Assignment 4

Maria Ren

Feburary 20,2018

```
library('tibble')
library("tidyverse")
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.2.1 ──
```

```
## ✔ ggplot2 2.2.1     ✔ purrr   0.2.4
## ✔ tidyr   0.7.2     ✔ dplyr   0.7.4
## ✔ readr   1.1.1     ✔ stringr 1.2.0
## ✔ ggplot2 2.2.1     ✔ forcats 0.2.0
```

```
## ── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library("dplyr")
```

1.

How can you tell if an object is a tibble?

```
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(as_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  21.0  6.00   160 110    3.90  2.62  16.5  0     1.00  4.00  4.00
##  2  21.0  6.00   160 110    3.90  2.88  17.0  0     1.00  4.00  4.00
##  3  22.8  4.00   108  93.0  3.85  2.32  18.6  1.00  1.00  4.00  1.00
##  4  21.4  6.00   258 110    3.08  3.22  19.4  1.00  0     3.00  1.00
##  5  18.7  8.00   360 175    3.15  3.44  17.0  0     0     3.00  2.00
##  6  18.1  6.00   225 105    2.76  3.46  20.2  1.00  0     3.00  1.00
##  7  14.3  8.00   360 245    3.21  3.57  15.8  0     0     3.00  4.00
##  8  24.4  4.00   147  62.0  3.69  3.19  20.0  1.00  0     4.00  2.00
##  9  22.8  4.00   141  95.0  3.92  3.15  22.9  1.00  0     4.00  2.00
## 10  19.2  6.00   168 123    3.92  3.44  18.3  1.00  0     4.00  4.00
## # ... with 22 more rows
```

 # A tibble prints only the first 10 rows of the data set, while the actual dataframe prints the ent ire dataset.

2.

Compare and contrast the following operations on dataframe, and equivalent tibble. What is different? why might the default data frame behaviors cause you frustration?

```
# On a data frame - given by the problem
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
```

```
class(df[, "xyz"])
```

```
## [1] "factor"
```

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

```
## [1] "data.frame"
```

```
# On a tibble
df <- tibble(abc = 1, xyz = "a")
df$x
```

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

```
## NULL
```

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

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

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

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

```
class(df[, "xyz"])
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

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

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
# when we type in df$x for data frame, it returns the result of Levels: a, the same result as for d
f[, "xyz"]. This could result in using the wrong variable. where for tibbles, it gives NULL, and a
 warning message. All the other commands are similar between tibbles and data frame. The default da
ta frame could return levels of factors, where subsetting tibble only returns tibbles (like shown a
bove).
```

3.

If you have the name of a variable stored in an object, e.g. var " mpg ", how can you extract the reference variable from a tibble?

```
# Use the double [[
# var <- "mpg"
# data[[var]]
```

4.

