STAT 6210 - R Session 1

Contents Operations and Assignment
Vectors
Matrix Operations
WHILE and FOR Loops
Random Number Generation
Statistics
List of all functions

Operations and Assignment Let's start simple. The R Console can be used as a calculator.

```
2 + 5
```

```
## [1] 7
```

The `[1]` in front of the answer shows that `7` is the first (and the only element) of the answer.

In order to store a value, we need the assignment operator `<-` (*'alt + -'* in R Studio).

```
x  <- 2+5
x
```

```
## [1] 7
```

You can also use `=` instead of `<-`.

```
x  = 2+5
x
```

```
## [1] 7
```

We can also perform numerical and logical operations on `x`

```
x < 4
```

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

```
2 + x
```

```
## [1] 9
```

```
3 * x
```

```
## [1] 21
```

```
x ^ 2
```

```
## [1] 49
```

```
sqrt(x)
```

```
## [1] 2.646
```

Note that the last example uses a function called `sqrt()`. This function is applied to element `x`.

Vectors Function `c()` (**c**oncatanate or **c**ombine) creates vectors.

```
x  <- c(4,8,15,16,23,42)
x
```

```
## [1]  4  8 15 16 23 42
```

To refer to a specific element of a vector, brackets `[]` are used,

```
x[1]
```

```
## [1] 4
```

```
x[3]
```

```
## [1] 15
```

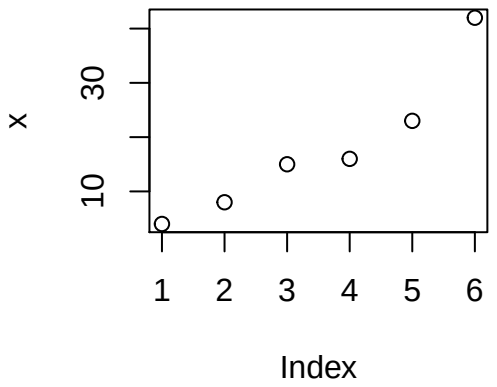
```
x[c(1,3,5)]
```

```
## [1]  4 15 23
```

The last line of code first creates a vector that contains 1, 3 and 5 and then R asks `x` for its 1st, 3rd and 5th value.

Objects can be plotted with the `plot()` function:

```
plot(x)
```



`sum()` returns the sum of the vector

```
sum(x)
```

```
## [1] 108
```

You can use `c()` to create character vectors:

```
y <- c("a","b","c","d")
y
```

```
## [1] "a" "b" "c" "d"
```

`length()` gives the length of a vector

```
length(x)
```

```
## [1] 6
```

```
length(y)
```

```
## [1] 4
```

You can obtain the list of variables in the *workspace* with `ls()`

```
ls()
```

```
## [1] "metadata" "x"        "y"
```

Variables can be removed from the *workspace* with `rm()`

```
rm(y)
ls()
```

```
## [1] "metadata" "x"
```

and the variable `y` is gone!

If you have a sequence of numbers and don't want to use the pesky `c()`, you can use `seq()`

```
y <- seq(from=1,to=5,by=0.2)
y
```

```
##  [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0 4.2
## [18] 4.4 4.6 4.8 5.0
```

```
y <- seq(from=1,to=5,length.out=21)
y
```

```
##  [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0 4.2
## [18] 4.4 4.6 4.8 5.0
```

The inputs don't need to be specified, the first three inputs for `seq()` are *from*, *to* and *by*. When we input three separate values to `seq()`, R automatically assigns them to the first three inputs.

```
y2 <- seq(1,9,1)
y2
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

If you want to create sequence that increases (or decreases) by 1, there is an even simpler way.

