CS 636 - Data Analytics with R Programming

# Week 1 - Class Overview, R Basics and Advanced Data Structures

## Class Overview

Quizzes will be 1 hour and based on previous class material

Midterm 2 hours

Final 2.5 hours

Homework is 10% (labs are not graded in summer course).

Worst grade is the only one graded, so don’t miss it!

Term projects are allowed to be submitted in Python.

Jupyter will be used regardless of language choice.

Group consists of 1-4 people in a group.

R is required for quizzes, they are every other week

Midterms and Final are open-note, open-internet.

Dev environment needs to be set up beforehand.

## R Basics

Because it uses data using large tabular data, R allows us to work with the data more efficiently. It was the first language to create dataframes to access these large tables.

A weakness of R is that it is only used for statistics, and does not really have any other applications out of its niche.

There are alot of R packages available to deal with statistics and data analysis - a lot of statistical researchers provide their methods as packages.

See Jupyter notebook for examples of R Basics

Arithmetic operations and functions

Remainder or modulo operator: %%

Matrix multiplication( The Dot Product): %\*%

Sin, cos, tan, a\* for inverse and \*h for hyperbolic

Abs, ceiling, and floor(drops to lowest)

Exp, log, log2, log(x, base), sqrt, trunc(drops decimal)

Max, min, mean, median

Seq

rbind , cbind

Order, sort, rank

## Advanced Data Structures, I/O & Control Structures

Matrix

Mat = matrix(values, nrows, ncols) # makes all zeros

Mat = rbind(c(x11, x12), c(x21, x22))

Mat - cbind(c(x11, x21), c(x12, x22))

Mat = matrix(1:10, 2, 5, byrow=T) # makes 1:5 in first row, 6:10 in second because byrow=T

Can name row and col names with:

rownames(x)-c(‘r1’, ‘r2’)

colnames()

Can use dim() to convert vector into matrix

t() to create transpose

Can access data in vectors with [ ]

Can access in matrices [x, y], [,y] for all rows of col y, [x,] for all col of row x

Can specify multiples elements with [c(1,2), c(2,3)]

Can use typeof() and class() to check type and class of element

X[-val] excludes element

Can create logical vectors by conditions

& , | for AND, OR

&& and || used to return comparison on first value in vector

al(condition) and any(condition) return single T or F value based on entire vector

x = 1:5

all(x>1) is FALSE

any(x>1) is TRUE

Reading Data:

library() # lists all installed packages

data() # lists all dataframes

library(MASS) # loads a library

head(survey)

tail(survey) # prints head and tail of dataframe

scan() reads data from CLI.

Can also read data from formatted data files.

Whales = scan(file=”whale.txt)

Whales = read.table(file=”whale.txt”, header=TRUE)

read.table(file=file.choose()) #Prompts for file name

Reads from internet:

read.table(file="http://statweb.stanford.edu/~rag/stat141/exs/whale.txt", header=T)

A Dataframe, known as a ‘data matrix’ or a ‘data set’

* Unlike a matrix, different columns can be of different types
* Row names have to be unique.
* Alphabet = data.frame(index=1:26, symbol=LETTERS)

# Week 2 - Functions and Functional Programming

## Functions

f = function() {

cat(“Hello world\n”)

}

f() # Executes the function

Functions return the last expression that is evaluated

Default argument values can be defined in functions.

Arguments can be matched positionally or by name.

## Lazy Evaluation and the … argument

Arguments to functions are evaluated lazily, so they are only evaluated as needed in the body of the function.

The … is a special argument used to indicate that you can pass any number of parameters in the function. This is usually used to be passed on to other functions.

Any arguments that follow the .... argument must be named explicitly, and cannot be partially matched or matched positionally.

Other stuff:

* Superassignment is shown by <<- which is involved in maintainig state within a function.
* str() prints function prototype
* rnorm() generates multivariate normal random variates in the space X.

## Functional Programming

Functions can be assigned and passed like arguments.

A key feature is that data is allowed to be processed in multi-threaded parallel.

