L05 Tibbles

Data Science I (STAT 301-1)

Shay Lebovitz

Contents

Overview

The goal of this lab is to better understand the basic data structure called a "tibble," which is used throughout the `tidyverse` instead of R's traditional `data.frame`. Tibbles are data frames, but they tweak some of R's older behaviors to make life a little easier by avoiding unintentional mistakes/errors. The `tibble` package provides the framework for these opinionated data frames which make working with data in the tidyverse possible – `tibble` package home page.

Datasets

This lab utilizes the `mtcars` and `flights` datasets contained in the packages `datasets` (automatically loaded with base R) and `nycflights13`, respectively. For one of the problems, you may opt to use one of our other familiar datasets, `diamonds` or `mpg`, which are contained in `ggplot2`.

Exercises

Please complete the following exercises. Be sure your solutions are clearly indicated and that the document is neatly formatted.

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

Load Packages

Exercise 1

Please read the vignette for the `tibble` package.

```
# Access vignette
vignette("tibble")
```

Exercise 2

Demonstrate how to manually input the data table below into R using each of these functions:

- `tibble()`
- `tribble()`

price	store	ounces
3.99	target	128
3.75	walmart	128
3.00	amazon	128

```
tibble(price = c(3.99, 3.75, 3.00), store = c('target', 'walmart', 'amazon'),
       ounces = c(128, 128, 128))
```

```
## # A tibble: 3 x 3
##   price store   ounces
##   <dbl> <chr>    <dbl>
## 1  3.99 target     128
## 2  3.75 walmart    128
## 3  3    amazon     128
```

```
tribble(
  ~price, ~store, ~ounces,
  #--/--/----
  3.99, 'target', 128,
  3.75, 'walmart', 128,
  3.00, 'amazon', 128
)
```

```
## # A tibble: 3 x 3
##   price store   ounces
##   <dbl> <chr>    <dbl>
## 1  3.99 target     128
## 2  3.75 walmart    128
## 3  3    amazon     128
```

Exercise 3

How can you tell if an object is a tibble? Consider including an example or two. (Hint: try printing `mtcars`, which is a regular data frame).

```
print(mtcars)
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
```

```
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
print(tibble(mtcars))
```

```
## # A tibble: 32 x 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
##  2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
##  3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
##  4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
##  5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
##  6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
##  7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
##  8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
##  9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
## 10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
## # ... with 22 more rows
```

To tell if a variable is a tibble, you can type `class(my_tibble)`, or you can `print(my_tibble)`. If only 10 lines show, it is most likely a tibble, as data frames will print many more.

Exercise 4 (Website: 10.5 Ex. 1)

Turn `mtcars` into a tibble and ask R to print only the first 4 observations/rows.

```
mtcars_tib = tibble(mtcars)
mtcars_tib %>%
  print(n = 4)
```

```
## # A tibble: 32 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
```

```
## 1  21       6   160   110  3.9   2.62  16.5     0     1     4     4
## 2  21       6   160   110  3.9   2.88  17.0     0     1     4     4
## 3  22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
## 4  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
## # ... with 28 more rows
```

Exercise 5 (Website: 10.5 Ex. 2)

Run the following operations on `df` as a data frame. Then turn `df` into a tibble and run the operations again. What changes? Why might the default data frame behaviors cause problems or frustration?

```
df <- data.frame(abc = 1, xyz = "a")
df$x
df[, "xyz"]
df[, c("abc", "xyz")]

df <- tibble(df)
df$x
df[, "xyz"]
df[, c("abc", "xyz")]
```

Firstly, you cannot access a variable by just what it starts with as you can with data frames (no partial matching). Also, a data frame subset will return strings in quotes but a tibble won't.

