Module 1: Introduction to R and RStudio by exploring eukaryotic genomes

Laurie Stevison and Nolan Bentley

Learning Goals

- Demonstrate basic skills in using the RStudio environment to explore genomic data with summary statistics and make a graphic of a single chromosome.
- Explain the concepts of genome coverage and reproducibility in bioinformatics.

Learning Objectives

- Be able to explain the concept of sequence coverage
- Conduct basic summary statistics to evaluate quantitative data in R
- Isolate columns and subset data in R
- Make simple plots to evaluate data graphically
- Interpret graphs displaying quantitative data
- Reinforce the concept of reproducibility
- Make an R script in RStudio

Prerequisites

- Knowledge of:
    - Chromosome structure (a chromosome is a continuous DNA molecule, basic understanding of chromosome arms)
    - Basic summary statistics such as mean, standard deviation and reading histograms
    - How to read and interpret graphs of data with x and y axes

Class Instruction

- Discuss the question: *What is reproducibility?* (Discuss with a partner, then as a class.) Consider how automating processes by developing code can improve reproducibility.
- Work through the RStudio Exercise, with pauses to discuss the answers to the questions.
- Conclude with an emphasis on the main points:
    - What is genome coverage?
    - What is the difference between a contig and a chromosome?
    - How do statistical summaries of data compare to visual summaries?
    - Anscombe's quartet provides a nice example of this!

Table of Contents

Part 1: Introduction to the Dataset

Genome Sequencing

A **genome** is the complete set of DNA sequence of an organism. In the last 20 years, there has been an increase in sequencing of whole genomes from organisms across the tree of life. These sequencing efforts have subsequently increased the availability of large and complex datasets, which has shifted the expectation for foundational skills for biologists. In addition to understanding basic genetics concepts, like gene structure and function, students now need to build tools that allow them to explore and manipulate large genomic datasets. Platforms such as R provide a starting point for skill development and for doing genome analysis.

Specifically, when scientists sequence a genome, they use a variety of molecular reagents to cut the DNA into short pieces. This is important because many of the platforms for sequencing genomes can only "read" short fragments of DNA at a time, so this step of cutting the DNA makes it easier to sequence the whole genome. Still, one of the most intensive computational steps in genomics is to put all the pieces back together again (think Humpty Dumpty!). There are two basic approaches to this step – assembly and alignment. We'll focus on alignment since it is easier and more common when dealing with model systems, like Drosophila.

Individual pieces of sequenced DNA are referred to as "**reads**" since they are the output that is *read* by sequencing machines. The raw output is very large and not very useful by itself. **Alignment** allows researchers to use an existing sequenced genome as a guide in putting the reads together. Computational algorithms are used to find matches between the **reference genome** sequence and the sequence reads. The reference can either be fully assembled chromosomes, or they can be shorter "contigs". **Contigs** are contiguous pieces of DNA, where researchers are not quite sure how each piece fits together to form whole chromosomes. Alignment is a necessary step in genomic analysis, which allows for comparison of sequences from different individuals of the same species, or individuals in the same population.

Genome Coverage

In model systems like Drosophila, there are several reference genomes to assist in the alignment process. Once they are aligned, the first way to assess the quality is to compute a statistic referred to as **coverage**, or depth. Figure 1 provides an example of this concept; Imagine we align four sequencing reads and examine the depth along the contig below. At position 1, only 1 read overlaps that site and therefore the depth is 1. In the highlighted column in light blue, all four reads overlap the position, and the depth is 4. In this exercise, you will use a file very similar to the bottom row that contains coverage along multiple chromosomes.

