Programming in Data science

Factors, Missing values and Data frames

Asad Waqar Malik
asad.malik@um.edu.my

Department of Information Systems
Faculty of Computer Science and Information Technology
University of Malaya, Malaysia

Oct 03, 2019

Outline

1. Factors
   - Factors
2. Missing Values
   - Missing Values
3. Data Frames
   - Comprehensive overview on data frames
   - Data Frames Attributes
   - Access Data Frame
   - Expanding data frames
4. Built-in data frame
   - Built-in data frame
5. Data Import
   - Data Import
6. Names
7. Attributes
8. Summary

Factors

- Factors are used to represent categorical data.
- One can think of a factor as an integer vector where each integer has a label.
- Factors are important in statistical modeling.
- Using factors with labels is better than using integers because factors are self-describing.

Examples

1. Having a variable that has values “Male” and “Female” is better than a variable that has values 1 and 2.
2. A data field such as marital status may contain only values from single, married, separated, divorced, or widowed.

Factors

- Factor objects can be created with the factor(..) function.

```
> x <- factor(c("single", "married", "married", "single"));
> x
[1] single  married married single
Levels: married single
```

- Levels may be predefined even if not used.

```
> x <- factor(c("single", "married", "married", "single"),
levels = c("single", "married", "divorced"));
> x
[1] single  married married single
Levels: single married divorced
```

Factors

Sample Code

```
> x <- factor(c("single","married","married","single"))
> str(x)
Factor w/ 2 levels "married","single": 2 1 1 2
```

- Levels are stored in a character vector and the individual elements are actually stored as indices.

Access components of a factor

- Accessing components of a factor is very similar to that of vectors.

Sample Code

```
> x[3]             # access 3rd element
>  x[c(2, 4)]       # access 2nd and 4th element
> x[-1]            # access all but 1st element
> x[c(TRUE, FALSE, FALSE, TRUE)]  # using logical vector
```

Modify Factors

- Components of a factor can be modified using simple assignments.

Sample Code

```
> x
[1] single  married married single
Levels: single married divorced
> x[2] <- "divorced"    # modify second element;
> x
[1] single   divorced married  single
Levels: single married divorced
```

Modify Factors

Invalid assignment

```
> x[3] <- "widowed"    # cannot assign values outside levels
Warning message:
In factor(*tmp*, 3, value = "widowed") :
invalid factor level, NA generated
> x
[1] single   divorced <NA>     single
Levels: single married divorced
```

- However, cannot choose values outside of predefined levels.

```
> k <- factor(c("Male"))
> k[1] <- "Female"
```

Modify Factors

- A workaround to this is to add the value to the level first.

Sample Code

```
>k <- factor(c("Male"))
> k
[1] Male
Levels: Male

> levels(k) <- c(levels(k), "Female")
> k
[1] Male
Levels: Male Female
```

- Often factors will be automatically created for you when you read a dataset in using a function like read.table(..).

Missing Values

- Missing values are denoted by NA (not available) or NaN (not a number) for undefined mathematical operations.
  - NA used to represent a value that is not known, as a placeholder.
  - NA says no result was available or the result is missing. It can be used in a matrix to fill in a value of a vector.
  - NaN implies a result that cannot be calculated for whatever reason.

Example – Sqrt of a negative number

```
sqrt(-1)
[1] NaN
Warning message:
In sqrt(-1) : NaNs produced
```

Missing Values

- Following functions can be used to test objects.
  - is.na(..) is used to test objects if they are NA
  - is.nan(..) is used to test for NaN
  - NA values have a class also, so there are integer NA, character NA, etc

is

```
## Create a vector with NAs in it
x <- c(1, 2, NA, 10, 3)
## Return a logical vector indicating which elements are NA
is.na(x)
[1] FALSE FALSE TRUE FALSE FALSE
## Return a logical vector indicating which elements are NaN
is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
```

Missing Values

- **Why is.na(..) can detect NA and NaN whereas, is.nan(..) only detect NaN?**

Sample Code

```
## Now create a vector with both NA and NaN values
x <- c(1, 2, NaN, NA, 4)
is.na(x)
[1] FALSE FALSE TRUE TRUE FALSE
is.nan(x)
[1] FALSE FALSE TRUE FALSE FALSE
```

Missing Values

- Excluding Missing Values from Analyses

Sample Code

```
> x <- c(1,2,NA,3)
> mean(x)
[1] NA
> mean(x, na.rm=TRUE)
[1] 2
```

Exclude Missing Values Using complete.cases(..)

- Excluding Missing Values from a vector using complete.cases(..)

