Introduction

MTH 214 has as part of its learning objectives an introduction to concepts of computing – compuational literacy is the goal, which will be taken to mean a familiarity with the following ideas: using commands to interact with a computer, the use of syntax to express these commands, the use of variables and assignment, the use of control structures to guide program flow, an understanding of data structures necessary to perform the task at hand. Over the course of the semester we will learn and use these concepts to do problems arising in statistics.

We will use the statistical software R for our compuations. This is freely available software (`www.r-project.org` that runs on Windows, Mac and Linux. The R software provides an environment for statistical work. The interface is a command interface – not a point and click interface.

The R software is used by several large companies (e.g., Google, the New York Times, ...), numerous academics and students around the globe.

To start R on a windows machine one double clicks the desktop icon that is installed as part of a standard installation. (For other operating systems, R is started in a manner as other software on the machine.)

R begins with a *command line prompt* at which we can type commands, e.g.:

The output contains a leading [1]. This is because R works with data which usually has more than one value. This [1] refers to the first value returned.

```
> 2 + 2

[1] 4
```

To achieve the output, `[1] 4`, we type the command and the the enter key. Behind the scenes this sends the command `2+2` to R to process. R can have troubles parsing the command (try `2_2` to see what happens), troubles evaulating the command (try `x`), or no trouble in which case R prints its response.

Question 0.1. Try it out. Find the following values using R not a calculator:

1. $3+5$
2. $123 \cdot 321$

3. 123^2
4. $12/(3+4)$.

Data

We will work with data in this class. In statistics a simple univariate data set is denoted:

$$x_1, x_2, x_3, \cdots, x_n.$$

In R data sets usually contain more than one variable. To enter such data sets in there is one main way (and several others). This involves *combining* the numbers together using the *function* `c`. For example, to enter the first few prime numbers we have

The `c` function for combining data is one of the most useful, hence its short name.

```
> c(2, 3, 5, 7, 11, 13)
[1]  2  3  5  7 11 13
```

Assignment

The values are echoed back – but then forgotten! To store the variables requires us to *assign* a name to the values For example, we assign the primes above to the variable `primes`:

Assignment in R is done using the left arrow operator. Assignment stores the values in memory in a manner that allows them to be referenced through the variable name.

```
> primes <- c(2, 3, 5, 7, 11, 13)
```

Nothing prints, but the values have been kept. To see what is referenced by a variable, one types its name at the command line, causing the values to be printed:

```
> primes
[1]  2  3  5  7 11 13
```

Question 0.2. A data set of text message lengths contains the following values. Store them in the variable `calls`

20, 18, 100, 8, 6, 25, 32

Question 0.3. A data set on IQ scores contains the values

110, 120, 115, 105, 140, 95, 110, 135, 100

Store these values in the variable `IQ` – case is important

IF YOU MAKE A TYPING ERROR in R, or simply wish to reuse a command, you can use the up arrow to scroll through previous commands. Once you found the one you want, you can edit the command by using the left and right arrows to move through the command. Unfortunately, with the windows interface, you can not use the mouse to move to a point in the command to edit.

Applying a function

We will see other ways to define data, but before doing so, we want to be able to do something with our data. In R we use *functions* to do this

A function in R is a command that takes some arguments and returns a value after performing some action. Thinking of the arguments as the domain and the output the range and you have the analog of a mathematical function. However, in statistical use it is better to think that the function finds some value from a data set that we want.

To use functions requires a few steps:

1. Knowing the name of the function
2. Knowing what arguments the function expects (in the simplest cases this is a data set, but for some functions may be nothing, and for others may require more than one argument. For example, plotting a data set may require extra arguments to specify attributes of the plot.)
3. Being able to interpret the functions output.

For new users the first step – knowing the name – is the most daunting part. This requires knowing something that you haven't yet learned!

Before beginning with the graphics functions to produce the standard plots, we show how to find an average using R.

