MTH 3270 Notes 5

5 Tidy Data and Iteration (Cont'd) (5)

5.4 Data Intake

- There are other ways to read data into R besides `read.table()` and `read.csv()`.
- ***Web scraping*** refers to reading data from an HTML web page. The `"rvest"` package (more specifically, the `"xml2"` package upon which `"rvest"` is built) has functions for **web scraping** (aka ”***harvesting***” data).

  Among those functions are the following.

  ```
  read_html()       # Read an HTML file into R from its URL.
  html_nodes()      # Select nodes (elements) from an HTML file that has
                    # been read into R.
  html_table()      # Convert an HTML table into a data frame.
  ```

- Reading data from a we page into R is a three-step process:

  1. Read the entire HTML file into R by downloading it from a URL using `read_html()`.
  2. Extract the table(s) from the HTML file using `html_nodes()`.
  3. Convert the table(s) into data frames using `html_table()`.

- For example (from pg 118 of our textbook *Modern Data Science with R*), the Wikipedia page

  `https://en.wikipedia.org/wiki/Mile_run_world_record_progression`.

  has tables showing the progression of world record times for the mile run. Each table corresponds to a particular group of runners (e.g. professionals, amateurs, males, females, etc.).

  To read the data into R, we first type:

  ```
  library(rvest)
  ```

```
url <- "https://en.wikipedia.org/wiki/Mile_run_world_record_progression"
tables <- url %>% read_html() %>% html_nodes("table")
```

The `tables` object isn't a data frame, it's a list, each element of which is the HTML code for one table:

```
is.list(tables)

## [1] TRUE

length(tables)

## [1] 12

tables

## {xml_nodeset (12)}
##  [1] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Athlete ...
##  [2] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Athlete ...
##  [3] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Athlete ...
##  [4] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Auto</t ...
##  [5] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Auto</t ...
##  [6] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Athlete ...
##  [7] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Athlete ...
##  [8] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Auto</t ...
##  [9] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Athlete ...
## [10] <table class="wikitable"><tbody>\n<tr>\n<th>Time</th>\n<th>Athlete ...
## [11] <table class="nowraplinks mw-collapsible autocollapse navbox-inner ...
## [12] <table class="nowraplinks navbox-subgroup" style="border-spacing:0 ...
```

To convert one of the tables (the third one, say) to a data frame, type:

```
Table3 <- html_table(tables[[3]])
Table3

##   Time             Athlete    Nationality             Date     Venue
## 1 4:52      Cadet Marshall United Kingdom 2 September 1852 Addiscome
## 2 4:45        Thomas Finch United Kingdom  3 November 1858    Oxford
## 3 4:45 St. Vincent Hammick United Kingdom 15 November 1858    Oxford
## 4 4:40       Gerald Surman United Kingdom 24 November 1859    Oxford
## 5 4:33       George Farran United Kingdom       23 May 1862    Dublin
```

Section 5.4 Exercises

Exercise 1 Using the approach described above (and the same Wikipedia website), create an R data frame containing the data from the **fourth** table of world record times for the mile run. Report your R commands.

5.5 Cleaning Data

5.5.1 Recoding

- Sometimes **categorical** data are coded as **integers**.

  We can **recode** them as `"character"` values by creating a ***codebook*** data frame indicating the correspondence between **integer** and `"character"` values, then using `"dplyr`'s `left_join()`.

- For example consider the following data on houses for sale (from pg 121 of our textbook *Modern Data Science with R*):

```
myURL <- "http://tiny.cc/dcf/houses-for-sale.csv"
Houses <- read.csv(myURL)
head(Houses)

##    price lot_size waterfront age land_value construction air_cond fuel
## 1 132500     0.09          0  42      50000            0        0    3
## 2 181115     0.92          0   0      22300            0        0    2
## 3 109000     0.19          0 133       7300            0        0    2
## 4 155000     0.41          0  13      18700            0        0    2
## 5  86060     0.11          0   0      15000            1        1    2
## 6 120000     0.68          0  31      14000            0        0    2
##   heat sewer living_area pct_college bedrooms fireplaces bathrooms rooms
## 1    4     2         906          35        2          1       1.0     5
## 2    3     2        1953          51        3          0       2.5     6
## 3    3     3        1944          51        4          1       1.0     8
## 4    2     2        1944          51        3          1       1.5     5
## 5    2     3         840          51        2          0       1.0     3
## 6    2     2        1152          22        4          1       1.0     8
```

