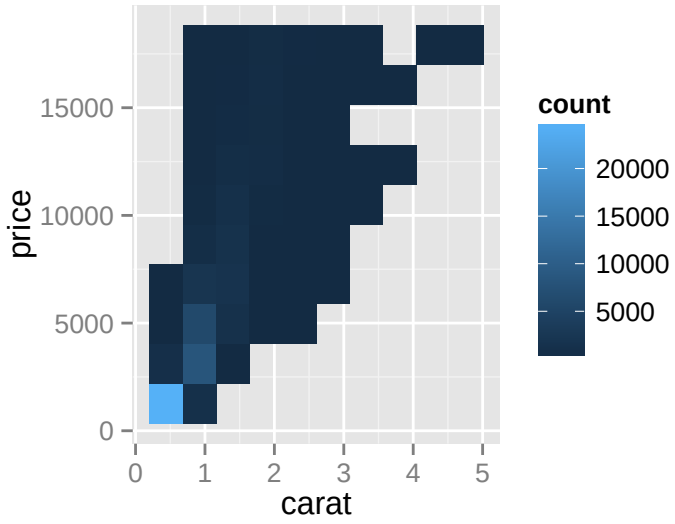
```
##   carat       cut color clarity depth table price    x    y    z
## 1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
## 2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
## 3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
## 4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
## 5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
## 6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

```
# Plot
p <- ggplot(diamonds, aes(carat, price))
p + geom_bin2d()
```



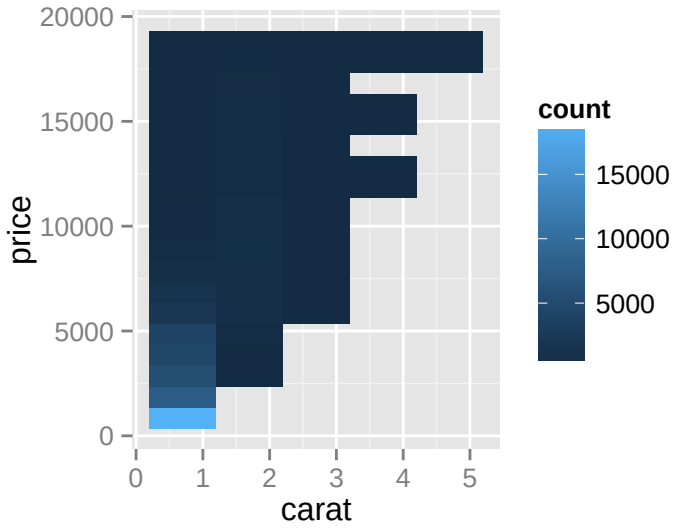
Change the number of bins :

```
# Change the number of bins
p + geom_bin2d(bins=10)
```



Or specify the width of bins :

```
# Or specify the width of bins
p + geom_bin2d(binwidth=c(1, 1000))
```



8.8 Scatter plot with marginal density distribution plot

Step 1/3. Create some data :

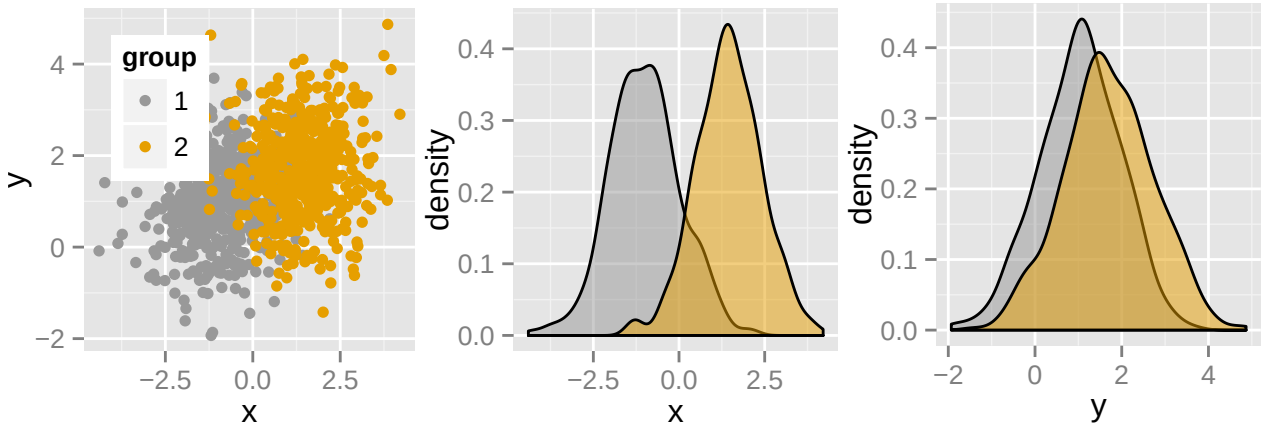
```
set.seed(1234)
x <- c(rnorm(500, mean = -1), rnorm(500, mean = 1.5))
y <- c(rnorm(500, mean = 1), rnorm(500, mean = 1.7))
group <- as.factor(rep(c(1,2), each=500))
df <- data.frame(x, y, group)
head(df)
```

```
##             x          y group
## 1 -2.20706575 -0.2053334     1
## 2 -0.72257076  1.3014667     1
## 3  0.08444118 -0.5391452     1
## 4 -3.34569770  1.6353707     1
## 5 -0.57087531  1.7029518     1
## 6 -0.49394411 -0.9058829     1
```

Step 2/3. Create the plots :

```
# scatter plot of x and y variables
# color by groups
scatterPlot <- ggplot(df,aes(x, y, color=group)) +
  geom_point() +
  scale_color_manual(values = c('#999999','#E69F00')) +
```

```
  theme(legend.position=c(0,1), legend.justification=c(0,1))
scatterPlot


# Marginal density plot of x (top panel)
xdensity <- ggplot(df, aes(x, fill=group)) +
  geom_density(alpha=.5) +
  scale_fill_manual(values = c('#999999','#E69F00')) +
  theme(legend.position = "none")
xdensity

# Marginal density plot of y (right panel)
ydensity <- ggplot(df, aes(y, fill=group)) +
  geom_density(alpha=.5) +
  scale_fill_manual(values = c('#999999','#E69F00')) +
  theme(legend.position = "none")
ydensity
```



Create a blank placeholder plot :

```
blankPlot <- ggplot()+geom_blank(aes(1,1))+
  theme(plot.background = element_blank(),
   panel.grid.major = element_blank(),
   panel.grid.minor = element_blank(),
   panel.border = element_blank(),
   panel.background = element_blank(),
   axis.title.x = element_blank(),
   axis.title.y = element_blank(),
   axis.text.x = element_blank(),
   axis.text.y = element_blank(),
   axis.ticks = element_blank()
     )
```

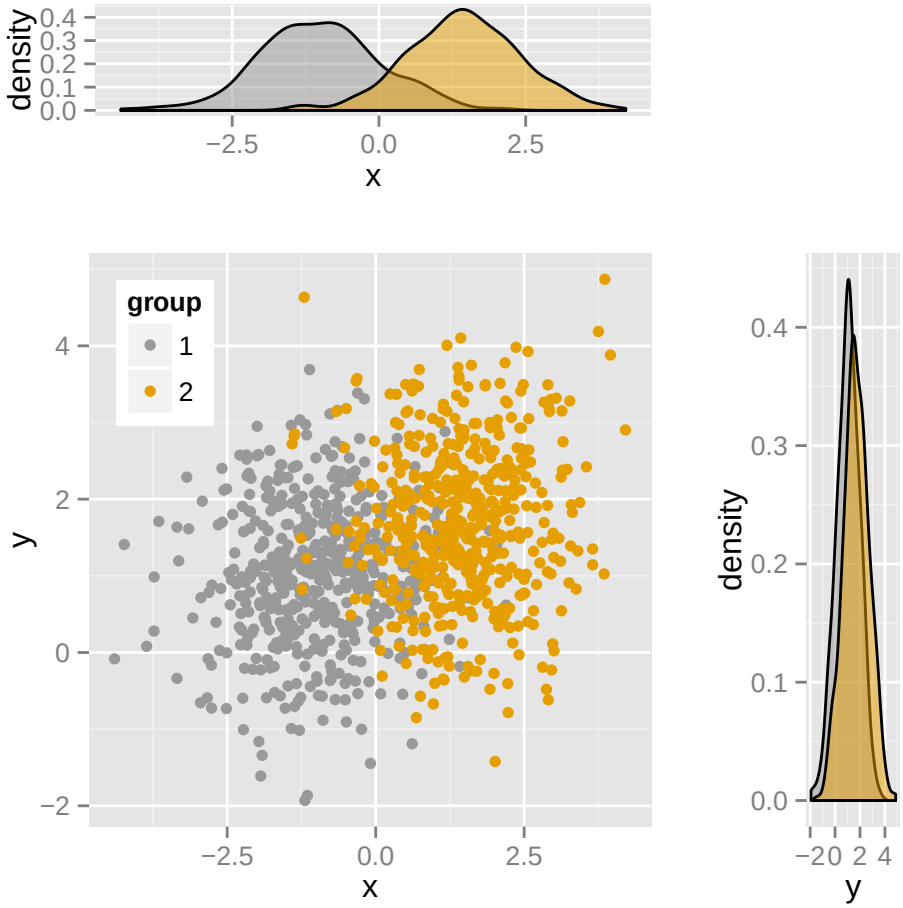
Step 3/3. Put the plots together:

To put multiple plots on the same page, the package **gridExtra** can be used. Install the package as follow :

```
install.packages("gridExtra")
```

Arrange ggplot2 with adapted height and width for each row and column :

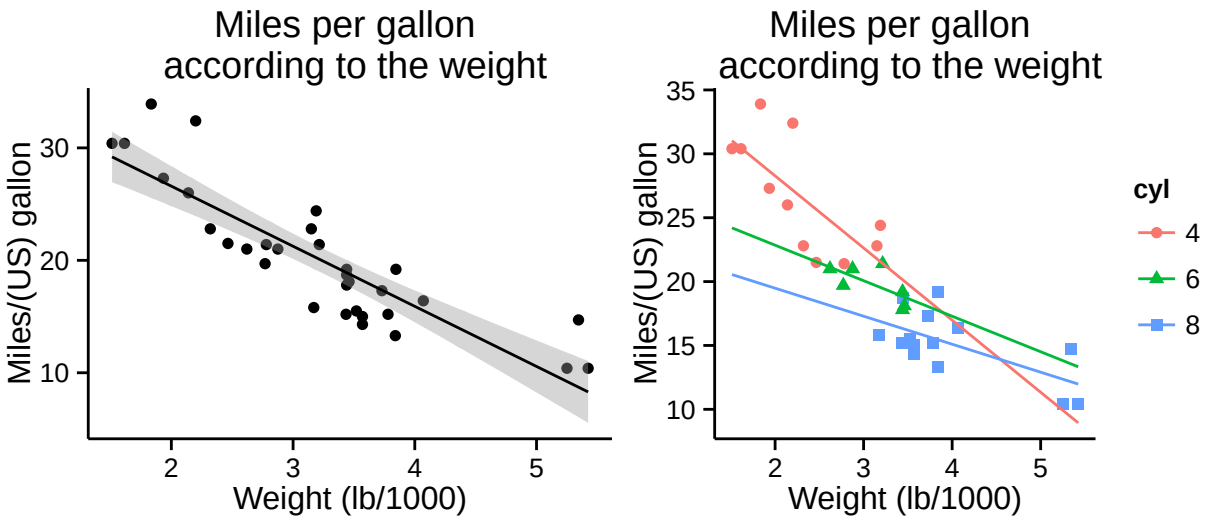
```
library("gridExtra")
grid.arrange(xdensity, blankPlot, scatterPlot, ydensity,
        ncol=2, nrow=2, widths=c(4, 1.4), heights=c(1.4, 4))
```



Read more on how to arrange multiple ggplots in one page : Chapter 28

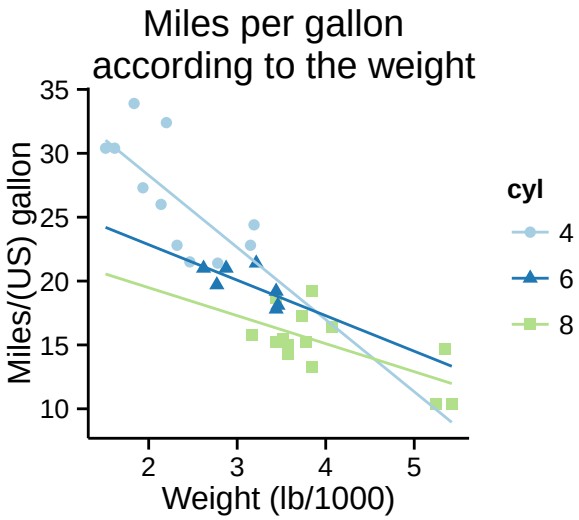
8.9 Customized scatter plots

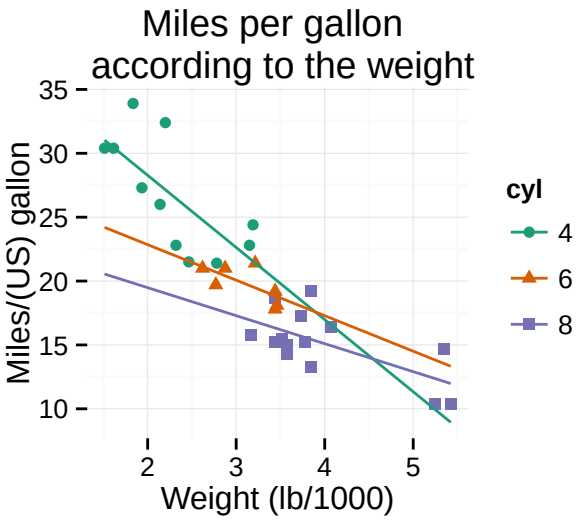
```
# Basic scatter plot
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth(method=lm, color="black")+
  labs(title="Miles per gallon \n according to the weight",
       x="Weight (lb/1000)", y = "Miles/(US) gallon")+
  theme_classic()

# Change color/shape by groups
# Remove confidence bands
p <- ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point()+
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)+
  labs(title="Miles per gallon \n according to the weight",
       x="Weight (lb/1000)", y = "Miles/(US) gallon")

p + theme_classic()
```

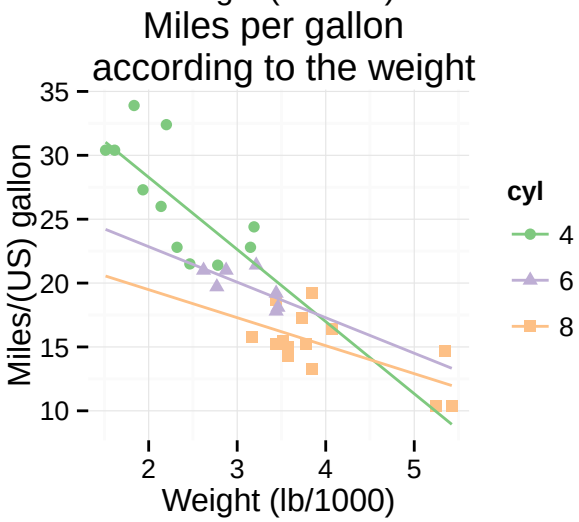


Change colors manually :

```
# Continuous colors
p + scale_color_brewer(palette="Paired") + theme_classic()

# Discrete colors
p + scale_color_brewer(palette="Dark2") + theme_minimal()

# Gradient colors
p + scale_color_brewer(palette="Accent") + theme_minimal()
```





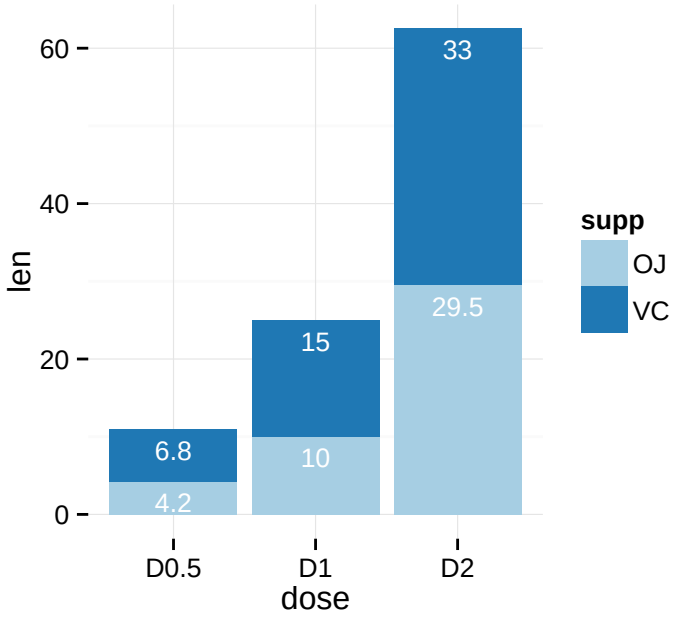


Read more on ggplot2 colors here : Chapter 18

Chapter 9

Bar plots

The function **geom_bar()** can be used to create a **bar plot**.



Key functions: **geom_bar()** and **geom_errorbar()**

9.1 Basic bar plots

9.1.1 Data

Data derived from *ToothGrowth* data sets are used. ToothGrowth describes the effect of Vitamin C on Tooth growth in Guinea pigs.

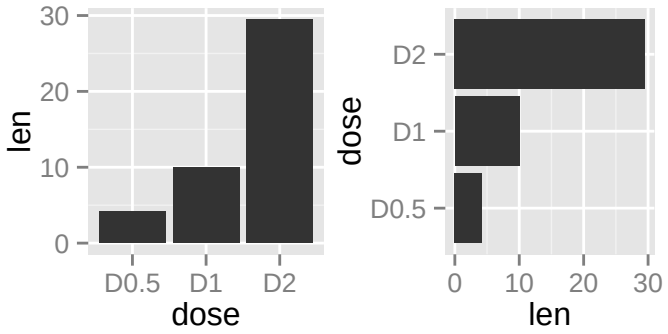
```
df <- data.frame(dose=c("D0.5", "D1", "D2"),
                 len=c(4.2, 10, 29.5))

head(df)
```

```
##   dose  len
## 1 D0.5  4.2
## 2   D1 10.0
## 3   D2 29.5
```

- *len* : Tooth length
- *dose* : Dose in milligrams (0.5, 1, 2)

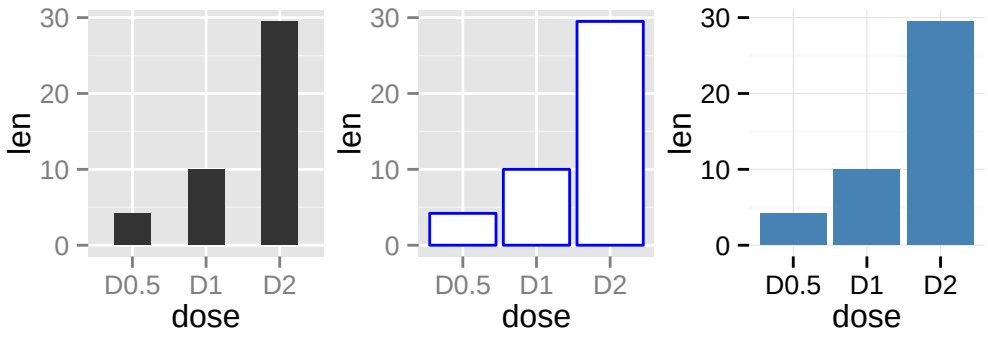
9.1.2 Create bar plots

```
library(ggplot2)
# Basic bar plot
p<-ggplot(data=df, aes(x=dose, y=len)) +
  geom_bar(stat="identity")
p

# Horizontal bar plot
p + coord_flip()
```



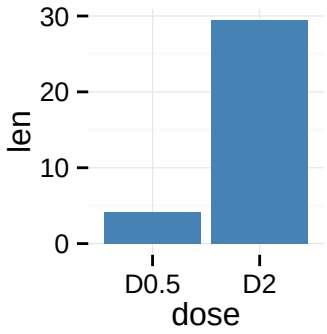
Change the width and the color of bars :

```
# Change the width of bars
ggplot(data=df, aes(x=dose, y=len)) +
  geom_bar(stat="identity", width=0.5)

# Change colors
ggplot(data=df, aes(x=dose, y=len)) +
  geom_bar(stat="identity", color="blue", fill="white")

# Minimal theme + blue fill color
p<-ggplot(data=df, aes(x=dose, y=len)) +
  geom_bar(stat="identity", fill="steelblue")+
  theme_minimal()
p
```



Choose which items to display :

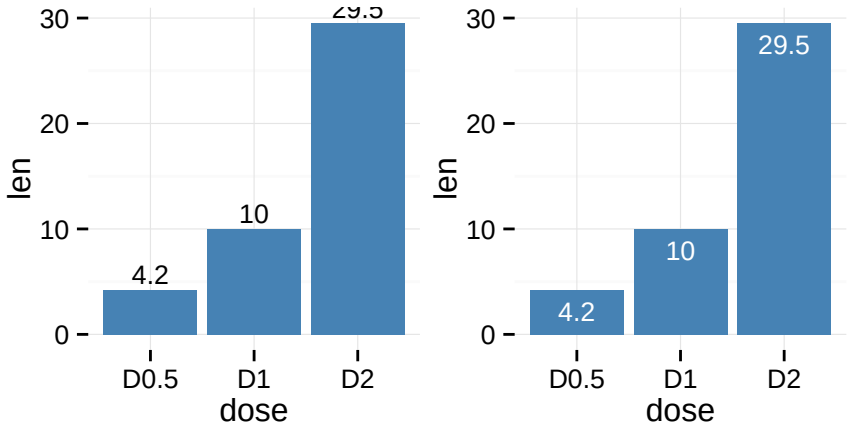
```
p + scale_x_discrete(limits=c("D0.5", "D2"))
```



9.1.3 Bar plot with labels

```
# Outside bars
ggplot(data=df, aes(x=dose, y=len)) +
  geom_bar(stat="identity", fill="steelblue")+
```

```
  geom_text(aes(label=len), vjust=-0.3, size=3.5)+
  theme_minimal()

# Inside bars
ggplot(data=df, aes(x=dose, y=len)) +
  geom_bar(stat="identity", fill="steelblue")+
  geom_text(aes(label=len), vjust=1.6, color="white", size=3.5)+
  theme_minimal()
```



9.1.4 Bar plot of counts

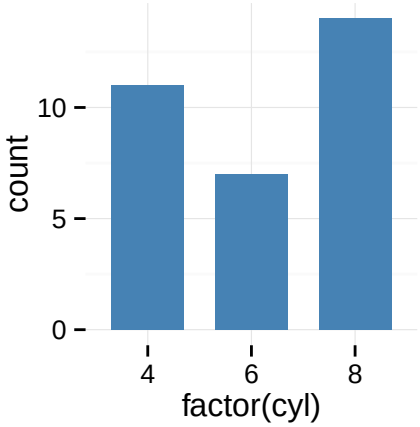
In the R code above, we used the argument **stat = "identity"** to make bar plots. Note that, the default value of the argument *stat* is *"bin"*. In this case, the height of the bar represents the count of cases in each category.

To make a bar plot of counts, we will use the *mtcars* data sets :

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# Don't map a variable to y
ggplot(mtcars, aes(x=factor(cyl)))+
  geom_bar(stat="bin", width=0.7, fill="steelblue")+
  theme_minimal()
```

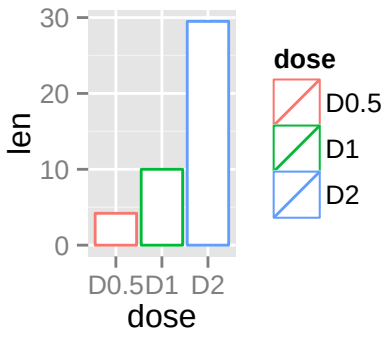


9.2 Change bar plot colors by groups

9.2.1 Change outline colors

Bar plot outline colors can be automatically controlled by the levels of the variable *dose* :

```
# Change bar plot line colors by groups
p<-ggplot(df, aes(x=dose, y=len, color=dose)) +
  geom_bar(stat="identity", fill="white")
p
```

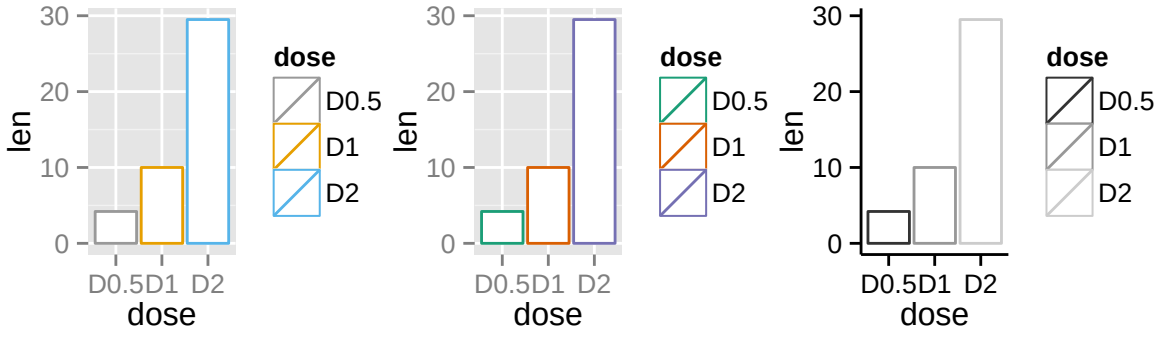


It is also possible to *change manually bar plot line colors* using the functions :

- *scale_color_manual()* : to use custom colors
- *scale_color_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_color_grey()* : to use grey color palettes

```
# Use custom color palettes
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
```

```
p+scale_color_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey() + theme_classic()
```

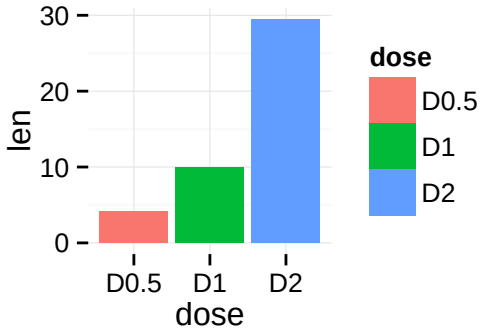


Read more on ggplot2 colors here : Chapter 18

9.2.2 Change fill colors

In the R code below, bar plot fill colors are automatically controlled by the levels of *dose* :

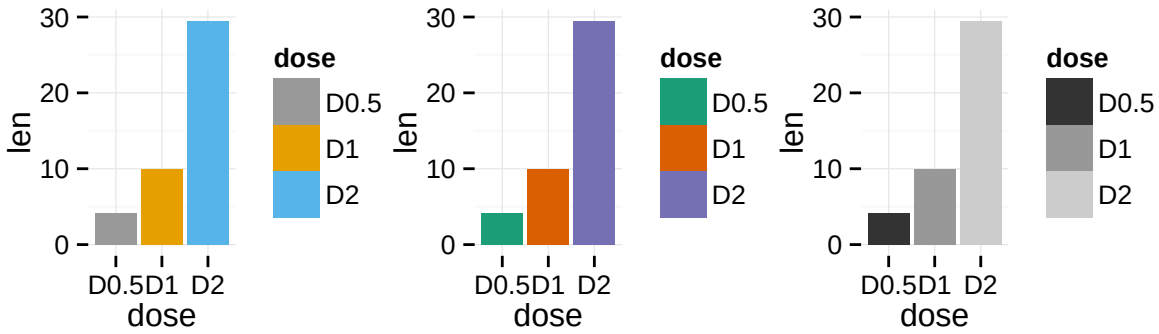
```
# Change bar plot fill colors by groups
p<-ggplot(df, aes(x=dose, y=len, fill=dose)) +
  geom_bar(stat="identity")+theme_minimal()
p
```



It is also possible to change manually bar plot fill colors using the functions :

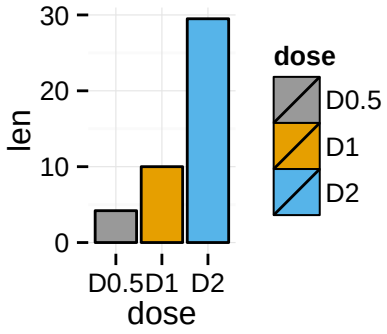
- *scale_fill_manual()* : to use custom colors
- *scale_fill_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_fill_grey()* : to use grey color palettes

```
# Use custom color palettes
p+scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# use brewer color palettes
p+scale_fill_brewer(palette="Dark2")

# Use grey scale
p + scale_fill_grey()
```



Use black outline color :

```
ggplot(df, aes(x=dose, y=len, fill=dose))+
geom_bar(stat="identity", color="black")+
scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))+
  theme_minimal()
```

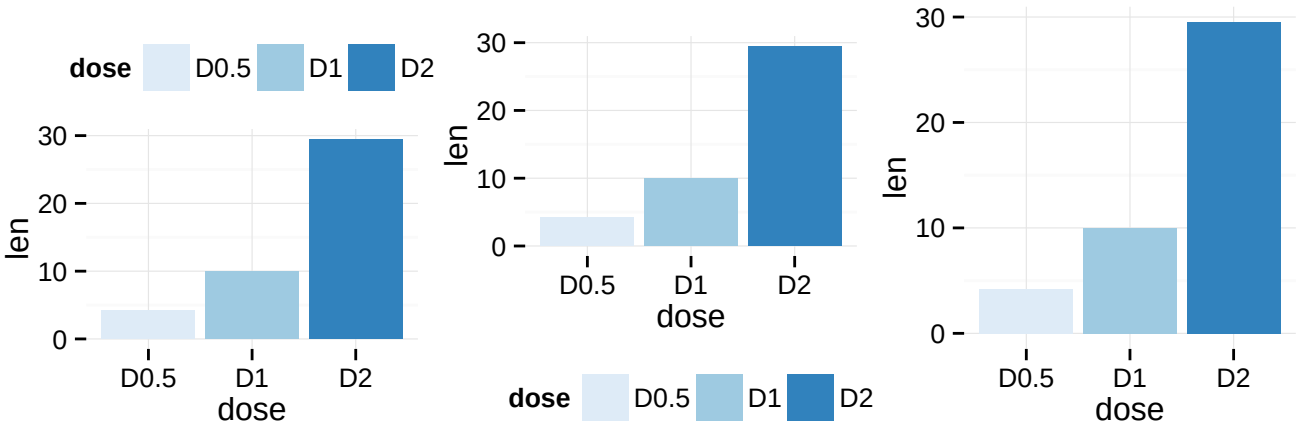


Read more on ggplot2 colors here : Chapter 18

9.3 Change the legend position

```
# Change bar fill colors to blues
p <- p+scale_fill_brewer(palette="Blues")
```

```
p + theme(legend.position="top")

p + theme(legend.position="bottom")

# Remove legend
p + theme(legend.position="none")
```



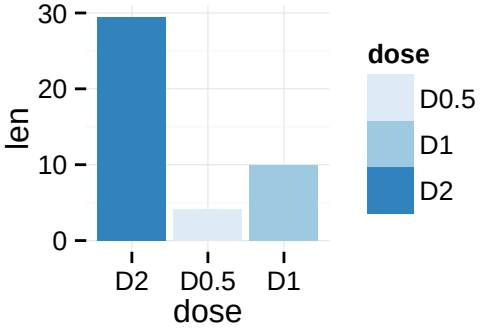
The allowed values for the arguments **legend.position** are : "left","top", "right", "bottom".

Read more on ggplot legend : Chapter 17

9.4 Change the order of items in the legend

The function **scale_x_discrete** can be used to change the order of items to "2", "0.5", "1" :

```
p + scale_x_discrete(limits=c("D2", "D0.5", "D1"))
```



9.5 Bar plot with multiple groups

9.5.1 Data

Data derived from *ToothGrowth* data sets are used. ToothGrowth describes the effect of Vitamin C on tooth growth in Guinea pigs. Three dose levels of Vitamin C (0.5, 1, and 2 mg) with each of two delivery methods [orange juice (OJ) or ascorbic acid (VC)] are used :

```
df2 <- data.frame(supp=rep(c("VC", "OJ"), each=3),
                  dose=rep(c("D0.5", "D1", "D2"),2),
                  len=c(6.8, 15, 33, 4.2, 10, 29.5))

head(df2)
```

```
##   supp dose  len
## 1   VC D0.5  6.8
## 2   VC   D1 15.0
## 3   VC   D2 33.0
## 4   OJ D0.5  4.2
## 5   OJ   D1 10.0
## 6   OJ   D2 29.5
```

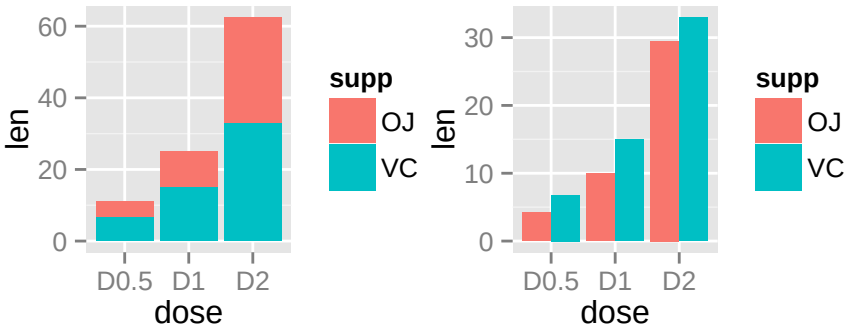
- *len* : Tooth length

- *dose* : Dose in milligrams (0.5, 1, 2)
- *supp* : Supplement type (VC or OJ)

9.5.2 Create bar plots

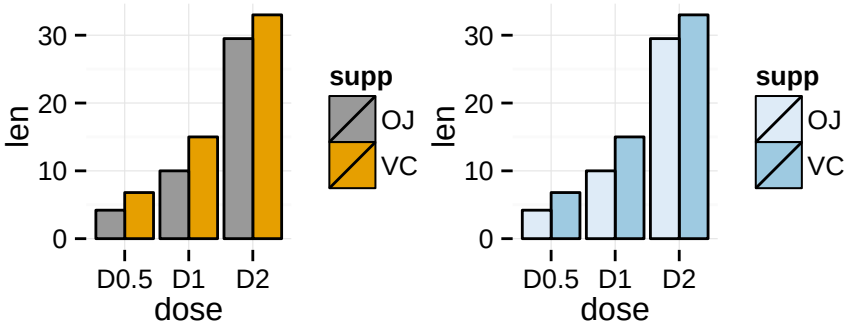
A stacked bar plot is created by default. You can use the function *position_dodge()* to change this. The bar plot fill color is controlled by the levels of *dose* :

```
# Stacked bar plot with multiple groups
ggplot(data=df2, aes(x=dose, y=len, fill=supp)) +
  geom_bar(stat="identity")

# Use position=position_dodge()
ggplot(data=df2, aes(x=dose, y=len, fill=supp)) +
geom_bar(stat="identity", position=position_dodge())
```



Change the color manually :

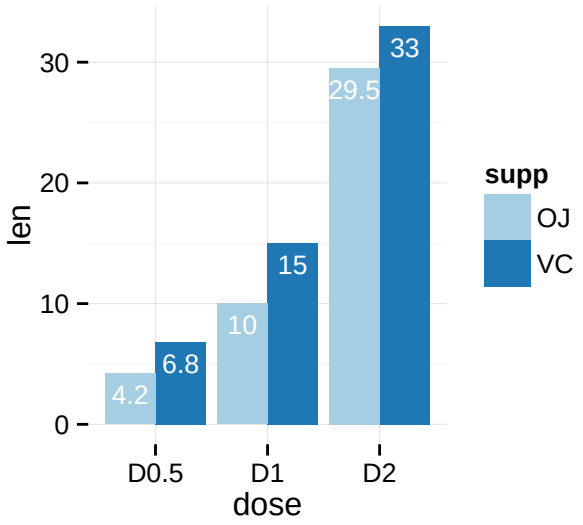
```
# Change the colors manually
p <- ggplot(data=df2, aes(x=dose, y=len, fill=supp)) +
geom_bar(stat="identity", color="black", position=position_dodge())+
  theme_minimal()

# Use custom colors
p + scale_fill_manual(values=c('#999999','#E69F00'))

# Use brewer color palettes
p + scale_fill_brewer(palette="Blues")
```



9.5.3 Add labels

Add labels to a dodged bar plot :

```
ggplot(data=df2, aes(x=dose, y=len, fill=supp)) +
  geom_bar(stat="identity", position=position_dodge())+
  geom_text(aes(label=len), vjust=1.6, color="white",
            position = position_dodge(0.9), size=3.5)+
  scale_fill_brewer(palette="Paired")+
  theme_minimal()
```



Add labels to a stacked bar plot : 3 steps are required

1. Sort the data by dose and supp : the package **plyr** is used
2. Calculate the cumulative sum of the variable *len* for each dose
3. Create the plot

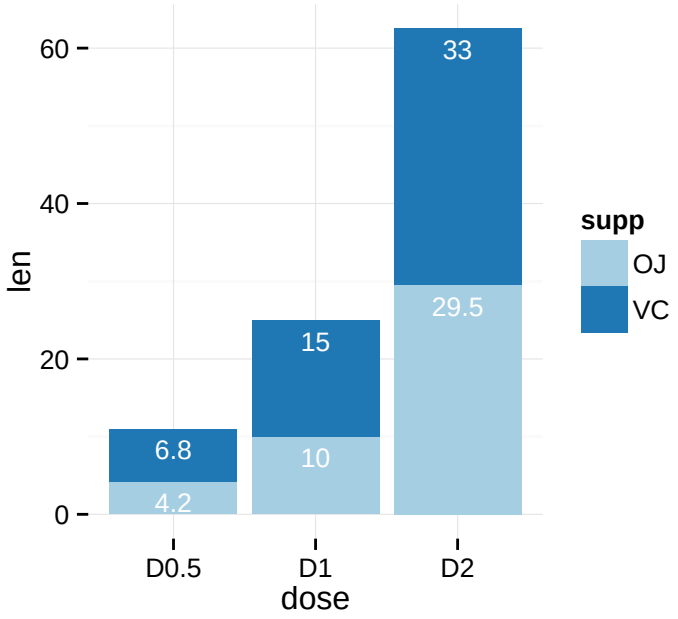
```
library(plyr)
# Sort by dose and supp
df_sorted <- arrange(df2, dose, supp)
head(df_sorted)
```

```
##   supp dose  len
## 1   OJ D0.5  4.2
## 2   VC D0.5  6.8
## 3   OJ   D1 10.0
## 4   VC   D1 15.0
## 5   OJ   D2 29.5
## 6   VC   D2 33.0
```

```
# Calculate the cumulative sum of len for each dose
df_cumsum <- ddply(df_sorted, "dose",
                   transform, label_ypos=cumsum(len))
head(df_cumsum)
```

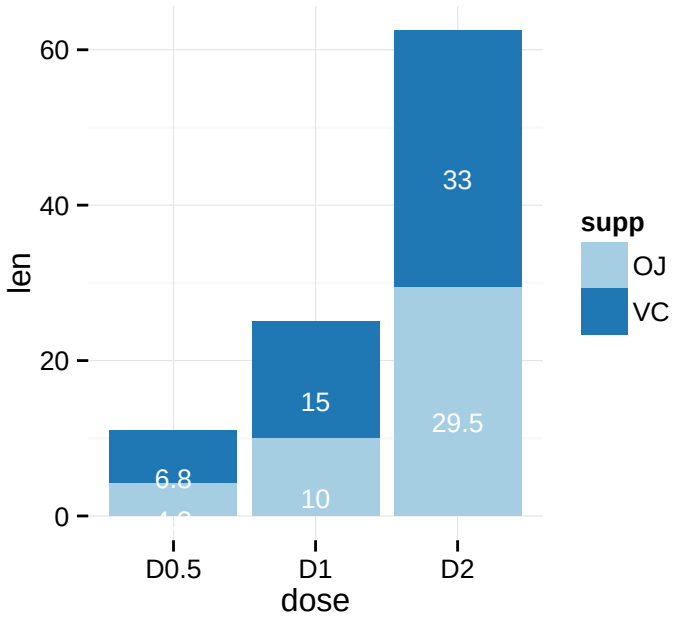
```
##   supp dose  len label_ypos
## 1   OJ D0.5  4.2        4.2
## 2   VC D0.5  6.8       11.0
## 3   OJ   D1 10.0       10.0
## 4   VC   D1 15.0       25.0
## 5   OJ   D2 29.5       29.5
## 6   VC   D2 33.0       62.5
```

```
# Create the bar plot
ggplot(data=df_cumsum, aes(x=dose, y=len, fill=supp)) +
  geom_bar(stat="identity")+
  geom_text(aes(y=label_ypos, label=len), vjust=1.6,
            color="white", size=3.5)+
  scale_fill_brewer(palette="Paired")+
  theme_minimal()
```



If you want to place the labels at the middle of bars, you have to modify the cumulative sum as follow :

```
df_cumsum <- ddply(df_sorted, "dose",
                   transform,
                   label_ypos=cumsum(len) - 0.5*len)

# Create the bar plot
ggplot(data=df_cumsum, aes(x=dose, y=len, fill=supp)) +
  geom_bar(stat="identity")+
  geom_text(aes(y=label_ypos, label=len), vjust=1.6,
            color="white", size=3.5)+
  scale_fill_brewer(palette="Paired")+
  theme_minimal()
```



9.6 Bar plot with a numeric x-axis

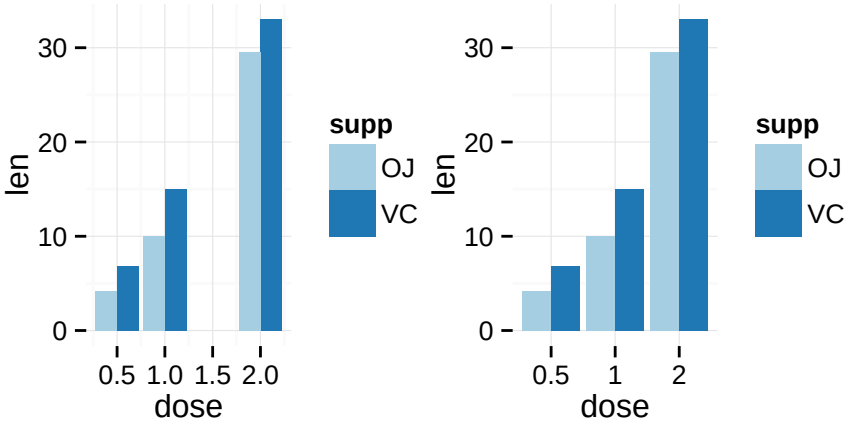
If the variable on x-axis is numeric, it can be useful to treat it as a continuous or a factor variable depending on what you want to do :

```
# Create some data
df2 <- data.frame(supp=rep(c("VC", "OJ"), each=3),
                  dose=rep(c("0.5", "1", "2"),2),
                  len=c(6.8, 15, 33, 4.2, 10, 29.5))
head(df2)
```

```
##   supp dose  len
## 1   VC  0.5  6.8
## 2   VC    1 15.0
## 3   VC    2 33.0
## 4   OJ  0.5  4.2
## 5   OJ    1 10.0
## 6   OJ    2 29.5
```

```
# x axis treated as continuous variable
df2$dose <- as.numeric(as.vector(df2$dose))
ggplot(data=df2, aes(x=dose, y=len, fill=supp)) +
  geom_bar(stat="identity", position=position_dodge())+
  scale_fill_brewer(palette="Paired")+
  theme_minimal()
```

```
# Axis treated as discrete variable
df2$dose<-as.factor(df2$dose)
ggplot(data=df2, aes(x=dose, y=len, fill=supp)) +
  geom_bar(stat="identity", position=position_dodge())+
  scale_fill_brewer(palette="Paired")+
  theme_minimal()
```



