Coursera: Getting and Cleaning Data

Contents

[About this course 2](#_Toc417058555)

[Raw and processed data 2](#_Toc417058556)

[Tidy data 2](#_Toc417058557)

[The four things you should have 2](#_Toc417058558)

[The raw data 2](#_Toc417058559)

[The tidy data 3](#_Toc417058560)

[The code book 3](#_Toc417058561)

[The instruction list 3](#_Toc417058562)

[Files and Directories 3](#_Toc417058563)

[Loading flat files 4](#_Toc417058564)

[Excel files 4](#_Toc417058565)

[XML files 5](#_Toc417058566)

[JSON files 6](#_Toc417058567)

[data.table 6](#_Toc417058568)

[Reading from mySQL 7](#_Toc417058569)

[Reading from HDF5 8](#_Toc417058570)

[Getting from the Web 9](#_Toc417058571)

[Getting Data from APIs 10](#_Toc417058572)

[Reading from other sources 10](#_Toc417058573)

[There is a package for that 10](#_Toc417058574)

[Subsetting and sorting 11](#_Toc417058575)

[Summarizing data 11](#_Toc417058576)

[Create New Variables 13](#_Toc417058577)

[Why create new variables 13](#_Toc417058578)

[Reshaping data 14](#_Toc417058579)

[Managing Data Frames with dplyr 15](#_Toc417058580)

[dplyr 15](#_Toc417058581)

[dplyr Verbs 15](#_Toc417058582)

[dplyr Properties 15](#_Toc417058583)

[Merging 17](#_Toc417058584)

# About this course

* Basic ideas behind getting data ready for analysis:
  + Finding and extracting raw data
  + Tidy data principles and how to make data tiny
  + Practical implementation through range of R packages
* Goal of this course
  + Raw data
  + -> processing script
  + -> tidy data
  + -> data analysis
  + -> data communication

## Raw and processed data

* Definition of data: “Data are values of qualitative or quantitative variables belong to a set of item”
* Raw data:
  + Original source of data
  + Often hard to use for data analyses
  + Data analysis *includes* processing
  + Raw data may only need to be processed once
* Processed data
  + Data that is ready for analysis
  + Processing can include merging, subsetting, transforming, etc
  + There may be standards for processing
  + All steps should be recorded **-> very important**

# Tidy data

## The four things you should have

1. The raw data
2. A tidy data set
3. A code book describing each variable and its values in the tidy data set
4. An explicit and exact recipe you used to go from 1 -> 2,3

## The raw data

* Examples:
  + strange binary file your measurement machine spits out
  + unformatted Excel file with 10 worksheets the company sent you
  + complicated JSON data from scraping Twitter API
  + hand-entered numbers collected
* Know it’s in right format (unadulterated) if:
  + Ran no software on the data
  + Did not manipulate any numbers in the data
  + Did not remove any data from the data set
  + Did not summarize the data in any way

## The tidy data

1. Each variable you measure should be in in one column
2. Each different observation of that variable should be in a different row
3. There should be one table for each “kind” of variable
4. If you have multiple tables, they should include a column in the table that allows them to be linked
5. Some other important tips:
   1. Include row at the top of each file with variable names
   2. Make variable names human readable AgeAtDiagnosis instead of AgeDx
   3. In general data should be saved in one file per table

## The code book

1. Information about the variables (including units!) in the data set not contained in the tidy data
2. Information about the summary choices you made
3. Information about the experimental study design you used
4. Some other important tips
   1. A common format for this document is a word/text file
   2. There should be a section called “Study design” that has a thorough description of how you collected the data
   3. There must be a section called “Code book” that describes each variable and its units

## The instruction list

* Ideally a computer script (in R .. or Python)
* 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 you should provide instructions like:
  + Step 1- take the raw file, run version 3.1.2. summarize software with parameters a=1, b=2, c=3
  + Step 2 - run the software separately for each sample
  + Step 3 - take column three of outputfile.out for each sample and that is the corresponding row in the output data set

# Files and Directories

|  |  |
| --- | --- |
| Command | Description |
| file.exists(“directoryName”) | checks to see if directory exists |
| dir.create(“directoryName”) | will create a directory (if it doesn’t exist, returns error) |
| if (!file.exists(“data”)) { dir.create(“data”)  } | creates a directory if it doesn’t already exist by first checking |
| setInternet2(use=TRUE) | use this before downloading https files |
| fileURL <- "https://data.baltimorecity.gov/api/views/dz54-2aru/rows.csv?accessType=DOWNLOAD" | save file URL to variable |
| download.file(fileURL,destfile="cameras.csv") | download file to destination. important parameters are url, destfile, method |
|  |  |

# Loading flat files

|  |  |
| --- | --- |
| Command | Description |
| read.table(“./cameras.csv”, sep = “,”, header=TRUE) | read data from table with comma delimeter and header  this is main function for reading data into R, important parameters are file, header, sep, row.names, nrows |
| quote | you can tell R whether there are any quoted values quote =”” means no quote |
| na.strings | set the character that represents missing value |
| nrows | how many rows to read of the file (e.g. nrows=10 reads 10 lines) |
| skip | number of lines to skip before starting to read |
|  |  |

