CHAPTER 2

Descriptive Statistics

One of the purposes of statistics is to reduce a large set of data into just a few descriptive numbers — data reduction. Of course reducing data results in the loss of information, but appropriate data reduction produces just those aspects you want to measure. When reducing the data to a single number, one usually wants an 'average' number, a measure of the central tendency of the data. When given the opportunity to reduce the data to two numbers, one usually wants a good measure of central tendency *and* some measure of dispersion or spread. This chapter examines many measures of central tendency and measures of dispersion.

2.1 Measures of Central Tendency

One of the most important numbers representing a set of data is the 'typical' value, an average number in some sense. There are several different measures of averages (a.k.a. of central tendency). Each measure attempts to summarize the data with a single value, which is to best represent the entire data. The most familiar measure is the (arithmetic) mean. Others exist and are superior to the mean (in some ways). Examples include the median, the trimmed mean, and the mode.

These measures all give approximately the same result under most circumstances. However, they are all different in important points.

2.1.1 The means

There are several "mean" measures. Each of these measures attempts to summarize the data with a single 'typical' value. The reason for the many types of means is that 'average' is defined according to what the data values are used for. Thus, the average height would be an arithmetic mean, whereas an average inflation rate would be a geometric mean.

Arithmetic mean

The arithmetic mean is the mean value with which we are most familiar. It is so common that 'arithmetic mean', 'mean', and 'average' are usually used interchangeably. The formula for the arithmetic mean is

$$\bar{x} := \frac{\sum_{i=1}^{n} x_i}{n} \tag{2.1}$$

The standard symbol for the arithmetic mean is a bar over the variable. The formula in Eqn 2.1 above requires $n + 1$ steps to calculate the mean; it is very fast. This is the most important advantage of the mean. A second advantage is that a lot of procedures you will learn later in this course are more easily performed using the mean.

The drawback is that it is not *robust* to outliers. An outlier is a datum that may not have been produced by the same real-world process as the rest of the data. As such, the outlier can alter the results of the analysis to the point that your results no longer reflect reality. Being robust indicates that the measure does not change drastically in the presence of an outlier. The robustness measure, R, is the fraction of outlier data that must be present before the measure becomes unusable.

For the arithmetic mean, $R = 1/n$. We know this because adding a single (extremely large) datum to the data can change the arithmetic mean to any arbitrary value. For instance, let our pure data be $1, 1, 3, 2, 3, 5, 3, 2, 1, 5, 5$. The arithmetic mean is $\bar{x} = 33/11 = 3$. If we add an outlier datum to the data with value $x_{12} = 1e12$, the mean becomes $\bar{x} = 83333333336.08333\ldots$, and the measure is worthless.[1]

[1] The number 1e12 is scientific notation for $1 \times 10^{12} = 1\,000\,000\,000\,000$.

Geometric mean

A second 'mean' measure is the geometric mean. If all of the data are positive real numbers ($x_i \in \mathbb{R}^+$), then the geometric mean is the nth root of the product of the data elements. The geometric mean only exists as a measure when the data are all positive.

$$GM := \left(\prod_{i=1}^{n} x_i \right)^{1/n} \tag{2.2}$$

Sometimes, the number of elements in the sample is so large that the computer will give an overflow error when you attempt to calculate the geometric mean. In such cases, an alternate formula may be of use. Remembering our logarithmic identities, we can see that the geometric mean can be calculated by $GM = \exp\left[\frac{1}{n} \sum \ln(x_i)\right]$; that is, the log of the geometric mean is also the arithmetic mean of the logarithms of the data. The number of calculations needed for this formula is on the order of $2n$, whereas the number of calculations needed for Eqn 2.2 is on the order of n — the same as for the arithmetic mean.

The geometric mean is never greater than the arithmetic mean.[2] As with the arithmetic mean, $R = 1/n$. The only times when the geometric mean is used in the research is when the underlying data values represent a relative (or percent) increase or decrease, such as the consumer price index (cpi) or interest rates.

Harmonic mean

The final mean measure we will be examining is the harmonic mean. The harmonic mean is also defined only when the data values are positive real numbers ($x_i \in \mathbb{R}^+$). It is appropriate when averages of several rates are needed.[3]

$$HM := \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} \tag{2.3}$$

One interesting relationship among the three means is that, when the measures are defined,

[2] They are equivalent only when all data values are equal.

