

# Introduction to R

Lin Cong

University of California, Riverside

*gradquant@ucr.edu*

October 11, 2019

- 1 Introduction
- 2 Data manipulation
- 3 Real Data Example

# Introduction

# What & Where

- R is a free software environment for statistical computing and graphics.
- R is a platform for the object-oriented statistical programming language S.
- To obtain R for Windows (or Mac) go to The Comprehensive R Archive Network (CRAN) at <http://www.r-project.org>. Just download the executable file, and it will install itself.

# Advantages and Disadvantages

## Advantages:

- Open Source (Free and lots of flexibility).
- Cross platform.
- Graphical Capabilities.

## Disadvantages:

- Not enough documentation sometimes.
- R commands give little thought to memory management, so R can consume all available memory.

## Advantages:

- easy to write scripts.
- convenient to view and interact with the objects stored in your environment.
- easy to set your working directory and access files on your computer.

- R script
- R console(a.k.a. terminal)
- Workspace
- Help

# Preliminaries on entering command

- R is case sensitive.
- Comments can be put in text, starting with a hashmark #.
- Command lines can be separated by a semi-colon (;), or simply by starting a new line.
- An object name must start with an alphabetical character, but may contain numeric characters thereafter. A period may also form part of the name of an object. For example, x.1 is a valid name.
- If a command is not complete at the end of a line, R will give a different prompt, by default (+).



# Help!

- `help(function.name) / ?function.name`
- `args(function.name)`
- `example(function.name)`
- `??keyword / help.search(keyword)`

# Data manipulation

# Working directory

A working directory is the default location where R looks for files.

- To see the current working directory:

```
> getwd()
```

```
"C:/Documents and Settings/user/My Documents"
```

- To change working directory:

```
> setwd("C:\\Documents\\.....") — For Windows
```

```
> setwd("/Documents/.....") — For Linux/Mac
```

- Note the direction of the slashes for different operating systems.

All data is represented in binary format, by bits (TRUE/FALSE, YES/NO, 1/0)

- **Booleans:** Direct binary values: TRUE or FALSE in R.
- **Integers:** whole numbers (positive, negative or zero), represented by a fixed-length block of bits.
- **Characters:** fixed-length blocks of bits, with special coding; strings = sequences of characters.
- **Floating point numbers:** a fraction (with a finite number of bits) times an exponent, like  $1.87 * 10^6$ , but in binary form.
- **Missing or ill-defined values:** NA, NaN, etc.

## Operators:

- >  $8+3$
- >  $8\wedge 3$
- >  $8\%3$  # the modulo operator
- >  $8\%/\%3$  # integer division

## Boolean operators:

- >  $8>3$
- >  $8!=3$
- >  $(8<3) \& (6*7==42)$
- >  $(8<3) | (6*7==42)$

- `typeof()` function returns the type.  
> `typeof(7)`
- `is.foo()` functions return Booleans for whether the argument is of type `foo`.  
> `is.numeric(7)`  
> `is.character("seven")`
- `as.foo()` (tries to) "cast" its argument to type `foo` to translate it sensibly into a `foo`-type value.  
> `as.character(5/6)`  
> `as.numeric(as.character(5/6))`  
> `5/6 == as.numeric(as.character(5/6))`

- ```
> options(digits=22)
```

```
> as.numeric(as.character(5/6))
```

```
> 5/6
```

## Notes on floating-point numbers:

- The R floating-point data type is a *double*, Finite precision  $\Rightarrow$  arithmetic on doubles  $\neq$  arithmetic on  $\mathbb{R}$ .
  - > `0.45 == 3*0.15`
  - > `0.45 - 3*0.15`
- Rounding errors tend to accumulate in long calculations. When results should be  $\approx 0$ , errors can flip signs. Usually better to use `all.equal()` than exact comparison.
  - > `(0.5 - 0.3) == (0.3 - 0.1)`
  - > `all.equal(0.5-0.3, 0.3-0.1)`



## Notes on Integers:

- Typing a whole number in the terminal doesn't make an integer; it makes a double, whose fractional part is 0.

```
> is.integer(7)
```

```
> as.integer(7)
```

```
> round(7) == 7
```

