Data Frames Versus Tibbles

Abbie M Popa

9/19/2018

Data Frames versus Tibbles!

First lets look at strings as factors versus strings as strings

Make a data frame (not a tibble, do not suppress the default behavior) named "test1" where the first column is the names of three of your friends and the second column is their ages

```
test1 <- data.frame(names = c("DJ", "Victor", "Quinn"), ages = c(20, 19, 42))
```

Make a tibble named "test2" where the first column is the names of three of your friends and the second column is their ages

```
library(tibble)
test2 <- as_tibble(test1)
test2 <- tibble(names = c("DJ", "Victor", "Quinn"), ages = c(20, 19, 42))
test2 <- tribble(~names, ~ages,
                 "DJ", 20,
                 "Victor", 19,
                 "Quinn", 42)
```

Print test1

```
test1
```

```
##    names ages
## 1     DJ   20
## 2 Victor   19
## 3  Quinn   42
```

Print test2

```
test2
```

```
## # A tibble: 3 x 2
##   names   ages
##   <chr>  <dbl>
## 1 DJ        20
## 2 Victor    19
## 3 Quinn     42
```

Even with these tiny data frames, what do you notice is different in how the generic data frame and the tibble are printed?

- Gives data type under each column header
- Indicates number of rows and columns

Paste is a function that allows you to combine strings. Type each of the following commands

```
paste(test1[1,], collapse = " ")
```

```
## [1] "1 20"
```

```
paste(test2[1,], collapse = " ")
```

```
## [1] "DJ 20"
```

What happened when paste tried to combine a factor with a string? Feel free to play around with functions like class and typeof to develop your answer * It changes the factors to the strings “1”, “2”, and “3”

Now make the following dataset

```
test3 <- data.frame(numbers_as_numbers = c(5, 3, -7, 1), numbers_as_strings = c("5", "3", "-7", "1"))
```

Look at your data frame and examine it with str

```
test3
```

```
##   numbers_as_numbers numbers_as_strings
## 1                  5                  5
## 2                  3                  3
## 3                 -7                 -7
## 4                  1                  1
```

```
str(test3)
```

```
## 'data.frame':    4 obs. of  2 variables:
##  $ numbers_as_numbers: num  5 3 -7 1
##  $ numbers_as_strings: Factor w/ 4 levels "-7","1","3","5": 4 3 1 2
```

Print the column named “numbers as strings”

```
test3$numbers_as_strings
```

```
## [1] 5  3  -7 1
## Levels: -7 1 3 5
```

Now, convert the column “numbers_as_strings” to an (additional) numeric column using the following command

```
test3$numbers_from_strings <- as.numeric(test3$numbers_as_strings)
```

Print your data frame and examine it with str…

```
test3
```

```
##   numbers_as_numbers numbers_as_strings numbers_from_strings
## 1                  5                  5                    4
## 2                  3                  3                    3
## 3                 -7                 -7                    1
## 4                  1                  1                    2
```

```
str(test3)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ numbers_as_numbers  : num  5 3 -7 1
##  $ numbers_as_strings  : Factor w/ 4 levels "-7","1","3","5": 4 3 1 2
##  $ numbers_from_strings: num  4 3 1 2
```

what happened when you converted the numbers_as_strings column to a numeric column? are the numbers what you expected? * It changes the factors to numbers based on their factor ordering, so -7 becomes 1, 1 becomes 2, 3 becomes 3, and 5 becomes 4.

Try the above “test3” but this time make a tibble instead of a data frame.

```
test3 <- tibble(numbers_as_numbers = c(5, 3, -7, 1), numbers_as_strings = c("5", "3", "-7", "1"))
test3$numbers_from_strings <- as.numeric(test3$numbers_as_strings)
test3
```

```
## # A tibble: 4 x 3
##   numbers_as_numbers numbers_as_strings numbers_from_strings
##                <dbl> <chr>                              <dbl>
## 1                  5 5                                      5
## 2                  3 3                                      3
## 3                 -7 -7                                    -7
## 4                  1 1                                      1
```

```
str(test3)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    4 obs. of  3 variables:
##  $ numbers_as_numbers  : num  5 3 -7 1
##  $ numbers_as_strings  : chr  "5" "3" "-7" "1"
##  $ numbers_from_strings: num  5 3 -7 1
```

Does it work better? * Yes, the numbers match.

