Introduction to R

Kevin Lee

Department of Statistics
Western Michigan University

Outline

Outline

What is R?

- R is a GNU project (free-software, mass-collaboration project).
- To find out about R go to `http://www.R-project.org/`.
- Check also the NY Times article
  `http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?pagewanted=all`.

How to Install R?

- To download R go to `http://cran.r-project.org/`.
- You may also want to install Rstudio. To download Rstudio go to `http://www.rstudio.com/`. But I will not use Rstudio in class.

Assignments

- R has symbolic variables, that can be used to represent values.
  - `x <- 2 # recommend`
  - `x = 2`
- Names of variables can be chosen quite freely in R. Letters, digits, underscore, and period (dot) can be used.
  - `height.1yr # can be used to describe the height of a child at the age of 1 year`
- Names are case-sensitive.
  (e.g. `Height` and `height` do not refer the same variable)
- Try not to use variable names same as popular basic function names.
  (e.g. `mean` and `diff`)

Vectorized Arithmetic

- R can handle entire data vectors as single objects.
- The construct `c()` is used to define vectors.
  - `weight <- c(60, 72, 57, 90, 95, 72) # (kg)`
  - `height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91) # (m)`
- You can do calculations with vectors just like ordinary numbers, as long as they are of the same length.
  - `BMI <- weight/height^2`
- Note that the operation is carried out element wise.
  (that is, the first value of `BMI` is 60/1.75^2)
- Try calculate the mean of weight ($\bar{x}$ = sum of all weights / number of observations) and compare the result with using `mean()` function.
  (Hint: use `sum()` and `length()`)

Graphics

- One of the most important aspects of the presentation and analysis of data is the generation of proper graphics.
- To check the relation between `weight` and `height`, the first idea is to plot one versus the other.
  - `plot(height, weight)`
  - `plot(height, weight, pch = 2) # pch (plotting character)`
- To add horizontal line or vertical line to an existing plot use function `abline()`
  - `abline(h = 70, col = "blue") # horizontal line`
  - `abline(v = 1.85, col = "red") # vertical line`

Outline

Outline

Vectors

- A generic function of constructing vectors is `c()`, "concatenate".
- A numeric vector can take the numbers.
  - `c(1, 3, 5)`
- A character vector can take the text strings, whose elements are specified and printed in quotes.
  - `c("Messi", "Ronaldo", "Hazard")`
- A logical vector can take the value `TRUE` or `FALSE`.
  - `c(TRUE, FALSE, FALSE)`
  - `c(T, F, F)`
- All elements of a vector have the same type. If you create a vector of different types, they will converted to the least "restrictive" type.
  - `c(1, "Messi")`

Other Functions to Create Vectors

- `seq()`, "sequence", is used for equidistant series of numbers.
  - `seq(4, 10)`
  - `seq(4, 10, 2) # a sequence in jumps of 2`
  - `seq(1.65, 1.90, 0.05) # a sequence in jumps of 0.05`
- `rep()`, "replicate", is used to generate repeated values.
  - `# Second argument is a number`
  - `rep(2, 5)`
  - `vec1 <- c(1, 2)`
  - `rep(vec1, 5)`
  - `# Second argument is a vector`
  - `rep(vec1, c(5, 10))`
  - `rep(vec1, c(10, 10))`
  - `rep(vec1, each = 10) # special case where there are equally many replications of each value`

Matrices

- A matrix in mathematics is just a two-dimensional array of numbers.
- Matrices are used a lot for numerical computation purpose.
- `matrix()` is used to construct a matrix.
  - `mat.elem <- seq(1,6)`
  - `mat1 <- matrix(mat.elem, nrow = 2, ncol = 3)`
  - `mat2 <- matrix(mat.elem, nrow = 2, ncol = 3, byrow = T)`
    `# the matrix to be filled in a rowwise`
- `rownames()` is used to name the rows of the matrix.
- `colnames()` is used to name the columns of the matrix.
  - `rownames(mat1) <- c("Male", "Female")`
  - `colnames(mat1) <- c("A", "B", "C")`
- `dim()` is mostly used to check the dimension of the matrix.
  - `dim(mat1)`

