Models in Ecology

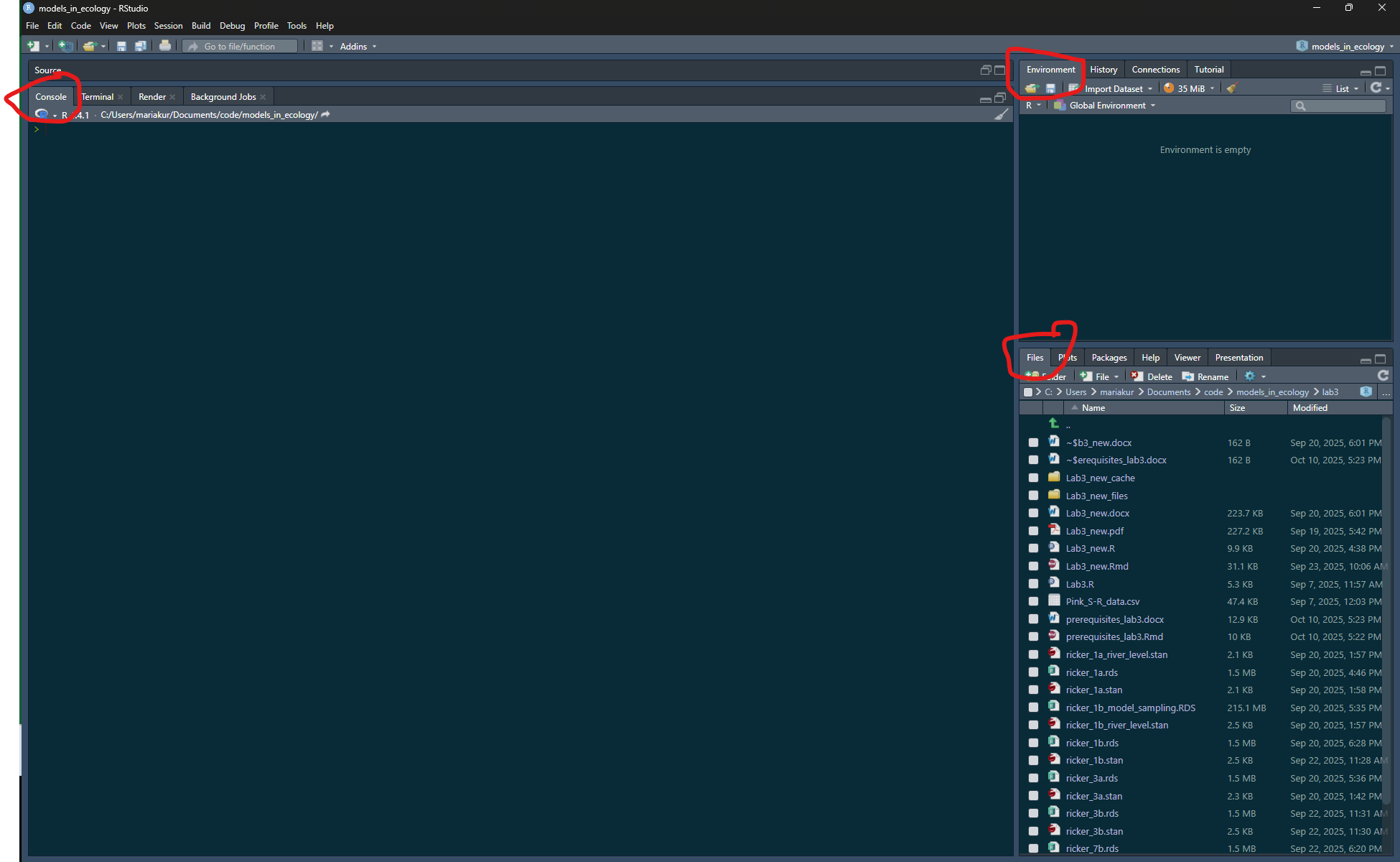
Prerequisites for Lab 3

Maria Kuruvilla

October 2025

### Introduction to effective coding in RStudio

When you open RStudio for the first time, (most of) you will see three windows in following layout:



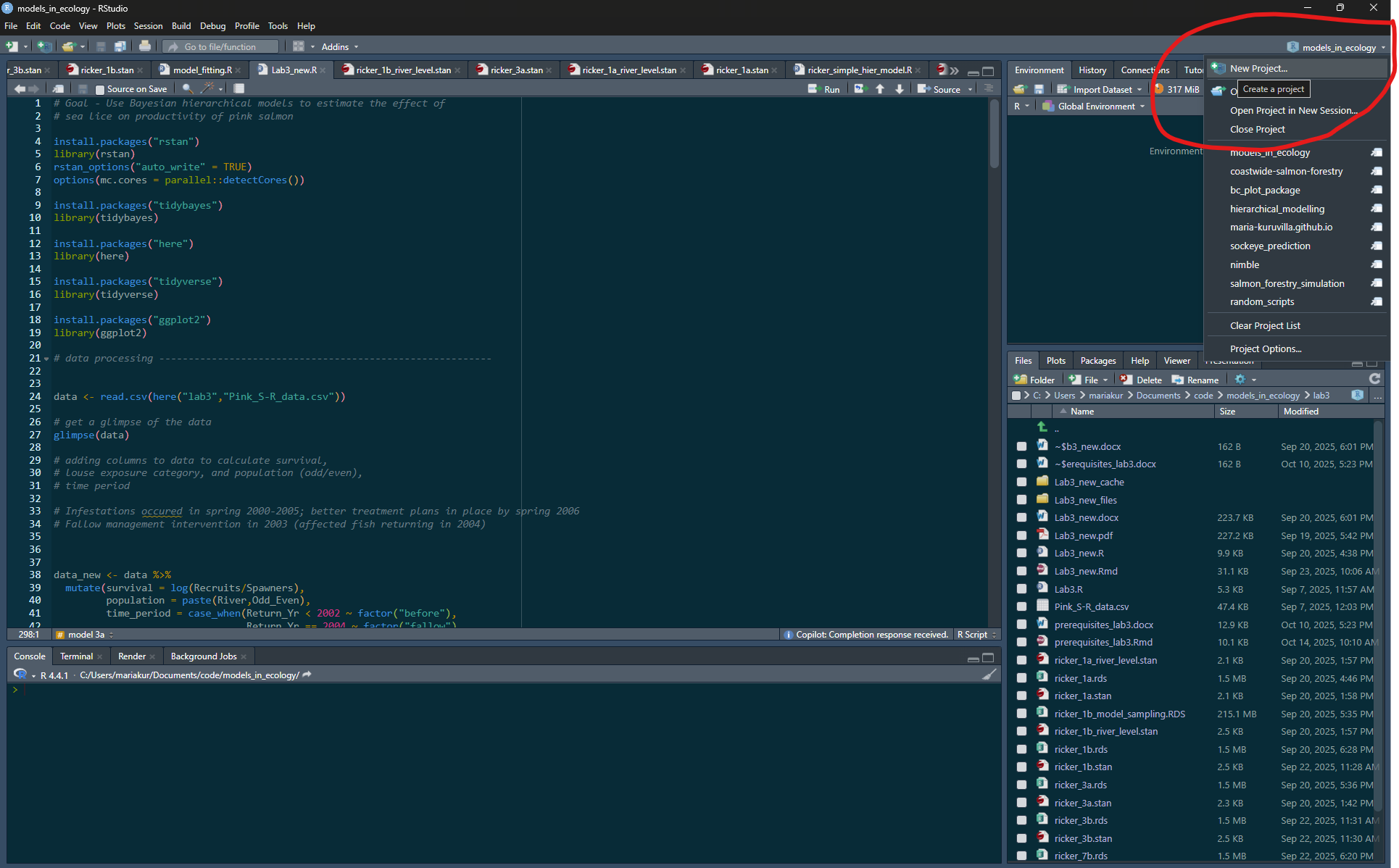
The large pane on the left is the Console Window. The window on the top right is the Environment / History / Connections pane and the bottom right window is the Files / Plots / Packages / Help / Viewer window. You can customise the location of each pane by clicking on the ‘Tools’ menu then selecting Global Options –> Pane Layout. You can resize the panes by clicking and dragging the middle of the window borders in the direction you want.