Now let's look at simplification versus preservation

first set-up a (generic) data frame and a tibble using the following code

```
tester <- data.frame(numbers = 1:10, fives = rep(5, 10), letters = LETTERS[1:10])
tester2 <- tibble(numbers = 1:10, fives = rep(5, 10), letters = LETTERS[1:10])
```

Print and examine both new objects

```
tester
```

```
##    numbers fives letters
## 1        1     5       A
## 2        2     5       B
## 3        3     5       C
## 4        4     5       D
## 5        5     5       E
## 6        6     5       F
## 7        7     5       G
## 8        8     5       H
## 9        9     5       I
## 10      10     5       J
```

```
str(tester)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ numbers: int  1 2 3 4 5 6 7 8 9 10
##  $ fives  : num  5 5 5 5 5 5 5 5 5 5
##  $ letters: Factor w/ 10 levels "A","B","C","D",..: 1 2 3 4 5 6 7 8 9 10
```

```
tester2
```

```
## # A tibble: 10 x 3
##    numbers fives letters
##      <int> <dbl> <chr>
##  1       1     5 A
##  2       2     5 B
```

```
##  3       3     5 C
##  4       4     5 D
##  5       5     5 E
##  6       6     5 F
##  7       7     5 G
##  8       8     5 H
##  9       9     5 I
## 10      10     5 J
```

```
str(tester2)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  3 variables:
##  $ numbers: int  1 2 3 4 5 6 7 8 9 10
##  $ fives  : num  5 5 5 5 5 5 5 5 5 5
##  $ letters: chr  "A" "B" "C" "D" ...
```

Save the first column as a subset for both dataframes using the following code

```
test_nums <- tester[ , 1]
test_nums2 <- tester2[ , 1]
```

Print and examine both test_nums and test_nums2

```
test_nums
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
str(test_nums)
```

```
##  int [1:10] 1 2 3 4 5 6 7 8 9 10
```

```
test_nums2
```

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

```
str(test_nums2)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  1 variable:
##  $ numbers: int  1 2 3 4 5 6 7 8 9 10
```

Run the following line of code as though you were trying to select the second row from a new data frame

```
#test_nums[2, ]
```

What happens, why? * You get an error “incorrect number of dimensions” * test_nums was simplified to a vector, so no longer has two dimensions as our subset implies

Now try it with test_nums2 instead of test_nums.

```
test_nums2[2, ]
```

```
## # A tibble: 1 x 1
##   numbers
##     <int>
## 1       2
```

Does it work? Why? * Yes it works because tibbles preserve by default, so it is still a tibble and therefore has two dimensions

There are several commands and operators in R that won't work on atomic vectors, try the following line of code

```
#test_nums[test_nums$numbers > 5, ]
```

What happens, why? Try to remember this error message and why it happened, it is an extremely common bug in R! * We get and error saying $ is invalid for atomic vectors * Since test_nums is an atomic vector you cannot subset using $

Now run the same code, but using test_nums2 instead of test_nums. Make sure you add the "2" everywhere you need to!

```
test_nums2[test_nums2$numbers > 5, ]
```

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

Now lets examine row name behavior in data frames and tibbles

Let's check on our friend the iris dataset, which is a (generic) data frame. Look at it and examine it with str

```
# iris
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 ...
```

Now make a tibble named iris2 that is generated by making the iris dataset into a tibble. Print it and examine it with str

```
iris2 <- as_tibble(iris)
iris2
```

```
## # 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
```

```
str(iris2)
```

```
## Classes 'tbl_df', 'tbl' and '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 ...
```

