Summary Statistics

Hao Li

Summary statistics

Header

This file belongs to the collection of

https://github.com/HaoLi111/All-stats-and-DS-on-iris

which intends to give

A collection of ALL POSSIBLE statistical and data science models on the iris data set in R.

with MIT License

https://github.com/HaoLi111/All-stats-and-DS-on-iris/blob/master/LICENSE

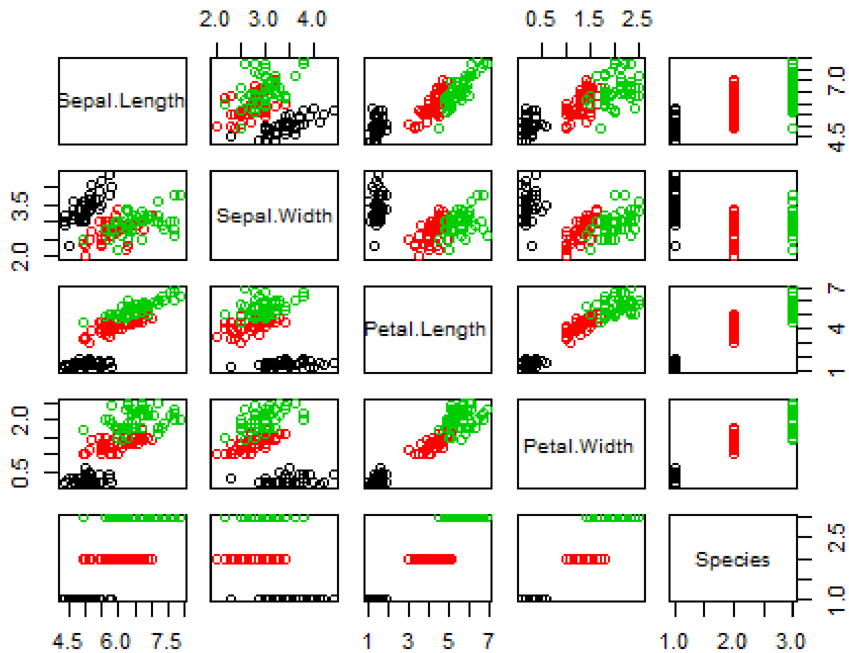
First Glance

R has basic functions to calculate numerical aspects of data. Perhaps the very first thing to look at your data is with summary() and plot(). You wouldn't expect to have a decent view of data, but a very general one.

You are expected to read the article Exploratory visualization before this part.

```
summary(iris)

##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
## 
```

```r
plot(iris,col= factor(iris$Species))
```



With summary(), you can access the min, max median, and other aspects of data. With plot(), you can have a general view of your data.

But keep in mind that the trend shown at the first glance can be misleading - both numerically and visually. There are lots of cases when things become counter-intuitive.

A better summary

To crunch the iris dataset, simply printing one-line commands isn't enough. It is suggested that R is a programming language, or more specifically, a scripting language. Programmability means flexibility.

Structure of Data

```r
class(iris)

## [1] "data.frame"

typeof(iris)

## [1] "list"
```

The 'Class' of the iris is "data.frame" and "type" is list. With most S3 methods the data can only be handled with the methods for "data.frame".

```
str(iris)

## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1
1 1 1 1 1 1 1 ...
```

Subsetting

For example, if I am only interested in the variable "Petal.Length"

```
(iris_Petal.Length = iris[,"Petal.Length"])

##   [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.
5 1.3
##  [18] 1.4 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.
5 1.4
##  [35] 1.5 1.2 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.
4 4.7
##  [52] 4.5 4.9 4.0 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.
5 4.1
##  [69] 4.5 3.9 4.8 4.0 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.
1 4.5
##  [86] 4.5 4.7 4.4 4.1 4.0 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1 6.
0 5.1
## [103] 5.9 5.6 5.8 6.6 4.5 6.3 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.
7 6.9
## [120] 5.0 5.7 4.9 6.7 4.9 5.7 6.0 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.
6 6.1
## [137] 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9 5.7 5.2 5.0 5.2 5.4 5.1
```

