# Get data from a website (HTML):

con <- url("<http://www.northeastern.com>","r")

x <- readLines(con)

# How to connect to MYSQL:

ucscDb <- dbConnect(MySQL(), user = “genome”, host = “genome.mysql.cse.ucsc.edu”)

result <- dbGetQuery(ucscDb, “show databases;”)

dbDisconnect(ucscDb)

1. You need to install RMySQL package first.

[*http://www.r-bloggers.com/installing-the-rmysql-package-on-windows-7/*](http://www.r-bloggers.com/installing-the-rmysql-package-on-windows-7/)

1. After the installation is completed, before you connect to the database:

Library(DBI)

Library(RMySQL)

1. Connecting and listing databases:

ucscDb <- dbConnect(MySQL(), user="genome", host="genome-mysql.cse.ucsc.edu")

result <- dbGetQuery(ucscDb,"show databases;"); dbDisconnect(ucscDb);

1. Connecting to certain database and listing tables:

hg19 <- dbConnect(MySQL(),user="genome",db="hg19",host="genome-mysql.cse.ucsc.edu")

allTables<-dbListTables(hg19)

1. Get dimensions of a specific table:

dbListFields(hg19,"affyU133Plus2")

dbGetQuery(hg19,"select count(\*) from affyU133Plus2")

## Note: hg19 is a database, “affyU133Plus2” is a table.

1. Read from the table:

affyData <- dbReadTable(hg19, "affyU133Plus2")

## affyData is a data frame.

1. Select a specific subset:

query <- dbSendQuery(hg19, "select \* from affyU133Plus2 where misMatches between 1 and 3")

affyMis <- fetch(query);quantile(affyMis$misMatches);

affyMisSmall <- fetch(query,n=10); dbClearResult(query);

# LOOP FUNCTIONS

Loop functions overview:

1. **Lapply: Loop over a list and evaluate a function on each element.**
2. **Sapply: Same as lapply but try to simplify the result.**
3. **Apply: Apply a function over the margins of an array.**
4. **Tapply: Apply a function over subsets of a vector.**
5. **Mapply: Multivariate version of lapply.**

USAGE OF LAPPLY:

**Simple usage:**

lapply(x,mean) ## x is a list/vector, “mean” is a function.

> x <- 1:4

> lapply(x, runif) # The uniform distribution. runif(n, min = 0, max = 1)

> lapply(x, runif, min = 0, max = 10) # With arguments of runif

**Anonymous function for extracting the first column of each matrix:**

x <- list(a=matrix(1:4,2,2), b=matrix(1:6,3,2))

lapply(x,function(elt) elt[,1])

USAGE OF SAPPLY:

The use of sapply is the same as lapply. Just the result differs.

USAGE OF APPLY:

str(apply)

function (X, MARGIN, FUN, ...)

x <- matrix(rnorm(200), 20,10) # Create a 20 by 10 matrix

apply(x,2,mean)

# Keep the 2nd dimension (10 columns). So the function will collapse the rows (calculate the mean of #each row).

[1] -0.2713273  0.1850763 -0.2436360  0.1181890  0.3120035 -0.1574476 -0.2974936  0.1067881

[9]  0.3212764 -0.2271914

**\*\*\*\*There are some other functions that are shortcuts of apply: [MUCH FASTER]**

rowSums = apply(x,1,sum)

rowMeans = apply(x,1,mean)

colSums = apply(x,2,sum)

colMeans = apply(x,2,mean)

> a <- array(rnorm(2\*2\*10), c(2,2,10))

> apply(a,c(1,2),mean)

            [,1]       [,2]

[1,] -0.06096704 -0.1286878

[2,] -0.16270320  0.1117720

USAGE OF TAPPLY:

Str(tapply)

USAGE OF SPLIT FUNCTION:

The split function can take a vector, working like tapply, but return a list!

We can use the split function in conjunction with lapply/sapply..

