Where to get the exercises

http://crio.uchicago.edu/?page_id=1207

If you have **Windows** laptop, install xMing:

http://sourceforge.net/projects/xming/

Follow instructions on how to install and connect via x-tunneling:

https://ibi.uchicago.edu/education/downloads/Installing_and_using_Xming.pdf

For Macs laptop:

https://ibi.uchicago.edu/education/downloads/tunneling_for_mac.pdf

University of Chicago
Center for Research Informatics

Introduction to R and Submitting to HPC

Alex Rodriguez
Jorge Andrade

Outline for the Day

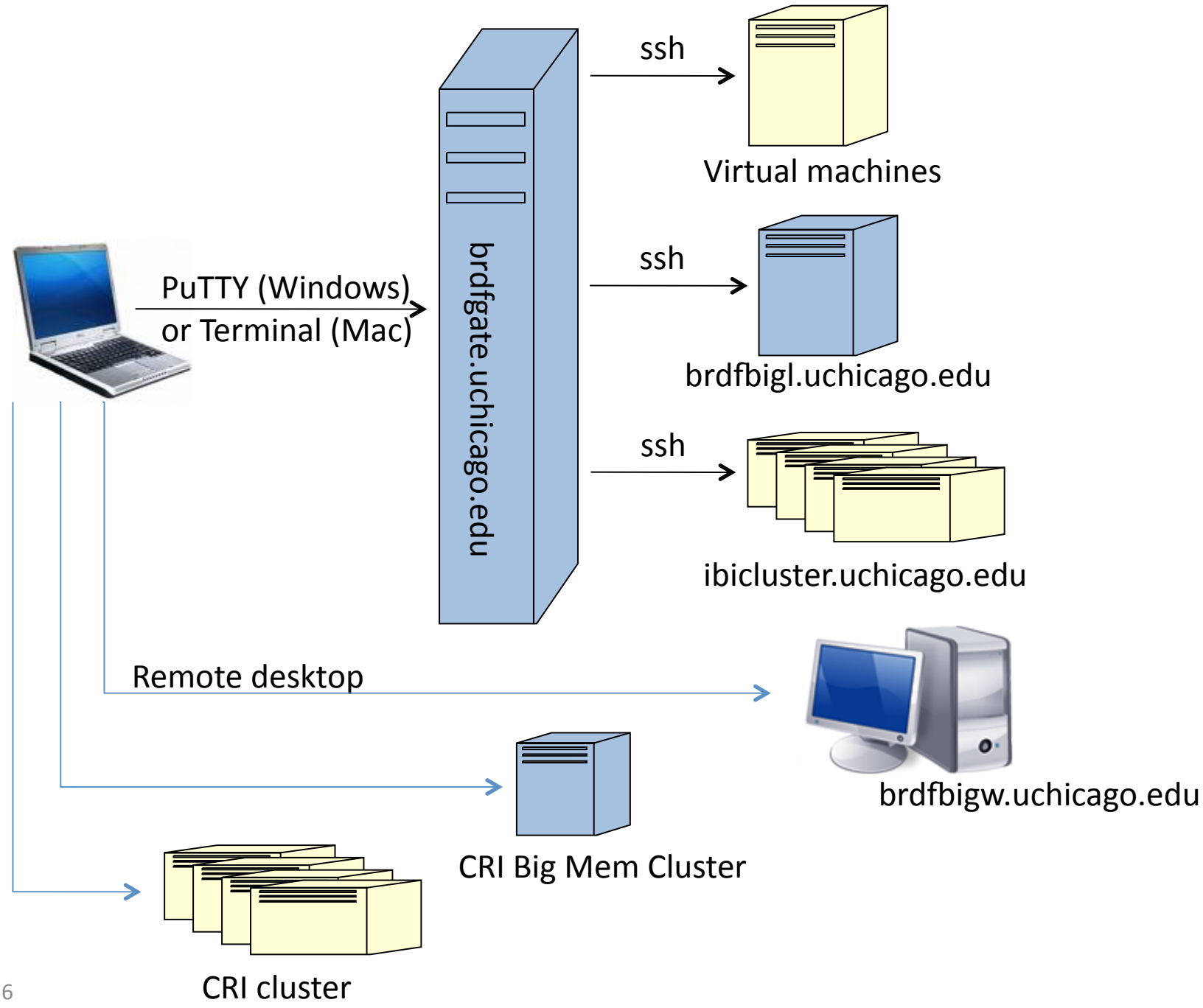
- CRI resources and support
- Accessing R on CRI systems
- Introduction to R
  - Objects
  - Basic operations
  - Statistics
  - Graphs
- Exercise 1 – creating a graph from biological data
- Exercise 2 – Submitting R script to Cluster

University of Chicago
Center for Research Informatics

CRI Support and Resources

Resources Summary

- Main Cluster (ibicluster)
  - 94 Nodes, 8 cores => 752 processors
  - 16GB RAM/per Node => 4GB per job
  - Each user can submit as many jobs as they want, but have at most 92 jobs running at one time
  - No reservations
  - Access to all Linux applications
- Large Memory Computation Linux (brdfbigl, l-cri-bmem)
  - brdfbigl ➔ 256 GB RAM shared
  - l-cri-bmem ➔ 1 TB RAM shared
  - No reservations
  - Access to all Linux applications
- Large Memory Computation Windows (brdfbigw)
  - 64 bit, 256 GB RAM
  - Access to many Windows applications
- Cloud applications (Galaxy)
- Project Shares


ssh
Virtual machines
ssh
brdfbigl.uchicago.edu
PuTTY (Windows)
or Terminal (Mac)
brdfgate.uchicago.edu
ssh
ibicluster.uchicago.edu
Remote desktop
brdfbigw.uchicago.edu
CRI Big Mem Cluster
CRI cluster

CRI Support

- CRI Website:
  - http://cri.uchicago.edu
- Support Email Address
  - cri-support@ci.uchicago.edu

Disclaimer

- R is typically used for statistics; there are several topics related to R that we are NOT going to cover:
  - Statistics in general/statistical analysis of data
  - Bayesian analysis
  - Linux commands

University of Chicago
Center for Research Informatics

Introduction to R

Ultra-short R introduction

Most life scientists use spreadsheet programs (like excel for data analysis) Why?

Ease to use

- Click buttons, select data by hand

- You see the data in front of you

- You can do limited programming

Disadvantages of spreadsheet

- Hard to handle large dataset (>1000 data points)
- Inflexible, few analyses available
- Hard to repeat analyses systematically with new data

