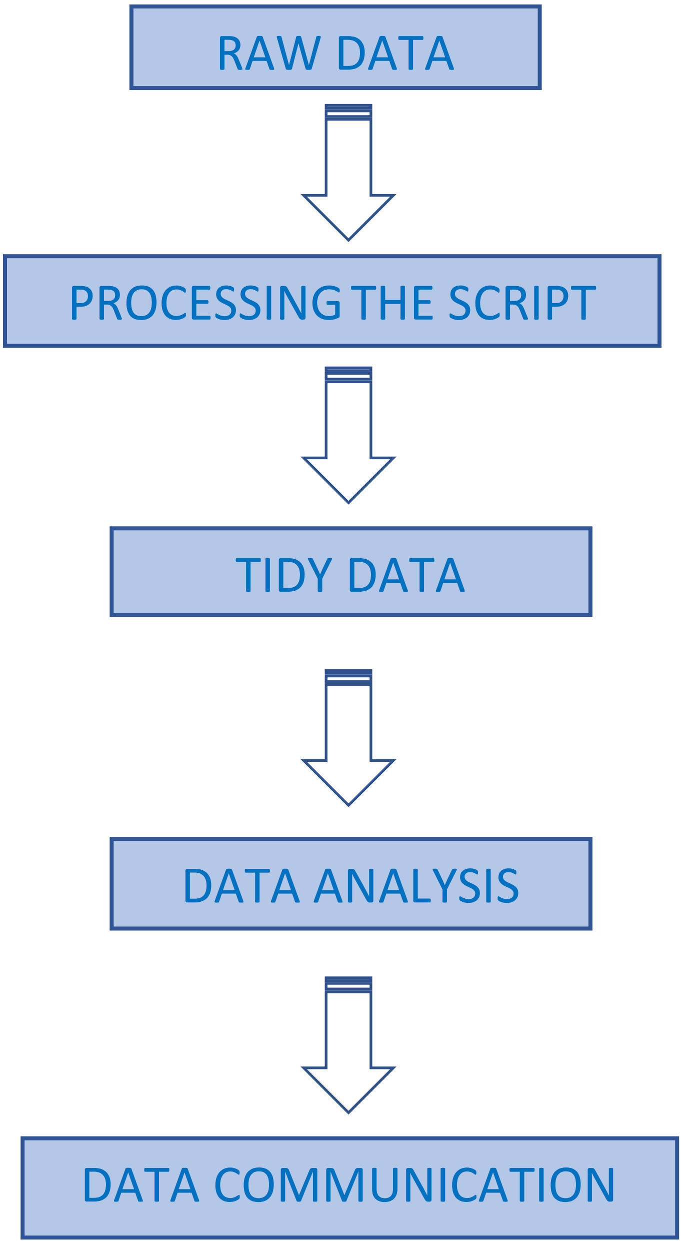
GETTING AND CLEANING DATA



RAW DATA VERSUS PROCESSED DATA

RAW DATA

- The original source of data.
- Often hard to use for data analysis.
- Data analysis requires processing of raw data.
- Raw data is the untouched data i.e. the data has not been manipulated.

PROCESSED DATA

- Data that is ready for analysis.
- Processing can include processing and sub-setting.
- There may be standards for processing.
- All steps should be recorded.

TIDY DATA

Four things that are required are :

- Raw Data
- A tidy dataset.
- Codebook describing each variable and values.
- Exact recipe you used to go from State 1 to state ‘X’ via ‘n’ states.

THE RAW DATA

- The strange binary file your machine (measurement machine) spits out is called the raw data.
- The unformatted excel sheet with more than one worksheet.
- The complicated JSON data you get from scrapping the web application.
- The hand entered numbers collected while looking through the microscope.

RAW DATA IS IN RIGHT FORMAT IF

i) Ran no software on data.
ii) Did not manipulated any of the numbers or alphabets in the data.
iii) Did not removed any data from the dataset.
iv) Did not summarise data in way.

THE TIDY DATA

- Each variable you measure should be in one column.
- Each different observation of that variable should be in a different row.

- Thee should be one row for each kind of variable.
- If you have multiple tables, they should include a column in the table that allows them to be linked.

=x=x=x=x=x=

THE INSTRUCTION LIST

- Ideally write a computer a computer script in R.
- The input for the script is the raw data.
- The output is the processed, tidy data.
- There are no parameters to the script.

In some cases, it will not be possible to script every step, in that case, instruction should be given as:

I) Take the raw file, run version 3.1.2 of summarise software with parameters a=1, b=2, c=3.