```
y3 <- 1:9
y3
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

We have created a bunch of variables, let's see how we're doing.

```
ls()
```

```
## [1] "metadata" "x"        "y"        "y2"       "y3"
```

Let's get rid of everything.

```
rm(list=ls())
ls()
```

```
## character(0)
```

If you need help about a function, you can use `?function_name` to learn about a function

```
?ls()
?rm()
```

R is a very kind program. You can even ask for examples if the help file was too confusing.

```
example(rm)
```

```
##
## rm> tmp <- 1:4
##
## rm> ## work with tmp  and cleanup
## rm> rm(tmp)
##
## rm> ## Not run:
## rm> ##D ## remove (almost) everything in the working environment.
## rm> ##D ## You will get no warning, so don't do this unless you are really sure.
## rm> ##D rm(list = ls())
## rm> ## End(Not run)
## rm>
## rm>
```

Matrix Operations Matrices are defined with `matrix()`.

```
z1 <- matrix(seq(1,9),nrow=3,ncol=3)
z1
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

The element in the ith row and jth column of a matrix can be accessed with `MATRIX_NAME[i,j]`

```
z1[1,1]
```

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

To get all of a row, instead specify which row you want and leave the column part empty.

```
z1[2,]
```

```
## [1] 2 5 8
```

Same thing works with columns too.

```
z1[,2]
```

```
## [1] 4 5 6
```

If you don't give all the values to the matrix, R fills them in by repetition.

```
z2 <- matrix(c(1,2),nrow=3,ncol=3)
```

```
## Warning: data length [2] is not a sub-multiple or multiple of the number
## of rows [3]
```

```
z2
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    1
## [2,]    2    1    2
## [3,]    1    2    1
```

R is (sometimes) smart at choosing a variable if you haven't specified it.

```
z3 <- matrix(1:9,nrow=3)
z3
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
z4 <- matrix(1:9,nrow=3,byrow=FALSE)
z4
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Matrix multiplication is done with `%*%`.

```
z2 <- matrix(11:19,3)
z1 %*% z2
```

```
##      [,1] [,2] [,3]
## [1,]  150  186  222
## [2,]  186  231  276
## [3,]  222  276  330
```

Matrices can be transposed with `t()`.

```
z1
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
t(z1)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

WHILE and FOR Loops The format for a for loop is a little tricky

```
z <- c(1,1)
for(i in 3:10){
  z[i] <- z[i-2] + z[i-1]
}
z
```

```
##  [1]  1  1  2  3  5  8 13 21 34 55
```

We can write the same with a while loop

```
z <- c(1,1)
i=3
while(i<=10){
    z[i] <- z[i-2] + z[i-1]
    i=i+1
}
z
```

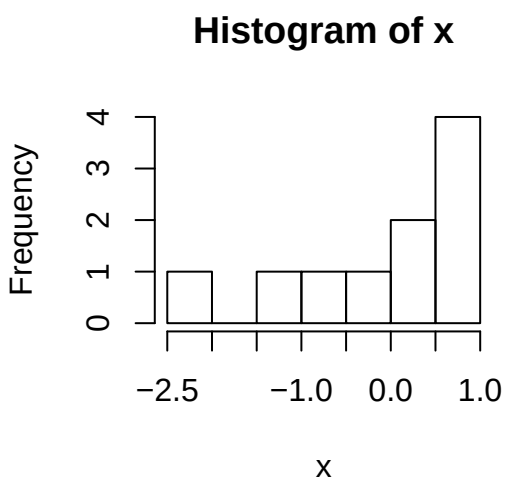
```
##  [1]  1  1  2  3  5  8 13 21 34 55
```

Random Number Generation R has very powerful random number generators

```
x <- rnorm(10)
x
```

```
##  [1] -0.53869  0.69514  0.65406  0.28678  0.76107  0.85893 -1.03110
##  [8] -0.01505  0.18896 -2.38331
```

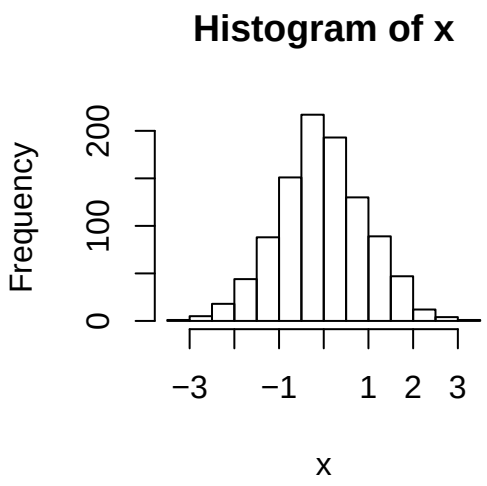
```
hist(x)
```



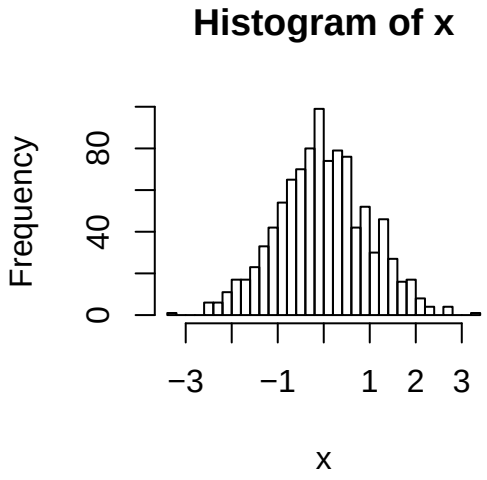

