R1: Basic Data Wrangling

Data manipulation with `{dplyr}` in `{tidyverse}`

Andreas Reschreiter

2023-11-12

Table of contents

1 Tidy data modelling

Packages used in this notebook:

```
suppressPackageStartupMessages({
  library(tidyverse)
})
```

1.1 Tidy data and tidyverse

The "`{tidyverse}`" is a set of related R packages designed for data science. One of the core packages is `{dplyr}` used for transformation of data frames or, better, "`tibble()`". A `tibble()` (from the abbreviation "tbl" for table) is basically a data frame that "behaves" and prints in a slightly better way than standard R `data.frame()`.

1.2 Tidy data

Tidy data satisfies the following criteria:

- Each column represents a variable (i.e. feature)
- Each row is an observation (i.e. case) of data

Data in `tibble()` and `data.frame()` is tipically tidy data.[1]

2 Pipe operator

The `{tidyverse}` pipe `%>%` (`magrittr` pipe) and base R pipe `|>` (native pipe).

2.1 The `{tidyverse}` pipe `%>%`

- The function `f(x,a)` can be written as `x %>% f(a)`.
- The pipe `%>%` simplifies the execution of several functions.
- For example `h(g(f(x,a),b),c)` becomes the more readable expression `x %>% f(a) %>% g(b) %>% h(c)` as the functions are placed one after the other.
- The `{tidyverse}` pipe `%>%` is part of the `{magrittr}` package that contains additional pipe operators. The `%>%` is therefore known as `{magrittr}` pipe.
- The pipe `%>%` requires tidy data and reads as "and then".

[1] The `{tidyverse}` has this tidy data principle in its name.

- The `%>%` operator must always appear at the end of lines.
- To persist changes
  - `data <- data %>% ...` or:
  - `data %<>% ...`
- By default piped into first function argument
  - `x %>% foo(y)` becomes `foo(x, y)`
- Non default piped into not first function argument
  - `y %>% foo(x, .)` becomes `foo(x, y)`
  - `d %>% lm(y~x,data=.)` becomes `lm(y~x,data=d)`
  - `mydata %>% .[[1]]` becomes `mydata[[1]]`

2.2 The base R pipe |> operator

- Since R version 4.1 a base R pipe `|>` operator exists. It is called **native pipe** (needs no loading of a package).
- By default `|>` pipes into first function argument
  `x |> foo(y)` becomes `foo(x, y)`
- Non default pipe by naming (all) arguments
  `y |> foo(a=x,b=)` becomes `foo(a=x,b=y)`
  `d |> lm(formula=y~x,data=)`
- Non default pipe by function definition
  `mydata |> \(x) lm(y~x, data = x)`
  `mydata |> \(.) lm(y~x, data = .)`

3 Make tibbles

From `data.frame()` with `as_tibble()`, column-wise from vectors with `tibble()` and row-wise with `tribble()`.

3.1 From data frames

Use the command `as_tibble()` to create a tibble from a data frame. Consider the `datasets::iris` data frame

```
str(iris)
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
data = as_tibble(iris)
str(data)
```

```
tibble [150 x 5] (S3: tbl_df/tbl/data.frame)
 $ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Note: The resulting `tibble()` is still also a `data.frame()` and all operations for a `data.frame()` still work (due to class inheritance).

```
class(data)
```

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

```
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
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# i 140 more rows
```

The `iris` data in the format of a `tibble` prints on "one screen" in contrast to the original `iris` data in the data frame that prints out the whole data (in the R-Markdown output).

3.2 Column-wise

Create a `tibble` column-wise from data vectors with the command `tibble()`:

```
tibble(A = 1:26, B = letters)
```

```
# A tibble: 26 x 2
       A B
   <int> <chr>
 1     1 a
 2     2 b
 3     3 c
 4     4 d
 5     5 e
 6     6 f
 7     7 g
 8     8 h
 9     9 i
10    10 j
# i 16 more rows
```

3.3 Row-wise

Create a tibble by row-wise data specification with the command `tribble()`. This is particularly useful for adding new cases of data (observations) via new lines of data entries.

