Andrew O. Finley and Jeffrey W. Doser

Forestry 472: Ecological Monitoring and Data Analysis

To my son,

without whom I should have finished this book two years earlier

Contents

List of Tables

List of Figures

Preface

This text is an introduction to data sciences for Forestry and Environmental students. Understanding and responding to current environmental challenges requires strong quantitative and analytical skills. There is a pressing need for professionals with data science expertise in this data rich era. The McKinsey Global Institute[1] predicts that "by 2018, the United States alone could face a shortage of 140,000 to 190,000 people with deep analytical skills as well as 1.5 million managers and analysts with the know-how to use the analysis of big data to make effective decisions". The Harvard Business Review dubbed *data scientist* "The Sexiest Job of the 21st Century"[2]. This need is not at all confined to the tech sector, as forestry professionals are increasingly asked to assume the role of *data scientists* and *data analysts* given the rapid accumulation and availability of environmental data (see, e.g. **?**). Thomson Nguyen's talk[3] on the difference between a data scientist and a data analyst is very interesting and contains elements relevant to the aim of this text. This aim is to give you the opportunity to acquire the tools needed to become an environmental data analyst. Following **?** a *data analyst* has the ability to make appropriate calculations, convert data to graphical representation, interpret the information presented in graphical or mathematical forms, and make judgements or draw conclusions based on the quantitative analysis of data.

[1] http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation

[2] https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century

[3] www.import.io/post/data-scientists-vs-data-analysts-why-the-distinction-matters

0

Data

0.1 FEF Tree Biomass Data Set

When thinking about data, we might initially have in mind a modest-sized and uncomplicated data set that serves a fairly specific purpose. For example, in forestry it is convenient to have a mathematical formula that relates a tree's diameter (or some other easily measured attribute) to stem or total biomass (i.e. we cannot directly measure tree biomass without destructive sampling). When coupled with forest inventory data, such formulas provide a means to estimate forest biomass across management units or entire forest landscapes. A data set used to create such formulas includes felled tree biomass by tree component for four hardwood species of the central Appalachians sampled on the Fernow Experimental Forest[4] (FEF), West Virginia **?**. A total of 88 trees were sampled from plots within two different watersheds on the FEF. Hardwood species sampled include *Acer rubrum*, *Betula lenta*, *Liriodendron tulipifera*, and *Prunus serotina*, all of which were measured in the summer of 1991 and 1992. Data include tree height, diameter, as well as green and dry weight of tree stem, top, small branches, large branches, and leaves. Table **??** shows a subset of these data

The size of this dataset is relatively small, there are no missing observations, the variables are easily understood, etc.

0.2 FACE Experiment Data Set

We often encounter data gleaned from highly structured and complex experiments. Such data typically present challenges in organization/storage, exploratory data analysis (EDA), statistical analysis, and interpretation of analysis results. An example data set comes from the Aspen Free-Air Carbon Diox-

[4] http://www.nrs.fs.fed.us/ef/locations/wv/fernow

TABLE 0.1: A subset of the tree biomass data from the FEF.

species	dbh_in	height_ft	stem_green_kg	leaves_green_kg
Acer rubrum	6.0	48.0	92.2	16.1
Acer rubrum	6.9	48.0	102.3	12.9
Acer rubrum	6.4	48.0	124.4	16.5
Acer rubrum	6.5	49.0	91.7	12.0
Acer rubrum	7.2	51.0	186.2	22.4
Acer rubrum	3.1	40.0	20.8	0.9
Acer rubrum	2.0	30.5	5.6	1.0
Acer rubrum	4.1	50.0	54.1	6.1
Acer rubrum	2.4	28.0	10.2	2.5
Acer rubrum	2.7	40.4	20.2	1.6

ide Enrichment[5] (FACE) Experiment conducted from 1997-2009 on the Harshaw Experimental Forest[6] near Rhinelander, Wisconsin. The Aspen FACE Experiment was a multidisciplinary study that assessed the effects of increasing tropospheric ozone and carbon dioxide concentrations on the structure and functioning of northern forest ecosystems. The design provided the ability to assess the effects of these gasses alone (and in combination) on many ecosystem attributes, including growth, leaf development, root characteristics, and soil carbon. The data set considered here comprises annual tree height and diameter measurements from 1997 to 2008 for *Populus tremuloides*, *Acer saccharum*, and *Betula papyrifera* grown within twelve 30 meter diameter rings in which the concentrations of tropospheric ozone and carbon dioxide were controlled **?**. Because there was no confinement, there was no significant change in the natural, ambient environment other than elevating these trace gas concentrations. Although the basic individual tree measurements are similar to those in the FEF data set we saw in Section~**??**, (i.e., height and diameter), the study design specifies various tree species clones, varying gas treatments, and treatment replicates. Further, because these are longitudinal data, (measurements were recorded over time) the data set presents many missing values as a result of tree mortality. Table **??** contains the first five records as well as 5 more randomly selected records in the data set. Here, a row identifies each tree's experimental assignment, genetic description, and growth over time.

Notice that several height measurements in 2008 contain missing data. If all year measurements were shown, we would see much more missing data. Also, notice that this data set is substantially larger than the FEF data set with 912 rows and 39 columns of data in the full data set.

[5] http://www.nrs.fs.fed.us/disturbance/climate_change/face
[6] http://www.nrs.fs.fed.us/ef/locations/wi/rhinelander/

TABLE 0.2: A small portion of the FACE experiment data set

	Rep	Treat	SPP	Clone	ID..	X1997_Height	X2008_Height
1	1	1	B	B	4360	51.0	632
2	1	1	A	216	4359	58.0	742
3	1	1	B	B	4358	24.0	916
4	1	1	A	216	4357	58.0	981
5	1	1	B	B	4356	41.0	914
183	1	3	B	B	5017	40.0	NA
625	3	1	B	B	6853	55.5	936
835	3	3	B	B	7573	48.0	NA
259	1	4	B	B	5327	48.0	659
96	1	2	A	216	4697	27.0	NA

0.3 PEF Inventory and LiDAR Data Set

Coupling forest inventory with remotely sensed Light Detection and Ranging (LiDAR) data sets using regression models offers an attractive approach to mapping forest variables at stand, regional, continental, and global scales. LiDAR data have shown great potential for use in estimating spatially explicit forest variables over a range of geographic scales (**?**), (**?**), (**?**), (**?**), (**?**). Encouraging results from these and many other studies have spurred massive investment in new LiDAR sensors, sensor platforms, as well as extensive campaigns to collect field-based calibration data.

