Precept 2 Answers

Bill Haarlow

Fall 2019

Contents

Objectives

- Getting familiar with functions: `hist()`, `unique()`, `sort()`, `%in%` and `rank()`
- Learn how to create a report with the Rmarkdown package
- Learn construction, extraction and assignment techniques for lists (and data frames)
- Learn how to interpret histograms

Rmarkdown Introduction

Rmarkdown is a **mark-up language** that can generate reports which includes R codes. As opposed to text processing programs that require marking the end-result in the document

itself, this type of method marks the desired formatting and the final document is achieved by rendering the document. For example, *italicized text* can be marked with a single underscores or asterisks, **boldfaced text** is marked with double underscores or asterisks, header is marked with one or more `#` at the beginning of a line, etc.

The first thing you see in this type of file is the header. The rendering process will use this to figure out the title and author name. You can also specify formatting option here such as, the type of output document (here a PDF instead of the default of HTML), include a table of content (toc = true or toc = yes), size of figures, adding captions to figure, etc.

Knitr

To generate an output file with a .Rmd file, you need to render the .Rmd file using the button "Knit" that is on the top of the upper-left panel in Rstudio. You can control the rendering options for the entire file using the following command, each option controls a different property in the file created:

What the options in knitr::opts_chunk$set() mean

The options set in knitr::opts_chunk$set() control for the entire report.

However, you can also use these options to control the settings for a particular code chunk; to do so you would put the option(s) inside the curly braces. E.g., all the graphs produced by the code in the following code chunk will be aligned left and will be 10 inches tall:

"'{r fig.align="left", fig.height = 10}

"'

Below are the commonly used options and their descriptions, along with their default values and the data types of the input values in the parentheses.

fig.align: ('default'; character) alignment of figures in the output document (possible values are left, right and center; default is not to make any alignment adjustments).

fig.height: (7; numeric) Height in inches for plots created in chunk.

fig.width: (7; numeric) Width in inches for plots created in chunk.

message: (TRUE; logical) whether to preserve messages emitted by message() (similar to warning).

collapse: (FALSE; logical; applies to Markdown output only) whether to, if possible, collapse all the source and output blocks from one code chunk into a single block.

prompt: (FALSE; logical) whether to add the prompt characters in the R code (see prompt and continue in ?base::options; note that adding prompts can make it difficult for readers to copy R code from the output, so prompt=FALSE may be a better choice.

comment: ('##'; character) the prefix to be put before source code output; default is to comment out the output by ##, which is good for readers to copy R source code since output is masked in comments (set comment=NA to disable this feature).

cache: (FALSE; logical) Whether to save results so that code blocks aren't re-run unless code changes, *or* a relevant earlier code block changed (see autodep).

autodep: (FALSE; logical) whether to figure out the dependencies among chunks automatically by analyzing the global variables in the code (may not be reliable) so that dependence does not need to be set explicitly.

options(width=63): (80; numeric) Controls the maximum number of columns on a line used in printing vectors, matrices and arrays.

Some code chunks for the demo

You can write **chunks** of code within the .Rmd file. This code can be incorporated as is, executed to include its output, or both. Below are a couple of examples for the use of code chunks using the `cars` dataset; the dataset lists the speed of 50 cars and the distances these 50 cars take to stop. We can see the size of the data frame using `dim`. We can use `head` to see that cars holds two vectors with numerical values.

```
> # within the code chunk everything is treated as if they were in
> # a .R script file; when you knit the .Rmd file all the code in
> # the code chunks in the report will be run sequentially
> dim(cars)
[1] 50  2
```

```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

Lists and data frames

Lists

Recall that for a data frame its elements can have different data types. The data frames that we will encounter in this course usually have vectors as the elements. E.g., for the titanic dataset:

```
> t.ship = read.csv("/Users/billhaarlow/Desktop/SML201/titanic.csv")
>
> head(t.ship)
  PassengerId Survived Pclass
1           1        0      3
2           2        1      1
3           3        1      3
4           4        1      1
```

```
5           5        0      3
6           6        0      3
                                                 Name    Sex
1                             Braund, Mr. Owen Harris   male
2 Cumings, Mrs. John Bradley (Florence Briggs Thayer) female
3                              Heikkinen, Miss. Laina female
4        Futrelle, Mrs. Jacques Heath (Lily May Peel) female
5                            Allen, Mr. William Henry   male
6                                    Moran, Mr. James   male
  Age SibSp Parch           Ticket    Fare Cabin Embarked
1  22     1     0        A/5 21171  7.2500              S
2  38     1     0         PC 17599 71.2833   C85        C
3  26     0     0 STON/O2. 3101282  7.9250              S
4  35     1     0           113803 53.1000  C123        S
5  35     0     0           373450  8.0500              S
6  NA     0     0           330877  8.4583              Q
>
> str(t.ship, strict.width = "cut")
'data.frame':   891 obs. of  12 variables:
 $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
 $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
 $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
 $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",....
 $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2..
 $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
 $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
 $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
 $ Ticket     : Factor w/ 681 levels "110152","110413",..: 52..
 $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
 $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 ..
 $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 ..
```

Note that all the vectors (i.e., the columns) in `t.ship` have the same length.

A *list* is a generalization of a data frame. For a list, not only that its elements can have different data types, they can also have different sizes. Thus, a data frame is just a special type of list where all its elements are of the same size. List is the most flexible object type in R.

Here is an example of a list.

