R-FOUNDATION TRAINING: DATA MANIPULATION

ANALYTIXLABS

26 October 2015

1. IMPORT & UNDERSTAND DATA:

```
stores <- read.csv(choose.files())
View(stores)
edit(stores)
fix(stores)
head(stores)
tail(stores)
str(stores)
dim(stores)
ncol(stores)
names(stores)
summary(stores)
require(plyr)
describe(stores)
```

2. SUBSETTING DATA:

a.Method-1:

```
s0<-stores[c("Total_Customers", "BasketSize","Tenure")]

s1<-stores[stores$TotalSales>100, 1:10]
s2<-stores[stores$TotalSales>100, -(1:10)]
s3<-stores[stores$TotalSales>100, c("AcqCostPercust" , "BasketSize")]
```

b.Method-2:

```
s0<-cbind(No_cust=stores$Total_Customers,
Avg_transValue=stores$BasketSize,No_years=stores$Tenure)
```

c.Method-3:

```
s1<-subset(stores,TotalSales>100, select=c(1:10) )
s2<-subset(stores,TotalSales>100, select=-c(1:10) )
s3<-subset(stores,TotalSales>100, select=c("AcqCostPercust" , "BasketSize") )
```

3.CREATING NEW VARIABLES:

a.Method-1:

```
stores$Total_Cost <- stores$AcqCostPercust*stores$Total_Customers  +
stores$OperatingCost
```

b.Method-2:

```
stores <- transform(stores, Total_cost1 = AcqCostPercust*Total_Customers +
OperatingCost)
```

c.Binning variable-1:

```
stores$storeclass1[stores$TotalSales > 240 ] <- "High"
stores$storeclass1[stores$TotalSales > 120 & stores$TotalSales <= 240] <-
"Average"
stores$storeclass1[stores$TotalSales < 120] <- "Low"
```

d.Binning Variable-2:

```
stores$storeclass2 <- cut(stores$TotalSales,
                          breaks=c(-Inf, 120, 240, Inf),
                          labels=c("Low","Average","High"))
```

4. SORTING DATA:

Method-1:

```
newstores1 <- stores[order(stores$OperatingCost, decreasing=TRUE),] # use
built in function
newstores2 <- stores[order(-(stores$OperatingCost)),]              # use
built in function

# Reverse sorting
newstores5 <- stores[order(stores$Location, stores$TotalSales,
decreasing=FALSE),] #using built in function
newstores6 <- with(stores, stores[order(StoreType, Location, -TotalSales),])
```

Method-2:

```
library(dplyr)
newstores3 <- arrange(stores, StoreName)                # Use arrange from
plyr package
newstores4 <- arrange(stores, StoreName,desc(Location) ) # Use arrange from
plyr package

# Use arrange from dplyr package
newstores7 <- arrange(stores, -BasketSize)         # Use arrange from
```

```
dplyr/plyr package
newstores8 <- arrange(stores, -BasketSize, Location ) # Use arrange from
dplyr/plyr package
```

5.REMOVING DUPLICATES:

Method-1

```
Unique <- unique(score1)
```

Method-2

```
Unique <- score1[!duplicated(score1) ,]
```

6.FINDING DUPLICATES:

```
stores[duplicated(stores),]
duplicates <- score1[duplicated(score1$Student) &
duplicated(score1$Section),]
```

7. CONVERTING DATA TYPES:

a. CHARACTER TO NUMERIC OR FACTOR

```
n <- 10:14
is.numeric(n)

# Numeric to Character
c <- as.character(n)
is.character(c)

# Numeric to Factor
f <- factor(n)
is.factor(f)
```

b. CHARACTER TO NUMERIC OR FACTOR:

```
# Character to Numeric
as.numeric(c)
# Character to Factor
factor(c)
```

c. FACTOR TO CHARACTER OR NUMERIC:

```
#Converting a Factor to a Character vector is straightforward:
# Factor to Character
as.character(f)
```

```r
#converting a Factor to a Numeric vector is a little trickier.
#If you just convert it with as.numeric, it will give you the numeric coding
of the factor, which probably isn't what you want

as.numeric(f)

# Another way to get the numeric coding, if that's what you want:
as.numeric(as.character(f))

#The way to get the proper values is to first convert it to a Character
vector, then a Numeric vector.
# Factor to Numeric
as.numeric(as.character(f))
```

