Week 3 Demonstration

Understand: Data Types and Data Structures

Dr. Anil Dolgun

14/03/2018

1 / 41

Course Announcements

Course Announcements

- First round of assessment is due on 25/03/2018. Details are available here.

Course Announcements

- First round of assessment is due on 25/03/2018. Details are available here.
- Suggested solutions to Week 2 Worksheet are available here

Course Announcements

- First round of assessment is due on 25/03/2018. Details are available here.
- Suggested solutions to Week 2 Worksheet are available here
- Answers to Module 2 Skill Builders are available here

Course Announcements

- First round of assessment is due on 25/03/2018. Details are available here.
- Suggested solutions to Week 2 Worksheet are available here
- Answers to Module 2 Skill Builders are available here
- Any other questions or concerns?

Course Announcements

- First round of assessment is due on 25/03/2018. Details are available here.
- Suggested solutions to Week 2 Worksheet are available here
- Answers to Module 2 Skill Builders are available here
- Any other questions or concerns?
- Please take a few minutes to provide feedback on the course. If you are happy, let me know. If you have some helpful ideas to improve the course, even better!

Course Announcements

- First round of assessment is due on 25/03/2018. Details are available here.
- Suggested solutions to Week 2 Worksheet are available here
- Answers to Module 2 Skill Builders are available here
- Any other questions or concerns?
- Please take a few minutes to provide feedback on the course. If you are happy, let me know. If you have some helpful ideas to improve the course, even better!
- Please fill out this form (Available from Tools --> Course Feedback on the website) at any stage of the semester.

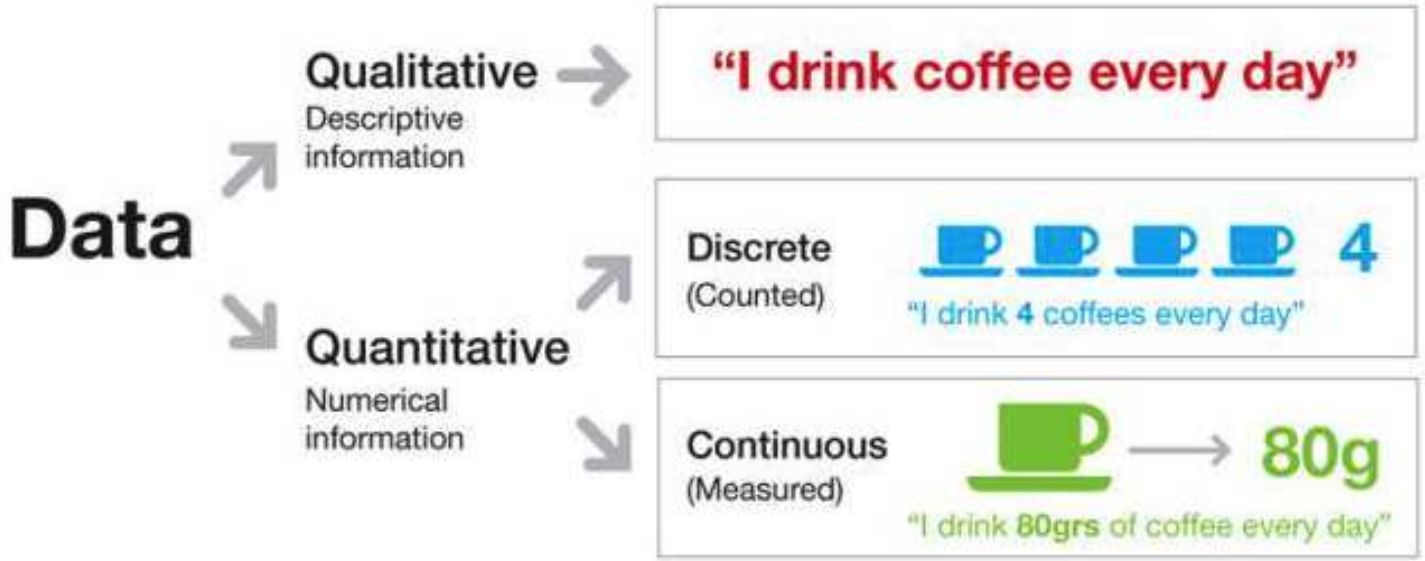
Understand: Data Types and Data Structures in R