Exercise 6 (Website: 10.5 Ex. 3)

If you have the name of a variable stored as an object, e.g. `var <- "mpg"`, how can you extract the specified variable from a tibble? Write code to demonstrate (we suggest selecting "mpg" from `mtcars`). **You would have to use the double brackets, so `df[[vars]]`. If you tried using the $ notation, R would look for a variable called `vars` instead of `'mpg'`.**

Exercise 7

How is subsetting via `[[ ]]` different from using `select()` when extracting columns of a tibble? (Hint: Investigate using one of our familiar datasets — `flights`, `diamonds`, or `mtcars`.)

```
diamonds %>%
  select(price)
```

```
## # A tibble: 53,940 x 1
##    price
##    <int>
##  1   326
##  2   326
##  3   327
##  4   334
##  5   335
##  6   336
##  7   336
##  8   337
##  9   337
## 10   338
```

```
## # ... with 53,930 more rows
```

```
diamonds[['price']]
```

`select()` will only show the first 10 entries, where as `[[]]` will show the whole set. I won't print `diamonds[['price']]` to save you from the scroll of death.

Exercise 8 (Website: 10.5 Ex. 4)

Practice referring to non-syntactic names in the following data frame by:

```
annoying <- tibble(
  `1` = 1:10,
  `2` = `1` * 2 + rnorm(length(`1`))
)
```

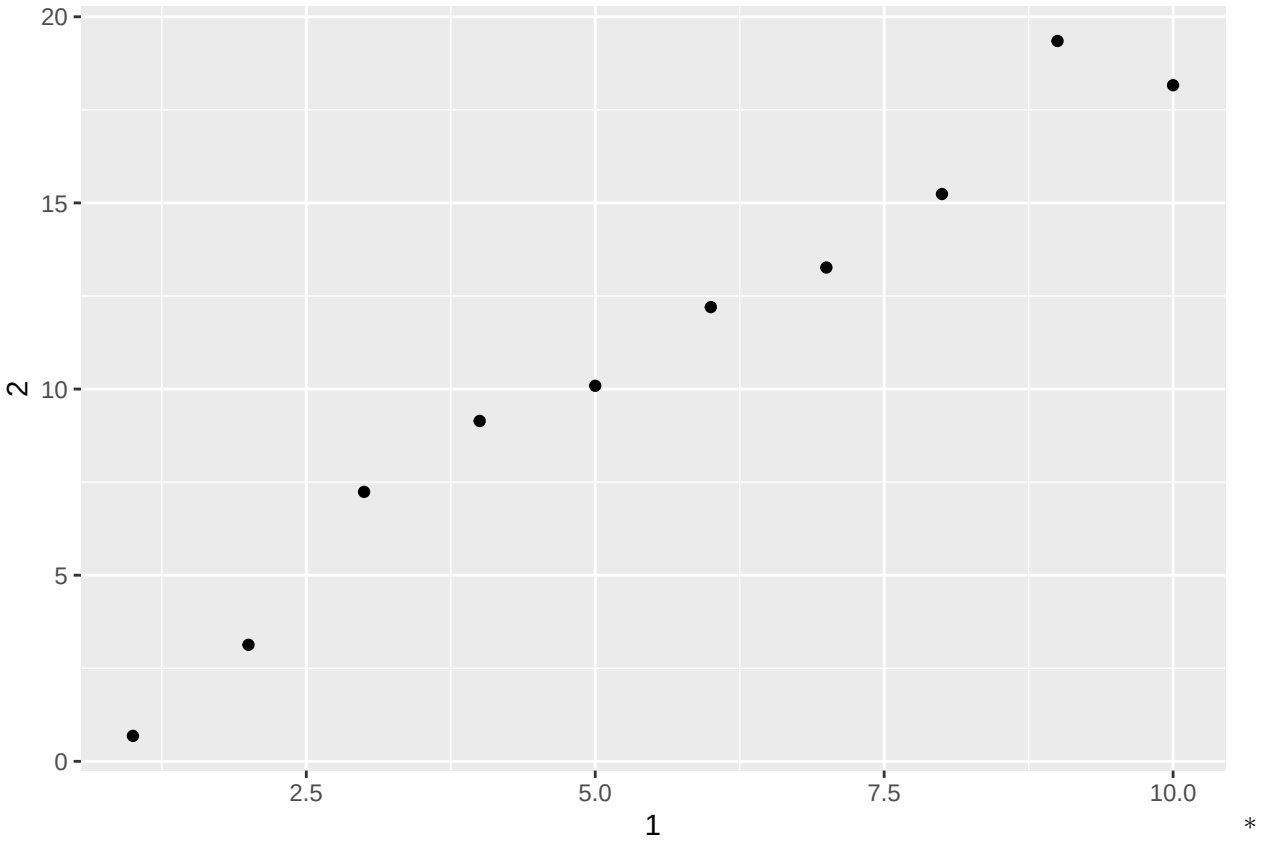
- Extracting the variable called 1.

```
annoying %>%
  select(1)
```

```
## # A tibble: 10 x 1
##      `1`
##    <int>
##  1     1
##  2     2
##  3     3
##  4     4
##  5     5
##  6     6
##  7     7
##  8     8
##  9     9
## 10    10
```