```
> # Create two lists for the example
> area.code2017 = list(area.code = 609, area = c("Princeton", "Trenton",
+     "Lawrenceville"))
> new.area.code2018 = list(new.area.code = 640, purpose = "overlay 609 in 2018")
>
> # You can concatenate two lists into one
> c(area.code2017, new.area.code2018)
$area.code
```

```
[1] 609

$area
[1] "Princeton"     "Trenton"       "Lawrenceville"

$new.area.code
[1] 640

$purpose
[1] "overlay 609 in 2018"
>
> #
> c(area.code2017, new.area.code2018)$area.code
[1] 609
```

How do we extract elements from a list? Recall that in terms of structure a list is like a vector of placeholders and inside of each placeholder there can be any R object.

Let's have a quick review on how extraction works with vectors.

```
> c(-7:4)
 [1] -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4
> c(-7:4)[5]  # extract out the part of the vector that contains the 5th element;
[1] -3
> class(c(-7:4)[5])  # still a vector
[1] "integer"
>
> c(-7:4)[5:7]  # extract out the part of the vector that contains the 5th to 7th element;
[1] -3 -2 -1
> class(c(-7:4)[5:7])  # still a vector
[1] "integer"
>
> c(-7:4)[length(c(-7:4))]  # extract out the part of the vector that contains the last element;
[1] 4
> class(c(-7:4)[length(c(-7:4))])  # still a vector
[1] "integer"
```

Now, see how extraction works with lists in R.

```
> # list extraction works in a way that is similar to vector
> # extraction
> area.code2017[2]
$area
[1] "Princeton"     "Trenton"       "Lawrenceville"
> class(area.code2017[2])  # still a list
[1] "list"
>
> area.code2017[1:2]
$area.code
```

```
[1] 609

$area
[1] "Princeton"     "Trenton"       "Lawrenceville"
> class(area.code2017[1:2])  # still a list
[1] "list"
```

```
> # Any difference between the following 3 ways?
> area.code2017[2]
$area
[1] "Princeton"     "Trenton"       "Lawrenceville"
> class(area.code2017[2])  # note that this is still a list
[1] "list"
>
> area.code2017[[2]]
[1] "Princeton"     "Trenton"       "Lawrenceville"
> class(area.code2017[[2]])  # note that this is a vector
[1] "character"
>
> area.code2017$area
[1] "Princeton"     "Trenton"       "Lawrenceville"
> class(area.code2017$area)  # note that this is a vector
[1] "character"
> # How do you extract out 'Princeton'?  What will you get with
> # this: area.code2017[2][1]?
```

Note that we used `[[]]` and `$` here; these are the extraction functions that reserved for lists. You might remember that we used `$` for the data frame `t.ship` in lecture before. This is because a data frame is just a special kind of list whose elements are vectors of the same length.

Data frames

Since a data frame is a special case of a list, a data frame inherits all the properties of a list. In addition, since a data frame has a two-dimensional appearance that is similar to a matrix, some of the extraction operations that work for matrices works for a data frame as well. Below are examples demonstrating how extraction works for data frames

```
> # The following extraction methods result in a vector:
> class(t.ship[, 1])  # first vector from the data frame
[1] "integer"
> head(t.ship[, 1])
[1] 1 2 3 4 5 6
>
> class(t.ship$PassengerI)  # same as above, but use the name of this column in the df
[1] "integer"
>
> head(t.ship[[1]])
[1] 1 2 3 4 5 6
```

```
> class(head(t.ship[[1]]))  # this goes into the first placeholder and take out the vector inside
[1] "integer"
> head(t.ship[[1:2]])  # this doesn't work as expected
[1] 2
> # because you are already inside of the placeholder.  This
> # actually gets the 2nd element of the first vector.
>
> # The following extraction methods result in a data frame:
> head(t.ship[1])  # extract out the part of the data frame that has the first element/column
  PassengerId
1           1
2           2
3           3
4           4
5           5
6           6
> class(head(t.ship[1]))  # still a data frame
[1] "data.frame"
> head(t.ship[1:2])  # this still works
  PassengerId Survived
1           1        0
2           2        1
3           3        1
4           4        1
5           5        0
6           6        0
>
> head(t.ship[c("Age", "Sex")])  # same as above, a data frame
  Age    Sex
1  22   male
2  38 female
3  26 female
4  35 female
5  35   male
6  NA   male
> class(t.ship[1])
[1] "data.frame"
> class(t.ship[c("Age", "Sex")])
[1] "data.frame"
```

In short, only using `[ ]` as how you would extract elements in a vector will result in a data frame.

How to use these functions: `unique()`, `sort()`, and `%in%`

```
> # create vector for the example:
> v = c(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, -4, 9)
```