9.7 Bar plot with error bars

The helper function below will be used to calculate the mean and the standard deviation, for the variable of interest, in each group :

```
#++++++++++++++++++++++++
# Function to calculate the mean and the standard deviation
  # for each group
#++++++++++++++++++++++++
# data : a data frame
# varname : the name of a column containing the variable
  #to be summariezed
# groupnames : vector of column names to be used as
  # grouping variables
data_summary <- function(data, varname, groupnames){
  require(plyr)
  summary_func <- function(x, col){
    c(mean = mean(x[[col]], na.rm=TRUE),
      sd = sd(x[[col]], na.rm=TRUE))
  }
  data_sum<-ddply(data, groupnames, .fun=summary_func, varname)
  data_sum <- rename(data_sum, c("mean" = varname))
 return(data_sum)
}
```

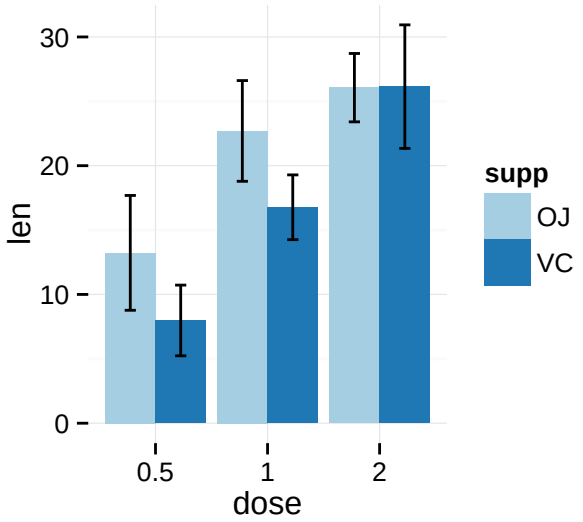
Summarize the data :

```
df3 <- data_summary(ToothGrowth, varname="len",
                    groupnames=c("supp", "dose"))
# Convert dose to a factor variable
df3$dose=as.factor(df3$dose)
head(df3)
```

```
##   supp dose   len       sd
## 1   OJ  0.5 13.23 4.459709
## 2   OJ    1 22.70 3.910953
## 3   OJ    2 26.06 2.655058
## 4   VC  0.5  7.98 2.746634
## 5   VC    1 16.77 2.515309
## 6   VC    2 26.14 4.797731
```

The function **geom_errorbar()** can be used to produce a bar graph with error bars :

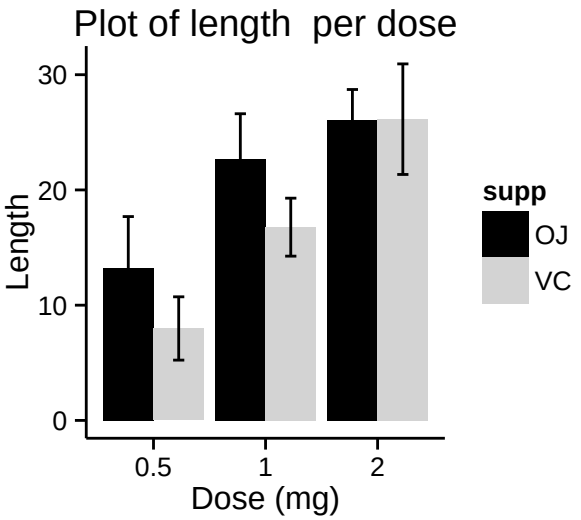
```
# Standard deviation of the mean as error bar
p <- ggplot(df3, aes(x=dose, y=len, fill=supp)) +
   geom_bar(stat="identity", position=position_dodge()) +
  geom_errorbar(aes(ymin=len-sd, ymax=len+sd), width=.2,
                position=position_dodge(.9))


p + scale_fill_brewer(palette="Paired") + theme_minimal()
```



Read more on error bars: Chapter 11

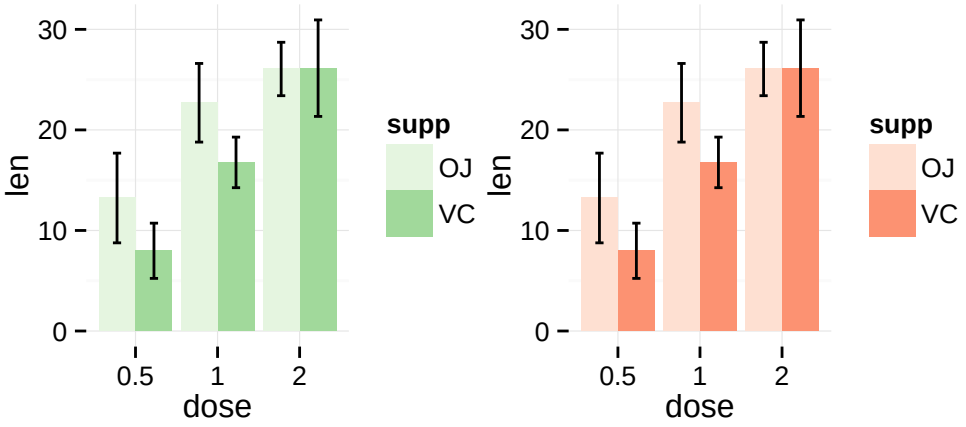
9.8 Customized bar plots

```
# Change color by groups
# Add error bars
p + labs(title="Plot of length  per dose",
         x="Dose (mg)", y = "Length")+
   scale_fill_manual(values=c('black','lightgray'))+
   theme_classic()
```



Change fill colors manually :

```
# Greens
p + scale_fill_brewer(palette="Greens") + theme_minimal()

# Reds
p + scale_fill_brewer(palette="Reds") + theme_minimal()
```



Chapter 10

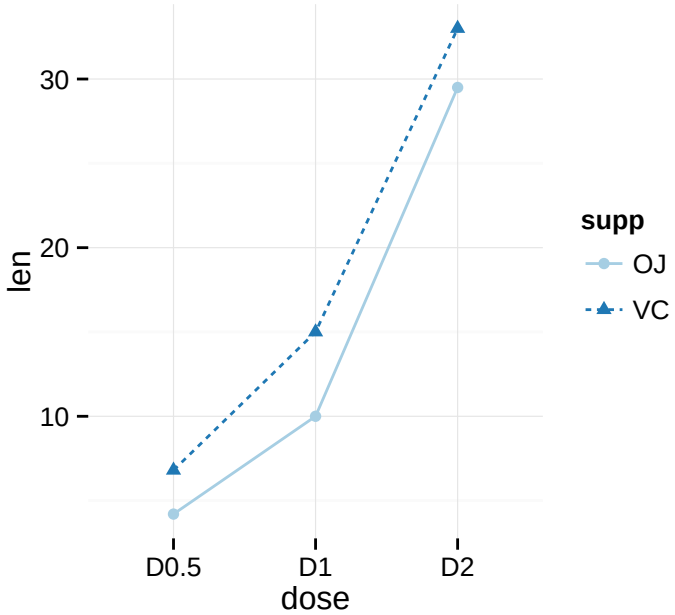
Line plots

In a line graph, observations are ordered by x value and connected.

The functions **geom_line()**, **geom_step()**, or **geom_path()** can be used.

x value (for x axis) can be :

- date : for a time series data
- texts
- discrete numeric values
- continuous numeric values



Key functions:

- geom_line()
- geom_step()

- geom_path()
- geom_errorbar()

10.1 Basic line plots

10.1.1 Data

Data derived from *ToothGrowth* data sets are used. ToothGrowth describes the effect of Vitamin C on tooth growth in Guinea pigs.

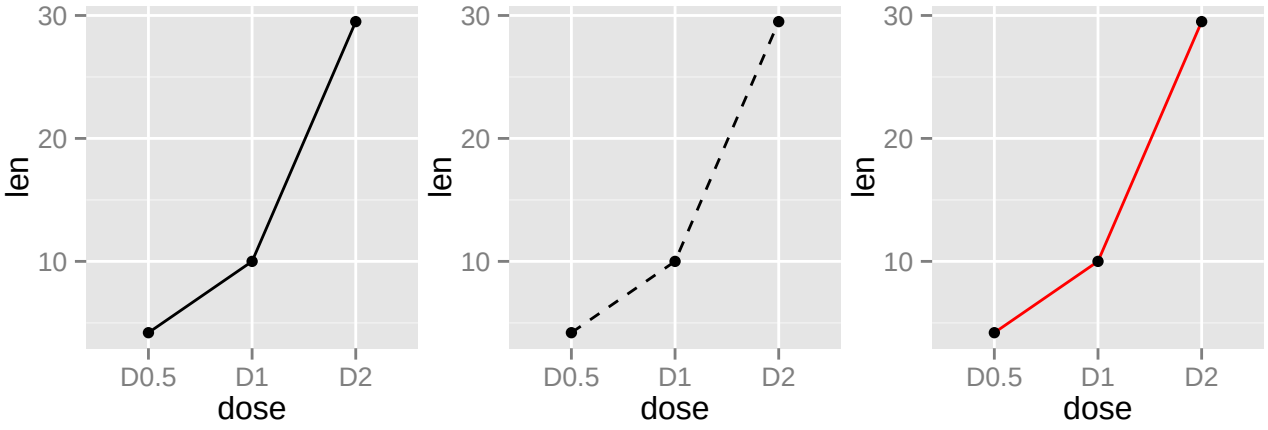
```
df <- data.frame(dose=c("D0.5", "D1", "D2"),
                 len=c(4.2, 10, 29.5))

head(df)
```

```
##   dose  len
## 1 D0.5  4.2
## 2   D1 10.0
## 3   D2 29.5
```

- *len* : Tooth length
- *dose* : Dose in milligrams (0.5, 1, 2)

10.1.2 Create line plots with points

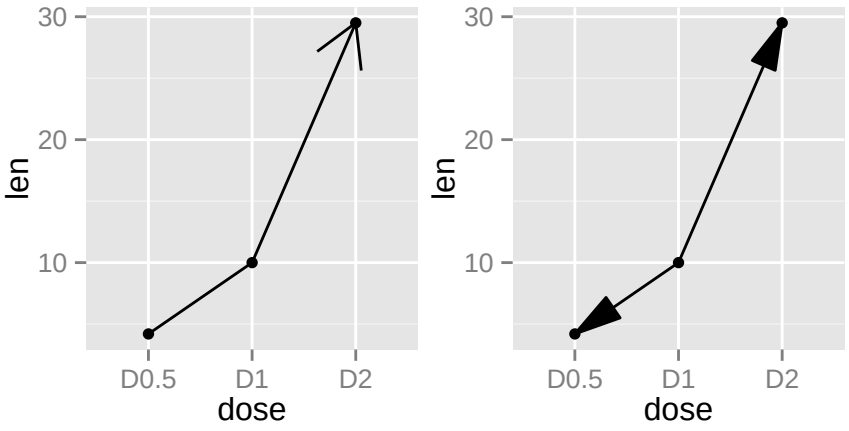
```
library(ggplot2)
# Basic line plot with points
ggplot(data=df, aes(x=dose, y=len, group=1)) +
  geom_line()+
  geom_point()

# Change the line type
ggplot(data=df, aes(x=dose, y=len, group=1)) +
  geom_line(linetype = "dashed")+
  geom_point()

# Change the color
ggplot(data=df, aes(x=dose, y=len, group=1)) +
  geom_line(color="red")+
  geom_point()
```



Read more on line types : Chapter 20

You can add an arrow to the line using the *grid* package :

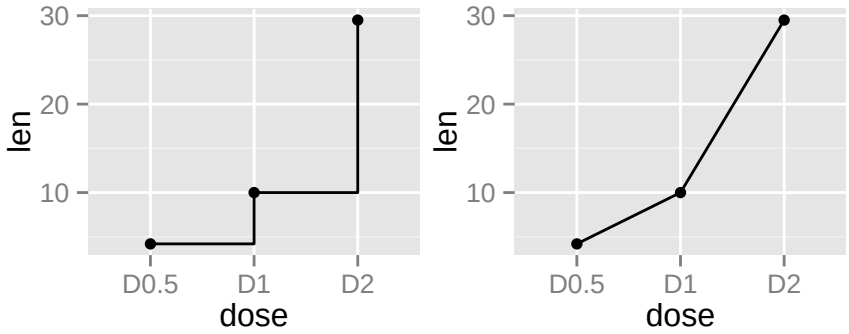
```
library(grid)
# Add an arrow
ggplot(data=df, aes(x=dose, y=len, group=1)) +
  geom_line(arrow = arrow())+
  geom_point()

# Add a closed arrow to the end of the line
myarrow=arrow(angle = 15, ends = "both", type = "closed")
ggplot(data=df, aes(x=dose, y=len, group=1)) +
  geom_line(arrow=myarrow)+
  geom_point()
```



Observations can be also connected using the functions **geom_step()** or **geom_path()** :

```
ggplot(data=df, aes(x=dose, y=len, group=1)) +
  geom_step()+
  geom_point()
```

```
ggplot(data=df, aes(x=dose, y=len, group=1)) +
  geom_path()+
  geom_point()
```



- **geom_line** : Connecting observations, ordered by x value
- **geom_path()** : Observations are connected in original order
- **geom_step** : Connecting observations by stairs

10.2 Line plot with multiple groups

10.2.1 Data

Data derived from *ToothGrowth* data sets are used. ToothGrowth describes the effect of Vitamin C on tooth growth in Guinea pigs. Three dose levels of Vitamin C (0.5, 1, and 2 mg) with each of two delivery methods [orange juice (OJ) or ascorbic acid (VC)] are used :

```
df2 <- data.frame(supp=rep(c("VC", "OJ"), each=3),
                  dose=rep(c("D0.5", "D1", "D2"),2),
                  len=c(6.8, 15, 33, 4.2, 10, 29.5))

head(df2)
```

```
##   supp dose  len
## 1   VC D0.5  6.8
## 2   VC   D1 15.0
## 3   VC   D2 33.0
## 4   OJ D0.5  4.2
## 5   OJ   D1 10.0
## 6   OJ   D2 29.5
```

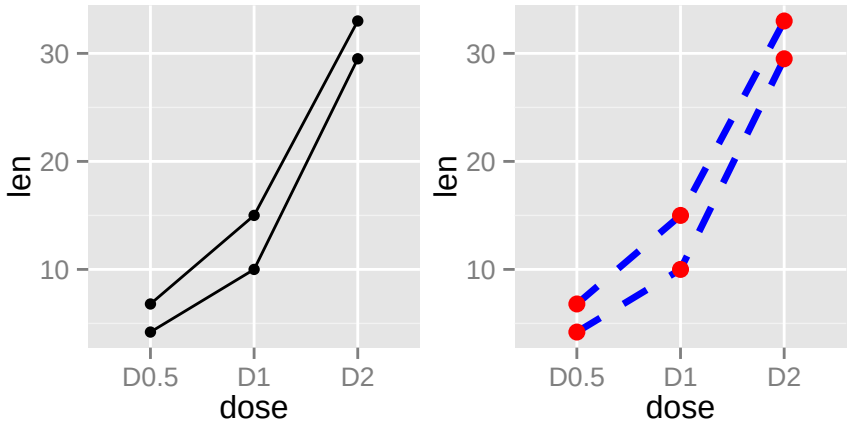
- *len* : Tooth length

- *dose* : Dose in milligrams (0.5, 1, 2)
- *supp* : Supplement type (VC or OJ)

10.2.2 Create line plots

In the graphs below, line types, colors and sizes are the same for the two groups :

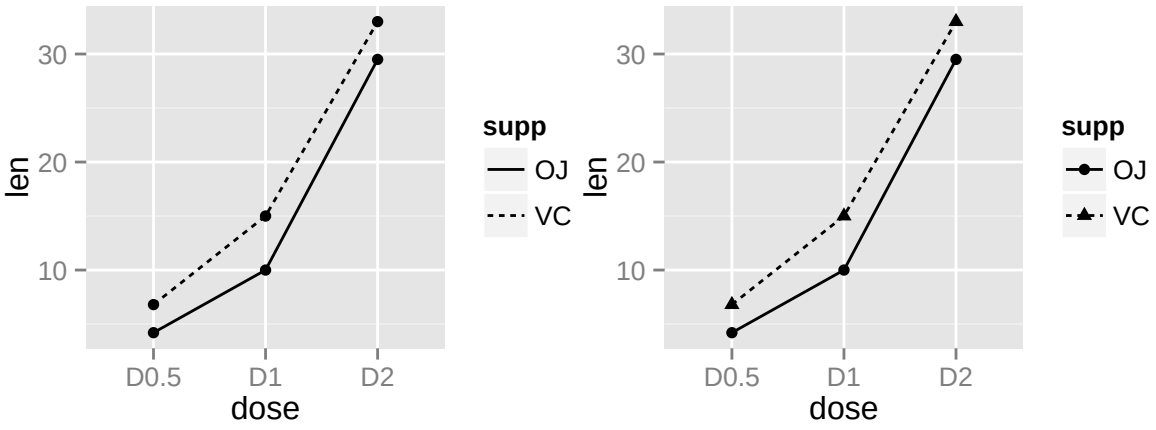
```
# Line plot with multiple groups
ggplot(data=df2, aes(x=dose, y=len, group=supp)) +
  geom_line()+
  geom_point()

# Change line types
ggplot(data=df2, aes(x=dose, y=len, group=supp)) +
  geom_line(linetype="dashed", color="blue", size=1.2)+
  geom_point(color="red", size=3)
```



10.2.3 Change line types by groups

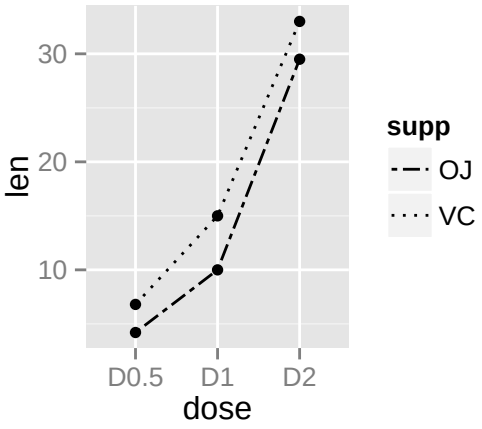
In the graphs below, line types and point shapes are controlled automatically by the levels of the variable *supp* :

```
# Change line types by groups (supp)
ggplot(df2, aes(x=dose, y=len, group=supp)) +
  geom_line(aes(linetype=supp))+
  geom_point()

# Change line types and point shapes
ggplot(df2, aes(x=dose, y=len, group=supp)) +
  geom_line(aes(linetype=supp))+
  geom_point(aes(shape=supp))
```



It is also possible to change manually the line types using the function **scale_linetype_manual()**.

```
# Set line types manually
ggplot(df2, aes(x=dose, y=len, group=supp)) +
  geom_line(aes(linetype=supp))+
  geom_point()+
  scale_linetype_manual(values=c("twodash", "dotted"))
```



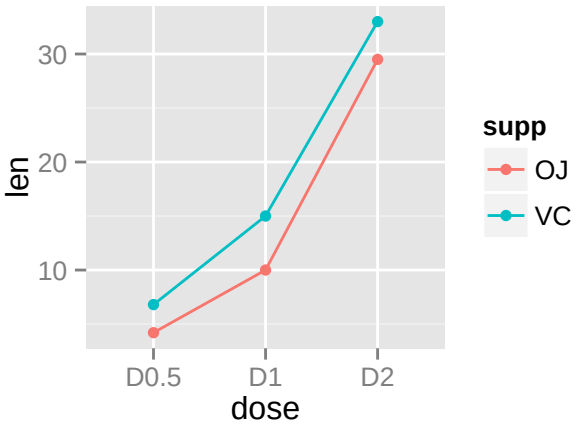
You can read more on line types here : Chapter 20

If you want to change also point shapes, read this article : Chapter 19

10.2.4 Change line colors by groups

Line colors are controlled automatically by the levels of the variable *supp* :

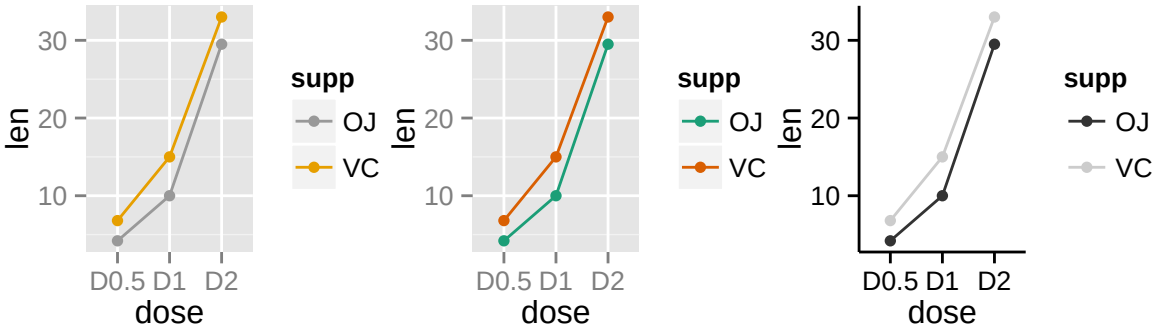
```
p<-ggplot(df2, aes(x=dose, y=len, group=supp)) +
  geom_line(aes(color=supp))+
  geom_point(aes(color=supp))
p
```



It is also possible to *change manually line colors* using the functions :

- *scale_color_manual()* : to use custom colors
- *scale_color_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_color_grey()* : to use grey color palettes

```
# Use custom color palettes
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
p+scale_color_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey() + theme_classic()
```

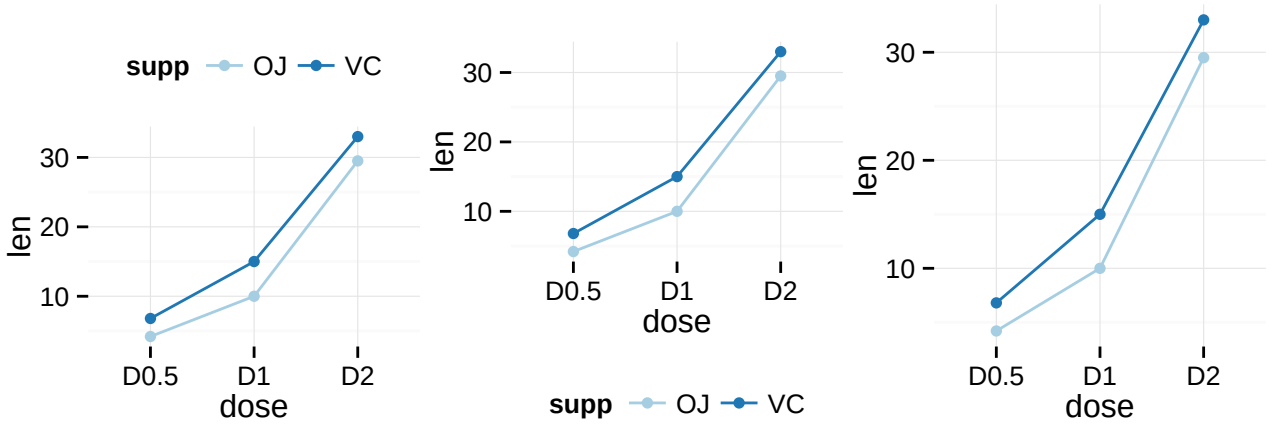


Read more on ggplot2 colors here: Chapter 18

10.3 Change the legend position

```
p <- p + scale_color_brewer(palette="Paired")+
  theme_minimal()
```

```
p + theme(legend.position="top")

p + theme(legend.position="bottom")

# Remove legend
p + theme(legend.position="none")
```



The allowed values for the arguments **legend.position** are : "left","top", "right", "bottom".

Read more on ggplot legend : Chapter 17

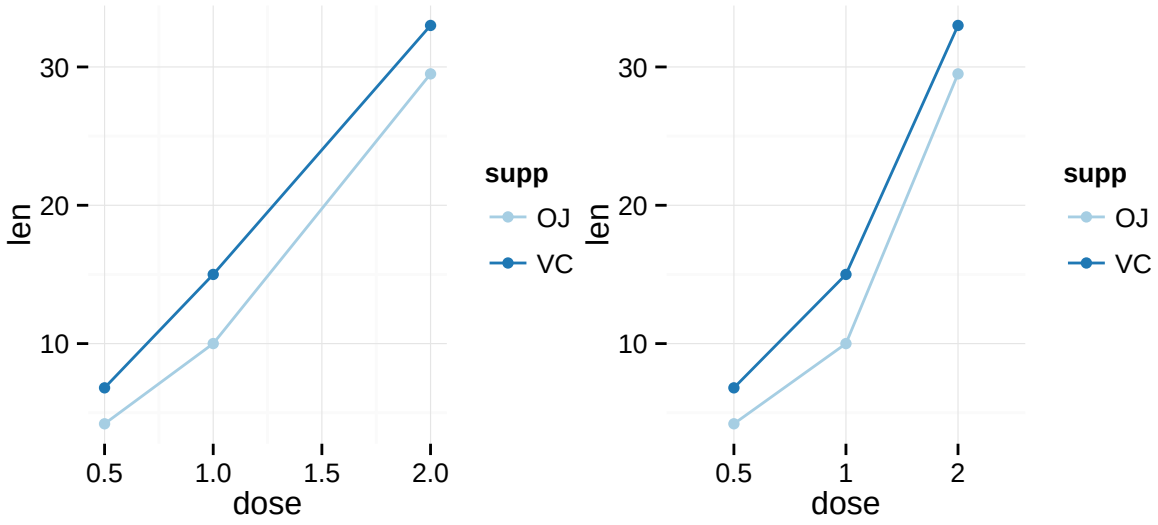
10.4 Line plot with a numeric x-axis

If the variable on x-axis is numeric, it can be useful to treat it as a continuous or a factor variable depending on what you want to do :

```
# Create some data
df2 <- data.frame(supp=rep(c("VC", "OJ"), each=3),
                  dose=rep(c("0.5", "1", "2"),2),
                  len=c(6.8, 15, 33, 4.2, 10, 29.5))
head(df2)
```

```
##   supp dose  len
## 1   VC  0.5  6.8
## 2   VC    1 15.0
## 3   VC    2 33.0
## 4   OJ  0.5  4.2
## 5   OJ    1 10.0
## 6   OJ    2 29.5
```

```
# x axis treated as continuous variable
df2$dose <- as.numeric(as.vector(df2$dose))
ggplot(data=df2, aes(x=dose, y=len, group=supp, color=supp)) +
  geom_line() + geom_point()+
  scale_color_brewer(palette="Paired")+
  theme_minimal()

# Axis treated as discrete variable
df2$dose<-as.factor(df2$dose)
ggplot(data=df2, aes(x=dose, y=len, group=supp, color=supp)) +
  geom_line() + geom_point()+
  scale_color_brewer(palette="Paired")+
  theme_minimal()
```



10.5 Line plot with dates on x-axis

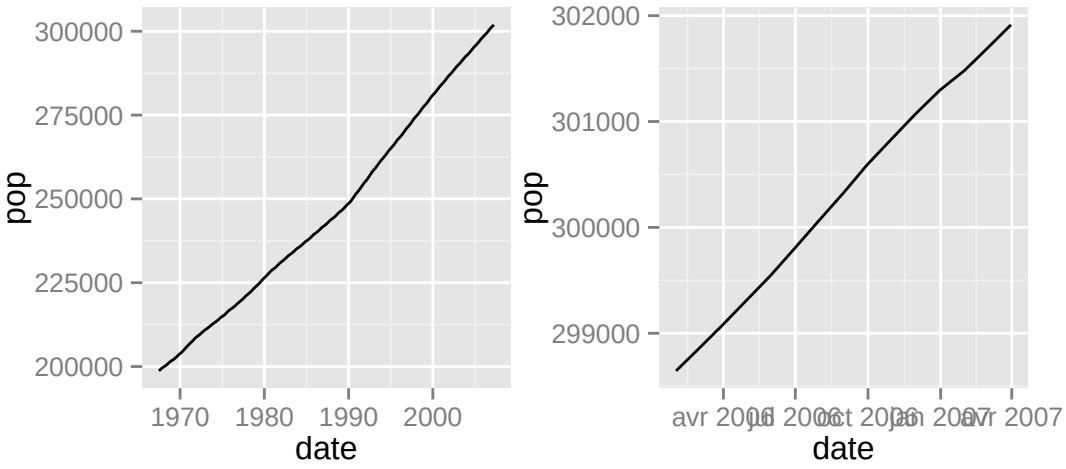
economics time series data sets are used :

```
head(economics)
```

```
##         date   pce    pop psavert uempmed unemploy
## 1 1967-06-30 507.8 198712     9.8     4.5     2944
## 2 1967-07-31 510.9 198911     9.8     4.7     2945
## 3 1967-08-31 516.7 199113     9.0     4.6     2958
## 4 1967-09-30 513.3 199311     9.8     4.9     3143
## 5 1967-10-31 518.5 199498     9.7     4.7     3066
## 6 1967-11-30 526.2 199657     9.4     4.8     3018
```

Plots :

```
# Basic line plot
ggplot(data=economics, aes(x=date, y=pop))+
  geom_line()

# Plot a subset of the data
ggplot(data=subset(economics, date > as.Date("2006-1-1")),
       aes(x=date, y=pop))+geom_line()
```



Change line size :

```
# Change line size
ggplot(data=economics, aes(x=date, y=pop, size=unemploy/pop))+
  geom_line()
```

10.6 Line graph with error bars

The function below will be used to calculate the mean and the standard deviation, for the variable of interest, in each group :

```
#++++++++++++++++++++++++
# Function to calculate the mean and the standard deviation
  # for each group
#++++++++++++++++++++++++
# data : a data frame
# varname : the name of a column containing the variable
  #to be summariezed
# groupnames : vector of column names to be used as
  # grouping variables
data_summary <- function(data, varname, groupnames){
  require(plyr)
  summary_func <- function(x, col){
    c(mean = mean(x[[col]], na.rm=TRUE),
      sd = sd(x[[col]], na.rm=TRUE))
  }
  data_sum<-ddply(data, groupnames, .fun=summary_func,
                  varname)
  data_sum <- rename(data_sum, c("mean" = varname))
 return(data_sum)
}
```

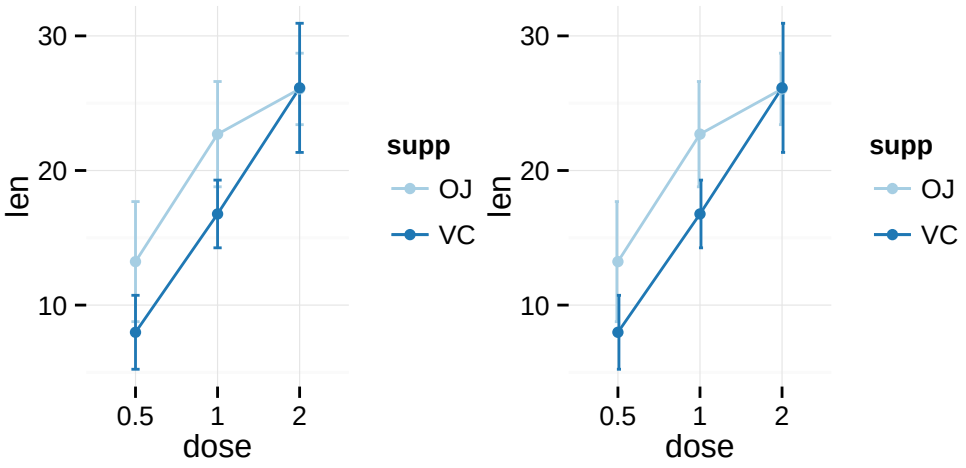
Summarize the data :

```
df3 <- data_summary(ToothGrowth, varname="len",
                    groupnames=c("supp", "dose"))
head(df3)
```

```
##   supp dose   len       sd
## 1   OJ  0.5 13.23 4.459709
## 2   OJ    1 22.70 3.910953
## 3   OJ    2 26.06 2.655058
## 4   VC  0.5  7.98 2.746634
## 5   VC    1 16.77 2.515309
## 6   VC    2 26.14 4.797731
```

The function **geom_errorbar()** can be used to produce a line graph with error bars :

```
# Standard deviation of the mean
ggplot(df3, aes(x=dose, y=len, group=supp, color=supp)) +
    geom_errorbar(aes(ymin=len-sd, ymax=len+sd), width=.1) +
    geom_line() + geom_point()+
   scale_color_brewer(palette="Paired")+theme_minimal()

# Use position_dodge to move overlapped errorbars horizontally
ggplot(df3, aes(x=dose, y=len, group=supp, color=supp)) +
    geom_errorbar(aes(ymin=len-sd, ymax=len+sd), width=.1,
    position=position_dodge(0.05)) +
    geom_line() + geom_point()+
   scale_color_brewer(palette="Paired")+theme_minimal()
```

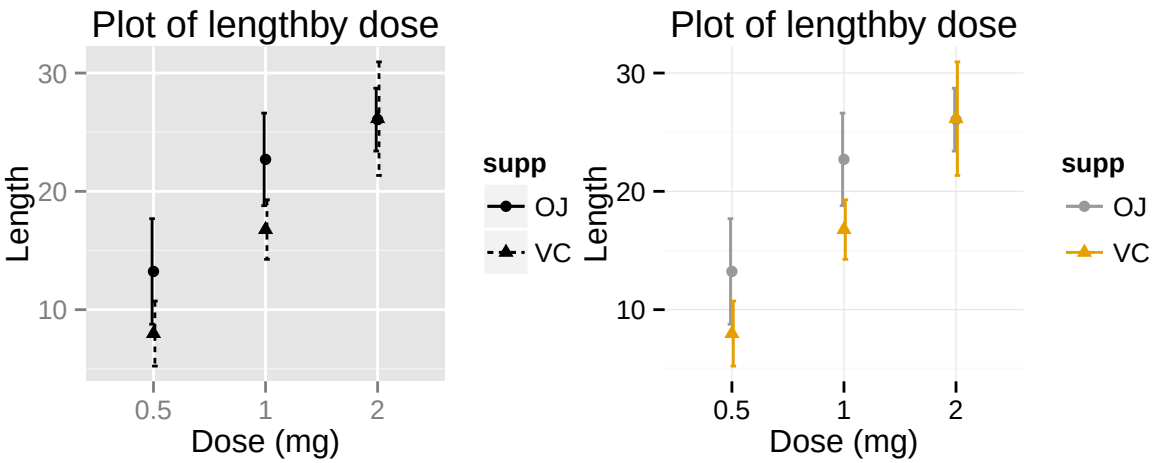


Read more on error bars: Chapter 11.

10.7 Customized line graphs

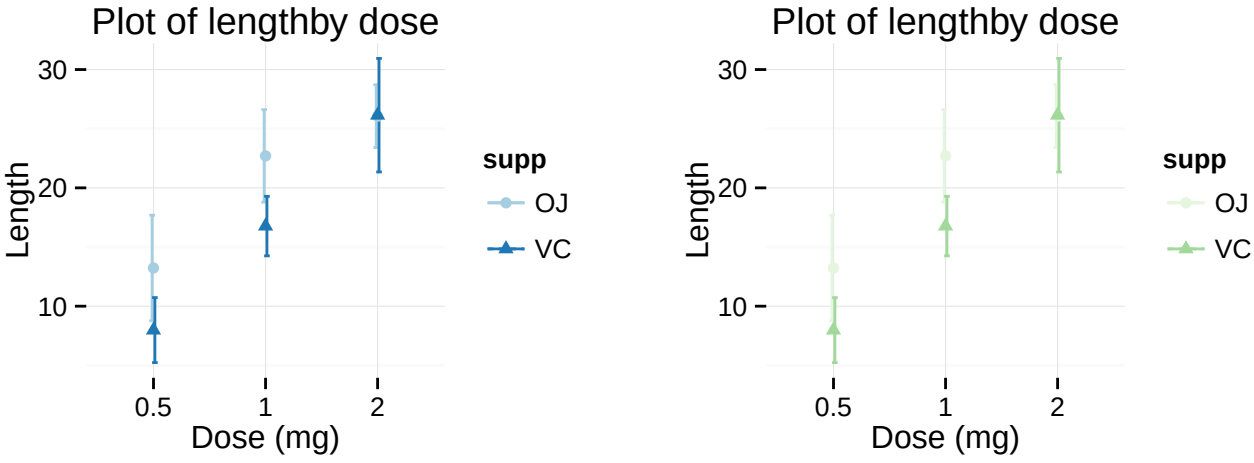
```
# Simple line plot
# Change point shapes and line types by groups
ggplot(df3, aes(x=dose, y=len, shape=supp, linetype=supp))+
    geom_errorbar(aes(ymin=len-sd, ymax=len+sd), width=.1,
    position=position_dodge(0.05)) +
    geom_line() +
    geom_point()+
    labs(title="Plot of lengthby dose",x="Dose (mg)", y = "Length")


# Change color by groups
# Add error bars
```

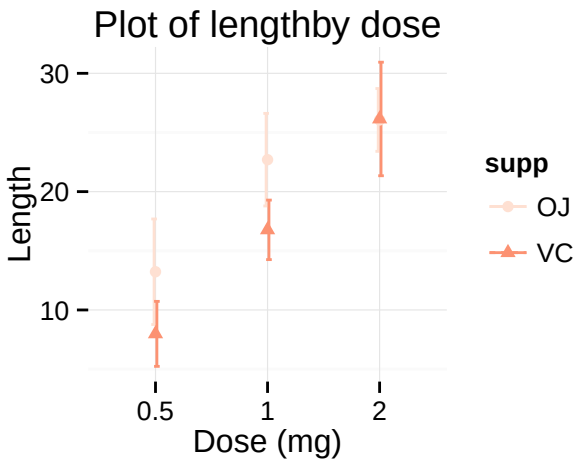
```
p <- ggplot(df3, aes(x=dose, y=len,  color=supp))+
    geom_errorbar(aes(ymin=len-sd, ymax=len+sd), width=.1,
    position=position_dodge(0.05)) +
    geom_line(aes(linetype=supp)) +
    geom_point(aes(shape=supp))+
    labs(title="Plot of lengthby dose",x="Dose (mg)", y = "Length")

p + theme_minimal() + scale_color_manual(values=c('#999999','#E69F00'))
```



Change colors manually :

```
p + scale_color_brewer(palette="Paired") + theme_minimal()

# Greens
p + scale_color_brewer(palette="Greens") + theme_minimal()

# Reds
p + scale_color_brewer(palette="Reds") + theme_minimal()
```




Plot of lengthby dose
Length
30
20
10
0.5
1
2
Dose (mg)
supp
OJ
VC

Chapter 11

Error bars

There are different types of *error bars* which can be created using the functions below :

- geom_errorbar()
- geom_linerange()
- geom_pointrange()
- geom_crossbar()
- geom_errorbarh()

11.1 Add error bars to a bar and line plots

11.1.1 Prepare the data

ToothGrowth data is used. It describes the effect of Vitamin C on tooth growth in Guinea pigs. Three dose levels of Vitamin C (0.5, 1, and 2 mg) with each of two delivery methods [orange juice (OJ) or ascorbic acid (VC)] are used :

```
library(ggplot2)
df <- ToothGrowth
df$dose <- as.factor(df$dose)
head(df)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

- *len* : Tooth length
- *dose* : Dose in milligrams (0.5, 1, 2)
- *supp* : Supplement type (VC or OJ)

In the example below, we'll plot the mean value of Tooth length in each group. The standard deviation is used to draw the error bars on the graph.

First, the helper function below will be used to calculate the mean and the standard deviation, for the variable of interest, in each group :

```
#++++++++++++++++++++++++
# Function to calculate the mean and the standard deviation
  # for each group
#++++++++++++++++++++++++
# data : a data frame
# varname : the name of a column containing the variable
  #to be summariezed
# groupnames : vector of column names to be used as
  # grouping variables
data_summary <- function(data, varname, groupnames){
  require(plyr)
  summary_func <- function(x, col){
    c(mean = mean(x[[col]], na.rm=TRUE),
      sd = sd(x[[col]], na.rm=TRUE))
  }
  data_sum<-ddply(data, groupnames, .fun=summary_func,
                  varname)
  data_sum <- rename(data_sum, c("mean" = varname))
 return(data_sum)
}
```

Summarize the data :

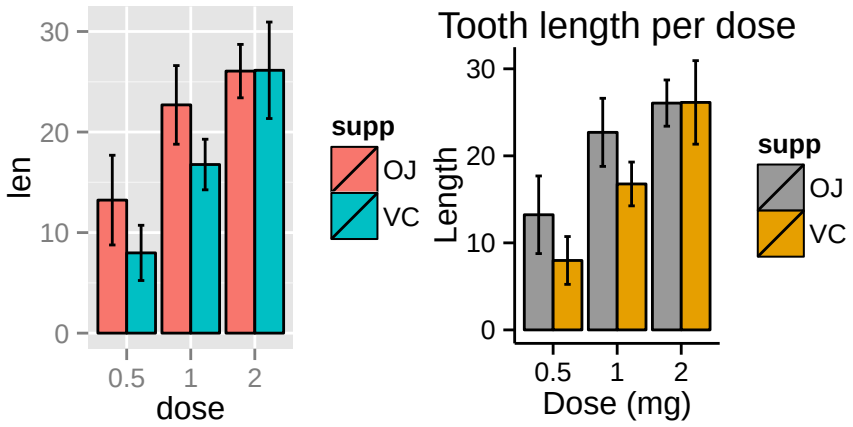
```
df2 <- data_summary(ToothGrowth, varname="len",
                    groupnames=c("supp", "dose"))
# Convert dose to a factor variable
df2$dose=as.factor(df2$dose)
head(df2)
```

```
##   supp dose   len       sd
## 1   OJ  0.5 13.23 4.459709
## 2   OJ    1 22.70 3.910953
## 3   OJ    2 26.06 2.655058
## 4   VC  0.5  7.98 2.746634
## 5   VC    1 16.77 2.515309
## 6   VC    2 26.14 4.797731
```

11.1.2 Barplot with error bars

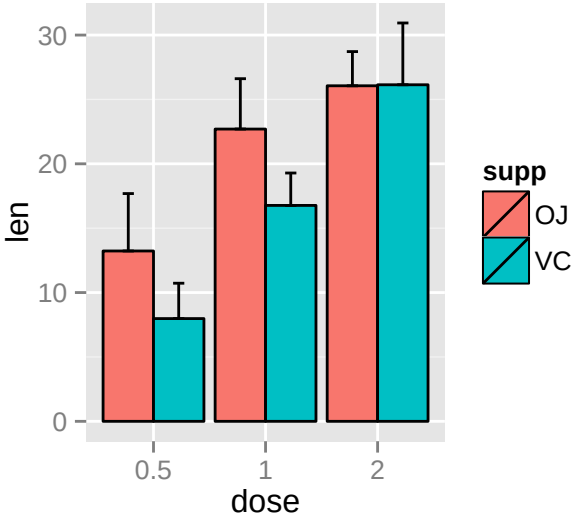
The function **geom_errorbar()** can be used to produce the error bars :

```
library(ggplot2)
# Default bar plot
p<- ggplot(df2, aes(x=dose, y=len, fill=supp)) +
  geom_bar(stat="identity", color="black",
           position=position_dodge()) +
  geom_errorbar(aes(ymin=len-sd, ymax=len+sd), width=.2,
                position=position_dodge(.9))
print(p)

