Exploratory Data Analysis with R

Data Manipulation with Base R

Xuemao Zhang
East Stroudsburg University

September 28, 2022

Outline

- Sorting
- Merging
- Subsetting
  - removing missing values
- Rename
- Adding/Removing variables
- Data type conversion
- Aggregating

Sorting

- We have discussed several data manipulation methods.
- To sort a data frame in R, use the `order( )` function. By default, sorting is ASCENDING.

```
quiz = read.csv("../data/quiz2.csv",header=TRUE, sep=",");
attach(quiz); #database is searched by R when evaluating a
#variable, no need to use
#detach it when it is not in use
quiz1 = quiz[order(Q1),];  #sort by Q1
head(quiz1);
```

```
##    ID Q1 Q2 Q3   Q4 Q5 Q6
## 4   4  6  5  9  8.0  5 NA
## 12 12  6  8  5  8.0 10  8
## 29 29  6  9  4  5.0  9  0
## 38 38  6  9 10  9.5  9  8
## 1   1  8  9 10  9.5 10  8
## 2   2  8  8  8 10.0  9  8
```

Sorting

```r
quiz2 = quiz[order(Q1,Q2),]; # sort by Q1 and Q2
head(quiz2);
```

```
##     ID Q1 Q2 Q3   Q4 Q5 Q6
## 4    4  6  5  9  8.0  5 NA
## 12 12  6  8  5  8.0 10  8
## 29 29  6  9  4  5.0  9  0
## 38 38  6  9 10  9.5  9  8
## 36 36  8  7  5  8.0  5  0
## 2    2  8  8  8 10.0  9  8
```

Sorting

```r
quiz3 = quiz[order(Q1,-Q2),];
# sort by Q1 (ascending) and Q2 (descending)
head(quiz3);
```

```
##    ID Q1 Q2 Q3  Q4 Q5 Q6
## 29 29  6  9  4 5.0  9  0
## 38 38  6  9 10 9.5  9  8
## 12 12  6  8  5 8.0 10  8
## 4   4  6  5  9 8.0  5 NA
## 10 10  8 10 10 4.0 10  7
## 1   1  8  9 10 9.5 10  8
```

```r
detach(quiz);
```

Merging

- To merge two data frames horizontally (adding columns), use the `merge()` function. You join two data frames by one or more common key variables.
- The merged data set is a data frame. The columns are the common columns followed by the remaining columns in the first data set and then those in the second data set.
- When common key variables have different names, we use `by.x` and `by.y` to match them. Otherwise, we use `by` to specify the common variables.
- The option `all=TRUE` includes all data from both datasets.

Merging

- Merging two datasets require that both have at least one variable in common (either string or numeric).

```
exam = read.csv("../data/exam.csv",header=TRUE, sep=",");
grade=merge(quiz,exam,by="ID",all=TRUE);
head(grade);
```

```
##   ID Q1 Q2 Q3   Q4 Q5 Q6   T1   T2   T3
## 1  1  8  9 10  9.5 10  8 21.5 16.5 23.5
## 2  2  8  8  8 10.0  9  8 21.0 16.0 20.0
## 3  3 10  7 10 10.0 10  8 30.5 31.0 30.0
## 4  4  6  5  9  8.0  5 NA 23.0 18.5 25.5
## 5  5 10  6  8  6.0 NA NA 25.5 13.5   NA
## 6  6 NA  9 10 10.0 10 NA 27.5 27.0 35.5
```

Subsetting

- Often you only want to look at subsets of a data set at any given time. As a review, elements of an R object are selected using the brackets ([and]).
- Recall that if `x` is a vector of numbers and we can select the second element of `x` using the brackets and an index (2):

```
x = c(1, 4, 2, 8, 10);
x[2];
```

```
## [1] 4
```

Subsetting

- We can select the second AND fifth elements below:

```
x = c(1, 2, 4, 8, 10);
x[c(2,5)];
```

```
## [1]  2 10
```

Subsetting

- You can put a minus (−) before integers inside brackets to remove these indices from the data.

```r
x[-2] # all but the second
```

```
## [1]  1  4  8 10
```

Note that you have to be careful with this syntax when dropping more than 1 element:

```r
x[-c(1,2,3)] # drop first 3
```

```
## [1]  8 10
```

```r
# x[-1:3] # wrong! R sees as -1 to 3
x[-(1:3)] # needs parentheses
```

```
## [1]  8 10
```

