Exercise: **Functions**

Day 4, Part B

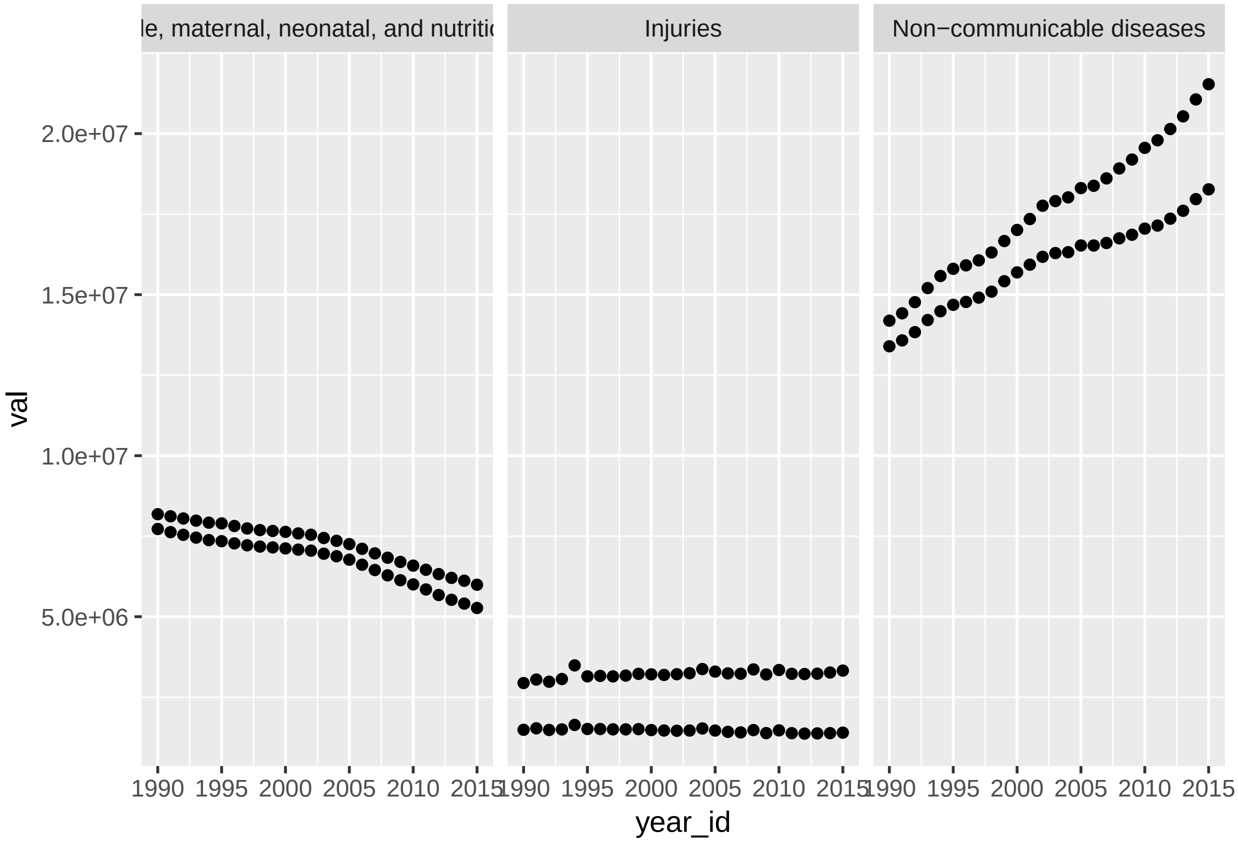
```
> library(foreign)
> library(ggplot2)
> library(reshape2)
```

1. Write a function where the arguments are `in_file`, `x_variable`, `y_variable`, `facet_variable`, and `out_file`, that loads the specified input file, creates a scatter plot with the specified x, y, and facetting variables using `ggplot`, and then saves this plot to the specified output file while also returning the ggplot object. Assume the input file is a csv and the output file is a pdf. Hint: in this particular case, `aes_string()` is more user-friendly than `aes()` when calling ggplot.

