R403: Probabilistic and Statistical Computations with R

Topic 7: Aggregating and Summarizing Data

Kaloyan Ganev

2022/2023

Lecture Contents

Introduction

Introductory Notes

- R has a lot of built-in functions that could be useful in aggregating and summarizing data
- We will make a brief tour through some of the most commonly used ones
- A part of the discussion will naturally contain some review of concepts already introduced

Summary Commands and Descriptive Statistics

Summary Commands and Descriptive Statistics

- Provide information which might suggest which analytical procedure would be most appropriate to apply to data
- With their aid, better understanding of the properties of data is achieved
- Also, they might point to data issues requiring additional pre-processing, cleaning, etc.

Summary Commands

- A good idea to start is to use the `ls()` command to get the list of all named objects in memory
- Of course, you can view the objects also in the Environment browser of RStudio
- Let's use some data so that we exemplify with it
- Load the data on new residential buildings (*residential.csv*)

```
residential <- read.csv2("residential.csv", header = TRUE, skip =
     4, blank.lines.skip = TRUE)
```

- We can have a look at the data by printing the data frame in the console or displaying it using RStudio's point-and-click tools

Summary Commands (2)

- If the data set is large, though, the data would be difficult to grasp
- Using the `str()` command, you can get a concise summary of the characteristics of the data:

```
str(residential)
```

- Of course, the same result is obtainable by clicking on the blue arrow next to the name of the data frame in the environment browser
- Note that `str()` is specifically suited to explore the structure of the object of interest

Summary Commands (3)

- In order to get a statistical summary, just type:

```
summary(residential)
```

- There might be nuances of this command's output when different object types are involved (character vectors, factors, matrices, models, etc.)
- Some more summary commands for lists and tables:

```
names() # Returns the names of list elements/data frame columns
```

- ...and some such for vectors, matrices, arrays, and data frames:

```
names()
colnames()
rownames()
dimnames()
```

Summary Statistics for Vectors

- Maximum value:

```
max(x, na.rm = FALSE)
```

- Usually missing observations should be removed from the calculation, therefore `na.rm = TRUE` is used
- Otherwise, you will get an *NA* value for a maximum
- Minimum value:

```
min(x, na.rm = FALSE) # Same logic as above
```

Summary Statistics for Vectors (2)

- Length of vector:

```
length(x)
```

- Note that *NA*'s are also values so they still count against the length of a vector (to omit *NA*'s, use `length(na.omit(x))`)
- Sum of all vector elements:

```
sum(x, na.rm = FALSE)
```

- Arithmetic mean, median, standard deviation, and variance:

```
mean(x, na.rm = FALSE)
median( x, na.rm = FALSE)
sd(x, na.rm = FALSE)
var(x, na.rm = FALSE)
```

Summary Statistics for Vectors (3)

- Median absolute deviation (a robust[1] measure of dispersion, sometimes better than the standard deviation):

$$\mathrm{MAD}(X) = \mathrm{median}(|X_i - \mathrm{median}(X)|)$$

```
mad(x, na.rm = FALSE)
```

[1] Meaning resilient to outliers in the present case.

Summary Statistics for Vectors (4)

- Quantiles:

```
quantile(x) # The default 0, 25, 50, 75, and 100% quantiles
```

- If you need to calculate different quantiles (e.g. 33, 66, and 99%):

```
quantile(x, probs = c(0.33, 0.66, 0.99), na.rm = TRUE, names =
    TRUE)
```

- Cumulative sum, product, maximum, and minimum:

```
cumsum(x)
cumprod(x)
cummax(x)
cummin(x)
```

Summary Statistics for Data Frames

- We've already discussed `summary()`
- The following are also clear: `min(), max(), length(), sum()`
- It also happened that we used `rowSums()` and `colSums()`; `colMeans()` and `rowMeans()` are analogical

Summary Statistics for Matrices

- To apply a statistical function to a part of a matrix, e.g. a column or a row:

```
m1 <- matrix(1:24, nrow = 4)
mean(m1[3,])
mean(m1[,2])
```

- `colMeans()` and `rowMeans()` can do this job, too
- The commands `rowSums()` and `colSums()` also work for matrices

Summary Statistics for Lists

- Not straightforward to have such (why?)
- $ notation can be used but it may be somewhat inconvenient
- Still, there some ways to do that, but we will leave matters for the topic on loops and loop commands

