Bayesian GLMs in R for Ecology

Mark Warren & Carl Smith

November 2021

Contents

"*The truth is not for all men, but only for those who seek it.*"

Ayn Rand

Preface

Our goal is to produce a set of accessible, inexpensive statistics guides for undergraduate and post-graduate students that are tailored to specific fields and that use R. These books present minimal statistical theory and are intended to allow students to understand the process of data exploration and model fitting and validation using datasets comparable to their own and, thereby, encourage the development of statistical skills.

To obtain the R script and data associated with each book chapter, type the following into your browser and download the data and script files:

https://www.dropbox.com/s/ofljovoiyp5u6ut/DataScript.zip

All profits from this book will be donated to the SAS Regimental Association.

Contributors

Mark Warren

Environment Agency, Tewkesbury GL20 8JG, UK

email: mark.warren@environment-agency.gov.uk

Carl Smith

Department of Ecology & Vertebrate Zoology, University of Łódź, 12/16 Banacha Street, 90-237 Łódź, Poland

and

Institute of Vertebrate Biology, Academy of Sciences of the Czech Republic, Květná 8, 603 65 Brno, Czech Republic

email: carl.smith@biol.uni.lodz.pl

Cover art

The cover art is the work of Laura Andrew (www.lauraandrew.com). Laura is located in central Lincoln, UK. After studying and working as an illustrator in London, Laura returned to her roots in Lincolnshire where she produces her art, and offers courses and workshops to people looking to learn new skills such as watercolour, oil painting and printmaking. Much of her art is inspired by the natural world, particularly birds. Working professionally as both an artist and illustrator Laura sells her art worldwide and her paintings have been exhibited in galleries locally and in London.

Chapter 1

Introduction to R and RStudio

In this chapter we will introduce R, which is a programming language for data analysis and graphics, and RStudio, which is an Integrated Development Environment (IDE) that allows you to interact more easily with R. We highly recommend using RStudio as your console for R. As you become familiar with it you will learn more of its functionality.

The advent of the statistical software package R has contributed substantially to an improvement in the quality and sophistication of data analyses performed in a range of scientific fields, including ecology. While not intuitive to use, R has become the industry standard, and time invested in learning to master R will be rewarded with an improved understanding of how to handle and model data. There are several benefits to using R. First, it is extremely flexible and permits analysis of almost any type of data. Second, there are extremely efficient packages that permit the import of various data types, joining and transforming data, and visualising data. R readily permits the sharing of code with collaborators or journal reviewers and can be archived with corresponding datasets for others to use and improve upon. Finally, R is freely distributed under General Public License for all major computing platforms (Windows, MacOS and Linux), and under continuous development by a large community of scientists.

To install the R software on your computer go to: https://www.r-

project.org and follow the instructions for downloading the latest version.

To install RStudio go to https://rstudio.com and download 'RStudio Desktop.'

You will need to install the R software before installing RStudio.

Once both are installed always start any R session by opening RStudio (R will be opened automatically).

1.1 Getting started with R and RStudio

1.1.1 Basic points

- R is command-line driven.
- It requires you to type commands after a command prompt (>) that appears when you open R. After typing a command in the R console and pressing 'Enter' on your keyboard, the command will run. If your command is not complete, R issues a continuation prompt ('+').
- You can also write *script* in the script window, and select a *command*, and click the Run button. This R script can be saved.
- Finally, you can also import R script - either saved by you previously, or written by someone else.
- R is case sensitive. Take care with spelling and capitalization.
- Commands in R are called 'functions' (see Section @ref(functions)).
- The up arrow (^) on your keyboard can be used to bring up previous commands that you have typed in the R console.
- The dollar symbol `$` is used to select a particular column within your data (called a 'dataframe') (e.g. "df$var1").
- You can include text in your script that R will not execute by including the 'hash tag' # symbol. R ignores the remainder of the script line following #. Using # enables you to insert comments and instructions in your script, or to make modifications to your analysis by 'hashing out' lines of script.

1.1.2 Navigating RStudio

The RStudio interface comprises four windows, which by default are organised as follows:

1. Bottom Left: *Console/Terminal/Jobs* window. For now we will just focus on the *Console* tab. Here you can directly enter commands after the ">" prompt. This is where you will see R execute your commands and where results will appear. However you cannot save anything written here. Instead, it is more efficient to write your commands on the *Source* window (see below) and leave this window for results.