[3] With this said, I have yet to see it in the social science literature. It may be in the engineering literature, but I am not sure. I am sure someone will let me know eventually. Actually, resisters in parallel operate this way: If my electronic circuit has 10 resisters connected in parallel, the effect would be the same as if I connected 10 resisters with resistance equal to the harmonic mean.

$AM \geq GM \geq HM$.[4] Its calculation speed in Eqn 2.3 is the same as for the arithmetic mean. Additionally, $R = 1/n$.

2.1.2 The median

One thing that all of the mean measures have in common is that they all are sensitive to outliers. For each, $R = 1/n$; that is, one single outlier can render the entire measure of central tendency worthless. The median is the most robust measure of central tendency available, $R = 1/2$. The reason is that the means depend on the data values equally, whereas the median only depends on the *rankings* of the data. Thus, changing a single value will only change the median if a below-median value becomes above-median, or vice-versa.

The median, $\tilde{x}$, is the middle-ranked number.[5] There is no nifty formula for the median, and that is its drawback. Whereas the means can be calculated in $n + 1$ flops,[6] One must sort the data before deterging the median — a relatively expensive step (on the order of $n \ln n$ for the best algorithms). For a sample of size 1,000,000, the mean takes 1,000,001 flops; the median takes at least 13,000,000.

Because of the calculation costs, much in statistics is based on the mean and not the median. This is slowly changing, since computing power is cheap. On a typical laptop computer, the system time required to execute a mean statement on a hundred million data (1×10^8) is approximately 0.33 seconds. The time to execute a median statement on the same dataset is 4.77 seconds. While the times are both small, the relative difference is astounding and is the reason so much in statistics is based on the mean.

2.1.3 The mode

I am including this for completeness. It is used only when attempting to give a 'typical' value for nominal data. It has no meaning when the data is continuous. It has little meaning when the data is ordinal.[7] It is important, however, when determining null models in the future.

[4] Alright, so it is not that interesting.

[5] Yes, that is a tilde above the x. Alternative symbols for the median include '*Med*', '*M*', and '*m*'. These last two may also represent means, therefore if you come across such a symbol, double-check what the author intends by that symbol.

[6] A flop is a floating point operation, a single calculation step.

[7] Nominal data is categorical, but without an inherent ordering. Ordinal data is categorical, but does have an inherent ordering that makes some sense. Continuous data is not categorical.

Measure	R function	Value
Arithmetic Mean	`sum(gdpcap)/length(gdp)`	15,320
Geometric Mean	`exp( (1/length(gdpcap)) * sum(log(gdpcap)) )`	7503
Harmonic Mean	`length(gdpcap) / sum(1/gdpcap)`	2937
Median	`median(gdpcap)`	8900
Mode	`Mode(gdpcap)`	900

Table 2.1: *Table of the various measures of central tendency, rounded, for the* `gdpcap` *dataset. Where a simple formula for the measure exists, it is provided to let you get a feel for how to do math in* `R`*.*

2.1.4 R and measures of central tendency

As with any good statistical package, `R` has the ability to calculate most of the above measures of central tendency. In fact, most of the functions are the same as their names.

Let us start by loading data from an external file. As the data I am using is stored in the file `gdpcap.csv`, we will use the `read.csv()` function. Assuming that the file is in the working directory, the line is `data <- read.csv("gdpcap.csv", header=TRUE)`

This command causes `R` to read in the csv file `gdpcap.csv` and store it in the variable `data`. The parameter `header=TRUE` specifies that the first row of the data consists of the variable names. The alternative (`header=FALSE`) specifies that the first row is data and that `R` needs to create default variable names `V1`, `V2`, ...

Typing `data` allows us to see the entire set of data. As we can see, the data file contains two variables — a variable for the state's name (`state`), and a variable for its GDP per capita (`gdpcap`). The former is a nominal variable, the latter is continuous.[8] We can graph a histogram of the data to help get a feel for it (`hist(data$gdpcap)`. As we see (Figure 2.1), the data is not symmetric; it is skewed right.[9] Because of the noticeable skew, the median would probably be a better measure of a 'typical' gdp per capita.[10] Even so, let us find all of the measures of central tendency for this data.