Tabulation

Summary Tables

- In order to summarize data samples, the `table()` command can be used
- It also allows creating, altering, and manipulating table objects including two types of special tables:
  - Contingency tables
  - Complex (flat) contingency tables

Table Summaries for Vectors

- Let's simulate tossing a coint 1000000 times

```
coin <- sample(c("H", "T"), 1000000)
```

- We can see a summary of counts using `table()` again:

```
table2 <- table(coin)
```

Contingency Tables

- The term was coined by Karl Pearson in 1904
- Also called cross-tabs/two-way tables
- Practically a table of counts; used to display the multivariate frequency distribution of the selected categorical variables
- In a sense, it is the analogue to scatter plots for continuous variables
- By this, they provide intuition on the potential statistical relationships among variables
- A complex (flat) table is a contingency table that is used to compress multiple dimensions of data to two dimensions only and a single table

An Example Simple Contingency Table

Gender / Hair colour	Black hair	Brown hair	Blond hair	Total
Male	70	25	40	135
Female	30	20	25	75
Total	100	45	65	210

- In this table, there are two variables: gender and hair colour
- The former determines the row categories, and the latter determines the column categories
- Each combination of row and column is called a **cell**

Requirements for Contingency Tables

- The requirements stem from the specifics of statistical methods applied to such tables
- First, observations should be **independent** from each other
- Second, categories should be **exclusive**, i.e. it is possible for an observation to fall only in one of the categories; in other words, categories cannot overlap
- Third, categories should be **exhaustive**, i.e. the full range of possibilities is represented making it impossible for an observation to fail falling in a category

How to Create a Contingency Table in R

- Get the values of the two variables in two vectors (here we make a new random sample so the numbers in above table will not be repeated):

```
gender <- sample(c("Male", "Female"), 210, replace = TRUE)
hair_col <- sample(c("Black", "Brown", "Blond"), 210, replace =
    TRUE)
```

- Make a data frame and then create the contingency table from it:

```
data1 <- as.data.frame(cbind(gender, hair_col))
table1 <- table(data1$gender, data1$hair_col)
```

- In general, this can be done for any data frame containing categorical variables

Flat Contingency Tables

- Let's take another example
- Assume you gather a random sample of data on educational degree, salary and productivity of several workers (20 in this example)
- We generate the data at random with some prior restriction with the following code:

```
degree <- sample(c("BSc", "MSc", "PhD"), 20, replace = TRUE)
salary <- sample(c("high", "medium size", "low"), 20, replace =
    TRUE)
productivity <- sample(seq(120,160, by = 0.5), 20, replace = TRUE
    )
```

Flat Contingency Tables (2)

- Let's get those data into a data frame:

```
df1 <- as.data.frame(cbind(degree, productivity, salary))
```

- There are some issues of the data type of productivity which we correct in the following way:

```
df1$productivity <- as.numeric(as.character(df1$productivity))
```

- Now, let's use the `table()` command to produce a summary:

```
table(df1)
```

- Because of the several dimensions, several tables are produced

Flat Contingency Tables (3)

- In order to get just one table instead of the many, we can create a flat contingency table that squeezes everything in two dimensions
- Flat contingency tables are created using the `ftable()` command
- For example:

```
ftable(df1)
```

- You can now take a close look at the table that is produced and make make a comparison with the previous `table()` output

Proportions Tables

- Can be generated on data frames, matrices or tables
- Note that data frames should contain only numerical variables in order not to get an error
- To begin with, let's use *table1* which is already available
- Type and run:

```
prop.table(table1)
```

- The command outputs proportions (frequencies) as shares of the grand total
- You can also get proportions as shares of row totals or column totals, respectively by running:

```
prop.table(table1, margin = 1)
prop.table(table1, margin = 2)
```

Proportions Tables (2)

- See what happens when the same is applied to a numerics-only data frame
- Generate the following salary vector:

```
salary2 <- sample(seq(2500,3500, by = 0.01), 20, replace = TRUE)
```

- Make the data frame:

```
df2 <- as.data.frame(cbind(productivity, salary2))
```

- Check out:

```
prop.table(as.matrix(df2))
```

- Try with the two other options

Further Readings

- Gardener, M. (2012): *Beginning R: The Statistical Programming Language*, Wiley (ch. 4)
- Spector's book (see syllabus), p. 101-107
- R documentation and help