```
x <- rnorm(1000)
x[1:100]
```

```
##   [1]  1.379524 -0.403134  0.087249  0.462906  0.604907  0.870217 -0.423075
##   [8] -0.394995  0.093820  1.405180  0.654193 -0.916504  0.761086  0.959304
##  [15]  0.001726  0.340274 -0.554494  0.195878 -0.725937 -0.989684 -0.786974
##  [22]  0.497481  0.437422  1.333004 -3.235294  0.501140  0.956375 -2.029773
##  [29]  0.278103 -1.112843  1.188996 -1.829925  1.662965 -0.038749  1.390722
##  [36]  0.235049  0.606802 -0.703951  1.357443 -0.548342  1.331683  0.349580
##  [43]  0.960182  0.389032  0.382553 -0.795031 -0.107406 -1.285839  0.488176
##  [50] -1.207984 -1.640576  0.637015  0.423307  0.574349 -0.308135 -1.203502
##  [57] -1.001286 -2.379988 -0.036260 -0.242071 -0.066301 -0.109525 -2.237784
##  [64]  1.095019  0.153763  0.697193 -0.747272 -0.173100 -0.268248 -1.120967
##  [71] -0.619776 -0.576813  1.104547  0.155813  0.441698 -1.963651  1.968933
##  [78] -0.598574 -0.979164  0.433345  0.072733 -0.056555  0.666135  0.250563
##  [85] -0.566654 -0.124854 -0.805948 -0.409745  0.665064  0.318229  1.431088
##  [92]  0.467130 -0.263937  0.653186 -1.978011 -0.224637 -0.332582  0.652889
##  [99] -0.095242  1.506605
```

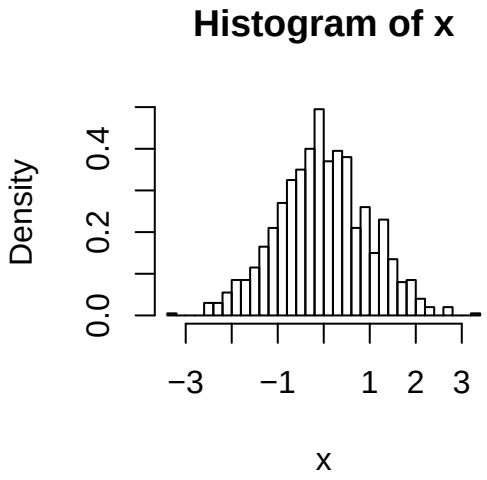
```
hist(x)
```