We'll use a *subset* of the variables, namely `fuel`, `heat`, `sewer`, and `construction`:

```
Houses_small <- select(Houses, fuel, heat, sewer, construction)
```

```
head(Houses_small)

##   fuel heat sewer construction
## 1    3    4     2            0
## 2    2    3     2             0
## 3    2    3     3             0
## 4    2    2     2             0
## 5    2    2     3             1
## 6    2    2     2             0
```

To *recode* `fuel` from **integers** to `"gas"`, `"electric"`, etc., and `sewer` to `"none"`, `"private"`, etc., we first create a *codebook* data frame that can be used to translate the **integers** to `"character"`:

```
Translations <- read.csv("http://tiny.cc/dcf/house_codes.csv",
                         stringsAsFactors = FALSE)
Translations

##    code system_type   meaning
## 1     0   new_const        no
## 2     1   new_const       yes
## 3     1  sewer_type      none
## 4     2  sewer_type   private
## 5     3  sewer_type    public
## 6     0 central_air        no
## 7     1 central_air       yes
## 8     2   fuel_type       gas
## 9     3   fuel_type  electric
## 10    4   fuel_type       oil
## 11    2   heat_type   hot air
## 12    3   heat_type hot water
## 13    4   heat_type  electric
```

The same information can also be presented in a wide format:

```
CodeVals <- Translations %>% spread(key = system_type,
                                    value = meaning,
                                    fill = "invalid")
CodeVals

##   code central_air fuel_type heat_type new_const sewer_type
## 1    0          no   invalid   invalid        no    invalid
## 2    1         yes   invalid   invalid       yes       none
## 3    2     invalid       gas   hot air   invalid    private
## 4    3     invalid  electric hot water   invalid     public
## 5    4     invalid       oil  electric   invalid    invalid
```

Now we use `left_join()` to merge `Houses_small` with `CodeVals`, matching rows in `CodeVals` by `code` to rows in `Houses_small` by `fuel` and then by `sewer`.

```
Houses_small <- Houses_small %>%
   left_join(CodeVals %>%
             select(code, fuel_type), by = c(fuel = "code")) %>%
   left_join(CodeVals %>%
             select(code, sewer_type), by = c(sewer = "code"))
```

Here's the resulting data set, with *recoded* `fuel` and `sewer` variables:

```
head(Houses_small)

##   fuel heat sewer construction fuel_type sewer_type
## 1    3    4     2            0  electric    private
## 2    2    3     2            0       gas    private
## 3    2    3     3            0       gas     public
## 4    2    2     2            0       gas    private
## 5    2    2     3            1       gas     public
## 6    2    2     2            0       gas    private
```

5.5.2 From Strings ("character") to Numbers

- Sometimes a **numeric** vector will inadvertently be read into R as `"character"`.

  We can convert it back to numeric using `as.numeric()`.

- For example, here `y` is `"character"`:

```
my_data <- data.frame(Name = c("Joe", "Kim", "Al", "Don", "Ann"),
                      y = c("2", "5", "6", "1", "7"),
                      stringsAsFactors = FALSE)
```

```
str(my_data)
```

```
## data.frame: 5 obs. of  2 variables:
##  $ Name: chr  "Joe" "Kim" "Al" "Don" ...
##  $ y   : chr  "2" "5" "6" "1" ...
```

We change `y` to numeric using `mutate()` and `as.numeric()` by typing:

```
my_data <- mutate(.data = my_data, y = as.numeric(y))
```

and now `y` is **numeric** as desired:

```
str(my_data)
```

```
## data.frame: 5 obs. of  2 variables:
##  $ Name: chr  "Joe" "Kim" "Al" "Don" ...
##  $ y   : num  2 5 6 1 7
```

- To go the other way (from **numeric** to `"character"`), use `as.character()`.

Section 5.5 Exercises

Exercise 2 Here's a data frame:

```
x <- data.frame(Name = c("Joe", "Lucy", "Tom", "Sally"),
                NumberChildren = c("2", "1", "0", "3"),
                stringsAsFactors = FALSE)
```

a) After creating the data frame `x`, type:

```
str(x)
```

What type of variable is `NumberChildren` (**numeric** or `"character"`)?

b) Write one or more commands that convert the `NumberChildren` column of `x` to **numeric**. Check your answer using `str()`. Report your R command(s).