```
> scatter_plot <- function(in_file, x_variable, y_variable, facet_variable, out_file) {
+   # load data
+   data <- read.csv(in_file)
+
+   # make plot
+   gg <- ggplot(data, aes_string(x = x_variable, y = y_variable)) +
+     facet_wrap(facet_variable) + geom_point()
+
+   # save plot
+   pdf(out_file)
+   print(gg)
+   dev.off()
+
+   # return plot
+   return(gg)
+ }
```

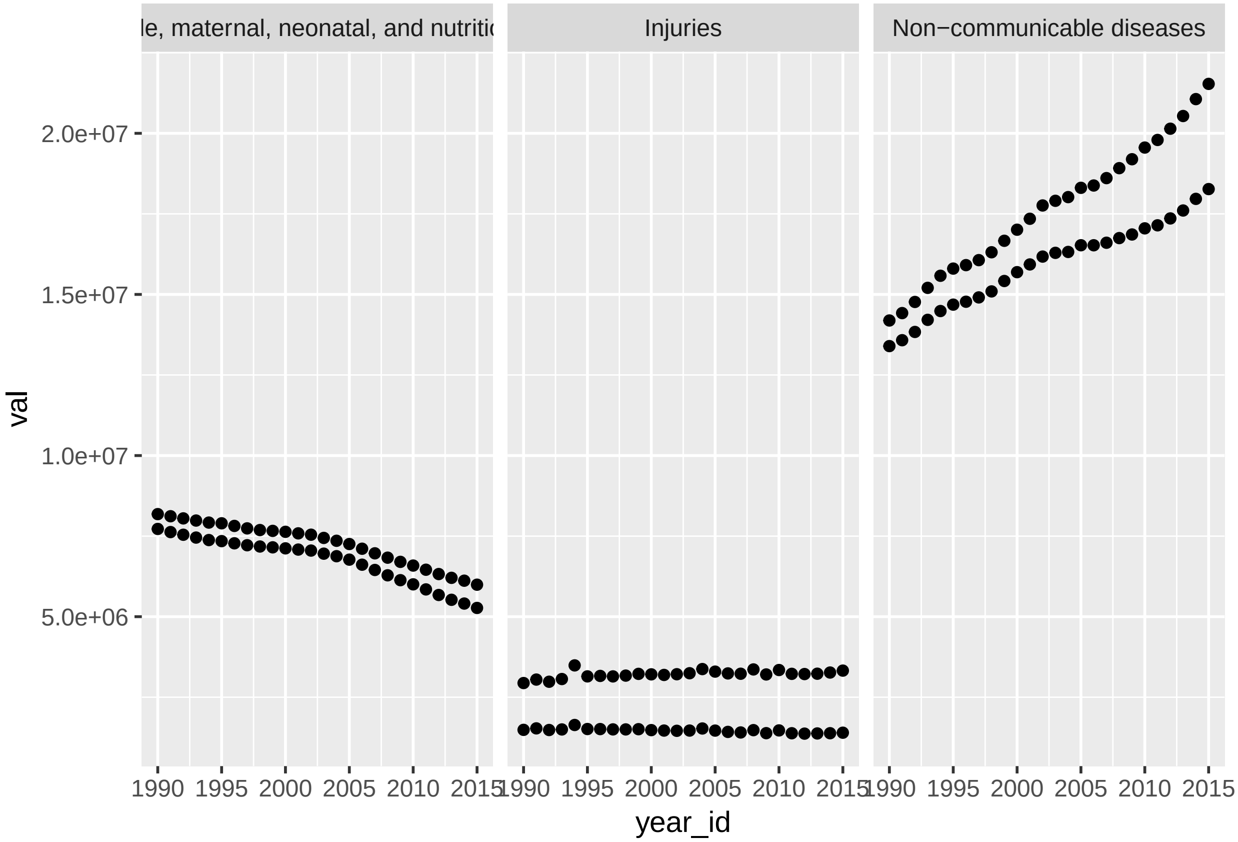
   a. Test this function using the data in 'data/gbd2015_global_deaths.csv', with year (`year_id`) on the x-axis, mean deaths (`val`) on the y-axis, and cause group (`cause_name`) as the facetting variable.

