Day 2: Data wrangling with data.table

Qingyin Cai

Department of Applied Economics
University of Minnesota

WebR Status

🟢 Ready!

🎯 Learning Objectives

- Use basic data wrangling skills with the `data.table` package
- Learn how to use the `%>%` operator from the `magrittr` package (Optional)

✳ Reference

- Introduction to data.table
- Efficient reshaping using data.table
- R for Data Science, Ch18: Pipes

Today's outline:

1. Data manipulation with data.table
    - What is `data.table`?
    - data.table syntax
    - Subset rows
    - Select columns
    - Compute on columns
    - Create a new column
    - Perform aggregations by group
    - Reshape datasets
    - Merge datasets
2. `%>%` operator (optional)
3. After-class Exercise Problems
4. Appendix

Introduction to data.table

What is `data.table`?

What is it?

- `data.table` is a package in R that provides an enhanced version of `data.frame`.
  - It is designed to be fast and memory efficient.

- There is another package called `dplyr` that is also popular for data wrangling. But `data.table` is much faster than `dplyr` particularly for large-scale data manipulation tasks.
  - See this for the speed comparison of dplyr and data.table.
  - This website compares dplyr vs data.table side by side. If you already know `dplyr` syntax, this website would be helpful to understand `data.table` syntax.

What is `data.table`?

Before Starting

1. Let's use `flights` data, which is obtained from `nycflights13`.

Run Code

```
# Load flights data from nycflights13 package.
flights <- nycflights13::flights
# Remove rows with missing values (just for convenience)
flights <- na.omit(flights)
# Check the class of object
class(flights)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

2. Converting to `data.table`

- To use the special features of the `data.table` package, the data must be in the `data.table` class.
- You can convert a `data.frame` (or tibble) into a `data.table` by using the `setDT()` function.

Run Code

```
# Load data.table package
library(data.table)
setDT(flights) # same as, flights <- as.data.table(flights)
# Now, flights is a data.table object.
class(flights)
```

data.table syntax

The general form of `data.table` syntax is

Run Code

```
# Don't run
DT[i, j, by]
```

- `i`: choose rows (filtering or subsetting)
- `j`: choose or transform columns (summaries, calculations, or selecting variables)
- `by`: group by variables (do the calculation in `j` separately for each group)

Simply put,

Start with a data.table `DT`. First pick rows using `i`, then work on columns with `j`, and if needed, repeat that operation for each group defined by `by`.

Using data.table syntax, we will see how to:

- subset rows
- select columns, compute on the selected columns, create a new column
- perform aggregations by group

1. Subset Rows

Basics

- `data.table` syntax: `DT[i, j, by]`
- To subset rows, put a condition on a **column** inside `i`.
  - Example: `DT[colA == "value", ]` selects rows where column `colA` equals `"value"`.

Example

Subset rows where `carrier` is `"AA"` (American Airlines):

Run Code

```
1  flights[carrier == "AA",] # a comma after the condition is not required
```

- What happens here?
  - `i`: selects rows where `carrier == "AA"`
  - `j`: no action (all columns)
  - `by`: no action (no grouping)

1. Subset Rows

In-class Exercise

Questions

Run Code

```
# 1. Subset rows where `carrier` is "AA" and `month` is 1 (January)

# 2. Subset rows where `carrier` is "AA" and `origin` is all the airports except "JFK"

# 3. Subset rows where delay in departure (`dep_delay`) is less than 0 or delay in arrival (`arr_delay`)
is less than 0. (Hint: use | for "or" condition)
```

Other row-related tasks

The key idea: all tasks related to **rows** are done inside `i`.

Example

Filter rows

Select flights where `carrier` is `"AA"`:

Run Code

```
1  flights[carrier == "AA"]
```

Other row-related tasks

The key idea: all tasks related to **rows** are done inside `i`.

Example

Select by row number

Return the first 5 rows:

Run Code