5.5.3 Dates

- Often **dates** end up being stored as `"character"` values in a data frame.

  When this is the case, R doesn't recognize the inherent ordering in the dates (e.g. `"16 December 2019"` should come *after* `"29 October 2019"`).

  It's preferable in this case to convert the variable to an object of class `"Date"`. R recognizes the ordering in objects that belong to the `"Date"` class.

- The `"lubridate"` package has several functions that are useful for working with date/time variables.

  The functions below convert dates stored as `"character"` vectors to `"Date"` objects.

```
ymd()       # Converts "character" (year, month, day) to a "Date" object
mdy()       # Converts "character" (month, day, year) to a "Date" object
dmy()       # Converts "character" (day, month, year) to a "Date" object
ymd_hms()   # Converts "character" (year, month, day, hour, minute,
            # second) to a "Date" object
```

(They can also be used to convert `"character"` vectors to so-called **POSIXct**) objects.)

For a complete list of the functions in the `"lubridate"` package, type:

```r
help(package = lubridate)
```

- Here are some examples:

```r
library(lubridate)
```

```r
myDate <- mdy("12/18/73")
myDate

## [1] "1973-12-18"

class(myDate)

## [1] "Date"
```

```r
myDates <- mdy(c("12/18/73", "12/19/73", "12/20/73"))
myDates

## [1] "1973-12-18" "1973-12-19" "1973-12-20"

class(myDates)

## [1] "Date"
```

- Internally, `"Date"` objects are stored in R as numerical values – the number of *days* **since 01-01-1970** (the so-called **UNIX epoch**):

```r
as.numeric(mdy("01-01-1970"))

## [1] 0

as.numeric(mdy("01-02-1970"))

## [1] 1

as.numeric(mdy("01-01-1971"))

## [1] 365
```

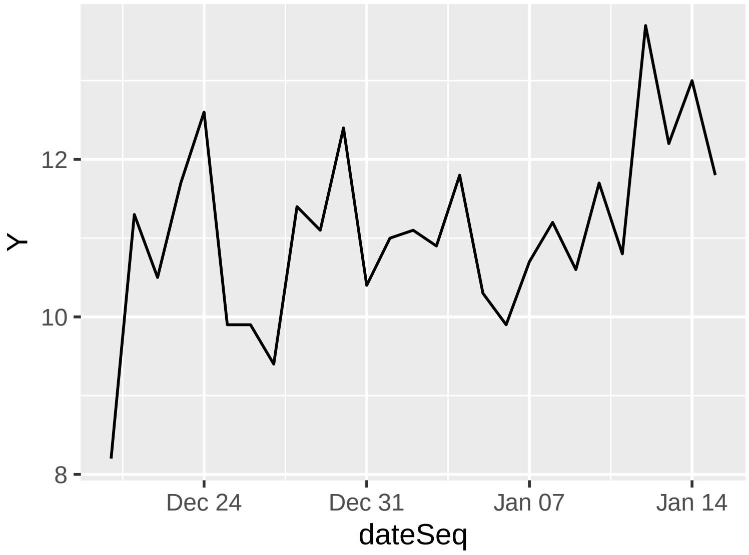
This allows for subtraction to find the elapsed number of days between two dates:

```
myDate1 <- mdy("12-20-1973")
myDate2 <- mdy("01-15-1974")
myDate2 - myDate1

## Time difference of 26 days
```

- It also allows for a vector of dates to be used as the x variable in a plot.

```
dateSeq <- seq(from = mdy("12-20-1973"), to = mdy("01-15-1974"), by = "days")
y <- c(8.2, 11.3, 10.5, 11.7, 12.6,  9.9, 9.9, 9.4, 11.4, 11.1, 12.4,
       10.4, 11.0, 11.1, 10.9, 11.8, 10.3, 9.9, 10.7, 11.2, 10.6, 11.7,
       10.8, 13.7, 12.2, 13.0, 11.8)
myData <- data.frame(Date = dateSeq, Y = y)
head(myData)

##         Date    Y
## 1 1973-12-20  8.2
## 2 1973-12-21 11.3
## 3 1973-12-22 10.5
## 4 1973-12-23 11.7
## 5 1973-12-24 12.6
## 6 1973-12-25  9.9
```

```
ggplot(data = myData, mapping = aes(x = dateSeq, y = Y)) +
  geom_line()
```



- Specific components of `"Date"` objects can be extracted using the following functions.

```
day()     # Get the day of the month from a "Date" object
mday()    # Same as day()
wday()    # Get the day of the week from a "Date" object
yday()    # Get the day of the year from a "Date" object
week()    # Get the week of the year from a "Date" object
```

- The `"Date"` class of objects (from the `"lubridate"` package) is most useful for dates that don't include the time of day.
- For **timestamp** data (also called **datetime** data), i.e. data that includes time of day (e.g. hour, minute, second), in which the **time zone** is important, the `"POSIXct"` and `"POSIXlt"` classes of objects are useful.

