Introduction to GNU R

Sven Laur

What is GNU R?

- It is a calculator
- It is a programming environment
- It is a collection of machine learning algorithms
- It is a tool for drawing graphics

Variables in GNU R

- By default all variables are vectors
- Vectors are indexed from 1 to length(x)
- Vector components have names
- Operations are carried out pointwise

```
a <- c(1:6)
names(a) <- c("A", "B", "C","D", "E", "F")
b <- c(1,2,3,3,3,3)
b + a
b * a
a^2
a[1]; a["A"]
a["Missing"]
sin(a)
```

Matrix magic

- Matrices are two dimensional arrays
- They can be formed by folding vectors
- They can be formed by outer products of vectors
- Real matrix multiplication is denoted by %*%

```
a <- c(1:12)
dim(a) <- c(3,4)
a;dim(a)
rbind(c(1:3), c(2:4))
cbind(c(1:3), c(2:4))
c(1:5) %o% c(1:5)
a[1,];a[,1]
a[1:3,c(2,3)]
a[,1:3] %*% a[,1:3]
```

Data frames

- Data frame is a table and matrix at the same time
- Dedicated operation for accessing columns
- You can do SQL operations with different syntax

```
a <- c(1:3)
b <- c("Alex", "Bob", "Eve")
tbl <- data.frame(a,b)
tbl[["Grade"]] <- c(1,1,4)
colnames(tbl) <- c("DBKey","Name", "Grade" )
tbl
subset(tbl, DBKey %in% c(1,3))
tbl1 <- tbl[c("DBKey", "Name")]
tbl2 <- tbl[c("DBKey", "Grade")]
merge(tbl1, tbl2, by = "DBKey" )
```

How to read data

- Function read.table parses text files
- Function read.cvs parses cvs files
- Functions load and save are for R specific files
- Pay attention to working directory

```
getwd();setwd("~/Downloads")
tbl <- read.csv("iris.data")
cols <-  c("S.Length", "S.Width", "P.Length", "P.Width", "Class")
colnames(tbl) <- cols
head(tbl); tail(tbl)
nrow(subset(tbl, Class == "Iris-setosa"))
aggregate(tbl$S.Width,tbl["Class"], length)
```

Aggregation magic

- By default functions are applied to all elements
- Function apply aggregates in one dimension
- Function mapply is sometimes useful, as well
- Function aggregate does most of the magic
- The package plyr contains more advanced magic

```
head(sin(tbl[-5]))
apply(tbl[-5], 2, sum)
apply(tbl[-5], 2, mean)
aggregate(tbl[-5], tbl[5], mean)
aggregate(tbl[c(1,2)], tbl[5], mean)
summary(tbl)
```

Basic plotting

- Function plot does basic plotting
- There are many additional arguments
- Special plots come with special functions
- Best results can be obtained with ggplot2

```
colors <- c("Red", "Green", "Blue")
names(colors) <- unique(tbl$Class)
plot(tbl$S.Length, tbl$S.Width, col = colors[tbl$Class])
plot(sort(tbl$S.Length), sort(tbl$S.Width), type = "l",
main = "QQ-plot", xlab = "Sepal length (cm)", ylab = "Sepal width (cm)")
points(sort(tbl$S.Length), sort(tbl$S.Width), pch=20)
pdf("test.pdf")
plot(tbl$S.Length, tbl$S.Width, col = colors[tbl$Class])
dev.off()
```

How to program in GNU R

- Basic scripting is done command by command
- The resulting file should have .R or .r extension
- Function source executes these scripts
- Function debug allows interactive debugging
- Try to avoid for cycles. They are slow
- Draw data shapes to verify what you are doing

```
x <-rep(NA, 100)
for(i in 1:100) x[i] <- 1/sin(i)
xx <- 1/sin(c(1:100))
print(x - xx)
```

If-then-else block

- If-then-else block may fail for mysterious reasons
- If-then-else block fails with vector arguments
- Positioning of curly brackets is important
- Use ifelse command for vectors and matrices
- Use conditional indexing instead

```
if(NA == 1) print("If branch") else print("Else branch")
if(NaN == 1) print("If branch") else print("Else branch")
if(c(1,2) == 1) print("If branch") else print("Else branch")
ifelse(c(NA,1,2) == 2, print("If branch"), print("Else branch"))
ifelse(c(NA,1,2) == 2, "If branch", "Else branch")
x <- c(1:5)
x[x > 3] <- -5; x
```

Safe loop constructions

- Loop syntax is similar to Python
- Standard for-cycle is not safe
- Use seq_len(x) and seq_along(x) instead

```
for(i in x) print(i)
for(i in 1:5) print(i)
for(i in 3:1) print(i)
for(i in seq_len(0)) print(i)
for(i in seq_along(x)) print(i)
for(i in seq_along(c())) print(i)
for(i in c(1:5)){
  if(i>3) next;
  print(i)
}
```

Functions

- Function are defined by assignment
- Function function defines a function
- The last evaluated expression is the return value
- Local variables shadow globals.
- There is a special super assignment <<- for globals

```
x <- 3
f1 <- function(a) x
f2 <- function(a) x <- a
f3 <- function(a) x <<- a
x; f1(1); x
x; f2(5); x
x; f3(5); x
```

Compulsory factorial

```r
f<- function(x){
  if(x == 0) 1
  else x*f(x-1)
}
```

```r
f <- function(x){
  if(x == 0) 1
  else x*(function(x){f(x)})(x-1)
}
```