Check the row names on the iris dataset and the new iris2 dataset

```
row.names(iris)
```

```
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"
##  [12] "12"  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"
##  [23] "23"  "24"  "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"
##  [34] "34"  "35"  "36"  "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"
##  [45] "45"  "46"  "47"  "48"  "49"  "50"  "51"  "52"  "53"  "54"  "55"
##  [56] "56"  "57"  "58"  "59"  "60"  "61"  "62"  "63"  "64"  "65"  "66"
##  [67] "67"  "68"  "69"  "70"  "71"  "72"  "73"  "74"  "75"  "76"  "77"
##  [78] "78"  "79"  "80"  "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"
##  [89] "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"  "97"  "98"  "99"
## [100] "100" "101" "102" "103" "104" "105" "106" "107" "108" "109" "110"
## [111] "111" "112" "113" "114" "115" "116" "117" "118" "119" "120" "121"
## [122] "122" "123" "124" "125" "126" "127" "128" "129" "130" "131" "132"
## [133] "133" "134" "135" "136" "137" "138" "139" "140" "141" "142" "143"
## [144] "144" "145" "146" "147" "148" "149" "150"
```

```
row.names(iris2)
```

```
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"
##  [12] "12"  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"
##  [23] "23"  "24"  "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"
##  [34] "34"  "35"  "36"  "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"
##  [45] "45"  "46"  "47"  "48"  "49"  "50"  "51"  "52"  "53"  "54"  "55"
##  [56] "56"  "57"  "58"  "59"  "60"  "61"  "62"  "63"  "64"  "65"  "66"
##  [67] "67"  "68"  "69"  "70"  "71"  "72"  "73"  "74"  "75"  "76"  "77"
##  [78] "78"  "79"  "80"  "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"
##  [89] "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"  "97"  "98"  "99"
## [100] "100" "101" "102" "103" "104" "105" "106" "107" "108" "109" "110"
## [111] "111" "112" "113" "114" "115" "116" "117" "118" "119" "120" "121"
## [122] "122" "123" "124" "125" "126" "127" "128" "129" "130" "131" "132"
## [133] "133" "134" "135" "136" "137" "138" "139" "140" "141" "142" "143"
## [144] "144" "145" "146" "147" "148" "149" "150"
```

Now take a subset of the iris data set named veri_iris that takes all columns, and only the rows where the species is “versicolor”

```
veri_iris <- iris[iris$Species == "versicolor", ]
row.names(veri_iris)
```

```
##  [1] "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"  "61"
## [12] "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"  "71"  "72"
## [23] "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"  "81"  "82"  "83"
## [34] "84"  "85"  "86"  "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"
## [45] "95"  "96"  "97"  "98"  "99"  "100"
```

What are the row names on this new dataset?

- The row names are 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100

Make a subset of the iris2 dataset named veri_iris_2 that takes all the columns, and only the rows where the species is "versicolor"

```
veri_iris_2 <- iris2[iris2$Species == "versicolor", ]
row.names(veri_iris_2)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15" "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28"
## [29] "29" "30" "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42"
## [43] "43" "44" "45" "46" "47" "48" "49" "50"
```

What are the row names on this new dataset? * The row names are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50

Print the mtcars dataset. Print the row names for the mtcars dataset.

```
# mtcars
row.names(mtcars)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
## [22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
## [25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
## [28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
## [31] "Maserati Bora"       "Volvo 142E"
```

Convert the mtcars dataset to a tibble and print it. Then print the row names of the new mtcars tibble.

```
mtcars_tibble <- as_tibble(mtcars)
mtcars_tibble
```

```
## # 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
```

```
row.names(mtcars_tibble)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
## [22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
## [25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
## [28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
## [31] "Maserati Bora"       "Volvo 142E"
```

Take a subset of rows (preserving all columns) of your mtcars tibble (you can use whatever criteria you like, so long as it's a subset).

```
mtcars_small <- mtcars_tibble[mtcars_tibble$wt < 3, ]
row.names(mtcars_small)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
```

Now what happens to the row names? * The row names were converted to numbers

What are two differences you notice (based on the previous exercises) about how tibbles and dataframes deal with row names? * When tibbles are subset the row names are replaced with the new row numbers, whereas data frames maintain the old row names * When printing tibbles row names are not displayed, instead row numbers are. Data frames display row names when printed.

Are there advantages to how tibbles deal with row names? Are there advantages to how (generic) data frames deal with rownames? * Row names are "volatile" and many functions delete them, so it's unwise to rely on them to store important information * If your data have no inherent order it may make more sense to you to think of rows by name rather than number

You can set the row names on a dataframe with the following syntax

```
row.names(test1) <- c("cube1", "cube2", "office")
```

What happens if you try to do this to your tibble, test2?

```
row.names(test2) <- c("cube1", "cube2", "office")
```

```
## Warning: Setting row names on a tibble is deprecated.
```

- If gives a warning message "Setting row names on a tibble is deprecated."

Looking back at the iris (generic) dataframe, iris tibble, mtcars (generic) dataframe, and mtcars tibble. What are the differences in how tibbles and dataframes are displayed? Which do you prefer? Why? * Tibbles give column type, dimensions, only print the first ten rows, and only print the number of columns that fit on the screen * Data frames display the whole data frame and print the row names * I prefer tibbles because they don't take over my whole console! (answer may vary)