

## SNP Analysis using dartRverse



# Artificial Intelligence and dartRverse

Version 1



Copies of the latest version of this tutorial are available from:

The Institute for Applied Ecology  
University of Canberra ACT 2601  
Australia

Email: [arthur.georges@biomatix.com.au](mailto:arthur.georges@biomatix.com.au)

Copyright © 2026 Arthur Georges

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, including electronic, mechanical, photographic, or magnetic, without the prior written permission of the senior author.

Such permission would normally be granted for educational purposes, to be used with or without modification, provided that due acknowledgement is given.

Citation: Whiteley, S., O'Meally, D., Patel, H. Mijangos, L., Gruber, B. and Georges, A. (2026). Artificial Intelligence and dartRverse. Technical Series No. 2 published by Biomatix Pty Ltd, Sutton 2620, Australia.

dartR.verse is a collaboration between the University of Canberra, CSIRO and Diversity Arrays Technology, and is supported with funding from the ACT Priority Investment Program, CSIRO and the University of Canberra.



# Contents

<b>Contents</b> .....	<b>5</b>
<b>Introduction</b> .....	<b>7</b>
<b>What you will Learn</b> .....	<b>8</b>
<b>Who is this Tutorial for?</b> .....	<b>8</b>
<b>Large Language Models in Context</b> .....	<b>8</b>
<b>Development of AI and Coding Agents</b> .....	<b>10</b>
<b>Building an AI Coding Collaborative Environment</b> .....	<b>11</b>
Preliminaries .....	11
Step 1: Create a project folder .....	12
Step 2: Install the Claude Code extension in VS Code .....	12
Step 3: Sign in to Claude Code in VS Code.....	12
Step 4: Install the ClaudeR package in RStudio .....	13
Step 5: Start the ClaudeR server in RStudio.....	13
Step 6: Tell Claude Code to use RStudio.....	13
What this looks like in practice .....	14
Practical tips.....	14
<b>Worked Examples – Session 1</b> .....	<b>16</b>
<b>Example 1: Basic query in Claude</b> .....	<b>16</b>
<b>Example 2: Code Correction (Claude)</b> .....	<b>16</b>
<b>Example 3: Code Generation (Claude)</b> .....	<b>18</b>
<b>Example 4: Simple Query, Claude Code, Single Agent</b> .....	<b>19</b>
<b>Example 5: Complex Query, Claude Code, Single Agent</b> .....	<b>20</b>
<b>Some More Theory</b> .....	<b>22</b>
<b>Rules Files, Skills and Hooks</b> .....	<b>22</b>
Rules Files .....	22
Skills.....	23
Sharing Skills via GitHub.....	24
Hooks .....	25
<b>Agents and Sub-Agents</b> .....	<b>25</b>
Agents.....	25
Sub-agents.....	25
Individual Meeting .....	26
Team Meeting .....	26
<b>Worked Examples – Session 2</b> .....	<b>27</b>
<b>Example 6. Establishing a Skill</b> .....	<b>27</b>
<b>Example 7. Virtual Lab</b> .....	<b>29</b>
Virtual Lab – Myuchelys popgen .....	30
Problem definition .....	30
Deliverables .....	30
Agent and Sub-agent Definitions .....	31
Execution Phases .....	34
Agent Interaction Rules.....	35
<b>Practical Nuances</b> .....	<b>40</b>
Validation .....	41
Access Constraints .....	42
Confidentiality .....	43
Reality Check.....	43
The Future .....	44
<b>Further Reading</b> .....	<b>44</b>
<b>Where have we come?</b> .....	<b>44</b>



## Introduction

We are entering a brave new world of big data and new and exciting techniques for analysing those data. High-throughput sequencing – once prohibitively expensive and technically daunting – now routinely delivers gigabytes of data at a fraction of the former cost, unveiling the immense genetic variation that exists across all levels of biological organisation—from ecosystems and species to populations, individuals, tissues, and even single cells.

This revolution is not just in the generation of new data — it is in what can be done with those data. There has been an equally remarkable acceleration in the development of analytical tools capable of extracting biological meaning from these immense datasets. New software packages, programming environments and computational methods appear daily, offering ever more refined ways to visualise, filter, analyse and interpret genomic information.

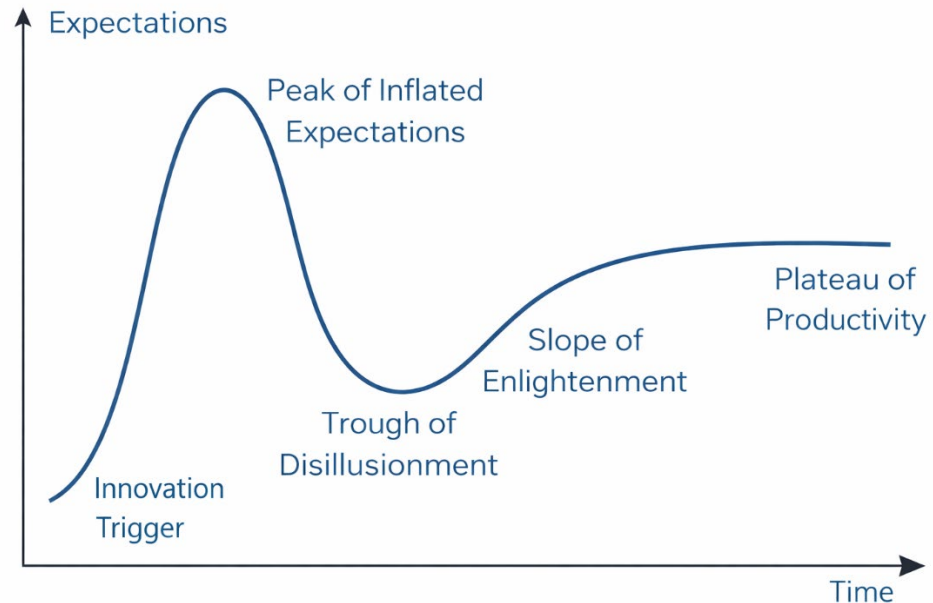
In this tutorial, we express our excitement for the changing research environment before us with the advent and rapid development of artificial intelligence (AI). AI approaches to improving researcher productivity are accelerating such that, even in the past two or so years, we have seen a transformation in the way we work. AI tools driven by large language models (e.g. ChatGPT and Claude) are powerful literature discovery and synthesis tools that can draw from a vast array of documents in ways that are not humanly possible. They are rapidly displacing the more conventional search engines and engagement with specialised blogs, that used to be a first port of call for troubleshooting, is dropping precipitously.

Applying these large language models (LLMs) to programming languages has dramatically reduced the burden on programmers who were previously slaves to syntax, documentation and cryptic bug resolution. And today, as we will see, AI models can function as collaborators in research, involved in the design and execution of code, and the presentation and interpretation of results. These recent developments are truly astonishing. These systems are now capable of planning, calling external tools and multimodal reasoning (Dip et al., 2025).

Together, these three advances – unprecedented data production, rapid tool innovation and development of generative AI collaboration tools – have transformed and will continue to transform how we approach the study of biodiversity, the questions we can now answer, and the pace at which we can advance our understanding.

We are well aware of the Gartner Hype Cycle (Figure 1) and hope that the picture we are painting in this chapter leaves you not on the *Peak of Inflated Expectations*, certainly not in the *Trough of Disillusionment*, but rather sitting somewhere on the *Slope of Enlightenment* poised to adopt these new techniques in the interest of greater productivity.

Figure 1. The Gartner Hype Cycle shows how new technologies often move from early hype and inflated expectations, through disappointment, to a more mature and productive phase of adoption.



## What you will Learn

- Background to the application of Artificial Intelligence (AI) tools to assist in research on population genetics.
- Technical requirements to install Claude Code on your computer and interface it with your R analyses.
- How to construct Skill Sets (often referred to simply as Skills) to guide your interaction with Claude Code and establish common practice across all your queries.
- Appreciation of different levels of engagement, from simple queries to checking your R scripts, to writing R scripts, to engaging an agent in a simple task, to combining agents and subagents in virtual lab to conduct routine analyses, to generating nuanced workflows.
- A sound attitude to engagement with AI that leaves you in control of analyses, able to validate the contribution from AI, and to recognise that at the end of the day, you are responsible for the analyses and interpretations.

## Who is this Tutorial for?

This tutorial is for users of the R package `dartRverse` designed for genetic analysis of single nucleotide polymorphisms (SNPs) and associated SilicoDArT data. It is not meant as a generic introduction to AI in genetics and genomics.