```
> main_dir <- "C:/Users/ngraetz/Documents/repos/r_training_penn/" # CHANGE TO YOUR LOCAL COP
> scatter_plot(in_file = paste0(main_dir, 'data/gbd2015_global_deaths.csv'),
+              x_variable = 'year_id', y_variable = 'val', facet_variable = 'cause_name',
+              out_file = paste0(main_dir, 'output/gbd2015_global_deaths_scatter.pdf'))
```



b. Add assertions to the function to check that the input and output files are the expected format, and to provide a helpful error message if not. Test this by specifying an output file that is a jpeg, rather than a pdf. Hint: think back to the string functions lecture.

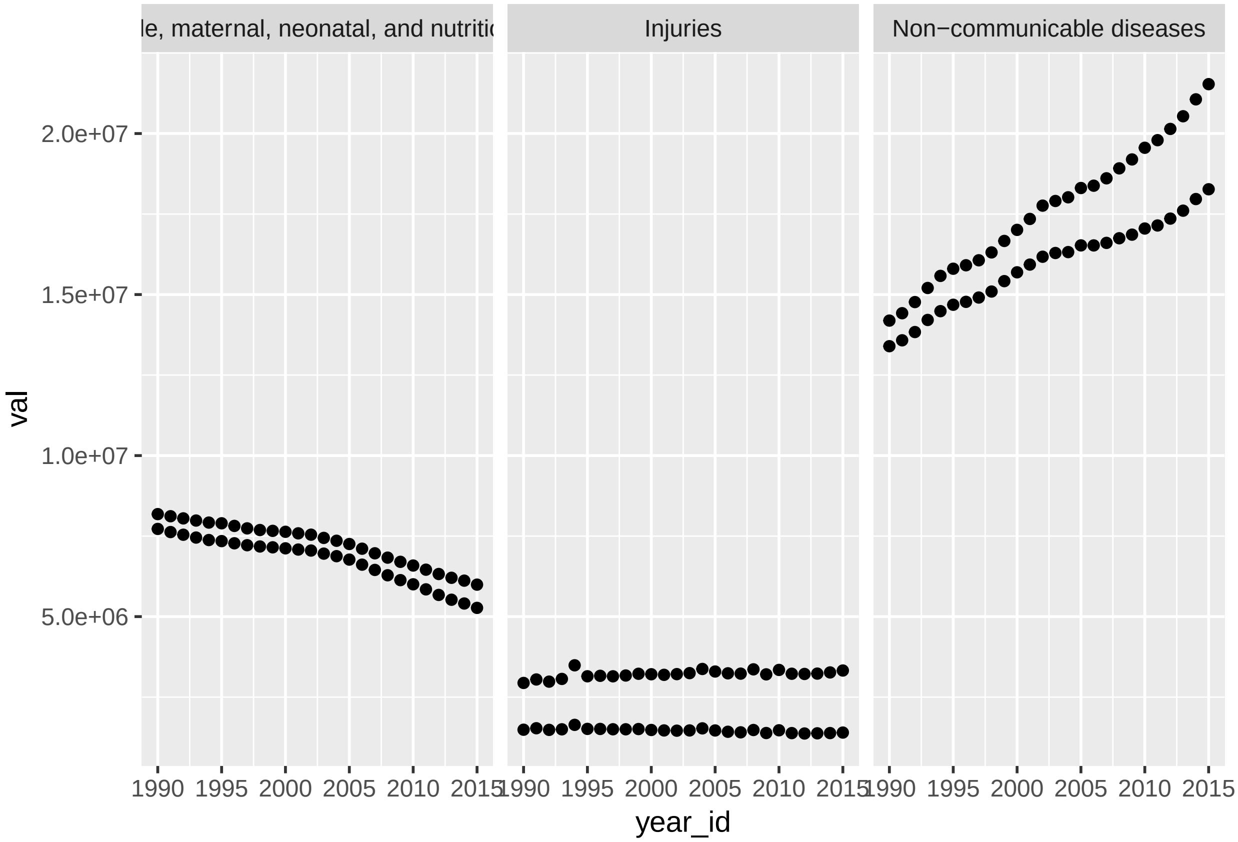
```
> scatter_plot <- function(in_file, x_variable, y_variable, facet_variable, out_file) {
+
+   # check input and output data format
+   if (!grepl('.csv$|.CSV$', in_file)) stop("'in_file' must be a csv file")
+   if (!grepl('.pdf$|.PDF$', out_file)) stop("'out_file' must be a pdf file")
+
+   # load data
+   data <- read.csv(in_file)
+
+   # make plot
+   gg <- ggplot(data, aes_string(x = x_variable, y = y_variable)) +
+     facet_wrap(facet_variable) + geom_point()
+
+   # save plot
+   pdf(out_file)
+   print(gg)
+   dev.off()
+
+   # return plot
+   return(gg)
+ }
>
> # this should work...
> scatter_plot(in_file = paste0(main_dir, 'data/gbd2015_global_deaths.csv'),
+      x_variable = 'year_id', y_variable = 'val', facet_variable = 'cause_name',
+      out_file = paste0(main_dir, 'output/gbd2015_global_deaths_scatter.pdf'))
