COMP824 2023 Week 5 Transforming Data

Dr Sarah Marshall
sarah.marshall@aut.ac.nz WZ Level 9

Department of Mathematical Sciences
Auckland University of Technology

Overview

The Process of Analytics

Tibbles

Transforming Data `dplyr`

Workflow: R scripts

Reading

Chapter 5, 6, 10 Wickham and Grolemund (2020), R for Data Science https://r4ds.had.co.nz/

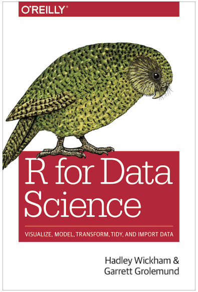
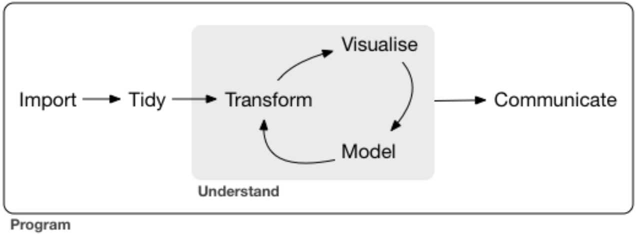


Figure 1: http://r4ds.had.co.nz/

The Process of Analytics



Learning objectives

- Understand the difference between data frames and tibbles
- Know how to filter, arrange, select, mutate and summarise data with `dplyr`
- Appreciate the beauty and usefulness of “the pipe” `%>%`
- Understand the key principles of using R scripts

Tibbles

Definition: Tibbles are “optionated data frames that make working in the tidyverse a little easier”. Wickham and Grolemund (2020)

Figure 3: https://https://www.tidyverse.org/

Example: Data frames

```r
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Example: Tibbles

```r
as_tibble(iris)
```

```
# A tibble: 150 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          5.1         3.5          1.4         0.2 setosa
2          4.9         3            1.4         0.2 setosa
3          4.7         3.2          1.3         0.2 setosa
# ... with 147 more rows
```

```r
class(as_tibble(iris))
```

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

Creating Tibbles

- Coerce a data frame `as_tibble`
- Create a new tibble `tibble`

Creating Tibbles

```
tibble(
  x = 1:3,
  y = 1,
  z = x ^ 2 + y,
  `x squared` = x^2
)
```

```
# A tibble: 3 x 4
      x     y     z `x squared`
  <int> <dbl> <dbl>       <dbl>
1     1     1     2           1
2     2     1     5           4
3     3     1    10           9
```

Use backticks ’ to define and access non-traditional column names (e.g. spaces, starting with numbers etc.)

Tribbles

Transposed tibbles – for easier data entry

```
tribble(
  ~x, ~y, ~z,
  #--|--|----
  "a", 2, 3.6,
  "b", 1, 8.5
)
```

```
# A tibble: 2 x 3
  x         y     z
  <chr> <dbl> <dbl>
1 a         2   3.6
2 b         1   8.5
```

Tibbles vs Data Frames: Default Printing Options

- Tibbles: max 10 rows, limited columns, variable type
- Data frame: (almost) all rows, all columns

Tibbles vs Data Frames: Customing Printing Options 1

- Tibble (local)

```r
nycflights13::flights %>%
  print(n = 10, width = Inf)
```

- Tibble (global)

```r
options(pillar.print_max = n,
        pillar.print_min = m,
        tibble.width = Inf)
```

Tibbles vs Data Frames: Customing Printing Options 2

- Tibble (view)

```
nycflights13::flights %>% View()
```

- Data frame

```
head(iris, 10)
tail(iris)
?print.data.frame
```

Tibbles vs Data Frames: Subsetting

- `$` extract a variable by name
- `[[]]` extract by a variable name or position

Tibbles vs Data Frames: Subsetting (Example)

```
tb <- tibble(x = 1:5,
             y = 11:15)

# Extract by name
tb$x; tb[["x"]]
```

```
[1] 1 2 3 4 5

[1] 1 2 3 4 5
```

```
# Extract by position
tb[[2]]
```

```
[1] 11 12 13 14 15
```

```
# Extract using a pipe - note the dot
tb %>% .$x
```

Tibbles to data frames

```
as.data.frame(tb)
```

```
  x  y
1 1 11
2 2 12
3 3 13
4 4 14
5 5 15
```

```
class(tb)
```

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

```
class(as.data.frame(tb))
```

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

Transforming Data `dplyr`

Description: `dplyr` is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges https://dplyr.tidyverse.org/

Figure 4: https://https://www.tidyverse.org/

Transforming data with dplyr

5 key functions

- filter() – pick observations based on criteria
- arrange() – reorder the rows based on a column value
- select() – select specified columns by name
- mutate() – create new variables from existing ones
- summarise() – create summaries

Also useful:

- group_by() – apply functions by group
- rename() – renames variables

2013 NYC Flights Data