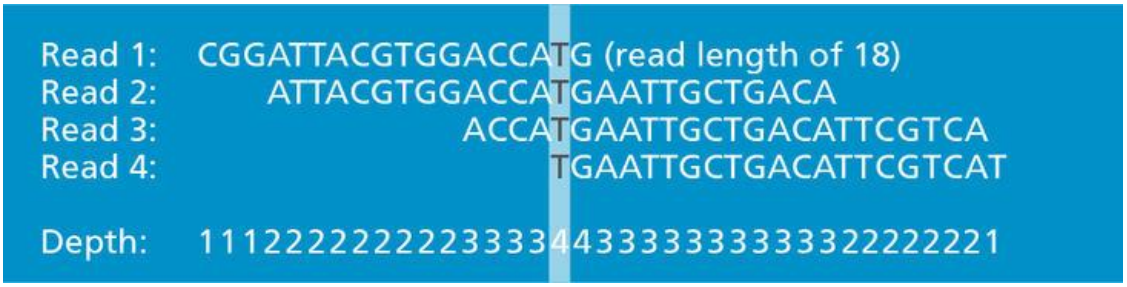


Figure 1. Example of the concept of depth of reads along a chromosome. Image source: By Genomics Education Programme - Read, read length and read depth - read depth of '4', CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=58405954.

Where did the data come from?

The data you will be working with today comes from a commonly used bioinformatics software package called **samtools**, which contains a suite of programs. This software works with genome alignment files to manipulate them and report quality metrics such as coverage. Here, we have used the program **samtools depth** to produce a simple three column output file (Table 1) that you will be working with throughout the activity:

- **Column 1:** The name of the sequence in the reference genome that reads were aligned to.
- **Column 2:** The base pair (bp) position being described in the sequence indicated by column 1
- **Column 3:** The number of reads that aligned to the current reference sequence and position.

Because the output is so large, we will use the program R to read in the output and try to interpret it.

The raw sequence reads used to make this file are from a published study, which is archived in NCBI's short read archive (SRP007802).The reads were aligned to the *Drosophila pseudoobscura* reference genome. You have been provided a subset of the total results, which includes the first 50 kilobases of each contig on the 4th chromosome.

Download the data files

Download and unzip this file to get started:
https://wustl.box.com/shared/static/h7t9cpnfkozno1l6acr3iqg2gv62eo11.zip

You can alternatively use the following R code:

```
url <- "https://wustl.box.com/shared/static/h7t9cpnfkozno1l6acr3iqg2gv62eo11.zip"
download.file(url, destfile =
"chr4.depth.out.zip")
unzip(zipfile = "chr4.depth.out.zip")
```

The data file is provided as a compressed zip archive file. Typically, you can expand the archive simply by double-clicking on the archive file on the three major platforms (Windows, Mac OS X, and Linux). There are also many third-party programs that can expand zip files. These tools include WinZip (http://www.winzip.com) and 7-zip (http://www.7-zip.org) for Windows, Stuff-It Expander for Mac OS X, and the gunzip command-line tool in both Mac OS X and Linux.

Diagram of read mapping

In the following diagram (Table 1) based on your chr4_depth.out file, we have simulated 8 reads mapped to the first 33 positions of chr4_group1. Compare this to the example in Figure 1.

These reads have been simulated to describe a single diploid individual genotype that is overall similar to the reference genome (i.e. it is the same species) but with some differences.

In this example, you can see how depth is a measurement of how many sequences aligned to the same position in the reference sequence (shown under "Ref"). Nucleotide calls that do not agree with the reference genome have been highlighted in yellow.

Contig	Position	Depth	Ref	Read 1	Read 2	Read 3	Read 4	Read 5	Read 6	Read 7	Read 8
chr4_group1	1	0	G								
chr4_group1	2	1	T	T							
chr4_group1	3	2	G	G	G						
chr4_group1	4	2	A	A	A						
chr4_group1	5	2	A	A	A						
chr4_group1	6	2	C	C	C						
chr4_group1	7	2	A	A	A						
chr4_group1	8	2	G	G	G						
chr4_group1	9	2	G	G	G						
chr4_group1	10	2	T	T	T						
chr4_group1	11	2	T	T	T						
chr4_group1	12	2	G		G	G					
chr4_group1	13	2	T			T	T				
chr4_group1	14	2	T			T	T				
chr4_group1	15	2	G			G	G				
chr4_group1	16	2	T			T	T				
chr4_group1	17	2	C			C	C				
chr4_group1	18	3	G			G	G	G			
chr4_group1	19	3	T			T	T	T			
chr4_group1	20	3	T			T	T	T			
chr4_group1	21	3	G			G	G	G			
chr4_group1	22	3	A				A	A	A		
chr4_group1	23	3	C					C	C	C	
chr4_group1	24	4	G					G	C	G	C
chr4_group1	25	4	T					T	T	T	T
chr4_group1	26	4	G					G	G	G	G
chr4_group1	27	4	G					C	C	C	C
chr4_group1	28	3	T						T	T	T
chr4_group1	29	3	T						T	T	T
chr4_group1	30	3	G						G	G	G
chr4_group1	31	3	G						G	G	G
chr4_group1	32	2	T							T	T
chr4_group1	33	1	G								G

