Module 5: Data manipulation

BIOSTAT 213: Introduction to Computing in the R Software Environment

Yea-Hung Chen, PhD, MS

Monday, October 21, 2019

Logical objects

Logical objects

- `is.element()`, revisited:

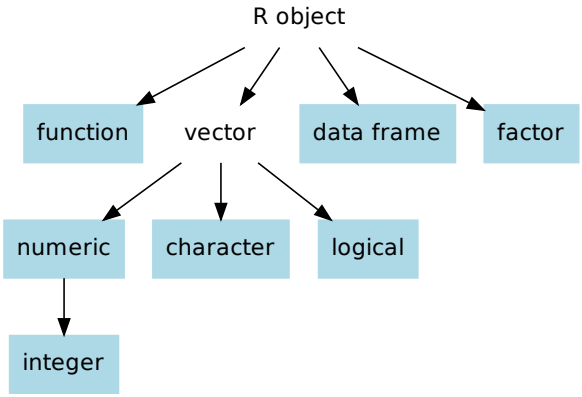
```
> is.element(1:6,5)
[1] FALSE FALSE FALSE FALSE  TRUE FALSE
```

Logical objects

```
> tt<-is.element(1:6,5)
> class(tt)
[1] "logical"
> is.vector(tt)
[1] TRUE
```

- `is.element()` returns a logical object
- logical objects are vectors

Logical objects



Logical objects

- logical objects store true/false values
- values appear as `TRUE` or `FALSE`

```
> is.element(1:6,2:3)
[1] FALSE  TRUE  TRUE FALSE FALSE FALSE
```

Equality

- to test for equality, use ==:

```
> pi==3.14
[1] FALSE
```

Equality

```
> years<-2016:2019
> years
[1] 2016 2017 2018 2019
> years==2017
[1] FALSE  TRUE FALSE FALSE
```

Equality

```
> animals<-c('giraffe','zebra','elephant','lion')
> animals
[1] "giraffe"  "zebra"    "elephant" "lion"
> animals=='elephant'
[1] FALSE FALSE  TRUE FALSE
> animals=='horse'
[1] FALSE FALSE FALSE FALSE
```

Inequality

- to test for inequality, use `!=`:

```
> years
[1] 2016 2017 2018 2019
> years!=2017
[1]  TRUE FALSE  TRUE  TRUE
```

OR

- for the OR operator, use |:

```
> ('elephant'=='elephant')|(pi==pi)
[1] TRUE
> ('elephant'=='elephant')|(pi==3)
[1] TRUE
> ('elephant'=='giraffe')|(pi==pi)
[1] TRUE
> ('elephant'=='giraffe')|(pi==3)
[1] FALSE
```

- the parentheses here are unnecessary, but can help readability

OR

```
> years==2017
[1] FALSE  TRUE FALSE FALSE
> animals=='lion'
[1] FALSE FALSE FALSE  TRUE
> (years==2017)|(animals=='lion')
[1] FALSE  TRUE FALSE  TRUE
```

OR

- as seen in Module 1 with mathematical operations, the vectors need not be of the same length: R will recycle the shorter vector as many times as necessary

```
> 1:4==2
[1] FALSE  TRUE FALSE FALSE
> 1:2==3
[1] FALSE FALSE
> (1:4==2)|(1:2==3)
[1] FALSE  TRUE FALSE FALSE
```

- however, as mentioned in Module 1, I find recycling of vectors confusing and discourage it

AND

- for AND, use &:

```
> ('elephant'=='elephant')&(3==3)
[1] TRUE
> ('elephant'=='elephant')&(3==-3)
[1] FALSE
> ('elephant'=='giraffe')&(3==3)
[1] FALSE
> ('elephant'=='giraffe')&(3==-3)
[1] FALSE
```

AND

```
> years==2020
[1] FALSE FALSE FALSE FALSE
> animals=='lion'
[1] FALSE FALSE FALSE  TRUE
> (years==2020)&(animals=='lion')
[1] FALSE FALSE FALSE FALSE
```