3 / 41

Module 3: Outline

- Data types in general
  - Qualitative (categorical)
    - Nominal variable
    - Ordinal variable
  - Quantitative (numerical)
    - Continuous variable
    - Discrete variable
- Data types in R
  - character
  - numeric
  - integer
  - factor
  - logical
- R's data structures
  - vector
  - list
  - matrix
  - data frame
- Converting Data Types/Structures
  - `is.` functions
  - `as.` functions

Preliminaries: Data types in a general sense



Data Types in R

- R is a programming language, it has own definitions of data types and structures.
- Technically, R classifies all the different types of data into four classes:
  -
  - (integer or double)
  -
  -
- Useful functions in R:
  - Use `class()` to check the class of an object
  - Use `typeof()` to check whether a numeric object is integer or double
  - Use `levels()` to see the levels of a factor object

Data Types in R: Logical class

- class consists of TRUE or FALSE (binary) values.
- A logical value is often created via comparison between variables.

```
x <- 10
y <- (x > 0)
y
```

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

```
class(y)
```

```
## [1] "logical"
```

Data Types in R: Numeric Class

- (integer or double): Quantitative values are called as numerics in R.
- It is the default computational data type.
- Numeric class can be integer or double.
- Integer types can be seen as discrete values (e.g., 2) whereas, double class will have floating point numbers (e.g., 2.16).
- To create a double numeric variable:

```
var1 <- c(4, 7.5, 14.5)
```

- To create an integer variable, place an `L` directly after each number:

```
var2 <- c(4L, 7L, 14L)
```

Data Types in R: Numeric Class

```
var1 <- c(4, 7.5, 14.5)

var2 <- c(4L, 7L, 14L)
```

- To check the class of numeric variable:

```
class(var1)
```

```
## [1] "numeric"
```

```
class(var2)
```

```
## [1] "integer"
```

- To check whether an object is integer or double, use `typeof()`.

```
typeof(var1)
```

```
## [1] "double"
```

```
typeof(var2)
```

```
## [1] "integer"
```

Data Types in R: Character Class

- : A character class is used to represent string values in R.
- The most basic way to generate a character object is to use quotation marks `" "` and assign a string/text to an object:

```
var3 <- c("debit", "credit", "Paypal")

class(var3)
```

```
## [1] "character"
```

Data Types in R: Factor Class

- class is used to represent qualitative data in R.
- Factors can be ordered or unordered.
- They store the nominal values as a vector of integers in the range (where is the number of unique values in the nominal variable), and an internal vector of character strings (the original values) mapped to these integers.
- Factor objects can be created with the `factor()` function:

```
var4 <- factor( c("Male", "Female", "Male", "Male") )

var4
```

```
## [1] Male   Female Male   Male
## Levels: Female Male
```

```
class(var4)
```

```
## [1] "factor"
```

Data Types in R: Factor Class Cont.

- To see the levels of a factor object `levels()` function will be used:

```
levels(var4)
```

```
## [1] "Female" "Male"
```

- By default, levels of the factors will be ordered alphabetically.

Data Types in R: Factor Class Cont.