```
hist(x,40)
```



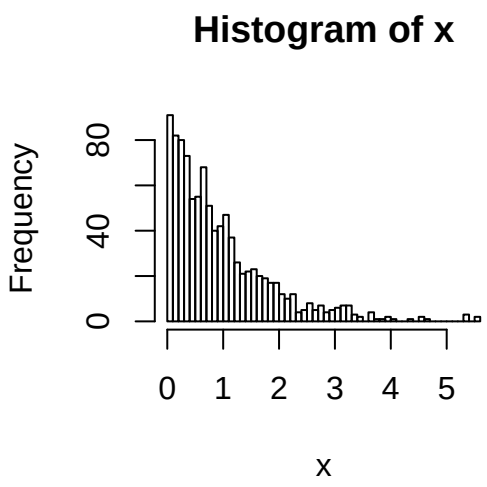

```
hist(x,40,freq=FALSE)
```



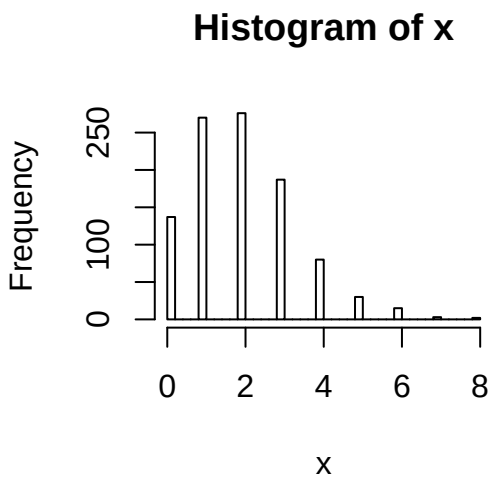
The **r** part of `rnorm()` is for **r**andom number and **norm** is for normal random variable.

Random variables from other distributions can be generated similarly

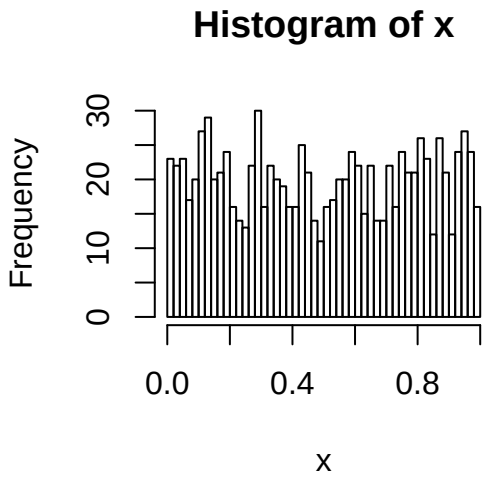
```
x <- rexp(1000)
hist(x,40)
```




```
x <- rpois(1000, lambda=2)
hist(x,40)
```




```
x <- runif(1000)
hist(x,40)
```



Besides random number generation, we can obtain pdf (probability density function) and cdf (cumulative density function) of distributions by changing "r" to "d" and "p", respectively

```
pnorm(1)
```

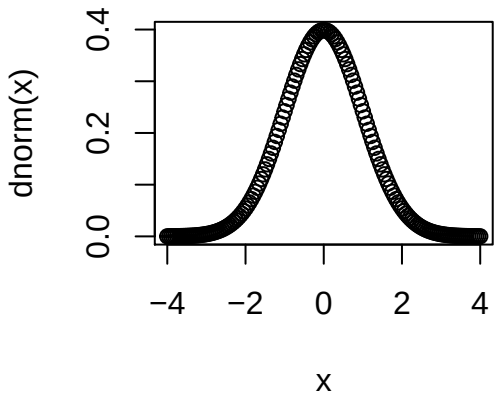
```
## [1] 0.8413
```

```
dnorm(.5)
```

```
## [1] 0.3521
```

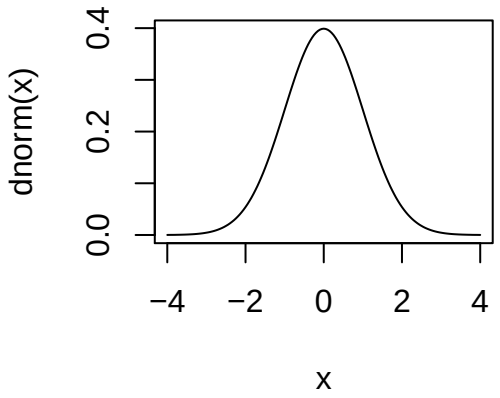
Let's plot the pdf of a standard Gaussian random variable.