d. CHARACTER TO DATE

```r
# Character to Date
# You can use the as.Date( ) function to convert character data to dates. The
format is as.Date(x, "format"), where x is the character data and format
gives the appropriate format.
# convert date info in format 'mm/dd/yyyy'
strDates <- c("01/05/1965", "08/16/1975")
is.character(strDates)
dates <- as.Date(strDates, "%m/%d/%Y")
#The default format is yyyy-mm-dd
strDates1<- c("2007-06-22", "2004-02-13")
is.character(strDates1)
mydates <- as.Date(strDates1)
str(mydates)
```

e. DATE TO CHARACTER

```r
# Date to Character
# You can convert dates to character data using the as.Character( ) function.
# convert dates to character data
strDates <- as.character(dates)
```

8. JOINING(MERGING) DATA SETS:

```r
dd <- read.csv("Demographic_Data.csv",header = T)
td <- read.csv("Transaction_Summary.csv",header = T)

#INNER JOIN
dd.td.Inner <- merge(dd,td, by.x = c("CustName"), by.y = c("CustomerName"),
all=F)
#FULL JOIN
dd.td.Full  <- merge(dd,td, by.x = c("CustName"), by.y = c("CustomerName"),
all=T)
```

```r
#LEFT JOIN
dd.td.Left  <- merge(dd,td, by.x = c("CustName"), by.y = c("CustomerName"),
all.x=T)
#RIGHT JOIN
dd.td.Right <- merge(dd,td, by.x = c("CustName"), by.y = c("CustomerName"),
all.y=T)
```

9. RESHAPING DATA SETS:

```r
store_sales <- read.table(choose.files(),sep = ",",header = T)

#RESHAPING WIDE TO LONG
v1<-names(store_sales)[3:13]

store_sales.Wide_Long1 <- reshape(store_sales,
                                  idvar = c("StoreID","City"), # by group
variables
                                  varying = v1,               # variables will
be transposed
                                  timevar = c("Month"),       # Name of the
transposed variable
                                  v.names = c("Sales"),       # Name of the
variable which contains value
                                  times =
c("Jan","Feb_sales","Mar_sales","Apr_sales","May_Sales",
"June_sales","Jul_sales", "Aug_sales","Sep_Sales","Oct_Sales","Nov_Sales"),
direction = "long")

#RESHAPING LONG TO WIDE
Score.Long_Wide <- reshape(store_sales.Wide_Long,
                           idvar = c("StoreID","City"),
                           v.names=c("Sales"),
                           timevar = c("Month"),
                           direction = "wide")
```

10. SAMPLING:

```r
#sampling
sample(stores$StoreCode,size=10)

#Stratified sampling
#install.packages("sampling")
require(sampling)
strata(data, stratanames=NULL, size, method=c("srswor","srswr","poisson",
                                              "systematic"),
pik,description=FALSE)
```

```
## Create a 10% sample, stratified by location
#stores<-read.csv(choose.files())

p=0.1
d=stores
stratum="Location"

Text=paste0("stores$",stratum)
size <- ceiling(table(eval(parse(text=Text))) *p);
strat <- strata(d, stratanames = stratum, size = size, method = "srswor");
dsample <- getdata(d, strat);
table(dsample$Location)
```

11. RENAMING VARIABLES IN A DATA FRAME:

```
# renaming variables in a data frame
df2 <- stores # working on a copy so later examples still work
names(df2)
names(df2)[names(df2)=="StoreCode"] <- "StoreID"
names(df2)

library(reshape)
mydata <- rename(stores, c(Store="StoreCode", storeLoc = "Location"))
```

12. RE-ORDERING COLUMNS:

```
stores1 <- stores[c("StoreName", "StoreType","Location","OperatingCost" ,
"Total_Customers",
                    "AcqCostPercust", "BasketSize"
,"ProfitPercust","OwnStore")]
stores1<- stores[c(5,4,2,1,3,6:10)]
```

13. APPENDING DATA:

```
s1<-table(stores$StoreType)
s2<-prop.table(table(stores$StoreType))
# ADDING ROWS
rbind(s1,s2)

# ADDING COLUMNS
cbind(s1, s2)
```

14. Formating variables:

```r
# Defined with a array() function with 3 arguments: vector of values,
dimensions, dimension names
#Example-1
format(Sys.time(),format="%b_%d_%y_%H_%M")

#Example-2
format(1:10)
format(1:10, trim = TRUE)

#Example-3
zz <- data.frame("(row names)"= c("aaaaa", "b"), check.names = FALSE)
format(zz)
format(zz, justify = "left")

#Example-4
## use of nsmall
format(13.7)
format(13.7, nsmall = 3)
format(c(6.0, 13.1), digits = 2)
format(c(6.0, 13.1), digits = 2, nsmall = 1)

## use of scientific
format(2^31-1)
format(2^31-1, scientific = TRUE)
```