```
>
> # returns a vector with the unique value by removing duplicates
> unique(v)
[1]  3  1  4  5  9  2  6  8 -4
>
> # returns a vector of the values sorted in increasing/decreasing
> # order
> sort(v)
 [1] -4  1  1  2  3  3  4  5  5  5  6  8  9  9
> sort(unique(v))
[1] -4  1  2  3  4  5  6  8  9
> sort(unique(v), decreasing = T)
[1]  9  8  6  5  4  3  2  1 -4
>
> # do 1, 3, 4 appear in v?
> v %in% c(1, 3, 4)  # note that this is different from c(1, 3, 4) %in% v
 [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
[10]  TRUE FALSE FALSE FALSE FALSE
> # extract out elements in v that are 1, 3, or 4
> v[v %in% c(1, 3, 4)]
[1] 3 1 4 1 3
```

Use `rank()` to get the ranks of the elements in a vector

```
>
> # rank() - gives ranks to the input vector; smallest value will
> # receive rank 1, next smallest value will receive rank 2, etc.
> # Equal values, 'ties', can be treated in different ways (see
> # details in help manual).
>
> # the line below just assigns some letters in alphabetical order
> # to the elements in the vector
> names(v) <- letters[1:14]
>
> v
 a  b  c  d  e  f  g  h  i  j  k  l  m  n
 3  1  4  1  5  9  2  6  5  3  5  8 -4  9
>
> rank(v)  # tie-ranks will be replaced by their average
   a    b    c    d    e    f    g    h    i    j    k    l
 5.5  2.5  7.0  2.5  9.0 13.5  4.0 11.0  9.0  5.5  9.0 12.0
   m    n
 1.0 13.5
> rank(-v)  # rank in decreasing order
   a    b    c    d    e    f    g    h    i    j    k    l
 9.5 12.5  8.0 12.5  6.0  1.5 11.0  4.0  6.0  9.5  6.0  3.0
```

```
   m    n
14.0  1.5
```

Histograms

We will use the `nhtemp` dataset (one of R's built-in datasets) for this week's demo. `nhtemp` records the mean annual temperature (in degrees Fahrenheit) in New Haven, Connecticut, from 1912 to 1971.

```
> `?`(nhtemp)
> class(nhtemp)
[1] "ts"
> # `nhtemp` is a time series R object; you can just treat
> # `nhtemp` as a vector
>
> head(nhtemp)
[1] 49.9 52.3 49.4 51.1 49.4 47.9
> # The Average Yearly Temperatures in New Haven, 1912-1971
> length(nhtemp)  #Number of observations
[1] 60
```

In the previous weeks we learned how to perform mathematical operations on data; for example,