Sample Code

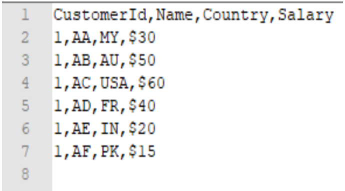
```
> vec <- c(a=c(1,2,3,4,NA), b=c(3,4,5,6,NA))
> vec
a1 a2 a3 a4 a5 b1 b2 b3 b4 b5
 1  2  3  4 NA  3  4  5  6 NA

> k <- complete.cases(vec)
> k
[1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE↩
      TRUE FALSE

> vec [k]
a1 a2 a3 a4 b1 b2 b3 b4
 1  2  3  4  3  4  5  6
```

Getting started with Data Frame

- Data frames are used to store tabular data in R.
- It is an Important type of object in R, used in a variety of statistical modeling applications.
- Unlike matrices, data frames can store different classes of objects in each column.
- Data frames have a special attribute called row.names which indicate information about each row of the data frame.

Data Frame

- Data frames are usually created by reading in a dataset using the `read.table(..)` or `read.csv(..)`.
- Data frames can also be created explicitly with the `data.frame(..)` function or they can be coerced from other types of objects like lists.
- Data frames can be converted to a matrix by calling `data.matrix(..)`.



```
CustomerId,Name,Country,Salary
1,AA,MY,$30
1,AB,AU,$50
1,AC,USA,$60
1,AD,FR,$40
1,AE,IN,$20
1,AF,PK,$15
```

Figure 1: CSV format

Creating a Data Frame

Example

```
>x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
>x
foo bar
1 TRUE
2 TRUE
3 FALSE
4 FALSE
```

```
>nrow(x)
[1] 4
>ncol(x)
[1] 2
```

Creating Data Frames – Explicitly like a list

- Creating data frame

```
> measrs <- data.frame(gender = c("M", "M","F"), ht = c(172, 186.5, 165), wt = c(91,99, 74))
> measrs
     gender   ht       wt
1    M        172.0    91
2    M        186.5    99
3    F        165.0    74
```

- Entries in a data.frame are indexed like a matrix:

```
measrs[1, 2]
```

Creating Data Frames from Vectors

- A data frame is used for storing data tables. It is a list of vectors of equal length.

Sample Code

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> df = data.frame(n, s, b)    # df is a data frame
```

Data Frame Attributes

- Accessing data frame attributes using names(..) function.

```
> names(measrs)
[1] "gender" "ht" "wt"
```

- Assign names to row.

```
> rownames(measrs) <- c("S1", "S2", "S3")
```

- Access column inside the data frame.

```
> measrs$ht
[1] 172.0 186.5 165.0
```

Data frame compenents as Vectors

- The components of a data frame can be extracted as a vector as in a list:

```
> height <- measrs$ht
> height
[1] 172.0 186.5 165.0
```

- Character vectors in a data frame are always stored as a factor. It's assumed that's what you should do.

```
> class(measrs$gend)
[1] "factor"
```

Expanding Data Frames

- Components can be added to a data frame in the natural way.

```
> measrs$age <- c(28, 55, 43)
> measrs
    gender  ht      wt age
S1  M       172.0   91 28
S2  M       186.5   99 55
S3  F       165.0   74 43
```

Expanding Data Frames

- If you expand the experiment to add data, use row binding to expand.

```
> m2 <- data.frame(gender = c("M", "F"), ht = c(170, 166), wt = c(68, 72), age = c(38, 22))
> rownames(m2) <- c("S4", "S5")
> measrs2 <- rbind(measrs, m2)
```

- If other data are kept on the same samples in another data frame it can be combined with the original using cbind.

Built-in Data Frame

- Lets explore the built-in data frames in R. For example, here is a built-in data frame in R, called mtcars.

mtcars data frame

```
> mtcars
                     mpg cyl  disp  hp drat
Mazda RX4           21.0   6 160.0 110 3.90
Mazda RX4 Wag       21.0   6 160.0 110 3.90
Datsun 710          22.8   4 108.0  93 3.85
Hornet 4 Drive      21.4   6 258.0 110 3.08
Hornet Sportabout   18.7   8 360.0 175 3.15
Valiant             18.1   6 225.0 105 2.76
Duster 360          14.3   8 360.0 245 3.21
```

- Top line of the table called the header contains the column names.
- Horizontal line afterward denotes a data row.

Retrieve data in a cell

- To retrieve data in a cell, use row and column coordinates in the single square bracket ”[]” operator, separated by a comma.
- The coordinates begins with row position, followed by comma, and ends with the column position.

Sample Code

```
> mtcars[1, 2]
[1] 6
```

- We can use the row and column names instead of the numeric coordinates.

```
> mtcars["Mazda RX4", "cyl"]
[1] 6
```

nrow and ncol functions

- Lastly, the number of data rows in the data frame is given by the nrow function.

Sample code

```
> nrow(mtcars)      # number of data rows
[1] 32
```

- And the number of columns of a data frame is given by the ncol function.

