Exploring data with R

Nov 7, 2018

In collaboration with the WHO VA Reference Group

Overview

Data exploration in R

In this lecture we will discuss practical data analysis in R.
In particular we will try to think about data and data analysis from a programing point of view.
Once you are familiar with R, you will see that it is a powerful tool for

- summarizing data
- navigating all or subset of data
- defining rules to change the data
- visual exploration of the data

We will discuss each of these aspects using examples in this lecture.

Data cleaning in R

We will also explore ways for data cleaning in R

Why doing data cleaning in R is a good idea?

- You can maintain copies of the data both before and after cleaning.
- You can separate the data and the procedure to clean up the data, and keep track of what is changed and how.
- If anything goes wrong, it is easy to undo them later without manually repeating the procedure on a spreadsheet.
- You can repeat your procedures on new datasets easily.

But it also requires

- more coding effort;
- a different perspective of data, compared to manipulating spreadsheets;
- and more coding effort, particularly if you are new to R.

VA data in particular

We will also discuss topics more directly related to VA data:

- How to quickly summarize and explore your data
- Some common tasks in cleaning up your data
- Some common tasks in plausibility checks

And a preview of VA algorithms:

- What input data does each VA algorithm expect
- How VA algorithms clean up the input data

Getting to know your data

Read data

You have seen the `read.csv` function to load CSV file into R.

```
data1 <- read.csv("../data/va16Data7_hw.csv", stringsAsFact
```

It reads the file in the `data` folder one level up from the current working directory.

- `header=TRUE` tells R that the first row is column names.
- `stringsAsFactors= FALSE` tells R to not turn character strings into factors.
- You can also check the default setting of these arguments using `?read.csv`

If you are more familiar with looking at data in a spreadsheet, the first challenge you may encounter in R is probably:

I don't see my data!

See your data

Your data (in the CSV file) is now read into the data frame called 'data1' in your R session.

- You can choose to see the data using different views.
- You can modify the data frame.
- You can write the modified data frame to a new file
- All these procedures 'live' in your R code script,
- They do not live in your data file, or R sessions.

The original data is not changed in the CSV file (unless you overwrite it).

Try and see what does the following commands do:

```
class(data1)
class(data1$i022a)
class(data1$ID)
dim(data1)
summary(data1)
head(data1)
head(data1[, c(1:10)])
```

Summaries

The easiest function to summarize data is the `summary()` function. It can be applied to the whole dataset or some columns/rows.

However, it is not very useful for characters, try `table()` instead.

```
table(data1$i022a)
```

```
##
##  -  0 No  Y
## 49 32  7 12
```

There seems to be some problem? We will deal with it later.

Five-number summaries

Let us read in another dataset,

```
data2 <- read.csv("../data/intro.csv", stringsAsFactors = FALSE)
```

To see the first few rows of the data,

```
head(data2)
```

To see a summary of the data (notice the summary of continuous variables),

```
summary(data2)
```

To see a spreadsheet-like dataset in RStudio

```
View(data2)
```

We can extract the date/time from the first character column by

```
data2$timedate2 = strptime(data2[, 1],
  format = '%m/%d/%y %H:%M', 'GMT')
```

For more information about time formats in R, see `?strptime`

Range checks

Check the range of data is correct

```
range(data2$age)
```

```
## [1] NA NA
```

```
range(data2$age, na.rm = TRUE)
```

```
## [1]  0.42 80.00
```

Find out which rows have NA values

```
which(is.na(data2$age))
data2$name[which(is.na(data2$age))]
data2[which(is.na(data2$age)), "name"]
```

Similarly you can do this to time variables

```
range(data2$timedate2)
```

```
## [1] "2011-01-01 04:00:00 GMT"
## [2] "2012-12-19 03:00:00 GMT"
```

Valid values

The reason we have been using `stringsAsFactors=FALSE` when reading the data is because factors are more difficult to work with in general.

- try read the data without this argument, and do a summary for the 'name' column.
- then try changing the name of the first record to "Peter, Mr. Russo", what do you see?

However, they are useful for representing categorical variables.

```
data2$sex <- factor(data2$sex, levels = c("female",
    "male"))
table(data2$sex, useNA = "always")
```

```
##
## female   male   <NA>
##    314    577      0
```

Valid values: things to be careful about

Remember to check if there are NA's in the data

```
data2_copy <- read.csv("../data/intro.csv",
    stringsAsFactors=FALSE)
data2_copy$sex[1:10] <- NA
table(data2_copy$sex)
table(data2_copy$sex, useNA = "always")
```

Remember to check if data are coded using the same scheme

```
data2_copy <- read.csv("../data/intro.csv")
data2_copy$sex[1] <- "m"
data2_copy$sex[data2_copy$sex == "m"] <- "male"
table(data2_copy$sex)
```

