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

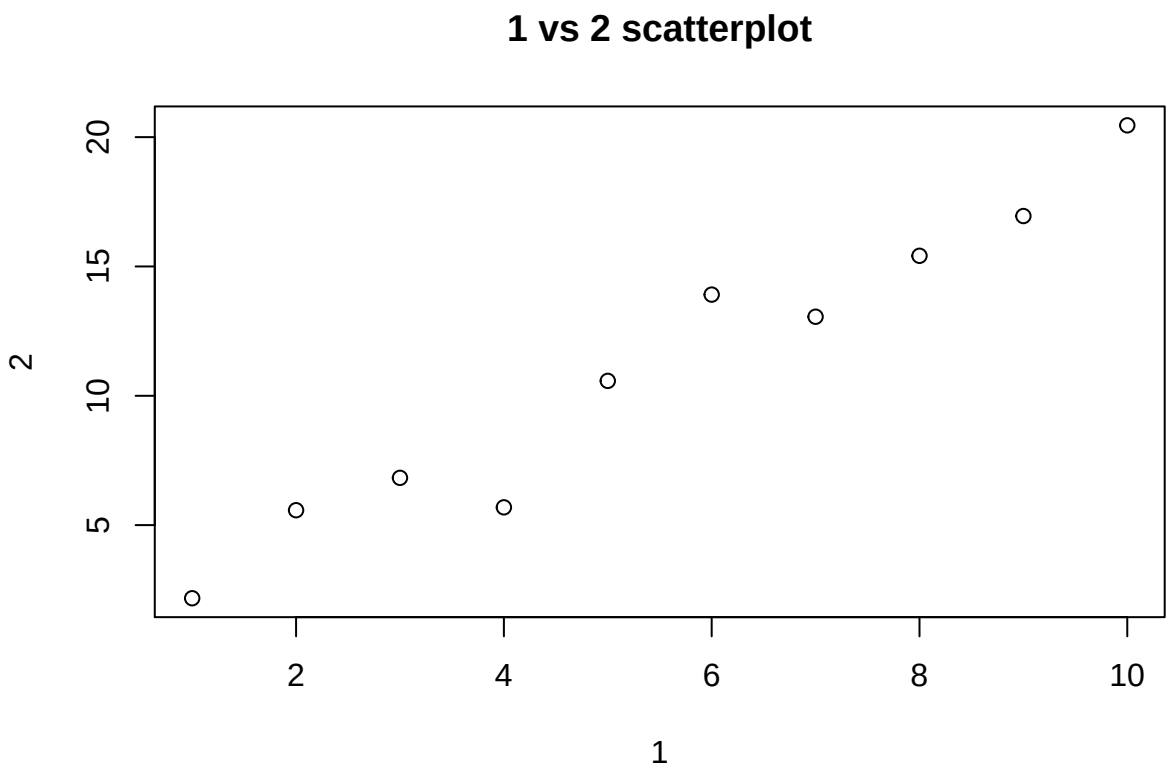
Extract the variable 1

```
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  2.17  2.17
##  2     2  5.58  2.79
##  3     3  6.83  2.28
##  4     4  5.69  1.42
##  5     5 10.6   2.12
##  6     6 13.9   2.32
##  7     7 13.1   1.87
##  8     8 15.4   1.93
##  9     9 16.9   1.88
## 10    10 20.5   2.05
```

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  2.17  2.17
##  2     2  5.58  2.79
##  3     3  6.83  2.28
```

```
##  4     4  5.69  1.42
##  5     5 10.6   2.12
##  6     6 13.9   2.32
##  7     7 13.1   1.87
##  8     8 15.4   1.93
##  9     9 16.9   1.88
## 10    10 20.5   2.05
```

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

12.6.1.1

`na.rm = T` is valid in this case because we have many missing values, and it wouldn't make sense in our data to attempt to count such values. They are explicitly missing, so we run this line to make it implicit. NA is different from 0 because NA denotes missing values while 0 denotes that we have observed 0 occurences with data.

12.6.1.2

`mutate(key = stringr::str_replace(key, "newrel", "new_rel"))` This line says to replace all occurences of "newrel" in the table keys with "new_rel". The effect of this is that the data becomes consistent. the two names refer to the exact same variables, so including both will inhibit data analysis later on and violate the rules of tidiness. We should not neglect this step.

12.6.1.3

country, iso2, and iso3 occurrences are deterministic. Every occurence of a given country is paired with the same occurences in iso2 and iso3. Because of this, the three variables are redundant, and we only need to select one. We then choose country.