**Example:**

library(datasets)

s <- split(airquality, airquality$Month) # The attribute “Month” is treated as a factor.

lapply(s,function(x) colMeans(x[,c("Ozone","Solar.R","Wind")]))

**Splitting on More than One Level:**

> x <- rnorm(10)> f1 <- gl(2,5)

> f2 <- gl(5,2)

> f1

[1] 1 1 1 1 1 2 2 2 2 2

Levels: 1 2

> f2

[1] 1 1 2 2 3 3 4 4 5 5

Levels: 1 2 3 4 5

> interaction(f1,f2)

[1] 1.1 1.1 1.2 1.2 1.3 2.3 2.4 2.4 2.5 2.5

Levels: 1.1 2.1 1.2 2.2 1.3 2.3 1.4 2.4 1.5 2.5

# SUBSETING AND SORTING

**Sample data frame:**

> set.seed(14321)

> X <- data.frame("var1"=sample(1:5),"var2" =sample(6:10),"var3"=sample(11:15))

> X <- X[sample(1:5),]; X$var2[c(1,3)] = NA

**Simple subsetting:**

X[,"var1"]

**Logical subsetting:**

X[X$var1 > 3 & !is.na(X$var2),]

**Dealing with missing values:**

X[which(X$var2 > 8),]

**Sorting:**

> sort(X$var1,decreasing = T)

[1] 5 4 3 2 1

> sort(X$var2,na.last = T)

[1]  6  9 10 NA NA

**Ordering:**

> X[order(X$var1),] # Ordering by the first column

  var1 var2 var3

3    1   NA   14

4    2   NA   15

5    3    9   11

2    4   10   12

1    5    6   13

# You can also sort by multiple variables.

**Ordering with plyr:**

> library(plyr)

> arrange(X,var1) # Arrange X by the order of var1

> arrange(X,desc(var1))

  var1 var2 var3

1    5    6   13

2    4   10   12

3    3    9   11

4    2   NA   15

5    1   NA   14

**Adding rows and columns to a data frame:**

X$var4 <- rnorm(5)

Y <- cbind(X,rnorm(5)) #Another way to do it.

#Rows:

> X

  var1 var2 var3        var4

4    2   NA   15  0.56447020

2    4   10   12 -0.73985890

3    1   NA   14 -0.47021083

5    3    9   11  0.07515538

1    5    6   13 -0.58327483

> Y <- rbind(c(1,2,3,4),X)  #Put the new row in the front

> Y

   var1 var2 var3        var4

1     1    2    3  4.00000000

4     2   NA   15  0.56447020

2     4   10   12 -0.73985890

3     1   NA   14 -0.47021083

5     3    9   11  0.07515538

11    5    6   13 -0.58327483

> Z <- rbind(X,c(1,2,3,4))  #Put the new row at the end

> Z

  var1 var2 var3        var4

4    2   NA   15  0.56447020

2    4   10   12 -0.73985890

3    1   NA   14 -0.47021083

5    3    9   11  0.07515538

1    5    6   13 -0.58327483

6    1    2    3  4.00000000

# SUMMARIZE DATA

**Downloading data from the web:**

> if(!file.exists("./data")) {dir.create("./data")}

> fileUrl <- <https://data.baltimorecity.gov/api/views/k5ry-ef3g/rows.csv?accessType=DOWNLOAD>

> download.file(fileUrl,destfile="./data/restaurants.csv",method="auto")

> restData <- read.csv("./data/restaurants.csv")

**Look at a bit of the data: head & tail**

**Make summary: summary(restData)**

> str(restData)

**Quantiles of quantitative variables:**

> quantile(restData$councilDistrict,na.rm=T)

> quantile(restData$councilDistrict,probs = c(0.5,0.75.0.9)

**Make table:**

table(restData$zipCode,useNA ="ifany")

# Give the count of each value. “ifany” make sure you don’t miss any NAs.

table(restData$councilDistrict,restData$zipCode)

**Check for missing values:**

sum(is.na(restData$councilDistrict))

any(is.na(restData$zipCode)) # To check if any of these values is NA

> all(restData$zipCode > 0)

[1] FALSE

**Rows and column sums:**

colSums(is.na(restData))

all(colSums(is.na(restData)) == 0)

**Values with specific characteristics:**

table(restData$zipCode %in% c("21212","21213"))

# If you try without the “table()”, you get a long vector,

# each indicating each value of zipCode is in that range or not.

restData[restData$zipCode %in% c("21212","21213"),] # Select those rows you want.