## Inputting Data as objects:

- Scalar:

```
> a <- 2
```

- Vector:

```
> b <- c(2,4,5)
```

- Matrix:

```
> c <- matrix(c(1,2,3,4,5,6),3,2)
```

To display Objects, type the object name::

- `> a`

- `> c`

- To check which objects are stored in memory:

```
> ls()
```

- Another way to do the same thing:

```
> objects()
```

- We can also remove objects from memory:

```
> rm(a)
```

Note: if you want to clear the whole workspace:

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

# Data Structure: Vectors

A vector is a sequence of values, all of the same type:

```
> x <- c(2, 3, 5, 8)
```

```
> x
```

```
> is.vector(x)
```

c() function returns a vector containing all its arguments in order

- To extract values from vector:

```
> x[1]
```

```
> x[2:3]
```

```
> x[c(2,4)]
```

```
> x[c(-1,-3)]
```

- To delete values from vector:

```
> x[-4]
```

- To make an empty vector of fixed length for filling things up:

```
> empty <- vector(length=5)
```

```
> empty[5] <- 8
```

```
> empty
```

# Data Structure: Vectors

- Vector arithmetic:

Operators apply to vectors "pairwise" or "elementwise"

```
> y <- c(-2, -3, -5, -8)
```

```
> x+y
```

- Recycling:

Recycling repeat elements in shorter vector when combined with longer

```
> x + c(-2,-3)
```

Single numbers are vectors of length 1 for purposes of recycling:

```
> 2*x
```

# Data Structure: Vectors

- Pairwise comparison:

```
> x > 4
```

- Elementwise comparison:

```
> (x > 4) & (x < 20)
```

- To compare whole vectors, use `identical()` or `all.equal()`:

```
> x == y
```

```
> identical(x, -y)
```

```
> identical(c(0.5-0.3,0.3-0.1),c(0.3-0.1,0.5-0.3))
```

```
> all.equal(c(0.5-0.3,0.3-0.1),c(0.3-0.1,0.5-0.3))
```



- Boolean vector:

```
> x>4
```

```
> x[x>4]
```

- which() turns a Boolean vector in vector of TRUE indices:

```
> index <- which(x > 4)
```

```
> index
```

```
> y[index]
```

## Named component:

- Give names to elements or components of vectors:
  - > `names(x) <- c("v1", "v2", "sum12", "sum23")`
  - > `names(x)`
- `names(x)` is just another vector (of characters):
  - > `sort(names(x))`
  - > `which(names(x)=="sum12")`

Functions on vectors:

- `mean()`, `median()`, `sd()`, `var()`, `max()`, `min()`, `length()`, `sum()`: return single numbers.
- `sort()` returns a new vector.
- `hist()` takes a vector of numbers and produces a histogram, a highly structured object, with the side-effect of making a plot.
- `ecdf()` produces a cumulative-density-function object.
- `summary()` gives a six-number summary of numerical vectors.

# Data Structure: Arrays

An array is a data structure that contains a group of elements.

```
> arr <- array(x,dim=c(2,2))
```

```
> arr
```

```
> dim(arr)
```

```
> typeof(arr)
```

```
> attributes(arr)
```

# Data Structure: Arrays

- Can access a 2-D array either by pairs of indices or by the underlying vector:
  - > `arr[1,2]`
  - > `arr[3]`
- Omitting an index means "all of it":
  - > `arr[c(1:2),2]`
  - > `arr[,2]`

Functions on arrays:

- Using a vector-style function:  
> `which(arr > 4)`
- Functions that preserve array structure:  
> `rowSums(arr)`

# Data Structure: Matrices

- A matrix is a specialization of a 2D array in R.  
> `mtrx <- matrix(c(2,3,5,8),nrow = 2, byrow = T)`
- Matrix multiplication  
> `sevens <- matrix(rep(7,4),ncol=2)`  
> `mtrx %*% sevens`  
> `mtrx * sevens`

Note: take care of the dimensions for multiplication.

```
> output <- c(10,20)
> mtrx %*% output
```

Note: When multiplying matrices and vectors, R silently casts the vector as either a row or a column matrix.