# Excel files

Still probably most widely used format for sharing data

|  |  |
| --- | --- |
| Command | Description |
| install.packages("xlsx")  library(xlsx) | library for connecting to excel files |
| fileURL <- "https://data.baltimorecity.gov/api/views/dz54-2aru/rows.xlsx?accessType=DOWNLOAD"  download.file(fileURL,destfile = "./cameras.xlsx", mode = "wb") | download excel file. use mode = “wb” because binary file. |
| cameraData <- read.xlsx("cameras.xlsx",sheetIndex=1,header=TRUE) | load excel data to table object with xlsx package |
| setInternet2(use=TRUE) | use this before downloading https files |
| colIndex <- 2:3  rowIndex <- 1:4  cameraDataSubset <- read.xlsx("cameras.xlsx",sheetIndex=1, colIndex = colIndex, rowIndex = rowIndex) | read subset of excel data based on row and column index |
| write.xlsx() | write out to Excel |
| read.xlsx2 | faster than read.xlsx but for reading subsets of rows may be unstable |
|  |  |

# XML files

|  |  |
| --- | --- |
| Command | Description |
| install.packages("XML")  library(XML) | library for connecting to XML files |
| xmlFile <- "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi ?db=pubmed&id= 25886956,23797912,25870761,25849898&retmode=xml"  doc <- xmlTreeParse(fileUrl,useInternal=TRUE) | Load XML file (this is example from PubMed) |
| top <- xmlRoot(doc) | gets the root or top (or main wrapper) |
| xmlName(top) | returns node name |
| names(top) | gets list of name of elements under node |
| length(names(top)) | number of elements under node |
| top[[1]] | gets first element under rootnode |
| top[[1]][[1]] | gets first sub-element under first element under rootnode |
| art = top[[1]][["MedlineCitation"]][["Article"]][["AuthorList"]] | load subset of XML into file using element name as reference |
| xmlSApply(art[[1]], xmlValue) | use xmlSApply to get each element content into text |
| xmlSApply(art,function(x) xmlSApply(x,xmlValue)) | use nested xmlSApply to get each element content for all |
| xpathSApply(top,'//PubMedPubDate',xmlValue) | gets value of all nodes tagged with element name |
| nodes = getNodeSet(top, "//History/PubMedPubDate[@PubStatus='pubmed']") | uses XPath language to get all content under History sub-element PubMedPubDate with attribute pubstatus = ‘pubmed’ |
| fileUrl <- "http://espn.go.com/nfl/team/\_/name/bal/ baltimore-ravens"  doc <- htmlTreeParse(fileUrl,useInternal=TRUE) | gets data from html page with XML data |
|  |  |

use XPath to better understand how to extract data: <http://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/XML.pdf>

xml introduction: <http://www.omegahat.org/RSXML/shortIntro.pdf>

# JSON files

|  |  |
| --- | --- |
| Command | Description |
| install.packages("curl")  install.packages("jsonlite")  library(jsonlite) | library for connecting to JSON file |
| jsonData <- fromJSON("https://api.github.com/users/jtleek/repos") | Load JSON data into data table |
| names(jsonData) | gets all names of jsonData |
| names(jsonData$owner) | gest all names of jsonData within owner, which is data fram within data frame |
| names(jsonData$owner$login) | drills in again to specific variable |
| myjson <- toJSON(iris,pretty = TRUE) | converts iris data frame to JSON |

# data.table

|  |  |
| --- | --- |
| Command | Description |
| DT = data.table(x=rnorm(9), y=rep(c("a","b","c"),each=3),z=rnorm(9)) | create data table with 3 variables and 9 rows |
| tables() | see all tables in memory |
| DT[2,] | subset DT to see second row |
| DT[DT$y=="a",] | subset DT to see all rows with variable y = “a” |
| DT[c(2,3)] | subset DT to see second and third rows |
| DT[,list(mean(x),sum(z))] | returns mean of x and sum of z |
| DT[,table(y)] | creates tables from y values |
| DT[,w:=z^2] | creates another variable (column) |
| DT2 <- DT  DT[,y:=2] | DT2 is not a copy but a reference to DT, so such a change will change both |
| DT2 = copy(DT) | Make a copy of a data table |
| DT[,m:={tmp <- (x+z); log2(tmp+5)}] | Multiple step operation to take log base two of (x+z)+5 |
| DT[,a:=x>0] | creates logical variable based on x |
| DT[,b:=mean(x+w),by=a] | creates variable b, which is mean of x+w grouped by column a (such that the same value will be for all a and all b) |
| DT[, .N, by=x] | uses special variable .N to return frequency of each value x |
| DT <- data.table(x=rep(c("a","b","c"),each=100), y=rnorm(300))  setkey(DT,x)  DT['a'] | create a key on column a  ,then subset DT by referencing a |
| DT1 <- data.table(x=c('a','a','b','dt1'),y=1:4)  DT2 <- data.table(x=c('a','b','dt2'),z=5:7)  setkey(DT1,x); setkey(DT2,x)  merge(DT1,DT2) | use keys to join two tables |
| big\_df <- data.frame(x=rnorm(1E6),y=rnorm(1E6) |  |
| big\_df <- data.frame(x=rnorm(1E6),y=rnorm(1E6))  file <- tempfile()  write.table(big\_df,file=file,row.names=FALSE, col.names=TRUE, sep"\t",quote=FALSE)  fread(file) | fast reading is much faster than read.table() |

