Introduction to R

Valentina Staneva
vms16@uw.edu

June 29, 2015

R

R - The R Project for Statistical Computing
Main website: http://www.r-project.org/

- programming language based on S
- scripting language
- good for statistics and data analysis (but not limited to)
- free and open source

CRAN - The Comprehensive R Archive Network
Packages: http://cran.us.r-project.org/

Application Areas
Task Views: http://cran.r-project.org/web/views/

Some Links

- R intro
  http://cran.r-project.org/doc/manuals/R-intro.html
- R Cookbook
  http://www.cookbook-r.com/
- Quick-R
  http://www.statmethods.net/
- R Cheat Sheets
  http://cran.r-project.org/doc/contrib/Short-refcard.pdf
  http://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf

RStudio

RStudio is an IDE environment for R.

Main components:

- R Console
- File editor
- Files, Plots, Packages, Help
- Workspace
- History

Installation

1. Installing R: http://cran.us.r-project.org/
   - *try to remember the directory where you install R!*
2. Installing RStudio:
   http://www.rstudio.com/products/rstudio/download/
3. Start RStudio
   - *if needed, provide the R's installation path to RStudio*

Looking for help:

```
>help.start()
```

Setting your working directory:

```
>getwd()
>setwd("path_to_your_folder")
```

R Types

Basic types:

- numeric: integer - 1,3,-5; double - 0.5,-2.,3.5
- complex - i,1-2i
- character - "a","ABBA","?", "456"
- logical - TRUE/FALSE

Compound types:

- vectors (1D data)
- matrices (2D data)
- arrays (multi-dimensional data)
- lists (sequence of elements)
- dataframes (table-like data)
- factors (categorical data)

Creating Variables

Variable names can consist of letters, numbers, dot, underscore and need to start with a letter or dot not followed by a number.

```r
a <- 1
b <- "a"
.number <- "4"
under_score <- "_"
```

- creating vectors

  ```r
  myNumericVector <- c(1,3,4)
  myCharacterVector <- c("a","b","c")
  myMixedVector <- c("a",1,"c")
  ```

  Note: you can check class and type by:

  ```r
  class(myNumericVector)
  typeof(myNumericVector)
  ```

  Vectors need to contain elements of the same type!

Creating variables

- creating matrices

```
M<-matrix(c(1,2,3,4),nrow = 2,ncol = 2)
print(M)

##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
M<-matrix(c("h","w","e","o","l","r","l","l","o","d"),nrow = 2)
print(M)

##      [,1] [,2] [,3] [,4] [,5]
## [1,] "h"  "e"  "l"  "l"  "o"
## [2,] "w"  "o"  "r"  "l"  "d"
```

Arrangement is columwise!

More Ways to Create Vectors and Matrices

- sequence of integers:

```
v <- -1:5
```

- sequence of numbers:

```
v <- seq(from = 1,to = 10,by = 0.1)
```

- sequence of 100 zeros:

```
ones <- rep(0,100)
```

- sequence of random values:

```
r <- rnorm(10)
```

- 2x3 matrix of zeros:

```
M <- matrix(0,2,3)
```

- matrix with ones on the diagonal:

```
I <- diag(1,3)
```

- diagonal matrix:

```
D <- diag(c(1,2,3))
```

Concatenating Vectors and Matrices

- replicate a vector

```
vv <- rep(1:4,2)
```

- merge rows

```
M <- rbind(1:4,5:8)
```

- merge columns

```
M <- cbind(1:4,5:8)
```

- merge matrices

```
M<-cbind(M,M)
```

Elementwise Operations

`+,-,*,/,^`: addition, substraction, multiplication,division, power element by element for vectors or matrices of the same size.

Exercise:

- create two vectors of length 3
- what happens when you add one of those vectors to a 3x3 matrix?
- what happens when you add a vector of length 2 to the matrix?
- what happens when you add a vector of length 3 to 2x3 matrix?

Useful Functions

- `length(v)` - length of a vector
- `dim(M)` - dimensions of a matrix
- `length(M)` - length of a matrix
- `sum(v),sum(M)` - sum of all elements
- `min(v),max(M)` - min,max values
- `mean(v),mean(M)` - mean value
- `colSums(),rowSums(),colMeans(),rowMeans()`
- `var(),std()`
- `sqrt(),log(),exp(),sin(),cos(),...`

Note: to learn how to use a function check out the documentation:

```
>help(function_name)
>?function_name
```

Accessing Values in Vectors and Matrices

- extracting individual elements: `v[3]`, `M[1,2]`, `M[5]`
- extracting a subvector: `v[1:3]`
- rearranging the elements: `v[c(3,1,2)]`
- extracting the odd elements: `v[seq(1,length(v),2)]`
- extracting a submatrix: `M[1:2,1:3]`
- extracting a row: `M[2,]`
- extracting a column: `M[,3]`
- extracting first and last column: `M[,c(1,dim(M)[2])]`

