# Course:

URL: <https://www.coursera.org/learn/data-cleaning>

# Week 1:

Refresh of getwd() and setwd()

Relative and absolute

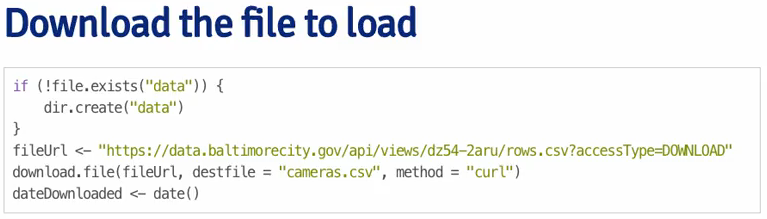
Absolute = setwd("C:/Users/steve.a.cooper/Google Drive/Training/datasciencedata")

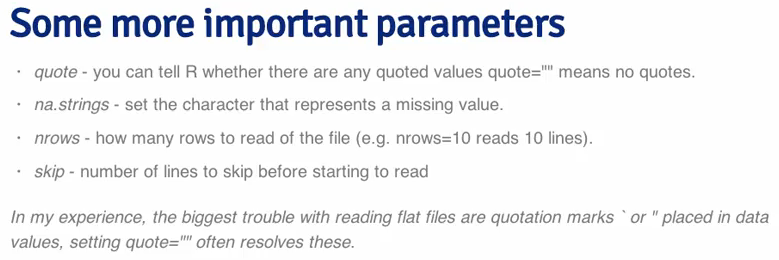
File.exists - check if a file exists

Dir.create – create a directory

if (!file.exists("check")) { dir.create("check") } – if the file doesn’t exist, it will create the directory

Check if file exists and download:

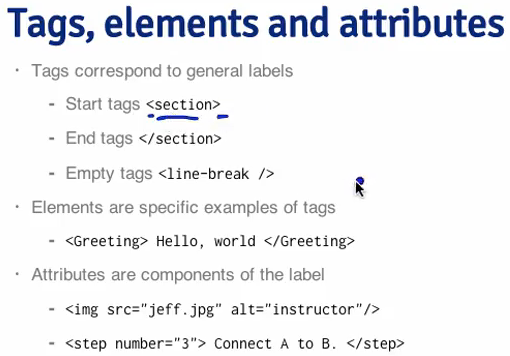




Reading the downloaded file to a table: example - IdahoHousing <- read.table("IdahoHousing.csv", sep =",", header = TRUE)

Reading specific rows / columns: dat <- read.xlsx("gas.xlsx", sheetIndex = 1, colIndex = colIndex, rowIndex = rowIndex)

XML



Read the contents of a specific element into a field: zip <- xpathSApply(rootNode, "//zipcode", xmlValue)

Count of the entries that equal a defined value (true): sum(zip == "21231") – remove NAs with sum(val == "24", na.rm = TRUE)

Web Scraping with R



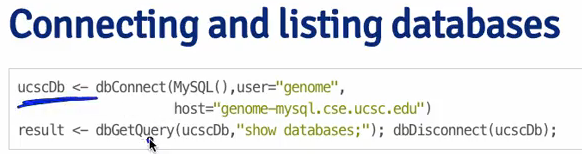
JSON is similar – you can use the jsonlite package to extract JSON to an R object with “fromJSON”, similarly, you can pass dataframes to JSON format with “toJSON”

# Week 2:

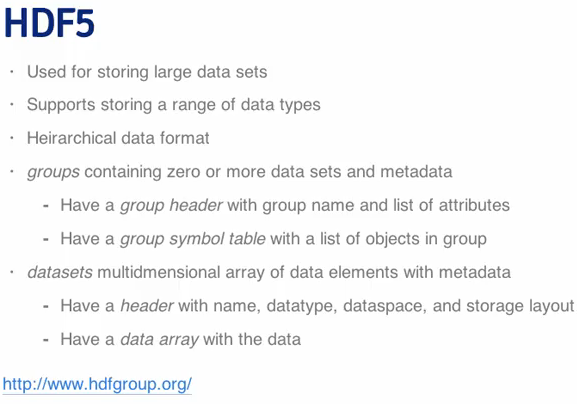
MySQL – connect – via internet or on your local machine