```



```
> # and this should throw an error...
> #scatter_plot(in_file = paste0(main_dir, 'data/gbd2015_global_deaths.csv'),
> #     x_variable = 'year_id', y_variable = 'val', facet_variable = 'cause_name',
> #     out_file = paste0(main_dir, 'output/gbd2015_global_deaths_scatter.jpeg'))
```

c. Add assertions to the function to check that `x_variable`, `y_variable`, and `facet_variable` all exist as columns in the input data file, and to provide a helpful error message if not. Test this by providing an incorrect variable name.

```
> scatter_plot <- function(in_file, x_variable, y_variable, facet_variable, out_file) {
+
+   # check input and output data format
+   if (!grepl('.csv$|.CSV$', in_file)) stop("'in_file' must be a csv file")
+   if (!grepl('.pdf$|.PDF$', out_file)) stop("'out_file' must be a pdf file")
+
+   # load data
+   data <- read.csv(in_file)
+
+   # check that all variables exist
+   if (!x_variable %in% names(data)) stop(paste(x_variable, 'is not a column in', in_file))
+   if (!y_variable %in% names(data)) stop(paste(y_variable, 'is not a column in', in_file))
+   if (!facet_variable %in% names(data)) stop(paste(facet_variable, 'is not a column in', i
+
+   # make plot
+   gg <- ggplot(data, aes_string(x = x_variable, y = y_variable)) +
+     facet_wrap(facet_variable) + geom_point()
+
+   # save plot
+   pdf(out_file)
+   print(gg)
+   dev.off()
+
+   # return plot
+   return(gg)
+ }
>
> # this should work...