Matrix operator:

- Transpose:  
> `t(mtrx)`
- Determinant:  
> `det(mtrx)`
- Inverting a matrix:  
> `solve(mtrx)`



## Diagonal matrices:

- extract the diagonal entries of a matrix:  
`> diag(mtrx)`
- change the diagonal:  
`> diag(mtrx) <- c(35,4)`
- Creating a diagonal or identity matrix:  
`> diag(c(3,4))`

- Names in matrices:

```
> rownames(mtrx) <- c("r1","r2")
```

```
> colnames(mtrx) <- c("c1","c2")
```

```
> mtrx
```

- Doing the same thing to each row or column:

```
> rowMeans(mtrx)
```

```
> summary(mtrx)
```

```
> apply(mtrx,1,mean)
```

# Data Structure: list

A list is a sequence of values, not necessarily all of the same type.

```
> dist.list <- list("exponential",7,FALSE)
```

```
> dist.list
```

- Accessing pieces of lists:

In R, `[[index]]` extracts only the element of the list without names and structures, while `[index]` extracts the whole element.

```
> is.character(dist.list[[1]])
```

```
> dist.list[[2]]^2
```

- Expanding and contracting lists:

```
> dist.list <- c(dist.list,7)
```

```
> length(dist.list)
```

```
> length(dist.list) <- 3
```

```
> dist.list
```

- Name some or all of the elements of a list:
  - > names(dist.list) <- c("family", "mean", "is.symmetric")
  - > dist.list
  - > dist.list[["family"]]
  - > dist.list\$family
- Creating a list with names:
  - > another <-  
list(family="gaussian", mean=7, sd=1, is.symmetric=TRUE)
  - > dist.list\$was.estimated <- FALSE
  - > dist.list[["last.updated"]] <- "2011-08-30"
  - > dist.list\$was.estimated <- NULL

# Data Structure: Data frames

- Dataframe is the classic data table, with  $n$  rows for cases,  $p$  columns for variables.
- Difference with matrix: columns can have different types.
- Many matrix functions also work for dataframes (`rowSums()`, `summary()`, `apply()`).
- No matrix multiplying dataframes, even if all columns are numeric.

# Data Structure: Dataframes

Generate a data frame.

```
> mat <- matrix(c(35,8,10,4),nrow=2)
> colnames(mat) <- c("v1","v2")
> mat
> df <- data.frame(mat,logicals=c(TRUE,FALSE))
> df
```

# Data Structure: Dataframes

- Extracting data:

```
> df$v1
```

```
> df[, "v1"]
```

```
> df[, 1]
```

- Adding rows and columns:

```
> rbind(df, list(v1=-3, v2=-5, logicals=TRUE))
```

```
> rbind(df, c(3, 4, 6))
```

# Real Data Example



# Statistical Analysis Pipeline

## Step 1: Data collection

Data can be from the internet or external/internal databases, and it must be extracted into a usable format (.csv, json, xlsx, etc.)

## Step 2: Data cleaning

Understand every feature, identify errors, and missing values. Clean data by throwing away, replacing, filling missing values/errors.

## Step 3: Model building

After cleaning the data and finding what features are most important, use model as a predictive tool for analysis.

## Step 3: Statistical inference

Understand and learn how to explain your findings and further diagnosis, including model comparisons, and hypothesis testing, etc.

# Modifying data

## Reading in data:

- R built-in data:

Many packages come with saved data objects; there's the convenience function `data()` to load them.

```
> data(cats,package="MASS")
```

```
> summary(cats)
```

- Non-R Data Tables:

`read.table()`: Presumes space-separated fields, one line per row.

`read.csv()`: reading comma-separated value (CSV) files.

- Main argument is the file name or URL.

- Returns a dataframe.

The foreign package on CRAN has tools for reading data files from lots of non-R statistical software.

# Modifying data

We are going to use the Birth Weight data built in the MASS package which contains Risk Factors Associated with Low Infant Birth Weight. The data were collected at Baystate Medical Center, Springfield, Mass during 1986.

```
> library(MASS)
> data(birthwt)
> summary(birthwt)
```

# Modifying data

## Variables Introduction:

*low*: indicator of birth weight less than 2.5 kg.

*age*: mother's age in years.

*lwt*: mother's weight in pounds at last menstrual period.

*race*: mother's race (1 = white, 2 = black, 3 = other).

*smoke*: smoking status during pregnancy.

*ptl*: number of previous premature labours.

*ht*: history of hypertension.

*ui*: presence of uterine irritability.

*ftv*: number of physician visits during the first trimester.

*bwt*: birth weight in grams.