2. Top Left: *Source* window. Here you can write commands (organised as script) that can be edited and saved. If no script has been selected the window will appear as "Untitled." It is in the Source window that you will do all your work - writing and editing script, and pasting in script from other sources. The contents of the entire window can be selected (CTRL+A) and then executed using the "Run" command at the top right of the window. Alternatively, you can run script one line at a time by placing the cursor anywhere on a line of interest and clicking RUN. You can also run a line of script from the keyboard using CTRL+ENTER. When a line of script is run in the Source window you will see that it is sent to the *Console* window to be executed.

If you cannot see this window just open it with:

File » NewFile » RScript

3. Top right: *Environment/History* window. If you select Environment you can see the data and values R has in its memory. If you select History you can see a history of what has been executed in the *Console* window.

4. Bottom right: *Files/Plots/Packages/Help/Viewer* window. Here you can open, delete, rename files, create folders, view current and previous plots, install and load packages or access help.

1.1.3 Basic settings in RStudio

The following are our recommendations about how to initially set up RStudio.

To access the setting menu (once you have opened RStudio):

Tools » Global Options

Select the 'General' tab and deselect the 'Restore .RData into workspace at startup,' and set 'Save workspace to .RData on exit' to 'Never.'

This procedure ensures that the content of any previous R session is not stored or reloaded between R sessions, which guarantees that the R session is 'clean' to begin with and does not have unexpected objects or settings that might interfere with the new code you will input.

Next, within 'Global Options' select 'Code.' We suggest you go to the 'Execution' section and change the drop-down menu after 'Ctrl+Enter executes:' to 'Current line'; as a beginner we recommend that you run and read R script line-by-line to better understand it.

Finally, still within 'Global Options,' select 'Appearance.' Here you can change font style, font size and the editor theme to whatever you find most pleasing to the eye under different working conditions and your personal preference. For example, the 'Editor theme' of 'Vibrant Ink' works well if your computer screen is in sunlight, whereas 'Xcode' works well under low light conditions.

1.1.4 Basic principles in R

In Section @ref(basic) we mentioned the term *command*, which is an instruction given to R and stored/written in a Script, which can be saved as a file to be shared with others. Commands can be simple mathematical operations; i.e. 1 + 1, or can be a complex set of instructions that will execute a full analysis of your data. In the latter case, R users like us are not expected to work out the different steps to carry out the analyses (e.g. a t-test), instead we call/request a function, which can be thought of as a data analysis method. Conceptually this is no different to using other software and pressing a menu button to run a function, except in R you run it via code. Functions are designed by advanced R

users, often called developers, and one day you will probably write your own functions to help speed up your analyses, which is another reason why R is so useful. We will routinely use functions in future sessions, so this concept will become clearer (see Section @ref(functions)). Some developers compile sets of functions into packages that are designed for particular types of data (i.e. data that contain lots of zeros) or analyses (i.e. for analysing evolutionary trees). See Section @ref(functions) for a fuller explanation of functions and packages.

One of the great attractions about learning R for data analysis is that thousands of developers are continually working to develop new functions and packages. Remarkably, they do this work entirely for free. Producers of commercial statistics packages, such as SPSS and Minitab, employ their own developers, but fewer than voluntarily contribute to R. The outcome is that more cutting-edge statistical methods are available to R users than those reliant on commercially produced statistical software.

Objects

R is referred to as an object-based programming language. Statistical analyses are based around creating and manipulating objects. Creating objects is straightforward, thus with the script:

```
a <- 1
```

We create an object 'a' with the value of 1. The symbol '<-' is termed the assignment operator, and it assigns whatever is on its right-hand side to whatever is on its left-hand side. It is a type of function (see Section @ref(functions)). The assignment operator keyboard shortcut is ALT and'-' (Windows) or Option and '-' (Mac). Remember this shortcut and use it to save typing and time!

We can examine the value of object a by typing its name in the Editor/Script window and clicking RUN (or CTRL+ENTER on the keyboard):

```
a
```

Which will return in the Console window:

```
1
```

Objects can be manipulated, thus:

```
a + 1
```

Which returns:

2

Unfortunately, this exciting result has now been lost (to recreate it we would need to type a + 1 again). Statisticians pride themselves on being efficient (lazy), so to reliably recreate the same outcome we can make another object:

```
b <- a + 1
```

We have created an object 'b' with the value a + 1. We can recall the value of object b by typing its name in the Editor/Script window (and CTRL + ENTER):

```
b
```

Which will return in the Console window:

2

Objects can contain anything: values (as above), vectors, matrices, dataframes, lists, graphs, tables, even R script.