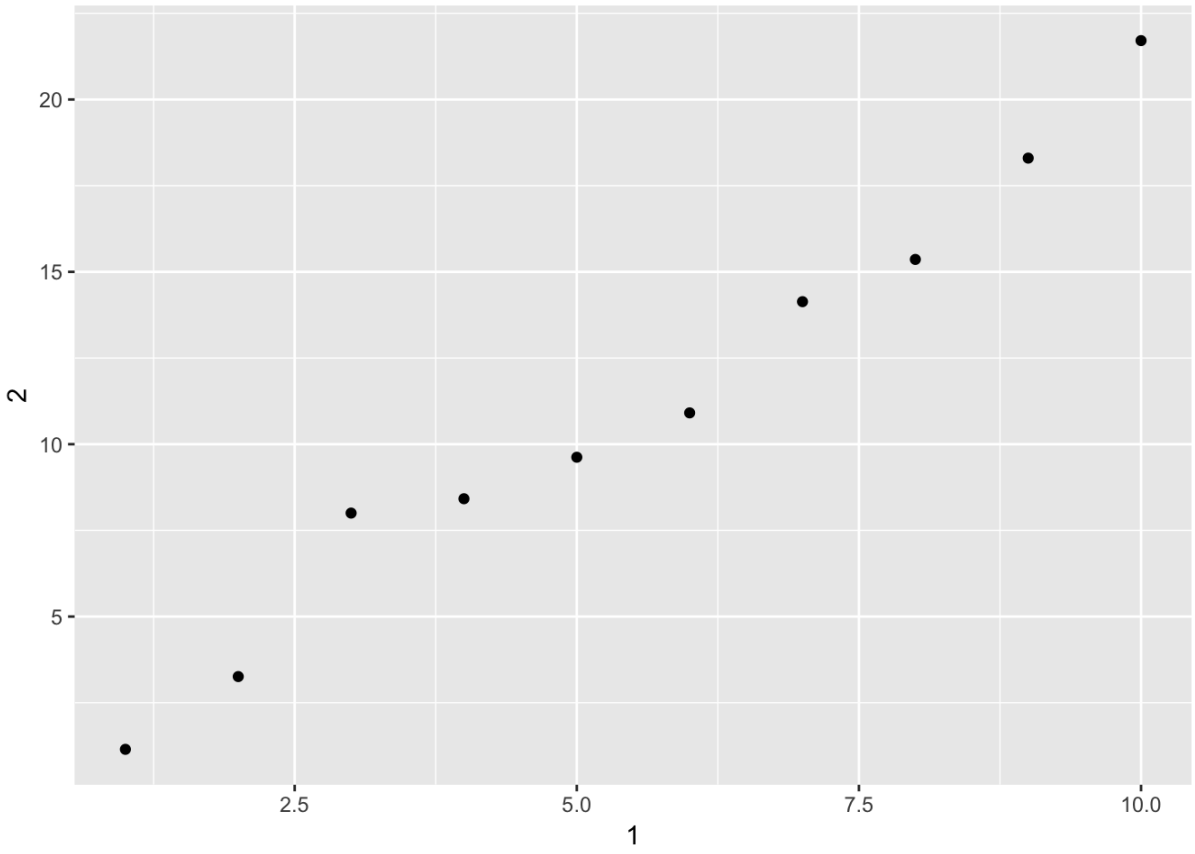
Practice referring to nonsyntactic names in the following data frame by:

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

# 1) Extracting the variable called 1.
annoying$`1`
```

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

```
# 2) Plotting a scatterplot of 1 vs 2.
ggplot(annoying, aes(`1`,`2`)) +
       geom_point()
```




```
# 3) Creating a new column called 3 which is 2 divided by 1.
annoying[["3"]] <- annoying$`2`/annoying$`1`
annoying
```

```
## # A tibble: 10 x 3
##       `1`   `2`   `3`
##    <int> <dbl> <dbl>
##  1     1  1.15  1.15
##  2     2  3.26  1.63
##  3     3  8.00  2.67
##  4     4  8.42  2.10
##  5     5  9.62  1.92
##  6     6 10.9   1.82
##  7     7 14.1   2.02
##  8     8 15.4   1.92
##  9     9 18.3   2.03
## 10    10 21.7   2.17
```

```
# 4) Renaming the columns to one, two and three.
annoying <- rename(annoying, one=`1`,two=`2`,three=`3`)
annoying
```

```
## # A tibble: 10 x 3
##      one   two three
##    <int> <dbl> <dbl>
##  1     1  1.15  1.15
##  2     2  3.26  1.63
##  3     3  8.00  2.67
##  4     4  8.42  2.10
##  5     5  9.62  1.92
##  6     6 10.9   1.82
##  7     7 14.1   2.02
##  8     8 15.4   1.92
##  9     9 18.3   2.03
## 10    10 21.7   2.17
```

```
# 5. What does tibble::enframe() do? When might you use it?
x <- c(a=2,b=9)
tibble::enframe(x)
```

```
## # A tibble: 2 x 2
##   name  value
##   <chr> <dbl>
## 1 a      2.00
## 2 b      9.00
```

```
?enframe()

# tibble::enframe() turns a vector with names into a tibble with two columns,as shown from above. Y
ou can use it when you have a named data vector,and you want to add that to another data frame.
```

5.

What option controls how many additional column names are printed at the footer of a tibble?