An average of a data set is simply the sum of the numbers divided by the number of data points. In statistics, "the average" is referred to as the mean, and the corresponding R function is also `mean`. To apply this function we type its name, **a pair of parentheses**, and any arguments to the function inside the parentheses. To find the mean of the primes, we have

```
> mean(primes)

[1] 6.833333
```

That was so easy, lets see a few other functions in practice– whose action you can infer from the function name:

```
> max(primes)

[1] 13

> min(primes)

[1] 2
```

Function	action
`c`	combine values
`range`	range of data
`median`	middle value
`IQR`	Inter quartile range
`sd`	standard deviation
`summary`	summary of object
`length`	number of elements
`names`	names attribute
`str`	short summary
`head`	first few items

Table 1: Some common R functions and their action.

```
> length(primes)

[1] 6
```

Question 0.4. What do the functions `max`, `min` and `length` do?

Question 0.5. Find the average value in the `calls` data set you entered previously.

Question 0.6. Find the maximum and minimum value in the `IQ` data set you entered previously.

The function `stem` will print out a stem and leaf plot for a data set. For example

```
> stem(primes)

  The decimal point is 1 digit(s) to the right of the |

  0 | 23
  0 | 57
  1 | 13
```

Question 0.7. Make a stem and leaf plot of the `IQ` data set

MORE THAN ONE ARGUMENT, NAMED ARGUMENTS
Some functions have several different possible arguments. To learn about the possible arguments one can read the help page for a function. The function `help` will open this page. For example, try `help("mean")` to see what happens (a shortcut is `?mean`). In looking at that you see the mean has arguments

mean(x, trim = 0, na.rm = FALSE, ...)

The help page mentions something about "Default S3 method:" we ignore this important R feature until later

The mean has 3 main arguments: `x` for the data; `trim` to indicate if a *trimmed mean* should be computed, and `na.rm` to trim out any values that are not available. A trimmed mean is a way of reducing the sensitivity of the mean to large and small values. Basically, if this value is given as 0.2, then 20% of the data from the top and bottom is thrown out before averaging. This is kind of what is done when an instructor drops the lowest – and highest – test scores before computing a final grade.

To perform this trimming, the argument must be specified. We use the syntax `name=value` to do so:

```
> mean(primes, trim = 0.2)

[1] 6.5
```

R will match arguments by name, then position. Here just `mean(primes, .2)` would have worked, but it is clearer being more verbose.

Question 0.8. Find the mean of the IQ data set after trimming 20% from the top and bottom. Compare to the mean without doing so.

Graphics functions

There are a handful of basic statistical graphics. R contains many different ways to produce thees graphics. Some prettier than others. The basic rule is the prettier ones are a tad harder to use. For now, we will use the base graphics.

Graphic	Base	lattice	ggplot2
dotplot	`stripchart`	`dotplot`	`qplot`, `geom=point`
histogram	`hist`	`histogram`	`qplot`, `geom=histog`
boxplot	`boxplot`	`bwplot`	`qplot`, `geom=boxplo`
qqplot	`qqplot`	`qqnorm`	`qplot`, `stat=qq`

Table 2: Functions for producing various graphics for the 3 main types of graphics engines. The Base graphics are very flexible, but not as pretty. The Lattice graphics are great for multi-variate panel displays. The ggplot2 graphics are prettier and relatively easy to produce.

dotplots

The basic dotplot is an excellent graphic for small, numeric data sets. They can be produced with the function `stripchart`. E.g,

```
> stripchart(primes)
```

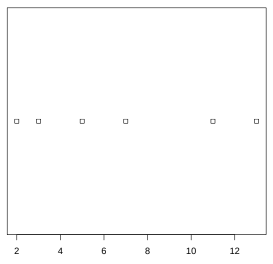

Figure 1: The basic dotplot produced by `stripchart`

The `stripchart` function does a pretty bad job of rendering the graphic. When data is repeated, the points overplot. To avoid this, the argument `method="stack"` is specified.

histograms

The histogram is a key statistical graphic for displaying numeric data, when there are a moderate to large number of data points. The center, spread and shape or the distribution is easy to see from the histogram.

```
> hist(primes)
```

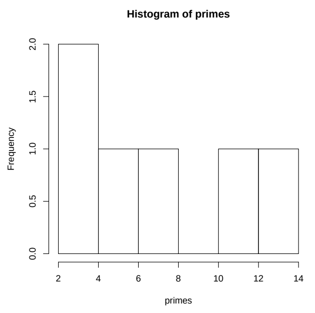

Figure 2: A basic histogram

Specifying the bins to use is done through the breaks argument.

