MA415 Assignment 4

Kevin Chan

February 21 2018

10.5 Exercises

1.

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

```r
# Intial setup.
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.4.3

## -- Attaching packages ---------------------------------- tidyverse 1.2.1 --

## v ggplot2 2.2.1     v purrr   0.2.4
## v tibble  1.4.2     v dplyr   0.7.4
## v tidyr   0.7.2     v stringr 1.2.0
## v readr   1.1.1     v forcats 0.2.0

## Warning: package 'ggplot2' was built under R version 3.4.3

## Warning: package 'tibble' was built under R version 3.4.3

## Warning: package 'tidyr' was built under R version 3.4.3

## Warning: package 'readr' was built under R version 3.4.2

## Warning: package 'purrr' was built under R version 3.4.3

## Warning: package 'dplyr' was built under R version 3.4.3

## Warning: package 'stringr' was built under R version 3.4.3

## Warning: package 'forcats' was built under R version 3.4.3

## -- Conflicts ------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 3.4.3
```

```r
# Exercise 1.
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
```

```r
class(mtcars)
```

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

```r
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
```

```r
class(as_tibble(mtcars))
```

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

2.

Compare and contrast the following operations on a `data.frame` and equivalent tibble. What is different? Why might the default data frame behaviours cause your frustration? Using "$" with a data frame will partially complete the column, which may result in accidentally using a different variable than the one desired. Using [, the type of object returned is dependent on the number of columns. If there is only one column, the object will return a vector and not a data frame. Otherwise, the object will return a data frame.

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

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

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

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

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

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

```r
tibdf <- as_tibble(df)
tibdf$x
```

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

```
## NULL
```

```r
tibdf[, "xyz"]
```

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

```r
tibdf[, c("abc", "xyz")]
```

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

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?

To extract the reference variable from a tibble, you can use the double bracket, e.g. `df[[var]]`.

4.

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

1.

```
  Extracting the variable called \texttt{1}.
```

```r
# This is given.
annoying <- tibble(
  `1` = 1:10,
```

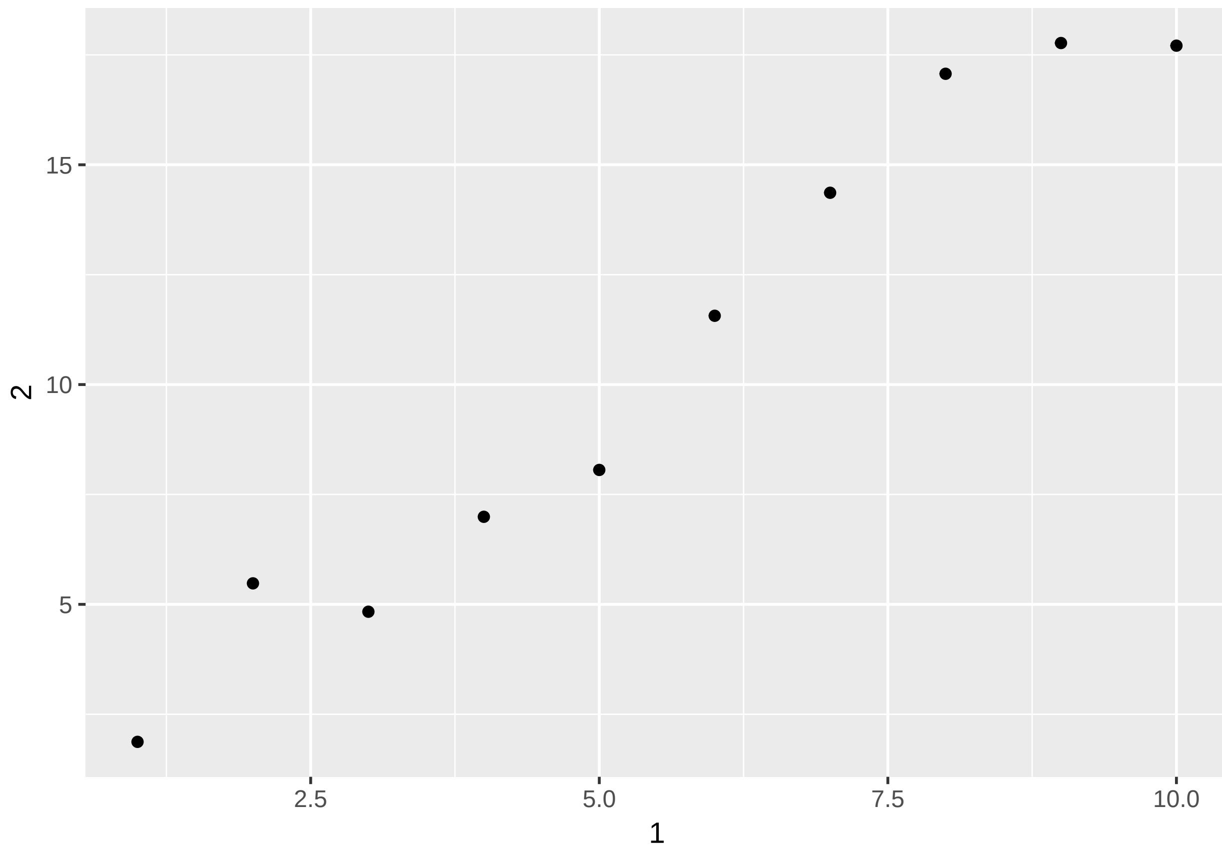
```
  `2` = `1` * 2 + rnorm(length(`1`))
)
# Part 1 of Problem 4
annoying[["1"]]
```