```
1  flights[1:5]
```

Other row-related tasks

The key idea: all tasks related to **rows** are done inside `i`.

Example

Remove rows

Exclude rows 1 to 10:

Run Code

```
flights[-(1:10)]
```

Other row-related tasks

The key idea: all tasks related to **rows** are done inside `i`.

Example

Sort rows

Sort by `month` (ascending) and then `day` (descending):

Run Code

```
1  flights[order(month, -day)]
```

2. Select Columns

Basics

- `data.table` syntax: `DT[i, j, by]`
- To select columns, use the `j` argument

Example:

Suppose we want to select `dep_time` column. Since we are not subsetting rows, we leave the `i` argument blank.

Run Code

```
# --- Select dep_time column as vector --- #
flights[, dep_time]

# --- Select dep_time column as data.table --- #
flights[, list(dep_time)]  # or flights[, .(dep_time)],
```

- If we wrap variables (column names) in `list()`, the result will be returned as a `data.table`.
- `.()` is simply shorthand for `list()` in `data.table` syntax.
- **Important**: In `data.table`, each column is internally stored as a list. When you use `.()` (or `list()`) in the `j` expression, each element of that list becomes a column in the resulting `data.table`.

2. Select Columns

Multiple columns

You can select multiple columns just like you did to select a single column.

Run Code

```
# --- Select dep_time and arr_time as data.table --- #
flights[, .(dep_time, arr_time)]

# --- Unselect columns using - or ! --- #
flights[, !c("dep_time", "arr_time")]
# or
# flights[, -c("dep_time", "arr_time")]
```

2. Select Columns

In-class exercise

Questions

Run Code

```
1  # 1. Select year, month, day, and carrier columns as data.table.
```

3. Compute on Columns

Basics

- `data.table` syntax: `DT[i, j, by]`
- `j` not only allows you to select columns but also to compute on columns

Example

Let's count the number of trips which have had total delay < 0 (i.e., total day = `dep_delay` + `arr_delay`).

Run Code

```
# count the number of trips with total delay < 0
flights[, sum((arr_delay + dep_delay) < 0)]  #Let's explore what's going on here.
```

What happens in this code?

- `i`: no action (all rows are used)
- `j`: takes the sum of the logical vector `arr_delay + dep_delay < 0`
- `by`: no action (no grouping)

Note: Since we skip the `i` expression, we must include a comma before the `j` expression.

3. Compute on Columns of the Subsetted Rows

Basics

- `data.table` syntax: `DT[i, j, by]`
- Using `i` and `j` expressions together, you can perform calculations on the selected columns of the subsetted rows.

Example

How many flights departed from “JFK” airport in the month of June?

Run Code

```
1  flights[origin == "JFK" & month == 6L, .N]
2  # NOTE: `.N` is a special variable that holds the number of rows in the current group.
```

What happens in this code?

- `i` : to select rows where origin airport equals “JFK”, and month equals 6.
- `j` : to count the number of rows in the subsetted data.
- `by` : no action (no grouping)

3. Compute on Columns of the Subsetted Rows

Multiple outputs

- You can assign names to the values you calculate in `j`.
- Recall that `.()` is a shorthand for `list()` in data.table syntax. You can name each element inside `.()` just like naming elements in a regular list.

Example

Count how many flights departed from JFK airport in June. For those flights, calculate the average departure delay (dep_delay).

Run Code

```
1  flights[origin == "JFK" & month == 6L, .(Count = .N, avg_dep_delay = mean(dep_delay))]
```

3. Compute on Columns of the Subsetted Rows

In-class exercise

Questions

1. Find the average arrival delay and the average departure delay for flights that departed from JFK in August.

- Hint:
  - Use the columns: `origin`, `month`, `arr_delay`, `dep_delay`
  - Use the `mean()` function to calculate averages

Run Code

```
1  # You can write your code here
```

4. Create a New Column

Basics

- `data.table` syntax: `DT[i, j, by]`
- In `j` expression, you can **add** or **update** a column in the data table using the `:=` operator.
  - Think of `:=` as a special assignment operator inside `data.table`. It modifies the data table **by reference** (changes the original table without making a copy).

Syntax