> scatter_plot(in_file = paste0(main_dir, 'data/gbd2015_global_deaths.csv'),
+      x_variable = 'year_id', y_variable = 'val', facet_variable = 'cause_name',
+      out_file = paste0(main_dir, 'output/gbd2015_global_deaths_scatter.pdf'))
```

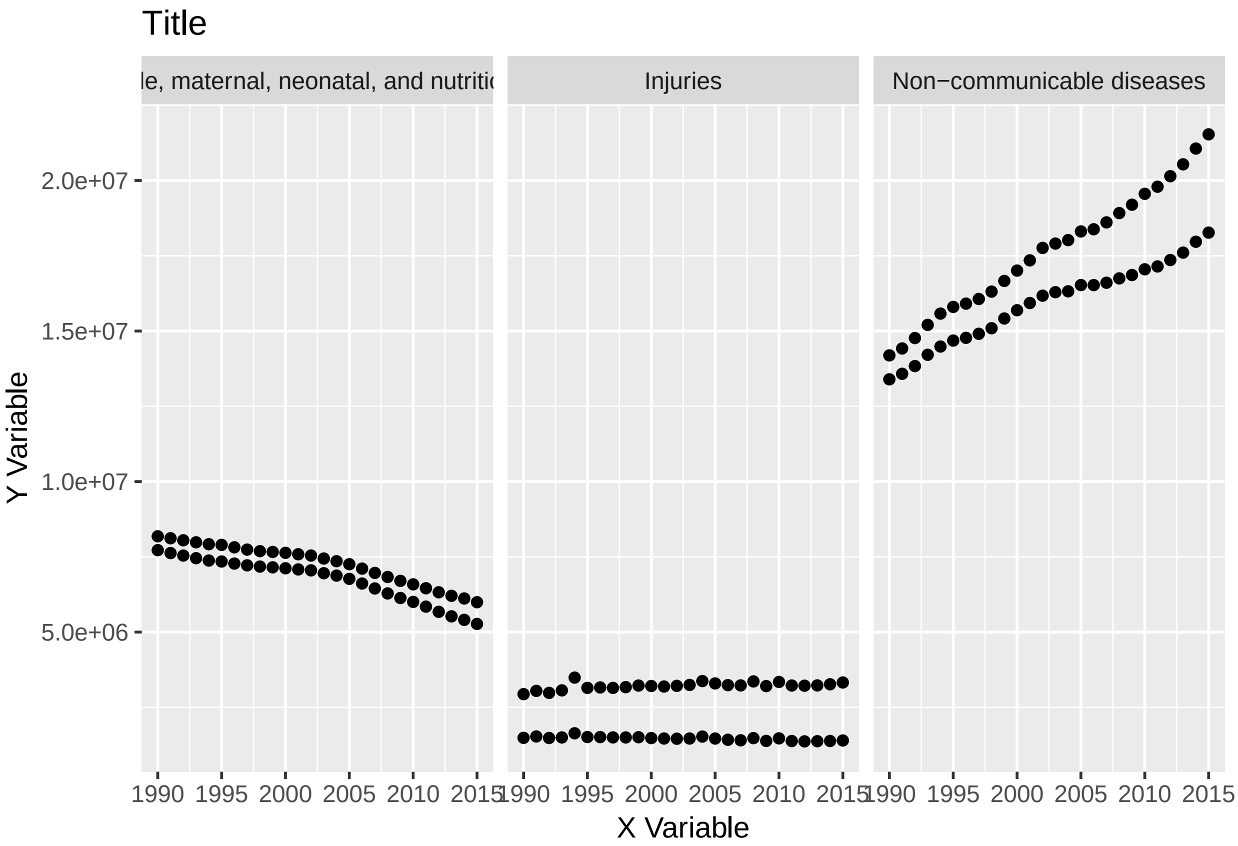


```
> # and this should throw an error...