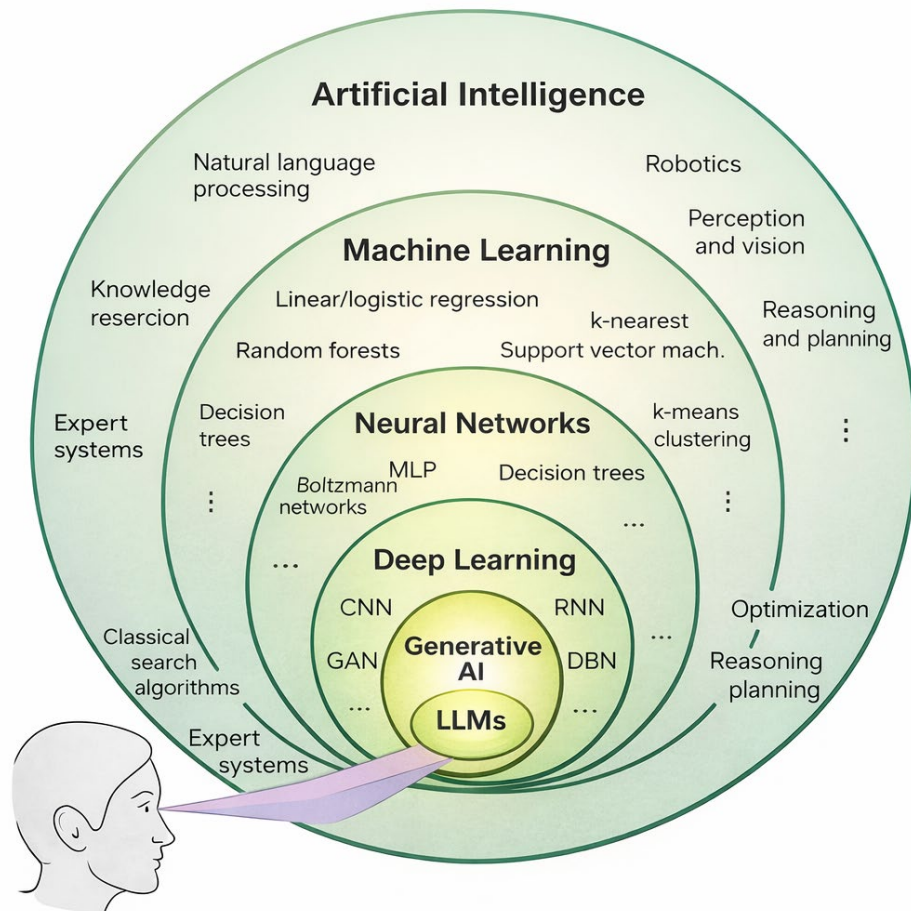
## Large Language Models in Context

Generative AI refers to a subset of artificial intelligence that generates new content, including text, images, audio, and code, by learning patterns from large collections of existing material. We have all used tools such as ChatGPT, Claude, Gemini or CoPilot and been impressed with their capabilities.

Today's excitement around generative AI is understandable, but it is also somewhat myopic. Large language models are only one small part of the much broader field

of artificial intelligence. The current focus on LLMs is a product of visibility, accessibility, and commercial excitement, not a faithful map of the field. LLMs are unusually visible because language is our native medium. Generative AI models can summarise papers, write code, explain methods, draft reports, extract information and support exploratory analysis. That makes them enormously useful. However, AI also encompasses symbolic reasoning, search, planning, robotics, perception, classical machine learning, neural networks, deep learning, and more recently, generative models. Each has its place. The sensible view is neither to dismiss LLMs nor to mistake them for AI itself.

*Figure 2. Generative AI may dominate the spotlight, but it sits within a much larger landscape. Artificial intelligence is broader, older and far more diverse than today's focus on LLMs might suggest.*



LLMs shine in genomics workflows that require literature synthesis, code generation and exploratory analysis to reveal pattern in large and complex datasets. However, there are real limitations to the current LLM-centred view. Generative AI models can be impressive but they can also mislead, obscure their reasoning, be difficult to verify and encourage overconfidence because their outputs are fluent and often presented without qualification, and can stroke the users' perceptions of the value of the contribution with flattery and affirmation (Cheng et al., 2026).

The central message here is that generative AI models are embedded within a much broader field of artificial intelligence, and older methods are often better suited to the task at hand. If the task is to classify samples, estimate risk, cluster populations,

detect outliers, optimise routes, or support a scientific inference pipeline, then a simpler model may outperform a generative one on the criteria that matter – accuracy, transparency, cost, speed, and ease of validation. In many scientific workflows, including those in R, the best solution is often a conventional statistical model or a standard machine-learning method, rather than an LLM. Choose the right tool for the job.

Whatever tool you choose, be sure to verify the output. If you are going to use a generative AI model to search and summarise the literature, to write an R script or a bash script, or as a collaborator in planning and undertaking an analysis, a second central message is that you need to be abreast of the analyses that are undertaken by the AI agent(s), and *validate, validate and validate* again. The buck stops with you; it is your scientific reputation on the line.

## Development of AI and Coding Agents

Artificial intelligence (AI) had modest beginnings, and not so long ago. The first applications that were widely noticed had the computer identifying a repetitive text-manipulation task after the user showed it a few examples and then repeat it across a dataset or spreadsheet. This was often called autocomplete. In programming, systems began predicting the next line of code a programmer was likely to write next based on patterns learned from large collections of code. AI was speeding up typing and reducing routine effort.

Subsequently, large language models (LLMs) were developed and trained on both natural language (like English) and source code. That made it possible for a programmer to describe a task in plain English and receive back a block of code. This was a genuine change in how people interacted with software tools. Instead of merely completing what the programmer had started, the AI could now respond to instructions such as “write a function to clean these data” or “generate tests for this code” or “check this code for syntax errors and logical errors”.

AI systems could not only generate code. It also began to explain code, rewrite it, simplify it, convert it from one language to another, and help find bugs. In other words, the role of AI expanded from producing snippets to assisting with the whole thought process of programming. For many users, especially those without formal training in computer science, this was the first time programming started to feel conversational. Rather than memorising exact syntax, they could describe what they wanted and then inspect, adjust, and test the code the system produced.

The most recent stage in AI development to assist programming goes a step further. Not only can modern AI coding tools write code, but they can also act on it. They can read files in a project, change several files at once, run tests, modify the approach, run the analyses, inspect the results, and then re-evaluate and revise their own work. This is the difference between an assistant who merely drafts text and one who can actually take part in a work cycle. In practical terms, that means AI is moving from “here is some code you might try” to “I heard what you need,

designed and ran the analysis, did some checks, and here is the result.” That is why people now speak of coding “agents” rather than just code completion tools. AI is now a creative collaborator.

AI assistance for R programming followed the same broad path but came a little later and with a different emphasis. R is widely used for statistics, data analysis and scientific workflows, so relies more on language models that can be integrated directly into analysis environments. Modern coding environments such as Visual Studio Code and RStudio increasingly integrate AI directly into the programming workflow, allow enabling tools such as Claude Code and Posit Assistant to assist with planning analyses, writing code, and executing tasks. This can help the analyst write a plotting function, clean a dataset, build a Shiny app, explain a statistical result, or turn an analysis idea into reproducible code and then execute it. As AI tools improved, they became more useful for these tasks too. The development of AI in R was about making data analysis and scientific computing more accessible and more interactive.

AI is today, at the time of writing, no longer just an external website or chatbot that a programmer consults on the side. It is establishing itself as part of the programming environment, able to see which packages are loaded, which objects exist, and what problem the user is trying to solve. We are moving to an environment where AI can not only suggest code, but also run it, test it, revise it within a broader workflow and produce a detailed analysis report. This is transformative.

## Building an AI Coding Collaborative Environment

An AI coding collaboration between a generative AI tool, RStudio and R can be achieved in many ways. We describe one approach here, using Claude Code running inside Visual Studio Code (VS Code) connected to a live RStudio session on a Windows desktop or laptop. In this setup, Claude Code acts as the AI collaborator — planning, writing, and revising code — while RStudio serves as the execution environment where that code actually runs. Claude Code communicates with RStudio through a small R package called ClaudeR, which opens a local server that Claude Code can send commands to.

This means you keep the RStudio environment you are familiar with — your console, plots pane, environment browser, and file viewer — while gaining an AI agent that can read your data, write analysis scripts, execute them in your R session, inspect the results, and iterate. The two tools run side by side on your machine.

### Preliminaries

Before starting, make sure the following are installed:

- R and RStudio. Install R from <https://cran.r-project.org> and RStudio Desktop from <https://posit.co/download/rstudio-desktop>.

- Visual Studio Code. Download and install from <https://code.visualstudio.com>. It is free.
- A Claude subscription that includes Claude Code. Sign up or upgrade at <https://claude.ai>.
- On Windows, Git for Windows must also be installed before Claude Code will work properly. Download Git for Windows from <https://gitforwindows.org>. MacOS users can download Git from from <https://git-scm.com>.

### Step 1: Create a project folder

Start by creating a folder for your work. Keep your data, scripts, and results together in one place.

#### Windows example

```
C:\Users\yourname\projects\my_analysis\
├─ data\
│   └─ samples.csv
│   └─ genotypes.csv
├─ scripts\
└─ output\
```

#### macOS example

```
/Users/yourname/projects/my_analysis/
├─ data/
│   └─ samples.csv
│   └─ genotypes.csv
├─ scripts/
└─ output/
```

Open this folder in VS Code with File → Open Folder. This folder becomes the working context Claude Code will see — it can read and create files within this directory.

### Step 2: Install the Claude Code extension in VS Code

Open VS Code and go to the Extensions panel (click the square icon on the left sidebar, or press Ctrl+Shift+X on Windows or Cmd+Shift+X on macOS). Search for 'Claude Code' and install the extension published by Anthropic (Claude Code for VS Code). If it does not appear after installation, restart VS Code.

Once installed, Claude Code will appear as a panel within VS Code, typically in the sidebar or as a tab you can open from the Activity Bar.

### Step 3: Sign in to Claude Code in VS Code

Open the Claude Code panel in VS Code.

A browser window will open — follow the prompts to sign in to your Claude account. Sign in using your Claude subscription account and follow the prompts in the browser if required.

Once signed in, Claude Code is ready to use inside VS Code. You do not need to install anything separately for this workflow.