Matrices

- You can "glue" vectors together to make a matrix.
- `rbind()` is used to glue vectors rowwise.
- `cbind()` is used to glue vectors columnwise.
  - `vec1 <- c(1, 2, 3)`
  - `vec2 <- c(4, 5, 6)`
  - `rbind(vec1, vec2)`
  - `cbind(vec1, vec2)`

Factors

- Factors are used to store categorical (or qualitative) variables. (e.g. gender, nationality, and eye color)
- `factor()` is used to construct a factor.
  - `gender <- factor(c("Male", "Female", "Male", "Male", "Female", "Female"))`
- You may specify the levels by yourself by setting `levels` argument.
  - `nationality <- factor(c("uk", "us", "au", "uk", "us", "us"), levels = c("us", "fr", "au", "uk"))`

Lists

- Lists are used to combine collection of objects (e.g. vectors, factors, etc.) into a larger composite object.
- `list()` is used to construct a list.
  - `pre <- c(5260, 5470, 5640, 6180, 6390, 6515)`
  - `post <- c(3910, 4220, 3885, 5160, 5645, 4680)`
  - `my.list <- list(before = pre, after = post)`
- The components of the list are named according to the argument names used in `list()`. To extract the named components:
  - `my.list$before`
  - `my.list$after`
- Many of R's built-in functions return their results in the form of list.

Data Frames

- Almost every real data set is stored in the form of data frame.
- Each row of a data frame is an observation, each column of a data frame is a variable.
- `data.frame()` is used to construct data frames.
  - `my.df <- data.frame(pre, post)`
- To extract the components of data frame:
  - `my.df$pre`
  - `my.df$post`

Difference between Lists and Data Frames

- Lists are the most flexible data structure in R.
- Lists have no restriction on the class, length or structure of each element.
- Data frames are lists as well, but they have a few restrictions:
  - All elements are vectors.
  - All elements have an equal length.
- These restrictions are resulting two-dimensional structure.
- Data frames can follow some of the behavior of matrices. (e.g. select rows, columns, and elements)

Outline

Indexing of Vectors

- Brackets are used for selection of data, also known as indexing.
  - `pre[3] # third element of vector`
- It can be also used to modify elements of vector.
  - `pre[3] <- 6000`
  - `pre[3] <- 5640`
- If you want a subvector then you can index with a vector:
  - `pre[c(1,3,5)]`
  - `pre[1,3,5] # wrong!`
- You can also use index vector stored in a variable.
  - `v <- c(1, 3, 5)`
  - `pre[v]`

Indexing of Matrices

- Matrices may be indexed by giving two indices in the form,
  - `my.mat <- cbind(pre, post)`
  - `my.mat[3,2] # my.mat[row, column]`
  - `my.mat[5,1] # fifth row and first column of matrix`
- It can be also used to modify elements of matrix.
  - `my.mat[3,2] <- 3575`
  - `my.mat[3,2] <- 3885`
- If you want a whole row or column from the matrix:
  - `my.mat[2, ] # second row of matrix`
  - `my.mat[, 1] # first column of matrix`
- If you want a submatrix then you can index with a vector:
  - `my.mat[c(3,4), ]`
  - `v <- c(3, 4)`
  - `my.mat[v, ]`

Indexing of Data Frames

- Data frames may be indexed by giving two indices in the form similar to matrix,
  - `my.df <- data.frame(pre, post)`
  - `my.df[1,2]`
- It can be also used to modify elements of data frame.
  - `my.df[4,2] <- 4760`
  - `my.df[4,2] <- 5160`
- If you want a whole row or column from the matrix:
  - `my.df[2, ] # second row of data frame`
  - `my.df[, 2] # second column of data frame`
  - `my.df$post # second column of data frame`

