Get start with R

Lea Lai

February 23, 2019

Contents

1 A start: Get use to R

(Partially credit to Nicole Kelbick, PhD. and introduction to R https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf)

Don't afriad to use R. It can be very simple. You can start by open R and type in only one line, and it will work. Try the following:

(The ones with ** are frequently used)

1.1 Common used operation or funtions

1.1.1 ** ":"

The operator ':' generates a sequence of integers.

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
10:1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

1.1.2 "<-" or "="

You can assign values to variables using '<-' OR '='.

```
x <- 5
x
```

```
## [1] 5
```

```
x=2
x
```

```
## [1] 2
```

1.1.3 “+” “-” "*" “/” “%%” These are basic arithmetic operations

```
x+5
```

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

```
x*5
```

```
## [1] 10
```

```
x/5
```

```
## [1] 0.4
```

```
x%%5 #give the remainder of x
```

```
## [1] 2
```

1.1.4 ** “seq(from,to,spacing)”

The ‘seq’ function generates a sequence of numbers with a specified spacing.
seq(from,to,spacing)

```
xn <- seq(1,10,.1)
xn
```

```
##  [1]  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3  2.4
## [16]  2.5  2.6  2.7  2.8  2.9  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9
## [31]  4.0  4.1  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.0  5.1  5.2  5.3  5.4
## [46]  5.5  5.6  5.7  5.8  5.9  6.0  6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9
## [61]  7.0  7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9  8.0  8.1  8.2  8.3  8.4
## [76]  8.5  8.6  8.7  8.8  8.9  9.0  9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8  9.9
## [91] 10.0
```

```
seq(1,10,length.out = 20) #use length.out to specify how many you need within the range
```

```
##  [1]  1.000000  1.473684  1.947368  2.421053  2.894737  3.368421  3.842105
##  [8]  4.315789  4.789474  5.263158  5.736842  6.210526  6.684211  7.157895
## [15]  7.631579  8.105263  8.578947  9.052632  9.526316 10.000000
```

1.1.5 “rev”

The ‘rev’ function reverses values of argument.

```
yn <- rev(xn)
yn
```

```
##  [1] 10.0  9.9  9.8  9.7  9.6  9.5  9.4  9.3  9.2  9.1  9.0  8.9  8.8  8.7  8.6
## [16]  8.5  8.4  8.3  8.2  8.1  8.0  7.9  7.8  7.7  7.6  7.5  7.4  7.3  7.2  7.1
## [31]  7.0  6.9  6.8  6.7  6.6  6.5  6.4  6.3  6.2  6.1  6.0  5.9  5.8  5.7  5.6
## [46]  5.5  5.4  5.3  5.2  5.1  5.0  4.9  4.8  4.7  4.6  4.5  4.4  4.3  4.2  4.1
## [61]  4.0  3.9  3.8  3.7  3.6  3.5  3.4  3.3  3.2  3.1  3.0  2.9  2.8  2.7  2.6
## [76]  2.5  2.4  2.3  2.2  2.1  2.0  1.9  1.8  1.7  1.6  1.5  1.4  1.3  1.2  1.1
## [91]  1.0
```

1.1.6 ** "c(elem1,elem2)"

The operator 'c'combines different elements into a vector

```
c(1,2)
```

```
## [1] 1 2
```

```
c("1",2) #the same as c("1","2"), they are all stored as strings.
```

```
## [1] "1" "2"
```

1.1.7 ** rep(arg1,n)

'rep(arg1, n)' repeats the first argument (arg1) n times

```
rep(2,7)
```

```
## [1] 2 2 2 2 2 2 2
```

```
y <- c(1, 3, 5.5, rep(2,7))
y
```

```
##  [1] 1.0 3.0 5.5 2.0 2.0 2.0 2.0 2.0 2.0 2.0
```

```
rep(c(1,3),3) # repeat 1,3 for 3 times
```

```
## [1] 1 3 1 3 1 3
```

```
rep(seq(1,3),2:4) # repeat 1,2,3 correspondingly for 2,3,4 times
```

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

1.1.8 Type casting: as.numeric and etc.

Change string to number; or change number to string

```
as.numeric("1") #when you add  " " , the content in the double quotation marks become strings
```

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

```
as.character(1)
```

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

1.1.9 ";"