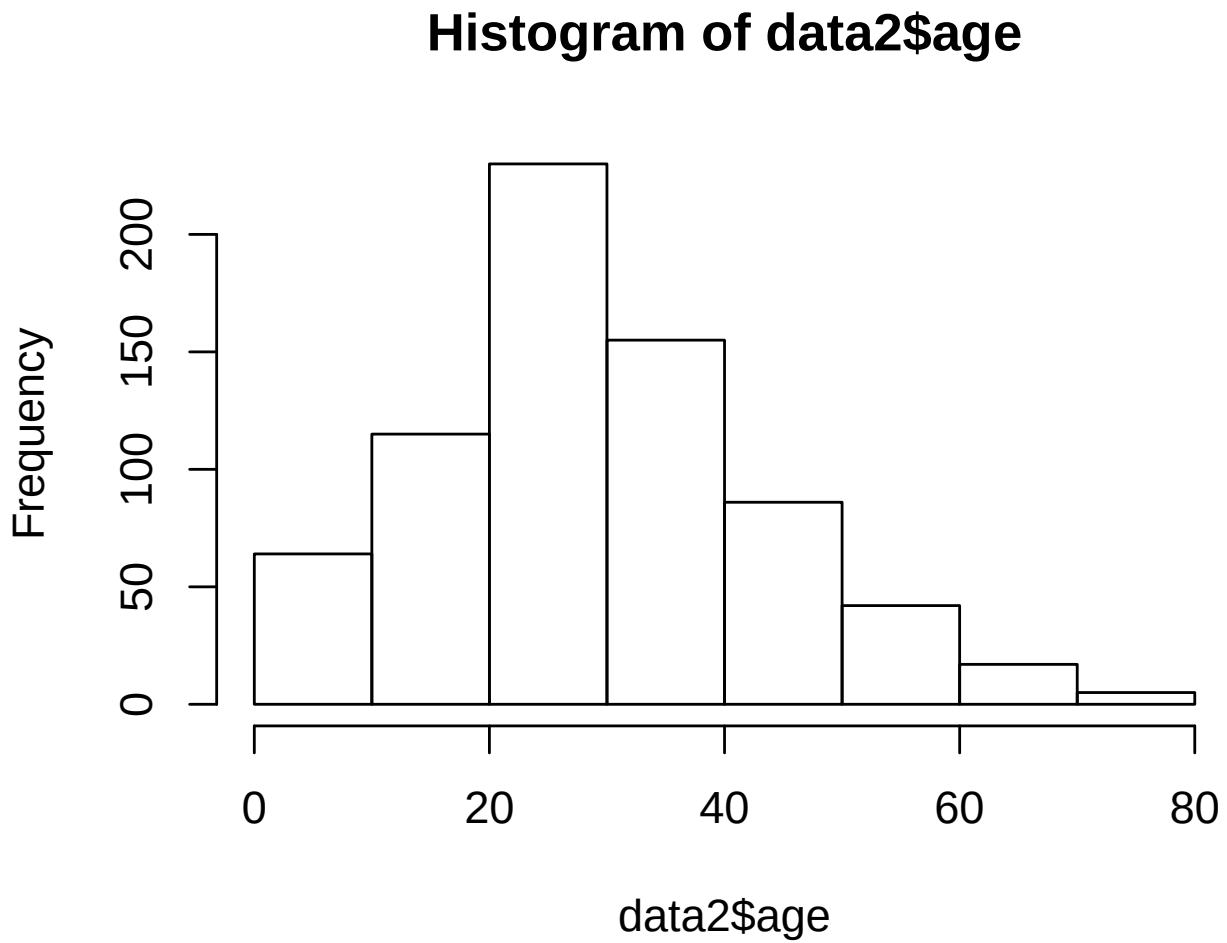
Know how to recode variables that are stored as factors

```
data2_copy <- read.csv("../data/intro.csv")
data2_copy$sex <- as.character(data2_copy$sex)
data2_copy$sex[data2_copy$sex == "male"] <- "m"
data2_copy$sex[data2_copy$sex == "female"] <- "f"
data2_copy$sex <- factor(data2_copy$sex)
table(data2_copy$sex)
```

Histograms

We will learn some more advanced visualizations later, but here we start with some basics

