Data Manipulation 1

STA 032: Gateway to data science Lecture 5

Jingwei Xiong

April 12, 2023

Today

- Data manipulation using tidyverse

Introduction

Up to now we have been manipulating vectors by subsetting them through indexing.

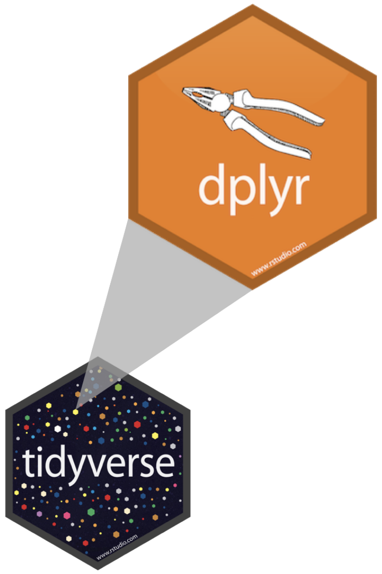
However, once we start more advanced analyses, the preferred unit for data storage is not the vector but the **data frame**.

We will learn to work **directly** with data frames, which greatly facilitate the organization of information.

We can load all the tidyverse packages at once by installing and loading the **tidyverse** package:

```r
#install.packages(tidyverse)
library(tidyverse)
```

Data manipulation using `dplyr` (included in tidyverse)



- `select`: pick columns by name
- `arrange`: reorder rows
- `slice`: pick rows using index(es)
- `filter`: pick rows matching criteria
- `distinct`: filter for unique rows
- `mutate`: add new variables
- `summarize`: reduce variables to values
- `group_by`: for grouped operations
- … (many more)

As we go over the examples, think about how you would do these in base R, you will understand how powerful it is.

Rules of `dplyr` functions

- First argument is always a data frame
- Subsequent arguments say what to do with that data frame
- Always return a data frame
- Don't modify in place
    - Meaning that you need an assignment operation if you want an "updated" version of the data frame

Data: Hotel bookings

- Data from two hotels: one resort and one city hotel
- Observations: Each row represents a hotel booking
- Goal for original data collection: Development of prediction models to classify a hotel booking's likelihood to be cancelled (Antonia et al., 2019)

```r
hotels <- readr::read_csv("data/hotels.csv")
```

Source: TidyTuesday

`glimpse`: What is in the data set?

Example   glimpse

```
dplyr::glimpse(hotels)
```

```
Rows: 119,390
Columns: 32
$ hotel                          <chr> "Resort Hotel", "Resort Hotel", "Resort…
$ is_canceled                    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, …
$ lead_time                      <dbl> 342, 737, 7, 13, 14, 14, 0, 9, 85, 75, …
$ arrival_date_year              <dbl> 2015, 2015, 2015, 2015, 2015, 2015, 201…
$ arrival_date_month             <chr> "July", "July", "July", "July", "July",…
$ arrival_date_week_number       <dbl> 27, 27, 27, 27, 27, 27, 27, 27, 27, 27,…
$ arrival_date_day_of_month      <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
$ stays_in_weekend_nights        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ stays_in_week_nights           <dbl> 0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 4, 4, 4, …
$ adults                         <dbl> 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, …
$ children                       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ babies                         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ meal                           <chr> "BB", "BB", "BB", "BB", "BB", "BB", "BB…
$ country                        <chr> "PRT", "PRT", "GBR", "GBR", "GBR", "GBR…
$ market_segment                 <chr> "Direct", "Direct", "Direct", "Corporat…
$ distribution_channel           <chr> "Direct", "Direct", "Direct", "Corporat…
$ is_repeated_guest              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ previous_cancellations         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ previous_bookings_not_canceled <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ reserved_room_type             <chr> "C", "C", "A", "A", "A", "A", "C", "C",…
$ assigned_room_type             <chr> "C", "C", "C", "A", "A", "A", "C", "C",…
$ booking_changes                <dbl> 3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ deposit_type                   <chr> "No Deposit", "No Deposit", "No Deposit…
$ agent                          <chr> "NULL", "NULL", "NULL", "304", "240", "…
$ company                        <chr> "NULL", "NULL", "NULL", "NULL", "NULL",…
$ days_in_waiting_list           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ customer_type                  <chr> "Transient", "Transient", "Transient", …
$ adr                            <dbl> 0.00, 0.00, 75.00, 75.00, 98.00, 98.00,…
```

`glimpse`: What is in the data set?

