EDS Vignettes

Andry Rajaoberison

2020-04-08

Contents

Prerequisites

The following tutorials are written in R. To install R on your computer, please visit this link: https://www.r-project.org/ You should also install RStudio: https://rstudio.com/products/rstudio/download/

We will use the packages in the **tidyverse** library:

```r
install.packages("tidyverse")
```

Chapter 1

Reading

In this section, we learn how to read data from different file format.

1.1 CSV

Reading csv with column headers and separated by `,`. These parameters are also the default values for the `read.csv` function.

```
data <- read.csv(file = '/path/to/csv', header = TRUE, sep = ',')
```

```
# Example
data <- read.csv(file = 'data/eds.data.hurricane.csv', header = TRUE)
```

1.2 Excel

The main advantage of Excel files is that they can store multiple tables. But reading these tables at once is different from a CSV. For this example, we're going to use the `readxl` package from the `tidyverse` collection. Please visit this [website] (https://www.tidyverse.org/) to learn more about `tidyverse`.

To read an excel file, you can use the `read_excel` function and specify at least the `path/to/the/file` and `sheet` you want to open. If you don't specify the `sheet`, `read_excel` will automatically open the first table in the spreadsheet. In the 'eds.excel.sample.xlsx' file, there are 2 tables: heatwave and hurricane. Here's how we load both tables into R:

```
library(readxl)
heatwave <-  read_excel(path='data/eds.excel.sample.xlsx', sheet = 'heatwave')
hurricane <-  read_excel(path='data/eds.excel.sample.xlsx', sheet = 'hurricane')
```

Once the tables are stored into individual R variable, you can perform exploration and analysis with them.

1.3 Google Spreadsheets

If the data is stored in a Google spreadsheet, we can read it using the `googledrive` and `googlesheet4` packages. We use the `googledrive` package to log into our Google Drive account and `googlesheets4` to read the speadsheets in our drive.

In the example below, I used a spreadsheet named `eds.sample.googlesheets` which contains the same tables in the previous Excel example (heatwave and hurricane). You can clone the spreadsheet via this [link] (https://drive.google .com/open?id=1uIsgrcsevbm9voZU-rzqhTg2LE5SgEPlGabSXKTcQtc if you'd like to repeat the steps below using your Google account.

Then authenticate to your drive using `drive_auth()`. When prompted: log in, authorized googledrive, and use the authorization code if provided. You only need to run `drive_auth()` once.

```r
library(googledrive)
# To authenticate and authorize googledrive package
drive_auth()
```

The following scripts show how to explore a Google Drive folder. Though this is not recommended as you might encounter performance issues.

```r
# NOT recommended
# Then, to view the list of files in a folder
drive_ls("EDS") # where "EDS" is the folder name
# To also get the files within the subfolders
drive_ls("EDS", recursive = TRUE)
# To view the list of spreadsheets within a folder
drive_ls("EDS", type="spreadsheet")
```

Also, because of Google authentification system, you may run into an error like below when running the previous code (using `drive_ls()`). Which is why it's not recommended.

```
#> Error in add_id_path(nodes, root_id = root_id, leaf = leaf) : !anyDuplicated(nodes$
```

To avoid this, you can use the folder url instead of the folder name. The folder url can be obtained by right-clicking on the folder and click `Get shareable link`. Then run the following code:

```r
# If using folder name doesn't work
folder_url = 'https://drive.google.com/open?id=1e0uJ9dwFcL34JA61F0tGSoaiMZ_xio_4'
drive_ls(folder_url, type="spreadsheet")
```

Then you can load the spreadsheet by using its `id`

```
eds.sample.spreadsheet <- drive_get(id = '1uIsgrcsevbm9voZU-rzqhTg2LE5SgEPlGabSXKTcQtc')
```

It also possible to read the spreadsheet right way by using its link / `path` (without using `drive_ls()`). I recommend using this to read any Google Drive files.

```
eds.sample.spreadsheet <- drive_get(path = 'https://drive.google.com/open?id=1uIsgrcsevbm9voZU-rz
```

Once the spreadsheet is loaded, we run a similar code used for the Excel files to read tables within the spreadsheet. But for Google Sheets, function is called `read_sheet`

```
library(googlesheets4)
# Authorizing the googlesheets4 package
sheets_auth(token=drive_token())
# Reading the tables
heatwave <- read_sheet(eds.sample.spreadsheet, sheet = 'heatwave')
hurricane <- read_sheet(eds.sample.spreadsheet, sheet = 'hurricane')
```

1.4 SPSS

```
library(haven)
data <- read_sav("data/eds.spss.sample.sav")
```

By default, the `read_sav()` will read the factor levels of non-numeric and non-character variables. If, instead, we want the labels, we can run the following code:

```
library(magrittr)
library(dplyr)
# Applying haven::as_factor() to labelled columns Here, we already know that
# variables Zone, Q4 and Q50 are not factor variables.
data %>% mutate_at(vars(-Zone, -Q4, -Q50), as_factor)
```

```
## # A tibble: 1,130 x 9
##    Zone     Q4 Q5             Q6    Q7              Q10     Q50 Q51   Q59
##    <chr> <dbl> <fct>          <fct> <fct>           <fct> <dbl> <fct> <fct>
##  1 A         2 3              0     Moderately Pr… No     1928 Male  $70,000-$99…
##  2 A         1 4              0     Moderately Pr… No     1962 Male  <NA>
##  3 A         3 4              0     Moderately Pr… No     1931 Fema… Over $200,0…
##  4 A         3 6              1     Fully Prepared No     1950 Male  $100,000-$1…
##  5 A         2 Not Worried …  0     Very Prepared  No     1948 Male  $100,000-$1…
##  6 A         5 4              0     Very Prepared  No     1938 Fema… <NA>
##  7 A         3 6              1     Moderately Pr… No     1977 Fema… <NA>
##  8 A         5 4              0     Moderately Pr… No     1964 Fema… <NA>
##  9 A          1 3              0     Moderately Pr… No     1976 Male  $40,000-$69…
## 10 A          2 6              0     Very Prepared  No     1964 Fema… Over $200,0…
## # … with 1,120 more rows
```

Because variables can be labelled in SPSS, we can use them as well to find out what each column represents.

```
# To get the labels of the variables / columns
as.vector(unlist(lapply(data, function(x) attributes(x)$label)))
```

```
## [1] "Q4. Since the beginning of 2009, how many hurricanes and tropical storms, if ar
## [2] "Q5. Generally speaking, when a hurricane or tropical storm is approaching your
## [3] "Q6. Since the beginning of 2009, how many times, if ever, did you leave your ho
## [4] "Q7. Generally speaking, how prepared were you for the storm(s) you experienced?
## [5] "Q10. Before Superstorm Sandy hit your area, did you leave your home to go somep
## [6] "Q50. In what year were you born?"
## [7] "Q51. Are you...?"
## [8] "Q59. Last year (in 2013), what was your total HOUSEHOLD income from all sources
```

To learn more about the `haven` package and how the variables are stored, please visit: https://haven.tidyverse.org/

Chapter 2

Extraction

2.1 Web Scraping

Web scraping is the process of fetching and extracting information / data from a webpage. It is very useful if you want to create a dynamic database that updates based on the content of a specific website.

To scrap a webpage, we first need to know how to get to the webpage, a url that you can use to directly access the content. For example, to obtain the Google search results for “data science”, you can simply copy and paste this url to your browser: https://www.google.com/search?q=data+science, without having to type “data science” on a Google search web page. Some websites like Twitter or Facebook will require to you to use an API and authenticate in order to access some of their data.

For this example, we’re going to use The Weather Channel website which do not require autentification. We’ll to extract the 10-day forecast for a specific location and store the data in a dataframe.

