Assignment4_10.5&12.6

10.5 Exercise

1 How can you tell if an object is a tibble?

(Hint: try printing mtcars, which is a regular data frame).

```
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
```

```
class(mtcars)
```

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

```
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 you frustration?

df <- data.frame(abc = 1, xyz = "a") df$x df[, "xyz"] df[, c("abc", "xyz")]

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

#tibble
tbl <- as.tibble(df)

df$x
```

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

```
tbl$x
```

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

```
## NULL
```

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

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

```
tbl[,"xyz"]
```

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

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

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

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

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

- Difference 1 Using $ a data.frame will partially complete the column. So even though we wrote df$xit returned df xyz. The advantage is it sometimes can save a few keystrokes, but on the otherside, it can result in accidentally using a different variable than you thought you were using.
- Difference 2 With data.frames, with [the type of object that is returned differs on the number of columns. If it is one column, it won't return a data.frame, but instead will return a vector. With more than one column, then it will return a data.frame.

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?

df[[var]]

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

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

annoying
```

```
## # A tibble: 10 x 2
##      `1`   `2`
##    <int> <dbl>
##  1     1  2.45
##  2     2  3.05
##  3     3  6.69
##  4     4  7.81
##  5     5 11.1
##  6     6 12.4
##  7     7 14.2
##  8     8 16.4
##  9     9 18.6
## 10    10 18.7
```

```
#(1)Extracting the variable called 1.
annoying[1]
```

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

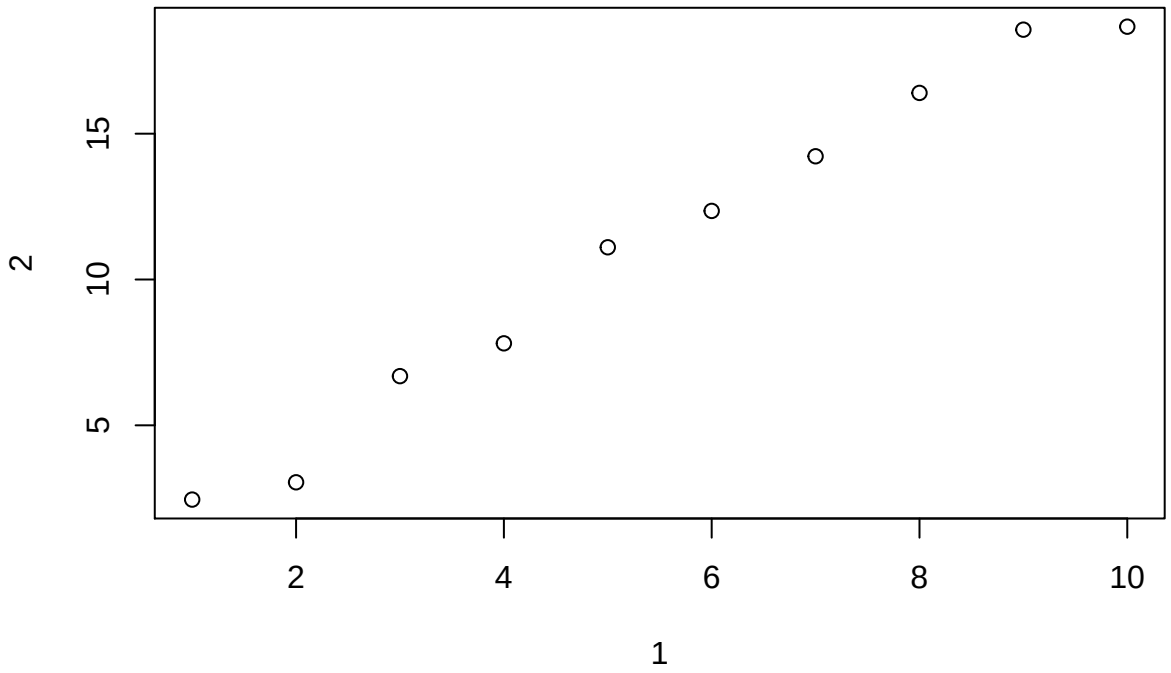
```
annoying[["1"]]
```

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

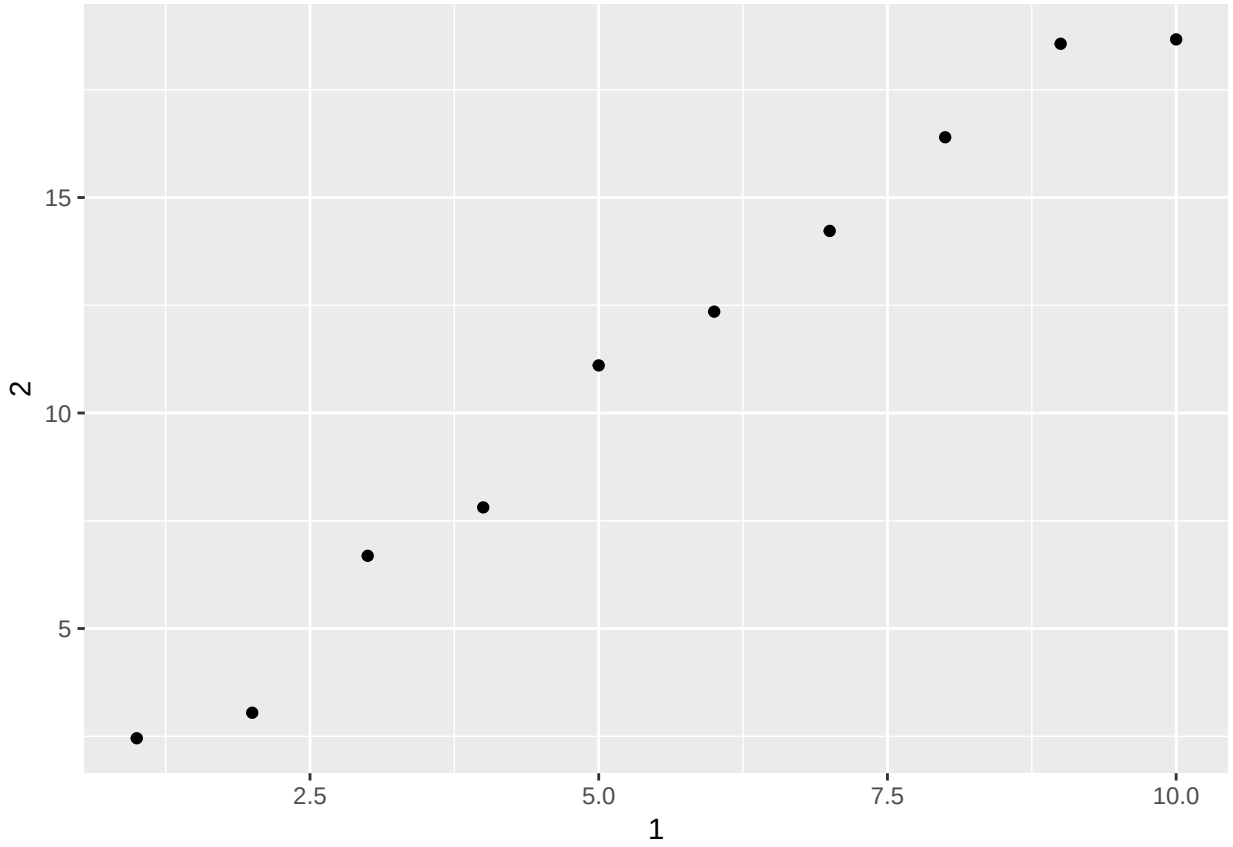
```
annoying$'1'
```

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