The operator ";" can be used as a seperation for each command when writing on the same line

```
print(x);print(y)
```

```
## [1] 2
```

```
##  [1] 1.0 3.0 5.5 2.0 2.0 2.0 2.0 2.0 2.0 2.0
```

1.1.10 “mean”;“var”;“sd”,“median”

```
x=1:10;x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
mean(x) #Get average
```

```
## [1] 5.5
```

```
median(x) #Get median
```

```
## [1] 5.5
```

```
var(x) #Get variation
```

```
## [1] 9.166667
```

```
sd(x) #Get Standard deviation
```

```
## [1] 3.02765
```

1.1.11 “paste(elm1,elm2,sep)”

the paste(element1,element2,sep="") function combines elements into strings

```
paste("Day",1:10,sep="")
```

```
##  [1] "Day1"  "Day2"  "Day3"  "Day4"  "Day5"  "Day6"  "Day7"  "Day8"  "Day9"
## [10] "Day10"
```

```
paste("Day",1:10,sep="_")
```

```
##  [1] "Day_1"  "Day_2"  "Day_3"  "Day_4"  "Day_5"  "Day_6"  "Day_7"  "Day_8"
##  [9] "Day_9"  "Day_10"
```

```
paste(c("X","Y"), 1:10, sep="")
```

```
##  [1] "X1"  "Y2"  "X3"  "Y4"  "X5"  "Y6"  "X7"  "Y8"  "X9"  "Y10"
```

1.2 Logical expressions

1.2.1 or “|” “||”

When comparing single value, you may use “|” or “||”

```
x=2;y=3
y > 0 | x >= 3
```

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

```
y > 0 || x >= 3
```

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

When comparing a vector, you use “|” to gain results of comparison by array

```
a=1:3;b=2:4
a>b | a==b
```

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

"||" gives only a single logic value when comparing a vetor

```
a>b || a==b
```

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

1.2.2 and "&" "&&"

```
a>b & a==b
```

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

```
y > 0 && x >= 3
```

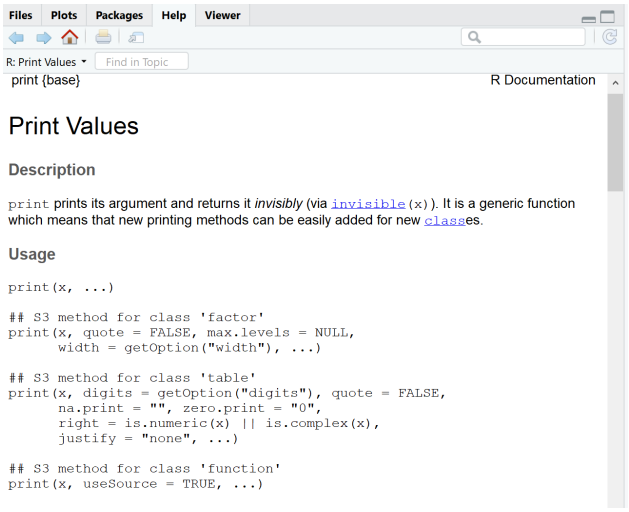
```
## [1] FALSE
```

1.3 About functions

1.3.1 Find help "?"

Find help/example/instruction for function, add a "?" before the function name:

```
?print
```



Use single quotes to get help on operators

```
?`:`
```

These helping information in the picture above will show up on the sidebar of R studio

1.3.2 "args": View arguments of function

To view the list of possible arguments a function can have use 'args'

```
args(png) #png is a function to export graph as png in your computer
```

```
## function (filename = "Rplot%03d.png", width = 480, height = 480,
##     units = "px", pointsize = 12, bg = "white", res = NA, ...,
##     type = c("cairo", "cairo-png", "Xlib", "quartz"), antialias)
## NULL
```

1.3.3 View the whole function

To see the whole function: type in the function name without "()" followed

```
var
```

```
## function (x, y = NULL, na.rm = FALSE, use)
## {
##     if (missing(use))
##         use <- if (na.rm)
##             "na.or.complete"
##         else "everything"
##     na.method <- pmatch(use, c("all.obs", "complete.obs", "pairwise.complete.obs",
##         "everything", "na.or.complete"))
##     if (is.na(na.method))
##         stop("invalid 'use' argument")
##     if (is.data.frame(x))
##         x <- as.matrix(x)
##     else stopifnot(is.atomic(x))
##     if (is.data.frame(y))
##         y <- as.matrix(y)
##     else stopifnot(is.atomic(y))
##     .Call(C_cov, x, y, na.method, FALSE)
## }
## <bytecode: 0x5633ab0483f0>
## <environment: namespace:stats>
```