Resources:

* The latest development version contains new functions like melt and dcast for data.tables <https://github.com/Rdatatable/data.table>
* Here is a list of differences between data.table and data.frame <http://stackoverflow.com/questions/13618488/what-you-can-do-with-data-frame-that-you-cant-in-data-table>
* Notes based on Raphael Gottardo's notes <https://github.com/raphg/Biostat-578/blob/master/Advanced_data_manipulation.Rmd> , who got them from Kevin Ushey.

# Reading from mySQL

For installing see: <http://biostat.mc.vanderbilt.edu/wiki/Main/RMySQL>  
Make sure the Rprofile.site file default working directory is not changed!

|  |  |
| --- | --- |
| Command | Description |
| load(RMySQL) |  |
| uscsDb <- dbConnect(MySQL(),user="genome",host="genome-mysql.cse.ucsc.edu") | Connect to external MySQL server |
| result <- dbGetQuery(uscsDb,"show databases;"); dbDisconnect(uscsDb); | get databases from server, then disconnect |
| hg19 <- dbConnect(MySQL(),user="genome",db="hg19",host="genome-mysql.cse.ucsc.edu") | connect to specific table |
| allTables <- dbListTables(hg19) | get list of all tables |
| dbListFields(hg19,"affyU133Plus2") | get list of fields in table |
| dbGetQuery(hg19,"select count(\*) from affyU133Plus2") | count records in table |
| affyData <- dbReadTable(hg19,"affyU133Plus2") | extract table into data frame |
| query <- dbSendQuery(hg19,"select \* from affyU133Plus2 where misMatches between 1 and 3") | send and execute query without getting data |
| fetch(query,n=10) | get only first 10 records from query |
| dbClearResult(query) | clear query |
| dbDisconnect(hg19) | disconnect from host, always do this before continueing |
|  |  |

Resources

* RMySQL vignette <http://cran.r-project.org/web/packages/RMySQL/RMySQL.pdf>
* List of commands <http://www.pantz.org/software/mysql/mysqlcommands.html>
* A nice blog post summarizing some other commands <http://www.r-bloggers.com/mysql-and-r/>

# Reading from HDF5

* Used for storing large data sets
* Supports storing a range of data types
* Hierarchical data format
* groups containing zero or more data sets and metadata
  + Have a group header with a group name and list of attributes
  + Have a group symbol table with a list of objects in group
* datasets multidimensional array of data elements with metadata
  + Have a header with name, datatype, dataspace and storage layout
  + Have a data array with the data
* [www.hdfgroup.org](http://www.hdfgroup.org)

|  |  |
| --- | --- |
| Command | Description |
| source("http://bioconductor.org/biocLite.R")  biocLite("rhdf5")  library(rhdf5) | Install and load R HDF5 package |
| created = h5createFile ("example.h5") | creates h5 file |
| created = h5createGroup("example.h5","foo")  > created = h5createGroup("example.h5","baa")  > created = h5createGroup("example.h5","foo/foobaa")6 | creates groups and subgroups |
| h5ls("example.h5") | lists groups in h5 file |
| A = matrix(1:10,nr=5,nc=2)  h5write(A,"example.h5","foo/A") | write to group |
| B = array(seq(0.1,2.0,by=0.1),dim=c(5,2,2))  attr(B,"scale") <- "liter"  h5write(B,"example.h5","foo/foobaa/B") | write to group with named attribute |
| df = data.frame(1L:5L,seq(0,1,length.out=5),  ("ab","cde","fghi","a","s"),stringsAsFactors=FALSE)  h5write(df,"example.h5","df") | write data set |
| readA = h5read("example.h5","foo/A") | read h5 data |
| h5write(c(12,13,14),"example.h5","foo/A",index=list(1:3,1)) | write a chunk to h5 |

Resources

* hdf5 can be used to optimize reading/writing from disc in R
* The rhdf5 tutorial: <http://www.bioconductor.org/packages/release/bioc/vignettes/rhdf5/inst/doc/rhdf5.pdf>
* The HDF group has informaton on HDF5 in general <http://www.hdfgroup.org/HDF5/>