Much of the interest in LiDAR based forest variable mapping is to support carbon monitoring, reporting, and verification (MRV) systems, such as defined by the United Nations Programme on Reducing Emissions from Deforestation and Forest Degradation[7] (UN-REDD) and NASA's Carbon Monitoring System[8] (CMS) (**?**), (**?**). In these, and similar initiatives, AGB is the forest variable of interest because it provides a nearly direct measure of forest carbon (i.e., carbon comprises $\sim 50\%$ of wood biomass, **?**). Most efforts to quantify and/or manage forest ecosystem services (e.g., carbon, biodiversity, water) seek high spatial resolution wall-to-wall data products such as gridded maps with associated measures of uncertainty, e.g., point and associated credible intervals (CIs) at the pixel level. In fact several high profile international initiatives include language concerning the level of spatially explicit acceptable error in total forest carbon estimates, see, e.g., **?** and **?**.

[7] http://www.un-redd.org
[8] http://carbon.nasa.gov

TABLE 0.3: A small portion of the PEF inventory data set

	MU	plot	easting	northing	biomass.mg.ha	stocking.stems.ha
118	17	24	530304	4965983	112.96	2325
403	31	11	530575	4964959	NA	NA
539	8	23	530004	4967094	71.05	10927
167	19	63	530436	4965217	NA	NA
62	14	21	530218	4966445	NA	NA
410	31	32	530657	4964999	NA	NA
308	27	31	530449	4965815	134.35	7872
471	6	13	529560	4967220	30.33	3016
556	9	14	529601	4966363	140.00	1284
65	14	24	530339	4966652	163.52	433

Here, we consider a data set collected on the Penobscot Experimental Forest[9] (PEF) in Bradley and Eddington, Maine. The dataset comprises LiDAR waveforms collected with the Laser Vegetation Imaging Sensor[10] (LVIS) and several forest variables measured on a set of 589 georeferenced forest inventory plots. The LVIS data were acquired during the summer of 2003. The LVIS instrument, an airborne scanning LiDAR with a 1064 nm laser, provided 12,414 LiDAR pseudo-waveform signals within the PEF. For each waveform, elevations were converted to height above the ground surface and interpolated at 0.3 m intervals. Figure **??** shows PEF LiDAR energy returns at 12 m above the ground, forest inventory plot locations, and management unit boundaries. The forest inventory data associated with each plot were drawn from the PEF's database of several on-going, long-term silvicultural experiments (see **?**). Below we provide a plot containng the geographic coordinates, biomass (mg/ha), basal area (m^2/ha), stocking (trees/ha), diameter class (cm), and management unit. Table **??** shows a subset of data for 10 randomly selected plots (where each row records plot measurements) in the forest inventory data set.

0.4 Zürichberg Forest inventory data set

Measuring tree diameter and height is a time consuming process. This fact makes the Zürichberg Forest inventory data set a rare and impressive investment. These data comprise a complete enumeration of the 589 trees in the

[9] www.fs.fed.us/ne/durham/4155/penobsco.htm
[10] http://lvis.gsfc.nasa.gov

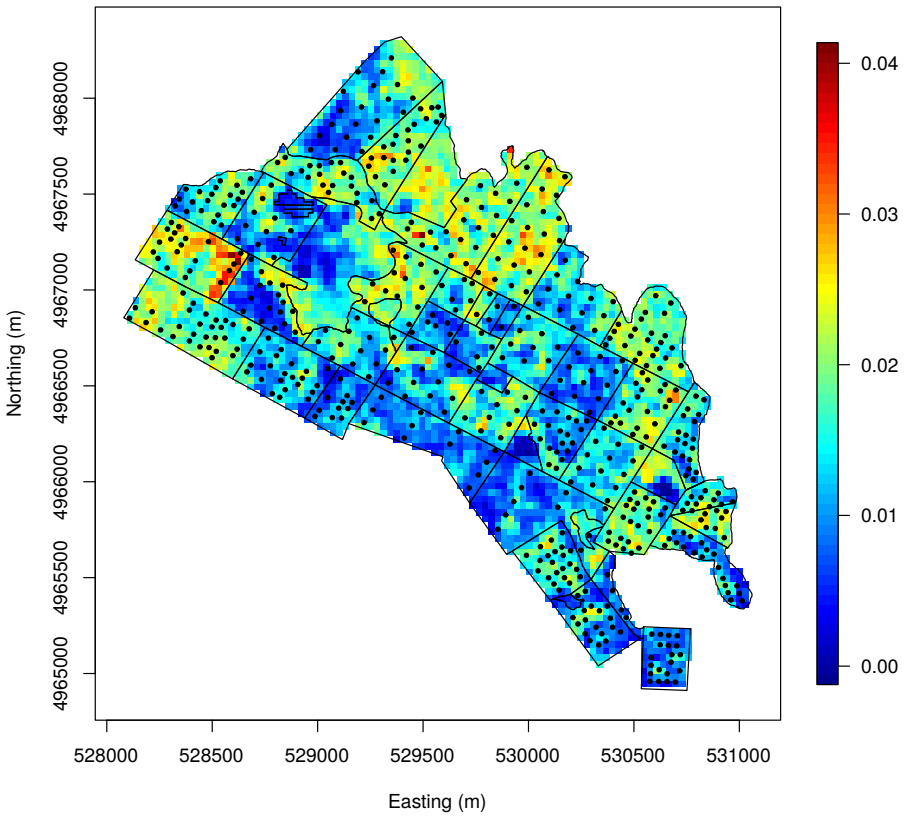


FIGURE 1: Surface of LiDAR energy returns at 12 m above the ground, forest inventory plot locations, and management unit boundaries on the PEF.

Zürichberg Forest, including species, diameter at breast height, basal area, and volume. The stem map colored by species is shown in Figure **??**.