```
who1 <- who %>% gather(new_sp_m014:newrel_f65, key = "key", value = "cases", na.rm = TRUE)
nrow(distinct(who1, country, iso2, iso3)) == nrow(distinct(who1,country))
```

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

Because the number of distinct rows using the three variables country, iso2, and iso3 is equivalent to the number of distinct rows using just country, there is a unique occurence of iso2 and iso3 for each country, and so the variables are redundant.

12.6.1.4

```
who1 <- who %>%
  gather(code, value, new_sp_m014:newrel_f65, na.rm = TRUE) %>%
  mutate(code = stringr::str_replace(code, "newrel", "new_rel")) %>%
  separate(code, c("new", "var", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)
#select relevant variables
who1 <- who1 %>% select(country, sex, year, value)
countrysum <- who1 %>% select(country, value) %>% group_by(country) %>% summarise(value = sum(value))
yearsum <- who1 %>% select(year, value) %>% group_by(year) %>% summarise(value = sum(value))
sexsum <-  who1 %>% select(sex, value) %>% group_by(sex) %>% summarise(value = sum(value))
countrysum
```

```
## # A tibble: 219 x 2
##    country              value
##    <chr>                <int>
##  1 Afghanistan         140225
##  2 Albania               5335
##  3 Algeria             128119
##  4 American Samoa          41
##  5 Andorra                103
##  6 Angola              308365
##  7 Anguilla                 2
##  8 Antigua and Barbuda     55
##  9 Argentina           117156
## 10 Armenia              15991
## # ... with 209 more rows
```

```
yearsum
```

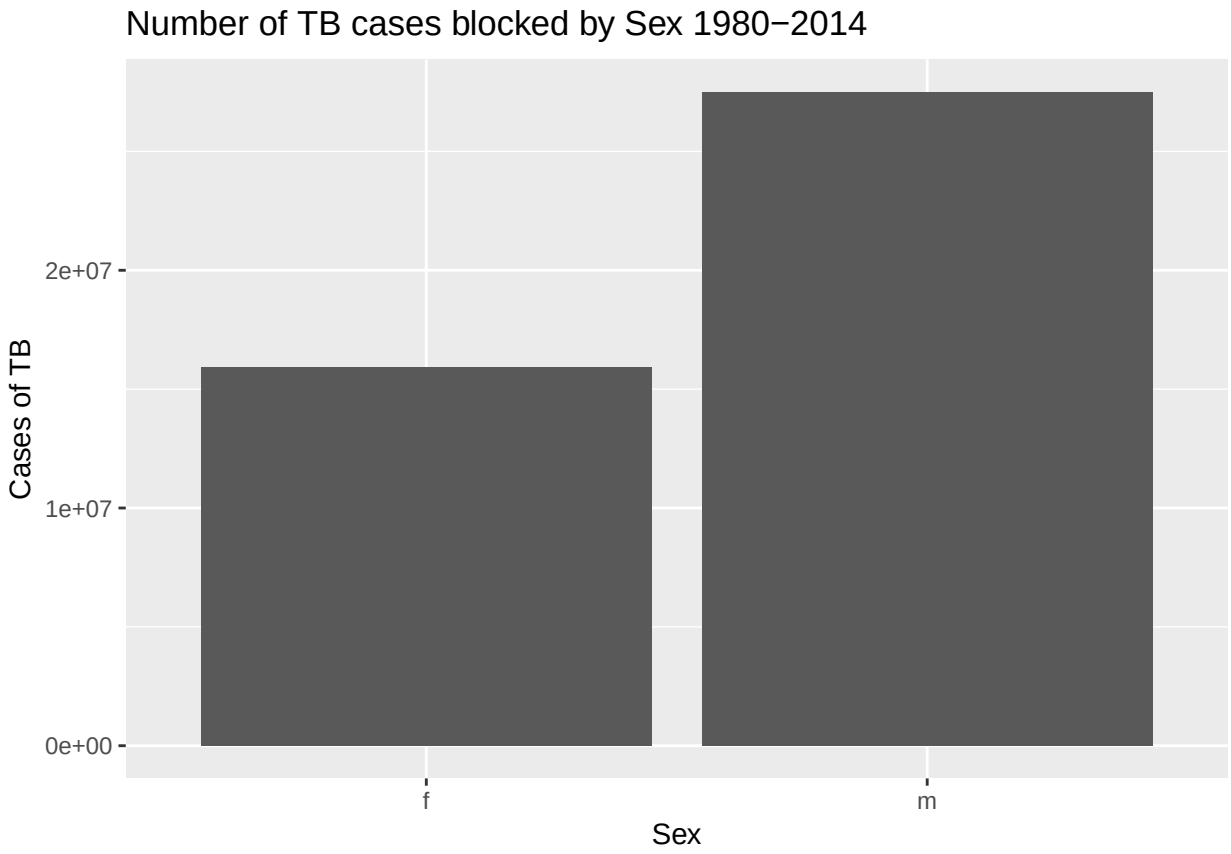
```
## # A tibble: 34 x 2
##     year value
##    <int> <int>
##  1  1980   959
##  2  1981   805
##  3  1982   824
##  4  1983   786
##  5  1984   814
##  6  1985   799
##  7  1986   754
##  8  1987   670
##  9  1988   682
## 10  1989   654
## # ... with 24 more rows
```