Negative Indexing

- Indexing is a special powerful ability of R.
- R can also do negative indexing.
- You can get all observations except numbers (or vectors) you specify.
- Negative indexing for vectors
  - `pre[-3]`
  - `pre[-c(1, 3, 5)]`
- Negative indexing for matrices
  - `my.mat[-3, ]`
- Negative indexing for data frames
  - `my.df[-3, ]`

Outline

Conditional Selection (Vectors)

- In practice, you often need to extract data that satisfy certain criterion.
  - `post[post < 4000]`
  - `post[pre > 6000]`
- `which()` function will return the position of the elements in a logical vector which are `TRUE`.
  - `post < 4000`
  - `which(post < 4000)`
- Comparison operators in R:
  - `< # less than`
  - `> # greater than`
  - `== # equal to`
  - `<= # less than or equal to`
  - `>= # greater than or equal to`
  - `!= # not equal to`

Conditional Selection (Vectors)

- In practice, you also need to extract data satisfying several criteria.
  - `post[post < 4000 | post > 5500]`
  - `post[pre > 6000 & pre <= 6400]`
- `which()` function usage
  - `post < 4000 | post > 5500`
  - `which(post < 4000 | post > 5500)`
- Logical operators in R:
  - `& # logical "and"`
  - `| # logical "or"`
  - `!  # logical "not"`

Conditional Selection (Matrices and Data Frames)

- In practice, it is very common to select just those rows of a matrix or data frame that meet some criteria.
- For matrix,
  - `my.mat[my.mat[, 1] > 6000, ]`
  - `my.mat[my.mat[, 2] < 5000, ]`
  - `my.mat[my.mat[, 1] > 6000 & my.mat[, 2] < 5000, ]`
- For data frame,
  - `my.df[my.df$pre > 6000, ]`
  - `my.df[my.df$post < 5000, ]`
  - `my.df[my.df$pre > 6000 & my.df$post < 5000, ]`
- Don't forget comma after the condition because we want to extract rows which are observations.

Outline

Missing Values

- In practical data analysis, a data is frequently unavailable.
  (e.g. the patient did not show up, an experiment failed, or the survey question left blank)
- R allows vectors to contain a special `NA` value.
- This `NA` is carried through in computations so that operations on `NA` yield `NA` result.
  - `weight <- c(60, 72, NA, 90, 95, 72)`
  - `weight - 75`

To Find Missing Values

- `is.na()` is used to find which elements of vector are recorded as missing, `NA`.
  - `midterm <- c(98, 72, NA, 89, 69, 92, 78, NA, 94, NA, 83)`
  - `is.na(midterm)`
  - `which(is.na(midterm))`
- `is.na()` is important because "`midterm == NA`" gives `NA` as the result for any value of midterm.

Outline

Outline

Workspace

- All variables created in R are store in a common workspace.
- `ls()` is used to see which variables are defined in workspace.
  - `a <- seq(1,10)`
  - `b <- 25:30`
  - `c <- rep(0, 20)`
  - `d <- c(17, 73, 92, 102, 234)`
  - `ls()`
- `rm()` is used to delete some of the objects.
  - `rm(b)`
  - `rm(a, c)`
  - `rm(list=ls()) # To delete all objects in workspace.`

Scripting

- Beyond a certain level of complexity, you don't want to work with R on a line-by-line basis.
- In this case you can work with R scripts, collection of lines of R code.
- You can also save your R code when you work with R scripts and load again later.

Packages

- Some packages are part of basic installation.
- Others can be downloaded from CRAN, which hosts over 1000 packages for various purpose.
- To install the package,
  - For Mac user go to "Packages & Data" tab and click "Package Installer".
  - For Window user go to ..
  - `install.packages("type package name")`
- Let's try install `ggplot2` package.

Packages

- `library()` is used to load the package that you installed.
  - `library(ggplot2)`
- The loaded packages are not considered part of the user workspace.
- If you terminate R and start again, then you will have to load the package again.

ggplot2: Better Graphics for R

- Since we installed ggplot2 package let us try using it!
  - `p <- ggplot(mtcars, aes(wt, mpg))`
  - `p + geom_point()`
- Compare the ggplot2 plot with original plot() function.
  - `dev.new() # opens new graphic window tab`
  - `plot(mtcars$wt, mtcars$mpg)`

Outline

attach()

- Let us use the famous data set `iris` which is available in R. Type `iris` directly to see what it looks like.