Table 2.1 gives the values calculated for each of the measures of central tendency discussed in this chapter. There are a few things I would like to point out about the table. First, note that the measures of central tendency do not provide the same (or similar) results. This is to be expected because of the skewed nature of the data. We can expect similar results only when the data is

[8] Technically, I suppose the `gdpcap` variable is ordinal, since we are rounding the values to the nearest 100, but it would be better to treat it as continuous.

[9] The direction of skew is the same as the direction of the *long* tail.

[10] As a rule of thumb, any income measure should use the median.

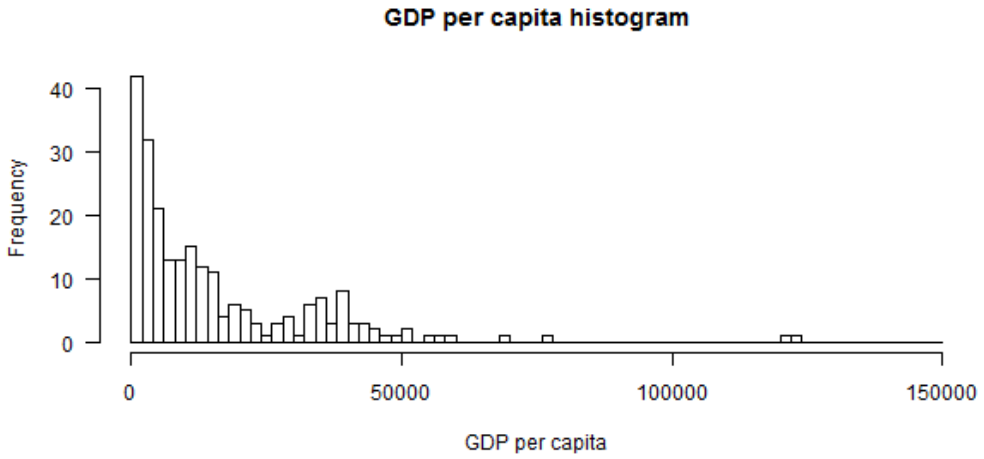

Figure 2.1: *A histogram of the gross domestic product per capita for the states of the world.*

symmetric and has a very low level of dispersion. The three means are equal only when the data consists of a single repeated value.

The second thing to notice are the formulae for the arithmetic, geometric, and harmonic means. The arithmetic mean has its own function, `mean()`, in the base package. The other require additional libraries for a dedicated function.[11] However, since that package is not contained in the base distribution, it is best to just write out the function. Also note that the formula used for the geometric mean is the alternative formula. When Eqn 2.2 is used as a basis for calculating the geometric mean, the product frequently causes a numeric overflow, rendering it useless.

Finally, the modal value was found using another library — the '`prettyR`' library. That library is also not with the standard distribution. As the mode is not of any great use for us, this may be the last time you see its calculation. When you need it in the future, you will discover a better method to calculate it.

If we compare the arithmetic mean and the median, we see that the mean is much larger than the median. This is because the data is highly skewed right. Were the data to be symmetric, then the mean and median would be very close. Were the data highly skewed left, the median would be larger than the mean. This allows us to get a feeling for the level (and direction) of skew in the data

[11] The `psych` library contains functions for the geometric mean (`geometric.mean`) and the harmonic mean (`harmonic.mean`).

— something that may be useful in the future (see Section XsomethingX for a better discussion of the skew).[12]

2.2 Measures of Dispersion

If we can only use one number to summarize our data, a measure of the center is our best choice. If we have the luxury of using a second number, we tend to choose a number describing the spread (or dispersion) of the data. We choose such a number because it also tells up how good that measure of the center represents each datum: Larger spreads indicate the average does not represent each data value well; small spreads indicate the average represents each data value well.

One easy dispersion measure to calculate is the range of the data. It uses just two data values — the minimum and the maximum — it ignores all other values (and data points). As a measure, it is of little use. However, it is useful to check that you have typed in the data correctly. If I had typed in the gdpcap data incorrectly and gave Norway a GDP per capita of -15000 because of a finger-slip, I could detect that from looking at the range (and by looking at the minimum value).