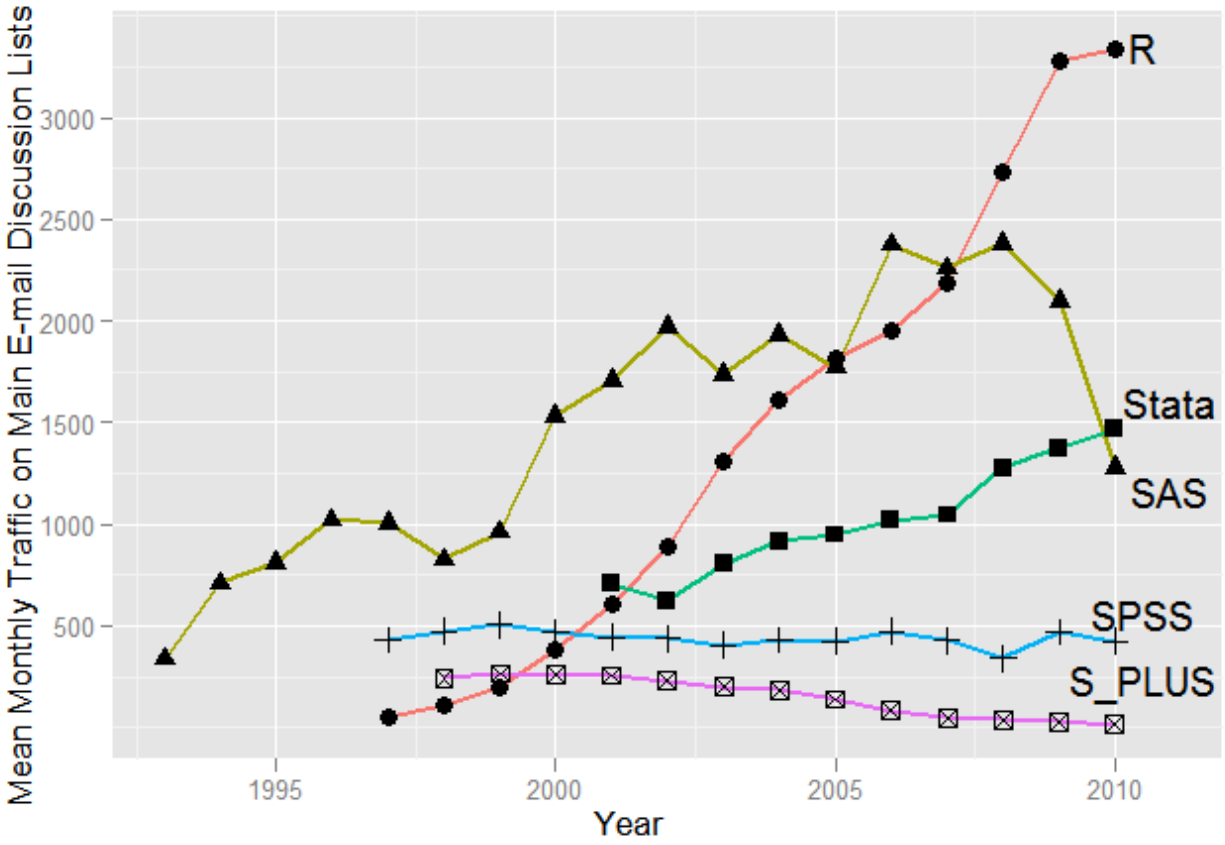
R

Advantages

- R is a computational environment - somewhere between a program and a programming language
- No buttons, no wizards: only a command line interface
- Is a professional statistics toolset – likely the only analyses tool you will ever need
- Is also a programming language
- Can handle large datasets
- Very powerful graphics
- State-of-the-art statistics program for **bioinformatics**
- Free, and open source!

R popularity

- R is a statistical tool
- Simple to use command line environment
- New packages added every month
- Used in many fields
- BioConductor
- Excel cannot handle 1000's of lines, R is almost instant



R Websites

- CRAN: http://cran.r-project.org/
  - Manuals: http://cran.r-project.org/manuals.html
  - FAQs: http://cran.r-project.org/faqs.html
  - Contributed Guides: http://cran.r-project.org/other-docs.html
- R Home: http://www.r-project.org/
  - R Wiki: http://wiki.r-project.org/
  - R Journal: http://journal.r-project.org/
  - Mailing Lists: http://www.r-project.org/mail.html
  - Bioconductor: http://www.bioconductor.org/

Where to find R

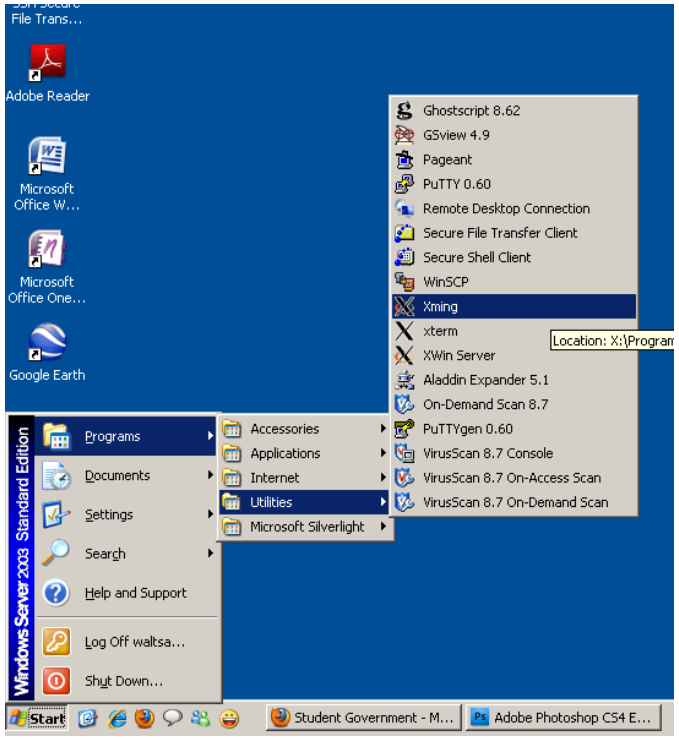
- R is on all of our Linux machines including the clusters
- R can be installed locally on your personal computer from the CRAN website
- In windows, R will exist in the start menu. Load it by clicking the icon.

Using R – Linux Shell

- If you don't need to see graphs, simply connect to brdfgate.uchicago.edu
- Login
- ssh to brdfbigl (`$ ssh brdfbigl`)
- Type 'R' in the command prompt (`$ R`)

Using R – Linux Shell with Graphics

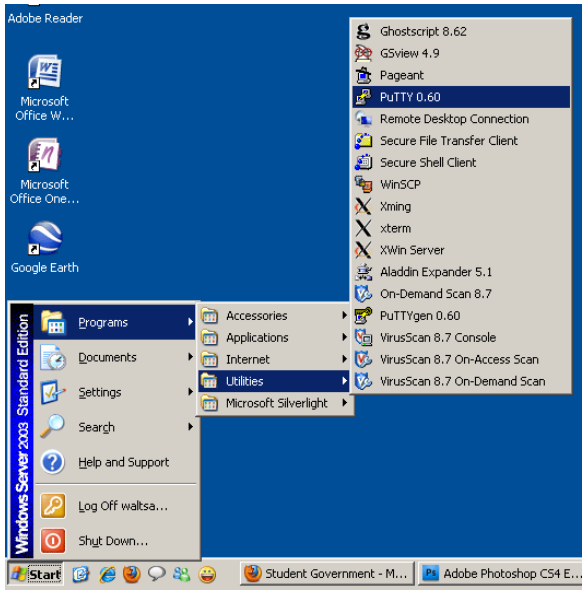
- To see graphs while using the CRI Linux server, we must use Xming

