# Problem Set 1

*whomsoever*

## Introduction

This week's problem set will walk you through some simple tools for analyzing data with R. Don't worry if you aren't yet comfortable with R—the coding exercises are mostly expositional.

To complete this problem set we've provided a few datasets for you to choose from. Pick whichever you fancy. If there's some other data you'd rather work on, you are welcome to—but please consult me first.

We produced this document using R Markdown. The great thing about R Markdown is it lets you present code and prose commentary in the same document (as we have done here). To complete this problem set you can simply use the template we've provided—just open the Rmd file and write your code in the code chunks we've left. For non-coding exercises, type your responses in the spaces provided.

To compile your R Markdown file (i.e. create the output document), click 'Knit'. Note that for pdf output (like this document), you'll need to have some form of TeX installed on your computer—https://www.latex-project.org/get/. Alternatively you can use html output, which doesn't require TeX. You'll need to specify `html_document` in the YAML metadata at the very top of the document.

For tips on R Markdown formatting and syntax, check out the course notes (chapter 1) or the cheatsheet we've provided.

We encourage you to use R Markdown for this problem set—our instructions assume you do—but if you'd rather use a different environment, e.g. Jupyter notebooks (etc), you are welcome to. Just make sure your final report is presentable, code and all.

Any time you are confused about a command in R, use `help()` to bring up its documentation. If you get an error message you don't understand, use google—it's very likely someone else has encountered the same error and written about it on Stack Overflow or the like.

The packages used in this problem set are:

- `tidyverse` (is a set of eight packages)
- `stats`

Make sure these are loaded in the setup chunk using the `library()` command. If you get the error message "there is no package called 'XXX' ", you need to first install the package using `install.packages('XXX')`, then load it into the session using `library()`.

## I – Some Warmup Questions

For this problem set we'll use the Economic Freedom dataset, sourced from the Fraser Institute: https://www.fraserinstitute.org/economic-freedom/.

Your first order of action is loading the data into your R session. First download the data file and save it somewhere convenient on your computer (e.g. the same directory your code file is in). To load csv data you can use the `read.csv()` function. You must give the function the file path to your data, in the form `read.csv('path-to-my-data/mydata.csv')`. If you don't know how to find a file path, give it a google.

If you're having trouble, try runing the command `getwd()`—this will show you what directory your current R session is in. You can also use `setwd()` to manually specify a working directory.

1. In the chunk below load the dataset into R. If your data is in the same directory as your code file, the sample code we've provided below should work—simply uncomment it and run it.

```
#efwdata <- read.csv('efw.csv')
```

Great. The next thing you'll want to do is view your data. There are two ways to do this:

- use the `View()` command to view the entire dataset in a separate window
- use the `head()` or `tail()` command to view the first or last few rows of the dataset

Try not to use `View()` for huge datasets as it's very memory intensive.

Now go ahead and view your dataset—make note of the variables it contains and how each is encoded (numerically, categorically, etc.). You don't need to show you did this.

Next, run the following checks on your dataset:

2. Check the class of your dataset using `class()`:

You should find that R has stored your dataset as a "data frame". A data frame is a 2-dimensional array where different columns can have different data types—this makes it a common structure for tabular data.

3. Check the dimensions of your dataset (the number of rows and columns) using `dim()`:

4. Check the column names of your dataset using `colnames()`:

5. Check the structure of your dataset using `str()`:

The `str()` function displays the internal structure of an object in R. If you feed it a data frame—as you just did—it should display each column in the dataset, its data type(s), and a preview of its observations.

6. Using the output from q5, identify which variables in your data are categorical, discrete, and continuous. How are they stored in R (i.e. in terms of data type)?

The extract operator, `$`, can be used to extract a named object from a larger data structure. You can use it, for instance, to extract a column from a data frame, in the form `dataset$column`.

7. Which years are recorded in the data? Use the `unique()` function on the `year` variable (you may have to use `$`):

# III – Simple Visualizations

Now let's make a few plots to get a sense for what your data looks like.

**Histograms** are a good way to visualize the distribution of data in a variable, since they display the frequency (or relative frequency) of observations within specified intervals or "bins".

Let's try to visualize the distribution of economic freedom scores, since this is the main variable of interest in the dataset. Note since there are observations across many years, you may want to first consider filtering your data to contain observations from a single year only. You can do this using the `filter()` function from the `dplyr` package.