As with measures of central tendency, there are several measures of dispersion. They can be divided into two groups: those based on the arithmetic mean, and those based on the median.[13]

2.2.1 Mean-based measures of dispersion

There is one (two, depending on how you count them) measure of dispersion based on the mean – the standard deviation (and the variance). The standard deviation is, in some sense, the average distance the values are from the mean. The units of standard deviation are the same as the units for the underlying measurement. For our data on gross domestic capita per capita, the standard deviation ($s = 18,188$) has units of dollars.

$$s := \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{2.4}$$

The formula used to calculate the standard deviation is Eqn 2.4. To calculate the standard deviation, first, calculate the deviances of each value from the mean ($x_i - \bar{x}$). Square those deviances.

[12] This section is forthcoming.

[13] There are measures based on the mode, the geometric mean, and the harmonic mean available, but the literature using them is sparse, so I am choosing to skip those.

Add those deviances together. Divide that sum by one less than the number of data points. Finally, take the square root of that quotient.

Here is the logic behind the standard deviation. The basis is the deviance of each data value from the mean. These deviances are squared because, if they were not, they sum to zero — always. The average of these squared deviances is then taken. Note that we divide by $n - 1$ instead of by n.[14] Were we to stop at this average, we would have the variance, s^2. However, we go the next step and return the measure to the originating units to get a measure of dispersion that is directly comparable to the underlying data.[15]

In `R`, the functions used to produce the standard deviation and the variance are `sd()` and `var()`.

2.2.2 Median-based measures of dispersion

The measure of dispersion based on the median is the interquartile range (IQR). Before we define the IQR, we have to define the quartiles. We already know that the median divides the ranked data into two halves. Each datum is either above the median or below the median. Quartiles divide the ranked data into fourths. The first quartile, Q1, is the value where one quarter of the data has a lower value. The second quartile, Q2, is the value where two-quarters of the data has a lower value. The third quartile, Q3, is the value where three-quarters of the data have a lower value. Note that the median is the second quartile.

$$IQR := Q3 - Q1 \tag{2.5}$$

The interquartile range is the difference between the third and first quartiles (Eqn 2.5). Fully one half of the data can be found between the first and third quartiles.

These quartiles are also used in calculating the boxplot (Figure 2.2). In a boxplot, there are several data features displayed. Thick line is the median value. The thin lines to either side of it, making up the left and right sides of the box are Q1 and Q3, respectively. The interquartile range is

[14] There is a long and complex reason why we divide by $n - 1$ in lieu of n — the mathematics requires it. A short, but unhelpful answer is that we are still dividing by the degrees of freedom (see Section XsomewhereX for a discussion on degrees of freedom). Since one degree of freedom was used in calculating the mean, we only have $n - 1$ remaining. Finally, the shortest answer, and still ultimately unhelpful, is we do it because we said so.

[15] There is also something called coefficient of variation, where the standard deviation is divided by the mean. The advantage to this is that one can compare them across different variables to answer such questions as Which variable is more disperse?

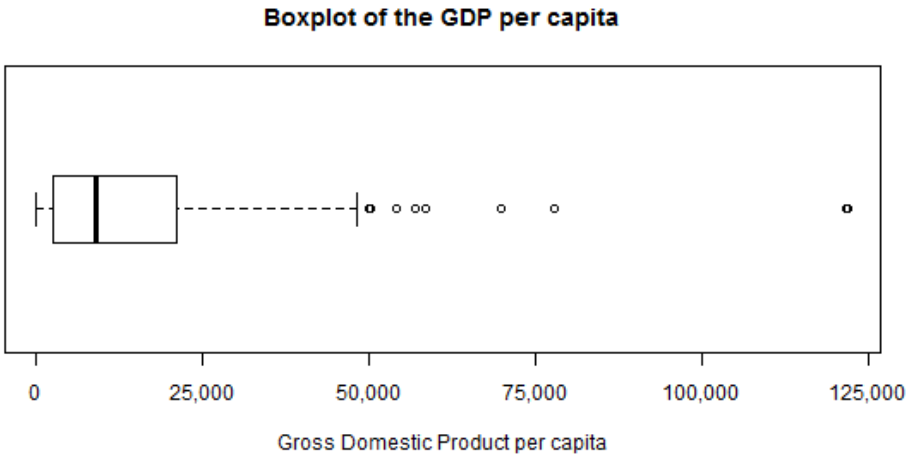

Figure 2.2: *A boxplot of the gross domestic product per capita for the states of the world.*