Subsetting

What about selecting rows based on the values of two variables? We use logical statements. Here we select only elements of x greater than 2:

```r
x
```

```
## [1]  1  2  4  8 10
```

```r
x > 2
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

```r
x[ x > 2 ]
```

```
## [1]  4  8 10
```

Subsetting

You can have multiple logical conditions using the following:

- `&` : AND
- `|` : OR

```
x[ x > 2 & x < 5 ]
```

```
## [1] 4
```

```
x[ x > 5 | x == 2 ]
```

```
## [1]  2  8 10
```

Subsetting

- The `which` functions takes in logical vectors and returns the index for the elements where the logical value is `TRUE`.

```
which(x > 5 | x == 2); # returns index
```

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

```
x[ which(x > 5 | x == 2) ];
```

```
## [1]  2  8 10
```

```
x[ x > 5 | x == 2 ];
```

```
## [1]  2  8 10
```

Subsetting - removing missing values

- The function `complete.cases()` checks which observations/rows have no missing values.
- The `subset( )` function is an easy way to select variables and observations in base R.

```
grade0 = grade[complete.cases(grade), ];#Keep the complete rows onl
#Or use the subset() function
grade0=subset(grade, complete.cases(grade) == T);
str(grade0); dim(grade0);
```

```
## 'data.frame':    32 obs. of  10 variables:
##  $ ID: int  1 2 3 7 8 9 10 12 13 14 ...
##  $ Q1: int  8 8 10 10 10 10 8 6 10 10 ...
##  $ Q2: int  9 8 7 5 10 10 10 8 10 10 ...
##  $ Q3: int  10 8 10 9 10 6 10 5 8 10 ...
##  $ Q4: num  9.5 10 10 8 9 7 4 8 10 10 ...
##  $ Q5: int  10 9 10 10 10 8 10 10 10 9 ...
##  $ Q6: int  8 8 8 7 7 10 7 8 9 8 ...
##  $ T1: num  21.5 21 30.5 25.5 27.5 26 24.5 24 26.5 26 ...
##  $ T2: num  16.5 16 31 16 18.5 20.5 27.5 19 25.5 27 ...
```

Subsetting - removing missing values

- Another method: `na.omit()` function removes all incomplete cases of a data object (typically of a data frame, matrix or vector).

```
grade00 = na.omit(grade);
dim(grade00);
```

```
## [1] 32 10
```

- `questionr::na.rm()` is similar to `na.omit()` but allows to specify a list of variables to take into account.

```
library(questionr);
grade000 = na.rm(grade);
dim(grade000);
```

```
## [1] 32 10
```

Subsetting - removing missing values

- The function `is.na()` indicates which elements are missing.
- For example, we replace all missing values in the data set “grade” by 0.

```
grade[is.na(grade)] = 0;
head(grade); dim(grade);
```

```
##   ID Q1 Q2 Q3   Q4 Q5 Q6   T1   T2   T3
## 1  1  8  9 10  9.5 10  8 21.5 16.5 23.5
## 2  2  8  8  8 10.0  9  8 21.0 16.0 20.0
## 3  3 10  7 10 10.0 10  8 30.5 31.0 30.0
## 4  4  6  5  9  8.0  5  0 23.0 18.5 25.5
## 5  5 10  6  8  6.0  0  0 25.5 13.5  0.0
## 6  6  0  9 10 10.0 10  0 27.5 27.0 35.5
```

```
## [1] 40 10
```

Subsetting columns

- Selecting (keeping) variables.
- We can grab a column using the $ operator.

```
grade$T1;
```

```
##  [1] 21.5 21.0 30.5 23.0 25.5 27.5 25.5 27.5 26.0 24.5 32.0 24.0
## [16] 27.0 22.0 32.0 30.5 29.5 24.5 30.0 32.0 32.0 24.5 25.5 28.5
## [31] 31.5 24.0 29.0 28.0 32.0 28.5 26.0 27.5 29.5 27.5
```

Subsetting columns

- Selecting (keeping) variables using `subset()`.