# Modifying data

- To know more about this data, using the help function:  
> `help(birthwt)`
- To have a look what the data is like:  
> `birthwt`  
> `head(birthwt)`  
> `tail(birthwt)`

# Modifying data

- To check what variables are in it:  
> `colnames(birthwt)`  
Note: We can also change the column names using the same function.
- Each column of this data frame is an object. To display a specific object:  
> `birthwt$bwt`
- To bypass having to specify the "dataframe\$object", we can attach the dataframe:  
> `attach(birthwt)`  
Now, we can display the column simply by:  
> `bwt`

# Modifying data

Let's first take a look of the data.

- Scatter plot:

```
> plot(birthwt$age, birthwt$bwt, xlab="mother's age", ylab="Birth weight", main="Scatter plot", type="p", col="blue")
```

| type | description                           |
|------|---------------------------------------|
| p    | points                                |
| l    | lines                                 |
| o    | overplotted points and lines          |
| b,c  | points (empty if "c") joined by lines |
| s,S  | stair steps                           |
| h    | histogram-like vertical lines         |
| n    | does not produce any points or lines  |

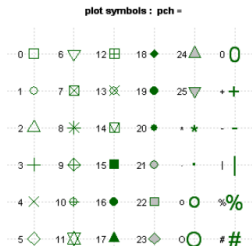
# Modifying data

## Graphical parameters:

- Text and Symbol Size:

| option                | description                                                                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>cex</code>      | number indicating the amount by which plotting text and symbols should be scaled relative to the default. 1=default, 1.5 is 50% larger, 0.5 is 50% smaller, etc. |
| <code>cex.axis</code> | magnification of axis annotation relative to <code>cex</code>                                                                                                    |
| <code>cex.lab</code>  | magnification of x and y labels relative to <code>cex</code>                                                                                                     |
| <code>cex.main</code> | magnification of titles relative to <code>cex</code>                                                                                                             |
| <code>cex.sub</code>  | magnification of subtitles relative to <code>cex</code>                                                                                                          |

- Plotting Symbols:





# Modifying data

## Graphical parameters:

- Lines:

| option           | description                                                          |
|------------------|----------------------------------------------------------------------|
| <code>lty</code> | line type. see the chart below.                                      |
| <code>lwd</code> | line width relative to the default (default= 1). 2 is twice as wide. |

- Colors:

| option                | description                                                                                      |
|-----------------------|--------------------------------------------------------------------------------------------------|
| <code>col</code>      | Default plotting color. Some functions (e.g. lines) accept a vector of values that are recycled. |
| <code>col.axis</code> | color for axis annotation                                                                        |
| <code>col.lab</code>  | color for x and y labels                                                                         |
| <code>col.main</code> | color for titles                                                                                 |
| <code>col.sub</code>  | color for subtitles                                                                              |
| <code>fg</code>       | plot foreground color (axes, boxes - also sets <code>col=</code> to same)                        |
| <code>bg</code>       | plot background color                                                                            |

## Graphical parameters:

- Fonts:

| option           | description                                                                                                       |
|------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>font</b>      | Integer specifying font to use for text.<br>1=plain, 2=bold, 3=italic, 4=bold italic, 5=symbol                    |
| <b>font.axis</b> | font for axis annotation                                                                                          |
| <b>font.lab</b>  | font for x and y labels                                                                                           |
| <b>font.main</b> | font for titles                                                                                                   |
| <b>font.sub</b>  | font for subtitles                                                                                                |
| <b>ps</b>        | font point size (roughly 1/72 inch)<br>text size=ps*cex                                                           |
| <b>family</b>    | font family for drawing text. Standard values are "serif", "sans", "mono", "symbol". Mapping is device dependent. |

- Add more features to the plot:
  - > `lines(birthwt$age, birthwt$bwt, col="red")`
  - > `legend("topleft", c("points"), fill=c("blue"))`

# Modifying data

- Missing data:

```
> na.omit(file.name)
```

- To make the factors more descriptive:

Creating a new column:

```
> birthwt$race.name <- factor(c("white", "black",  
"other"))[birthwt$race]
```

Keeping the original column:

```
> birthwt$smoke <- factor(c("No", "Yes"))[birthwt$smoke + 1])
```

## Coding for the data—Dummy Coding:

- For two categories:

Low birthweight is a term used to describe babies who are born weighing less than 2,500 grams (5 pounds, 8 ounces). Over 8 percent of all newborn babies in the United States have low birthweight. So we want to compare the lowest category of birthweight to the rest.

```
> birthwt$btw.dummy <- as.numeric(birthwt$bwt < 2500)
```

```
> birthwt$btw.dummy == birthwt$low
```

- For more categories:  
According to the Human Birth Weight Classification, there are 3 categories for the birth weight.