- The notation for accessing variables in the data set gets annoying if we repeatedly have to write longish commands like
  - `iris[iris$Sepal.Length > 6.5 & iris$Sepal.Width < 3.0, ]`

- `attach()` is used to make R to look for objects among the variables in a given data frame.
  - `Sepal.Length # before using attach() it gives you an error`
  - `iris$Sepal.Length # this was correct way`

  - `attach(iris)`
  - `Sepal.Length`
  - `iris[Sepal.Length > 6.5 & Sepal.Width < 3.0, ]`

subset()

- subset() is used to select subsets of data frame.
  - `new.iris1 <- subset(iris, Sepal.Length > 6.5)`
  - `new.iris2 <- subset(iris, Sepal.Length > 6.5 & Sepal.Width < 3.0)`
  - `new.iris3 <- subset(iris, Species == "setosa")`
  - `new.iris4 <- subset(iris, select = c(Sepal.Length))`
  - `new.iris5 <- subset(iris, select = c(Sepal.Length, Sepal.Width))`

transform()

- `transform()` is used to create new data frames with transformed variables.
  - `new.iris6 <- transform(iris, log.Sepal.Length = log(Sepal.Length))`
  - `new.iris7 <- transform(iris, diff.Sepal = Sepal.Length - Sepal.Width)`

Outline

Plot Layout

- `plot()` is generic function for plotting R objects. By default, draw a scatter plot.
  - `plot(Sepal.Length, Sepal.Width)`
- `main = "put overall title"` is used to put overall title of the plot.
- `xlab = "put x-axis title"` is used to change the x-axis title.
- `ylab = "put y-axis title"` is used to change the y-axis title.
  - `plot(Sepal.Length, Sepal.Width, main = "Sepal Iris", xlab = "Length", ylab = "Width")`
- `abline()` is used to add straight line to the plot
  - `abline(a = 0.3, b = 0.5) # y = a + bx`
  - `abline(h = 3.0) # horizontal line`
  - `abline(v = 6.0) # vertical line`

Using par()

- `par()` is used to set many graphical parameters. Here, we focus on one main usage.
- `par(mfrow = c())` is used to put multiple figures simultaneously.
  - `par(mfrow = c(1, 2))`
  - `plot(Sepal.Length, Sepal.Width, main = "Sepal Iris", xlab = "Length", ylab = "Width")`
  - `plot(Petal.Length, Petal.Width, main = "Petal Iris", xlab = "Length", ylab = "Width")`

Outline

Data Input from a Text File (.txt)

- `read.table()` is used to import a text file to R.
- Please follow the steps below:
  - **Step 1**: Prepare a new folder, say `Rdemo`. Download the data set (e.g. Auto.txt) to the folder.
  - **Step 2**: Set the working directory of R to the folder where the data set is located.
  - **Step 3**: Read in the data set by calling:
    ```
    autotxt <- read.table("Auto.txt")
    ```
- Assign the result of `read.table()` is always desirable.

Data Input from an Excel File (.csv)

- `read.csv()` is used to import an Excel file to R.
- Please follow the steps below:
  - **Step 1**: Prepare a new folder. Download the data set (e.g. Auto.csv) to the folder.
  - **Step 2**: Set the working directory of R to the folder where the data set is located.
  - **Step 3**: Read in the data set by calling:
    `autocsv <- read.csv("Auto.csv")`
- Assign the result of `read.csv()` is always desirable.

Data Output to a Text File / Excel File

- `write.table()` is used to export a data frame object to a text file.
- `write.csv()` is used to export a data frame object to a csv file.
  - `write.table(autotxt, file = "textauto.txt")`
  - `write.csv(autocsv, file = "csvauto.csv")`
- Check your directory folder to make sure you have a new file.