# Introduction to Data Visualization and Plotting Using R

Pramudita Satria Palar, Ph.D.

17/8/2022

## 1 Introduction

Data visualization is one of the most important features that can be done using R. Data visualization in the form of plots greatly help in aiding users to understand the data better and draw important insight. Our brain is hard-wired to extract information faster from figures rather than tables, which is why it is important to visualize your data properly. This tutorial will primarily use R built-in functions to draw plots.

There are some R built-in functions that we can use to draw plots, including `plot()`, `barplot()`, `histogram()`, and `boxplot()`. In addition, some additional packages are specialized in creating more expressive and beautiful plots, such as `ggplot2`. However, in this tutorial, we will only focus on R's built-in functions and discuss more advanced visualization tools in other documents.

## 2 Using the `plot()` function

The main function that we will use is `plot()`. We can use `plot()` to create a variety of plots, including line or scatter plots. Let us start by downloading the `cars` data into our R session. The `cars` data is a simple data set consisting of the `speed` and `distance` of various cars, in which the former and the latter correspond to the velocity and the distance required to stop the car fully. The data will be saved in the data frame format, so make sure you are familiar with this data structure. We will save the data into a single variable `car_data` as in the following example:

```
car_data <- cars # The data set
car_data$speed # Velocity
```

```
##  [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14 15 15
## [26] 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25
```

```
car_data$dist # Distance
```

```
##  [1]   2  10   4  22  16  10  18  26  34  17  28  14  20  24  28  26  34  34  46
## [20]  26  36  60  80  20  26  54  32  40  32  40  50  42  56  76  84  36  46  68
## [39]  32  48  52  56  64  66  54  70  92  93 120  85
```

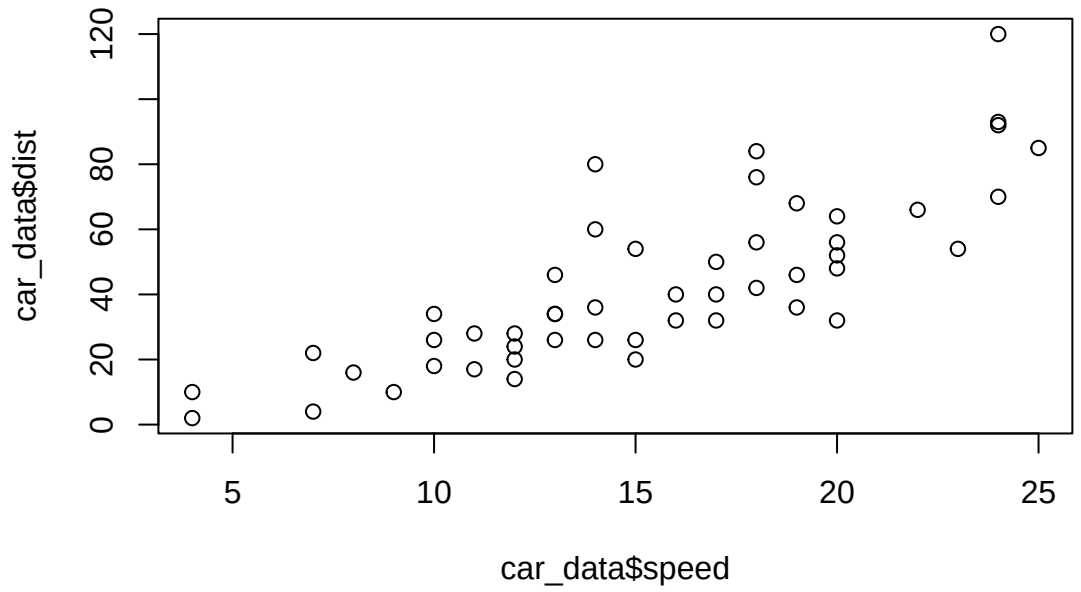
Remember that the Dollar sign (`$S$`) is for extracting a column from the data frame. Another way to view the `car_data` data in a more convenient fashion is to use `View()`:

```
View(car_data)
```

## 2.1 Creating your first plot

Let us begin by creating our simple plot that draws the relationship between `speed` and `dist` using the `plot` function:

```
plot(car_data$speed,car_data$dist)
```




Congratulations! You just drew your very first R plot. As expected, we can observe that the distance required to stop the car fully correlates positively with the car's velocity. The plot above is very simple, and we can make it more eye-pleasing. Besides, you need to add labels to the $x$ and $y$ axis and also give title to the plot. You can easily add labels and titles using additional arguments `xlab`, `ylab`, and `main` to `plot()`. You can also change the plot color using an extra argument `col` (for example, `col="blue"`). Furthermore, you can change the style of your marker using an extra argument `pch`. The `pch` argument should be filled with the marker's identity. For example, `pch=2` will draw a triangle, while `pch=17` will give us a filled triangle. Lastly (but this is not all), you can change the marker's size using `cex`. Please try the following example:

```
plot(car_data$speed,car_data$dist, main = "Speed vs distance", xlab = "Speed",
     ylab = "Jarak", col = "blue", pch=17, cex=1)
```

You can also try changing the font size for the label and the title using `cex.main` and `cex.lab`. Please try by adding `cex.main=2` to the plot above.