0.5 Looking Forward

The four examples above illustrate a variety of data sets that might be encountered in practice, and each provides its own challenges. For the FACE data, the challenges are more statistical in nature. Complications could arise related to the complex study design and how that design might affect methods of analysis and conclusions drawn from the study. The other data sets present different challenges, such as how to:

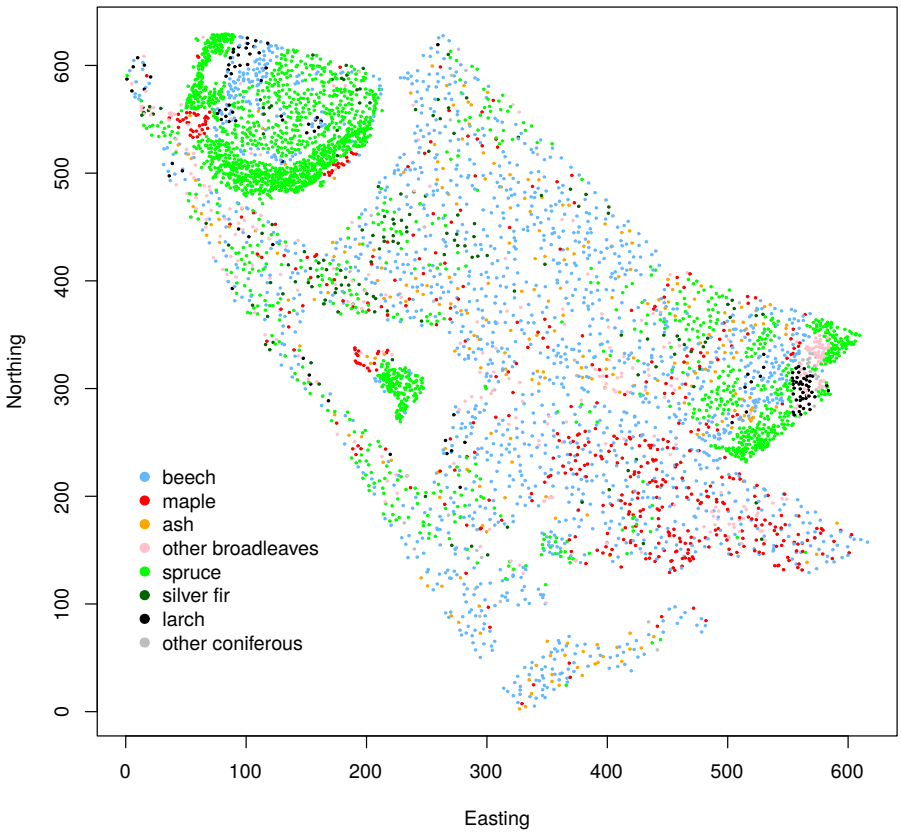


FIGURE 2: Location and species of all trees in the Zürichberg Forest.

1. Develop biomass equations suitable for population inference from the FEF's small sample of 88 trees
2. Work with spatially indexed data in the case of the PEF and Zürichberg inventory data
3. Effectively and efficiently process the PEF's high-dimentional LiDAR signal data for use in predictive models of forest variables.

This book and associated material introduce tools to tackle some of the challenges in working with real data sets within the context of the R statistical system. We will focus on important topics such as

- Obtaining and manipulating data
- Summarizing and visualizing data
- Communicating findings about data that support reproducible research
- Programming and writing functions
- Working with specialized data structures, e.g., spatial data and databases

0.6 How to Learn (The Most Important Section in This Book!)

There are several ways to engage with the content of this book and associated materials.

One way is not to engage at all. Leave the book closed on a shelf and do something else with your time. That may or may not be a good life strategy, depending on what else you do with your time, but you won't learn much from the book!

Another way to engage is to read through the book "passively", reading all that's written but not reading the book with R open on your computer, where you could enter the R commands from the book. With this strategy you'll probably learn more than if you leave the book closed on a shelf, but there are better options.

A third way to engage is to read the book while you're at a computer with R, and to enter the R commands from the book as you read about them. You'll likely learn more this way.

A fourth strategy is even better. In addition to reading and entering the commands given in the book, you think about what you're doing, and ask yourself questions (which you then go on to answer). For example, after working through some R code computing the logarithm of positive numbers you might ask yourself, "What would R do if I asked it to calculate the logarithm of a negative number? What would R do if I asked it to calculate the logarithm of a really large number such as one trillion?" You could explore these questions easily by just trying things out in the R Console window.

If your goal is to maximize the time you have to binge-watch *Stranger Things* Season 2 on Netflix, the first strategy may be optimal. But if your goal is to learn a lot about computational tools for data science, the fourth strategy is probably going to be best.

0

Introduction to R and RStudio

Various statistical and programming software environments are used in data science, including R, Python, SAS, C++, SPSS, and many others. Each has strengths and weaknesses, and often two or more are used in a single project. This book focuses on R for several reasons:

1. R is free
2. It is one of, if not the, most widely used software environments in data science
3. R is under constant and open development by a diverse and expert core group
4. It has an incredible variety of contributed packages
5. A new user can (relatively) quickly gain enough skills to obtain, manage, and analyze data in R

Several enhanced interfaces for R have been developed. Generally such interfaces are referred to as *integrated development environments (IDE)*. These interfaces are used to facilitate software development. At minimum, an IDE typically consists of a source code editor and build automation tools. We will use the RStudio IDE, which according to its developers "is a powerful productive user interface for R.[11] RStudio is widely used, it is used increasingly in the R community, and it makes learning to use R a bit simpler. Although we will use RStudio, most of what is presented in this book can be accomplished in R (without an added interface) with few or no changes.

0.7 Obtaining and Installing R

It is simple to install R on computers running Microsoft Windows, macOS, or Linux. For other operating systems users can compile the source code di-

[11] http://www.rstudio.com/

rectly.[12] Here is a step-by-step guide to installing R for Microsoft Windows.[13] macOS and Linux users would follow similar steps.

1. Go to `http://www.r-project.org/`
2. Click on the `CRAN` link on the left side of the page
3. Choose one of the mirrors.[14]
4. Click on `Download R for Windows`
5. Click on `base`
6. Click on `Download R 3.5.0 for Windows`
7. Install R as you would install any other Windows program

0.8 Obtaining and Installing RStudio

You must install R prior to installing RStudio. RStudio is also simple to install:

1. Go to `http://www.rstudio.com`
2. Click on the link `RStudio` under the `Products` tab, then select the `Desktop` option
3. Click on the `Desktop` link
4. Choose the `DOWNLOAD RSTUDIO DESKTOP` link in the `Open Source Edition` column
5. On the ensuing page, click on the `Installer` version for your operating system, and once downloaded, install as you would any program

0.9 Using R and RStudio

Start RStudio as you would any other program in your operating system. For example, under Microsoft Windows use the Start Menu or double click on the shortcut on the desktop (if a shortcut was created in the installation process). A (rather small) view of RStudio is displayed in Figure **??**.