- To see the levels of a factor object `levels()` function will be used:

```
levels(var4)
```

```
## [1] "Female" "Male"
```

- By default, levels of the factors will be ordered alphabetically.
- Using the `levels()` argument, we can control the ordering of the levels while creating a factor:

```
var5 <- factor( c("Male", "Female", "Male", "Male"),
                    levels = c("Male", "Female") )
var5
```

```
## [1] Male   Female Male   Male
## Levels: Male Female
```

```
levels(var5)
```

```
## [1] "Male"   "Female"
```

Data Types in R: Ordered Factor Class

- We can also create ordinal factors in a specific order using the `ordered = TRUE` argument:

```
var6 <- factor(c("DI", "HD", "PA", "NN", "CR", "DI", "HD", "PA"), lev
    "PA", "CR", "DI", "HD"), ordered = TRUE)

var6
```

```
## [1] DI HD PA NN CR DI HD PA
## Levels: NN < PA < CR < DI < HD
```

- The ordering will be reflected as `NN < PA < CR < DI < HD` in the output.

Data Structures in R

- A data set is a collection of measurements or records which can be in any class (i.e., logical, character, numeric, factor, etc.).
- Typically, data sets contain many variables of different length and type of values.
- In R, we can store data sets using vectors, lists, matrices and data frames and these are called .
- R's base data structures can be organised by their (i.e., 1-D, 2-D, or n-D) and whether they're (i.e., all variables must be of the same type) or (i.e., variables can be of different types):

one-dimension	Atomic vector	List
two-dimension	Matrix	Data frame
n-dimension	Array	--

Data Structures in R: Vectors

- A vector is the basic structure in R, which consists of one-dimensional sequence of data elements of the same basic type (i.e., integer , double , logical, or character).
- Vectors are created by combining multiple elements into one dimensional array using the combine `c()` function.
- The one-dimensional examples illustrated previously are considered vectors:

```
var1 <- c(4, 7.5, 14.5) # a double numeric vector

var2 <- c(4L, 7L, 14L) # an integer vector

var3 <- c(T, F, T, T) # a logical vector
```

Data Structures in R: Vectors Cont.

- All elements of a vector must be the same type, if you attempt to combine different types of elements they will be coerced to the most flexible type possible.

Data Structures in R: Vectors Cont.

- All elements of a vector must be the same type, if you attempt to combine different types of elements they will be coerced to the most flexible type possible.

Vector of characters + numerics:

```
ex1 <- c("a", "b", "c", 1, 2, 3)
```

Vector of numerics + logical:

```
ex2 <- c(1, 2, 3, TRUE, FALSE)
```

Vector of logical + characters:

```
ex3 <- c(TRUE, FALSE, "a", "b",
```

Data Structures in R: Vectors Cont.

- All elements of a vector must be the same type, if you attempt to combine different types of elements they will be coerced to the most flexible type possible.

Vector of characters + numerics:

```
ex1 <- c("a", "b", "c", 1, 2, 3)
```

--> a character vector

```
## [1] "character"
```

Vector of numerics + logical:

```
ex2 <- c(1, 2, 3, TRUE, FALSE)
```

--> a numeric vector

```
## [1] "numeric"
```

Vector of logical + characters:

```
ex3 <- c(TRUE, FALSE, "a", "b",
```

--> a character vector

```
## [1] "character"
```

Data Structures in R: Vectors Cont.

- To add additional elements to a vector use `c()` function.
- Let's add two elements (4 and 6) to the `ex2` vector:

```
ex4 <- c(ex2, 4, 6)

ex4
```

```
## [1] 1 2 3 1 0 4 6
```

Data Structures in R: Vectors Cont.

- To subset a vector, we can use square brackets `[ ]` with positive or negative integers, logical values or names.

```
ex4
```

```
## [1] 1 2 3 1 0 4 6
```

Take the third element `ex4`:

```
ex4[3]
```

```
## [1] 3
```

Take first three elements in `ex4`:

```
ex4[1:3]
```

```
## [1] 1 2 3
```

Data Structures in R: Vectors Cont.