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

2.

  Plotting a scatterplot of \texttt{1} vs \texttt{2}.

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



3.

  Creating a new column called \texttt{3}, which is \texttt{2} divided by \texttt{1}.

```
annoying[["3"]] <- annoying[["2"]]/annoying[["1"]]
```

4.

  Renaming the columns to \texttt{one}, \texttt{two}, and \texttt{three}.

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

```
## Observations: 10
## Variables: 3
## $ one   <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
## $ two   <dbl> 1.871761, 5.477952, 4.832513, 6.991914, 8.057822, 11.564...
## $ three <dbl> 1.871761, 2.738976, 1.610838, 1.747978, 1.611564, 1.9274...
```

5.

What does `tibble::enframe()` do? When might you use it?

```
?tibble::enframe()
```

```
## starting httpd help server ... done
```

`tibble::enframe()` converts named vectors to two-column data frames with names and values. The natural sequence is used as name column for unnamed vectors. ## 6. What option controls how many additional column names are printed at the footer of a tibble? Using the print function for tibbles, which is print.tbl_df, the option n_extra controls how many extra column names are printed.

12.6.1 Exercises

```
# Code necessary for the exercises below (from throughout the chapter):
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)
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.3
```

```
## # 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
```

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

```
## Observations: 76,046
## Variables: 6
## $ country <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanis...
## $ iso2    <chr> "AF", "AF", "AF", "AF", "AF", "AF", "AF", "AF", "AF", ...
## $ iso3    <chr> "AFG", "AFG", "AFG", "AFG", "AFG", "AFG", "AFG", "AFG"...
```

```
## $ year    <int> 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, ...
## $ key     <chr> "new_sp_m014", "new_sp_m014", "new_sp_m014", "new_sp_m...
## $ cases   <int> 0, 30, 8, 52, 129, 90, 127, 139, 151, 193, 186, 187, 2...
```

```
who2 <- who1 %>%
 mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
who3 <- who2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
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
```

```
who3 %>%
  count(new)
```

```
## # A tibble: 1 x 2
##   new       n
##   <chr> <int>
## 1 new   76046
```

```
who4 <- who3 %>%
  select(-new, -iso2, -iso3)
who5 <- who4 %>%
  separate(sexage, c("sex", "age"), sep = 1)
who5
```

```
## # A tibble: 76,046 x 6
##    country      year type  sex   age   cases
##  * <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? More information must be known in order to know more about the data generating process. There are zeros in the data, which may indicate no cases.

```
who1 %>%
  filter(cases == 0) %>%
  nrow()
```

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

It appears that either a country has all of its values as non-missing if the World Health Organization collected data for that country or all of its values are truly non-missing. Therefore, it is okay to treat explicitly and implicitly missing values equally, and we do not lose any information by dropping the missing values.

```
gather(who, new_sp_m014:newrel_f65, key = "key", value = "cases") %>%
  group_by(country, year)  %>%
  mutate(missing = is.na(cases)) %>%
  select(country, year, missing) %>%
  distinct() %>%
  group_by(country, year) %>%
  filter(n() > 1)
```

```
## # A tibble: 6,968 x 3
## # Groups:   country, year [3,484]
##    country      year missing
##    <chr>       <int> <lgl>
##  1 Afghanistan  1997 F
##  2 Afghanistan  1998 F
##  3 Afghanistan  1999 F
##  4 Afghanistan  2000 F
##  5 Afghanistan  2001 F
##  6 Afghanistan  2002 F
##  7 Afghanistan  2003 F
##  8 Afghanistan  2004 F
##  9 Afghanistan  2005 F
## 10 Afghanistan  2006 F
## # ... with 6,958 more rows
```

2.

What happens if you neglect the `mutate()` step? (mutate(key = stringr::str_replace(key, "newrel", "new_rel"))) If you neglect the mutate() step, separate will give the warning message "too few values". If we check the rows for keys beginning with "newrel_", sexage is missing and type is equal to m014.