Initially the RStudio window contains three smaller windows. For now our

[12] Windows, macOS, and Linux users also can compile the source code directly, but for most it is a better idea to install R from already compiled binary distributions.

[13] New versions of R are released regularly, so the version number in Step 6 might be different from what is listed below.

[14] The `http://cran.rstudio.com/` mirror is usually fast. Otherwise choose a mirror in Michigan.

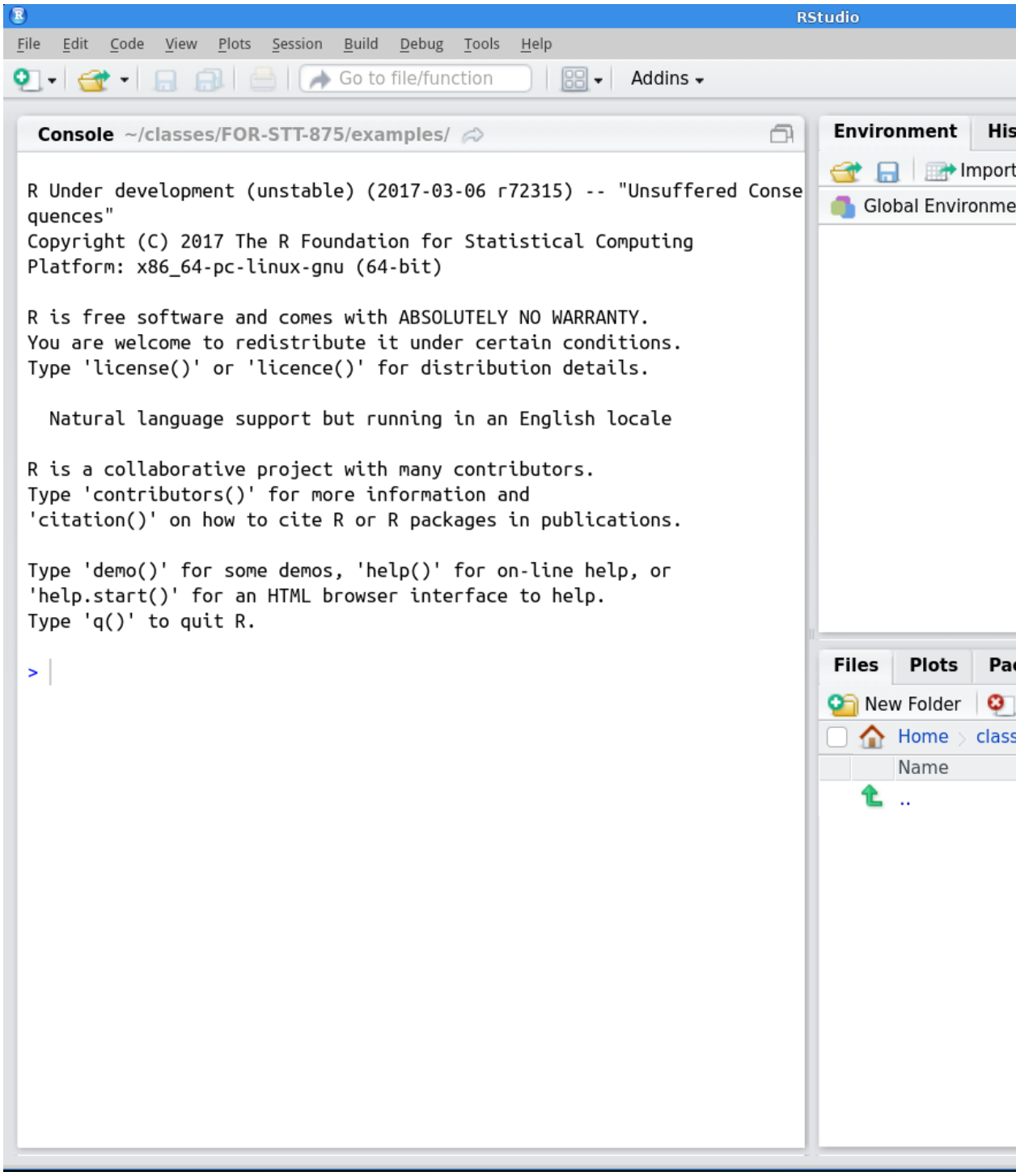


FIGURE 3: The Rstudio IDE.

main focus will be the large window on the left, the `Console` window, in which R statements are typed. The next few sections give simple examples of the use of R. In these sections we will focus on small and non-complex data sets, but of course later in the book we will work with much larger and more complex sets of data. Read these sections at your computer with R running, and enter the R commands there to get comfortable using the R console window and RStudio.

0.9.1 R as a calculator

R can be used as a calculator. Note that `#` is the comment character in R, so R ignores everything following this character. Also, you will see that R prints `[1]` before the results of each command. Soon we will explain its relevance, but ignore this for now. The command prompt in R is the greater than sign `>`.

```
> 34+20*sqrt(100)  ## +,-,*,/ have the expected meanings
```

```
## [1] 234
```

```
> exp(2)  ##The exponential function
```

```
## [1] 7.389
```

```
> log10(100)  ##Base 10 logarithm
```

```
## [1] 2
```

```
> log(100)  ##Base e logarithm
```

```
## [1] 4.605
```

```
> 10^log10(55)
```

```
## [1] 55
```

Most functions in R can be applied to vector arguments rather than operating on a single argument at a time. A *vector* is a data structure that contains elements of the same data type (i.e. integers).

```
> 1:25 ##The integers from 1 to 25
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## [18] 18 19 20 21 22 23 24 25
```

```
> log(1:25) ##The base e logarithm of these integers
```

```
##  [1] 0.0000 0.6931 1.0986 1.3863 1.6094 1.7918 1.9459
##  [8] 2.0794 2.1972 2.3026 2.3979 2.4849 2.5649 2.6391
## [15] 2.7081 2.7726 2.8332 2.8904 2.9444 2.9957 3.0445
## [22] 3.0910 3.1355 3.1781 3.2189
```

```
> 1:25*1:25 ##What will this produce?
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100 121 144
## [13] 169 196 225 256 289 324 361 400 441 484 529 576
## [25] 625
```

```
> 1:25*1:5 ##What about this?
```

```
##  [1]   1   4   9  16  25   6  14  24  36  50  11  24
## [13]  39  56  75  16  34  54  76 100  21  44  69  96
## [25] 125
```

```
> seq(from=0, to=1, by=0.1) ##A sequence of numbers from 0 to 1
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
> exp(seq(from=0, to=1, by=0.1)) ##What will this produce?
```

```
##  [1] 1.000 1.105 1.221 1.350 1.492 1.649 1.822 2.014
##  [9] 2.226 2.460 2.718
```

Now the mysterious square bracketed numbers appearing next to the output make sense. R puts the position of the beginning value on a line in square brackets before the line of output. For example if the output has 40 values, and 15 values appear on each line, then the first line will have `[1]` at the left, the second line will have `[16]` to the left, and the third line will have `[31]` to the left.