15. DATA MANIPULATION USING dplyr package:

```r
#nstall.packages("dplyr")
library(dplyr)

#Subsetting
filter(iris, Sepal.Length > 7)
slice(iris, 10:15)

select(iris, Sepal.Width, Petal.Length, Species)
select(iris, contains("."))
select(iris, ends_with("Length"))
select(iris, everything())
select(iris, matches(".t."))
select(iris, num_range("x", 1:5))
select(iris, one_of(c("Species", "Genus")))
select(iris, starts_with("Sepal"))
select(iris, Sepal.Length:Petal.Width)
select(iris, -Species)

#New Variable Creation
```

```r
mutate(iris, sepal = Sepal.Length + Sepal.Width)
transmute(iris, sepal = Sepal.Length + Sepal.Width)

#Removing duplicates
distinct(iris)

#Sorting
arrange(iris,Petal.Width, desc(Sepal.Length) )

#Getting top observations based on variable from data set
top_n(iris, 2, Sepal.Width)

# Summarization
summarise(iris, avg = mean(Sepal.Length))
summarise_each(iris, funs(mean))
View(iris)
group_by(iris, Species)
summarise(group_by(iris, Species), tot=sum(Sepal.Length),
          avg = mean(Sepal.Length))

# Flow

iris %>% group_by(Species) %>% summarise(tot=sum(Sepal.Length),
                                         avg = mean(Sepal.Length))
iris %>% group_by(Species) %>% mutate(tot=sum(Sepal.Length),
                                      avg = mean(Sepal.Length))

# Counting the number of observations based on variable
count(iris, Species, wt = Sepal.Length)

# Getting top observations from the data set based on variable for each
grouped variable
top_n(group_by(iris, Species), 2, Sepal.Width)

# Structure of data set(similar to str() function)
glimpse(iris)

# Renaming variables
rename(iris, wt = Sepal.Length)

# Joining(merging) the data sets

a <- read.csv("Demographic_Data.csv",header = T)
b <- read.csv("Transaction_Summary.csv",header = T)

left_join(a, b, by = c("CustName" = "CustomerName")) #- Join matching rows
from b to a.
right_join(a, b, by = c("CustName" = "CustomerName")) #- Join matching rows
from a to b.
```

```r
inner_join(a, b, by = c("CustName" = "CustomerName")) #- Join data. Retain
only rows in both sets.
full_join(a, b, by = c("CustName" = "CustomerName")) #- Join data. Retain all
values, all rows
semi_join(a, b, by = c("CustName" = "CustomerName")) #- All rows in a that
have a match in b.
anti_join(a, b, by = c("CustName" = "CustomerName")) #- All rows in a that do
not have a match in b.

# Set operations(appending etc.)
y <- read.csv("Score.csv",header = T)
z <- read.csv("Score1.csv",header = T)

intersect(y, z) #- Rows that appear in both y and z.
union(y, z) #- Rows that appear in either or both y and z.
setdiff(y, z) #- Rows that appear in y but not z.
bind_rows(y, z) #- Append z to y as new rows.
bind_cols(y, z) #- Append z to y as new columns.

# Simple Random sampling

sample_frac(iris, 0.5, replace = TRUE)
sample_n(iris, 10, replace = TRUE) ###Use replace = TRUE to perform a
bootstrap sample, and optionally weight the sample with the weight argument.
```

16. DATA MANIPULATION TOOLS:

a. USING SQL:

```r
## using SQL statements
library(sqldf)
newdf <- sqldf("Select * from dd as a left join td as b on a.CustName =
b.CustName")

sqldf("select avg(OperatingCost) as avg_OperatingCost, avg(TotalSales) as
avg_TotalSales,
      Tenure from stores where Staff_Cnt > 50 group by Tenure")
```

b. READING External R-SCRIPT

```r
setwd(choose.dir())
source("MyFirstRScript.R")
```