Logical Operations

We can perform logical operations on the whole vector/matrix

```
A <- matrix(rnorm(9),3,3)
```

```
A<0

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

- negate statement

  ```
  !A<0
  ```

- set the negative values to zero:

  ```
  A[A<0] = 0
  ```

- find the locations of the zeros

  ```
  which(A==0)
  ```

Data Frames

Data Frames are powerful data structures for data in a table form:

```
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb

Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
```

Data Frame Example

R has some built-in datasets:

```
data()
data(mtcars)
summary(mtcars)
```

```
##       mpg             cyl             disp             hp
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
##       drat             wt             qsec             vs
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
##        am              gear            carb
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000
```

Useful Commands

- `colnames(mtcars)` - a vector of column names
- `rownames(mtcars)` - a vector of row names
- `dim(mtcars),nrow(mtcars),ncol(mtcars)` - dimensions of the data frame
- `length(mtcars)` - number of columns in the data frame
- `mtcars[5,4],mtcars[5:6,4:7]` - accessing elements
- `mtcars$mpg` - acessing columns
- `mtcars["Mazda RX4", "mpg"]`
- `mtcars[c("Mazda RX4","Mazda RX4 Wag"),]`
- `mtcars[,c("hp","mpg")]`

Note: rownames cannot be repeated, but colnames can ...

Creating Data Frames

We can have different types of variables in the columns.

- creating a data frame from vectors

  ```
  > numbers = c(1,2,3)
  > characters = c("a","b","c")
  > df = data.frame(numbers,characters)
  ```

- creating a data frame from a matrix

  ```
  > df = data.frame(M)
  ```

- creating an empty dataframe

  ```
  > df =
  data.frame(numbers=numeric(10),characters=character(10)))
  ```

- merging two data frames

  `df = rbind(df1,df2)` - need same colnames and types!

  `df = cbind(df1,df2)` - need same rownames!

- adding an extra column

  `df$new_name = values` - need correct size

- removing a column

  ```
  df$new_name <- NULL
  ```

Importing and Exporting Data

- store variables in .RData format (readable only by R)
- read, write spreadsheet-like files .csv (text file - very clean!)

```
data<-read.csv("matrix.csv")
```

```
write.csv(data,"matrix1.csv")
```

Importing and Exporting Data

- store variables in .RData format (readable only by R)
- read, write spreadsheet-like files .csv (text file - very clean!)

  ```
  data<-read.csv("matrix.csv")
  ```

  ```
  write.csv(data,"matrix1.csv")
  ```

Converting first column into row names:

```
data<-read.csv("matrix.csv",row.names = 1)
```

Using row names as an index:

```
write.csv(data,"matrix1.csv",row.names = FALSE)
```

FOR Loops

Example 1:

```
for (i in 1:10){
  print(i)
}
```

Example 2:

```
for (i in seq(1.5,10,0.5)){
  print(i)
}
```

Example 3:

```
for (i in c("a","b","c")){
  print(i)
}
```

IF Statements

Example 1:

```r
for (i in 1:10){
  if (i>5){
    print(i)
  }
}
```

Example 2:

```r
for (i in 1:10){
  if (i>5 && i<8){
    print("5<i<8")
  }
  else if (i<=5){
    print("i<=5")
  }
  else{
    print("i>=8")
  }
}
```

Functions

Functions in R can be stored as ‘variables’.

```
myFunction <- function(a,b){
    c = a + b
    return(c)
}
```

We can set default values:

```
myFunction <- function(a = 0,b = 0){
    c = a + b
    return(c)
}
```

and then we don’t have to specify all the values when in the call:

```
myFunction(3)

## [1] 3

myFunction(b=2)

## [1] 2
```

Functions with multiple outputs

Rule: one can return only one object!

Case 1: outputs are of same type:

```
 myFunction <- function(a,b){
    c = a + b
    d = a*b
    result <-c(c,d)
    names(result) = c("c","d")
    return(result)
 }
 result <- myFunction(1,3)
 print(sprintf('The sum is %d and the product is %d.',
               result["c"],result["d"]))

## [1] "The sum is 4 and the product is 3."
```

Functions with multiple outputs

Rule: one can return only one object!

Case 2: outputs are of different type:

```r
myFunction <- function(n){
   v = rnorm(n)
   M = matrix(rnorm(n*n),nrow = n)
   result <-list(v = v,M = M)
   return(result)
}
result <- myFunction(3)
result$v
result$M
```

A list is a sequence of any type of objects!

Apply Functions over Array Margins

`apply(X, MARGIN, FUN)` - apply function FUN to the first or second dimension of X
rowMeans:

```
apply(M,1,mean)
```

colMeans:

```
apply(M,2,mean)
```

You can easily vectorize your own functions!
`vapply` - for vectors
`lapply` - for lists
`sapply` - for vectors or lists
`mapply` - for multidimensional objects
`do.call` - similar to lapply

Exercise

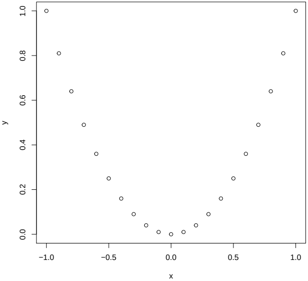
1) create a function which takes a vector which might have missing values, and returns a vector where the missing values are substituted with the mean of the remaining values.