Apply a handle to each connection, go to the connection and get all databases on it…

“show databases;” is a MySQL command, being sent through the R function



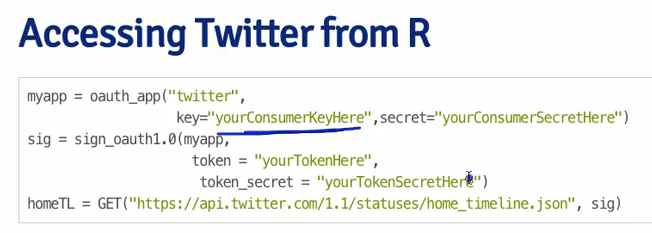
HDF5

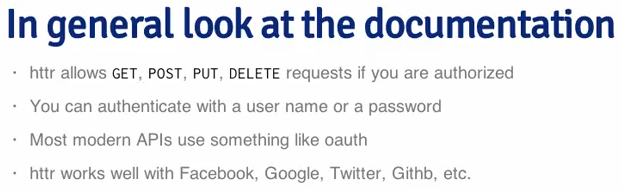


Web Scraping – can work with a number of functions; readLines, XML, GET from HTTR (which also allows you authenticate):



Using the HTTR package, you can access APIs:





# Week 3:

Subsetting by Column x[,1] or x[,”var1”]

Subsetting by rows and columns x[1:2,”var2”]

Subsetting with logical statements AND x[(x$var1 <= 3 & x$var3 > 11),]

OR x[(x$var1 <= 3 | x$var3 > 11),]

Use WHICH to ignore NA: x[(x$var2 > 8),] vs. x[which((x$var2 > 8)),]

Sorting sort(x$var1) – can be increasing, put NA at the end

Order the data frame by a particular variable x[order(x$var1),]

PLYR package can do the same with the arrange(x,var1) or descending

You can add rows x$var4 <- rnorm(5)

Column bind (cbind) does the same thing..

Summarizing data:

Head

Tail

Summary – to provide count of text fields, min/quartiles of numerics etc.

Str – tells you the class of variable, dimensions, classes of the columns

Quantile – of quantitative variables, including the probability of these quantile(x$var1, probs=c(0.5, 0.9))

Table – to make table of a variable, with count of frequency (use the function “useNA=”ifany” to highlight any NAs)

Two-dimensional tables, by passing two variables table(x$var1, x$var3)

Check for missing variables with sum(is.na(x$var2)) – you can use the any command to find any instances of NA

Use the all command to find if any statements are true

ColSums to apply sum to all columns

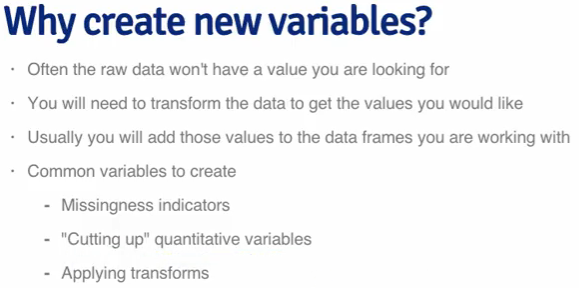
Values with a specific characteristic – table(x$var1 %in% 1)

.. you can also subset the dataset by this logical variable

Use cross tabs to create summaries by dimensions: xt = xtabs(Freq ~ Gender + Admit, data=dt)

Object.size to find the size of a data set (you can specify the units)