```
sexsum
```

```
## # A tibble: 2 x 2
##   sex      value
##   <chr>    <int>
## 1 f     15907024
## 2 m     27490494
```

```
#Histogram of Number of TB Cases by Sex
g <- ggplot(sexsum, aes(x=sex, weight=sexsum["value"])) + xlab("Sex") + ylab("Cases of TB")
g + geom_bar() + ggtitle("Number of TB cases blocked by Sex 1980-2014")
```

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```

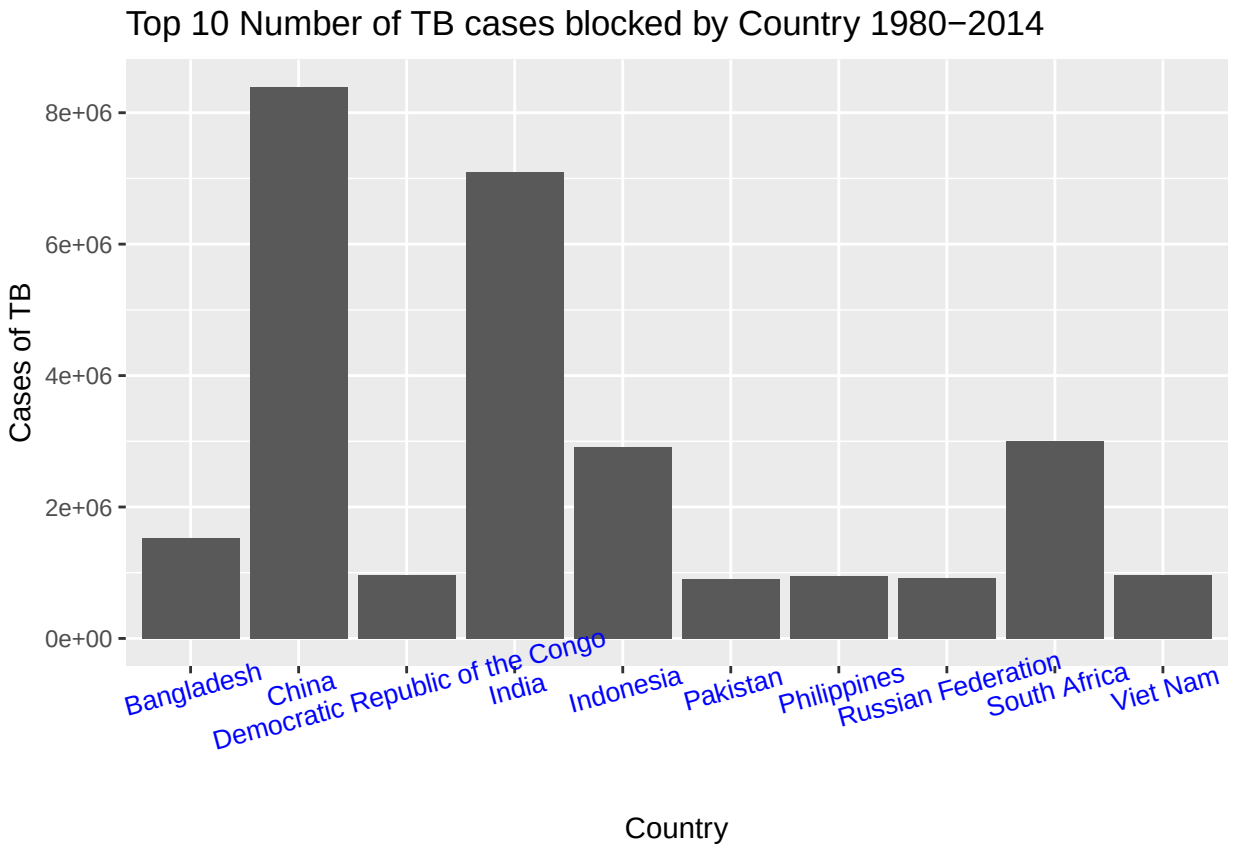