1. Filter your dataset for observations from the year 2017, and assign your filtered observations to a new object with an appropriate name. Below is some sample code showing two ways to do this:

```
## one way
#efw2017 <- filter(efwdata, year == 2017)

## another way, using a pipe
#efw2017 <- efwdata %>% filter(year == 2017)
```

The latter method uses a pipe, `%>%`. This is a `dplyr` feature that forwards or "pipes" the values on its left hand side into the expressions(s) on its right hand side. The advantage of using a pipe may not be apparent in the simple example above, but when running many functions simultaneously it is far superior to the other method (as you will soon see).

To make a plot use the `ggplot()` function (note although the package name is `ggplot2`, the function call is not appended with a "2"). The `ggplot()` function takes two arguments:

- `data`, a data frame whose variables you want to plot
- `mapping`, an aesthetic mapping for which variable(s) go on which axes

You'll also need to specify a geometric mapping, in the form `+ geom_XXX()`, to specify which variable(s) go on which axes (i.e. geometrically map the data onto the $x$ and $y$ axes). The following examples will demonstrate how to use `ggplot()`.

2. Let's now make a histogram of the economic freedom scores in 2017 by plotting a histogram. Below is some sample code to help you do this. Note histograms only require an aesthetic mapping for the $x$-axis (since the $y$-axis is simply frequency). Make note of the syntax for specifying aesthetic and geometric mappings.

```
# ggplot(data = efw2017, mapping = aes(x = economic_freedom)) +
#   geom_histogram(bins = 50)
```

Great. You can adjust the number of bins by specifying the argument `bins = XXX` in the geometric mapping. Alternatively you can use `binwidth = XXX` to specify that each bin should have a certain width.

3. Now run the following code, which produces a histogram similar to the one above, but with *relative frequency* on the $y$-axis instead of frequency. You can do this by specifying `aes(y = ..density..)` in the geometric mapping. Note also the additional aesthetic parameters, which change the labels/colors/theme of the plot. Note how each is appended onto the main function call with a `'+'`. Change the generic title and axis label to something more relevant.

```
# ggplot(data = efw2017, mapping = aes(x = economic_freedom)) +
#   geom_histogram(bins = 50, aes(y = ..density..), fill = 'violet') +
#   ggtitle('a title') +
#   xlab('an axis label') +
#   theme_light()
```

4. What is the area contained by this relative frequency histogram?

When there are categorical variables in the data, it's often useful to compare the distribution of a numeric variable across each of the categories. Thankfully `ggplot()` has a feature that allows you to do just that—`facet_wrap()`. This function takes a categorical variable and splits a plot into constituent categories or "facets".

5. Now run the following code, which uses `facet_wrap()` to create individual plots for each continent:

```
# ggplot(data = efw2017, aes(x = economic_freedom)) +
#   geom_histogram(bins = 50) +
#   facet_wrap(~continent)
```

6. Describe what you see (in the plot for q5). Are the distributions similar across the categories?

Another way you can compare the distribution of a continuous variable across different categories is using a **box and whisker plot**. These require two variables—one should be categorical and the other numeric. To make a box and whisker plot you should specify `+ geom_boxplot()` as the geometric mapping.

7. Make a box and whisker plot of `economic_freedom` (numeric) on `continent` (categorical). Note since box and whisker plots require two variables, you'll need two aesthetic mappings—it should look something like `aes(x = ..., y = ...)` etc. Usually the categorical variable goes on the $x$-axis. What do you see?

**Line plots** are a good way to visualize how a variable evolves over time. To make a line plot you should specify `+ geom_line()` as the geometric mapping.

8. Let's visualize how the economic freedom score has evolved over time in a particular country. Choose a country from the dataset, and create a filtered dataset with observations from that country only. Then make a line plot of `economic_freedom` (on the $y$-axis) on `year` (on the $x$-axis) for this country's data. Use `geom_line()`. What do you see? Does the score change over the years recorded in the data?

9. Now let's compare how economic freedom has evolved over time across several countries. Choose five countries from the dataset, and create a list containing their names (make sure to write them exactly as they are in the data). Then create a filtered dataset that contains observations from only these countries (you may have to use `%in%`).

```
# my_countries <- c('country1', 'country2', 'country3', 'country4', 'country5')

# efw_reduced <- efwdata %>% filter(country %in% mycountries)
```

10. Make a line plot of `economic_fredom` on `year`, using the reduced dataset with observations on your five chosen countries. To make separate lines for each country, add a color argument to your aesthetic mapping, e.g. `mapping = aes(x = ..., y = ..., color = country)`. What do you see?

Another way to make a line plot is using `stat_smooth(method = 'loess')` instead of `geom_line()`. This will fit a smoothed line using the method of loess (this is a modeling technique that uses weighted regression to fit a smooth curve).