```
# === Add one column === #
DT[, "new_column_name" := .(valueA)]

# or you can drop the quotes and `.()` for convenience
DT[, new_column_name := valueA]
```

4. Create a New Column

Example

Using the dataset below, create a new column c that is the sum of columns a and b.

Run Code

```
# === Simple dataset === #
simple_data <- data.table(a = 1:5, b = 6:10)

# === Create a new column === #
simple_data[, "c" := .(a + b)]

# or simply, you can do
simple_data[, c := a + b]
```

Important Rule

The operator `:=` creates new columns by updating the data **in place** (by reference). This means the original data table is directly modified.

4. Create a New Column

Multiple new columns

Here is how you define multiple variables at the same time.

Run Code

```
# === Newly add two columns (formal syntax)  === #
simple_data[, c("c", "d") := .(a + b, a - b)]

# Simplified version
simple_data[, `:=`(
  c = a + b,
  d = a - b
)]
```

4. Create a New Column

Note

- The `:=` operator in `data.table` does not allow you to reference newly created or modified columns within the same `[` expression.
- If you want to use a new column in another calculation, you need a second `[` step.

Example

- Let's create two new columns: (1) `c` by adding `a` and `b`, and (2) `d` by dividing `c` by `a`.

Run Code

```
# This code does not work
flights[, `:=`(
  c = a + b,
  d = c/a,
]

# Instead do this
flights[, c := a + b]
flights[, d := c/a]
```

4. Create a New Column

Update with a condition

- Using `i` and `j` expressions together, you can change the column values for rows that satisfy certain conditions.

Example:

Run Code

```
# === Create a simple data === #
simple_data <- data.table(a = 1:5, b = 6:10)

# === Update column b by adding 10 only for the rows with a >= 3  === #
simple_data[a >= 3, b := b + 10]
```

Keeping the original data:

- If you want to keep the original dataset unchanged, use the `data.table::copy()` function to create a duplicate.
- The object created with `copy()` is completely independent: changes to one will not affect the other.

4. Create a New Column

In-class exercise

Questions

Create two new columns in the `flights` data:

- `total_delay`: the sum of `dep_delay` and `arr_delay`.
- `speed`: the ratio of `distance` to `air_time` (i.e, `distance/air_time`.)

Run Code

```
1  # You can write your code here
```

5. Perform Aggregations by Group (Grouped Operations)

Basics

- `data.table` syntax: `DT[i, j, by]`
- To perform grouped operations, use `by` argument.

Syntax

```
1 DT[, .(new_column = function(column)), by = .(group_variable)]
```

Example: Let's find the number of flights by `origin`.

Run Code

```
1 flights[, .(.N), by = .(origin)]
```

What happens in this code?

- `i`: no action (all rows)
- `j`: count the number of rows in each group defined by `by` argument
- `by`: group the data by `origin`

5. Perform Aggregations by Group (Grouped Operations)

Group by multiple columns

Nothing special. Just provide multiple columns to `by` argument.

Example: Find the average time of departure delay and arrival delay by `carrier` and `origin`.

Run Code

```
1  flights[, .(avg_dep_delay = mean(dep_delay), avg_arr_delay = mean(arr_delay)), by = .(carrier, origin)]
```

5. Perform Aggregations by Group (Grouped Operations)

Grouped operations for select observations

By combining the `i` argument with by, you can perform grouped operations on a subset of rows.

Example 1: Get the number of flights for each origin airport for carrier code “AA” (American Airlines).

Run Code

```
1  flights[carrier == "AA", .N, by = .(origin)]
```

What happens in this code? - `i`: subset rows where `carrier` is “AA” - `j`: count the number of rows in each group defined by `by` argument - `by`: group the data by `origin`

Example 2: Find the number of flights by `origin` and `month` for carrier code “AA” (American Airlines).

Run Code