**Cross tabs**

> data(UCBAdmissions)

> DF = as.data.frame(UCBAdmissions) # Load the built-in dataset

> xt <- xtabs(Freq~Gender + Admit, data=DF)

**Flat tables:**

> data(warpbreaks)

> warpbreaks$replicate <- rep(1:9, len = 54)

> xt = xtabs(breaks ~., data=warpbreaks) # Break down by all variables

> ftable(xt) # Make it look better than xtabs.

             replicate  1  2  3  4  5  6  7  8  9

wool tension

A    L                 26 30 54 25 70 52 51 26 67

     M                 18 21 29 17 12 18 35 30 36

     H                 36 21 24 18 10 43 28 15 26

B    L                 27 14 29 19 29 31 41 20 44

     M                 42 26 19 16 39 28 21 39 29

     H                 20 21 24 17 13 15 15 16 28

**Size of a dataset**

> object.size(warpbreaks)

3080 bytes

> print(object.size(warpbreaks),units="Mb")

0 Mb

# Create new variables

1. **Creating sequences:**

s1 <- seq(1,10,by=2) # From 1 to 10 increase by 2

s2 <- seq(1,10,length = 3) # From 1 to 10, generating 3 numbers

> x <- c(1,3,76,2,4,12)

> s3 <- se1(along=x); s3 # Create an index vector along x

1. **Creating binary variables:**

restData$zipWrong = ifelse(restData$zipCode < 0, T,F)

1. **Creating categorical variables:**

> restData$zipGroups = cut(restData$zipCode, breaks=quantile(restData$zipCode))

> table(restData$zipGroups)

(-2.123e+04,2.12e+04]  (2.12e+04,2.122e+04] (2.122e+04,2.123e+04]

                  337                   375                   282

(2.123e+04,2.129e+04]

                  332

> table(restData$zipGroups,restData$zipCode) # To see which zipcode fall into which group

# Cutting creates factor

1. **Easier cutting:**

Library(Hmisc)

restData$zipGroups = cut2(restData$zipCode, g = 4)

1. **Creating factor variales:**

restData$zcf <- factor(restData$zipCode)

1. **Levels of factor variables: (Methods to manipulate levels)**

> yesno <- sample(c("yes","no"),size = 10, replace=T)

> yesnofac = factor(yesno)

> relevel(yesnofac,ref="yes") # Relevel the factor, use the “yes” as the main level?

> fac = factor(yesno,levels = c("yes","no")) # Another way is to assign at first.

> as.numeric(fac) # Levels are represented in numbers.

1. **Using the mutate function (Another way to add new column to a data frame)**

|  |
| --- |
| > library(Hmisc)  > library(plyr)  > restData2 = mutate(restData, zipGroups=cut2(zipCode,g=4)) |
|  |
| |  | | --- | |  | |

1. **Common transforms:**

Abs(x)

Sqrt(X)

Ceiling(x), floor(x)

Round(x,digits=n) #round(3.475, digits = 2) is 3.48

Signif(x,digits=n) signif(3.475,digits=2) is 3.5

Cos(x) ..

Log(x)

Log2() , log10(x)

Exp(x)

# RESHAPE DATA

**The goal is tidy data:**

1. Each variable forms a column
2. Each observation forms a row
3. Each table/file stores data about one kind of observation (e.g. People/hospitals)

library(reshape2) #Don’t forget this!!!

**Melting data frames:**

> mtcars$carname <- rownames(mtcars)

> carMelt <- melt(mtcars,id=c("carname","gear","cyl"),measure.vars=c("mpg","hp"))

# carname,gear,cyl together to be the id of each row, and the variables mpg,hp will be count

> head(carMelt)

            carname gear cyl variable value

1         Mazda RX4    4   6      mpg  21.0

2     Mazda RX4 Wag    4   6      mpg  21.0

3        Datsun 710    4   4      mpg  22.8

4    Hornet 4 Drive    3   6      mpg  21.4

5 Hornet Sportabout    3   8      mpg  18.7

6           Valiant    3   6      mpg  18.1

**Casting data frames:**

> cylData <- dcast(carMelt, carname + gear ~ variable) # summarize the data by “carname” and “gear”

**Averaging values:**

> tapply(InsectSprays$count,InsectSprays$spray,sum)

    ## Another way of doing this:

> spIns <- split(InsectSprays$count,InsectSprays$spray)

> sprCount <- lapply(spIns,sum)

> unlist(sprCount)

## Same as:

> sapply(spIns,sum)

# MERGING DATA

**Merging data – merge():**

See the example in ?merge. # A little complicated.

**Using join in the plyr package:**

> df1 = data.frame(id=sample(1:10),x=rnorm(10))

> df2 = data.frame(id=sample(1:10),y=rnorm(10))

> arrange(join(df1,df2),id) # Only workable when they share same names.