the distance represented in the box. The next pair of lines (left-most and right-most vertical lines) are the either the minimum and maximum values or the largest data values inside the 'fence.' The fence is used to help determine if the data set has any outliers. For usual applications, the upper fence is $\tilde{x} + 1.5 \times IQR$, and the lower fence is $\tilde{x} - 1.5 \times IQR$. The left-most vertical bar is either the minimum value or the smallest value greater than the lower fence. Likewise, the right-most vertical line is either the maximum value or the largest data value smaller than the upper fence. The last feature of a boxplot are the outlier marks. Each outlier (those data values beyond the fences) is marked with a dot corresponding to its value.

In this boxplot (Figure 2.2), we know the left-most vertical line is the minimum value as there are no outliers to the left. The right-most vertical line is the largest value less than the upper fence. The dots to the right of that are outliers, values beyond the upper fence. There are nine (9) outliers in this data set: Liechtenstein ($122,100), Qatar ($121,700), Luxembourg ($78,000), Bermuda ($69,900), Norway ($58,600), Jersey ($57,000), Kuwait ($54,100), Singapore ($50,300), and Brunei ($50,100).[16] Also note, as with any boxplot, half of the data values fall within the box, half of the data values fall below the medial line, and half fall above the medial line.

[16]If the resolution is poor, you may only see seven dots (outliers). However, there are two values near 150,000 and two values near 50,000. By the way, the source of this data is the CIA — a good source for such information. Also by the way, the United States comes in at 11^{th} with a GDP per capita of $46,400. The Faroe Islands form $Q3$ with a GDP per capita of $48,200.

Measure	R function	Value	Units
Range	`diff(range(gdpcap))`	122000	dollars
Standard Deviation	`sd`	18188	dollars
Variance	`var`	330816287	dollars^2
IQR	`IQR`	18600	dollars
Q0 (minimum)	`min(gdpcap)`	100	dollars
Q1	`quantile(gdpcap, 0.25)`	2600	dollars
Q2	`quantile(gdpcap, 0.50)`	8900	dollars
Q3	`quantile(gdpcap, 0.75)`	21200	dollars
Q4 (maximum)	`max(gdpcap)`	122100	dollars

Table 2.2: *Table of the various measures of dispersion, rounded, for the `gdpcap` dataset. The values of the five quartiles is included for completeness, even though they are not measures of dispersion in themselves.*

2.2.3 R and measures of dispersion

Before we begin ending this chapter, let us see how R handles measures of dispersion. As we already have the `gdpcap` data in memory, let us calculate the various measures of dispersion on it. Table 2.2 provides the measure name, the R function, the value calculated, and the units of the measure. Note that the only measure of dispersion that is not in the same units as the underlying data is the variance.

Percentiles

In R, there is a handy function used to calculate any or all of the percentiles. A percentile is to 100 as a quartile is to 4, and as a median is to 2 — it breaks the ranked data into 100 equal groups. Thus, if p is the 90th percentile, then we know that $x_i < p$ for 90% of the data.[17] The function is the `quantile` function. If you only pass it the data, it will produce the five quantiles (Q0 – Q4). If you pass it a value (or a vector of values), it will give percentiles corresponding to the values you provide. Thus, the 90th percentile can be calculated using `quantiles(gdpcap, 0.90)`. To calculate multiple percentiles at once, pass the function a *vector* of values; for instance, the 5th, 10th, 90th, and 95th percentiles can be calculated using `quantiles(gdpcap, c(0.05, 0.10, 0.90, 0.95))`.[18]

[17] When we discuss confidence intervals and bootstrapping, we will see this again.

[18] I need to discuss vectors, but there is not much interesting about them. In R, vectors can be created using the `c()` constructor function. Thus, `c(0.05, 0.10, 0.90, 0.95)` is a vector of four numeric values. There are other ways of creating vectors — the colon method and the repeat method. If we wish to create a vector of the numbers between 1 and 100, inclusive, we can use the colon operator as such `1:100`. If we wish to create a vector of length 100 of only the number

2.3 A sample univariate analysis

Let me now present the script that produced the analysis and the graphs in this chapter. It offers a nice introduction to extending several of the graphing functions, as well as the `getwd()` and `setwd()` functions, which may become very important to your future projects.