The most common data objects in R are *vectors* and *dataframes*. A vector is a single column in a spreadsheet and will often be classed as either numeric, character or logical.

Consider two vectors representing the abundance of five fish species captured in a stretch of the River Pilica in Poland:

species	frequency
pike	40
roach	99
chub	31
perch	35
asp	0

The vector *species* is a character vector and *frequency* is a numeric vector.

It is easy to input a vector object in R, just write the following in the Editor/Script window and run (CTRL + ENTER):

```
freq <- c(40,99,31,35,0)
```

We have created a vector object 'freq.' The 'c()' expression is referred to as the 'concatenate' function, combining all the elements in the parentheses into a vector. Confirm the object contains the correct values by typing:

```
freq
```

Which will return in the Console window:

40, 99, 31, 35, 0

To check the class of vector type:

```
class(freq)
```

Which returns:

numeric

We can similarly create an object vector to represent categories:

```
species <- c("pike", "roach", "chub", "perch", "asp")
```

Confirm the object contains the values by typing:

```
species
```

Which will return in the Console window:

pike, roach, chub, perch, asp

Another type of object is a *dataframe*, what you might consider as a 'table' of information. You can create dataframes by combining vectors. So:

```
spec_freq <- data.frame(species,freq)
```

Confirm the object contains the values by typing:

```
spec_freq
```

Again, use the `class()` function to see what type of object `spec_freq` is.

Dataframes are the fundamental objects used within our statistical analyses. In reality, we usually do not create dataframes by typing data directly into R when we want to analyse them. Instead, we import them directly from their source. But first, we need to set up where we will not only access our data but also our code, outputs and other documentation associated with particular analyses.

A note on tibbles

Although we have mentioned dataframes, there also exist special types of dataframe called *tibbles*. Tibbles are dataframes that have been tweaked to overcome older R behaviours and make life a little easier. For now simply view tibble as an alias for dataframe and for brevity we will use the term *dataframe* throughout to cover both. There is a tibble package, part of the tidyverse set of packages, which we use in this book. A good starting point to become familiar with *tidyverse* packages and tidy coding style is the book *R for Data Science* by Hadley Wickham & Garrett Grolemund (2016).

1.1.5 Working with RStudio projects

When we started using R in 2011, the insightful developers at RStudio had only just released a beta version of the IDE. Version 1.0 was released in 2016 and since then the IDE and the 'ecosystem' of packages and add-ins has grown enormously to allow anyone to use R in a more organised way. Here we want to highlight what we believe is fundamental to an efficient workflow with the IDE; *projects*.

RStudio projects make it straightforward to divide and manage your work into multiple contexts, each with their own working directory, workspace, history, and source documents. Projects are associated with particular working directories that you set up, basically a folder somewhere on your computer or network where you store your data. Once a project is set up and associated with a folder (working directory) your data, R code and outputs from R will be stored there. Projects set up a neat and easy to use mini environment for your work and using them is good practice.

1.1.5.1 Creating projects

RStudio projects are associated with R working directories. You can create an RStudio project:

File » New Project

You then have the following options:

- In a brand new directory

- In an existing directory where you already have R code and data
- By cloning a version control (Git or Subversion) repository (this is beyond the scope of this book but will feature in our forthcoming book *Reproducible Research in R*).

When a new project is created RStudio will:

- Create a project file (with an .Rproj extension) within the project directory. This file can be used as a shortcut for opening the project directly from the filesystem.
- Create a hidden directory (named .Rproj.user) where project-specific temporary files (e.g. auto-saved source documents, window-state, etc.) are stored.
- Load the project into RStudio and display its name in the Projects toolbar (which is located on the far right side of the main toolbar)

1.1.5.2 Opening and closing projects

There are several ways to open a project but the most common are:

- Using the Open Project command (available from both the Projects menu and the Projects toolbar) to browse for and select an existing project file (e.g. MyProject.Rproj).
- Selecting a project from the list of most recently opened projects (also available from both the Projects menu and toolbar).

When a project is opened within RStudio the following actions are taken:

- A new R session (process) is started
- The .Rprofile file in the project's main directory is sourced by R
- The .Rhistory file in the project's main directory is loaded into the RStudio History pane (and used for Console Up/Down arrow command history).
- Previously edited source documents are restored into editor tabs
- Other RStudio settings (e.g. active tabs, splitter positions, etc.) are restored to where they were the last time the project was closed.