```
tribble(~A, ~B,
        1,  "bla",
        2,  "foo")
```

```
# A tibble: 2 x 2
      A B
  <dbl> <chr>
1     1 bla
2     2 foo
```

4 Basic data manipulation

The ground functions of the `{dplyr}` package are: `select()`, `filter()`, `mutate()`, `arrange()`, and `group_by()` / `summarize()`.

4.1 `select()` columns

`select()` is used to **choose** and **exclude** *columns*:

```
select(data, Petal.Width, Species)
select(data, -Sepal.Length, -Sepal.Width, -Petal.Length)
```

```
# A tibble: 150 x 2
   Petal.Width Species
         <dbl> <fct>
 1         0.2 setosa
 2         0.2 setosa
 3         0.2 setosa
 4         0.2 setosa
 5         0.2 setosa
 6         0.4 setosa
 7         0.3 setosa
 8         0.2 setosa
 9         0.2 setosa
10         0.1 setosa
# i 140 more rows
```

```
# A tibble: 150 x 2
   Petal.Width Species
         <dbl> <fct>
 1         0.2 setosa
 2         0.2 setosa
 3         0.2 setosa
 4         0.2 setosa
 5         0.2 setosa
 6         0.4 setosa
 7         0.3 setosa
 8         0.2 setosa
 9         0.2 setosa
10         0.1 setosa
# i 140 more rows
```

`select()` on **parts of column name**:

```
# NOTE uses chunk option: #| layout-ncol: 3
select(data, starts_with("Sepal")) %>% head() # shorten printout
select(data, ends_with("Length"))  |>  head() # native pipe
select(data, contains("l.L"))      %>% head   # shorten printout
```

4.2 `filter()` rows

`filter()` to **choose** or **exclude** rows

```
# NOTE filter first and select afterwards
filter(data, Species == "versicolor") |> select(1:2)
```

```
# A tibble: 6 x 2
  Sepal.Length Sepal.Width
         <dbl>       <dbl>
1          5.1         3.5
2          4.9         3
3          4.7         3.2
4          4.6         3.1
5          5           3.6
6          5.4         3.9
```

```
# A tibble: 6 x 2
  Sepal.Length Petal.Length
         <dbl>        <dbl>
1          5.1          1.4
2          4.9          1.4
3          4.7          1.3
4          4.6          1.5
5          5            1.4
6          5.4          1.7
```

```
# A tibble: 6 x 2
  Sepal.Length Petal.Length
         <dbl>        <dbl>
1          5.1          1.4
2          4.9          1.4
3          4.7          1.3
4          4.6          1.5
5          5            1.4
6          5.4          1.7
```

```
filter(data, Species != "versicolor") |> select(last_col())
```

```
# A tibble: 50 x 2
   Sepal.Length Sepal.Width
          <dbl>       <dbl>
 1          7           3.2
 2          6.4         3.2
 3          6.9         3.1
 4          5.5         2.3
 5          6.5         2.8
 6          5.7         2.8
 7          6.3         3.3
 8          4.9         2.4
 9          6.6         2.9
10          5.2         2.7
# i 40 more rows
```

```
# A tibble: 100 x 1
   Species
   <fct>
 1 setosa
 2 setosa
 3 setosa
 4 setosa
 5 setosa
 6 setosa
 7 setosa
 8 setosa
 9 setosa
10 setosa
# i 90 more rows
```

4.2.1 `filter()` composed conditions

```
# NOTE uses chunk option: #| layout-nrow: 2
filter(data, Sepal.Length > 5 & Sepal.Width > 4) ## same as next
filter(data, Sepal.Length > 5 , Sepal.Width > 4) ## same as above
```

```
# A tibble: 3 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          5.7         4.4          1.5         0.4 setosa
2          5.2         4.1          1.5         0.1 setosa
3          5.5         4.2          1.4         0.2 setosa
```

```
# A tibble: 3 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          5.7         4.4          1.5         0.4 setosa
2          5.2         4.1          1.5         0.1 setosa
3          5.5         4.2          1.4         0.2 setosa
```


4.2.2 `filter()` specified range

```
filter(data, between(Petal.Length, 1, 1.2))
```

```
# A tibble: 4 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          4.3         3            1.1         0.1 setosa
2          5.8         4            1.2         0.2 setosa
3          4.6         3.6          1           0.2 setosa
4          5           3.2          1.2         0.2 setosa
```

4.3 `mutate()` variables

4.3.1 Modify existing variables

```
mutate(data, Sepal.Length = round(Sepal.Length)) %>%
  head(3)
```

```
# A tibble: 3 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1            5         3.5          1.4         0.2 setosa
2            5         3            1.4         0.2 setosa
3            5         3.2          1.3         0.2 setosa
```

4.3.2 Add new variable (based on an existing ones)