This is printed as a numeric vector. If I still want a data.frame (with 1 list). Do this(for convenience only printing the first 6 rows of it),

```
iris_Petal.Length_notDrop =iris[,"Petal.Length",drop=F]
head(iris_Petal.Length)

## [1] 1.4 1.4 1.3 1.5 1.4 1.7
```

Calculate Summary Statistics Aspects

You can run summary() on a single numeric vector the same way you do it on dataframes. Also it is possible to get the index of the max and min values. Note that which.max() and which.min() only support 1D vector, so use drop=T. For Example,

```
summary(iris_Petal.Length)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.600   4.350   3.758   5.100   6.900
```

```r
mean(iris_Petal.Length)
```

```
## [1] 3.758
```

```r
max(iris_Petal.Length);min(iris_Petal.Length)
```

```
## [1] 6.9
```

```
## [1] 1
```

```r
which.max(iris_Petal.Length);which.min(iris_Petal.Length)
```

```
## [1] 119
```

```
## [1] 23
```

```r
quantile(iris_Petal.Length,.25)
```

```
## 25%
## 1.6
```

```r
quantile(iris_Petal.Length,.75)
```

```
## 75%
## 5.1
```

Also the numeric functions are availiable. We can calculate sd(Standard Derivation by taking the square root of the derivation).

$$sd = \sqrt{\frac{\Sigma x^2}{n} - (\frac{\Sigma x}{n})^2}$$

```r
length(iris_Petal.Length)
```

```
## [1] 150
```

```r
var(iris_Petal.Length)
```

```
## [1] 3.116278
```

```r
sd(iris_Petal.Length)
```

```
## [1] 1.765298
```

```r
sqrt(sum(iris_Petal.Length^2)/length(iris_Petal.Length) - (sum(iris_Pet
al.Length/length(iris_Petal.Length)))^2)
```

```
## [1] 1.759404
```

As you can see the sd that is returned from the function sd() differs from that calculated from the equation, why?

Let's access the help file, and it says,

"Like var this uses denominator n-1"

This is an unbiased estimator of sd, with,

$$sd = \sqrt{\frac{\Sigma x^2}{n-1} - \frac{(\Sigma x)^2}{(n-1)*n}}$$

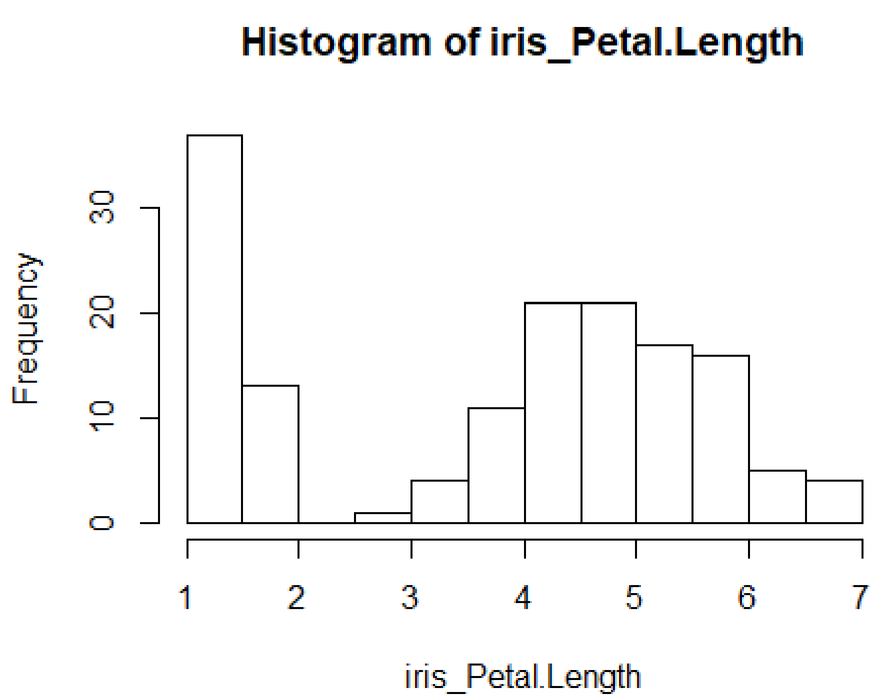
Let's modify our expression a little bit:

```
sqrt(sum(iris_Petal.Length^2)/(length(iris_Petal.Length)-1)-(sum(iris_Petal.Length)^2/((length(iris_Petal.Length)-1)*length(iris_Petal.Length))))# actually, don't do this, use sd()

## [1] 1.765298
```

Aggregating/ Facetting/ summarizing?