```
# In the function print.tbl_df, the n_extra argument determines the number of extra columns to prin
t abbreviated information for. The argument tibble.max_extra_cols determines the most extra column
s.
```

12.6.1

Exercises

```
# Original Data and Code
tidyr::who
```

```
## # A tibble: 7,240 x 60
##    country     iso2  iso3   year new_sp_m014 new_sp_m1524 new_sp_m2534
##    <chr>       <chr> <chr> <int>       <int>        <int>        <int>
##  1 Afghanistan AF    AFG    1980          NA           NA           NA
##  2 Afghanistan AF    AFG    1981          NA           NA           NA
##  3 Afghanistan AF    AFG    1982          NA           NA           NA
##  4 Afghanistan AF    AFG    1983          NA           NA           NA
##  5 Afghanistan AF    AFG    1984          NA           NA           NA
##  6 Afghanistan AF    AFG    1985          NA           NA           NA
##  7 Afghanistan AF    AFG    1986          NA           NA           NA
##  8 Afghanistan AF    AFG    1987          NA           NA           NA
##  9 Afghanistan AF    AFG    1988          NA           NA           NA
## 10 Afghanistan AF    AFG    1989          NA           NA           NA
## # ... with 7,230 more rows, and 53 more variables: new_sp_m3544 <int>,
## #   new_sp_m4554 <int>, new_sp_m5564 <int>, new_sp_m65 <int>,
## #   new_sp_f014 <int>, new_sp_f1524 <int>, new_sp_f2534 <int>,
## #   new_sp_f3544 <int>, new_sp_f4554 <int>, new_sp_f5564 <int>,
## #   new_sp_f65 <int>, new_sn_m014 <int>, new_sn_m1524 <int>,
## #   new_sn_m2534 <int>, new_sn_m3544 <int>, new_sn_m4554 <int>,
## #   new_sn_m5564 <int>, new_sn_m65 <int>, new_sn_f014 <int>,
## #   new_sn_f1524 <int>, new_sn_f2534 <int>, new_sn_f3544 <int>,
## #   new_sn_f4554 <int>, new_sn_f5564 <int>, new_sn_f65 <int>,
## #   new_ep_m014 <int>, new_ep_m1524 <int>, new_ep_m2534 <int>,
## #   new_ep_m3544 <int>, new_ep_m4554 <int>, new_ep_m5564 <int>,
## #   new_ep_m65 <int>, new_ep_f014 <int>, new_ep_f1524 <int>,
## #   new_ep_f2534 <int>, new_ep_f3544 <int>, new_ep_f4554 <int>,
## #   new_ep_f5564 <int>, new_ep_f65 <int>, newrel_m014 <int>,
## #   newrel_m1524 <int>, newrel_m2534 <int>, newrel_m3544 <int>,
## #   newrel_m4554 <int>, newrel_m5564 <int>, newrel_m65 <int>,
## #   newrel_f014 <int>, newrel_f1524 <int>, newrel_f2534 <int>,
## #   newrel_f3544 <int>, newrel_f4554 <int>, newrel_f5564 <int>,
## #   newrel_f65 <int>
```

```
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)
```

```
## # A tibble: 76,046 x 6
##    country      year var   sex   age   value
##  * <chr>       <int> <chr> <chr> <chr> <int>
##  1 Afghanistan  1997 sp    m     014       0
##  2 Afghanistan  1998 sp    m     014      30
##  3 Afghanistan  1999 sp    m     014       8
##  4 Afghanistan  2000 sp    m     014      52
##  5 Afghanistan  2001 sp    m     014     129
##  6 Afghanistan  2002 sp    m     014      90
##  7 Afghanistan  2003 sp    m     014     127
##  8 Afghanistan  2004 sp    m     014     139
##  9 Afghanistan  2005 sp    m     014     151
## 10 Afghanistan  2006 sp    m     014     193
## # ... with 76,036 more rows
```

1.

In this case study I set $na.rm = TRUE$ just to make it easier to check that we had the correct values. Is this reasonable? Think about how missing values are represented in this dataset. Are there implicit missing values? What's the difference between an NA and zero?