- To subset a vector, we can use square brackets `[ ]` with positive or negative integers, logical values or names.

```
ex4
```

```
## [1] 1 2 3 1 0 4 6
```

Take the third element `ex4`:

```
ex4[3]
```

```
## [1] 3
```

Take first three elements in `ex4`:

```
ex4[1:3]
```

```
## [1] 1 2 3
```

Take the 1st, 3rd, and 5th element:

```
ex4[c(1,3,5)]
```

```
## [1] 1 3 0
```

Take all elements except first:

```
ex4[-1]
```

```
## [1] 2 3 1 0 4 6
```

Take all elements less than 3:

```
ex4[ ex4 < 3 ]
```

```
## [1] 1 2 1 0
```

Data Structures in R: Lists

- A list is an R structure that allows you to combine elements of different types and lengths.
- In order to create a list we can use the `list()` function.

```
list1 <- list(1:3, "a", c(TRUE, FALSE, TRUE), c(2.5, 4.2))
```

- To see the detailed structure within an object use the structure function `str()`:

```
str(list1)
```

```
## List of 4
##  $ : int [1:3] 1 2 3
##  $ : chr "a"
##  $ : logi [1:3] TRUE FALSE TRUE
##  $ : num [1:2] 2.5 4.2
```

- Note how each of the four list items above are of different classes (integer, character, logical, and numeric) and different lengths.

Data Structures in R: Lists Cont.

- To add on to lists we can use the `append()` function. Let's add a fifth element to the `list1` and store it as `list2`:

```
list2 <- append(list1, list(c("credit", "debit", "Paypal")))

str(list2)
```

```
## List of 5
##  $ : int [1:3] 1 2 3
##  $ : chr "a"
##  $ : logi [1:3] TRUE FALSE TRUE
##  $ : num [1:2] 2.5 4.2
##  $ : chr [1:3] "credit" "debit" "Paypal"
```

A remark : Attributes

- R objects can also have attributes, which are like metadata for the object.
- These metadata can be very useful in that they help to describe the object. Some examples of R object attributes are:
    - names, dimnames
    - dimensions (e.g. matrices, arrays)
    - class (e.g. integer , numeric)
    - length
    - other user-defined attributes/metadata
- Attributes of an object (if any) can be accessed using the `attributes()` function. Let's check if `list2` has any attributes.

```
attributes(list2)
```

```
## NULL
```

Data Structures in R: Lists Cont.

- We can add names to lists using `names()` function.

```
# add names to a pre-existing list

names(list2) <- c ("item1", "item2", "item3", "item4", "item5")

str(list2)
```

```
## List of 5
##  $ item1: int [1:3] 1 2 3
##  $ item2: chr "a"
##  $ item3: logi [1:3] TRUE FALSE TRUE
##  $ item4: num [1:2] 2.5 4.2
##  $ item5: chr [1:3] "credit" "debit" "Paypal"
```

- Now, you can see that each element has a name and the names are displayed after a dollar `$` sign.

Data Structures in R: Lists Cont.

- In order to subset lists, we can use dollar `$` sign, square brackets `[ ]` or double square brackets `[[ ]]`:

```
list2[1]  # take the first list item in list2
```

```
## $item1
## [1] 1 2 3
```

```
list2[[1]]  # take the first list item in list2 without attributes
```

```
## [1] 1 2 3
```

```
list2$item1 # take the first list item in list2 using $
```

```
## [1] 1 2 3
```

```
list2$item5[3] # take the third element out of fifth list item
```

```
## [1] "Paypal"
```

Data Structures in R: Matrices

- A matrix is a collection of data elements arranged in a two-dimensional rectangular layout.
- In R, the elements of a matrix must be of same class (i.e. all elements must be numeric, or character, etc.) and all columns of a matrix must be of same length.
- We can create a matrix using the `matrix()` function using `nrow` and `ncol` arguments.

```
m1 <- matrix(1:6, nrow = 2, ncol = 3)

m1
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Data Structures in R: Matrices Cont.

- Matrices can also be created using the column-bind `cbind()` and row-bind `rbind()` functions.
- Note that the vectors that are being binded must be of equal length and mode.