II) Run the software separately for each sample.

III) Take column 03 of outputfile.out for each sample and that is the corresponding row in the output dataset.

=x=x=x=x=x=

DOWNLOADING FILES

- A basic component of working with data is knowing your working directory.
- The two main components are getwd() and setwd().
- Be aware of relative and absolute path.
    a. Relative: setwd(" ../")
    b. Absolute: setwd("/users/REDBull/Documents/")

In windows, we have to use "\"(back-slash) instead of "/" to set the path of working directory.

CHECKING AND CREATING DIRECTORY

- File.exists(“dir_name”) will check if there is a directory on the local PC.
- File.crete(“dir_name”) will create a directory with the specified name in the locl PC.

```
If(! File.exists(“data”))

{

            dir.create(“data”)

}
```

GETTING DATA FROM THE INTERNET

I. DOWNLOAD.FILE()

- Downloads a file from the internet.
- Even user can do this(download) by UI, it helps with reproducibility.
- Important parameters are url, dest_file, method.
- Useful for downloading ‘tab-delimited’, ‘csv’, etc. types of files.

GETTING DATA FROM THE LOCAL DRIVES

- read.table()
- Use sep = ‘,’ for comma separated values.
- Use sep = ‘\t’ for tab separated values.
- CSV files can also be read by read.csv() without using the ‘sep’ attribute.

WRITING TO EXCEL FILE

Use write.xlsx() for writing data into an excel file.

READING XML FILE

- Frequently used for storing structured data.
- Extracting XML is the basis for most web scrapping.
- Components :
    a. Markup: labels that give text structure.
    b. Content: the actual text document.

```
x <- read_xml("<bar>123</bar>")
xml_name(x)

y <- read_xml("<bar><baz>1</baz>abc<foo /></bar>")
z <- xml_children(y)
xml_name(xml_children(y))
```

```
file_url <- "http"//www.x………………..."
doc <- xmlTreeParse(File_url, useInternal = TRUE)
rootNode <- xmlRoot(doc)
xmlName(rootNode)
```

xPathSApply

```
xpathSApply(rootNode, "//Name", xmlValue)
xpathSApply(rootNode, "//Price", xmlValue)
```

EXTRACT CONTENT BY ATTRIBUTE

```
fileURL <- "https://....................."
doc <- xmlTreeParse(doc,//[@class = 'score'], xmlValue)
scores
```

[1] "49-27" "14-6" "30-9" "23-20" "26-23"

READING JSON FILES

```
Library(jsonlite)
Jsondata <- fromJSON("…………..")
Names(jsondata)
```

Data.table

- Inherits from data.frame
  -All functions that accepts data.frame also works fine with data.table.
- Written in C, so it is much faster.
- Even more faster at subsetting group and updating.

SPECIAL VARIABLES

.N

- An integer
- Length is 1

```
Library(data.table)
DF<- data.frame( x= rnorm(9), y= rep(c(“a”,”b”,”c”), each =
3, z= rnorm(9)
Head(DF,3)

 X      Y      Z

0.122   a      -0.025

0.085   a      0.137

1.058   a      2.164
```

- Containing the number.

MULTIPLE OPERATIONS

DT [, m: = {tmp <- x+z ; log2(tmp+5) }]

PLYR LIKE OPERATION

DT[, a : = x>0]
DT[, b : = mean(x+w), by = a]

JOINS IN DATA TABLE

```
DT1 <- data.table(X = c('a', 'b', 'c') , DT, Y = 1:4)
Dt2 <- data.table(X = c('a', 'b', DT1), Y = 5:7)

Setkey <- (DT1, X) ; setKey(DT2, X)
Merge(DT1, DT2)
```

	X	Y	Z
1	A	1	5
2	A	2	5
3	B	3	6

FAST READING

```
Big_df <- data.frame( X = rnorm(1E6, Y =
rnorm(1E7))
File <- tempfile()
Write.table(big_df, file = file, row.names = F, col.names = T, sep = '\t',
quote = F)
System.time(fread(file))
```

DPLYR PACKAGE

- Used for manipulating tabular data.