```
> sum(nhtemp)  #Sum of all observations
[1] 3069.6
> head(sqrt(nhtemp))
[1] 7.063993 7.231874 7.028513 7.148426 7.028513 6.920983
> # Vectorization computation: take square root of each element
> min(nhtemp)  #The lowest recorded average temp
[1] 47.9
> max(nhtemp)  #The highest recorded average temp
[1] 54.6
> mean(nhtemp)  #The average temp across all years
[1] 51.16
> range(nhtemp)  #The lowest and highest recorded average temp
[1] 47.9 54.6
```

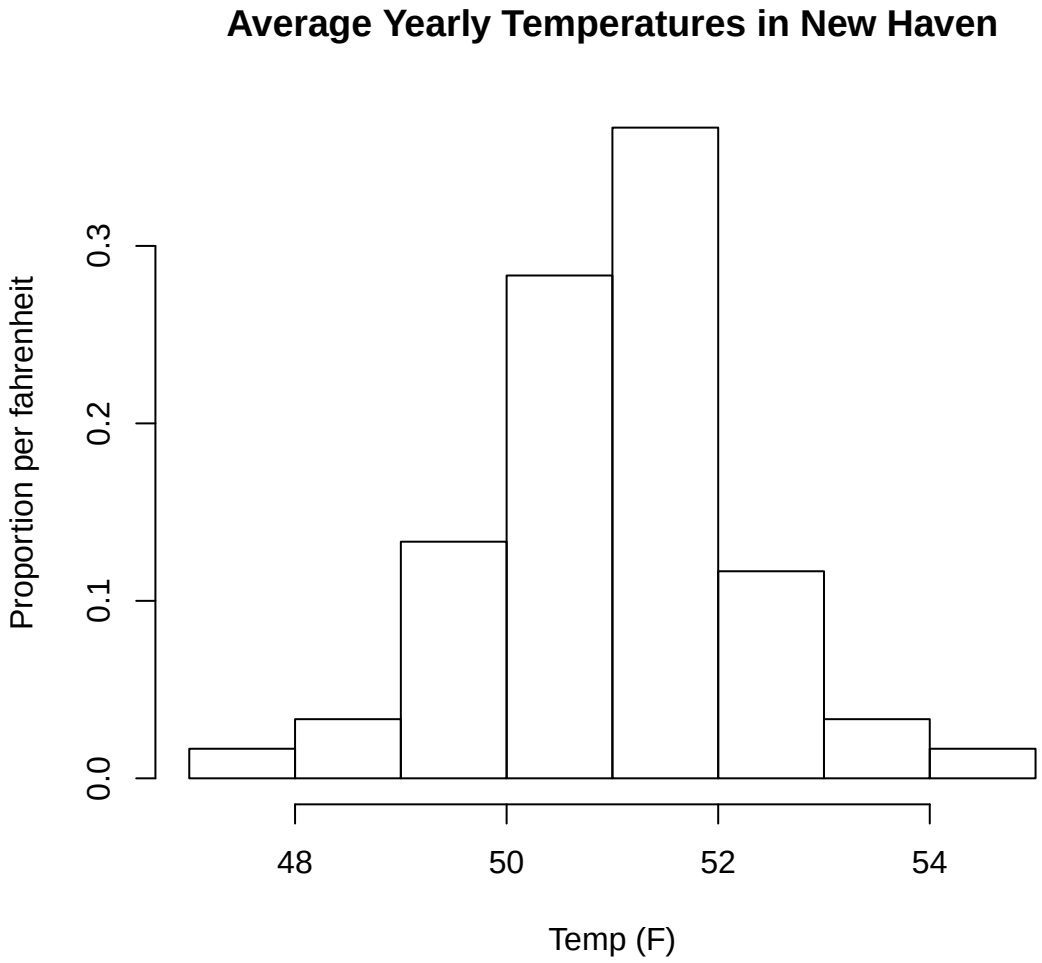
To understand the distribution of the data, we can display the data graphically with a histogram. The basic function to create a histogram is hist(). Similar to previously learned functions, we can adjust the values for its input arguments to modify the output of this function.

In addition, we can use the function par() to adjust the values for the graphic parameters that are used to display the graph. You can check out the help