```
#Histogram of Top 10 Number of TB Cases by Country
sorted_country  <-  top_n(arrange(countrysum), 10)
```

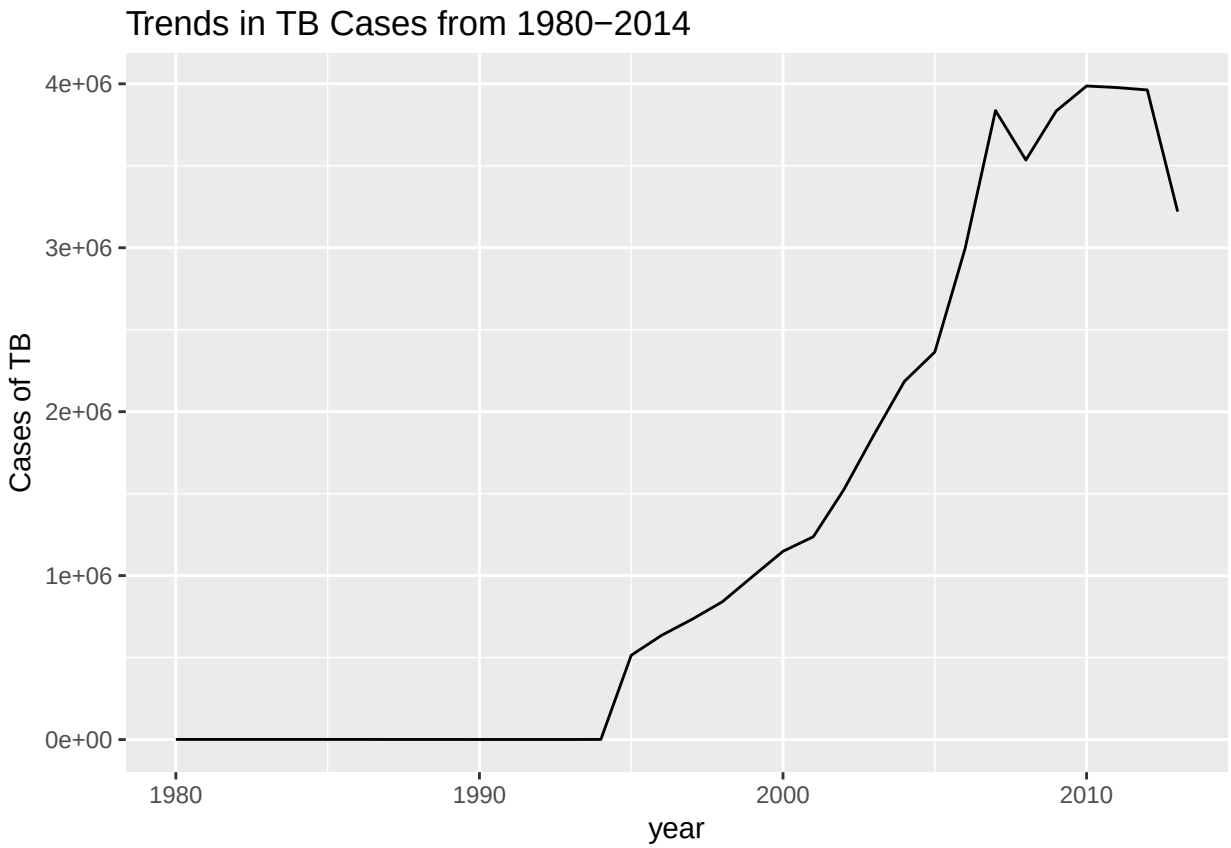
```
## Selecting by value
```

```
g <- ggplot(sorted_country, aes(x=country, weight=sorted_country["value"])) + xlab("Country") + ylab("Ca
g + geom_bar() + ggtitle("Top 10 Number of TB cases blocked by Country 1980-2014") + theme(axis.text.x =
```

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```




```
#Line graph of Number of TB Cases from 2010 to 2014
g <- ggplot(yearsum, aes(x=year, y=value)) + xlab("year") + ylab("Cases of TB")
g + geom_line() + ggtitle("Trends in TB Cases from 1980-2014")
```


Trends in TB Cases from 1980−2014
Cases of TB
4e+06
3e+06
2e+06
1e+06
0e+00
1980
1990
2000
2010
year