  - My_df <- read.csv(“path2csv”, stringsASFactor = F)
  - Dim(my_df)
  - Cran <- tbl_df(my_df)
- It has 5 verbs :
  - Select()
  - Filter()
  - Arrange()
  - Mutate()
  - Summarise()/summarize()

SELECT()

- Starts_with, ends_with, contains
- Matches()
- Num_range()
- One_of()
- Everything()
- Group_cols()

```
Select(cran, r_arch : country)
Select(cran, country : r_arch)
Select(cran, -time)
```

FILTER()

```
Filter(cran, !is.na(r_version))
```

ARRANGE()

```
Cran2 <- select(cran, size : ip_id)
Arrange(cran2,ip_id)
Arrange(cran2, desc ip_id)
```

MUTATE()

```
Cran3 <- select(cran2, size:Ip_id)
Mutate(crans3, size_mb = size/2^20)
```

SUMMARIZE()

```
Summarize(cran, avg_bytes = mean(size))
```

=x=x=x=x=x=

CONNECTING AND LISTING DATABASES

```
Ucscdb <- dbConnect(Mysql(), uses = “genome”, host = “genome-mysql-cse.ucsc.edu”)
Result <- dbGetQuery(ucscdb, “Show databases;”);  dbDisconnect(ucscDb)

Hg19 <- dbConnect(MySQL(),user=“genome”,db=“hg19”, host=genome-mysql-ucsc.org.edu”)

AllTables <- dbListTables(hg19)
Length(allTables)
```

READ FROM THE TABLE

```
affyData <- dbReadTable(hg19,”affyU133Plus2”
head(affyData)
```

SELECT A SPECIFIC SUBSET

```
Query<-dbSendQuery(hg19,“SELECT * FROM affyU133Plus2 WHERE mismatches between 1 and 3”)
affMis <- fetch(query) ; quantile(affyMis$mismatches)
affyMissSmall <- fetch(query, n=w) ; dbCleanResult()

dim(affMissSmall)
dbDisconnect(hg19)
```

READING FROM HDF5

Install R HDF5 Package

```
Source(http://bioconductor.org/biocLite.R)
biocLite(“rhdf5”)

library(rhdf5)
created <- h5CreateFile(“example.h5”)
created

[1] TRUE
```

CREATE GROUP IN HDF5

```
Created <- h5CreateGroup(“example.h5”, “foo”)
Created <- h5CreateGroup(“example.h5”, “bar”)
Created <- h5CreateGroup(“example.h5”, “foo/baa”)
H5ls(“example.h5”)
```

WRITE TO GROUPS

```
A = matrix(1:10, 5, 2)
H5CreateGroup(A, “example.h5”, “foo/A”)
B = array(seq(0.1, 2.0, by = 0.1), dim = c(5,2,2))
Attr(B, “scale”) <- “Liter”
H5write(B, “example.h5”, “foo/baaa/B”)
H5ls(“example.h5”)
```

WRITE A DATASET

```
Df<- data.frame(1L:5L, seq(0,1,length.out = 5), c(“a”,”b”,”cde”,”fghi”,”a”,”s”),
stringsAsFactor = F)
H5write(df, “example.h5”,”df”)
H5ls(“example.h5”)
```

READING DATA

```
readA <- h5read(“example.h5”,”foo/A”)
readB <- h5read(“example.h5”, “foo/baaa/B”)
readDF <- h5read(“example.h5”, df)
```

WRITING AND READING CHUNKS

```
H5write(c(12,13,14), “example.h5” , “foo/A”, index = List(1:3,1))
H5read(“example.h5”, “foo/A”)
```

READING DATA FROM WEB

```
Con <- url(http://scholar.google.com/....... user = HI-1600AAAA L=en)
Htmlcode <- readLines(con)
Close(con)
Htmlcode
```

[1] “<1DOCTYPE <html> …………………………………………….</html>

=x=x=x=x=x=

SUMMARIZING DATA

```
Head(restData, n = 3)
Tail(restData, n = 3)
Summary(restData)
Str(restData)
Quantile(restData)
Quantile(restData$councilDistrict, probs = c(0.5, 0.75, 0.9))
Table(restData$zipCode, useNA = 'If any')
Sum(is.na(restData$councilDistrict))
[1] 0

Any(is.na(restData$councilDistrict))
[1] FALSE

colSums(is.na(restData))
all(colSums(is.na(restData))==0)
table(restData$zipcode %in% c(21212))

               or

table(restData$zipcode == c(21212))
```

CROSS TABS

```
xt <- xtabs(Freq ~ Gender + Admit, data =df)
```

FLAT TABLES

```
Ftables(xt)
Object.size(faketable)
```

CREATING NEW VARIABLES

```
S1 <- seq(1, 10, by =2) ;s1
S2 <- seq(1,10, length = 3) ; s2
```

```
X <- c(1,3,8,25,100)
```

SUBSETTING VARIABLES

```
restData$nearMe <- restData$neighbour %in% c(“Roland Park”,”Homeland”)
table(restData$nearMe)
```