0.9.2 Basic descriptive statistics and graphics in R

Of course it is easy to compute basic descriptive statistics and to produce standard graphical representations of data. For illustration consider the first 14 observations of tree height and DBH (diameter at breast height) from the FEF data set. We will begin by entering these data "by hand" using the `c()` function, which concatenates its arguments into a vector. For larger data sets we will clearly want an alternative way to enter data.

A style note: R has two widely used methods of assignment: the left arrow, which consists of a less than sign followed immediately by a dash: `<-` and the equals sign: `=`. Much ink has been used debating the relative merits of the two methods, and their subtle differences. Many leading R style guides (e.g., the Google style guide at https://google.github.io/styleguide/Rguide.xml and the Bioconductor style guide at http://www.bioconductor.org/developers/how-to/coding-style/) recommend the left arrow `<-` as an assignment operator, and we will use this throughout the book.

Also you will see that if a command has not been completed but the ENTER key is pressed, the command prompt changes to a `+` sign.

```
> dbh <- c(6, 6.9, 6.4, 6.5, 7.2, 3.1, 2, 4.1, 2.4, 2.7, 3.7, 6.3, 5.2, 5.1, 6.4)
> ht <- c(48, 48, 48, 49, 51, 40, 30.5, 50, 28, 40.4, 42.6, 53, 55, 50, 50)
> dbh
```

```
##  [1] 6.0 6.9 6.4 6.5 7.2 3.1 2.0 4.1 2.4 2.7 3.7 6.3
## [13] 5.2 5.1 6.4
```

```
> ht
```

```
##  [1] 48.0 48.0 48.0 49.0 51.0 40.0 30.5 50.0 28.0 40.4
## [11] 42.6 53.0 55.0 50.0 50.0
```

Next we compute some descriptive statistics for the two numeric variables

```
> mean(dbh)
```

```
## [1] 4.933
```

```
> sd(dbh)
```

```
## [1] 1.782
```

```
> summary(dbh)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    3.40    5.20    4.93    6.40    7.20
```

```
> mean(ht)
```

```
## [1] 45.57
```

```
> sd(ht)
```

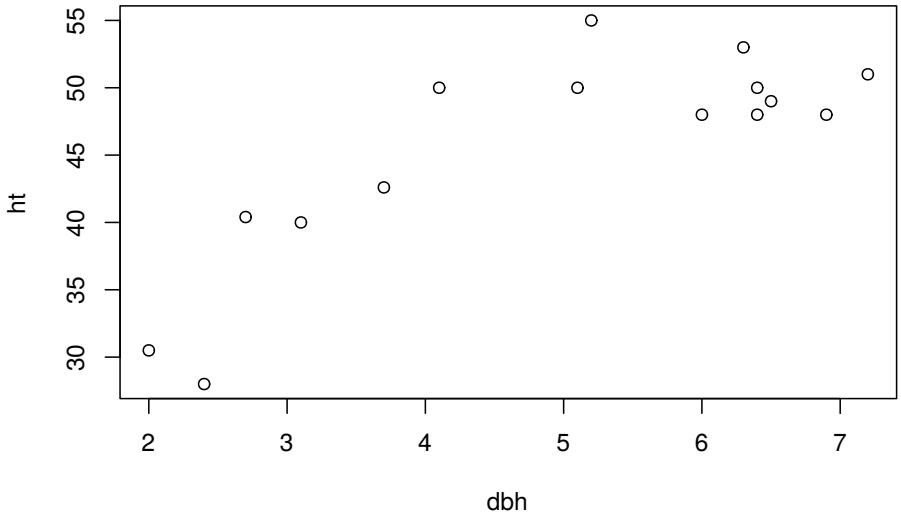
```
## [1] 7.857
```

```
> summary(ht)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    28.0    41.5    48.0    45.6    50.0    55.0
```

Next, a scatter plot of `dbh` versus `ht`:

```
plot(dbh, ht)
```



Unsurprisingly as DBH increases, height tends to increase. We'll investigate this further using simple linear regression in the next section.

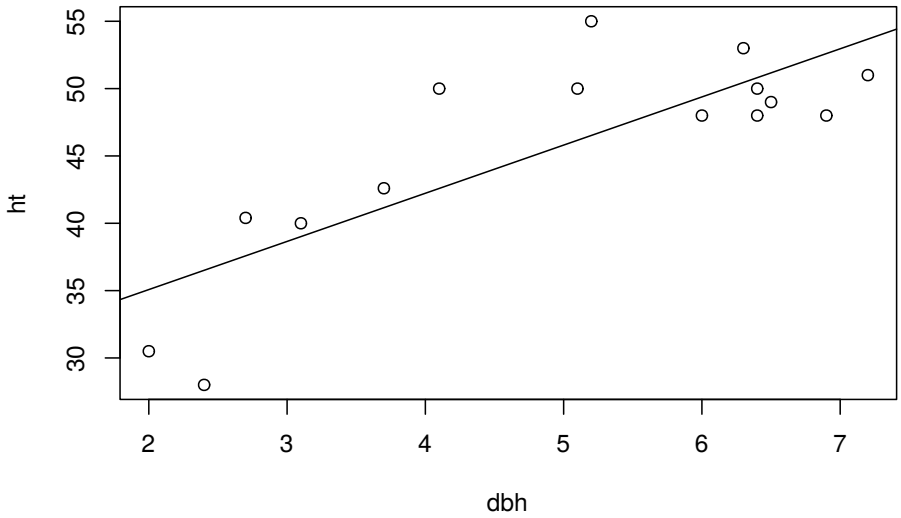
0.9.3 Simple linear regression in R

The `lm()` function is used to fit linear models in R, including simple linear regression models. Here it is applied to the DBH height data.

```
> ht.lm <- lm(ht ~ dbh) ##Fit the model and save it in ht.lm
> summary(ht.lm)  ##Basic summary of the model
```

```
##
## Call:
## lm(formula = ht ~ dbh)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.507 -2.742 -0.812  2.683  8.480
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   27.925      3.739    7.47  4.7e-06 ***
## dbh            3.576      0.716    5.00  0.00024 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.77 on 13 degrees of freedom
## Multiple R-squared:  0.658,  Adjusted R-squared:  0.631
## F-statistic:   25 on 1 and 13 DF,  p-value: 0.000244
```

```
> plot(dbh, ht) ##Scatter plot of the data
> abline(ht.lm) ##Add the fitted regression line to the plot
```



We will work extensively with such models later in the text. We will also talk about why it might not be a good idea to assume a linear relationship between DBH and height—can you guess why this is by looking at the data scatter and model fitted line in the plot above?

