WIFI

Wifi Name: AISG
Password: AISG@2018!NUS

R for data cleaning

Download materials here:
https://tinyurl.com/dssg-intro-to-r

Lin Chun

Agenda

1. Introduction & Set up (Live Demo)
2. Navigating in Jupyter notebook (Live Demo)
3. What is R?
4. R basics
5. Dataframe operations
6. Data Cleaning

What is R?

1. R is an open source development tool that is supported by a large community of statisticians and computer scientists. It has over 4000 packages that implements various statistical analysis tools related to hypothesis testing, model fitting, clustering techniques, and machine learning.

   https://www.quora.com/What-is-R-programming-language

2. R is usually paired with RStudio (Graphical interface) for usage

Data types

1. R has 6 atomic data types:
    a. character
    b. numeric (real or decimal)
    c. integer
    d. logical
    e. Complex (not covered)
2. By ***atomic***, we mean the vector only holds data of a single type

Numeric & Integer

1. Numeric

```
> x <- 1
> x
[1] 1
> class(x)
[1] "numeric"
> typeof(x)
[1] "double"
```

```
> is.integer(x)
[1] FALSE
```

```
> y
[1] 1.2
> class(y)
[1] "numeric"
> typeof(y)
[1] "double"
```

2. Integer

```
> y <- as.integer(3)
> y
[1] 3
> class(y)
[1] "integer"
> typeof(y)
[1] "integer"
> is.integer(y)
[1] TRUE
```

Character & Logical

1. Character data types

```
> z <- 'hello'
> w <- 'world'
> z
[1] "hello"
> w
[1] "world"
> typeof(z)
[1] "character"
> typeof(w)
[1] "character"
```

2. Logical Data types

```
> v <- TRUE
> w <- FALSE
> class(v)
[1] "logical"
> class(w)
[1] "logical"
```

Basic operations

1. Arithmetic operations
   a. +
   b. -
   c. /
   d. *
   e. % - modulo

```
> x <- 5
> y <- 2
> x + y
[1] 7
> x - y
[1] 3
> x * y
[1] 10
> x / y
[1] 2.5
> x %% y
[1] 1
```

2. Relational operations
   a. ==
   b. !=
   c. <
   d. >
   e. &
   f. ||

```
> x == y
[1] FALSE
> x != y
[1] TRUE
> x < y
[1] FALSE
> x > y
[1] TRUE
> x & y
[1] TRUE
> x || y
[1] TRUE
```

Try it out!

1. V == 1
2. ‘Hello’ * 3
    a. What does the statement evaluate to?
3. Try the following statements:

    a <- as.integer(3)

    b <- a * 3

    Before you check the typeof(b), what do you think the type of b is?

Commenting of code

What is code commenting?

It is typing out lines to convey what **your code does** in ‘English’

Comments are specially marked lines of text in your program that is not evaluated.

Why do we comment code?

1. For reproducibility
    a. You want to let others understand what your code does without scrutinizing your code
    b. You want to remember what your own code does

Best practices

1. Comment what your code does before you actually code

Data Structures

1. Vector
2. Matrix
3. Dataframe

Vectors

1. A vector is a collection of elements that are most commonly of mode character, logical, integer or numeric.
2. How do you create a vector?

   type(length)

   Examples

```
> numeric(5)
[1] 0 0 0 0 0
> character(7)
[1] "" "" "" "" "" "" ""
> logical(10)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Vectors

1. How do you create a vector?

   c(insert elements here, separated by a comma)

   c(1,2,3,4,5)

   Examples

```
> number_vector <- c(1,2,3,4,5)
> number_vector
[1] 1 2 3 4 5
> char_vector<-c('A',1,'TWO',TRUE)
> char_vector
[1] "A"    "1"    "TWO"  "TRUE"
```

   Vector does not support collection of different types, will perform coercion

Vectors

1. Adding to vectors

   c(vector, 'new element')

```
> new_number_vector<-c(number_vector,20)
> new_number_vector
[1]  1  2  3  4  5 20
```

2. Accessing vectors

```
> new_number_vector<-c(6,2,3,1,33,55)
> new_number_vector
[1]  6  2  3  1 33 55
> new_number_vector[3]
[1] 3
> new_number_vector[2]
[1] 2
> new_number_vector[1:4]
[1] 6 2 3 1
> new_number_vector[-1]
[1]  2  3  1 33 55
> new_number_vector[-4]
[1]  6  2  3 33 55
```

Matrix

1. Extension of the numeric or character vectors with multiple dimensions (row & col)

2. Constructing a matrix:

```
> m <- matrix(nrow = 2, ncol = 2)
> m
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
```

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrix

1. Constructing a matrix:

```
> mdat <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol = 3, byrow = TRUE)
> mdat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   11   12   13
```

2. Accessing a matrix:

```
> mdat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   11   12   13
> row = 1
> col = 1
> mdat[row,col]
[1] 1
> mdat[row,]
[1] 1 2 3
> mdat[,col]
[1]  1 11
> mdat[1,2]
[1] 2
```

Dataframe

1. Is the de-facto data structure to use when cleaning data, performing analysis etc in R
2. The structure is very similar to a matrix

Dataframe Operations

1. Creating a dataframe
    a.
```
> dat <- data.frame(first_col = 1:10, second_col = 11:20)
> dat
   first_col second_col
1          1         11
2          2         12
3          3         13
4          4         14
5          5         15
6          6         16
7          7         17
8          8         18
9          9         19
10        10         20
```

    b. Reading in from a source (csv):

```
df<-read.csv('iris.csv')
```

Dataframe Operations

1. Common functions:
    a. **head()** - shows first 6 rows
    b. **tail()** - shows last 6 rows
    c. **nrow()** - number of rows
    d. **ncol()** - number of columns
    e. **dim()** - returns the dimensions of data frame (i.e. number of rows and number of columns)
    f. **str()** - structure of data frame - name, type and preview of data in each column

```
> dim(df)
[1] 150   5
> str(df)
'data.frame':   150 obs. of  5 variables:
 $ sepal_length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ sepal_width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ petal_length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ petal_width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Try it out!

1. Read in the csv provided in the same folder
2. After reading in the csv, observe the types of the columns you've read in:
    a. str(df)
    b. Note: we commonly use df as a variable name instead of dataframe because dataframe is a reserved keyword in R and this may cause some confusion.
3. What are the types and what do you notice for the column 'Species'

Coercion and Casting

1. as.integer(3) - casting a double ‘3.0’ to an integer 3
2. Coercion (implicit) by read_csv function to convert string values into factors
3. On Factors
    a. Factors are basically groupings made by R to group strings into categories, it allows for efficient processing with R, however may not be what we intend to have.
    b. To prevent the coercion:
        i. read_csv(’iris.csv’, stringsAsFactors=FALSE)
4. Commonly coerced types:
    a. String -> Date: as.date(‘String’)

Dataframe Operations

1. Accessing dataframe elements

   - Similar to accessing elements in a matrix

   There are multiple ways to access element(s) in a matrix:

   Single element

   1. Access by name:
      a. *df$Sepal.Length[20]*
      b. *df[20, 'Sepal.Length']*
   2. Access by index:
      a. *df[20,1]*

Dataframe Operations

1. Accessing dataframe elements

   - Similar to accessing elements in a matrix

   There are multiple ways to access element(s) in a matrix:

   Entire column

   1. Access by name:
      a. *df$Sepal.Length*
      b. *df[, 'Sepal.Length']*
   2. Access by index:
      a. *df[,1]*

Dataframe Operations

1. Accessing dataframe elements

   - Similar to accessing elements in a matrix

   There are multiple ways to access element(s) in a matrix:

   Multiple columns

   1. Access by name:
      a. *df[, c('Sepal.Length', 'Petal.Width')]*
   2. Access by index:
      a. *df[,c(1,4)]*

Dataframe Operations

1. Accessing dataframe elements

   - Similar to accessing elements in a matrix

   There are multiple ways to access element(s) in a matrix:

   Entire row(s)

   1. Access by index:
      a. *df[20,]*
      b. *df[c(20,21,30:35,36),] - gives you row 20,21,30 to 35 and 36*

Dataframe Operations

1. Mutating data frame elements

- Dataframes are mutable, hence you can change the values simply by assigning another value over it

Mutating value

a. *df[20, 'Sepal.Length'] <- 7.0*
b. *df[20,1] <- 7.0*

Try it out!