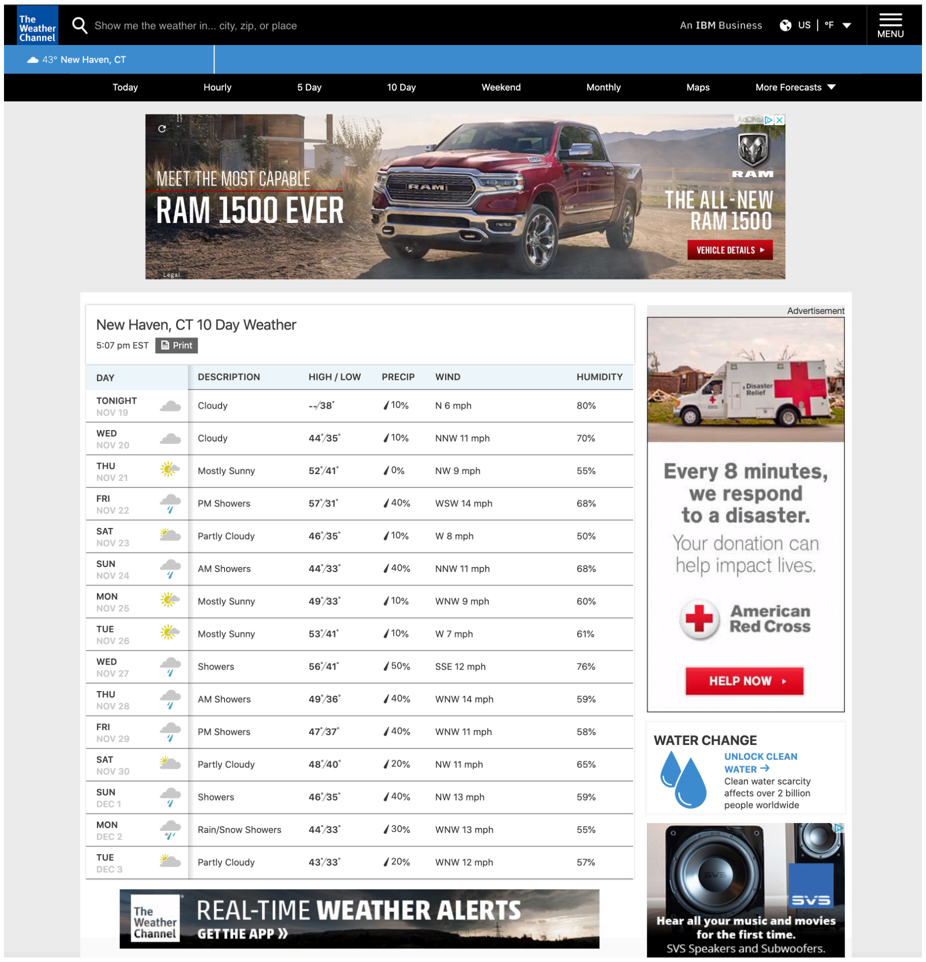
After inspecting the website and it’s url, I have noticed that you can view the weather data by zip code using this url pattern:

`https://weather.com/weather/ + forecast type + /l/ + zip_code + :4:US`

For example, if we want to view the 10-day forecast for New Haven, we can go to: https://weather.com/weather/tenday/l/06511:4:US. And for today’s forecast: https://weather.com/weather/today/l/06511:4:US

Once we have the webpage url, we can read it into R and extract the data using `rvest` from the `tidyverse` collection.

The New Haven 10-day forecast webpage looks like this:



New Haven, CT 10 Day Weather

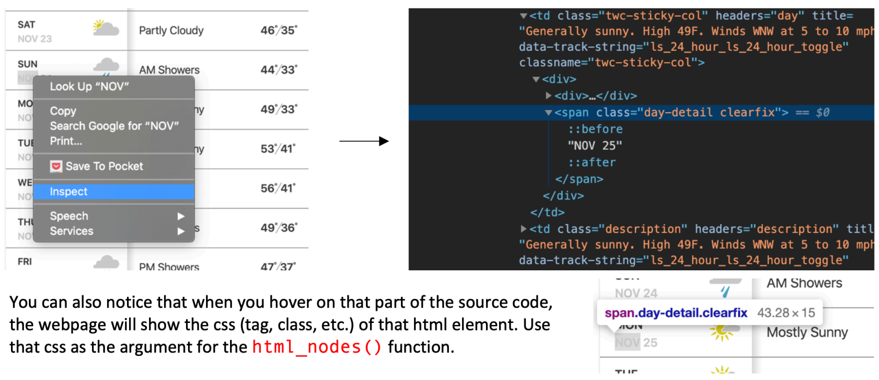
5:07 pm EST Print

DAY	DESCRIPTION	HIGH / LOW	PRECIP	WIND	HUMIDITY
TONIGHT NOV 19	Cloudy	--/38°	10%	N 6 mph	80%
WED NOV 20	Cloudy	44°/35°	10%	NNW 11 mph	70%
THU NOV 21	Mostly Sunny	52°/41°	0%	NW 9 mph	55%
FRI NOV 22	PM Showers	57°/31°	40%	WSW 14 mph	68%
SAT NOV 23	Partly Cloudy	46°/35°	10%	W 8 mph	50%
SUN NOV 24	AM Showers	44°/33°	40%	NNW 11 mph	68%
MON NOV 25	Mostly Sunny	49°/33°	10%	WNW 9 mph	60%
TUE NOV 26	Mostly Sunny	53°/41°	10%	W 7 mph	61%
WED NOV 27	Showers	56°/41°	50%	SSE 12 mph	76%
THU NOV 28	AM Showers	49°/36°	40%	WNW 14 mph	59%
FRI NOV 29	PM Showers	47°/37°	40%	WNW 11 mph	58%
SAT NOV 30	Partly Cloudy	48°/40°	20%	NW 11 mph	65%
SUN DEC 1	Showers	46°/35°	40%	NW 13 mph	59%
MON DEC 2	Rain/Snow Showers	44°/33°	30%	WNW 13 mph	55%
TUE DEC 3	Partly Cloudy	43°/33°	20%	WNW 12 mph	57%

Figure 2.1: weatherpage

Basically, what we want is the table that has the weather information. In order to extract the values that we want, we have to know where in the source code they are located. For example, in the "DAY" column, we want to extract the `exact date` instead of the `days of the week`. And we can do that by:

- inspecting the tag or class of exact date from the website. Move the cursor to the exact date, right-click, then choose `Inspect`
- then, a window will open, which will point directly to the location of the `exact date` in the source code. Take notes of the css (tag or class name), and use it to get the `exact date` value using the `html_nodes()` function.



```
▼<td class="twc-sticky-col" headers="day" title=
"Generally sunny. High 49F. Winds WNW at 5 to 10 mph
data-track-string="ls_24_hour_ls_24_hour_toggle"
classname="twc-sticky-col">
  ▼<div>
    ▶<div>…</div>
    ▼<span class="day-detail clearfix"> == $0
       ::before
       "NOV 25"
       ::after
     </span>
   </div>
 </td>
▶<td class="description" headers="description" titl
"Generally sunny. High 49F. Winds WNW at 5 to 10 mph
data-track-string="ls_24_hour_ls_24_hour_toggle"
```

You can also notice that when you hover on that part of the source code, the webpage will show the css (tag, class, etc.) of that html element. Use that css as the argument for the `html_nodes()` function.

Figure 2.2: weatherpage

Here is how we extract the dates:

```r
library(rvest)
# Get the webpage url
url = 'https://weather.com/weather/tenday/l/06511:4:US'
# Load the webpage using the url
webpage <- read_html(url)

# Getting the exact date
# Filtering the relevant css / location
date_locations <- html_nodes(webpage, "span.day-detail.clearfix")
# Extracting the exact value
raw_date <- html_text(date_locations)
raw_date
```

```
##  [1] "APR 8"  "APR 9"  "APR 10" "APR 11" "APR 12" "APR 13" "APR 14" "APR 15"
##  [9] "APR 16" "APR 17" "APR 18" "APR 19" "APR 20" "APR 21" "APR 22"
```

```r
# Because the value are formatted like "NOV 21" we have to convert to a date format
exact_date <- as.Date(raw_date, format="%b %d") # b = month, d = day
exact_date
```

```
##  [1] "2020-04-08" "2020-04-09" "2020-04-10" "2020-04-11" "2020-04-12"
##  [6] "2020-04-13" "2020-04-14" "2020-04-15" "2020-04-16" "2020-04-17"
## [11] "2020-04-18" "2020-04-19" "2020-04-20" "2020-04-21" "2020-04-22"
```