# Getting from the Web

* It can be a great way to get data: <http://www.theatlantic.com/technology/archive/2014/01/how-netflix-reverse-engineered-hollywood/282679/>

|  |  |
| --- | --- |
| Command | Description |
| library(XML)  library(RCurl) | Packages required to web scrape with HTTPS |
| fileUrl <- "https://scholar.google.com/citations?user=HI-I6C0AAAAJ"  xData <- getURL(fileUrl,ssl.verifypeer = FALSE)  html <- htmlTreeParse(xData,useInternalNodes=T)  xpathSApply(html,"//title",xmlValue)  xpathSApply(html,"//td[@class='gsc\_a\_c']",xmlValue) | get title  get values where class = gsc\_a\_c |
| library(httr)  content2 = content(html2,as="text")  parsedHtml = htmlParse(content2,asText=TRUE)  xpathSApply(parsedHtml,"//title",xmlValue) | using GET and httr package to read HTML data |
| pg1 = GET("http://httpbin.org/basic-auth/user/passwd",authenticate("user","passwd")) | using httr package to access website with password |
| google = handle("http://google.com")  pg1=GET(handle=google,path="/")  pg2=GET(handle=google,path="search") | using handles to save authentication across multiple accessses |

Resources

* R Bloggers has a number of examples of web scraping <http://www.r-bloggers.com/?s=Web+Scraping>
* The httr help file has useful examples <http://cran.r-project.org/web/packages/httr/httr.pdf>
* See later lectures on APIs

# Getting Data from APIs

|  |  |
| --- | --- |
| Command | Description |
| library(httr)  myapp <- oauth\_app("githdata-013jm", key="f0d908de7aa13d5bb112", secret="bcdef23eb5514138bc678fe93e464c976fa6b8f3") | establish connection to API. githdata-013jm, in this case |
| github\_token <- oauth2.0\_token(oauth\_endpoints("github"), myapp)  gtoken <- config(token = github\_token) | create token for API. github is specific to github |
| req <- GET("https://api.github.com/users/jtleek/repos", gtoken) | create request to application |
| stop\_for\_status(req) | check to see if request connection is good |
| content(req) | get the content from the request |

# Reading from other sources

## There is a package for that

* nice video on how there are R packages for most things you want to access
* best way to find “data storage mechanism R package” (e.g. MySQL R package)
* Interacting more directly with files:
  + file - open a connection to a text file
  + url - open a connection to a url file
  + gzfile - open a connection to a .gz file
  + bzfile - open a connection to a .bz3 file
  + ?connections for more information
  + **Remember to close connections**
* Foreign package - useful for other statistical software
  + Minitab, S, SAS, SPSS, Stata, Systat
* RPostresSQL
* RODBCS - interface to multiple databases
* RMongo - interface to MongoDb
* Reading images
  + jpeg
  + readbitmap
  + png
  + EBImage
* GIS data
  + rdgal
  + rgeos
  + raster
* Music data
  + tuner
  + seewave

# Subsetting and sorting

|  |  |
| --- | --- |
| Command | Description |
| set.seed(13435)  x <- data.frame("var1"=sample(1:5),"var2"=sample(6:10)  ,"var3"=sample(11:15))  x <- x[sample(1:5),]; x$var2[c(1,3)] = NA | scramble order and make values in var2 missing |
| x[,1] | subset specific column on index |
| x[,"var1"] | subset specific column on variable name |
| x[1:2, "var2"] | subset first two rows and var2 variable |
| x[(x$var1 <= 3 & x$var3 > 11),] | subset rows where var1 is <= 3 and var3 is > 11 |
| x[(x$var1 <= 3 | x$var3 > 15),] | subset rows where var1 is <= 3 or var3 is > 15 |
| x[which(x$var2 > 8),] | subset rows where var2 is greater than 8, ignoring missing values |
| sort(x$var1) | sort values in var1 in increasing order |
| sort(x$var1,decreasing = TRUE) | sort values in var1 in decreasing order |
| sort(x$var2,na.last=TRUE) | sort values in var1 in increasing order, making NA values last |
| x[order(x$var1),] | order entire data frame by var1 |
| x[order(x$var1,x$var3),] | order by multiple variables |
| library(plyr)  arrange(x,var1)  arrange(x, desc(var1)) | ordering using plyr library |
| x$var4 <- rnorm(5) | add row to data frame |
| y <- cbind(x,rnorm(5)) | create a new data frame using column bind |
| z <- rbind(y,rnorm(5)) | create a new data frame using row bind |

