Name :- Manvi Pandya
Class:- TE10 (L-10)
Roll No. 33235

Assignment: 5

AIM: Perform different operations using R/Python

PROBLEM STATEMENT:

Perform the following operations using R/Python on the Amazon book review and Facebook metrics data sets

a. Create data subsets

b. Merge Data

c. Sort Data

d. Transposing Data

e. Melting Data to long format

f. Casting data to wide format

OBJECTIVE:

To learn R/Python programming to learn different data pre-processing techniques.

THEORY:

Subsetting Data

R has powerful indexing features for accessing object elements. These features can be used to select and exclude variables and observations. The following code snippets demonstrate ways to keep or delete variables and observations and to take random samples from a dataset.

Selecting (Keeping) Variables

```
# select variables v1, v2, v3

myvars <- c("v1", "v2", "v3")

newdata <- mydata[myvars]


# another method

myvars <- paste("v", 1:3, sep="")

newdata <- mydata[myvars]
```

```
# select 1st and 5th thru 10th variables
newdata <- mydata[c(1,5:10)]
```

Excluding (DROPPING) Variables

```
# exclude variables v1, v2, v3
myvars <- names(mydata) %in% c("v1", "v2", "v3")
newdata <- mydata[!myvars]
# exclude 3rd and 5th variable
newdata <- mydata[c(-3,-5)]

# delete variables v3 and v5
mydata$v3 <- mydata$v5 <- NULL
```

Selecting Observations

```
# first 5 observations
newdata <- mydata[1:5,]

# based on variable values
newdata <- mydata[ which(mydata$gender=='F'
& mydata$age > 65), ]
# or
attach(mydata)
newdata <- mydata[ which(gender=='F' & age > 65),]
detach(mydata)
```

Selection using the Subset Function The subset( ) function is the easiest way to select variables and observations. In the following example, we select all rows that have a value of age greater than or equal to 20 or age less then 10. We keep the ID and Weight columns.

```
# using subset function
newdata <- subset(mydata, age >= 20 | age < 10,
select=c(ID, Weight))
```

In the next example, we select all men over the age of 25 and we keep variables weight through income (weight, income and all columns between them).

using subset function (part 2)

newdata <- subset(mydata, sex=="m" & age > 25,

select=weight:income)

Merging Data

Adding Columns To merge two data frames (datasets) horizontally, use the merge function. In most cases, you join two data frames by one or more common key variables (i.e., an inner join).

merge two data frames by ID

total <- merge(data frameA,data frameB,by="ID")

merge two data frames by ID and Country

total <- merge(data frameA,data frameB,by=c("ID","Country"))

Adding Rows To join two data frames (datasets) vertically, use the rbind function. The two data frames must have the same variables, but they do not have to be in the same order. total <- rbind(data frameA, data frameB)

If data frameA has variables that data frameB does not, then either:

1. Delete the extra variables in data frameA or 2. Create the additional variables in data frameB and set them to NA (missing) before joining them with rbind( ).

Going Further To practice manipulating data frames with the dplyr package, try this interactive course on data frame manipulation in R.

Sorting Data To sort a data frame in R, use the order( ) function. By default, sorting is ASCENDING. Prepend the sorting variable by a minus sign to indicate DESCENDING order. Here are some examples.

sorting examples using the mtcars dataset

attach(mtcars)

sort by mpg

newdata <- mtcars[order(mpg),]

sort by mpg and cyl

newdata <- mtcars[order(mpg, cyl),]

#sort by mpg (ascending) and cyl (descending)

newdata <- mtcars[order(mpg, -cyl),]

detach(mtcars)

Transpose

Use the t() function to transpose a matrix or a data frame. In the later case, rownames become variable (column) names.

example using built-in dataset

mtcars

t(mtcars)

The Reshape Package

Hadley Wickham has created a comprehensive package called reshape to massage data.

