# Advanced R Programming

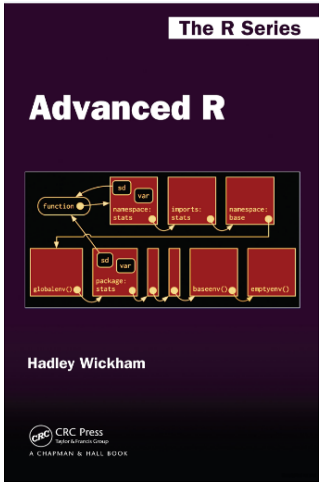
Farid Cheraghi
PhD student in GIS
University of Tehran

# How to master R

- Learn to read the documentation
  - Prerequisites (covered today)
    - Data structure
    - Vectorization
    - Object orientation (OO) in R
- First master the R core then go to the contributed packages

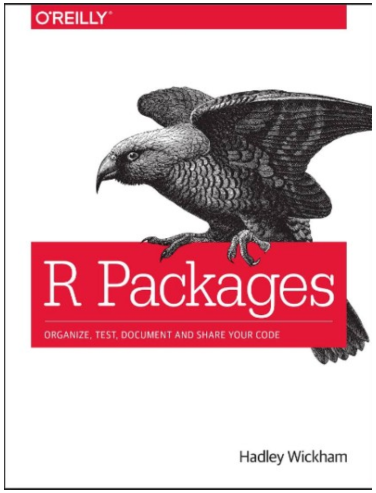
# How to master R

- There are more advanced topics that we don’t cover in this session
  - Environment
  - Exceptions and debugging
  - Functional
  - expressions
  - Non-standard evaluation (Subset(),transform())
  - Memory (Profiling)
  - Rcpp



http://adv-r.had.co.nz/

# How to master R



- R packaging and modular design
- Benefits
  - Encapsulate individual project into one package
    - More manageable and tidy
  - When you learn how to create a package it is easier to understand the mechanism underlying R built-in and contributed packages
  - Reproducibility (data + code)
  - Sharable (github etc)

# Learn to read the documentation

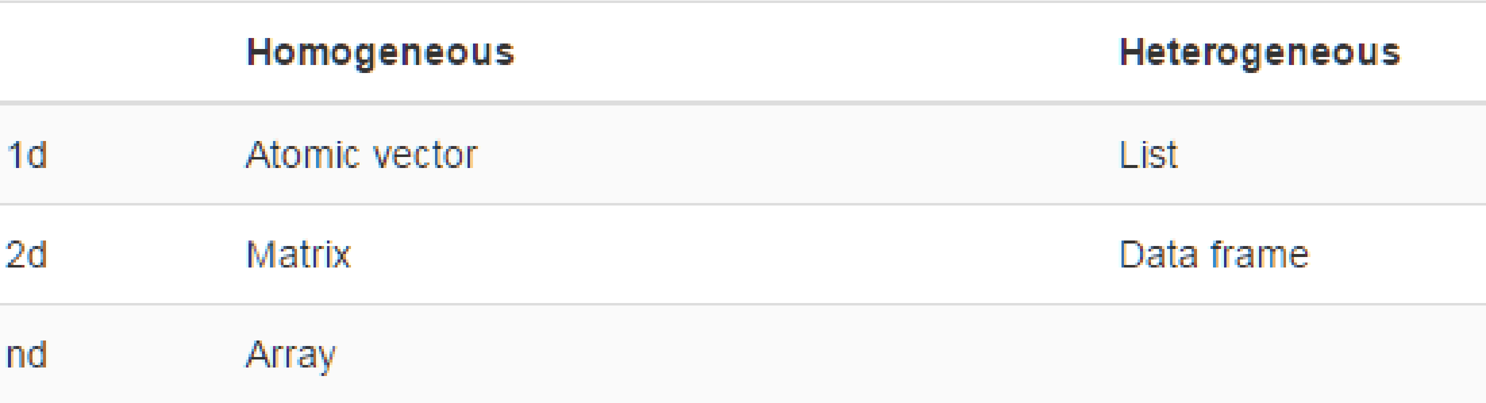
- Benefits
  - No book/person is required to teach you how to use R
  - Every package comes with it’s manual
  - all methods of a generic function in one place (aggregate)
  - Functions are grouped and linked in the doc (i.e. lapply). Easier to remember and organize in brain
  - If you only know the function name, you can use it with doc page.
- Reading the doc is difficult specially when you don’t know the terms

# Data structure. Questions?

- What are the three properties of a vector, other than its contents?
- What are the four common types of atomic vectors? What are the two rare types?
- What are attributes? How do you get them and set them?
- How is a list different from an atomic vector? How is a matrix different from a data frame?
- Can you have a list that is a matrix? Can a data frame have a column that is a matrix?