- Plotting a scatterplot of 1 vs 2.

```
ggplot(data = annoying) +
  geom_point(mapping = aes(x = `1`, y = `2`))
```



Creating a new column called 3 which is 2 divided by 1.

```
annoying$`3` = annoying$`2` / annoying$`1`
```

- Renaming the columns to one, two and three.

```
rename(annoying, 'one' = `1`, 'two' = `2`, 'three' = `3`)
```

```
## # A tibble: 10 x 3
##      one    two three
##    <int>  <dbl> <dbl>
##  1     1  0.685 0.685
##  2     2  3.13  1.56
##  3     3  7.24  2.41
##  4     4  9.14  2.29
##  5     5 10.1   2.02
##  6     6 12.2   2.03
##  7     7 13.3   1.90
##  8     8 15.2   1.90
##  9     9 19.3   2.15
## 10    10 18.2   1.82
```

Exercise 9 (Website: 10.5 Ex. 5 — modified)

What does `tibble::enframe()` do? When might you use it? (Hint: A named vector is one where each value has a specified name – examples in code below.)

```
# example of a named vector
named_vector <- c("I" = 3.14, "love" = 2.72, "stats!" = 1.61)
```

```
# Consider how enframe() would be helpful in the following example:

# random sample of 1000 from a normal ditribution
foo <- rnorm(n = 1000, mean = 100, sd = 15)
summary(foo)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   49.80   89.71  100.82  100.46  110.90  152.64
```

```
enframe(named_vector)
```

```
## # A tibble: 3 x 2
##   name   value
##   <chr>  <dbl>
## 1 I       3.14
## 2 love    2.72
## 3 stats!  1.61
```

```
enframe(summary(foo))
```

```
## # A tibble: 6 x 2
##   name    value
##   <chr>   <table>
## 1 Min.     49.80423
## 2 1st Qu.  89.71398
## 3 Median  100.82084
## 4 Mean    100.45521
## 5 3rd Qu. 110.89995
## 6 Max.    152.64061
```

You can store the summary of `foo` in a tibble for easier reference in future analysis.

Exercise 10

Apply `tibble::glimpse()` to the `flights` dataset. Then apply `print()` to `flights`. When/why might `glimpse()` be more useful than `print()`?

```
flights %>%
  glimpse()
```

```
## Rows: 336,776
## Columns: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 60...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2,...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 8...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 8...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7,...
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6"...
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301...
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LG...
```

```
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IA...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 73...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-0...
```

```
print(flights)
```

```
## # 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
## # ... with 336,766 more rows, and 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>
```

`glimpse()` is a transposed version of `print`, where the column names run down the screen instead of across. This is more useful than print when you have a lot of variables in a tibble and want to see all of them at once.

Exercise 11 (Website: 10.5 Ex. 6 — modified)

What option controls how many additional column names are printed at the footer of a tibble? (Hint: `package?tibble`) Provide an example using `flights`.

```
print(flights)
```

```
## # 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
## # ... with 336,766 more rows, and 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>
```

```
print(flights, n_extra = 1)
```

```
## # 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
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>, ...
```

The `n_extra` option allows you to control how many additional column names are printed at the footer of a tibble.