CS636 Data Analytics with R Programing

David Li

Course Logistics

- Basic information
- Requirements
- Goal

Requirements

- Homework & computing lab exercise (10%)
- Quiz (20%)
- Term Project (20%)
- Midterm (20%)
- Final (30%)

Homework (10 %)

- Homework assignments
  - Only use R in homework
  - Try to do it independently, discussions allowed, but copying is forbidden.
- Homework Grading Policy
  - Your homework: may have several homework assignments, but pick only one (the worst one) to grade. Namely, if you miss one assignment, you get 0.
- Late homework policy
  - 25% penalization per late day;
  - Not accepted more than 3 days late

Lab exercise

- Have a lab session every week
- Lab exercises
  - Focus on R computing exercises
  - We will solve some simple problems
  - Post your answers by replying on canvas
  - Some answers may be selected for discussion by the end of lab session.
  - Some problems may become part of homework

Two Term Projects (20%)

- You can choose R or Python for your projects
- Use Jupyter
- Submit code and report to summarize what you have done and results you obtained.
- Prepare for presentation and demo.
- 1~4 students a group.
- More details to be announced soon
- Cheating/Copying is strictly prohibited. I will report to Dean and you will get F in this course.
- If you think your group members don't make contribution, talk to me.

Quiz (20%)

- Focus on course materials.
- Every other week
- Only R is allowed

Two Exams (50%)

- One midterm and one Final (20%+30%)
  - In-class
  - Open book
  - Final is cumulative
  - Only R

Some tips

- Set up your dev environment for exams. It may require to write code.
- Prior to quiz/exam, restudy the slides and Jupyter sample code
- If I discover cheating, I will report the incident to the Dean of Student's office Re: Academic Integrity. (TAs report the incident to the course instructor)

Goal

- We use R to teach this class but the content is for generic data science
- Focus on the skills that can be transferred to Python
- Familiarize you with the commonly used analytical techniques in Data Science
- Develop the way of data science thinking
  - Learn how to preprocess, explore and interpret real data
  - Learn how to model real problems using computational techniques

Prerequisites

- Basic programming skills
- Linear algebra
- Probability
- Statistics

Tentative course topics

(Subject to changes according to progress)

- R libraries for data science. The most common knowledge that you can easily apply to Python.
- Big Data and Graph Analytics
- Visualization with probability and statistics basics
- Regular Expression and NLP for text processing
- Machine learning algorithms (a lot of math)
- Model/Feature Selection(more math)
- May cover advanced big data and deep learning

Intro to R

David Li

What is R?

- Statistical computer language similar to S-plus
- Interpreted language (like Matlab)
- Has many built-in (statistical) functions
- Easy to build your own functions
- Good graphic displays
- Extensive help files

Strengths

- Many built-in functions
- Can get other functions from the internet by downloading libraries
- Relatively easy data manipulations

Weaknesses

- Not as commonly used by non-statisticians
- Not a compiled language, language interpreter can be very slow, but allows to call own C/C++ code

R packages

- Packaging: a crucial infrastructure to efficiently produce, load and keep consistent software libraries from (many) different sources / authors
- Statistics
  - most packages deal with statistics and data analysis
  - State of the art: many statistical researchers provide their methods as R packages

A sample job opening

- Experience using Statistical and Machine Learning algorithms on real data, in commercial environments. Experience in the Telecom Domain is a big plus.
- Excellent and wide ranging experience in supervised and unsupervised learning. Reinforcement learning a plus.
- Experienced in the use and design of logistic regression, support vector machines, ensemble trees, and neural networks. Optimization problems a plus
- Familiarity and experience with the standard machine learning packages, such as numpy, scipy scikit-learn, TensorFlow, keras and Theano
- Experience in data munging, data cleansing etc.
- Experience in feature selection, feature engineering and development of recommender systems.
- Very Good Knowledge on one and more Statistical Tool like R/Python
- Excellent communication and interpersonal skills, with proven ability to take initiative and build strong, productive relationships.
- Good to have experience in Network Analytics. Building a strong intuitive understanding of the problem domain (Next Generation Access Networks).
- Influence and transform the end-to-end delivery process to maximize the value Customers gain from Analytics
- Should be able to handle multiple projects and liaison with customer different teams to bring overall value add.
- Should have strong people skills and good team management experience.
- Identify testable hypotheses to explain interesting phenomena in this domain
- Constructing an automated system test framework
- Develop and communicate goals, strategies, tactics, project plans, timelines, and key performance metrics to reach goals
- Experience with public cloud (AWS) desirable
- Great communication skills
- Proficiency in using query languages such as SQL
- Good scripting and programming skills, such as R, Python, or Spark

When to use R?

- When
  - Requires standalone computing or analysis on individual servers.
  - Great for exploratory work: it's handy for almost any type of data analysis because of the huge number of packages and necessary tools to get up and running quickly
  - R can even be part of a big data solution.

How to use/learn R?