> #scatter_plot(in_file = paste0(main_dir, 'data/gbd2015_global_deaths.csv'),
> #     x_variable = 'year_id', y_variable = 'mean', facet_variable = 'cause_name',
> #     out_file = paste0(main_dir, 'output/gbd2015_global_deaths_scatter.pdf'))
```

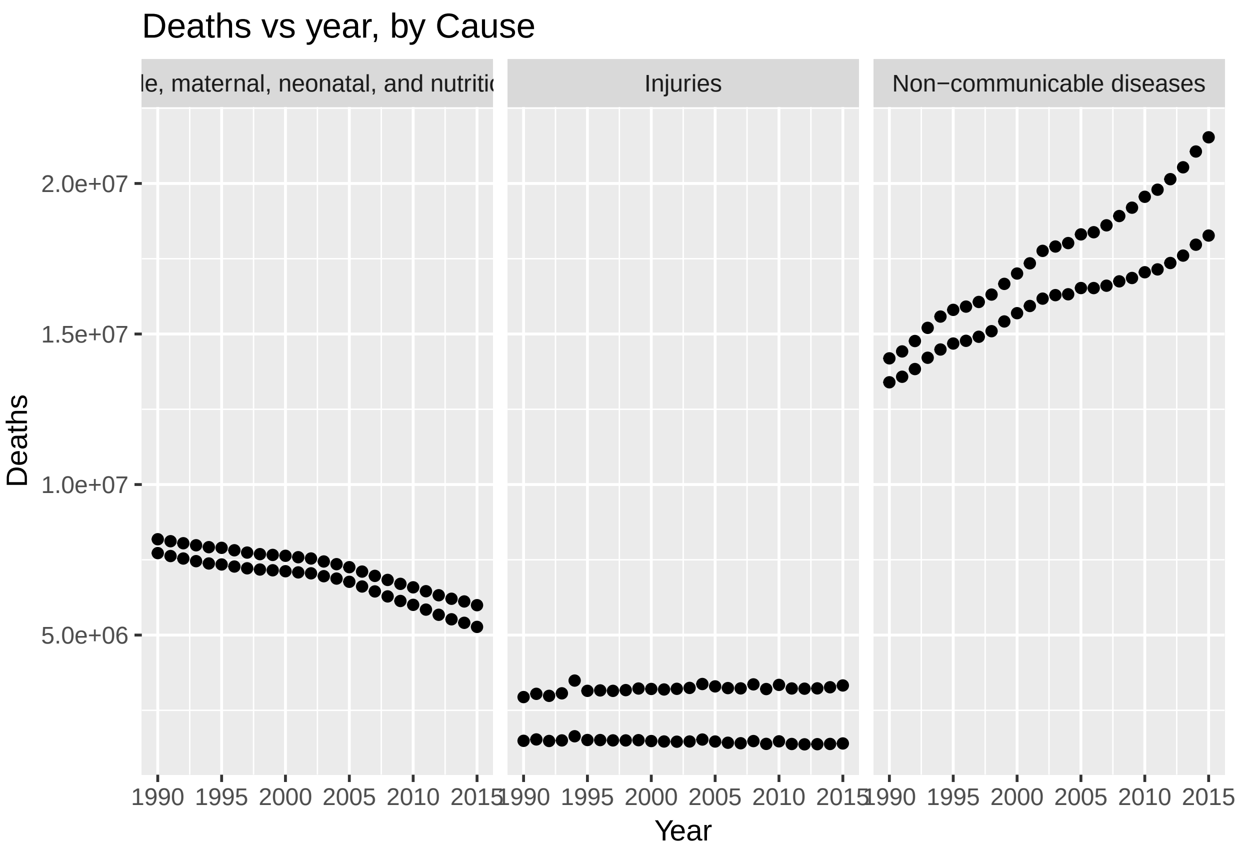
d. Add arguments `x_label`, `y_label`, and `title` to provide labels to the plot. Set them to default to "X Variable", "Y Variable", and "Title", respectively. Test that this works as expected, both when values are supplied for these arguments and when they are not.