Basically, you "melt" data so that each row is a unique id-variable combination. Then you "cast" the melted data into any shape you would like. Here is a very simple example

mydata id time x1 x2

1 1 5 6

1 2 3 5

2 1 6 1

2 2 2 4

 # example of melt function

library(reshape)

mdata <- melt(mydata, id=c("id","time"))

newdata id time variable value

1 1 x1 5

1 2 x1 3

2 1 x1 6

2 2 x1 2

1 1 x2 6

1 2 x2 5

2 1 x2 1

2 2 x2 4

cast the melted data

 # cast(data, formula, function)

subjmeans <- cast(mdata, id~variable, mean)

timemeans <- cast(mdata, time~variable, mean)

subjmeans id x1 x2

1 4 5.5

2 4 2.5

timemeans time x1 x2

1 5.5 3.5

2 2.5 4.5

Melting:

There are many situations where data is presented in a format that is not ready to dive straight to exploratory data analysis or to use a desired statistical method. The reshape2 package for R provides useful functionality to avoid having to hack data around in a spreadsheet prior to import into R.

The melt function takes data in wide format and stacks a set of columns into a single column of data. To make use of the function we need to specify a data frame, the id variables (which will be left at their settings) and the measured variables (columns of data) to be stacked. The default assumption on measured variables is that it is all columns that are not specified as id variables.

Consider the following set of data:

```
> dat
  FactorA FactorB     Group1     Group2     Group3      Group4
1     Low     Low -1.1616334 -0.5228371 -0.6587093  0.45064563
2  Medium     Low -0.5991478 -1.0461138 -0.1942979  2.47985577
3    High     Low  0.8420797 -1.5413266  0.6318852 -0.98948125
4     Low  Medium  1.6225569 -1.2706469 -0.8026467 -0.32332181
5  Medium  Medium -0.3450745 -1.3377985  1.4988363  0.36541918
6    High  Medium  1.6025044  0.7631882 -0.5375833  0.85028148
7     Low    High -1.2991011 -0.2223622 -0.6321478 -1.57284216
8  Medium    High -0.4906400 -1.1802192  0.1235253  0.09891793
9    High    High  0.3897769 -0.3832142  0.6671101  0.23407257
```

There four groups are to used as part of a statistical analysis so we want to stack them into a single column and create an factor variable to indicate which group the measurement corresponds to and the **melt** function does the trick:

```
> melt(dat)
Using FactorA, FactorB as id variables
   FactorA FactorB variable       value
1      Low     Low   Group1 -1.16163338
2   Medium     Low   Group1 -0.59914783
3     High     Low   Group1  0.84207974
4      Low  Medium   Group1  1.62255690
5   Medium  Medium   Group1 -0.34507455
6     High  Medium   Group1  1.60250438

...
36    High    High   Group4  0.23407257
```

Consider a second set of data where there are two groups but we only want to retain the FactorB variable in the molten data set:

```
   FactorA   FactorB   Group1   Group2
1      Low  Very Low 6.851828 3.061329
2   Medium  Very Low 7.352169 1.303077
3     High  Very Low 6.918091 2.477875
4      Low       Low 7.402351 2.450527
```

```
8   Medium    Medium 8.251806 4.384492
9     High    Medium 8.339398 3.443789
10     Low      High 5.127386 2.868952
11  Medium      High 8.561181 3.616898
12    High      High 6.993838 3.450634
13     Low Very High 7.880877 2.950622
14  Medium Very High 9.439892 3.220295
15    High Very High 8.799447 3.106060
```

We now need to specify both the **id.vars** and **measure.vars** arguments in the **melt** function to get the desired output:

```
> melt(dat, id.vars = "FactorB", measure.vars = c("Group1", "Group2"))
     FactorB variable    value
1   Very Low   Group1 6.851828
2   Very Low   Group1 7.352169
3   Very Low   Group1 6.918091
4        Low   Group1 7.402351
5        Low   Group1 6.928385
6        Low   Group1 7.400626
...
30 Very High   Group2 3.106060
```

Conclusion:

We understood all the data preprocessing techniques and have successfully implemented all of the techniques on datasets.