manual for par() to see all the available input arguments for the function but here are some useful ones that we are going to talk about today:

mfrow: a vector for the number of rows and number of columns c(nr,nc), to describe the layout of the plotting area. Used to design the layout of several plots.

cex.main, cex.lab, cex, cex.aixs: used to change the sizes of the font for the title, the x- and y- axis labels, the symbols and the axis annotations on a plot, respectively. These arguments can also be used in other graphical functions, such as, hist(), plot(), points(), lines(), abline()–the last 4 functions will be discussed in next week's precept.

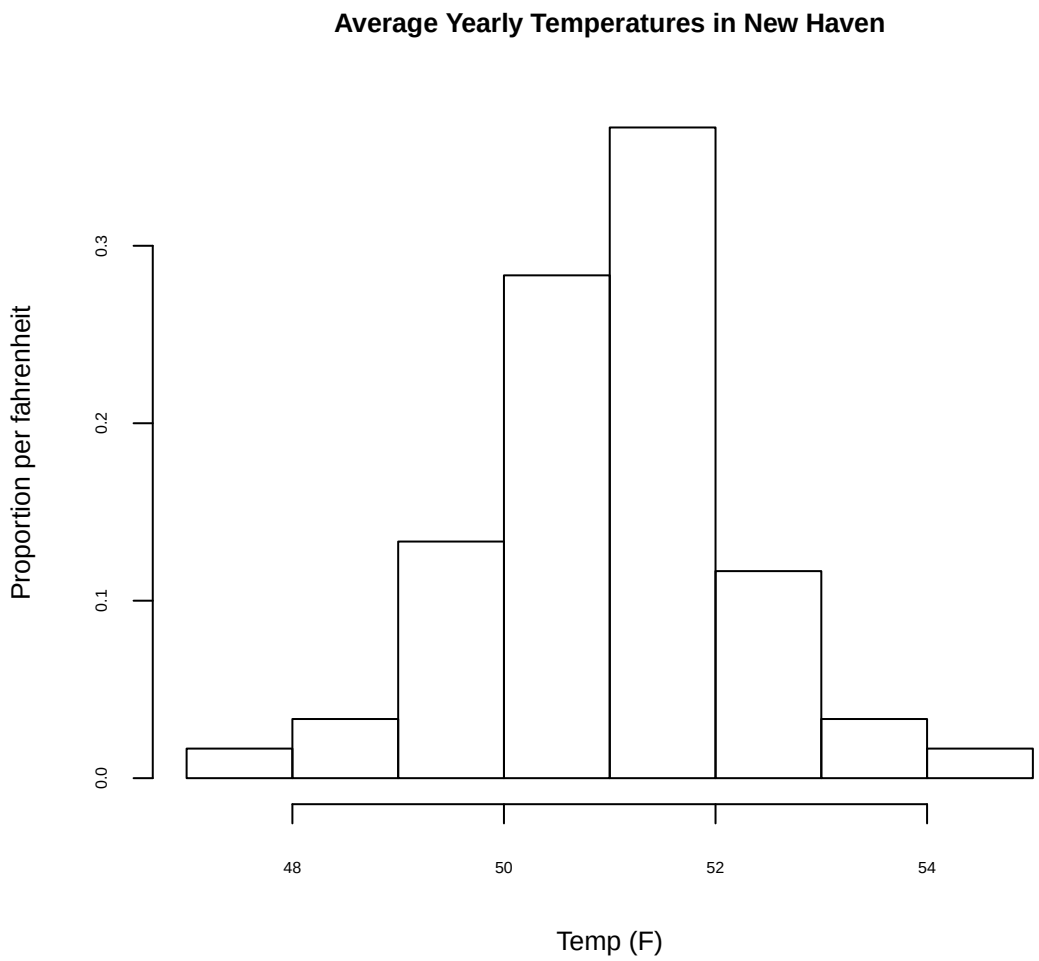
```
> # Add lables to the histogram
> hist(nhtemp,freq = F,
+      main='Average Yearly Temperatures in New Haven',
+      xlab = 'Temp (F)', ylab = 'Proportion per fahrenheit' )
```



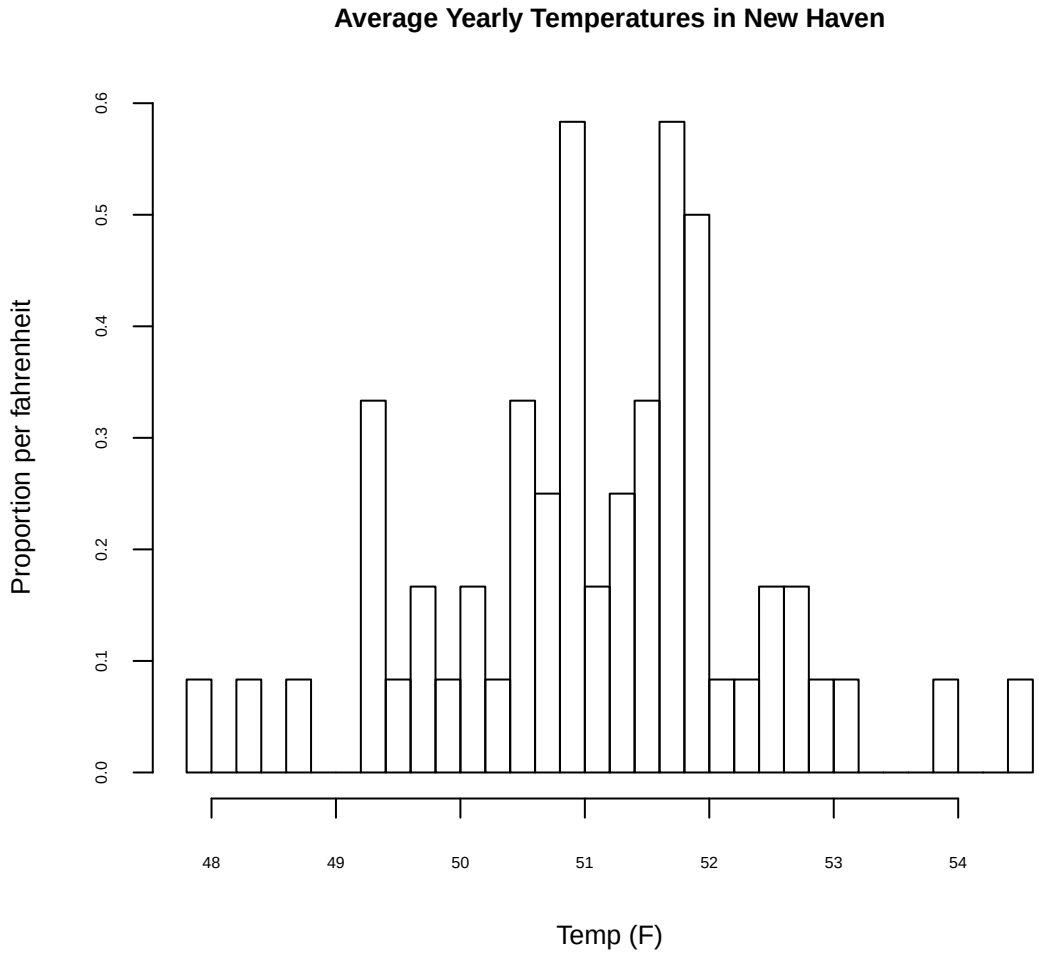