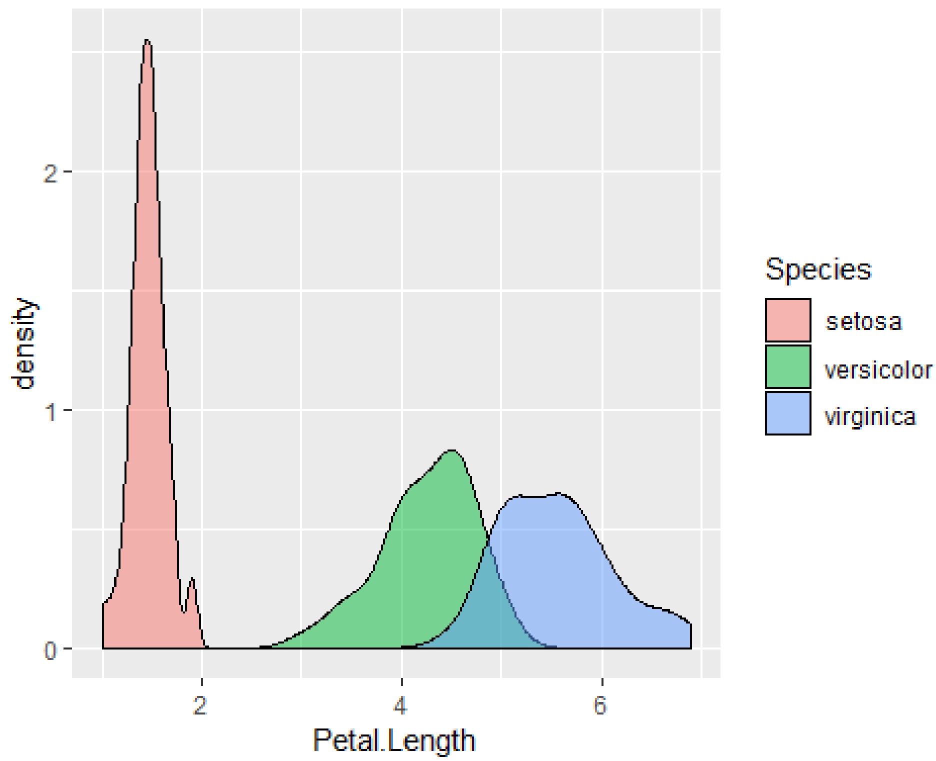
Recall that we can make histogram with

```
hist(iris_Petal.Length)
```



At the first glance it has 2 peaks. Recall that we can facet the distribution with ggplot.

```
library(ggplot2)

## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang

ggplot(data=iris,aes(x=Petal.Length,fill = Species)) +
  geom_density(alpha=.5)
```



A base solution : aggregate

```
attach(iris)
aggregate(Sepal.Length, by = list(Species=Species),
          FUN = sd)

##      Species         x
## 1     setosa 0.3524897
## 2 versicolor 0.5161711
## 3  virginica 0.6358796

aggregate(Sepal.Length,by =list(Species = Species),
          FUN = pastecs::stat.desc)
```

```
##       Species    x.nbr.val   x.nbr.null     x.nbr.na        x.min
## 1      setosa  50.00000000   0.00000000   0.00000000   4.30000000
## 2 versicolor  50.00000000   0.00000000   0.00000000   4.90000000
## 3  virginica  50.00000000   0.00000000   0.00000000   4.90000000
##           x.max      x.range        x.sum     x.median       x.mean
## 1    5.80000000   1.50000000 250.30000000   5.00000000   5.00600000
## 2    7.00000000   2.10000000 296.80000000   5.90000000   5.93600000
## 3    7.90000000   3.00000000 329.40000000   6.50000000   6.58800000
##       x.SE.mean x.CI.mean.0.95        x.var    x.std.dev   x.coef.var
## 1    0.04984957     0.10017646   0.12424898   0.35248969   0.07041344
## 2    0.07299762     0.14669422   0.26643265   0.51617115   0.08695606
## 3    0.08992695     0.18071498   0.40434286   0.63587959   0.09652089
```

For the same reason you can apply customized functions

```
Skewness =function(x){
  m = mean(x)
  n = length(x)
  s = sd(x)
  sum((x-m)^3/s^3)/n
}
aggregate(Sepal.Length,by =list(Species = Species),
          FUN =Skewness   )
```

```
##       Species          x
## 1      setosa 0.11297784
## 2 versicolor 0.09913926
## 3  virginica 0.11102862
```

Since positive, all 3 species are with right skewed distribution of Sepal.Length.