# Finished bar plot
p+labs(title="Tooth length per dose", x="Dose (mg)", y = "Length")+
   theme_classic() +
   scale_fill_manual(values=c('#999999','#E69F00'))
```



Note that, you can chose to keep only the upper error bars

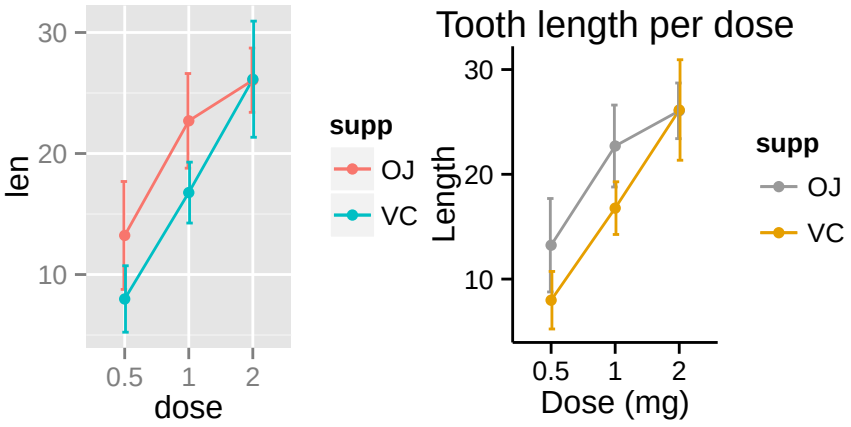
```
# Keep only upper error bars
 ggplot(df2, aes(x=dose, y=len, fill=supp)) +
  geom_bar(stat="identity", color="black", position=position_dodge()) +
  geom_errorbar(aes(ymin=len, ymax=len+sd), width=.2,
                position=position_dodge(.9))
```



Read more on ggplot2 bar graphs : Chapter 9

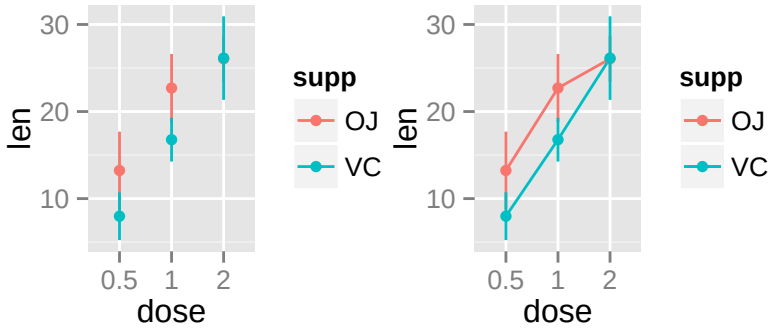
11.1.3 Line plot with error bars

```
# Default line plot
p<- ggplot(df2, aes(x=dose, y=len, group=supp, color=supp)) +
  geom_line() +
  geom_point()+
  geom_errorbar(aes(ymin=len-sd, ymax=len+sd), width=.2,
                position=position_dodge(0.05))
print(p)

# Finished line plot
p+labs(title="Tooth length per dose", x="Dose (mg)", y = "Length")+
   theme_classic() +
   scale_color_manual(values=c('#999999','#E69F00'))
```



You can also use the functions **geom_pointrange()** or **geom_linerange()** instead of using **geom_errorbar()**

```
# Use geom_pointrange
ggplot(df2, aes(x=dose, y=len, group=supp, color=supp)) +
geom_pointrange(aes(ymin=len-sd, ymax=len+sd))

# Use geom_line()+geom_pointrange()
ggplot(df2, aes(x=dose, y=len, group=supp, color=supp)) +
  geom_line()+
  geom_pointrange(aes(ymin=len-sd, ymax=len+sd))
```



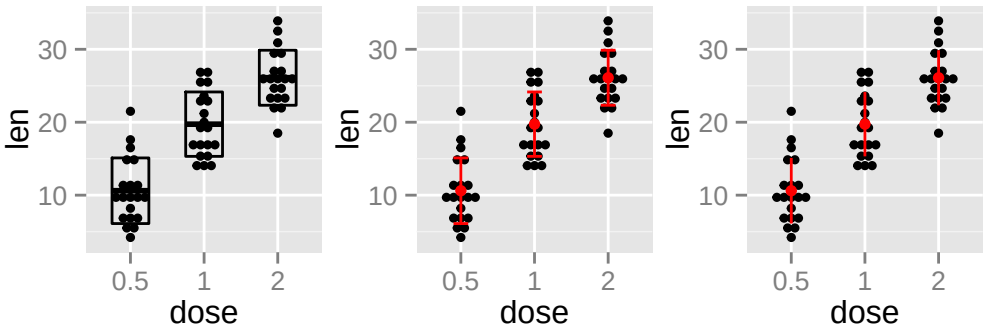
Read more on ggplot2 line plots : Chapter 10

11.2 Dot plot with mean point and error bars

The functions **geom_dotplot()** and **stat_summary()** are used :

The mean +/- SD can be added as a *crossbar* , a **error bar** or a *pointrange* :

```
p <- ggplot(df, aes(x=dose, y=len)) +
    geom_dotplot(binaxis='y', stackdir='center')

# use geom_crossbar()
p + stat_summary(fun.data="mean_sdl", mult=1,
                 geom="crossbar", width=0.5)

# Use geom_errorbar()
p + stat_summary(fun.data=mean_sdl, mult=1,
        geom="errorbar", color="red", width=0.2) +
  stat_summary(fun.y=mean, geom="point", color="red")

# Use geom_pointrange()
p + stat_summary(fun.data=mean_sdl, mult=1,
                 geom="pointrange", color="red")
```



Read more on ggplot2 dot plots : Chapter 4

Chapter 12

Pie charts

The function **coord_polar()** is used to produce a **pie chart**, which is just a stacked bar chart in polar coordinates.

12.1 Simple pie charts

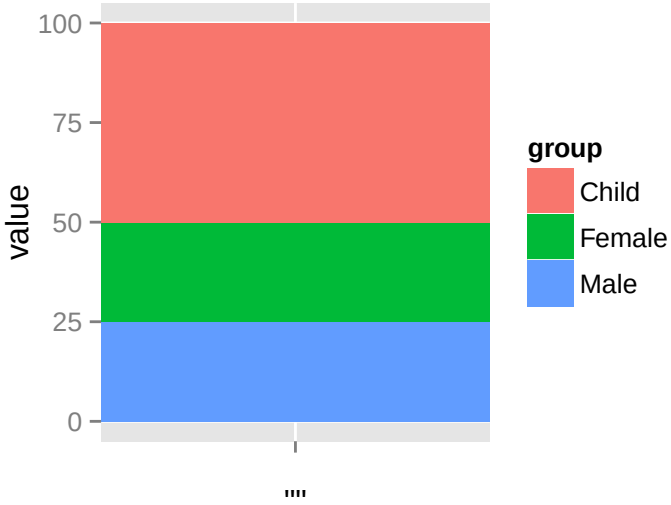
Create some data :

```
df <- data.frame(
  group = c("Male", "Female", "Child"),
  value = c(25, 25, 50)
  )
head(df)
```

```
##    group value
## 1   Male    25
## 2 Female    25
## 3  Child    50
```

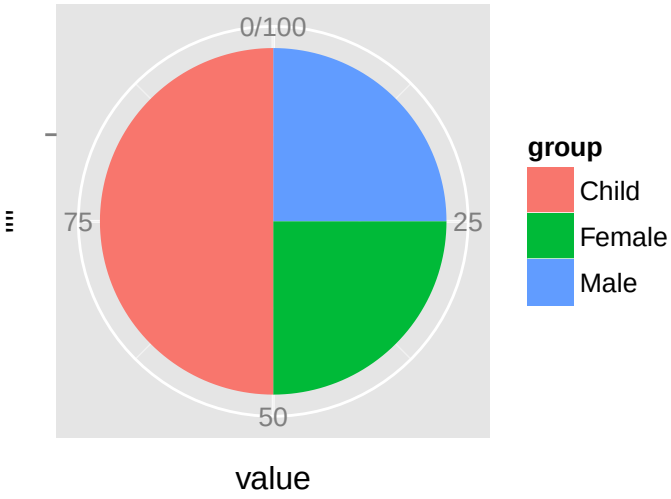
Use a bar plot to visualize the data :

```
# Barplot
bp<- ggplot(df, aes(x="", y=value, fill=group))+
geom_bar(width = 1, stat = "identity")
bp
```



Create a pie chart :

```
pie <- bp + coord_polar("y", start=0)
pie
```

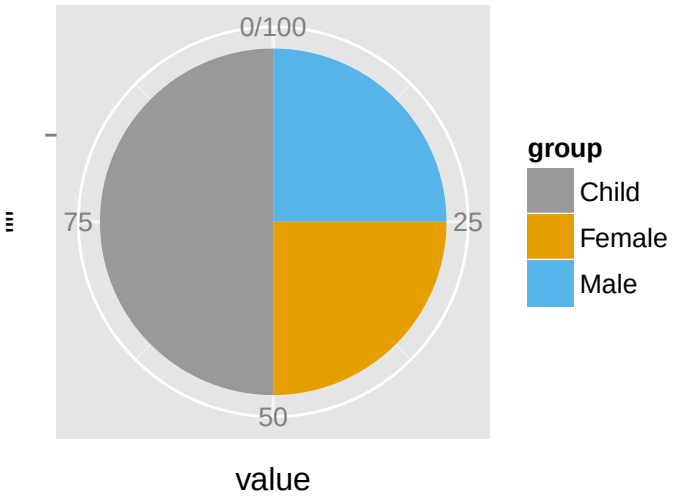


12.2 Change the pie chart fill colors

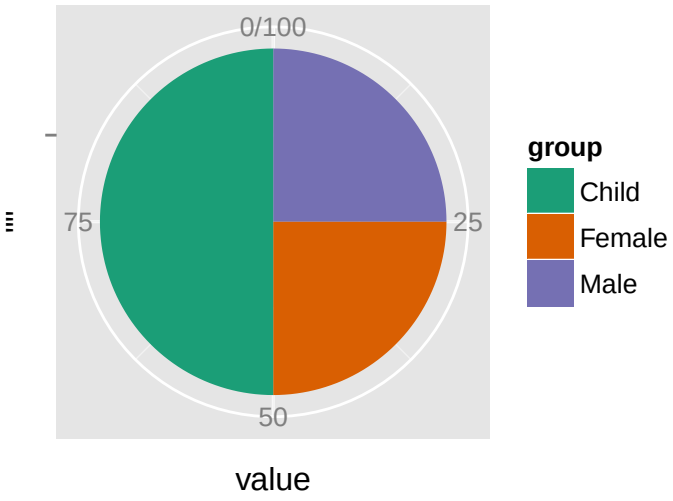
It is possible to change manually the **pie chart fill colors** using the functions :

- *scale_fill_manual()* : to use custom colors
- *scale_fill_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_fill_grey()* : to use grey color palettes

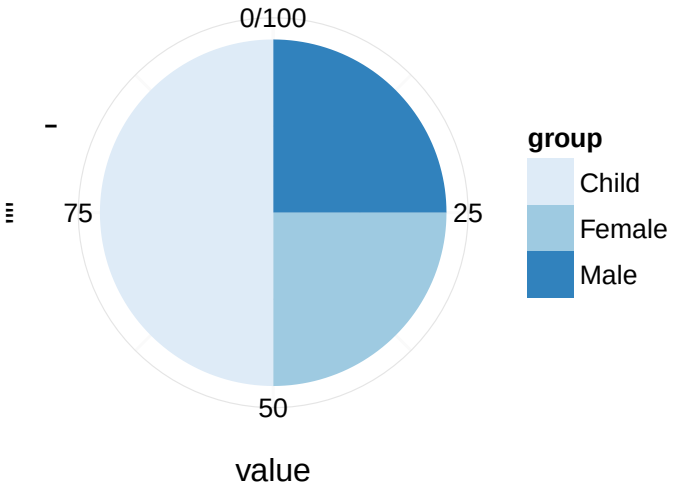
```
# Use custom color palettes
pie + scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))
```



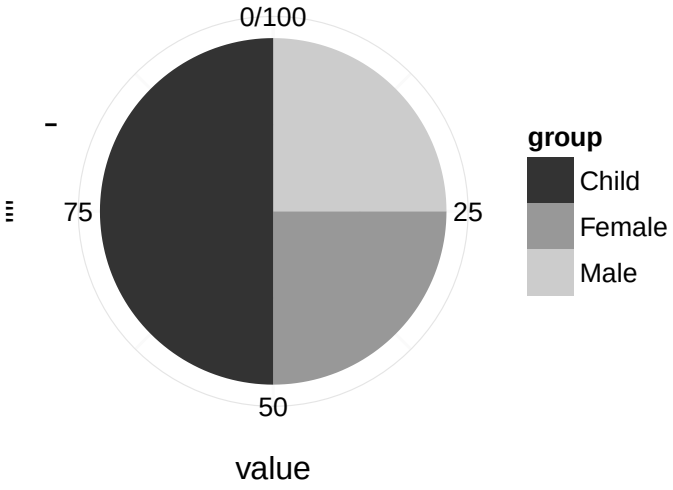

```
# use brewer color palettes
pie + scale_fill_brewer(palette="Dark2")
```




```
pie + scale_fill_brewer(palette="Blues")+
  theme_minimal()
```




```
# Use grey scale
pie + scale_fill_grey() + theme_minimal()
```



Read more on ggplot2 colors here: Chapter 18

12.3 Create a pie chart from a factor variable

PlantGrowth data is used :

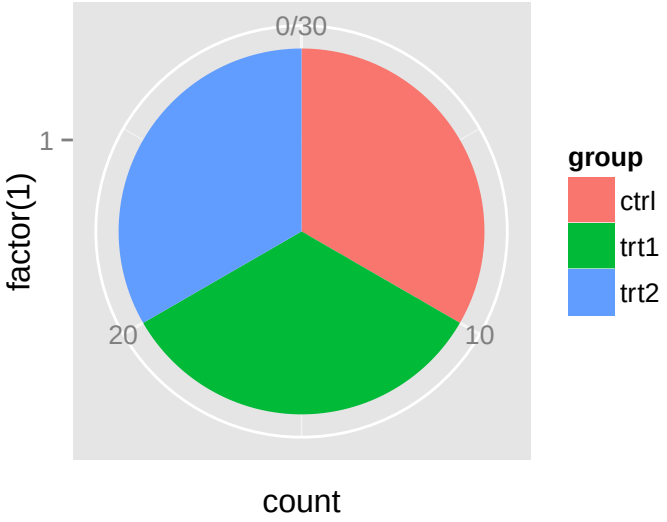
```
head(PlantGrowth)
```

```
##   weight group
## 1   4.17  ctrl
## 2   5.58  ctrl
## 3   5.18  ctrl
## 4   6.11  ctrl
```

```
## 5   4.50  ctrl
## 6   4.61  ctrl
```

Create the pie chart of the count of observations in each group :

```
ggplot(PlantGrowth, aes(x=factor(1), fill=group))+
  geom_bar(width = 1)+
  coord_polar("y")
```



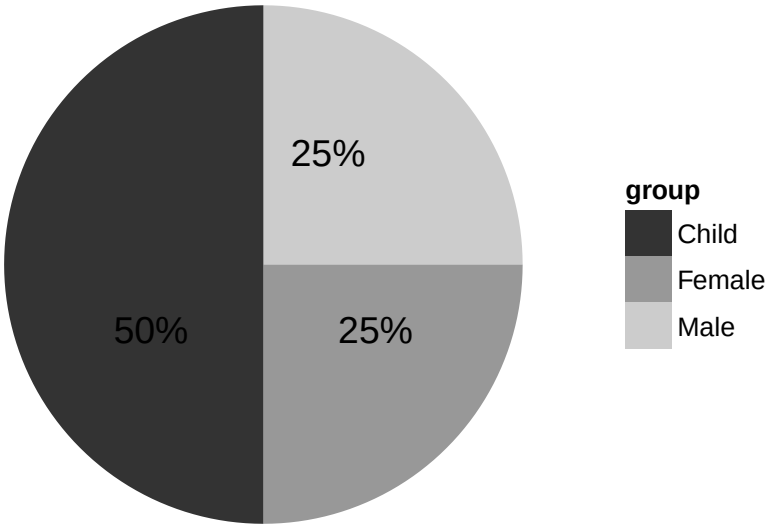
12.4 Customized pie charts

Create a blank theme :

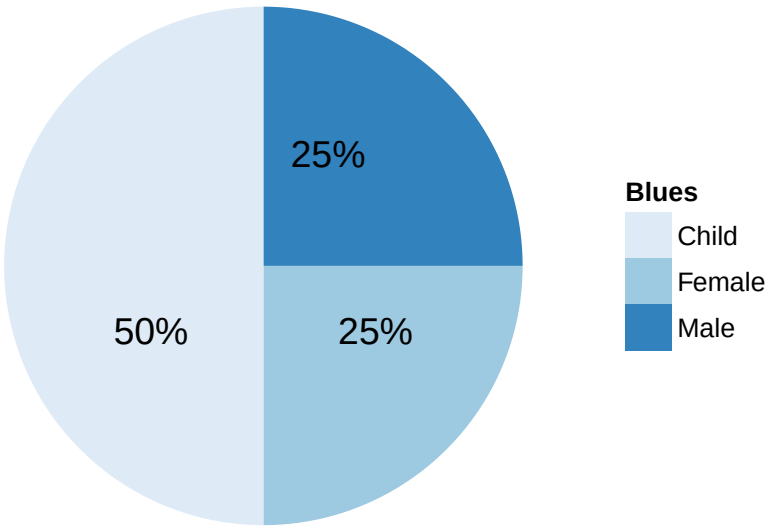
```
blank_theme <- theme_minimal()+
  theme(
  axis.title.x = element_blank(),
  axis.title.y = element_blank(),
  panel.border = element_blank(),
  panel.grid=element_blank(),
  axis.ticks = element_blank(),
  plot.title=element_text(size=14, face="bold")
  )
```

1. Apply the blank theme
2. Remove axis tick mark labels
3. Add text annotations : The package **scales** is used to format the labels in *percent*

```
# Apply blank theme
library(scales)
pie + scale_fill_grey() +  blank_theme +
  theme(axis.text.x=element_blank()) +
  geom_text(aes(y = value/3 + c(0, cumsum(value)[-length(value)]),
            label = percent(value/100)), size=5)
```



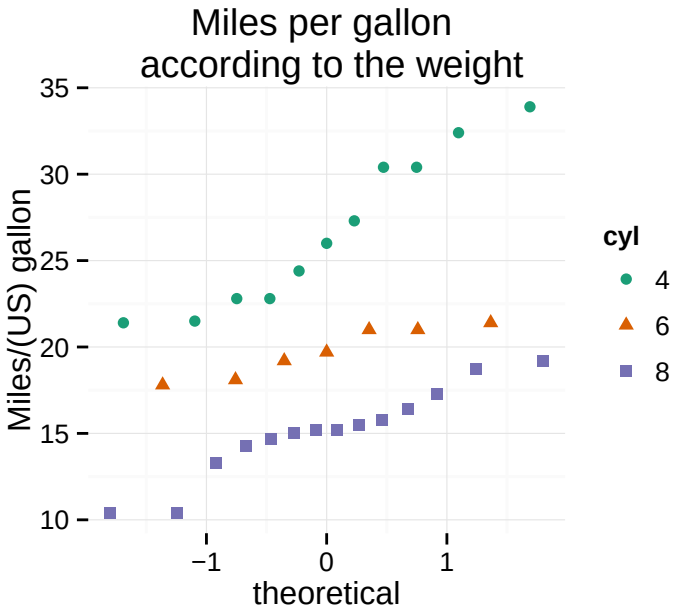

```
# Use brewer palette
pie + scale_fill_brewer("Blues") + blank_theme +
  theme(axis.text.x=element_blank())+
  geom_text(aes(y = value/3 + c(0, cumsum(value)[-length(value)]),
                label = percent(value/100)), size=5)
```



Chapter 13

QQ Plots

QQ-plots (or Quantile - Quantile plots) are used to check whether a given data follows *normal distribution*. The function **stat_qq()** or **qplot()** can be used to create qq-plots.



13.1 Prepare the data

mtcars data sets are used in the examples below.

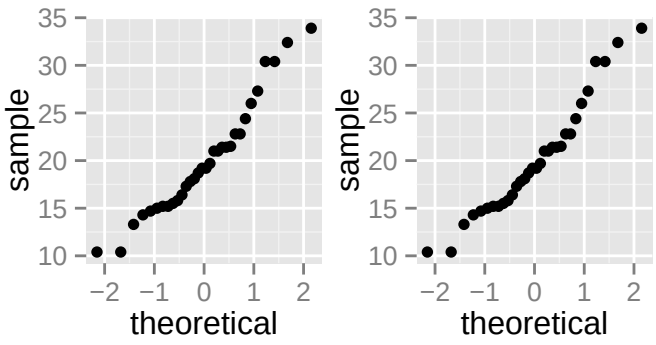
```
# Convert cyl column from a numeric to a factor variable
mtcars$cyl <- as.factor(mtcars$cyl)
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

13.2 Basic qq plots

In the example below, the distribution of the variable *mpg* is explored :

```
library(ggplot2)

# Solution 1
qplot(sample = mpg, data = mtcars)

# Solution 2
ggplot(mtcars, aes(sample=mpg))+stat_qq()
```

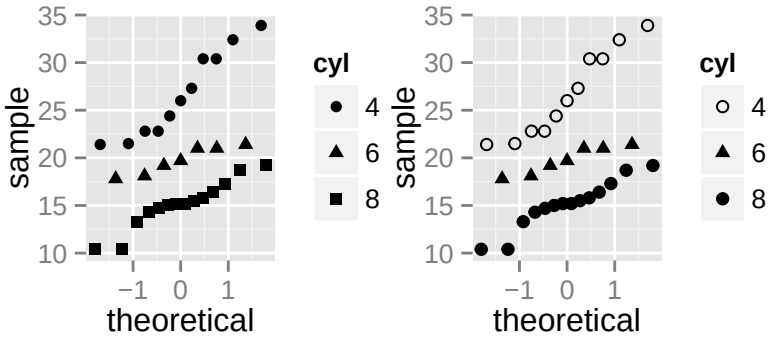


13.3 Change qq plot point shapes by groups

In the R code below, point shapes are controlled automatically by the variable *cyl*.

You can also set point shapes manually using the function **scale_shape_manual()**

```
# Change point shapes by groups
p<-qplot(sample = mpg, data = mtcars, shape=cyl)
p

# Change point shapes manually
p + scale_shape_manual(values=c(1,17,19))
```

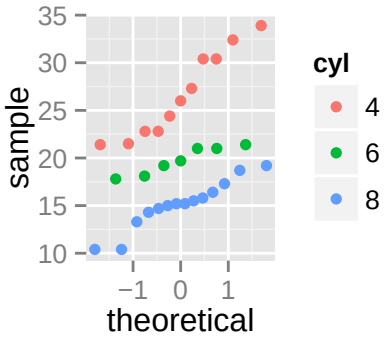


Read more on point shapes: Chapter 19

13.4 Change qq plot colors by groups

In the R code below, point colors of the qq plot are automatically controlled by the levels of *cyl* :

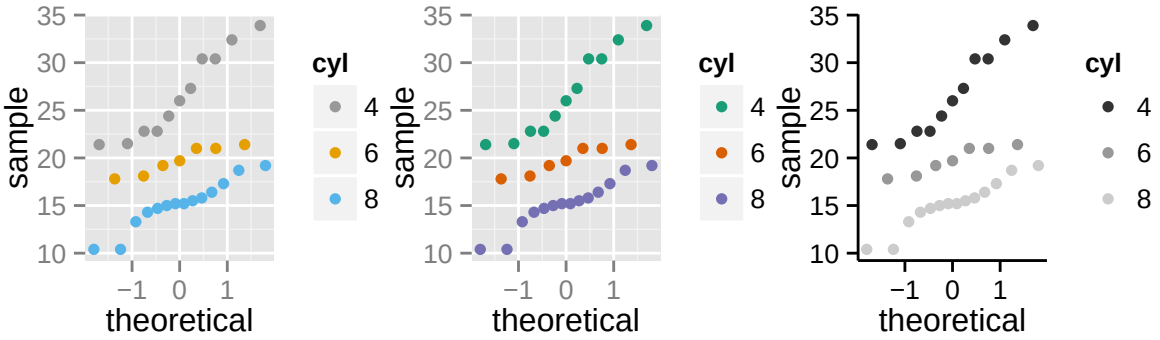
```
# Change qq plot colors by groups
p<-qplot(sample = mpg, data = mtcars, color=cyl)
p
```



It is also possible to *change manually qq plot colors* using the functions :

- *scale_color_manual()* : to use custom colors
- *scale_color_brewer()* : to use color palettes from *RColorBrewer* package
- *scale_color_grey()* : to use grey color palettes

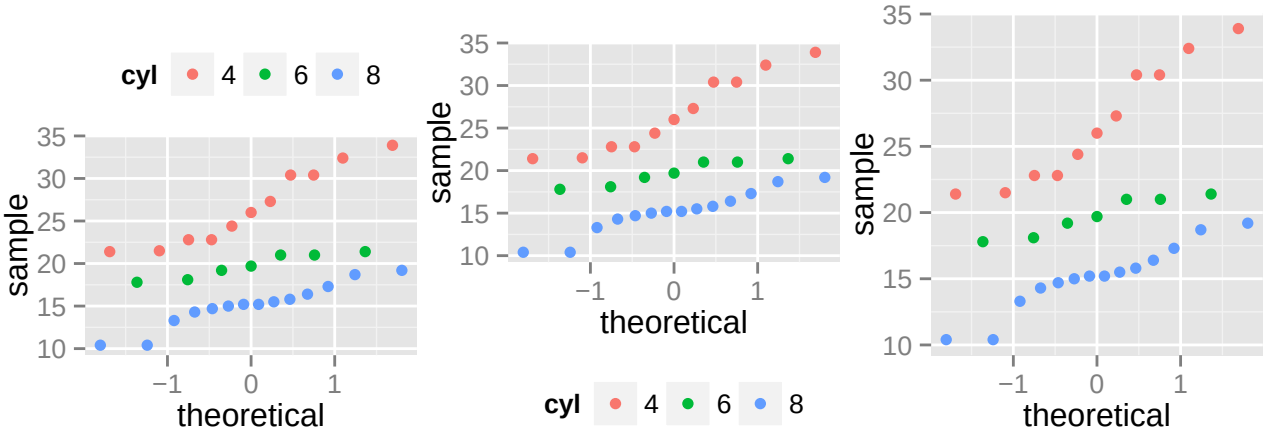
```
# Use custom color palettes
p+scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
p+scale_color_brewer(palette="Dark2")

# Use grey scale
p + scale_color_grey() + theme_classic()
```



Read more on ggplot2 colors here: Chapter 18

13.5 Change the legend position

```
p + theme(legend.position="top")

p + theme(legend.position="bottom")

p + theme(legend.position="none") # Remove legend
```

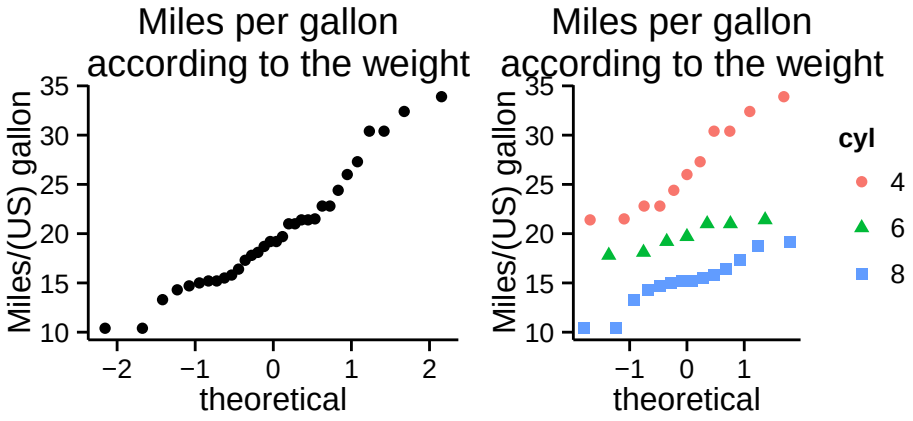


The allowed values for the arguments **legend.position** are : "left","top", "right", "bottom".

Read more on ggplot legends : Chapter 17

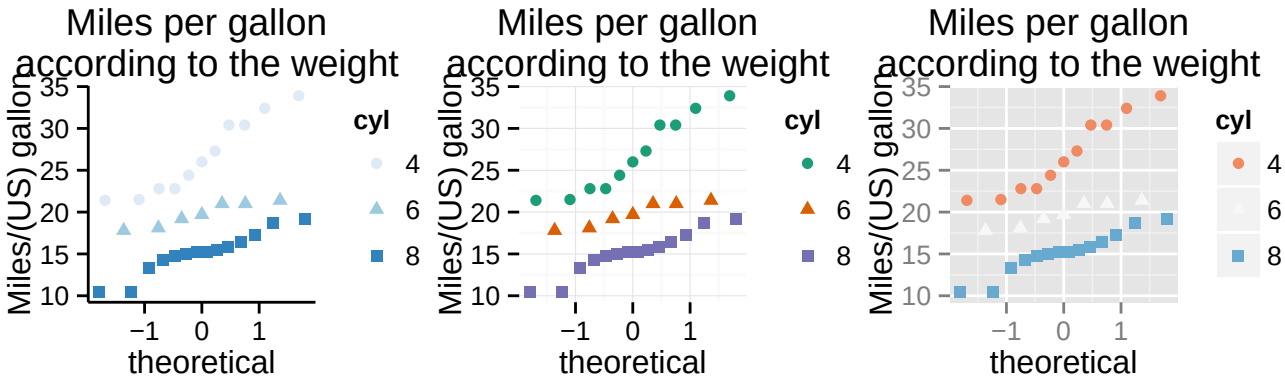
13.6 Customized qq plots

```
# Basic qq plot
qplot(sample = mpg, data = mtcars)+
labs(title="Miles per gallon \n according to the weight",
       y = "Miles/(US) gallon")+
theme_classic()

# Change color/shape by groups
p <- qplot(sample = mpg, data = mtcars, color=cyl, shape=cyl)+
labs(title="Miles per gallon \n according to the weight",
       y = "Miles/(US) gallon")

p + theme_classic()
```



Change colors manually :

```
# Continuous colors
p + scale_color_brewer(palette="Blues") + theme_classic()

# Discrete colors
p + scale_color_brewer(palette="Dark2") + theme_minimal()

# Gradient colors
p + scale_color_brewer(palette="RdBu")
```



Read more on ggplot2 colors here : Chapter 18

Chapter 14

ECDF plots

ECDF (Empirical Cumulative Density Function) reports for any given number the percent of individuals that are below that threshold.

The function **stat_ecdf()** or **qplot()** can be used.

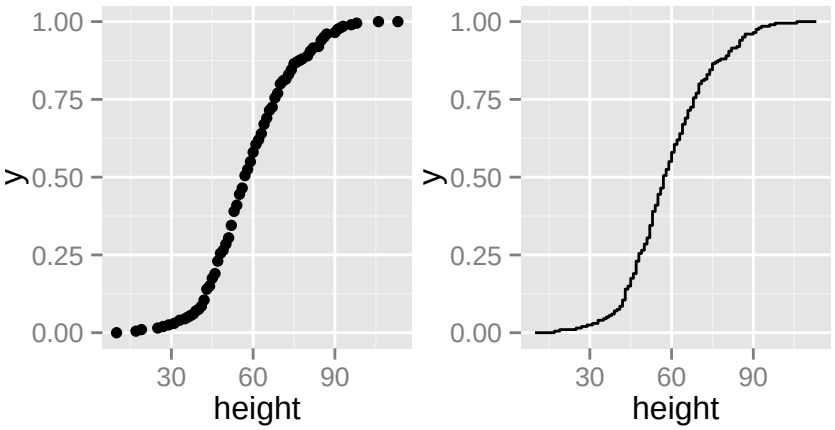
14.1 Create some data

```
set.seed(1234)
height <- round(rnorm(200, mean=60, sd=15))
head(height)
```

```
## [1] 42 64 76 25 66 68
```

14.2 ECDF plots

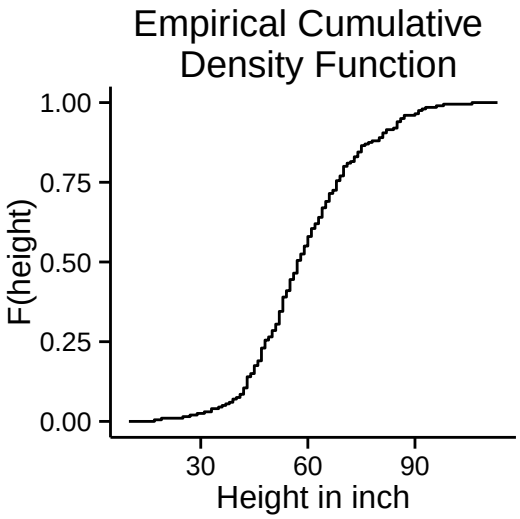
```
library(ggplot2)

qplot(height, stat = "ecdf", geom="point")

qplot(height, stat = "ecdf", geom="step")
```



For any value, say, height = 50, you can see that about 25% of our individuals are shorter than 50 inches

14.3 Customized ECDF plots

```
qplot(height, stat = "ecdf", geom="step")+
labs(title="Empirical Cumulative \n Density Function",
     y = "F(height)", x="Height in inch")+
theme_classic()
```



Chapter 15

ggsave(): Save ggplots

To print directly a ggplot to a file, the function **print()** is used:

```
# Print the plot to a pdf file
pdf("myplot.pdf")
myplot <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
print(myplot)
dev.off()
```

For printing to a **png** file, use:

```
png("myplot.png")
myplot <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
print(myplot)
dev.off()
```

It's also possible to make a ggplot and to save it from the screen using the function **ggsave()**:

```
# 1. Create a plot
# The plot is displayed on the screen
ggplot(mtcars, aes(wt, mpg)) + geom_point()

# 2. Save the plot to a pdf
ggsave("myplot.pdf")
```

For saving to a **png** file, use:

```
ggsave("myplot.png")
```

Part II

Graphical parameters

Chapter 16

Main title, axis labels and legend titles

This chapter describes how to modify **plot titles** (**main title**, **axis labels** and **legend titles**) using **ggplot2** package.

The functions below can be used :

```
ggtitle(label) # for the main title
xlab(label) # for the x axis label
ylab(label) # for the y axis label
labs(...) # for the main title, axis labels and legend titles
```

The argument *label* is the text to be used for the main title or for the axis labels.

16.1 Data

ToothGrowth data is used in the following examples.

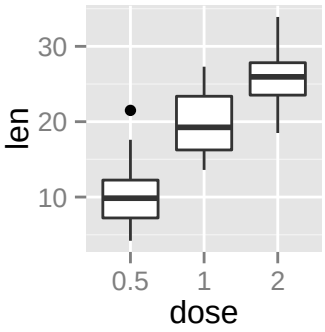
```
# convert dose column from a numeric to a factor variable
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that the variable *dose* is converted as a factor using the above R script.

16.2 Example of plot

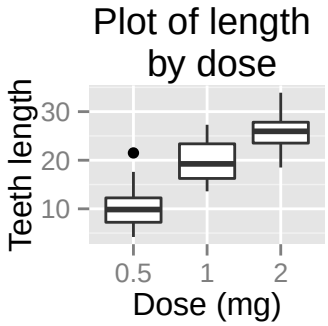
```
library(ggplot2)
p <- ggplot(ToothGrowth, aes(x = dose, y = len)) + geom_boxplot()
p
```



16.3 Change the main title and axis labels

Change **plot titles** by using the functions *ggtitle()*, *xlab()* and *ylab()* :

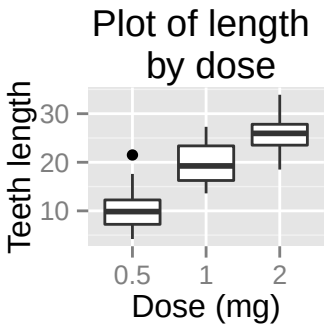
```
p + ggtitle("Plot of length \n by dose") +
  xlab("Dose (mg)") + ylab("Teeth length")
```



Note that, you can use **\n** to split long title into multiple lines.

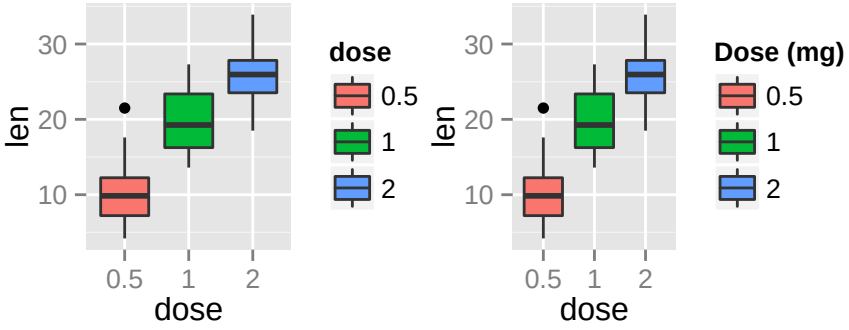
Change **plot titles** using the function **labs()** as follow :

```
p +labs(title = "Plot of length \n by dose",
        x = "Dose (mg)", y = "Teeth length")
```



It is also possible to change **legend titles** using the function **labs()**:

```
# Default plot
p <- ggplot(ToothGrowth, aes(x = dose, y = len, fill = dose))+
  geom_boxplot()
p

# Modify legend titles
p + labs(fill = "Dose (mg)")
```



16.4 Change the appearance of the main title and axis labels

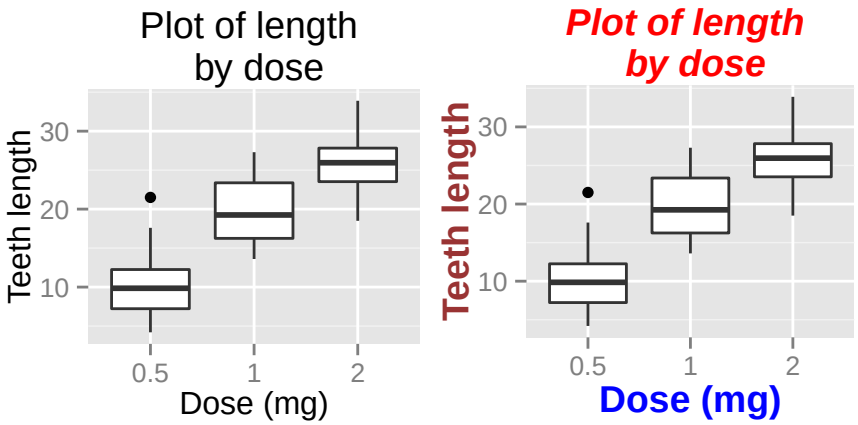
Main title and, x and y axis labels can be customized using the functions **theme()** and **element_text()** as follow :

```
# main title
p + theme(plot.title = element_text(family, face, colour, size))

# x axis title
p + theme(axis.title.x = element_text(family, face, colour, size))

# y axis title
p + theme(axis.title.y = element_text(family, face, colour, size))
```

The arguments below can be used for the function *element_text()* to change the appearance of the text :

- **family** : font family
- **face** : font face. Possible values are “plain”, “italic”, “bold” and “bold.italic”
- **colour** : text color
- **size** : text size in pts
- **hjust** : horizontal justification (in [0, 1])
- **vjust** : vertical justification (in [0, 1])
- **lineheight** : line height. In multi-line text, the *lineheight* argument is used to change the spacing between lines.
- **color** : an alias for colour

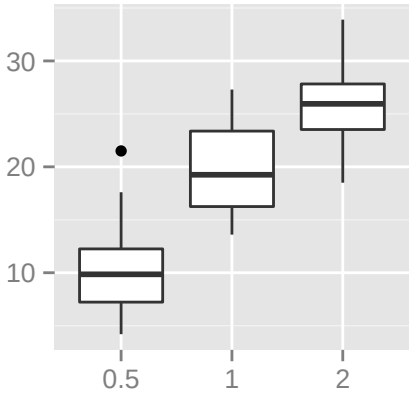
```
# Default plot
p <- ggplot(ToothGrowth, aes(x = dose, y = len)) +
  geom_boxplot() +
  ggtitle("Plot of length \n by dose") +
  xlab("Dose (mg)") + ylab("Teeth length")
p

# Change the color, the size and the face of
# the main title, x and y axis labels
p + theme(
plot.title = element_text(color="red", size=14, face="bold.italic"),
axis.title.x = element_text(color="blue", size=14, face="bold"),
axis.title.y = element_text(color="#993333", size=14, face="bold")
)
```



16.5 Remove x and y axis labels

It’s possible to hide the **main title** and **axis labels** using the function **element_blank()** as follow :

```
# Hide the main title and axis titles
p + theme(
  plot.title = element_blank(),
  axis.title.x = element_blank(),
  axis.title.y = element_blank())
```



Chapter 17

Change the position and the appearance of plot legends

This chapter describes how to change the **legend** of a graph generated using **ggplot2**.
Key functions: guides(), guide_legend() and guide_colourbar().

17.1 Data

ToothGrowth data is used in the examples below :

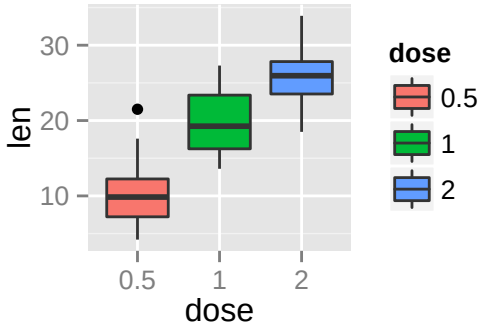
```
# Convert the variable dose from numeric to factor variable
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that the variable *dose* is converted as a factor variable using the above R script.

17.2 Example of plot

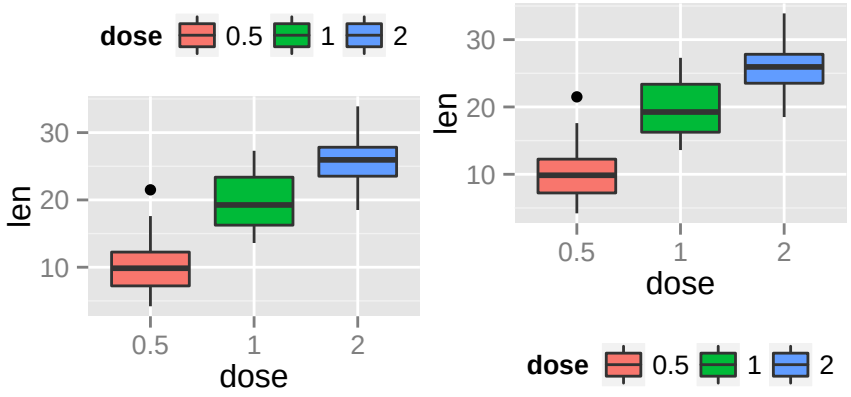
```
library(ggplot2)
p <- ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_boxplot()
p
```



17.3 Change the legend position

The position of the legend can be changed using the function theme() as follow :

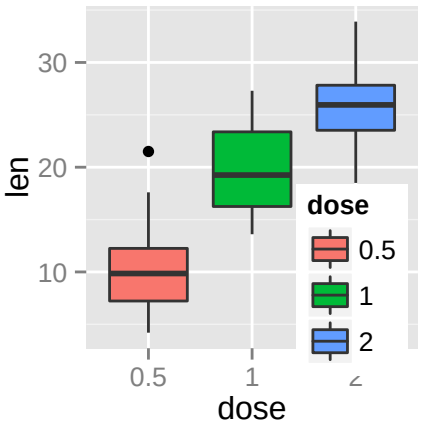
```
p + theme(legend.position="top")

p + theme(legend.position="bottom")
```



The allowed values for the arguments **legend.position** are : “left”,“top”, “right”, “bottom”.