- How
  - (optional) Install and Use Rstudio IDE
  - (optional) Install Jupyter with R kernel
  - Getting started with R (Basic grammars)
  - Get to use/learn those popular packages
    - dplyr, plyr and reshape2 for data manipulation
    - stringr for string operation
    - ggplot2 for data visualization
    - …
  - Do (a lot of) practices including real projects

Install RStudio

- An integrated development environment (IDE) available for R
  - a nice editor with syntax highlighting
  - there is an R object viewer
  - there are a number of other nice features that are integrated
- How to install
  - https://www.youtube.com/watch?v=9-RrkJQQYqY

Install Jupyter with R kernel

1. Install R and Rstudio
2. Download and install the latest Anaconda at https://www.anaconda.com/download/
3. In windows, add your R bin path and Anaconda3 Scripts path to your environmental variable "Path"
   - In my computer the R bin path is C:\Program Files\R\R-3.5.1\bin
   - Anaconda3 Scripts path is C:\ProgramData\Anaconda3\Scripts, the paths in your computer may vary.
   - How to set the path and environment variables in Windows https://www.computerhope.com/issues/ch000549.htm
   - Install R kernel to Jupyter (PLEASE DO THIS STEP IN R CONSOLE, not in Rstudio or RGui)
     https://irkernel.github.io/installation/
     https://stackoverflow.com/questions/44056164/jupyter-client-has-to-be-installed-but-jupyter-kernelspec-version-exited-wit
   - Then you can start "Jupyter Notebook" from the start menu.

Starting and stopping R

- Starting
  - Windows: **`Double click on the R icon`**
  - Unix/Linux: type **`R (or the appropriate path on your machine)`**

- Stopping
  - **`Type q()`**
  - q( )is a function execution
  - Everything in R is a function
  - q merely returns the content of the function

Writing R code

- Can input lines one at a time into R
- Can write many lines of code in any of your favorite text editors (including Rstudio) and run all at once
  - Simply paste the commands into R
  - Use function source("path/yourscript"), to run in batch mode the codes saved in file "yourscript" (use options(echo=T) to have the commands echoed)

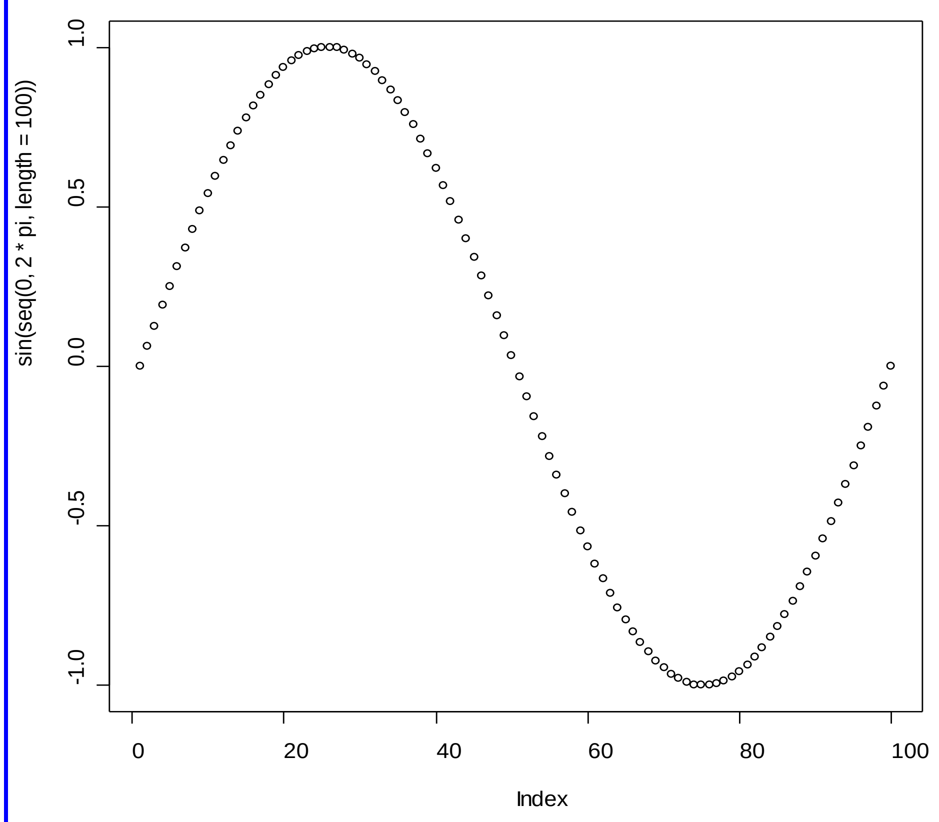
R as a Calculator

```
> log2(32)
[1] 5

> sqrt(2)
[1] 1.414214

> seq(0, 5, length=6)
[1] 0 1 2 3 4 5

> plot(sin(seq(0,
 2*pi, length=100)))
```



Recalling Previous Commands

- In WINDOWS/UNIX one may use the **`arrow up key`** or the **`history`** command under the menus

- Given the history window then one can copy certain commands or else past them into the console window

Language layout

- Three types of statement
  - expression: it is evaluated, printed, and the value is lost (3+5)
  - assignment: passes the value to a variable but the result is not printed automatically (out<-3+5)
  - comment: (#This is a comment)