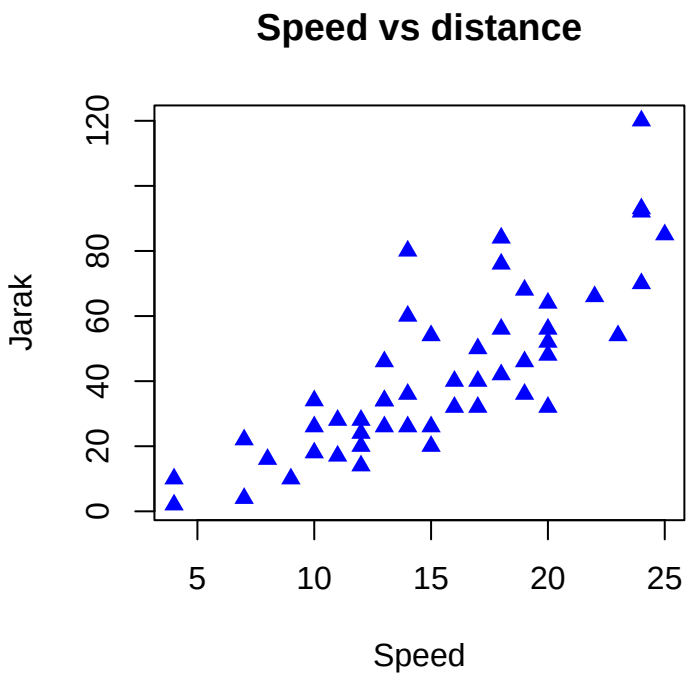



Figure 1: Plot kecepatan vs jarak

## 2.2 Drawing multiple curves in a single plot.

Occasionally (or perhaps frequently), you want to draw multiple curves in a single graph, like when you want to compare trends for two different categories or treatments. There are several methods to add multiple curves in R.

Before that, we will connect dots using a line (and not a scatter plot) so we need to change the type of the plot by adding an extra argument `type = 'o'` to `plot()`. In a similar fashion, a dotless line can be drawn using `type='l'` or `type='b'`. The main difference between `type='b'` dan `type='l'` is that the line in the former does not overlay the dots. Besides, we will use `grid()` to add grids to the plot. Finally, the function `axis()` will be used to adjust the ticks on the $x$- and $y$-axis.

The main function that we will use to add extra lines is `lines()`. Notice that in MATLAB we will use `hold on` and then subsequently add another plot. This method of adding additional plots in R may initially be inconvenient for MATLAB's users. The function `lines()` works similarly to `plot()`, but it is primarily used to add lines/dots, not as the main plot itself.

See the example below (note that `xlim` and `ylim` adjust the limits of the axes):

```
x <- seq(1,7,1) # To create a vector of sequence
y_1 <- x + 2
plot(x,y_1,type = 'b',col='red',pch=20,cex=2,xlim=c(0,8),ylim=c(2,12))

# Adjusting the axis
axis(1,at=seq(0,10,1)) # 1 indicates the x-axis
axis(2,at=seq(0,16,2)) # 2 indicates the y-axis
# Add a grid
grid(nx = NULL, ny = NULL,
```

```
     lty = 2,       # Grid line type
     col = "gray", # Grid  line colour
     lwd = 2)       # Grid line width
# Add the second and the third line
y_2 <- 1.25*x + 2
y_3 <- 1.1*x +1
lines(x,y_2,type='o',col='blue',pch=18,cex=2)
lines(x,y_3,type='o',col='green',pch=16,cex=2,lty=2)
legend(0,10,legend=c('y_1','y_2','y_3'),col=c('red','blue','green'),
       lty=c(1,2,2),pch=c(20,18,16))
```