Example glimpse

- glimpse from the dplyr package
- It allows you to quickly inspect the structure of a data frame
- It provides a compact and informative summary of the data frame
- Shows the number of observations, variables, and data type of each variable
- Useful for data exploration and identifying issues or inconsistencies
- Can verify that the data frame has been loaded correctly and variables are in the expected format

`select()`: Select columns

single column | multiple columns | exclude columns

View only `lead_time` (number of days between booking and arrival date):

```
select(hotels, lead_time)
```

```
# A tibble: 119,390 × 1
   lead_time
       <dbl>
 1       342
 2       737
 3         7
 4        13
 5        14
 6        14
 7         0
 8         9
 9        85
10        75
# … with 119,380 more rows
```

- First argument: data frame we're working with , `hotels`
- Second argument: variable we want to select, `lead_time`
- Result: data frame with 119390 rows and 1 column
- This is an alternative to `hotels$lead_time`

`select()`: Select columns

single column | **multiple columns** | exclude columns

View only the `hotel` type and `lead_time` columns:

```
select(hotels, hotel, lead_time)
```

```
# A tibble: 119,390 × 2
   hotel        lead_time
   <chr>            <dbl>
 1 Resort Hotel       342
 2 Resort Hotel       737
 3 Resort Hotel         7
 4 Resort Hotel        13
 5 Resort Hotel        14
 6 Resort Hotel        14
 7 Resort Hotel         0
 8 Resort Hotel         9
 9 Resort Hotel        85
10 Resort Hotel        75
# … with 119,380 more rows
```

`select()`: Select columns

single column | multiple columns | **exclude columns**

- We saw earlier that `select()` keeps variables
- `select()` can also exclude variables, using the – sign

```
select(hotels, -agent)
```

```
# A tibble: 119,390 × 31
   hotel  is_ca…¹ lead_…² arriv…³ arriv…⁴ arriv…⁵ arriv…⁶ stays…⁷ stays…⁸ adults
   <chr>    <dbl>   <dbl>   <dbl> <chr>     <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
 1 Resor…       0     342    2015 July         27       1       0       0      2
 2 Resor…       0     737    2015 July         27       1       0       0      2
 3 Resor…       0       7    2015 July         27       1       0       1      1
 4 Resor…       0      13    2015 July         27       1       0       1      1
 5 Resor…       0      14    2015 July         27       1       0       2      2
 6 Resor…       0      14    2015 July         27       1       0       2      2
 7 Resor…       0       0    2015 July         27       1       0       2      2
 8 Resor…       0       9    2015 July         27       1       0       2      2
 9 Resor…       1      85    2015 July         27       1       0       3      2
10 Resor…       1      75    2015 July         27       1       0       3      2
# … with 119,380 more rows, 21 more variables: children <dbl>, babies <dbl>,
#   meal <chr>, country <chr>, market_segment <chr>,
#   distribution_channel <chr>, is_repeated_guest <dbl>,
#   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>,
#   reserved_room_type <chr>, assigned_room_type <chr>, booking_changes <dbl>,
#   deposit_type <chr>, company <chr>, days_in_waiting_list <dbl>,
#   customer_type <chr>, adr <dbl>, required_car_parking_spaces <dbl>, …
```