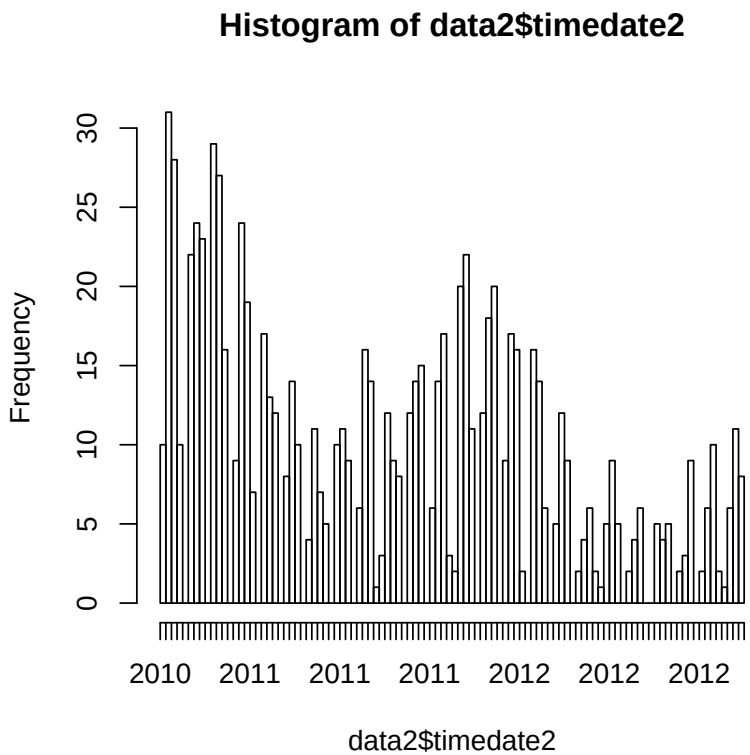
```
hist(data2$age)
```

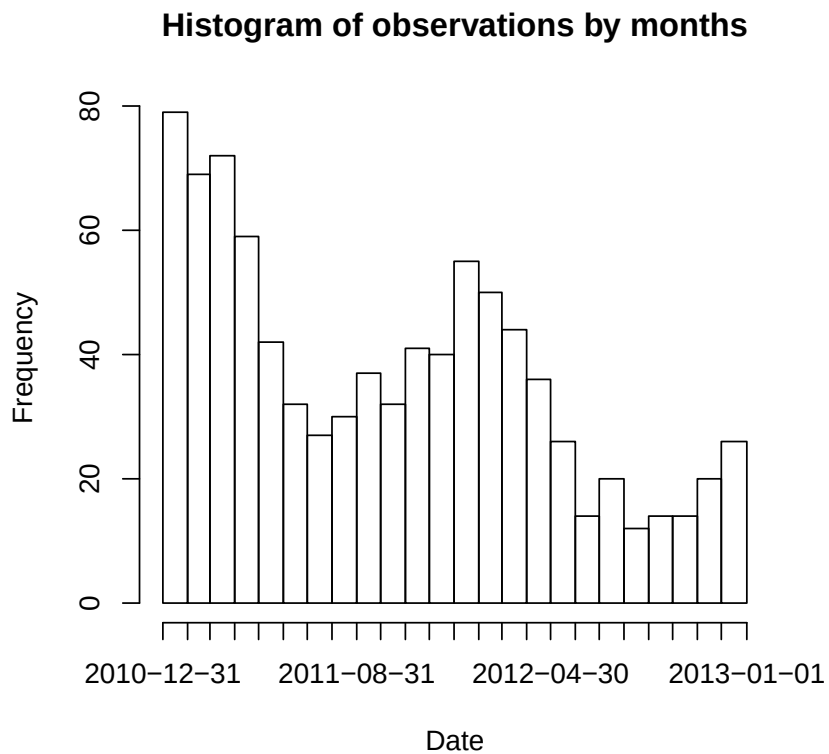


Histogram

We can also plot histogram of time objects, and customize the plots

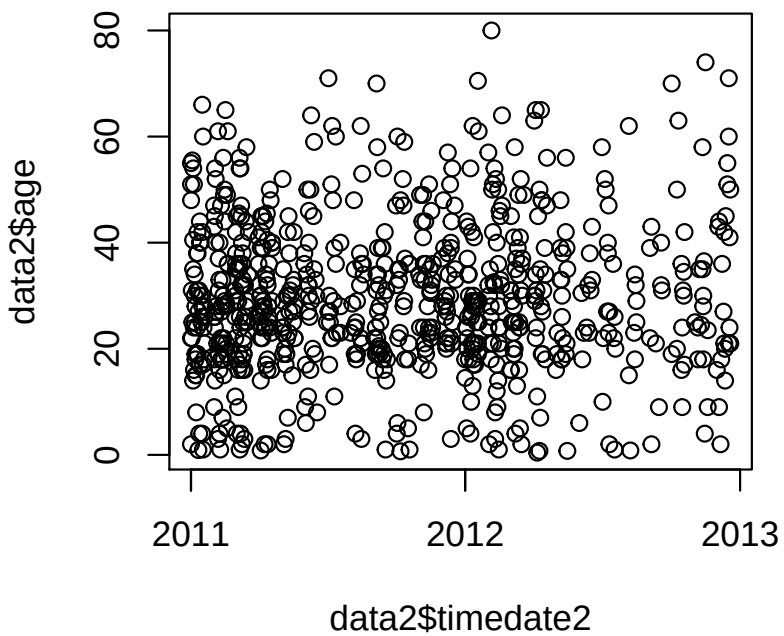
```
par(mfrow = c(1, 2))
hist(data2$timedate2, breaks = "weeks", freq = TRUE)
hist(data2$timedate2, breaks = "months", xlab="Date",
  main = "Histogram of observations by months", freq = TRUE)
```