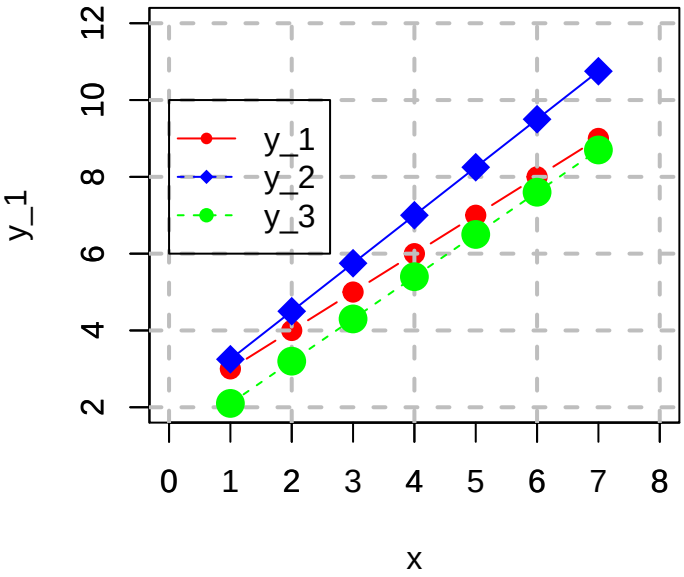



Figure 2: Plot of speed vs distance, with modifications

# 3 Boxplot

## 3.1 A simple example

Boxplot, also known as a "box-and-whiskers" plot, is a very useful plot that depicts the summary statistics of the data. In this regard, the "box" depicts three important statistics, namely, the median (Q2), lower quartile (Q1), and upper quartile (Q3). On the other hand, the "whiskers" depicts two important information, namely, the "maximum" and the "minimum" (in a quotation, because the definition of "maximum" and "minimum" is user-dependent). The most common definition for the "minimum" and the "maximum" is $Q1 - 1.5IQR$ and $Q3 + 1.5IQR$, respectively, where $IQR = Q3 - Q1$ is the interquartile range. The example below creates a data set with 1000 observations drawn from a normal distribution with $\mu = 0$ and $\sigma = 1$ (i.e., the mean the standard deviation of a normal distribution) using `rnorm()`. Notice that it is easier to denote a standard

normal distribution by using the notation $\mathcal{N}(0,1)$. The arguments taken by `rnorm()` are `n` (sample size), `mean` ($\mu$), and `sd` ($\sigma$).

The function for creating a boxplot is `boxplot()`, which takes a minimal one argument, i.e., the data. See an example below:

```
x_norm_1 <- rnorm(1000,mean=0,sd=1) # 1000 random observations from a standard normal distribution
boxplot(x_norm_1) # Create a histogram
```

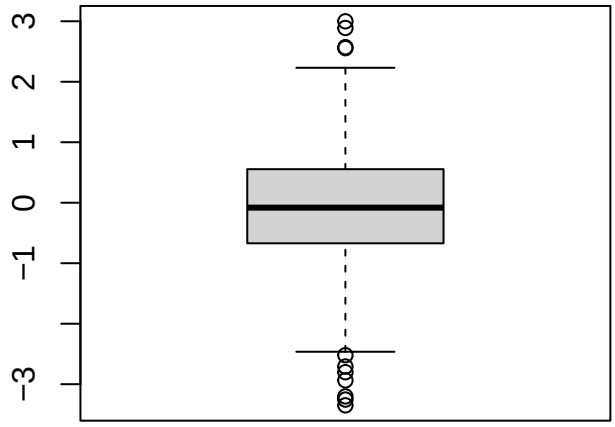



Figure 3: An example of boxplot

To include two or more boxplots from different data (e.g., plot for each category), the best approach is to combine the data into a list first (use `list()`). Besides, we will add extra arguments to make the boxplot more informative:

```
x_norm_1 <- rnorm(1000,mean=0,sd=1)
x_norm_2 <- rnorm(1000,mean=1,sd=2)
boxplot(list(x_norm_1,x_norm_2)
        ,names=c('Data 1','Data 2')
        ,xlab='Category'
        ,ylab='Value'
        ,main='A boxplot example'
        )
```

## 3.2 Boxplot from a data frame

It is very common to have a data set in the data frame format, and this is probably the most convenient data structure for data science. However, I need to make a note here that it is more convenient to use