```
who1 <- who %>%
  gather(new_sp_m014:newrel_f65, key = "key", value = "cases", na.rm = TRUE)
nrow(who1 %>% filter(cases==0))
```

```
## [1] 11080
```

```
nrow(who1 %>% filter(cases==NA))
```

```
## [1] 0
```

```
# It is reasonable to treat missing values the same .Who1 combines all the columns from new-sp_m014
 to newrel-f65. There are 11080 cases where the count of cases equals 0, which indicates no cases n
a.rm would not cause any lost of information.
```

2.

What happens if you neglect the $mutate()$ step?

$$(mutate(key = stringr :: str_replace(key, " newrel ", " new_rel ")))$$

```
who3 <- who1 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
```

```
## Warning: Too few values at 2580 locations: 73467, 73468, 73469, 73470,
## 73471, 73472, 73473, 73474, 73475, 73476, 73477, 73478, 73479, 73480,
## 73481, 73482, 73483, 73484, 73485, 73486, ...
```

```
who3
```

```
## # A tibble: 76,046 x 8
##    country     iso2  iso3   year new   type  sexage cases
##  * <chr>       <chr> <chr> <int> <chr> <chr> <chr>  <int>
##  1 Afghanistan AF    AFG    1997 new   sp    m014       0
##  2 Afghanistan AF    AFG    1998 new   sp    m014      30
##  3 Afghanistan AF    AFG    1999 new   sp    m014       8
##  4 Afghanistan AF    AFG    2000 new   sp    m014      52
##  5 Afghanistan AF    AFG    2001 new   sp    m014     129
##  6 Afghanistan AF    AFG    2002 new   sp    m014      90
##  7 Afghanistan AF    AFG    2003 new   sp    m014     127
##  8 Afghanistan AF    AFG    2004 new   sp    m014     139
##  9 Afghanistan AF    AFG    2005 new   sp    m014     151
## 10 Afghanistan AF    AFG    2006 new   sp    m014     193
## # ... with 76,036 more rows
```

```
# As shown above, if we skip the mutate code, it gives us a warning message Warning message:Too few
 values at 2580 locations: 73467, 73468, 73469, 73470,73471, 73472, 73473, 73474, 73475, 73476, 734
77, 73478, 73479, 73480, 73481,73482, 73483, 73484, 73485, 73486, ... The mutate step changes the n
ew_rel and newrel strings so that they would be consistent. But if we skip the mutate step, we woul
d be missing values once we separate the column "new" from type and sexage.
```

3.

I claimed that $iso2$ and $iso3$ were redundant with country. Confirm this claim.

```
select(who3, country, iso2, iso3) %>%
  distinct() %>%
# select unique rows from the data group of who3, country, iso2 and iso3
  group_by(country) %>%
  filter(n() > 1)
```

```
## # A tibble: 0 x 3
## # Groups:   country [0]
## # ... with 3 variables: country <chr>, iso2 <chr>, iso3 <chr>
```

```
# When we group together the three columns country, iso2 and iso3, and try to find
# unique rows from the data, we found that none of the values in the columns
# have different values from each other, therefore these three columns are redundant.
```

4.

For each country, year, and sex compute the total number of cases of TB. Make an informative visualisation of the data.

```
who1 <- who %>%
  gather(new_sp_m014:newrel_f65, key = "key", value = "cases", na.rm = TRUE)
who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
who3 <- who2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
who4 <- who3 %>%
  select(-new, -iso2, -iso3)
who5 <- who4 %>%
  separate(sexage, c("sex", "age"), sep = 1)

who5 %>%
  group_by(country,sex,age) %>%
  summarize(total_cases=sum(cases))
```

```
## # A tibble: 3,065 x 4
## # Groups:   country, sex [?]
##    country     sex   age   total_cases
##    <chr>       <chr> <chr>       <int>
##  1 Afghanistan f     014          8211
##  2 Afghanistan f     1524        22206
##  3 Afghanistan f     2534        24250
##  4 Afghanistan f     3544        16265
##  5 Afghanistan f     4554        10978
##  6 Afghanistan f     5564         7439
##  7 Afghanistan f     65           4005
##  8 Afghanistan m     014          5953
##  9 Afghanistan m     1524         9583
## 10 Afghanistan m     2534         8390
## # ... with 3,055 more rows
```

```
# The total number of cases for TB is 43397518


who5 %>%
  group_by(country,sex,age) %>%
  summarize(total_cases=sum(cases))%>%
  unite(df,"country","sex") %>%
  ggplot(aes(x = "year", y = "cases", group = df)) +
  geom_line()
```