Resources: <http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf>

# Summarizing data

|  |  |
| --- | --- |
| Command | Description |
| fileURL <- "https://data.baltimorecity.gov/Culture-Arts/Restaurants/k5ry-ef3g"  download.file(fileURL,destfile="rest.csv")  restData <- read.csv("rest.csv") | for example |
| head(restData,n=3) | show first three rows |
| tail(restData,n=3) | show last three rows |
| summary(restData) | show distribution of each variable |
| str(restData) | describes structure of data frame |
| quantile(restData$councilDistrict,na.rm=TRUE) | summarizes quantiles for councilDistrict column |
| quantile(restData$councilDistrict,probs=c(0.5,0.75,.9)) | summarizes quantiles for specific percentiles |
| table(restData$zipCode,useNA="ifany") | summarizes distribution of variable into table. If any missing values will show number in NA. |
| table(restData$councilDistrict,restData$zipCode) | summarizes by two variables |
| sum(is.na(restData$councilDistrict)) | checks for number of missing values |
| any(is.na(restData$councilDistrict)) | checks if there is any na values in variable |
| all(restData$zipCode > 0) | checks if all values in restData are greater than 0 |
| colSums(is.na(restData)) | returns sum of each column on conditional test |
| all(colSums(is.na(restData))==0) | checks if all variables have no missing variables |
| table(restData$zipCode %in% c("21212","21213")) | returns number that have variable equal to one of the values |
| restData[restData$zipCode %in% c("21212","21213"),] | subsets data frame where variable is one of the values |
| data(UCBAdmission)  DF = as.data.frame(UCBAdmissions)  xt <- xtabs(Freq ~ Gender + Admit, data=DF) | create cross-tab with frequency by two variables specified |
| data(warpbreaks)  warpbreaks$replicate <- rep(1:9,len=54)  xt = xtabs(breaks ~.,data=warpbreaks) | create multiple cross-tabs on first two variables where each is third variable (replicate) |
| ftable(xt) | make flat table from multiple cross-tabs |
| fakeData = rnorm(1e5)  object.size(fakeData)  print(object.size(fakeData),units="Mb") | see size of object (default is bytes) |

# Create New Variables

## Why create new variables

* Often raw data won’t have value you are looking for
* You will need to transform data to get values you like
* Add those values to data frames working with
* Common variable creation
  + Missingness indicators
  + “Cutting up” quantitative variables
  + Applying transforms

|  |  |
| --- | --- |
| Command | Description |
| fileURL <- "https://data.baltimorecity.gov/Culture-Arts/Restaurants/k5ry-ef3g"  download.file(fileURL,destfile="rest.csv")  restData <- read.csv("rest.csv") | for example |
| restData$nearMe = restData$neighborhood %in% c("Roland Park","Homeland") | create logical variable based on neighborhood |
| restData$zipWrong = ifelse(restData$zipCode < 0, TRUE, FALSE) | create logical variable based on condition |
| restData$zipGroups = cut(restData$zipCode, breaks=quantile(restData$zipCode)) | creates categorical variable based on variable within four quantiles |
| library(Hmisc)  restData$zipGroups = cut2(restData$zipCode, g=4) | easier way to cut using Hmisc library |
| restData$zcf <- factor(restData$zipCode) | create factor variable from integer |
| yesno <- sample(c("yes","no"),size=10,replace=TRUE)  yesnofac = factor(yesno,levels=c("no","yes"))  yesnofac = relevel(yesnofac,ref="yes") | create factor variable  relevel so yes becomes 1 instead of 2 |
| as.numeric(yesnofac) | convert factor to numeric |
| library(plyr)  restData2 = mutate(restData,zipGroups=cut2(zipCode,g=4)) | using plyr mutate function to create new data frame with added variables |
| common transforms  abs(x)  sqrt(x)  ceiling(x)  floor(x)  round(x,digits=n)  signif(x,digits=n)  cos(x), sin(x)  log(x)  log2(x), log10(x)  exp(x) |  |

Resources:

* http://www.biostat.jhsph.edu/~ajaffe/lec\_winterR/Lecture%202.pdf
* <http://statmethods.net/management/functions.html>
* A tutorial from the developer of plyr - http://plyr.had.co.nz/09-user/
* Andrew Jaffe's R notes <http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf>

# Reshaping data

The goal of 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)

|  |  |
| --- | --- |
| Command | Description |
| library(reshape2)  mtcars$carname <- rownames(mtcars)  carMelt <- melt(mtcars,id=c("carname","gear","cyl"),measure.vars=c("mpg","hp")) | use melt to reshape data so that it is tall and skinny, one row for every measure.vars |
| cylData <- dcast(carMelt,cyl~variable) | puts cyl as rows, and aggregated variable as columns (defaults to length) |
| cylData <- dcast(carMelt,cyl~variable,mean) | puts cyl as rows, and aggregated variable as columns as mean |
| tapply(InsectSprays$count,InsetSprays$spray,sum) | Apply to count along the index spray the function sum. This returns a table which shows sum of count by spray |
| spIns = split(InsectSprays$count,InsectSprays$spray)  sprCount = lapply(spIns,sum) | using lapply to do the same, but return as list instead of table |
| unlist(sprCount)  sapply(spIns,sum) | to get table from list |
| library(plyr)  ddply(InsectSprays,.(spray),summarize,sum=sum(count)) | using plyr to do the same |
| spraySums <- ddply(InsectSprays,.(spray),summarize,sum=ave(count,FUN=sum)) | adding new variable, where you get the same length data set with sum for all A values |