And here is the full code that extract the complete table:

```
library(rvest)
# Get the webpage url
url = 'https://weather.com/weather/tenday/l/06511:4:US'
# Load the webpage using the url
webpage <- read_html(url)

# Getting the exact date
# Filtering the relevant css / location
date_locations <- html_nodes(webpage, "span.day-detail.clearfix")
# Extracting the exact value
raw_date <- html_text(date_locations)
# Because the value are formatted like "Nov 21" we have to convert to a date format
exact_date <- as.Date(raw_date, format="%b %d") # b = month, d = day

# Getting the weather description
desc_loc <- html_nodes(webpage, "td.description")
desc <- html_text(desc_loc)

# Getting the temperature
temp_loc <- html_nodes(webpage, "td.temp")
temp <- html_text(temp_loc)
# High and Low temperature values
high_temp <- rep(NA, length(temp))
low_temp <- rep(NA, length(temp))
for (i in 1:length(temp)){
all <- unlist(strsplit(temp[i], "°"))
if (length(all) > 1){
high_temp[i] <- all[1]
low_temp[i] <- all[2]
} else {
low_temp[i] <- 38
}
}

# Getting the precipitation
precip_loc <- html_nodes(webpage, "td.precip")
precip <- as.numeric(sub("%", "", html_text(precip_loc))) / 100

# Getting the wind
wind_loc <- html_nodes(webpage, "td.wind")
```

```r
wind <- html_text(wind_loc)
# Wind direction and speed
wind_dir <- rep(NA, length(wind))
wind_speed <- rep(NA, length(wind))
for (i in 1:length(wind)){
all <- unlist(strsplit(wind[i], " "))
wind_dir[i] <- all[1]
wind_speed[i] <- all[2]
}

# Getting the humidity
humidity_loc <- html_nodes(webpage, "td.humidity")
humidity <- as.numeric(sub("%", "", html_text(humidity_loc))) / 100

# Save the data in tibble
library(tibble)
new_haven_forecast <- tibble('day' = exact_date, 'description' = desc,
                        'high_temp' = high_temp, 'low_temp' = low_temp,
                        'precip' = precip, 'wind_dir' = wind_dir,
                        'wind_speed' = wind_speed, 'himidity' = humidity)
new_haven_forecast
```

```
## # A tibble: 15 x 8
##    day        description high_temp low_temp precip wind_dir wind_speed himidity
##    <date>     <chr>       <chr>     <chr>     <dbl> <chr>    <chr>         <dbl>
##  1 2020-04-08 Mostly Cle… <NA>      38          0.1 S        4              0.9
##  2 2020-04-09 Thundersto… 59        39          1   S        18             0.81
##  3 2020-04-10 Partly Clo… 50        38          0   WNW      21             0.47
##  4 2020-04-11 Partly Clo… 52        35          0   WNW      16             0.41
##  5 2020-04-12 Partly Clo… 57        50          0.1 SSE      13             0.53
##  6 2020-04-13 Rain/Wind   64        48          0.9 S        21             0.82
##  7 2020-04-14 Partly Clo… 58        41          0.2 WNW      10             0.55
##  8 2020-04-15 Showers     51        38          0.4 NNW      9              0.53
##  9 2020-04-16 Mostly Sun… 51        38          0.2 WNW      13             0.47
## 10 2020-04-17 Partly Clo… 51        39          0.1 WNW      12             0.49
## 11 2020-04-18 Showers     51        42          0.5 NE       11             0.59
## 12 2020-04-19 Showers     56        43          0.4 WNW      12             0.59
## 13 2020-04-20 Showers     56        43          0.4 WNW      13             0.580
## 14 2020-04-21 AM Showers  57        46          0.3 NW       11             0.56
## 15 2020-04-22 Showers     59        48          0.5 WSW      11             0.61
```

Chapter 3

Structuring

Data structuring is the process of correcting or removing inaccurate records of a "raw data" so that, after the treatment, the transformed data will be easy to analyze and/or consistent with an existing dataset. More explicitly, the variable names, types, and values will be consistent and uniform. The focus here is on the 'appearance' of the data.

3.1 Inspecting the data

In order to structure a dataset, first, we need to be able to detect the anomalies within the data. Types of anomalies include the values that are stored in the wrong format (ex: a number stored as a string), the values that fall outside of the expected range (ex: outliers), values with inconsistent patterns (ex: dates stored as mm/dd/year vs dd/mm/year), trailing spaces in strings (ex: "data" vs "data"), etc.

One method of detecting these anomalies is the summary statistics of the variables, which can be obtained by using `summary()`. Here is an example using the hurricane data:

```
# Structure of the data
str(data)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1130 obs. of  9 variables:
##  $ Zone: chr  "A" "A" "A" "A" ...
##   ..- attr(*, "format.spss")= chr "A9"
##   ..- attr(*, "display_width")= int 1
##  $ Q4  : num  2 1 3 3 2 5 3 5 1 2 ...
##   ..- attr(*, "label")= chr "Q4. Since the beginning of 2009, how many hurricanes and tropical
##   ..- attr(*, "format.spss")= chr "F2.0"
##   ..- attr(*, "display_width")= int 2
## $ Q5  : 'haven_labelled' num  3 4 4 6 1 4 6 4 3 6 ...
##   ..- attr(*, "label")= chr "Q5. Generally speaking, when a hurricane or tropical st
##   ..- attr(*, "format.spss")= chr "F1.0"
##   ..- attr(*, "labels")= Named num  1 7
##   .. ..- attr(*, "names")= chr  "Not Worried At All" "Extremely Worried"
## $ Q6  : num  0 0 0 1 0 0 1 0 0 0 ...
##   ..- attr(*, "label")= chr "Q6. Since the beginning of 2009, how many times, if eve
##   ..- attr(*, "format.spss")= chr "F2.0"
##   ..- attr(*, "display_width")= int 2
## $ Q7  : 'haven_labelled' num  3 3 3 1 2 2 3 3 3 3 2 ...
##   ..- attr(*, "label")= chr "Q7. Generally speaking, how prepared were you for the s
##   ..- attr(*, "format.spss")= chr "F1.0"
##   ..- attr(*, "labels")= Named num  1 2 3 4 5
##   .. ..- attr(*, "names")= chr  "Fully Prepared" "Very Prepared" "Moderately Prepare
## $ Q10 : 'haven_labelled' num  2 2 2 2 2 2 2 2 2 2 2 ...
##   ..- attr(*, "label")= chr "Q10. Before Superstorm Sandy hit your area, did you lea
##   ..- attr(*, "format.spss")= chr "F1.0"
##   ..- attr(*, "labels")= Named num  1 2
##   .. ..- attr(*, "names")= chr  "Yes" "No"
## $ Q50 : num  1928 1962 1931 1950 1948 ...
##   ..- attr(*, "label")= chr "Q50. In what year were you born?"
##   ..- attr(*, "format.spss")= chr "F4.0"
## $ Q51 : 'haven_labelled' num  1 1 2 1 1 2 2 2 1 2 ...
##   ..- attr(*, "label")= chr "Q51. Are you...?"
##   ..- attr(*, "format.spss")= chr "F1.0"
##   ..- attr(*, "labels")= Named num  1 2
##   .. ..- attr(*, "names")= chr  "Male" "Female"
## $ Q59 : 'haven_labelled' num  4 NA 6 5 5 NA NA NA 3 6 ...
##   ..- attr(*, "label")= chr "Q59. Last year (in 2013), what was your total HOUSEHOLD
##   ..- attr(*, "format.spss")= chr "F1.0"
##   ..- attr(*, "display_width")= int 1
##   ..- attr(*, "labels")= Named num  1 2 3 4 5 6
##   .. ..- attr(*, "names")= chr  "Less than $15,000" "$15,000-$39,999" "$40,000-$69,9
```

```r
# Summary for a numerical variables
summary(data$Q4)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.000   2.000   2.000   2.537   3.000  20.000     134
```

```r
# Summary for a categorical variable
summary(as_factor(data$Q7))
```

```
##      Fully Prepared       Very Prepared Moderately Prepared   A Little Prepared
##                  93                 326                 438                 137
## Not at all Prepared                NA's
##                  22                 114
```

Other ways of exploring the data include:

```r
# First 10 rows
head(data, 10)
```

```
## # A tibble: 10 x 9
##    Zone     Q4           Q5    Q6           Q7     Q10   Q50     Q51         Q59
##    <chr> <dbl>    <dbl+lbl> <dbl>    <dbl+lbl> <dbl+l> <dbl> <dbl+l>   <dbl+lbl>
##  1 A         2 3                0 3 [Moderate…  2 [No]  1928 1 [Mal…  4 [$70,00…
##  2 A         1 4                0 3 [Moderate…  2 [No]  1962 1 [Mal… NA
##  3 A         3 4                0 3 [Moderate…  2 [No]  1931 2 [Fem…  6 [Over $…
##  4 A         3 6                1 1 [Fully Pr…  2 [No]  1950 1 [Mal…  5 [$100,0…
##  5 A         2 1 [Not Worr…     0 2 [Very Pre…  2 [No]  1948 1 [Mal…  5 [$100,0…
##  6 A         5 4                0 2 [Very Pre…  2 [No]  1938 2 [Fem… NA
##  7 A         3 6                1 3 [Moderate…  2 [No]  1977 2 [Fem… NA
##  8 A         5 4                0 3 [Moderate…  2 [No]  1964 2 [Fem… NA
##  9 A         1 3                0 3 [Moderate…  2 [No]  1976 1 [Mal…  3 [$40,00…
## 10 A         2 6                0 2 [Very Pre…  2 [No]  1964 2 [Fem…  6 [Over $…
```

```r
# Last 10 rows
tail(data, 10)
```

```
## # A tibble: 10 x 9
##    Zone     Q4      Q5    Q6             Q7      Q10   Q50      Q51         Q59
##    <chr> <dbl> <dbl+l> <dbl>      <dbl+lbl> <dbl+lb> <dbl> <dbl+lb>    <dbl+lbl>
##  1 B         1       2     0 4 [A Little P…  2 [No]   1980 1 [Male] 3 [$40,000-…
##  2 B         2       2     0 3 [Moderately…  2 [No]   1977 2 [Fema… 4 [$70,000-…
##  3 B         4       4     1 2 [Very Prepa…  1 [Yes]  1962 2 [Fema… 2 [$15,000-…
##  4 B         2       5     0 1 [Fully Prep…  2 [No]   1946 1 [Male] 5 [$100,000…
##  5 B        NA       4    NA 1 [Fully Prep…  1 [Yes]  1957 2 [Fema… 1 [Less tha…
##  6 B         1       4     1 4 [A Little P…  1 [Yes]  1987 2 [Fema… 6 [Over $20…
##  7 B         2       5     0 3 [Moderately…  2 [No]   1953 1 [Male] 4 [$70,000-…
##  8 B        NA       4     4 2 [Very Prepa…  2 [No]   1973 2 [Fema… 1 [Less tha…
##  9 B         2       5     0 3 [Moderately…  2 [No]   1980 1 [Male] 5 [$100,000…
## 10 B         2       2     0 4 [A Little P…  2 [No]     NA 2 [Fema… 3 [$40,000-…
```

```r
# Total number of rows
nrow(data)
```

```
## [1] 1130
```

```r
# Total number of columns
ncol(data)
```