1.3.4 Get/set Working dictionary: "getwd()" "setwd()"

Get current working dictionary:

```
getwd()
```

```
## [1] "/home/shulai/Insync/shulai@iu.edu/Google Drive/CAIDE Lab/R-tt"
```

Set current working dictionary:

```
setwd("C:/Users/naszh/Desktop")
```

As we started, (e.x.: a=c(1,2,...)) is a way to combine elelments and create vectors. There also are other ways to create vectors:

1.4 vector or array

Create vector or array

```
x=vector()
x[3]=10
x
```

```
## [1] NA NA 10
```

```
y=array()
y[4]=1
y
```

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

Difference between array and vector is that array can have more dimensions than vector:

```
array(dim=c(1,2))
```

```
##      [,1] [,2]
## [1,]   NA   NA
```

dim stands for dimension at here.

1.4.1 Assign names for a vector or array

Vectors can have names for each element, and array can have column names and row names

```
x=1:10 #x become a vector
names(x)=paste("X",1:10,sep="")
x
```

```
##  X1  X2  X3  X4  X5  X6  X7  X8  X9 X10
##   1   2   3   4   5   6   7   8   9  10
```

```
y=array(2:3,dim=c(1,2))
colnames(y)=c("col1","col2") #define column names
rownames(y)="row1" #define row names
y
```

```
##      col1 col2
## row1    2    3
```

1.4.2 Call elements in vector or array

call by names (use x and y value from above)

```
x["X1"] #don't forget to add " " on the name inside []
```

```
## X1
##  1
```

```
y["row1","col1"]
```

```
## [1] 2
```

Call by index number

```
#vector:
x=1:10
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
x[1]
```

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

```
x[1:2]
```

```
## [1] 1 2
```

```
#array:
y=array(1:20,dim=c(4,5))
y
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

```
y[2,3] #row2, column 3
```

```
## [1] 10
```

```
y[2, ] #present row 2
```

```
## [1]  2  6 10 14 18
```

1.4.3 Select only certain things in the array

```
x=1:10
x[x>5]
```

```
## [1]  6  7  8  9 10
```

"x>5" is a logical statement and give an array of logical vlaues:

```
x>5
```

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

This logical value array can be put into

##Dataframe

create dataframe:

```
d = data.frame("Feature1" = c(1,10,rep(12,2)))
d
```

```
##   Feature1
## 1        1
## 2       10
## 3       12
## 4       12
```

Call a column:

```
d$Feature1
```

```
## [1]  1 10 12 12
```

Add a column:

```
d = cbind(d, "Feature2" = c(10,29,9,2)) #cbind stand for column bind;
    #similarly, rbind is for row bind.
d
```

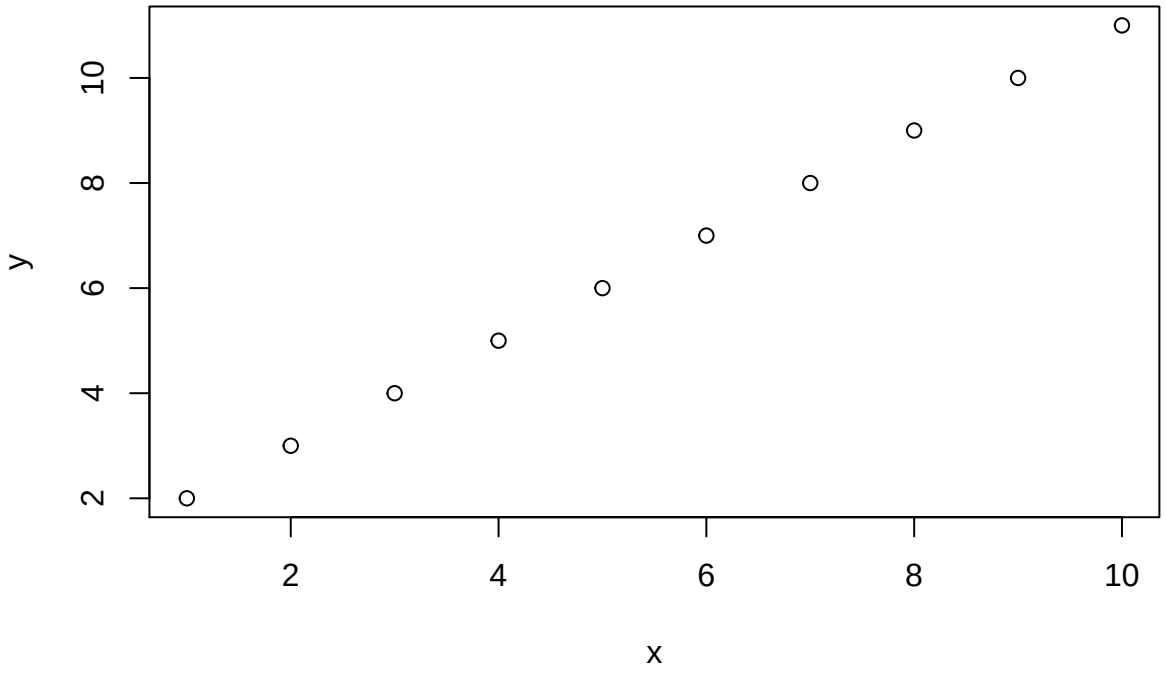
```
##   Feature1 Feature2
## 1        1       10
## 2       10       29
## 3       12        9
## 4       12        2
```

##Ploting

1.4.4 "plot"

You can plot data using the function 'plot'