Table 1. Simulated read mapping diagram

Questions about the data

The following questions reference Table 1 which is based on your sample data although some of the depths have been changed. Please provide short answer responses to the following:

Q1: Based on the diagram, how many reads mapped to position 2 on contig "chr4_group1"?

Q2: Which position(s) indicate differences between the sequenced genotype and the reference genotype?

Q3: Which position(s) indicate that the sequenced genotype might be heterozygous?

Q4: How long is each read?

Part 2: Setting up the R environment, and loading of data

Getting RStudio Started

In order to continue you should already have both R and RStudio installed on your system. We will use RStudio to interact with the programming language R.

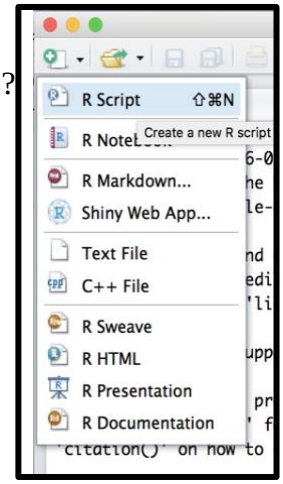


Figure 2. At the top left, click the white rectangle with the green plus sign, which will open the following drop down menu. Select the first option "R Script" to open a new pane in your RStudio session containing an empty R Script file.

The goals for this section are to:

- Organize your data into an otherwise empty directory.
- Create a script file to store and organize your R code
- Setup your R **environment** so that its **working directory** is where your data is located.

To accomplish this:

1. Move the downloaded zip file into a new folder called Module 1.
2. If you have not already done so, unzip the zip file locally so that you can read it into R.
3. Open RStudio and open an empty script file as you saw in the video (Figure 2). This will open a new **source pane** where you will place your code throughout this exercise.
4. Set the **working directory** (the current reference point within the filesystem) in RStudio to the new folder you made in Step 1 (Figure 3).
   a) Select the option "Session" at the top.
   b) Select "Set Working Directory"
   c) Select "Choose Directory"
   d) Browse to the directory storing your unzipped file.
5. After selecting the directory, RStudio tells R to change the working directory by running R code in your **console pane**.

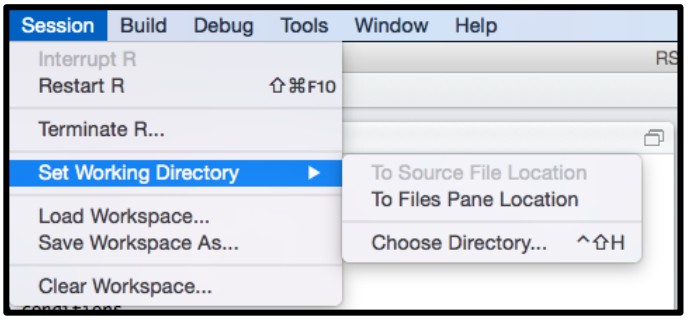


Figure 3. To set the working directory in RStudio.

6. Copy this code into your **source pane** to record what it did (except the ">" symbol). Remember, this pane is where you will write and store code *adding new code underneath previous commands*.
7. Save your script file as "coverage.R" so you will have a record of what code you have run.