boxplots

The boxplot is an excellent graphic for *comparing* distributions. Each boxplot succinctly shows the center, spread, range and skew of a distribution in a graphic that lends itself to side-by-side comparisons. The boxplot function produces a single boxplot, as used below.

```
> boxplot(primes)
```

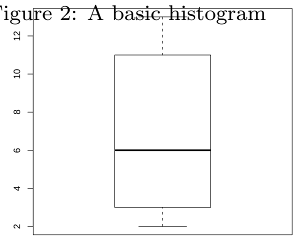

Figure 3: A basic boxplot

quantile-quantile plots

The quantile-normal plot is used to graphically compare a distribution of values to some other distribution of values – usually the theoretical “normal” distribution.

```
> qqnorm(primes)
```

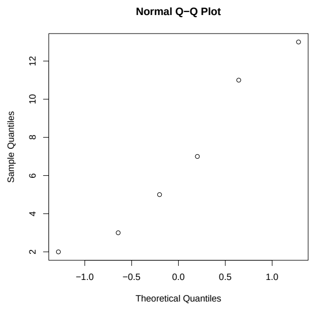

Figure 4: A basic quantile-quantile plot

Question 0.9. For the calls and IQ data sets produce each of the 4 graphics: a dot plot, histogram, boxplot and quantile-normal plot.

Built-in data sets

R can be extended by its users. This is one of the reasons R has gained wide spread use. R does this by allowing users to create “packages” which a user can a) download then b) load into their R session. Some packages are standard with R some are not and must be downloaded. This is actually quite easy, as there are functions to do so within R. Anyways, many packages have build in data sets. These do not need to be typed in! For example, the rivers data set is already there:

The rivers data set lives in the datasets package, which is loaded by default when R is started.

```
> mean(rivers)

[1] 591.1844
```

Other data sets are in packages which must be loaded first. To load a package, the `library` function is used. The `MASS` package comes bundled with R but is not loaded by default. To do so, we issue this command:

```
> library(MASS)
```

Now the data sets are available. For example, the `deaths` data set lists "monthly deaths from bronchitis, emphysema and asthma in the UK, 1974-1979"

```
> mean(deaths)
```

```
[1] 2056.625
```

Question 0.10. The `rivers` data set lists the lengths in miles of major North American rivers.

1. How many rivers are in the data set?
2. Compare the mean to the 10% trimmed mean.
3. Make a histogram of the data and describe the shape.

Question 0.11. For the `deaths (MASS)` data set:

1. Find the number of data points, n.
2. Compare the mean and 10% trimmed mean
3. Make a histogram and comment on the shape.

Data frames, data types

Data comes in a few different flavors: we have numeric data which can be on a continuous scale (no two values are expected to be the same) or a discrete scale and categorical data. Additionally, we can be considering univariate, bivariate or multivariate data sets (one, two or two or more variables). R must be able to store all of these in an efficient manner.

The basic R data types are

numeric This is what we have already seen and is how R stores numbers. R does not distinguish between discrete and continuous.

```
> x <- c(1, 2, 3, 45.1)
> x

[1]  1.0  2.0  3.0 45.1

> class(x)

[1] "numeric"
```

character Strings are stored as characters:

```
> x <- c("red", "white", "and", "blue")
> x

[1] "red"   "white" "and"   "blue"

> class(x)

[1] "character"
```

factor Categorical variables can be stored as characters, but more often they appear as factors. This is R's way of knowing a variable is categorical. Factors have a specific set of categories that are possible called `levels`. Notice how factors print differently from character data sets:

```
> x <- factor(c("freshman", "sophmore", "sophmore", "freshman",
+     "junior"))
> x

[1] freshman sophmore sophmore freshman junior
Levels: freshman junior sophmore

> class(x)

[1] "factor"
```

logical Logical values will be seen to be useful later. A logical variable is either `TRUE` or `FALSE`:

```
> x <- c(TRUE, FALSE, TRUE)
> x

[1]  TRUE FALSE  TRUE

> class(x)

[1] "logical"
```

Some things to know:

1. If a value is not available, the special value `NA` should be used.
2. If you mix types, R will coerce all the values to the type that will accomodate all the values – usually character:

   ```
   > x <- c(1, "two", 3)
   > class(x)

   [1] "character"
   ```

bivariate or multivariate data

A typical data set in statistics might involve several measurements on the same case. For instance, a nurse might record height, weight, pulse, blood pressure, smoking status ... for several patients. A rectangular grid of numbers – like that in a spread sheet – is a convenient means of recording these values. R provides a data frame to store such data types.