Creating new variables

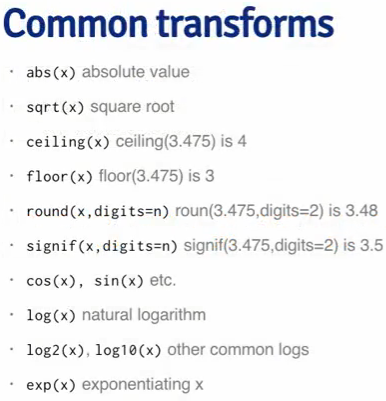


You can subset variables, create a new variable based on calculation: restData$nearMe = restData$neighborhood %in% c("Roland Park", "Homeland")

Create binary variables with the IFELSE function

Create categorical variables with the CUT and CUT2 functions – this is based on quantiles of the variables

Create factor variables with FACTOR function



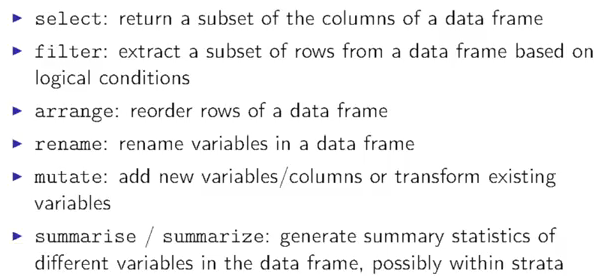
Can use the melt function to reshape the data – for example,

Create a data frame summarizing the data with dcast function

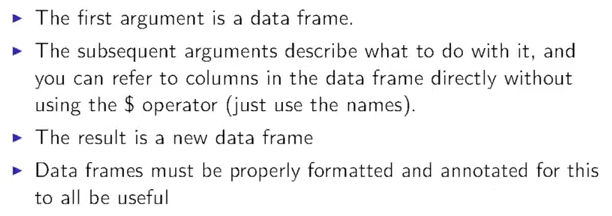
Apply to count, along the spray, the function sum – tapply(insect$count, insect$spray, sum)

You can also do this with “split, apply, combine”

Dplyr verbs:



Dplyr syntax:



Using dplyr:

Dim() show rows and columns

Names() show column names

Select by column names – select(Chicago, city:dptp)

Select, excluding column names – select(Chicago, -(city:dptp))

Filter rows – filter(Chicago, pm25tmean > 30 & tmp > 80)

Sorting – arrange(Chicago, date) or arrange(Chicago, desc(date))

Rename column names – rename(Chicago, pm25 = pm25tmean)

Transform existing or create new variables – mutate(Chicago, pmdetrend = pm25-mean(pm25))

Split a dataframe according to certain attributes – group\_by(Chicago, temp\_cat)

Summarize with – summarize(chicago, tmpd = mean(tmpd, na.rm = TRUE))

Pipeline variable %>% allows you to chain together a bunch of operations without having to build temporary variables

Merging data frames – merge(reviews, solutions, by.x=”solution.id”, by.y=”id”, all = TRUE)

You can merge in plyr (join) too, but it requires common names – not as full featured as merge

Plyr join is good for join\_all to merging multiple data frames

Swirl lesson

A powerful version of the datafram is tbl\_df – this provides more information to the print

?Comparison – for relational operators

Filter out NA values - filter(cran, !is.na(r\_version))

| The 'count' column, created with n(), contains the total number of rows (i.e. downloads) for each package. The 'unique'

| column, created with n\_distinct(ip\_id), gives the total number of unique downloads for each package, as measured by the

| number of distinct ip\_id's. The 'countries' column, created with n\_distinct(country), provides the number of countries

| in which each package was downloaded. And finally, the 'avg\_bytes' column, created with mean(size), contains the mean

| download size (in bytes) for each package.

