STQD6134: Business Analytics

Data Extraction

Introduction to R

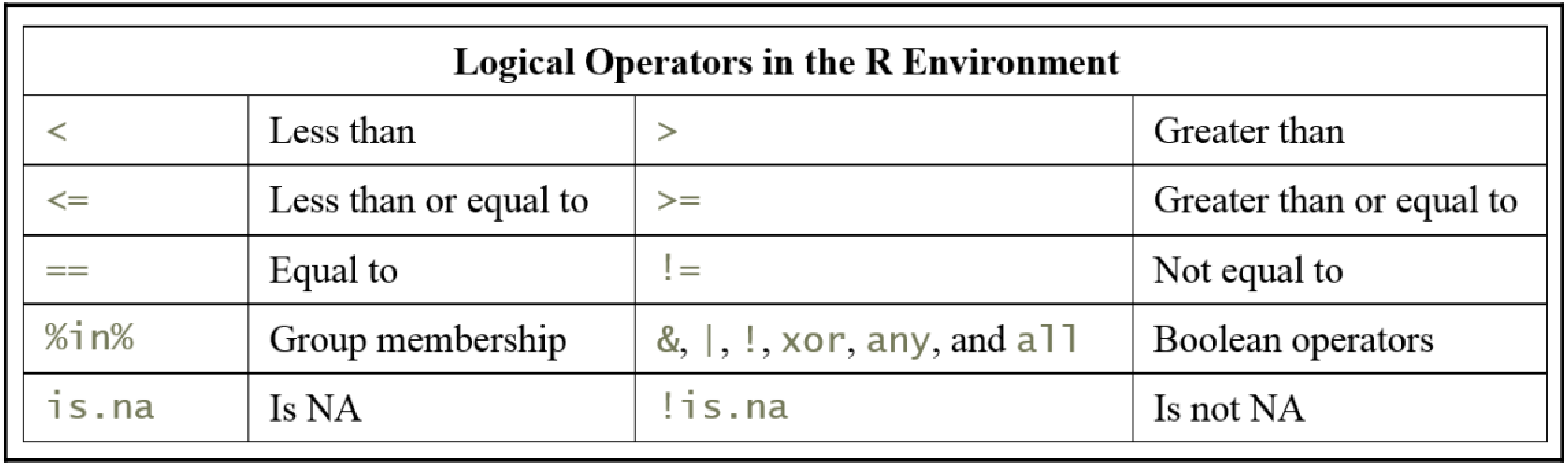
Install the software: https://cran.r-project.org/bin/windows/base/

Basic tutorials: https://www.statmethods.net/r-tutorial/index.html

Importing data

- `bike <- read.csv("{filename}.csv")`
- `getwd():` This will return the current working directory as a path in the R console
- `setwd():` This is used in the console to change the working directory to you pass in the function
- `str(bike)`
- `bike <- read.table("./data/Ch1_bike_sharing_data.csv", sep = ",", header = TRUE)`

Logic operators in R



Logical Operators in the R Environment

`<`	Less than	`>`	Greater than
`<=`	Less than or equal to	`>=`	Greater than or equal to
`==`	Equal to	`!=`	Not equal to
`%in%`	Group membership	`&`, `|`, `!`, `xor`, `any`, and `all`	Boolean operators
`is.na`	Is NA	`!is.na`	Is not NA

Transforming data

- Install dplyr package

```
library(dplyr)
extracted_rows <- filter(bike, registered == 0,
                         season == 1 | season == 2)
dim(extracted_rows)
```

```
using_membership <- filter(bike, registered == 0, season %in% c(1, 2))
identical(extracted_rows, using_membership)
```

Adding a calculated column from existing data

- Casual renters pay five dollars for a day pass. You figured out that all you have to do is multiply the number of casual renters by five to get the revenues for each day in your data frame.

```
add_revenue <- mutate(extracted_columns, revenue = casual * 5)
```

Aggregate data into groups

```
grouped <- group_by(add_revenue, season)

report <- summarise(grouped, sum(casual), sum(revenue))
```

Export data

- To .csv or tab-delimited text file

```
write.csv(report, "revenue_report.csv", row.names = FALSE)

write.table(report, "revenue_report.txt", row.names = FALSE, sep = "\t")
```

Summarizing data

```
bike <- read.csv("./data/Ch2_raw_bikeshare_data.csv",
                 stringsAsFactors = FALSE)
str(bike)
```

```
'data.frame':  17379 obs. of  13 variables:
 $ datetime  : chr  "1/1/2011 0:00" "1/1/2011 1:00" "1/1/2011  ...
 $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
 $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
 $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
 $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
 $ humidity  : chr  "81" "80" "80" "75" ...
 $ windspeed : num  0 0 0 0 0 ...
 $ casual    : int  3 8 5 3 0 0 2 1 1 8 ...
 $ registered: int  13 32 27 10 1 1 0 2 7 6 ...
 $ count     : int  16 40 32 13 1 1 2 3 8 14 ...
 $ sources   : chr  "ad campaign" "www.yahoo.com" "www.google.fi" ...
```

You can immediately interpret some key characteristics of the bike sharing data:

- `17379` observations and `13` variables
- Three data types: three character (`chr`), seven integer (`int`), and three numeric (`num`)
- The first observations of each variable give you a sense of the content

- `dim()` provides you with the row and column dimensions of a data frame
- `head()` shows the top six lines of the data frame to see the surrounding data
- `tail()` shows the bottom six lines of the data frame

Finding and fixing flawed data

```
table(is.na(bike))
```

You found that the `humidity` variable was problematic when you reviewed the data summary. `humidity` is a numerical value, but the summary showed it stored as a character data type. You will now determine what is causing the problem in this column.