Joining by: id

   id          x          y

1   1  0.6387582 -0.4684626

2   2 -0.4549968  0.5238525

3   3 -1.3341411 -0.1630504

4   4 -1.8221198  0.2500387

5   5 -0.2652369 -1.2513703

6   6  0.4457522 -1.1447864

7   7 -0.1056125  1.0974847

8   8  0.6169069 -0.3407762

9   9  1.0611811  0.3624752

10 10  0.4103733 -1.4780403

**Multiple data frames:**

> df3 = data.frame(id=sample(1:10),z=rnorm(10))

> dfList = list(df1,df2,df3)

> join\_all(dfList)

Joining by: id

Joining by: id

   id          x          y          z

1   9  1.0611811  0.3624752  1.1821808

2   3 -1.3341411 -0.1630504 -0.7885041

3   8  0.6169069 -0.3407762 -0.9179093

4   4 -1.8221198  0.2500387  1.0743263

5   6  0.4457522 -1.1447864 -0.6128987

6   5 -0.2652369 -1.2513703  0.6073069

7   2 -0.4549968  0.5238525  1.9913784

8   7 -0.1056125  1.0974847 -0.3459946

9  10  0.4103733 -1.4780403 -1.1809147

10  1  0.6387582 -0.4684626 -2.1031183

# Edit text variables

1. **Fixing character vectors: tolower(),toupper():**

> names(cameraData)

[1] "address"      "direction"    "street"       "crossStreet"  "intersection"

[6] "Location.1"

> tolower(names(cameraData))

[1] "address"      "direction"    "street"       "crossstreet"  "intersection"

[6] "location.1"

1. **Split the string vector: strsplit()**

> splitnames <- strsplit(names(cameraData),"[\\.](file:///\\.)") ## “.” is reserved

> splitnames

[[1]]

[1] "address"

[[2]]

[1] "direction"

[[3]]

[1] "street"

[[4]]

[1] "crossStreet"

[[5]]

[1] "intersection"

[[6]]

[1] "Location" "1"

1. **Substitution:**

> names(cameraData)

[1] "address"      "direction"    "street"       "crossStreet"  "intersection"

[6] "Location.1"

> sub("[\\.](file:///\\.)","",names(cameraData)) #sub() will only replace the first matching character.

[1] "address"      "direction"    "street"       "crossStreet"  "intersection"

[6] "Location1"

## Replace all the matching characters :

> testName <- "this\_is\_a\_test"

> sub("\_","",testName)

[1] "thisis\_a\_test"

> gsub("\_","",testName)

[1] "thisisatest"

1. **Finding values (Pattern matching):**

> **grep**("Alameda",cameraData$intersection)

[1]  4  5 36 #This pattern appears in the 4th,5th,36th record.

> **grepl**("Alameda",cameraData$intersection)

# This will return a vector indicating T/F : whether this pattern appears in this record.

> table(grepl("Alameda",cameraData$intersection))

FALSE  TRUE

   77     3

> **grep**("Alameda",cameraData$intersection, value = T)

## instead of showing the index of appearance, it shows the content.

[1] "The Alameda  & 33rd St"   "E 33rd  & The Alameda"

[3] "Harford \n & The Alameda"

1. **Useful string operations: (Don’t need to library(stringr)**

> nchar("asdfasdfsadf")

[1] 12

> substr("asdfasdfasdf",1,3)

[1] "asd"

> paste("Yaxin"," Huang") # default sep = “ “

[1] "Yaxin  Huang"

   > **paste0**("Yaxin","Huang")

   [1] "YaxinHuang"

> str\_trim("Yaxin    ")

Error: could not find function "str\_trim"

**> library(stringr)**

> str\_trim("Yaxin    ")

[1] "Yaxin"

# REGULAR EXPRESSIONS

## Metacharacters

1. **Match the start of a line:**

> grep("**^**I think",s) # s is a vector of strings. Will not match the middle

[1] 1 3 5l

1. **Match the end of a line:**

grep("morning$",s)

1. **List a set of characters we will accept at a given point in the match:**

> s <- c("name the worst thing about Bush!","ush")

> grep("[Bb][Uu][Ss][Hh]",s,value=T)

# This will match “bush” in lower or upper case at any point of the string.

# But if you only have words like “ush” in your string, it won’t match.

[1] "name the worst thing about Bush!"

Range of characters/numbers: [a-z] [a-zA-Z] [0-9]

1. **Match character NOT in the indicated class:**

> s <- c("i like basketballs", "dont worry... we all die anyway!", "why?","Lets go.")