An agreed way of summarizing?

Problem is, everyone wants to rule the world. By “the world”, I mean their “Imagine Estacy”, therefore, a number of different summarizing toolbox has been defined with different packages.

Worse, they don’t always work with aggregate(), nor with summarize in tidyverse.

Doing such is easy.

```
print(Hmisc::describe(iris[,1]))
```

```
## iris[, 1]
##        n  missing distinct     Info     Mean      Gmd      .05      .
10
##      150        0       35    0.998    5.843   0.9462    4.600    4.
800
##      .25      .50      .75      .90      .95
##    5.100    5.800    6.400    6.900    7.255
```

```
##
## lowest : 4.3 4.4 4.5 4.6 4.7, highest: 7.3 7.4 7.6 7.7 7.9
```

```
print(psych::describe(iris[,1]))
```

```
##    vars   n mean   sd median trimmed  mad min max range skew kurtosi
s   se
## X1    1 150 5.84 0.83    5.8    5.81 1.04 4.3 7.9   3.6 0.31    -0.6
1 0.07
```

```
pastecs::stat.desc(iris[,1],basic = T,
                   desc = F,
                   norm = F)
```

```
##  nbr.val nbr.null   nbr.na      min      max    range      sum
##    150.0      0.0      0.0      4.3      7.9      3.6    876.5
```

```
pastecs::stat.desc(iris[,1],basic =F,
                   desc = T,
                   norm = F)
```

```
##       median         mean      SE.mean CI.mean.0.95          var
##   5.80000000   5.84333333   0.06761132   0.13360085   0.68569351
##      std.dev     coef.var
##   0.82806613   0.14171126
```

```
pastecs::stat.desc(iris[,1],basic =F,
                   desc = F,
                   norm = T)
```

```
##    skewness    skew.2SE    kurtosis    kurt.2SE  normtest.W  normtes
t.p
##  0.30864073  0.77924478 -0.60581253 -0.76961200  0.97609027  0.01018
116
```

but to do aggregation for multiple outputs, you will inevitably end up using other wrappers to save time, (and sometimes by() breaks!)