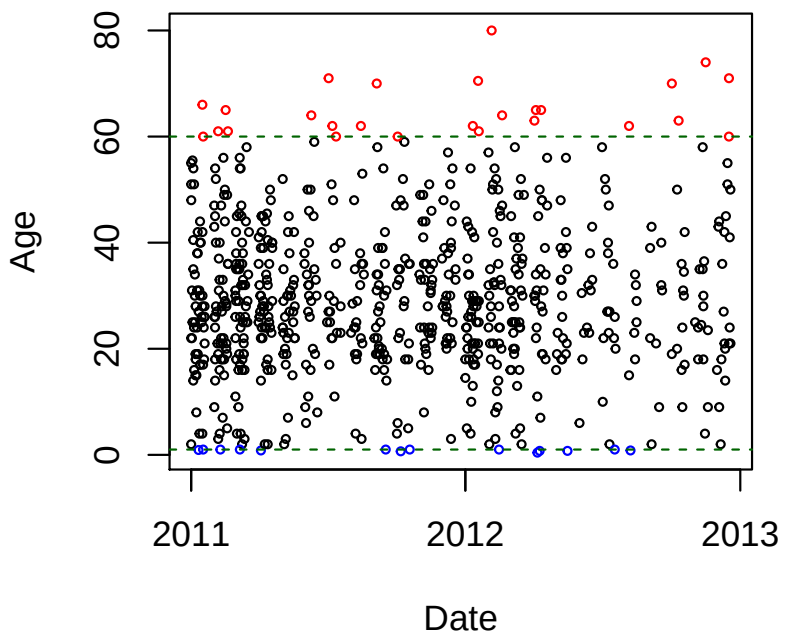




Scatter plots

```
par(mfrow = c(1, 2))
plot(data2$timedate2, data2$age)
color <- rep("black", dim(data2)[1])
color[data2$age >= 60] <- "red"
color[data2$age <= 1] <- "blue"
plot(data2$timedate2, data2$age, col = color, xlab = "Date",
    ylab = "Age", cex = 0.5)
abline(h = c(1, 60), col = "darkgreen", lty = 2)
```





More visualization

What other plots you want to see? We can discuss more in the afternoon session!

The basic R plots are great for producing quick sketch. They can get very complicated if you want to customize the look of your plot a lot.

Later this week we will also see some other visualization tools for more complicated graphs (with a steeper learning curve).

Clean up your data

Standardize codings

Now we revisit the dataset at the beginning of this lecture

```
data1 <- read.csv("../data/va16Data7_hw.csv", stringsAsFactors = FALSE)
table(data1[, 2])
```

```
##
##  -  Y
## 40 60
```

```
table(data1[, 6])
```

```
##
##  -  0 No  Y
## 49 32  7 12
```

So it seems some columns are coded with mixed values. Let us see which codes are used in this dataset

```
unique(as.character(unlist(data1[, -1])))
```

```
## [1] "Y"   "-"   "0"   "No"  "1"   "Yes" "N"
```

Clean up your data: a basic approach

There are a few ways to standardize the coding scheme. We will focus on this basic codes below. It loops over all the columns and change the coding schemes for each column.

```
for (i in 2:dim(data1)[2]) {
    data1[which(data1[, i] == "Yes"), i] <- "Y"
    data1[which(data1[, i] == "No"), i] <- "N"
    data1[which(data1[, i] == "0"), i] <- "N"
    data1[which(data1[, i] == "1"), i] <- "Y"
}
table(as.character(unlist(data1[, -1])), useNA = "always")
```

```
##
##     -     N     Y  <NA>
## 31199  1414  2687     0
```

Clean up your data: a more advanced approach

There are many packages for tidying data in R more efficiently. We will not focus on them, but here is an example

```
library(plyr)
data1 <- read.csv("../data/va16Data7_hw.csv")
for (i in 2:dim(data1)[2]) {
    data1[, i] <- revalue(data1[, i], c(Yes = "Y",
        No = "N", `1` = "Y", `0` = "N"), warn_missing = FALSE)
}
table(as.character(unlist(data1[, -1])), useNA = "always")
```

```
##
##     -     N     Y  <NA>
## 31199  1414  2687     0
```

Notice it automatically handles factors. The basic approach will fail for factors. When input column is factor, the output column is also factor.

if you are interested, google with key word 'tidyverse data cleaning'

Clean up your data: for VA data

There is also a function in openVA package that is specifically designed for VA input data. Here is an example

```
library(openVA)
data1 <- read.csv("../data/va16Data7_hw.csv")
data1 <- ConvertData(data1, yesLabel = c("Yes", "Y",
    "1"), noLabel = c("No", "N", "0"), missLabel = c("-",
    "."))
table(as.character(unlist(data1[, -1])), useNA = "always")
```

```
##
##             .     Y  <NA>
##  1414 31199  2687     0
```

Notice the function also works for factor columns, but change them back into characters.