`select()`: Select columns, continue

a range of variables | starts_with | ends_with

- Instead of writing out all the variable names, `select()` also accepts a range of variables
- This follows the order they are listed in the data frame

```r
colnames(hotels)[1:10]
```

```
 [1] "hotel"                       "is_canceled"
 [3] "lead_time"                   "arrival_date_year"
 [5] "arrival_date_month"          "arrival_date_week_number"
 [7] "arrival_date_day_of_month"   "stays_in_weekend_nights"
 [9] "stays_in_week_nights"        "adults"
```

```r
select(hotels, hotel:arrival_date_month)
```

```
# A tibble: 119,390 × 5
   hotel        is_canceled lead_time arrival_date_year arri
   <chr>              <dbl>     <dbl>             <dbl> <chr
 1 Resort Hotel           0       342              2015 July
 2 Resort Hotel           0       737              2015 July
 3 Resort Hotel           0         7              2015 July
 4 Resort Hotel           0        13              2015 July
 5 Resort Hotel           0        14              2015 July
 6 Resort Hotel           0        14              2015 July
 7 Resort Hotel           0         0              2015 July
 8 Resort Hotel           0         9              2015 July
 9 Resort Hotel           1        85              2015 July
10 Resort Hotel           1        75              2015 July
# … with 119,380 more rows
```

`select()`: Select columns, continue

a range of variables | **starts_with** | ends_with

- We can also select columns with certain characteristics

```r
select(hotels, starts_with("arrival"))
```

```
# A tibble: 119,390 × 4
arrival_date_year arrival_date_month arrival_date_week_number arrival_date_…¹
           <dbl> <chr>                                  <dbl>           <dbl>
1              2015 July                                      27               1
2              2015 July                                      27               1
3              2015 July                                      27               1
4              2015 July                                      27               1
5              2015 July                                      27               1
6              2015 July                                      27               1
7              2015 July                                      27               1
8              2015 July                                      27               1
9              2015 July                                      27               1
10              2015 July                                      27               1
# … with 119,380 more rows, and abbreviated variable name
#   ¹arrival_date_day_of_month
```

`select()`: Select columns, continue

a range of variables | starts_with | **ends_with**

- We can also select columns with certain characteristics

```
select(hotels, ends_with("type"))
```

```
# A tibble: 119,390 × 4
   reserved_room_type assigned_room_type deposit_type customer_type
   <chr>              <chr>              <chr>        <chr>
 1 C                  C                  No Deposit   Transient
 2 C                  C                  No Deposit   Transient
 3 A                  C                  No Deposit   Transient
 4 A                  A                  No Deposit   Transient
 5 A                  A                  No Deposit   Transient
 6 A                  A                  No Deposit   Transient
 7 C                  C                  No Deposit   Transient
 8 C                  C                  No Deposit   Transient
 9 A                  A                  No Deposit   Transient
10 D                  D                  No Deposit   Transient
# … with 119,380 more rows
```

Select helpers

- `starts_with()`: Starts with a prefix
- `ends_with()`: Ends with a suffix
- `contains()`: Contains a literal string
- `num_range()`: Matches a numerical range like x01, x02, x03
- `one_of()`: Matches variable names in a character vector
- `everything()`: Matches all variables
- `last_col()`: Select last variable, possibly with an offset
- `matches()`: Matches a regular expression (a sequence of symbols/characters expressing a string/pattern to be searched for within text)

> But you can always select by listing column names in this course. These methods are for situations where there are lots of variables.

See help for any of these functions for more info, e.g. `?everything`.

`select()`, then `arrange()`

What if we wanted to select these columns, and then sort the data in order of lead time?

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(lead_time)
```

```
# A tibble: 119,390 × 2
   hotel        lead_time
   <chr>            <dbl>
 1 Resort Hotel         0
 2 Resort Hotel         0
 3 Resort Hotel         0
 4 Resort Hotel         0
 5 Resort Hotel         0
 6 Resort Hotel         0
 7 Resort Hotel         0
 8 Resort Hotel         0
 9 Resort Hotel         0
10 Resort Hotel         0
# … with 119,380 more rows
```

> Wait, what is that **%>%**???

Pipes

In Data science we can perform a series of operations, in the previous example we `select` and then `arrange`, by sending the results of one function to another using what is called the *pipe operator*: `|>`. or `%>%`

To illustrate the operations, it can be shown as:

$$\text{original data} \rightarrow \text{select} \rightarrow \text{arrange}$$

For such an operation, we can use the pipe `|>`. The code looks like this:

```
hotels |>  select(hotel, lead_time) |> arrange(lead_time)
```

Pipes

In Data science we can perform a series of operations, in the previous example we `select` and then `arrange`, by sending the results of one function to another using what is called the *pipe operator*: |>. or `%>%`. They are equivalent

- Start with the data frame `hotels`, and pass it to the `select()` function,

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(lead_time)
```

```
# A tibble: 119,390 × 2
   hotel        lead_time
   <chr>            <dbl>
 1 Resort Hotel         0
 2 Resort Hotel         0
 3 Resort Hotel         0
 4 Resort Hotel         0
 5 Resort Hotel         0
 6 Resort Hotel         0
 7 Resort Hotel         0
 8 Resort Hotel         0
 9 Resort Hotel         0
10 Resort Hotel         0
# … with 119,380 more rows
```

Pipes

In Data science we can perform a series of operations, in the previous example we `select` and then `arrange`, by sending the results of one function to another using what is called the *pipe operator*: |>. or `%>%`. They are equivalent

- Start with the data frame `hotels`, and pass it to the `select()` function,
- then we select the variables `hotel` and `lead_time`,

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(lead_time)
```

```
# A tibble: 119,390 × 2
   hotel        lead_time
   <chr>            <dbl>
 1 Resort Hotel         0
 2 Resort Hotel         0
 3 Resort Hotel         0
 4 Resort Hotel         0
 5 Resort Hotel         0
 6 Resort Hotel         0
 7 Resort Hotel         0
 8 Resort Hotel         0
 9 Resort Hotel         0
10 Resort Hotel         0
# … with 119,380 more rows
```

In Data science we can perform a series of operations, in the previous example we `select` and then `arrange`, by sending the results of one function to another using what is called the *pipe operator*: |>. or `%>%`. They are equivalent

- Start with the data frame `hotels`, and pass it to the `select()` function,
- then we select the variables `hotel` and `lead_time`,
- and then we arrange the data frame by `lead_time`.

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(lead_time)
```

```
# A tibble: 119,390 × 2
   hotel        lead_time
   <chr>            <dbl>
 1 Resort Hotel         0
 2 Resort Hotel         0
 3 Resort Hotel         0
 4 Resort Hotel         0
 5 Resort Hotel         0
 6 Resort Hotel         0
 7 Resort Hotel         0
 8 Resort Hotel         0
 9 Resort Hotel         0
10 Resort Hotel         0
# … with 119,380 more rows
```

Note that the pipe operator is implemented in the package `magrittr`, but is **automatically loaded** when we use `library(dplyr)` or `library(tidyverse)`.

How does a pipe work?

- You can think about the following sequence of actions - find keys, unlock car, start car, drive to work, park.
- Expressed as a set of nested functions in R pseudocode this would look like:

```
park(drive(start_car(find("keys")), to = "work"))
```

- Writing it out using pipes give it a more natural (and easier to read) structure:

```
find("keys") %>%
  start_car() %>%
  drive(to = "work") %>%
  park()
```

- Note this is the **coding style** you need to follow. Each line represents an action about the dataset, and connected with pipes.

Simple example

- We can write `exp(1)` with pipes as `1 %>% exp`, and `log(exp(1))` as `1 %>% exp %>% log`

```
exp(1)
```

```
[1] 2.718282
```

```
1 %>% exp
```

```
[1] 2.718282
```

```
1 %>% exp %>% log
```

```
[1] 1
```

- **Remarks: Tidyverse functions are at their best when composed together using the pipe operator**

`arrange()` in ascending or descending order

- We saw earlier that `arrange()` defaults to ascending order
- For descending order, use `desc()`

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(lead_time)
```

```
# A tibble: 119,390 × 2
   hotel        lead_time
   <chr>            <dbl>
 1 Resort Hotel         0
 2 Resort Hotel         0
 3 Resort Hotel         0
 4 Resort Hotel         0
 5 Resort Hotel         0
 6 Resort Hotel         0
 7 Resort Hotel         0
 8 Resort Hotel         0
 9 Resort Hotel         0
10 Resort Hotel         0
# … with 119,380 more rows
```

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
# A tibble: 119,390 × 2
   hotel        lead_time
   <chr>            <dbl>
 1 Resort Hotel       737
 2 Resort Hotel       709
 3 City Hotel         629
 4 City Hotel         629
 5 City Hotel         629
 6 City Hotel         629
 7 City Hotel         629
 8 City Hotel         629
 9 City Hotel         629
10 City Hotel         629
# … with 119,380 more rows
```