When you are within a project and choose to either Quit, close the project, or open another project the following actions are taken:

- .RData and/or .Rhistory are written to the project directory (if current options indicate they should be)
- The list of open source documents is saved (so it can be restored next time the project is opened)
- Your RStudio settings are saved.
- The R session is terminated.

1.1.5.3 Working with multiple projects

You can work with more than one RStudio project at a time by simply opening each project in its own instance of RStudio. The simplest way to accomplish this is:

File » Open Project in New Session

You can then navigate to the working directory (or folder) where the project is saved. Then select the project file (.Rproj) and select open in the dialogue box. A new RStudio and R session will be available in a new window. This is helpful when you are starting an analysis that is similar to one you did a while back, having the older project open means you can copy/paste similar code into new project scripts.

1.1.6 Importing data

Once you have created a project associated with a directory that contains data, you can import them into R from a variety of formats, such as .csv, .txt, .xls, etc. For simplicity, we will work with tab-delimited files (.txt) and comma-separated files (.csv).

We will start by importing a set of data on Eurasian blue tit (*Cyanistes caeruleus*) nesting success from Wytham Woods. Wytham Woods is a tract of ancient woodland in Oxfordshire, UK that has been managed by the University of Oxford since 1942. It is the most intensively researched area of woodland in the world and has been used for pioneering ecological research for decades.

Blue tits are abundant in Wytham Woods and readily use artificial nest boxes placed by researchers. These data (note that 'data' is a plural

word - the singular of 'data' is 'datum') are a subset of 438 records collected during the breeding seasons of 2001–2003 in Wytham Woods. The aim of data collection was to investigate which variables predicted variation in size of nests. An analysis of the full dataset has been published previously (ONeill et al., 2018).

Data for blue tit nests are saved in the tab-delimited file 'cyan.txt' and can be imported into a dataframe in R using the command:

```
cyan <- read.table(file = "cyanistes.txt", header = TRUE,
dec = ".")
```

After importing the dataframe, select Environment in the *Environment / History* window (top right) and you will see an entry for 'cyan' showing that R now has the data stored in its memory.

1.1.7 Functions and packages

Commands performed on values, vectors, objects, and other structures in R are executed using *functions*. A function is a type of object.

Every function has the form `function.name()` with arguments given inside the brackets. Functions require you to give at least one argument.

Many functions in R come in *packages* although you can create your own as we will see in later chapters. Packages are folders containing all the script that is needed to run particular functions. When you first install R it comes with a set of default packages (base R) and whenever you open R or RStudio, some of these are loaded to ensure you have basic functionality. However, as you develop your statistical and R-coding skills, you will need to load specific packages to do particular jobs.

As an example, we can use the function `describe()` from the package *psych* to report basic summary statistics for the variable `depth` in the `cyan` dataframe. The package psych is not part of base R, which means that to use the function `describe()` we must first install the package *psych*:

```
install.packages("psych")
```

R will go online, download the package from the package repository and install it to the R library on your computer. Note that to install packages the package name is wrapped in quotes ("package")

When a package is installed it is necessary to then *load* it from the library so that R can access its functions, which is done using the `library()` command:

```
library(psych)
```

Note that loading a package does not require the quotes around the name that install.packages() needed. You only need to *install* a package once. However, after you close R you will have to *load* any packages that you need again. It is good practice to load all the packages you will need for your data processing, visualisation, and analysis at the beginning of your script. You will see this in the script we provide for analysis in this book.

Now that the package psych is installed and loaded, we can use the function `describe()` to summarise the variable `depth` in the `cyan` dataframe. The variable `depth` is an estimate of the fraction of each nest box that was filled by a blue tit nest. Note that the `$` symbol is used to select the column `depth` in the `cyan` dataframe.

```
describe(cyan$depth, skew = FALSE)
```

	vars	n	mean	sd	min	max	range	se
X1	1	438	0.33	0.1	0.17	0.75	0.58	0

Which gives us the number of summarised variables (`vars`), the sample size (`n`), mean of the variable (`mean`), standard deviation (`sd`), minimum value (`min`), maximum value (`max`), range of the variable (`range`), and standard error of the mean (`se`).