Clean up your data: for VA data

The default coding scheme is different than what we are using here, but since they have been turned into characters, you can easily change them without looping.

```
data1[data1 == ""] <- "N"
data1[data1 == "."] <- "-"
table(as.character(unlist(data1[, -1])), useNA = "always")
```

```
##
##     -     N     Y  <NA>
## 31199  1414  2687     0
```

We will discuss what coding scheme is used for each algorithm/input on Friday.

Additional considerations for data cleaning

Pay attention to missing values.

Pay attention to values that are close to the standard coding schemes,

- e.g., “Y ”(with an extra space) in place of “Y”.

Pay attention to column names and extra columns.

Pay attention to IDs and make sure they are unique, so that you can map results back to input.

Save data

After performing all the cleaning steps, you may save the clean data into a different spreadsheet with a new name.

```
write.csv(data1, file = "../data/va16Data7_clean.csv",
    row.names = FALSE)
```

Again, remember to also maintain both the original data and the codes you used to clean up the data, so that you can always revisit to make changes.

How algorithms checks your data

Algorithms data consistency checks

We will revisit this in more detail on Thursday and Friday.

Most algorithms have its internal data consistency checks,

- WHO format: InterVA and InSilicoVA
- PHMRC format: SmartVA-Analyze

In a nutshell, they checks for

- symptoms that are inconsistent with each other
- determine which causes are impossible given the symptoms

Example: InterVA consistency check logic

Algorithm 1 Data Check

Require:

$\mathbf{s}_j$ = vector of $j = 1$ to 245 indicators for one death

notask(s): higher-level indicator(s) whose presence ensures indicator s does not exist

anc(s): higher-level indicator(s) that must exist if indicator s exists

Ensure: $\mathbf{s}^*$ = modified vector of $j = 1$ to 245 indicators for one death

1: **for** each indicator $j = 1$ to 245 **do**
2: Set $\mathbf{s}_j^* \leftarrow \mathbf{s}_j$ ▷ Initialize
3: **end for**
4: **for** each indicator $j = 1$ to 245 **do**
5: **if** there exists $\mathbf{s}_\ell^*$ in notask($\mathbf{s}_j^*$) and $\mathbf{s}_\ell^*$ is 'yes' **then**
6: $\mathbf{s}_j^* \leftarrow$ 'no' ▷ This indicator is nonsensical, set to 'No'
7: **end if**
8: **if** there exists $\mathbf{s}_\ell^*$ in anc($\mathbf{s}_j$) and $\mathbf{s}_j^*$ is 'yes' **then**
9: $\mathbf{s}_\ell^* \leftarrow$ 'yes' ▷ Set more general version of this indicator to 'yes'
10: **end if**
11: **end for**
12: Repeat loop in lines 4 to line 11 again ▷ Processes 2-level hierarchies

InSilicoVA inherits similar check procedures with some modifications

Self-check: Female/male symptoms appearing for wrong sex?

For key symptoms of interest, you can check the data yourself as well.

For example, for symptoms i019a (male), i019b (female), and i022l (female 12-19 years old)

```
table(data1$i019a, data1$i019b)
```

```
##
##       -  Y
##    -  0 44
##    Y 56  0
```

```
table(data1$i019a, data1$i022l)
```

```
##
##       -  Y
##    - 43  1
##    Y 56  0
```

Self-check: Age-dependent symptoms appearing for wrong ages?

Similarly, for symptoms i022g (neonate), i127o (diagnosis by health professional of HIV/AIDS)

```
table(data1$i022g, data1$i127o)
```

```
##
##       -
##    - 69
##    Y 31
```

More on checking date/time

Interview times and durations

For this last topic, let us look at the following dataset

```
data3 <- read.csv("../data/interviewduration.csv")
dim(data3)
```

```
## [1] 6202    4
```

```
head(data3)
```

```
##   SubmissionDate interviewstarttime interviewdate
## 1 1/1/2018 14:36     1/1/2018 12:53      1-Jan-18
## 2 1/1/2018 15:41     1/1/2018 13:57      1-Jan-18
## 3 1/1/2018 21:36     1/1/2018 15:57      1-Jan-18
## 4 2/1/2018 19:40     2/1/2018 19:10      2-Jan-18
## 5 2/1/2018 20:55     2/1/2018 20:33      2-Jan-18
## 6 3/1/2018 15:06     3/1/2018 15:02      3-Jan-18
##   interviewendtime
## 1   1/1/2018 14:36
## 2   1/1/2018 15:40
## 3   1/1/2018 21:35
## 4   2/1/2018 19:40
## 5   2/1/2018 20:53
## 6   3/1/2018 15:08
```