```
1  head(flights[carrier == "AA", .N, by = .(origin, month)])
```

5. Perform Aggregations by Group (Grouped Operations)

In-class exercise

Questions

1. For each `month` and each `carrier`, calculate the total number of flights, average departure delay, and average arrival delay.

Run Code

```
1  # You can write your code here
```

2. (Optional) Define seasons (Winter: Dec-Feb, Spring: Mar-May, Summer: Jun-Aug, Fall: Sep-Nov) and summarize the total number of flights, average departure delay, and average arrival delay for each season and each carrier.

Run Code

```
1  # You can write your code here
```

Summary

So far, we have covered the **basic operations** in the `data.table` package.

Focus on these key ideas:

- The general syntax is `DT[i, j, by]`:
  - `i` → rows
  - `j` → columns
  - `by` → groups
- Use `i` for anything related to **rows**.
  - Example: filter rows with conditions.
- Use `j` for anything related to **columns**.
  - Example: select columns, compute new values (use `.()`), or add/update columns with `:=`.
- Use `by` for anything related to **grouped operations**.
  - Example: calculate summaries by group.

With just these three pieces (`i`, `j`, and `by`), you can handle most data manipulation tasks in `data.table`.

Next, we will see a few advanced topics:

- Reshaping Data
- Merging Multiple Datasets
- (and the `%>%` operator if we have time).

6. Reshape Data

Basics

Data often comes in two formats: **long** or **wide**.

Example:

Long data:

Each student appears in multiple rows (one per year).

```
   student  year  math reading
    <char> <num> <num>   <num>
1:   Alice  2021    78      82
2:   Alice  2022    85      88
3:     Bob  2021    92      90
4:     Bob  2022    95      93
5: Charlie  2021    88      85
6: Charlie  2022    90      87
7:   Diana  2021    70      75
8:   Diana  2022    80      83
```

Wide data

Each student appears in one row, with columns for each year's scores.

```
   student math_2021 math_2022 reading_2021 reading_2022
    <char>     <num>     <num>        <num>        <num>
1:   Alice        78        85           82           88
2:     Bob        92        95           90           93
3: Charlie        88        90           85           87
4:   Diana        70        80           75           83
```

- We can convert one format to another using `dcast()` and `melt()` functions of `data.table` package.

6. Reshape Data

Long to wide

- Use `dcast()` function converts long form to wide form

Basic Syntax:

```
1 dcast(data, LHS ~ RHS , value.var = c("var1", "var2"))
```

- LHS: set of id variables (variables (columns) that you don't want change).
- RHS: set of variables to be used as the column index.
- `value.var`: set of variables whose values will be filled to cast.

Example:

Suppose that we want to organize the data so that each student's math and reading scores appear in the same row.

Run Code

```
1 student_wide <- dcast(student_long, student ~ year, value.var = c("math", "reading"))
2 student_wide
```

```
Error: object 'student_long' not found
```

Tips - Before coding a reshape, first visualize the format you want the data to take. - I often sketch a small example table. - This helps me to understand what variables I need to use as LHS, RHS, and `value.var`.

6. Reshape Data

Wide to long

- Use `melt()` function to convert wide form to long form

Basic Syntax:

```
melt(data, id.var = c("id_var1", "id_var2"), measure.vars = c("var1", "var2"))
```

- `id.vars`: the set of id variables (variables (columns) that you don't want change).
- `measure.vars`: the set of columns you want to collapse (or combine) together.
- `value.name`: (optional) the name of the new column that will store the values of the variables in `measure.vars`, the default is `value`

Example:

Let's get back to the original data format `student_long` from `student_wide`.

Run Code

```
# Collect math and reading scores for each year into long format
col_math <- paste0("math_", 2021:2022)
col_read <- paste0("reading_", 2021:2022)

student_long2 <- melt(
  student_wide,
  id.vars = "student",
  measure.vars = list(col_math, col_read),
  value.name = c("math","reading")
```