11. Make the same plot as in q11, but this time use `stat_smooth(method = 'loess')`. The grey region you'll see around the line is its standard error. You can remove this by including the argument `se = FALSE`.

# IV – Summarizing Data

Summary statistics are point values that describe or "summarize" some aspect of a distribution. They can help extract meaning from a distribution of data.

A **measure of central tendency** is a summary statistic that describes the central or typical value of a distribution. The mean and median are both examples.

The **mean** is the sum of all observations divided by the number of observations:

$$\bar{x} = \frac{1}{n}\sum_{i}^{n} x_i$$

The **median** is middle value or 50th percentile of a distribution:

$$m = \frac{1}{2}\left(x_{(n/2)} + x_{(n/2)+1}\right)$$

1. Use `mean()` and `median()` to compute the mean and median values of the variable `sound_money` in 2017 (use the filtered 2017 dataset you created earlier). Are the values similar, or are they different? If they are different, can you think of a reason why?

2. Plot a relative frequency histogram of `sound_money` in 2017 to visualize its distribution. Add two vertical lines to your plot showing where the mean and median are. You can use `geom_vline()` to add vertical lines and `geom_text()` to annotate your plot. Below is some sample code to help you do this.

```
# ggplot(data = efw2017, aes(x = sound_money)) +
#   geom_histogram(bins = 30, aes(y = ..density..)) +
#   geom_vline(xintercept = mean(efw2017$sound_money), color = 'red') +
#   geom_text(x = mean(efw2017$sound_money)-0.5, y = 30, color = 'red', label = 'mean') +
```

```
#   geom_vline(xintercept = median(efw2017$sound_money), color = 'blue') +
#   geom_text(x = median(efw2017$sound_money)+0.5, y = 30, color = 'blue', label = 'median')
```

3. In the above plot, what is the area contained by the histogram to the left of the median?

4. Comment on whether the mean or median is a more appropriate measure of central tendency in this case.

A **measure of spread** is a summary statistic that describes the variation or spread of a distribution.

The **standard deviation** is the most common measure of spread. It's defined as the average distance of each observation from the mean:

$$s = \sqrt{\frac{1}{n-1}\sum_{i}^{n}(x_i - \bar{x})^2}$$

5. Use `sd()` to compute the standard deviation of the variable `sound_money` in 2017.

**Aggregating data** is useful if you want to summarize the information in a large dataset across subsets or categories in the data. To aggregate data you need a grouping variable (categorical) and you must specify some operation to perform on the nongrouping variable (usually a sum or a mean).

6. Let's say you want to compare the average economic freedom score across each of the continents for each year in the data. To do this you'll need to create an aggregated datase. In this case your grouping variables are `continent` and `year`, and you'll want to calculate a mean for the nongrouping variable, `economic_freedom`. First, select the relevant variables from the full dataset. Then use `group_by()` to specify the grouping variables and `summarize()` to specify the function you want to perform on the nongrouping variables. Below is some sample code to help you do this.

```
# efw_aggregated <- efwdata %>%
#   select(continent, year, economic_freedom) %>%
#   group_by(continent, year) %>%
#   summarize(avg_economic_freedom = mean(economic_freedom, na.rm = TRUE))
```

Great. If you call `View()` on your aggregated dataset, you should be able to see the aggregated statistics.

7. Using your aggregated dataset, make a smoothed line plot of `avg_economic_freedom` on `year`, using different colors for each continent. What do you see?

# V – Intervals and Uncertainty

A summary statistic such as the mean is known as a *point* statistic since it uses a single value to describe some aspect of a distribution (in this case central tendency). When computed from sample data, it's referred to as the *sample* mean since it's an **estimate**—specifically, a ***point* estimate**—of the thing we're actually trying to measure, i.e. *true* mean.

The true mean of a distribution is the value you'd get if you somehow had access to *all* the relevant data on whatever you're measuring. Most of the time you only have samples of data to work with.

1. Compute the mean economic freedom score in 2017.

This value is a point estimate for the *true* mean economic freedom score, which is unknown. From any one sample of data you often don't know how close the sample mean is to the true mean—there is usually some unknown error in your estimate. Thus when using sample data to describe some property of a distribution, such as central tendency, often its better to provide a *range* of values rather than a single value, since a range of values will better capture the uncertainty in your result.

A **confidence interval for the mean** is just that—it's an **interval estimator** of the center of a distribution. Confidence intervals are incredibly useful for capturing the uncertainty in a point statistic.