`slice()` for certain row numbers

This is an alternative indexing option for `hotels[1:5, ]`

```r
hotels %>%
  slice(1:5)
```

```
# A tibble: 5 × 32
  hotel    is_ca…¹ lead_…² arriv…³ arriv…⁴ arriv…⁵ arriv…⁶ stays…⁷ stays…⁸ adult
  <chr>      <dbl>   <dbl>   <dbl> <chr>     <dbl>   <dbl>   <dbl>   <dbl>  <c
1 Resort…        0     342    2015 July         27       1       0       0
2 Resort…        0     737    2015 July         27       1       0       0
3 Resort…        0       7    2015 July         27       1       0       1
4 Resort…        0      13    2015 July         27       1       0       1
5 Resort…        0      14    2015 July         27       1       0       2
# … with 22 more variables: children <dbl>, babies <dbl>, meal <chr>,
#   country <chr>, market_segment <chr>, distribution_channel <chr>,
#   is_repeated_guest <dbl>, previous_cancellations <dbl>,
#   previous_bookings_not_canceled <dbl>, reserved_room_type <chr>,
#   assigned_room_type <chr>, booking_changes <dbl>, deposit_type <chr>,
#   agent <chr>, company <chr>, days_in_waiting_list <dbl>,
#   customer_type <chr>, adr <dbl>, required_car_parking_spaces <dbl>, …
```

Reminder: comments in R

- Any text following # will be printed as is, and won't be run as code
- This is useful for leaving comments and for temporarily disabling certain lines of code (for debugging, trying out different things)

```
hotels %>%
  # slice the first five rows  # this line is a comment
  #select(hotel) %>%           # this one doesn't run
  slice(1:5)                   # this line runs
```

```
# A tibble: 5 × 32
  hotel   is_ca…¹ lead_…² arriv…³ arriv…⁴ arriv…⁵ arriv…⁶ stays…⁷ stays…⁸ adults
  <chr>     <dbl>   <dbl>   <dbl> <chr>     <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
1 Resort…       0     342    2015 July         27       1       0       0      2
2 Resort…       0     737    2015 July         27       1       0       0      2
3 Resort…       0       7    2015 July         27       1       0       1      1
4 Resort…       0      13    2015 July         27       1       0       1      1
5 Resort…       0      14    2015 July         27       1       0       2      2
# … with 22 more variables: children <dbl>, babies <dbl>, meal <chr>,
#   country <chr>, market_segment <chr>, distribution_channel <chr>,
#   is_repeated_guest <dbl>, previous_cancellations <dbl>,
#   previous_bookings_not_canceled <dbl>, reserved_room_type <chr>,
#   assigned_room_type <chr>, booking_changes <dbl>, deposit_type <chr>,
#   agent <chr>, company <chr>, days_in_waiting_list <dbl>,
#   customer_type <chr>, adr <dbl>, required_car_parking_spaces <dbl>, …
```

`filter()` to select a subset of rows

```r
# bookings in City Hotels
hotels %>%
  filter(hotel == "City Hotel")
```

```
# A tibble: 79,330 × 32
   hotel  is_ca…¹ lead_…² arriv…³ arriv…⁴ arriv…⁵ arriv…⁶ stays…⁷ stays…⁸ adults
   <chr>    <dbl>   <dbl>   <dbl> <chr>     <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
 1 City …       0       6    2015 July         27       1       0       2      1
 2 City …       1      88    2015 July         27       1       0       4      2
 3 City …       1      65    2015 July         27       1       0       4      1
 4 City …       1      92    2015 July         27       1       2       4      2
 5 City …       1     100    2015 July         27       2       0       2      2
 6 City …       1      79    2015 July         27       2       0       3      2
 7 City …       0       3    2015 July         27       2       0       3      1
 8 City …       1      63    2015 July         27       2       1       3      1
 9 City …       1      62    2015 July         27       2       2       3      2
10 City …       1      62    2015 July         27       2       2       3      2
# … with 79,320 more rows, 22 more variables: children <dbl>, babies <dbl>,
#   meal <chr>, country <chr>, market_segment <chr>,
#   distribution_channel <chr>, is_repeated_guest <dbl>,
#   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>,
#   reserved_room_type <chr>, assigned_room_type <chr>, booking_changes <dbl>,
#   deposit_type <chr>, agent <chr>, company <chr>, days_in_waiting_list <dbl>,
#   customer_type <chr>, adr <dbl>, required_car_parking_spaces <dbl>, …
```