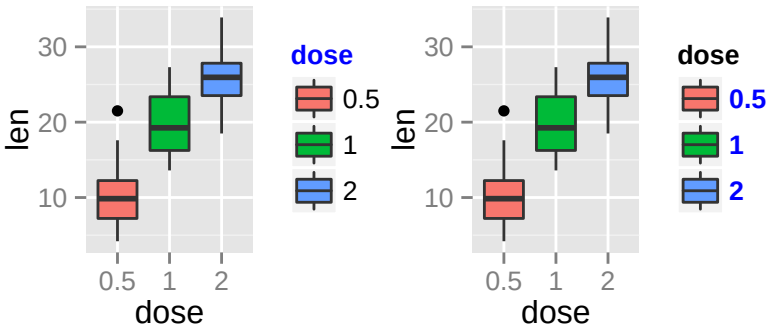
Note that, the argument **legend.position** can be also a numeric vector c(x,y). In this case it is possible to position the legend inside the plotting area. x and y are the coordinates of the legend box. Their values should be between 0 and 1. c(0,0) corresponds to the “bottom left” and c(1,1) corresponds to the “top right” position.

```
p + theme(legend.position = c(0.8, 0.2))
```



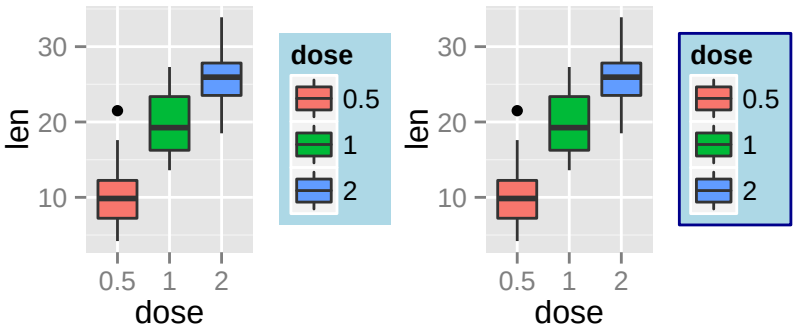
17.4 Change the legend title and text font styles

```
# legend title
p + theme(legend.title = element_text(colour="blue", size=10,
                                      face="bold"))

# legend labels
p + theme(legend.text = element_text(colour="blue", size=10,
                                     face="bold"))
```



17.5 Change the background color of the legend box

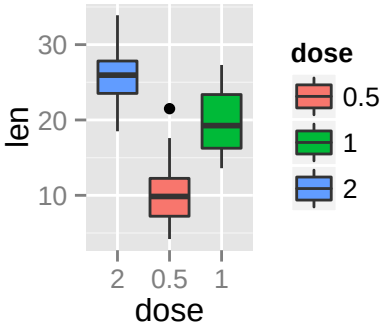
```
# legend box background color
p + theme(legend.background = element_rect(fill="lightblue",
                                   size=0.5, linetype="solid"))

p + theme(legend.background = element_rect(fill="lightblue",
                                   size=0.5, linetype="solid",
                                   colour ="darkblue"))
```



17.6 Change the order of legend items

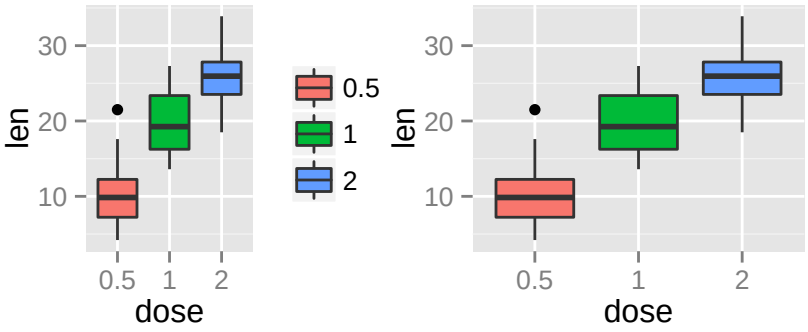
To change the order of items to "2", "0.5", "1" :

```
p + scale_x_discrete(limits=c("2", "0.5", "1"))
```



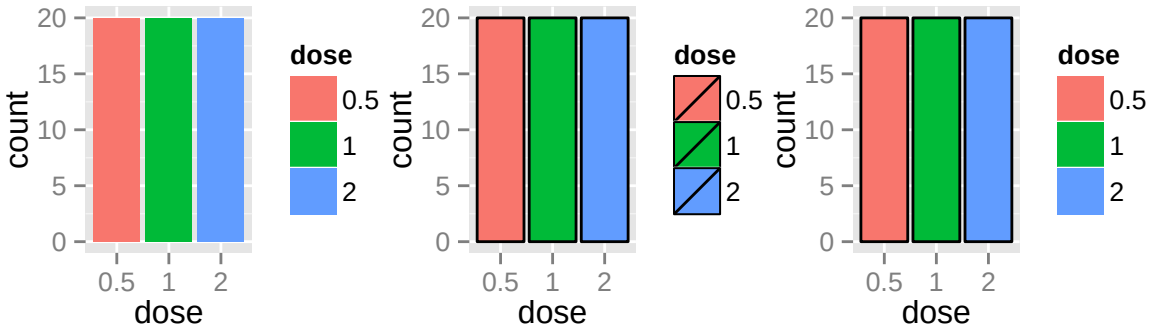
17.7 Remove the plot legend

```
# Remove only the legend title
p + theme(legend.title = element_blank())

# Remove the plot legend
p + theme(legend.position='none')
```



17.8 Remove slashes in the legend of a bar plot

```
# Default plot
ggplot(data=ToothGrowth, aes(x=dose, fill=dose)) + geom_bar()

# Change bar plot border color,
# but slashes are added in the legend
ggplot(data=ToothGrowth, aes(x=dose, fill=dose)) +
  geom_bar(colour="black")

# Hide the slashes:
  #1. plot the bars with no border color,
  #2. plot the bars again with border color, but with a blank legend.
ggplot(data=ToothGrowth, aes(x=dose, fill=dose))+
  geom_bar() +
  geom_bar(colour="black", show_guide=FALSE)
```



17.9 guides() : set or remove the legend for a specific aesthetic

It's possible to use the function **guides()** to set or remove the legend of a particular aesthetic(fill, color, size, shape, etc).

mtcars data is used :

```
# Prepare the data : convert cyl and gear to factor variables
mtcars$cyl<-as.factor(mtcars$cyl)
mtcars$gear <- as.factor(mtcars$gear)
head(mtcars)
```

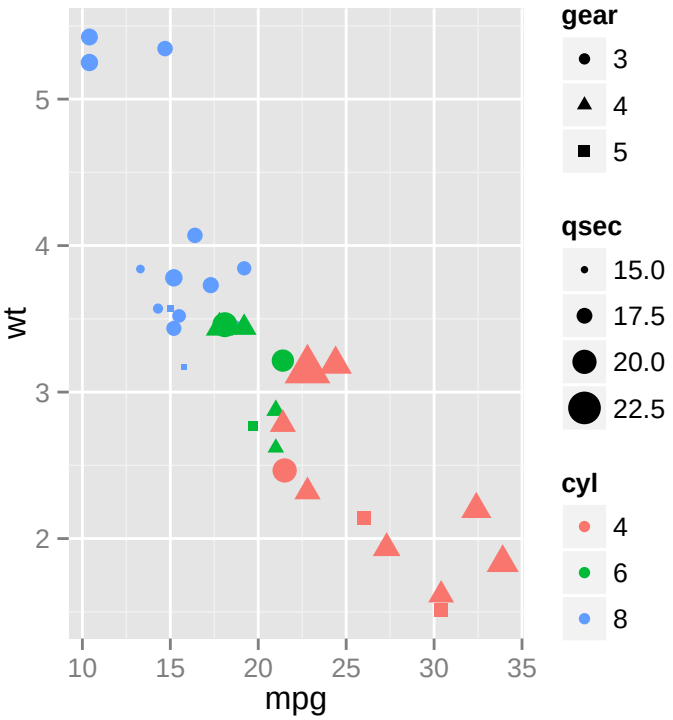
```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
```

```
## Mazda RX4 Wag      21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710         22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive     21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant            18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

17.9.1 Default plot without guide specification

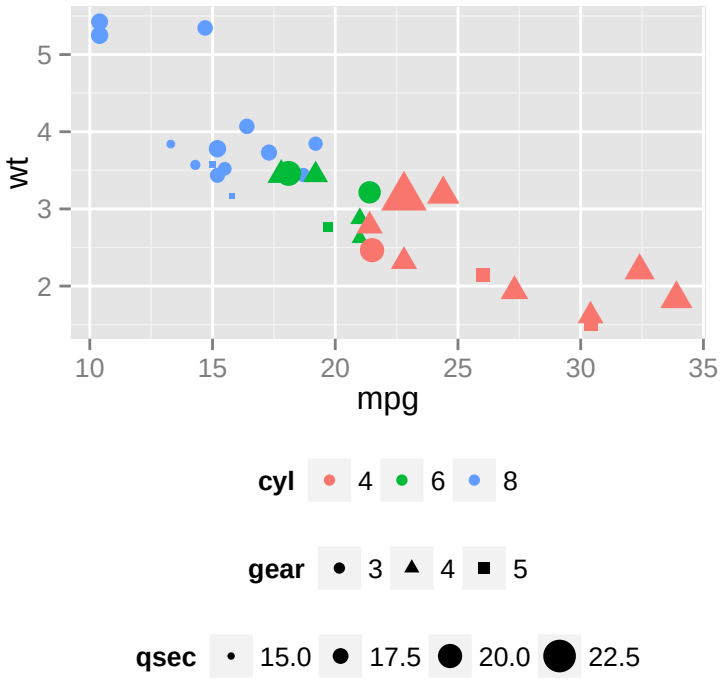
The R code below creates a scatter plot. The color and the shape of the points are determined by the factor variables cyl and gear, respectively. The size of the points are controlled by the variable qsec.

```
p <- ggplot(data = mtcars,
    aes(x=mpg, y=wt, color=cyl, size=qsec, shape=gear))+
    geom_point()

# Print the plot without guide specification
p
```

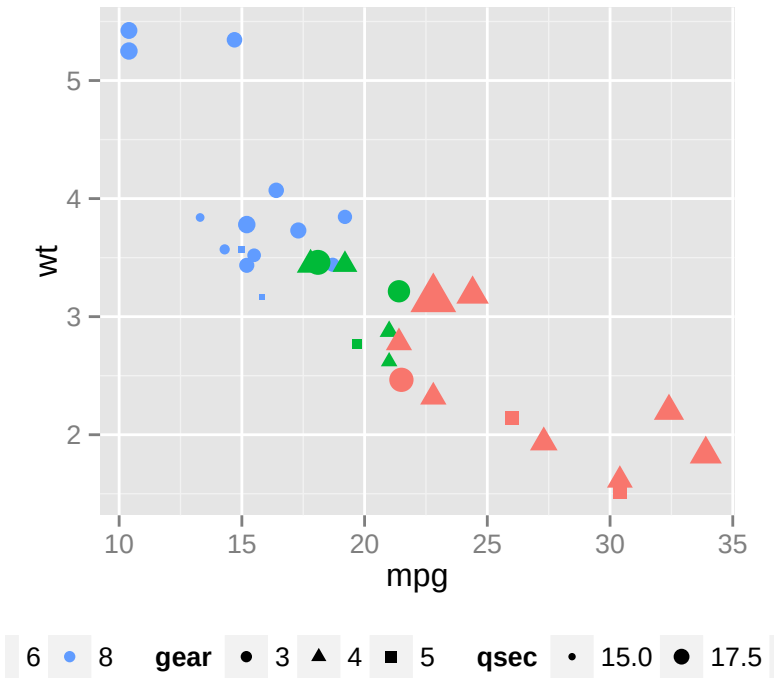


17.9.2 Change the legend position for multiple guides

```
# Change the legend position
p +theme(legend.position="bottom")
```



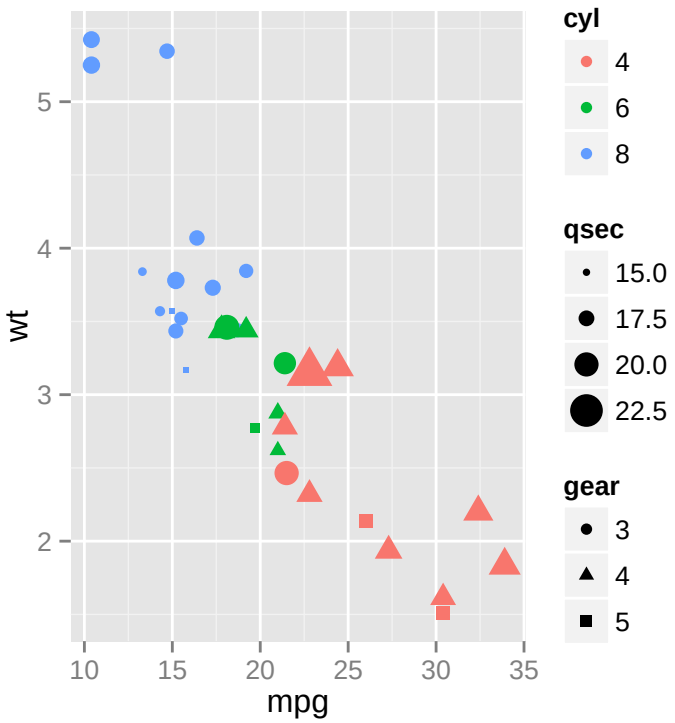

```
# Horizontal legend box
p +theme(legend.position="bottom", legend.box = "horizontal")
```



17.9.3 Change the order for multiple guides

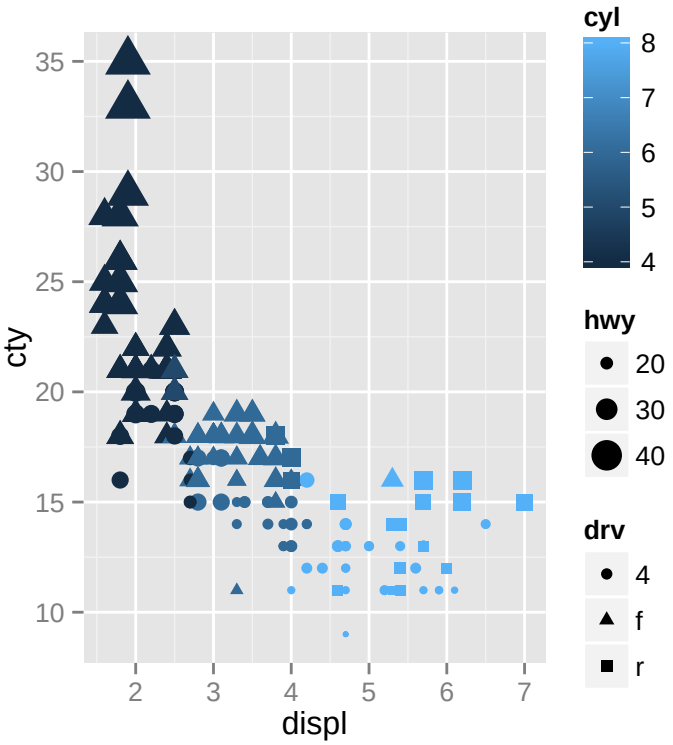
The function **guide_legend()** is used :

```
p+guides(color = guide_legend(order=1),
         size = guide_legend(order=2),
         shape = guide_legend(order=3))
```



If a *continuous color* is used, the order of the *color guide* can be changed using the function *guide_colourbar()* :

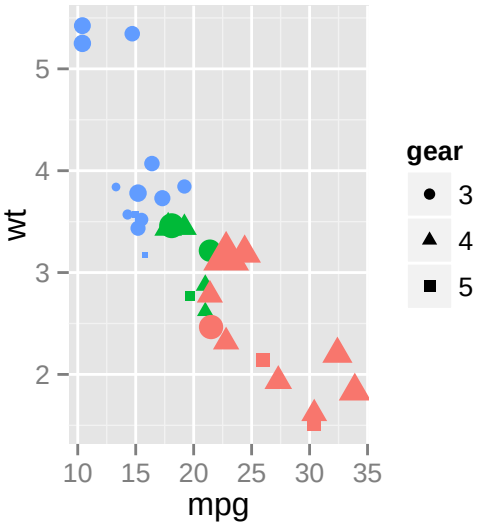
```
qplot(data = mpg, x = displ, y = cty, size = hwy,
      colour = cyl, shape = drv) +
  guides(colour = guide_colourbar(order = 1),
         alpha = guide_legend(order = 2),
         size = guide_legend(order = 3))
```



17.9.4 Remove a legend for a particular aesthetic

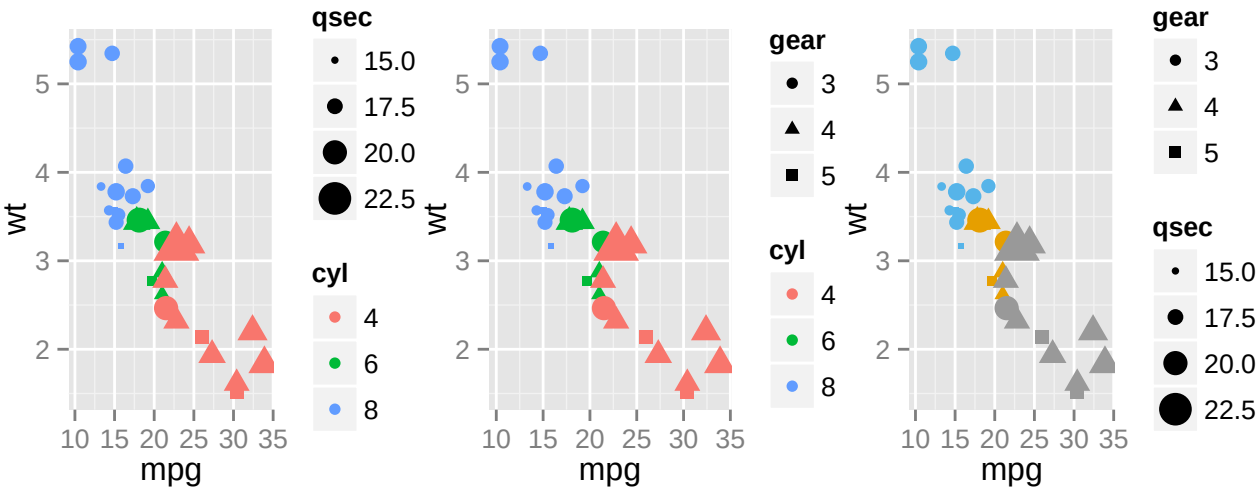
The R code below removes the legend for the aesthetics color and size :

```
p+guides(color = FALSE, size = FALSE)
```



Removing a particular legend can be done also when using the functions *scale_xx*. In this case the argument *guide* is used as follow :

```
# Remove legend for the point shape
p+scale_shape(guide=FALSE)

# Remove legend for size
p +scale_size(guide=FALSE)

# Remove legend for color
p + scale_color_manual(values=c('#999999','#E69F00','#56B4E9'),
                       guide=FALSE)
```



Chapter 18

Change colors automatically and manually

This chapter describes how to change the **color** of a graph generated using **ggplot2** package. A color can be specified either by name (e.g.: "red") or by hexadecimal code (e.g. : "#FF1234").

You will learn how to :

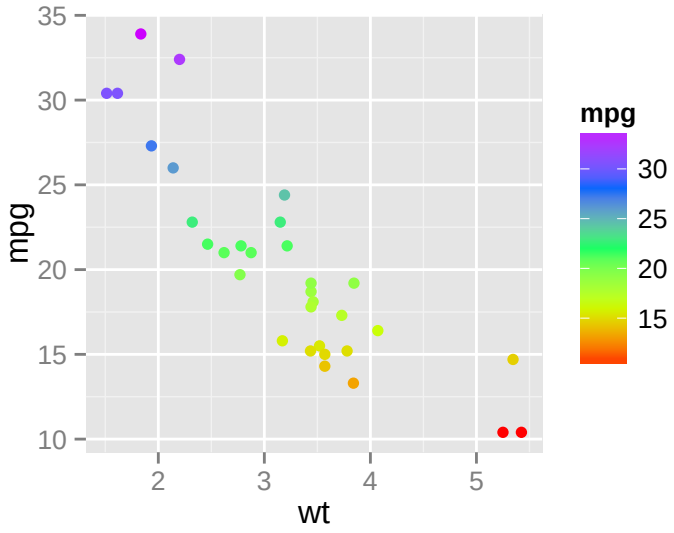
- change colors by groups (automatically and manually)
- use RColorBrewer and Wes Anderson color palettes
- use gradient colors

Key functions:

1. Brewer palettes : scale_colour_brewer(), scale_fill_brewer(), scale_color_brewer()
2. Grey colors: scale_color_grey(), scale_fill_grey()
3. Manual colors: scale_color_manual(), scale_fill_manual()
4. scale_colour_hue()
5. Gradient, continuous colors

- scale_color_gradient(), scale_fill_gradient()
- scale_fill_continuous, scale_color_continuous

6. Gradient, diverging colors: scale_color_gradient2(), scale_fill_gradient2(), scale_colour_gradientn()



18.1 Data

ToothGrowth and *mtcars* data sets are used in the examples below.

```
# Convert dose and cyl columns from numeric to factor variables
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
mtcars$cyl <- as.factor(mtcars$cyl)
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

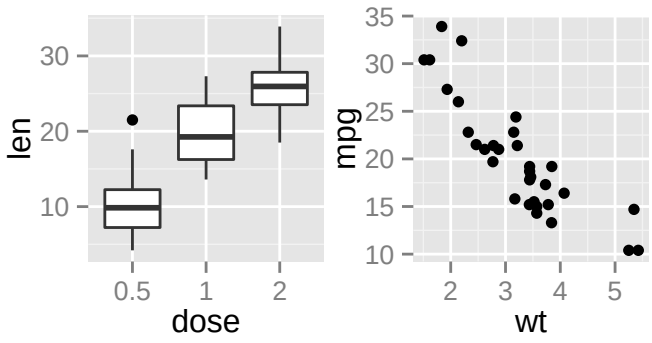
```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Make sure that the columns *dose* and *cyl* are converted as factor variables using the R script above.

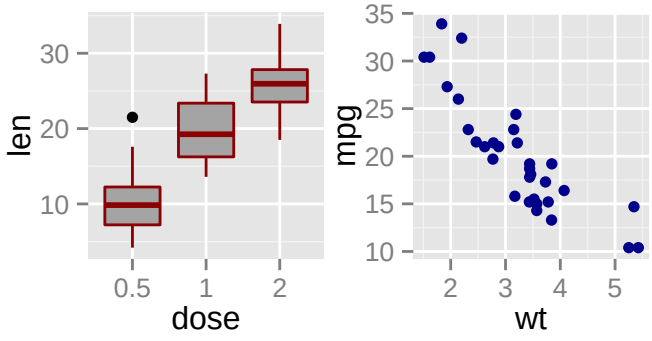
18.2 Simple plots

```
library(ggplot2)
# Box plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +geom_boxplot()

# scatter plot
ggplot(mtcars, aes(x=wt, y=mpg)) + geom_point()
```



18.3 Use a single color

```
# box plot
ggplot(ToothGrowth, aes(x=dose, y=len)) +
  geom_boxplot(fill='#A4A4A4', color="darkred")

# scatter plot
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(color='darkblue')
```

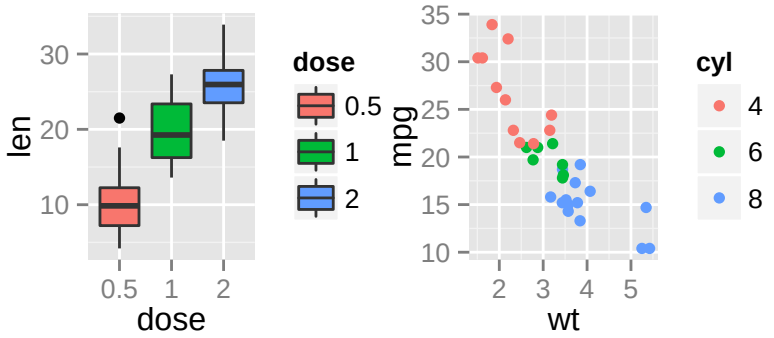


18.4 Change colors by groups

18.4.1 Default colors

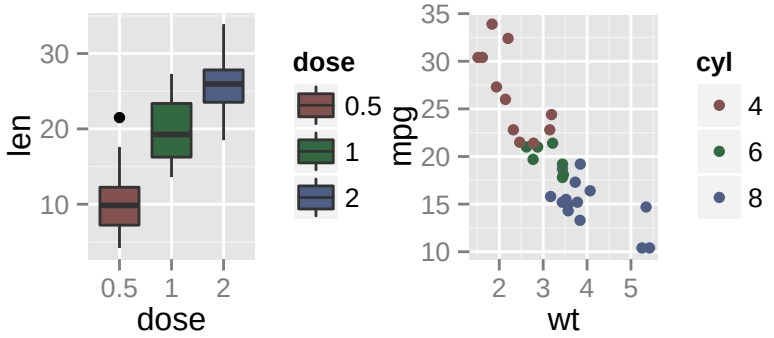
The following R code changes the color of the graph by the levels of *dose* :

```
# Box plot
bp<-ggplot(ToothGrowth, aes(x=dose, y=len, fill=dose)) +
  geom_boxplot()
bp

# Scatter plot
sp<-ggplot(mtcars, aes(x=wt, y=mpg, color=cyl)) + geom_point()
sp
```



The lightness (l) and the chroma (c, intensity of color) of the default (hue) colors can be modified using the functions *scale_hue* as follow :

```
# Box plot
bp + scale_fill_hue(l=40, c=35)

# Scatter plot
sp + scale_color_hue(l=40, c=35)
```



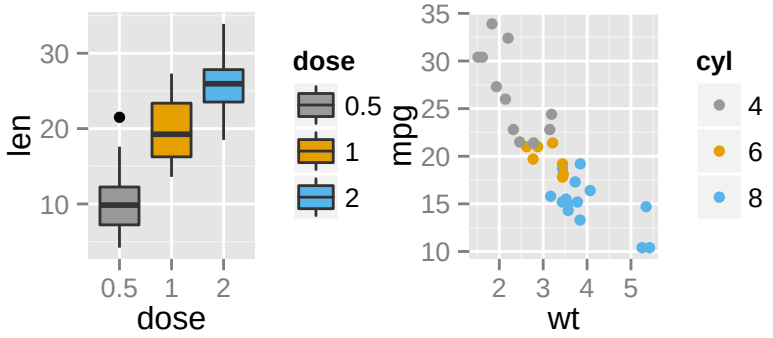
Note that, the default values for l and c are : l = 65, c = 100.

18.4.2 Change colors manually

A custom color palettes can be specified using the functions :

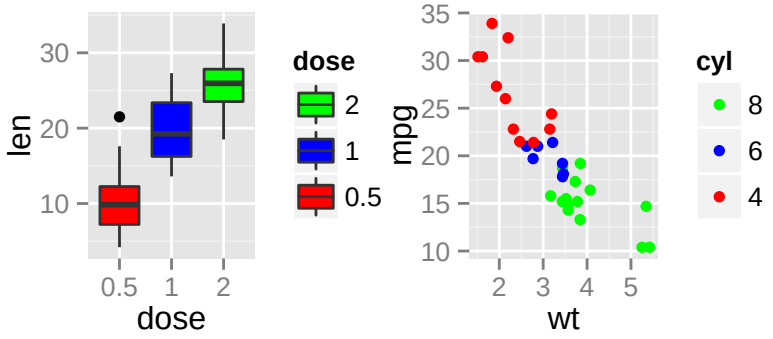
- **scale_fill_manual()** for box plot, bar plot, violin plot, etc
- **scale_color_manual()** for lines and points

```
# Box plot
bp + scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Scatter plot
sp + scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))
```



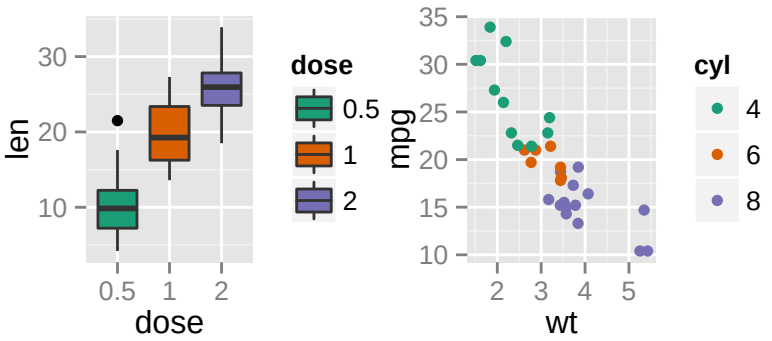
Note that, the argument *breaks* can be used to control the appearance of the legend. This holds true also for the other *scale_xx()* functions.

```
# Box plot
bp + scale_fill_manual(breaks = c("2", "1", "0.5"),
                       values=c("red", "blue", "green"))

# Scatter plot
sp + scale_color_manual(breaks = c("8", "6", "4"),
                        values=c("red", "blue", "green"))
```

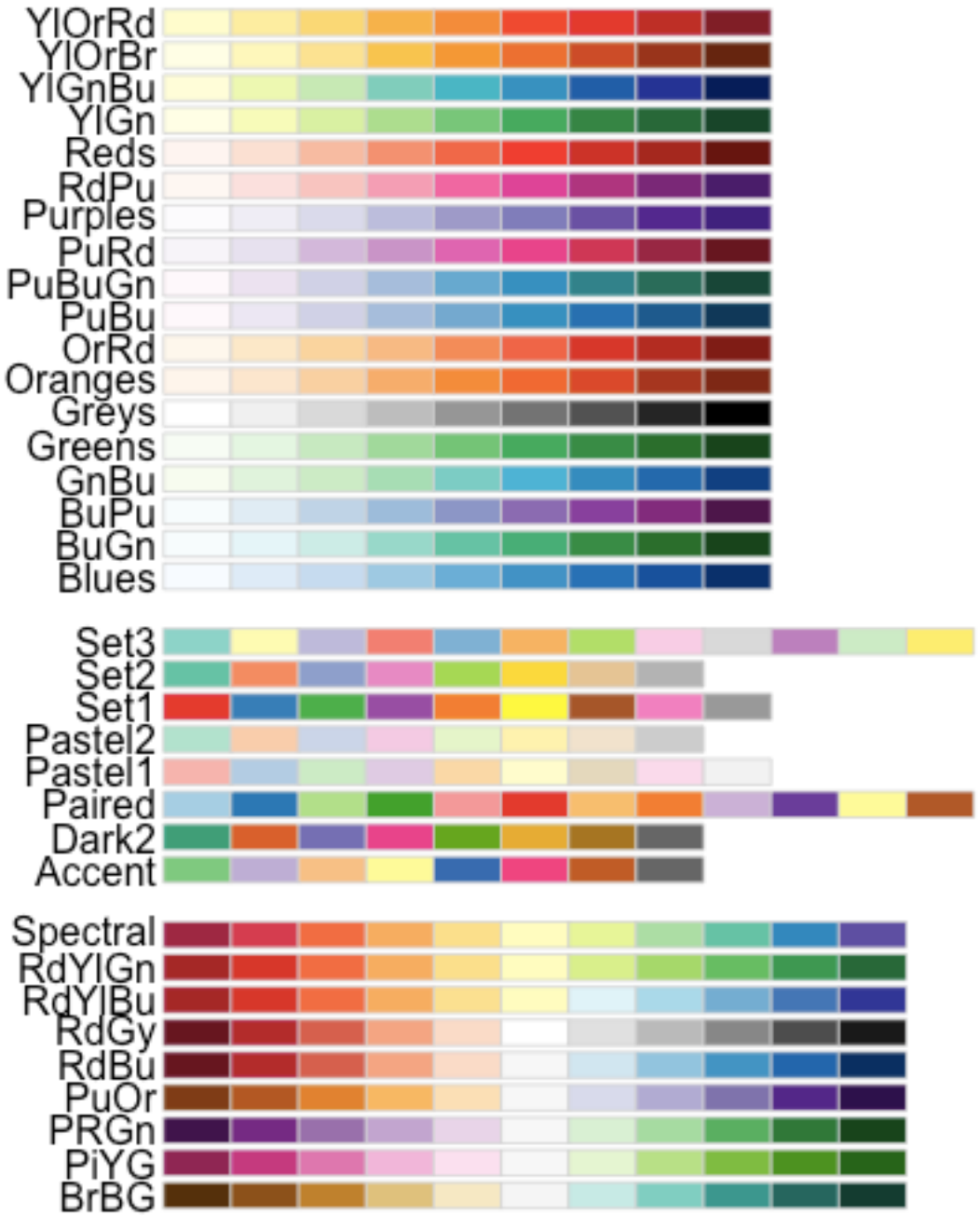


18.4.3 Use RColorBrewer palettes

```
# Box plot
bp + scale_fill_brewer(palette="Dark2")

# Scatter plot
sp + scale_color_brewer(palette="Dark2")
```



The available color palettes in the RColorBrewer package are :



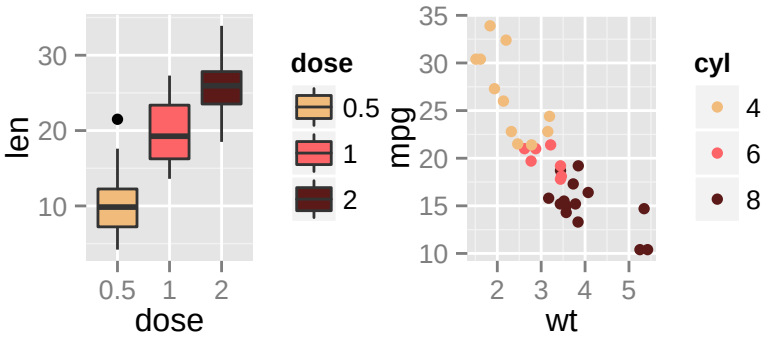
18.4.4 Use Wes Anderson color palettes

Install and load the color palettes as follow :

```
# Install
install.packages("wesanderson")
# Load
library(wesanderson)
```

The available color palettes are :


```
library(wesanderson)
# Box plot
bp+scale_fill_manual(values=wes_palette(n=3, name="GrandBudapest"))

# Scatter plot
sp+scale_color_manual(values=wes_palette(n=3, name="GrandBudapest"))
```



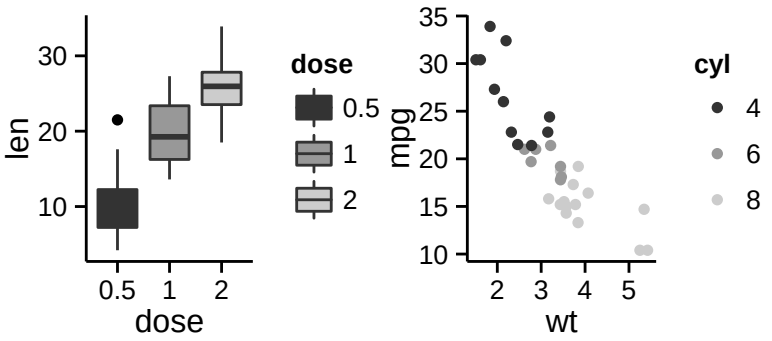
18.5 Use gray colors

The functions to use are :

- **scale_colour_grey()** for points, lines, etc

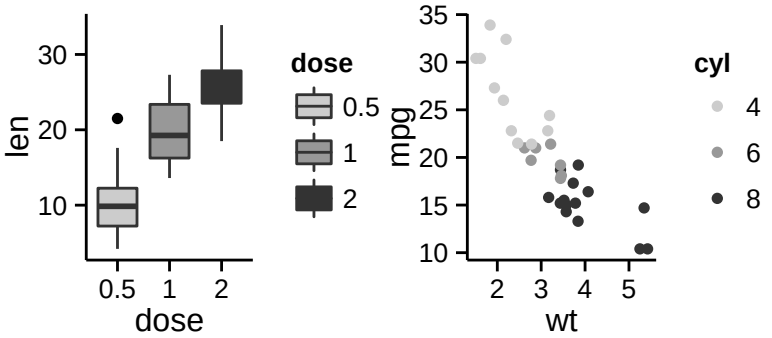
- **scale_fill_grey()** for box plot, bar plot, violin plot, etc

```
# Box plot
bp + scale_fill_grey() + theme_classic()

# Scatter plot
sp + scale_color_grey() + theme_classic()
```



Change the gray value at the low and the high ends of the palette :

```
# Box plot
bp + scale_fill_grey(start=0.8, end=0.2) + theme_classic()

# Scatter plot
sp + scale_color_grey(start=0.8, end=0.2) + theme_classic()
```



Note that, the default value for the arguments *start* and *end* are : start = 0.2, end = 0.8

18.6 Continuous colors

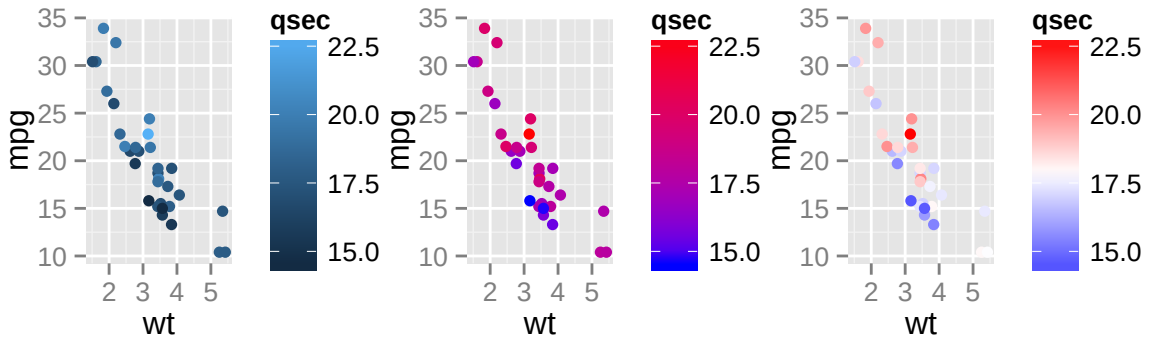
The graph can be colored according to the values of a continuous variable using the functions :

- scale_color_gradient(), scale_fill_gradient() for sequential gradients between two colors
- scale_color_gradient2(), scale_fill_gradient2() for diverging gradients
- scale_color_gradientn(), scale_fill_gradientn() for gradient between n colors

18.6.1 Gradient colors for scatter plots

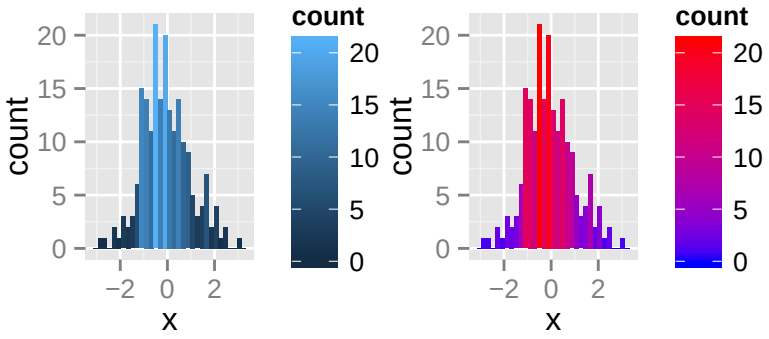
The graphs are colored using the *qsec* continuous variable :

```
# Color by qsec values
sp2<-ggplot(mtcars, aes(x=wt, y=mpg, color=qsec)) + geom_point()
sp2

# Change the low and high colors
# Sequential color scheme
sp2+scale_color_gradient(low="blue", high="red")

# Diverging color scheme
mid<-mean(mtcars$qsec)
sp2+scale_color_gradient2(midpoint=mid,
                  low="blue", mid="white", high="red" )
```



18.6.2 Gradient colors for histogram plots

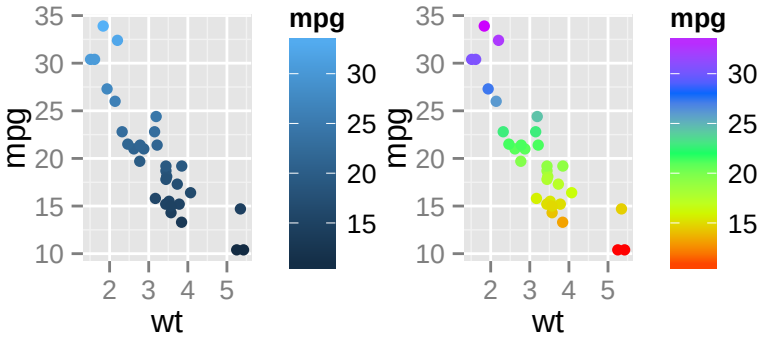
```
set.seed(1234)
x <- rnorm(200)
# Histogram
hp<-qplot(x =x, fill=..count.., geom="histogram")
hp

# Sequential color scheme
hp+scale_fill_gradient(low="blue", high="red")
```



Note that, the functions **scale_color_continuous()** and **scale_fill_continuous()** can be used also to set gradient colors.

18.6.3 Gradient between n colors

```
# Scatter plot
# Color points by the mpg variable
sp3<-ggplot(mtcars, aes(x=wt, y=mpg, color=mpg)) + geom_point()
sp3

# Gradient between n colors
sp3+scale_color_gradientn(colours = rainbow(5))
```



Chapter 19

Point shapes

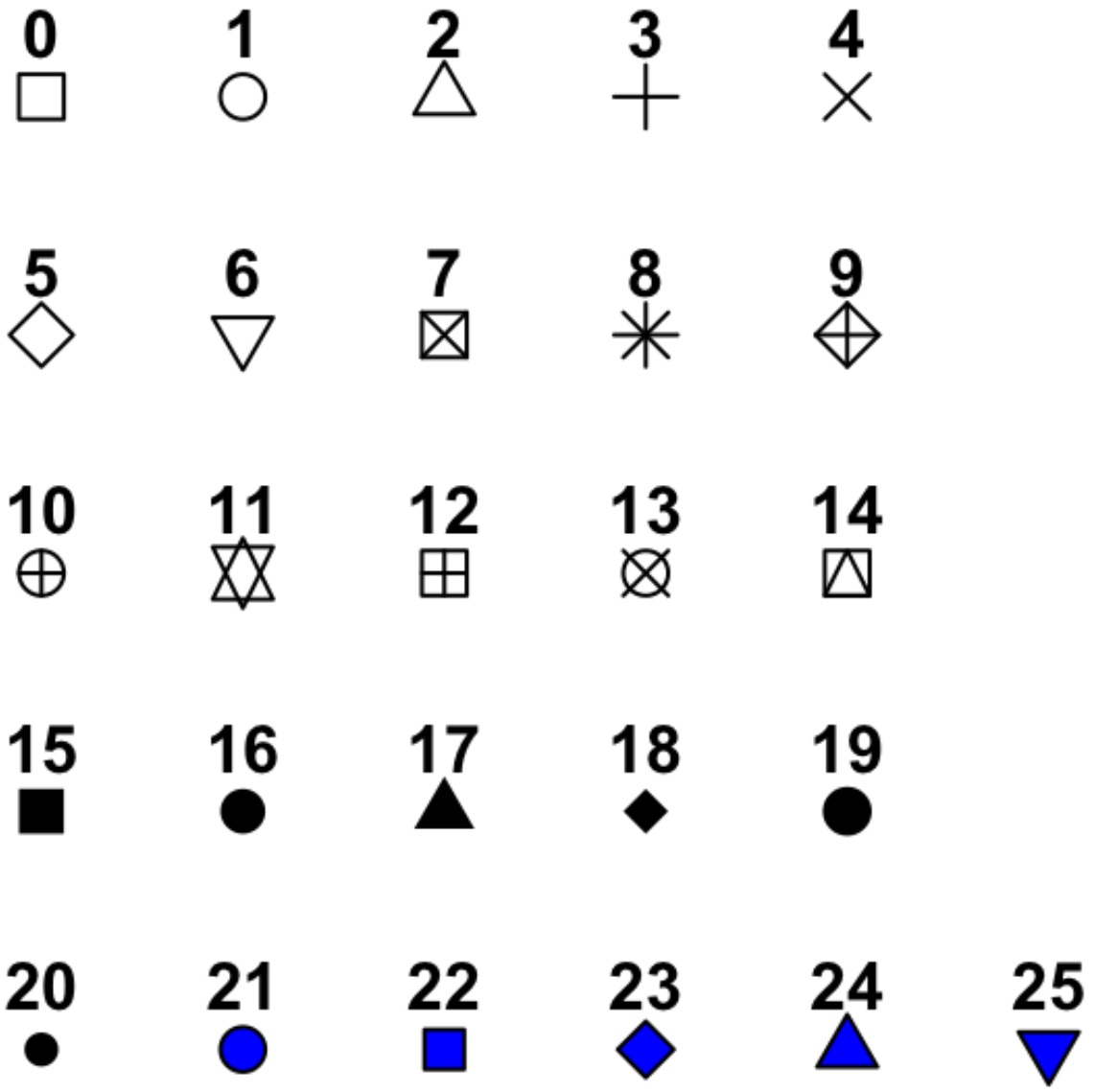
This chapter describes how to change the **point shapes** of a graph generated using **ggplot2**.

Key functions:

- scale_shape_manual() : to change point shapes
- scale_color_manual() : to change point colors
- scale_size_manual() : to change the size of points

19.1 Point shapes in R

The different **points shapes** commonly used in **R** are illustrated in the figure below :



19.2 Data

mtcars data is used in the following examples.