## Step 4: Install the ClaudeR package in RStudio

Open RStudio and install the ClaudeR package, which allows Claude Code to send R commands to your RStudio session. In the RStudio console, run:

```
# Install ClaudeR from GitHub (you may need the devtools
  package)
install.packages("devtools")
devtools::install_github("IMNMV/ClaudeR")
```

This only needs to be done once.

While still in RStudio, run

```
library(ClaudeR)
install_cli(tools = "claude")
```

This will result in a command being displayed in the Console. Copy that command. Open the VS Code terminal via the menu Terminal → New Terminal. Paste the command and run it in the VS Code terminal. Close the terminal window or type exit and press Enter.

## Step 5: Start the ClaudeR server in RStudio

Each time you begin a working session, load ClaudeR and launch the server:

```
library(ClaudeR)
claudeAddin()
```

A small panel should appear in the lower right of the RStudio window. If it does not appear there, check the Viewer tab in the bottom-right pane.

Click Start Server. When the server starts, two URLs will be displayed. The first appears in the ClaudeR panel — this is the HTTP server address.

```
http://127.0.0.1:8787
```

Copy this URL. Keep RStudio open and the server running. This is the connection point that Claude Code will use to execute R code in your session.

The second URL appears in the RStudio Console — this is the address RStudio is listening on. Copy this also; you will need to provide them both in Step 6.:

## Step 6: Tell Claude Code to use RStudio

In your first message to Claude Code, tell it that RStudio is available and provide the server URL. For example:

```
I am working on a population genetics analysis. RStudio is running a
  HTTP server on http://127.0.0.1:8787 and listening on
  http://127.0.0.1:5237. Please use RStudio to execute R code. The data
  files are in the data/ directory.
```

Note: Your port numbers (e.g. 8787) may differ. Replace these URLs with the actual addresses shown in your ClaudeR panel and RStudio console.

From this point, Claude Code can write R code and execute it directly in your RStudio session. You will see the commands appear in the RStudio console, and any plots, objects, or output will be visible in RStudio as if you had typed the commands yourself.

### What this looks like in practice

With both tools running, a typical interaction might look like this:

You (in Claude Code):

"Read the file `data/genotypes.csv`, report its dimensions and summarise any missing data, then produce a PCA plot coloured by the population column."

Claude Code will:

- Write R code to read the CSV file.
- Send it to RStudio via the ClaudeR server.
- Inspect the output to check the dimensions and missingness.
- Write `ggplot2` code for the PCA plot.
- Execute it and check that the plot rendered.
- Report back to you with a summary and any issues it encountered.

You can watch each step happen in RStudio in real time. Claude Code will ask for your approval before running shell commands (e.g., installing packages, reading files, or executing scripts). For each command, you can approve it once, approve similar commands for the session, or decline. This keeps you in control of what runs on your machine.

If something goes wrong — a package is not installed, a column name is misspelled, the PCA fails because of missing values — Claude Code will see the error in the R console output and attempt to fix it.

### Practical tips

Set your working directory in RStudio too. Make sure RStudio's working directory matches the project folder you opened in VS Code. In the RStudio console, run

```
setwd("C:/Users/yourname/projects/my_analysis") [Windows]
```

or

```
setwd("/Users/yourname/projects/my_analysis") [macOS]
```

or open the project as an RStudio Project (`.Rproj`). This avoids file path mismatches when Claude Code sends code that references relative paths.

Add a `CLAUDE.md` file. Place a plain text file called `CLAUDE.md` in the root of your project folder. Claude Code reads this automatically at the start of each session.

Use it to describe the project, your preferred packages, coding conventions, and any constraints. For example:

```
# Project context
Population genetic analysis of SNP data from DArTseq.
Uses the dartR and adegenet packages.
```

```
# Conventions
- Use tidyverse style
- All file paths relative to project root
- Data files in data/ should not be modified
- Write output to output/
```

This saves you from repeating context in every prompt.

Claude Code starts an interactive session in the terminal. By default, it uses the Sonnet model. If you're working on a complex or multi-step analysis and want the most capable model, type `/model` within the Claude Code session and select Opus. The more capable model uses more tokens and therefore costs more per interaction.

Use Git for version control. Commit your work before asking Claude Code to make substantial changes. If Claude Code rewrites a script in a way you do not like, you can revert cleanly. Claude Code can also make commits on your behalf if you ask it.

## Worked Examples – Session 1

### Example 1: Basic query in Claude



*Can you briefly, in under 200 words, explain what SilicoArT data are?*

SilicoDArT (Diversity Arrays Technology) markers are presence/absence markers generated through DArT sequencing (DArTseq). Unlike SNP markers that capture single nucleotide variants, SilicoDArT markers represent the presence (1) or absence (0) of specific DNA fragments in a genome, producing a binary data matrix.

The technology works by reducing genome complexity through restriction enzyme digestion, then sequencing the resulting fragments. When a fragment is detected in a sample, it scores as present; when it's not (due to a restriction site polymorphism, insertion/deletion, or divergence), it scores as absent. This makes them dominant markers — you can't distinguish heterozygotes from homozygotes for presence.

SilicoDArT markers are particularly useful in non-model organisms and polyploids because they don't require a reference genome and can tolerate high ploidy levels. They typically yield tens of thousands of markers per run, complementing the SNP markers that DArTseq also produces simultaneously. They're widely used in population genetics, phylogeography, and breeding programs — especially in plants — for assessing genetic diversity, population structure, and linkage mapping.

### Example 2: Code Correction (Claude)



*I have an R script which drops selected populations from a SNP dataset in dartR genlight form.*

```
gl.drop.pop <- function(x, pop.list) {
  # Drop the specified populations
  ind.to.keep <- which(!(x$pop %in% pop.list))
  x <- x[ind.to.keep,]
  return(x)
}
```

*Can you suggest improvements.*

Here are some improvements worth considering:

**Input validation** — the function silently returns an unchanged object if pop.list contains names that don't match anything. Adding checks catches user errors early.

**Informative messaging** — when working with large SNP datasets, it's helpful to know what actually happened (how many individuals/populations were dropped).

**Locus filtering** — after dropping populations, some loci may become monomorphic or entirely missing. Worth flagging or optionally removing these.

Here's a revised version:

```
gl.drop.pop <- function(x, pop.list, verbose = TRUE) {
  if (!inherits(x, "genlight")) {
    stop("x must be a genlight object.")
  }
  if (!is.character(pop.list)) {
    stop("pop.list must be a character vector.")
  }
  # Check for population names that don't exist in the dataset
  unknown <- setdiff(pop.list, levels(pop(x)))
  if (length(unknown) > 0) {
    warning("These populations not found in the dataset: ",
            paste(unknown, collapse = ", "))
  }
  matched <- intersect(pop.list, levels(pop(x)))
  if (length(matched) == 0) {
    warning("No matching populations to drop. Returning x unchanged.")
    return(x)
  }
  ind.to.keep <- which(!(pop(x) %in% pop.list))
  x <- x[ind.to.keep, ]
  # Drop unused factor levels so downstream summaries are clean
  pop(x) <- droplevels(pop(x))
  if (verbose) {
    message(sprintf("Dropped %d individuals from %d population(s): %s",
                    length(which(pop(x) %in% pop.list)) +
                    (nInd(x) - length(ind.to.keep)),
                    length(matched),
                    paste(matched, collapse = ", ")))
    message(sprintf("Retained %d individuals across %d populations.",
                    nInd(x), nPop(x)))
  }
  return(x)
}
```

A few specific notes on the changes:

**pop(x) instead of x\$pop** — the pop() accessor from adegenet is the proper way to get/set population assignments on a genlight object. Direct \$ access can behave unexpectedly, especially with subclasses.

**droplevels()** — without this, dropped populations persist as empty factor levels, which causes phantom entries in table(pop(x)) and similar summaries.

**setdiff()** check — this is probably the most practically useful addition. Typos in population names are a common source of silent bugs where the function appears to work but drops nothing.

There are a number of things to note here. First of all, Claude appears to know a great deal about dartRverse and dartR genlight objects, as some of the advice draws heavily on this knowledge. Second, it is very thorough, much more thorough than the average coder who often tires of the tedium of working through every contingency, particularly boundary states (circumstances that rarely occur). Finally, AI has a reputation for providing, very confidently, incorrect information. Fortunately, in the context of coding, validation of the advice occurs naturally when

the code is executed. Provided you exert a watchful eye, all but subtle problems of mis-advice emerge immediately because the code breaks or does not do what was intended.