```
x <- seq(-4,4,by=.05)
plot(x,dnorm(x))
```



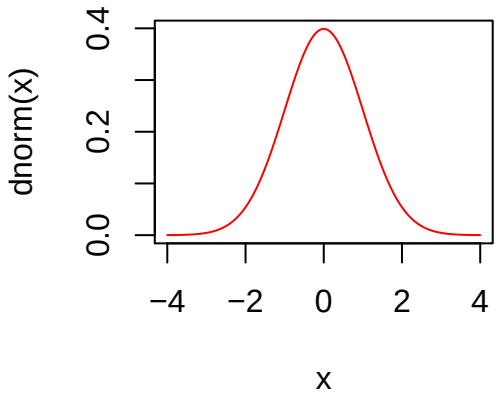
This is how you do a line plot:

```
plot(x,dnorm(x),type='l')
```

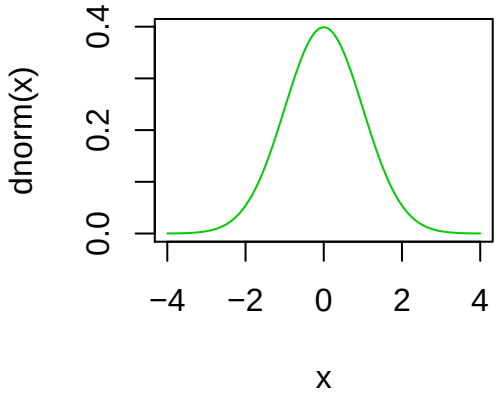


And you can always change the color

```
plot(x,dnorm(x),type='l',col=2)
```

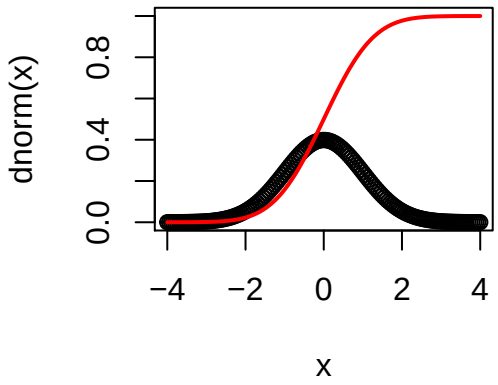



```
plot(x,dnorm(x),type='l',col=3)
```

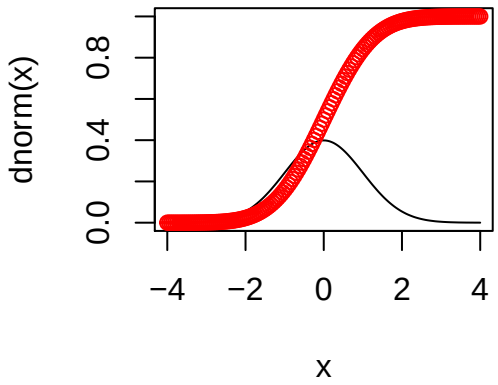


`plot()` generates a new plot and you can add a second layer using `points()` or `lines()`