```
df <- mtcars[, c("mpg", "cyl", "wt")]
df$cyl <- as.factor(df$cyl)
head(df)
```

```
##                   mpg cyl    wt
```

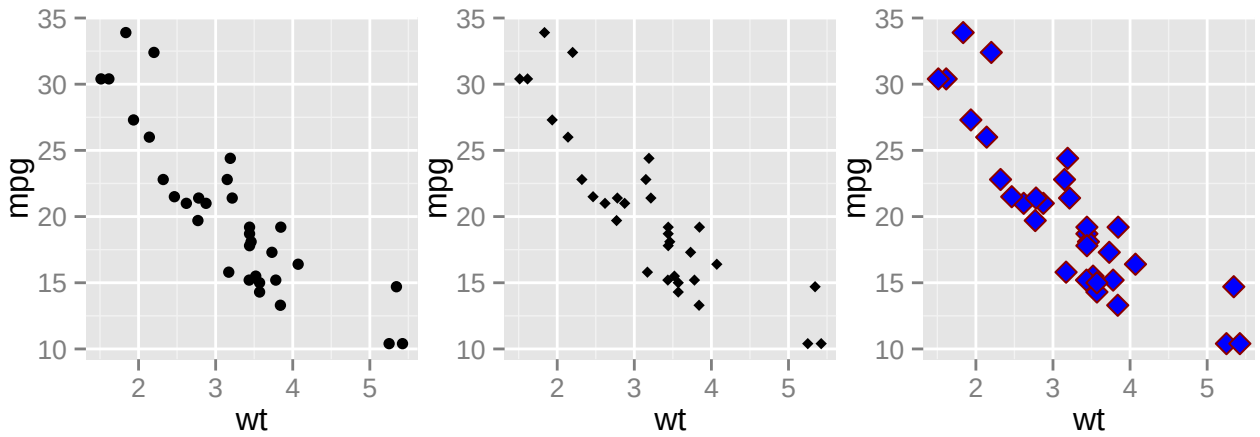
```
## Mazda RX4          21.0   6 2.620
## Mazda RX4 Wag      21.0   6 2.875
## Datsun 710         22.8   4 2.320
## Hornet 4 Drive     21.4   6 3.215
## Hornet Sportabout 18.7   8 3.440
## Valiant            18.1   6 3.460
```

Make sure to convert the column *cyl* from a numeric to a factor variable.

19.3 Basic scatter plots

Create a scatter plot and change point shapes using the argument **shape** :

```
library(ggplot2)
# Basic scatter plot
ggplot(df, aes(x=wt, y=mpg)) +
  geom_point()

# Change the point shape
ggplot(df, aes(x=wt, y=mpg)) +
  geom_point(shape=18)

# change shape, color, fill, size
ggplot(df, aes(x=wt, y=mpg)) +
  geom_point(shape=23, fill="blue", color="darkred", size=3)
```



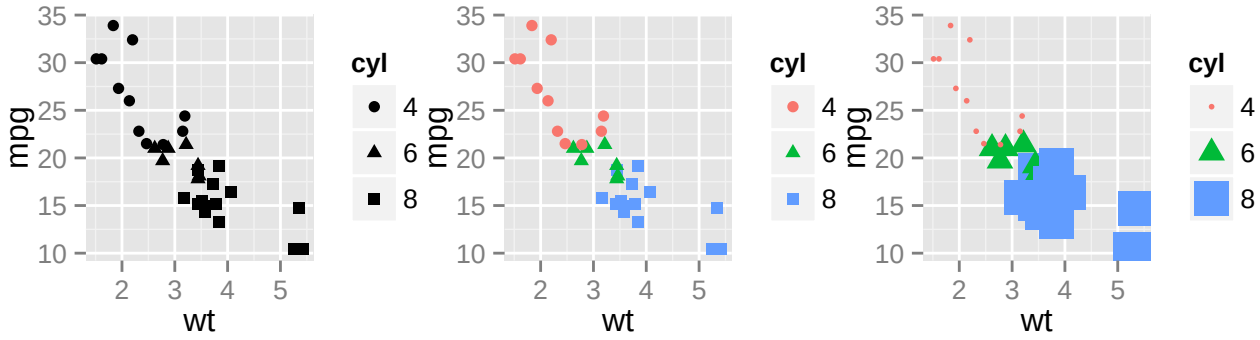
Note that, the argument *fill* can be used only for the point shapes 21 to 25

19.4 Scatter plots with multiple groups

19.4.1 Change the point shapes, colors and sizes automatically

In the R code below, point shapes, colors and sizes are controlled automatically by the variable *cyl*:

```
library(ggplot2)
# Scatter plot with multiple groups
# shape depends on cyl
ggplot(df, aes(x=wt, y=mpg, group=cyl)) +
  geom_point(aes(shape=cyl))

# Change point shapes and colors
ggplot(df, aes(x=wt, y=mpg, group=cyl)) +
  geom_point(aes(shape=cyl, color=cyl))

# change point shapes,  colors and sizes
ggplot(df, aes(x=wt, y=mpg, group=cyl)) +
  geom_point(aes(shape=cyl, color=cyl, size=cyl))
```



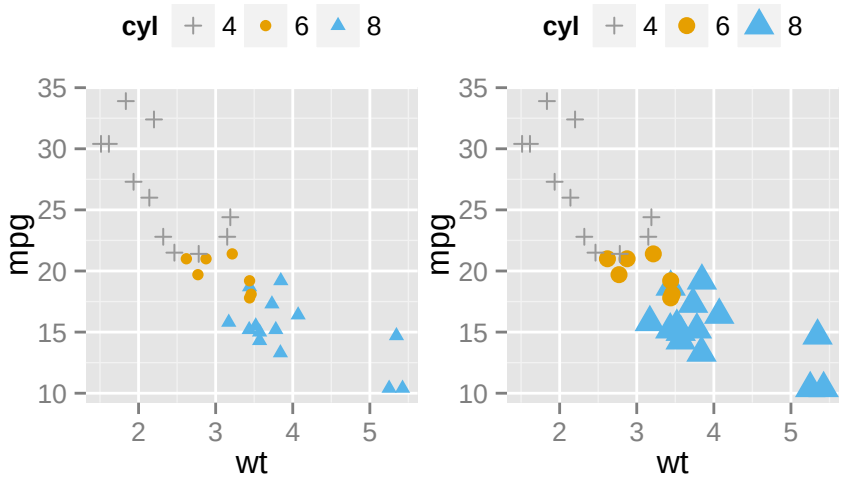
19.4.2 Change point shapes, colors and sizes manually :

The functions below can be used :

- scale_shape_manual() : to change point shapes
- scale_color_manual() : to change point colors
- scale_size_manual() : to change the size of points

```
# Change colors and shapes manually
ggplot(df, aes(x=wt, y=mpg, group=cyl)) +
  geom_point(aes(shape=cyl, color=cyl), size=2)+
  scale_shape_manual(values=c(3, 16, 17))+
```

```
  scale_color_manual(values=c('#999999','#E69F00', '#56B4E9'))+
  theme(legend.position="top")

# Change the point size manually
ggplot(df, aes(x=wt, y=mpg, group=cyl)) +
  geom_point(aes(shape=cyl, color=cyl, size=cyl))+
  scale_shape_manual(values=c(3, 16, 17))+
  scale_color_manual(values=c('#999999','#E69F00', '#56B4E9'))+
  scale_size_manual(values=c(2,3,4))+
  theme(legend.position="top")
```



Chapter 20

Line types

This chapter describes how to change **line types** of a graph generated using **ggplot2** package.

Key functions:

- scale_linetype()
- scale_linetype_manual()
- scale_color_manual()
- scale_size_manual()

20.1 Line types in R

The different line types available in **R software** are : **"blank", "solid", "dashed", "dotted", "dotdash", "longdash", "twodash"**.

Note that, line types can be also specified using numbers : **0, 1, 2, 3, 4, 5, 6**. 0 is for "blank", 1 is for "solid", 2 is for "dashed", ...

A graph of the different line types is shown below :

20.2 Basic line plots

20.2.1 Data

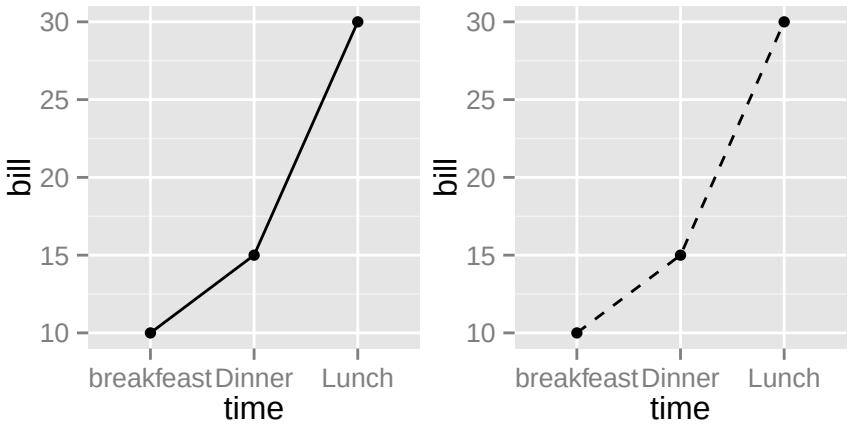
```
df <- data.frame(time=c("breakfeast", "Lunch", "Dinner"),
                 bill=c(10, 30, 15))
head(df)
```

```
##         time bill
## 1 breakfeast   10
## 2      Lunch   30
## 3     Dinner   15
```

20.2.2 Create line plots and change line types

The argument **linetype** is used to change the line type :

```
library(ggplot2)
# Basic line plot with points
ggplot(data=df, aes(x=time, y=bill, group=1)) +
  geom_line()+
  geom_point()

# Change the line type
ggplot(data=df, aes(x=time, y=bill, group=1)) +
  geom_line(linetype = "dashed")+
  geom_point()
```



20.3 Line plot with multiple groups

20.3.1 Data

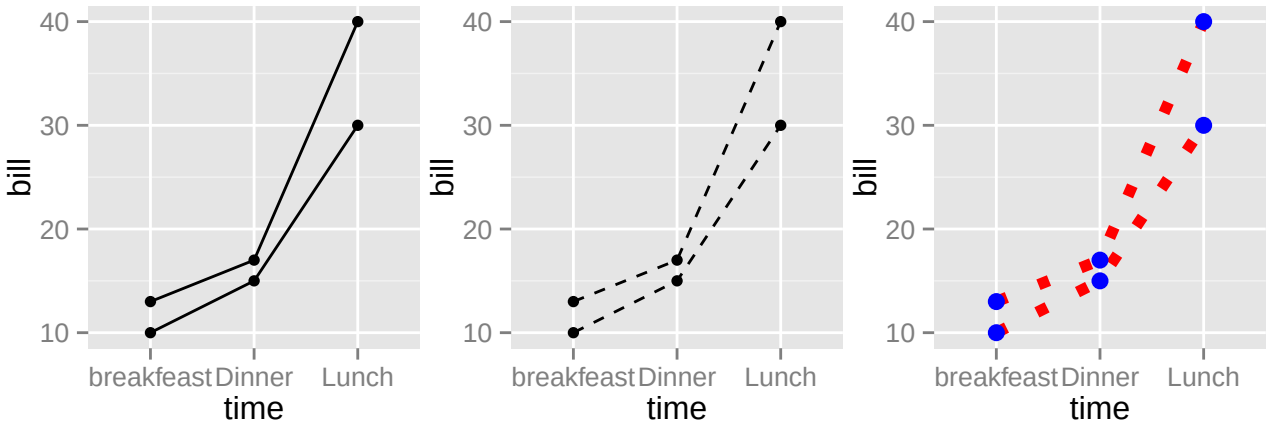
```
df2 <- data.frame(sex = rep(c("Female", "Male"), each=3),
                  time=c("breakfeast", "Lunch", "Dinner"),
                  bill=c(10, 30, 15, 13, 40, 17) )
head(df2)
```

```
##      sex       time bill
## 1 Female breakfeast   10
## 2 Female      Lunch   30
## 3 Female     Dinner   15
## 4   Male breakfeast   13
## 5   Male      Lunch   40
## 6   Male     Dinner   17
```

20.3.2 Change globally the appearance of lines

In the graphs below, line types, colors and sizes are the same for the two groups :

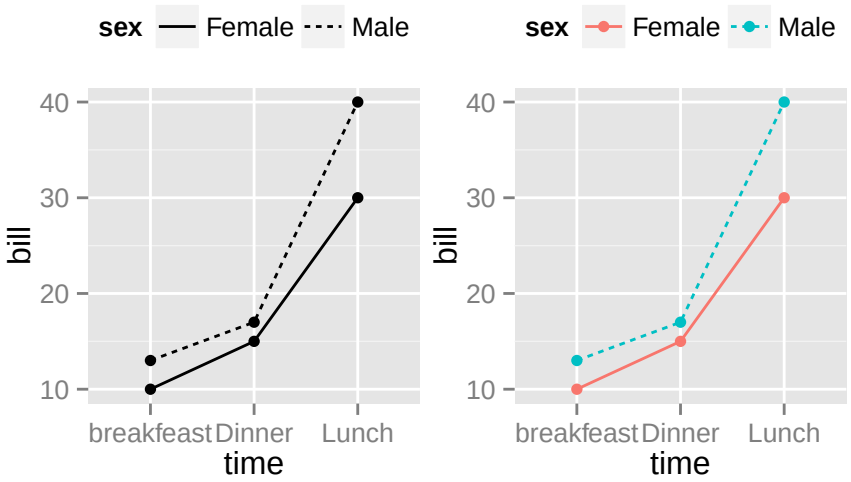
```
library(ggplot2)
# Line plot with multiple groups
ggplot(data=df2, aes(x=time, y=bill, group=sex)) +
  geom_line()+
  geom_point()

# Change line types
ggplot(data=df2, aes(x=time, y=bill, group=sex)) +
  geom_line(linetype="dashed")+
  geom_point()

# Change line colors and sizes
ggplot(data=df2, aes(x=time, y=bill, group=sex)) +
  geom_line(linetype="dotted", color="red", size=2)+
  geom_point(color="blue", size=3)
```



20.3.3 Change automatically the line types by groups

In the graphs below, line types, colors and sizes are changed automatically by the levels of the variable *sex* :

```
# Change line types by groups (sex)
ggplot(df2, aes(x=time, y=bill, group=sex)) +
  geom_line(aes(linetype=sex))+
  geom_point()+
  theme(legend.position="top")

# Change line types + colors
ggplot(df2, aes(x=time, y=bill, group=sex)) +
  geom_line(aes(linetype=sex, color=sex))+
  geom_point(aes(color=sex))+
  theme(legend.position="top")
```



20.3.4 Change manually the appearance of lines

The functions below can be used :

- scale_linetype_manual() : to change line types
- scale_color_manual() : to change line colors
- scale_size_manual() : to change the size of lines

```
# Set line types manually
ggplot(df2, aes(x=time, y=bill, group=sex)) +
  geom_line(aes(linetype=sex))+
  geom_point()+
  scale_linetype_manual(values=c("twodash", "dotted"))+
  theme(legend.position="top")

# Change line colors and sizes
ggplot(df2, aes(x=time, y=bill, group=sex)) +
  geom_line(aes(linetype=sex, color=sex, size=sex))+
  geom_point()+
  scale_linetype_manual(values=c("twodash", "dotted"))+
  scale_color_manual(values=c('#999999','#E69F00'))+
  scale_size_manual(values=c(1, 1.5))+
  theme(legend.position="top")
```

Chapter 21

Add text annotations to a graph

To **add a text** to a plot generated using **ggplot2**, the functions below can be used :

- geom_text()
- annotation()
- annotation_custom()

21.1 Data

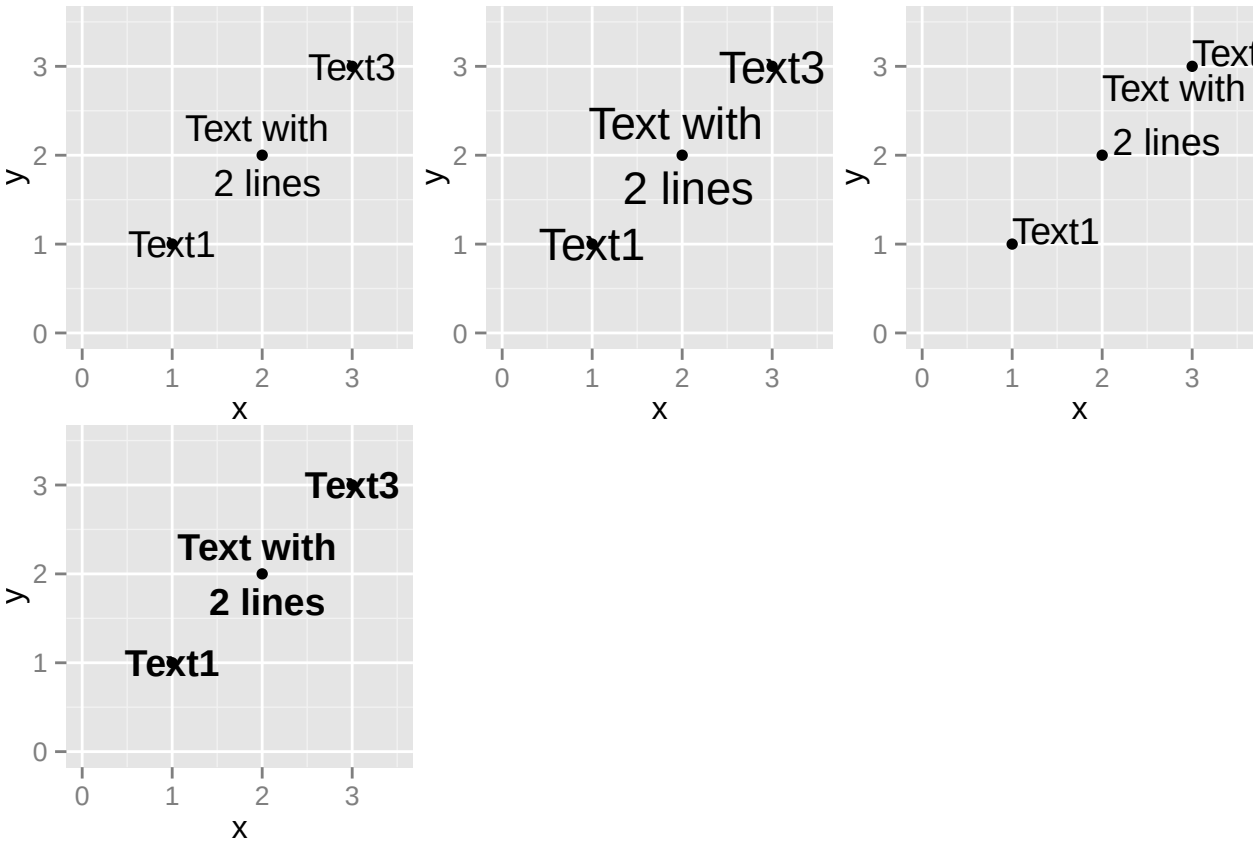
```
df <- data.frame(x=1:3, y=1:3,
                 name=c("Text1", "Text with \n 2 lines", "Text3"))
head(df)
```

```
##   x y                 name
## 1 1 1                Text1
## 2 2 2 Text with \n 2 lines
## 3 3 3                Text3
```

21.2 Text annotations using the function geom_text

```
library(ggplot2)

# Simple scatter plot
sp <- ggplot(data = df, aes(x, y, label=name)) +
  geom_point()+xlim(0,3.5)+ylim(0,3.5)
```

```
# Add texts
sp + geom_text()

# Change the size of the texts
sp + geom_text(size=6)

# Change vertical and horizontal adjustement
sp +  geom_text(hjust=0, vjust=0)

# Change fontface. Allowed values : 1(normal),
# 2(bold), 3(italic), 4(bold.italic)
sp + geom_text(aes(fontface=2))
```

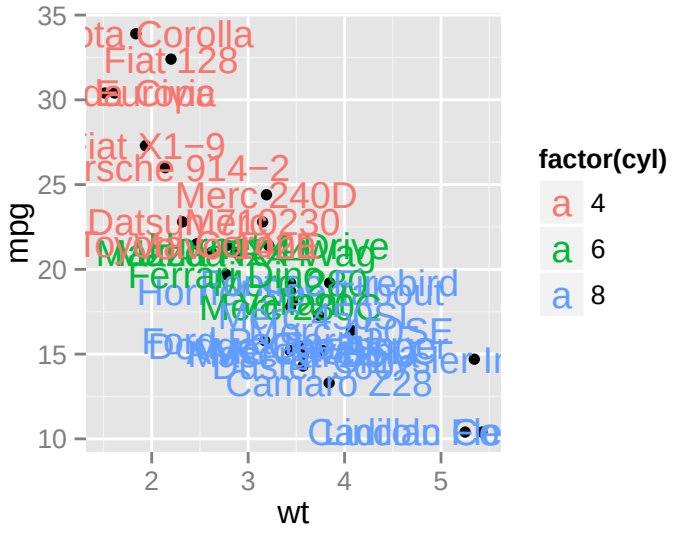


21.3 Change the text color and size by groups

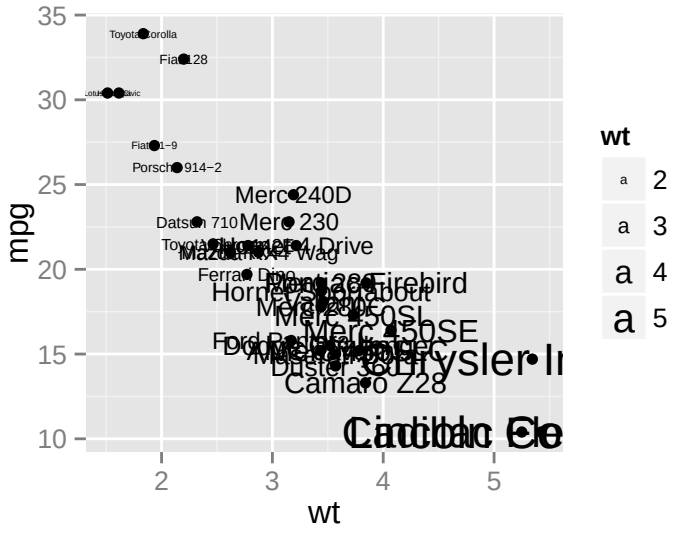
It's possible to change the appearance of the texts using aesthetics (color, size,...) :

```
sp2 <- ggplot(mtcars, aes(x=wt, y=mpg, label=rownames(mtcars)))+
  geom_point()
```

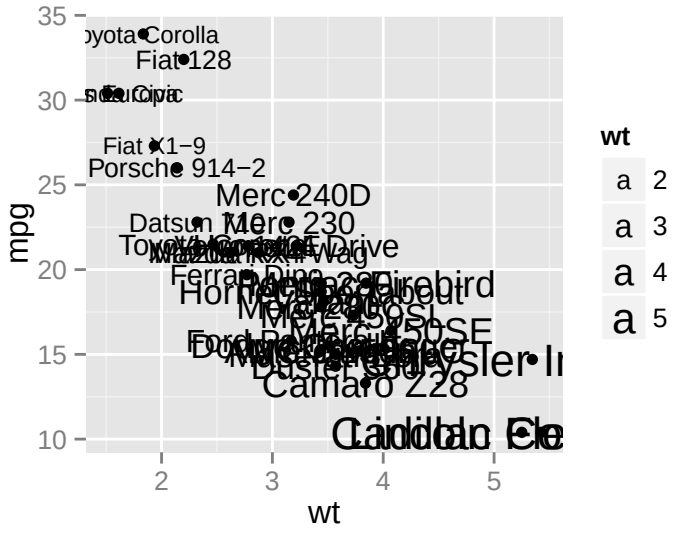
```
# Color by groups
sp2 + geom_text(aes(color=factor(cyl)))
```




```
# Set the size of the text using a continuous variable
sp2 + geom_text(aes(size=wt))
```



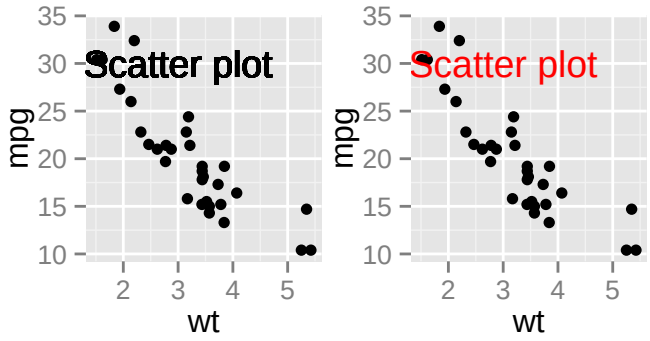

```
sp2 + geom_text(aes(size=wt)) + scale_size(range=c(3,6))
```



21.4 Add a text annotation at a particular coordinate

The functions **geom_text()** and **annotate()** can be used :

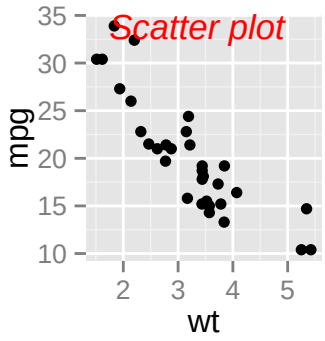
```
# Solution 1
sp2 + geom_text(x=3, y=30, label="Scatter plot")

# Solution 2
sp2 + annotate(geom="text", x=3, y=30, label="Scatter plot",
               color="red")
```



21.5 annotation_custom : Add a static text annotation in the top-right, top-left, ...

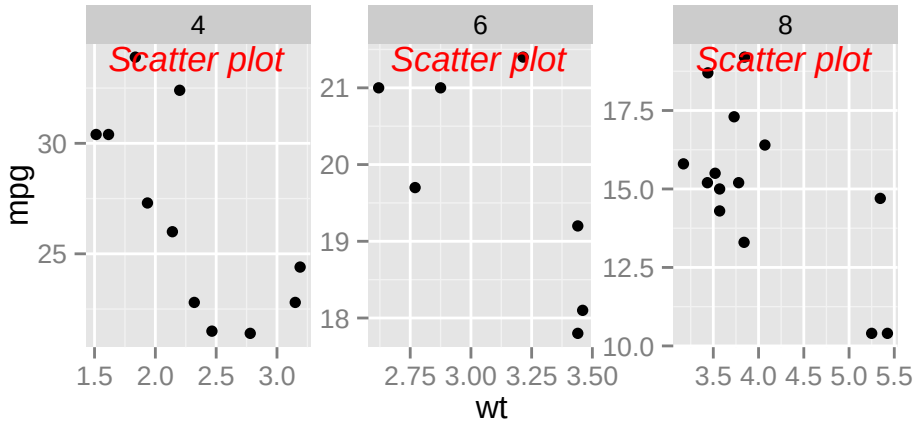
The functions **annotation_custom()** and **textGrob()** are used to add static annotations which are the same in every panel.The *grid* package is required :

```
library(grid)
# Create a text
grob <- grobTree(textGrob("Scatter plot", x=0.1,  y=0.95, hjust=0,
  gp=gpar(col="red", fontsize=13, fontface="italic")))
# Plot
sp2 + annotation_custom(grob)
```



Facet : In the plot below, the annotation is at the same place (in each facet) even if the axis scales vary.

```
sp2 + annotation_custom(grob)+facet_wrap(~cyl, scales="free")
```



Chapter 22

Add straight lines to a plot: horizontal, vertical and regression lines

This chapter describes how to add one or more **straight lines** to a **graph** generated using **ggplot2**.

The R function below can be used :

- **geom_hline()** for horizontal lines
- **geom_abline()** for regression lines
- **geom_vline()** for vertical lines
- **geom_segment()** to add segments

22.1 geom_hline : Add horizontal lines

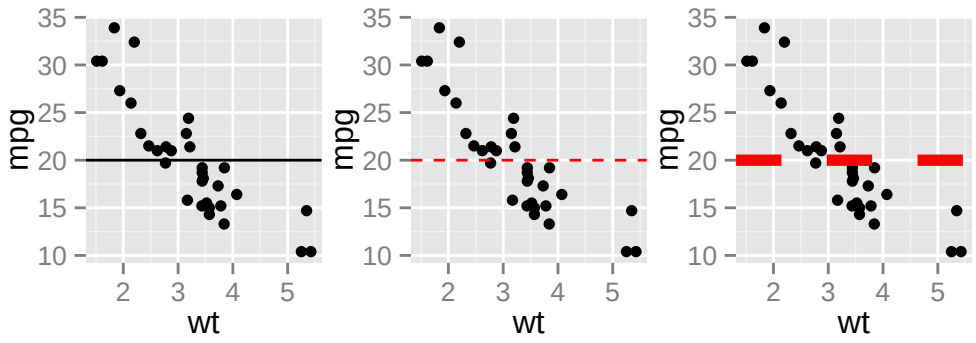
A simplified format of the function **geom_hline()** is :

```
geom_hline(yintercept, linetype, color, size)
```

It draws a horizontal line on the current plot at the specified 'y' coordinates :

```
library(ggplot2)

# Simple scatter plot
sp <- ggplot(data=mtcars, aes(x=wt, y=mpg)) + geom_point()

# Add horizontal line at y = 20
sp + geom_hline(yintercept=20)
```

```
# Change line type and color
sp + geom_hline(yintercept=20, linetype="dashed", color = "red")

# Change line size
sp + geom_hline(yintercept=20, linetype="dashed",
                color = "red", size=2)
```



Read more on line types here: Chapter 20

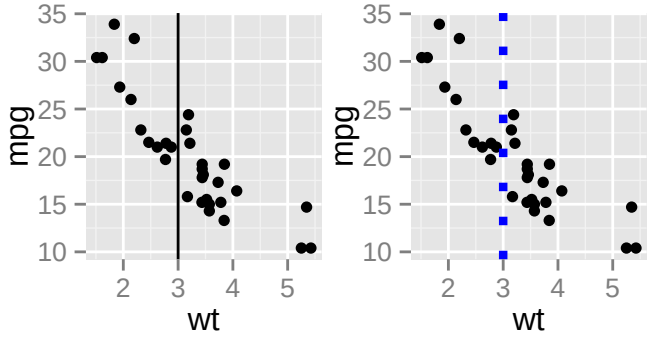
22.2 geom_vline : Add vertical lines

A simplified format of the function **geom_vline()** is :

```
geom_vline(xintercept, linetype, color, size)
```

It draws a vertical line on the current plot at the specified 'x' coordinates :

```
library(ggplot2)

# Add a vertical line at x = 3
sp + geom_vline(xintercept = 3)

# Change line type, color and size
sp + geom_vline(xintercept = 3, linetype="dotted",
                color = "blue", size=1.5)
```



22.3 geom_abline : Add regression lines

A simplified format of the function **geom_abline()** is :

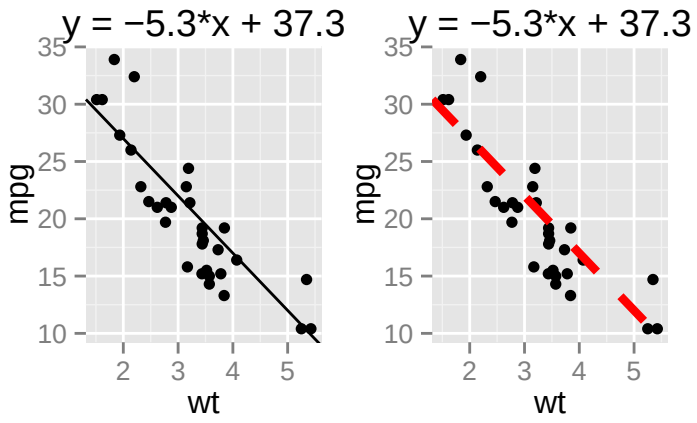
```
geom_abline(intercept, slope, linetype, color, size)
```

The function **lm()** is used to fit linear models.

```
# Fit regression line
require(stats)
reg<-lm(mpg ~ wt, data = mtcars)
reg
```

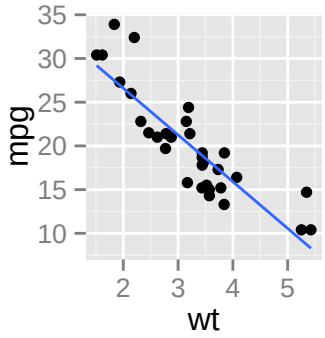
```
##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Coefficients:
## (Intercept)           wt
##      37.285       -5.344
```

```
coeff=coefficients(reg)
# Equation of the line :
eq = paste0("y = ", round(coeff[2],1), "*x + ", round(coeff[1],1))

# Plot
sp + geom_abline(intercept = 37, slope = -5)+
  ggtitle(eq)

# Change line type, color and size
sp + geom_abline(intercept = 37, slope = -5, color="red",
                 linetype="dashed", size=1.5)+
  ggtitle(eq)
```



Note that, the function **stat_smooth()** can be used for fitting smooth models to data.

```
sp + stat_smooth(method="lm", se=FALSE)
```



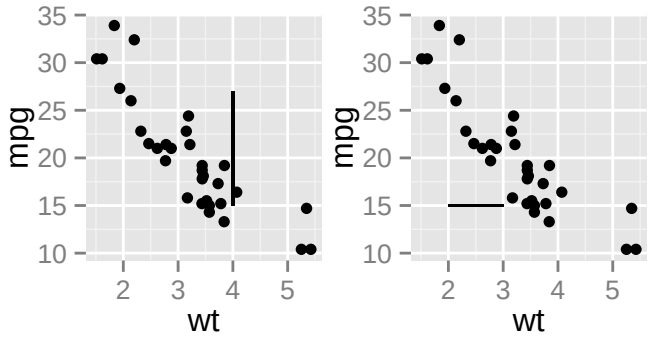
22.4 geom_segment : Add a line segment

A simplified format of the function **geom_segment()** is :

```
geom_segment(aes(x, y, xend, yend))
```

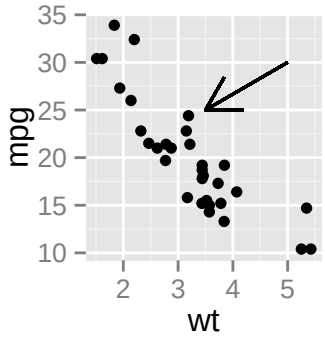
It's possible to use it as follow :

```
# Add a vertical line segment
sp + geom_segment(aes(x = 4, y = 15, xend = 4, yend = 27))

# Add horizontal line segment
sp + geom_segment(aes(x = 2, y = 15, xend = 3, yend = 15))
```



Note that, you can add an arrow at the end of the segment. *grid* package is required

```
library(grid)
sp + geom_segment(aes(x = 5, y = 30, xend = 3.5, yend = 25),
                  arrow = arrow(length = unit(0.5, "cm")))
```


35
30
25
20
15
10
mpg
2
3
4
5
wt

Chapter 23

Axis scales and transformations

This chapter describes how to modify **x and y axis limits** (minimum and maximum values) using **ggplot2**. **Axis transformations** (**log scale**, sqrt, ...) and date axis are also covered in this section.

Key functions:

- expand_limits() : x and y axis limits
- xlim() and ylim()
- scale_x_continuous, scale_y_continuous
- scale_x_log10(), scale_y_log10() : for log10 transformation
- scale_x_sqrt(), scale_y_sqrt() : for sqrt transformation
- coord_trans()
- scale_x_reverse(), scale_y_reverse()
- annotation_logticks
- scale_x_date, scale_y_date
- scale_x_datetime, scale_y_datetime

23.1 Data

ToothGrowth data is used in the following examples :

```
# Convert dose column dose from a numeric to a factor variable
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
head(ToothGrowth)
```

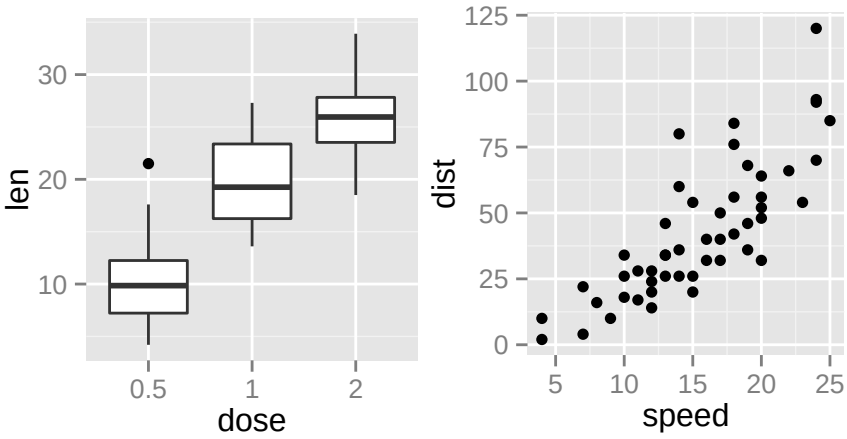
```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
```

```
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that *dose* column is converted as a factor using the above R script.

23.2 Example of plots

```
library(ggplot2)
# Box plot
bp <- ggplot(ToothGrowth, aes(x=dose, y=len)) + geom_boxplot()
bp

# scatter plot
sp<-ggplot(cars, aes(x = speed, y = dist)) + geom_point()
sp
```



23.3 Change x and y axis limits

There are different functions to set axis limits :

- xlim() and ylim()
- expand_limits()
- scale_x_continuous() and scale_y_continuous()

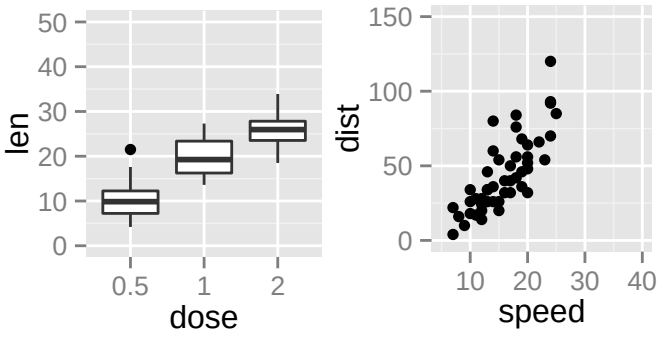
23.3.1 Use xlim() and ylim() functions

To change the range of a continuous axis, the functions **xlim()** and **ylim()** can be used as follow :

```
# x axis limits
sp + xlim(min, max)

# y axis limits
sp + ylim(min, max)
```

min and max are the minimum and the maximum values of each axis.

```
# Box plot : change y axis range
bp + ylim(0,50)

# scatter plots : change x and y limits
sp + xlim(5, 40)+ylim(0, 150)
```

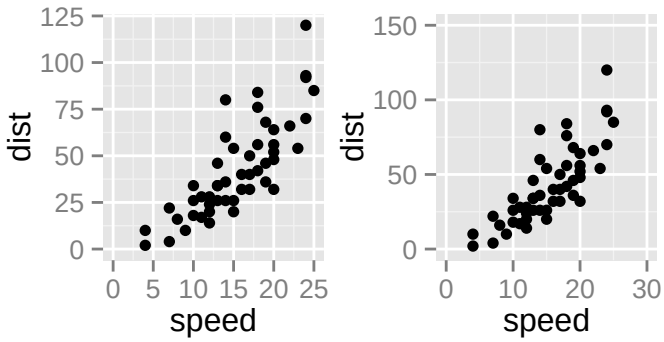


23.3.2 Use expand_limts() function

Note that, the function **expand_limits()** can be used to :

- quickly set the intercept of x and y axes at (0,0)
- change the limits of x and y axes

```
# set the intercept of x and y axis at (0,0)
sp + expand_limits(x=0, y=0)

# change the axis limits
sp + expand_limits(x=c(0,30), y=c(0, 150))
```



23.3.3 Use scale_xx() functions

It is also possible to use the functions **scale_x_continuous()** and **scale_y_continuous()** to change x and y axis limits, respectively.

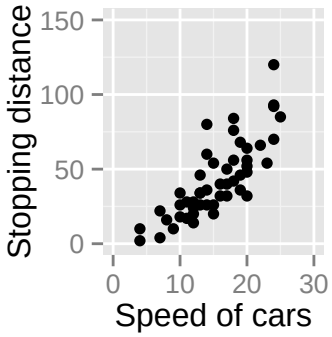
The simplified formats of the functions are :

```
scale_x_continuous(name, breaks, labels, limits, trans)

scale_y_continuous(name, breaks, labels, limits, trans)
```

- **name** : x or y axis labels
- **breaks** : to control the breaks in the guide (axis ticks, grid lines, . . .). Among the possible values, there are :
  - *NULL* : hide all breaks
  - **waiver()** : the default break computation
  - a **character** or **numeric** vector specifying the breaks to display
- **labels** : labels of axis tick marks. Allowed values are :
  - **NULL** for no labels
  - **waiver()** for the default labels
  - **character vector** to be used for break labels
- **limits** : a numeric vector specifying x or y axis limits (min, max)
- **trans** for axis transformations. Possible values are “log2”, “log10”, . . .