CREATING BINARY VARIABLE

```
restData$zipWrong <- ifelse(restData$zipcode < 0, TRUE, FALSE)
table(restData$zipcode < 0, restData$zipWrong)
```

CREATING CATEGORICAL VARIABLE

```
restData$zipGroup <-cutrestData$zipCode,breaks =quantile(restData$zipCode))
table(restData$zipCode)
```

CREATING FACTOR VARIABLE

```
restData$zcf <- factor(restData$zipCode)
class(restData$zcf)

yesno <- sample(c( “YES”, “NO”), size = 10, replace = TRUE)

yesnoFac <- factor(yesno, Levels = (“YES”, “NO”)
relevel(yesnoFac, ref = “YES”)
```

COMMON TRANSFORMS

```
Abs(x)

Sqrt(x)

Ceiling(x)

Floor(x)

Round(x, digit = n)

Cos(x)

Sin(x)

Log2(x)

Log10(x)
```

```
Exp(x)
```

RESHAPING DATA

```
Library(reshape2)
Head(mtcars)

Mtcars$carname <- rownames(mtcars)
Carmelt <- melt(mtcars, id = c("carname", "gear", "Cyl"), measure.vars =
c("mpg", "hp"))

CylData <- dcast(carmelt, Cyl ~ Variable)
CylData <- dcast(carmelt, Cyl ~ variable,mon)
Cyldata
Tpply(InsectSpray$Count, InsectSprays$spray, sum)
SpIns <- split(InsectSpray$count, InsectSprays$spray)
spIns
```

or

```
sprCount <- lapply(spins,sum)
sprCount
```

or

```
ddply(InsectSpray, .(spray), summarize, sum = sum(count))
```

MANAGING DATAFRAMES WITH DPLYR

DPLYR VERBS :

i. Select
ii. Filter
iii. Arrange
iv. Rename
v. Mutate
vi. Summarize/summarise

```
Library(dplyr)
Options(width = 105)

Chicago <- readRDS("Chicago.rds")
Dim(Chicago)

Str(Chicago)

Names(Chicago)
```

```
Head(select(Chicago, city:dptp))

Head(select(Chicago, -(city:dptp))

I <-  match(“city”, names(Chicago))

J <- match(“dptp”, names(Chicago))

Head(Chicago [ , -(i:j)]

Chic.f <- filter(Chicago, pm25team2 > 30)

Head(chic.f, 10)



Chic.f <- filter(Chicago, pm25team2>30 & tmpd > 80)
Head(chic,f, 10)

Chicago <- arrange(Chicago, desc(date))
Chicago  <- rename(Chicago,pm25 = pm25tmean2, decopoint = dptp)

Chicago <- mutate(Chicago , pm25detrend = pm25 – mean(pm25, na.rm = TRUE))

Head(select(Chicago, pm25, pm25detrend))

Chicago <- mutate(Chicago, tempcat = factor(1 *(temp>80), labels =
c(“COLD”, “HOT”)))

Hot_cold <- group_by(Chicago, tempcat)

Summarize(hot_cold, pm25 = mean(pm25), O3 = max(O3tmean2), NO2 =
median(no2mean2))
```

=x=x=x=x=x=

MERGING DATA

- Merges Dataframes.
- Important parameters:
  - X
  - Y
  - By
  - By.x
  - By.y
  - All

```
mergedData <- merge(reviews, solution, by.x = “solution_id”, by.y = “id”,
all = TRUE)

intersect(names(solution), names(reviews))

mergedData2 <-  merge(reviews, solution, all =TRUE)
head(mergedData)
head(mergedData2)
```

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)
```

If you have multiple dataframes:

```
Df1 <- data.frame(id = sample(1:10), x = rnorm(10)
Df2 <- data.frame(id = sample(1:10), y = rnorm(10)
Df3 <- data.frame(id = sample(1:10), y = rnorm(10)

Df_list = list(df1, df2, df3)
Arrange(join_all(df_list))
```

=x=x=x=x=x=

EDITING TEXT VARIABLES

FIXING CHARACTER VARIABLES

-strsplit()

- Good for automatically splitting variable names
- Important parameters: x, split

```
Splitnames <- strsplit(names(cameraData), "\\.")
```

- This command will split the names of column according to the(".") operator.

QUICK ASIDE – LISTS

```
Mylist <- list(letters = c(“a”,”b”, “c”), numbers = 1:3, matrix(1:25, ncol
= 5))

Head(mylist)

Splitnames[[6]][1]

firstElement <- function(x) { x[1] }

sapply(splitnames, firstelement)
```

FIXING CHARACTER VECTORS

-sub()

- It substitutes in place of given value with the passed value.
- Important parameters: (pattern, replacement, x)

```
Names(reviews)
Sub(“_”,””,names(reviews))
```

Sub() command only replaces the first occurrence of given pattern. If you want to remove all the occurance of given pattern, use gsub() command.