```r
library(nycflights13)
library(tidyverse)
```

```r
?flights
```

2013 NYC Flights Data

```r
flights
```

```
# A tibble: 336,776 x 19
   year month   day dep_time sched~1 dep_d~2 arr_t~3 sched~4
  <int> <int> <int>    <int>   <int>   <dbl>   <int>   <int>
1  2013     1     1      517     515       2     830     819
2  2013     1     1      533     529       4     850     830
3  2013     1     1      542     540       2     923     850
# ... with 336,773 more rows, 11 more variables:
#   arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>, and abbreviated
#   variable names 1: sched_dep_time, 2: dep_delay,
#   3: arr_time, 4: sched_arr_time
```

Filter

`filter` – pick observations based on criteria

```
(jan2 <- filter(flights, month == 1, day == 2))
```

```
# A tibble: 943 x 19
   year month   day dep_time sched~1 dep_d~2 arr_t~3 sched~4
  <int> <int> <int>    <int>   <int>   <dbl>   <int>   <int>
1  2013     1     2       42    2359      43     518     442
2  2013     1     2      126    2250     156     233    2359
3  2013     1     2      458     500      -2     703     650
# ... with 940 more rows, 11 more variables:
#   arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>, and abbreviated
#   variable names 1: sched_dep_time, 2: dep_delay,
#   3: arr_time, 4: sched_arr_time
```

Filter - with logic

More examples:

```
on_schedule <- filter(flights, arr_delay <= 20, dep_delay <= 20)

on_time_departure <- filter(flights, between(dep_delay, -1, 1))


# The following two statements give the same results
nov_dec <- filter(flights,  month == 11 | month == 12)
nov_dec_v2 <- filter(flights,  month %in% c(11, 12))
```

Arrange

`arrange` – reorder the rows based on a column value

```r
arrange(flights, month, day, sched_dep_time)
```

```
# A tibble: 336,776 x 19
   year month   day dep_time sched~1 dep_d~2 arr_t~3 sched~4
  <int> <int> <int>    <int>   <int>   <dbl>   <int>   <int>
1  2013     1     1      517     515       2     830     819
2  2013     1     1      533     529       4     850     830
3  2013     1     1      542     540       2     923     850
# ... with 336,773 more rows, 11 more variables:
#   arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>, and abbreviated
#   variable names 1: sched_dep_time, 2: dep_delay,
#   3: arr_time, 4: sched_arr_time
```

Arrange - Further examples

Question: Which flights had the longest departure delays?

```
arrange(flights, dep_delay)
arrange(flights, desc(dep_delay)) #Descending order
```

```
arrange(flights, -dep_delay)      #Descending order
```

```
# A tibble: 336,776 x 19
   year month   day dep_time sched~1 dep_d~2 arr_t~3 sched~4
  <int> <int> <int>    <int>   <int>   <dbl>   <int>   <int>
1  2013     1     9      641     900    1301    1242    1530
2  2013     6    15     1432    1935    1137    1607    2120
3  2013     1    10     1121    1635    1126    1239    1810
# ... with 336,773 more rows, 11 more variables:
#   arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>,
```

Select

`select` – select specified columns by name

```
select(flights, year, month, day)
```

```
# A tibble: 336,776 x 3
   year month   day
  <int> <int> <int>
1  2013     1     1
2  2013     1     1
3  2013     1     1
# ... with 336,773 more rows
```

Select - Further examples 1

```
# Select all columns from year to day
select(flights, year:day)

#Select all columns except year:day
select(flights, -(year:day))

# Select columns related to departures + tailnum
select(flights, starts_with("dep"), tailnum)
select(flights, contains("dep"))

# Move time_hour and air_time to start then
# include remaining columns
select(flights, time_hour, air_time, everything())
```

Select - Further examples 2

```
(flights_sml <- select(flights,
                      year:day,
                      ends_with("delay"),
                      distance,
                      air_time
))
```