The functions *scale_x_continuous()* and *scale_y_continuous()* can be used as follow :

```
# Change x and y axis labels, and limits
sp + scale_x_continuous(name="Speed of cars", limits=c(0, 30)) +
  scale_y_continuous(name="Stopping distance", limits=c(0, 150))
```



23.4 Axis transformations

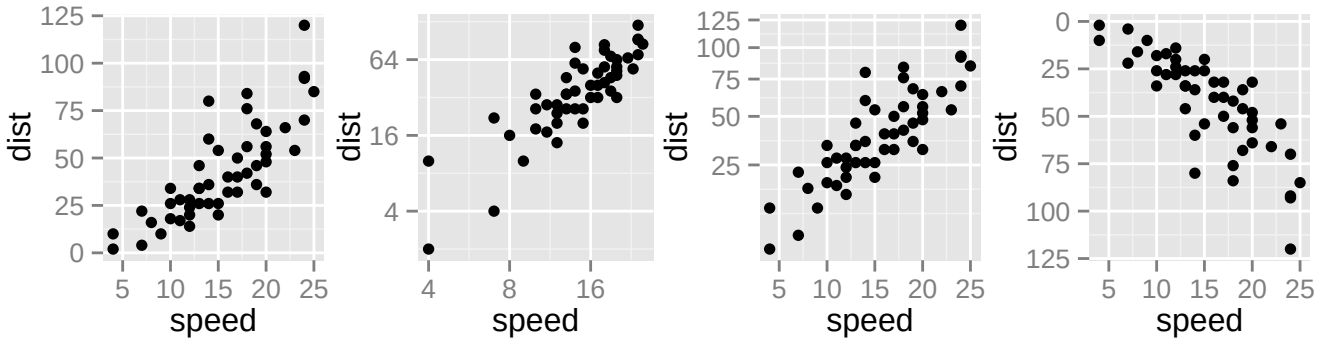
23.4.1 Log and sqrt transformations

Built in functions for axis transformations are :

- scale_x_log10(), scale_y_log10() : for log10 transformation
- scale_x_sqrt(), scale_y_sqrt() : for sqrt transformation
- scale_x_reverse(), scale_y_reverse() : to reverse coordinates
- coord_trans(x =“log10”, y=“log10”) : possible values for x and y are “log12”, “log10”, “sqrt”, ...
- scale_x_continuous(trans=‘log2’), scale_y_continuous(trans=‘log2’) : another allowed value for the argument *trans* is ‘log10’

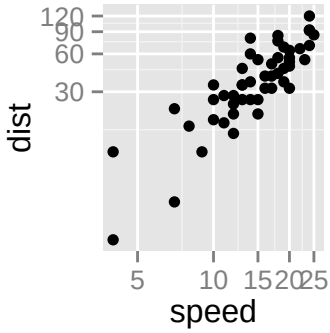
These functions can be used as follow :

```
# Default scatter plot
sp <- ggplot(cars, aes(x = speed, y = dist)) + geom_point()
sp

# Log transformation using scale_xx()
# possible values for trans : 'log2', 'log10','sqrt'
sp + scale_x_continuous(trans='log2') +
  scale_y_continuous(trans='log2')

# Sqrt transformation
sp + scale_y_sqrt()

# Reverse coordinates
sp + scale_y_reverse()
```



The function **coord_trans()** can be used also for the axis transformation

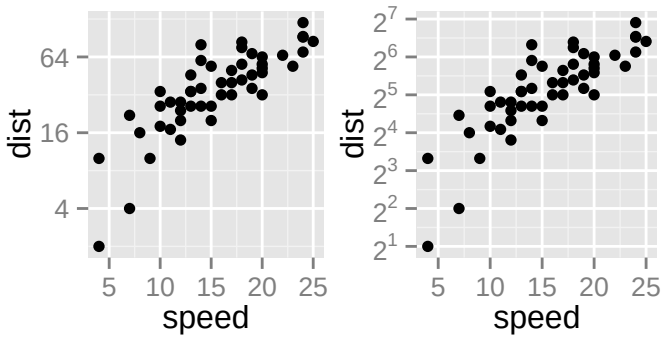
```
# Possible values for x and y : "log2", "log10", "sqrt", ...
sp + coord_trans(x="log2", y="log2")
```



23.4.2 Format axis tick mark labels

Axis tick marks can be set to show exponents. The **scales** package is required to access break formatting functions.

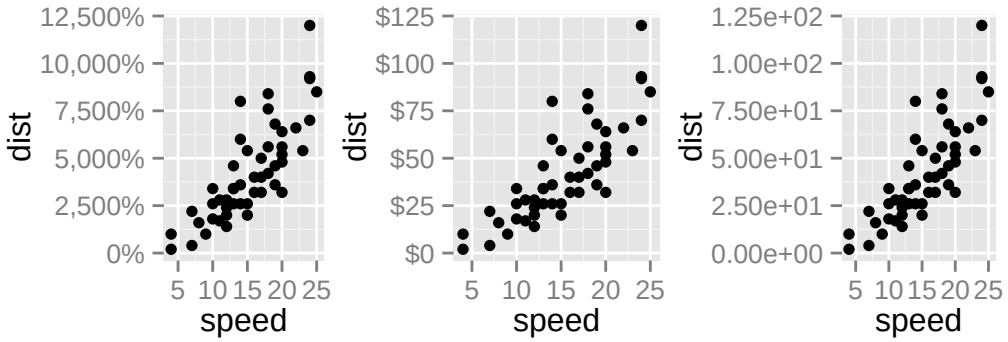
```
# Log2 scaling of the y axis (with visually-equal spacing)
library(scales)
sp + scale_y_continuous(trans = log2_trans())

# show exponents
sp + scale_y_continuous(trans = log2_trans(),
    breaks = trans_breaks("log2", function(x) 2^x),
    labels = trans_format("log2", math_format(2^.x)))
```



Note that many transformation functions are available using the **scales** package : log10_trans(), sqrt_trans(), etc. Use help(trans_new) for a full list.

Format axis tick mark labels :

```
library(scales)
# Percent
sp + scale_y_continuous(labels = percent)

# dollar
sp + scale_y_continuous(labels = dollar)

# scientific
sp + scale_y_continuous(labels = scientific)
```



23.4.3 Display log tick marks

It is possible to add log tick marks using the function **annotation_logticks()**.

Note that, these tick marks make sense only for base 10

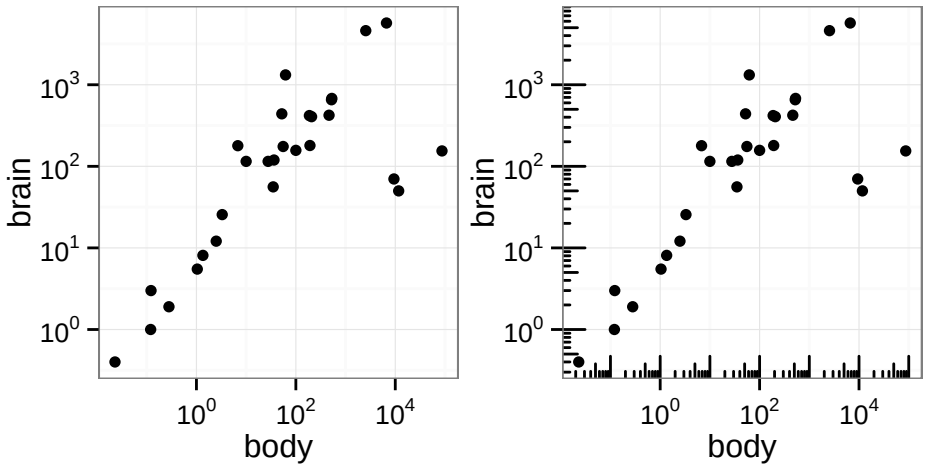
The *Animals* data sets, from the package *MASS*, are used :

```
library(MASS)
head(Animals)
```

```
##                      body brain
## Mountain beaver      1.35   8.1
## Cow                465.00 423.0
## Grey wolf           36.33 119.5
## Goat                27.66 115.0
## Guinea pig           1.04   5.5
## Dipliodocus      11700.00  50.0
```

The function **annotation_logticks()** can be used as follow :

```
library(MASS) # to access Animals data sets
library(scales) # to access break formatting functions
# x and y axis are transformed and formatted
p2 <- ggplot(Animals, aes(x = body, y = brain)) + geom_point() +
     scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x),
                   labels = trans_format("log10", math_format(10^.x))) +
     scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                   labels = trans_format("log10", math_format(10^.x))) +
     theme_bw()
# log-log plot without log tick marks
p2

# Show log tick marks
p2 + annotation_logticks()
```



Note that, default log ticks are on bottom and left.

To specify the sides of the log ticks :

```
# Log ticks on left and right
p2 + annotation_logticks(sides="lr")
```

```
# All sides
p2+annotation_logticks(sides="trbl")
```

Allowed values for the argument *sides* are :

- t : for top
- r : for right
- b : for bottom
- l : for left
- the combination of t, r, b and l

23.5 Format date axes

The functions **scale_x_date()** and **scale_y_date()** are used.

23.5.1 Example of data

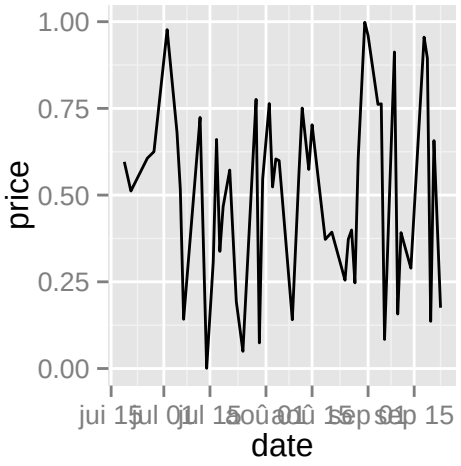
23.5.2 Create some time serie data

```
df <- data.frame(
  date = seq(Sys.Date(), len=100, by="1 day")[sample(100, 50)],
  price = runif(50)
)
df <- df[order(df$date), ]
head(df)
```

```
##          date     price
## 50 2015-06-19 0.5959110
## 27 2015-06-21 0.5117742
## 8  2015-06-26 0.6065630
## 19 2015-06-28 0.6251975
## 26 2015-07-02 0.9770535
## 6  2015-07-05 0.6805039
```

23.5.3 Plot with dates

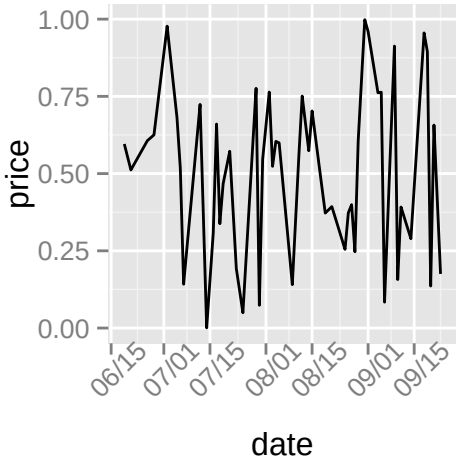
```
# Plot with date
dp <- ggplot(data=df, aes(x=date, y=price)) + geom_line()
dp
```

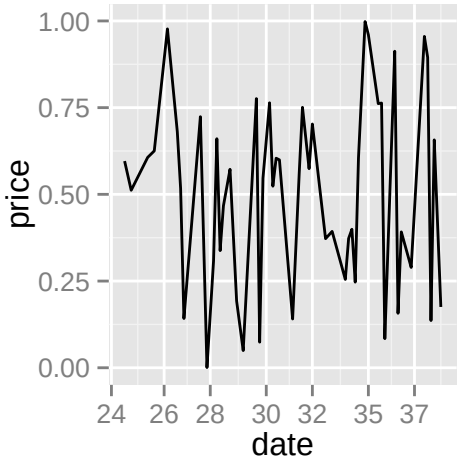


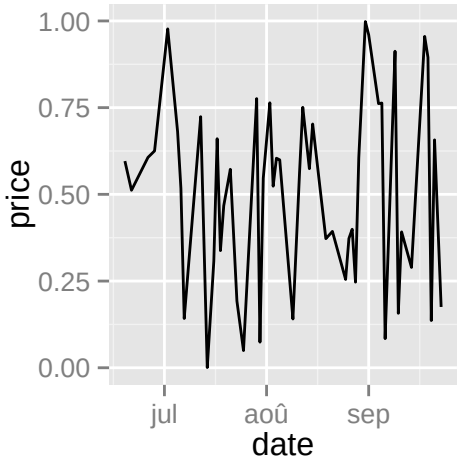
23.5.4 Format axis tick mark labels

Load the package *scales* to access break formatting functions.

```
library(scales)
# Format : month/day
dp + scale_x_date(labels = date_format("%m/%d")) +
  theme(axis.text.x = element_text(angle=45))

# Format : Week
dp + scale_x_date(labels = date_format("%W"))

# Months only
dp + scale_x_date(breaks = date_breaks("months"),
  labels = date_format("%b"))
```







23.5.5 Date axis limits

US economic time series data sets (from ggplot2 package) are used :

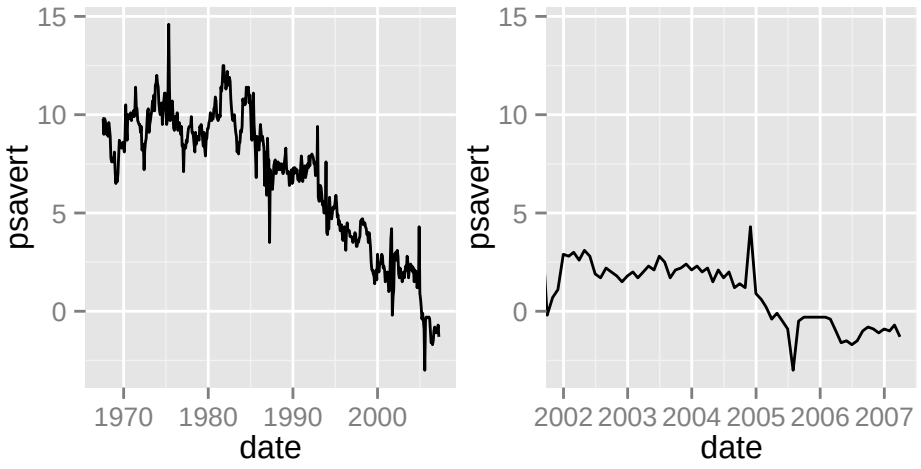
```
head(economics)
```

```
##         date   pce    pop psavert uempmed unemploy
## 1 1967-06-30 507.8 198712     9.8     4.5     2944
## 2 1967-07-31 510.9 198911     9.8     4.7     2945
## 3 1967-08-31 516.7 199113     9.0     4.6     2958
## 4 1967-09-30 513.3 199311     9.8     4.9     3143
## 5 1967-10-31 518.5 199498     9.7     4.7     3066
## 6 1967-11-30 526.2 199657     9.4     4.8     3018
```

Create the plot of psavert by date :

- date : Month of data collection
- psavert : personal savings rate

```
# Plot with dates
dp <- ggplot(data=economics, aes(x=date, y=psavert)) + geom_line()
dp

# Axis limits c(min, max)
min <- as.Date("2002-1-1")
max <- max(economics$date)
dp+ scale_x_date(limits = c(min, max))
```



23.6 Read also

See also the function **scale_x_datetime()** and **scale_y_datetime()** to plot a data containing date and time.

Chapter 24

Axis ticks : customize tick marks and labels

This chapter describes how to customize **axis tick marks** and **labels** using **ggplot2** package.

Key functions:

- scale_x_discrete(), scale_y_discrete()
- scale_x_continuous(), scale_y_continuous()

24.1 Data

ToothGrowth data is used in the examples hereafter.

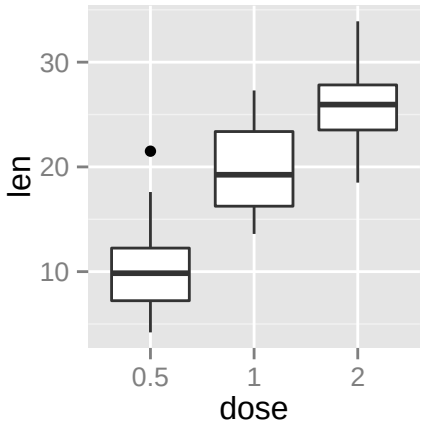
```
# Convert dose column from numeric to factor variable
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that *dose* column are converted as a factor using the above R script.

24.2 Example of plots

```
library(ggplot2)
p <- ggplot(ToothGrowth, aes(x=dose, y=len)) + geom_boxplot()
p
```



24.3 Change the appearance of the axis tick mark labels

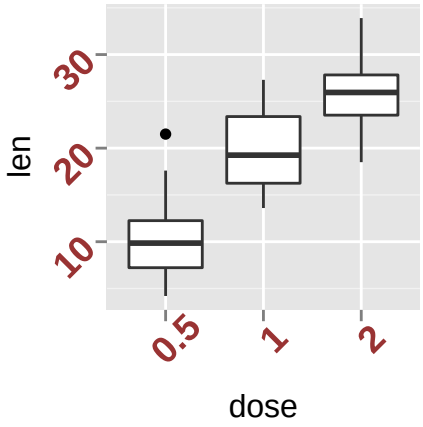
The **color**, the **font size** and the **font face** of axis tick mark labels can be changed using the functions **theme()** and **element_text()** as follow :

```
# x axis tick mark labels
p + theme(axis.text.x= element_text(family, face, colour, size))

# y axis tick mark labels
p + theme(axis.text.y = element_text(family, face, colour, size))
```

The following arguments can be used for the function *element_text()* to change the appearance of the text :

- **family** : font family
- **face** : font face. Possible values are “plain”, “italic”, “bold” and “bold.italic”
- **colour** : text color
- **size** : text size in pts
- **angle** : angle (in [0, 360])

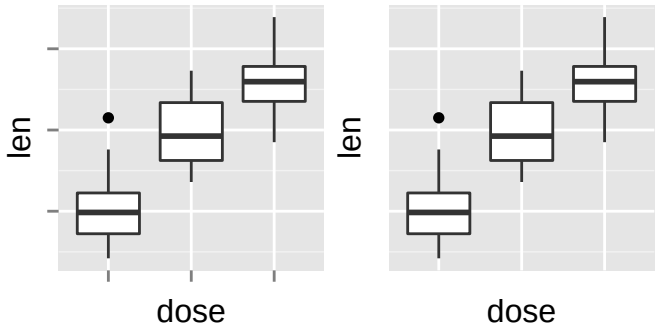
```
# Change the appearance and the orientation angle
# of axis tick labels
p + theme(axis.text.x = element_text(face="bold", color="#993333",
                          size=14, angle=45),
          axis.text.y = element_text(face="bold", color="#993333",
                          size=14, angle=45))
```



24.4 Hide x and y axis tick mark labels

axis ticks and **tick mark labels** can be removed using the function **element_blank()** as follow :

```
# Hide x an y axis tick mark labels
p + theme(
  axis.text.x = element_blank(),
  axis.text.y = element_blank())

# Remove axis ticks and tick mark labels
p + theme(
  axis.text.x = element_blank(),
  axis.text.y = element_blank(),
  axis.ticks = element_blank())
```



24.5 Change axis lines

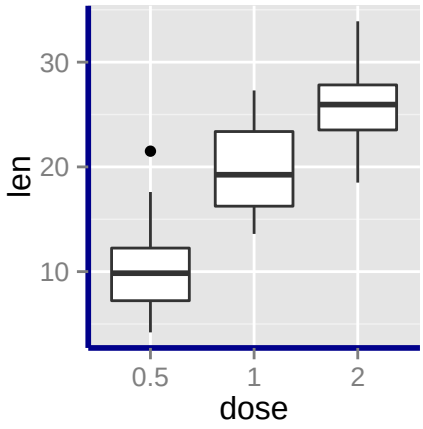
Axis lines can be changed using the function **element_line()** as follow :

```
p + theme(axis.line = element_line(colour, size, linetype,
                                   lineend, color))
```

The arguments of **element_line()** are :

- **colour**, **color** : line color
- **size** : line size
- **linetype** : line type. **Line type** can be specified using either text ("blank", "solid", "dashed", "dotted", "dotdash", "longdash", "twodash") or number (0, 1, 2, 3, 4, 5, 6). Note that linetype = "solid" is identical to linetype=1. The available line types in R are described in this post : Line type in R software
- **lineend** : line end. Allowed values for line end are : "round", "butt" or "square"

```
# Change the line type and color of axis lines
p + theme( axis.line = element_line(colour = "darkblue",
                      size = 1, linetype = "solid"))
```



24.6 Set axis ticks for discrete and continuous axes

x or y axis can be discrete or continuous. In each of these two cases, the functions to be used for setting axis ticks are different.

24.6.1 Customize a discrete axis

The functions **scale_x_discrete()** and **scale_y_discrete()** are used to customize discrete x and y axis, respectively.

It is possible to use these functions to change the following x or y axis parameters :

- axis titles
- axis limits (data range to display)
- choose where tick marks appear
- manually label tick marks

The simplified formats of **scale_x_discrete()** and **scale_y_discrete()** are :

```
scale_x_discrete(name, breaks, labels, limits)

scale_y_discrete(name, breaks, labels, limits)
```

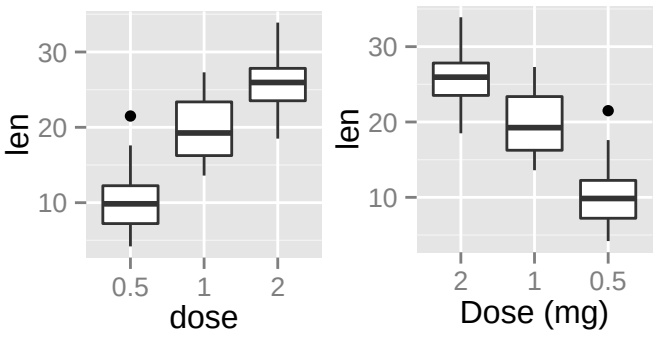
- **name** : x or y axis labels
- **breaks** : control the breaks in the guide (axis ticks, grid lines, . . .). Among the possible values, there are :
  - *NULL* : hide all breaks
  - **waiver()** : the default break computation
  - a **character** or **numeric** vector specifying which breaks to display
- **labels** : labels of axis tick marks. Allowed values are :
  - **NULL** for no labels
  - **waiver()** for the default labels
  - **character vector** to be used for break labels
- **limits** : a character vector indicating the data range

Note that, in the examples below, we'll use only the functions **scale_x_discrete()** and **xlim()** to customize x axis tick marks. The same kind of examples can be applied to a discrete y axis using the functions **scale_y_discrete()** and **ylim()**.

24.6.1.1 Change the order of items

The argument *limits* is used to change the order of the items :

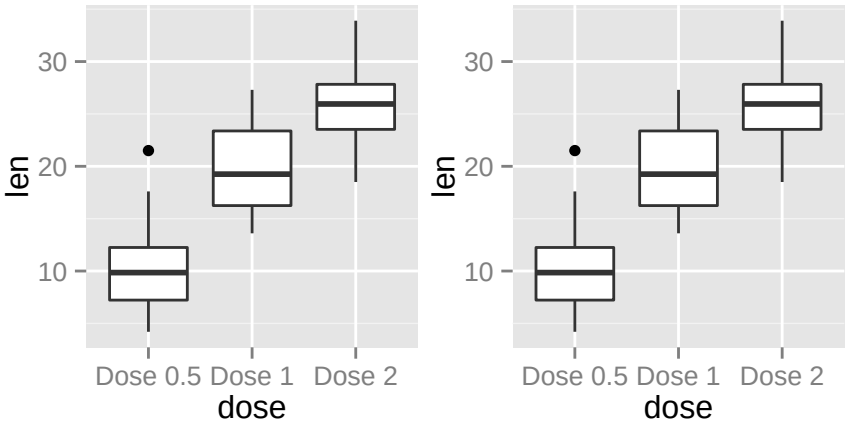
```
# default plot
p

# Change the order of items
# Change the x axis name
p + scale_x_discrete(name ="Dose (mg)",
                     limits=c("2","1","0.5"))
```



24.6.1.2 Change tick mark labels

The name of tick mark texts can be changed as follow :

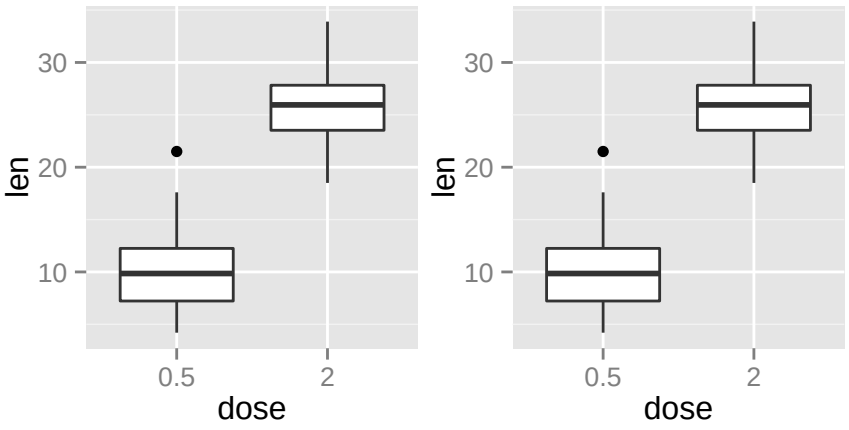
```
# Solution 1
p + scale_x_discrete(breaks=c("0.5","1","2"),
        labels=c("Dose 0.5", "Dose 1", "Dose 2"))

# Solution 2 : same plot as solution 1
p + scale_x_discrete(labels=c("0.5" = "Dose 0.5", "1" = "Dose 1",
                              "2" = "Dose 2"))
```



24.6.1.3 Choose which items to display

The R code below shows the box plot for the first item (dose = 0.5) and the last item (dose = 2) :

```
# Solution 1
p + scale_x_discrete(limits=c("0.5", "2"))

# Solution 2 : same result as solution 1
p + xlim("0.5", "2")
```



24.6.2 Customize a continuous axis

The functions **scale_x_continuous()** and **scale_y_continuous()** are used to customize continuous x and y axis, respectively.

Using these two functions, the following x or y axis parameters can be modified :

- axis titles
- axis limits (set the minimum and the maximum)
- choose where tick marks appear
- manually label tick marks

The simplified formats of **scale_x_continuous()** and **scale_y_continuous()** are :

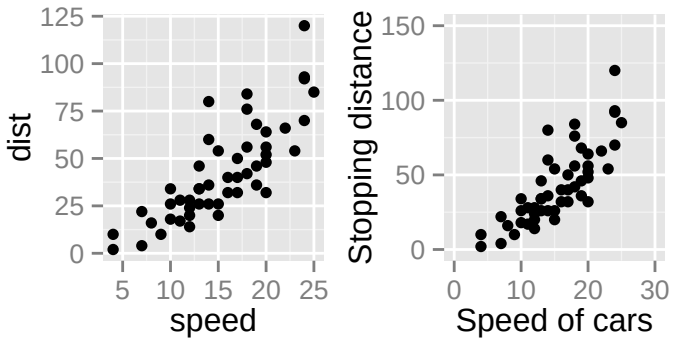
```
scale_x_continuous(name, breaks, labels, limits, trans)

scale_y_continuous(name, breaks, labels, limits, trans)
```

- **name** : x or y axis labels
- **breaks** : control the breaks in the guide (axis ticks, grid lines, . . .). Among the possible values, there are :
  - *NULL* : hide all breaks
  - **waiver()** : the default break computation
  - a **character** or **numeric** vector specifying the breaks to display
- **labels** : labels of axis tick marks. Allowed values are :
  - **NULL** for no labels
  - **waiver()** for the default labels
  - **character vector** to be used for break labels
- **limits** : a numeric vector specifying x or y axis limits (min, max)

- **trans** for axis transformations. Possible values are “log2”, “log10”, “sqrt”, etc

These functions can be used as follow :

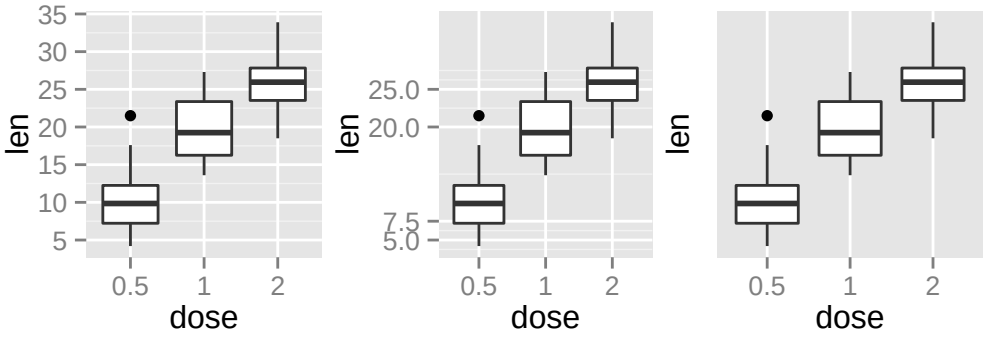
```
# scatter plot
sp<-ggplot(cars, aes(x = speed, y = dist)) + geom_point()
sp

# Change x and y axis labels, and limits
sp + scale_x_continuous(name="Speed of cars", limits=c(0, 30)) +
  scale_y_continuous(name="Stopping distance", limits=c(0, 150))
```



24.6.2.1 Set the position of tick marks

The R code below set the position of tick marks on the y axis of the box plot. The function **scale_y_continuous()** and the argument **breaks** are used to choose where the tick marks appear :

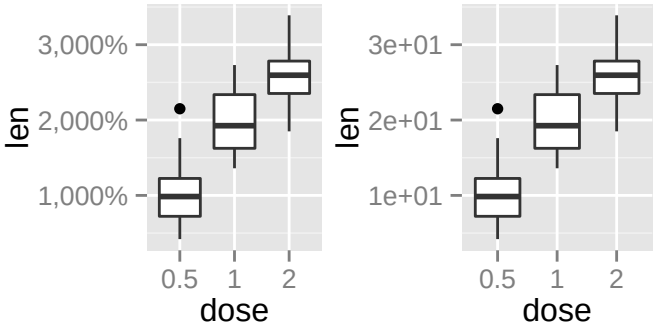
```
# Set tick marks on y axis
# a tick mark is shown on every 5
p + scale_y_continuous(breaks=seq(0,40,5))

# Tick marks can be spaced randomly
p + scale_y_continuous(breaks=c(5,7.5, 20, 25))

# Remove tick mark labels and gridlines
p + scale_y_continuous(breaks=NULL)
```



24.6.2.2 Format the text of tick mark labels

Tick mark labels can be formatted to be viewed as percents, dollars or scientific notation. The package **scales** is required.

```
library(scales)

# Format labels as percents
p + scale_y_continuous(labels = percent)

# Format labels as scientific
p + scale_y_continuous(labels = scientific)
```

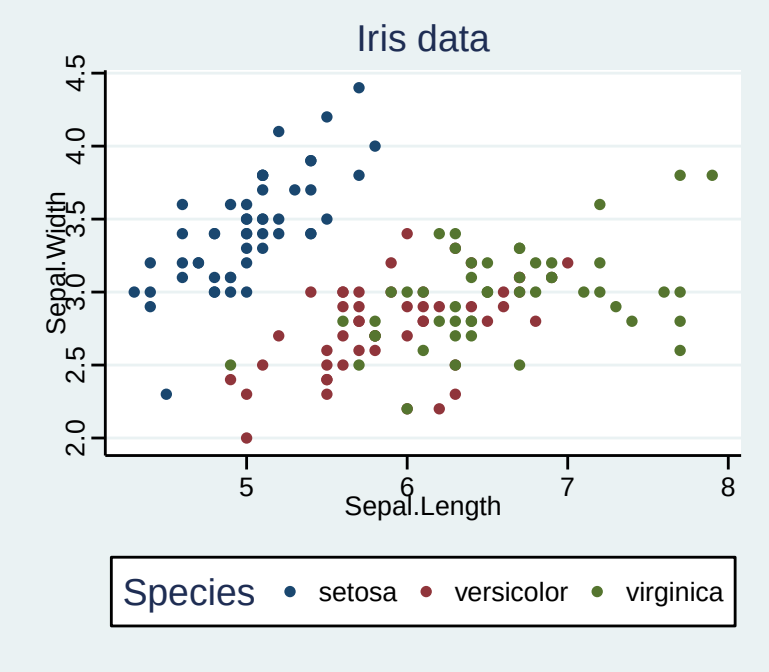


Possible values for labels are comma, percent, dollar and scientific. For more examples, read the documentation of the package *scales* : ?scales::trans_new

Chapter 25

Themes and background colors

This chapter describes how to change the look of a plot **theme** (**background color**, **panel background color** and **grid lines**) using **ggplot2** package. You'll also learn how to use the base themes of ggplot2 and to create your own theme.



25.1 Data

ToothGrowth data is used :

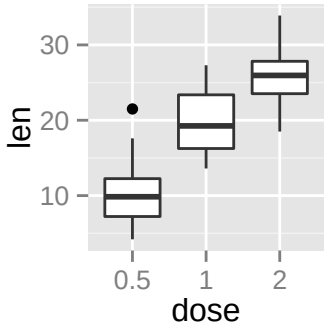
```
# Convert the column dose from numeric to factor variable
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that the variable *dose* is converted as a factor using the above R script.

25.2 Example of plot

```
library(ggplot2)
p <- ggplot(ToothGrowth, aes(x=dose, y=len)) + geom_boxplot()
p
```

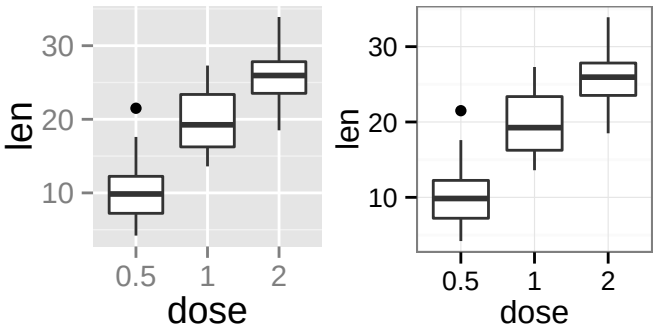


25.3 Quick functions to change plot themes

Several functions are available in ggplot2 package for changing quickly the theme of plots :

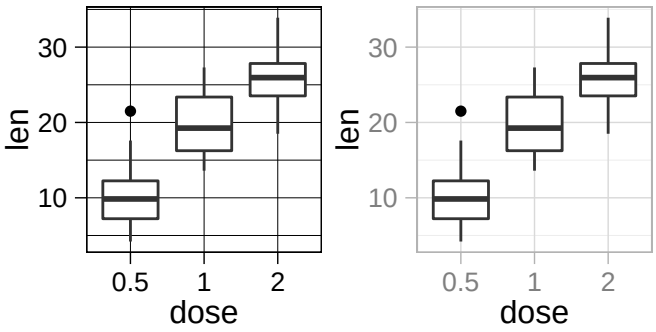
- **theme_gray** : gray background color and white grid lines
- **theme_bw** : white background and gray grid lines

```
p + theme_gray(base_size = 14)

p + theme_bw()
```



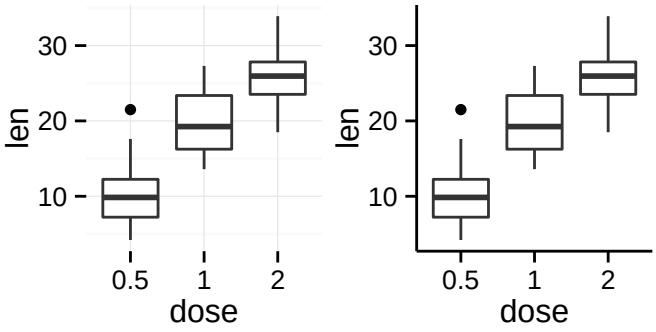
- **theme_linedraw** : black lines around the plot
- **theme_light** : light gray lines and axis (more attention towards the data)

```
p + theme_linedraw()

p + theme_light()
```



- **theme_minimal**: no background annotations
- **theme_classic** : theme with axis lines and no grid lines

```
p + theme_minimal()

p + theme_classic()
```

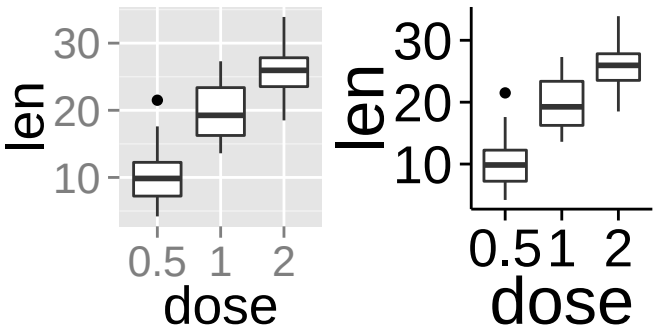


The functions theme_xx() can take the two arguments below :

- **base_size** : base font size (to change the size of all plot text elements)
- **base_family** : base font family

The size of all the plot text elements can be easily changed at once :

```
# Example 1
theme_set(theme_gray(base_size = 20))
ggplot(ToothGrowth, aes(x=dose, y=len)) + geom_boxplot()

# Example 2
ggplot(ToothGrowth, aes(x=dose, y=len)) + geom_boxplot()+
  theme_classic(base_size = 25)
```



Note that, the function **theme_set()** changes the theme for the entire session.

25.4 Customize the appearance of the plot background

The function **theme()** is used to control non-data parts of the graph including :

- **Line elements** : axis lines, minor and major grid lines, plot panel border, axis ticks background color, etc.
- **Text elements** : plot title, axis titles, legend title and text, axis tick mark labels, etc.
- **Rectangle elements** : plot background, panel background, legend background, etc.

There is a specific function to modify each of these three elements :

- **element_line()** to modify the line elements of the theme
- **element_text()** to modify the text elements
- **element_rect()** to change the appearance of the rectangle elements

Note that, each of the theme elements can be removed using the function **element_blank()**

25.4.1 Change the colors of the plot panel background and the grid lines

1. The functions **theme()** and **element_rect()** are used for changing the plot panel background color :

```
p + theme(panel.background = element_rect(fill, colour, size,
                                          linetype, color))
```

- **fill** : the fill color for the rectangle
- **colour**, **color** : border color
- **size** : border size

2. The appearance of **grid lines** can be changed using the function **element_line()** as follow :

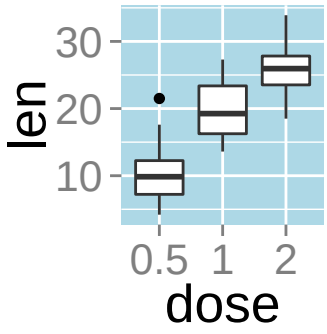
```
# change major and minor grid lines
p + theme(
  panel.grid.major = element_line(colour, size, linetype,
                                  lineend, color),
  panel.grid.minor = element_line(colour, size, linetype,
                                  lineend, color)
  )
```

- **colour**, **color** : line color
- **size** : line size
- **linetype** : line type. **Line type** can be specified using either text ("blank", "solid", "dashed", "dotted", "dotdash", "longdash", "twodash") or number (0, 1, 2, 3, 4, 5, 6). Note that linetype = "solid" is identical to linetype=1. The available line types in R are described here : Line types in R software
- **lineend** : line end. Possible values for line end are : "round", "butt" or "square"

The R code below illustrates how to modify the appearance of the plot panel background and grid lines :

```
# Change the colors of plot panel background to lightblue
# and the color of grid lines to white
p + theme(
  panel.background = element_rect(fill = "lightblue",
                                  colour = "lightblue",
                                  size = 0.5, linetype = "solid"),
  panel.grid.major = element_line(size = 0.5, linetype = 'solid',
```

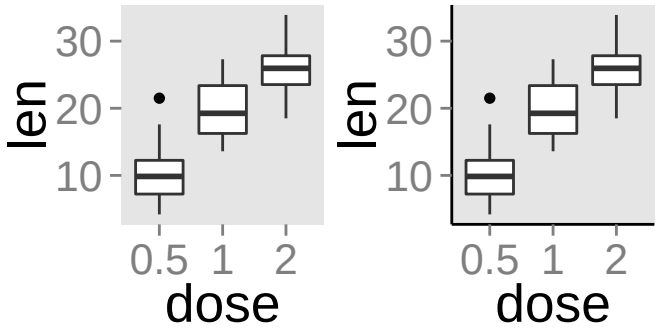
```
                                colour = "white"),
  panel.grid.minor = element_line(size = 0.25, linetype = 'solid',
                                colour = "white")
  )
```



25.4.2 Remove plot panel borders and grid lines

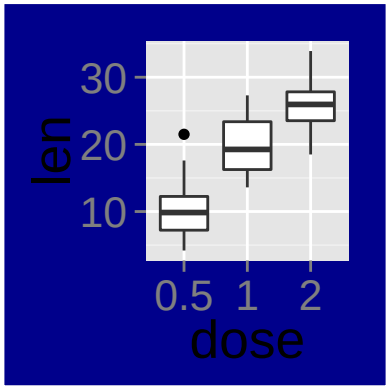
It is possible to hide plot panel borders and grid lines with the function **element_blank()** as follow :

```
# Remove panel borders and grid lines
p + theme(panel.border = element_blank(),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank())

# Hide panel borders and grid lines
# But change axis line
p + theme(panel.border = element_blank(),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          axis.line = element_line(size = 0.5, linetype = "solid",
                                   colour = "black"))
```



25.4.3 Change the plot background color (not the panel)

```
p + theme(plot.background = element_rect(fill = "darkblue"))
```



25.5 Use a custom theme

You can change the entire appearance of a plot by using a custom theme. Jeffrey Arnold has implemented the library **ggthemes** containing several custom themes.

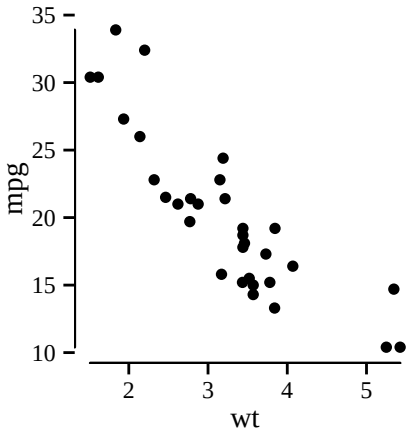
To use these themes install and load ggthemes package as follow :

```
install.packages("ggthemes") # Install
library(ggthemes) # Load
```

ggthemes package provides many custom themes and scales for ggplot.

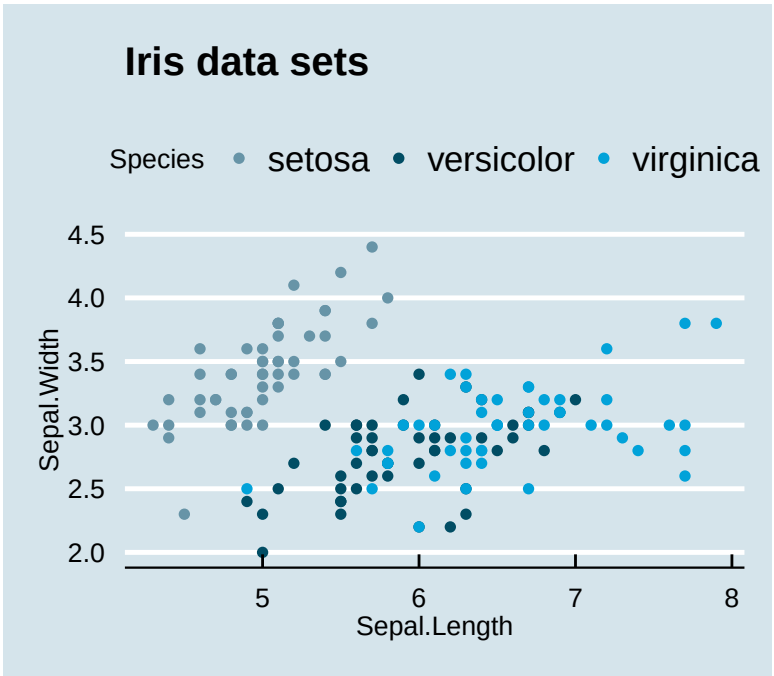
25.5.1 theme_tufte : a minimalist theme

```
# scatter plot
ggplot(mtcars, aes(wt, mpg)) +
  geom_point() + geom_rangeframe() +
  theme_tufte()
```



25.5.2 theme_economist : theme based on the plots in the economist magazine

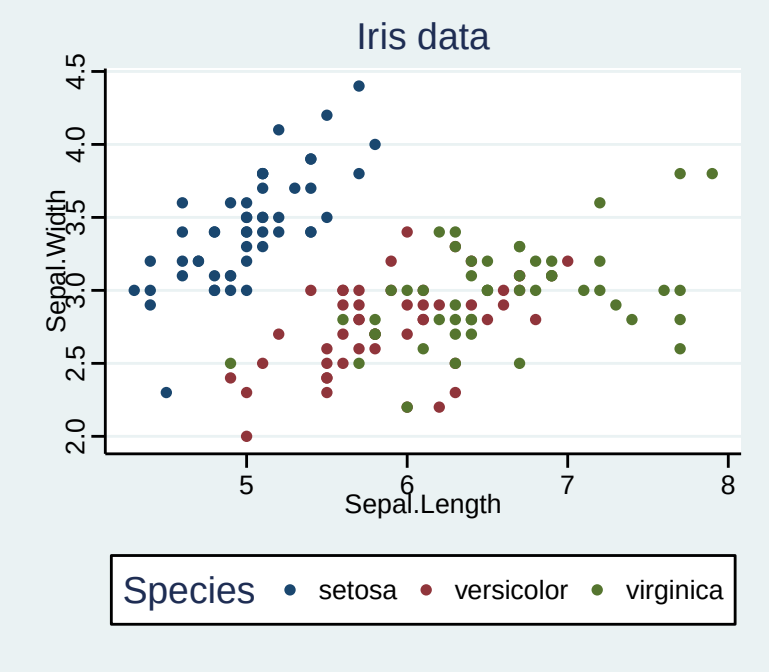
```
p <- ggplot(iris, aes(Sepal.Length, Sepal.Width, colour = Species))+
  geom_point()

# Use economist color scales
p + theme_economist() +
  scale_color_economist()+
  ggtitle("Iris data sets")
```



Note that, the function **scale_fill_economist()** are also available.