First, in the following listing, note that there is no real end of line character that signifies the end of the command line. `R` is flexible to the point that it will continue accepting characters until the grouping punctuation, usually parentheses, balance out before it executes the line. This means *extra* white space (like spaces and tabs) is usually ignored. It also means that we can split a particularly lengthy command across several lines to ease *our* interpretation.

```
1  ## Prologue
2  library(prettyR)
3  data <- read.csv("gdpcap.csv")
4  attach(data)
5
6  # Measures of central tendency
7  mean(gdpcap)
8  median(gdpcap)
9  Mode(gdpcap)
10 exp( (1/length(gdpcap)) * sum(log(gdpcap)) )     # geometric mean
11 length(gdpcap) / sum(1/gdpcap)                   # harmonic mean
12
13 # Measures of dispersion
14 diff(range(gdpcap))
15 sd(gdpcap)
16 var(gdpcap)
17 quantile(gdpcap, 0:4/4)
18 IQR(gdpcap)
19
20 # Histogram
21 png("hist-gdpcap.png", width=600, height=300)
22 hist(gdpcap, main="GDP per capita histogram", xlab="GDP per capita",
23     breaks=0:75*2000, las=1)
24 dev.off()
25
26 # Boxplot
27 png("bp-gdpcap.png",width=600,height=300)
28 boxplot(gdpcap, las=1, horizontal=TRUE, xaxt="n",
29     pars=list(xlab="Gross Domestic Product per capita",
30     main="Boxplot of the GDP per capita",
31     boxwex=0.5) )
32 axis( 1, at=0:6*25000,
33     labels=c("0", "25,000", "50,000", "75,000","100,000","125,000",
34     "150,000") )
35 dev.off()
```

5 (happens more often than you know), we can use the repeat method as `rep(5,100)`.

There are a few things to point out in this script listing. The first line tells `R` that you are requesting a non-standard function library to supplement the functions already included in `R`. Here, we are loading the library `prettyR`, which gives us the `Mode()` function.

The next line loads the data, `gdpcap.csv`, into the variable `data`.

★ **Notice**: *Always load a dataset into a variable; it allows you to actually use the data rather than just displaying it.*

Now, here is the big question: Where is the data loaded from? That all depends on the current working directory. `R` will always load from the current working directory. If you started `R` in the default manner, the default directory is not your project directory. [19] The most important thing to mention is the issue of the working directory.

2.3.1 The working directory

As with program, `R` must know where to find the files you reference. By default, `R` assumes all referenced files are in the working directory. If you are running `R`, there is a good chance that the working directory is `C://R-2.11.1/bin`, which is not your project folder. Here are three good solutions.

First, if you are running `R` from a USB drive, then you will need to include the following single line in all of your scripts:

```
setwd("../../project/")
```

This command sets your working directory (hence, `setwd()`) to a folder called `project` located in the highest level of your USB drive. This line assumes two things. First, that you installed `R` to the highest level of the USB drive, which is default behavior. Second, that your project folder is at the same level. If your project folder is contained in a POLS6123 folder, then the appropriate line would be `setwd("../../POLS6123/project/")`.

Second, if you are running `R` from your computer's hard drive (if you installed it to your computer), then you have a couple options. The first is similar to the previous solution. Let us say that the absolute path to your project folder is `C://POLS6123/project/`. Then, the line

[19] To check the current directory, use the function `getwd()`.

you would insert into your scripts would be `setwd("C://POLS6123/project/")`. The second option is to save an `R` workspace to your project folder and not worry about adding a line to your scripts.

To save an `R` workspace, follow these steps:

1. Open `R`.
2. Once you see the Console window, click on `File|Save Workspace...`
3. Save it in your project file.
4. Close `R`.
5. Finally, go to your project folder and click on the `.RData` file.

Warning: *When you save the workspace, `R` saves all current register information, which includes the current folder. Unfortunately, this last method only works when `R` is fully installed on the computer. As such, it will not work if it is only located on your USB drive.*

2.3.2 Attaching the data

Now, we have the data stored in the the variable `data`. This data variable *itself* contains two variables: `state` (character) and `gdpcap` (numeric). At this point, were we to type `data` in the console, `R` would display the entire contents of the variable `data`. Were we, however, to type `gdpcap`, we would receive an error message; there is no variable named `gdpcap` defined. But wait, how do we access the variable `gdpcap` that is in the `data` variable (which is what we really want to do)? There are three ways. The first is useful if you are only using this data once or twice. The second two are more useful if you are using it more often (like us).