```
plot(x,dnorm(x),ylim=c(0,1))
lines(x,pnorm(x),col=2,lwd=2)
```

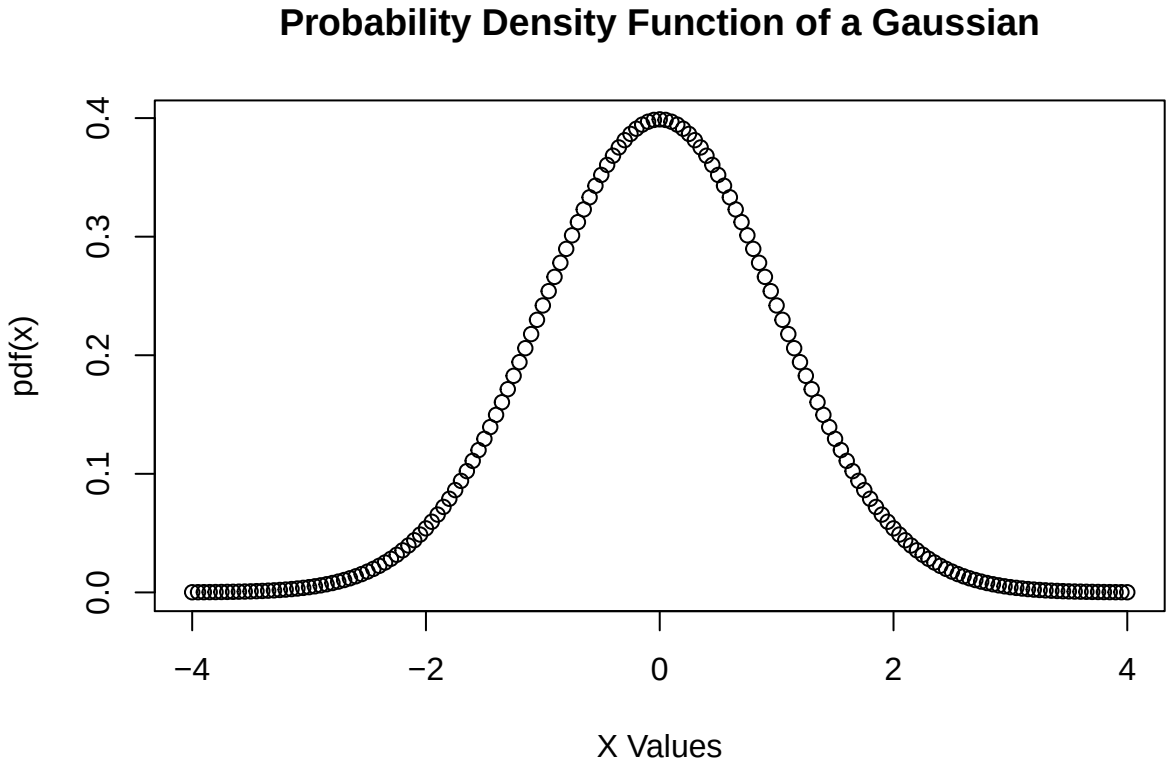



```
plot(x,dnorm(x),ylim=c(0,1),type='l')
points(x,pnorm(x),col=2)
```



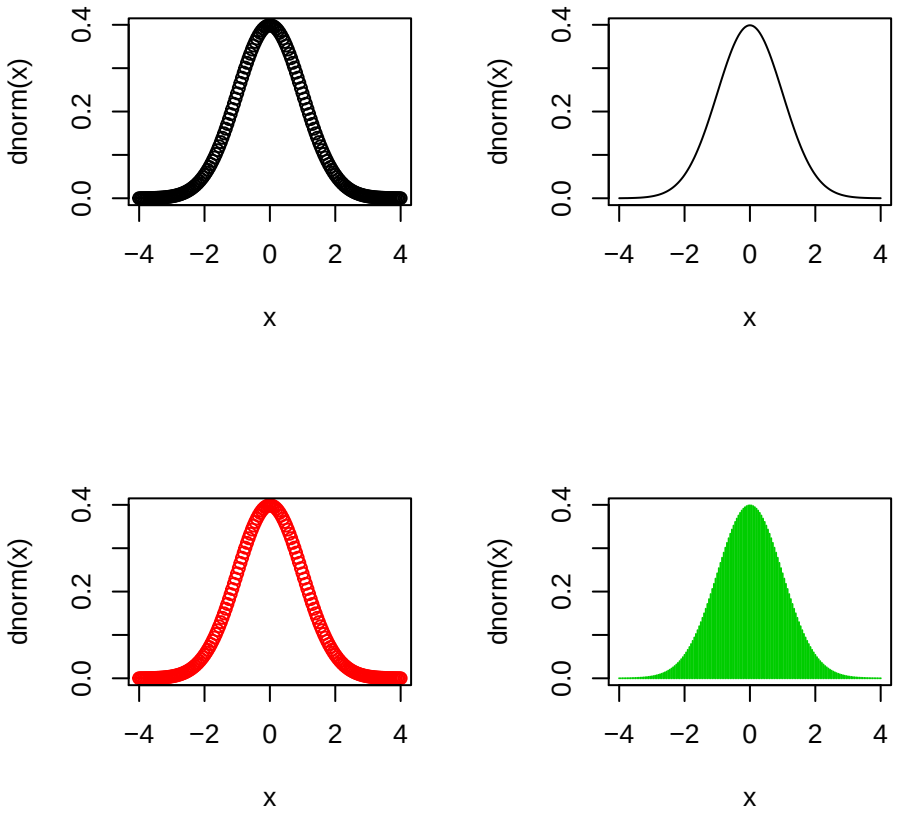
`plot()` has a lot of different settings.