#### Using R Projects

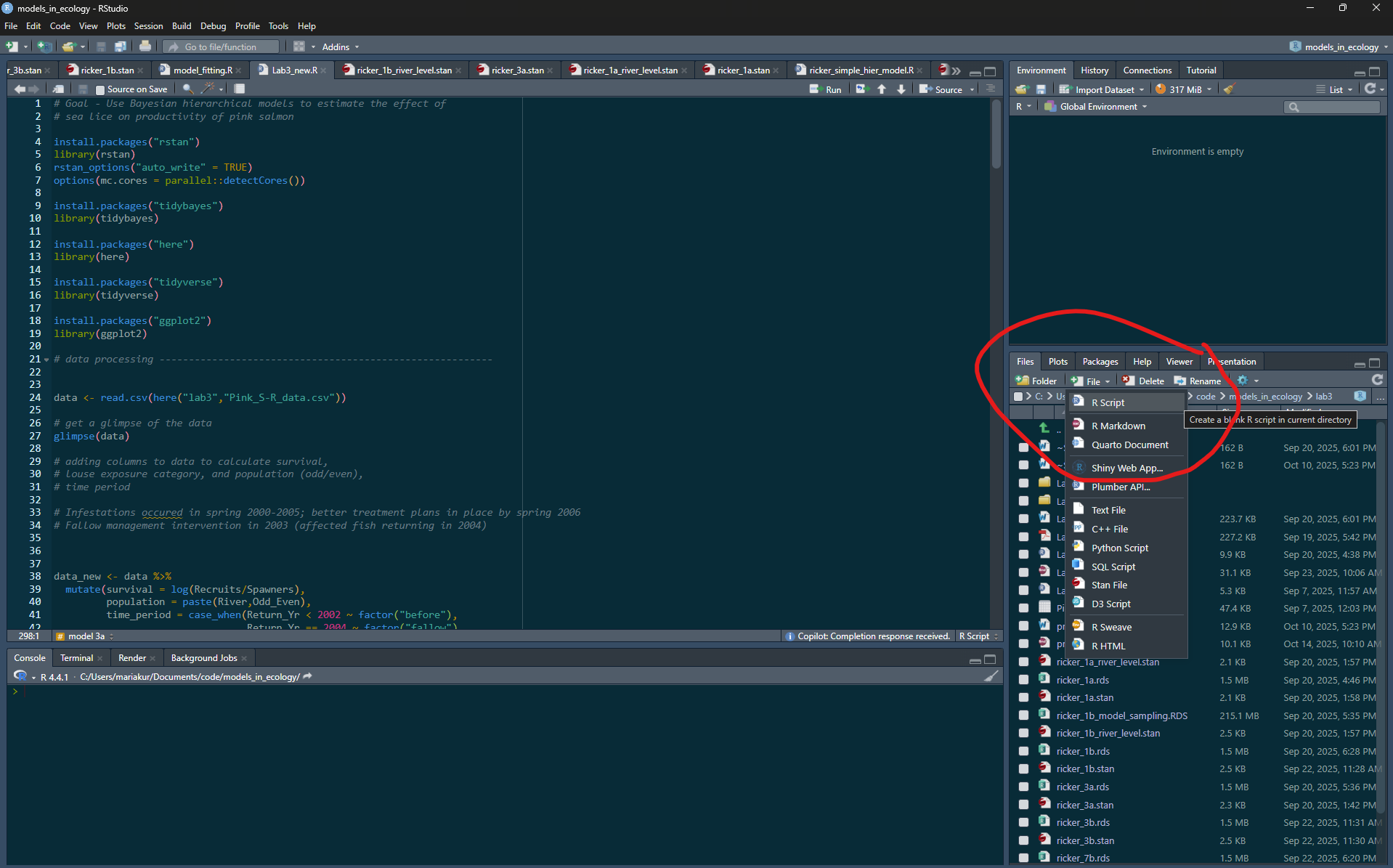
It is recommended that your code for this lab and class is within one folder which is also an Rproject for project management.

Before making a new project, you should copy all the files for this class into a folder/directory and call it models\_in\_ecology. You can make a new project, by clicking on the ‘New Project’ option on the top right, and when prompted click on existing directory and redirect Rstudio to the folder/directory. This will create a .Rproj file within the folder. Whenever you open RStudio, you can open this file by clicking on ‘Open Project in New Session’ and then clicking on this file. This will help you open all the files relevant to this project and set the working directory to this folder location. This folder can have folders within it. For example, you can make a folder called lab3 and put all files related to lab3 within it.