```
v1 <- c( 1, 4, 5)
v2 <- c( 6, 8, 10)
```

```
# create a matrix using column-b
m2 <- cbind(v1, v2)
m2
```

```
##      v1 v2
## [1,]  1  6
## [2,]  4  8
## [3,]  5 10
```

```
# create a matrix using row-binc
m3 <- rbind(v1, v2)
m3
```

```
##    [,1] [,2] [,3]
## v1    1    4    5
## v2    6    8   10
```

Data Structures in R: Matrices Cont.

- We can also use `cbind()` and `rbind()` functions to add onto matrices.

```
v3 <- c(9, 8, 7)

m4 <- rbind(m3, v3)
m4
```

```
##    [,1] [,2] [,3]
## v1    1    4    5
## v2    6    8   10
## v3    9    8    7
```

Data Structures in R: Matrices Cont.

- We can add names to the rows and columns of a matrix using `rownames` and `colnames`.

```
rownames(m4) <- c("subject1", "subject2", "subject3")

colnames(m4) <- c("var1", "var2", "var3")

attributes(m4)
```

```
## $dim
## [1] 3 3
##
## $dimnames
## $dimnames[[1]]
## [1] "subject1" "subject2" "subject3"
##
## $dimnames[[2]]
## [1] "var1" "var2" "var3"
```

Data Structures in R: Matrices Cont.

- In order to subset matrices we use the `[ ]` operator.
- As matrices have two dimensions we need to specify subsetting arguments for both row and column dimensions like: matrix [rows, columns]:

```
m4
```

```
##          var1 var2 var3
## subject1    1    4    5
## subject2    6    8   10
## subject3    9    8    7
```

```
m4[1,2] # take the value in the first row and second column
```

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

Data Structures in R: Matrices Cont.

```
m4[1:2, ] # subset for rows 1 and 2 but keep all columns
```

```
##          var1 var2 var3
## subject1    1    4    5
## subject2    6    8   10
```

```
m4[ , c(1, 3)] # subset for columns 1 and 3 but keep all rows
```

```
##          var1 var3
## subject1    1    5
## subject2    6   10
## subject3    9    7
```

```
m4[1:2, c(1, 3)] # subset for both rows and columns
```

```
##          var1 var3
## subject1    1    5
## subject2    6   10
```

Data Structures in R: Data Frames

- The most common way of storing data in R and, generally, is the data structure most often used for data analyses.
- A data frame (DF) is a list of equal-length vectors and they can store different classes of objects in each column (i.e., numeric, character, factor).
- DFs are usually created by importing/reading in a data set using the functions covered in Module 2.
- Can also be created explicitly with the `data.frame()` function or they can be coerced from other types of objects like lists.

Data Structures in R: Data Frames Cont.

```
df1 <- data.frame (col1 = 1:3,
                   col2 = c ("credit", "debit", "Paypal"),
                   col3 = c (TRUE, FALSE, TRUE),
                   col4 = c (25.5, 44.2, 54.9))
str(df1)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ col1: int  1 2 3
##  $ col2: Factor w/ 3 levels "credit","debit",..: 1 2 3
##  $ col3: logi  TRUE FALSE TRUE
##  $ col4: num  25.5 44.2 54.9
```

Data Structures in R: Data Frames Cont.

- With `stringsAsFactors = FALSE` argument:

```
df1 <- data.frame (col1 = 1:3,
                   col2 = c ("credit", "debit", "Paypal"),
                   col3 = c (TRUE, FALSE, TRUE),
                   col4 = c (25.5, 44.2, 54.9),
                   stringsAsFactors = FALSE)

str(df1)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ col1: int  1 2 3
##  $ col2: chr  "credit" "debit" "Paypal"
##  $ col3: logi  TRUE FALSE TRUE
##  $ col4: num  25.5 44.2 54.9
```

Data Structures in R: Data Frames Cont.

- We can add columns (variables) and rows (items) on to a data frame using `cbind()` and `rbind()` functions:

```
# create a new vector

v4 <- c("VIC", "NSW", "TAS")