Naming conventions

- Any roman letters, digits, underline, and '.' (non-initial position)
- Avoid using system names: c, q, s, t, C, D, F, I, T, diff, mean, pi, range, rank, tree, var
- Hold for variables, data and functions
- Variable names are case sensitive

Arithmetic operations and functions

- Most operations in R are similar to Excel and calculators
- Basic: +(add), -(subtract), *(multiply), /(divide)
- Exponentiation: ^
- Remainder or modulo operator: %%
- Matrix multiplication: %*%
- sin(x), cos(x), cosh(x), tan(x), tanh(x), acos(x), acosh(x), asin(x), asinh(x), atan(x), atan(x,y) atanh(x)
- abs(x), ceiling(x), floor(x)
- exp(x), log(x, base=exp(1)), log10(x), sqrt(x), trunc(x) (the next integer closer to zero)
- max(), min(), mean(), median()

Defining new variables

- Assignment symbol, use "<-" (shortcut: alt -) or =
- Scalars

```
>scal<-6
>value<-7
```

- Vectors; using c() to enter data

```
>whales<-c(74,122,235,111,292,111,211,133,16,79)
>simpsons<-c("Homer", "Marge", "Bart", "Lisa", "Maggie")
```

- Factors

```
>pain<-c(0,3,2,2,1)
>fpain<-factor(pain,levels=0:3)
>levels(fpain)<-c("none", "Mild", "medium", "severe")
```

Use functions on a vector

- Most functions work on vectors exactly as we would want them to do

  >sum(whales)

  >length(whales)

  >mean(whales)

  – sort(), min(), max(), range(), diff(), cumsum()

- Vectorization of (arithmetic) functions

  >whales + whales

  >whales - mean(whales)

  – Other arithmetic funs: sin(), cos(), exp(), log(), ^, sqrt()

  – Example: calculate the standard deviation of whales

$$\mathrm{SD}(X) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2}.$$

NJIT New Jersey's Science & Technology University

THE EDGE IN KNOWLEDGE

Functions that create vectors

- Simple sequences

```
>1:10
>rev(1:10)
>10:1
```

```
>c(1:10, 10:1)
>fractions(1/(2:10))
>library(MASS) #to have fractions()
```

- Arithmetic sequence
  - a+(n-1)*h: how to generate 1, 3, 5, 7, 9?

```
>a=1; h=2; n=5
>a+h*(0:(n-1))
```

OR

```
>seq(1,9,by=2)
>seq(1,9,length=5)
```

- Repeated numbers

```
>rep(1,10)
>rep(1:2, c(10,15))
```

  - getting help: ?rep or help(rep)
  - help.search("keyword") or ??keyword

Advanced data structures, I/O & Control Structures

David Li

Matrix

- There are several ways to make a matrix
- To make a 2x3 (2 rows, 3 columns) matrix of 0's:

```
>mat<-matrix(0,2,3)
```

- To make the following matrix:

71	172
73	169
69	160
65	130

```
>mat2<-rbind(c(71,172),c(73,169),c(69,160),c(65,130))
>mat3<-cbind(c(71,73,69,65),c(172,169,160,130))
```

- To make the following matrix:
  - `mat4<-matrix(1:10,2,5, byrow=T)`

1	2	3	4	5
6	7	8	9	10

Revisit vectors: access data

- Accessing individual observations

```
>whales[2]
```

- Slicing

```
>whales[2:5]
```

- Negative indices

```
>whales[-1]
```

- Logical values

```
>whales[whales>100]
>which(whales>100)
>which.max(whales)
```

Indexing of vector/matrix

- x=1:10

ith element	`x[2]` ($i = 2$)
all *but* ith element	`x[-2]` ($i = 2$)
first k elements	`x[1:5]` ($k = 5$)
specific elements.	`x[c(1,3,5)]` (First, 3rd and 5th)
all greater than some value	`x[x>3]` (the value is 3)
bigger than or less than some values	`x[ x< -2 | x > 2]`

- mat=matrix(1:24, nrow=4)

```
mat[,2] # 2nd  column
mat[2,] # 2nd  row
mat[c(2,4),] # 2nd and 4th row
mat[1:3,1]  # 1 to 3 element in column 1
mat[-c(2,4),] # all but row 2 and 4
```

Create logical vectors by conditions

- Logical operators: <, <=, >, >=, ==, !=
- Comparisons
  - Vectors: AND &; OR |
  - Longer forms &&, ||: return a single value
  - all() and any()
- Examples
  - X=1:5
  - X<5; X>1
  - X >1 & X <5; X >1 | X <5;
  - all(X<5); any(X>1); all(X<5) && any(X>1)
- %in% operator: x %in% c(2,4)

Missing values

- R codes missing values as NA
- is.na(x) is a logical function that assigns a T to all values that are NA and F otherwise

```
>x[is.na(x)]<-0
>mean(x, na.rm=TRUE)
```

Reading in other sources of data

- Use R’ s built-in libraries and data sets

```
>range(lynx) #lynx is a built-in dataset
>library(MASS) # load a library
>data(survey)  # load a dataset in the library
>data(survey, package="MASS")#load just data
>head(survey)
>tail(survey)
```