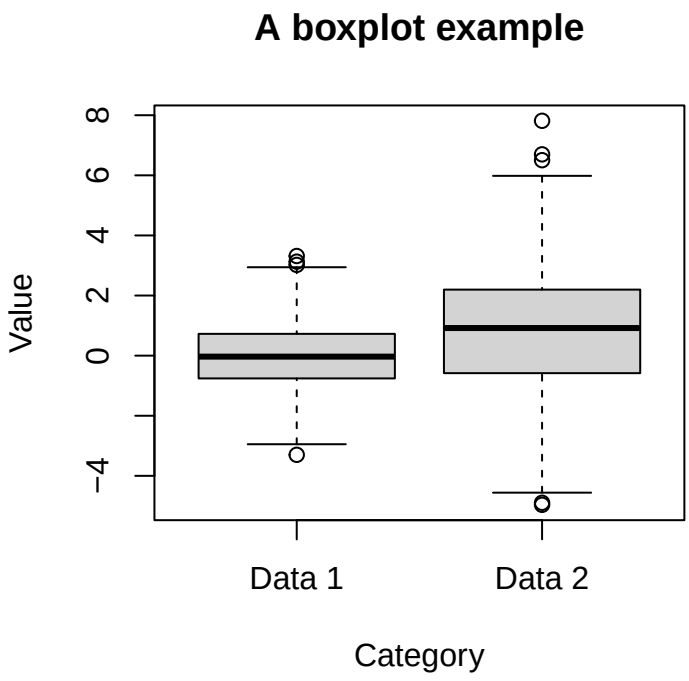



Figure 4: Boxplots with different categories

matrix to perform more advanced statistical computations. Let us try a simple example using a built-in data from R, i.e., `mtcars`. The `mtcars` data comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (the data is quite old, it was extracted from a 1974 magazine). For simplicity, only the `mpg` (Miles/US Gallon) and `cyl` (number of cylinders) will be used in this tutorial. To be more exact, the boxplot will display `mpg` with `cyl` as the category. Let us execute the code below:

```
data_cars <- mtcars[,c('mpg','cyl')] # Save the data into variable `data_cars`
head(data_cars) # Display the first 6 rows of the data
```

```
##                   mpg cyl
## Mazda RX4         21.0   6
## Mazda RX4 Wag     21.0   6
## Datsun 710        22.8   4
## Hornet 4 Drive    21.4   6
## Hornet Sportabout 18.7   8
## Valiant           18.1   6
```

Alternatively, you can also type `View("data_cars")` to display the whole data set.

The first important argument that should be included is the name of the data (i.e., `data`). In this regard, we have to fill out `data` with `data=data_cars`. The next step is to fill out the `formula` argument in the format of `y~grp`, where `y` is the numerical data that is categorized based on groups (`grp`). Because `mpg` will be factored based on `cyl`, then the formula would be `formula=mpg~cyl`.

```
boxplot(formula=mpg~cyl,data=data_cars)
```

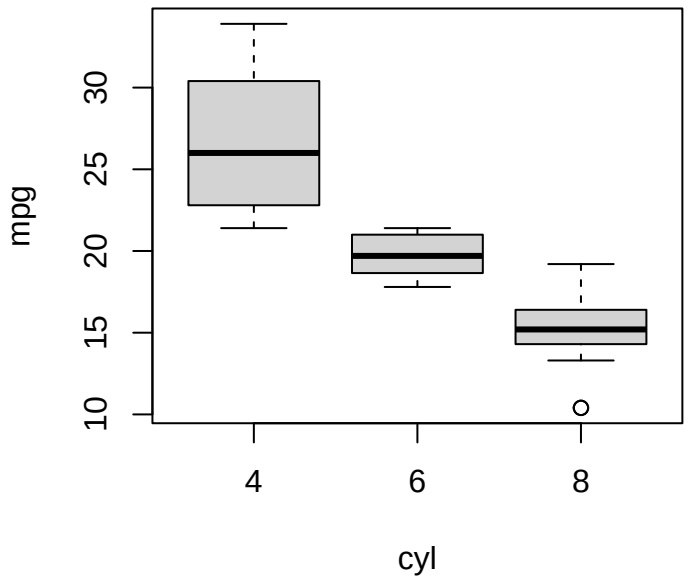



Figure 5: An example of boxplot from a data frame

Let us step further by giving the labels, colors, and names for all categories. It is also possible to change the orientation of the boxplot from vertical to horizontal by adding the argument `horizontal=TRUE` (do not forget to change the label because the $x$- and the $y-$ axis will switch places). Note that colors and names are added using `col` and `names`. For multiple boxplots, the input to `col` and `names` is a vector (e.g., `col=c('red','green','blue')`) ‘

```
boxplot(formula=mpg~cyl,data=data_cars,xlab='Number of cylinders',ylab='Miles per gallon',
        col=c('red','green','blue'),names=c('High','Medium','Low'))
```

Individual observations can be displayed together with the boxplot in the form of scatter plot. The function to perform this particular job is `stripchart()`. We will also add an argument `add=TRUE` to `stripchart()` so as to denote that the stripchart is put on top of the boxplot.

```
boxplot(formula=mpg~cyl,data=data_cars,xlab='Number of cylinders',ylab='Miles per gallon',
        names=c('High','Medium','Low'))
stripchart(mpg~cyl,data=data_cars,vertical=TRUE,method='jitter',
           add=TRUE, col=c('red','black','blue'))
```

These are just several tricks to create a boxplot. I suggest you explore different techniques to make your boxplot more informative and prettier.