pack\_sum <- summarize(by\_package,

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size))

tidyr – great package for tidying data, example:

gather – takes multiple columns and collapses into key-value pairs – gather(students, sex, count, -grade)

separate – separate one column into multiple columns… separate(res, sex\_class, c("sex", "class"))

spread – spread a key-value across multiple columns

# Week 4:

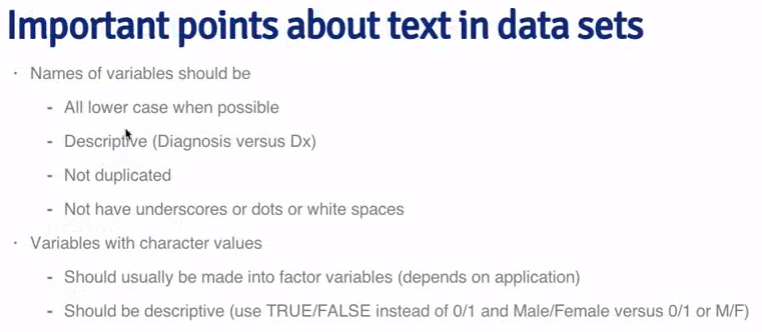
Convert character variables to lower – tolower(names(xx))

Convert character variables to upper – toupper(names(xx))

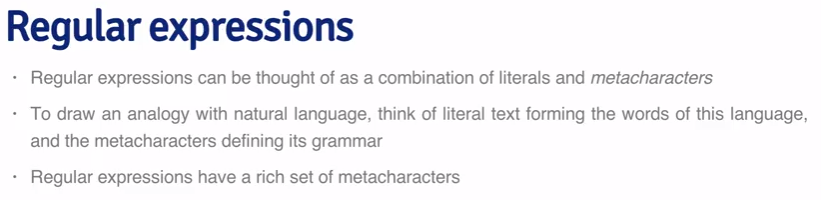
Split variable names by, e.g. period – strsplit(names(xx), “//.”)

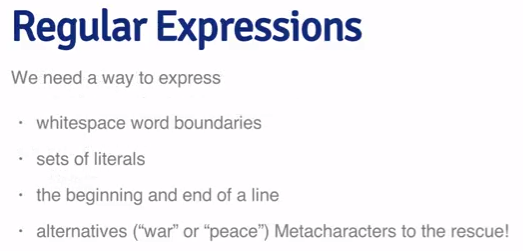
Substitute characters with others – sub(“\_”, “”, names(xx)) (hint: use gsub if there are more than one character in the variable)

Finding variables – grep(“Steve”, people$firstname) (hint: use grepl to find instances of true of the searched term e.g. table(grepl(“Steve”, people$firstname)) – you can also subset by using [!grepl…]



Regular expressions





Metacharacters:

Represent the start of the line with ^I think – the carrot

The end of the line with morning$ - the dollar sign

Character capitalization with [Tt] – the square brackets

You can combine these, for example ^[Ii] am

You can specify ranges - ^[0-9]

The carrot, at the end of the line is the same as “not” – [^?.]$

The dot refers to any character – 9.11 would be 9-11

The pipe means “or” – flood|fire

This can be combined with other metacharacters ^Good|Bad or ^(Good|Bad)

The question mark means that it is optional

Note: the use the escape (backslash) to remove the metacharacters usage e.g. \.

Repeat any number of times - \*

Repeat at least once - +

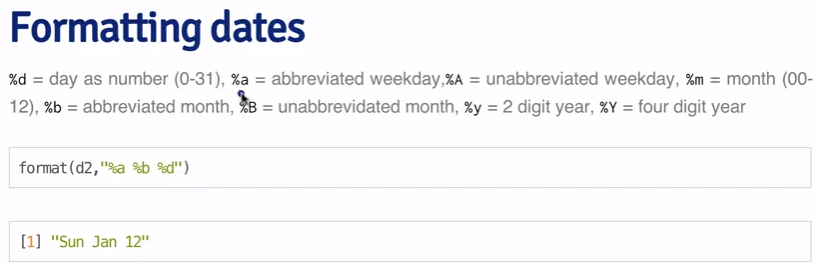
{} are interval qualifiers – how many times something should occur

Replicate with \

Dates

Current date – date()

System date – Sys.Date() (note: this is a date variable)



Lubridate library is good for looking for dates in any format