A data frame has one or more columns containing variables. The values in a particular column are all of the same type (eg. numeric or factor), but the different columns can be different (weight is numeric, smoking status categorical). Each row corresponds to the case or subject (or patient in this example.) Most of the built in data sets in R are data frames. As these can be quite big, we mention the function `head` to return the first few rows, `names` to give the variable names and `str` to return a summary of each variable.

The `head`, `str` and `names` functions

For example, the `Aids2` data set in the `MASS` package has

```
> head(Aids2)
```

```
  state sex  diag death status T.categ age
1   NSW   M 10905 11081      D      hs  35
2   NSW   M 11029 11096      D      hs  53
3   NSW   M  9551  9983      D      hs  42
4   NSW   M  9577  9654      D    haem  44
5   NSW   M 10015 10290      D      hs  39
6   NSW   M  9971 10344      D      hs  36
```

```
> names(Aids2)
```

```
[1] "state"   "sex"     "diag"    "death"   "status"  "T.categ" "age"
```

```
> str(Aids2)
```

```
'data.frame':        2843 obs. of  7 variables:
 $ state  : Factor w/ 4 levels "NSW","Other",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ sex    : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ diag   : int  10905 11029 9551 9577 10015 9971 10746 10042 10464 10439 ...
 $ death  : int  11081 11096 9983 9654 10290 10344 11135 11069 10956 10873 ...
 $ status : Factor w/ 2 levels "A","D": 2 2 2 2 2 2 2 2 2 2 ...
 $ T.categ: Factor w/ 8 levels "hs","hsid","id",..: 1 1 1 5 1 1 8 1 1 2 ...
 $ age    : int  35 53 42 44 39 36 36 31 26 27 ...
```

The `str` fuction shows some variables are factors and some are integers (numeric).

Question 0.12. The `Cars93` data set in the `MASS` package has many variables on different car models. How many variables are factors?

How do we get the variable in a data frame? The data frame name refers to all the variables as a data set. The `names` function return the variable names. To get the data in a given variable we need notation to refer to the variable within the data frame. The dollar sign notation does this through the pattern `dataframe$variable`. E.g.,

```
> mean(Aids2$age)
```

```
[1] 37.40907
```

```
> mean(Cars93$Weight)
```

```
[1] 3072.903
```

Question 0.13. Make histograms of the variables `MPG.highway`, `Price` and `Weight` stored in the `Cars93` data set. Comment on their shape.

Data from the internet

R can read data in from the internet. This is useful for doing your homework! Why, the data sets are online. For instance, the first homework assignment for section 1.2 is **ex**cercise **56** of chapter **1**. The data appears in the file `ex01_056.txt` on the website `http://www.math.csi.cuny.edu/verzani/classes/MTH214/R/Data`. To read that data into an R session we have

```
> f <- "http://www.math.csi.cuny.edu/verzani/classes/MTH214/R/Data/ex01_056.txt"
> d <- read.table(f, header = TRUE)
> names(d)
```

```
[1] "DBH"
```

The `read.table` command returned a data frame with only one variable `DBH`. The homework is answered by

```
> x <- d$DBH
> fivenum(x)
```

```
[1]  2.20 10.95 28.50 41.90 69.30
```

```
> boxplot(x)
> hist(x)
```

My short summary might be:

> For this data, the boxplot shows a data set which appears symmetric. However, the boxplot does not show *modes* in a data set. The histogram picks up the apparent bimodality in the data with a peak between 0 and 10 and another between 40 and 50. This might be due to a forest planted at two different times.

Question 0.14. Download the data for problem 60 of chapter 1, `ex01_060.txt`. Make a stem and leaf plot, histogram, dotplot and boxplot. Which shows the distribution of values better?