Load the data into RStudio

8. Copy and paste the 4 lines of code below into your **source pane** file (save each time you edit!):

```
getwd()
samtools.depth <- read.table(file="chr4.depth.out", stringsAsFactors = F)
class(samtools.depth)
samtools.depth.dim <- dim(samtools.depth)
```

9. You are now ready to run your first line of code. Place the text cursor at the beginning of the first line of code in the **source pane** and press either "Control + Enter" on a PC or "Command + Enter" on a Mac. Repeat for each line. This will cause the lines of code to be run one at a time in the **console pane**. A couple of things will happen at this point:
10. First, you will see the code you ran appear in your console below.
    a) If executed correctly, you should also see the object "samtools.depth" in the environment tab at the top right (Figure 4).
    b) You can also highlight the entire 4 lines and run them all at together.
    c) See troubleshooting below if nothing happens or you get an error.

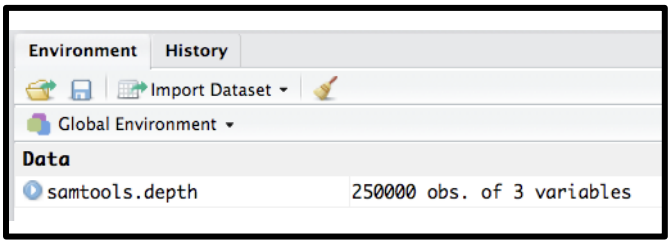


Figure 4. Once you have correctly read in the dataset, your environment tab should look like this.

11. Repeat copying and pasting the 2 pieces of code below to your source pane. Below each line of code there is a description of what should happen after you run the code in your console:
    a) `getwd()`
        i) returns a **character vector** of the **working directory**
        ii) Since the output character vector is unassigned, it is printed by default
        iii) This will be different between different computers and how you set your working directory.
    b) `samtools.depth <- read.table(file="chr4.depth.out", stringsAsFactors = F)`
        i) The assign function `<-` creates an object or overwrites a preexisting object called `samtools.depth`. The data to the right of `<-` is directed to this object.
        ii) The data returned by `read.table` is called a **data.table** and results from opening and interpreting a file named "chr4.depth.out" inside the current working directory.
        iii) The **argument** `stringsAsFactors` has been defined to be FALSE for this line of code. This is no longer necessary in newer versions of R (versions ≥4.0), but it is needed to consistently interpret the code on older installations.
        iv) When run, you won't see anything happen besides the object being added to the environment pane.
    c) `class(samtools.depth)`
        i) This returns the class of the object `samtools.depth` which tells you what type of information this object holds (e.g. numeric, integer, character, data.frame, etc).

ii) Since the output character vector is unassigned, it is printed by default.

d) `samtools.depth.dim <- dim(samtools.depth)`

i) The function `dim` returns the dimensions of an object. However, since the output is assigned to the object `samtools.depth.dim`, nothing is printed when this code is run.

ii) In order to visualize the dimensions you will have to view the contents of `samtools.depth.dim`, using the `print()` function. Alternatively, you can simply type the name of the object `samtools.depth.dim` in the console and press enter (this works because the value of objects is printed by default without requiring an R function).

iii) For tabular data like matrixes and data.frames, this comprises 2 dimensions: the number of rows and the number of columns.

Troubleshooting loading the data

The most common errors here are:

- **Your file isn't at the path you told R to look or your name contains a typo:**
  In the following example, I have added an "s" to the name. It therefore cannot find a file with that name and prints off an error code. Note how it prints out the name of the file I requested in the error.

```
> samtools.depth <- read.table(file= "chr4.depths.out", stringsAsFactors = F)
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file 'chr4.depths.out': No such file or directory
```

- **You don't use quotation marks when indicating the file name to open:**
  In the following example I have left off quotation marks from around "chr4.depth.out". This means that R tries to interpret it as if it was an object of some type instead of interpreting it as the literal name of a file. Therefore, the error it produces indicates that there is no such object.

```
> samtools.depth <- read.table(file= chr4.depth.out, stringsAsFactors = F)
Error in read.table(file = chr4.depth.out, stringsAsFactors = F) :
  object 'chr4.depth.out' not found
```