Resources

* A tutorial from the developer of plyr - http://plyr.had.co.nz/09-user/
* A nice reshape tutorial http://www.slideshare.net/jeffreybreen/reshaping-data-in-r
* A good plyr primer - http://www.r-bloggers.com/a-quick-primer-on-split-apply-combine-problems/
  + See also the functions
  + acast - for casting as multi-dimensional arrays
  + arrange - for faster reordering without using order() commands
  + mutate - adding new variables

# Managing Data Frames with dplyr

The data frame is a key data structure in statistics and in R

* one observation per row
* each column represents a variable or measure or characteristic
* primary implementation that you will use is the default R implementations
* Other implementations, particularly relational database systems

## dplyr

* developed by Hadley Rickham of Rstudio
* optimized distilled version of plyr
* no new functionality per se, but greatly simplifies existing functionality in R
* Provides a “grammar” (in particular, verbs) for data manipulation
* Is very fast, as many key operations coded in C++

## dplyr Verbs

* select: return a subset of columsn of a data frame
* filter: extract a subset of rows from a data frame based on logical conditions
* arrange: reorder rows of a data frame
* rename: rename variables in a data frame
* mutate: add new variables/columns or transform existing variables
* summarise/summarize: generate summary statistics of different variables in data frame, possibly within strata
* There is a handy print method that prevents you from printing a lot of data to the console

## dplyr Properties

* first argument always data frame
* subsequent arguemtns describe what to do with it, refer ro columns in DF directly without using $ operator
* result is new data frame
* DF must be properly formatted and annotated for this to all be useful (factors properly leveled, variable names all there)

|  |  |
| --- | --- |
| Command | Description |
| install.packages("dplyr")  library(dplyr)  chicago <- readRDS("chicago.rds") | install & load dplyr  read chicago.rds for example |
| head(select(chicago,city:dptp)) | select only city, dptp and columns in between |
| head(select(chicago,-(city:dptp))) | select all columns except city, dptp and those in between |
| chic.f <- filter(chicago,pm25tmean2 > 30 & tmpd > 80) | filter table on two variable conditions |
| chic.f <- filter(chicago,pm25tmean2 %in% c(30,35)) | filter where pm25tmean2 is 30 or 35 |
| chicago <- arrange(chicago,date) | order chicago by date in ascending order |
| chicago <- arrange(chicago,desc(date)) | order chicago by date in descending order |
| chicago <- rename(chicago, pm25 = pm25tmean2, dewpoint = dptp) | rename columns pm25tmean2 as pm25, dptp as dewpoint |
| chicago <- mutate(chicago,pm25detrend = pm25-mean(pm25,na.rm=TRUE)) | add new variable pm25detrend which is deviation from the mean of the entire data set |
| chicago <- mutate(chicago, tempcat = factor(1 \* (tmpd > 80),labels = c("cold","hot")))  hotcold <- group\_by(chicago,tempcat)  summarize(hotcold,pm25=mean(pm25, na.rm=TRUE),o3=max(o3tmean2),no2 =median(no2tmean2)) | create a group\_by on tempcat, so that summarize will return results by that group |
| chicago <- mutate(chicago,year = as.POSIXlt(date)$year + 1900)  years <- group\_by(chicago,tempcat)  summarize(years,pm25=mean(pm25, na.rm=TRUE),o3=max(o3tmean2),no2 =median(no2tmean2)) | do the same by year |
| chicago %>% mutate(month = as.POSIXlt(date)$mon + 1) %>% group\_by(month) %>% summarize(pm25= mean (pm25, na.rm = TRUE),o3 = max(o3tmean2), no2 = median(no2tmean2)) | use pipeline operator (%>%) to summarize by month in one line of code |
| cran <- tbl\_df(mydf) | convert data frame to data frame tbl for better printing |
| mutate(data, DateTime = as.POSIXct(strptime(paste(Date,Time),"%d/%m/%Y %H:%M:%S"))) | using mutate to convert value from string to datetime, final must be POSIXct as POSIXlt will not work with dplyr |

* dplyr can work with other data frame “backends”
* data.table for large fast tables
* SQL interface for relational databases via the DBI package <http://master.bioconductor.org/help/course-materials/2006/rforbioinformatics/labs/thurs/DBI-howto.pdf>

# Merging

Important parameters: x,y,by,by.x,by.y,all

|  |  |
| --- | --- |
| Command | Description |
| fileUrl1 = "https://dl.dropboxusercontent.com/u/7710864/data/reviews-apr29.csv"  fileUrl2 = "https://dl.dropboxusercontent.com/u/7710864/data/solutions-apr29.csv"  download.file(fileUrl1,destfile="reviews.csv")  download.file(fileUrl2,destfile="solutions.csv")  reviews = read.csv("reviews.csv"); solutions <- read.csv("solutions.csv") | for example |
| mergedData = merge(reviews,solutions,by.x="solution\_id",by.y="id",all=TRUE) | joins reviews and solutions on reviews.solution\_id = solutions.id. all=TRUE means include records that do not match (with NAs for other columns) |
| mergedData2 = merge(reviews,solutions,all=TRUE) | by default will merge on all commonly named columns |
| arrange(join(reviews,solutions),id) | uses dplyr join to join table. Can only do left outer join |
| 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),x=rnorm(10))  dfList = list(df1,df2,df3)  join\_all(dfList) | dplyr join\_all to join on multiple columns |