  The `"POSIXct"` and `"POSIXlt"` classes can generally be treated the same, but internally they're stored differently.

  `"POSIXct"` objects are stored as numerical values – the number of *seconds* **since 01-01-1970**. `"POSIXlt"` objects are stored as a *list* of year, month, day, hour, etc. `"character"` values.

Section 5.5 Exercises

Exercise 3 The functions `ymd()`, `mdy()`, etc. (from the `"lubridate"` package) recognize `"character"` dates in a variety of formats, and in each case covert from `"character"` to the `"Date"` class. Guess what each of the following commands returns, then check your answers.

a) `mdy("Dec 18, 1973")`

b) `mdy("December 18, 1973")`

c) `mdy("12/18/1973")`

d) `mdy("12/18/73")`

e) `mdy("12-18-1973")`

f) `mdy("12-18-73")`

Exercise 4 Be careful when using `ymd()`, `mdy()`, etc. with `"character"` dates for which the century isn't given. Does `mdy()` interpret `"11/14/23"` as referring to the year 2023 or 1923? Try it.

```
mdy("11/14/23")
```

Exercise 5 How many elapsed days are there between January 15, 2007 (`"1/15/07"`) and October 4, 2019 (`"10/4/19"`)?

Exercise 6 Guess what each of the following commands does, then check your answers.

a)
```
seq(from = mdy("12-20-1993"), to = mdy("01-15-2004"),
       by = "days")
```

b)
```
seq(from = mdy("12-20-1993"), to = mdy("01-15-2004"),
       by = "weeks")
```

c)
```
seq(from = mdy("12-20-1993"), to = mdy("01-15-2004"),
    by = "years")
```

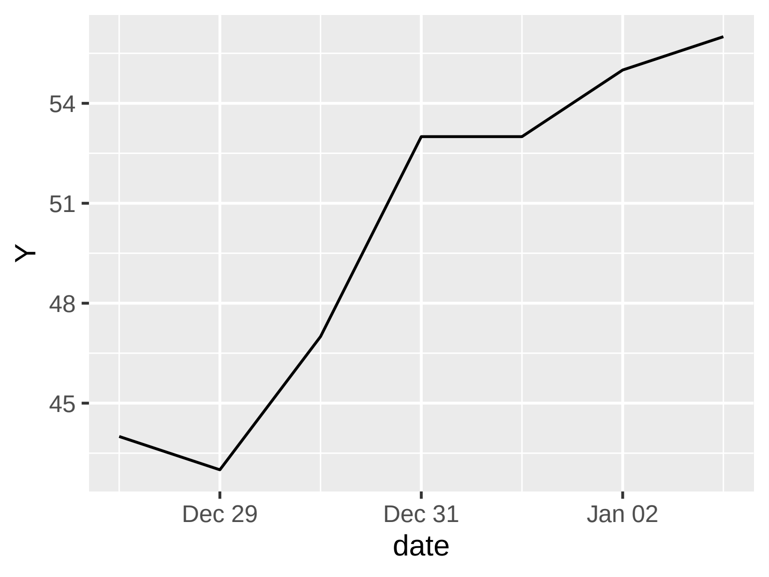
Exercise 7 Here's a data frame:

```
my.data <- data.frame(date = c("12/28/2017",  "12/29/2017", "12/30/2017",
                        "12/31/2017", "1/1/2018", "1/2/2018", "1/3/2018"),
                      Y = c(44, 43, 47, 53, 53, 55, 56))
```

a) Why doesn't the following plot command work?

```
ggplot(data = my.data, mapping = aes(x = date, y = Y)) +
  geom_line()
```

b) How can you use `mutate()` (from the `"dplyr"` package) and `mdy()` to fix the problem? Do it and report your R commands. You should end up with this:


54
51
48
45
Y
Dec 29
Dec 31
Jan 02
date