```
keep=c("Q1","Q2","T1","T2"); #Keep these 4 variables only
grade1= subset(grade, select=keep);
str(grade1);
```

```
## 'data.frame':    40 obs. of  4 variables:
##  $ Q1: num  8 8 10 6 10 0 10 10 10 8 ...
##  $ Q2: num  9 8 7 5 6 9 5 10 10 10 ...
##  $ T1: num  21.5 21 30.5 23 25.5 27.5 25.5 27.5 26 24.5 ...
##  $ T2: num  16.5 16 31 18.5 13.5 27 16 18.5 20.5 27.5 ...
```

Subsetting columns

- Excluding (dropping) variables.

```
grade2= subset(grade, select=-c(Q4,Q5,Q6,T3));
 #drop these 4 variables
str(grade2);
```

```
## 'data.frame':    40 obs. of  6 variables:
##  $ ID: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Q1: num  8 8 10 6 10 0 10 10 10 8 ...
##  $ Q2: num  9 8 7 5 6 9 5 10 10 10 ...
##  $ Q3: num  10 8 10 9 8 10 9 10 6 10 ...
##  $ T1: num  21.5 21 30.5 23 25.5 27.5 25.5 27.5 26 24.5 ...
##  $ T2: num  16.5 16 31 18.5 13.5 27 16 18.5 20.5 27.5 ...
```

Subsetting columns

- We can also subset a `data.frame` using the bracket `[, ]` subsetting.
- For `data.frames` and matrices (2-dimensional objects), the brackets are `[rows, columns]` subsetting. We can grab the `x` column using the index of the column or the column name (“`carb`”)

```
mtcars[, 11];
```

```
##  [1] 4 4 1 1 2 1 4 2 2 4 4 3 3 3 4 4 4 1 2 1 1 2 2 4 2 1 2 2 4 6
```

```
mtcars[, "carb"];
```

```
##  [1] 4 4 1 1 2 1 4 2 2 4 4 3 3 3 4 4 4 1 2 1 1 2 2 4 2 1 2 2 4 6
```

Subsetting columns

- We can select multiple columns using multiple column names:

```
mtcars[, c("mpg", "cyl")];
```

```
##                       mpg cyl
## Mazda RX4            21.0   6
## Mazda RX4 Wag        21.0   6
## Datsun 710           22.8   4
## Hornet 4 Drive       21.4   6
## Hornet Sportabout    18.7   8
## Valiant              18.1   6
## Duster 360           14.3   8
## Merc 240D            24.4   4
## Merc 230             22.8   4
## Merc 280             19.2   6
## Merc 280C            17.8   6
## Merc 450SE           16.4   8
## Merc 450SL           17.3   8
## Merc 450SLC          15.2   8
## Cadillac Fleetwood   10.4   8
## Lincoln Continental 10.4   8
```

Subsetting columns

- We can select multiple columns using logic values

```
selected=names(mtcars) %in% c("mpg", "cyl")
mtcars[selected]
```

```
##                      mpg cyl
## Mazda RX4           21.0   6
## Mazda RX4 Wag       21.0   6
## Datsun 710          22.8   4
## Hornet 4 Drive      21.4   6
## Hornet Sportabout   18.7   8
## Valiant             18.1   6
## Duster 360          14.3   8
## Merc 240D           24.4   4
## Merc 230            22.8   4
## Merc 280            19.2   6
## Merc 280C           17.8   6
## Merc 450SE          16.4   8
## Merc 450SL          17.3   8
## Merc 450SLC         15.2   8
## Cadillac Fleetwood  10.4   8
```

Subsetting rows

- Selecting observations.
- We can subset rows of a `data.frame` with indices:

```
grade[c(1,3, 5),];
```

```
##   ID Q1 Q2 Q3   Q4 Q5 Q6   T1   T2   T3
## 1  1  8  9 10  9.5 10  8 21.5 16.5 23.5
## 3  3 10  7 10 10.0 10  8 30.5 31.0 30.0
## 5  5 10  6  8  6.0  0  0 25.5 13.5  0.0
```

```
grade3 = grade[1:5,];  # first 5 observations
grade3;
```

```
##   ID Q1 Q2 Q3   Q4 Q5 Q6   T1   T2   T3
## 1  1  8  9 10  9.5 10  8 21.5 16.5 23.5
## 2  2  8  8  8 10.0  9  8 21.0 16.0 20.0
## 3  3 10  7 10 10.0 10  8 30.5 31.0 30.0
## 4  4  6  5  9  8.0  5  0 23.0 18.5 25.5
## 5  5 10  6  8  6.0  0  0 25.5 13.5  0.0
```

```
grade4 = subset(grade, grade$ID >= 31);
# based on variable values
```

Rename

We can use the `colnames` function to directly reassign column names of a dataframe. Consider the data `mtcars`.

