Tibbles and Tibble Indexing

Jarred Robidoux

2023-02-18

Tibbles

__Prerequisites___

In this chapter we'll explore the **tibble** package, part of the core tidyverse.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0     v purrr   1.0.1
## v tibble  3.1.8     v dplyr   1.1.0
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.3     v forcats 1.0.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Creating Tibbles

Almost all of the functions that you'll use in this book produce tibbles, as tibbles are one of the unifying features of the tidyverse. Most other R packages use regular data frames, so you might want to coerce a data frame to a tibble. You can do that with **as__tibble()**

```
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
## # ... with 140 more rows
```

You can create a new tibble from individual vectors with **tibble()**

```
tibble(
  x=1:5,
  y=1,
  z=x^2+y
)
```

```
## # A tibble: 5 x 3
##       x     y     z
##   <int> <dbl> <dbl>
## 1     1     1     2
## 2     2     1     5
## 3     3     1    10
## 4     4     1    17
## 5     5     1    26
```

It's possible for a tibble to have column names that are not valud R variable names, aka **non-syntactic** names. To refer these variable, you need to surround them with backticks, ‘

```
tibble(
  `:-(` = "frowny face",
  `2000` = "number"
)
```

```
## # A tibble: 1 x 2
##   ‘:-(‘       ‘2000‘
##   <chr>       <chr>
## 1 frowny face number
```

Another way to create a tibble is with **tribble()**, short for transposed tibble. **tribble()** is customised for data entry in code: column heading are defind by formulas (they start with ~), and entries are separated by commas.

```
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.frame

There are two main differences in the usage of a tibble vs a classic data.frame: printing and subsetting

Printing

Tibbles have a refined print method that shows only the first 10 rows, and all the columns that fit on the screen. This makes it much easier to work with large data. In addition to its name, each column reports its type, a nice feature borrowed from **str()**

Tibbles are designed so that you don't accidentally overwhelm your console when you print large data frame. But sometimes you need more output than the default display. There are a few options that can help.

First, you can explicitly **print()** the data frame and control the number of rows (n) and the **width** of the display. width = Inf will display all columns

```
install.packages("nycflights13", repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/Valued Customer/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
## package 'nycflights13' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Valued Customer\AppData\Local\Temp\Rtmp88CWVK\downloaded_packages
```

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

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <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
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
##    arr_delay carrier flight tailnum origin dest  air_time distance  hour minute
##        <dbl> <chr>    <int> <chr>   <chr>  <chr>    <dbl>    <dbl> <dbl>  <dbl>
##  1        11 UA        1545 N14228  EWR    IAH        227     1400     5     15
##  2        20 UA        1714 N24211  LGA    IAH        227     1416     5     29
##  3        33 AA        1141 N619AA  JFK    MIA        160     1089     5     40
##  4       -18 B6         725 N804JB  JFK    BQN        183     1576     5     45
##  5       -25 DL         461 N668DN  LGA    ATL        116      762     6      0
##  6        12 UA        1696 N39463  EWR    ORD        150      719     5     58
##  7        19 B6         507 N516JB  EWR    FLL        158     1065     6      0
##  8       -14 EV        5708 N829AS  LGA    IAD         53      229     6      0
##  9        -8 B6          79 N593JB  JFK    MCO        140      944     6      0
## 10         8 AA         301 N3ALAA  LGA    ORD        138      733     6      0
##    time_hour
##    <dttm>
```

```
##  1 2013-01-01 05:00:00
##  2 2013-01-01 05:00:00
##  3 2013-01-01 05:00:00
##  4 2013-01-01 05:00:00
##  5 2013-01-01 06:00:00
##  6 2013-01-01 05:00:00
##  7 2013-01-01 06:00:00
##  8 2013-01-01 06:00:00
##  9 2013-01-01 06:00:00
## 10 2013-01-01 06:00:00
## # ... with 336,766 more rows
```

Subsetting

So far all the tools you've learned have worked with complete data frames. If you want to pull out a single variable, you need some new tools, $**and**[[***.***[[**canextractbynameorposition*; $ only extracts by name but is a little less typing.

```
df <- tibble(
  x = runif(5),
  y = runif(5)
)
```

Extract by name

```
df$x
```

```
## [1] 0.6543853 0.6719294 0.9830780 0.2572913 0.2077841
```

Extract by position

```
df[[1]]
```

```
## [1] 0.6543853 0.6719294 0.9830780 0.2572913 0.2077841
```

To use these in a pipe, you'll need to use the special placeholder `.`

```
df %>%
  .$x
```

```
## [1] 0.6543853 0.6719294 0.9830780 0.2572913 0.2077841
```