You can read more about what the advantages of using Rprojects are here - [Why Use Projects?](https://thedavidchen.github.io/post/rstudio-why-use-projects/)



You can type code into the console directly, but it becomes cumbersome when you have to write a lot of reproducible code. It is better to make a new file (R script) which contains your lines of code. Before making a new file, make sure that the path shown in your ‘files’ panel in your RStudio is pointing at the folder called models\_in\_ecology or wherever you want to make the new file. Then click on ‘+ file’ and then choose R Script to make a new .R file, which you can name “lab3\_setup.R”.



#### Installing packages

For lab 3, we will use some new packages, namely ‘here’, ‘tidyverse’, and ‘rstan’.

First, in the console copy and paste the following line of code:

install.packages("here")

You can read more about the package [here](https://here.r-lib.org/), which enables easy file referencing.

Repeat to install “tidyverse”:

install.packages("tidyverse")

Finally, we need to install [Rstan](https://mc-stan.org/rstan/) to fit Bayesian models. This is optional for undergraduate students.

Prior to installing rstan, you need to configure your R installation to be able to compile C++ code. This might look different depending on your operating system, so please use the [following instructions](https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started) for your system.

You can then install the rstan package and run an example model by copying and running the following lines of code:

install.packages("rstan", repos = c('https://stan-dev.r-universe.dev', getOption("repos")))  
example(stan\_model, package = "rstan", run.dontrun = TRUE)

There are additional options in the rstan installation because we are installing a more recent version of the Stan language than the latest official RStan release.

The example model might take a few minutes to finish running and will print out lines in the console while fitting. These lines should give you information on the number of chains and how many samples have been collected

### Fitting mixed-effects models in Stan

For Lab 3, graduate students will use the rstan package to fit a Bayesian mixed-effects (or hierarchical) model in R. Undergraduate students will have the option to fit a Bayesian mixed-effects (or hierarchical) model in R, but are not required to.

[Stan](https://mc-stan.org/) is a probabilistic programming language that allows us to run Bayesian statistical models - from simple linear regression to more complicated multi-level models and time-series analysis.

A Stan program is organized into a sequence of named blocks:

functions {  
 // ... function declarations and definitions ...  
}  
data {  
 // ... declarations ...  
}  
transformed data {  
 // ... declarations ... statements ...  
}  
parameters {  
 // ... declarations ...  
}  
transformed parameters {  
 // ... declarations ... statements ...  
}  
model {  
 // ... declarations ... statements ...  
}  
generated quantities {  
 // ... declarations ... statements ...  
}

All of the blocks are options. Note that comments start with “//” and all lines end with a semicolon “;”.

Here is an example of a simple linear regression for one salmon population which can be expressed as where is survival in year , is the growth rate, is the density dependence, and is the spawners at year t. Copy the following code into a new stan file (click on new file and then select stan file) within your Rproject and save it as “simple.stan”.

You can make a new stan file in the same way as you made a new R script. Click on ‘+ file’ in the bottom right files window and then select ‘Stan File’. You can name the first file ‘simple.stan’ and copy and paste the following code in it. You can check if the code is syntactically correct by clicking on the ‘check’ button on the top right of the window. If you see this warning “Warning message: In readLines(file, warn = TRUE) :incomplete final line found on ‘simple.stan’”, then just add an empty line to the end of your code by clicking enter. In order to run this code, we will have to write some lines of code in an R Script.

data {  
 int<lower=0> N; //number of observations or years in this case  
 vector[N] S; //predictor of size N  
 vector[N] y; //response variable of size N  
}  
parameters {  
 real r; //intercept to be estimated  
 real b; //slope to be estimated  
 real<lower=0> sigma; //variance to be estimated  
}  
model {  
  
 r ~ normal(1.2, 2); //prior for intercept, based on literature  
 b ~ normal(0, 10); //prior for slope  
 sigma ~ normal(0, 1); //prior for variance  
 y ~ normal(r + b \* S, sigma); // linear model  
}

#### Running Stan code

Download the spawner-recruit “Pink S-R data” as a .csv file. Read those data into R. I have downloaded into a folder called “lab3”. The “here” function makes a relative file path to Pink\_S-R\_data.csv which is within the lab3 folder.

library(here)  
library(tidyverse)  
  
data <- read.csv(here("lab3","Pink\_S-R\_data.csv"))  
  