```
)

student_long2

# If you are familiar with regular expressions, you can do:
# melt(student_wide, id.vars = "student", measure.vars = patterns("^math", "^reading"), value.name = c
("math", "reading"))
```

- Notice that the year info is stored as variable (1, 2).

Run Code

```
student_long2[, year := ifelse(variable == 1, 2021, 2022)][, variable := NULL]
student_long2
```

6. Reshape Data

Why reshape data?

Summarizing is easier in long form.

- Example: average math/reading score by year.

Run Code

```
# --- using long-form --- #
student_long[, .(
  avg_math = mean(math),
  avg_reading = mean(reading)
), by = year]

# --- using wide-form --- #
student_wide[, `:=`(
  avg_math = (math_2021 + math_2022) / 2,
  avg_reading = (reading_2021 + reading_2022) / 2
)]

student_wide[, .(student, avg_math, avg_reading)]
```

Visualization is easier in long form.

6. Reshape Data

In-class exercise

Questions

Using the following long-form data named `long_data`, can you get back `student_long`?

Run Code

```
# === create long_dt (run this code) === #
student_wide <- dcast(student_long, student ~ year, value.var = c("math", "reading"))
long_data <- melt(student_wide, id.var = "student")
```

Run Code

```
# You can write your code here
# Hint: you can use `tstrsplit()` function to split the variable column by "_"
```

7. Merge Multiple Datasets

Basics

You can use the `merge()` function from the `data.table` package to merge two datasets.

Basic Syntax:

```
# Merge data y to data x keeping all rows from data x
merge(x, y, by = "key_column", all.x = TRUE)
```

- `x`, `y`: data tables.
- `by`: specifies variables that let you merge two datasets.
- `all.x = TRUE` means that all rows from `x` are maintained in the merged dataset, and only matching rows from `y` are included.

Note: The order of the datasets matter.

7. Merge Multiple Datasets

Example

Instructions | Merge

Let’s play around with the `merge()` function using the following small data.

7. Merge Multiple Datasets

In-class exercise1

Questions

1. In the `flights` data, the `carrier` column contains two-letter codes for airlines. Let's translate these codes into the full name of the airline.

Airline data from `nycflights13` package contains the full name of the airline corresponding to the two-letter code. The following code loads the airline data.

Run Code

```
1  airlines <- nycflights13::airlines
2  head(airlines)
```

```
# A tibble: 6 × 2
  carrier name
  <chr>   <chr>
1 9E      Endeavor Air Inc.
2 AA      American Airlines Inc.
3 AS      Alaska Airlines Inc.
4 B6      JetBlue Airways
5 DL      Delta Air Lines Inc.
6 EV      ExpressJet Airlines Inc.
```

Merge `flights` and `airlines` data, keeping all rows from the `flights` data. Which variable should be used as a key column?

Run Code

```
1  # You can write your code here
```

7. Merge Multiple Datasets

In-class exercise 2

Questions

Run the following code to create two datasets: `yield_data` and `weather_data`.

Run Code

```r
math_data <-
  data.table(
    student = rep(c("Alice", "Bob", "Carol", "David", "Eva"), each = 2),
    year = rep(2021:2022, times = 5),
    math = runif(n = 10, min = 60, max = 100)
  )

reading_data <-
  data.table(
    student = rep(c("Alice", "Bob", "Carol", "Frank", "Grace"), each = 4),
    year = rep(2021:2024, times = 5),
    reading = runif(20, min = 60, max = 100)
  )
```

Merge these two datasets, keeping all rows from `math_data`. Which variable(s) should be used as key columns?

Run Code

```r
# you can write your code here
```

```
1 # you can write your code here
```

%>% operator

Motivation

- In R, you need to assign the result of each operation to a new object if you want to use the result in the subsequent process.
- But sometimes, some objects are just intermediate results that you don't need to keep.

Example

Let's first create `flights_mini` data from `flights` data of `nycflights13` package in the `data.table` format.

Run Code