```
> # Setting the correct scale for the y-axis with `freq` the input argument.
> # Set freq = F, the histogram displays density so that the total
> # area equals 1. the area of the rectangles is the fraction of data points
> # falling into the cells, that is $density = proportion per x-axis unit$
```

```
> # Adjust title, axis label sizes:
> # cex.main=.8: sets the font of the title to be 80% of the default size
> # cex.lab=.8: sets the font of the x- and y- labels to be 80% of the default size
> # cex.axis=.5: sets the font of the axis annotations to be 50% of the default size
```

```
>
> hist(nhtemp,freq = F, cex.main=.8, cex.lab=.8, cex.axis=.5,
+      main='Average Yearly Temperatures in New Haven',
+      xlab = 'Temp (F)', ylab = 'Proportion per fahrenheit' )
```



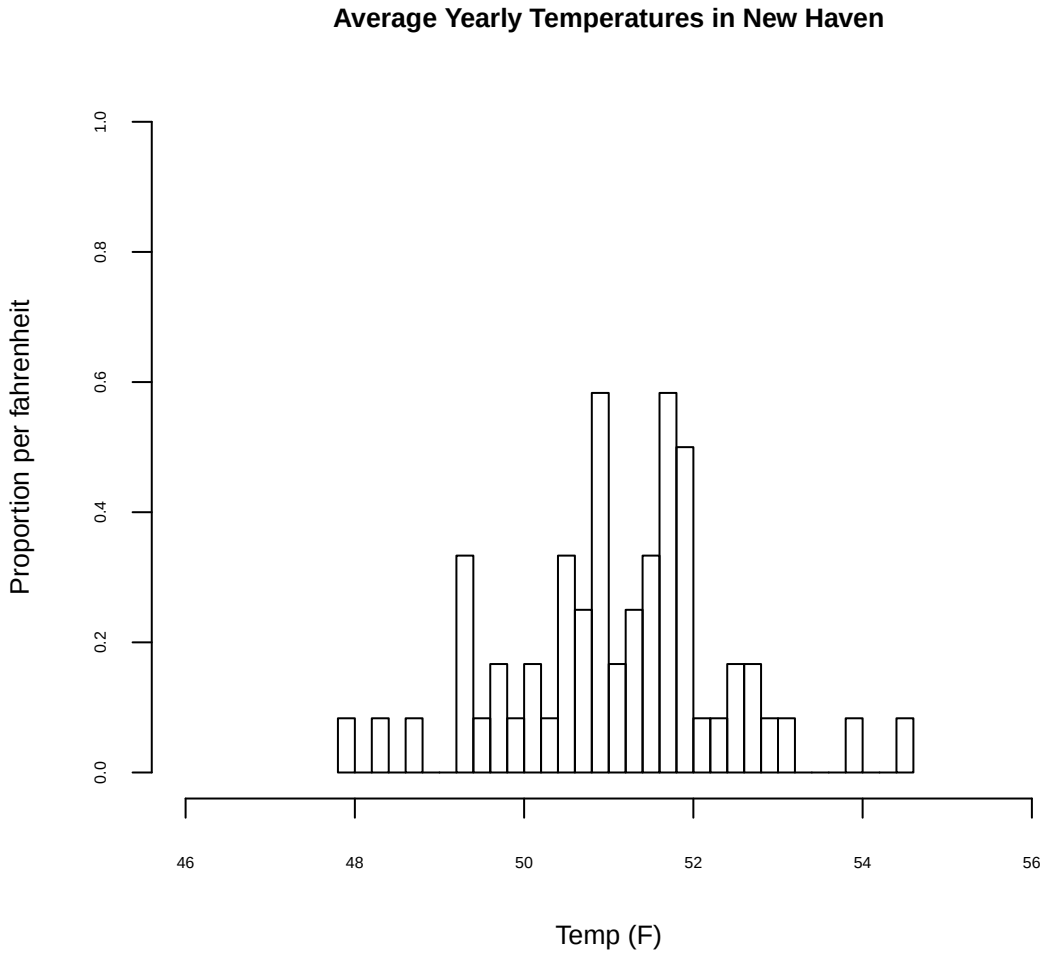

```
> # Add more breaks
> hist(nhtemp, freq = F, cex.main = 0.8, cex.lab = 0.8, cex.axis = 0.5,
+     main = "Average Yearly Temperatures in New Haven", xlab = "Temp (F)",
+     ylab = "Proportion per fahrenheit", breaks = 30)
```



Please adjust the value for the input argument `breaks` to see how the graph changes and why 30 is a good value for breaks.

We can also use **xlim** and **ylim** in graphical functions to set the ranges for the x- and y-axes for the graphs. For example, the following sets the minimum and the maximum of the x-axis to be 46 and 56, and the minimum and the maximum of the y-axis to be 0 and 1.

```
> hist(nhtemp, freq = F, cex.main = 0.8, cex.lab = 0.8, cex.axis = 0.5,
+     main = "Average Yearly Temperatures in New Haven", xlab = "Temp (F)",
+     ylab = "Proportion per fahrenheit", breaks = 30, xlim = c(46,
+         56), ylim = c(0, 1))
```



The following links is helpful for learning R Markdown (especially the first one).

http://www.stat.cmu.edu/~cshalizi/rmarkdown/#mark-up-markdown

https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

https://yihui.name/knitr/options/

Exercises

Please include your answers in code chunks and make sure that you can knit your file successfully.

Question 1

(a) Try placing one at a time of the following arguments into the curly brackets for the first code chunk and observe what happens when you compile (knit) the file.

include = FALSE

echo = FALSE
eval = FALSE

i.e., for the first one do this

Side note: note that you can also add one of these arguments to knitr::opts_chunk$set(), and the option will then controls all the code chunk globally.

(b) Change 'toc: no' in the YAML header to 'toc: yes' and observe what happens when you compile the file. Also, type your name for the `author` field in the header.

Add a code chunk to each of the following questions to evaluate the answers

Question 2

`cars` is a built-in data frame in R.