```
plot(x,dnorm(x),main="Probability Density Function of a Gaussian", xlab="X Values", ylab="pdf(x)")
```



You can plot many graphs simultaneously by changing the display settings.

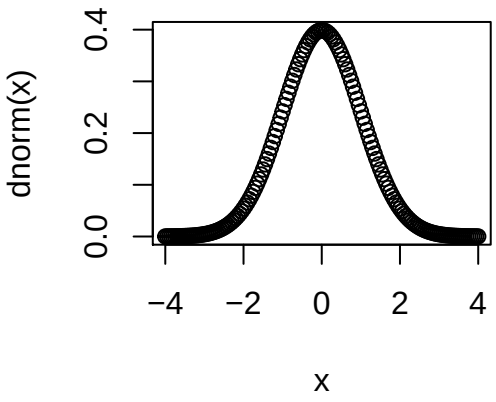
```
par(mfrow=c(2,2))
plot(x,dnorm(x))
plot(x,dnorm(x),type='l')
plot(x,dnorm(x),type='b',col=2)
plot(x,dnorm(x),type='h',col=3)
```



To go back to the one plot display, simply type:

```
par(mfrow=c(1,1))
```

```
plot(x,dnorm(x))
```



In addition to this, matrices and grids can be plotted with `image()`.

Statistics Let's draw a random sample.

```
x <- rnorm(100)
```

The mean is:

```
mean(x)
```

```
## [1] -0.1703
```

the median is

```
median(x)
```

```
## [1] -0.1608
```

the variance is

```
var(x)
```

```
## [1] 1.247
```

and the standard deviation is given by

```
sd(x)
```

```
## [1] 1.117
```

We can obtain the major quantiles (*quartiles*) with the `quantile()` function

```
quantile(x)
```

```
##      0%     25%     50%     75%    100%
## -3.0249 -0.9849 -0.1608  0.5837  2.8250
```

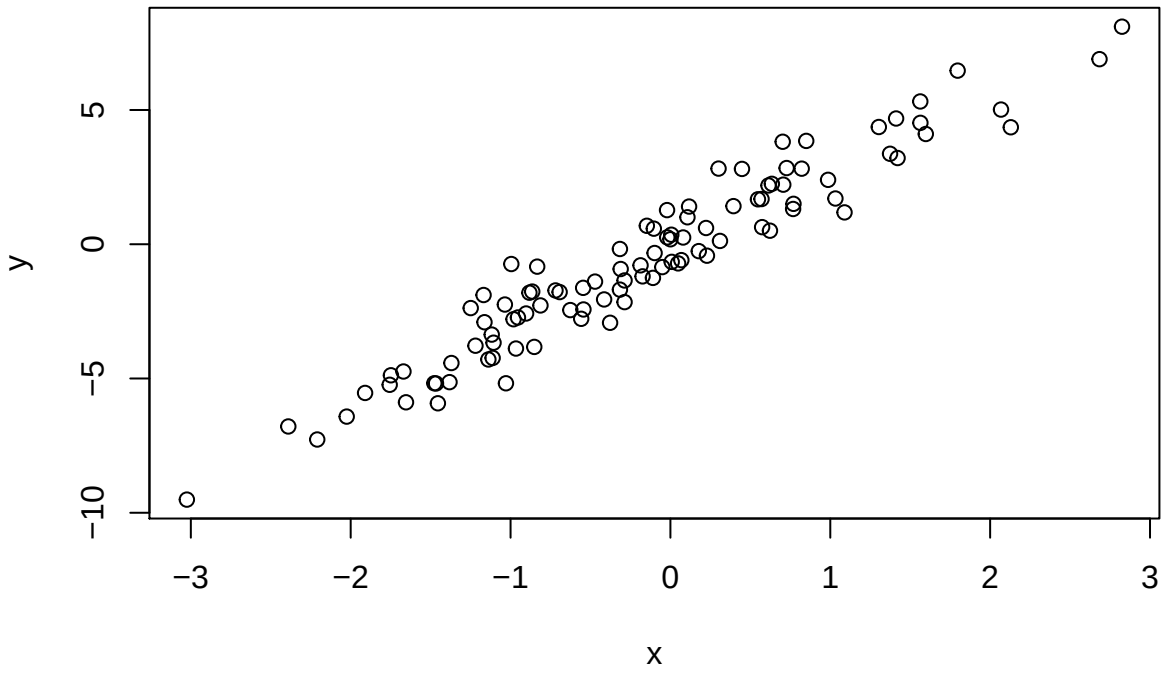
Instead, we can calculate the 15th, 30th and 50th percentiles,