```
mutate(data, Sepal = ifelse(Sepal.Length > 5, "Long", "Short"), .after = 2) %>%
  head()
```

```
# A tibble: 6 x 6
  Sepal.Length Sepal.Width Sepal Petal.Length Petal.Width Species
         <dbl>       <dbl> <chr>        <dbl>       <dbl> <fct>
1          5.1         3.5 Long           1.4         0.2 setosa
2          4.9         3   Short          1.4         0.2 setosa
3          4.7         3.2 Short          1.3         0.2 setosa
4          4.6         3.1 Short          1.5         0.2 setosa
5          5           3.6 Short          1.4         0.2 setosa
6          5.4         3.9 Long           1.7         0.4 setosa
```

4.3.3 Add new variable (independent variable)

```
add_column(data,
           ID = paste("id", 1:150),
           .before = "Sepal.Length") %>% head()
```

```
# A tibble: 6 x 6
  ID    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
  <chr>        <dbl>       <dbl>        <dbl>       <dbl> <fct>
1 id 1           5.1         3.5          1.4         0.2 setosa
2 id 2           4.9         3            1.4         0.2 setosa
3 id 3           4.7         3.2          1.3         0.2 setosa
4 id 4           4.6         3.1          1.5         0.2 setosa
5 id 5           5           3.6          1.4         0.2 setosa
6 id 6           5.4         3.9          1.7         0.4 setosa
```

4.3.4 Remove variable

```
mutate(data, Species = NULL) %>% head()
```

```
# A tibble: 6 x 4
  Sepal.Length Sepal.Width Petal.Length Petal.Width
         <dbl>       <dbl>        <dbl>       <dbl>
1          5.1         3.5          1.4         0.2
2          4.9         3            1.4         0.2
3          4.7         3.2          1.3         0.2
4          4.6         3.1          1.5         0.2
5          5           3.6          1.4         0.2
6          5.4         3.9          1.7         0.4
```

4.4 `rename()` variables

```
rename(data, Sepal_Length = "Sepal.Length") %>% head()
```

```
# A tibble: 6 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
4          4.6         3.1          1.5         0.2 setosa
5          5           3.6          1.4         0.2 setosa
6          5.4         3.9          1.7         0.4 setosa
```

4.5 `arrange()` rows to sort data

Sort ascending:

```
arrange(data, Sepal.Length) # NOTE sort ascending is default
```

```
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          4.3         3            1.1         0.1 setosa
```

```
 2          4.4         2.9          1.4         0.2 setosa
 3          4.4         3            1.3         0.2 setosa
 4          4.4         3.2          1.3         0.2 setosa
 5          4.5         2.3          1.3         0.3 setosa
 6          4.6         3.1          1.5         0.2 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          4.6         3.6          1           0.2 setosa
 9          4.6         3.2          1.4         0.2 setosa
10          4.7         3.2          1.3         0.2 setosa
# i 140 more rows
```

Sort `Species` descending and then according to `Sepal.Length` ascending:

```
arrange(data, desc(Species), Sepal.Length)
```

```
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          4.9         2.5          4.5         1.7 virginica
 2          5.6         2.8          4.9         2   virginica
 3          5.7         2.5          5           2   virginica
 4          5.8         2.7          5.1         1.9 virginica
 5          5.8         2.8          5.1         2.4 virginica
 6          5.8         2.7          5.1         1.9 virginica
 7          5.9         3            5.1         1.8 virginica
 8          6           2.2          5           1.5 virginica
 9          6           3            4.8         1.8 virginica
10          6.1         3            4.9         1.8 virginica
# i 140 more rows
```

4.6 `top_n()` for top rows

```
top_n(data, 5, Sepal.Width)
```

```
# A tibble: 6 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
```

```
1          5.4         3.9          1.7         0.4 setosa
2          5.8         4            1.2         0.2 setosa
3          5.7         4.4          1.5         0.4 setosa
4          5.4         3.9          1.3         0.4 setosa
5          5.2         4.1          1.5         0.1 setosa
6          5.5         4.2          1.4         0.2 setosa
```

Note that 6 rows are returned because of ties, and data are not sorted.

4.7 `summarize()` columns

Create summary statistics (UK `summarise()` is same as `summarize()`):

```
summarize(data,
          N = n(),
          minimum = min(Sepal.Length),
          maximum = max(Sepal.Length),
          mean = mean(Petal.Width))
```