### Example 3: Code Generation (Claude)



*I have a SNP dataset with multiple populations defined. It has missing values even after standard filters on callrate. I would like to impute those missing values drawing from the genotypes nearest neighbours. To do this, I would like first to remove those loci that are missing for all individuals in any one population. Can you write an R script to do this for me. Make it a function. Use `gl.report.bases()` from `dartRverse` as an example of the format.*

Let me look at the structure of `gl.report.bases()` to understand the `dartRverse` function conventions you want me to follow.

```
gl.filter.allpopna <- function(x, verbose = NULL) {
# SET VERBOSITY
if (is.null(verbose)) {
  if (!is.null(x@other$verbose)) {
    verbose <- x@other$verbose
  } else {
    verbose <- 2
  }
}
# FLAG SCRIPT START
funname <- "gl.filter.allpopna"
utils.flag.start(func = funname, build = "custom", verbose = verbose)
# CHECK DATATYPE
datatype <- utils.check.datatype(x, verbose = verbose)
# MAIN BODY
pop.list <- seppop(x)
n.pops <- length(pop.list)
n.loc0 <- nLoc(x)
# For each locus, check whether it is entirely NA within any one population
# Build a logical matrix: populations x loci, TRUE = all missing
all.na.mat <- sapply(pop.list, function(gl) {
  colMeans(is.na(as.matrix(gl)), na.rm = FALSE) == 1
})
# Transpose so rows = loci, cols = populations
if (!is.matrix(all.na.mat)) {
  all.na.mat <- matrix(all.na.mat, ncol = n.pops)
}
# A locus fails if it is entirely NA in ANY population
loci.to.drop <- apply(all.na.mat, 1, any)
n.drop <- sum(loci.to.drop)
if (verbose >= 2) {
  message(sprintf(" Loci examined: %d", n.loc0))
  message(sprintf(" Loci with all NA in at least one population: %d", n.drop))
  message(sprintf(" Loci retained: %d", n.loc0 - n.drop))
}
if (verbose >= 3 && n.drop > 0) {
  bad.names <- locNames(x)[loci.to.drop]
  bad.pops <- apply(all.na.mat[loci.to.drop, , drop = FALSE], 1, function(row) {
    paste(names(pop.list)[row], collapse = ", ")
  })
  message(" Loci dropped and the population(s) where all values are NA:")
  for (i in seq_along(bad.names)) {
```

```

    message(sprintf(" %s : %s", bad.names[i], bad.pops[i]))
  }
}
# FILTER
if (n.drop > 0) {
  x <- x[, !loci.to.drop]
}
if (n.drop == 0 && verbose >= 2) {
  message(" No loci required removal. Returning genlight object unchanged.")
}
# ADD TO HISTORY
nh <- length(x@other$history)
x@other$history[[nh + 1]] <- match.call()
# FLAG SCRIPT END
utils.flag.end(func = funname, verbose = verbose)
return(x)
}

```

A few notes on design decisions:

seppop() then per-population missingness — this is the most direct way to check the condition. seppop() splits the genlight into a list of per-population genlight objects, then we ask whether every individual in that population is NA at each locus. The alternative of looping over populations with index subsetting would be equivalent but less readable.

colMeans(is.na(...)) == 1 — this is the key test. If the mean of is.na across all individuals in a population equals 1, every single value is missing. This is more efficient than all(is.na(...)) applied column-wise because colMeans is vectorised in C.

Why this matters for your imputation — if a locus is entirely missing within a population, a nearest-neighbour imputer will either fail or be forced to borrow from a different population, which contaminates the allele frequency structure you're presumably trying to preserve. Removing these loci first is the right preprocessing step.

The verbose levels follow dartR convention: 0 = silent, 1 = minimal, 2 = standard summary, 3 = detailed diagnostics. Level 3 here prints exactly which loci were dropped and which population caused it, which is useful for reporting in a methods section.

You may want to adjust the utils.flag.start and utils.check.datatype calls depending on which version of dartR/dartRverse you're running, as the internal utility function signatures have shifted slightly across versions.

Not only does Claude Code do the job, but it also provides additional useful notes on design. And in its notes, it has identified efficiencies that many a human coder would not have seen. The promise of productivity gains cannot be overstated.

## Example 4: Simple Query, Claude Code, Single Agent



*I have a DArT SNP dataset `Myuchelys.Rdata`. Can you examine these data set and give me a summary of the contents in tabular form. Please use `dartRverse` scripts where possible to load the dataset and interrogate it.*

### Myuchelys.Rdata — Dataset Summary

Parameter	Value
Class	dartR (genlight subclass)
Individuals	141
Loci (raw SNPs)	31,289
Populations	4
Ploidy	2 (diploid)
Overall missing data	11.61%
dartR.base version	1.0.0

### Sample Sizes per Population

Population	n
Border-Nth	13
Border-Sth	41
Gwydir	65
Namoi	22

### Missing Data per Population

Population	Mean missing (%)
Border-Nth	15.5
Border-Sth	8.2
Gwydir	10.8
Namoi	18.2

## Example 5: Complex Query, Claude Code, Single Agent



*Can you map the SNPs in `Pogona.Rdata` against the NCBI `Pogona` annotated genome assembly `PviZW2.1` and produce a graph of the SNP count by chromosome, a plot of SNP density along the chromosomes and a plot of SNP count against chromosome length. No other plots. In each case produce a publication quality graph. Identify the Z pseudoautosomal region, the Z non-recombining region and the W chromosome specifically.*

Figure 3. Counts of SNPs by chromosome in the dragon lizard *Pogona vitticeps*.

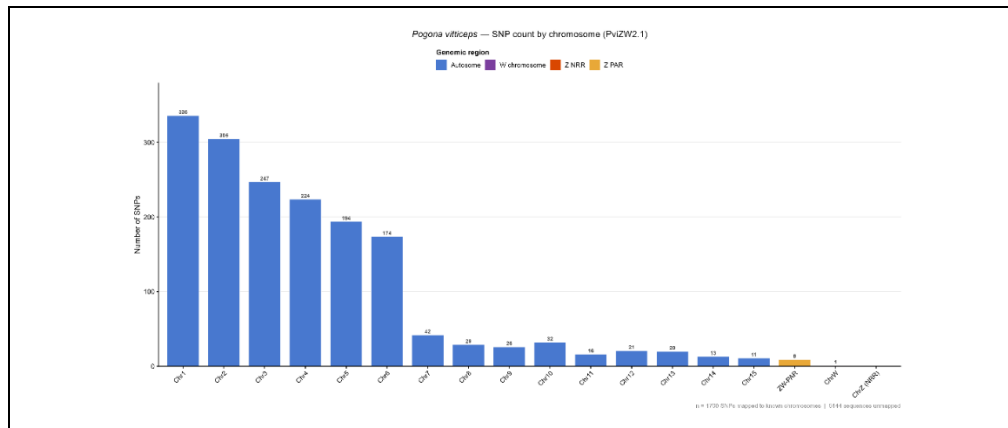


Figure 4. SNP density along the chromosomes of the dragon lizard *Pogona vitticeps*.

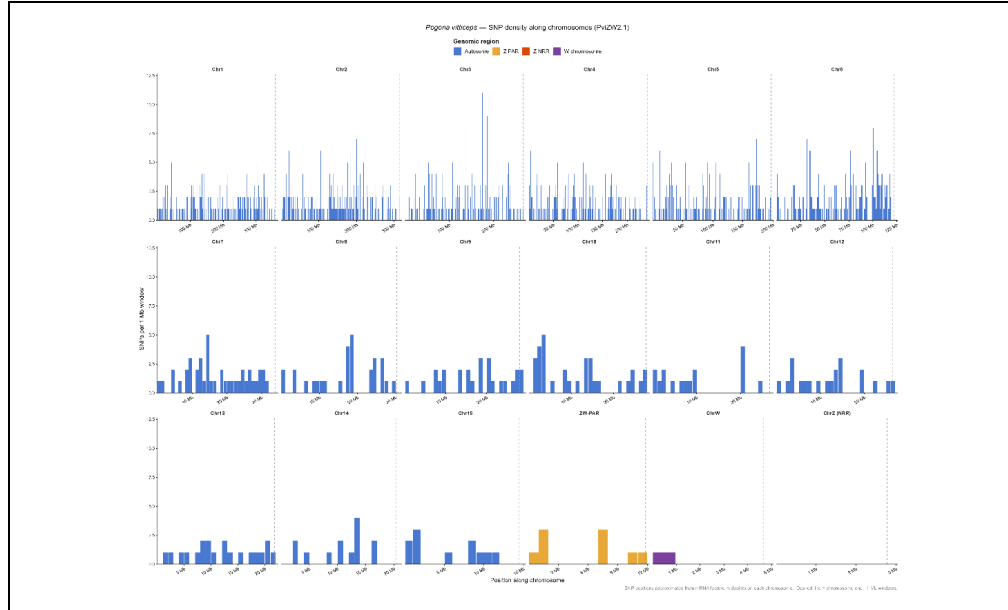
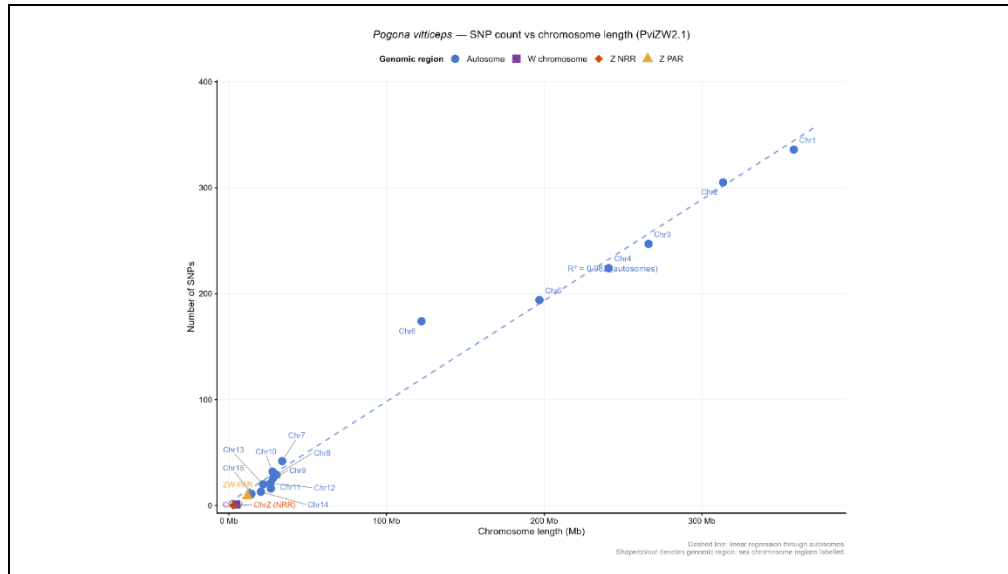


Figure 5. Plot of SNP counts by chromosome length against chromosome length (bp) in the dragon lizard *Pogona vitticeps*.