```
# A tibble: 336,776 x 7
   year month   day dep_delay arr_delay distance air_time
  <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
1  2013     1     1         2        11     1400      227
2  2013     1     1         4        20     1416      227
3  2013     1     1         2        33     1089      160
# ... with 336,773 more rows
```

Mutate - create new variables

```
mutate(flights_sml,
       gain = arr_delay - dep_delay,
       hours = air_time / 60,
       gain_per_hour = gain / hours
)
```

```
# A tibble: 336,776 x 10
   year month   day dep_delay arr_de~1 dista~2 air_t~3  gain
  <int> <int> <int>     <dbl>    <dbl>   <dbl>   <dbl> <dbl>
1  2013     1     1         2       11    1400     227     9
2  2013     1     1         4       20    1416     227    16
3  2013     1     1         2       33    1089     160    31
# ... with 336,773 more rows, 2 more variables:
#   hours <dbl>, gain_per_hour <dbl>, and abbreviated
#   variable names 1: arr_delay, 2: distance, 3: air_time
```

Transmute - only keep new variables

```
#Keep only new variables
transmute(flights,
          gain = arr_delay - dep_delay,
          hours = air_time / 60,
          gain_per_hour = gain / hours
)
```

```
# A tibble: 336,776 x 3
   gain hours gain_per_hour
  <dbl> <dbl>         <dbl>
1     9  3.78          2.38
2    16  3.78          4.23
3    31  2.67         11.6
# ... with 336,773 more rows
```

Mutate + Modular arithmetic

```r
transmute(flights,
  dep_time,
  hour = dep_time %/% 100,
  minute = dep_time %% 100
)
```

```
# A tibble: 336,776 x 3
  dep_time  hour minute
     <int> <dbl>  <dbl>
1      517     5     17
2      533     5     33
3      542     5     42
# ... with 336,773 more rows
```

Summarise and Group By

`group_by` – create groups within dataframe

`summarise` – summarise by group

Discussion: Which month has the greatest average departure delay?

Summarise and Group By: Departure Delays By Month

```r
by_month <- group_by(flights, month)
summarise(by_month, delay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 12 x 2
  month delay
  <int> <dbl>
1     1  10.0
2     2  10.8
3     3  13.2
# ... with 9 more rows
```

Summarise and Group By: Departure Delays By Month 2

```r
by_month <- group_by(flights, month)
delay_by_month <- summarise(by_month,
                            delay = mean(dep_delay, na.rm = TRUE))
arrange(delay_by_month, -delay)
```

```
# A tibble: 12 x 2
  month delay
  <int> <dbl>
1     7  21.7
2     6  20.8
3    12  16.6
# ... with 9 more rows
```

Summarise and Group By: Arrival Delays by Month

```
summarise(by_month,
  avg_arr_delay1 = mean(arr_delay, na.rm = TRUE),
  avg_arr_delay2 = mean(arr_delay[arr_delay > 0], na.rm = TRUE)
                 # the average positive delay
)
```

```
# A tibble: 12 x 3
  month avg_arr_delay1 avg_arr_delay2
  <int>          <dbl>          <dbl>
1     1           6.13           34.5
2     2           5.61           33.7
3     3           5.81           40.6
# ... with 9 more rows
```

Summarise and Group By: Number of flights per day

```r
daily <- group_by(flights, year, month, day)
(per_day   <- summarise(daily, flights = n()))
```

```
# A tibble: 365 x 4
# Groups:   year, month [12]
   year month   day flights
  <int> <int> <int>   <int>
1  2013     1     1     842
2  2013     1     2     943
3  2013     1     3     914
# ... with 362 more rows
```

Summarise and Group By: Number of flights per month

Each `summarise` removes a layer of the grouping.

```
(per_month <- summarise(per_day, flights = sum(flights)))
```

```
# A tibble: 12 x 3
# Groups:   year [1]
   year month flights
  <int> <int>   <int>
1  2013     1   27004
2  2013     2   24951
3  2013     3   28834
# ... with 9 more rows
```

Summarise and Group By: Number of flights per year

```r
(per_year  <- summarise(per_month, flights = sum(flights)))
```

```
# A tibble: 1 x 2
   year flights
  <int>   <int>
1  2013  336776
```

The pipe %>%

Figure 5: https://https://www.tidyverse.org/

The pipe %>%

The pipe operator %>% makes analysis much easier.

It “pipes” the results of a function into the first argument of the next function.