```
x = 1:10;y=2:11
plot(x,y)
```



1.4.5 Export plot: "pdf", "png"

Export plot as a pdf (or other formats).

```
pdf(file="homework1_plot2.pdf", height=3, width=3)
plot(y,x)
plot(x,x)
dev.off()
```

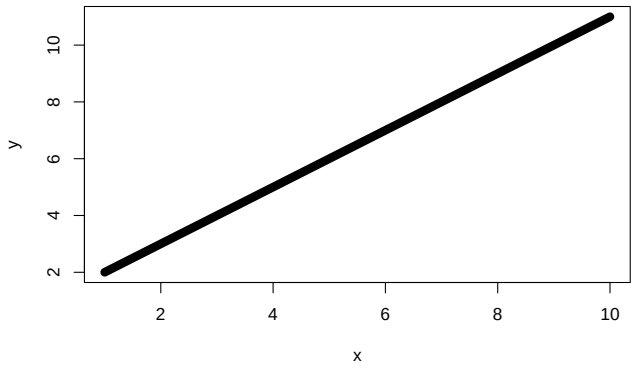
```
## pdf
##   2
```

```
  #Run these functions together
  #the first commend "pdf" startsthe graphics device to pdf,
  #and the following graphics would be produced in the pdf
  #When finish plotting, use dev,off to turns off the connection to the graphical device.
  #The file will show up in whatever your current working directory is.
```

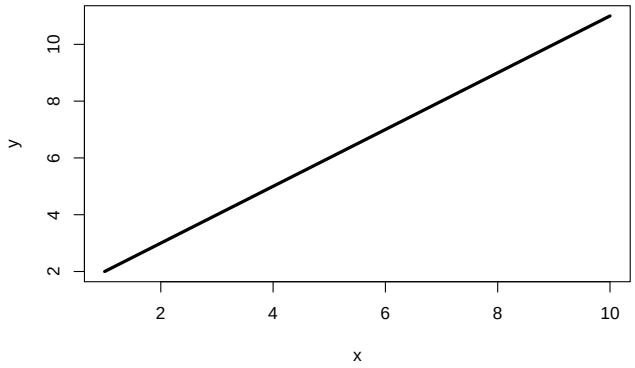
1.4.6 Change font size: "plot(. . . , lwd=)"

Use larger font for axis labels