- **Your script contains a typo:**
  In the following example I have capitalized the "T" in "read.table". Notice how the error message says it cannot find the function I requested. This makes sense; it doesn't exist!

```
> samtools.depth <- read.Table(file= "chr4.depth.out", stringsAsFactors = F)
Error in read.Table(file = "chr4.depth.out", stringsAsFactors = F) :
  could not find function "read.Table"
```

- You never unzipped your file:

```
> samtools.depth <- read.table(file= "chr4.depth.out.zip", stringsAsFactors = F)
Error in scan(file = file, what = what, sep = sep, quote = quote, dec = dec,  :
  line 1 did not have 2 elements
In addition: Warning messages:
1: In read.table(file = "chr4.depth.out.zip", stringsAsFactors = F) :
  line 1 appears to contain embedded nulls
2: In read.table(file = "chr4.depth.out.zip", stringsAsFactors = F) :
  incomplete final line found by readTableHeader on 'chr4.depth.out.zip'
```

Questions about setting up the environment and loading the data

Q5: What is the result of `getwd()` for your system after changing your working directory?

Q6: If I wanted to set my working directory to my downloads folder (this likely doesn't exist for you!) at "C:/Users/DocS/Downloads", what R code should I run? **Hint:** go look at steps 4-6 in "Getting RStudio Started".

Q7: What is the object class of `samtools.depth`? **Hint:** See 11c.

Q8: What is the object class of `samtools.depth.dim`? **Hint:** you did not run this code, so try to edit the code in 11c to answer this question.

Part 3: Basics of R functions

Using help

12. Use the `View()` function to demonstrate how to read help files in R (*Note: V in View is capitalized!*)
    a) In the environment pane of RStudio, click the object "samtools.depth" (Figure 4).
    b) It should open a second tab in the source pane that has a view of your file (Figure 5).
    c) It also will have sent the code RStudio used to generate this pane to the console.
    d) As a reminder, the three columns are V1: the contig name (chr4 stands for chromosome 4 and the group number indicates a region of this chromosome), V2: the position along this contig in base pair (bp), and V3: the number of reads (depth).

	V1	V2	V3
1	chr4_group1	2	1
2	chr4_group1	3	2
3	chr4_group1	4	2
4	chr4_group1	5	2
5	chr4_group1	6	2
6	chr4_group1	7	2
7	chr4_group1	8	2
8	chr4_group1	9	2

Figure 5. Screen shot of the View pane of samtools.depth.

13. Open the help file for `View()`
    a) Learning to use R means you will need to get comfortable asking questions and problem solving!
    b) A great place to start getting your answers is the "Help" tab in RStudio (typically located in the bottom right quadrant of RStudio depending on your version).
        i) Click this tab and type the new command "View" into the search field (Figure 6).
        ii) You can also open it by running the R code `?View`

Figure 6. This shows the search field in the "Help" tab of RStudio. Use this often to look up the functions as you learn to understand what they do and how to use them.

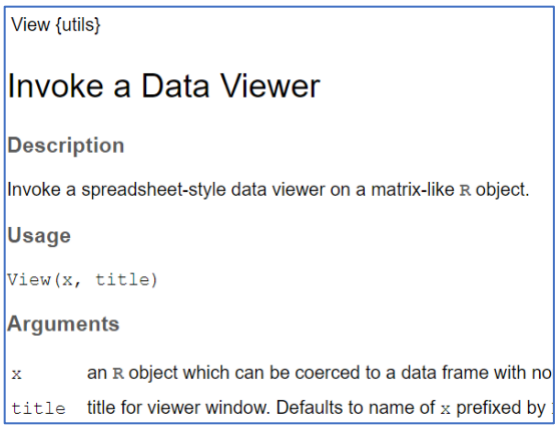
14. Read the help file for `View()`
    a) Note that the help file for `View()` (Figure 7) contains both a "usage" section as well as examples at the bottom.
    b) Note that it contains two arguments named `x` and `title`
    c) The way that you use this information is that it tells you the order of the arguments and what is needed:
    d) `x` needs to be sent an R object (as opposed to the quoted name of the object) such as `samtools.depth`
    e) `title` is ambiguous in that it doesn't say what type of data to send it. However, the description says that it has a default value. Therefore, you don't have to specify it.