- Copy and paste by scan()

```
>whales=scan()
1: 74 122 235 111 292 111 211 133 156 79
11:
Read 10 items
```

Read formatted data

- Read data from formatted data files, e.g. a file of numbers from a single file, a table of numbers separated by space, comma, tab etc, with or without header

```
>whale=scan(file="whale.txt")
 "whale.txt" :
74 122 235 111 292 111 211 133 156 79
>whale=read.table(file="whale.txt", header=TRUE)
 "whale.txt" :
    texas florida
1   74     89
2   122    254
3   ....   ....
>read.table(file=file.choose()) # specify the file
>read.table(file="http://statweb.stanford.edu/~rag/stat141/exs/whale.txt",header=T)    # read from internet
```

Data frame

- A "data matrix" or a "data set"
  - it likes a matrix (rectangular grid)
  - But unlike matrix, different columns can be of different types
  - Row names have to be unique
- >alphabet<-data.frame(index=1:26, symbol=LETTERS)
- read.table() stores data in a *data frame*
- Access var in a dataset: $, attach(), with()

```
>library(ISwR)  #load the package that provides thuesen data
>data(thuesen)
>names(thuesen)  #variable names
> blood.glucose # not visible
>length(thuesen$blood.glucose)
>with(thuesen, range(blood.glucose))
```

```
>attach(thuesen)
>range(blood.glucose)
>detach(thuesen)
```

More about data frame

- Indexing of data frames is the same as that of vector and matrix >energy[energy$stature== "lean",]
- Sorting rows by order()
  >energy[order(energy$expend),]
  >energy[with(energy, order(stature, expend)),]
- Selecting subsets of data by subset()
  >subset(energy, stature=="lean" & expend>8)
- Splitting data
  >split(energy$expend, energy$stature)

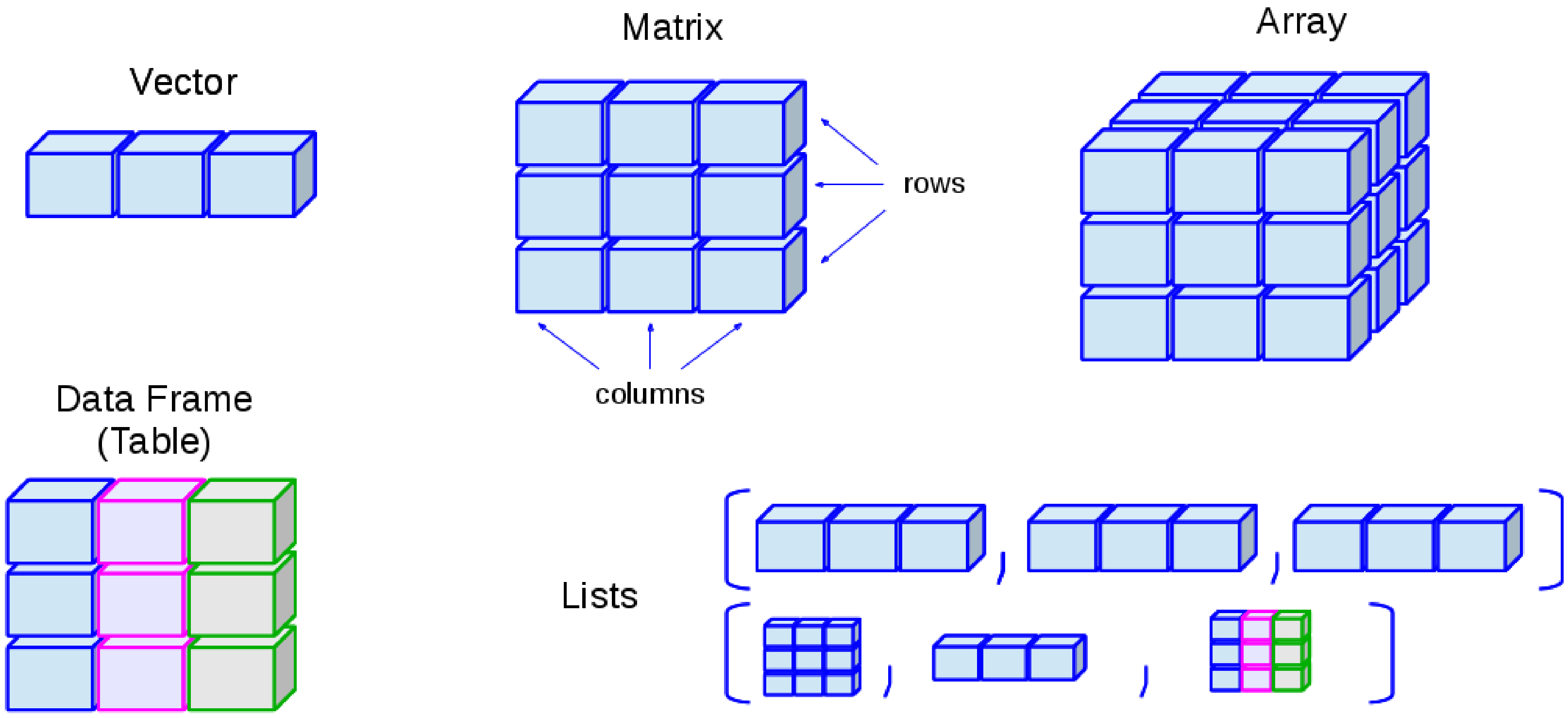
Lists

- A larger composite object for combining a collection of objects
  - Different from data frame, each object can be of different length, in additional to being of different types

```
>a=list(whales=c(74,122,235,111,292,111,211,133,16,79),
 simpsons=c("Homer", "Marge", "Bart", "Lisa", "Maggie"))
```