1. With the iris.csv read in as a dataframe:
    a. Find the 10th row and 4th column value
    b. List out all the the 2nd, 4th, 7th and 9th rows only
    c. List out the last row only
    d. Add +1 cm to all the values of ‘Sepal.Length’
    e. Change all of the values of types of iris to uppercase values

Dataframe Operations

1. What if you wanted to access only rows that have sepal length of > 5?

   - Two step process

   1. Check which rows have sepal length of > 5
      a. ***df$Sepal.Length**>5*
      b. *df[,1]>5*

   This will return a vector of true/false values which serves to indicate whether the element is > 5 (true) or not (not)

   2. Return rows where sepal length is greater than 5
      a. *df[**df$Sepal.Length>5**,]*
      b. *df[df[,1]>5]*

   This will return the rows where the vector is true

Dataframe Operations

1. How do you get the specific position(s) of iris that have > 5.2 sepal length?
    a. which(df$Sepal.Length>5.2)
2. How do we calculate the **average/mean**, **max**, **min** of the sepal length?
    a. mean(df$Sepal.Length)
    b. max(df$Sepal.Length)
    c. min(df$Sepal.Length)
3. How do we get the summary statistics of the dataframe?
    a. summary(df)

```
In [345]: summary(df)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :6.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:7.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :7.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :7.843   Mean   :3.054   Mean   :3.759   Mean   :1.199
 3rd Qu.:8.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :9.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
           Species
 IRIS-SETOSA    :50
 IRIS-VERSICOLOR:50
 IRIS-VIRGINICA :50
```

Dataframe Operations

1. How do we deal with NA values?
    a. First, we need to determine if we care about the NA values.
        i. Finding out NAs
        ii. Omitting NAs
        iii. Complete cases
2. Finding out NAs
    a. ***which(is.na(df$Sepal.Length))*** *- will find out all the rows in the Sepal Length Column that contains NA values*
3. Omitting NAs
    a. ***na.omit(df)*** *- returns with a special dataframe with all the rows (that has NA) omitted*
4. Complete cases
    a. ***df[complete.cases(df),]*** *- returns a dataframe containing a rows which are complete cases - no NAs*

Dataframe Operations

1. Difference between na.omit and complete.cases?
    a. Na.omit adds an additional information attribute which lets you know which rows were removed
    b. This may be cumbersome in some cases
    c. Complete cases provides better control

```
In [356]: df_na_omitted <- na.omit(df_2)

In [357]: str(df_na_omitted)

'data.frame':   134 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5.4 4.6 4.4 4.9 5.4 4.8 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.9 3.4 2.9 3.1 3.7 3.4 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.7 1.4 1.4 1.5 1.5 1.6 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.1 0.2 0.2 ...
 $ Species     : Factor w/ 3 levels "Iris-setosa$",..: 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "na.action")=Class 'omit'  Named int [1:16] 5 8 22 37 44 45 49 62 71 78 ...

In [399]: df_complete_cases <- df_2[complete.cases(df_2),]

In [400]: str(df_complete_cases)

'data.frame':   147 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5.8 5.4 4.6 5.8 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "Iris-setosa$",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Dataframe Operations

1. Filling in missing data
    a. Most of the time, you would not want to just delete/omit your missing data
    b. You can actually fill it in with some heuristics such as ‘filling missing values with the mean,median,mode’
2. But how?
    a. 2 step process:
        i. Find out where the missing values are located at
        ii. Find out what the mean of the column is
        iii. Replace the missing value with the mean
        iv. <u>Try it!</u>

Try it out!

1. Read in **‘iris_2.csv’** as df_2.
2. Which of the following rows are empty in ‘Sepal.Length’
3. Challenge:
    a. Fill empty rows of ‘Sepal.Length’ with the median value
    b. Fill empty rows of ‘Sepal.Width’ with the mean value

String manipulation

In data cleaning, you will often see strings that aren’t clean.

Such as:

- prefix values: 0001-setosa
- postfix values: setosa-0001
- dirty data: setosa$!

How do we clean them?

Using regular expression or REGEX for short.

String manipulation

For text, we usually default to using the function gsub:

gsub(pattern, replacement, string)

- prefix values: 0001/setosa

pattern: 0001/

replacement: “”

string:0001/setosa

gsub(“0001/”, “”, “0001/setosa”)

String manipulation

What happens if the prefix keeps changing, but has a consistent theme being that it is non-alphabetical

gsub(pattern, replacement, string)

- prefix value 1 : 0001/setosa
- prefix value 2 : 0002/versicolor

gsub("[^[:alpha:]]","",a)

Find out more here
http://www.endmemo.com/program/R/gsub.php

R series

R for data cleaning

Thank you!