0.10 How to Learn

There are several ways to engage with the content of this book and associated learning materials.

A comprehensive, but slightly overwhelming, cheatsheet for RStudio is available here `https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf`. As we progress in learning R and RStudio, this cheatsheet will become more useful. For now you might use the cheatsheet to locate the various windows and functions identified in the coming chapters.

0.11 Getting help

There are several free (and several not free) ways to get R help when needed.

Several help-related functions are built into R. If there's a particular R function of interest, such as `log`, `help(log)` or `?log` will bring up a help page for that function. In RStudio the help page is displayed, by default, in the `Help` tab in the lower right window.[15] The function `help.start` opens a window which allows browsing of the online documentation included with R. To use this, type `help.start()` in the console window.[16] The `help.start` function also provides several manuals online and can be a useful interface in addition to the built in help.

Search engines provide another, sometimes more user-friendly, way to receive answers for R questions. A Google search often quickly finds something written by another user who had the same (or a similar) question, or an online tutorial that touches on the question. More specialized is `https://rseek.org/`, which is a search engine focused specifically on R. Both Google and `https://rseek.org` are valuable tools, often providing more user-friendly information then R's own help system.

In addition, R users have written many types of contributed documentation. Some of this documentation is available at `http://cran.r-project.org/other-docs.html`. Of course there are also numerous books covering general and specialized R topics available for purchase.

0.12 Workspace, working directory, and keeping organized

The *workspace* is your R session working environment and includes any objects you create. Recall these objects are listed in the `Global Environment` window. The command `ls()`, which stands for list, will also list all the objects in your workspace (note, this is the same list that is given in the `Global Environment` window). When you close RStudio, a dialog box will ask you if you want to save an image of the current workspace. If you choose to save your workspace, RStudio saves your session objects and information in a `.RData` file (the period makes it a hidden file) in your *working directory*. Next time you start R or RStudio it checks if there is a `.RData` in the working directory, loads it if it exists, and your session continues where you left off. Otherwise R starts

[15] There are ways to change this default behavior.

[16] You may wonder about the parentheses after `help.start`. A user can specify arguments to any R function inside parentheses. For example `log(10)` asks R to return the logarithm of the argument 10. Even if no arguments are needed, R requires empty parentheses at the end of any function name. In fact if you just type the function name without parentheses, R returns the definition of the function. For simple functions this can be illuminating.

with an empty workspace. This leads to the next question—what is a working directory?

Each R session is associated with a working directory. This is just a directory from which R reads and writes files, e.g., the `.RData` file, data files you want to analyze, or files you want to save. On Mac when you start RStudio it sets the working directory to your home directory (for me that's `/Users/andy`). If you're on a different operating system, you can check where the default working directory is by typing `getwd()` in the console. You can change the default working directory under RStudio's `Global Option` dialog found under the `Tools` dropdown menu. There are multiple ways to change the working directory once an R session is started in RStudio. One method is to click on the `Files` tab in the lower right window and then click the `More` button. Alternatively, you can set the session's working directory using the `setwd()` in the console. For example, on Windows `setwd("C:/Users/andy/for472/exercise1")` will set the working directory to `C:/Users/andy/for472/exercise1`, assuming that file path and directory exist (Note: Windows file path uses a backslash, `\`, but in R the backslash is an escape character, hence specifying file paths in R on Windows uses the forward slash, i.e., `/`). Similarly on Mac you can use `setwd("/Users/andy/for472/exercise1")`. Perhaps the most simple method is to click on the `Session` tab at the top of your screen and click on the `Set Working Directory` option. Later on when we start reading and writing data from our R session, it will be very important that you are able to identify your current working directory and change it if needed. We will revisit this in subsequent chapters.

As with all work, keeping organized is the key to efficiency. It is good practice to have a dedicated directory for each R project or exercise.

0.13 Quality of R code

Writing well-organized and well-labeled code allows your code to be more easily read and understood by another person. (See xkcd's take on code quality in Figure **??**.) More importantly, though, your well-written code is more accessible to you hours, days, or even months later. We are hoping that you can use the code you write in this class in future projects and research.

Google provides style guides for many programming languages. You can find the R style guide here[17]. Below are a few of the key points from the guide that we will use right away.