Chapter 2

Introduction to Bayesian inference

Placeholder

2.1 The differences between Bayesian and frequentist approaches

2.1.1 Frequentist approach

2.1.2 Bayesian approach

2.1.3 Bayes' theorum

2.1.4 A frequentist or Bayesian framework?

2.2 Fitting Bayesian GLMs

2.2.1 Steps in fitting a Bayesian GLM

2.3 Priors

2.3.1 Non-informative priors

2.3.2 Weakly-informative priors

2.3.3 Informative priors

2.3.4 Conjugate priors

2.4 The posterior distribution

2.5 Bayesian computational methods

2.5.1 Markov chain Monte Carlo sampling (MCMC)

2.5.2 Numerical approximation

2.6 The advantages of Bayesian inference

2.7 Criticism of Bayesian inference

Chapter 3

Data exploration

Placeholder

3.1 Six-step data exploration protocol

3.1.1 Outliers

3.1.2 Normality and homogeneity of the dependent variable

3.1.3 Lots of zeros in the response variable

3.1.4 Multicollinearity among covariates

3.1.5 Relationships among dependent and independent variables

3.1.6 Independence of response variable

3.2 Results of data exploration

3.3 Conclusions

Chapter 4

Bayesian GLM

Placeholder

4.1 European bitterling territoriality

4.2 Steps in fitting a Bayesian GLM

4.2.1 State the question

4.2.2 Data exploration

4.2.2.1 Outliers

4.2.2.2 Normality and homogeneity of the dependent variable

4.2.2.3 Balance of categorical variables

4.2.2.4 Multicollinearity among covariates

4.2.2.5 Zeros in the response variable

4.2.2.6 Relationships among dependent and independent variables

4.2.2.7 Independence of response variable

4.2.3 Selection of a statistical model

4.2.4 Specification of priors

4.2.4.1 Pilot study

4.2.4.2 Frequentist linear model

4.2.4.3 Priors on the fixed effects

4.2.4.4 Priors on the hyperparameter

4.2.5 Fit the model

4.2.6 Obtain the posterior distribution

4.2.6.1 Model with default priors

4.2.6.1.1 Fixed effects

4.2.6.1.2 Hyperparameter

4.2.6.2 Model with informative priors

4.2.6.2.1 Fixed effects

4.2.6.2.2 Hyperparameter

4.2.6.3 Comparison with frequentist Gaussian GLM

4.2.7 Conduct model checks

4.2.7.1 Model selection using the Deviance Information Criterion (DIC)

4.2.7.2 Posterior predictive checks

4.2.7.3 Cross-validation model checking

4.2.7.4 Bayesian residuals analysis

4.2.7.5 Prior sensitivity analysis

4.2.7.6 Conclusions from model checks

4.2.8 Interpret and present model output

4.2.9 Visualise the results

4.3 Conclusions

Chapter 5

Bayesian Poisson GLM

Placeholder

5.1 Stickleback lateral plate number

5.2 Steps in fitting a Bayesian GLM

5.2.1 State the question

5.2.2 Data exploration

5.2.2.1 Outliers

5.2.2.2 Distribution of the dependent variable

5.2.2.3 Balance of categorical variables

5.2.2.4 Multicollinearity among covariates

5.2.2.5 Zeros in the response variable

5.2.2.6 Relationships among dependent and independent variables

5.2.2.7 Independence of response variable

5.2.3 Selection of a statistical model

5.2.4 Specification of priors

5.2.4.1 Existing data

5.2.4.2 Priors on the fixed effects

5.2.5 Fit the models

5.2.6 Obtain the posterior distribution

5.2.6.1 Model with default priors

5.2.6.2 Model with informative priors

5.2.6.3 Fixed effects

5.2.6.4 Comparison with frequentist Gaussian GLM

5.2.7 Conduct model checks

5.2.7.2.2 Simulate regression parameters from the posterior distribution

5.2.7.2.3 Calculate predicted values

5.2.7.2.4 Simulate count data using `rpois`

5.2.7.2.5 Calculate summary statistic

5.2.7.2.6 Repeat simulation

5.2.7.2.7 Compare dispersion of simulated and observed data

5.2.7.3 Posterior predictive checks

5.2.7.4 Cross-validation model checking

5.2.7.5 Bayesian residuals analysis

5.2.7.6 Prior sensitivity analysis

5.2.7.7 Conclusions from model checks

5.2.8 Interpret and present model output

5.2.9 Visualise the results

5.3 Conclusions

Chapter 6

Bayesian negative binomial GLM

Placeholder

6.1 Coral abundance

6.2 Steps in fitting a Bayesian GLM

6.2.1 State the question

6.2.2 Data exploration

6.2.2.1 Outliers

6.2.2.2 Distribution of the dependent variable