  - Access by $ or [[]]: a$simpsons or a[[2]]

Summary of data structures



Manage the work environment

- What if there are more variables defined than can be remembered?
- ls() list all the objects(var, fun, etc) in a given environment
- rm(a, b): delete variables a and b
  - rm(list=ls()) will ?
- Get and set working directory

  >getwd()

  >setwd("working/directory/path")
- Save and load working environment

  >save.image(file="filename.RData")

  >load(file="filename.RData")

scripting

- Edit your commands using your favorite text editors
- How to run

Inside R: >source(filename)

  - Takes the input and runs them
  - Do syntax-check before anything is executed
  - Set echo=T to print executed commands

  OR copy & paste

Outside R: R CMD BATCH filename

output is in *.Rout

Or: Rscript filename

How to install packages

- To install CRAN packages, execute from the R console the following command:

  > install.packages('UsingR')

  OR download the package and install it directly

  R CMD INTALL aPackage_0.1.tar.gz

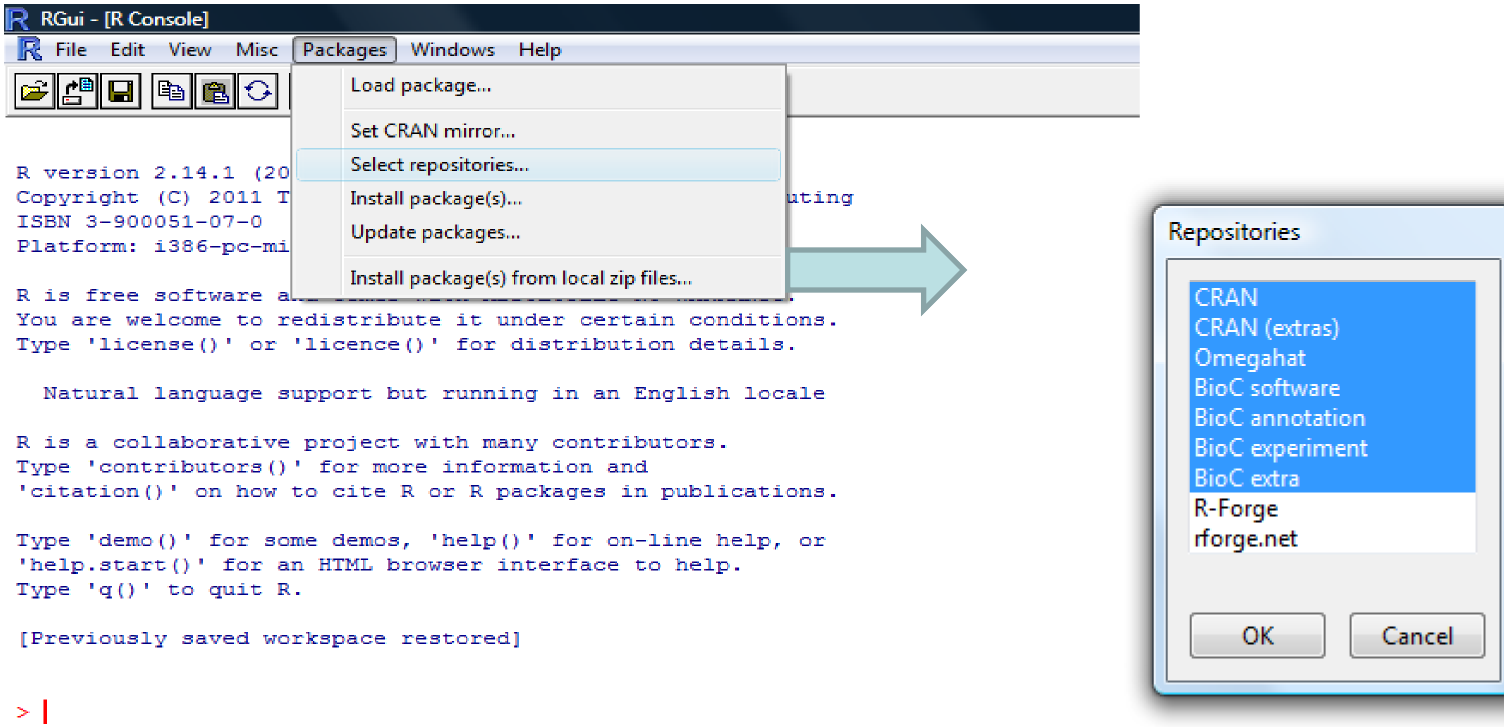
- Load a library

  >library("UsingR")