[17] https://google.github.io/styleguide/Rguide.xml

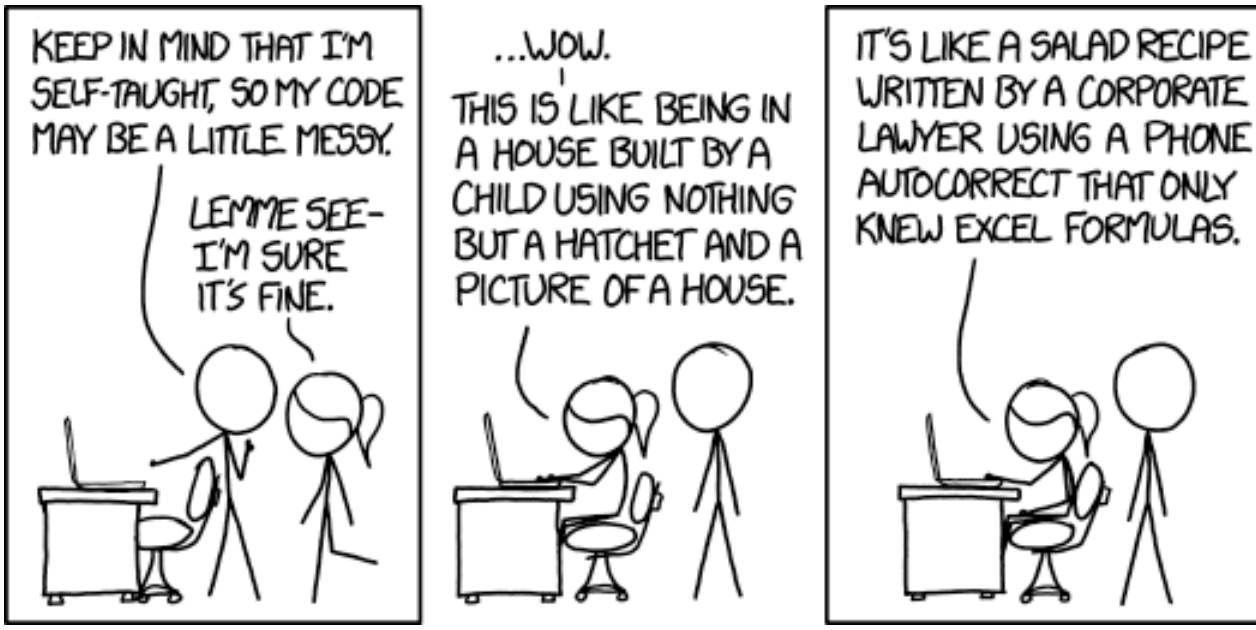


FIGURE 4: xkcd: Code Quality

0.13.1 Naming Files

File names should be meaningful and end in `.R`. If we write a script that analyzes a certain species distribution:

- GOOD: `african_rhino_distribution.R`
- GOOD: `africanRhinoDistribution.R`
- BAD: `speciesDist.R` (too ambiguous)
- BAD: `species.dist.R` (too ambiguous and two periods can confuse operating systems' file type auto-detect)
- BAD: `speciesdist.R` (too ambiguous and confusing)

0.13.2 Naming Variables

- GOOD: `rhino.count`
- GOOD: `rhinoCount`
- GOOD: `rhino_count` (We don't mind the underscore and use it quite often, although Google's style guide says it's a no-no for some reason)
- BAD: `rhinocount` (confusing)

0.13.3 Syntax

- Keep code lines under 80 characters long.
- Indent your code with two spaces. (RStudio does this by default when you press the TAB key.)

0

Scripts, R Markdown, and Reproducible Research

Doing work in data science, whether for homework, a project for a business, or a research project, typically involves several iterations. For example, creating an effective graphical representation of data can involve trying out several different graphical representations, and then tens if not hundreds of iterations when fine-tuning the chosen representation. And each of these representations may require several R commands to create. Although this all could be accomplished by typing and re-typing commands at the R Console, it is easier and more effective to write the commands in a *script file* that can then be submitted to the R console either a line at a time or all together.[18]

In addition to making the workflow more efficient, R scripts provide another large benefit. Often we work on one part of a homework assignment or project for a few hours, then move on to something else, and then return to the original part a few days, months, or sometimes even years later. In such cases we may have forgotten how we created a graphical display that we were so proud of, and will again need to spend a few hours to recreate it. If we save a script file, we have the ingredients immediately available when we return to a portion of a project.[19]

Next consider a larger scientific endeavor. Ideally a scientific study will be reproducible, meaning that an independent group of researchers (or the original researchers) will able to duplicate the study. Thinking about data science, this means that all the steps taken when working with the data from a study should be reproducible, from selection of variables to formal data analysis. In principle this can be facilitated by explaining, in words, each step of the work with data. In practice, on the other hand, it is typically difficult or impossible to reproduce a full data analysis based on a written explanation. It is much more effective to include the actual computer code that accomplished the data work in the report, whether the report is a homework assignment or a research paper. Tools in R such as *R Markdown* facilitate this process.

[18] Unsurprisingly it is also possible to submit several selected lines of code at once.

[19] In principle the R history mechanism provides a similar record. But with history we have to search through a lot of other code to find what we're looking for, and scripts are a much cleaner mechanism to record our work.

0.14 Scripts in R

As noted above, scripts help to make working with data more efficient and provide a record of how data were managed and analyzed. Here we describe an example using the FEF data.[20] First we read the FEF data into R using the code below.

```
> face.dat <- read.csv(
+     file="http://blue.for.msu.edu/FOR472/data/FACE_aspen_core_growth.csv"
+ )
```

Next we print the names of the variables in the data set. Don't be concerned about the specific details. Later we will learn much more about reading in data and working with data sets in R.

```
> names(face.dat)
```

```
##  [1] "Rep"                "Treat"
##  [3] "Clone"              "E.Clone"
##  [5] "Row"                "Col"
##  [7] "ID.."               "X1997Initial_Height"
##  [9] "X1997Initial_Diam"  "X1997Final_Height"
## [11] "X1997Final_Diam"    "X1998_Height"
## [13] "X1998_Diam"         "X1999_Height"
## [15] "X1999_Diam"         "X2000_Height"
## [17] "X2000_Diam"         "X2001_Height"
## [19] "X2001_AvgDiam"      "X2001_Diam.3cm"
## [21] "X2001_Diam.10cm"    "X2002_Height"
## [23] "X2002_Diam.10cm"    "X2003_Height"
## [25] "X2003_Diam.10cm"    "X2003_DBH"
## [27] "X2004_Height"       "X2004_Diam.10cm"
## [29] "X2004_DBH"          "X2005_Height"
## [31] "X2005_Diam.10cm"    "X2005_DBH"
## [33] "X2006_Height"       "X2006_DBH"
## [35] "X2007_Height"       "X2007_DBH"
## [37] "X2008_Height"       "X2008_DBH"
## [39] "Notes"              "Comment1"
## [41] "Comment2"           "Comment3"
## [43] "Comment4"           "Comment5"
```

[20] The example uses features of R that we have not yet discussed, so don't worry about the details but rather about how it motivates the use of a script file.

```
## [45] "Comment6"
```

Let's create a scatter plot of 2008 DBH versus height. To do this we'll first create variables for DBH and height taken in the year 2008 and print out the first ten values of each variable.[21]

```
> dbh <- face.dat$X2008_DBH
> ht <- face.dat$X2008_Height
> dbh[1:10]
```