Using arguments

15. In the top right quadrant, switch back to your coverage.R script tab again. Now, try using `View()` to view the `samtools.depth` object.
16. Based on the help file (and the example RStudio generated), you probably used either of the following:

```
View(x = samtools.depth) #This works!
View(samtools.depth) #This also works!
```

    a) The first of these works because the x argument is the only argument in `View()` without a default, therefore it works if we specify it.
    b) The second of these works because x is the first argument. If we leave the data passed to it unnamed, R still accepts the syntax.


View {utils}

Invoke a Data Viewer

Description

Invoke a spreadsheet-style data viewer on a matrix-like R object.

Usage

```
View(x, title)
```

Arguments

x	an R object which can be coerced to a data frame with no
title	title for viewer window. Defaults to name of x prefixed by

Figure 7. Image of the help file for the View() function.

17. On the other hand, you may have made a common mistake of adding quotation marks:

```
View("samtools.depth") #This just views the string "samtools.depth", not the object
```

    a) This is a mistake because in R objects are typically referred to using their names without quotes.
    b) You typically use quotation marks only when specifying literal strings of characters.
    c) Often makes the names of objects that we use to refer to the objects distinct from the content of the objects themselves. Therefore in order to view the contents of `samtools.depth`, x needs to not be in quotes in order to pass `View()` the contents and not just the name of the object.
18. Because title is the second argument, we can override its default value by adding a second argument:

```
View(samtools.depth, "Hello world") #This makes the window have a custom title
```

    a) In this example, I literally want to name the title “Hello world” (not refer to an object called Hello world), so I use quotation marks.
    b) Because title is the second argument, it does not need to be manually specified (although you can).
19. Finally, we can submit arguments out of order if we name them:

```
View(title = "Hello world again", samtools.depth) #This is atypically ordered
```

    a) This still works because even though we gave the title first since we manually specified that the first argument was the title.
    b) R was then able to determine that the second argument must be the first unspecified argument x.

Making histograms to explore datasets visually

20. For a change of pace, use the `hist()` function with default parameters to visualize the overall read depths in a histogram by running the following code (note: this may open in the plots tab at the bottom right or in a new window altogether):

```
hist(samtools.depth$V3, xlab = "Read depth")
```

    a) This code works because there is only one required argument (x) in `hist()`.

b) In this example, we have selected the column (or vector) in `samtools.depth` named "V3". This "**extraction**" was accomplished using the `$` symbol.

c) The `xlab` argument can be found in the function's help file. Here we have provide this argument a string (literal characters as opposed to an object's name). This being a string is indicated to R by the quotation marks. (*Note: histograms only require the x axis information since the y axis is always frequency*.)

d) This should produce a plot similar to that on the right (Figure 8).

e) In the help file, you can see that there are many parameters that you can modify (including the number of bins or "breaks" in the data). You will modify these arguments below.

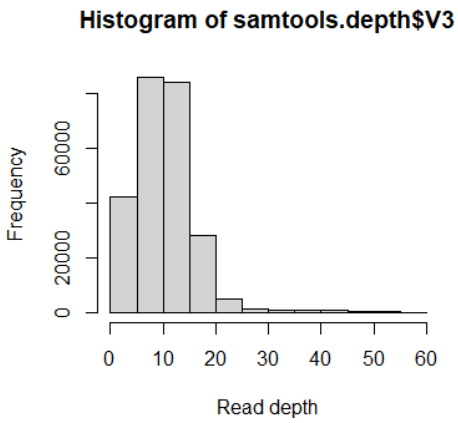


Figure 8. A histogram of the overall read depths

f) Extracting, subsetting, modifying, and otherwise transforming data is an important part of data science, and is covered below in #25 and throughout Module 4.