# get a glimpse of the data  
glimpse(data)

## Rows: 1,762  
## Columns: 6  
## $ Return\_Yr <int> 1972, 1978, 1980, 1982, 1984, 1986, 1988, 1990, 1992, 1972, …  
## $ Odd\_Even <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, …  
## $ River <chr> "Pine", "Pine", "Pine", "Pine", "Pine", "Pine", "Pine", "Pin…  
## $ Area <int> 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, …  
## $ Spawners <int> 9000, 4500, 1000, 600, 2000, 700, 550, 25, 130, 12000, 5000,…  
## $ Recruits <int> 5891, 2143, 1702, 4014, 1200, 2661, 61, 260, 306, 29457, 853…

The Return\_Yr is the year that the Recruits came back to spawn, meaning that the Spawners column refers to the spawners that started in year Return\_Yr-2. Area refers to the Department of Fisheries and Oceans statistical management area, which is the spatial scale at which catch data are reported (see <http://www.pac.dfo-mpo.gc.ca/fm-gp/maps-cartes/areas-secteurs/index-eng.html>). Note that because odd and even year returns do not mix and are treated as separate populations, a population is given by a combination of River and Odd\_Even. Create a column for population (by combining river and Odd\_Even). Then calculate survival as log(Recruits/Spawners). Create a column for exposure category (by combining exposure and time\_period). Finally, create a column for time series length, start row and end row for each population. We will need this to differentiate the data from different populations in stan.

Now use the unique and the length functions in R to check how many return years, management areas and populations there are in the data frame.

data\_new <- data %>%   
 mutate(survival = log(Recruits/Spawners),  
 population = paste(River,Odd\_Even),  
 time\_period = case\_when(Return\_Yr < 2002 ~ factor("before"),  
 Return\_Yr == 2004 ~ factor("fallow"),  
 Return\_Yr >= 2002 & Return\_Yr <= 2006 ~ factor("during"),  
 Return\_Yr > 2006 ~ factor("after")),  
 exposure = as.factor(ifelse(Area==12, "exposed","unexposed"))) %>%   
 mutate(exposure\_category = paste(exposure, time\_period),  
 row\_number = row\_number()) %>%   
 # add a column with the start row number and another with the end row number for  
 # the time series of each population  
 group\_by(population) %>%  
 mutate(time\_series\_length = n(),  
 start\_row = first(row\_number),  
 end\_row = last(row\_number)) %>%   
 ungroup()  
   
   
  
glimpse(data\_new)

## Rows: 1,762  
## Columns: 15  
## $ Return\_Yr <int> 1972, 1978, 1980, 1982, 1984, 1986, 1988, 1990, 199…  
## $ Odd\_Even <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, …  
## $ River <chr> "Pine", "Pine", "Pine", "Pine", "Pine", "Pine", "Pi…  
## $ Area <int> 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, …  
## $ Spawners <int> 9000, 4500, 1000, 600, 2000, 700, 550, 25, 130, 120…  
## $ Recruits <int> 5891, 2143, 1702, 4014, 1200, 2661, 61, 260, 306, 2…  
## $ survival <dbl> -0.423798815, -0.741870680, 0.531804030, 1.90061387…  
## $ population <chr> "Pine 2", "Pine 2", "Pine 2", "Pine 2", "Pine 2", "…  
## $ time\_period <fct> before, before, before, before, before, before, bef…  
## $ exposure <fct> unexposed, unexposed, unexposed, unexposed, unexpos…  
## $ exposure\_category <chr> "unexposed before", "unexposed before", "unexposed …  
## $ row\_number <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, …  
## $ time\_series\_length <int> 9, 9, 9, 9, 9, 9, 9, 9, 9, 16, 16, 16, 16, 16, 16, …  
## $ start\_row <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 10, 10, 10, 10, 10, 10, …  
## $ end\_row <int> 9, 9, 9, 9, 9, 9, 9, 9, 9, 25, 25, 25, 25, 25, 25, …

Let us make a subset of the data with one population using the filter function, to fit our simple.stan model. Note the number of rows and columns.

data\_subset <- data\_new %>%   
 filter(population == "Embly 2")  
   
glimpse(data\_subset)