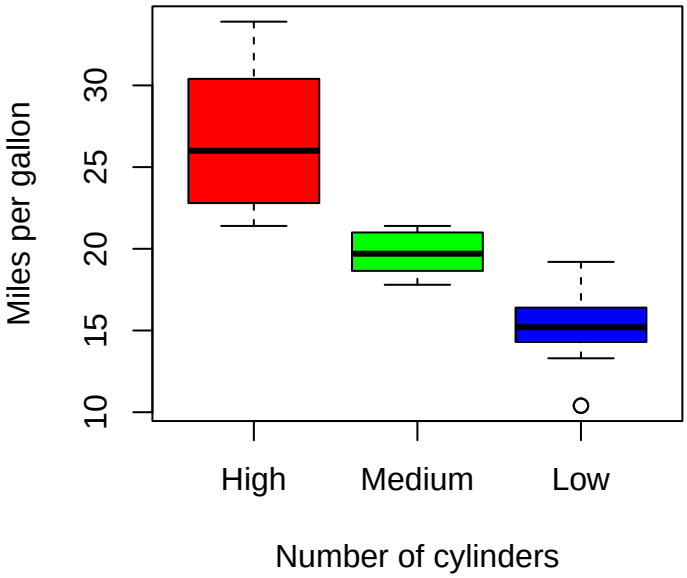



Figure 6: An example of boxplot with adjusted colors

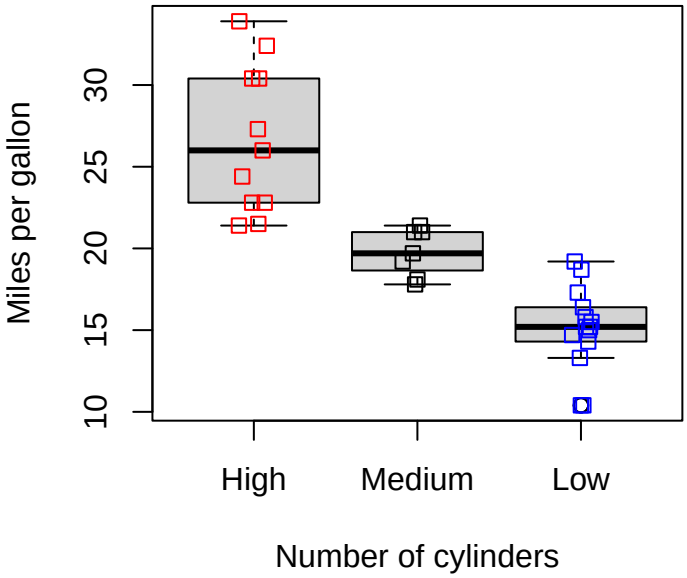



Figure 7: An example of boxplot with stripchart

# 4 Histogram

A histogram is a very useful plot for visualizing the distribution of the data. The data will be depicted as a barplot, in which the height of each bar shows the frequency of observations that falls within the corresponding bin. For illustration purposes, we will use data generated from a standard normal distribution $\mathcal{N}(0, 1)$.

After that, we need to call the function `hist()` to create a histogram from our data. Notice that `hist()` automatically chooses the number of bins using the method of Sturges. However, users can add an extra argument `breaks` to tune the number of bins (use an integer). It is worth noting that the value is just a recommendation because R will readjust the histogram again for optimum visualization. The input to `breaks` does not always need to be an integer. Instead, you can define the name of the method. Alternatives to Sturges' method include `breaks='Freedman-Diaconis'`or `breaks='Scott'`.

```
x_norm_1 <- rnorm(1000,mean=0,sd=1) # 1000 random numbers from N(0,1)
hist(x_norm_1,breaks=10) # Creating a histogram with 10 bins
```

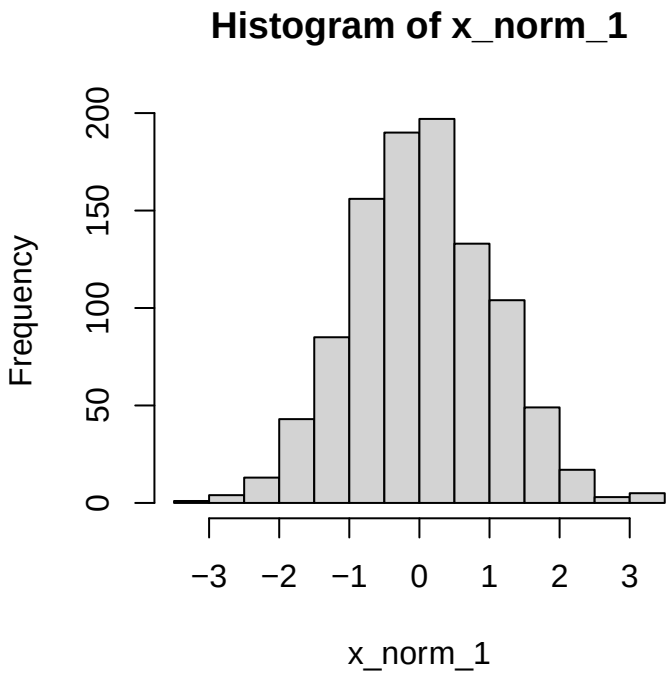



Figure 8: A histogram of a standard normal distribution

Now let us try a different inputting method for `breaks`. The argument `breaks` can also be a numerical vector consisting of the actual breaks of the bins. Using this method, each bin might have a different width as in the following example (notice that you may need to repeat the execution because the numerical vector should take into account the minimum and maximum of the data):

```
x_norm_1 <- rnorm(1000,mean=0,sd=1) # 1000 random values from standard normal distribution
hist(x_norm_1,breaks=c(-5,-3,-2,0,1,2,4,5)) # Create a histogram with arbitrary breaks
```