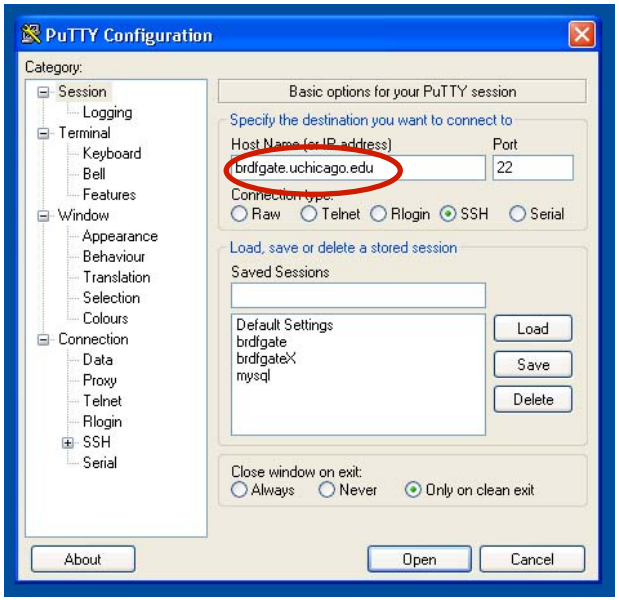


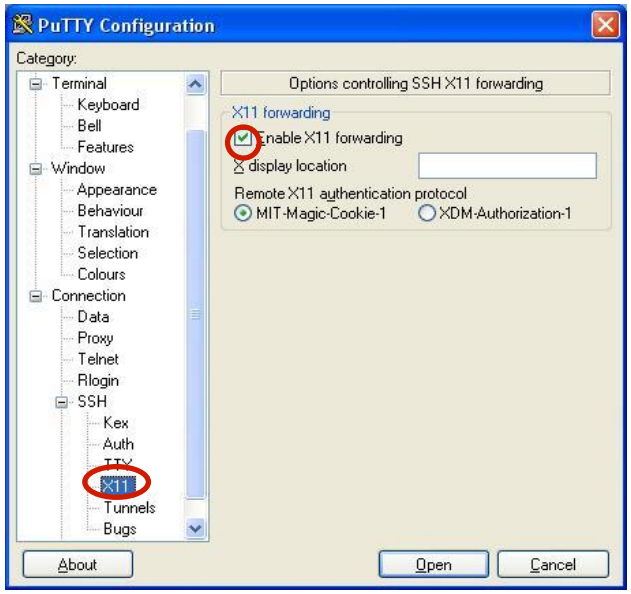
- We can see X running in the tray

Using R – Linux Shell with Graphics

- Connect with PuTTY, be sure to enable X11 forwarding





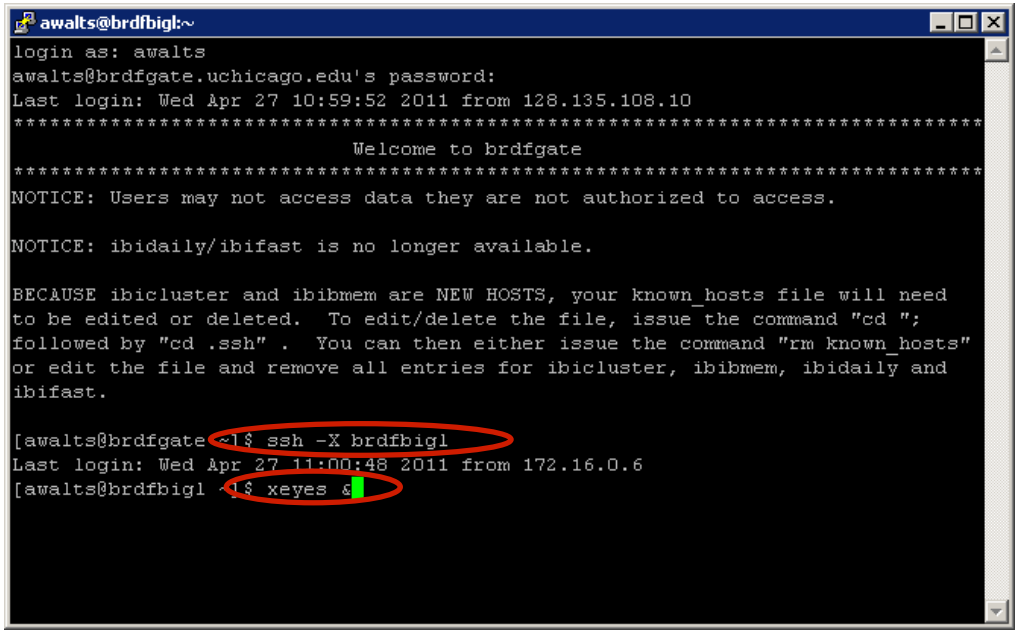


Using R – Linux Shell with Graphics

- Login with the workshop account or your personal account
- ssh to brdfgate with –Y flag:

  `$ ssh -Y brdfgate`
- Ssh to brdfbigl with `-X` flag:

  `$ ssh -X brdfbigl`



Basics

R Command Line

- R's main prompt is:

  `>`

  This means R is ready for a command
- R's secondary prompt is:

  `+`

  This means R is waiting for you to complete the current command
- Ctrl-c or Esc will stop processing.
- Quit R with: `> q()`

Commands

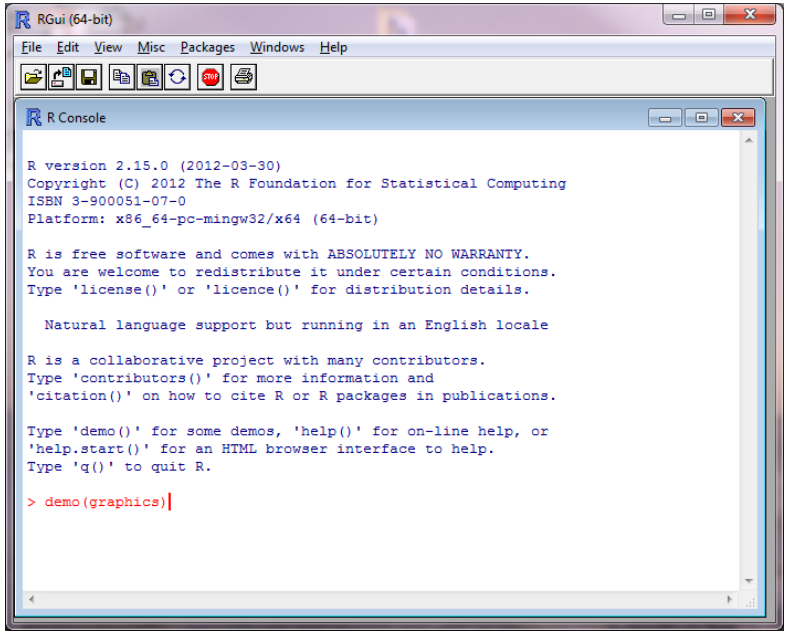
- Elementary commands are either expressions or assignments.
  - Expressions will be evaluated, printed, and the value is lost
  - Assignments also evaluated an expression, but it passes the value to a variable for use later. The result is not automatically printed.
- Commands are separated by a semi-colon(;) or a newline

Entering Commands

- R is case-sensitive
- Name your variables with letters, numbers, the dot (.), and the underscore (_)
- Begin variable names with a letter
- Tab-completion works
- Use up and down arrows to see commands you have used during the current session

First challenge

- Start R
- Type: demo(graphics)
- Hit enter a few times



Getting help in R

- R has several built-in ways to get help

```
> help("data.frame")
> ? data.frame
> browseVignettes("Biobase")
```

- R mailing list
  - http://www.r-project.org/mail.html , R-help section
  - Read the posting guide before sending anything
  - Be sure to include the information from the `sessionInfo()` command:

```
> sessionInfo()
```

Objects

- All entities created in R are known as objects. Some examples are:
  - Variables
  - Arrays of numbers
  - Character strings
  - Functions
- The collection of objects currently stored is the workspace. To see the objects in the workspace use either:

```
> objects()
```

or

```
> ls()
```

Libraries (Packages)

- R has a number of libraries (packages) available for use
- Most are user-contributed and have been tested to varying degrees
- To see packages available to load:

```
> library()
```

- To load a package so you can use its functions:

```
> library(packageName)
```

- To see the packages currently loaded:

```
> search()
```

Getting New Packages

- The CRAN website maintains a list of packages for download: Package list
- There are a lot of libraries that don't come with the basic R install package
- The Best Practice is to notify us of a package you want
- We'll install it so everyone can use it