Numerical comparisons

- `<` for less than
- `<=` for less than or equal to
- `>` for greater than
- `>=` for greater than or equal to

Numerical comparisons

```
> 3.14<pi
[1] TRUE
> 1:6<=3
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE
> 1:5>4
[1] FALSE FALSE FALSE FALSE  TRUE
```

Numerical comparisons

```
> x<-1:6
> y<-c(3,6,2,1,2,4)
> x>y
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
> x<=y
[1]  TRUE  TRUE FALSE FALSE FALSE FALSE
```

Negation

- for negation, use `!`:

```
> years
[1] 2016 2017 2018 2019
> is.element(years,c(2016,2018))
[1]  TRUE FALSE  TRUE FALSE
> !is.element(years,c(2016,2018))
[1] FALSE  TRUE FALSE  TRUE
```

Negation

```
> years==2017
[1] FALSE  TRUE FALSE FALSE
> !years==2017
[1]  TRUE FALSE  TRUE  TRUE
> !(years==2017)
[1]  TRUE FALSE  TRUE  TRUE
> years!=2017
[1]  TRUE FALSE  TRUE  TRUE
```

Negation

```
> years>=2018
[1] FALSE FALSE  TRUE  TRUE
> !years>=2018
[1]  TRUE  TRUE FALSE FALSE
> !(years>=2018)
[1]  TRUE  TRUE FALSE FALSE
> years<2018
[1]  TRUE  TRUE FALSE FALSE
```

Logical objects and mathematical operations

```
> tt<-is.element(years,c(2016,2018))
> tt
[1]  TRUE FALSE  TRUE FALSE
> tt*1
[1] 1 0 1 0
> tt+1
[1] 2 1 2 1
> sum(tt)
[1] 2
```

- in mathematical operations, TRUE is treated as 1 and FALSE is treated as 0

Missing values

Missing values

- missing values in the NHANES data:

```
> head(ii$alq110)
[1] NA NA  1  1 NA NA
```

- missing values appear in R as `NA`

Missing values and `read.csv()`

- `read.csv()` automatically recognized the empty "cells" in `nhanes.csv` as missing values
- to recognize other character codings for missing values, use `read.csv()`'s `na.strings` argument
- the default is `''`, meaning an empty cell, as mentioned above

is.na()

- to check for missing values, use `is.na()`

```
> animals
[1] "giraffe"  "zebra"    "elephant" "lion"
> is.na(animals)
[1] FALSE FALSE FALSE FALSE
> head(ii$alq110)
[1] NA NA  1  1 NA NA
> is.na(head(ii$alq110))
[1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE
```

is.na()

- notice that `is.na()` returns a logical object:

```
> is.na(head(ii$alq110))
[1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE
> class(is.na(head(ii$alq110)))
[1] "logical"
```

Checking for missing values

```
> table(is.na(ii$lbdhdd))

FALSE  TRUE
 4014   195
> sum(is.na(ii$lbdhdd))
[1] 195
```

- notice that the second example takes advantage of the inherent numeric values of TRUE and FALSE

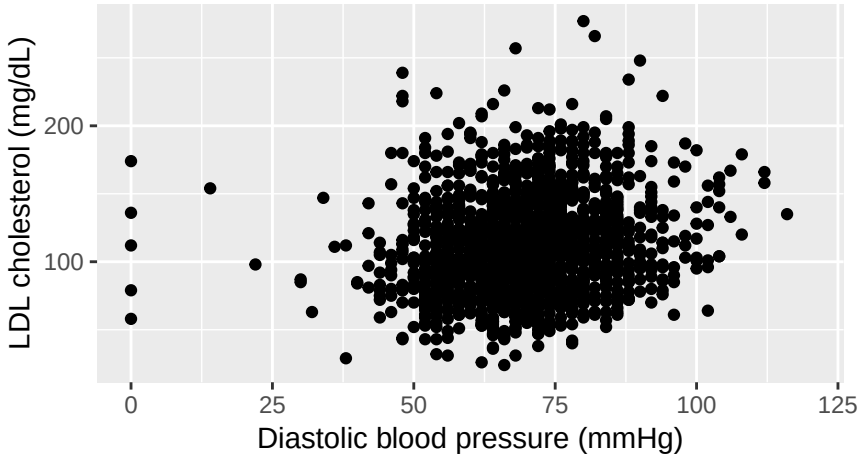
Assigning missing values

```
> x<-15
> x
[1] 15
> is.na(x)
[1] FALSE
> is.na(x)<-TRUE
> x
[1] NA
> is.na(x)
[1] TRUE
```