True, you can write some customized functions.

```
ply_by = function(x,.f,param = 1,groups = ncol(x),as.list = FALSE,...){
  Levels = levels(x[,groups])
  re=list()
  l =length(Levels)
  for(i in 1:l){
    print(Levels[i])
    print(.f(x[x[,groups]==Levels[i],param],...))
  }
  #re
}

ply_by(iris,.f = summary)
```

```
## [1] "setosa"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   4.300   4.800   5.000   5.006   5.200   5.800 
## [1] "versicolor"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   4.900   5.600   5.900   5.936   6.300   7.000 
## [1] "virginica"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   4.900   6.225   6.500   6.588   6.900   7.900
```

```
ply_by(iris,.f = Hmisc::describe)
```

```
## [1] "setosa"
## x[x[, groups] == Levels[i], param] 
##        n  missing distinct     Info     Mean      Gmd      .05      .
10 
##       50        0       15    0.989    5.006   0.4004     4.40     4.
59 
##      .25      .50      .75      .90      .95 
##     4.80     5.00     5.20     5.41     5.61 
## 

## Value        4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.0  5.1  5.2  5.3  5.
4
## Frequency      1    3    1    4    2    5    4    8    8    3    1    
 5
## Proportion 0.02 0.06 0.02 0.08 0.04 0.10 0.08 0.16 0.16 0.06 0.02 0.
10
## 
## Value        5.5  5.7  5.8
## Frequency      2    2    1
## Proportion 0.04 0.04 0.02
## [1] "versicolor"
## x[x[, groups] == Levels[i], param] 
##        n  missing distinct     Info     Mean      Gmd      .05      .
10 
##       50        0       21    0.995    5.936   0.5918    5.045    5.
380 
##      .25      .50      .75      .90      .95 
##    5.600    5.900    6.300    6.700    6.755 
## 
## lowest : 4.9 5.0 5.1 5.2 5.4, highest: 6.6 6.7 6.8 6.9 7.0
## [1] "virginica"
## x[x[, groups] == Levels[i], param] 
##        n  missing distinct     Info     Mean      Gmd      .05      .
10 
##       50        0       21    0.995    6.588   0.7171    5.745    5.
800 
##      .25      .50      .75      .90      .95 
##    6.225    6.500    6.900    7.610    7.700
```

```
##
## lowest : 4.9 5.6 5.7 5.8 5.9, highest: 7.3 7.4 7.6 7.7 7.9
```

```
ply_by(iris,.f = psych::describe)
```

```
## [1] "setosa"
##    vars  n mean   sd median trimmed mad min max range skew kurtosis
  se
## X1    1 50 5.01 0.35      5       5 0.3 4.3 5.8   1.5 0.11    -0.45
0.05
## [1] "versicolor"
##    vars  n mean   sd median trimmed  mad min max range skew kurtosis
   se
## X1    1 50 5.94 0.52    5.9    5.94 0.52 4.9   7   2.1  0.1    -0.69
 0.07
## [1] "virginica"
##    vars  n mean   sd median trimmed  mad min max range skew kurtosis
   se
## X1    1 50 6.59 0.64    6.5    6.57 0.59 4.9 7.9     3 0.11     -0.2
 0.09
```

```
ply_by(iris,pastecs::stat.desc,basic =F,
                   desc = F,
                   norm = T)
```

```
## [1] "setosa"
##   skewness   skew.2SE   kurtosis   kurt.2SE normtest.W normtest.p
##  0.1129778  0.1678217 -0.4508724 -0.3405852  0.9776985  0.4595132
## [1] "versicolor"
##    skewness    skew.2SE    kurtosis    kurt.2SE  normtest.W  normtes
t.p
##  0.09913926  0.14726538 -0.69391378 -0.52417661  0.97783568  0.46473
704
## [1] "virginica"
##   skewness   skew.2SE   kurtosis   kurt.2SE normtest.W normtest.p
##  0.1110286  0.1649263 -0.2032597 -0.1535407  0.9711794  0.2583147
```

It is working, but is bumpy,because R package developments are very decentralized and lots of the algorithms has been written for some time with the old conventions.

Sometimes it becomes hard to work with these old conventioned packages -they don't even work in a reproducible way. If this happens, maybe go to their source for help- you know how to build things with packages? alright, I know how packages are built lol.

Operators, functors and Data pipelines?

The package dplyr in tidyverse offers a better way of facetting.

We can use data pipeline to create something that is similar to the idea of "Pivot table" in Excel.

```
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

iris %>% arrange(Species) %>% group_by(Species) %>%
  summarize(Count = n(),Min = min(Petal.Length),LQ = quantile(Petal.Len
gth,.25),Median = median(Petal.Length,.25),Mean = mean(Petal.Length),Sk
ew = Skewness(Petal.Length),UQ = quantile(Petal.Length,.75),Max =max(Pe
tal.Length))

## # A tibble: 3 x 9
##   Species    Count   Min    LQ Median  Mean   Skew    UQ   Max
##   <fct>      <int> <dbl> <dbl>  <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1 setosa        50   1     1.4   1.5   1.46  0.100  1.58   1.9
## 2 versicolor    50   3     4     4.35  4.26 -0.571  4.6    5.1
## 3 virginica     50   4.5   5.1   5.55  5.55  0.517  5.88   6.9
```

which is very handy. The idea of pipeline makes it possible for us to compare not only data and aspects of which, but also certain model applied to which. I will save that in a future post.