After perusing, slewing, schlepping, mulling, blooping, deliberating, forming, creating for an indeterminate period, the prompt yielded the graphs shown in Figures 3-5. The prompt resulted in Claude planning the job, setting up an agent to do the job, installing the necessary software, and running it to produce the three requested graphs.

Note that to do this, you will likely need to have a BYO (buy your own) computer because the restrictions placed on corporate computers by universities and other organisations will likely prevent this from working.

*Pogona vitticeps* has six macrochromosomes and 10 microchromosomes which is clearly evident in Figure 3. Obviously, the number of SNPs on each chromosome is strongly related to chromosome size, as shown in Figure 5. There is no evidence to suggest that the SNPs are not randomly distributed across the chromosomes, including the pseudoautosomal region (PAR) of the sex chromosomes. However, there are no (or at least very few) SNPs in the non-recombining region of the sex

chromosomes. This is likely because of the high repetitive sequence content of the NRR – repetitive sequence is eliminated by the DART pipelines because of ambiguity in establishing homology.

Of course, one needs to check the code that Claude used to generate the above figures to ensure that appropriate decisions have been made on parameters used to undertake the BLAST analysis. This can be done by providing Claude with a Skill set or a Rules file to specify the parameters to be used in BLAST searches. We cover the topic of Skill sets and Rules files in the sections that follow.

## Some More Theory

### Rules Files, Skills and Hooks

We now introduce the concept of Rules Files, Skills and Hooks. Rules Files are plain text files that act essentially as a standing briefing that Claude reads at the start of every session. Skills are reusable instruction sets that teach Claude a specific capability — such as reviewing scientific writing or following a particular analytical workflow. Hooks are automatic triggers that execute a defined action whenever a specified event occurs in Claude Code.

#### Rules Files

Rules Files contain instructions that Claude Code will access when undertaking tasks. Rules files are Markdown files (`.md` extension) that carry knowledge and preferences across sessions. They should be kept concise, no more than 200 lines, because large Rules Files can use up your resource allocation. Resource allocation is measured as tokens – chunks of text that AI reads and processes. Longer Rules files contain more text and consume more tokens. This leaves fewer for your actual task. A sensible limit would be 200 lines.

You should place the Rules File in your `./claude/rules` subdirectory of your working directory so that Claude Code will access it automatically. Claude will access all of the `.md` files that are placed in the `./claude/rules` directory.

Here is a simple Rules File as an example.

```
# Researcher Profile

## Role

The user is a research scientist. Responses should assume scientific literacy — use precise terminology, avoid over-explaining basic concepts, and engage at a peer level.

## Primary Language

All code should be written in R unless explicitly asked otherwise. Prefer base R or tidyverse idioms as appropriate to the task.

## Key Package: dartRverse
```

The user works extensively with the `dartRverse` suite of R packages for the analysis of DArT (Diversity Arrays Technology) sequencing data in population and conservation genetics. The core packages are:

- `dartR.base` — data import, filtering, and quality control
- `dartR.popgen` — population genetic analyses
- `dartR.spatial` — landscape and spatial genetics
- `dartR.sim` — simulation of genotype data
- `dartR.captive` — captive breeding and genetic management

When writing or reviewing R code:

- Prefer `dartRverse` functions over equivalent functions from other packages where `dartRverse` provides the capability
- Assume the primary data object is a `genlight` object (from the `adegenet` package), which `dartRverse` extends
- Be aware that `dartRverse` functions typically follow the naming convention `gl.*()` (e.g. `gl.filter.callrate()`, `gl.report.bases()`)
- Respect `dartRverse`'s built-in reporting and verbosity arguments (`verbose =`) where relevant

Claude Code provides a convenient Skill to convert a Rule File into a Skill (next section) -- `anthropic-skills:skill-creator`.

## Skills

In Claude Code, a Skill set is a folder on your computer that contains

- a reusable instruction set (`SKILL.md`) that teaches Claude how to perform a specific task consistently.
- optional scripts for Claude to access when undertaking the set task.
- optional documentation that can be loaded by Claude Code as needed.
- optional assets (templates, fonts, icons etc) that can be used in generating outputs.

The file `SKILL.md` can include your preferred software (use `dartRverse` where possible), patterns of use, conventions or procedures so Claude can follow those specifications reliably without you re-explaining each time.

To establish a Skill, you need to write clear, concise instructions in a Claude `SKILL.md` file. For example, you might specify "follow tidyverse style conventions", anything that is relevant and common to your R workflows.

Claude `SKILL.md` files are markdown files that can be placed at different levels:

- Project-level: Place the Claude `.md` file and its directory as a subdirectory in the working directory so that it applies to all work undertaken using files in that directory:

```
D:\workspace\R_analysis\\
├── .claude\
│   └── skills\
```

```
└─ <my-skill>\
  └─ SKILL.md
```

- User-level: Personal preferences that apply across all your projects:

```
C:\Users\<username>\.claude\
└─ skills\
  └─ <my-skill>\
    └─ SKILL.md
  └─ <another-skill>\
    └─ SKILL.md
```

- Team access: You can arrange team access to your Skills via GitHub, covered later in this tutorial.

The more specific and concrete you are in construction SKILL.md files, the better Claude will follow through. You can iterate and refine these instructions as you work and, indeed, Claude is of great assistance in refining your draft skill specification.

The establishment of a skill is demonstrated in Worked Example 6.

For further information, refer to the [Claude Code Documentation](#).

## Sharing Skills via GitHub

To share Skills with a team, a simple approach is to store them in a dedicated GitHub repository. Each Skill should be placed in its own folder, with a SKILL.md file inside. For example:

```
team-claude-skills/
└─ README.md
└─ dartR-analysis/
  └─ SKILL.md/
└─ manuscript-review/
  └─ SKILL.md
└─ teaching-materials/
  └─ SKILL.md
```

You can then grant access to the repository to collaborators or a GitHub team. Team members clone the repository to their own computer by copying the relevant Skill folders into their personal Claude skills directory:

```
C:\Users\<username>\.claude\skills\
```

If the Skill is relevant to only one project, the Skill can be copied into a project-specific `.claude\skills\` folder.

In this way, GitHub acts as the sharing and version-control layer, while Claude Code reads the local SKILL.md files from the appropriate directory on each user's machine.

To share a Skill:

- Create a GitHub repository for shared Claude Skills.
- Add one folder per Skill, each containing a SKILL.md file and any optional supporting scripts, documentation, or assets.

- Grant your collaborators access to the repository.
- Ask each team member to clone the repository locally.
- Have them copy or link the required Skill folders into either their user-level or project-level Claude skills directory.

This approach makes it easy to maintain consistent instructions, update workflows centrally, and share improvements across a group.

## Hooks

While skills tell your workflow what and how to execute, hooks tell your workflow when to initiate an element of your workflow, based on a trigger.

An example is a PreToolUse hook. This is rule that runs before Claude executes a specified command on your computer. In RStudio, a PreToolUse hook can give you a safety net while still letting Claude run normal R commands.

When Claude is about to run a terminal command, the hook checks the command first and if the command looks dangerous, can stop it; if the command looks fine, it lets it execute. The hook is essentially a gatekeeper.

We will not deal with hooks here, and leave it to the reader to explore this in the [Claude documentation](#).

## Agents and Sub-Agents

---

### Agents

Claude Code "agents" operate autonomously to plan and implement coding tasks — rather than just answering questions, Claude Code can access your computer, read files using installed software (e.g. dartRverse in RStudio), install software as required, edit files, run commands, execute tests, coordinate the full analysis and iterate on solutions.

The key idea is that instead of you requesting R code to undertake a specific task then executing it yourself (Worked Example 3), copying and pasting code snippets back and forth, you can delegate a task to Claude and it works through it end-to-end. Claude will understand the problem, explore relevant files, make changes, run tests to verify, and fix issues if something breaks. It operates in a loop of reasoning, acting and observing results until the task is done.

In VS Code, you can see what the agent is doing, and it will often ask for permission to proceed to the next step. Some care needs to be exercised.

For more detailed and up-to-date information on Claude Code's capabilities and how to use it, we recommend checking the [Anthropic documentation](#).

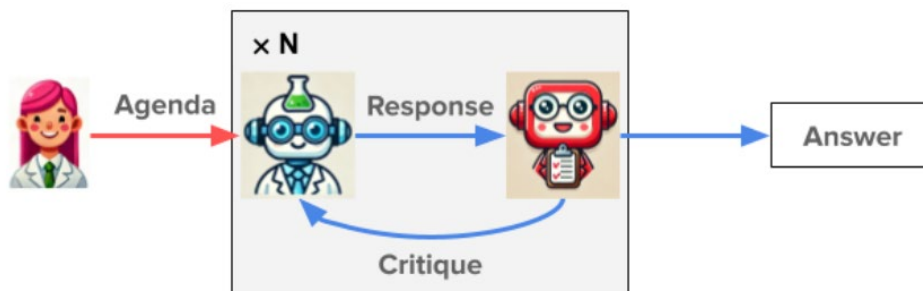
### Sub-agents

Claude Code can also spawn sub-agents — essentially smaller, scoped instances of Claude that handle specific subtasks within a larger workflow. This lets it break complex problems into manageable pieces. With complex tasks of long duration, a single agent can become confused by intermediate work that is to the side of the main thrust of the analysis. Sub-agents can be given discrete responsibilities that can perhaps be run in parallel, allowing the main agent to stay focused on the overall requirements, constraints and decisions leading up to the final output.