> grep("[^?.]$",s) #”?” and “.” NOT at the end.

[1] 1 2

1. **“.” Is used to refer to any character:**

> s <-c("9111","its stupid the post 9-11 rules.","203.169.114.66","9no11")

> grep("9.11",s)

[1] 1 2 3 # Notice it doesn’t match the last one!!

1. **“|” means can match either:**

> grep("flood|fire|earthquake",s)

Complicated one:

^[Gg]ood|[Bb]ad

Match “Good” or “good” at the beginning and “Bad” or “bad” at any point.

^([Gg]ood|[Bb]ad)

Match good or bad AT THE BEGINNING

1. **Question mark indicates the expression is optional:**

[Gg]eorge**( [Ww]\.)?** [Bb]ush    # Please Escape the dot.

It can match George W. Bush but George bush can also be matched.

1. \* and + : same as common sense.

(.\*) will match any thing (even no string) that surrounded by parenthesizes.

1. **{} are referred to as interval quantifiers; Specify the MAX and MIN  number of occurrence of an expression.**

> s <- c("Bush has historically won all major debates he's done.","in my view, Bus doen'st need these debates..", "bush does not need the debates?", "That's what Bush suporters are doing about the fierce great debate.")

> grep("[Bb]ush( +[^ ]+ +){1,5} debate",s)

integer(0)

> s[2] <- "in my view, bush doesn't need these debates..."

> grep("[Bb]ush( +[^ ]+ +){1,5} debate",s)

integer(0)

# The course material is wrong at this point. This exp will match nothing.

# See the change:

> grep("[Bb]ush( +[^ ]+ **\***){1,5} debate",s) #Not right but it avoid needing two spaces between two words.

[1] 1 2 3

> news <- c("Bush  has  historically  won  all  major  debates  he's  done.","in my view, bush does not need debates.")

> grep("[Bb]ush( +[^ ]+ +){1,5} debate",news)

[1] 1  # news[1] has double spaces.

{m,n} min m max n

{m} exactly m

{m,} at least m

1. **\* is greedy so it always matches the longest possible string.**

^s(.\*)s  will match “sitting at starbucks” (whole thing)

# DEALING WITH DATES

**Ordinary String:**

> d <- date()

> d

[1] "Thu Jul 24 14:31:32 2014"

# Shows the date and time now.

> class(d)

[1] "character"

**Date Class:**

> d2 <- Sys.Date(); d2

[1] "2014-07-24"

> class(d2)

[1] **"Date"**

**Formatting Dates:**

> format(d,"%a %b %d")

Error in format.default(d, "%a %b %d") : invalid 'trim' argument

> format(d2,"%a %b %d")

[1] "Thu Jul 24"

Notes:

%d = day as number

%a = abbreviated weekdays

%A = unabbreviated weekdays

%m = month (00-12)

%b = abbreviated month

%y = 2 digit year

%Y = 4 digit year

**Create dates:**

> d

[1] "Thu Jul 24 14:31:32 2014"

> as.Date(d,"%a %b %d %H:%M:%S %Y") # d can be a vectory (actually it is now a vector!)

[1] "2014-07-24"

**Calculate date difference:**

> x <- c("1jan1960","2feb1990")

> z <- as.Date(x,"%d%b%Y"); z

[1] "1960-01-01" "1990-02-02"

> z[1] - z[2]

Time difference of -10990 days

> as.numeric(z[1]-z[2])

[1] -10990

**Convert to Julian:**

> d2

[1] "2014-07-24"

> weekdays(d2)

[1] "Thursday"

> months(d2)

[1] "July"

> julian(d2)

[1] 16275

attr(,"origin") # Days count starting from 1970/01/01

[1] "1970-01-01"

**Pakcage lubridate:**

> library(lubridate)

> ymd("20140108")

[1] "2014-01-08 UTC"

> mdy("01082014")

[1] "2014-01-08 UTC"

> dmy("08012014")

[1] "2014-01-08 UTC"

**> mdy("01/08-2014")**

[1] "2014-01-08 UTC"

**Dealing with times (also package lubridate):**

> ymd\_hms("2011/08-03 10-15:03")

[1] "2011-08-03 10:15:03 UTC"

> ymd\_hms("2011/08-03 10-15:03", tz = "Pacific/Auckland")

[1] "2011-08-03 10:15:03 **NZST**"

## Check : ?Sys.timezone