# Tidyr

|  |  |
| --- | --- |
| Command | Description |
| library(tidyr)  gather(student,sex,count,-grade) | gathers (unpivots) table student, to create key on sex and count as value (-grade means grade column is kept as is) |
| separate(data=res,col=sex\_class,into=c("sex","class")) | separates column into two, splitting by detected separator |
| spread(res,test,grade) | spreads the values in column test into separate variable tables |
| extract\_numeric("class5") | extracts number from character |
| bind\_rows(passed,failed) | combines rows from two DF into one |
| select(contains( |  |

Resource: Hadley Wickham discusses his philosophy of tidy data:

<http://vita.had.co.nz/papers/tidy-data.pdf>

# Editing text variables

|  |  |
| --- | --- |
| Command | Description |
| tolower(names(cameraData)) | make all names lower case |
| splitNames = strsplit(names(cameraData),"\\.") | split based on delimiter |
| mylist <- list(letters=c("A","b","c"),numbers=1:3, matrix(1:25,ncol=5))  mylist[1]  mylist$letters  mylist[[1]] | reminder about lists  subsets out element of list  subsets out vector from element using name  subsets out vector from element using index |
| firstElement <- function(x){x[1]}  sapply(splitNames,firstElement) | use sapply to take out first element only |
| sub("\_","",names(reviews),) | substitute one character for another (replaces first only) |
| gsub("\_","",names(reviews)) | substitutes one character for another (replaces all) |
| grep("Alameda",cameraData$intersection) | finds all instances of string in another object |
| table(grepl("Alameda",cameraData$intersection)) | grepl returns binary for each element including string |
| cameraData2 <- cameraData[!grepl("Alameda", cameraData$intersection),] | subset df removing all records with substring found |
| grep("Alameda",cameraData$intersection,value=TRUE) | returns actual values containing substring |
| length(grep("Jeffstreet",cameraData$intersection)) | shows number of elements with that value |
| library(stringr)  nchar("Jeffrey Leek") | using stringr package, return number of characters in string |
| substr("Jeffrey Leek",1,7) | return substring 7 characters long, starting at first character |
| paste("Jeffrey","Leek") | combine strings, default delimiter is space |
| paste0("Jeffrey","Leek") | paste with no separater |
| str\_trim("Jeff ") | removes trailing spaces |
|  |  |
|  |  |

## Important points about text in data sets

* Names of variables should be
  + All lower case when possible
  + descriptive (Diagnosis versus Dx)
  + Not duplicated
  + Not have underscores or dots or white spaces
* Variables with character values
  + Should usually be made into factor variables (depends on application)
  + Should be descriptive (use TRUE/FALSE instead of 0/1, and Male/Female versus 0/1 or M/F)

# Regular expressions

* Regular expressions can be thought of as combinatiosn of literals and metacharacters
* Think of literal text forming the words of this language, and the metacharacters defining its grammar
* Regular expressions have a rich set of metacharacters

## Literals

Simplest pattern consists only of literals.

* Simplest pattern consists only of literals; a match occurs if the sequence of literals occur anywhere in the text being tested
* What if we only want the word “Obama”? or sentences that end in the word “Clinton”, or “clinton” or “clinto”?
* We need a way to express:
  + whitespaces word boundaries
  + sets of literals
  + the beginning and end of a line
  + alternatives (“war” or “peace) Metacharacters to the rescue!