```
# A tibble: 1 x 4
      N minimum maximum  mean
  <int>   <dbl>   <dbl> <dbl>
1   150     4.3     7.9  1.20
```

4.8 `group_by()` to group columns data

Summaries on grouped data:

```
summarize(group_by(data, Species),
          N = n(),
          minimum = min(Sepal.Length),
          maximum = max(Sepal.Length),
          mean = mean(Petal.Width))
```

```
# A tibble: 3 x 5
  Species        N minimum maximum  mean
  <fct>      <int>   <dbl>   <dbl> <dbl>
1 setosa        50     4.3     5.8 0.246
2 versicolor    50     4.9     7   1.33
3 virginica     50     4.9     7.9 2.03
```

Examples of using `group_by()` without aggregation:

```
  mutate(group_by(data, Species), id = 1:n())
```

```
# A tibble: 150 x 6
# Groups:   Species [3]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species    id
          <dbl>       <dbl>        <dbl>       <dbl> <fct>   <int>
 1          5.1         3.5          1.4         0.2 setosa      1
 2          4.9         3            1.4         0.2 setosa      2
 3          4.7         3.2          1.3         0.2 setosa      3
 4          4.6         3.1          1.5         0.2 setosa      4
 5          5           3.6          1.4         0.2 setosa      5
 6          5.4         3.9          1.7         0.4 setosa      6
 7          4.6         3.4          1.4         0.3 setosa      7
 8          5           3.4          1.5         0.2 setosa      8
 9          4.4         2.9          1.4         0.2 setosa      9
10          4.9         3.1          1.5         0.1 setosa     10
# i 140 more rows
```

This gives each group in the Species an index `id`:

```
  slice(mutate(group_by(data, Species), id = 1:n()), 1:3)
```

```
# A tibble: 9 x 6
# Groups:   Species [3]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species       id
         <dbl>       <dbl>        <dbl>       <dbl> <fct>      <int>
1          5.1         3.5          1.4         0.2 setosa         1
2          4.9         3            1.4         0.2 setosa         2
3          4.7         3.2          1.3         0.2 setosa         3
4          7           3.2          4.7         1.4 versicolor     1
5          6.4         3.2          4.5         1.5 versicolor     2
6          6.9         3.1          4.9         1.5 versicolor     3
7          6.3         3.3          6           2.5 virginica      1
8          5.8         2.7          5.1         1.9 virginica      2
9          7.1         3            5.9         2.1 virginica      3
```

Using the pipe %>% to achieve the same result:

```
data %>%  group_by(Species) %>% mutate(id = 1:n()) %>% slice(1:3)
```

```
# A tibble: 9 x 6
# Groups:   Species [3]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species       id
         <dbl>       <dbl>        <dbl>       <dbl> <fct>      <int>
1          5.1         3.5          1.4         0.2 setosa         1
2          4.9         3            1.4         0.2 setosa         2
3          4.7         3.2          1.3         0.2 setosa         3
4          7           3.2          4.7         1.4 versicolor     1
5          6.4         3.2          4.5         1.5 versicolor     2
6          6.9         3.1          4.9         1.5 versicolor     3
7          6.3         3.3          6           2.5 virginica      1
8          5.8         2.7          5.1         1.9 virginica      2
9          7.1         3            5.9         2.1 virginica      3
```

Group on two columns:

```
data %>% mutate(Sepal = ifelse(Sepal.Length > 5, "Long", "Short")) %>%
  group_by(Species, Sepal) %>%
  mutate(id = 1:n()) %>%
  slice(1:2)
```

```
# A tibble: 11 x 7
# Groups:   Species, Sepal [6]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species    Sepal    id
          <dbl>       <dbl>        <dbl>       <dbl> <fct>      <chr> <int>
 1          5.1         3.5          1.4         0.2 setosa     Long      1
 2          5.4         3.9          1.7         0.4 setosa     Long      2
 3          4.9         3            1.4         0.2 setosa     Short     1
 4          4.7         3.2          1.3         0.2 setosa     Short     2
 5          7           3.2          4.7         1.4 versicolor Long      1
 6          6.4         3.2          4.5         1.5 versicolor Long      2
 7          4.9         2.4          3.3         1   versicolor Short     1
```

```
 8          5           2           3.5         1   versicolor Short     2
 9          6.3         3.3         6           2.5 virginica  Long      1
10          5.8         2.7         5.1         1.9 virginica  Long      2
11          4.9         2.5         4.5         1.7 virginica  Short     1
```

Note: operations using `group_by()` follow the split/transform/combine paradigm.