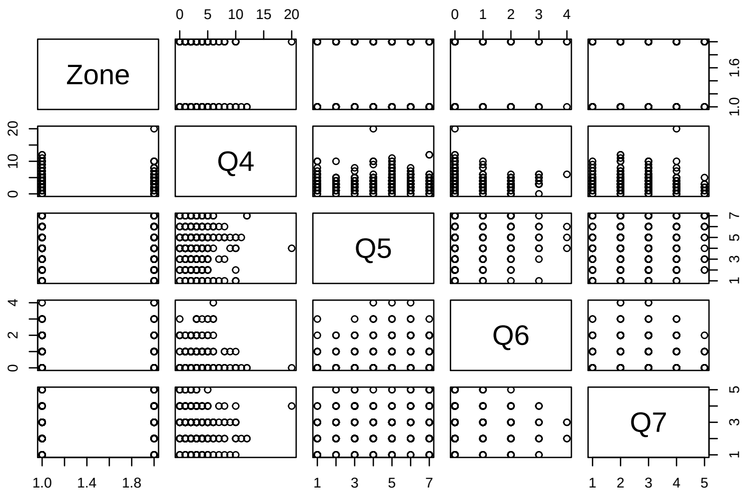
```
## [1] 9
```

```r
# Column names
names(data) # also colnames(data)
```

```
## [1] "Zone" "Q4"   "Q5"   "Q6"   "Q7"   "Q10"  "Q50"  "Q51"  "Q59"
```

We can also plot the data to visualize the distribution of variables

```
# Plotting the first 5 columns
plot(data[,1:5])
```



While these plots could help in understanding the dataset, they could be misleading if the variables are not set to their correct data type.

3.2 Data types

One type of anomaly that we may also encounter is the coercion of irrelevant data types to a variables. This is very common for numerically coded variables or ones that has levels.

For example, if we read in the same SPSS data from the Reading data section, we get the coded values instead of the labels.

```
## # A tibble: 6 x 9
##   Zone     Q4          Q5    Q6          Q7     Q10   Q50     Q51         Q59
##   <chr> <dbl>   <dbl+lbl> <dbl>   <dbl+lbl> <dbl+l> <dbl> <dbl+l>   <dbl+lbl>
## 1 A         2 3              0 3 [Moderate…  2 [No]  1928 1 [Mal…  4 [$70,000…
## 2 A         1 4              0 3 [Moderate…  2 [No]  1962 1 [Mal… NA
## 3 A         3 4              0 3 [Moderate…  2 [No]  1931 2 [Fem…  6 [Over $2…
## 4 A         3 6              1 1 [Fully Pr…  2 [No]  1950 1 [Mal…  5 [$100,00…
## 5 A         2 1 [Not Worr…   0 2 [Very Pre…  2 [No]  1948 1 [Mal…  5 [$100,00…
## 6 A         5 4              0 2 [Very Pre…  2 [No]  1938 2 [Fem… NA
```

So if we run `summary(data)` right away then we'll get this unintended result:

```
##       Zone                 Q4               Q5              Q6
##  Length:1130        Min.    : 0.000   Min.    :1.000   Min.    :0.0000
##  Class :character   1st Qu.: 2.000   1st Qu.:3.000   1st Qu.:0.0000
##  Mode  :character   Median : 2.000   Median :4.000   Median :0.0000
##                     Mean   : 2.537   Mean   :4.235   Mean   :0.4191
##                     3rd Qu.: 3.000   3rd Qu.:5.000   3rd Qu.:1.0000
##                     Max.   :20.000   Max.   :7.000   Max.   :4.0000
##                     NA's   :134      NA's   :111     NA's   :116
##        Q7              Q10              Q50             Q51             Q59
##  Min.   :1.000   Min.   :1.000   Min.   :  19   Min.   :1.00   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:1944   1st Qu.:1.00   1st Qu.:3.000
##  Median :3.000   Median :2.000   Median :1955   Median :2.00   Median :4.000
##  Mean   :2.674   Mean   :1.796   Mean   :1944   Mean   :1.55   Mean   :3.715
##  3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.:1966   3rd Qu.:2.00   3rd Qu.:5.000
##  Max.   :5.000   Max.   :2.000   Max.   :1992   Max.   :2.00   Max.   :6.000
##  NA's   :114     NA's   :123     NA's   :47     NA's   :38     NA's   :112
```

`Q4` and `Q50` are the only variables that are supposed to be numeric. But here everything is treated as numeric which is incorrect. Also, it is best if we read `Zone` as factor as well so that we find out the possible values.

We can easily convert data types into factor using `dplyr::mutate_at()` and applying `as.factor` function to the variables.

```r
# Converting data types
updated_data <- data %>% mutate_at(vars(-Q4, -Q6, -Q50), as_factor)
```

And now we can get the full summary statistics that we want:

```
##  Zone          Q4               Q5            Q6
##  A:684   Min.   : 0.000   5      :244   Min.   :0.0000
##  B:446   1st Qu.: 2.000   4      :211   1st Qu.:0.0000
##          Median : 2.000   3      :169   Median :0.0000
##          Mean   : 2.537   6      :129   Mean   :0.4191
##          3rd Qu.: 3.000   2      :104   3rd Qu.:1.0000
##          Max.   :20.000   (Other):162   Max.   :4.0000
##          NA's   :134      NA's   :111   NA's   :116
##                         Q7         Q10           Q50            Q51
##  Fully Prepared      : 93   Yes :205   Min.   :  19   Male  :491
##  Very Prepared       :326   No  :802   1st Qu.:1944   Female:601
##  Moderately Prepared:438   NA's:123   Median :1955   NA's  : 38
##  A Little Prepared   :137              Mean   :1944
##  Not at all Prepared: 22              3rd Qu.:1966
##  NA's                :114              Max.   :1992
##                                        NA's   :47
##                  Q59
##  Less than $15,000: 81
##  $15,000-$39,999  :169
##  $40,000-$69,999  :215
##  $70,000-$99,999  :190
##  $100,000-$199,999:220
##  Over $200,000    :143
##  NA's             :112
```

As we can see from the summary, there might be some anomalies with the variables:

- `Zone`: as most of the respondents are from Zone A. But this is basically related to the survey method which would later require that some weighting of the variables would be applied.
- `Q4: Number of storms experienced`: where the mean value is 2.5 but some response have the value of 20.
- `Q50: Birth year`: where some respondent answered 19 which is incorrect. Also this column is probably better if it's in age instead of birth year.

We can also notice some missing values.