```
who3nomut <- 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, ...
```

```
filter(who3nomut, new == "newrel") %>% head()
```

```
## # A tibble: 6 x 8
```

```
##   country     iso2  iso3   year new    type  sexage cases
##   <chr>       <chr> <chr> <int> <chr>  <chr> <chr>  <int>
## 1 Afghanistan AF    AFG    2013 newrel m014  <NA>    1705
## 2 Albania     AL    ALB    2013 newrel m014  <NA>      14
## 3 Algeria     DZ    DZA    2013 newrel m014  <NA>      25
## 4 Andorra     AD    AND    2013 newrel m014  <NA>       0
## 5 Angola      AO    AGO    2013 newrel m014  <NA>     486
## 6 Anguilla    AI    AIA    2013 newrel m014  <NA>       0
```

3.

I claimed that `iso2` and `iso3` were redundant with country. Confirm this claim. Based on the output, `iso2` and `iso3` were redundant with country.

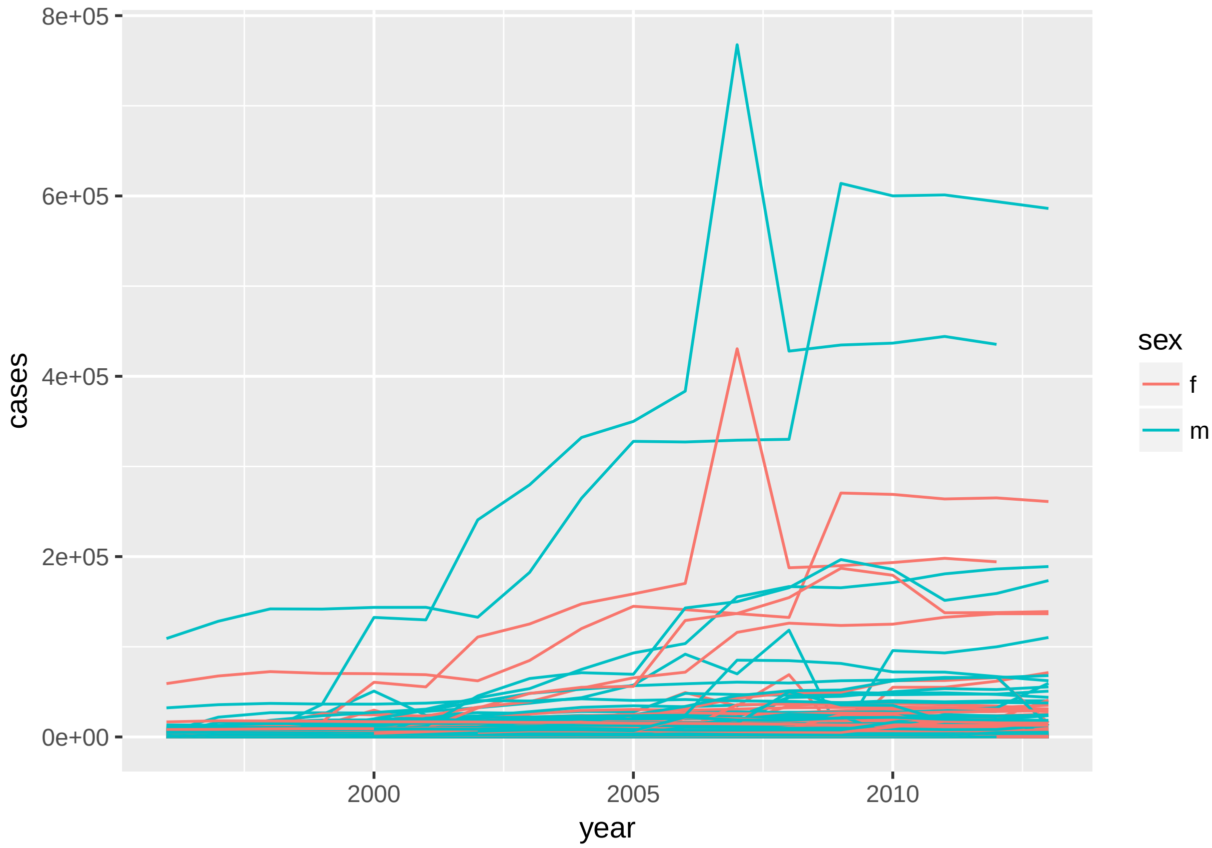
```r
select(who3, country, iso2, iso3) %>%
  distinct() %>%
  group_by(country) %>%
  filter(n() > 1)
```

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

4.

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

```r
who5 %>%
  group_by(country, year, sex) %>%
  filter(year > 1995) %>%
  summarize(cases = sum(cases)) %>%
  unite(country_sex, country, sex, remove = FALSE) %>%
  ggplot(aes(x = year, y = cases, group = country_sex, colour = sex)) + geom_line()
```


8e+05
6e+05
4e+05
2e+05
0e+00
cases
2000
2005
2010
year
sex
f
m