The apply() functions:

* apply() - apply a function over the margins of an array
  + Used to apply a function to the rows or columns of a matrix
* lapply() - loop over a list and evaluate a function on each element
  + Returns list of FUN applied to each element in provided list
  + If X is not a list, it will coerced to a list using as.list()
  + When you pass a function to lapply(), lapply() takes elements of the list and passes them as the first argument of the function you are applying
* sapply() - behaves similarly to lapply(), only different is the return value
  + Tries to intelliegently return right type
    - If returns list where every element is length 1, return vector
    - If returns list where every element is is vector of same length greater than 1, matrix is returned
    - If it cant figure it out, a list is returned.
* split() - combination of split() and lapply()sapply() is common
* mapply() - A multivariate apply of sorts
  + A simple way to send over multiple vectors as input to a function
* tapply() - applies a function over a subset of a vector
  + Passing in an index vector (I think to crete sub-groups)

mapply() can be used to “vectorize” a function - taking a function that takes a single argument and create a new function that can take vector arguments.

Vectorize() can create a vectorized version of your function.

# Week 3 - Manipulating Dataframes and Interactive Data Visualization

## Manipulating Dataframes

Like a matrix, however, only columns must be of the same type, and row names must be unique.

Use $ to access column name of a dataframe.+

Sort rows by order()

Selecting subsets of data by subset()

Splitting the data using split()

Retrieving data in a cell

* By index: mtcars[1,2] # rowName, colName
* By name: mtcars[“Mazda RX4”. “cyl”] # rowName, colName

Remember, dataframes are stored in memory as a list of vectors (of the same length).

* Column data can be referenced with *double square bracket* [[ ]]
  + >mtcar[[1]]
  + >mtcars[[“mpg”]]
* Using $ operator
  + >mtcars$mpg
* Using single square bracket
  + >mtcars[,”mpg”]

### Column slicing and Row slicing

The above methods return a vector. A smaller dataframe can be returned by using column slicing:

* Numeric Indexing
* Name Indexing

Row Slicing

* Numeric Indexing
* Name Indexing
* Logical Indexing

### Introduction to dplyr

Provides a “grammar” for data manipulation and for operating on dataframes.

**Important dplyr functions:**

* select() : select columns
* filter() : filter rows
* arrange() : re-order or arrange rows (i.e. sort)
* mutate() : create new columns
* summarise() : summarise values
* group\_by() : allows for group operations in the “split-apply-combine” concept

**Common dplyr Function Properties**

* The first argument is a dataframe
* Column names are referred by column name without $
* Returns a new dataframe
* Dataframes must be properly formatted and annotated in order to use these functions.

>select(msleep, name, sleep\_total) # selects name and sleep\_total from msleep

* Can use starts\_with(), ends\_with(), contains(0, matches(), one\_of(“x”, “y”)

>filter(mtcars, cyl == 8 & color == “green”)

>arrange(mtcars, cyl, disp)

>arrange(mtcats, desc(disp))

>rename(mtcars, new.name=old.name)

>mutate() and transmute()

group\_by() used to generate summary statistics from the data frame within strata defined by a variable

>by\_cyl = group\_by(mtcars, cyl)

summarise(by\_cyl, mean(disp), mean(hp))

Pipe operator - %>% - passes the result of one step as input for the next step in a sequence of operations

>lhs %>% rhs === rhs(lhs)

>lhs %>% rhs(a = 1) === rhs(lhs, a=1)

> lhs %>% rhs(a=1, b=. ) === rhs(a=1, b=lhs)

There is an example on slide 23 to reference use of %>%

### Introduction to reshape2 package

Makes it easy to transform data between wide and long formats.