Data Structures

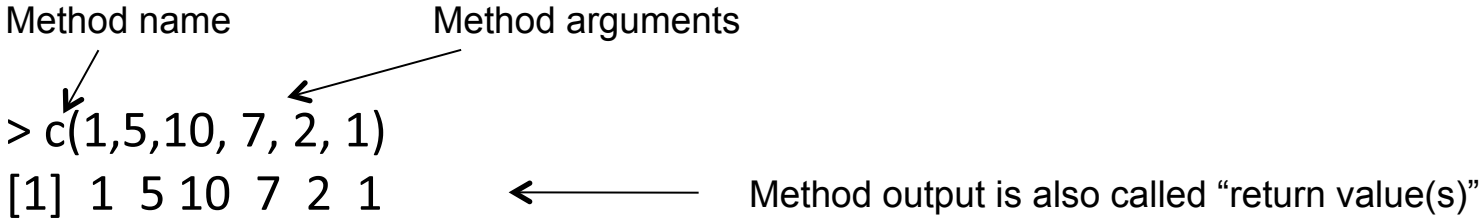
Basic Structures

- vector — An indexed array containing elements that are all the same type
- factor — A vector of categorical descriptors
- data.frame — A matrix-like structure in which the columns can be of different types (most likely numeric and categorical)

Vectors

- The simplest structure in R is a numeric vector
- It is a single entity consisting of an ordered collection of numbers.
- One-based indexing
- A vector is an collection of numbers and/or strings:
  - (“jorge”, “alex”, “ron”)
  - (10, 5.2, 1, 7, 2, 21)
  - (3)
- There are a number of functions we can use to create vectors:
  - `c()` :concatenate
  - `:` (the colon) :integer sequence
  - `seq()` :general sequence
  - `rep()` :repetitive patterns
  - `vector()` : vector of the given length with default value

In R, we make a vector by the c() command (for *concatenate*)



> c('jorge', 'alex', 'ron')
[1] "jorge" “alex" “ron"

When we input strings or characters, we have to surround them with “ or ‘
If making vectors of size 1, we can skip c()

> 3
[1] 3

```
> ls()                          # List the contents of the workspace.
> rm(list=ls())                 # This completely clears the workspace.
> ls()
character(0)                    # This means "nothing to see here"
```

Challenge:

1) Make the following vector

45,5,12,10

2) What happens with the following command?

c(1:100)

c(50:2)

A vector is a *data structure*, and the most fundamental in R. Almost everything in R is some kind of vector, although sometimes in several dimensions – vectors within vectors

Examples: Other methods for creating vectors

- **The colon :**

```
> 1:3
> 2*3:15
> 30:1
```

- **`seq()`**

```
> seq(1,3)
> seq(3,1)
> seq(-5, 5, by=.2)
```

- **`rep()`**

```
> rep(1:2, 3)
```

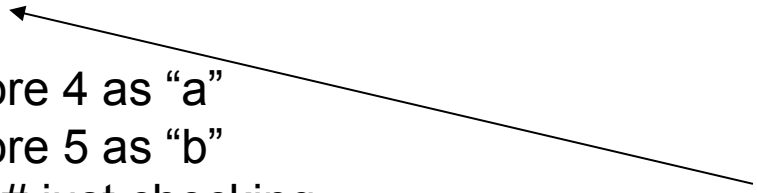
Reference sheet is your friend!

- You will get over-whelmed by different command names fast
- Use the reference sheet to remind yourself in all exercises

Assignment to memory

- The c() command is almost useless in itself - we want to keep the vector for other analyses
- The assignment concept:

```
> 4+5          # add 4 and 5
[1] 9
> a<-4         # store 4 as “a”
> b<-5         # store 5 as “b”
> a                  # just checking
[1] 4
> b
[1] 5
> a+b          # add a+b (4+5)
[1] 9
```



Anything after a # will be disregarded by R. Used for making comments

Expanding this to a whole vector
my_vector<- c(1,5,10, 7, 2)

Note that there is no “return value” now - this is caught by the “my_vector”.

What happens if you type my_vector after this?

my_vector is a *variable*, with the variable name: my_vector. Variable names are totally arbitrary!
The anatomy of the vector:

Name	my_vector				
Values	1	5	10	7	2
Index	[1]	[2]	[3]	[4]	[5]

```
> my_vector[c(1,3,5)]
[1]  1 10  2
> my_vector[1:4]
[1]  1  5 10  7
```

Making a series of integers:
A:B will give a vector of A,A+1, A+2…B

```
>my_vector[4:1]   # also works backwards
[1]  7 10  5  1
```

We can access part of the vector like this:

```
my_vector[5]   # will give the 5th item in the vector
```

What happens if you do this?

```
my_vector<- c(1,5,10, 7, 2)          # define the vector

my_vector [c(1,3,5)]

my_vector[1:4]

my_vector[4:1]
```

Subsetting vectors – Positive Indices

- Access subsets of vectors via `[subscript]`

```
> x<-11:20
> x[2:4]
  # Returns values 2 through 4 in vector x
```

- Subset expressions can appear on the left side of an assignment:

```
> x[8:10] <-44
  # 44 is assigned to values 8 through 10 in vector x
```

Subsetting vectors – Negative Indices

- Can use negation to select all indices except a given subset

```
> x[-(1:3)]
```

  # Returns all elements in vector x EXCEPT 1 through 3

- Negative subscripts can appear on the left side of an assignment

```
> x[-(8:10)] = 33
```

  # Assign 33 to all values in vector x EXCEPT 8 through 10

- Cannot mix positive and negative subscripts

Subsetting vectors – Logical Predicates

- Can use logical vectors to obtain subsets

```
> x <- 11:20
> x > 15
  # Returns a logical vector.  Each element is either true or false accordingly
> x[x > 15]
  # Returns a vector of all the values of x that are greater than 5
```

- The subset vector can be longer than the base vector; values selected beyond the end of the vector produce `NA`s.

Examples – Subsetting Vectors

- Subsetting

```
> x<-11:20
> x[2:4]
> x[8:10] <-44
> x[-(5:8)] = 33
> x<-11:20
> x[x > 16]
> logic <- x > 17
```

Challenge

- figure out at least three ways of making R print your vector in the other direction

```
my_vector<- c(1,5,10, 7, 2)          # define the vector
```

Challenge Solution

```
> my_vector[5:1]
> c <- c(my_vector[5],my_vector[4],my_vector
  [3],my_vector[2], my_vector[1])
> my_vector[c(5,4,3,2,1)]
> rev(my_vector)
```

Factors

- A vector object used to specify a discrete classification, or grouping, of the components of other vectors of the same length
- One-based indexing

```
> col <- c("green", "blue", "hazel",
"hazel", "brown", "hazel", "blue",
"brown")
> factor(col)
> col.factor<- factor(col)
```

- Subsetting, including assignment, also works on factors

data.frame

- Data frames are a special R structure used to hold a set of spreadsheet like table. In a `data.frame`, the observations are the rows and the covariates are the columns.
- Can contain objects of different types
- One-based indexing
- Data frames can be treated like matrices and be indexed with two subscripts. The first subscript refers to the observation (row), the second to the variable (column).
- Data frames are really lists, and list subsetting can also be used on them.

Reading Data from a File

- There are several `read` methods
- `read.table`, `read.csv`, `read.delim`
- All have different defaults and arguments you can set
- All read a file in table format and create a data.frame from it. Cases are the rows in the file; variables are the columns in the file.

