R Bootcamp: Data Types

August 23-24, 2021

UBC MFRE | Master of Food and Resource Economics

Learning Objectives

- To know different data types available in R and the commands to inspect them
- To convert data types (e.g., numeric to character)
- To understand the basics of working with dates in R, specifically specifying the date formats
- To know how to subset or extract elements from vectors, matrices, dataframes

UBC MFRE

Data object types

- There are different types of objects in R, but common ones are numeric, character, factor, and logical

```
numeric_var <- 1
character_var <- “one”
factor_var <- factor(1, labels = ‘one’)
logical_var <- TRUE
```

- You can use **class()**, **typeof()**, **str()** commands to understand and inspect these objects
- You can convert numeric to character with **as.character(numeric_var)**

UBC MFRE

Vectors

- A **vector** is composed of a series of values, which be either numbers or characters.
  - All elements inside the vector are of the same type
- We assign a series of values to a vector using the **c()** function
- Types of vectors
  - character, numeric, integer, logical

UBC MFRE

Vectors

countries <- c(“Canada”, “Kenya”, “United States”)

countries

- quotes around the text are important to indicate the data type character. If not, R will think it’s an object, and since these objects don’t exist in R, you will get an error

emissions <- c(53700, 14300,5250000)

emissions

UBC MFRE

Vectors

- We can add values to existing vectors

```
countries <- c(countries, “China”)
```

UBC MFRE

Inspecting vectors

- Use the function **length()** to inspect the number of elements in a vector

length(countries)

- Use the function **class()** to inspect the type of elements in a vector

class(countries)

- Use the function **str()** to inspect the structure of an object and its elements

str(countries)

UBC MFRE

Vector and data types

- What if we mix different data types together?

```
trythis <- c(1, 2, 3, “a”)
class(trythis)
```

```
trythis <- c(“a”, 1, TRUE, “b”)
class(trythis)
```

UBC MFRE

Converting vectors

- Convert to character

```
emissions <- as.character(emissions)
class(emissions)
```

- Convert back to numeric

```
emissions <- as.numeric(emissions)
class(emissions)
```

UBC MFRE

Subsetting vectors

- We use the index position of an element in square brackets to extract one or more elements from a vector
  - Index starts at 1

emissions[1]

emissions[c(1,3)]

emissions[c(2:3)]

UBC MFRE

Conditional subsetting

- We can use logical tests to subset vectors
  - <, <=, >, >=
  - & and |

emissions[emissions > 100000]

- Select only emissions that are greater than 100000

emissions[emissions > 0 & emissions < 55000]

- Select only emissions that are greater than 0 and less than 55000

UBC MFRE

Value matching

- Use **%in%** to check whether an object is contained within or matches with a list of items

emissions %in% c(“Canada”)

countries %in% c(“Canada”, “United States”)

emissions %in% c(14300, 50000)

UBC MFRE

Factors

- Factors deal with categorical data (useful in survey analysis FRE518)

- In R, they are stored as integers with labels
  - Ordered (birth order, high/medium/low, etc.)
  - Unordered (color, country, etc.)

- The pre-defined set of values are called “levels”
  - By default, R sorts these levels alphabetically

UBC
MFRE

Factors

- Create factor variable with 5 levels
  - R assigns 1 = Africa, 2 = Americas, 3 = Europe

```
regions <- factor(c("Americas", "Americas", "Europe", "Africa", "Europe"))
```

- To know the levels

```
levels(regions)
```

- To know the number of levels

```
nlevels(regions)
```

UBC MFRE

Factors

- Sometimes, the order matters (low < medium < high)

```
responses <- factor(c("low", "low", "high", "medium", "high", "low"))
responses # current order

plot(responses)
```

UBC MFRE

Factors

- R now sorts it by the levels you specified

```
responses <- factor(responses,
  levels = c("low", "medium", "high"))
responses # after re=ordering
plot(responses)
```

- To establish the order

```
responses_ordered <- factor(responses,
ordered = TRUE)
responses_ordered
```

UBC MFRE

Recoding factors

- Let’s say you want to recode the second level from “medium” to “not sure”

```
levels(responses)[2] <- "not sure"
responses
```

UBC MFRE

Dates

- R will not always recognize your date variable as dates. Usually R will read it as a character or numeric vector.

```
dates <- c("2021/08/01", "2021/08/02", "2021/08/03")
class(dates)
```

- You will need to convert the variable to date. You can use the **as.Date()** function of the **{base}** package.
  - Other packages: **{lubridate}**

UBC MFRE

Dates

- You will need to specify the format of your date variable. Specifying the incorrect format will lead to parsing errors.