One strategy that you can use is to search for all instances of character values in `humidity` and save them to a `bad_data` object. The `str_subset()` function of `stringr` applies a **regular expression** string to a column and extracts all matching instances into an object:

```
bad_data <- str_subset(bike$humidity, "[a-z A-Z]")
```

Printing the `bad_data` object reveals a single error where the `x61` value is included in the column. You are familiar with the idea of imputing data and are aware that the technique can be misused. In this case, it appears that the error is typographical and you choose to impute the value rather than eliminate the row. Passing the elements of `bad_data` into the `str_detect()` function will assign the row position of `x61` to `location`:

```
location <- str_detect(bike$humidity, bad_data)
```

```
bike[location, ]
```

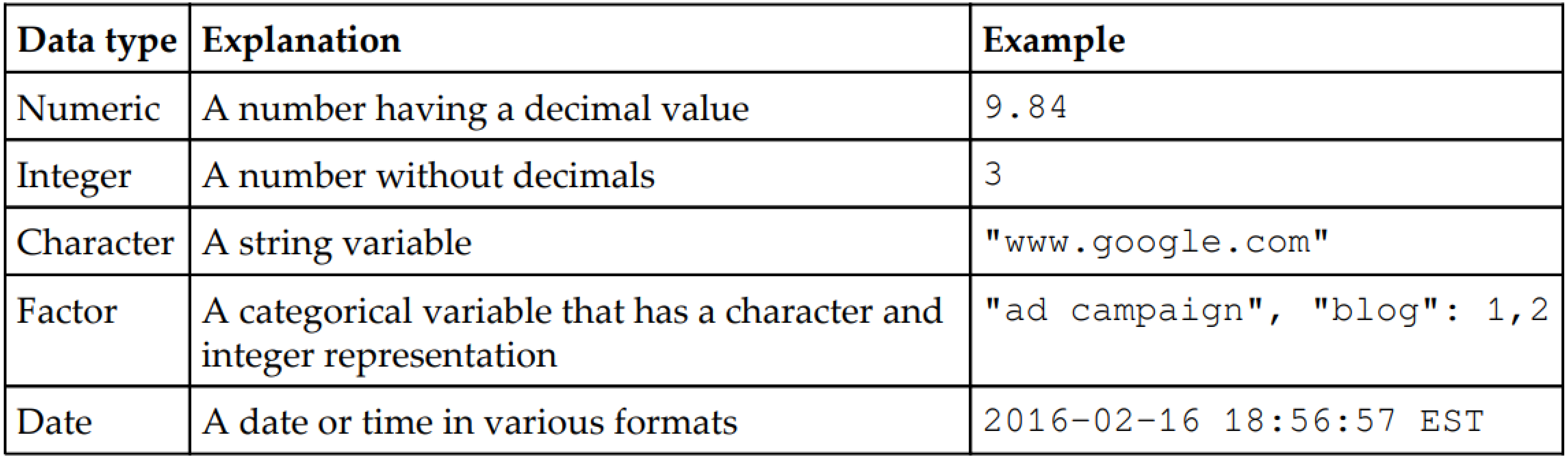
The output is as follows:

```
            datetime season holiday workingday weather  temp atemp
14177 8/18/2012 21:00      3       0          0       1 27.06 31.06
      humidity windspeed casual registered count       sources
14177      x61         0     90        248   338 www.bing.com
```

```
bike$humidity <- str_replace_all(bike$humidity, bad_data, "61")
```

```
                datetime season holiday workingday weather  temp atemp
14177 8/18/2012 21:00      3       0          0       1 27.06 31.06
      humidity windspeed casual registered count       sources
14177       61         0     90        248   338 www.bing.com
```

Converting data types



Data type	Explanation	Example
Numeric	A number having a decimal value	`9.84`
Integer	A number without decimals	`3`
Character	A string variable	`"www.google.com"`
Factor	A categorical variable that has a character and integer representation	`"ad campaign", "blog": 1,2`
Date	A date or time in various formats	`2016-02-16 18:56:57 EST`

- **Character to Numeric**: The typographic error in the `humidity` data coerced it to a character type, but you need to convert it back to numeric.