```
> class(cars)
[1] "data.frame"
> dim(cars)
[1] 50  2
```

Without running any code, say what kind of R object each of the following line will return. Once you finish, verify your answers by using the `class()` function. (You do not have to show the code to apply the `class()` function; the verifications are just for your own benefit.)

```
> cars[, 1]  # numeric
 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13
[20] 14 14 14 14 15 15 15 16 16 17 17 17 18 18 18 18 19 19 19
[39] 20 20 20 20 20 22 23 24 24 24 24 25
> cars$speed  # numeric
 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13
[20] 14 14 14 14 15 15 15 16 16 17 17 17 18 18 18 18 19 19 19
[39] 20 20 20 20 20 22 23 24 24 24 24 25
> head(cars[1])  # data.frame
  speed
1     4
2     4
3     7
4     7
5     8
6     9
> head(cars["speed"])  # data.frame
  speed
1     4
2     4
3     7
4     7
5     8
6     9
> cars[[1]]  # numeric
```

```
 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13
[20] 14 14 14 14 15 15 15 16 16 17 17 17 18 18 18 18 19 19 19
[39] 20 20 20 20 20 22 23 24 24 24 24 25
```

Question 3

Use the dataset `babynames` in the `babynames` package.

(a) Make a data frame "high.freq" by extracting out the rows for female names that were used at least 20000 times in a given year between year 2000 and year 2017 inclusively. Hint: You should break down this problem into parts. Your thought/work process should follow steps similar to the following:

- Step 1: I am asked to extract out the rows that satisfy the following conditions:
  a) for female names
  b) for names used at least 20000 times in a given year
  c) for years between year 2000 and year 2017 inclusively
- Step 2: How do I extract out rows that satisfy each of the conditions above?
- Step 3: Work out the code for Step 2 and test the code by checking the results produced by the code (e.g., use functions, such as `range()`, `table()`, `summary()`, `head()` etc.)
- Step 4: How do I go from the results in step 3 to the requested result?

```
> library(babynames)  # Loads package `babynames`
> babynames$sex == "F"  # Extracts 'sex' column & extracts all female names
> female_baby = babynames$sex == "F"  # Assigns female names to `female_baby`
> high.freq = babynames[female_baby, ]  # Creates data frame `high.freq` using `female_baby`
> years2000.2017 = high.freq$year %in% c(2000:2017)  # Extracts all data from 2000 to 2017 from
> high.freq = high.freq[years2000.2017, ]  # Reassigns year-extracted `high.freq` to `high.freq`
> high.freq
> times = high.freq$n >= 20000  # Extracts all names with a frequency over 20000 from `high.freq
> high.freq = high.freq[times, ]  # Reassigns times-extracted `high.freq` to `high.freq`
> high.freq  # Outputs data frame `high.freq`
```

```
> summary(high.freq)
      year          sex                name
 Min.   :2000   Length:26          Length:26
 1st Qu.:2002   Class :character   Class :character
 Median :2004   Mode  :character   Mode  :character
 Mean   :2006
 3rd Qu.:2011
 Max.   :2015
       n              prop
 Min.   :20197   Min.   :0.01003
 1st Qu.:20926   1st Qu.:0.01058
 Median :21805   Median :0.01103
 Mean   :22275   Mean   :0.01123
 3rd Qu.:23036   3rd Qu.:0.01166
 Max.   :25953   Max.   :0.01301
> str(high.freq)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   26 obs. of  5 variables:
 $ year: num  2000 2000 2001 2001 2001 ...
 $ sex : chr  "F" "F" "F" "F" ...
 $ name: chr  "Emily" "Hannah" "Emily" "Madison" ...
 $ n   : int  25953 23080 25055 22164 20712 24463 21773 25688 22704 20197 ...
 $ prop: num  0.013 0.0116 0.0127 0.0112 0.0105 ...
```

(b) What are the names that appear in `high.freq`? How many are there (do not include repeats)?