```
x <- 1:4

# Without a pipe
mean(x^2)
```

```
[1] 7.5
```

```
# With a pipe
x^2 %>% mean()
```

```
[1] 7.5
```

The pipe %>%

```
# Without a pipe
by_month <- group_by(flights, month)
delay_by_month <- summarise(by_month,
                            delay = mean(dep_delay, na.rm = TRUE))
arrange(delay_by_month, -delay)

# With a pipe
flights %>%
  group_by(month) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE)) %>%
  arrange(-delay)
```

The new pipe |>

In R 4.1 and later there is a new pipe operator built-in to Base R.

It works a similar way to the `maggritr` pipe, but there are some differences

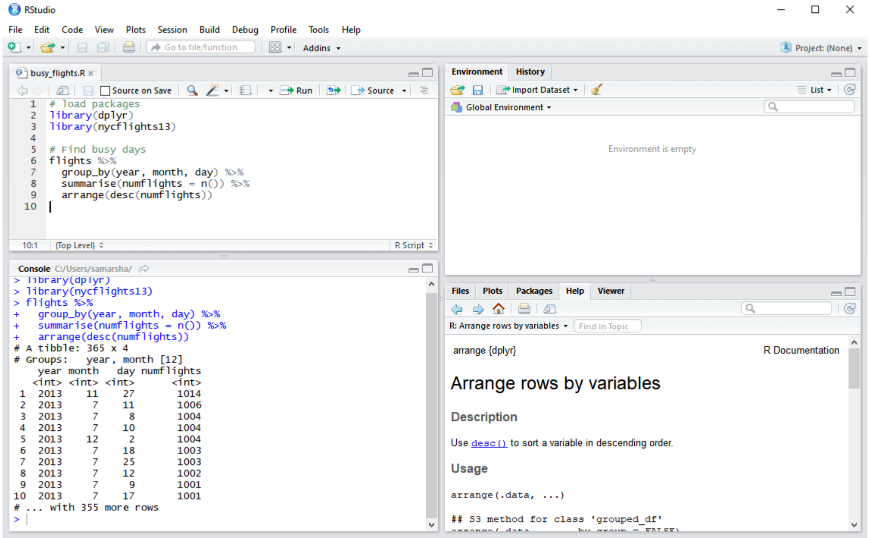
```
x %>% mean()
x %>% mean
x |> mean()
x |> mean   # error: the new pipe needs brackets
```

For more info: https://www.infoworld.com/article/3621369/use-the-new-r-pipe-built-into-r-41.html

Workflow: R scripts

R scripts

R scripts are files containing R commands



```
# load packages
library(dplyr)
library(nycflights13)

# Find busy days
flights %>%
  group_by(year, month, day) %>%
  summarise(numflights = n()) %>%
  arrange(desc(numflights))
```

```
> library(dplyr)
> library(nycflights13)
> flights %>%
+   group_by(year, month, day) %>%
+   summarise(numflights = n()) %>%
+   arrange(desc(numflights))
# A tibble: 365 x 4
# Groups:   year, month [12]
    year month   day numflights
   <int> <int> <int>      <int>
 1  2013    11    27       1014
 2  2013     7    11       1006
 3  2013     7     8       1004
 4  2013     7    10       1004
 5  2013    12     2       1004
 6  2013     7    18       1003
 7  2013     7    25       1003
 8  2013     7    12       1002
 9  2013     7     9       1001
10  2013     7    17       1001
# ... with 355 more rows
>
```

arrange {dplyr} R Documentation

Arrange rows by variables

Description

Use desc() to sort a variable in descending order.

Usage

```
arrange(.data, ...)

## S3 method for class 'grouped_df'
```

R scripts

Good practice

- Load packages at the start
- Use comments
- Store code not results
- Never save or restore .Rdata into workspace (Tools/Global Options/General)
- Use relative not absolute paths
- If sharing files – including `install.packages()` or `setwd()` is “anti-social”

Keyboard shortcuts

- `Cmd/Ctrl + Enter`: Run current R expression
- `Cmd/Ctrl + Shift + S`: Run/Source complete R script
- `Cmd/Ctrl + Shift + F10`: Restart RStudio

Learning objectives

- Understand the difference between data frames and tibbles
- Know how to filter, arrange, select, mutate and summarise data with `dplyr`
- Appreciate the beauty and usefulness of “the pipe” `%>%`
- Understand the key principles of using R scripts

References

Wickham, Hadley, and Garrett Grolemund. 2020. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*.