- **Character to Factor**: A common strategy for using `read.csv()` is to set `stringsAsFactors = FALSE`. This means categorical features are not automatically detected and you will later choose which ones are factors.
- **Character to Date**: A common situation is incorrect date conversion during file read. All your dates are characters. This is a difficult conversion, but it is detailed in the *Date and time conversions* section.

```r
bike$humidity <- as.numeric(bike$humidity)


bike$holiday <- factor(bike$holiday, levels = c(0, 1),
                       labels = c("no", "yes"))
bike$workingday <- factor(bike$workingday, levels = c(0, 1),
                          labels = c("no", "yes"))

bike$weather <- factor(bike$weather, levels = c(1, 2, 3, 4),
                       labels = c("clr_part_cloud",
                                  "mist_cloudy",
                                  "lt_rain_snow",
                                  "hvy_rain_snow"),
                       ordered = TRUE)

bike$season <- factor(bike$season, levels = c(1, 2, 3, 4),
                      labels = c("spring", "summer",
                                 "fall", "winter"),
                      ordered = TRUE)
```

```
library(lubridate)
bike$datetime <- mdy_hm(bike$datetime)
```

Adapting data to standard

Your implied standard is that `sources` should be a categorical variable. What might happen if you use the `as.factor(bike$sources)` function? This will convert the data, but before you do that, you should consider a couple of questions:

- How many unique kinds of advertising source are in `sources`?
- How many categories would you like to have in your analysis dataset?

You can answer the first question with `unique(bike$sources)` to see all unique values:

```
 [1] "ad campaign"       "www.yahoo.com"     "www.google.fi"
 [4] "AD campaign"       "Twitter"           "www.bing.com"
 [7] "www.google.co.uk" "facebook page"     "Ad Campaign"
[10] "Twitter      "     NA                 "www.google.com"
[13] "direct"            "blog"
```

It is worth discussing this output in detail to understand data adaption before we perform any string manipulation. What do you see in the results?

Observed	**Explained**
Two Twitter strings	Extra white space at element 10 treats these as different values
Multiple ad campaigns	R is case-sensitive-making elements 1, 4, and 9 all unique cases
The `NA` value	As discovered in the *Finding flaws in datasets* section

The `stringr` R package contains many functions to manipulate strings. The following code shows two functions, as well the use of subsetting to convert all strings to lower case, trim excess white space, and replace all `NA` values with the `unknown` string:

```
bike$sources <- tolower(bike$sources)
bike$sources <- str_trim(bike$sources)
na_loc <- is.na(bike$sources)
bike$sources[na_loc] <- "unknown"
```

Rerunning the `unique(bike$sources)` function produces the fruits of your labor. You have reduced the original 14 unique types of advertising source to 11, as follows:

```
 [1] "ad campaign"      "www.yahoo.com"    "www.google.fi"
 [4] "twitter"          "www.bing.com"     "www.google.co.uk"
 [7] "facebook page"    "unknown"          "www.google.com"
[10] "direct"           "blog"
```

Combine data into new catagories

Having eleven different advertising sources is not *bad* in the sense that there is no right or wrong answer. *Miller's Law* helps analysts think about analysis design. Consider histograms: too many categories is just as problematic as too few categories. For this project, you will decide to have five to nine categories of advertising sources in the dataset.

After talking with people experienced in advertising, you will learn that, when it comes to websites, the important thing is not which search engine they used. It is only important to know that the person found Bike Sharing services on the Web. You may decide to adapt all search engine sources by labeling them all as `web`. There is an R package for that.

The `DataCombine` package is a user-friendly library of functions to combine data into newly defined categories. The following code accomplishes this adaption:

```
library(DataCombine)
web_sites <- "(www.[a-z]*.[a-z]*)"
current <- unique(str_subset(bike$sources, web_sites))
replace <- rep("web", length(current))
```

The preceding code is described as follows:

- Assign a regular expression string into the `web_sites` variable, which is now a search string to find all variants of strings starting with `www`.
- Use `str_subset()` to find these instances and assign them to `current`.
- Create an instance of the `web` replacement string for each item contained in the `current` variable and assign them to `replace`. The `length(current)` function tells the code how many instances of `web` are needed. It should be as many as there are elements in `current`.

Create a cross-reference table by storing the `current` and `replace` variables as vectors in a `replacements` data frame:

```
replacements <- data.frame(from = current, to = replace)
```

You are now ready to use the `FindReplace()` function to adapt the strings in `bike$sources` from five different Web search engine names to the `web` string. Notice how the `FindReplace()` function uses the `from` and `to` elements of the `replacements` data frame that you just created as input parameters to the function in the following code:

```
bike <- FindReplace(data = bike, Var = "sources", replacements,
                    from = "from", to = "to", exact = FALSE)
unique(bike$sources)
```

The output is as follows:

```
[1] "ad campaign"   "web"           "twitter"       "facebook page"
[5] "unknown"       "direct"        "blog"
```

Running the `unique()` function shows that you have reduced the original 14 unique types of advertising source down to 7. George Miller would be proud of you. The last thing to do is fully adapt the string variable by converting it to a factor with the `as.factor()` function:

```
bike$sources <- as.factor(bike$sources)
```