3.3 Subsetting and Filtering

We can remove incorrect or missing row values by using `dplyr::filter`:

```
# Removing rows where birth year is irrelevant
# Here we decided that all birth year must be greater 1900
updated_data <- data %>% filter(Q50 > 1900)
# Now if we re-run its summary
summary(updated_data$Q50)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1908    1945    1955    1956    1966    1992
```

```
# Removing rows with birth year greater than 1900 and missing responses for Q4
updated_data <- data %>% filter(Q50 > 1900, !is.na(Q4))
summary(updated_data$Q50)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1908    1944    1954    1955    1965    1990
```

```
summary(updated_data$Q4)
```

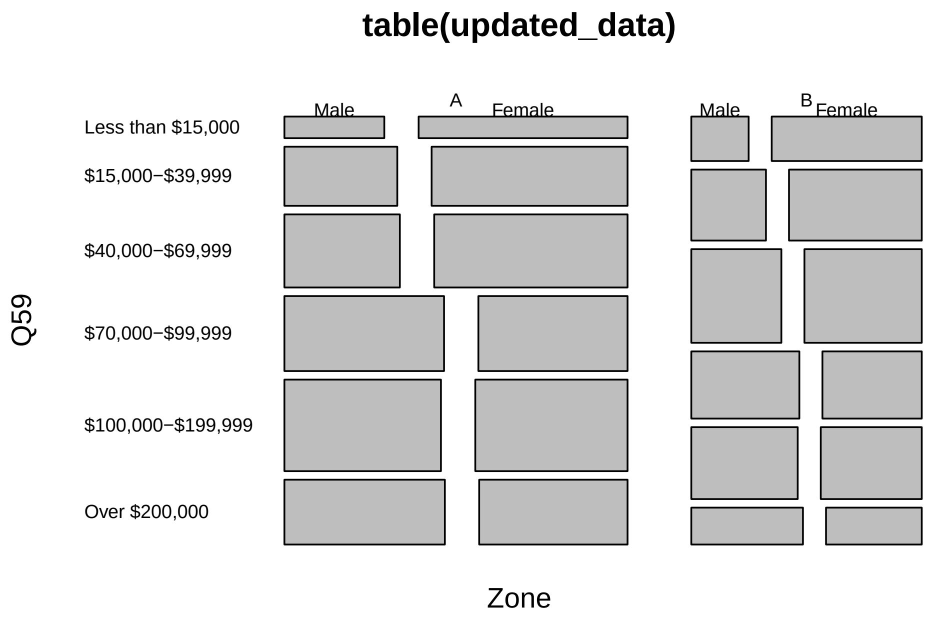
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   2.000   2.000   2.522   3.000  20.000
```

We can also select only the variables that we are interested in using `dplyr::select`:

```
# Creating a new dataframe with only zone, gender, and income column
updated_data <- data %>% select(Zone, Q59, Q51)
head(updated_data, 10)
```

```
## # A tibble: 10 x 3
##    Zone  Q59               Q51
##    <fct> <fct>              <fct>
##  1 A     $70,000-$99,999    Male
##  2 A     <NA>               Male
##  3 A     Over $200,000      Female
##  4 A     $100,000-$199,999 Male
##  5 A     $100,000-$199,999 Male
##  6 A     <NA>               Female
##  7 A     <NA>               Female
##  8 A     <NA>               Female
##  9 A     $40,000-$69,999    Male
## 10 A     Over $200,000      Female
```

```
plot(table(updated_data), las=1)
```



It is also possible to split the dataset into multiple dataframe by number of rows using `split()`.

```
# To split the dataset into multiple dataframe of 10 rows each
max_number_of_rows_per_dataframe <- 10
total_number_rows_in_the_current_dataset <- nrow(data)
sets_of_10rows_dataframes <- split(data,
                                   rep(1:ceiling(total_number_rows_in_the_current_dataset/max_num
                                       each=max_number_of_rows_per_dataframe,
                                       length.out=total_number_rows_in_the_current_dataset)
                                   )
# Here are the first 2 dataframes
sets_of_10rows_dataframes[[1]] # or sets_of_10rows_dataframes$`1`
```

```
## # A tibble: 10 x 9
##    Zone     Q4 Q5               Q6 Q7             Q10     Q50 Q51   Q59
##    <fct> <dbl> <fct>         <dbl> <fct>          <fct> <dbl> <fct> <fct>
##  1 A         2 3                 0 Moderately Pr… No     1928 Male  $70,000-$99…
##  2 A         1 4                 0 Moderately Pr… No     1962 Male  <NA>
##  3 A         3 4                 0 Moderately Pr… No     1931 Fema… Over $200,0…
##  4 A         3 6                 1 Fully Prepared No     1950 Male  $100,000-$1…
##  5 A         2 Not Worried …     0 Very Prepared  No     1948 Male  $100,000-$1…
##  6 A         5 4                 0 Very Prepared  No     1938 Fema… <NA>
##  7 A         3 6                 1 Moderately Pr… No     1977 Fema… <NA>
##  8 A         5 4                 0 Moderately Pr… No     1964 Fema… <NA>
##  9 A         1 3                 0 Moderately Pr… No     1976 Male  $40,000-$69…
## 10 A         2 6                 0 Very Prepared  No     1964 Fema… Over $200,0…
```

```
sets_of_10rows_dataframes[[2]]
```

```
## # A tibble: 10 x 9
##    Zone     Q4 Q5               Q6 Q7             Q10     Q50 Q51   Q59
##    <fct> <dbl> <fct>         <dbl> <fct>          <fct> <dbl> <fct> <fct>
##  1 A         2 Extremely Wo…     2 Fully Prepared Yes    1937 Fema… <NA>
##  2 A         3 5                 0 Very Prepared  No     1943 Male  $70,000-$99…
##  3 A         2 Extremely Wo…     0 Very Prepared  No     1954 Fema… $100,000-$1…
##  4 A         2 5                 0 Very Prepared  No     1959 Fema… $100,000-$1…
##  5 A         4 Not Worried …    NA Very Prepared  No     1936 Fema… Over $200,0…
##  6 A         1 3                 1 Moderately Pr… Yes    1963 Male  Over $200,0…
##  7 A         2 3                 1 Very Prepared  Yes    1950 Fema… $100,000-$1…
##  8 A         4 6                 0 Moderately Pr… No       NA <NA>  <NA>
##  9 A         0 4                 0 Very Prepared  No     1941 Male  $100,000-$1…
## 10 A        NA <NA>             NA <NA>           <NA>   1952 Fema… $100,000-$1…
```

3.4 Changing cell values

As we mentionned earlier, it is best if Q50 is stored as an age variable instead of the default birth year. `Q50` is a numeric variable and we can simply change it by using `dplyr::mutate()`

```
# Replacing Q50 values to their age in 2020
updated_data <- data %>% mutate(Q50 = 2020 - Q50)
head(updated_data, 10)
```

```
## # A tibble: 10 x 9
##    Zone     Q4 Q5               Q6 Q7             Q10     Q50 Q51   Q59
##    <fct> <dbl> <fct>         <dbl> <fct>          <fct> <dbl> <fct> <fct>
##  1 A         2 3                 0 Moderately Pr… No       92 Male  $70,000-$99…
##  2 A         1 4                 0 Moderately Pr… No       58 Male  <NA>
##  3 A         3 4                 0 Moderately Pr… No       89 Fema… Over $200,0…
##  4 A         3 6                 1 Fully Prepared No       70 Male  $100,000-$1…
##  5 A           2 Not Worried …     0 Very Prepared   No        72 Male  $100,000-$1…
##  6 A           5 4                 0 Very Prepared   No        82 Fema… <NA>
##  7 A           3 6                 1 Moderately Pr… No        43 Fema… <NA>
##  8 A           5 4                 0 Moderately Pr… No        56 Fema… <NA>
##  9 A           1 3                 0 Moderately Pr… No        44 Male  $40,000-$69…
## 10 A           2 6                 0 Very Prepared   No        56 Fema… Over $200,0…
```

```
# It is also possible to leave Q50 untouched and store the results into a new column
updated_data <- data %>% mutate(age = 2020 - Q50)
head(updated_data, 10)
```

```
## # A tibble: 10 x 10
##    Zone     Q4 Q5               Q6 Q7            Q10     Q50 Q51   Q59           age
##    <fct> <dbl> <fct>         <dbl> <fct>         <fct> <dbl> <fct> <fct>       <dbl>
##  1 A         2 3                 0 Moderately … No     1928 Male  $70,000-$…     92
##  2 A         1 4                 0 Moderately … No     1962 Male  <NA>           58
##  3 A         3 4                 0 Moderately … No     1931 Fema… Over $200…     89
##  4 A         3 6                 1 Fully Prepa… No     1950 Male  $100,000-…     70
##  5 A         2 Not Worrie…       0 Very Prepar… No     1948 Male  $100,000-…     72
##  6 A         5 4                 0 Very Prepar… No     1938 Fema… <NA>           82
##  7 A         3 6                 1 Moderately … No     1977 Fema… <NA>           43
##  8 A         5 4                 0 Moderately … No     1964 Fema… <NA>           56
##  9 A         1 3                 0 Moderately … No     1976 Male  $40,000-$…     44
## 10 A         2 6                 0 Very Prepar… No     1964 Fema… Over $200…     56
```

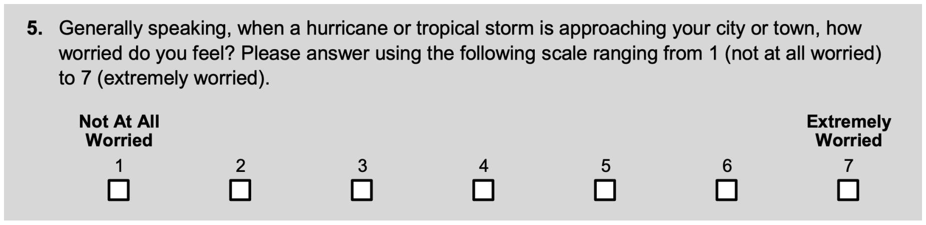
```
summary(updated_data$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   28.00   54.00   65.00   64.14   75.00  112.00
```

For a categorical variable, we use a different function `dplyr::recode_factor()` or `dplyr::recode()`. We will apply this to `Q5` as we have noticed in the previous section that not all of its values were labelled from SPSS. Here is its summary:

```
## Not Worried At All                  2                  3                  4
##                 58                101                164                197
##                  5                  6  Extremely Worried               NA's
##                232                123                 97                104
```

Looking back at the questionnare, here is how it was phrased:



5. Generally speaking, when a hurricane or tropical storm is approaching your city or town, how worried do you feel? Please answer using the following scale ranging from 1 (not at all worried) to 7 (extremely worried).

Not At All Worried						Extremely Worried
1	2	3	4	5	6	7
☐	☐	☐	☐	☐	☐	☐

Because the survey itself doesn't have labels, the recoding will be up to the user. Here we chose to remove replace the extreme values with 1 and 7. As mentionned in the documentation: `dplyr::recode()` will preserve the existing order of levels while changing the values, and `dplyr::recode_factor()` will change the order of levels to match the order of replacements.

```
# Recoding Q5
recoded.with.recode <- recode(data$Q5, `Not Worried At All`="1", `Extremely Worried`="
summary(recoded.with.recode)
```

```
##    1    2    3    4    5    6    7 NA's
##   58  101  164  197  232  123   97  104
```

```
recoded.with.recode_factor <- recode_factor(data$Q5, `Not Worried At All`="1", `Extrem
summary(recoded.with.recode_factor)
```

```
##    1    7    2    3    4    5    6 NA's
##   58   97  101  164  197  232  123  104
```

We can also change cell values without external libraries like `dplyr` by running the following code:

```
# Add column age where the values are 2020 - Q50
data$age <- 2020 - data$Q50
# Replace Q5 with value "Not Worried At All" to "1"
data$Q5[data$Q5 == "Not Worried At All"] <- 1
```

```
## Warning in `[<-.factor`(`*tmp*`, data$Q5 == "Not Worried At All", value =
## structure(c(3L, : invalid factor level, NA generated
```

3.5 Pivoting the dataset

In some cases, we may want to split a column based on values, or merge multiple columns into fewer columns. These process can be done using `tidyr` package. For example, to convert the dataframe into long-format with only `Zone`, `question`, and `value` as columns:

```
library(tidyr)
# We have to pivot by variable type
# Pivot longer for factor variables
pivoted.longer <- data %>%
  select_if(is.factor) %>%
  pivot_longer(-Zone, names_to = "question", values_to = "value")
pivoted.longer
```

```
## # A tibble: 5,380 x 3
##    Zone  question value
##    <fct> <chr>    <fct>
##  1 A     Q5       3
##  2 A     Q7       Moderately Prepared
##   3 A     Q10      No
##   4 A     Q51      Male
##   5 A     Q59      $70,000-$99,999
##   6 A     Q5       4
##   7 A     Q7       Moderately Prepared
##   8 A     Q10      No
##   9 A     Q51      Male
##  10 A     Q59      <NA>
## # … with 5,370 more rows
```

```
# Then we can reshape it back to the original
pivoted.wider <- pivoted.longer %>%
  group_by(question) %>% mutate(row = row_number()) %>%
  pivot_wider(names_from = question, values_from = value) %>%
  select(-row)
pivoted.wider
```

```
## # A tibble: 1,076 x 6
##    Zone  Q5    Q7                  Q10   Q51    Q59
##    <fct> <fct> <fct>               <fct> <fct>  <fct>
##  1 A     3     Moderately Prepared No    Male   $70,000-$99,999
##  2 A     4     Moderately Prepared No    Male   <NA>
##  3 A     4     Moderately Prepared No    Female Over $200,000
##  4 A     6     Fully Prepared      No    Male   $100,000-$199,999
##  5 A     <NA>  Very Prepared       No    Male   $100,000-$199,999
##  6 A     4     Very Prepared       No    Female <NA>
##  7 A     6     Moderately Prepared No    Female <NA>
##  8 A     4     Moderately Prepared No    Female <NA>
##  9 A     3     Moderately Prepared No    Male   $40,000-$69,999
## 10 A     6     Very Prepared       No    Female Over $200,000
## # … with 1,066 more rows
```

`tidyr::spread()` and `tidyr::gather()` are the outdated equivalent of `tidyr::pivot_wider()` and `tidyr::pivot_longer()`.

To merge or split columns, we can use `tidyr::unite()` or `tidyr::separate()`. For example, to merge `Q7` and `Q10`:

```
# Creating a new column with responses from both Q7 and Q10
merged <- data %>% unite("Q7_Q10", Q7:Q10, sep = "__", remove = TRUE, na.rm = FALSE)
merged
```

```
## # A tibble: 1,076 x 9
##    Zone     Q4 Q5       Q6 Q7_Q10                  Q50 Q51   Q59               age
##    <fct> <dbl> <fct> <dbl> <chr>                 <dbl> <fct> <fct>           <dbl>
##  1 A         2 3         0 Moderately Prepared…  1928 Male  $70,000-$99,9…     92
##  2 A         1 4         0 Moderately Prepared…  1962 Male  <NA>               58
##  3 A         3 4         0 Moderately Prepared…  1931 Fema… Over $200,000      89
##   4 A          3 6          1 Fully Prepared__No    1950 Male  $100,000-$199…    70
##   5 A          2 <NA>       0 Very Prepared__No     1948 Male  $100,000-$199…    72
##   6 A          5 4          0 Very Prepared__No     1938 Fema… <NA>              82
##   7 A          3 6          1 Moderately Prepared…  1977 Fema… <NA>              43
##   8 A          5 4          0 Moderately Prepared…  1964 Fema… <NA>              56
##   9 A          1 3          0 Moderately Prepared…  1976 Male  $40,000-$69,9…    44
##  10 A          2 6          0 Very Prepared__No     1964 Fema… Over $200,000     56
## # … with 1,066 more rows
```

```
# To split it back
merged %>% separate(Q7_Q10, c("Q7", "Q10"), sep = "__", remove = TRUE)
```