Assigning missing values

```
> x<-1:7
> x
[1] 1 2 3 4 5 6 7
> is.na(x)
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> is.na(x)<-(x>5)
> x
[1]  1  2  3  4  5 NA NA
> is.na(x)
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
```

Assigning missing values



Assigning missing values

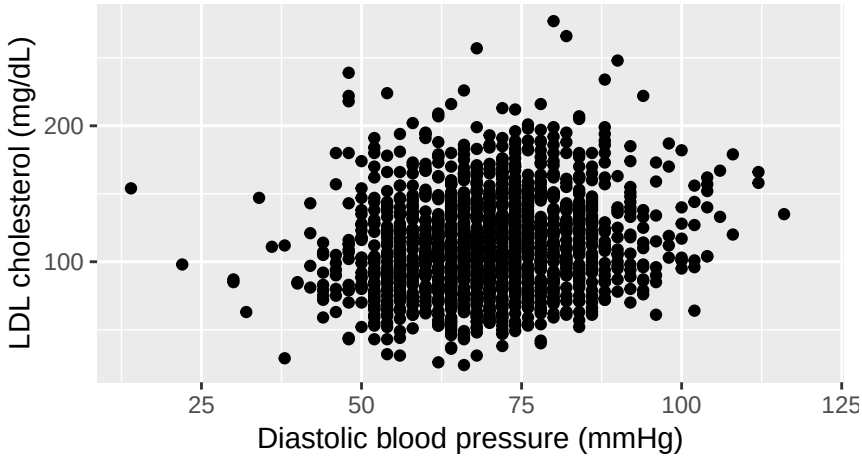
```
> table(is.na(ii$bpxdi1))

FALSE  TRUE
 4023   186
> is.na(ii$bpxdi1)<-(ii$bpxdi1<=10)
> table(is.na(ii$bpxdi1))

FALSE  TRUE
 4012   197
```

- you should, of course, consult with scientific experts and those involved in data collection prior to such recoding

Assigning missing values



Preservation of missing values

```
> x
[1]  1  2  3  4  5 NA NA
> x+3
[1]  4  5  6  7  8 NA NA
> x*2
[1]  2  4  6  8 10 NA NA
> x>=3
[1] FALSE FALSE  TRUE  TRUE  TRUE    NA    NA
```

- missing values are preserved in mathematical and logical operations

Categorization

Categorization

```
> ii$hdl3<-(ii$lbdhdd>=0)+(ii$lbdhdd>=40)+(ii$lbdhdd>=60)
> ii$hdl3<-factor(ii$hdl3,1:3)
> levels(ii$hdl3)<-c('Low','Medium','High')
> table(ii$hdl3,useNA='always')

   Low Medium   High   <NA>
   793   1976   1245    195
> sum(is.na(ii$lbdhdd))
[1] 195
```

- the first line takes advantage of the inherent numeric values of `TRUE` and `FALSE`
- notice the preservation of missing values

Categorization

- equivalently, using `with()`:

```
> ii$hdl3<-with(ii,(lbdhdd>=0)+(lbdhdd>=40)+(lbdhdd>=60))
> ii$hdl3<-factor(ii$hdl3,1:3)
> levels(ii$hdl3)<-c('Low','Medium','High')
> table(ii$hdl3,useNA='always')

   Low Medium   High   <NA>
   793   1976   1245    195
> sum(is.na(ii$lbdhdd))
[1] 195
```