Organize interview times and durations

The coding for time is a little messy in this dataset

```
data3$SubmissionDate <- strptime(data3$SubmissionDate,
    format = "%d/%m/%Y %H:%M", "GMT")
data3$interviewstarttime <- strptime(data3$interviewstarttime,
    format = "%d/%m/%Y %H:%M", "GMT")
data3$interviewdate <- strptime(data3$interviewdate,
    format = "%d-%b-%y", "GMT")
data3$interviewendtime <- strptime(data3$interviewendtime,
    format = "%d/%m/%Y %H:%M", "GMT")
```

+ Calculate other quantities of interest

```
data3$month <- format(data3$interviewstarttime, format = "%m")
data3$year <- format(data3$interviewstarttime, format = "%Y")
data3$weekday <- format(data3$interviewstarttime, format = "%A")
data3$duration <- difftime(data3$interviewendtime,
    data3$interviewstarttime, units = "mins")
data3$duration.minute <- as.numeric(data3$duration)
```

Before we move on to visualization,

It seems the reported start/end time are also not always right

```
data3[which.max(data3$duration.minute), ]
```

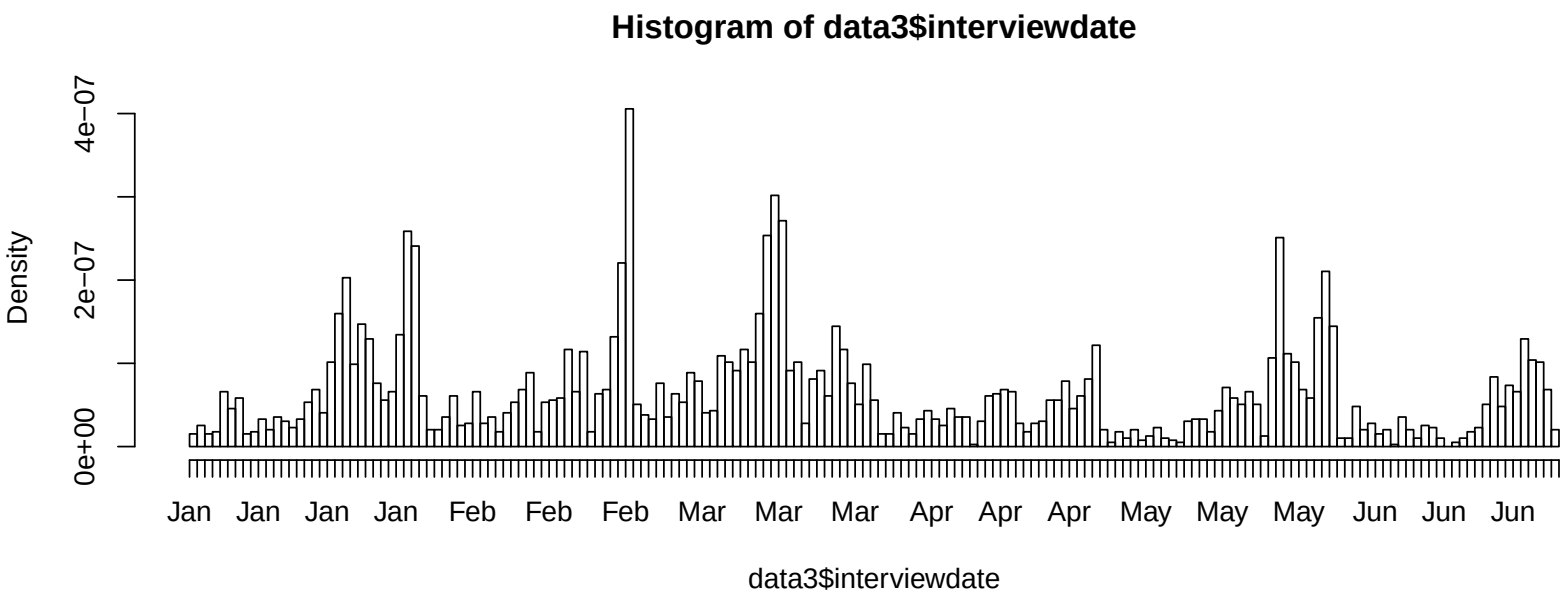
```
##           SubmissionDate  interviewstarttime
## 5347 2018-05-29 13:15:00 2018-01-01 21:09:00
##      interviewdate    interviewendtime month year
## 5347    2018-01-01 2018-05-29 13:13:00    01 2018
##      weekday    duration duration.minute
## 5347  Monday 212644 mins          212644
```