Similar to the other plots, it is also possible to change the label and the title of the histogram (see the

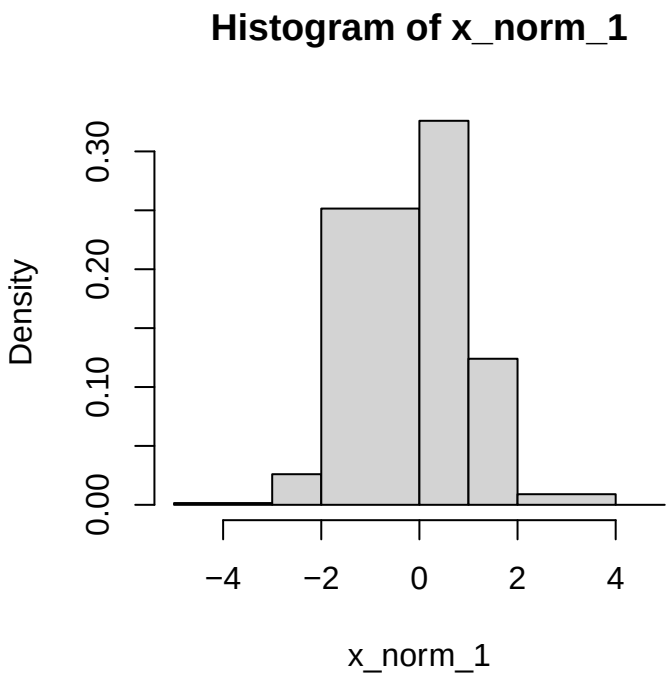



Figure 9: A histogram with arbitrary breaks

example below). Here, the `col` and `border` arguments are added to change the color of the histogram and the accompanying border, respectively. For more details, please type `?hist` in your R Console:

```
x_norm_1 <- rnorm(1000,mean=0,sd=1) # 1000 random numbers from standard normal distribution
hist(x_norm_1,breaks=10,main='A histogram example'
     ,xlab='Data',col='blue',border='red')
```

## 5 Piechart

A pie chart is typically used to show data in the form of proportion. Imagine showing your data in the form of pizza slices (in which you want the biggest slice). Making a pie chart with R is relatively straightforward through the `pie()` function that takes a minimum one argument (that is, the data). Let us use `pie()` with a simple data `pie_data <- c(1,5,10,5)`. The `pie()` function automatically calculates the proportion based on the given data. See the following example:

```
pie_data <- c(1,5,10,5)
pie(pie_data,
    labels = c('Group A','Group B','Group C','Group D'),
    col=c('red','green','blue','white'),
    main='A simple pie chart'
    )
```

For more informative visualization, you might want to show the corresponding values for each group by putting extra arguments. Try the following alternative:

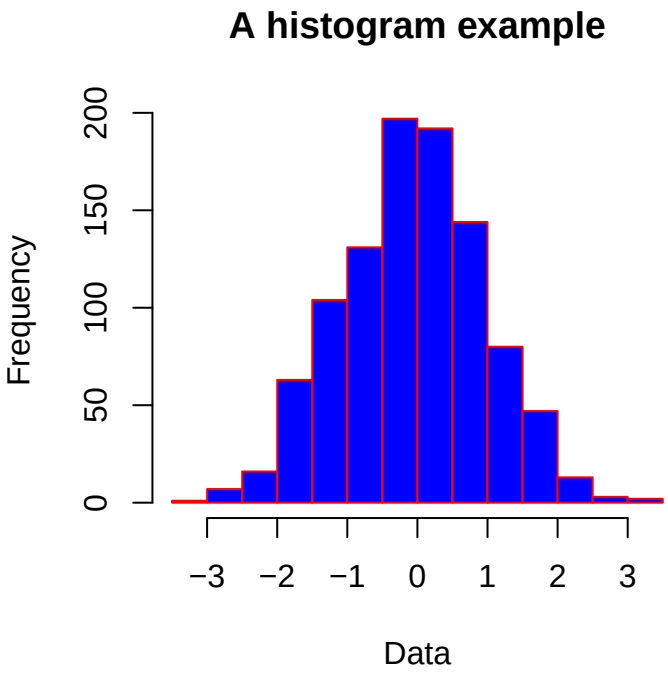



Figure 10: A histogram with extra arguments

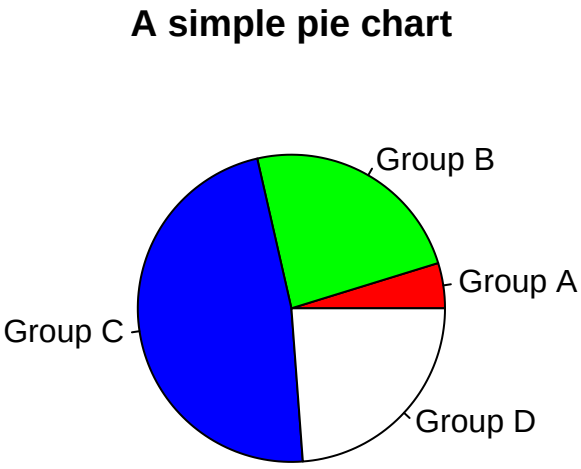



Figure 11: An example of pie chart

```
pie_data <- c(1,5,10,5)
label_names <- c('Group A','Group B','Group C','Group D')
pie(pie_data,
    labels = pie_data,
    col=c('red','green','blue','white'),
    main='A simple pie chart'
    )

legend("topright",label_names,fill=c('red','green','blue','white'))
```

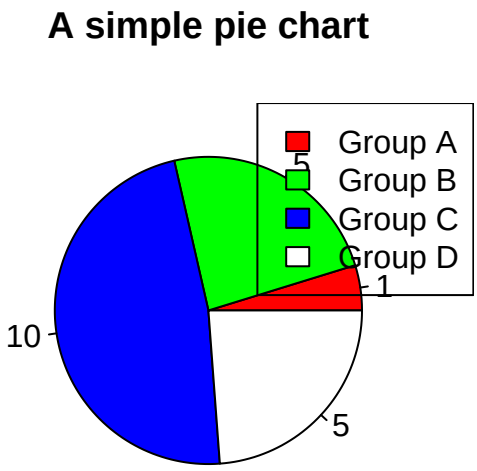



Figure 12: Another example of pie chart

## 6 Scatter plot matrix

An individual scatter plot can be easily created using `plot()`. How if we want to make a scatter plot matrix, that is, when we have more than two variables? This is when we need to use `pairs()` to solve this problem. Notice that `pairs()` accepts data either in the matrix format or data frame.

Let us start by generating the data matrix first. There are three individual features: `x_1` from $\mathcal{N}(0,1)$, `x_2` from $x_1 + \mathcal{N}(0, 0.1)$, and `x_3` from $\mathcal{N}(0,2)$. It is easy to see that `x_1` is correlated with `x_2`, while `x_3` does not bear any relationship with the other variables. The code below generates the data for the three variables and then subsequently uses `pairs()`. Notice that we need to combine the three data into a single matrix by using `cbind()` (binding vectors/matrices column-wise):

```
x_1 <- rnorm(200,mean=0,sd=1)
x_2 <- x_1 + rnorm(200,mean=0,sd=0.1)
```

```r
x_3 <- rnorm(200,mean=0,sd=2)

x_data <- cbind(x_1,x_2,x_3) # Combine into a single matrix
pairs(x_data,labels=c('Var1','Var2','Var3'))
```

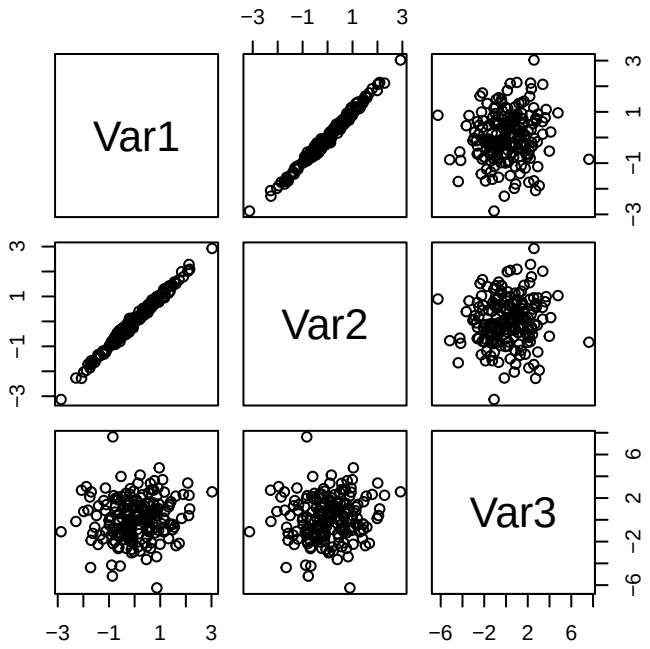



Figure 13: An example of scatter plot matrix

Nicely done. Now we can see that each scatter plot draws a relationship between variables indicated by the row and the column. For example, the plot shown in the first row and second column depicts the scatter plot for the first and the second variable (i.e., $x_1$ and $x_2$, respectively). The diagonal of this scatter plot matrix shows the variable's name. It is important to mention that some other implementations show the histogram of each data in the diagonal component of the scatter plot matrix.