21. Now let us try to modify the arguments of `hist` make the visualization more informative.

a) Open the **help** file for `hist()`. Note, this function allows for a lot more customization in the "Arguments" section!

b) In your source pane, start a new line of code using the function `hist()`. In the parentheses, pass the following values to the your argument based on your reading of the help file:

i) Set the argument that matches the description "*a vector of values for which the histogram is desired*" equal to:

```
samtools.depth$V3
```

ii) Set the argument that matches the description "*a single number giving the number of cells for the histogram*" equal to:

```
100
```

iii) Set the argument that matches the description "*main title*" equal to:

```
"Overall read depth"
```

c) If successful, your graph will look like Figure 9, but it may be stretched wider or shorter depending on the size of your **plot pane**.

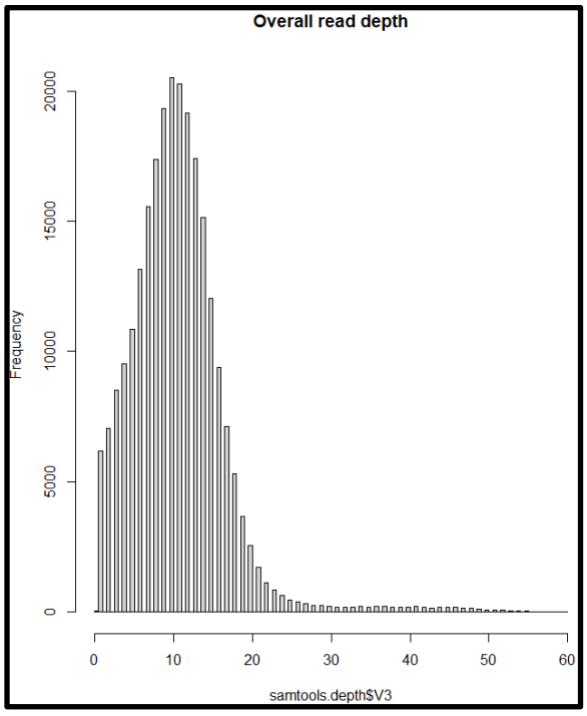


Figure 9. A modified histogram of the overall read depths.

Specifying arguments for various functions

22. Use the commands "head" and "tail" on `samtools.depth`, but print off the first 10 lines instead of the default 6. These are quick commands to let you see the top and bottom of the file.
    a) Use the above tips to open up the help file for head.
    b) Figure out what you need to change to print the first 10 lines instead of the default number lines.
    c) The output of these commands should look like the following:

```
             V1 V2 V3
1  chr4_group1  2  1
2  chr4_group1  3  2
3  chr4_group1  4  2
4  chr4_group1  5  2
5  chr4_group1  6  2
6  chr4_group1  7  2
7  chr4_group1  8  2
8  chr4_group1  9  2
9  chr4_group1 10  2
10 chr4_group1 11  2
```

Example head output

```
                V1    V2 V3
249991 chr4_group5 57290  9
249992 chr4_group5 57291  9
249993 chr4_group5 57292  9
249994 chr4_group5 57293  9
249995 chr4_group5 57294  8
249996 chr4_group5 57295  7
249997 chr4_group5 57296  7
249998 chr4_group5 57297  7
249999 chr4_group5 57298  6
250000 chr4_group5 57299  6
```

Example tail output

23. Edit the following code so that the command runs and titles the plot "Science rules!".

```
hist("samtools.depth$V3", xlab = "Read depth", main = Science rules!)
```

24. Edit the underscore in the following code so that the command runs and produces a histogram summarizing the read depths with only 3 bins. *Note: this is providing you the incorrect code that you need to correct!*

```
hist(3 ,__ = samtools.depth$V3)
```

Q9: Provide the code needed to produce the modified histogram plots.

Subset a particular contig

25. In this file, you will notice that there is more than one contig. Determine the number of contigs by using the command "**unique**" and specifying the column with the contig names.
26. To plot coverage along a single chromosome, you must first subset one of the contigs. Use the command subset to extract the contig "`chr4_group5`" from the `samtools.depth` object. This will create a new object that is smaller.

```
chr4_group5=subset(samtools.depth, samtools.depth$V1=="chr4_group5")
```

Q10: How many contigs are in this file?