Every confidence interval has a confidence level—e.g. 80%, 90%, 95%—that describes the approximate probability the interval contains the true parameter. We won't ask you to think much about probabilities here, but for now all you need to know is that the higher the confidence level, the higher the likelihood the interval contains the true value.

Just as the `mean()` function takes a vector array of numbers and returns its mean, you can also construct a function that takes a vector array of numbers and returns a confidence interval for the mean. Although there isn't currently an inbuilt function that does this, we've written one for you:

```
confidence_interval = function(data, conflevel) {
  xbar = mean(data)           # sample mean
  n = length(data)            # sample size
  SE = sd(data) / sqrt(n)     # standard error
  alpha = 1 - conflevel       # alpha

  lb = xbar + qt(alpha/2, df = n-1) * SE    # calculate lower bound
  ub = xbar + qt(1-alpha/2, df = n-1) * SE  # calculate upper bound

  cat(paste(c('sample mean =', round(xbar,3), '\n',
              conflevel*100, '% confidence interval:', '\n',
              'lower bound =', round(lb,3), '\n',
              'upper bound =', round(ub,3))))
}
```

Running this code will define a new function in the Environment, called `confidence_interval()`. Once defined, this function can be used to compute a confidence interval for the mean of an array of data. The way it's written, it takes two arguments: `data`, a vector array of numbers, and `conflevel`, the desired confidence level. E.g. if you want a 90% confidence interval, specify `conflevel = 0.9`.

2. Use the function to compute a 95% confidence interval for the mean economic freedom in 2017.

3. Do you anticipate that a 99% confidence interval is larger or smaller than the one above? Why?

# VI – Tests

Broadly speaking, a **statistical hypothesis** is an assumption about the *true* value of something. Many studies start with a hypothesis of some sort—e.g. we believe the true mean economic freedom scor is 4.5—after which sample data is collected and analyzed to determine whether the initial hypothesis was indeed correct.

In a hypothesis test, you essentially compare two competing models that attempt to describe the true value of something. You then use the comparison to reject one of the models.

A hypothesis test is usually formulated in terms of two hypotheses, as follows:

- the **null hypothesis**, $H_0$, a proposed model for the true value of something
- the **alternative hypothesis**, $H_1$, that the proposed model is not true

We say the test is **statistically significant** if the new data gives strong evidence to support the alternative hypothesis—i.e. if the sample under scrutiny is an unlikely realization of the null hypothesis.

Let's suppose that before you saw any of the economic freedom data, you hypothesized that the true mean economic freedom score in 2017 was 4.5. You now want to use the data you have to test this hypothesis.

1. For the test described above, state the null and alternative hypotheses.

This is known as a one-sample $t$-test for a mean. In R you can use the `t.test()` function to perform the test. It needs two arguments: `mu`, the proposed value of the mean under the null hypothesis, and the vector array of data you're using to test it.

2. Use `t.test()` to perform the hypothesis test. Below is some sample code to help you:

```
#t.test(mu = 4.5, efw2017$economic_freedom)
```

The **observed value** of the test is printed at the bottom of the output—this is the sample mean or the "observed" mean of your data.

3. What is the observed value in this test? Is it close to your hypothesized value, or is it far off?

We tend to reject the null hypothesis if the new data gives strong evidence against the null hypothesis—i.e. a result that is very unlikely under the null hypothesis. The $p$-value of a test is one way to quantify this likelihood.

4. What is the $p$-value of this test?

5. How might you interpret this $p$-value? Does it say the observed result is likely or unlikely under the null hypothesis?

Conventionally we reject the null hypothesis if the $p$-value is smaller than 0.05—i.e. we reject if the likelihood of seeing a result as extreme as the observed result is smaller than 5%.

6. Decide whether to reject your null hypothesis, and state why.

You can also use $t$-tests to compare whether two samples have the same mean. This is known as a two-sample $t$-test for a difference in means.

7. Create a filtered dataset for observations from 1985. Suppose you want to test whether the average economic freedom has changed between 1985 and 2017. How might you construct a hypothesis test to check this? What would the null and alternative hypotheses be?

8. Use `t.test()` to perform the two-sample $t$-test described above. Comment on your results, and decide whether you should reject the null hypothesis you stated in q7.

9. Do the same thing as in q8, but this time test whether the economic freedom index has changed between 2015 and 2017. Again, state the null and alternative hypotheses, and comment on your results.

# VII – Relationships Between Variables

Two variables are said to be **associated** or **dependent** if there is a relationship between them.

You can use scatterplots to visualize the relationship between two numeric variables.