In the first case, you merely have to specify that the variable is a part of the `data` variable as such: `data$gdpcap`. Thus, to find the arithmetic mean of the gdp per capita in the world, we could type `mean(data$gdpcap)`.

The second way is to specifically create new variables matching the `data` variables:

```
state  <- data$state
gdpcap <- data$gdpcap
```

I will frequently do something similar to this in my initial analyses when the variable names are long; I will rename the `x`, `y`, or such.

Finally, the third way uses the `attach()` function. When you execute the `attach(data)` line, you are quickly doing the second way (above) without the extra typing. Once you attach the data, the variables in the data are available without having to specify that you are asking for the variables in the `data` variable.[20] If you ever want to undo the `attach()` function and use those variable names for something else, you will need to issue a `detach()` command.

2.3.3 Univariate functions

The next five lines calculate and display the five primary measures of central tendency discussed in this chapter: arithmetic mean, median, mode, geometric mean, and harmonic mean.

After the measures of central tendency come the measures of dispersion. The first line calculates the range of the data. The next two are the standard deviation and the variance. The next line produces the five quartiles (Q0, Q1, Q2, Q3, and Q4). The median is the same as Q2. The minimum and maximum are the same as Q0 and Q4, respectively. The fifth line calculates the interquartile range, Q3-Q1.

Finally come the two graphs shown in this chapter: the histogram and the boxplot. Each of the two graphing functions is preceded by a `png()` function and succeeded by the `dev.off()` function. The former tells `R` to save the graph to the working directory as the filename in quotation marks, with the provided width and height, in pixels. The latter closes that 'window' so that all output goes to the console window.

The histogram function only requires a numeric variable. If that is all that you provide, it will produce a standard histogram. The histogram function also takes several optional graphical parameters (most of which are standard across all graphics in `R`/). The optional parameters I provided the function altered the displayed graph. The `main` parameter specifies the content of the main title of the histogram. The `xlab` parameter specifies the label on the x-axis. The `breaks` parameter specifies the bins in the histogram. Finally, the `las` parameter specifies the orientation of the axis numbers. A '1' forces all axis numerals to be horizontal.[21]

[20] The variable `data` is actually a data frame — a type of variable that is very useful for data analysis. `R` actually has several data types available, including scalar, vector, matrix, and list. Searching the `R` help files may be worth your time if this interests you.

[21] For additional options, see the `hist` and the `par` help pages.

The boxplot is quite different from the standard graphic in R, due to its different requirements. Again, it only requires the name of the numeric variable. The options change the default behavior. The `horizontal` toggle specifies that the boxplot is to be graphed horizontally, instead of the traditional vertical orientation. The boxplot is different in that it requires the standard graphical parameters to be included in a `list()` function.[22] Finally, I included the `xaxt="n"` parameter, which turns off the plotting of the x-axis. I decided I would be able to get better results by specifying the tick marks on the x-axis myself, which I do in the following `axis()` line.

The `axis()` function allows you to take direct control of the plotting of any of the four axes available. Here, I specify that the x-axis (the `1`) has seven (7) tick marks, one every 25,000 dollars starting at zero.[23] Finally, the `labels` parameter specifies what I want displayed at each of the values provided in the `at` parameter.

2.4 Conclusion

In this chapter, we discussed reducing the entire data to one number (measure of central tendency). We then provided a number which described the goodness of that measure of central tendency as a description of each data value. We then showed two graphs, a histogram and a boxplot, which describes the data in a graphical manner.

Finally, an R script was presented and explained. The importance of the working directory was emphasized, as were three ways of ensuring that the working directory is your project directory.

The next chapter will consider the ubiquitous t-test, its theory, and its uses (and under-uses) in statistics.

2.5 Extension

Here are a few things you can do to practice (and extend) what you read in this chapter.

[22] The `list` function merely allows several parameters to be passed at one time. The boxplot command is rather old in that it does not natively take advantage of the many graphical parameters available in R; it requires an external function to deal with them. None of this really matters to us users, as it is all done transparently.