What was the base R alternative that we saw?

`filter()` for many conditions at once

```
hotels %>%
  filter(
    adults == 0,
    children >= 1
    ) %>%
  select(adults, babies, children)
```

```
# A tibble: 223 × 3
   adults babies children
    <dbl>  <dbl>    <dbl>
 1      0      0        3
 2      0      0        2
 3      0      0        2
 4      0      0        2
 5      0      0        2
 6      0      0        3
 7      0      1        2
 8      0      0        2
 9      0      0        2
10      0      0        2
# … with 213 more rows
```

`filter()` for more complex conditions

```
# bookings with no adults and some children or babies in the room
hotels %>%
  filter(
    adults == 0,
    children >= 1 | babies >= 1
    ) %>%
  select(adults, babies, children)
```

```
# A tibble: 223 × 3
   adults babies children
    <dbl>  <dbl>    <dbl>
 1      0      0        3
 2      0      0        2
 3      0      0        2
 4      0      0        2
 5      0      0        2
 6      0      0        3
 7      0      1        2
 8      0      0        2
 9      0      0        2
10      0      0        2
# … with 213 more rows
```

Reminder: Logical operators in R

operator	definition	operator	definition
`<`	less than	`x | y`	`x` OR `y`
`<=`	less than or equal to	`is.na(x)`	test if `x` is `NA`
`>`	greater than	`!is.na(x)`	test if `x` is not `NA`
`>=`	greater than or equal to	`x %in% y`	test if `x` is in `y`
`==`	exactly equal to	`!(x %in% y)`	test if `x` is not in `y`
`!=`	not equal to	`!x`	not `x`
`x & y`	`x` AND `y`		

`mutate()` to add a new variable

```
hotels %>%
  mutate(little_ones = children + babies) %>%
  select(children, babies, little_ones) %>%
  arrange(desc(little_ones))
```

```
# A tibble: 119,390 × 3
   children babies little_ones
      <dbl>  <dbl>       <dbl>
 1       10      0          10
 2        0     10          10
 3        0      9           9
 4        2      1           3
 5        2      1           3
 6        2      1           3
 7        3      0           3
 8        2      1           3
 9        2      1           3
10        3      0           3
# … with 119,380 more rows
```

What are these functions doing? How do to the same in base R? Think about it after class!

A small question

```
hotels %>%
  mutate(little_ones = children + babies) %>%
  select(children, babies, little_ones) %>%
  arrange(desc(little_ones))
```

Notice that here we used `children` and `babies` inside the function, which are objects that are **not** defined in our workspace. But why don't we get an error?

This is one of **dplyr**'s main features. Functions in this package, such as `mutate`, know to **look for variables** in the data frame provided in the first argument. In the call to mutate above, `children` will have the values in `hotels$children`. This approach makes the code much more **readable**.