```
1  flights <- nycflights13::flights # Load flights data from nycflights13
2  flights_dt <- as.data.table(flights) # change the data to data.table class
3  flights_mini <- flights_dt[,.(year, month, origin, dest, carrier, air_time, dep_delay, arr_delay)] #
   select some columns
4  flights_mini <- na.omit(flights_mini) # remove rows with missing values
```

The first three lines yield intermediate results to make the final `flight_mini`, and you don't need to keep those.

You can create `flights_mini` without using those intermediate steps with the chaining operation in `data.table` package, but it's hard to read!

Run Code

```
1  flights_mini <- na.omit(data.table(nycflights13::flights)[,.(year, month, origin, dest, carrier, air_time,
   dep_delay, arr_delay)])
```

Introduction

What is %>%?

- `%>%` a special symbol in R, called a pipe operator. It comes from the `magrittr` package.
- It's a powerful tool to write linear sequence of operations in a more readable way.

Note: When you load the `dplyr` package, `magrittr` package is automatically loaded as well. So, you don't need to load the `magrittr` package separately to use `%>%`.

Introduction

Basics

%>% takes the output of the code on its left and feeds it as the first argument to the function on its right.

Example 1

```
1 fun1(input1)
```

is the same as

```
1 input1 %>% fun1()
```

Example 2

```
1 output1 <- fun1(input1)
2 output2 <- fun2(output1)
```

is the same as

```
1 output2 <- fun1(input1) %>% fun2()
```

Introduction

Refer to the Preceding Object

What if you want to use the object defined before `%>%` as the second or third argument of the subsequent function?

You can refer the preceding object by `.` in the subsequent function.

Example

Run Code

```
# Let's use this function
print_three_words <- function(x, y, z) paste(c(x, y, z),collapse = " ")
# For example, this function prints three words with spaces between them
print_three_words(x="I", y="love", z="R")

# pass the input to the first argument
"I" %>% print_three_words(x=., y="love", z="R")

# pass the input to the second argument
"love" %>% print_three_words(x="I", y=., z="R")

# pass the input to the third argument
"R" %>% print_three_words(x="I", y="love", z=.)
```

Tip - Whenever you use `%>%`, I recommend you always use `.` in the subsequent function to **explicitly denote the destination of the object** defined before `%>%` even if it is the first argument.

Example

Without %>%

Run Code

```
flights <- nycflights13::flights
flights_dt <- as.data.table(flights)
flights_mini <- flights_dt[,.(year, month, origin, dest, carrier, air_time, dep_delay, arr_delay)]
flights_mini <- na.omit(flights_mini)
```

With %>%

Run Code

```
library(dplyr)

flights_mini <-
  nycflights13::flights %>%
  as.data.table(.) %>%
  .[,.(year, month, origin, dest, carrier, air_time, dep_delay, arr_delay)] %>%
  na.omit(.)
```

Note that the order of execution is the same as the order in which the functions are written.

Summary

The topics in the second part of this lecture were more advanced, so you don't need to memorize every function right away.

What I want you to remember are the following key ideas:

1. You can reshape data using the functions `dcast()` and `melt()`. Depending on your goal, one format (wide or long) may be easier to analyze than the other.
2. You can merge datasets using the `merge()` function, but you must have at least one common key column between the datasets.

You don't need to use `%>%` operator, unless you thinks it would be more convenient.

After-class Exercise Problems

Exercise 1

Instructions

1. Find the flight company with the longest departure delay. (Hint: use `max()` function to find the maximum value of `dep_delay` column)
2. Subset the information of flights that headed to MSP (Minneapolis-St Paul International Airport) in February. Let's name it "msp_feb_flights". How many flights are there?
3. Calculate the median, interquartile range ($IQR = Q3 - Q1$) for `arr_delays` of flights in in the `msp_feb_flights` dataset and the number of flights, grouped by `carrier`. Which carrier has the most variable arrival delays?

- Hints: IQR = Q3 − Q1 (the difference between the 75th percentile and the 25th percentile.) Use `quantile()` function to calculate the quantiles.

Exercise 1

Answers

Run Code