Code	Value
%Y	Year (4 digit | i.e., 2021)
%y	Year (2 digit | i.e., 21 for 2021)
%d	Day of month (in number)
%m	Month (in number)
%b	Month (in text | abbreviated | i.e. Aug for August)
%B	Month (in text | i.e., August)

https://www.stat.berkeley.edu/~s133/dates.html

UBC MFRE

Dates

- Because we put the dates in quotation marks, R reads them as character, and not dates.

```
dates_ymd1 <- c("2021/08/01", "2021/08/02", "2021/08/03")
dates_ymd2 <- c("21/08/01", "21/08/02", "21/08/03")
dates_mdy1 <- c("08/01/2021", "08/02/2021", "08/03/2021")
dates_mdy2 <- c("Aug 1, 2021", "Aug 2, 2021", "Aug 3, 2021")
dates_mdy3 <- c("August 1, 2021", "August 2, 2021", "August 3, 2021")

class(dates_ymd1)
```

UBC MFRE

Convert to Date format

- For **dates_ymd1** – 2021/08/01
  - big Y for YYYY (2021)
  - "/" because that is the separator used

```
convert_dates_ymd1 <- as.Date(dates_ymd1, format = "%Y/%m/%d")
```

- For **dates_ymd2** – 21/08/21
  - small y for YY (21 instead of 2021)

```
convert_dates_ymd2 <- as.Date(dates_ymd2, format = "%y/%m/%d")
```

UBC MFRE

Convert to Date format

- For **dates_mdy1** – 08/01/2021
  - Note the different format because month comes first
  - "/" because that is the separator used

```
convert_dates_mdy1 <- as.Date(dates_mdy1, format = "%m/%d/%Y")
```

- For **dates_mdy2** – Aug 1, 2021
  - Note the separator used – comma and space now instead of /
  - Small b for abbreviated month

```
convert_dates_mdy2 <- as.Date(dates_mdy2, format = "%b %d, %Y")
```

UBC MFRE

Convert to Date format

- For **dates_mdy3** – August 1, 2021
  - Note the separator used – comma and space now instead of /
  - Capital B for month

```
convert_dates_mdy3 <- as.Date(dates_mdy3, format = "%B %d, %Y")
```

- Will take practice to get used to it
- Always check your objects after converting to date to make sure you got it right; you will see NA if you misspecified the format

UBC MFRE

Convert to Date format

- With the as.Date() function, you have to have a day, as in you can't have 01/2021. You have to have 01/01/2021.

```
dates_noday <- c("01/2021", "02/2021", "03/2021")
```

UBC MFRE

Convert to Date format

- One option is to add a day with **paste()**; use 1 as an arbitrary day
  - paste() is R's concatenate or append function; equivalent to "&" in Excel

paste_dates_noday <- paste("01", dates_noday, sep = "/")

- This code will append "01" to dates_noday, and / is the separator
- So 01/01/2021, 01/02/2021, 01/03/2021
- If we did paste(dates-noday, "01", sep = "/") instead, the output will be 01/2021/01, 02/2021/01, 03/2021/

convert_dates_noday <- as.Date(paste_dates_noday, format = "%d/%m/%Y")

 # can do the above in 1 line

convert_dates_noday <- as.Date(paste("01", dates_noday, sep = "/"), format = "%d/%m/%Y")

UBC MFRE

Convert to Date format

- You can also use different functions in the {lubridate} package
  - **as_date()**
  - **ymd()**
  - **mdy()**
  - **dmy()**
  - **parse_date_time()**

UBC MFRE

Convert to Date format

```
dates_ymd1 <- c("2021/08/01", "2021/08/02", "2021/08/03")
dates_ymd2 <- c("21/08/01", "21/08/02", "21/08/03")
dates_mdy1 <- c("08/01/2021", "08/02/2021", "08/03/2021")
dates_mdy2 <- c("Aug 1, 2021", "Aug 2, 2021", "Aug 3, 2021")
dates_mdy3 <- c("August 1, 2021", "August 2, 2021", "August 3, 2021")

lubridate_convert_dates_ymd1 <- ymd(dates_ymd1)
lubridate_convert_dates_ymd2 <- ymd(dates_ymd2)
lubridate_convert_dates_mdy1 <- mdy(dates_mdy1)
lubridate_convert_dates_mdy2 <- mdy(dates_mdy2)
lubridate_convert_dates_mdy3 <- mdy(dates_mdy3)
```