  or

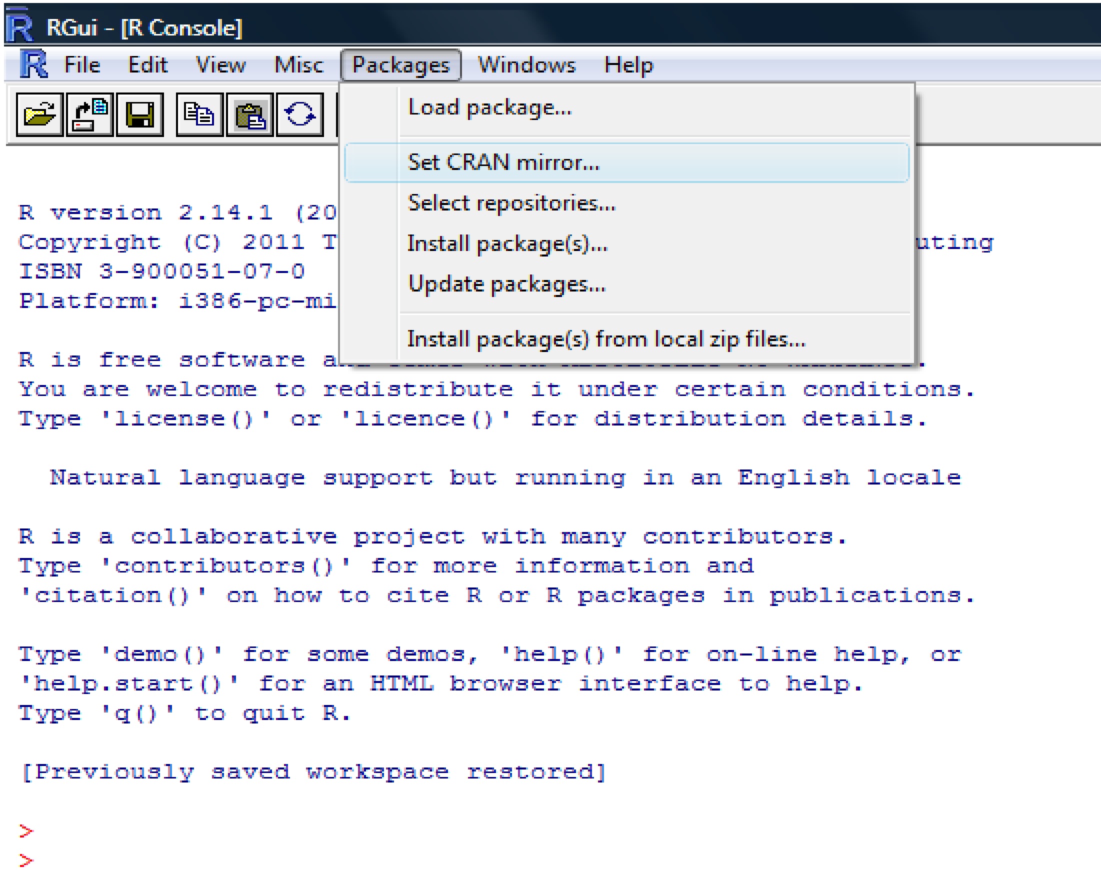
  >library(UsingR)

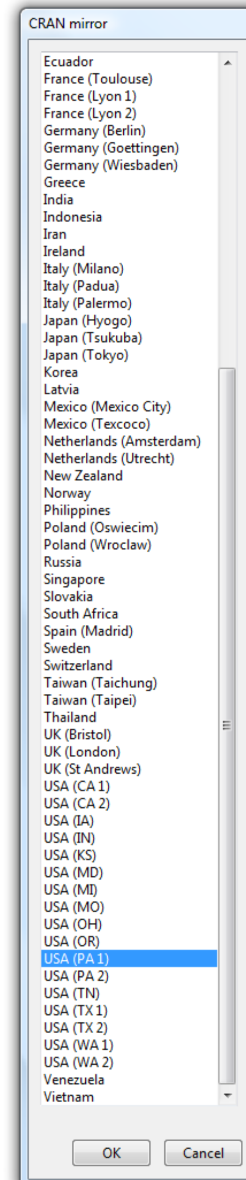
Windows: Set repositories



- Make sure you include necessary repositories (you may simply select all of them)