Using external files

- Commands stored in a file in the working directory can be executed via:

```
> source("commands.R")
```

- Output can be diverted to a file via:

```
> sink("recordOutput.txt")
```

  – All subsequent output will be diverted to "recordOutput.txt"

- To restore output to the console use:

```
> sink()
```

R Scripts on the Command Line or Cluster

- Write an R script
  - R commands typed on the R shell
- This runs R from the command line in batch mode or can be run on cluster environment
  - make sure packages are installed on cluster
  - the log file is output to track R's progress

```
$ R CMD BATCH --vanilla path/to/file.R file.log &
```

OR

```
$ Rscript path/to/file.R &
```

Exercise 1

Diabetes in Female Pima Indians

Exercise Overview

- The data are from the R package ‘faraway’.
- Do an initial data analysis
- Clean the data
- Make some graphs

```
#Clear existing data and graphics
rm(list=ls())
graphics.off()
#Load Hmisc library
library(Hmisc)
#Read Data
data=read.csv('/biodbA/WorkshopDir/R_BioConductor/DATA_WHBIOCONDUCTOR_DEMO.CSV')
#Setting Labels

label(data$study_id)="Subject ID"
label(data$pregnant)="Number of pregnancies:"
label(data$glucose)="Plasma glucose concentration at 2 hours in an oral glucose tolerance test"
label(data$diastolic)="Diastolic blood pressure"
label(data$triceps)="Triceps skin fold thickness"
label(data$insulin)="2 hour serum insulin"
label(data$bmi)="Body mass index"
label(data$age_pima)="Age"
label(data$test)="Test for signs of diabetes"
label(data$pima_complete)="Complete?"
#Setting Units

#Setting Factors(will create new variable for factors)
data$test.factor = factor(data$test,levels=c("1","0"))
data$pima_complete.factor = factor(data$pima_complete,levels=c("0","1","2"))

levels(data$test.factor)=c("Positive","Negative")
levels(data$pima_complete.factor)=c("Incomplete","Unverified","Complete")
```

Using the Data

- The data must be in R's current working directory OR you must provide the absolute path to the data.
- Find out what the current working directory is and change it with the following commands:

```
> getwd()
> setwd("dir")
```

- Use the source file via:

```
>source("/biodbA/WorkshopDir/R_Bioconductor/
  EXPORT_BIOCONDUCTOR_DEMO.R")
```

- The data are now loaded

Initial Data Analysis

- By default, the data will be loaded into a data.frame object called 'data'
- We can see the object's class via the `class()` command:

```
> class(data)
```

- I'm going to rename the data

```
> pima<-data
```

- Look at the data:

```
> pima
> head(pima)
> summary(pima)
```

summary()

- Looking at `pregnant`, we see a max value of 17. This is large, but not impossible.
- We see that the minimum values for `glucose`, `diastolic`, `triceps`, `insulin`, and `bmi` are all 0. This is definitely a problem!
- Look at the sorted values:

```
> sort(pima$diastolic)
```

Clean the Data

- We will set all the zero values of the five mentioned variables to `NA`, the missing value code used by R:

```
> pima$diastolic[pima$diastolic==0] <-NA
> pima$glucose[pima$glucose==0] <-NA
> pima$triceps[pima$triceps==0] <-NA
> pima$insulin[pima$insulin==0] <-NA
> pima$bmi[pima$bmi==0] <-NA
```

- Now if we use the `summary()` command again, things look better

Factor Variables

- This dataset has one factor variable, `test`
- The R script has done some work for us and has converted the `test` variable into a factor named `test.factor`
- The R script has coded the variable values for 0 being negative and 1 being positive

Creating Factor Variables

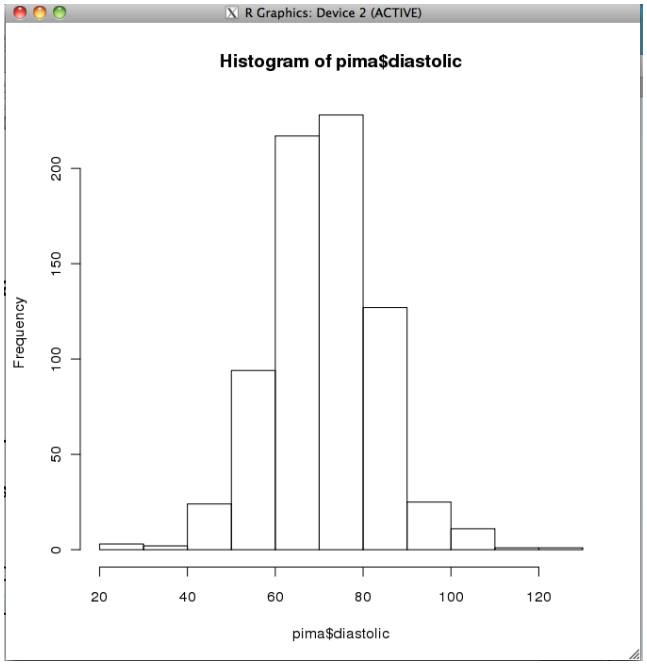
- If you have a dataset in which this conversion was not done for you, the following commands will create a factor variable

```
> dataSet$factorVar <- factor(dataSet$factorVar)
> summary(dataSet$factorVar )
> levels(dataSet$factorVar ) <- c("name1", "name2", …)
```