Using summary statistics to explore data numerically

Above, you used a histogram to explore your data graphically. Another way to explore your data is through the use of summary statistics. This allows you compare different datasets numerically.

27. Use the R commands **mean** and **sd** to calculate average coverage and standard deviation of coverage in the column V3 for the main dataset `samtools.depth`.
28. Now using the extracted contig `chr4_group5`, calculate the coverage of only that specific contig.

Q11: What is the mean and standard deviation of coverage in the entire file?

Q12: What is the mean and standard deviation of coverage in chr4_group5?

Q13: How do the two values compare to each other? What are some reasons you would expect coverage to be different in one chromosome or one region of a chromosome as compared to the whole genome?

Plot coverage along a single contig

29. Repeat the steps above to make a histogram for the subsetted contig chr4_group5. **Hint:** Look back at the code you made for #21 and edit it to specify `chr4_group5`.

Q14: How do the two histograms compare? **Hint:** Figure 9 represents the whole dataset.

Q15: How do the numerical summary statistics versus the graphical histograms help you in comparing these two datasets?

Now, let's continue to explore this dataset graphically. The final thing you will do for this lab activity is to plot the coverage along a single contig to see how it varies by position.

30. You might also be interested in how coverage varies along each position in the contig. To do this, we will use the command `plot()`. Unlike `hist()`, this command requires you to specify both an x and y axis. For the x axis, we will use the column V2, which contains the position along the contig (see Table 1). The y axis is column V3, which contains coverage.

```
plot(chr4_group5$V2,chr4_group5$V3)
```

31. Use the **help** pane to read about the command `plot()`. Try adding color, a title and names for your two axes to finalize your graph. Finally, write your final plot to an output file (select PNG). If successful, your graph will look like Figure 10, but different based on the customizations you made to your own graph.
32. When you are done, save your coverage.R file so that you can reproduce all of your work on similar files anytime in the future!

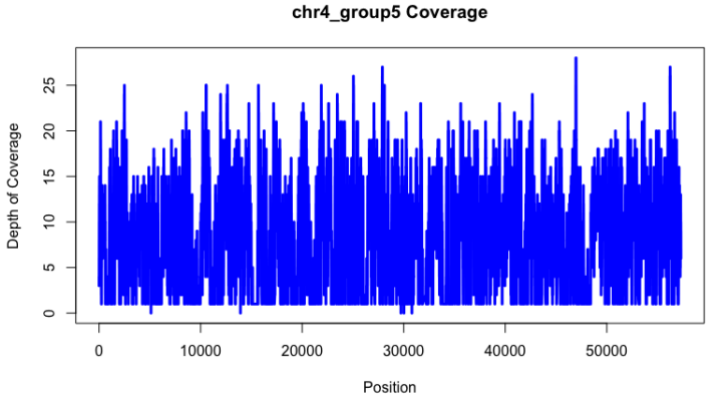


Figure 10. Example plot of coverage along the contig chr4_group5.

Conclusion

In this activity, you learned how to use R and the Rstudio environment to begin to build a skillset that is fundamental to future biologists. To summarize:

- Genome sequencing cannot sequence entire molecules of DNA, meaning that computers are needed to put the short pieces of DNA back together using a reference genome as a guide.
- One way to assess the quality of a genome dataset is to estimate coverage along each chromosome and overall. Certain regions of chromosomes may have dramatically different values of coverage that reveal potential chromosomal structure, such as repetitive regions at chromosome middles (centromeres) and ends (telomeres).
- Because coverage can only be positive, it appears as a non-normal distribution of data.
- Summary statistics like mean and standard deviation reveal quantitatively what summary plots like histograms reveal graphically. Together, both can be informative for exploring datasets. A fun example of this concept is Anscombe's quartet.
- R has a variety of ways to subset data and to customize plots.
- By saving all of your code into a script, you can make your work reproducible such that it could be edited to work on ANY dataset that used a similar input file.