Model a 1-var distribution

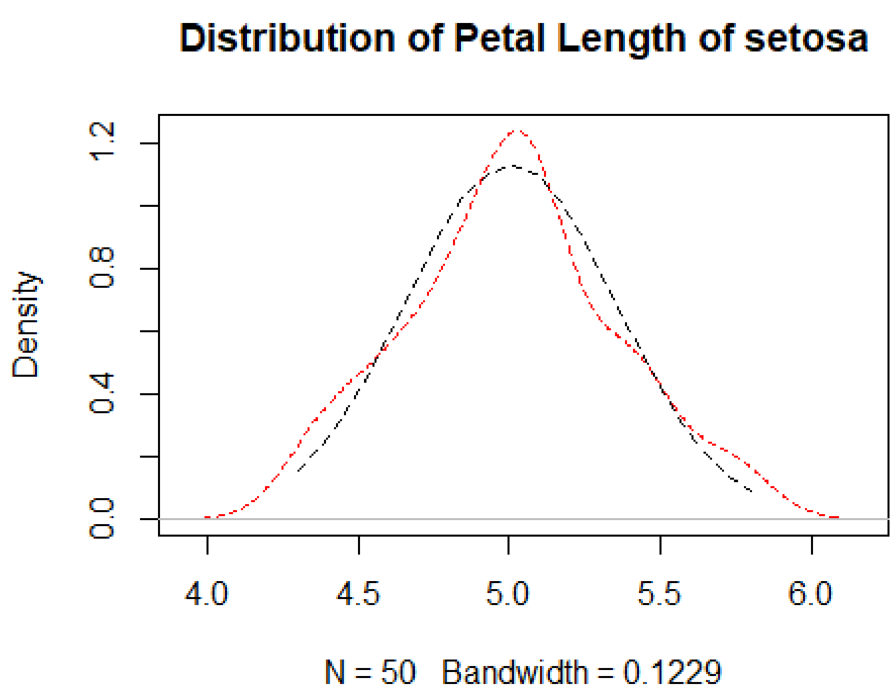
We can test if a 1-var distribution follow certain assumptions or not, recall that the code below give the likelyhood of a set of observations to follow the normal distribution.

```
ply_by(iris,pastecs::stat.desc,basic =F,
                   desc = F,
                   norm = T)

## [1] "setosa"
##   skewness   skew.2SE   kurtosis   kurt.2SE normtest.W normtest.p
##  0.1129778  0.1678217 -0.4508724 -0.3405852  0.9776985  0.4595132
## [1] "versicolor"
##    skewness    skew.2SE    kurtosis    kurt.2SE  normtest.W  normtes
t.p
##  0.09913926  0.14726538 -0.69391378 -0.52417661  0.97783568  0.46473
704
## [1] "virginica"
```

```
##    skewness    skew.2SE    kurtosis    kurt.2SE normtest.W normtest.p
##   0.1110286   0.1649263 -0.2032597  -0.1535407  0.9711794  0.2583147
```

As for the graph (we will use setosa as an example)

```
#hist(iris_setosa[,1],freq= F)
iris_setosa = subset(iris,Species=="setosa")
plot(density(iris_setosa[,1]),col = 'red',lty=2,main ="Distribution of
Petal Length of setosa")

#hence write a function that add normal density line to certain dataset

lines_dnorm = function(x,n=20){
  xbase = seq(from=min(x),to=max(x),length.out=n)
  lines(xbase,dnorm(xbase,mean = mean(x),sd =sd(x)),color='blue',lty=2)
}

lines_dnorm(iris_setosa[,1])

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "color"不是图
形参数
```