1. Make a scatterplot of `economic_freedom` on `business_regulation`. Use the geometric mapping `+ geom_point()`. What do you see? Are the variables associated? If so, what kind of relationship do they have?

**Correlation** is a measure of the *linear* association between two variables. A common measure of correlation is Pearson's correlation coefficient, denoted $r$, which takes a value between -1 and 1.

In R you can use `cor()` to compute Pearson's correlation coefficient between two variables.

2. Compute Pearson's correlation coefficient between the two variables you plotted in q1. What is the result? What does it imply?

You can also use the `cor()` function to create a **correlation matrix** that displays the correlations between several numeric variables.

3. Select five numeric variables from the dataset, and assign them to a new object—use the `select()` function from `dplyr`. Then apply `cor()` to the new object to generate a correlation matrix for these variables.

# VIII – Linear Models

When two variables are related, you can build a statistical model that identifies the mathematical relationship between them. When variables are linearly related, you can model them with a linear function (a straight line). This modeling technique is known as **linear regression**.

You may be familiar with the mathematical equation for a straight line, $y = mx + c$, where $m$ is the slope and $c$ is the $y$-intercept. In this notation $y$ is usually called the *dependent* variable and $x$ the *independent* variable, since $y$ is expressed as a function of $x$.

In linear regression there is slightly different terminology—but the idea is the same. A simple linear model has one **response variable** ($y$) and one **predictor variable** or **explanatory variable** ($x$). The response variable is specified as a function of the predictor.

1. First let's try to visualize a linear model. Pick two variables you found in the previous section that were demonstrably correlated with each other. If you were going to model their relationship with a linear function, which variable should you use as the predictor and which the response? Why?

2. Make a scatterplot of these two variables using `geom_point()`. Add another geometric mapping, `stat_smooth(method = 'lm')`. This will overlay the scatterplot with a straight line that is fitted using the "lm" method (linear model).

Great. The line you've fitted is the **regression line**—you can think of it as essentially a line of best fit for the data.

Mathematically we describe the functional form of the regression line as follows:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

where $\beta_0$ is the $y$-intercept and $\beta_1$ is the slope of the line. The error term, $\varepsilon$, is included to illustrate that data points don't like *exactly* on the regression line—as you can see from your plot in q2.

3. The error term describes the vertical distance from each point to the regression line. What is the sum of all the error terms in the data?

The intercept and slope, $\beta_0$ and $\beta_1$, are known as the **coefficients** or **parameters** of a linear model. The goal of linear regression is to estimate the parameters of the model. This is because a linear function is uniquely parametrized by slope and intercept. Once you have these two quantities, you can draw a straight line.

In R you can use the `lm()` function to perform linear regression and compute the coefficients. It takes two arguments:

- `formula`: the regression formula, specifying the reponse and predictor variables in the form `y ~ x`
- `data`: the dataset whose variables you are using for regression

4. Use `lm()` to compute the coefficients of the regression line you plotted in q2. Below is some sample code to help you. Report your results by calling `summary()` on the saved regression data. Are the results what you expect?

```
# reg1 <- lm(response ~ predictor, data = efwdata)
# summary(reg1)
```

Once you've computed the coefficients, you have a linear model. You can now use this model to predict the value of the response variable for a given value of the predictor. Naturally, the predicted values of a linear model are points that lie on the regression line. Predicted values are usually denoted $\hat{y}$, and you can write the predicted equation of a linear model as follows:

$$\hat{y} = \beta_0 + \beta_1 x$$

5. Predict the value of your chosen response variable for some arbitrary value of your explanatory variable.

6. When answering q5, hopefully you chose a value for the predictor that is within the range of observed data. Take a look at your scatterplot—what is the range of values on the $x$-axis? Making predictions outside this range is known as **extrapolation**. Can you think of a reason why extrapolation might be a bad idea?

Of course in reality a response variable can be influenced by a number of factors—-not just one. Thus when trying to model the behavior of a response variable, you often need more than one predictor variable to properly capture the nuance.

A linear model with more than one predictor variable is known as a **multiple regression model**.

7. From the correlation matrix you made in the previous part, choose a few variables that are correlated with the response variable–other than the one you've already used. Use `lm()` to create a multiple regression model, adding these predictors to the one you have already. Show the results.

In a multiple regression model the coefficient on any one predictor describes its effect on the response variable *while holding the other predictors constant.*

8. Is the coefficient on your original predictor (the one in the simple model above) different in the multiple model? If so, why do you think that might be?

9. Add a categorical variable (don't use country—choose one with relatively few categories, like continent) to your multiple regression model and show the results. You'll see that R treats each category as a separate predictor with its own coefficient.