- `with()` allows for omission of dollar-sign notation
- the first argument to `with()` is the data frame

Categorization

- equivalently, using `cut()`:

```
> ii$hdl3<-cut(ii$lbdhdd,breaks=c(0,40,60,250),right=FALSE)
> table(ii$hdl3,useNA='always')

  [0,40)  [40,60) [60,250)     <NA>
     793     1976     1245      195
```

- the `right=FALSE` specifies left-inclusive intervals

subset()

subset()

- subsetting refers to systematically selecting part of the data
- there are many ways to subset data in R
- the `subset()` function is one method

subset()

```
> years
[1] 2016 2017 2018 2019
> subset(years,years<=2017)
[1] 2016 2017
```

- the first argument should be the object you want to subset
- the second argument should be some logical vector

subset()

```
> years
[1] 2016 2017 2018 2019
> subset(years,1:4<=2)
[1] 2016 2017
```

- note that the second argument does not need to refer to the object being subsetted

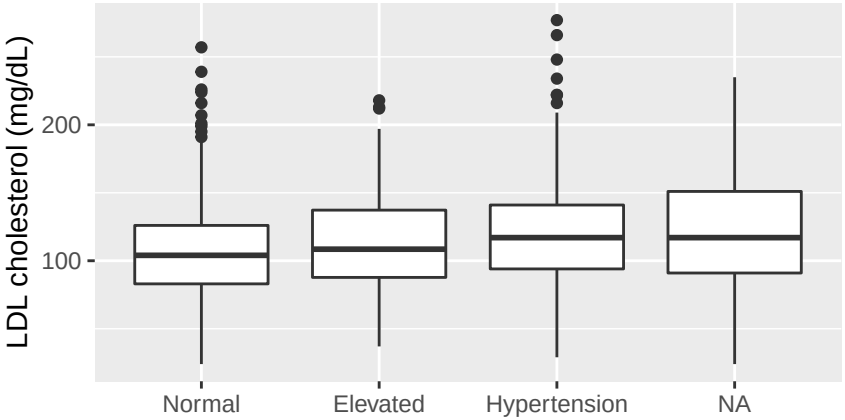
subset()

- to use subset() on data frames (two-dimensional objects):

```
> ss<-subset(ii,ridageyr<=30)
> nrow(ss)
[1] 1119
```

- note that the second argument does not require dollar-sign notation!

subset()

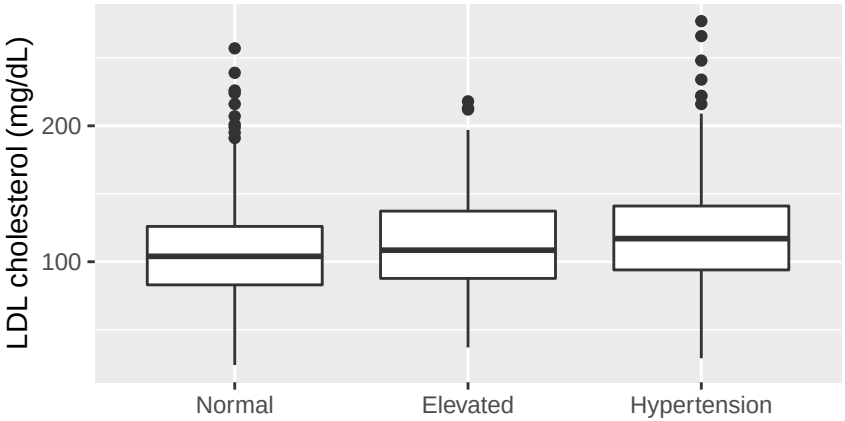


subset()

- one method of omitting the NA group from the plot is to change the data argument to ggplot():

```
gg<-ggplot(mapping=aes(x=bpcat,y=lbdldl),
           data=subset(ii,!is.na(bpcat)))
gg+
  geom_boxplot()+
  xlab('')+
  ylab('LDL cholesterol (mg/dL)')
```

subset()



subset()

```
gg+
  geom_violin(draw_quantiles=c(0.25,0.5,0.75))+
  xlab('')+
  ylab('LDL cholesterol (mg/dL)')
```

subset()

Indexing of one-dimensional objects

What is indexing?