```
unique(high.freq$name)
```

The 6 unique names in `high.freq` are "Emily", "Hannah", "Madison", "Emma", "Isabella" & "Sophia".

(c) Make a data frame "popular.names" by extracting out the rows from `babynames` for the female names that you found in part (b) for years between 2000 and 2017 inclusively. Compare the size of this data frame with that of `high.freq`.

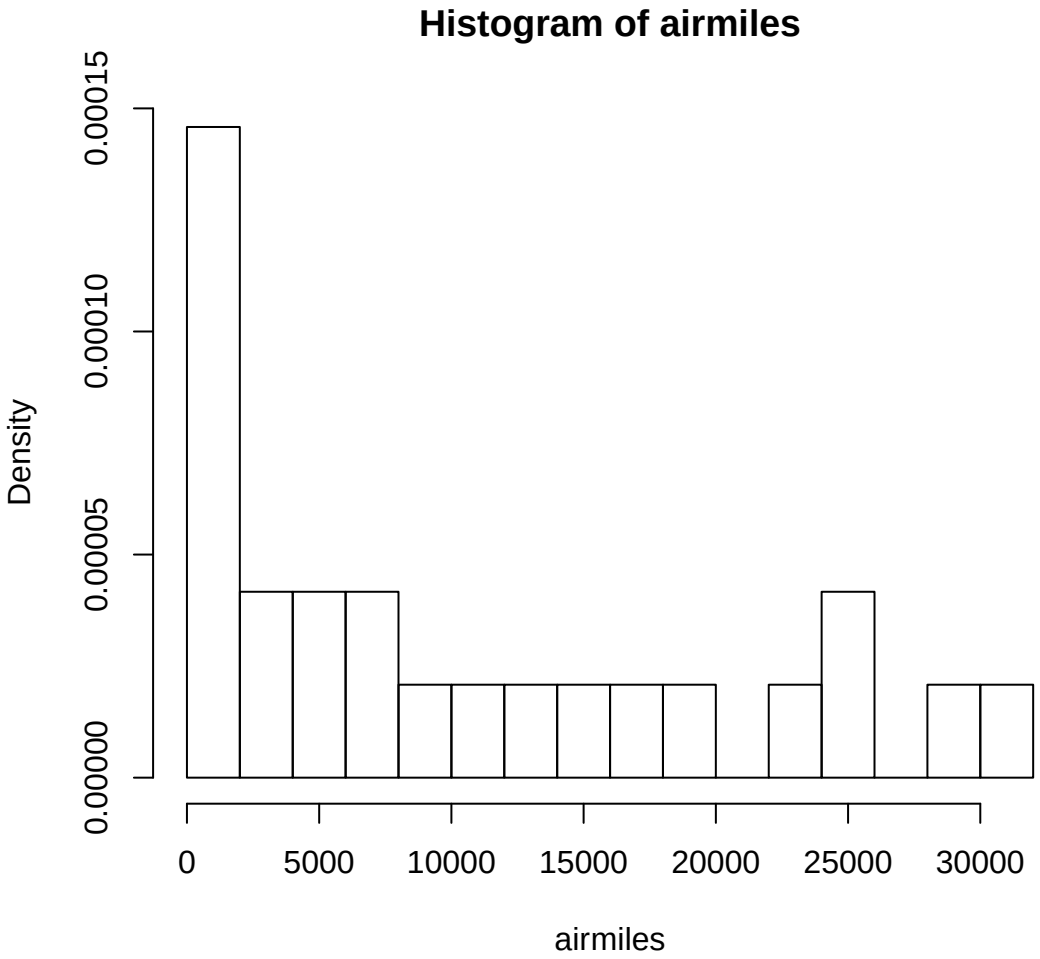
```
> library(babynames)
> babynames$sex == "F"  # Extracts 'sex' column & extracts all female names
> female_baby_p = babynames$sex == "F"  # Assigns female names to `female_baby_p`
> female.names = babynames[female_baby_p, ]  # Creates data frame `female.names` using `female_b
> placeholder = female.names$name == c("Emily", "Hannah", "Madison",
+     "Emma", "Isabella", "Sophia")
Warning in female.names$name == c("Emily", "Hannah", "Madison",
"Emma", : longer object length is not a multiple of shorter
object length
> popular.names = female.names[placeholder, ]  # Creates data frame `popular.names` using `placel
> years2000.2017.p = popular.names$year %in% c(2000:2017)  # Extracts all data from 2000 to 2017
> popular.names = popular.names[years2000.2017.p, ]  # Reassigns year-extracted `popular.names`
> popular.names  ### Up through here is good
> popular.names
> str(popular.names)
```

```
> dim(high.freq)
[1] 26  5
> dim(popular.names)
[1] 25  5
```

Question 4

Plot the histogram for the built-in dataset `airmiles` by using hist(). Answer the following with just the histogram (i.e., without any computations).

```
> hist(airmiles, freq = F, breaks = 16)
```




```
a. What does the area for each bin represent?
The area for each bin represents the proportion of observations for a given number of airmiles.
b. Describe the shape of the distribution of the data (e.g., left- or right- skewed? Symmetric or
This distribution is right-skewed, with a long right tail. It is asymmetric and unimodal.
c. What is the mode of the data?
The mode of the data appears to be within [0, 2000) airmiles.
d. What is the median of the data?  Is the mean bigger/smaller than the median?
The median appears to be within [6000, 8000) airmiles and the mean appears to be smaller than the
```

Question 5: Most popular babygirl names

Suppose that most of you in this class are turning 19 to 22 this year; this means that you were born between 1997 and 2000. We will investigate the baby names in the dataset for this time period .

Part a

Extract out the rows in `babynames` that correspond to the names of the babies born between 1997 and 2000, inclusively, and name the resulting subset `names1997.2000`. How many rows and columns does `names1997.2000` have?

Part b

Use `names1997.2000` from Part a for this Part. For the period of 1997 to 2000 find out the top 20 most popular female names each year (it is okay to just print these four lists out in the code chunk; no need to include these names in the write-up). Among these four lists of top 20 female names, which names stay in the top 20 for all four years for the period of 1997 to 2000?

Please make sure that you print out only the requested names; printing out all the names in your dataset will likely cause Rstudio to have compiling issues since there will be many pages of names to be printed out.