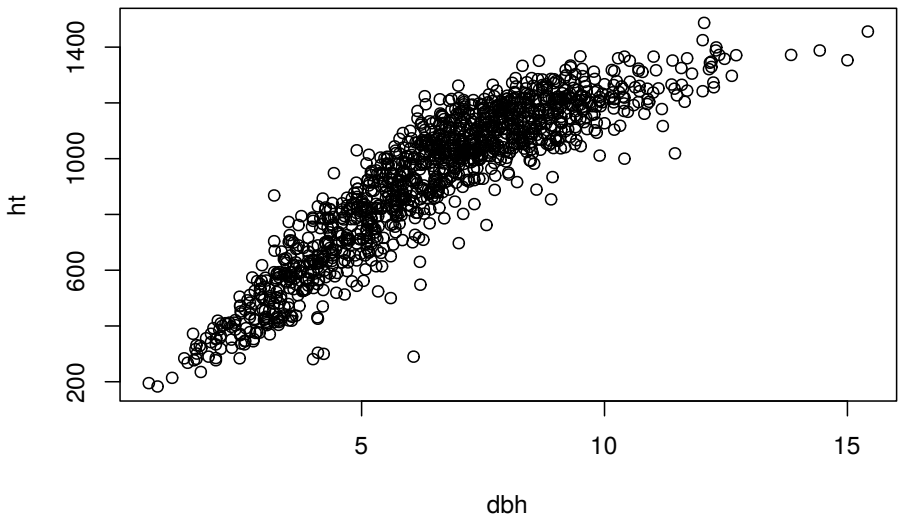
```
##  [1]   NA 9.55 2.00 9.00 3.11 6.35 4.60   NA   NA 1.42
```

```
> ht[1:10]
```

```
##  [1]   NA 1225  334 1079  370  859  818   NA   NA  268
```

The `NA` is how missing data are represented in R. Their presence here suggests several trees in this data set are dead or not measured for some reason in 2008. Of course at some point it would be good to investigate which trees have missing data and why. The `plot()` function in R will omit missing values, and for now we will just plot the non-missing data. A scatter plot of the data is drawn next.

```
> plot(dbh, ht)
```



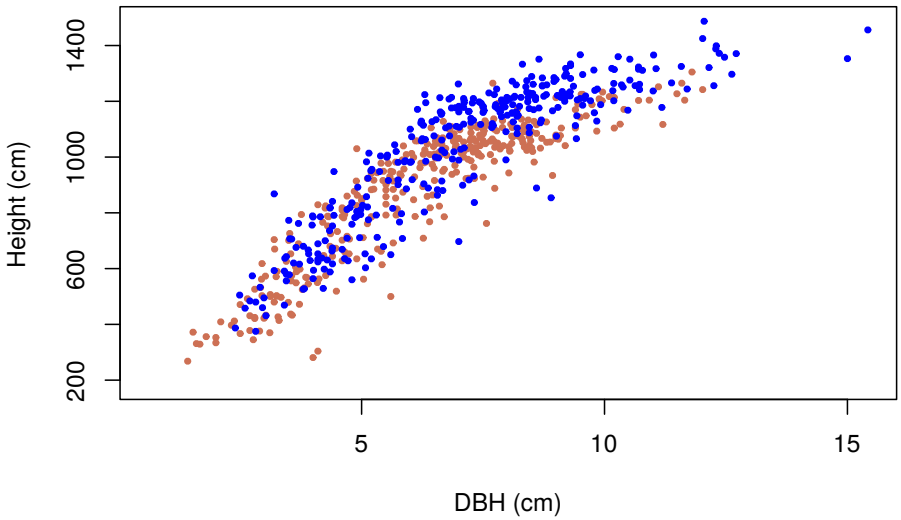
[21] Neither of these steps are necessary, but are convenient for illustration.

Not surprisingly, the scatter plot shows that DBH and height are positively correlated and the relationship is nonlinear. Now that we have a basic scatter plot, it is tempting to make it more informative. We will do this by adding a feature that identifies which trees belong to the control and elevated CO_2 environment treatments. We do this by first separating DBH and height into their respective treatment groups.

```
> treat <- face.dat$Treat
> dbh.treat.1 <- dbh[treat==1] ##Treatment 1 is the control
> ht.treat.1 <- ht[treat==1]
>
> dbh.treat.2 <- dbh[treat==2] ##Treatment 2 is the elevated CO2
> ht.treat.2 <- ht[treat==2]
```

To make a more informative scatter plot we will do two things. First make a plot for treatment 1 data, but ensure the plot region is large enough to include the treatment 2 data. This is done by specifying the range of the plot axes via `xlim` and `ylim` arguments in the `plot()` function. Here the `xlim` and `ylim` are set to the range of `dbh` and `ht` values, respectively, using `range()` (try and figure out what the `na.rm` argument does in the range function). Second we add treatment 2 data via the `points()` function. There are several other arguments passed to the plot function, but don't worry about these details for now.

```
> plot(dbh.treat.1, ht.treat.1, xlim=range(dbh, na.rm=TRUE),
+      ylim=range(ht, na.rm=TRUE), pch=19, col="salmon3", cex=0.5,
+      xlab="DBH (cm)", ylab="Height (cm)")
> points(dbh.treat.2, ht.treat.2, pch=19, col="blue", cex=0.5)
```



Of course we should have a plot legend to tell the viewer which colors are associated with the treatments, as well as many other aesthetic refinements. For now, however, we will resist such temptations.[22]

Some of the process leading to the completed plot is shown above. We read in the data, created an intermediate plot by adding treatment identifiers, creating variables representing the 2008 measurements of DBH and height, and so on. However, a lot of the process isn't shown. For example, I made several mistakes in the process of getting the code and plot the way I wanted it—forgot the `na.rm=TRUE` initially then fiddled around with the treatment colors a bit.

Now imagine trying to recreate the plot a few days later. Possibly someone saw the plot and commented that it would be interesting to see similar plots for each year in the study period. If we did all the work, including all the false starts and refinements, at the console it would be hard to sort things out. This would take much longer than necessary to create the new plots. This would be especially true if a few months had passed, rather than just a few days.

Creating the new scatter plots would be much easier with a script file, especially if it had a few well-chosen comments. Fortunately it is quite easy to create and work with script files in RStudio.[23] Just choose `File > New File > New script` and a script window will open up in the upper left of the full RStudio window.

An example of a script window (with some R code already typed in) is shown

[22] As an aside, by only looking at the plotted data and thinking about basic plant physiology, can you guess which color is associated with the elevated CO_2 treatment?

[23] It is also easy in R without RStudio. Just use `File > New script` to create a script file, and save it before exiting R.