## Rows: 19  
## Columns: 15  
## $ Return\_Yr <int> 1972, 1974, 1976, 1978, 1980, 1982, 1984, 1986, 198…  
## $ Odd\_Even <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, …  
## $ River <chr> "Embly", "Embly", "Embly", "Embly", "Embly", "Embly…  
## $ Area <int> 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,…  
## $ Spawners <int> 100000, 70000, 7000, 13000, 25000, 12000, 7000, 120…  
## $ Recruits <int> 200284, 158931, 22854, 28088, 45165, 15824, 11008, …  
## $ survival <dbl> 0.69456617, 0.81997490, 1.18321601, 0.77039308, 0.5…  
## $ population <chr> "Embly 2", "Embly 2", "Embly 2", "Embly 2", "Embly …  
## $ time\_period <fct> before, before, before, before, before, before, bef…  
## $ exposure <fct> exposed, exposed, exposed, exposed, exposed, expose…  
## $ exposure\_category <chr> "exposed before", "exposed before", "exposed before…  
## $ row\_number <int> 1744, 1745, 1746, 1747, 1748, 1749, 1750, 1751, 175…  
## $ time\_series\_length <int> 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19,…  
## $ start\_row <int> 1744, 1744, 1744, 1744, 1744, 1744, 1744, 1744, 174…  
## $ end\_row <int> 1762, 1762, 1762, 1762, 1762, 1762, 1762, 1762, 176…

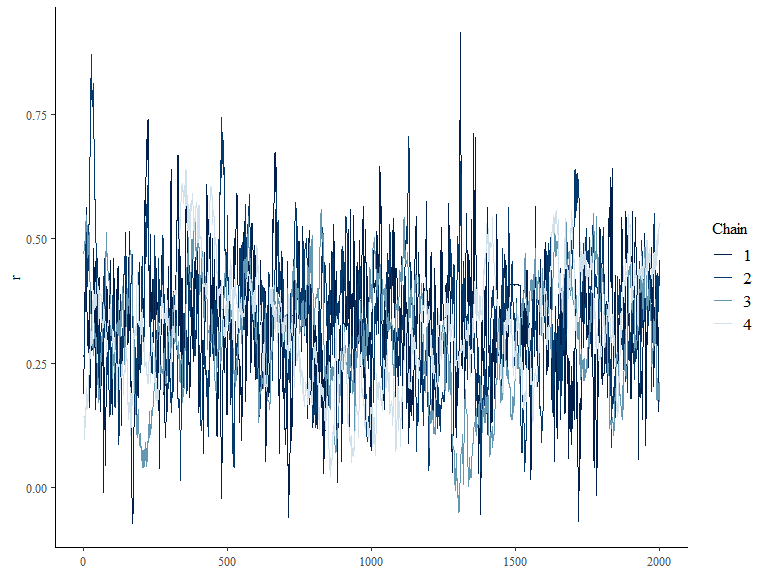
Now we need to call the stan model and give it a name within the R Script. Here I am calling it ricker\_simple\_model and using the stan\_model() function to call the stan model. It might take a minute to run. Then we need to make a data list for the first block of the stan model. Make sure the variables match the variables in the data block of the stan code. We then draw samples from the posterior distribution by using the sampling() function. This function has inputs which are the stan model and the data we provide. We can also specify the number of chains we want to use, how many samples or iterations (iter) we want, and how many warmup iterations to use. In this context, chains refer to the sequence of samples from the distribution. We usually want more than 1 chain which start at different points to make sure that all chains converge to the same posterior distribution after multiple iterations. We also discard the first 1000 samples (warmup) because we do not want to make inference from these samples, which might be influenced by the starting values of the parameters in the chains.

This might take ~2 minutes to run.

# call model  
  
library(rstan)  
ricker\_simple\_model <- stan\_model(file = here("lab3","simple.stan"))  
  
  
# create data list for model   
  
data\_list\_simple <- list(  
 N = nrow(data\_subset),  
 S = data\_subset$Spawners,  
 y = data\_subset$survival  
)  
  
# options for rstan  
rstan\_options("auto\_write" = TRUE)  
  
# runs the chains in parallel  
options(mc.cores = parallel::detectCores())  
  
# sample from model  
ricker\_simple\_model\_sampling <- rstan::sampling(ricker\_simple\_model,  
 data = data\_list\_simple,  
 chains = 4,  
 iter = 3000,  
 warmup = 1000)

We can assess convergence by looking at the chains/sequence of samples. The chains should resemble a hairy catterpillar. These chains don’t look so good because we don’t have a lot of data. This leads us to using multi-stock model with data from several populations by accounting for random effects in Lab 3.

# check trace plots to look for caterpillar like sampling plots  
  
bayesplot::mcmc\_trace(ricker\_simple\_model\_sampling, pars = "r")



# plot posterior probability distribution  
bayesplot::mcmc\_areas(ricker\_simple\_model\_sampling, pars = c("r"))