```
# === Part 1 === #
flights[dep_delay == max(dep_delay), .(carrier)]

# === Part 2 === #
msp_feb_flights <- flights[dest=="MSP" & month==2L]
nrow(msp_feb_flights)

# === Part 3 === #
msp_feb_flights[,.(
  median = median(arr_delay),
  IQR = quantile(arr_delay, 0.75) - quantile(arr_delay, 0.25),
  n_flights = .N
  ), by = carrier]
```

Exercise 2

Instructions

If you were selecting an airport simply based on on-time departure percentage, which NYC airport would you choose to fly out of? - To address this question, first, define a new variable which indicates on-time departure. On-time-departure can be defined as a departure delay of less than or equal to 0. Then, calculate the on-time departure rate for each airport.

Run Code

```
1  # You can write your code here
```

Exercise 2

Answers

Run Code

```
flights <- data.table(nycflights13::flights)

flights[, .(on_time_rate = mean(dep_delay <= 0, na.rm = TRUE)), by = origin]

flights[, on_time := dep_delay <= 0] %>%
  .[, .(on_time_rate = mean(on_time, na.rm = TRUE)), by = origin]
```

Exercise 3

Data

For this exercise problem, we will use `journal` data from the `AER` package.
- First, load the data and convert it to data.table object using `setDT` function (or. `as.data.table()`). Take a look at the data.
- Also, type `?journal` to see the description of the data.

Run Code

```
# If you have not installed the package, run the following code
# install.packages("AER")

# load the package
library(AER)
# load the data from AER
data("Journals")

# To see the descriptions of the data,
# type `?Journals` in the console
?Journals

setDT(Journals)
```

Exercise 3

Instructions

1. Calculate the average number of pages and price for the entire dataset.
2. Show the `title`, `citations`, `price`, and `subs` columns for the top 5 journals (`title`) with the highest number of citations (`citations`). (Hint: use `order()` function to sort the data by `citations` in descending order.).
3. This dataset is created in the year 2000. Calculate the age (`age`) of each journal by subtracting the start year (`foundingyear`) of the journal from 2000. Select the columns, `price`, `subs`, `citations`, and `pages`, and `age`. Use that data to create a correlation matrix between those variables using the `cor()` function. (Hint: use this syntax: `cor(data)`). Can you find anything interesting from the correlation matrix?

Run Code

```
1  # You can write your code here
```

Exercise 3

Solutions

Run Code

```
# === Part 1 === #
Journals[, .(
  avg_pages = mean(pages, na.rm = TRUE),
  avg_price = mean(price, na.rm = TRUE)
)]

# === Part 2 === #
Journals[order(-citations), .(title, citations, price, subs)][1:5]

# === Part 3 === #
Journals[, age := 2000 - foundingyear]
cor(Journals[, .(price, subs, citations, pages, age)])
```

Appendix

Useful functions

- `.N`
- `copy()`
- `setnames()`
- `order()`
- `shift()`
- `duplicated()`: find duplicates
- `unique()`: find unique observations
- `fcase()`

fcase()

- `fcase()` function is useful when you want to define a variable that takes different values based on conditions.
- `fcase()` function returns the first value for which the corresponding condition is `TRUE`. If no condition is `TRUE`, it returns the default value.

Run Code

```
x = 1:10
fcase(
    x <= 5L, 1L,
    x > 5L, 3L
)
```

Example: Define seasons (Winter: Dec-Feb, Spring: Mar-May, Summer: Jun-Aug, Fall: Sep-Nov)

Run Code

```
# --- Define season --- #
flights[,season := fcase(
  month %in% c(12, 1, 2), "Winter",
  month %in% c(3, 4, 5), "Spring",
  month %in% c(6, 7, 8), "Summer",
  default = "Fall" #otherwise, "Fall`"
)]
flights
```