Exercise: **Functions**

Day 4, Part B

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.

    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.

    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.

    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.

    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.

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.:

```
> 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.

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.

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.