---

| <b>Low Birth Weight</b> | <b>Normal Birth Weight</b> | <b>High Birth Weight</b> |
|-------------------------|----------------------------|--------------------------|
| < 2500                  | 2500 - 3999                | > 3999                   |

---

```
> birthwt$btw.d3[birthwt$bwt<2500] <- 1
```

```
> birthwt$btw.d3[birthwt$bwt>3999] <- 3
```

So how do we create the second category?

```
> birthwt$btw.d3[birthwt$bwt>=2500 & birthwt$bwt<=3999] <- 2
```

# Modifying data

Logical Statements:

Logical statements in R are evaluated as to whether they are TRUE or FALSE.

Table 1: Logical Operators in R

| Operator | Means                    |
|----------|--------------------------|
| <        | Less Than                |
| <=       | Less Than or Equal To    |
| >        | Greater Than             |
| >=       | Greater Than or Equal To |
| ==       | Equal To                 |
| !=       | Not Equal To             |
| &        | And                      |
|          | Or                       |

We can not only identify data based on the logical statements, but also subtract data by them.

- Find the records with low birthweight with the mother smoking:  
> `birthwt$smoke.low <- birthwt$low==1 & birthwt$smoke==1`  
To subtract these records:  
> `birth.sub <- birthwt[birthwt$low==1 & birthwt$smoke==1, ]`  
Note: without comma, it cannot work.
- Check the age for the records with low birthweight with the mother smoking:  
> `birthwt$age[birthwt$low==1 & birthwt$smoke==1]`



Is there any relationship between birth weight and mothers' smoking behavior?

- Estimating an OLS model in R is done with the `lm` command:  
> `lm.1 <- lm (bwt ~ smoke, data=birthwt)`  
> `summary(lm.1)`
- R is an object-oriented environment.:  
Whenever we run a model or create a data frame, we create an object, which consists of several components. So, we can call on these components of the `lm.1` object to obtain several other regression outputs including the coefficients, residuals, `cov.unscaled` (the variance-covariance matrix) and `fitted.values`.  
> `lm.1$coefficients`

- Two-sample t-test:

```
> t.test (birthwt$bwt[birthwt$smoke == "Yes"],  
birthwt$bwt[birthwt$smoke == "No"])
```

# More complex Models

- Including interaction term:
  - > `lm.2 <- lm (bwt ~ age + smoke*race, data=birthwt)`
  - > `summary(lm.2)`
  - > `lm.3 <- lm (bwt ~ age + smoke:race, data=birthwt)`
  - > `summary(lm.3)`
- Including all predictors:
  - > `lm.4 <- lm (bwt ~ ., data=birthwt)`
  - > `summary(lm.4)`

Notice: the birthweight is a function of the indicator variable *low*

- Model without indicator *low*:
  - > `lm.5 <- lm (bwt ~ .- low, data=birthwt)`
  - > `summary(lm.5)`

# Generalized Linear Models

To use the `glm` command you must specify your formula, the family, and the link function.

- To predict whether an infant has low birth weight, let's first fit a logistic model with all the predictors in the data.

```
> glm.1 <- glm (low ~ .- bwt, data=birthwt,  
family=binomial(link=logit))  
> summary(glm.1)
```

# Generalized Linear Models

The `glm` command allows you to run a variety of models.

| Family           | Default Link Function                      |
|------------------|--------------------------------------------|
| binomial         | (link = "logit")                           |
| gaussian         | (link = "identity")                        |
| Gamma            | (link = "inverse")                         |
| inverse.gaussian | (link = "1/mu^2")                          |
| poisson          | (link = "log")                             |
| quasi            | (link = "identity", variance = "constant") |
| quasibinomial    | (link = "logit")                           |
| quasipoisson     | (link = "log")                             |

# Question?

# Thank You

# Welcome to GradQuant