Indexing

- indexing is another way to subset data in objects
- it is an extremely powerful method
- today we'll discuss indexing of one-dimensional objects
- next week, we'll discuss indexing of two-dimensional objects

Indexing

- indexing works by specifies indexes of the desired data
- for indexing of one-dimensional objects, the general form of the code is: `object[index]`
- the index can be a logical vector, a numeric vector, or a character vector

Using numeric vectors

Indexing using numeric vectors

```
> animals
[1] "giraffe"  "zebra"    "elephant" "lion"
> animals[2:3]
[1] "zebra"    "elephant"
```

- indexing returns the values in the positions indicated by the numeric vector

Indexing using numeric vectors

```
> animals
[1] "giraffe"  "zebra"    "elephant" "lion"
> animals[length(animals)]
[1] "lion"
> animals[3:length(animals)]
[1] "elephant" "lion"
```

Indexing using numeric vectors

```
> ii$bmxwt[1:10]
 [1]  94.8  90.4 109.8  55.2  76.6  64.5 108.3  86.2  76.2  71.2
> ii$bmxwt[seq(1,100,10)]
 [1]  94.8 117.8  75.6  60.4  71.2  61.3  50.5  73.2  49.7  73.2
```

Indexing using numeric vectors

- indexing using negative numbers:

```
> animals
[1] "giraffe"  "zebra"    "elephant" "lion"
> animals[-3]
[1] "giraffe" "zebra"   "lion"
> animals[c(-1,-length(animals))]
[1] "zebra"    "elephant"
```

Indexing using numeric vectors

- `order()` and `sort()`, revisited:

```
> animals
[1] "giraffe"  "zebra"    "elephant" "lion"
> order(animals)
[1] 3 1 4 2
> animals[order(animals)]
[1] "elephant" "giraffe"  "lion"     "zebra"
> sort(animals)
[1] "elephant" "giraffe"  "lion"     "zebra"
```

Indexing using numeric vectors

- safer re-ordering of levels:

```
> levels(ii$bmicat)
[1] "Class I obesity"              "Class II obesity"
[3] "Class III obesity"            "Overweight"
[5] "Underweight or normal weight"
> ii$bmicat<-factor(ii$bmicat,levels(ii$bmicat)[c(5,4,1,2,3)])
> levels(ii$bmicat)
[1] "Underweight or normal weight" "Overweight"
[3] "Class I obesity"              "Class II obesity"
[5] "Class III obesity"
```

Using logical vectors

Indexing using logical vectors

```
> animals
[1] "giraffe"  "zebra"    "elephant" "lion"
> animals[c(TRUE,FALSE,TRUE,FALSE)]
[1] "giraffe"  "elephant"
```

- indexing returns the values in the TRUE position
- note that the logical vector inside the brackets has the same length as the vector we are indexing
- it's possible to use a shorter indexing vector, but R will recycle the shorter vector (which, as I've mentioned, can be confusing)

Indexing using logical vectors

```
> animals
[1] "giraffe"  "zebra"    "elephant" "lion"
> is.element(animals,c('lion','kangaroo','giraffe'))
[1]  TRUE FALSE FALSE  TRUE
> animals[is.element(animals,c('lion','kangaroo','giraffe'))]
[1] "giraffe" "lion"
```

Indexing using logical vectors

```
> animals
[1] "giraffe"  "zebra"    "elephant" "lion"
> substr(animals,0,1)=='e'
[1] FALSE FALSE  TRUE FALSE
> animals[substr(animals,0,1)=='e']
[1] "elephant"
```