```
colnames(mtcars)[1:3] = c("MPG", "CYL", "DISP")
head(mtcars);
```

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

```
colnames(mtcars)[1:3] = c("mpg", "cyl", "disp") #reset
```

Adding variables

- We can add new variables to a dataframe.

```
grade[ , c("Total","Final")] = NA;
# add two more variables with values missing
str(grade);
```

```
## 'data.frame':    40 obs. of  12 variables:
##  $ ID   : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Q1   : num  8 8 10 6 10 0 10 10 10 8 ...
##  $ Q2   : num  9 8 7 5 6 9 5 10 10 10 ...
##  $ Q3   : num  10 8 10 9 8 10 9 10 6 10 ...
##  $ Q4   : num  9.5 10 10 8 6 10 8 9 7 4 ...
##  $ Q5   : num  10 9 10 5 0 10 10 10 8 10 ...
##  $ Q6   : num  8 8 8 0 0 0 7 7 10 7 ...
##  $ T1   : num  21.5 21 30.5 23 25.5 27.5 25.5 27.5 26 24.5 ...
##  $ T2   : num  16.5 16 31 18.5 13.5 27 16 18.5 20.5 27.5 ...
##  $ T3   : num  23.5 20 30 25.5 0 35.5 21.5 36 33 32.5 ...
##  $ Total: logi  NA NA NA NA NA NA ...
##  $ Final: logi  NA NA NA NA NA NA ...
```

Adding variables

```r
grade$Letter = NA;
# add one more variable with values missing
str(grade);
```

```
## 'data.frame':    40 obs. of  13 variables:
##  $ ID    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Q1    : num  8 8 10 6 10 0 10 10 10 8 ...
##  $ Q2    : num  9 8 7 5 6 9 5 10 10 10 ...
##  $ Q3    : num  10 8 10 9 8 10 9 10 6 10 ...
##  $ Q4    : num  9.5 10 10 8 6 10 8 9 7 4 ...
##  $ Q5    : num  10 9 10 5 0 10 10 10 8 10 ...
##  $ Q6    : num  8 8 8 0 0 0 7 7 10 7 ...
##  $ T1    : num  21.5 21 30.5 23 25.5 27.5 25.5 27.5 26 24.5 ...
##  $ T2    : num  16.5 16 31 18.5 13.5 27 16 18.5 20.5 27.5 ...
##  $ T3    : num  23.5 20 30 25.5 0 35.5 21.5 36 33 32.5 ...
##  $ Total : logi  NA NA NA NA NA NA ...
##  $ Final : logi  NA NA NA NA NA NA ...
##  $ Letter: logi  NA NA NA NA NA NA ...
```

Data type conversion

- Use `is.Type()` to test for data Type.
- Use `as.Type` to explicitly convert it.

```
is.numeric(),    as.numeric()
is.character(),  as.character()
is.vector(),     as.vector()
is.matrix(),     as.matrix()
is.data.frame(), as.data.frame()
```

```
mtcars1=mtcars;
mtcars1$cyl=as.factor(mtcars1$cyl);
str(mtcars1);
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ..
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
```

Aggregating Data

- aggregate() function splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.
- The by variables must be in a list (even if there is only one).

```
# for numeric variables only
# aggregate data frame mtcars by cyl
# returning means

attach(mtcars);
aggdata =aggregate(mtcars, by=list(cyl),FUN=mean, na.rm=TRUE);
print(aggdata);
```

```
##   Group.1      mpg cyl     disp        hp     drat       wt     c
## 1       4 26.66364   4 105.1364  82.63636 4.070909 2.285727 19.13
## 2       6 19.74286   6 183.3143 122.28571 3.585714 3.117143 17.97
## 3       8 15.10000   8 353.1000 209.21429 3.229286 3.999214 16.77
##          am     gear     carb
## 1 0.7272727 4.090909 1.545455
## 2 0.4285714 3.857143 3.428571
## 3 0.1428571 3.285714 3.500000
```

License