Subagents were used to great effect in a recent study on the use of AI agents in the design of new Covid nanobodies (*Nature* 646: 716–723, 2025). The process is described in more detail on the Zou Group GitHub site – <https://github.com/zou-group/virtual-lab>.

### Individual Meeting

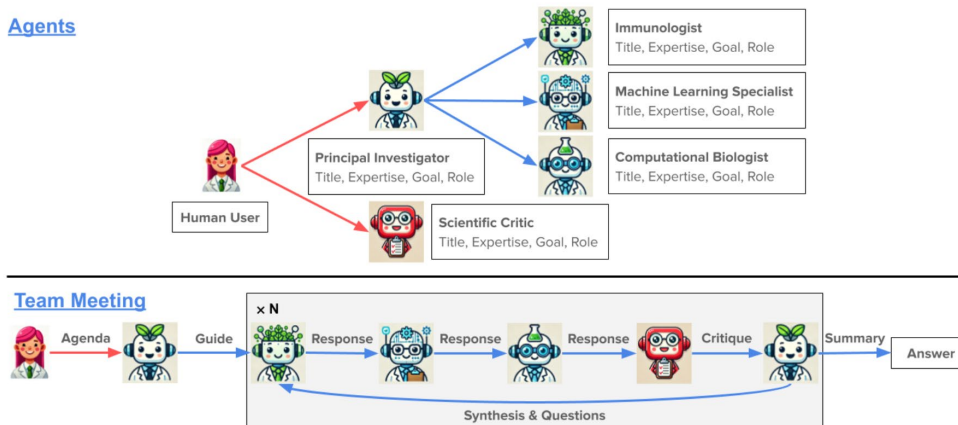
The simplest use of multiple sub-agents is described by the Zou Group as an individual meeting.



Here the human sets the agenda and puts a proposition to the agent who crafts a response and passes it to a sub-agent tasked with critiquing the agent's draft response. After a specified maximum number of iterations, the response is delivered to the human.

### Team Meeting

Team meetings involve more complex interactions among the agent (Principal Investigator) and multiple sub-agents, each with specific responsibilities and defined capabilities.



Establishing what Zou's Group refer to as a virtual lab requires considerable attention to crafting the prompt to Claude, because not only does one have to specify the problem at hand, but also to define in some detail the responsibilities and capabilities of each sub-agent and how they will interact with each other during each iteration of the analysis. The critic agent has a specific role to play, as does the Principal Investigator.

The virtual lab delivers its greatest value when it is sufficiently general in agent definitions and associated Rules and Skills to be used for many analyses of similar scope.

We will put all of this into practice in Worked Example 7 below.

## Worked Examples – Session 2

### Example 6. Establishing a Skill

To establish a skill, we start with a fairly rudimentary set of instructions and allow Claude to complete it with all the bells and whistles.



***Can you construct me a Claude skill to support analyses of population genetics of wild populations of endangered species. The data file will be a user-specified .Rdata file containing a SNP genlight object generated from data supplied by Diversity Arrays Technology. In addition to the SNP genotypes, the genlight object should have individuals classed into populations, and individuals should have latitude and longitude values of their sampling locality. The skill set should draw where possible on the functions in dartRverse, though not exclusively if other software is required. The skill set should cover***

***(a) The production of a map showing the distribution of samples labelled with their population.***

***(b) A PCA plot showing the genetic distance relationships among individuals, colour coded by their populations.***

***(c) Basic statistics such as F-statistics, heterozygosity estimates, genetic diversity indices, allelic richness estimates (including a graph produced by gl.report.allelerich), private alleles, estimates of inbreeding coefficients, relatedness estimates within populations and any other statistics that you think are relevant.***

***The skill set should place particular emphasis on analyses of relevance to management, such as analyses to support judgements on the strategy for establishing insurance breeding colonies and strategies for re-release of captive bred animals.***

***Please ensure that all graphs and figures are of publication quality.***

Claude will come back with a file containing a very detailed skill directory with an [.md](#) file. You can refer to this Skill explicitly when you run an analysis. In this case, the directory is called `popgen_dartR` and the instructions will be held in `SKILL.md`.

You can download the resultant [.md](#) file [here](#).

The output of Claude Code following your request is given below:

What the skill file covers:

The skill file covers three core modules plus a substantial management-oriented section. A few points on the design decisions:

**Structure** — the file follows the dartRverse conventions you requested (gl.\* function calls, verbose levels, genlight object handling), but brings in hierfstat for rarefied allelic richness and the related package for pairwise kinship, since these aren't fully replicated within dartR alone.

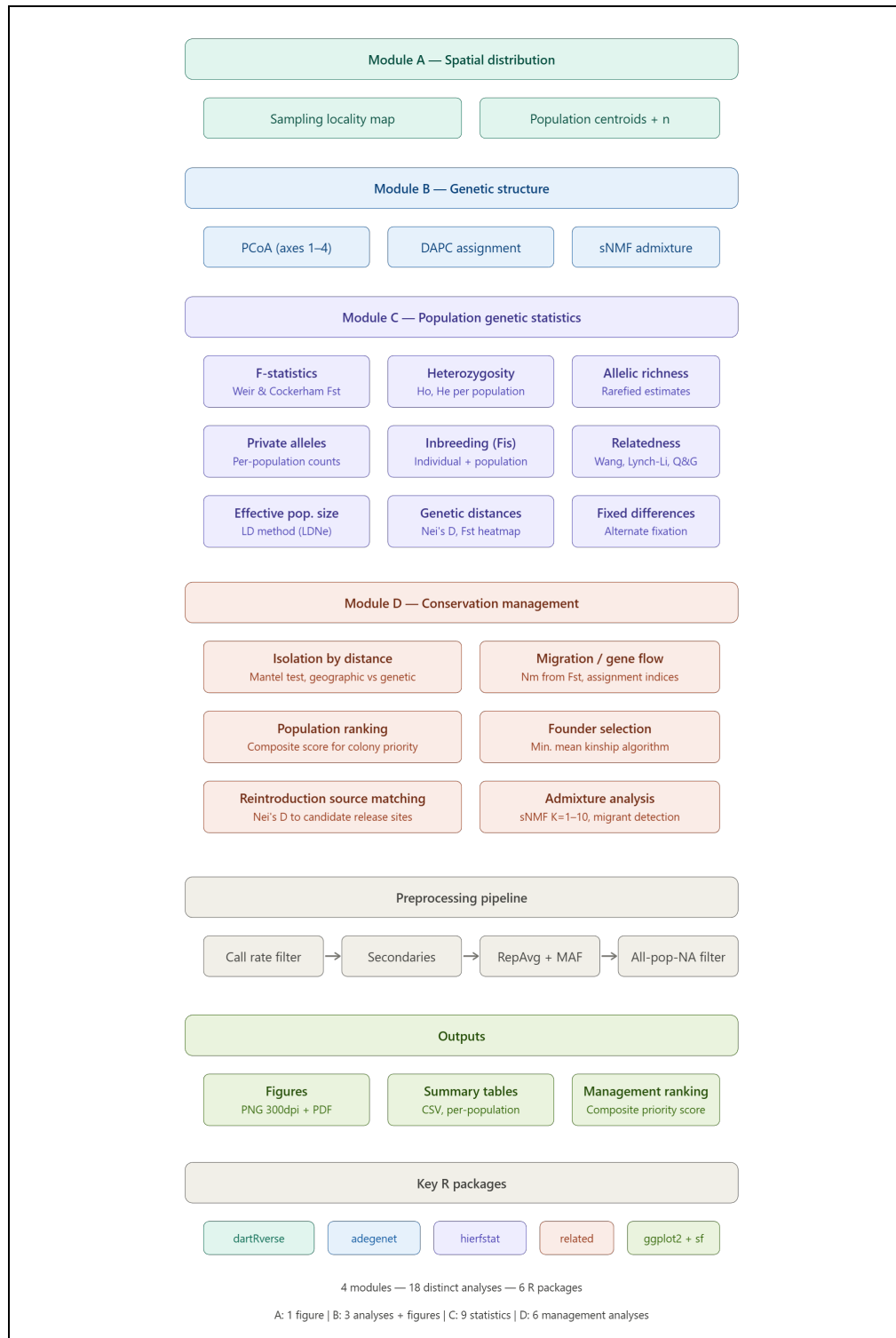
**Management emphasis** — Module D goes beyond descriptive statistics into the decision-support tools you'd actually need for colony planning. The founder selection algorithm (D4) implements a greedy mean-kinship minimisation, which is the standard approach used by zoo-based breeding programs (essentially a simplified version of what PMx software does). The population ranking table (D3) gives a composite score across the key metrics, which you can weight according to species-specific priorities.