Indexing using logical vectors

```
> years
[1] 2016 2017 2018 2019
> years[years<=2018]
[1] 2016 2017 2018
```

Indexing using logical vectors

```
> ii$bpxdi1[!is.na(ii$bpxdi1)&(ii$bpxdi1<=30)]
[1] 22 30 30 14
```

Indexing using logical vectors

```
> ii$bmxwt[(ii$ridageyr==20)&(ii$gender=='Male')]
 [1]  77.1  70.6 153.9  67.1  93.4  63.8  93.8  62.9 103.9  84.9
[11]  63.1 121.2  91.8  62.0  63.0  84.4  81.2  61.6  90.2  70.2
[21]  75.9  63.5  54.7  66.3 106.1 155.6 101.1 162.1  74.3  79.0
[31]  61.2  59.8  61.8  97.0  81.9  74.7  68.0
```

Indexing using logical vectors

```
> mean(ii$bmxwt[ii$gender=='Female'])
[1] 78.01787
> mean(ii$bmxwt[ii$gender=='Male'])
[1] 87.90966
```

- indexing is very useful for stratified analysis

Indexing using logical vectors

```
> names(ii)[nchar(names(ii))==3]
[1] "vwa" "vra"
```

Using character vectors

Indexing using character vectors

- `quantile()`, revisited:

```
> qq<-quantile(ii$bmxwt,seq(0.1,0.9,0.1))
> names(qq)
[1] "10%" "20%" "30%" "40%" "50%" "60%" "70%" "80%" "90%"
> qq[c('60%','80%')]
 60%  80%
84.7 99.3
```

- indexing returns the values with the names indicated by the character vector

Indexing using character vectors

```
> tt<-t.test(ii$bmxbmi~ii$gender)
> tt$estimate
mean in group Female   mean in group Male
            30.24384             29.01464
> names(tt$estimate)
[1] "mean in group Female" "mean in group Male"
> tt$estimate['mean in group Female']
mean in group Female
            30.24384
```

Indexing using character vectors

```
> tt$estimate['mean in group Female']
mean in group Female
            30.24384
> tt$estimate[1]
mean in group Female
            30.24384
```

- indexing by the character vector is arguably safer
- for example, you might change the ordering of the levels, and forget that you did so
- if you did that, `tt$estimate[1]` would actually refer to males

Indexing using character vectors

```
> mm<-lm(dr1tkcal~gender+dr1tprot,data=ii)
> mm$coefficients
(Intercept)  genderMale    dr1tprot
  690.04101   186.12621    16.40565
> mm$coefficients['genderMale']
genderMale
  186.1262
```

Exercise

Exercise

You may work in groups of 2–3.

Use `presidencies.csv` for the following problems. You might choose to first explore the data using `names()` and `head()`. Note that `name` contains the names of the presidents.

1. Find all presidents from the state of Virginia.
2. The `order` variable indicates the order of each presidency. Use that variable to find the president who served the 33rd term.
3. List the start date (`start`) of all presidents from the state of Ohio.

More on the next slide...

Exercise

Continue to use `presidencies.csv` for the following problems.

4. One president served two non-consecutive terms, and is considered to have served two presidencies. Identify that president.
5. Find all presidents from the Democratic party from the state of New York.
6. Find all presidents from the Republican party from the state of Ohio.
7. Find all presidents from the state of Ohio or the state of Texas.

More on the next slide...

Exercise

Use `nhanes.csv` data for the following problems.

8. Find the mean body mass index (`bmxbmi`) among individuals with a history of smoking (`hs`). Find the mean body mass index among individuals without a history of smoking.
9. Restricting analysis to individuals 30 years of age (`ridageyr`) or younger, conduct a t-test, using body mass index as the y variable (`bmxbmi`) and history of military service (`military`) as the x variable. Hint: use `data` argument to `t.test()`.
10. How many individuals are missing a value for any (in other words, one or more) of the following 3 variables: history of smoking (`hs`), history of military service (`military`), and asthma (`asthma`)?