Graph the Data

- Now the data is clean, we're ready to make some graphs:
- Histogram

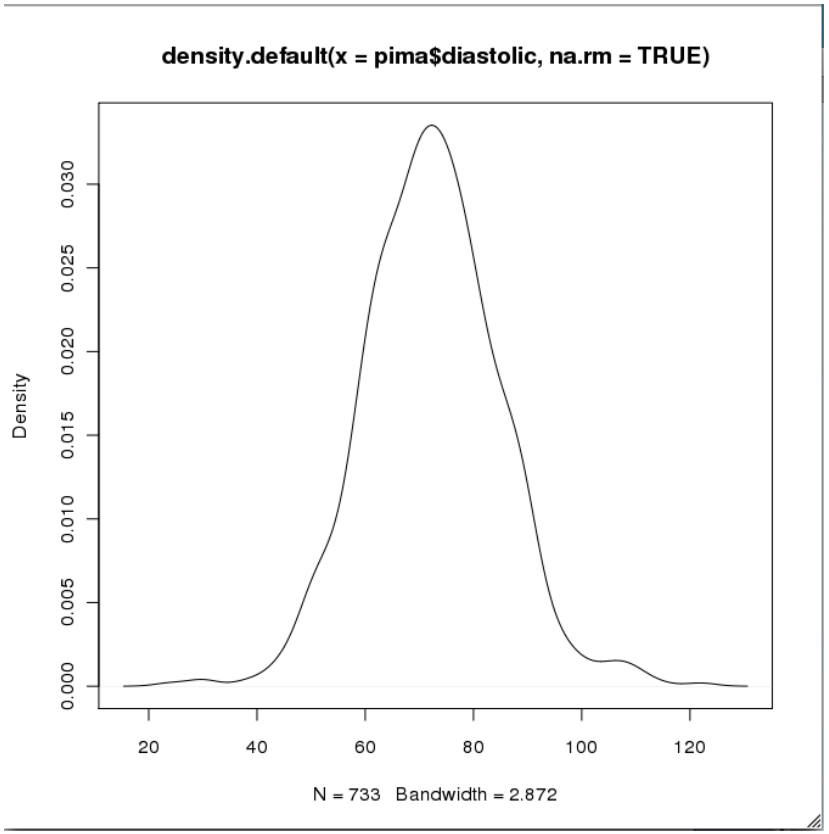
```
> hist(pima$diastolic)
```



Graph the Data

- Kernel Density Plot

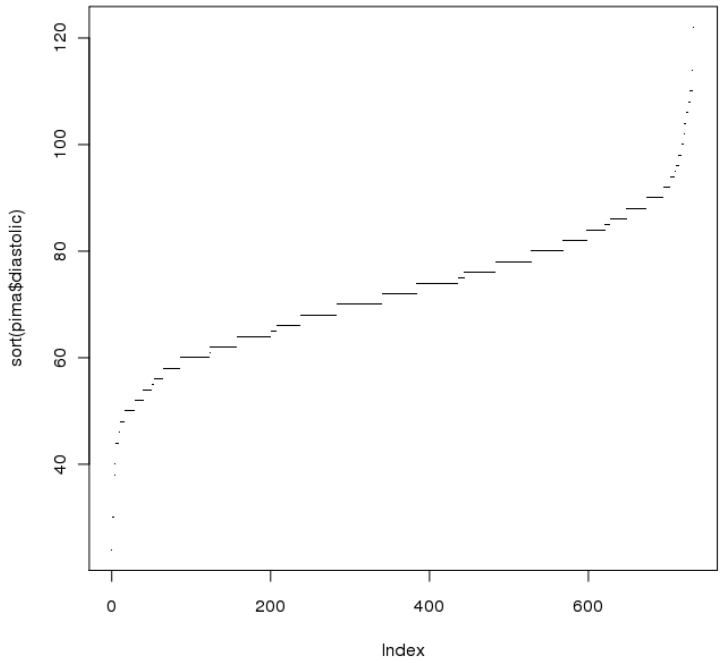
```
> plot(density(pima$diastolic,
na.rm=TRUE))
```



Graph the Data

- Index plot

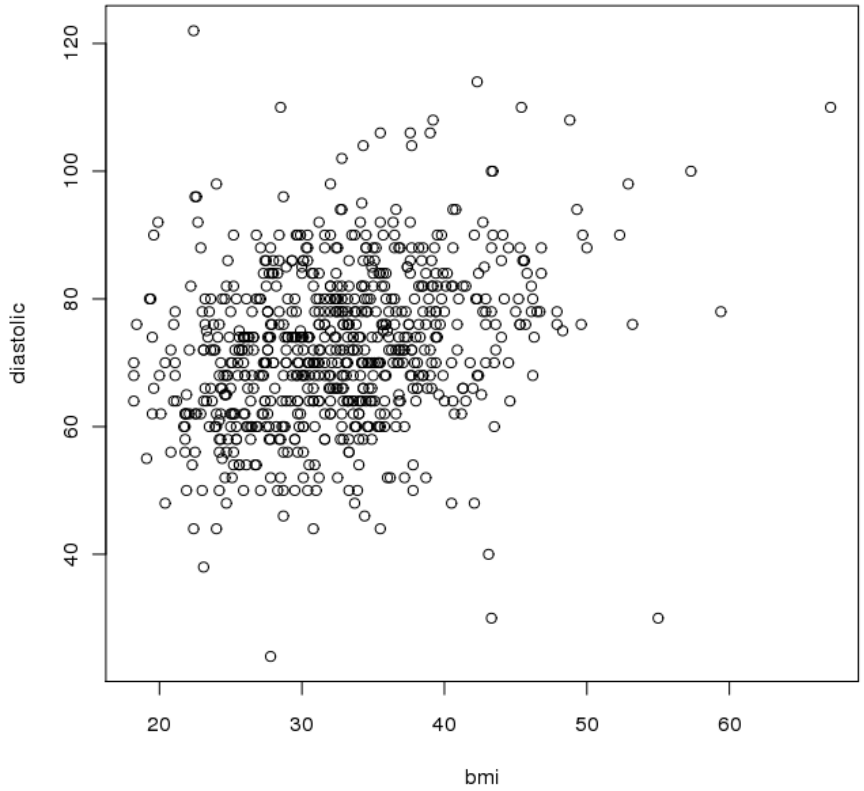
```
> plot(sort(pima$diastolic),pch=".")
```



Graph the Data

- Bivariate plots
  - Scatterplot

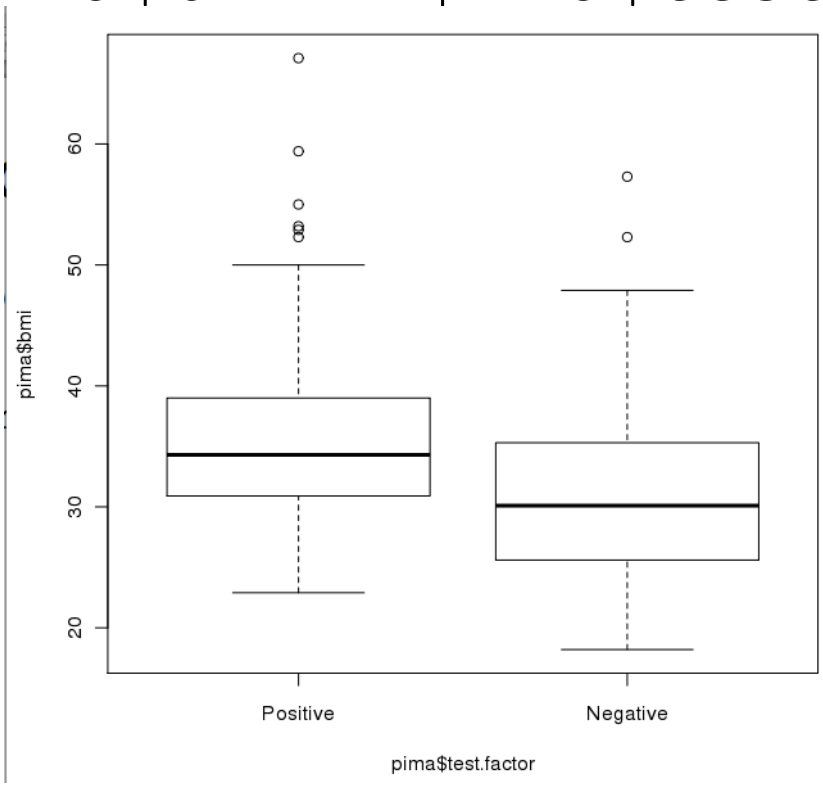
```
> plot(diastolic ~ bmi, pima)
```



Graph the Data

- Bivariate plots
  - Boxplot

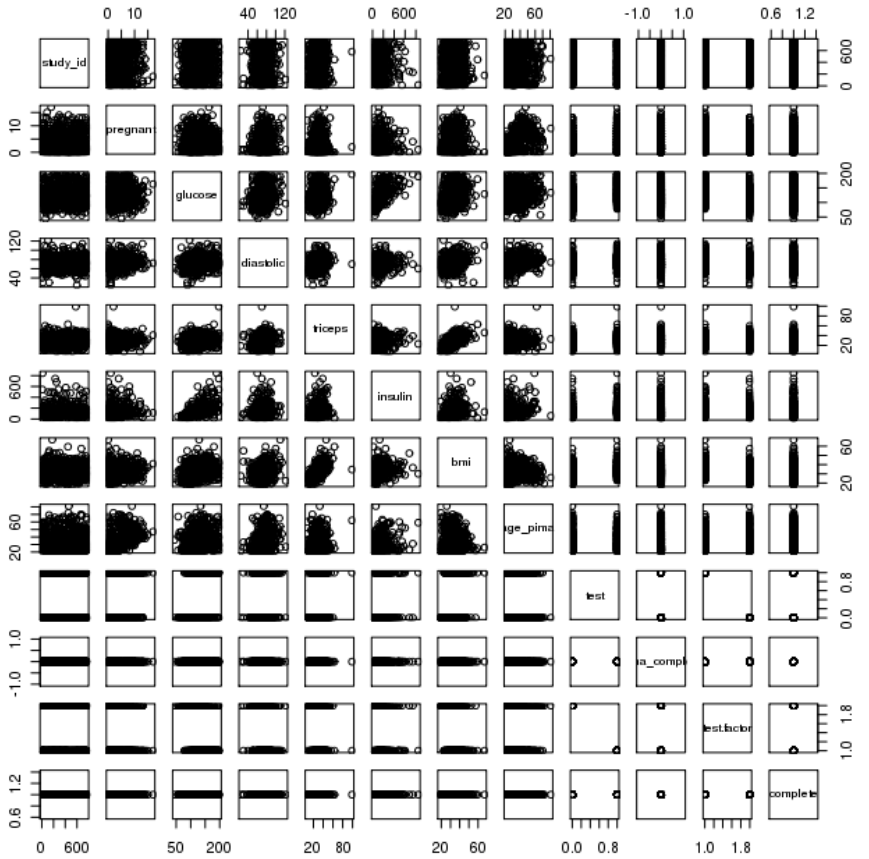
```
> plot(pima$bmi ~ pima$test.factor)
```