```
Testname <- this_is_a_test
Sub(“_”,””,testname)
[1] thisis_a_test

Gsub(“_”,””,testname)
[1] thisisatest
```

FINDING VALUES – GREP() AND GREPL()

```
Grep(“Alameda”, cameraData$intersection)
[1] 4 5 36
```

The above output tells that the word’Almeda’ appeared in 4th, 5th and 36th line

```
Table(grepl(“Alameda”,cameraData$intersection)

       FALSE TRUE
       77     3

cameraData2<-cameraData[!grepl(“Alameda”, cameraData$intersection),]
grep(“Alameda”, cameraData$interscetion, value = TRUE)
```

MORE USEFUL STRING FUNCTION

```
Library(stringr)
Nchar(“Prem Prakash”)

[1] 12

Substr(“Prem Prakash”, 1,7)
[1] “Prem”

Paste(“Prem”,”Prakash”,”Illa”)
[1]”Prem Prakash Illa”

Paste0(“Prem”,”Prakash”,”Illa”)
[1]” PremPrakashIlla

Strtrim(“Prem       “)
[1] Prem
```

=x=x=x=x=x=

REGULAR EXPRESSION

METACHARACTER

^ - REPRESENTS THE START OF THE LINE
$ - REPRESENTS THE END OF THR LINE.

[Bb] will check for either B or b in the text.

[Cc][Oo][Pp]

will check for COP, COp, CoP, Cop, cOP, cOp, coP,cop.

- We can specify a range of letters in []. To check for character in a-z, we will write [a-z].
- To check for A-Z, we will write [A-Z].

The period operator(.) is used to represent any character.

- 9.11 will check for :
    - 9_11
    - 9/11
    - 9-11
    - 9;11
    - 9:11
    - Etc.

The pipe operator(|) is used to represent or. A or B is equivalent to A|B in Regular Expressions.

'+' and '*'

'+' : Repeat at least one time.
'*' : Repeat any number of time including none.

{ } : These are known as interval quantifiers.

- {m,n} means minimum 'm' and maximum 'n' no of times.
- {m} means exactly 'm' no of times.
- {m,} means minimum m number of times with no maximum limit.

Revisited :

```
\1 : Means repeated 1 time.
\2 : Means repeated 2 times.
\3 : Means repeated 3 times.
```

```
Ex -   night is here.            ([Nn]ight +\1)
       And and than.             ([Aa]nd +\2)
       So so so cold.            ([Ss]o +\3)
```

WORKING WITH DATES

```
d1 <- date()
d1

[1] “Sun Jan 12 17:48:00 2014”

Class(d1)

[1] “character”

D2 <- sys.date()
D2

[1]2019-12-3

Class(d2)

[1] “Date”
```

FORMATTING DATE

%d	:	day as no(1-31)
%a	:	abbreviated weekday
%A	:	unabbreviated weekday
%m	:	month(00-12)
%b	:	abbreviated month
%B	:	unabbreviated month
%y	:	2-digit year
%Y	:	4-digit year

```
Format(d2, “%a%b%d”)

[1] “Sun Jan 12”
```

CREATING DAYS

```
X <- c(“1Jan1960”, “2Jan1960”, “ 31Mar1960”, “30Jul1960”)
Z <- as.Date(x, “%Y%m%d)

[1] 1960-01-01       1960-01-02    1960-03-31    1960-07-30

Z[1] – z[2]
[1] -1
```

CONVERTING TO JULIAN

```
WEEKDAYS(D2)
[1] "Sunday"

Months(d2)
[1] "January"
Julian(d2)

[1] 16082
Attr( , "origin")

[1] "1970-0101"
```

THE LUBRIDATE LIBRARY

```
Library(libridate)
Ymd(20191204)

[1] 2019-12-04

Mdy(08/04/2013)

[1] 2013-08-04
```

DEALING WITH DATE

```
Ymd_hms("2014-08-03 10:15:03")

[1] "2014-08-03 10:15:03 UTC"

Ymd_hms("2019-12-04 10:15:03, tz = "Pacific")

[1] "2019-12-04 10:15:03 NZST"
```

=x=x=x=x=x=