```
quantile(x,c(.15,.30,.50))
```

```
##     15%     30%     50%
## -1.2240 -0.8696 -0.1608
```

To make things more interesting, let's form a new variable `y`. `y` is equal to 3 times `x` and some random noise.

```
y <- 3*x + rnorm(100)
plot(x,y)
```



Covariance of `x` and `y` is given by

```
cov(x,y)
```

```
## [1] 3.653
```

Correlation can be calculated with `cor()`

```
cor(x,y)
```

```
## [1] 0.9632
```

You should all know that

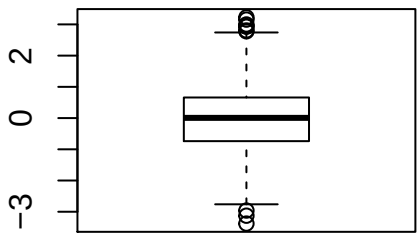
$$\text{Correlation}(x, y) = \frac{\text{Covariance}(x, y)}{\sqrt{\sigma_x^2 \sigma_y^2}}.$$

```
cov(x,y)/(sd(x)*sd(y))
```

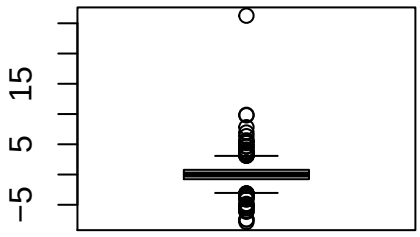
```
## [1] 0.9632
```

We can also obtain box-plots with the `boxplot()` function:

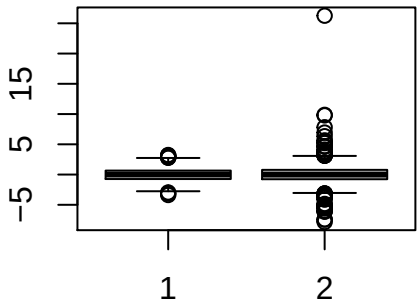
```
x <- rnorm(1000)
boxplot(x)
```



```
y <- rt(1000,df=3)
boxplot(y)
```




```
boxplot(x,y)
```



List of all functions

- `<-` : Assignment operator
- `c()` : Combine
- `sqrt()`: Square root
- `sum()`: Sum of vector
- `plot()`: Plots values
- `length()`: Length of a vector
- `ls()`: List of all elements in the *workspace*
- `rm()`: Removes an element from the *workspace*
- `seq()`: Makes a sequence of numbers
- `matrix()`: Creates a matrix
- `%\*%`: Matrix multiplication
- `t()`: Matrix transpose
- `rnorm()`: Generates random Gaussian variables
- `rexp()`: Generates random Exponential variables
- `rpois()`: Generates random Poisson variables
- `pnorm(z)`: Gives that probability that P(Z<z) where Z is a Gaussian variable
- `dnorm(z)`: Gives that probability density function of a Gaussian variable at value z
- Functions `ppois(), dpois()` and `pexp(), dexp()` are the equivalent of `pnorm()` and `dnorm()` for Poisson and exponential random variables. For more distributions, see the help file.

```
?Distributions
```

- `plot()`: Main plotting function
- `lines()`: Adds lines to the main plot

- `points()`: Adds points to the main plot
- `par(mfrow=c(1,2))`: Changes the plot settings to display two plots. More plots and a different style can be obtained by playing with the numbers 1, 2.
- `mean()`: Calculates mean of a vector
- `median()`: Calculates median of a vector
- `var()`: Calculates variance of a vector
- `sd()`: Calculates standard deviation
- `quantile()`: Calculates quantiles
- `cov()`: Covariance
- `cor()`: Correlation
- `boxplot()`: Creates a box-plot