Graph the Data

- Scatter plot matrix

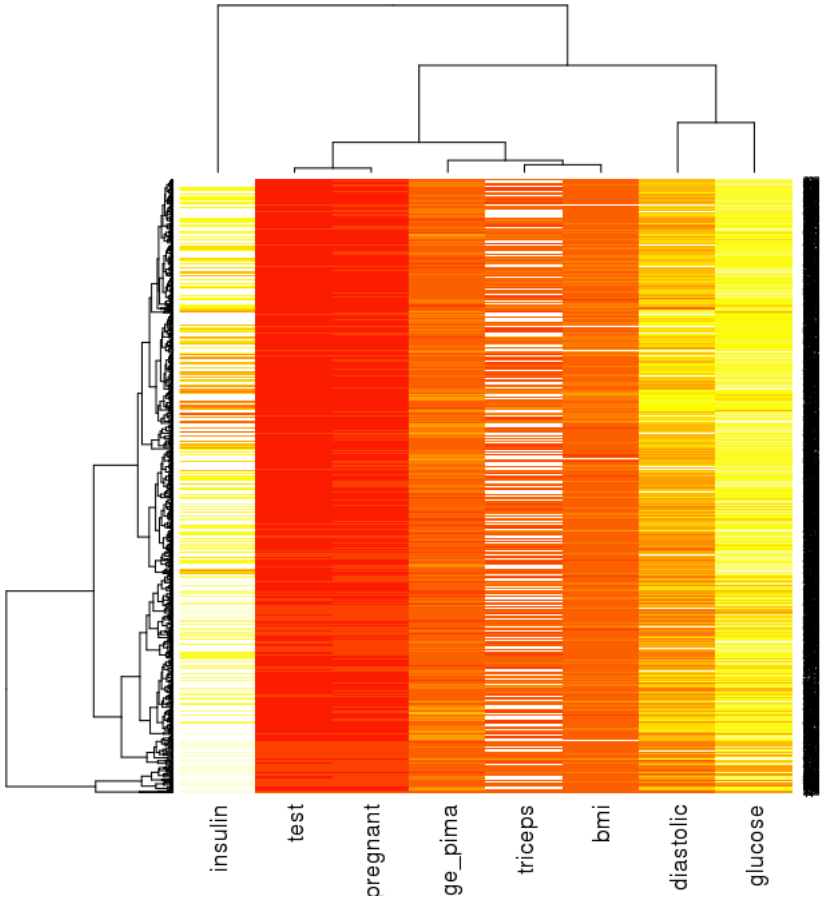
```
> pairs(pima)
```



Graph the Data

- Heatmap

```
> pimaMat <-
    as.matrix(pima[2:9])
> heatmap(pimaMat)
```



Run Exercise 1 in Command Line

- We can also create a file with all our R commands and run it from a Linux Bash Shell

```
source("/biodbA/WorkshopDir/R_BioConductor/EXPORT_BIOCONDUCTOR_DEMO.R")
pima<-data
pima$diastolic[pima$diastolic==0] <-NA
pima$glucose[pima$glucose==0] <-NA
pima$triceps[pima$triceps==0] <-NA
pima$insulin[pima$insulin==0] <-NA
pima$bmi[pima$bmi==0] <-NA
par(mfrow=c(3,2))
hist(pima$diastolic)
plot(density(pima$diastolic, na.rm=TRUE))
plot(sort(pima$diastolic),pch=".")
plot(diastolic ~ bmi, pima)
plot(pima$bmi ~ pima$test.factor)
pairs(pima)
pimaMat <- as.matrix(pima[2:9])
heatmap(pimaMat)
```

  - `R CMD BATCH /biodbA/WorkshopDir/R_BioConductor/pima_plots_for_cluster.R`

- The output will be written to a file called “Rplots.pdf” stored in the directory where you ran the command

Run R Script in Cluster

- We can also run R scripts on a Cluster environment
- We need to create a bash submit script so that the “head node” can schedule the job on a “cluster node”
- The output will be written to a user specified directory
- Log on to cluster

```
$ ssh ibicluster
```

Writing a Submit Shell Script

- What is a submit shell script (wrapper script)?
- Create a shell script as the input
  - All commands which would normally be run from the command line will go inside the wrapper script
  - All parameters (metadata and options) about the job to be run on the cluster will also be included in the submit script
    - Job name
    - Setting working directory
    - Standard output and error file name and location
    - Type of shell to run
    - Job dependencies
- Submit the shell script to the SGE cluster

```
qsub [options] shell_submit_script
```

  - Will queue the job as soon as nodes are available
  - What is the scheduling priority?

Anatomy of Wrapper Script

```
#!/bin/bash  first line defines the shell

# The next section contains SGE flags for your job
   submission. They start with "#$ "
#$ -cwd
#$ -N nameit
#$ -S /bin/bash

# Execute your profile to set the env. variables
. ~/.profile

# List of commands you wish the cluster node to run
R CMD BATCH /biodbA/WorkshopDir/R_BioConductor/
   pima_plots_for_cluster.R
```

Submitting a job to the queue: qsub

`qsub [options] *scriptfile*`

`qsub` : submit a batch job to Sun Grid queuing system.

Options:

`-cwd` : -- Place the output file (.e, .o) in the current working directory. The default is to place them in the users home directory

`-S [shell path]` : Specify the shell to use when running the job script

`-N [name]` : The name of the job

`-e [stderr]` : The name of the file where all stderr should be directed

`-o [stdout]` : The name of the file where all stoud should be directed

`-j [join]` : join stdout and stderr to one file

`-m [email times]` : specify what times email will be sent at

`-M [email ID]` : specify the email the notices should be sent to

`-l h_vmem=[memory size]` : limit the job's total memory use in M or G

`-l mem_free=[memory needed]` : specify the amount of memory

`-pe threaded [num threads]` : tell SGE the job is threaded and how many threads

Scriptfile:

`directory/name` : name of the scriptfile to be run

Example:

```
$ qsub -cwd dateScript.sh
```

Check your job status: qstat

`qstat [options][job_ID]`

`qstat` : is used to request the status of jobs, queues, or a batch server. The request is written to standard out

Options:

- `-u` : -- display jobs for a specific user
- `-j` : -- specify that a full status displays to be written to standard out
- `-N` : -- the name of the job

Examples:

```
$ qstat
$ qstat -u userID
$ qstat -j jobID
```

Submit

- You can submit your script by typing:

```
qsub /biodbA/WorkshopDir/R_BioConductor/submit_pima_R_to_cluster.sh
```

- The job should complete in approximately 10 seconds
- Monitor your job by submitting the “qstat” command
- The output will be written to your home directory as a pdf file.

Exercise 2

A Microarray Example

Exercise Summary

- Microarray expression data from 128 individuals with acute lymphoblastic leukemia has been provided as a comma separated file exprsMat.csv.
- Several covariates such as age, sex, type, stage of the disease etc. have been provided as another comma separated file pData.csv.
- Goals:
  - Read in the expression data and the covariates into R objects.
  - Perform data manipulations such as subsetting to arrive at a smaller expression dataset with samples that we are interested in.
  - Generate MA plots from our subsetted data using the lattice package.