c. User defined function(UDF)

```r
ds_sum<-function(ds){
  h<-head(ds)
  t<-tail(ds)
  s<-str(ds)
```

```
  d<-dim(ds)
  n<-ncol(ds)
  na<-names(ds)
  l<-length(ds)
  return(list(head=h,tail=t,structre=s,dim=d,ncol=n,names=na,l))
}

stores_sum<-ds_sum(stores)  #calling function

stores_sum$names  # extracting objects from list(output)
```

d. Conditional Statements

```
#ifelse function

a = c(5,7,2,9)
ifelse(a %% 2 == 0,"even","odd")

#if statements
x <- 5
if(x > 0){
  print("Positive number")
}

#if else statment
x <- -5
if(x > 0){
  print("Non-negative number")
} else {
  print("Negative number")
}

#Nested if else
x <- -5
y <- if(x > 0) 5 else 6

x <- 0
if (x < 0) {
  print("Negative number")
} else if (x > 0) {
  print("Positive number")
} else
  print("Zero")
```

e. LOOPS

```
#For loop
x <- c(2,5,3,9,8,11,6)
count <- 0
for (val in x) {
```

```
  if(val %% 2 == 0)  count = count+1
}
print(count)

#While loop
i <- 1

while (i < 6) {
  print(i)
  i = i+1
}

#Break statement
x <- 1:5

for (val in x) {
  if (val == 3){
    break
  }
  print(val)
}

#Next statement
x <- 1:5

for (val in x) {
  if (val == 3){
    next
  }
  print(val)
}

#Repeat Loop
x <- 1

repeat {
  print(x)
  x = x+1
  if (x == 6){
    break
  }
}
```

f. APPLY FUNCTIONS – ALTERNATIVE FOR LOOPS

```
# Understanding the LOOPS VS. APPLY FUNCTIONS
for(i in 1:15)          # usage of loops
{
  print("Missing value summary")
```

```r
  print(colnames(stores)[i])
  print(class(stores[,i]))
  print(sum(is.na(stores[i]))/length(stores[i]))
  #print(summary(train[,i]))
}

 for(i in 1:15)          # usage of loops
{
   summary1[i]= sum(is.na(stores[i]))/length(stores[i])
}

#apply functions
# create a matrix of 10 rows x 2 columns
m <- matrix(c(1:10, 11:20), nrow = 10, ncol = 2)
# mean of the rows
apply(m, 1, sum)
apply(m, 2, sum)

# mean of the columns
apply(m, 2, mean)

# divide all values by 2
apply(m, 1:2, function(x) x/2)

# lapply
# Description: lapply returns a list of the same length as X, each element of
# which is the result of applying FUN to the corresponding element of X
# create a list with 2 elements
l <- list(a = 1:10, b = 11:15)
# the mean of the values in each element
lapply(l, mean)
lapply(l, sum)
sapply(l, mean)

# sapply
# Description:sapply is a user-friendly version of lapply by default
returning a vector or matrix if appropriate

# create a list with 2 elements
l <- list(a = 1:10, b = 11:20)
# mean of values using sapply
l.mean <- sapply(l, mean)
# what type of object was returned?
class(l.mean)
# it's a numeric vector, so we can get element "a" like this
l.mean[['a']]

# vapply
# Description: vapply is similar to sapply, but has a pre-specified type of
```

```r
# return value, so it can be safer (and sometimes faster) to use

l <- list(a = 1:10, b = 11:20)
# fivenum of values using vapply

l.fivenum <- vapply(l, fivenum, c(Min.=0, "1st Qu."=0, Median=0, "3rd Qu."=0,
Max.=0))
class(l.fivenum)
# [1] "matrix"
# let's see it
l.fivenum

# mapply
# Description: mapply is a multivariate version of sapply.
# mapply applies FUN to the first elements of each argument,
# the second elements, the third elements, and so on
l1 <- list(a = c(1:10), b = c(11:20))
l2 <- list(c = c(21:30), d = c(31:40))

# sum the corresponding elements of l1 and l2
mapply(sum, l1$a, l1$b, l2$c, l2$d)


# rapply
# Description: rapply is a recursive version of lapply.

# let's start with our usual simple list example
l <- list(a = 1:10, b = 11:20)

# log2 of each value in the list
rapply(l, log2)
rapply(l, log2, how = "list")

# what if the function is the mean?
rapply(l, mean)
rapply(l, mean, how = "list")


# tapply
# Description: Apply a function to each cell of a ragged array,
# that is to each (non-empty) group of values given by a
# unique combination of the levels of certain factors.

attach(iris)
# mean petal length by species
tapply(iris$Petal.Length, Species, mean)
```