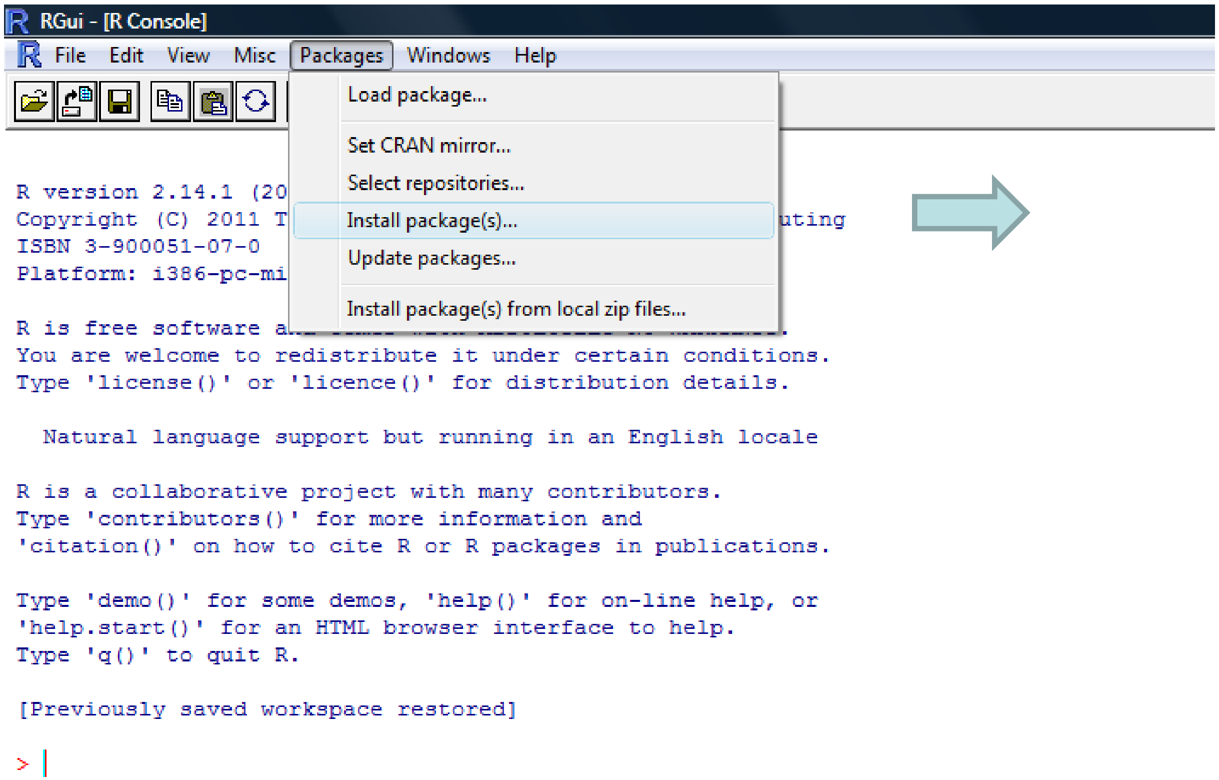
Windows: Set CRAN mirror

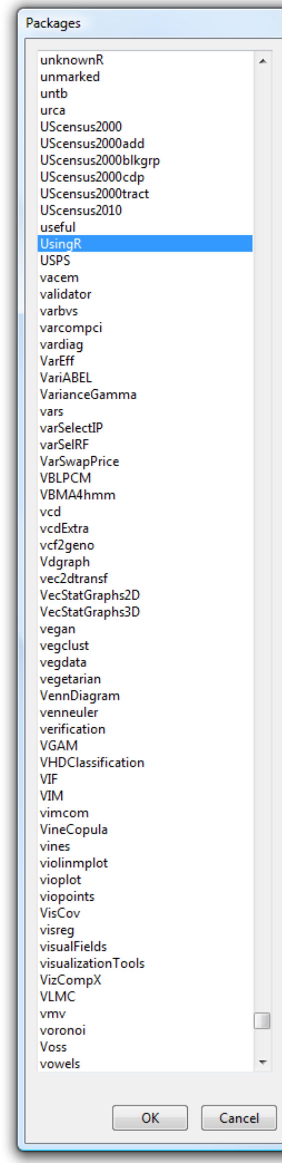




- You can choose anyone but physically close ones are preferred

Windows: install packages





Additional references

- Beginners should print out the R Reference Card

http://cran.r-project.org/doc/contrib/Short-refcard.pdf

- The R-FAQ (Frequently Asked Questions on R)

http://cran.r-project.org/doc/FAQ/R-FAQ.html

- A rather terse introduction to R online

http://cran.r-project.org/doc/manuals/R-intro.html

- Bioconductor

http://www.bioconductor.org/

- A useful online manual for R & Bioconductor

http://manuals.bioinformatics.ucr.edu/home/R_BioCondManual

Acknowledgments

- Jeff Solka: for some of the slides adapted or modified from his lecture slides at George Mason University
- Brian Healy: for some of the slides adapted or modified from his lecture slides at Harvard University

Getting Data In and Out of R

Principal functions reading data

- read.table, read.csv, for reading tabular data
- readLines, for reading lines of a text file
- source, for reading in R code files (inverse of dump)
- dget, for reading in R code files (inverse of dput)
- load, for reading in saved workspaces
- unserialize, for reading single R objects in binary form

Principal functions writing data

- write.table, for writing tabular data to text files (i.e. CSV) or connections
- writeLines, for writing character data line-by-line to a file or connection
- dump, for dumping a textual representation of multiple R objects
- dput, for outputting a textual representation of an R object
- save, for saving an arbitrary number of R objects in binary format (possibly compressed) to a file.
- serialize, for converting an R object into a binary format for outputting to a connection (or file).