Load Microarray Expression data into an Rsession

- Read the data in

```
> library(ibiRbioCWorkshop2011)
> phenoPath <- system.file( "extdata", "pData.csv", package="
   ibiRbioCWorkshop2011")
> pdOrig <- read.table(phenoPath, check.names = FALSE)
> colnames(pdOrig)
 [1] "cod"            "diagnosis"      "sex"            "age"
 [5] "BT"             "remission"      "CR"             "date.cr"
 [9] "t(4;11)"        "t(9;22)"        "cyto.normal"    "citog"
[13] "mol biol"       "fusion protein" "mdr"            "kinet"
[17] "ccr"            "relapse"        "transplant"     "f.u"
[21] "date last seen"
> expPath <- system.file("extdata", "exprsMat.csv", package="
   ibiRbioCWorkshop2011")
> expOrig <- read.table(expPath, check.names = FALSE)
```

Subsetting Data

- We make use of some of concepts of indexing, subsetting, factors etc. to split our `expOrig` and `pdOrig` variables to narrow down to the samples that we are interested in, specifically, B-cell tumors with molecular biology type "NEG" or "BCR/ABL".

```
> bcell <- grep("^B", as.character(pdOrig
  $BT))
> types <- c("NEG", "BCR/ABL")
> moltyp <- which(as.character(pdOrig
  [["mol biol"]]) %in% types)
> indx <- intersect(bcell, moltyp)
> psubData <- pdOrig[indx,]
> exprsMat <- expOrig[, indx]
```

Recode the factor levels

- The covariate data for some variables, for example BT is represented using a variable of type factor with distinct levels.
- These variables can only take a distinct number of categorical values. The levels can be obtained by using the levels function on the factor variable.
- Operations such as subsetting on a factor variable subsets the factor but leaves the levels of the variable unchanged.
- In this exercise, we shall take a look at the levels of the factor variables that we have just subsetted and attempt to correct this problem.

Recode the factor levels

- Observe the levels for the `mol biol` and `moltyp` variables. Do you notice any problem ?

Recode the factor levels

- Recode the factor levels for the `mol biol` and `moltyp` variables using the `factor` function.

```
> psubData$BT <-factor(psubData$BT)
> levels(psubData$BT)
[1] "B" "B1" "B2" "B3" "B4"
> psubData[["mol biol"]] <- factor
  (psubData[["mol biol"]])
> levels(psubData[["mol biol"]])
[1] "BCR/ABL" "NEG"
```

Compute summary statistics

- Create some summary statistics on the meta data variable `psubData` using the `aggregate` and `xtabs` functions

```
> aggregate( psubData[, "age", drop = FALSE], by= list("sex"= psubData
    $sex,"molBiol"= psubData[["mol biol"]]), FUN = mean,na.rm = TRUE )
 sex molBiol      age
1   F BCR/ABL 39.93750
2   M BCR/ABL 40.50000
3   F     NEG 29.75000
4   M     NEG 24.85714
> aggregate(age ~ sex + `mol biol`, data = psubData, FUN = mean)
  sex mol biol      age
1   F  BCR/ABL 39.93750
2   M  BCR/ABL 40.50000
3   F      NEG 29.75000
4   M      NEG 24.85714
> xtabs( relapse ~ sex + `mol biol` , data = psubData)
   mol biol
sex BCR/ABL NEG
  F       7   3
  M       9  18
```

Data visualization

- Create M-A plots for the first 20 samples in our subsetted expression intensity `data.frame exprsMat`, using the `xyplot` function from the *lattice* package.

```
> library(lattice)
> mat <- exprsMat[,1:20]
> A <- rowMeans(log2(mat))
> M <- log2(unlist(mat)) - A
> Sample <- rep(colnames(mat), each=nrow(mat))
> df <- data.frame(M, A, Sample, row.names=NULL,
  check.names = FALSE)
> plt <- xyplot(M ~ A | Sample, df,
  panel=panel.smoothScatter)
```

Data visualization

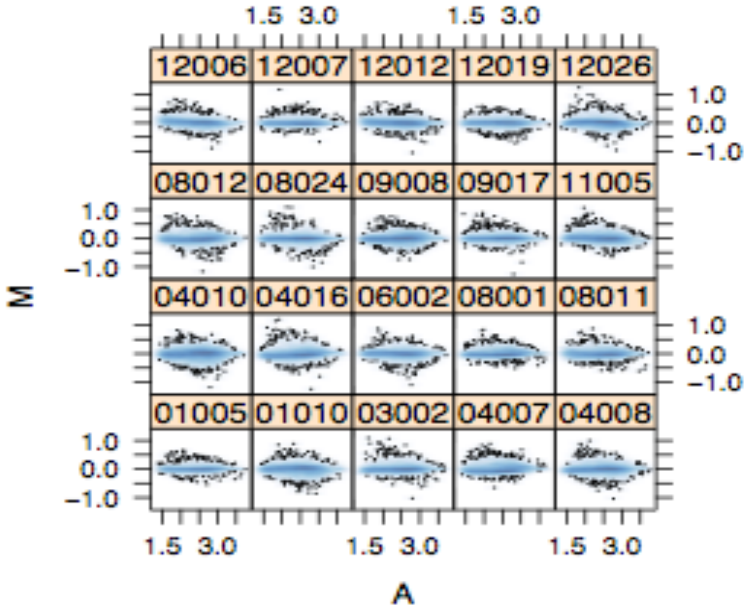


Figure 1: M-A plot for 20 Samples produced using `xyplot`

Contact us

- If you have any questions contact us at:
  - arodrigu1@bsd.uchicago.edu
  - jandrade@bsd.uchicago.edu

- Or visit our website:
  - http://cri.uchicago.edu

Flow Control

Flow Control

- R has the standard set of flow control functions
  - Loops: `for`, `while`, and `repeat`
  - Conditional evaluation: `if` and `switch`
- Can also use the `apply` family of functions
  - `apply`, `sapply`, `lapply`, `mapply`, `eapply`

```
> apply(x, 2, mean)
```

computes the column means of a matrix x

- `apply` is not usually faster than a for loop, but it is more elegant.

References

- Faraway, Julian J. *Linear Models with R*. Boca Raton, FL: Chapman & Hall/CRC, 2005. Print.
- Gopalakrishnan, Nishant. "Introduction to R." Introduction to R and Bioconductor. Fred Hutchinson Cancer Research Center, Seattle, WA. 9-10 Dec. 2010. Lecture.
- Venables, Bill, and David M. Smith. "An Introduction to R." *The Comprehensive R Archive Network*. Ed. R Project. The R Foundation for Statistical Computing. Web. 14 Apr. 2011. <http://cran.r-project.org/doc/manuals/R-intro.html>.

Lists

- An ordered set of elements that can be any type of R object (vectors, other lists, functions, etc.)
- Are useful for grouping related things. Many R functions return lists.
- One-based indexing
- To create list use:

```
> lst<- list(a=1:3, b="ciao", c= sqrt)
> lst
> lst$c(81)
```

Subsetting lists

- Use the `[]` operator to extract a sublist

  ```
  > lst[1]
  ```

- Use the `[[]]` operator to extract a list element

  ```
  > lst[[1]]
  ```

- Use the $ operator to get a named element of the list

  ```
  > lst$a
  ```

- To get a specific value in "a" add the `[]` operator

  ```
  > lst$a[2]
  ```

- Creating subsets with logical expressions is also possible
- Can assign values as well