25.5.3 theme_stata: theme based on Stata graph schemes.

```
p + theme_stata() + scale_color_stata() +
  ggtitle("Iris data")
```



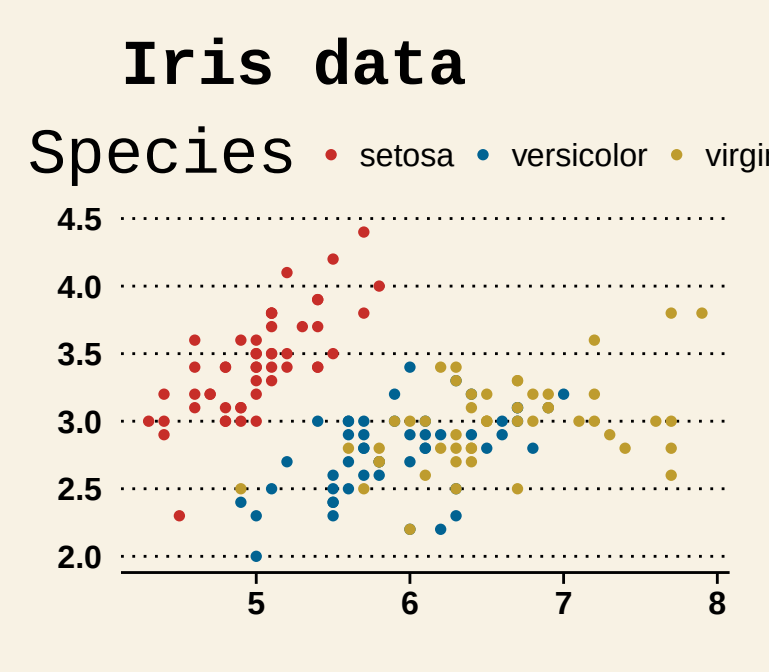
The stata theme color scales can be used as follow :

```
scale_fill_stata(scheme = "s2color", ...)

scale_color_stata(scheme = "s2color", ...)
```

The allowed values for the argument *scheme* are one of “s2color”, “s1rcolor”, “s1color”, or “mono”.

25.5.4 theme_wsj: theme based on plots in the Wall Street Journal

```
p + theme_wsj()+ scale_colour_wsj("colors6")+
  ggtitle("Iris data")
```



The Wall Street Journal color and fill scales are :

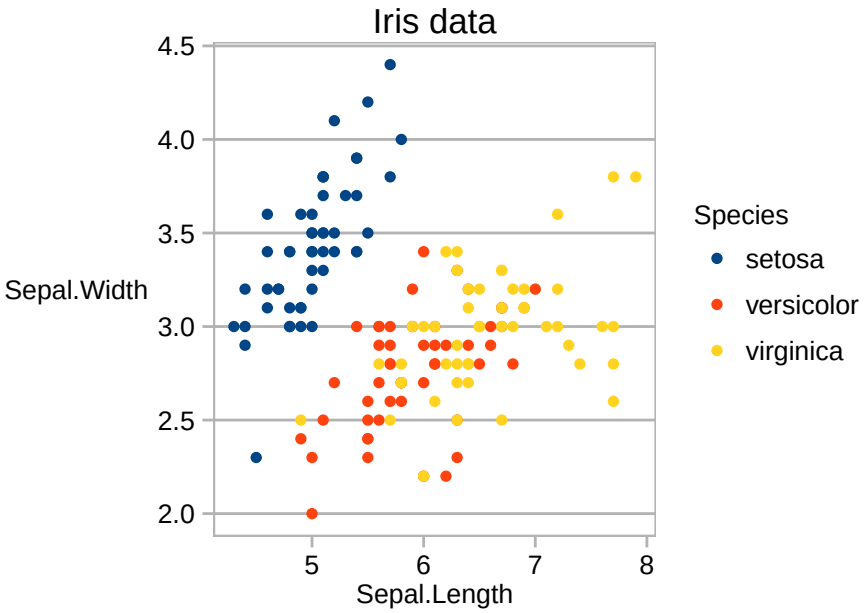
```
scale_color_wsj(palette = "colors6", ...)

scale_fill_wsj(palette = "colors6", ...)
```

The color palette to use can be one of “rgby”, “red_green”, “black_green”, “dem_rep”, “colors6”.

25.5.5 theme_calc : theme based on LibreOffice Calc

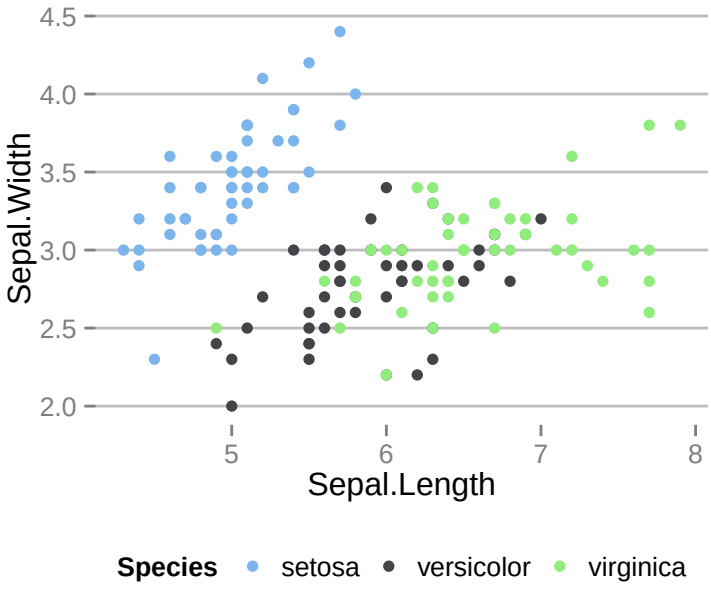
These themes are based on the defaults in Google Docs and LibreOffice Calc, respectively.

```
p + theme_calc()+ scale_colour_calc()+
  ggtitle("Iris data")
```



25.5.6 theme_hc : theme based on Highcharts JS

```
p + theme_hc()+ scale_colour_hc()
```



25.6 Create a custom theme

1. You can change the theme for the current R session using the function theme_set() as follow :

```
theme_set(theme_gray(base_size = 20))
```

2. You can extract and modify the R code of theme_gray :

```
theme_gray

function (base_size = 12, base_family = "")
{
theme(
    line = element_line(colour = "black", size = 0.5,
                        linetype = 1, lineend = "butt"),

    rect = element_rect(fill = "white",  colour = "black",
                        size = 0.5, linetype = 1),
    text = element_text(family = base_family, face = "plain",
                        colour = "black", size = base_size,
                        hjust = 0.5, vjust = 0.5, angle = 0,
                        lineheight = 0.9),

    axis.text = element_text(size = rel(0.8), colour = "grey50"),
    strip.text = element_text(size = rel(0.8)),
    axis.line = element_blank(),
    axis.text.x = element_text(vjust = 1),
    axis.text.y = element_text(hjust = 1),
    axis.ticks = element_line(colour = "grey50"),
    axis.title.x = element_text(),
    axis.title.y = element_text(angle = 90),
    axis.ticks.length = unit(0.15, "cm"),
    axis.ticks.margin = unit(0.1, "cm"),

    legend.background = element_rect(colour = NA),
    legend.margin = unit(0.2, "cm"),
    legend.key = element_rect(fill = "grey95", colour = "white"),
    legend.key.size = unit(1.2, "lines"),
    legend.key.height = NULL,
    legend.key.width = NULL,
    legend.text = element_text(size = rel(0.8)),
    legend.text.align = NULL,
    legend.title = element_text(size = rel(0.8), face = "bold",hjust=0),
    legend.title.align = NULL,
    legend.position = "right",
    legend.direction = NULL,
    legend.justification = "center",
    legend.box = NULL,
```

```
    panel.background = element_rect(fill = "grey90", colour = NA),
    panel.border = element_blank(),
    panel.grid.major = element_line(colour = "white"),
    panel.grid.minor = element_line(colour = "grey95", size = 0.25),
    panel.margin = unit(0.25, "lines"),
    panel.margin.x = NULL,
    panel.margin.y = NULL,

    strip.background = element_rect(fill = "grey80", colour = NA),
    strip.text.x = element_text(),
    strip.text.y = element_text(angle = -90),

    plot.background = element_rect(colour = "white"),
    plot.title = element_text(size = rel(1.2)),
    plot.margin = unit(c(1, 1, 0.5, 0.5), "lines"), complete = TRUE)
}
```

Note that, the function **rel()** modifies the size relative to the base size

Chapter 26

Rotate a graph

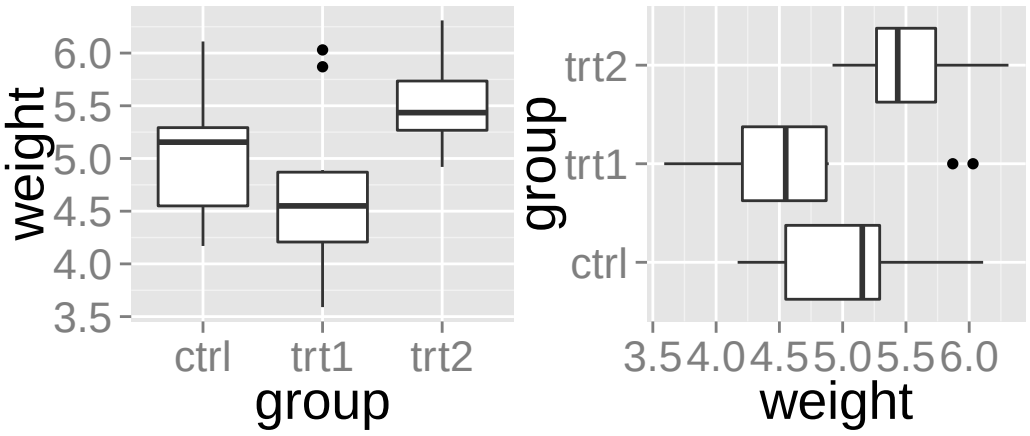
This chapter describes how to *rotate* a plot created using **R software** and **ggplot2** package. The functions are :

- coord_flip() to create horizontal plots
- scale_x_reverse(), scale_y_reverse() to reverse the axes

26.1 Horizontal plot : coord_flip()

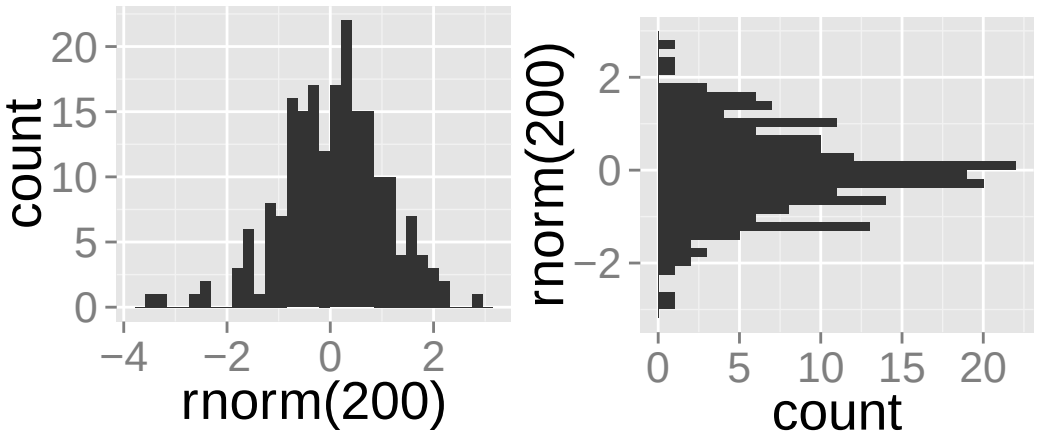
Box plot :

```
library(ggplot2)

# Basic box plot
bp <- ggplot(PlantGrowth, aes(x=group, y=weight))+
  geom_boxplot()
bp

# Horizontal box plot
bp + coord_flip()
```



Histogram :

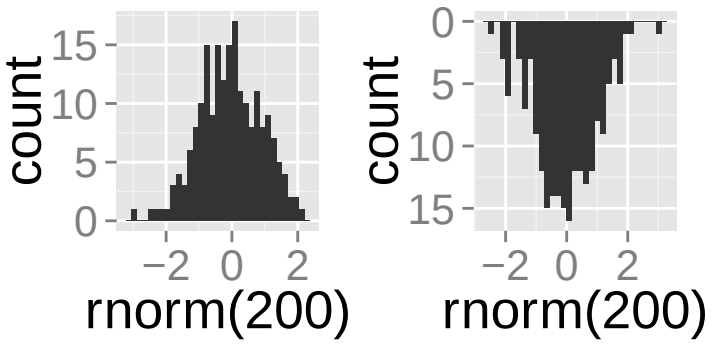
```
set.seed(1234)
# Basic histogram
hp <-qplot(x=rnorm(200), geom="histogram")
hp

# Horizontal histogram
hp + coord_flip()
```



26.2 Reverse y axis

The function scale_y_reverse() can be used as follow :

```
# Basic histogram
hp

# Y axis reversed
hp + scale_y_reverse()
```


count
15
10
5
0
−2
0
2
rnorm(200)
count
0
5
10
15
−2
0
2
rnorm(200)

Chapter 27

Facets: split a plot into a matrix of panels

The **facet** approach partitions a plot into a matrix of panels. Each panel shows a different subset of the data.

There are two main functions for faceting :

- facet_grid()
- facet_wrap()

27.1 Data

ToothGrowth data is used in the following examples.

```
# Convert dose from numeric to factor variables
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
df <- ToothGrowth
head(df)
```

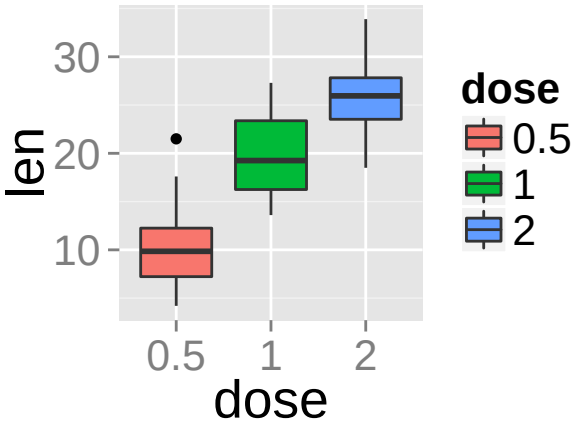
```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Make sure that the variable *dose* is converted as a factor using the above R script.

27.2 Basic box plot

Create a basic box plot filled by groups :

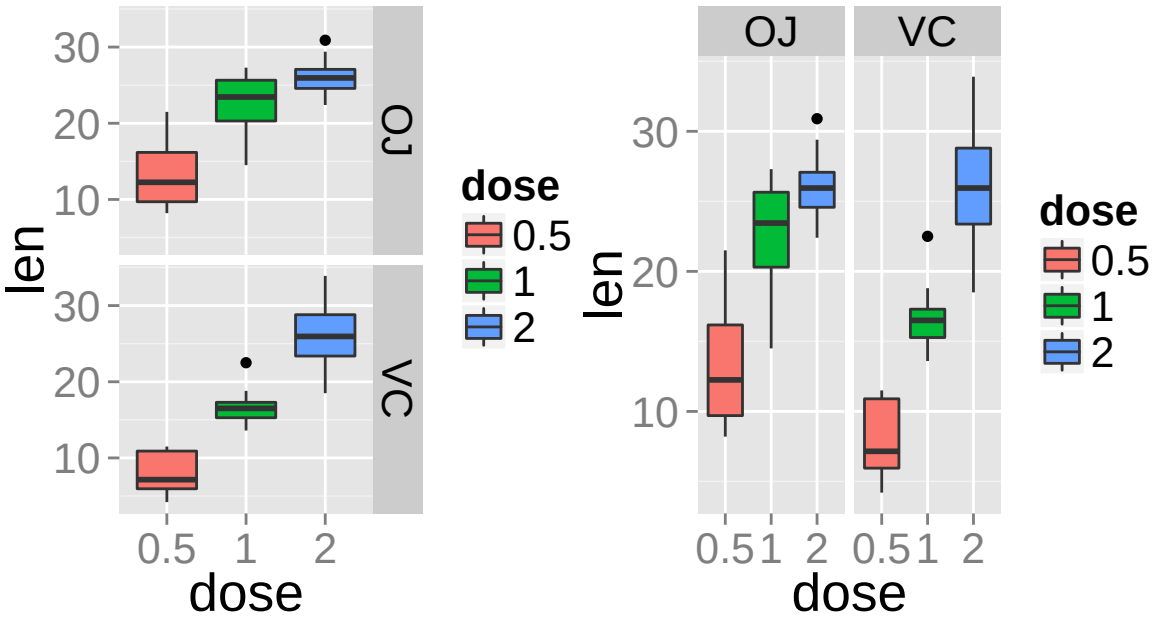
```
library(ggplot2)
bp <- ggplot(df, aes(x=dose, y=len, group=dose)) +
  geom_boxplot(aes(fill=dose))
bp
```



27.3 Facet with one variable

The graph is partitioned in multiple panels by levels of the group “supp”:

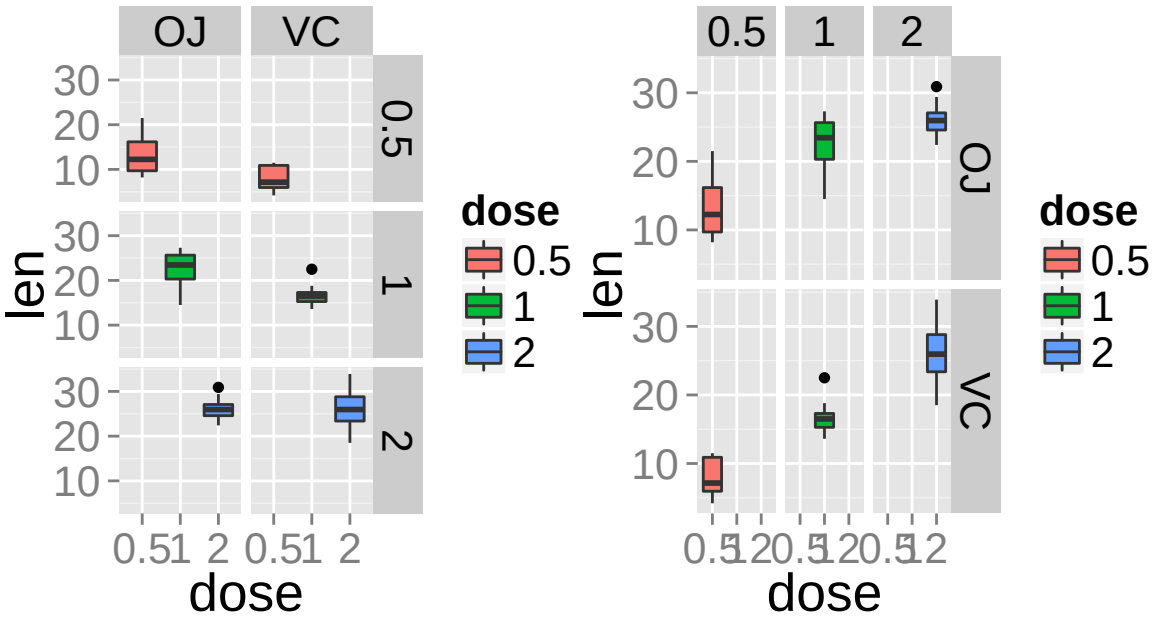
```
# Split in vertical direction
bp + facet_grid(supp ~ .)

# Split in horizontal direction
bp + facet_grid(. ~ supp)
```



27.4 Facet with two variables

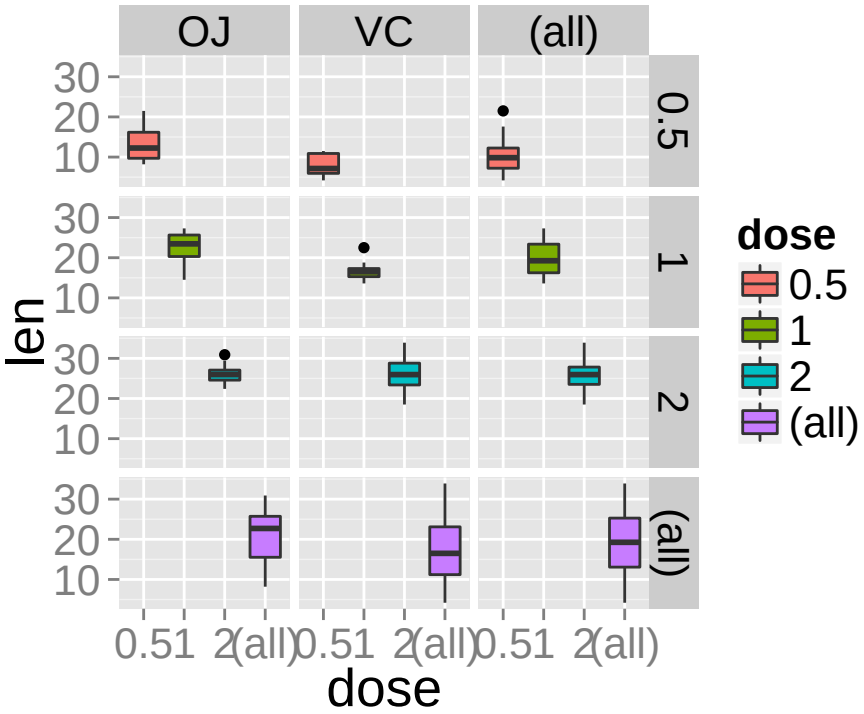
The graph is partitioned by the levels of the groups "dose" and "supp" :

```
# Facet by two variables: dose and supp.
# Rows are dose and columns are supp
bp + facet_grid(dose ~ supp)

# Facet by two variables: reverse the order of the 2 variables
# Rows are supp and columns are dose
bp + facet_grid(supp ~ dose)
```



Note that, you can use the argument *margins* to add additional facets which contain all the data for each of the possible values of the faceting variables

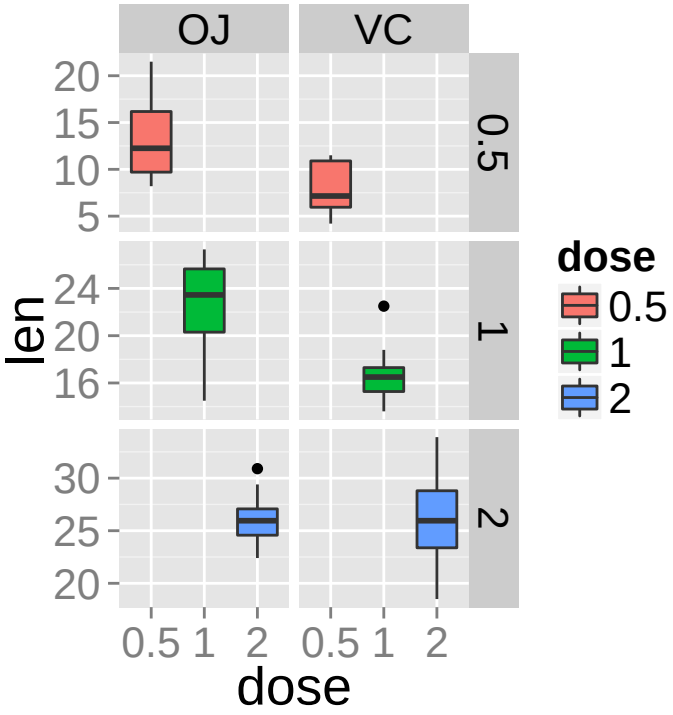
```
bp + facet_grid(dose ~ supp, margins=TRUE)
```



27.5 Facet scales

By default, all the panels have the same scales (`scales="fixed"`). They can be made independent, by setting scales to `free`, `free_x`, or `free_y`.

```
bp + facet_grid(dose ~ supp, scales='free')
```

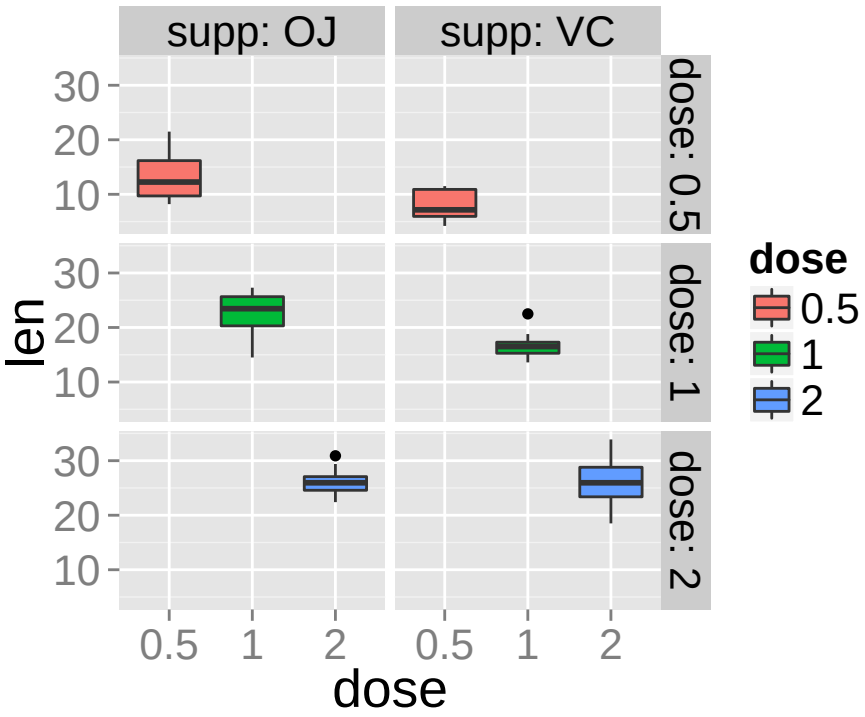


As you can see in the above plot, y axis have different scales in the different panels.

27.6 Facet labels

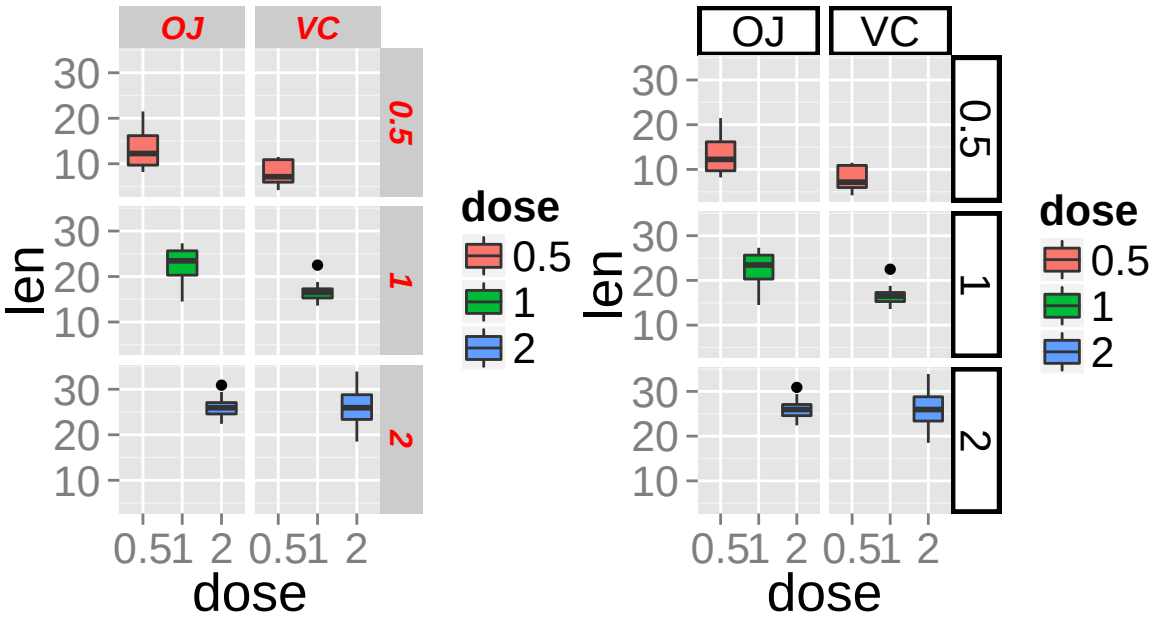
The argument *labeller* can be used to control the labels of the panels :

```
bp + facet_grid(dose ~ supp, labeller=label_both)
```



The appearance of facet labels can be modified as follow :

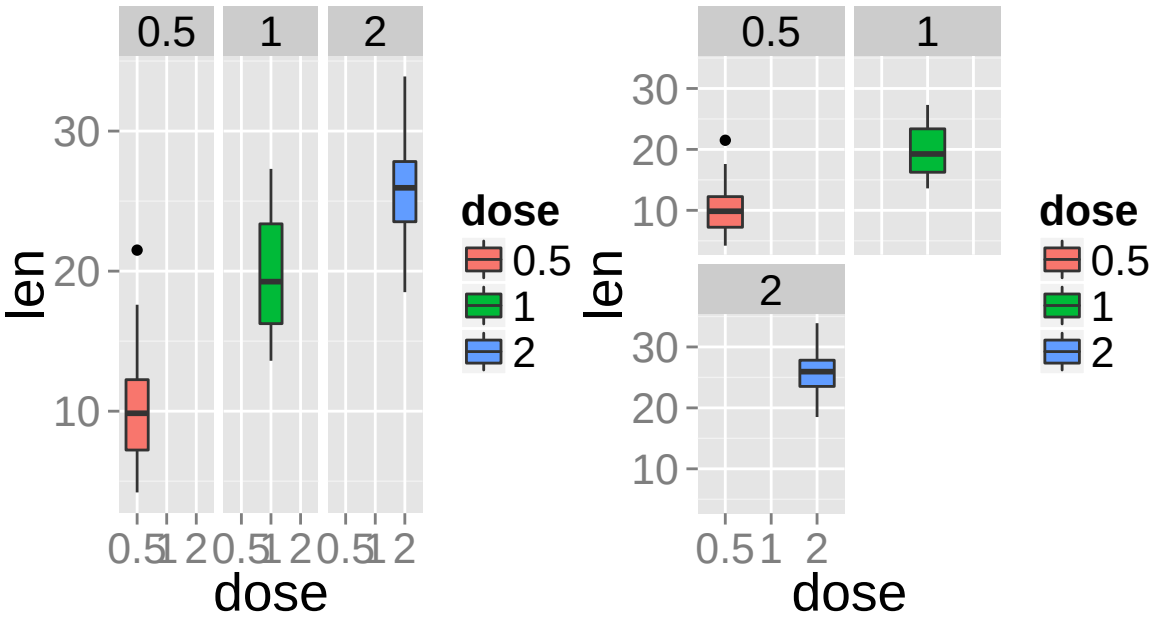
```
# Change facet text font. Possible values for the font style:
  #'plain', 'italic', 'bold', 'bold.italic'.
bp + facet_grid(dose ~ supp)+
    theme(strip.text.x = element_text(size=12, color="red",
                                      face="bold.italic"),
          strip.text.y = element_text(size=12, color="red",
                                      face="bold.italic"))

# Change the apperance of the rectangle around facet label
bp + facet_grid(dose ~ supp)+
 theme(strip.background = element_rect(colour="black", fill="white",
                                       size=1.5, linetype="solid"))
```



27.7 facet_wrap

Facets can be placed side by side using the function **facet_wrap()** as follow :

```
bp + facet_wrap(~ dose)

bp + facet_wrap(~ dose, ncol=2)
```



Part III

ggplot2 extensions

Chapter 28

Mix multiple graphs on the same page

To **arrange multiple ggplot2 graphs on the same page**, the standard R functions - par() and layout() - cannot be used.

This chapter will show you, step by step, how to put several ggplots on a single page.

The functions **grid.arrange()**[in the package **gridExtra**] and **plot_grid()**[in the package **cowplot**], will be used.

28.1 Install and load required packages

28.1.1 Install and load the package gridExtra

```
install.packages("gridExtra")
library("gridExtra")
```

28.1.2 Install and load the package cowplot

cowplot can be installed as follow:

```
install.packages("cowplot")
```

OR

as follow using devtools package (devtools should be installed before using the code below):

```
devtools::install_github("wilkelab/cowplot")
```

Load **cowplot**:

```
library("cowplot")
```

28.2 Data

ToothGrowth data is used :

```
df <- ToothGrowth
# Convert the variable dose from a numeric to a factor variable
df$dose <- as.factor(df$dose)
head(df)
```

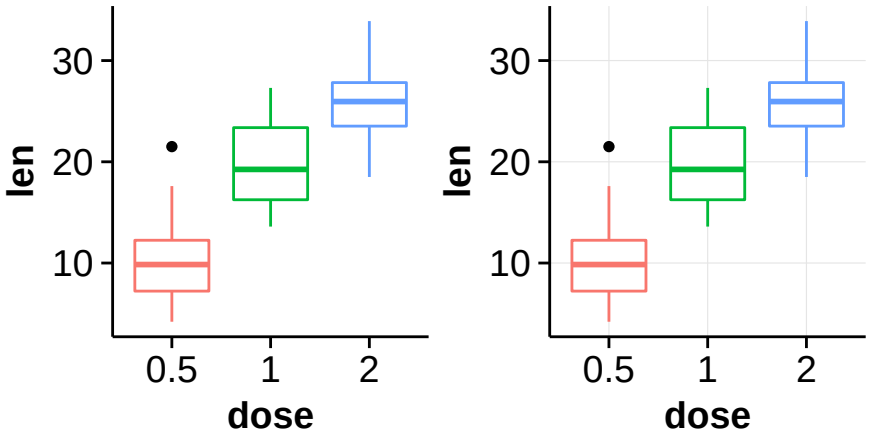
```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

28.3 Cowplot: Publication-ready plots

The **cowplot** package is an extension to **ggplot2** and it can be used to provide a publication-ready plots.

28.3.1 Basic plots

```
library(cowplot)
# Default plot
bp <- ggplot(df, aes(x=dose, y=len, color=dose)) +
  geom_boxplot() +
  theme(legend.position = "none")
bp

# Add gridlines
bp + background_grid(major = "xy", minor = "none")
```

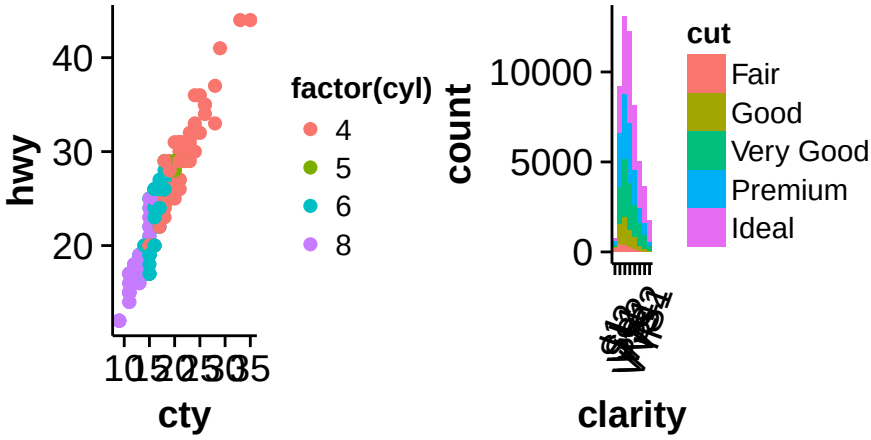


Recall that, the function **ggsave()**[in **ggplot2** package] can be used to save ggplots. However, when working with **cowplot**, the function **save_plot()** [in **cowplot** package] is preferred. It's an alternative to **ggsave** with a better support for multi-figur plots.

```
save_plot("mpg.pdf", plot.mpg,
          base_aspect_ratio = 1.3 # make room for figure legend
          )
```

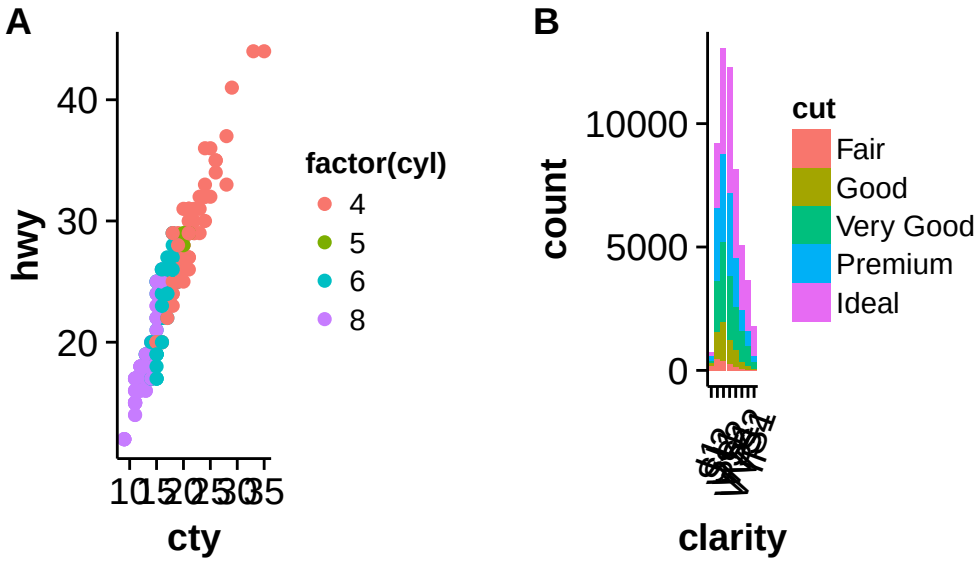
28.3.2 Arranging multiple graphs using cowplot

```
# Scatter plot
sp <- ggplot(mpg, aes(x = cty, y = hwy, colour = factor(cyl)))+
  geom_point(size=2.5)
sp

# Bar plot
bp <- ggplot(diamonds, aes(clarity, fill = cut)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle=70, vjust=0.5))
bp
```



Combine the two plots (the scatter plot and the bar plot):

```
plot_grid(sp, bp, labels=c("A", "B"), ncol = 2, nrow = 1)
```



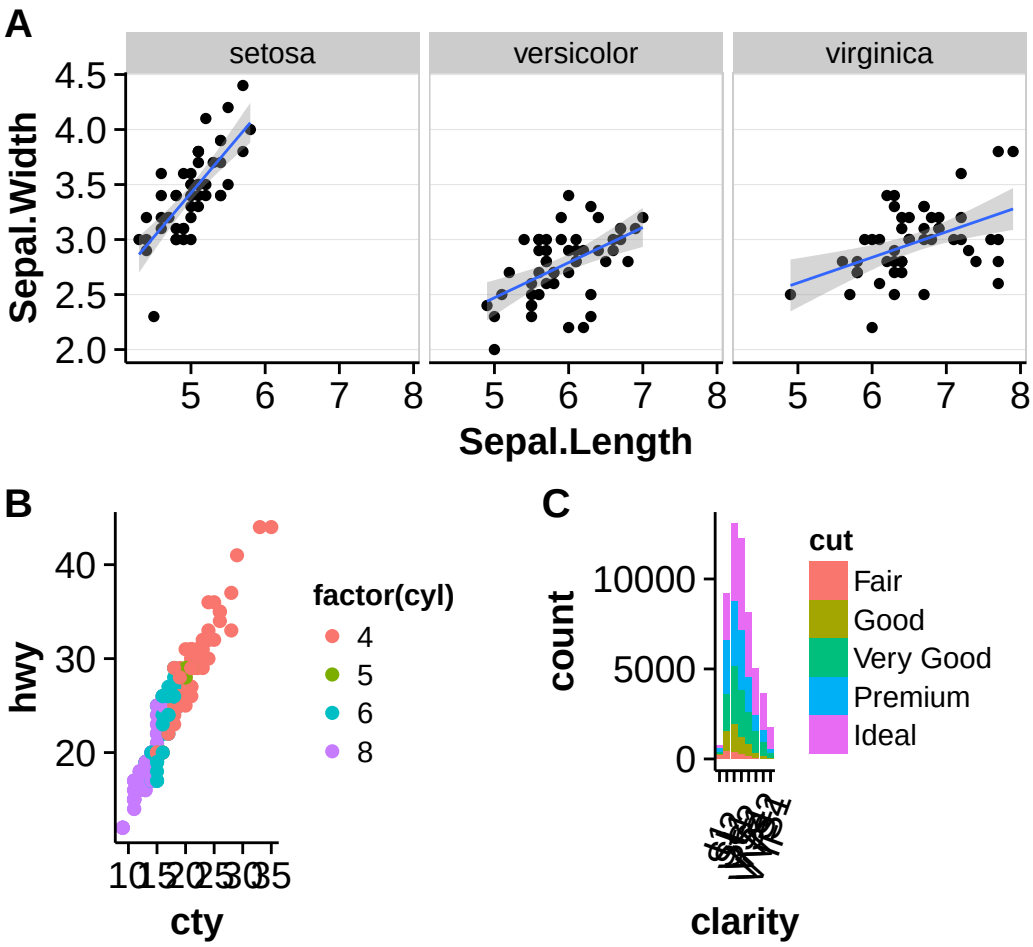
The function **draw_plot()** can be used to place graphs at particular locations with a particular sizes. The format of the function is:

```
draw_plot(plot, x = 0, y = 0, width = 1, height = 1)
```

- **plot**: the plot to place (ggplot2 or a gtable)
- **x**: The x location of the lower left corner of the plot.
- **y**: The y location of the lower left corner of the plot.
- **width, height**: the width and the height of the plot

The function **ggdraw()** is used to initialize an empty drawing canvas.

```
plot.iris <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point() + facet_grid(. ~ Species) + stat_smooth(method = "lm") +
  background_grid(major = 'y', minor = "none") + # add thin horizontal lines
  panel_border() # and a border around each panel
# plot.mpt and plot.diamonds were defined earlier
ggdraw() +
  draw_plot(plot.iris, 0, .5, 1, .5) +
  draw_plot(sp, 0, 0, .5, .5) +
  draw_plot(bp, .5, 0, .5, .5) +
  draw_plot_label(c("A", "B", "C"), c(0, 0, 0.5), c(1, 0.5, 0.5), size = 15)
```