```
## # A tibble: 1,076 x 10
##    Zone     Q4 Q5       Q6 Q7               Q10     Q50 Q51   Q59              age
##    <fct> <dbl> <fct> <dbl> <chr>            <chr> <dbl> <fct> <fct>          <dbl>
##  1 A         2 3         0 Moderately Pre… No     1928 Male  $70,000-$99,…     92
##  2 A         1 4         0 Moderately Pre… No     1962 Male  <NA>              58
##  3 A         3 4         0 Moderately Pre… No     1931 Fema… Over $200,000     89
##  4 A         3 6         1 Fully Prepared  No     1950 Male  $100,000-$19…     70
##  5 A         2 <NA>      0 Very Prepared   No     1948 Male  $100,000-$19…     72
##  6 A         5 4         0 Very Prepared   No     1938 Fema… <NA>              82
##  7 A         3 6         1 Moderately Pre… No     1977 Fema… <NA>              43
##  8 A         5 4         0 Moderately Pre… No     1964 Fema… <NA>              56
##  9 A         1 3         0 Moderately Pre… No     1976 Male  $40,000-$69,…     44
## 10 A         2 6         0 Very Prepared   No     1964 Fema… Over $200,000     56
## # … with 1,066 more rows
```

Chapter 4

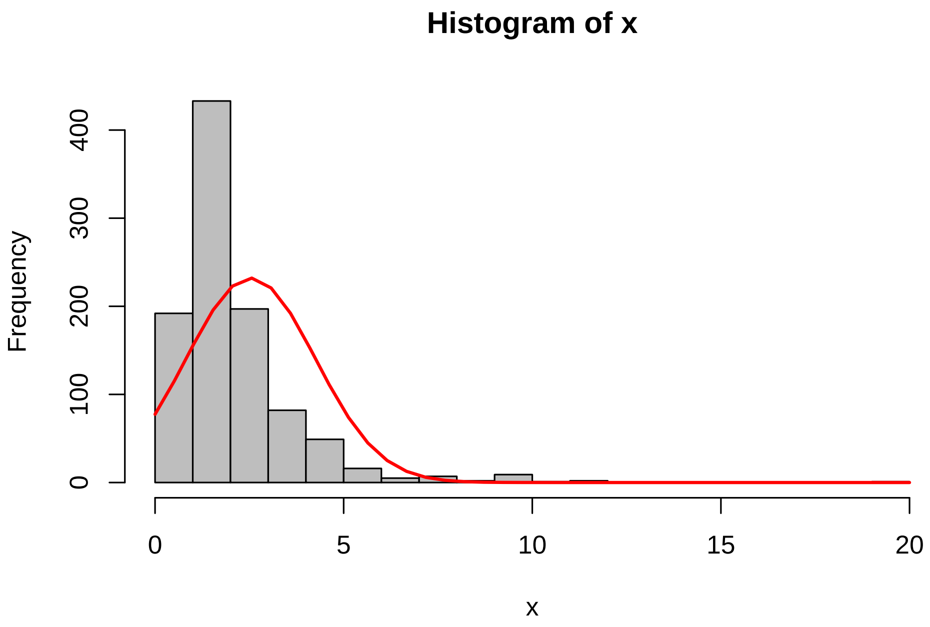
Cleaning

Data cleaning is the process of getting the data ready for statistical analysis. In contrast to “Structuring the data” the target anomalies in this case are the varibale values such as missing values, outliers, distribution, etc.

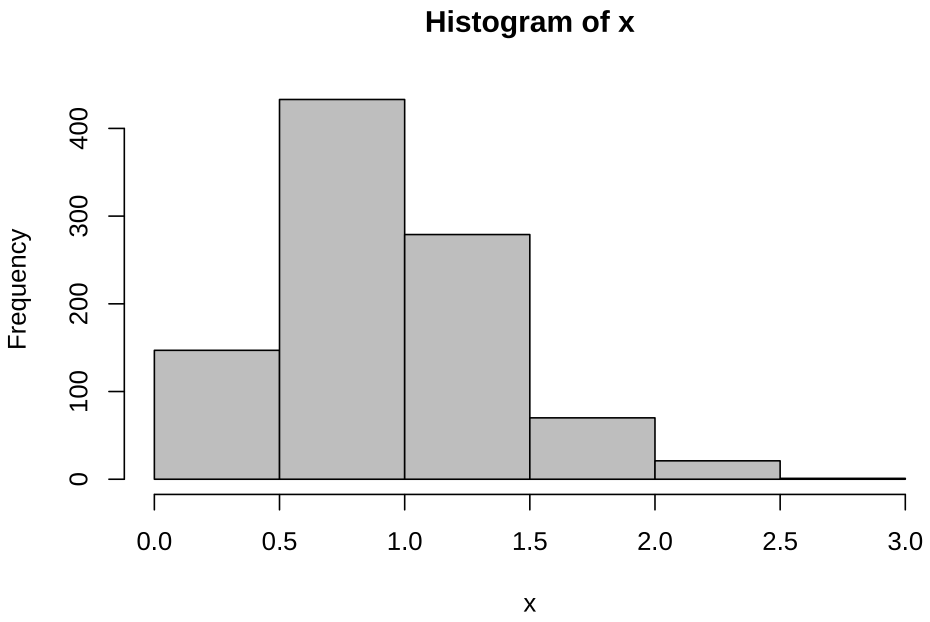
4.1 Fixing skewed distribution

A data is skewed when it’s distribution is not symetrical but rather distored on the left or right. Sometimes, to facilitate statistical analysis, we need to transform that skewed data so that it becomes normally distributed instead.

```r
# Example of skewed data
{x <- data$Q4
h <- hist(x, breaks = 20, col = "grey")
xfit <- seq(min(x),max(x),length=40)
yfit <- dnorm(xfit,mean=mean(x),sd=sd(x))
yfit <- yfit*diff(h$mids[1:2])*length(x)
lines(xfit, yfit, col="red", lwd=2)}
```



```
# Log-transformation
{x <- log(data$Q4)
h <- hist(x, breaks = 10, col = "grey")}
```



For `nstorm` and `nevac`, we can better investigate what's going on by actually visualizing them in a histogram using the `hist()` or `boxplot()`.

```r
par(mfrow=c(2,2)) # 4 figures arranged in 2 rows and 2 columns
hist(hurricane$nstorm, breaks=10, xlab="Number of storms", main=NA)
boxplot(hurricane$nstorm)
hist(hurricane$nevac, breaks=10, xlab="Number of evacuations", main=NA)
boxplot(hurricane$nevac)
mtext("How many storms have you experienced?", side=3, outer=TRUE, line=-2)
mtext("How many times have you evacuated?", side=3, outer=TRUE, line=-16)
```

Using `select()`, `mutate()`, `filter()`, etc.

4.2 Fixing outliers

As we can see, both variables are not normally distributed but skewed. And there are several method of treating such variables based on the objective of the analysis: log-transformation, conversion to categorical variables, or simply removing the outliers, etc.

4.3 Fixing missing values

We can also notice from the summary above that the there are missing values (`NA`) as well. They can also be detected using `anyNA()`. And the best way to treat them is by removing all of the corresponding observations using `drop_na()` from the `tidyr` package. Or, in some cases, removing the variable itself.

```r
tidyr::drop_na(hurricane)
```

However, if dropping all of the rows with missing values affect the quality of the data, then another option is to replace the missing values with the mean/median/mode of the variable or predict using an appropriate algorithm. There are several packages out there that are solely dedicated to treating missing values including `VIM` and `MICE`.

In this next example, we'll try to predict the 15 missing values in the variable `nstorm` (number of storms the survey respondents have experienced) using the variables that has no missing values: `zone`, `lat`, and `long`.

```r
# Imputation using MICE
library(mice)

# Building the mice model
mice_model <- mice(select(hurricane, zone, lat, long, nstorm), method="rf") # select() is from th
# Predicting the missing values
mice_prediction <- complete(mice_model)  # generate the completed data.
anyNA(mice_prediction)
```

Then we can visualize the data to see how well the imputation has performed.

However, the best way to assess the accuracy is to compare actual values with predicted values using measures such as: `MSE`, `MAE`, `MAPE`, etc.

```
# Visualizing the prediction
non_na_latitude <- hurricane$lat[!is.na(hurricane$nstorm)]
non_na_nstorm <- hurricane$nstorm[!is.na(hurricane$nstorm)]
na_latitude <- mice_prediction$lat[is.na(hurricane$nstorm)]
na_nstorm <- mice_prediction$nstorm[is.na(hurricane$nstorm)]
plot(non_na_nstorm, non_na_latitude, col="grey", pch="•", ylab="Latitude", xlab="Numbe
points(na_nstorm, na_latitude, col="red", pch="•", cex=2)
legend("topright", c("Existing values", "Predicted missing values"), col=c("grey", "re
```

Chapter 5

Visualization

5.1 Simple graph with `ggplot2`

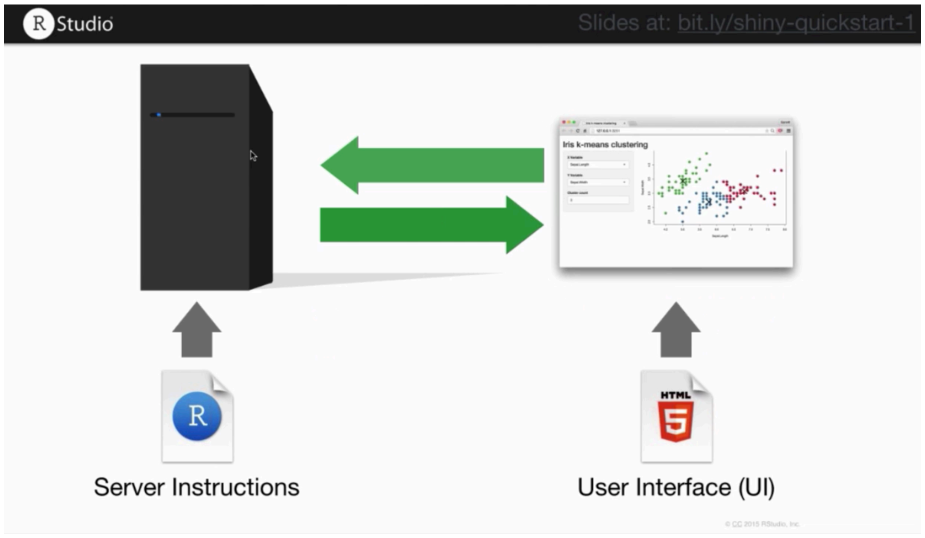
5.2 Interactive graph with `plotly` and `gganimate`

5.3 Web app with RShiny

This Shiny tutorial was edited based on the official tutorial on the website

With RShiny, it is possible to make the functions in your R script available to people who don't necessarily know R. For example, in this app, you can create different types of graph by selecting a site and other parameters. Basically, it's web development using the power of R libraries.

So, just like any web tools, an RshinyApp has components on the User and



Server side.

The visual appearance of the app can be modified in the user component. You can use it to change layout, font size, color, etc. Whereas on the server side, you can customize how your app responds to user inputs/interactions.

A basic **ShinyApp starter template** looks like this:

```r
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui=ui, server=server)
```

If you run the script above, it will give you and empty page. Some content will show when you add some (text) elements with `fluidPage`.

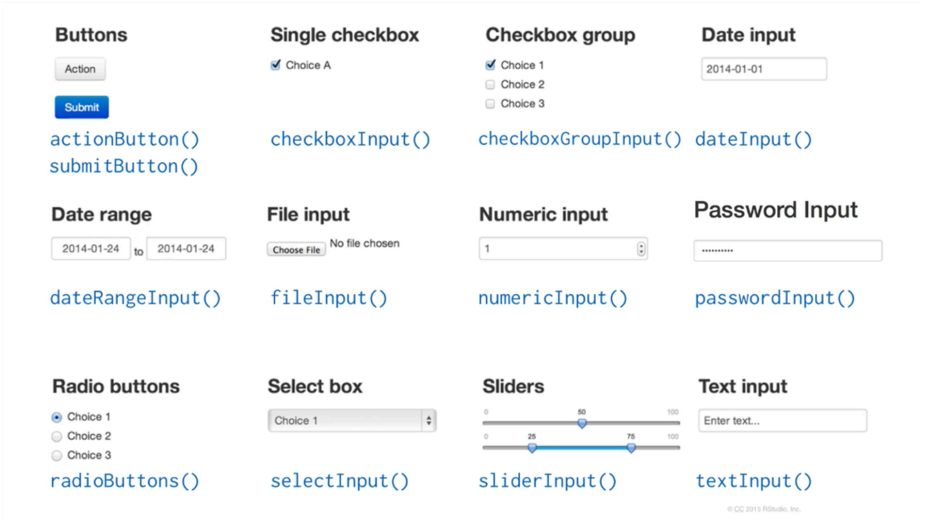
```r
ui <- fluidPage('Hello world')
server <- function(input, output){}
shinyApp(ui=ui, server=server)
```

5.3.1 Input functions

For richer content, the `shiny` package has built-in functions that will allow you create them. `sliderInput` for example adds a slider in your web app.

```r
ui <- fluidPage(
  sliderInput(inputId='num',
              label='Choose a number',
              value=25, min=1, max=100)
)
server <- function(input, output){}
shinyApp(ui=ui, server=server)
```

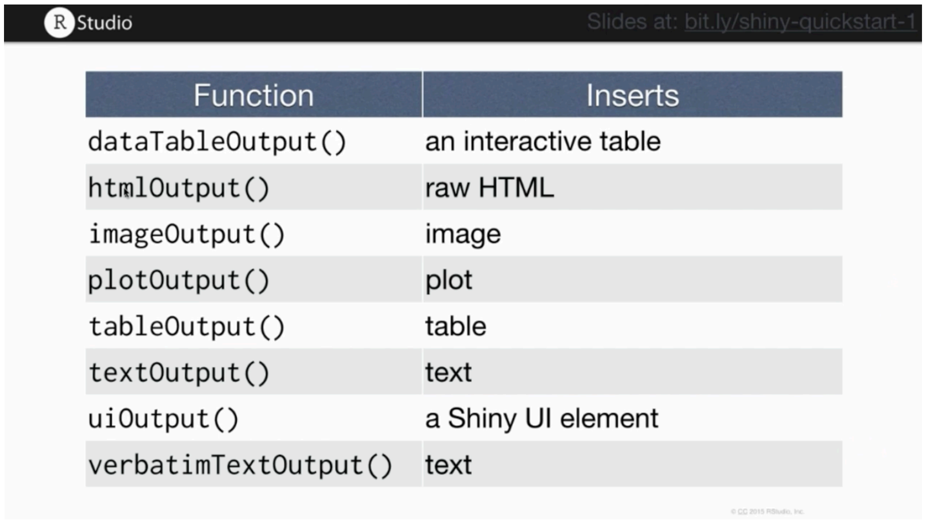
`sliderInput` is part of a group of function called **inputs**. These functions allow an Rshiny developper like yourself to add html element that will serve as user input. Here are some input functions you can try:



All input functions take as first 2 arguments: `inputId` and `label`. `inputId` is an ID that R will use create the input element in a webpage and to reference it later. `label` is just a text that the user will see, describing what the input element is for. The rest of the arguments are function-specific.

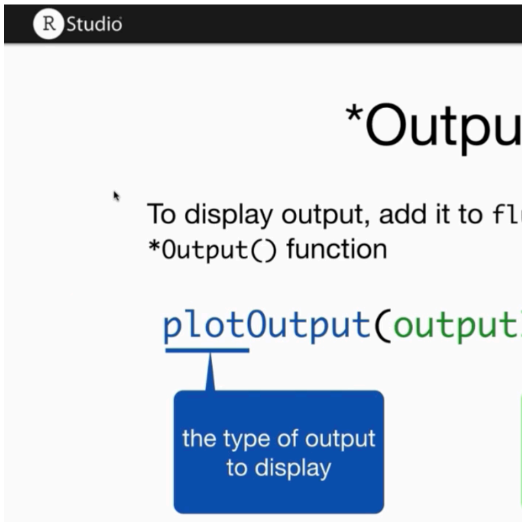
5.3.2 Output functions

Output functions, on the other hand, allow you to add outputs of R into your web page. As you know, these outputs can be image, plot, table, text, etc. And there are specific output functions for each type of output.


Slides at: bit.ly/shiny-quickstart-1

Function	Inserts
dataTableOutput()	an interactive table
htmlOutput()	raw HTML
imageOutput()	image
plotOutput()	plot
tableOutput()	table
textOutput()	text
uiOutput()	a Shiny UI element
verbatimTextOutput()	text

Figure 5.1: Rshiny output functions


*Outpu

To display output, add it to fl
*Output() function

plotOutput(output

the type of output to display

Output functions are called similarly to the input functions:

Here is our app with an output function:

```
ui <- fluidPage(
  sliderInput(inputId='num',
              label='Choose a number',
              value=25, min=1, max=100),
  plotOutput('hist')
)
server <- function(input, output){}
```

```
shinyApp(ui=ui, server=server)
```

Running the previous script won't show any plot though. It just reserve a space in the webpage for plot with id: 'hist'. To actually create the plot, you must use the server function.

5.3.3 Server function

Server function creates the interactivity in your web application, i.e. this is where you set up how an output (ex: graph) chnages with the user input (ex: some number). However, there are 3 rules that has to be followed and it is summarized in this script:

```
function(input, output){
  #1 outputId is the outputId you defined with output function
  #2 to render on a webpage. To render a plot, use renderPlot()
  output$outputId <- renderSomething({
    #3 inputId is the inputId you defined with input function
    someFunction(input$inputId)
  })
}
# Anything inside the curly braces is an R code. And it can be multiple lines of code.
```

In summary, you must define output variable by the `outputId` and prefixing it with `output$`. You must call the R script that will produce the output with a `render*({})` function. And you must call the user input with its `inputId` and prefixing it with `input$`.

In our example script, we can render it as follow:

```
ui <- fluidPage(
  sliderInput(inputId='num',
              label='Choose a number',
              value=25, min=1, max=100),
  plotOutput('hist')
)
server <- function(input, output){
    output$hist <- renderPlot({
      title <- paste(input$num, 'random normal values')
      hist(rnorm(input$num), main=title, xlab='values')
  })
}
shinyApp(ui=ui, server=server)
```

To make much more advanced app, simply read the instructions on Rshiny website.

5.3.4 Sharing your app

Now you can create your own Shiny app. But for now you can only run it in your computer and no one else has access to it. To make available to the public:

1. Save the `ui` and `server` objects/scripts into a stand alone R script, name it `app.R`, and save it in a separate folder. You must name it `app.R` as that's the file that the server will look for when you deploy your app. For this example, I'd save this as `app.R`

```r
ui <- fluidPage(
  sliderInput(inputId='num',
              label='Choose a number',
              value=25, min=1, max=100),
  plotOutput('hist')
)
server <- function(input, output){
    output$hist <- renderPlot({
      title <- paste(input$num, 'random normal values')
      hist(rnorm(input$num), main=title, xlab='values')
  })
}
```

2. Go to shinyapps.io and log in or sign for an account.
3. Now simply run the app on your computer
4. In the top right cover of R viewer window, there is a `Publish` button that you can use to publish your app.
5. Simply follow the instructions.

Chapter 6

Analysis

Example studies conducted by FES Professors.

Chapter 7

Resources

Data

sea (fish vs phys), land (human, animal, plant, phys), air (compos), region climate, weather air quality hydrology water quality earth quake data

Business and the Environment

Climate Change Science and Solutions

Ecosystems and Land Conservation and Management

Energy and the Environment

Environmental Policy Analysis

Forestry

Industrial Ecology and Green Chemistry

People, Equity, and the Environment

Urban

Water Resource Science and Management

Other

Yale Course Map