`summarize()` for summary stats

```r
# mean average daily rate for all bookings
hotels %>%
  summarize(mean_adr = mean(adr))
```

```
# A tibble: 1 × 1
  mean_adr
     <dbl>
1     102.
```

- `summarize()` changes the data frame entirely
- Rows are collapsed into a single summary statistic
- Columns that are irrelevant to the calculation are removed

`summarize()` is often used with `group_by()`

- For grouped operations
- There are two types of `hotel`, city and resort hotels
- We want the mean daily rate for bookings at city vs. resort hotels

```
hotels %>%
  group_by(hotel) %>%
  summarize(mean_adr = mean(adr))
```

```
# A tibble: 2 × 2
  hotel        mean_adr
  <chr>           <dbl>
1 City Hotel      105. 
2 Resort Hotel     95.0
```

- `group_by()` can be used with more than one group

Multiple summary statistics

`summarize` can be used for multiple summary statistics as well.

```
hotels %>%
  summarize(
    n = n(), # frequencies
    min_adr = min(adr),
    mean_adr = mean(adr),
    median_adr = median(adr),
    max_adr = max(adr)
    )
```

```
# A tibble: 1 × 5
       n min_adr mean_adr median_adr max_adr
   <int>   <dbl>    <dbl>      <dbl>   <dbl>
1 119390   -6.38     102.       94.6    5400
```

head() and the top n

When datasets are large, output all results will fill up the page with the entire dataset.

head top_n

- We can use head() to show the first several rows

```
hotels |> head(5)
```

```
# A tibble: 5 × 32
  hotel    is_ca…¹ lead_…² arriv…³ arriv…⁴ arriv…⁵ arriv…⁶ stays…⁷ stays…⁸ adult
  <chr>      <dbl>   <dbl>   <dbl> <chr>     <dbl>   <dbl>   <dbl>   <dbl>  <c
1 Resort…        0     342    2015 July         27       1       0       0
2 Resort…        0     737    2015 July         27       1       0       0
3 Resort…        0       7    2015 July         27       1       0       1
4 Resort…        0      13    2015 July         27       1       0       1
5 Resort…        0      14    2015 July         27       1       0       2
# … with 22 more variables: children <dbl>, babies <dbl>, meal <chr>,
#   country <chr>, market_segment <chr>, distribution_channel <chr>,
#   is_repeated_guest <dbl>, previous_cancellations <dbl>,
#   previous_bookings_not_canceled <dbl>, reserved_room_type <chr>,
#   assigned_room_type <chr>, booking_changes <dbl>, deposit_type <chr>,
#   agent <chr>, company <chr>, days_in_waiting_list <dbl>,
#   customer_type <chr>, adr <dbl>, required_car_parking_spaces <dbl>, …
```

head() and the top n

When datasets are large, output all results will fill up the page with the entire dataset.

head top_n

- Or top_n() to show the rows regarding to the largest variables.

*Note that rows are not sorted by `adr`, only filtered. If we want to sort, we need to use `arrange`.

- Note that if the second argument is left blank, `top_n` filters by the last column.

```
hotels |> top_n(3, adr)
```

```
# A tibble: 3 × 32
  hotel    is_ca…¹ lead_…² arriv…³ arriv…⁴ arriv…⁵ arriv…⁶ stays…⁷ stays…⁸ adult
  <chr>      <dbl>   <dbl>   <dbl> <chr>     <dbl>   <dbl>   <dbl>   <dbl>  <c
1 Resort…        0       1    2015 July         29      15       0       1
2 City H…        1      35    2016 March        13      25       0       1
3 City H…        0       0    2017 May          19       9       0       1
# … with 22 more variables: children <dbl>, babies <dbl>, meal <chr>,
#   country <chr>, market_segment <chr>, distribution_channel <chr>,
#   is_repeated_guest <dbl>, previous_cancellations <dbl>,
#   previous_bookings_not_canceled <dbl>, reserved_room_type <chr>,
```

Tidyverse coding style

```
result <- dataset |>
  mutate SOMETHING |>
  filter SOMETHING |>
  select SOMETHING |>
  group_by SOMETHING |>
  summarise SOMETHING
```

If you write your code according to this workflow, it will be very clear and easy to understand. In homework 2 I suggest you try to practice writing code like this.

Summary

- Data manipulation tools
    - `select()`: selects columns by name
    - `arrange()`: reorders rows
    - `slice()`: selects rows using index(es)
    - `filter()`: selects rows matching criteria
    - `mutate()`: adds new variables
    - `summarize()`: reduces variables to values
    - `group_by()`: for grouped operations

Readings

- Chapter 4:The tidyverse
- R for Data Science Chapter 5