28.4 grid.arrange: Create and arrange multiple plots

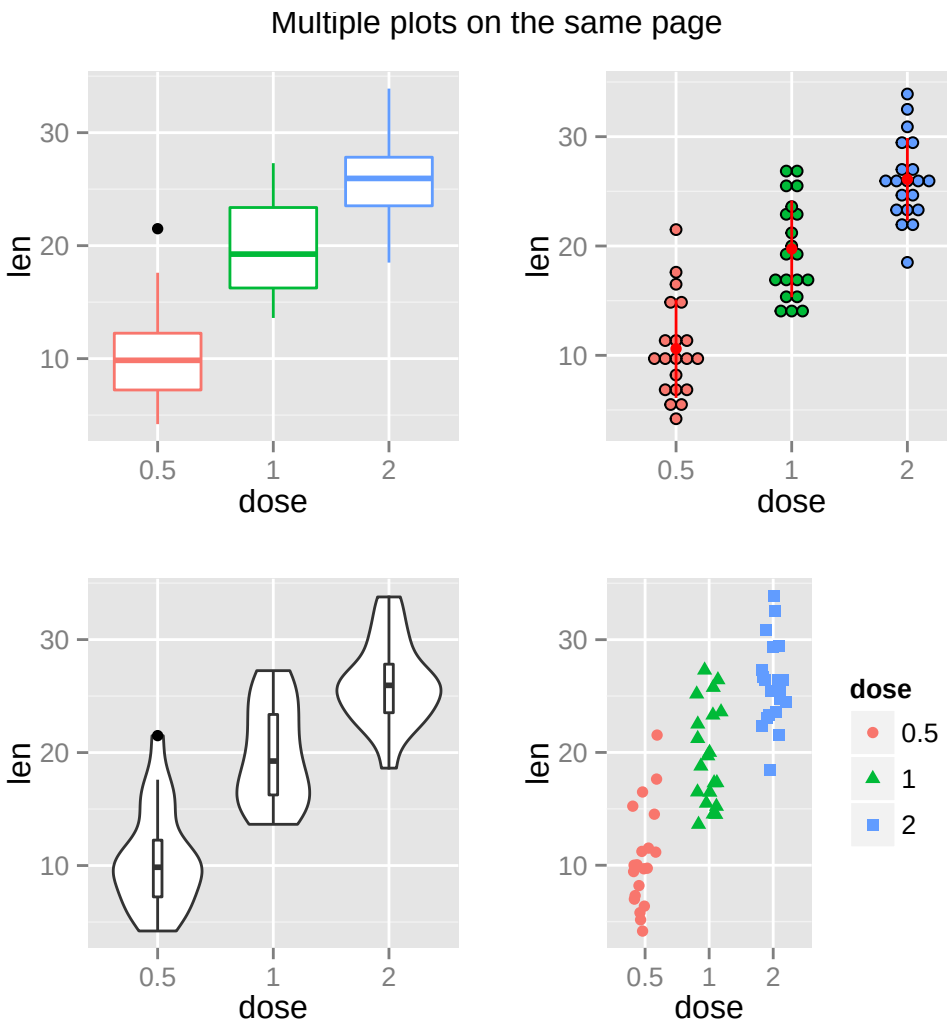
The R code below creates a box plot, a dot plot, a violin plot and a stripchart (jitter plot) :

```
library(ggplot2)
# Create a box plot
bp <- ggplot(df, aes(x=dose, y=len, color=dose)) +
  geom_boxplot() +
  theme(legend.position = "none")

# Create a dot plot
# Add the mean point and the standard deviation
dp <- ggplot(df, aes(x=dose, y=len, fill=dose)) +
  geom_dotplot(binaxis='y', stackdir='center')+
  stat_summary(fun.data=mean_sdl, mult=1,
                 geom="pointrange", color="red")+
   theme(legend.position = "none")
```

```
# Create a violin plot
vp <- ggplot(df, aes(x=dose, y=len)) +
  geom_violin()+
  geom_boxplot(width=0.1)

# Create a stripchart
sc <- ggplot(df, aes(x=dose, y=len, color=dose, shape=dose)) +
  geom_jitter(position=position_jitter(0.2))+
  theme(legend.position = "none") +
  theme_gray()
```

Combine the plots using the function **grid.arrange()** [in **gridExtra**] :

```
library(gridExtra)
grid.arrange(bp, dp, vp, sc, ncol=2,
             main="Multiple plots on the same page")
```



28.4.1 Add a common legend for multiple ggplot2 graphs

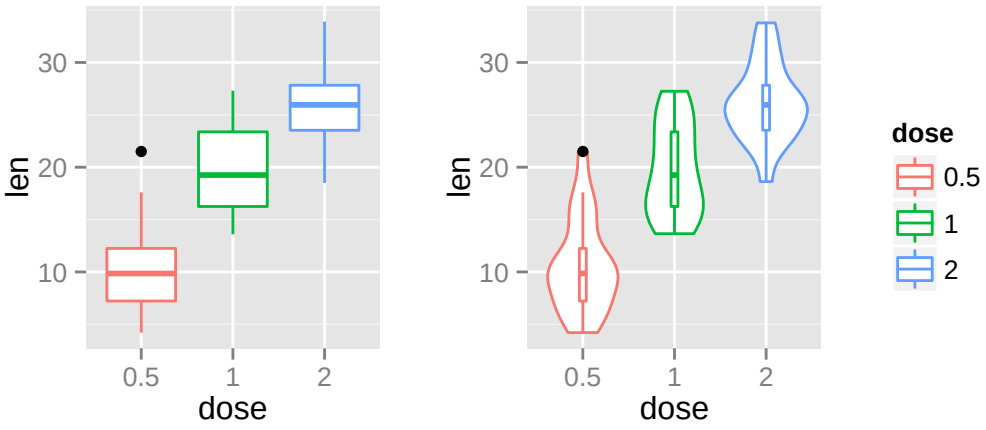
This can be done in four simple steps :

1. Create the plots : p1, p2, ...
2. Save the legend of the plot p1 as an external graphical element (called a "grob" in Grid terminology)
3. Remove the legends from all plots
4. Draw all the plots with only one legend in the right panel

To **save the legend** of a ggplot, the helper function below can be used :

```
library(gridExtra)
get_legend<-function(myggplot){
  tmp <- ggplot_gtable(ggplot_build(myggplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
  legend <- tmp$grobs[[leg]]
  return(legend)
}
```

```
# 1. Create the plots
#++++++++++++++++++++++++++++++++
# Create a box plot
bp <- ggplot(df, aes(x=dose, y=len, color=dose)) +
  geom_boxplot()

# Create a violin plot
vp <- ggplot(df, aes(x=dose, y=len, color=dose)) +
  geom_violin()+
  geom_boxplot(width=0.1)+
  theme(legend.position="none")

# 2. Save the legend
#++++++++++++++++++++++
legend <- get_legend(bp)

# 3. Remove the legend from the box plot
#++++++++++++++++++++++
bp <- bp + theme(legend.position="none")

# 4. Arrange ggplot2 graphs with a specific width
grid.arrange(bp, vp, legend, ncol=3, widths=c(2.3, 2.3, 0.8))
```



28.4.2 Scatter plot with marginal density plots

Step 1/3. Create some data :

```
set.seed(1234)
x <- c(rnorm(500, mean = -1), rnorm(500, mean = 1.5))
y <- c(rnorm(500, mean = 1), rnorm(500, mean = 1.7))
group <- as.factor(rep(c(1,2), each=500))
df2 <- data.frame(x, y, group)
head(df2)
```

```
##             x          y group
## 1 -2.20706575 -0.2053334     1
## 2 -0.72257076  1.3014667     1
## 3  0.08444118 -0.5391452     1
## 4 -3.34569770  1.6353707     1
## 5 -0.57087531  1.7029518     1
## 6 -0.49394411 -0.9058829     1
```

Step 2/3. Create the plots :

```
# Scatter plot of x and y variables and color by groups
scatterPlot <- ggplot(df2,aes(x, y, color=group)) +
  geom_point() +
  scale_color_manual(values = c('#999999','#E69F00')) +
  theme(legend.position=c(0,1), legend.justification=c(0,1))


# Marginal density plot of x (top panel)
xdensity <- ggplot(df2, aes(x, fill=group)) +
  geom_density(alpha=.5) +
```

```
  scale_fill_manual(values = c('#999999','#E69F00')) +
  theme(legend.position = "none")

# Marginal density plot of y (right panel)
ydensity <- ggplot(df2, aes(y, fill=group)) +
  geom_density(alpha=.5) +
  scale_fill_manual(values = c('#999999','#E69F00')) +
  theme(legend.position = "none")
```

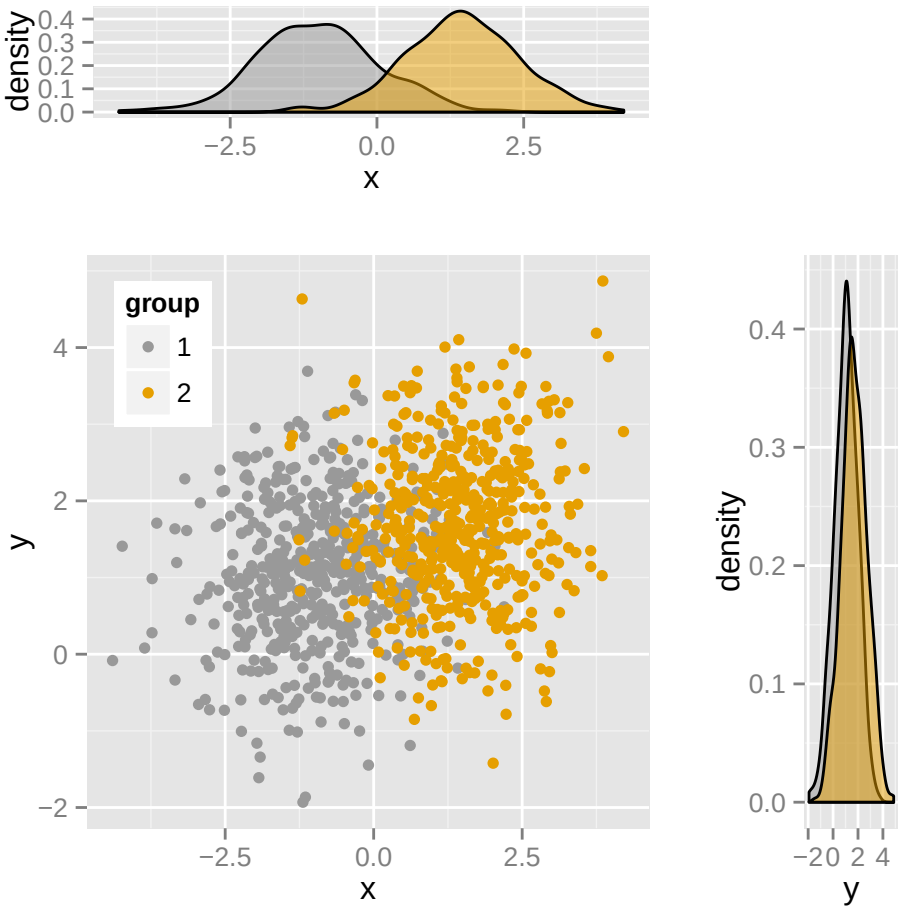
Create a blank placeholder plot :

```
blankPlot <- ggplot()+geom_blank(aes(1,1))+
  theme(
    plot.background = element_blank(),
   panel.grid.major = element_blank(),
   panel.grid.minor = element_blank(),
   panel.border = element_blank(),
   panel.background = element_blank(),
   axis.title.x = element_blank(),
   axis.title.y = element_blank(),
   axis.text.x = element_blank(),
   axis.text.y = element_blank(),
   axis.ticks = element_blank(),
   axis.line = element_blank()
     )
```

Step 3/3. Put the plots together:

Arrange ggplot2 with adapted height and width for each row and column :

```
library("gridExtra")
grid.arrange(xdensity, blankPlot, scatterPlot, ydensity,
        ncol=2, nrow=2, widths=c(4, 1.4), heights=c(1.4, 4))
```



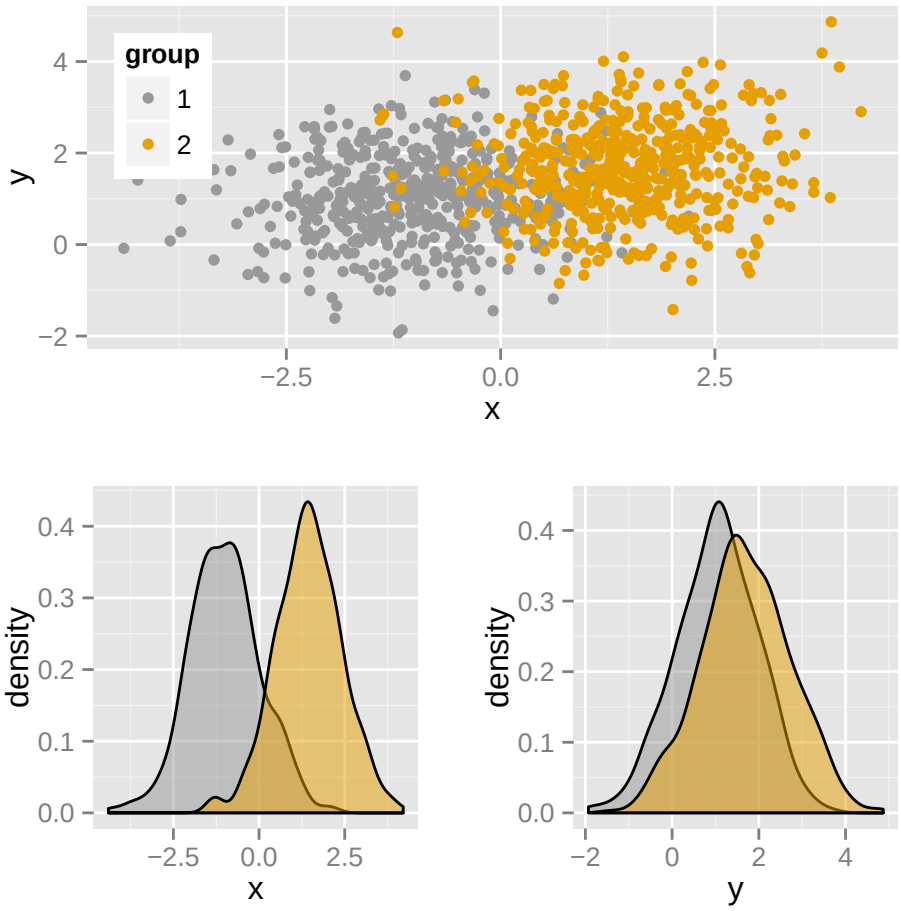
28.4.3 Create a complex layout using the function viewport()

The different steps are :

1. Create plots : p1, p2, p3, ...
2. Move to a new page on a grid device using the function **grid.newpage()**
3. Create a layout 2X2 - number of columns = 2; number of rows = 2
4. Define a grid viewport : a rectangular region on a graphics device
5. Print a plot into the viewport

```
# Move to a new page
grid.newpage()

# Create layout : nrow = 2, ncol = 2
pushViewport(viewport(layout = grid.layout(2, 2)))

# A helper function to define a region on the layout
define_region <- function(row, col){
  viewport(layout.pos.row = row, layout.pos.col = col)
```

```
}

# Arrange the plots
print(scatterPlot, vp=define_region(1, 1:2))
print(xdensity, vp = define_region(2, 1))
print(ydensity, vp = define_region(2, 2))
```



28.5 Insert an external graphical element inside a ggplot

The function **annotation_custom()** [in **ggplot2**] can be used for adding tables, plots or other grid-based elements. The simplified format is :

```
annotation_custom(grob, xmin, xmax, ymin, ymax)
```

- **grob**: the external graphical element to display
- **xmin, xmax** : x location in data coordinates (horizontal location)

- **ymin, ymax** : y location in data coordinates (vertical location)

The different steps are :

1. Create a scatter plot of y = f(x)
2. Add, for example, the box plot of the variables x and y inside the scatter plot using the function **annotation_custom()**

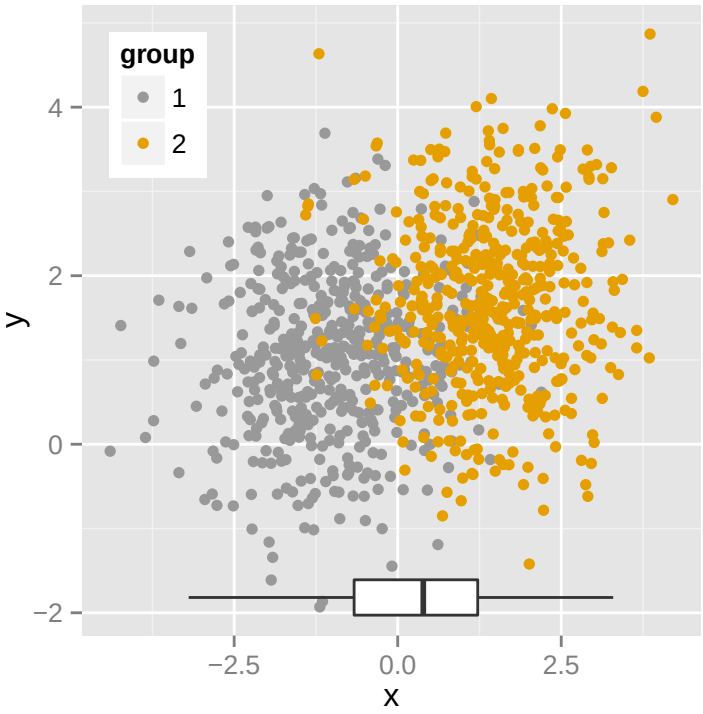
As the inset box plot overlaps with some points, a **transparent background** is used for the box plots.

```
# Create a transparent theme object
transparent_theme <- theme(
 axis.title.x = element_blank(),
 axis.title.y = element_blank(),
 axis.text.x = element_blank(),
 axis.text.y = element_blank(),
 axis.ticks = element_blank(),
 panel.grid = element_blank(),
 axis.line = element_blank(),
 panel.background = element_rect(fill = "transparent",colour = NA),
 plot.background = element_rect(fill = "transparent",colour = NA))
```

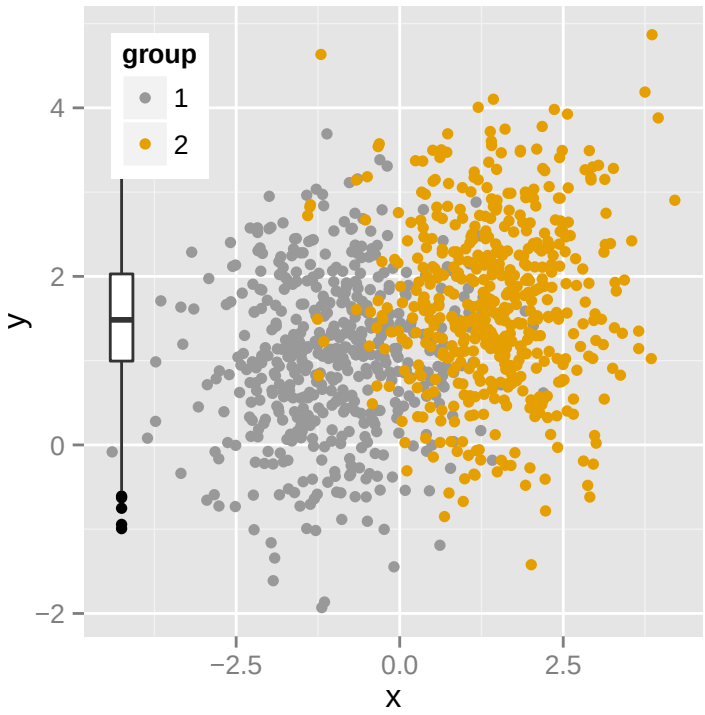
Create the graphs :

```
p1 <- scatterPlot # see previous sections for the scatterPlot

# Box plot of the x variable
p2 <- ggplot(df2, aes(factor(1), x))+
  geom_boxplot(width=0.3)+coord_flip()+
  transparent_theme

# Box plot of the y variable
p3 <- ggplot(df2, aes(factor(1), y))+
  geom_boxplot(width=0.3)+
  transparent_theme

# Create the external graphical elements
# called a "grop" in Grid terminology
p2_grob = ggplotGrob(p2)
p3_grob = ggplotGrob(p3)
```

```
# Insert p2_grob inside the scatter plot
xmin <- min(x); xmax <- max(x)
ymin <- min(y); ymax <- max(y)
p1 + annotation_custom(grob = p2_grob, xmin = xmin, xmax = xmax,
                       ymin = ymin-1.5, ymax = ymin+1.5)
```




```
# Insert p3_grob inside the scatter plot
p1 + annotation_custom(grob = p3_grob,
                       xmin = xmin-1.5, xmax = xmin+1.5,
                       ymin = ymin, ymax = ymax)
```



28.6 Mix table, text and ggplot2 graphs

The functions below are required :

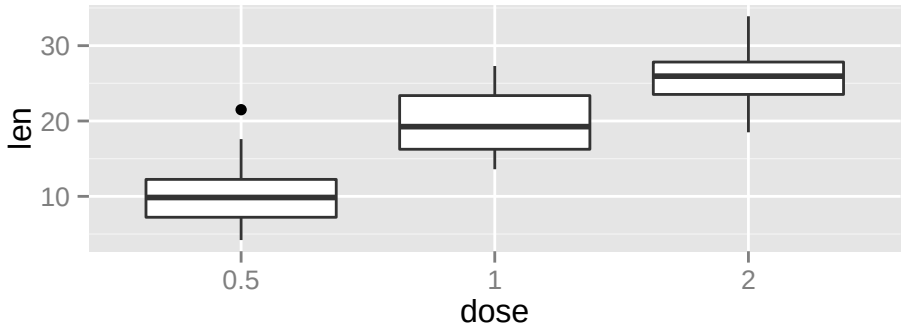
- **tableGrob()** [in the package *gridExtra*] : for adding a data table to a graphic device
- **splitTextGrob()** [in the package *RGraphics*] : for adding a text to a graph

Make sure that the package **RGraphics** is installed.

```
library(RGraphics)
library(gridExtra)

# Table
p1 <- tableGrob(head(ToothGrowth))

# Text
text <- "ToothGrowth data describes the effect of Vitamin C on tooth growth in Guinea pi
p2 <- splitTextGrob(text)

# Box plot
p3 <- ggplot(df, aes(x=dose, y=len)) + geom_boxplot()

# Arrange the plots on the same page
grid.arrange(p1, p2, p3, ncol=1)
```

	len	supp	dose
1	4.2	VC	0.5
2	11.5	VC	0.5
3	7.3	VC	0.5
4	5.8	VC	0.5
5	6.4	VC	0.5
6	10.0	VC	0.5

ToothGrowth data describes the effect of Vitamin C on tooth growth in Guinea pigs. Three dose levels of Vitamin C (0.5, 1, and 2 mg) with each of two delivery methods [orange juice (OJ) or ascorbic acid (VC)] are used.



Chapter 29

Correlation matrix heatmap

The visualization of a correlation matrix can be easily done using the function **ggcorr()** [in the package **GGally**], which is just an extension of ggplot2. I'll show you also, step by step, how to do this using only the package ggplot2.

29.1 Data

mtcars data is used :

```
mydata <- mtcars[, c(1,3,4,5,6,7)]
head(mydata)
```

```
##                    mpg disp  hp drat    wt  qsec
## Mazda RX4         21.0  160 110 3.90 2.620 16.46
## Mazda RX4 Wag     21.0  160 110 3.90 2.875 17.02
## Datsun 710        22.8  108  93 3.85 2.320 18.61
## Hornet 4 Drive    21.4  258 110 3.08 3.215 19.44
## Hornet Sportabout 18.7  360 175 3.15 3.440 17.02
## Valiant           18.1  225 105 2.76 3.460 20.22
```

29.2 GGally

The package GGally can be installed as follow:

```
# Installation
install.packages("GGally")
```

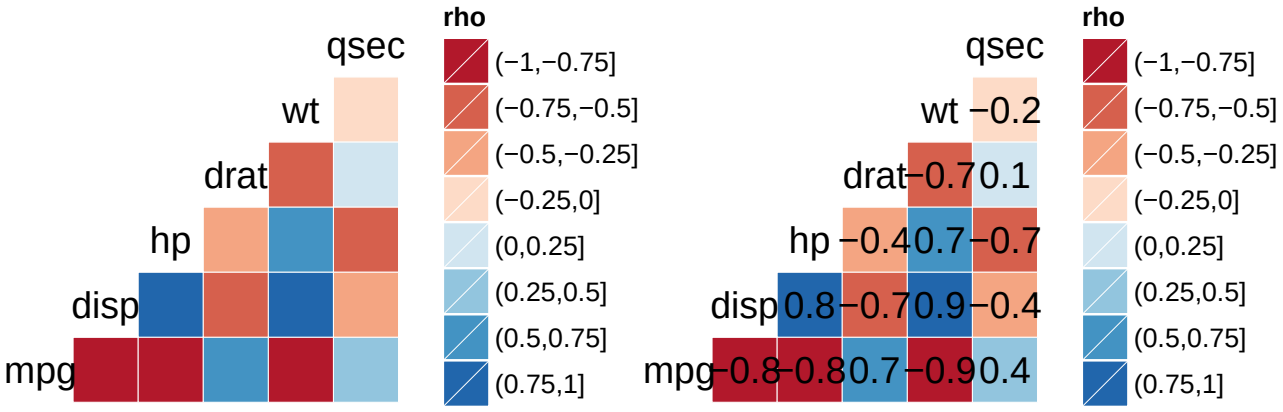
Load the package:

```
library("GGally")
```

29.2.1 ggcorr(): Plot a correlation matrix

The function **ggcorr()** can be used as follow:

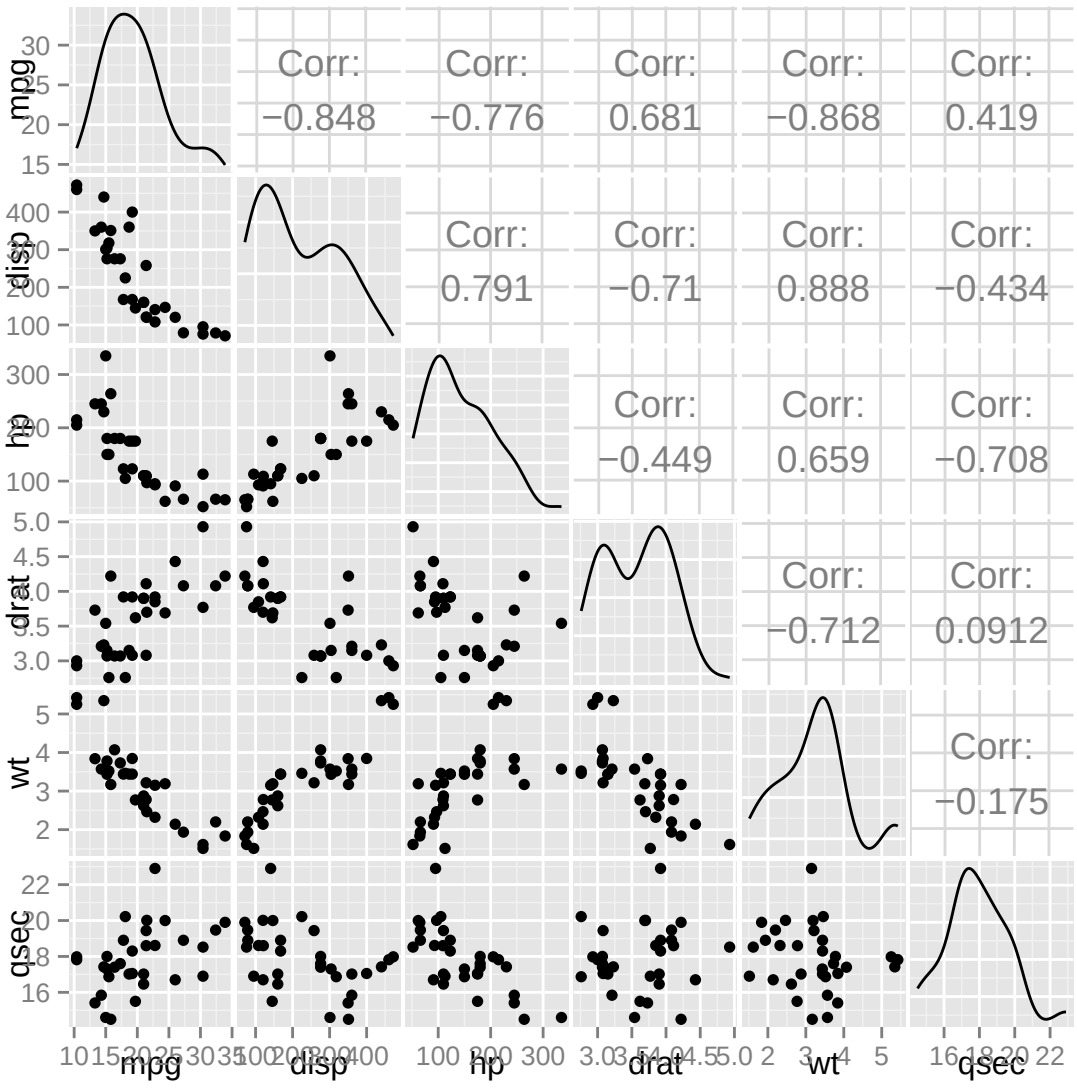
```
# Default plot
ggcorr(mydata, palette = "RdBu")

# Add labels
ggcorr(mydata, palette = "RdBu", label = TRUE)
```



Note that, the argument **palette** corresponds to ColorBrewer palettes.

It's also possible to make a matrix of scatter plots to visualize the correlation between variables. The function **ggpairs()**[in *GGally* package] is used:

```
# Default plot
ggpairs(mydata)
```



29.3 Build and visualize step by step a correlation matrix

29.3.1 Compute the correlation matrix

Correlation matrix can be created using the R function **cor()** :

```
cormat <- round(cor(mydata),2)
head(cormat)
```

```
##        mpg  disp    hp  drat    wt  qsec
## mpg   1.00 -0.85 -0.78  0.68 -0.87  0.42
## disp -0.85  1.00  0.79 -0.71  0.89 -0.43
```

```
## hp   -0.78  0.79  1.00 -0.45  0.66 -0.71
## drat  0.68 -0.71 -0.45  1.00 -0.71  0.09
## wt   -0.87  0.89  0.66 -0.71  1.00 -0.17
## qsec  0.42 -0.43 -0.71  0.09 -0.17  1.00
```

29.3.2 Create the correlation heatmap with ggplot2

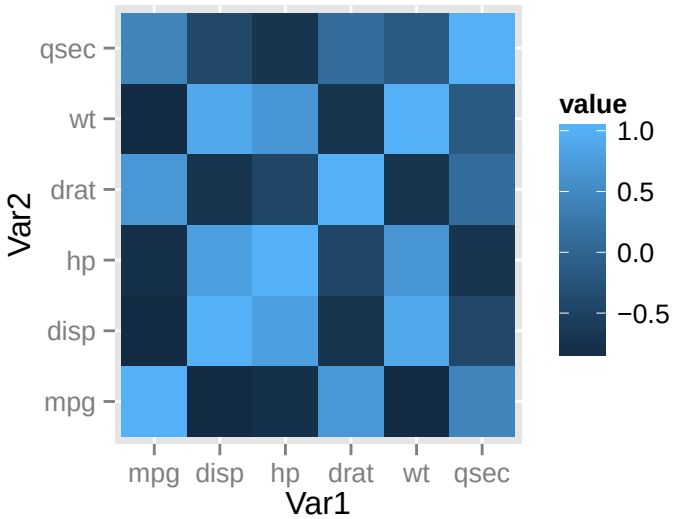
The package **reshape** is required to **melt** the correlation matrix :

```
library(reshape2)
melted_cormat <- melt(cormat)
head(melted_cormat)
```

```
##   Var1 Var2 value
## 1  mpg  mpg  1.00
## 2 disp  mpg -0.85
## 3   hp  mpg -0.78
## 4 drat  mpg  0.68
## 5   wt  mpg -0.87
## 6 qsec  mpg  0.42
```

The function **geom_tile()** [in **ggplot2** package] is used to visualize the correlation matrix :

```
library(ggplot2)
ggplot(data = melted_cormat, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile()
```



The default plot is very ugly. We'll see in the next sections, how to change the appearance of the heatmap.

Note that, if you have lot of data, it's preferred to use the function **geom_raster()** which can be much faster.

29.3.3 Get the lower and upper triangles of the correlation matrix

Note that, a correlation matrix has redundant information. We'll use the functions below to set half of it to NA.

Helper functions :

```
# Get lower triangle of the correlation matrix
  get_lower_tri<-function(cormat){
    cormat[upper.tri(cormat)] <- NA
    return(cormat)
  }

  # Get upper triangle of the correlation matrix
  get_upper_tri <- function(cormat){
    cormat[lower.tri(cormat)]<- NA
    return(cormat)
  }
```

Usage :

```
upper_tri <- get_upper_tri(cormat)
upper_tri
```

```
##      mpg  disp    hp  drat    wt  qsec
## mpg    1 -0.85 -0.78  0.68 -0.87  0.42
## disp  NA  1.00  0.79 -0.71  0.89 -0.43
## hp    NA    NA  1.00 -0.45  0.66 -0.71
## drat  NA    NA    NA  1.00 -0.71  0.09
## wt    NA    NA    NA    NA  1.00 -0.17
## qsec  NA    NA    NA    NA    NA  1.00
```

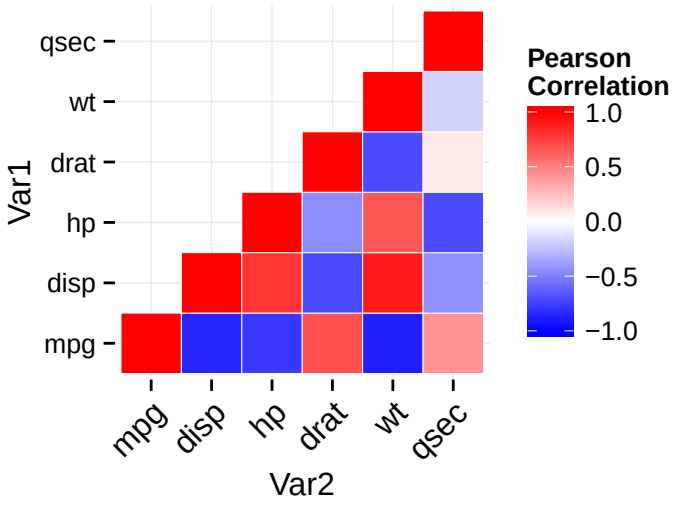
Note that, if you want to remove the diagonal values, use the R code below:

```
diag(upper_tri) <- NA
```

29.3.4 Finished correlation matrix heatmap

Melt the correlation data and drop the rows with NA values :

```
# Melt the correlation matrix
library(reshape2)
melted_cormat <- melt(upper_tri)
melted_cormat <- na.omit(melted_cormat)

# Heatmap
library(ggplot2)
ggplot(data = melted_cormat, aes(Var2, Var1, fill = value))+
 geom_tile(color = "white")+
 scale_fill_gradient2(low = "blue", high = "red", mid = "white",
   midpoint = 0, limit = c(-1,1), name="Pearson\nCorrelation") +
  theme_minimal()+
 theme(axis.text.x = element_text(angle = 45, vjust = 1,
    size = 12, hjust = 1))+
 coord_fixed()
```



In the figure above :

- **negative correlations** are in blue color and **positive correlations** in red. The function **scale_fill_gradient2** is used with the argument *limit = c(-1,1)* as correlation coefficients range from -1 to 1.
- **coord_fixed()**: this function ensures that one unit on the x-axis is the same length as one unit on the y-axis.

29.3.5 Reorder the correlation matrix

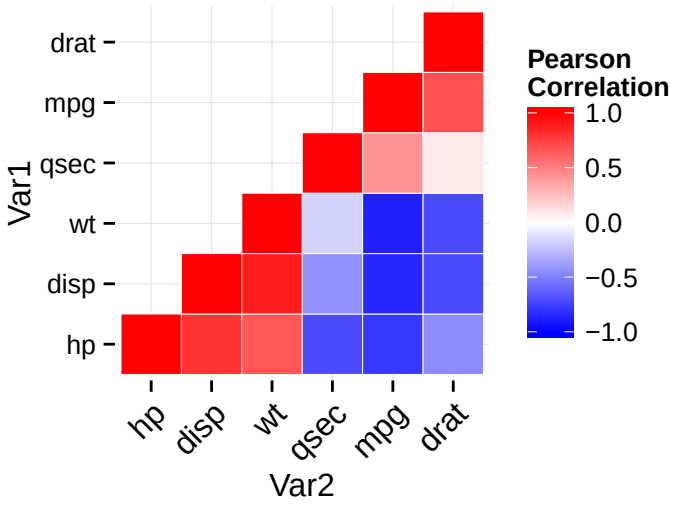
This section describes how to reorder the correlation matrix according to the correlation coefficient. This is useful to identify the hidden pattern in the matrix. **hclust** for hierarchical clustering order is used in the example below.

Helper function to reorder the correlation matrix :

```
reorder_cormat <- function(cormat){
# Use correlation between variables as distance
dd <- as.dist((1-cormat)/2)
hc <- hclust(dd)
cormat <-cormat[hc$order, hc$order]
}
```

Reordered correlation data visualization :

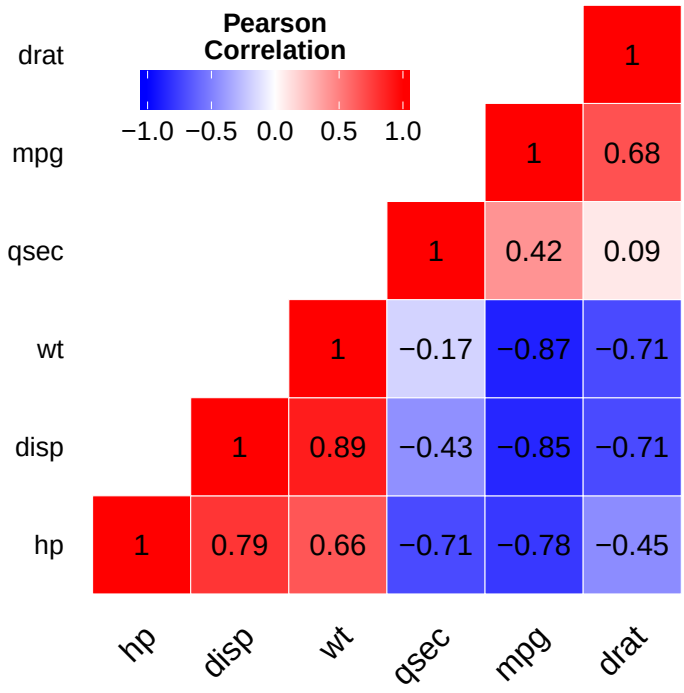
```
# Reorder the correlation matrix
cormat <- reorder_cormat(cormat)
upper_tri <- get_upper_tri(cormat)

# Melt the correlation matrix
melted_cormat <- melt(upper_tri)
melted_cormat <- na.omit(melted_cormat)

# Create a ggheatmap
ggheatmap <- ggplot(melted_cormat, aes(Var2, Var1, fill = value))+
 geom_tile(color = "white")+
 scale_fill_gradient2(low = "blue", high = "red", mid = "white",
   midpoint = 0, limit = c(-1,1), name="Pearson\nCorrelation") +
  theme_minimal()+ # minimal theme
 theme(axis.text.x = element_text(angle = 45, vjust = 1,
    size = 12, hjust = 1))+
 coord_fixed()

# Print the heatmap
print(ggheatmap)
```



29.3.6 Add correlation coefficients on the heatmap

1. Use **geom_text()** to add the correlation coefficients on the graph
2. Use a **blank theme** (remove axis labels, panel grids and background, and axis ticks)
3. Use **guides()** to change the position of the legend title

```
ggheatmap +
geom_text(aes(Var2, Var1, label = value), color = "black", size = 4) +
theme(
  axis.title.x = element_blank(),
  axis.title.y = element_blank(),
  panel.grid.major = element_blank(),
  panel.border = element_blank(),
  panel.background = element_blank(),
  axis.ticks = element_blank(),
  legend.justification = c(1, 0),
  legend.position = c(0.6, 0.7),
  legend.direction = "horizontal")+
  guides(fill = guide_colorbar(barwidth = 7, barheight = 1,
                title.position = "top", title.hjust = 0.5))
```



Chapter 30

Survival curves

The function **ggsurv()** [in the package **GGally**] can be used to draw survfit objects using ggplot2.

The simplified format is:

```
ggsurv(s, surv.col = "gg.def")
```

- **s**: an object of class survfit
- **surv.col**: colour of the survival estimate. The default value is **black** for one stratum; default ggplot2 colors for multiple strata. It can be also a vector containing the color names for each stratum.

30.1 Required packages

You should install the packages **survival** and **GGally** in order to use the function **ggsurv()**.

Installation of the packages:

```
install.packages("survival")

install.packages("GGally")
```

Loading of the packages:

```
library(survival)
library(GGally)
```

30.2 Data

We'll use the data *lung* from the package *survival*:

```
data(lung, package = "survival")
head(lung)
```

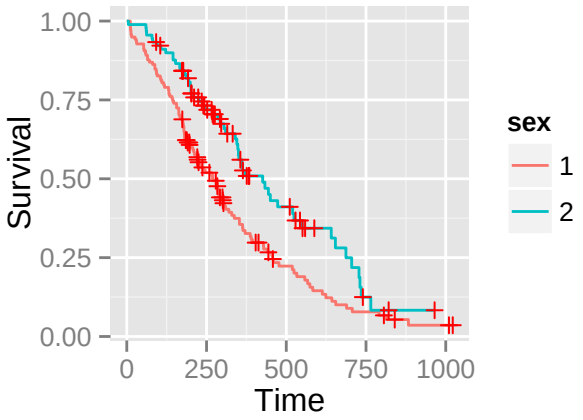
```
##   inst time status age sex ph.ecog ph.karno pat.karno meal.cal wt.loss
## 1    3  306      2  74   1       1       90       100     1175      NA
## 2    3  455      2  68   1       0       90        90     1225      15
## 3    3 1010      1  56   1       0       90        90       NA      15
## 4    5  210      2  57   1       1       90        60     1150      11
## 5    1  883      2  60   1       0      100        90       NA       0
## 6   12 1022      1  74   1       1       50        80      513       0
```

The data above includes:
- time: Survival time in days - status: censoring status 1 = censored, 2 = dead - sx: Male = 1; Female = 2

In the next section we'll plot the survival curves of male and female.

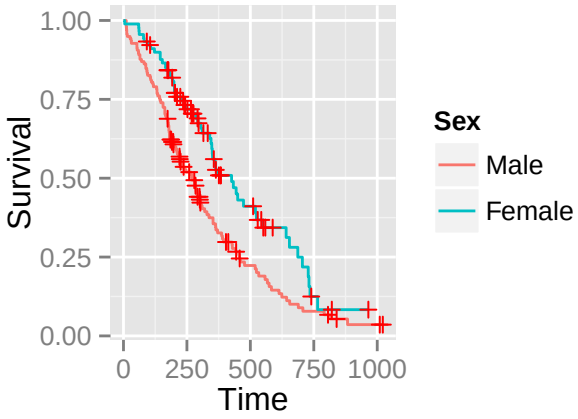
30.3 Survival curves

```
library("survival")
# Fit survival functions
surv <- survfit(Surv(time, status) ~ sex, data = lung)
# Plot survival curves
surv.p <- ggsurv(surv)
surv.p
```



It's possible to change the legend of the plot as follow:

```
surv.p +
  guides(linetype = FALSE) +
  scale_colour_discrete(
    name   = 'Sex',
    breaks = c(1,2),
    labels = c('Male', 'Female')
  )
```



30.4 References and further reading

- Hadley Wickman. Elegant graphics for data analysis. Springer 2009. http://ggplot2.org/book/
- Hadley Wickman. ggplot2 official documentation. http://docs.ggplot2.org/current/
- Winston Chang. R graphics cookbook. O'Reilly 2012. http://www.cookbook-r.com/
- Alboukadel Kassambara. Data analysis and visualization. http://www.sthda.com/english/wiki/ggplot2-introduction