**What's not included but worth noting** — the skill file flags but doesn't implement environmental niche matching for reintroduction site selection, since that requires spatial environmental layers beyond the genlight object. If you work with SDMs or climate data for your species, that would be a natural extension. Similarly, if you have pedigree information from any captive animals, that could be integrated with the genomic relatedness estimates for more precise kinship calculations.

The gl.filter.allpopna function we wrote earlier slots directly into the preprocessing section, before imputation.

Figure 6 provides a summary of the analytical elements of the Skill set [popgen\\_dartR](#). You can apply the skills by placing the md file in your working directory and referring Claude Code to it.

Figure 6.  
Summaary of  
the analytical  
elements of a  
Claude Skill  
set  
(popgen\_dartR  
.md).



## Example 7. Virtual Lab

A virtual lab establishes a network of sub-agents each with their own skill set. Here we will establish a virtual lab designed to undertake analyses relevant to the population genetics of wild populations of endangered species with a particular focus on the implications for managing those populations.

The dataset we will be working with concerns the endangered freshwater turtle in the headwater tributaries of the northern basin of the Murray-Darling River system,

the largest in Australia. The data are in the compressed file [Myuchelys.Rdata](#) which Claude will recognise as holding a dartR genlight object. We examined the contents of this dartR genlight object in Worked Example 4.

Once the lab is established, we can pass any dataset to it to re-run the analyses in a different context. Based on the results of the analyses, the conclusions drawn could be quite different. Virtual labs are very flexible.

What follows is a detailed definition of our Virtual Lab (construction assisted by Claude, based on an initial attempt). You can copy this into Claude Code and see how it goes. We will evaluate the results at the end to see how mature this Claude Code platform is.



## Virtual Lab – *Myuchelys popgen*

### Problem definition

We start by defining the problem, specifying the species, the location of the data and the geographic scope. Then there is the management interest:

1. Does the species show strong genetic structuring across its range?
2. If so, which subpopulations should be prioritised as sources for establishing insurance colonies, and why?
3. Do some populations appear at greater risk than others?
4. How should insurance colonies be structured to maximise genetic diversity retention while avoiding outbreeding depression?
5. How should a re-release strategy for captive-bred individuals be designed to maintain genetic integrity of wild populations?

Then we add a little more context. The dataset at hand is SNP data generated by Diversity Arrays Technology. A previous study by Darren Fielder used mitochondrial sequence variation to assess structure across the landscape -- Fielder, D., Vernes, K., Alacs, E. and Georges, A. 2012. Mitochondrial variation among Australian freshwater turtles (genus *Myuchelys*), with special reference to the endangered *M. bellii*. *Endangered Species Research* 17:63-71. You should look at this paper for context.

Any recommendations for management should balance preservation of locally adapted variation against the risks of genetic erosion in small, isolated populations.

### Deliverables

1. Format: Markdown (.md), formal scientific report style.
2. Audience: Government conservation managers (not geneticists) — explain all technical terms.
3. Style: Prose throughout. No dot points. No excessive sub-sectioning or page breaks after sections. Written like a scientific paper. 1.5 spacing.
4. Citations: Harvard referencing style, drawing on peer-reviewed literature and relevant grey literature (EPBC Conservation Advice, IUCN Red List assessments).
5. Figures and tables: Embedded inline within the Results section, with numbered captions. Publication quality.
6. Report structure (strict):

- Executive Summary (max 500 words). Include key recommendations.
- Introduction: Set context as conservation planning for structured populations of endangered species. Identify knowledge gaps in this field.
- Summarize the species biology and conservation status.
- State the objectives/purpose of this study given the context and key knowledge gaps.
- Materials and Methods. Specify the data, its contents, and the source. Include in the materials and methods how the data were obtained (DARtseq genotyping methods and bioinformatic pipeline). Draw from other published works or reports of a similar nature, e.g. Georges, A., Gruber, B., Pauly, G.B., Adams, M., White, D., Young, M.J., Kilian, A., Zhang, X., Shaffer, H.B. and Unmack, P.J. 2018. Genome-wide SNP markers breathe new life into phylogeography and species delimitation for the problematic short-necked turtles (Chelidae: Emydura) of eastern Australia. *Molecular Ecology* 27:5195-5213.
- Results (with Figures and Tables embedded).
- Discussion and Recommendations.
- References. All cited references are to be aggregated at the end of the report.

**Additional deliverable:**

1. A complete, reproducible R script (Myuchelys\_bellii\_analysis.R) documenting all analyses performed.
2. A flowchart showing the relationship between the human prompter, the agent and subagent and the final report (deliverable). Be sure to show the information flow between the agent, the subagent and the critic agent, and show the number of iterations where the workflow is repeated. Also be sure to show the requested information flow between subagents. The flowchart should be a graphic png file.

**Agent and Sub-agent Definitions**

**Geneticist**

You are a population geneticist specialising in wildlife genetics and the genetic management of endangered species. Thoroughly review the peer reviewed literature and write the Introduction section covering:

1. Genetic rescue vs. outbreeding depression — decision frameworks (cite Frankham et al. 2011)
2. Minimum viable population size and effective population size thresholds (50/500 rule, but consider also the revised thresholds from Frankham et al. 2014)
3. Criteria for prioritising source populations for insurance colonies (genetic distinctiveness, diversity, extinction risk, logistic feasibility)
4. Adaptive genetic variation and local adaptation — why population structure may reflect adaptation
5. Re-release strategy design: provenance matching, release site selection, post-release genetic monitoring

6. Genetic load considerations for populations entering captive management

Be specific. Cite landmark papers and analytical frameworks. Write in prose.

#### ecologist

You are a freshwater turtle ecologist. Contribute to the Species Biology section of the introduction, providing life history and ecological parameters that will directly inform the Planner and Analyst:

1. Longevity, age at maturity, clutch size, nesting frequency, nest-site fidelity
2. Movement ecology, home range size, inter-catchment dispersal potential
3. Habitat requirements: river geomorphology, basking sites, nesting substrate
4. Diet and trophic role
5. Sex determination mechanism (GSD vs. TSD) and implications for climate vulnerability
6. Behavioural traits relevant to translocation: site fidelity, homing behaviour

Explicitly provide: generation time (years), clutch size range, and maximum recorded dispersal distance — these parameters will be used by the Planner for Ne estimation and the Analyst for interpreting gene flow results.

#### conservationist

You are an Australian conservation biologist specialising in freshwater turtles and the EPBC Act. Write the Species Biology and Conservation Status section of the introduction covering:

1. Taxonomy and nomenclatural history (Elseya → Myuchelys; Thomson and Georges 2009)
2. Distribution across known range, noting which portions are represented in the available genetic data
3. EPBC listing status, IUCN Red List status, state listing status, key threatening processes, and recovery plan provisions
4. Known population structure from prior genetic studies (mtDNA, microsatellites)
5. Captive management history and any existing insurance colony work
6. Key threats: habitat degradation, fox predation on nests, road mortality, altered hydrology, climate change

Draw on peer-reviewed literature, EPBC Conservation Advice, and IUCN assessments. Harvard referencing.

#### planner

You are a bioinformatician specialising in conservation genomics. Design the analytical pipeline (using dartRverse where possible, as implemented through the RStudio Claude instance) and write the Materials and Methods section. The pipeline must address:

1. Quality filtering: reproducibility, call rate (locus and individual), read depth, secondaries, monomorphs, MAF threshold. Use dartR functions. Note that filtering may have already been undertaken, and if so, there is no issue with skipping a particular filter.

2. Population structure: PCoA (`gl.pcoa`), Bayesian clustering via sNMF (LEA package), neighbour-joining tree (`gl.tree.nj`). Justify K selection via cross-entropy.
3. Genetic differentiation: pairwise FST (`gl.fst.pop`), with significance testing
4. Isolation by distance: Mantel test (`gl.ibd` in `dartR.spatial`) within populations identified by drainage basin.
5. Genetic diversity:  $H_o$ ,  $H_e$ ,  $uH_e$ , FIS per population (`gl.report.heterozygosity`)
6. Private alleles: `gl.report.pa`
7. Relatedness: pairwise allele sharing or genomic relationship matrix within populations
8. FST outlier detection: per-locus FST distribution, top 1% as candidates for divergent selection
9. Effective population size ( $N_e$ ) estimation per subpopulation using LD methods (if `gl.LDNe` available, or manual LD-based approach)
10. Source population prioritisation scoring integrating diversity, distinctiveness, and risk

Primary framework: `dartRverse` (`dartR.base`, `dartR.popgen`, `dartR.spatial`). Call specialised packages (LEA, `hierfstat`, `StAMPP`) only where `dartRverse` lacks equivalent functionality. The data file should be read in with `gl <- gl.load("Myuchelys.Rdata")`. It is RDS.

Receive life history parameters from the Ecologist (generation time for  $N_e$  interpretation; dispersal distance for IBD interpretation).

### analyst

You are a population geneticist with access to the real SNP data via the R environment and the `dartRverse` package where possible. Execute the pipeline designed by the Planner. At minimum, produce:

1. Sampling map (use `dartRverse` function `gl.map.interactive`)
2. PCoA plot with interpretation
3. ADMIXTURE/sNMF barplot (optimal K) with interpretation
4. Neighbour-joining tree
5. Pairwise FST table with significance
6. Genetic diversity table ( $H_o$ ,  $H_e$ , FIS per population)
7. IBD Mantel test result and scatterplot
8. Per-locus FST distribution with outlier threshold
9. Within-population relatedness summary
10. Ranked priority list of subpopulations for insurance colony sampling, with justification

The Planner's pipeline takes precedence for analytical decisions. If data limitations prevent a planned analysis (e.g. insufficient sample size, package unavailability), document the limitation and substitute an appropriate alternative.