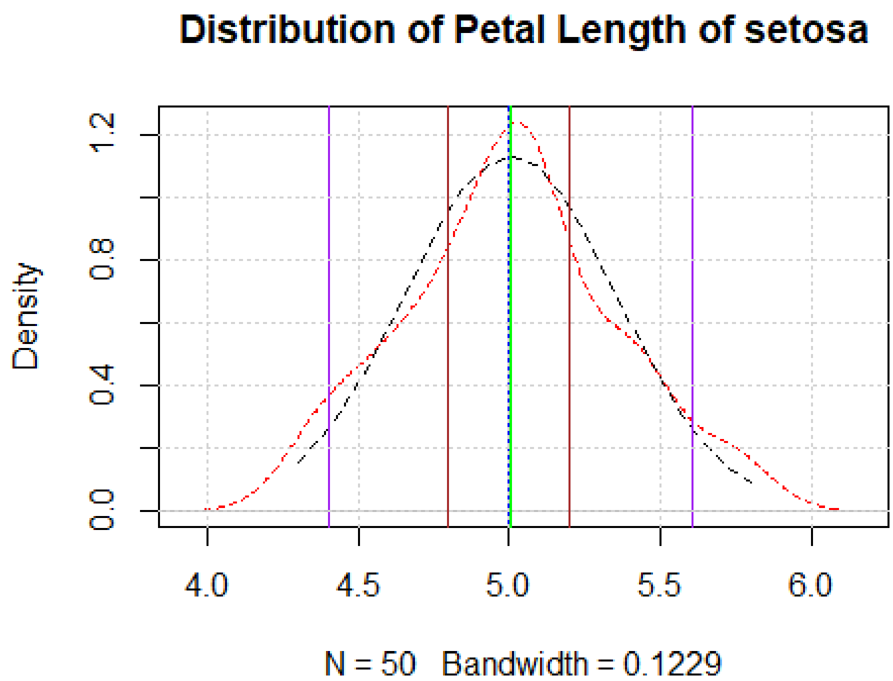
Add some vertical lines to help us read the plot.

```
# line up
# mid point
```

```
abline_summary = function(x,type = 'v'){
  L = list(list(mean(x),col = "green"),
           list(median(x),col = "blue"),
           list(quantile(x,.25),col = "brown"),
           list(quantile(x,.75),col = "brown"),
           list(quantile(x,.05),col = "purple"),
           list(quantile(x,.95),col = "purple"))
  for(i in 1:length(L)){
    x = L[[i]]
    names(x)[1] = type
    do.call(abline,x)
  }
}
```

Wrap the lines into a function, and test it – as always, it is important to know how to adjust the plots.

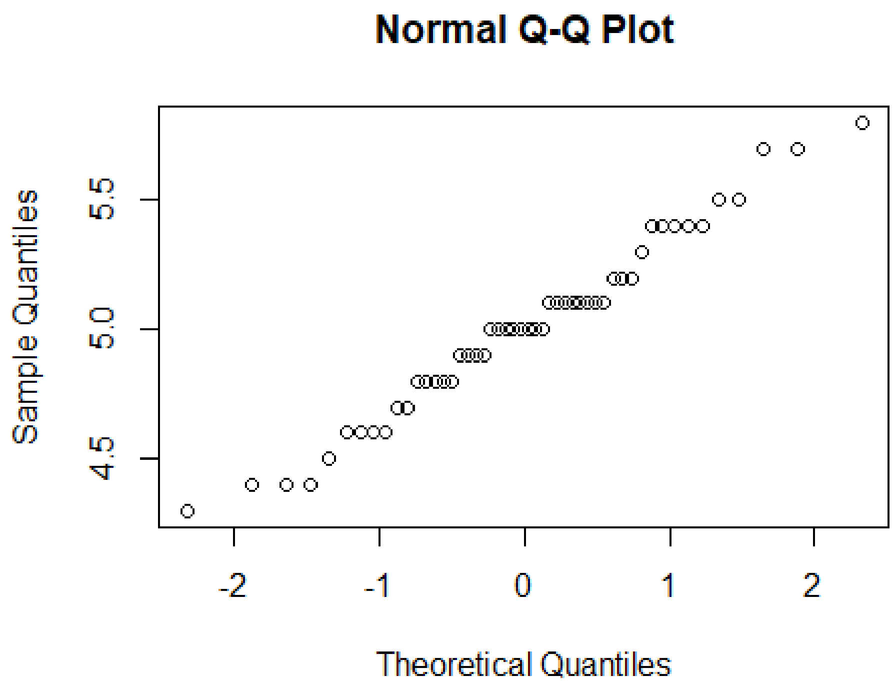
```
plot(density(iris_setosa[,1]),col = 'red',lty=2,main ="Distribution of Petal Length of setosa")

lines_dnorm(iris_setosa[,1])

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "color"不是图形参数

abline_summary(iris_setosa[,1])
#add grid
grid()
```



QQ plot

A qq plot is another way of visualizing how well the normal distribution applies. To use it, call qqnorm()

```
qqnorm(iris_setosa[,1])
```



This is simmilar to the cumulative density funcition but compared a "theoretical scale" with the origional one.