```
> scatter_plot <- function(in_file, x_variable, y_variable, facet_variable, out_file,
+                          x_label = "X Variable", y_label = "Y Variable", title = "Title")
+
+   # check input and output data format
+   if (!grepl('.csv$|.CSV$', in_file)) stop("'in_file' must be a csv file")
+   if (!grepl('.pdf$|.PDF$', out_file)) stop("'out_file' must be a pdf file")
```

```
+
+   # load data
+   data <- read.csv(in_file)
+
+   # check that all variables exist
+   if (!x_variable %in% names(data)) stop(paste(x_variable, 'is not a column in', in_file))
+   if (!y_variable %in% names(data)) stop(paste(y_variable, 'is not a column in', in_file))
+   if (!facet_variable %in% names(data)) stop(paste(facet_variable, 'is not a column in', i
+
+   # make plot
+   gg <- ggplot(data, aes_string(x = x_variable, y = y_variable)) +
+     facet_wrap(facet_variable) + geom_point() +
+     labs(x = x_label, y = y_label, title = title)
+
+   # save plot
+   pdf(out_file)
+   print(gg)
+   dev.off()
+
+   # return plot
+   return(gg)
+ }
>
> # using the defaults for labels/titles...
> scatter_plot(in_file = paste0(main_dir, 'data/gbd2015_global_deaths.csv'),
+      x_variable = 'year_id', y_variable = 'val', facet_variable = 'cause_name',
+      out_file = paste0(main_dir, 'output/gbd2015_global_deaths_scatter.pdf'))
```



```
> # providing label/title arguments
> scatter_plot(in_file = paste0(main_dir, 'data/gbd2015_global_deaths.csv'),
+      x_variable = 'year_id', y_variable = 'val', facet_variable = 'cause_name',
+      out_file = paste0(main_dir, 'output/gbd2015_global_deaths_scatter.pdf'),
+      x_label = 'Year', y_label = 'Deaths', title = 'Deaths vs year, by Cause')
```



2. Write a function that takes one argument `data`, a data frame, and returns a report of the number of missing and non-missing values for each variable, as well as the number of unique values, e.g.:

```
> # note that there are many ways to do this, and more than one valid output format
> missing_report <- function(data) {
+   report <- data.frame(vars = names(data),
+                        missing = NA,
+                        non_missing = NA,
+                        values = NA)
+
+   for (v in names(data)) {
+     report[report$vars == v, 'missing'] <- sum(is.na(data[, v]))
+     report[report$vars == v, 'non_missing'] <- sum(!is.na(data[, v]))
+     report[report$vars == v, 'values'] <- length(unique(data[, v]))
+   }
+
+   return(report)
+ }
```

```
> data <- read.csv(paste0(main_dir, "data/ebola_polygon_data.csv"))
> missing_report(data)
vars missing non_missing values
1      UNIQ_ID       0          54     54
```

```
2         NAME       0          54     53
3      Country       0          54      6
4        Virus       0          54      4
5    CASE_TYPE       0          54      3
6    DATA_TYPE       0          54      1
7          LAT       0          54     46
8         LONG       0          54     45
9    SPR_ORDER      18          36      6
10    SOURCE_1      32          22     20
11    SOURCE_2      54           0      1
12    SOURCE_3      54           0      1
13     STR_DAY      42          12     11
14    STR_MNTH      18          36     10
15    STR_YEAR       4          50     15
16     END_DAY      52           2      3
17    END_MNTH      52           2      3
18    END_YEAR      52           2      3
19    REP_CASE      28          26     18
20   REP_DEATH      39          15     12
21       OB_ID       0          54     23
22  OB_STR_DAY       3          51     13
23 OB_STR_MNTH       0          54     10
24 OB_STR_YEAR       0          54     15
25  OB_END_DAY      13          41     12
26 OB_END_MNTH       0          54     11
27 OB_END_YEAR       0          54     14
28     OB_CASE       0          54     18
29    OB_DEATH       0          54     19
```