+ In practice, you may want to figure out what caused the problem. For now, let us simply remove those data points and check the interviews shorter than 12 hours.

```
data3 <- data3[-which(data3$duration.minute > 12 *
    60), ]
```

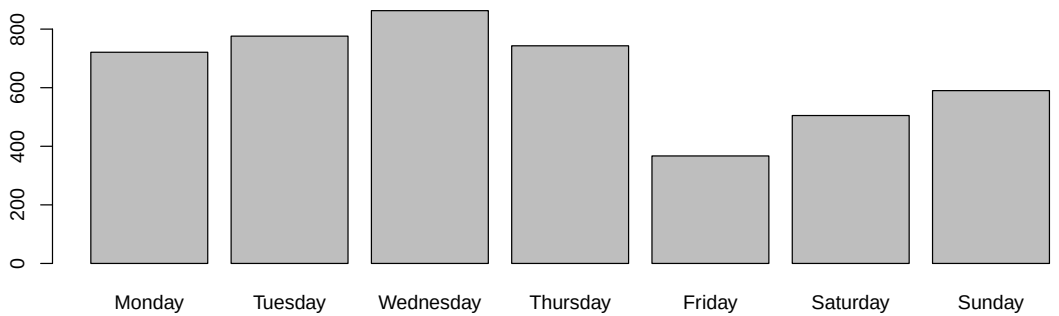
Visualize the distribution of interview date

```
hist(data3$interviewdate, breaks = "day")
```



Visualize the distribution of interview date

```
data3$weekday <- factor(data3$weekday, levels = c("Monday",
    "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
    "Sunday"))
barplot(table(data3$weekday))
```

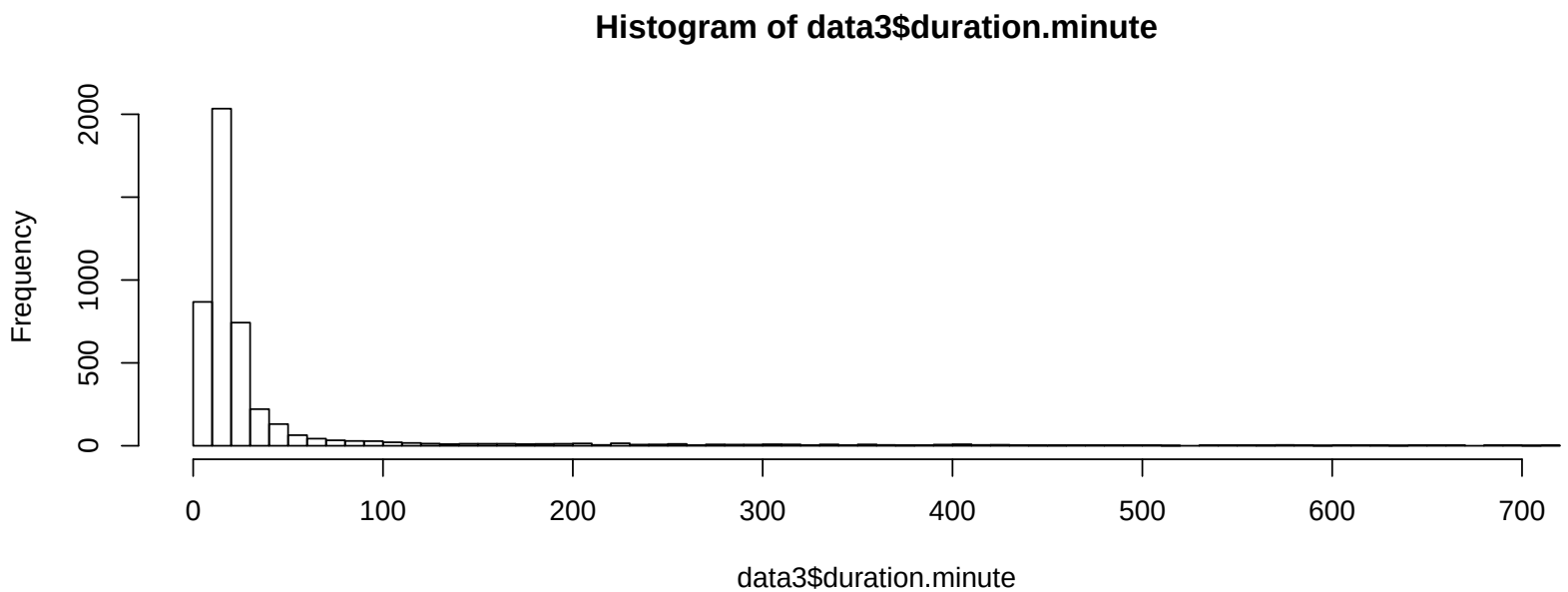


Visualize the distribution of interview duration

```
summary(data3$duration.minute)
```

