STA 141A
Fundamentals of Statistical Data Science

Fall 2016

Instructor: Debashis Paul

Lecture 6

Agend for this week

- Working with random variables
- Basic set operations
- Working with data frames; Use of **dplyr** and **plyr** packages
- String manipulations; Regular expressions; Use of **stringr** package
- Reading structured data into R (partly covered in discussion 1)
- Quiz on Thursday (10/13) [less than 10 minutes]

Working with random variables

- R uses 4 types of functions: (i) to generate random variables; (ii) to compute pdf/pmf of their distributions; (iii) to compute probabilities; (iv) to compute quantiles of the distributions.
- Uniform distribution: **runif( ), duinf( ), punif( ), qunif( )**
- Normal distribution: **rnorm( ), dnorm( ), pnorm( ), qnorm(**)
- Other examples: **rbinom( ), rpois( ), rexp( ), rchisq( ), rf( ), rgamma( )**
- For **reproducibility** of the results, we would like to set the random seed to a specified value

```
rnorm(10) # generate a sample of 10 i.i.d. N(0,1) random variables  (repeat a few times)
set.seed(171); rnorm(10)  # set random seed to 171, then generate 10 i.i.d. N(0,1) r.v. (repeat)
```

Basic set operations

```
x = 1:10

y = c(2,4,3,7,8,12,5)

z = c("a","b",2,3,4)

union(x,y)  # union of sets x and y

intersect(x,y) # intersection of sets x and y

setdiff(x,y)  # set difference of y from x

setequal(x,y) # returns FALSE

intersect(x,z) # treats all the elements as characters

as.numeric(intersect(x,z))  # after obtaining common elements, treat them as numeric values
```

Merging (or joining) two data frames

- One can combine or ***join*** two data frames by common column names by making use of the R function **merge( )**. Syntax: merge(x,y,by=…)
- By default, the *two data frames are merged by their common columns* (separate specifications possible through the arguments by.x and by.y). Rows of the two data frames that ***match*** on the specified columns are extracted and joined together to form a new data frame.
- If there is more than one match (nonunique records), all possible matches contribute one row each.
- One can control the collection of rows in the resulting data frame by making use of the arguments all.x, all.y and all. If all.x = TRUE, all the non-matching cases of x are appended to the result, with NA filling the corresponding columns of y, analogously for all.y.

Example : merging data frames

```
students = data.frame(name=c("Bob","Jane","Alex","Luke"),
                 year = c("Junior","Junior","Sophomore","Senior"),   GPA=c(3.2,3.5,2.9,3.8))
grades = data.frame(name=c("Jane","Bob","Luke","Alex","Dale"),
                grade=c("B","A","C","B","A"),  major =c("Psy","Sta","Bio","CS","Soc"),
                year = c("Junior","Junior","Senior","Sophomore","Junior"))
merge(students, grades, by="name")    # returns data frame with additional columns
                                      # (treating “year” as different between two data frames)
merge(students, grades, by=c("name","year"))  # ensures unique identification of the records and variables
merge(students, grades, by="year")  # returns a larger data frame with additional  columns
                                    # since "year" does not uniquely identify the records
```

Working with data frames : **dplyr** package

- **dplyr** package has a set of functions that make the tasks of extracting information from a data frame significantly easier. These functions can also be interpreted as a set of "verbs". Some key examples are:
- select( ) : returns a subset of columns of a data frame, using a flexible notation
- filter( ) : extracts a subset of rows from a data frame based on some logical conditions
- arrange( ) : reorders rows of a data frame
- rename( ) : renames the variables (columns) in a data frame
- mutate( ) : add new variables (columns) or transforms existing variables
- group_by( ) : converts an existing data frame to a grouped (or stratified) data frame
- summarize( ) : generates summary statistics of different variables (applying functions), possibly within strata
- %>% : the "pipe" operator used to connect multiple verb actions together in a pipeline

Application of dplyr package

- We illustrate the use of **dplyr** package by applying them to extract information from a data set on effect of air quality on mortality in chicago metropolitan area. The data set is available as part of the package **gamair** .

data(chicago) # A data frame with 7 columns and 5114 rows. Each row refers to one day. The variables (columns) are:

death : total deaths (per day)

pm10median : median number of particles of diameter 2.5-10 micometer per cubic meter

pm25median : median number of particles of diameter < 2.5 micrometer per cubic meter (more dangerous)

o3median : Ozone in parts per billion

so2median : median Sulpher dioxide measurement

time : time in days

tmpd : temperature in fahrenheit

select( ) and filter( )

```
library(dplyr)  # load the package dplyr
select(chicago,death:o3median)  # or select(chicago,1:4), creates data frame with first 4 columns
select(chicago,-(so2median:time))  # excludes certain columns
select(chicago,ends_with("median")) # selects columns whose names end with “median”
select(chicago,starts_with("pm")) # selects columns whose names start with “pm”
filter(chicago,pm10median > 4) # extracts rows for which pm10median > 4
filter(chicago, !is.na(pm25median)) # extracts rows for which pm25median is not NA
filter(chicago, pm10median > 4 & so2median< 5 ) # extracts rows where both conditions are met
```

arrange( ), rename( ), mutate( ), transmute( )

```
arrange(chicago,tmpd) # reorder rows acoording to increasing values of tmpd
arrange(chicago,desc(tmpd)) # reorder rows acoording to decreasing values of tmpd
chicago.ren = rename(chicago,pm10=pm10median,o3=o3median,fht=tmpd)
# renames column pm10median as pm10 and o3median as o3 in resulting data frame
chicago.mute = mutate(chicago,pm10detrend=pm10median -mean(pm10median,na.rm=TRUE))
# transforms the variable pm10median to pm10detrend by removing its mean and appends it
chicago.trans = transmute(chicago, death=death,
                    pm10detrend=pm10median-mean(pm10median,na.rm=TRUE))
# mutate and then keep only the variables that are specified
```

group_by( ) and summarize( )

- We want to compute the mean of o3median within the strata defined by deciles of pm10median

```
pm10.qq = quantile(chicago$pm10median,seq(0,1,0.1), na.rm=TRUE)
# computes deciles of pm10median
chicago = mutate(chicago, pm10dec=cut(pm10median, pm10.qq)) # adds an extra column (pm10dec)
                                          # consisting of intervals with end points as successive values of the deciles
chicago.decile = group_by(chicago,pm10dec)  # treats pm10dec as a factor and creates the stratified
                                      # data frame  with strata defined by the values of pm10dec (decile intervals)
summarize(chicago.decile,o3.mean=mean(o3median,na.rm=TRUE))   # creates a data frame with values
# of mean(o3median) for each stratum of the data (as determined by the intervals or levels of pm10dec)
```

Use of "pipes" through %>%

- The ***pipeline operator*** %>% is pieces together multiple **dplyr** functions in a sequence of operations.

```
first(x) %>% second %>% third
```

- Here the (fictitious) function first( ) is applied to object (data frame) x, and the resulting object is then operated on by function second( ), and the resulting object is then subjected to the function third( ). Thus, the command above is equivalent to the following *composition of functions:*

```
third(second(first(x)))
```

- Example: given the deciles pm10.qq in the previous example, we can get the result as

```
chicago.decile = mutate(chicago, pm10dec=cut(pm10median, pm10.qq)) %>%
                 group_by(pm10dec)  %>%
                 summarize(o3.mean = mean(o3median, na.rm = TRUE))
```