Video

- https://youtu.be/Z_dc_FADyi4
- ?read.table

read.table(file="http://statweb.stanford.edu/~rag/stat141/exs/whale.txt"),header=T) # read from internet

- ?readLines

readLines("http://statweb.stanford.edu/~rag/stat141/exs/whale.txt")

Using dput() and dump()

- dput()/dget()
  - y <- **data.frame**(a = 1, b = "a")
  - dput(y)
  - dput(y, file = "y.R")
- dump()/source()
  - x <- "foo"; y <- **data.frame**(a = 1L, b = "a")
  - dump(c("x", "y"), file = "data.R")
  - **rm**(x, y)
  - **source**("data.R")
  - str(y)

Difference between dput() and dump()

- dump can be used to output multiple objects
- dump adds the object name and can be source()'d

Binary Formats save()/load()

- a <- **data.frame**(x = rnorm(100), y = runif(100))
- b <- **c**(3, 4.4, 1 / 3)
- *## Save 'a' and 'b' to a file*
- **save**(a, b, file = "mydata.rda")
- *## Load 'a' and 'b' into your workspace*
- **load**("mydata.rda")
- *## Save everything to a file*
- **save.image**(file = "mydata.RData")
- *## load all objects in this file*
- **load**("mydata.RData")

Reading in Larger Datasets with read.table

- Video https://youtu.be/BJYYIJO3UFI
- A tip

```
> initial <- read.table("datatable.txt", nrows = 100)
> classes <- sapply(initial, class)
> tabAll <- read.table("datatable.txt", colClasses = classes)
```

Summary

- write.csv() and write.table() are used when you want to exchange data in tabular text format.
- dput() saves single data object in R code
- dump() saves multiple data objects and their metadata in R code
- save() is similar to dump() but saves in binary format or ASCII
- save.image() saves workspace in binary format

Calculating Memory Requirements for R Objects

- An example: a data frame with 1,500,000 rows and 120 columns, all of which are numeric data.
  - 1,500,000 × 120 × 8 bytes/numeric

Control Structures

Commonly used control structures

- if and else: testing a condition and acting on it
- for: execute a loop a fixed number of times
- while: execute a loop *while* a condition is true
- repeat: execute an infinite loop (must break out of it to stop)
- break: break the execution of a loop
- next: skip an iteration of a loop

if-else

- **if**(<condition>) {

do something

}

Continue with rest of code

- **if**(<condition>) {

do something

}

else {

do something else

}

if-else {if-else}

```
if(<condition1>) {
## do something
} else if(<condition2>) {
## do something different
} else {
## do something different
}
#-----------------------
if(<condition1>) {
}
if(<condition2>) {
}
```

Example

```
x <- runif(1, 0, 10)
```

- ```
  if(x > 3) {
    y <- 10
  } else {
    y <- 0
  }
  ```
- ```
  y <- if(x > 3) {
    10
  } else {
    0
  }
  ```
- ```
  y <- ifelse(x>3, 10, 0)
  ```
  ```

ifelse()

- x <- c(6:-4)
- sqrt(x) #- gives warning
- sqrt(ifelse(x >= 0, x, NA)) # no warning
- ## Note: the following also gives the warning !
- ifelse(x >= 0, sqrt(x), NA)

- ## example of different return modes:

yes <- 1:3
no <- pi^(0:3)
typeof(ifelse(NA, yes, no)) # logical
typeof(ifelse(TRUE, yes, no)) # integer
typeof(ifelse(FALSE, yes, no)) # double

for Loops

- **for**(i **in** 1:10) {

 print(i)

}

x <- **c**("a", "b", "c", "d")

- **for**(i **in** 1:4) {

Print out each element of 'x'

 print(x[i])

}

for Loops (cont'd)

- seq_along() function is commonly used in conjunction with for loops

```
for(i in seq_along(x)) {
    print(x[i])
}
```

- It is not necessary to use an index-type variable

```
for(letter in x) {
    print(letter)
}
```

- One line loops (curly braces are not required)

```
for(i in 1:4) print(x[i])
```

Nested for loops

```
x <- matrix(1:6, 2, 3)
for(i in seq_len(nrow(x))) {
  for(j in seq_len(ncol(x))) {
    print(x[i, j])
  }
}
```

while Loops

while (<condition>) {
  ## do something
}

Example:

```
count <- 0
while(count < 10) {
 print(count)
 count <- count + 1
}
```

While loops can potentially result in infinite loops if not written properly. Use with care!

repeat

```
x0 <- 1
tol <- 1e-8
repeat {
    x1 <- computeEstimate()
    if(abs(x1 - x0) < tol) { ## Close enough?
      break
    } else {
      x0 <- x1
    }
}
```

next, break

- next is used to skip an iteration of a loop.

```
for(i in 1:100) {
  if(i <= 20) {
    ## Skip the first 20 iterations
    next
  }
## Do something here
}
```

- break is used to exit a loop immediately, regardless of what iteration the loop may be on.

```
for(i in 1:100) {
  print(i)
  if(i > 20) {
   ## Stop loop after 20 iterations
   break
  }
}
```

Summary

- Control structures like if, while, and for allow you to control the flow of an R program
- Infinite loops should generally be avoided, even if (you believe) they are theoretically correct.
- Control structures mentioned here are primarily useful for writing programs; for commandline interactive work, the "apply" functions are more useful (will cover later)
- It is more efficient to use built-in functions or vectorization rather than control structures whenever possible.