Next, let us try building a scatter plot matrix from a data frame. The data that we will use is the `mtcars` data again:

```r
mtcars_data <- mtcars # Saving the data in mtcars_data
head(mtcars_data) # View the first parts of the data
```

```
##                   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
```

As previously discussed, the `mtcars_data` has eleven variables, but we will not use all of them. For this tutorial, we will create a scatter plot matrix using the following variables: `wt` (weight in 1000 lbs), `mpg`

(Miles/ US gallon), `disp` (displacement), and `cyl` (number of cylinders). The function `pairs()` will be used slightly differently because the data is now a data frame (previously, it was a matrix). Using a data frame, it is necessary to include the following arguments: `data` (your data) and `formula`. A formula is written such as `~var_1+var_2+...+var_n`, where `var_1,..,var_n` indicates the names of the variables. See the following demonstration:

```
pairs(formula=~wt+mpg+disp+cyl,data=mtcars_data)
```

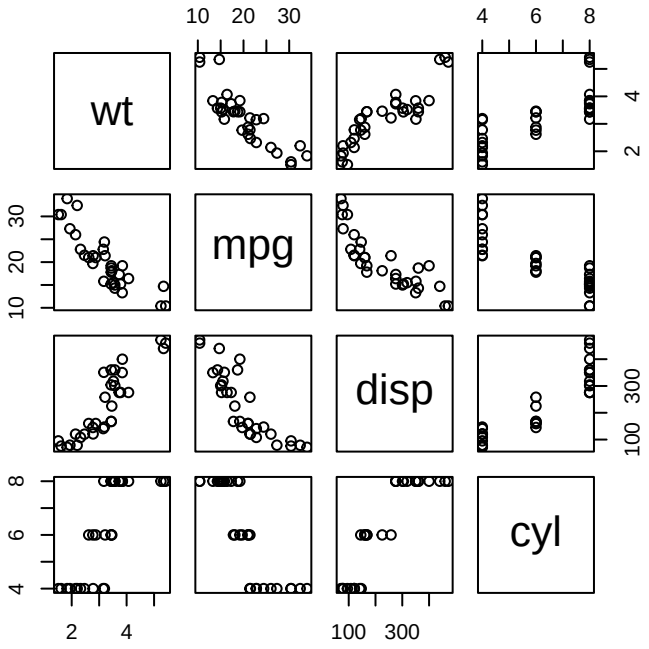



Figure 14: An example of scatter plot matrix from a data frame

There are some important trends that can be observed from the plot above. For example, heavier vehicles correspond to smaller Miles/US Gallon compared to lighter vehicles (i.e., the variables are negatively correlated, try typing `cor(mtcars$mpg,mtcars$wt)` in your R console). Furthermore, the more cylinders we have, the less the value of the `mpg`, etc.

Try experimenting with your own data set.

## 6.1 Barplot

To create a barplot in R can be simply done by using `barplot()`. Just like other R functions, `barplot` can take data frame or direct numerical data as its inputs. There are at least two important arguments for `barplot()`, namely the `height` (heights of the bar) and `names.arg` (names for each bar). Additionally, we can also set the color by using an extra argument `col`. See the example below:

```
heightcol <- c(10,5,2)
name <- c('A','B','C')
barplot(heightcol,names.arg=name,col=c('red','blue','green'))
```

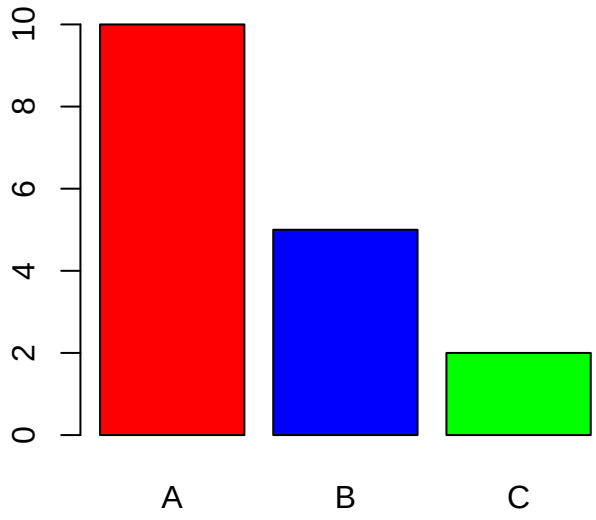



Figure 15: An example of simple barplot

# 7 Final thoughts

What I explain in this tutorial only touches the tip of the iceberg of plotting and visualization methods in R. There are so many things you can explore, but at least this short tutorial can give you important basics regarding how to use R for data visualization and plotting. Furthermore, I think using R is much more convenient than Spreadsheet softwares for data visualization. That is, R has more features, not to mention that the plots are more eye-pleasing (and you can make them better with, e.g., `ggplot2`).

What is next? Try exploring R's plotting feature using various online sources or books. You can also try playing with arguments in R's plotting functions because there is a lot of them. Besides, I also suggest you to try `ggplot2` for even better plotting and data visualization. Please wait for my next tutorial.