UBC MFRE

Convert to Date format

```
dates_noday <- c("01/2021", "02/2021", "03/2021")

convert_with_lubridate <- parse_date_time(dates_noday, orders =
"m/Y")
```

UBC
MFRE

Date Sequence

- We can also create a Date class object using the **seq()** function

```
dates_seq <- seq(as_date("2021/08/01"), length = 5, by = "days")
dates_seq
```

UBC MFRE

Matrices

- Matrices are two-dimensional objects
- Elements must be of the same data type
- Elements are arranged in rows and columns

UBC
MFRE

Matrices

- We can construct a matrix using the **matrix()** function.

```
k <- matrix(nrow = 3, ncol = 2)
k
class(k)
dim(k)
```

UBC MFRE

Matrices

- Matrices are filled column-wise

- Let's create a 2x3 matrix called **I** filled with values 1 to 6.

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

UBC MFRE

Matrices

- We can also create a matrix using existing vectors; we created an **emissions** and **countries** vectors earlier

m <- matrix(c(countries, emissions), nrow = 3)

```
> m
     [,1]           [,2]
[1,] "Canada"       "53700"
[2,] "Kenya"        "14300"
[3,] "United states" "5250000"
```

UBC MFRE

Matrices

- We can create a matrix from vectors using the **cbind()** function

- Since vectors must be of the same length, let's rewrite our countries object again

```
matrix <- cbind(countries, emissions)
matrix
```

UBC MFRE

Naming columns and rows

- The column names are the names of the data objects specified in the **cbind()** function.

- You can use the **colnames()** and **rownames()** functions to specify or rename the columns and rows.

```
colnames(matrix)[2] <- “emissions_new”
rownames(matrix) <- c(“c1”, “c2”, “c3”)
matrix
```

```
> matrix
   countries       emissions_new
c1 "Canada"        "53700"
c2 "Kenya"         "14300"
c3 "United states" "5250000"
```

UBC MFRE

Subsetting matrices

- We also use square brackets to subset matrices.

- Matrices are 2 dimensional, so we need to indicate the row and column positions of the element/s we want to extract.
  - Syntax: **matrix[row_position, column_position]**
  - If you leave the row (column) position blank, then R assumes you want the whole row (column).

UBC MFRE

Subsetting matrices

- Extract the first element of the second column
  matrix[1,2]
- Extract the first row (leave column position blank)
  matrix[1,]
- Extract the first column (leave row position blank)
  matrix[,1]
- Extract second row by row name
  matrix[“c2”,]

UBC MFRE

Subsetting vectors

- Extract first column
  matrix[,1]

- Extract by column name
  matrix[, c(“emissions_new”)

- Extract first two rows of the first column
  matrix[1:2, 1]

UBC
MFRE

Lists

- A list is a flexible R object that contains multiple data types
  - Example: regression results

- We can subset objects using **[[ ]]** or **$**

UBC MFRE

Lists

```
first_list <- list(a = 1:5, b = 6:10, c = c("food", "resource", "economics"))
first_list

first_list["a"] #output is a list

first_list[['a']] #output is integer

first_list$c #output is character
```

UBC MFRE

Data frames

- Data frames are lists composed of vectors of the same length.
  - Each element can be thought of as a column.
  - The length of each element of the list is the number of rows.
- You can think of a data frame as an Excel worksheet that contains columns of different data types but all have the same rows.
- You will mostly work with data frames or tibbles, a special type of data frame

UBC MFRE

Data frames

- Create new data frame (df for short)

```
first_df <- data.frame(countries = c("Canada", "Kenya", "United States"), emissions = c(53700, 14300, 5250000))
first_df
```

- Convert matrix to df

```
matrix_df <- as.data.frame(matrix)
matrix_df
```

UBC MFRE

Data frames

- Here are some functions to inspect the data frame
  - dim(), str(), head(), and names()

dim(first_df)

str(first_df)

head(first_df)

names(first_df)

UBC MFRE

Extracting elements

- We can use [], [[]], and $ to extract elements from the data frame.

```
first_df["countries"] #output is a data frame
first_df[["countries"]] #output is a character vector
first_df$countries #output is a character vector
```

UBC MFRE

What we just did

- Learned numeric, character, factors, and logical data types are in relation to R
- Created, inspected, subset vectors
- Created ordered and un-ordered factors
- Worked with dates and different ways of specifying date formats
- Created, inspected, and subset matrices, lists, and data frames

UBC
MFRE

UBC MFRE

mfre.landfood.ubc.ca