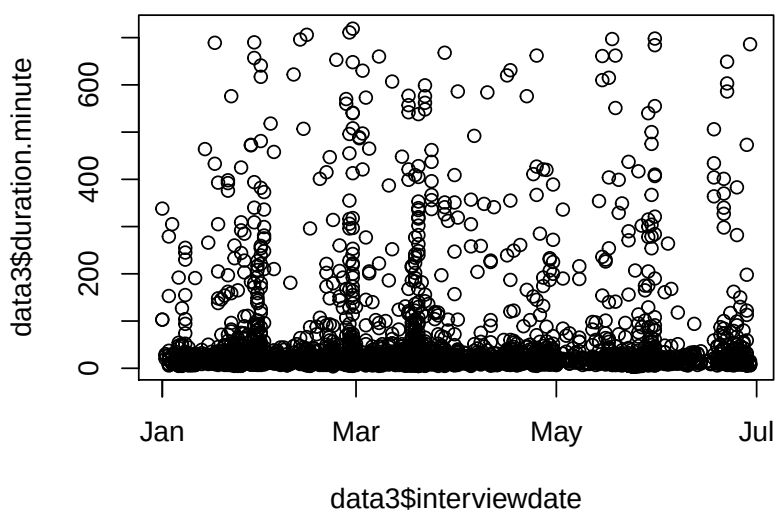
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4.00   12.00   17.00   42.73   26.00  719.00
```

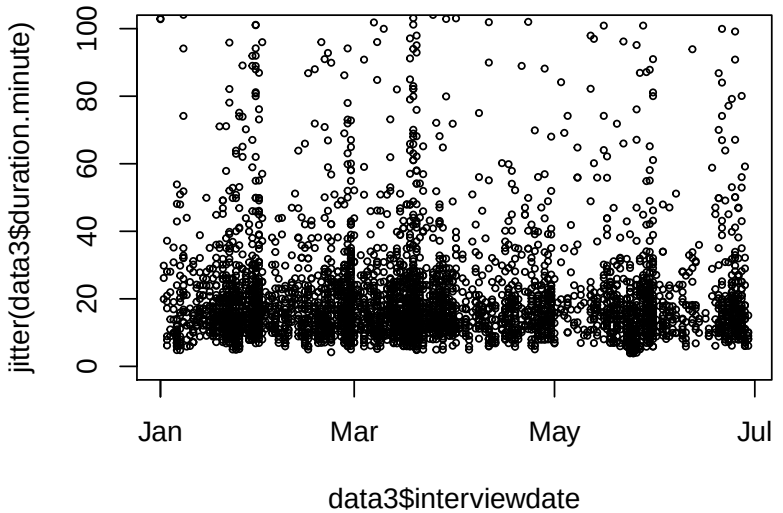
```
hist(data3$duration.minute, breaks = 100)
```



Visualize the distribution of interview duration by date

```
par(mfrow = c(1, 2))
plot(data3$interviewdate, data3$duration.minute)
plot(data3$interviewdate, jitter(data3$duration.minute),
    ylim = c(0, 100), cex = 0.5)
```





Summary of duration by weekday

```
aggregate(duration.minute ~ weekday, data = data3,
    FUN = median)
```

```
##       weekday duration.minute
## 1      Monday              17
## 2     Tuesday              17
## 3   Wednesday              17
## 4    Thursday              16
## 5      Friday              15
## 6    Saturday              17
## 7      Sunday              17
```

Summary of duration by month and weekday

```
aggregate(duration.minute ~ month + weekday, data = data3,
    FUN = median)
```

```
##    month   weekday duration.minute
## 1     01    Monday            18.0
## 2     02    Monday            16.0
## 3     03    Monday            18.0
## 4     04    Monday            17.0
## 5     05    Monday            19.0
## 6     06    Monday            18.0
## 7     01   Tuesday            19.0
## 8     02   Tuesday            15.0
## 9     03   Tuesday            16.0
## 10    04   Tuesday            18.0
## 11    05   Tuesday            18.0
## 12    06   Tuesday            16.0
## 13    01 Wednesday            17.0
## 14    02 Wednesday            18.0
## 15    03 Wednesday            16.0
## 16    04 Wednesday            17.0
## 17    05 Wednesday            16.0
## 18    06 Wednesday            15.0
```

Exercise

Data cleaning and exploration

1. Read in the datasets *va16Data8_hw.csv*, *va16Data9_hw.csv*, and *va16Data10_hw.csv*.
2. Explore the data. Are there any issues in the data?
3. Write a paragraph describing if there are any issues in the data and how you propose to clean them up.
4. Review the standard format of WHO 2016 input data for InterVA5. What do you think of the quality of these three datasets? Are they plausible? Write down what you think and provide (quantitatively or visually) evidence to support your statement.