6.2.2.3 Balance of categorical variables

6.2.2.4 Multicollinearity among covariates

6.2.2.5 Zeros in the response variable

6.2.2.6 Relationships among dependent and independent variables

6.2.2.7 Independence of response variable

6.2.3 Selection of a statistical model

6.2.4 Specification of priors

6.2.4.1 Previous study

6.2.4.2 Priors on the fixed effects

6.2.5 Fit the models

6.2.6 Obtain the posterior distribution

6.2.6.1 Model with default priors

6.2.6.2 Model with informative priors

6.2.6.3 Comparison of models with uninformative and informative priors

6.2.6.4 Comparison with frequentist Poisson GLM

Chapter 7

Bayesian Bernoulli GLM

Placeholder

7.1 Common cuckoo parasitism of great reed warbler nests

7.2 Steps in fitting a Bayesian GLM

7.2.1 State the question

7.2.2 Data exploration

7.2.2.1 Outliers

7.2.2.2 Distribution of the dependent variable

7.2.2.3 Balance of categorical variables

7.2.2.4 Multicollinearity among covariates

7.2.2.5 Zeros in the response variable

7.2.2.6 Relationships among dependent and independent variables

7.2.2.7 Independence of response variable

7.2.3 Selection of a statistical model

7.2.4 Specification of priors

7.2.4.1 Pilot study

7.2.4.2 Model pilot data

7.2.4.3 Priors on the fixed effects

7.2.5 Fit the models

7.2.6 Obtain the posterior distribution

7.2.6.1 Model with default priors

7.2.6.2 Model with informative priors

7.2.6.3 Comparison of models with uninformative and infor-

Chapter 8

Bayesian gamma GLM

Placeholder

8.1 Common seal dive duration

8.2 Steps in fitting a Bayesian GLM

8.2.1 State the question

8.2.2 Data exploration

8.2.2.1 Outliers

8.2.2.2 Distribution of the dependent variable

8.2.2.3 Balance of categorical variables

8.2.2.4 Multicollinearity among covariates

8.2.2.5 Zeros in the response variable

8.2.2.6 Relationships among dependent and independent variables

8.2.2.7 Independence of response variable

8.2.3 Selection of a statistical model

8.2.4 Specification of priors

8.2.4.1 Priors on the fixed effects

8.2.4.2 Priors on the hyperparameter

8.2.5 Fit the model

8.2.6 Obtain the posterior distribution

8.2.6.1 Model with default priors

8.2.6.2 Model with informative priors

8.2.6.3 Comparison of models with uninformative and informative priors

8.2.6.4 Comparison with frequentist gamma GLM

Chapter 9

Implementing and assessing Bayesian GLMs

Placeholder

9.1 Prior information

9.1.1 The results of previous research

9.1.2 Logical considerations

9.1.3 Expert knowledge

9.1.4 Pilot data

9.2 Presenting the results of a Bayesian GLM

9.3 Reviewing Bayesian GLMs

9.4 Misuse of Bayesian inference

9.5 Conclusions

Chapter 10

Coda

How can we use new data to change what we currently believe? As ecologists we often make decisions in the face of uncertainty and incomplete information. Bayesian inference offers a framework for incrementally accruing scientific knowledge by explicitly building on the conclusions of previous knowledge.

However, despite the attraction in using Bayesian inference to tackle ecological questions, there are many pitfalls to its implementation. Sovereign against many of these problems is transparency; clearly reporting how priors were obtained, why they are specified as they are, careful description of their impacts, and presentation of sensitivity analyses. Ultimately, Bayesian methods do not offer a panacea, but they are a valuable tool for the ecologist that encourages full use of the available data - whatever form those data take.

We hope this book is useful in extending your understanding of Bayesian data analysis with R. We are always interested to receive feedback; positive or negative, and also welcome questions about your own analyses; feel free to email us.

ONeill, L. G., Parker, T. H., & Griffith, S. C. (2018). Nest size is predicted by female identity and the local environment in the blue tit (cyanistes caeruleus), but is not related to the nest size of the genetic or foster mother. *Royal Society Open Science*, *5*(4), 172036. https://doi.org/10.1098/rsos.172036