## Metacharacters

* Some metacharacters represent the start of a line - ^i think
* $ represents the end of a line
* We can list a set of characters we will accept at a given point in the match. Example - this will match both upper and lower [Bb] [Uu] [Ss] [Hh]
* You can combine such as ^[Ii] am
* You can specify a range of letters [a-z] or [a-zA-Z], the order doesn’t matter
* When used at the beginning of a character class, the “^” is also a metacharacters and indicates matching characters not in the indicates class [^?.]$ returns any lines not ending in ? or .
* “.” is used to refer to any character. 9.11 matches 9-11
* | is or metacharacters, example flood|fire
* | can combine with other expressions, such as ^[Gg]ood|[Bb]ad
* Subexpressions are often contained in parantheses to constrain the alternatives, e.g. ^([Gg]ood|[Bb]ad) will look for either word but at start of line only
* Question mark indicates the indicated expression is optional, e.g. [Gg]eorge ( [Ww\.)? [Bb]ush
* Escaping metacharacters use backslash - e.g. \.
* \* and + are metacharacters means repetition; \* means “any number, including none, of the items” and + means at least one of them, e.g. [0-9]+ (.\*)[0-9]+ means at least one number then any number of characters followed but at least one number
* { and } are interval quantifiers; they let us specify the minimum and maximum number of matches of an expression: [Bb]ush( +[^ ]+ +){1,5} debate means at least one space, followed by something not a space, followed by at least one space between 1 and 5 times
  + m,n means at least m but not more than n matches
  + m means exactly m matches
  + m, means at least m matches
* Parentheses not only limit the scope of regular expressions, they also limit the scope of alternati alternatives divided by a “|”, but also can be used to “remember” text matched by the subexpression enclosed
  + We refer to the matched text with \1, \2
  + Expression +([a-zA-Z]+) +\1 +, is at least one space, followed by any number of characters followed by at least one space, then the whole thing repeated again
* \* is greedy so always matches the longest possible string that satisfies the regular expression
  + ^s(.\*)s is a line starting followed by any number of characters (including spaces) ending in s
* greediness can be turned off with the ?, as in
  + ^s(.\*?)s$

## Summary

* Regular expressions are used in many different languages; not unique in R
* Regular expressions are composed of literals and metacharacters that represent sets of classes of characters/words
* Text processing via regular expressions is a very powerful way to extract data from “unfriendly” sources (not all data comes as a CSV file)
* Used with the functions grep, grepl, sub, gsub and others that involve searching for text strings (Mark Hansen helped with material)

# Working with dates

|  |  |
| --- | --- |
| Command | Description |
| date() | returns current date and time as character |
| Sys.Date() | returns current date (no time) as Date |
| format(d2,"%a %b %d") | Format date in a specific format. [see](http://www.statmethods.net/input/dates.html) |
| x = c("1jan1960","2jan1960","31mar1960")  z = as.date(x,"%d%b%Y") | create dates from strings |
| z[1] - z[2] | display difference in days between two dates |
| as.numeric(z[1]-z[2]) | returns days difference as number |
| weekdays(d2) | show weekday |
| months(d2) | show months |
| julian(d2) | convert to Julian month, number of days since origin of 1970-01-01 |
| library(lubridate)  ymd("20140108")  mdy("08/04/2013")  dmy("03-04-2013") | use lubridate package to convert string to date in a given format |
| ymd\_hms("2011-08-03 10:15:03") | convert string to datetime |
| ymd\_hms("2011-08-03 10:15:03",tz="Pacific/Auckland") | convert string to datetime with specific timezone |
| x <- dmy(c("1jan2013","2jan2013","31mar2013"))  wday(x[1]) | get weekday as number |
| wday(x[1], label=TRUE) | show day name |
| this\_moment <- update(this\_moment,hours=8, minutes=34,seconds=55) | update multiple components of a date/time |
| depart <- nyc + days(2) | add days to a date/time |
| how\_long <- new\_interval(last\_time,arrive) | create interval between two times |
|  |  |
|  |  |

Resources

* More information in this nice lubridate tutorial <http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/>
* The lubridate vignette is the same content <http://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html>
* Ultimately you want your dates and times as class "Date" or the classes "POSIXct", "POSIXlt". For more information type ?POSIXlt

# Data resources

## Open Government Sites

* United Nations http://data.un.org/
* U.S. http://www.data.gov/
* List of cities/states with open data
* United Kingdom http://data.gov.uk/
* France http://www.data.gouv.fr/
* Ghana http://data.gov.gh/
* Australia http://data.gov.au/
* Germany https://www.govdata.de/
* Hong Kong http://www.gov.hk/en/theme/psi/datasets/
* Japan http://www.data.go.jp/
* Many more <http://www.data.gov/opendatasites>

## Gapminder

Human health across the planet

[**http://www.gapminder.org/**](http://www.gapminder.org/)

## Survey Data from United States

http://www.asdfree.com/

## Infochimps Marketplace

<http://www.infochimps.com/>

## Kaggle

Data science competitions

<http://www.kaggle.com/>

## Collections by data scientists

* Hilary Mason <http://bitly.com/bundles/hmason/1>
* Peter Skomoroch <https://delicious.com/pskomoroch/dataset>
* Jeff Hammerbacher <http://www.quora.com/Jeff-Hammerbacher/Introduction-to-Data-Science-Data-Sets>
* Gregory Piatetsky-Shapiro <http://www.kdnuggets.com/gps.html>
* <http://blog.mortardata.com/post/67652898761/6-dataset-lists-curated-by-data-scientists>

## More specialized collections

* Stanford Large Newtork Data
* UCI Machine Learning
* KDD Nugets Datasets
* CMU Statlib
* Gene expression omnibus
* ArXiv Data
* Public Data Sets on Amazon Web Services

## APIs with R interfaces

* twitter and twitteR package
* figshare and rfigshare
* PLoS and rplos
* rOpenSci
* Facebook and RFacebook
* Google maps and RGoogleMaps