Feb 21 Exercises

Matthew D. CIaramitaro

February 21, 2018

Exercises 10.5

10.5.1

We can tell if a dataframe is a tibble by printing it and seeing if it follows the print behavior of a tibble. If it prints more than 10 rows it is not a tibble. However, this method doesn't work for all cases.The function `is.tibble` below would work for all cases.

```
df <- mtcars
tb <- as.tibble(mtcars)
print(is.tibble(df))
```

```
## [1] FALSE
```

```
print(is.tibble(tb))
```

```
## [1] TRUE
```

10.5.2

```
#first with the data.frame
df <- data.frame(abc = 1, xyz = "a")
df$x
```

```
## [1] a
## Levels: a
```

```
df[, "xyz"]
```

```
## [1] a
## Levels: a
```

```
df[, c("abc", "xyz")]
```

```
##   abc xyz
## 1   1   a
```

```
#now with the tibble
df <- as.tibble(data.frame(abc = 1, xyz = "a"))
df$x
```

```
## Warning: Unknown or uninitialised column: 'x'.
```

```
## NULL
```

```
df[, "xyz"]
```

```
## # A tibble: 1 x 1
##   xyz
##   <fct>
## 1 a
```

```
df[, c("abc", "xyz")]
```

```
## # A tibble: 1 x 2
##     abc xyz
##   <dbl> <fct>
## 1  1.00 a
```

The tibble doesn't have access to the same indexing keys as the data frame. We can no longer use the old methods to obtain the data by row/column name. Dataframes can be frustrating because unlike tibbles, they do not inform you that you accessed a column that doesn't exist and do partial matches like `df$x` yielding `df$xyz` in the above.

10.5.3

We can use piping or normal indexing to extract

```
var <- "mpg"
#let's use tb from before
tb[[var]]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

```
tb %>% .[[var]]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

10.5.4

Using the following dataframe we will answer some questions

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

10.5.4.1

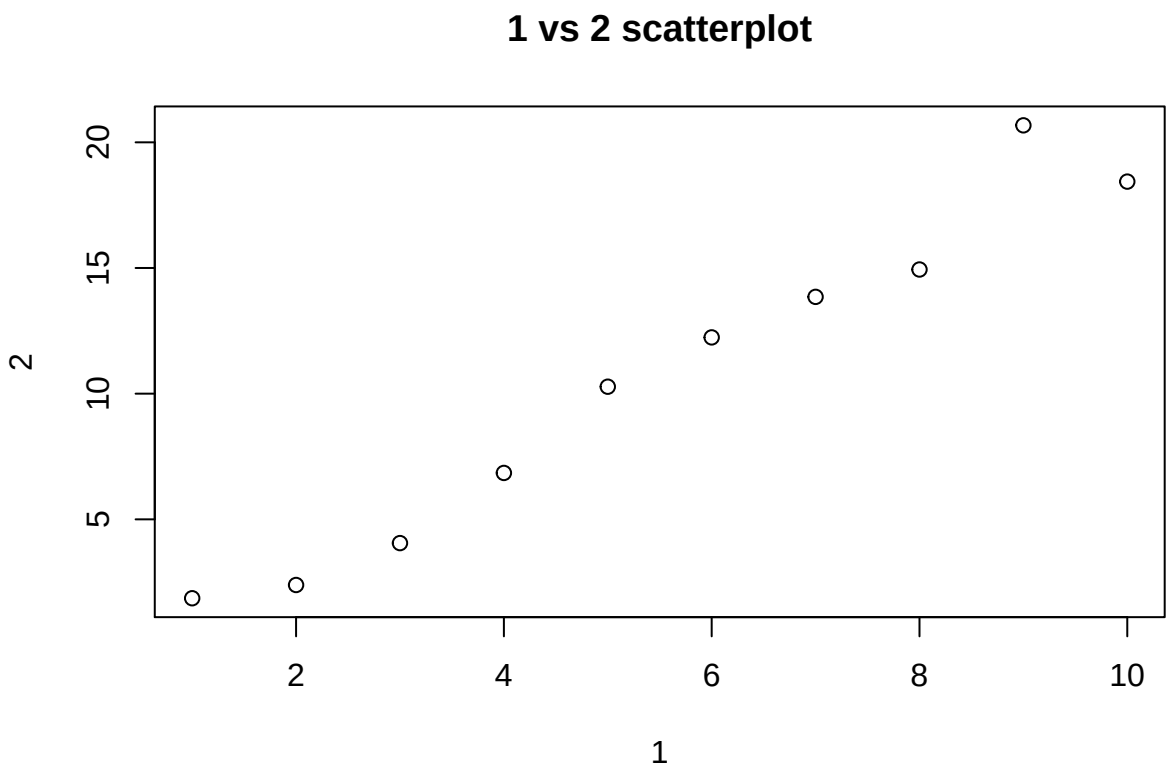
Extract the variable 1

```
annoying %>% .[["1"]]
```

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

10.5.4.2

```
plot(x = annoying %>% .[["1"]], y = annoying %>% .[["2"]], main="1 vs 2 scatterplot", type="p", xlab =
```



10.5.4.3

We use the add column function

```
annoying <- add_column(annoying, "3"= annoying[["2"]]/annoying[["1"]])
annoying
```

```
## # A tibble: 10 x 3
##      `1`   `2`   `3`
##    <int> <dbl> <dbl>
##  1     1  1.86  1.86
##  2     2  2.39  1.19
##  3     3  4.05  1.35
##  4     4  6.85  1.71
##  5     5 10.3   2.06
##  6     6 12.2   2.04
##  7     7 13.9   1.98
##  8     8 14.9   1.87
##  9     9 20.7   2.30
## 10    10 18.4   1.84
```

10.5.4.4

```
annoying <- rename(annoying, "one" = "1", "two" = "2", "three" = "3")
annoying
```

```
## # A tibble: 10 x 3
##      one   two three
##    <int> <dbl> <dbl>
##  1     1  1.86  1.86
##  2     2  2.39  1.19
##  3     3  4.05  1.35
```

```
##  4     4  6.85  1.71
##  5     5 10.3   2.06
##  6     6 12.2   2.04
##  7     7 13.9   1.98
##  8     8 14.9   1.87
##  9     9 20.7   2.30
## 10    10 18.4   1.84
```

10.5.5

This tidies the data, giving it two columns, one for variable and one for data.

10.5.6

`options(tibble.width = n)` will make the number of columns printed n. If n is `Inf` all columns will be printed.

##Exercises 12.6.1