```
> ncol(mtcars)      # number of columns
[1] 11
```

Data Frame Column Vector

- We reference a data frame column with the double square bracket "[[]]" operator.
- To retrieve the ninth column vector of the built-in data set mtcars, we write mtcars[[9]].

Sample Code

```
> mtcars[[9]]     # access 9th column
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...
```

- We can retrieve the same column vector by its name.

```
> mtcars[["am"]]
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...
```

Data Frame Column Slice

- We retrieve a data frame column slice with the single square bracket "[]" operator.

Sample Code

```
> mtcars[1]
                   mpg
Mazda RX4         21.0
Mazda RX4 Wag     21.0
Datsun 710        22.8
```

- We can retrieve the same column slice by its name.

```
> mtcars["mpg"]             # access single column
> mtcars[c("mpg", "hp")]    # access multiple ↩
    column
```

Data Frame Row Slice

- Retrieve rows from a data frame with the single square bracket operator, in additional to an index vector of row positions, append an extra comma character.

```
> mtcars[24,]
             mpg cyl disp  hp drat   wt  ...
Camaro Z28 13.3   8  350 245 3.73 3.84  ...
```

- To retrieve more than one rows, use a numeric index vector.

```
> mtcars[c(3, 24),]   # reterive multiple row
> mtcars["Camaro Z28",] #retrieve a row by its name.
> mtcars[c("Datsun 710", "Camaro Z28"),]  # reterive multiple row
```

Data Import from CSV File

- CSV : The data is in comma separated format.
- The first row represents column names.

```
Col1,Col2,Col3
100,a1,b1
200,a2,b2
300,a3,b3
```

- Code to read from "mydata.csv",using `read.csv(..)`.

```
> mydata = read.csv("mydata.csv")  # read csv file
> mydata
   Col1 Col2 Col3
1  100   a1   b1
2  200   a2   b2
3  300   a3   b3
```

Working directory

- The data files must be located in the R working directory, which can be found and set with the following functions.

```
> getwd()                # get current working ↩
    directory
> setwd("<new path>")    # set working directory
> setwd("C:/MyDoc")      # Note that the forward ↩
    slash should be used as the path separator even↩
     on Windows platform.
```

Names

- R objects can have names, which is very useful for writing readable code and self-describing objects.

Sample Code

```
> x <- 1:3
> names(x)
NULL
> names(x) <- c("New York", "Seattle", "Los Angeles")
> x
New York Seattle Los Angeles
1          2       3
>names(x)
[1] "New York" "Seattle" "Los Angeles"
```

Names

- Lists can also have names, which is often very useful.

Example

```
> x <- list("Los Angeles" = 1, Boston = 2, London = 3)
> x
$ Los Angeles
[1] 1
$Boston
[1] 2
$London
[1] 3

>names(x)
[1] "Los Angeles" "Boston" "London"
```

Names

- Matrices can have both column and row names.

Example

```
> m <- matrix(1:4, nrow = 2, ncol = 2)
> dimnames(m) <- list(c("a", "b"), c("c", "d"))
> m
  c d
a 1 3
b 2 4
```

Names in Data frame

- for data frames, there is a separate function for setting the row names, the row.names() function.
- data frames do not have column names, they just have names (like lists).
- to set the column names of a data frame just use the names() function.

Object	Set column names	Set row names
data frame	names()	row.names()
matrix	colnames()	rownames()

Table 1: Quick Summary.

Attributes

- R objects can have attributes, which are like metadata for the object.
- metadata can be very useful in that they help to describe the object.
- For example, column names on a data frame help to tell us what data are contained in each of the columns.

Example

- names, dimnames
- dimensions (e.g. matrices, arrays)
- class (e.g. integer, numeric)
- length
- other user-defined attributes/metadata

Reading from Excel

```
> install.packages("readxl")
> library("readxl")
> my_data <- read_excel("C:/myfile.xlsx")
```

```
#Select file through browsing
>my_data <- read_excel(file.choose())
```

```
# Specify sheet by its name
my_data <- read_excel("my_file.xlsx", sheet = "data")
```

Summary

There are a variety of different builtin-data types in R. In this section we have reviewed the following.

- atomic classes: numeric, logical, character, integer, complex
- vectors, lists
- factors
- missing values
- data frames and matrices

Practise questions

⇒ Create the following data frame, afterwards invert Sex for all individuals.

```
           Age  Height  Weight  Sex
Alex         25   177     57     F
Lily         20   165     69     F
Mark         21   145     70     M
Oliver         24   165     65     M
Martha         22   175     71     F
```

⇒ How many rows and columns does the new data frame have?
⇒ What class of data is in each column?

Exc

⇒ Create this data frame

```
    Working
Alex    YES
Lily    NO
Mark    YES
Oliver  NO
Martha  YES
```

⇒ Add this data frame column-wise to the previous one.

⇒ How many rows and columns does the new data frame have?

⇒ What class of data is in each column?

Exc

⇒ Check what class of data is the (built-in data set) state.center and convert it to data frame. Hine, see as.data.frame?