3. Repeat question 1 from lecture 4a using `lapply()` instead of a `for` loop.

```
> data <- lapply(1997:2015, function(year) {
+   if (year < 2004) {
+     sub <- read.csv(paste0(main_dir, "/data/wa_income_", year, ".csv"))
+   } else {
+     sub <- read.dta(paste0(main_dir, "/data/wa_income_", year, ".dta"))
+     sub <- plyr::rename(sub, c("FIPS" = "fips", "median_income" = "income_median"))
+   }
+   return(sub)
+ })
> data <- do.call('rbind', data)
> summary(data)
fips             year      income_median      poverty
 Min.   :53001   Min.   :1997   Min.   :27453   Min.   : 6.60
 1st Qu.:53019   1st Qu.:2001   1st Qu.:36992   1st Qu.:11.50
 Median :53039   Median :2006   Median :42369   Median :14.10
 Mean   :53039   Mean   :2006   Mean   :43726   Mean   :14.26
 3rd Qu.:53059   3rd Qu.:2011   3rd Qu.:48693   3rd Qu.:16.40
 Max.   :53077   Max.   :2015   Max.   :81816   Max.   :32.30
```

Bonus:

4. Repeat the first part of question 3 from lecture 4a using `sapply()` or `tapply()` instead of a `for` loop. Then do the same again, but calculate the ratio of the maximum to the minimum value instead of the mean.

```
> # mean with sapply...
> sapply(1997:2015, function(yy) {
+   mean(data[data$year == yy, 'income_median'])
+ })
[1] 36051.05 37450.33 37636.90 39212.51 38255.59 38989.44 39935.90 41323.79 41837.21
[10] 43795.28 46171.38 48219.13 47017.18 46621.33 47544.08 48141.85 49234.31 50873.38
[19] 52487.00
```

```
> # mean with tapply...
> tapply(data$income_median, data$year, mean)
1997     1998     1999     2000     2001     2002     2003     2004     2005
36051.05 37450.33 37636.90 39212.51 38255.59 38989.44 39935.90 41323.79 41837.21
    2006     2007     2008     2009     2010     2011     2012     2013     2014
43795.28 46171.38 48219.13 47017.18 46621.33 47544.08 48141.85 49234.31 50873.38
    2015
52487.00
```

```
> # max/min with sapply...
> sapply(1997:2015, function(yy) {
+   x <- data[data$year == yy, 'income_median']
+   max(x)/min(x)
+ })
[1] 1.868648 1.872010 1.816942 1.852431 1.857221 1.819798 1.761449 1.822441 1.842353
[10] 1.928604 1.906808 2.005580 1.945463 1.853792 1.985815 2.136076 2.010447 2.004605
[19] 2.034768
```

```
> # max/min with tapply...
> tapply(data$income_median, data$year, function(x) max(x) / min(x))
1997     1998     1999     2000     2001     2002     2003     2004     2005
1.868648 1.872010 1.816942 1.852431 1.857221 1.819798 1.761449 1.822441 1.842353
    2006     2007     2008     2009     2010     2011     2012     2013     2014
1.928604 1.906808 2.005580 1.945463 1.853792 1.985815 2.136076 2.010447 2.004605
    2015
2.034768
```

5. Load the US cigarette data ('data/us_state_cigarette_data.rdata') and put all four of the data frames into a single list. Use the `Reduce()` function to merge all of these data sets in one go, rather than by calling merge directly three separate times. Do this once retaining only the rows where there are matches, and a second time retaining all rows.

```
> # load data and combine in a list
> load(paste0(main_dir, "/data/us_state_cigarette_data.rdata"), verbose = T)
Loading objects:
  cig_tax
  cig_csmp
  pop
  locs
```

```
> cig_data_list <- list(cig_tax, cig_csmp, pop, locs)
>
> # for only matching rows, we are totally fine with defaults
> cig_data <- Reduce(merge, cig_data_list)
> dim(cig_data)
[1] 204   6
```

```
> # for all rows, we need to set `all=T`, which requires defining a function that calls
> # `merge()` with this setting
> cig_data <- Reduce(function(x, y) merge(x, y, all = T), cig_data_list)
> dim(cig_data)
[1] 306   6
```