```
NA2Mean <- function(v){
  #fill here
  return(v_new)
}
NA2Mean(c(1,NA,3))
  c(1,2,3)
```

2) Apply your function to each column of data

Basic Plotting

```
x = seq(-1,1,0.1)
y = x^2
plot(x,y)
```



- plotting lines:
  `plot(x,y,type = "l")`
- plotting points and lines:
  `plot(x,y,type = "o")`

More options

- changing color

```
plot(x,y,col = "green")
```

- typical colors: ”blue”,”black”,”red”,”yellow”,”orange”,...

```
colors()
```

- changing the symbol

```
plot(x,y,pch = 20)
```

- changing the linewidth

```
plot(x,y,lwd = 4)
```

More parameters

- `xlim = c(0,1), ylim = c(0,1)` (domain)
- `main, xlab, ylab` (the labels can be set with `title()`)
- `cex,cex.lab` (symbol size, label size)
- `text(),legend()`

Get parameters:

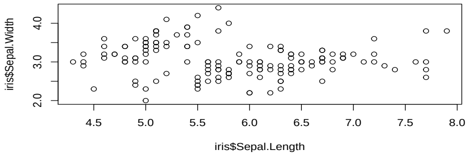
```
> par()
```

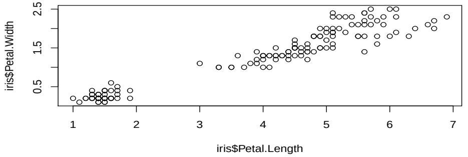
Set parameters:

```
> par(lwd = 5)
```

Iris Data Example

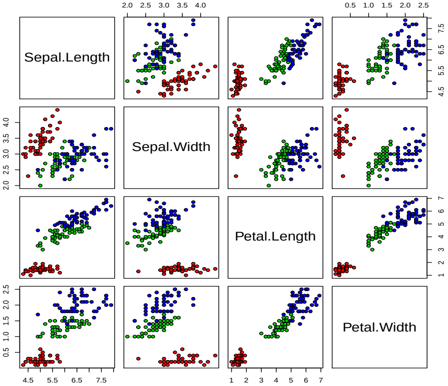
```
data(iris)
par(mfrow = c(2,1))
plot(iris$Sepal.Length,iris$Sepal.Width)
plot(iris$Petal.Length,iris$Petal.Width)
```





Iris Data Example

```
pairs(iris[1:4], pch = 21,
bg = c("red", "green3", "blue")[unclass(iris$Species)])
```

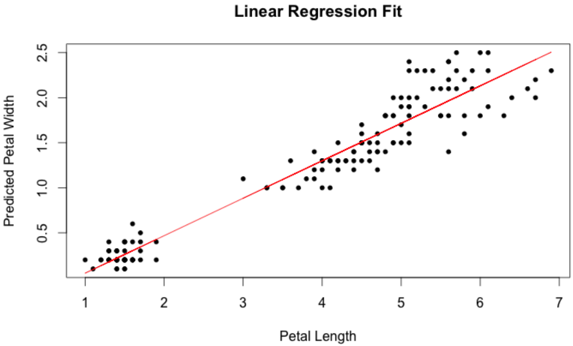


Iris Data Example

```
fit <- lm(Petal.Width ~ Petal.Length, data = iris)
summary(fit)

##
## Call:
## lm(formula = Petal.Width ~ Petal.Length, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56515 -0.12358 -0.01898  0.13288  0.64272
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.363076   0.039762  -9.131  4.7e-16 ***
## Petal.Length  0.415755   0.009582  43.387  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2065 on 148 degrees of freedom
## Multiple R-squared:  0.9271,Adjusted R-squared:  0.9266
## F-statistic:  1882 on 1 and 148 DF,  p-value: < 2.2e-16
```

```
predicted<-predict(fit)
plot(iris$Petal.Length,iris$Petal.Width,pch = 20,
     xlab = "Petal Length",ylab = "Predicted Petal Width",
     main = "Linear Regression Fit")
lines(Petal.Length,predicted,col = "red")
```



Exporting graphs

```
png("myPlot.png")
plot(x,y)
dev.off()
```

Supports `png`,`jpeg`,`svg`,`pdf`,`postscript`,...
You can set height, width, ...

Fancier graphs

- package: ggplot2 - the Grammar of Graphics
- book: *R Graphics Cookbook*, by Winston Chung (2012) (ebook in the library)
- tutorial: `http://www.cookbook-r.com/Graphs/`

Interactive graphs

- Plotly: `https://plot.ly/r/`
- rCharts: `http://ramnathv.github.io/rCharts/`
- Shiny apps: `http://shiny.rstudio.com`