# Data structure

- Type hierarchy:
  - NULL < raw < logical < integer < double < complex < character < list < expression
  - Impilict coercion
    - What is the type of c(1,1+2i,’c’)?



| | Homogeneous | Heterogeneous |
|---|---|---|
| 1d | Atomic vector | List |
| 2d | Matrix | Data frame |
| nd | Array | |

# Data structures

- Vectors
  - Atomic vectors
  - Lists
- Attributes
  - Factors
- Matrices and array
- Data frames
  - List columns

# Data structures tricks

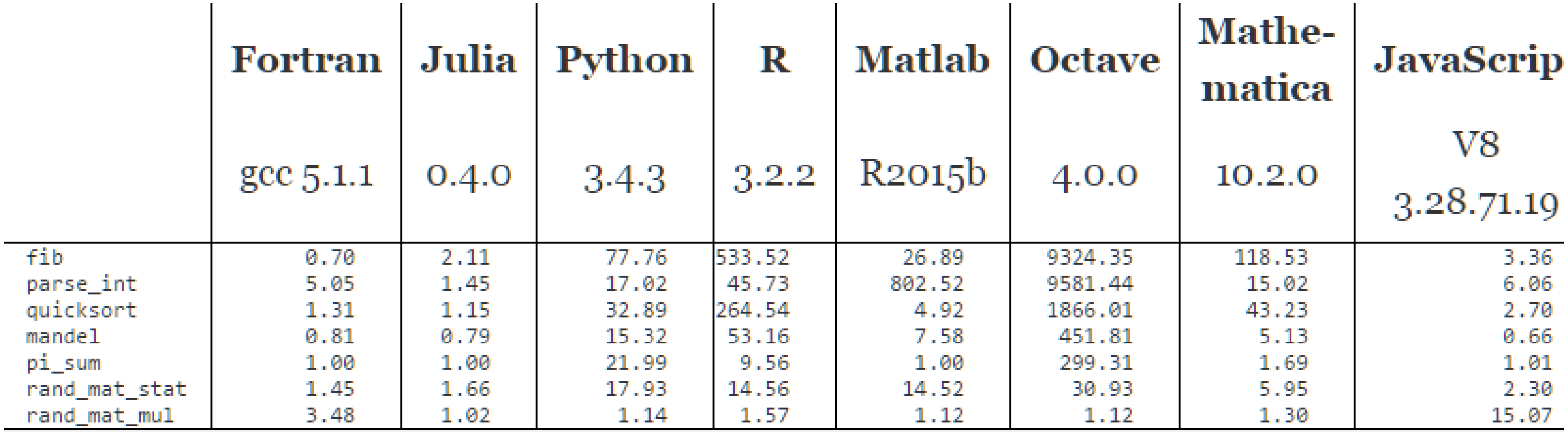
- Useful functions
  - mode() , Typeof()
  - class()
  - dput()
  - Str()
- When you face a new function, ASK:
  - What is the type (data structure) of input and the output?
  - It makes it easier to chain the functions

# R language

- Pros
  - R is easy!!!
    - Easy compared to other programming languages such as C#, C++, Java
  - Succinct and abstract
  - The best tool for scientific purposes i.e. for fast prototyping of a given algorithm or idea
  - Is being used and developed in big IT companies: Google, Microsoft, Facebook etc
- Cons
  - Not fast
    - Interface with C++, Fortran etc
    - Parallel processing (there are packages)
  - Memory hungry
    - Not viable with tables larger than 10-100 K
    - Solution (data.table, DBMS)
  - Slow with **loops**
    - Solution: vectorization

# Unfair benchmarks

- Using loops in R



| | Fortran | Julia | Python | R | Matlab | Octave | Mathematica | JavaScrip |
|---|---|---|---|---|---|---|---|---|
| | gcc 5.1.1 | 0.4.0 | 3.4.3 | 3.2.2 | R2015b | 4.0.0 | 10.2.0 | V8<br>3.28.71.19 |
| fib | 0.70 | 2.11 | 77.76 | 533.52 | 26.89 | 9324.35 | 118.53 | 3.36 |
| parse_int | 5.05 | 1.45 | 17.02 | 45.73 | 802.52 | 9581.44 | 15.02 | 6.06 |
| quicksort | 1.31 | 1.15 | 32.89 | 264.54 | 4.92 | 1866.01 | 43.23 | 2.70 |
| mandel | 0.81 | 0.79 | 15.32 | 53.16 | 7.58 | 451.81 | 5.13 | 0.66 |
| pi_sum | 1.00 | 1.00 | 21.99 | 9.56 | 1.00 | 299.31 | 1.69 | 1.01 |
| rand_mat_stat | 1.45 | 1.66 | 17.93 | 14.56 | 14.52 | 30.93 | 5.95 | 2.30 |
| rand_mat_mul | 3.48 | 1.02 | 1.14 | 1.57 | 1.12 | 1.12 | 1.30 | 15.07 |

# Vectorization

- Rule of thumb: avoid loops
  - For
  - while
- Vectorized functions call internal c code and implicitly use multiple CPU cores.
- lapply, apply, sapply, mapply, do.call, vapply, split, tapply, aggregate, eapply, rapply, replicate, simplify2array
  - Combining with plot functions is very useful
- Cumsum, cumprod, cummax, cummin

# Object orientation (OO)

Central to any object-oriented system are the concepts of class and method. A **class** defines the behaviour of **objects** by describing their attributes and their relationship to other classes. The class is also used when selecting **methods**, functions that behave differently depending on the class of their input. Classes are usually organised in a hierarchy: if a method does not exist for a child, then the parent’s method is used instead; the child **inherits** behaviour from the parent.

# OO

- Generic functions and methods
  - Aggregate
  - Boxplot
- R should know which method to use based on the class of the object that is sent to a generic function.

# R OO systems

- S3
  - Simple
  - Succinct
  - Widespread
  - The best to go
- S4
  - Multiple dispatch (dispatch method based on the class of more than one argument)
  - Formal class definition
    - Slots and inheritance
  - Verbose and clunky
  - Matrix and stats4 in Core. Sp package for spatial data
- RC
  - very new and immature.
  - I haven’t seen it being used anywhere yet

# Lab

Given the mtcars data.frame, how do you do boxplot of all columns for each cylinder class?
You should use this combo: lapply, split, mapply