Getting and Cleaning Data - Week 3

Carmelo Ramirez

26/10/2020

SUBSETTING AND SORTING

Subsetting - Quick Review

```
set.seed(13435)

X <- data.frame("var1" = sample(1:5), "var2" = sample(1:5), "var2" = sample(6:10), "var3" = sample(11:15
X <- X[sample(1:5),]; X$var2[c(1,3)] = NA
X

## Subsetting examples
X[,1]
X[,"var1"]
X[1:2, "var2"]

##Logicals And’s and Or’s
X[(X$var1 <= 3 & X$vars3 > 11),]
X[(X$var1 <= 3 | X$var3 > 15),]

## Dealing with missing values
X[which(X$var2 > 8),] ## Will return the indexes where the validation is true,
                      ## excluding NAs values
```

Sorting

```
sort(X$var1)

sort(X$var1, decreasing = TRUE)

sort(X$var2, na.last = TRUE)
```

Ordering

```
X[order(X$var1),]

## Sort first by var1, and if two rows have the same value,
## then it will order it by var3
X[order(X$var1, X$var3),]
```

Adding Rows and Columns

```
X$var4 <- rnorm(5)

## Using column bind command
Y <- cbind(X, rnorm(5))
```

SUMMARIZING DATA

Look at a bit of the data

```
head(data, n = 3) ## by default, head/tail methods return 6 rows

tail(data, n = 3)
```

Make a Summary

```
summary(data)
```

More in Depth Information

```
str(data)
```

Make table

```
## Create table of the data, a NA column is added at the end with its count.
## By default, table method ignores NAs values
table(data$var, useNA = "ifany")
```

Check for missing values

```
sum(is.na(data$var))

any(is.na(data$var))

all(data$var > 0)
```

Row and column sums

```
## Get the sums of each columns
colSums(is.na(data))
```

Values with specific characteristics

```
table(data$var %in% c("value"))

## returns true/false if the variable in the dataframe equals EITHER one of the
## values in the vector
table(data$var %in% c("value", "value2"))

## Get dataset with only the rows that equal values required
data[data$var %in% c("value", "value2"),]
```

Cross Tabs

```
data("UCBAdmissions")

DF = as.data.frame(UCBAdmissions)

## Freq will be the values displayed inside table
xt <- xtabs(Freq ~ Gender + Admit, data = DF)

xt
```

```
##         Admit
## Gender   Admitted Rejected
##   Male       1198     1493
##   Female      557     1278
```

CREATING NEW VARIABLES

Why create new variables

- Often the raw data won't have a value you are looking for
- You will need to tranform the data to get the values you would like
- Usually you will add those values to the data frames you are working with
- Common variables to create:
  - Missingness indicators
  - "Cutting up" quantitatives variables
  - Applying transforms

Creating Sequences

Sometimes you need an index for your data set

```
s1 <- seq(1, 10, by = 2) ; s1
```

```
## [1] 1 3 5 7 9
```

```
s2 <- seq(1, 10, length = 3) ; s2
```

```
## [1]  1.0  5.5 10.0
```

```
x <- c(1, 3, 8, 25, 100); seq(along = x)
```

```
## [1] 1 2 3 4 5
```

Subsetting variables

```
restData$nearMe = restData$neighborhood %in% c("Roland Park", "Homeland")
```

Creating Binary Values

```
restData$zipWrong = ifelse(restData$zipCode < 0, TRUE, FALSE)
```

Creating Categorical Variables

```
library(Hmisc)
restData$zipGroups = cut2(restData$zipCode, g = 4)
table(restData$zipGroups)
```

RESHAPING DATA

The Goal is Tidy Data

Tidy data principles: 1. Each variable forms a column 2. Each observation forms a row 3. Each table/file stores data about one kind of observation

Melting Data Frames

```
library(reshape2)

data(mtcars)

head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
mtcars$carname <- rownames(mtcars)

carMelt <- melt(mtcars, id = c("carname", "gear", "cyl"), measure.vars = c("mpg","hp"))

head(carMelt, n = 3)
```

```
##         carname gear cyl variable value
## 1     Mazda RX4    4   6      mpg  21.0
## 2 Mazda RX4 Wag    4   6      mpg  21.0
## 3    Datsun 710    4   4      mpg  22.8
```

```
tail(carMelt, n = 3)
```

```
##          carname gear cyl variable value
## 62  Ferrari Dino    5   6       hp   175
## 63 Maserati Bora    5   8       hp   335
## 64    Volvo 142E    4   4       hp   109
```