Two major functions:

* melt() : Takes wide-format data and melts in into long-format data
* cast() : takes long-format data and cast it into wide-format data

id.vars is used as composite key for columns within table for melting, that way we can build back

aql < melt(df, id.vars = c(“month”, “day”))

dcast() is what we use for casting dataframes. We must provide ID variables in order to retain proper instance values.

aqw <dcast(aql, month + day ~ variable)

## Interactive Data Visualization

Because data visualization has changed in the past 10 years, and we are not solely using visualization in publications dynamic visualization has emerged.

ggplot2 is used for static plotting

plotly is used for dynamic plotting

### Basic Plotting: ggplot2

Install tidyverse, as it contains ggplot2 for visualization, dplyer for manipulation, tidyr for data tidying and others.

There are three components of a graph:

ggplot(data=mpg, mapping = aes(x=cty, y=hwy)) + geom\_point()

**Types of Statistical Data**

Most data fall into one of two groups: numerical or categorical

* Numerical - of you can take an average of the data, it is most likely numerical
  + Discrete
    - Represent items that can be counted
    - Take on possible values that can be listed out.
    - The list of possible values bay be *finite* or *countably infinite*
  + Continuous
    - Represent measurements
    - Their possible values cannot be counted and can only be described using intervals on the real number line
* Categorical
  + Used to represent characteristics such as a person’s gender, marital status, or hometown
  + Can take on numerical values, but numbers don’t have mathematical meaning.

**Basic barplot**

Uses dataframe for ‘data’ argument.

> ggplot(data, aws(x=name, y=value)) + geom\_bar(stat=”identity”)

# stat=”identity” uses sum of y for each category of x and is the height of the bar.

**Basic barplot without y variable**

Uses stat=”count” by default

**Grouped barplot**

Used to display multiple bars in one group.

Allows us to add a third dimension of data.

The aesthetics argument ‘aes()’ uses arg fill=<some\_condition> or factor

* A T/F condition would return 2 subgroups
* A factor would split the results based on that factor

geom\_bar(position= ) needs to be set to “dodge” to display bars side by side.

geom\_bar(stat= ) needs to be set to “identity”

*\* Note when using grouped barplots with a boolean condition as the fill arg, on the max factor will be displayed, other factors will be hidden underneath.*

**Stacked barplot**

Aes value is similar to grouped barplot

geom\_bar(position=”stack”, stat=”identity)

**Percent stacked barplot**

Aes value is similar to grouped barplot

geom\_bar(position=”fill”, stat=”identity)

**Small multiple barplot**

Can be used as an alternative of stacking or grouping, using ‘facet\_wrap()’ to specify the first level grouping

> ggplot(mtcars, aes(fill=factor(cyl), y=hp, x=factor(gear))) + geom\_bar(position=”dodge”, stat=”identity”) + facet\_wrap(~mpg>20)

**Differences between bar and histogram**

> + geom\_bar()

> + geom\_histogram()

**Line chart**

Requires at least two columns:

* An **ordered** numeric variable for X
* Another numeric value for Y

If x is not sorted, it will be sorted by R

Can add multiple geom types to one plot:

> + geom\_line()

Color, size, alpha, linetype

> + geom\_point()

> + geom\_smooth()

**Line Chart for Time Series**

ggplot2 will recognize data format automatically, and will use a specific type of X axis

* Use str(data) to confirm type
* If not read as data, use ‘lubridate’ or ‘anytime’ to convert

> scale\_x\_date(limit=c(as.Date(“3433-34-23). as.Date(“3434-32-53”)))

**Multi groups line chart for Time Series**

The input data frame is composed by 3 columns:

* An **ordered** numeric variable for the X axis. For time series, it is the Data column.
* Another numeric variable for the Y axis
* A categorical variable that specify the group of the observation

The idea is to draw one line per group

> see slides 29-31

### Introduction to plotly

Used to create interactive and dynamic web graphics for data analysis

In plotly, a *figure* has two key components

* Data (aka traces)
  + Defines a mapping from data and visuals
  + Every trace has a *type* (e.g. histogram, pie, scatter, etc.)
  + The trace type determines what other attributes are available
* Layout
  + layout() function anticipates a plotly object in its first argument
  + Other arguments add and/or modify various layout components of that object (e.g, the title).