```
#(2)Plotting a scatterplot of 1 vs 2.
plot(annoying)
```




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




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

```
#(4)Renaming the columns to one, two and 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> 2.454134, 3.046401, 6.686756, 7.810543, 11.104764, 12.35...
## $ three <dbl> 2.454134, 1.523200, 2.228919, 1.952636, 2.220953, 2.0586...
```

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

```
?enframe()
enframe(c(a = 5, b = 7, c=25))
```

```
## # A tibble: 3 x 2
##   name  value
##   <chr> <dbl>
## 1 a      5.00
## 2 b      7.00
## 3 c     25.0
```

enframe() converts named atomic vectors or lists to two-column data frames. For unnamed vectors, the natural sequence is used as name column.

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

```
?print.tbl_df()
```

The print function for tibbles is in print.tbl_df: n_extra : Number of extra columns to print abbreviated information for, if the width is too small for the entire tibble. If NULL, the default, will print information about at most tibble.max_extra_cols extra columns

12.6.1 Exercise

```
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>
```

```
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...
```

```
who1
```

```
## # A tibble: 76,046 x 6
##    country     iso2  iso3   year key         cases
##  * <chr>       <chr> <chr> <int> <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
```

```
who2 <- who1 %>%
 mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
who2
```

```
## # A tibble: 76,046 x 6
##    country     iso2  iso3   year key         cases
```

```
##    <chr>       <chr> <chr> <int> <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 <- 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?

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

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

Yes, I think it is reasonable. Because the rows with the missing values(NA)were redundant as they were not giving any useful information. These missing values may either were inserted explicitily by whoever created the dataset, or they were really missing values. The main difference between NA and 0 is that 0 is an actual value indicating that there were no cases of tubercolosis in a country in a given year, whereas a NA value tells us we have no information on the observation at all.

And after we set the na.rm=TRUE to all the data (treat them the same), the explicitly and implicitly missing values are not displayed in who1. And this is reasonable because if we removed all the missing values(including explicitly and implicitly) from the data set, we will have fewer data to analyze.

2 What happens if you neglect the mutate() step? (mutate(key = stringr::str_replace(key, "newrel", "new_rel")))

```
who3a <- who1 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
```

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 2580 rows
## [73467, 73468, 73469, 73470, 73471, 73472, 73473, 73474, 73475, 73476,
## 73477, 73478, 73479, 73480, 73481, 73482, 73483, 73484, 73485, 73486, ...].
```

separate causes the warning message saying the values are too few

```
filter(who3a, 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
```

if we check the rows for keys beginning with "newrel_", we see that sexage is messing, and type = m014.

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

```
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 visualization of the data.

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