```
plot(x, y, type='l', lwd=8)
```

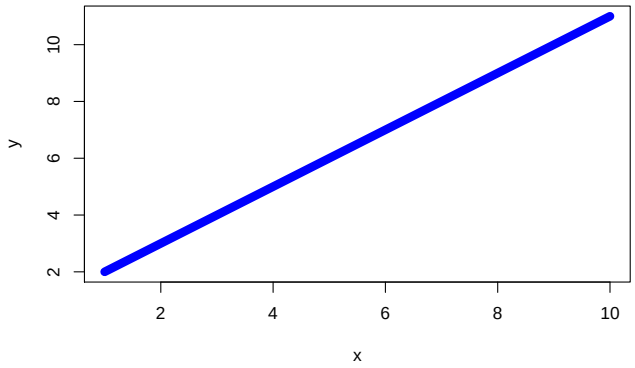



```
plot(x, y, type='l', lwd=3)
```



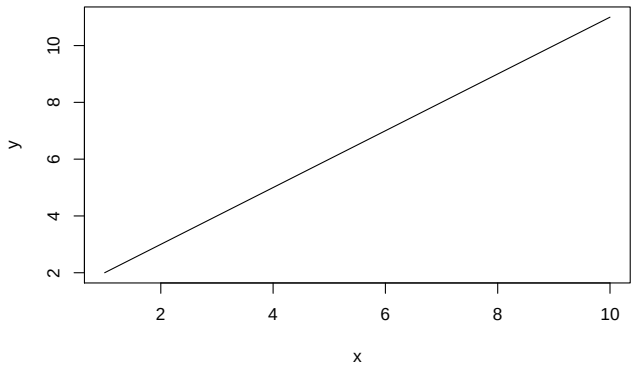
1.4.7 Change font color: "plot(... , col=)"

```
plot(x, y, type='l', lwd=8, col='blue')
```

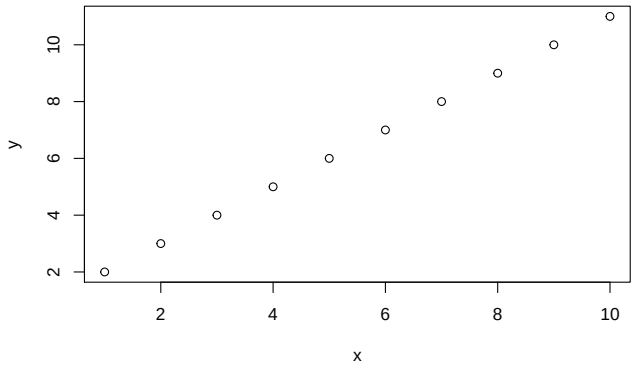


1.4.8 Change type of plot: "plot(..., type=)"

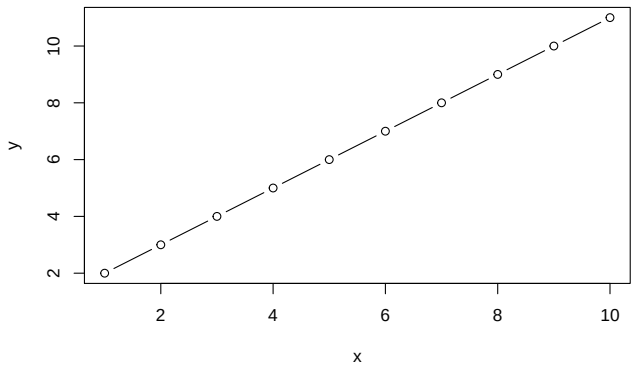
```r
plot(x, y, type='l') # "l" for lines
```




```r
plot(x, y, type='p') # "p" for points
```



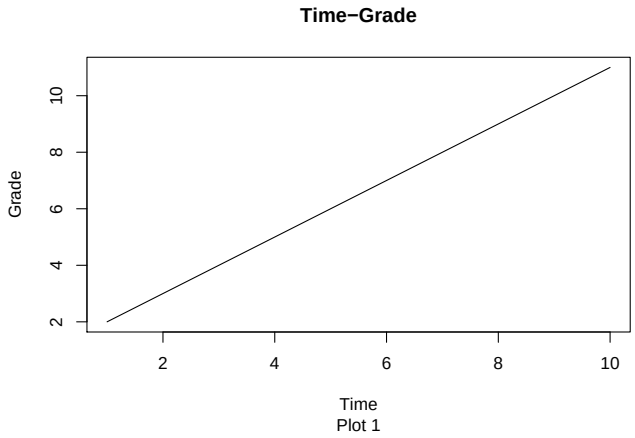

```r
plot(x, y, type='b') # "b" for both
```



More types usage see "?plot"

1.4.9 Change title/ lab names

```r
plot(x, y, type='l',xlab="Time",ylab="Grade",main="Time-Grade",sub="Plot 1")
```


Time−Grade
Grade
2
4
6
8
10
2
4
6
8
10
Time
Plot 1