Casting Data Frames

```
## dcast(dataframeMelted, row ~ column)
cylData <- dcast(carMelt, cyl ~ variable)
```

```
## Aggregation function missing: defaulting to length
```

```
cylData
```

```
##   cyl mpg hp
## 1   4  11 11
## 2   6   7  7
## 3   8  14 14
```

```
cylData <- dcast(carMelt, cyl ~ variable, mean)
cylData
```

```
##   cyl      mpg        hp
## 1   4 26.66364  82.63636
## 2   6 19.74286 122.28571
## 3   8 15.10000 209.21429
```

DPLYR PACKAGE

dplyr verbs:

- *select*: return a subset of the columns of a data frame
- *filter*: extract a subset of rows from a data frame based on logical conditions
- *arrange*: reorder rows of a data frame
- *rename*: rename variables in a data frame
- *mutate*: add new variables/columns or transform existing variables
- *summarise/summarize*: generate summary statistics of different variables in the dataframe, possibly within strata

dplyr properties: - The first argument is a data frame - The subsequent arguments describe what to do with it, and you can refer to columns in the data frame directly without using the *$* operator (just use the names). - The result is a new data frame - Data frames must be properly formatted and annotated for this to all be useful.

Select Function

```r
## Select columns between column1 and columnN
select(dataFrame, column1:columnN)

## Select columns EXCEPT the ones specified
select(dataFrame, -(column1 : columnN))
```

Filter Function

```r
filter(dataFrame, column1 > 0 & columnX < 10)
```

Arrange Function

```r
## Arrange Data Frame by a specific column
arrange(dataFrame, columnX)

## Arrange Data Frame by a specific column, in DESCENDING order
arrange(dataFrame, desc(columnX))
```

Rename Function

```r
## Rename specific column name in Data Frame
rename(dataFrame, newNameCol1 = oldNameCol1, newNameCol2 = oldNameCol2)
```

Mutate Function

```
mutate(dataFrame, newColumn = value)
```

Group By Function

```
group_by(dataFrame, variable_to_be_grouped_by)
```

Summarize/Summarise Function

```
summarize(dataFrame, colName = value, col2Name = value, col3Name = value)
```

Summarize by year in a data frame:

```
chicago <- mutate(chicago, year = as.POSIXlt(date)$year + 1900)
years <- group_by(chicago, year)
summarize(years, pm25 = mean(pm25, na.rm = TRUE), o3 = max(o3tmean2), no2 = median(no2tmean2))
```

Chain *dplyr* functions

```
dataFrame %>% mutate(month = as.POSIXlt(date)$mon + 1) %>% group_by(month) %>% summarize(pm25 = mean(pm
```

MERGING DATA

Mergind Data - merge()

- Merged data
- Important parameters: *x*, *y*, *by*, *by.x*, *by.y*, *all*

```
mergedData = merge(reviews, solutions, by.x = "solution_id", by.y = "id", all = TRUE)
```

Using *join* in the plyr package

Faster, but less full featured - defaults to left join

```
library(plyr)
df1 = data.frame(id = sample(1:10), x = rnorm(10))
df2 = data.frame(id = sample(1:10), y = rnorm(10))
arrange(join(df1, df2), id)
```

```
## Joining by: id
##    id           x          y
## 1   1 -1.23968784 -0.4104048
## 2   2 -1.34747233 -0.2839854
## 3   3 -1.32685239 -2.3784514
## 4   4  0.94515412  0.1509111
## 5   5 -1.36037660 -0.2004282
## 6   6 -0.67303387  0.9295641
## 7   7 -0.79799272  0.7510976
## 8   8 -1.35389608 -0.8447342
## 9   9  0.12516612  0.4048521
## 10 10  0.05371413  0.2495960
```

If you have mulitple data frames

```
df1 = data.frame(id = sample(1:10), x = rnorm(10))
df2 = data.frame(id = sample(1:10), y = rnorm(10))
df3 = data.frame(id = sample(1:10), z = rnorm(10))

dfList = list(df1, df2, df3)

join_all(dfList)
```

```
## Joining by: id
## Joining by: id
```

```
##    id           x          y           z
## 1   6  2.01787173 -1.5750181 -0.85325709
## 2   2 -0.15710439  0.9412353 -1.11085227
## 3  10  1.03095278 -0.5006197  0.12817335
## 4   4 -0.12577961 -0.3135369 -0.68844259
## 5   7 -0.33917081 -2.8365960 -1.02026646
## 6   3  0.75468991 -0.4140041  0.01300922
## 7   5  0.24359469  0.3362831 -0.41044660
## 8   1 -2.09807385 -1.7780328  0.28137848
## 9   8 -0.09926964 -1.1362447  0.64253062
## 10  9 -0.45821114  1.0458963  2.21750170
```