[23] The `at` parameter designates the locations along the axis at which it is to display the labels (provided later). The value given to `at` needs to be a vector of values, since you are giving it more than one value. There are three main ways of specifying a vector of values in R (see page 2.2.3). In this case, I am specifying a vector of seven values using the colon operator. The vector contains the values 25000 times each number between 0 and 6: $0, 25000, 50000, 75000, 100000, 125000, 150000$.

1. Write a script that loads the `gdpcap.csv` dataset into your computer, calculates all of the measures of central tendency and measures of dispersion discussed in this chapter.

2. Write a script that calculates all 101 percentiles of the `gdpcap` data set.

3. Write a script that produces the boxplot in Figure **??**. Make sure you label the axes and create the title. Statements and parameters you may want to examine include `boxplot`, `xlab`, `main`, and `axis`.

2.6 R functions used

This section provides a list of the `R` functions and statements used in this chapter. It may be useful. Let me know. Assume that `x` is a vector of data values as `gdpcap` has been in this chapter.

File functions

getwd() This function, with nothing in the parentheses, will display the path of the current working directory. This will be important if you run `R` from your USB drive.

setwd(path) This function sets the current working directory to the folder specified in the `path` parameter. The path can be absolute or relative. If absolute, it will be of the form `E://project/`. If relative, it will be relative from the current working directory, usually the `R/bin/` folder. If `R` is run from a USB drive, and if your project directory is (for instance) `E://project/`, then `setwd(../../project/)` will change the working directory from the default to your project directory.

read.csv(filename) This returns the contents of the csv file `filename.csv`. It assumes that the first row of the csv file contains the variable names. If this is not true, then use the toggle `header=FALSE`. If you use `read.csv(filename, header=FALSE)`, then the default variables are `V1`, `V2`, ... Since this returns an object, not just a single value, you should store the contents in a variable. Hence, you will often see me use `data <- read.csv(filename)`. By the way, make sure you enclose the filename in quotation marks.

attach(data) This attaches a data frame so that its variables are more easily available for use in the analysis.

detach(data) This is the opposite of the `attach()` function.

Analytical functions

mean(x) This calculates the arithmetic average of x.

median(x) This calculates the median (or Q2) of x.

Mode(x) This calculates the modal value of x. Note that this function is a part of a little-used library `prettyR`. You will have to install and load `prettyR` if you wish to use this to calculate the modal value. Do not use the mode(x) function; it returns the type of object x is, not the modal value.

sd(x) This returns the standard deviation of x.

var(x) This returns the variance of x, which is the square of the standard deviation.

IQR(x) This returns the interquartile range, $Q3 - Q1$ of x.

max(x) This returns the maximum of x.

min(x) This returns the minimum of x.

quantile(x, p) This returns the p-th percentile of x. Some valuable percentiles are p=0.50, the median; p=0.25, which is Q1; p=0.75, which is Q3; p=1.0, which is the maximum; and p=0.0, which is the minimum.

range(x) This returns two values: the minimum value and the maximum value. Thus, range(x)[1] is the minimum value and range(x)[2] is the maximum value.

diff(x) This returns the difference between the elements of the vector. That is, if we represent the difference by Δ, the $\Delta_i = x_{i+1} - x_i$. This function is usually used in time series, but I used it here to calculate the range of the data.

Graphics functions

hist(x) This returns a standard histogram of the data. The produced graph is customizable (as are all graphs in `R`).

boxplot(x) This returns a standard box (and whiskers) plot of the data. The produced graph is customizable (as are all graphs in `R`).

Vectors

c(·) This returns a vector of elements you included between the parentheses. Thus, if you want to create a vector of values 1, 2, 3, 2, 3, 1, and 2, you would use `c(1, 2, 3, 2, 3, 1, 2)`.

rep(n, times) this returns a vector of length `times` filled with the value `n`. Thus, to create a vector of length 50 of the value 1, use `rep(1,50)`. This also works with letters: `rep("asd",20)` gives a vector of length 20 with each element being the string `asd`.

a:b This creates a vector consisting of the numbers between `a` and `b`. Thus, `1976:2016` is a vector of the integers between 1976 and 2016. To make this more useful, we can perform operations on the vector to create vectors of different spacings. For instance, `0:6*50000` produces the vector 0, 50000, 100000, 150000, 200000, 250000, and 300000.