# add a column (variable) to df1

df2 <- cbind(df1, v4)
```

Data Structures in R: Data Frames Cont.

- To add attributes to data frames we use `rownames()` and `colnames()`

```
rownames(df2) <- c("subj1", "subj2", "subj3") # add row names

colnames(df2) <- c("number", "card_type", "fraud", "transaction", "st

str(df2)
```

```
## 'data.frame':    3 obs. of  5 variables:
##  $ number     : int  1 2 3
##  $ card_type  : chr  "credit" "debit" "Paypal"
##  $ fraud      : logi  TRUE FALSE TRUE
##  $ transaction: num  25.5 44.2 54.9
##  $ state      : Factor w/ 3 levels "NSW","TAS","VIC": 3 1 2
```

```
attributes(df2)
```

```
## $names
## [1] "number"      "card_type"   "fraud"       "transaction" "state"
##
## $row.names
```

Data Structures in R: Data Frames Cont.

- Data frames possess the characteristics of both lists and matrices.
- If you subset with a single vector, they behave like lists and will return the selected columns with all rows and if you subset with two vectors, they behave like matrices and can be subset by row and column.

```
df2
```

```
##       number card_type fraud transaction state
## subj1      1    credit  TRUE        25.5   VIC
## subj2      2     debit FALSE        44.2   NSW
## subj3      3    Paypal  TRUE        54.9   TAS
```

```
df2[2:3, ] # subset by row numbers, take second and third rows only
```

```
##       number card_type fraud transaction state
## subj2      2     debit FALSE        44.2   NSW
## subj3      3    Paypal  TRUE        54.9   TAS
```

Data Structures in R: Data Frames Cont.

```
df2[c("subj2", "subj3"),  ] # same as above but uses row names
```

```
##       number card_type fraud transaction state
## subj2      2     debit FALSE        44.2   NSW
## subj3      3    Paypal  TRUE        54.9   TAS
```

```
df2[, c(1,4)] # subset by column numbers, take first and forth colum
```

```
##       number transaction
## subj1      1        25.5
## subj2      2        44.2
## subj3      3        54.9
```

```
df2[, c("number", "transaction")] # same as above but uses column nar
```

```
##       number transaction
## subj1      1        25.5
## subj2      2        44.2
## subj3      3        54.9
```

```
df2[2:3, c(1, 4)] # subset by row and column numbers
```

```
##       number transaction
## subj2      2        44.2
## subj3      3        54.9
```

```
df2[c("subj2", "subj3"), c("number", "transaction")] # same as above
```

```
##       number transaction
## subj2      2        44.2
## subj3      3        54.9
```

```
df2$fraud # subset using $: take the column (variable) fraud
```

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

```
df2$fraud[2] # take the second element in the fraud column
```

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

Converting Data Types/Structures

- `as.` functions will convert the object to a given type (whenever possible) and `is.` functions will test for the given data type and return a logical value (`TRUE` or `FALSE`).

as.		is.	
as.numeric()	numeric	is.numeric()	numeric
as.integer()	integer	is.integer()	integer
as.double()	double	is.double()	double
as.character()	character	is.character()	character
as.factor()	factor	is.factor()	factor
as.logical()	logical	is.logical()	logical
as.vector()	vector	is.vector()	vector
as.list()	list	is.list()	list
as.matrix()	matrix	is.matrix()	matrix
as.data.frame()	data frame	is.data.frame()	data frame

What do you need to know by Week 3

- Understand R's basic data types (i.e., character, numeric, integer, factor, and logical).
- Understand R's basic data structures (i.e., vector, list, matrix, and data frame) and main differences between them.
- Learn to check attributes (i.e., name, dimension, class, levels etc.) of R objects.
- Learn how to convert between data types/structures.
- Practice!

Class Worksheet

- Working in small groups, complete the following class worksheet

Week 3 Class Worksheet

- Once completed, feel free to work on your Assignment and/or Skill Builders

Return to Data Preprocessing Website