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.

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've 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'. The template we've provided will produce a document in HTML format, since this is probably the easiest to work with. If you rather a pdf format (like this document) check out the course notes (chapter 1) on rendering pdf output files.

In general, for tips on R Markdown formatting and syntax, check out the course notes 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're 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 specify 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 file is in the same directory as your code file, the sample code 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

Note the `View()` command is typically used in the console—don't leave it in a code chunk as your document won't knit properly. Also, try not to use `View()` on massive 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.

2. Identify some variables in your data that are categorical, discrete, and continuous.

3. Which years are recorded in the data? Use the `unique()` function on the `year` variable, as shown in the sample code below:

```
#unique(efwdata$year)
```

4. (optional) Some additional checks you can run on your data: use `colnames()` to check the column names; use `class()` to check the data type of a particular column; use `str()` to check the structure of your dataset—this will produce a table showing each column, its data type(s), and a preview of its observations. How are different variables (e.g. continuous, categorical, etc) stored in R?

II – 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 data. 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's 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 the kind of plot you want (i.e. to geometrically map the data onto the x and y axes). The following examples will demonstrate how to use `ggplot()`.

2. Let's now visualize the distribution of 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).

```
# 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.

```
# ggplot(data = efw2017, mapping = aes(x = economic_freedom)) +
#   geom_histogram(bins = 50, aes(y = ..density..), fill = 'violet') +
#   ggtitle('distribution of economic freedom scores 2017') +
#   xlab('economic freedom index (out of 10)') +
#   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 for this—`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. Are the distributions similar across the categories?

Box and whisker plots are another way to compare the distribution of a numeric variable across different categories. 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). Use `geom_boxplot()`. 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.

8. What do box and whisker plots show that histograms don't? What do the lines on a box and whisker plot represent?

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.

9. 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?

Note how `geom_line()` draws a straight line connecting each data point. To make a smoothed line plot instead, you can use `geom_smooth()`—this will fit a smoothed line using the method of loess (a line fitting technique that uses weighted regression).

10. (optional) Make the same plot as in q9, this time fitting a smoothed line to the data.

11. (optional) Compare how economic freedom has evolved over time across several countries. Choose some countries from the dataset, and create a filtered dataset with observations from these countries only. Make a line plot similar to the ones above, this time specifying a color argument in the aesthetic mapping, e.g. `mapping = aes(x = ..., y = ..., color = country)`—this will use a different color for each country.

III – Summary Statistics

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**, $\bar{x}$, is the sum of all observations divided by the number of observations. The **median**, m, is middle value or 50th percentile of a distribution.

1. Say you had a set of observations, $x_1, x_2, ..., x_n$, which had some mean $\bar{x}$. How would you anticipate the mean changing if you multiplied all the observations by 2? What about if you added 5 to all the observations?

2. 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?

3. 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')
```

4. In the above plot, what is the area contained by the histogram to the left of the median? What does this imply?

5. Based on your results, 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**, s, is the most common. It's defined as the average distance of each observation from the mean.

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

7. How would you anticipate the standard deviation changing if you multiplied all the observations by 2? What about if you added 5 to all the observations?

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).

8. (optional) 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 dataset. 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))
```

9. (optional) 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?

IV – Tests

Summary statistics computed from sample data are known as **estimates**—specifically, ***point* estimates**—of what we're actually trying to measure, i.e. *true* mean.

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 score is 4.5—after which sample data is collected and analyzed to determine whether the initial hypothesis was indeed correct.

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 is an *unlikely* realization of the null hypothesis (i.e. if it gives strong evidence to support the alternative 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 it—i.e. if it produces a result that is very unlikely under the null hypothesis. The p-value of a test is one way to quantify this

likelihood. Speaking in relative terms, if the p-value is small, it means the observed result is *unlikely* under the null hypothesis. If the p-value is large, it means the observed value is *likely* under the null hypothesis.

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

5. Decide whether to reject your null hypothesis, and say why.

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

6. (optional) 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? Use `t.test()` to perform the two-sample t-test described above. Comment on your results, and decide whether you should reject the null.

7. (optional) Do the same thing as in q6, 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.

V – Association

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. To make a scatterplot use the geometric mapping `+ geom_point()`.

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

Correlation describes 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 `cor()` to create a **correlation matrix** that displays the correlations between several numeric variables.

3. (optional) 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.

VI – 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 (x) and which the response (y)? 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 model as follows:

$$y = b_0 + b_1 x + e$$

where b_0 is the y-intercept and b_1 is the slope of the line. The error term, e, 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 the data points to the regression line. Each point has its own error term. What is the sum of all the error terms?

The intercept and slope, b_0 and b_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.

5. Predict the value of your chosen response variable for some arbitrary value of your explanatory variable. You don't need to write code to do this—the computation should be simple enough to do by hand (just plug in your coefficients to the linear function).

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?

In reality, using a *single* variable to predict the behavior of a response is often insufficient. This is because most response variables are influenced by a number of factors, not just one—e.g. think of how many variables might influence the economic freedom score of a country.

Thus when modeling a response variable it's often useful to have *several* predictor variables—this may increase the predictive power of your model and help you control for **confounding variables**.

A linear model with more than one predictor is known as a **multiple regression model**. Functionally it has the form:

$$y = b_0 + b_1 x_1 + b_2 x_2 + ... + b_k x_k + e$$

where $x_1, ..., x_k$ are a set of k predictor variables, each of which has its own coefficient.

7. Choose a few numeric 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.

```
#reg2 <- lm(response ~ predictor1 + predictor2 + predictorK, data = efwdata)
#summary(reg2)
```

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 from q4) different in the multiple model? If so, can you think of a reason why?

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