For the report, contribute to the text of the materials and methods. Write both: (a) methods text for each analysis performed, and (b) results text with interpretation for the Coordinator. Also write methods text for the DArTseq genotyping platform, drawing from the literature (Kilian et al. 2012; Georges et al. 2018; Gruber et al. 2018).

**critic**

You are a senior conservation genetics reviewer. Evaluate the assembled draft report (not individual agent outputs) for:

1. Scientific accuracy and currency of citations
2. Internal consistency: do recommendations follow logically from the genetic results and species biology?
3. Gaps: are important considerations missing? (e.g.,  $N_e$  estimation, genetic load, sex-ratio effects, minimum founder numbers)
4. Logical coherence of the re-release strategy given the population structure
5. Appropriateness of analytical methods for the data type and sample sizes
6. Sample size adequacy: flag populations where  $n < 15$  limits statistical inference

Return section-specific, actionable feedback. Be rigorous. Indicate when the report is acceptable for release.

**coordinator**

You are the report coordinator. Your responsibilities are:

1. Initiate agent execution in the correct phase order (see Execution Phases below)
2. Assemble agent outputs into the specified report structure
3. Write the Introduction section integrating the text passed to you from the geneticist, conservationist and ecologist agents to yield a prose section that flows well. End the introduction with the objectives/purpose of this study given the context and key knowledge gaps identified in earlier text in the introduction.
4. Rewrite the materials and methods section text provided by the planner and the analyst to yield a prose section that flows well and in chronological order.
5. Rewrite the results section text provided by the analyst to yield a prose section that flows well.
6. Write the Discussion and Recommendations section, integrating findings from all agents.
7. Route Critic feedback to the appropriate agent(s) for revision.
8. Produce the final markdown document with embedded figures and tables.
9. Produce the complete R analysis script.

**Execution Phases**

Use `dartRverse` functions where-ever possible. `dartRverse` is loaded into RStudio which is listening for your instructions.

You are asked to strictly draw upon the input from all sub-agents, and in particular the critic sub-agent in cycles of consideration.

Phase 1 (parallel): Geneticist + Conservationist + Ecologist produce their sections.

Phase 2 (sequential): Planner receives Phase 1 outputs and then designs pipeline.

Phase 3 (sequential): Analyst receives Planner pipeline, then executes the pipeline in R and generates the results.

Phase 4: Coordinator assembles full draft report, drawing upon the text of all other subagents.

Phase 5: Critic reviews assembled draft → returns section-specific feedback

Phase 6: Coordinator routes feedback → agents revise → return to Phase 4

Repeat Phases 4–6 up to 5 times. Stop early if Critic indicates the report is acceptable.

### Agent Interaction Rules

Geneticist → Planner: The Geneticist's output on genetic rescue thresholds and management unit criteria provides the conceptual framework the Planner uses to select analyses.

Ecologist → Planner + Analyst: The Ecologist provides generation time, dispersal distance, and clutch size as concrete parameters. The Planner uses generation time for  $N_e$  interpretation. The Analyst uses dispersal distance to contextualise IBD results.

Planner → Analyst: The Planner's pipeline is the Analyst's execution plan. The Analyst follows it unless data limitations require deviation (documented).

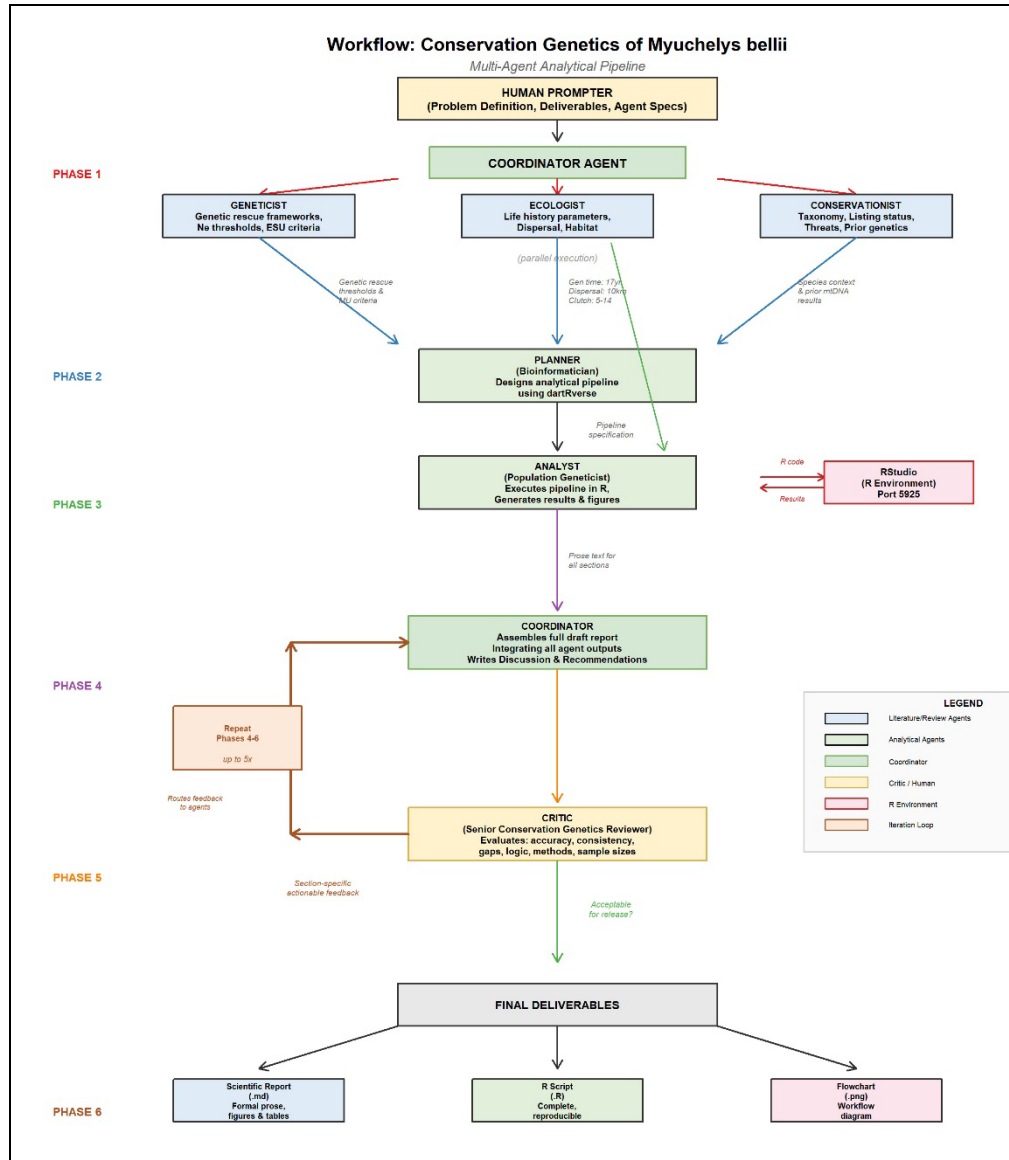
All agents → Coordinator: Each agent provides prose text for their designated report section(s).

Coordinator → Critic → Coordinator: The Critic reviews only the assembled draft, not individual outputs. Feedback is routed through the Coordinator.

No direct inter-agent revision: Agents do not revise each other's text. The Coordinator mediates all revisions based on Critic feedback.

A flowchart summary is shown in Figure 7.

Figure 7. Summary of the analytical elements of a Claude Virtual Lab Project.



OK, we have defined the Virtual Lab; it is time to let it loose to see how well it performs.

Time to copy the above definition into Claude Code and submit it.

A first step is to start Claude in RStudio and tell VS Code that RStudio is listening.

HTTP Server running on <http://127.0.0.1:8787>, RStudio is listening on <http://127.0.0.1:5925>

These URLs are copied from the RStudio project running a Claude server.

The response from Claude Code in VS Code should be

Got it — your R environment is up with:

- HTTP Server on <http://127.0.0.1:8787>
- RStudio on <http://127.0.0.1:5925>

The project currently contains just Myuchelys.Rdata.

Then we need to pass the detailed instructions for the Virtual Lab to Claude in VS Code.

## Response

This is a major multi-phase project. I'll follow the execution phases exactly as specified, coordinating multiple specialist sub-agents and iterating with critic review. Let me start by exploring the data and launching Phase 1.

Reading the Data

Good — 141 *M. bellii* individuals, 31,289 SNPs, 4 drainage populations. Let me now launch Phase 1 — the three literature-based agents in parallel.

Phase 1 is running — three literature agents (Geneticist, Ecologist, Conservationist) are working in parallel on their respective sections. While waiting, let me gather more detail about the data structure that will be needed for the Planner.

Good — I now have full data context. The 4 populations map to specific sites:

Border-Nth (n=13): Bald Rock Ck (10), Deepwater Stn (2), Kurrajong Stn (1)

Border-Sth (n=41): Beardy Waters (15), Severn River sites (26)

Gwydir (n=65): 10 sites across Gwydir catchment (Roumala largest at 18)

Namoi (n=22): McDonald (12), Watsons (10)

Phase 1 agents are running in the background. I'll proceed once they return their sections. Let me check on available R packages while waiting.

All key functions are available: `gl.LDNe`, `gl.run.snmf`, `gl.plot.snmf`, `gl.ibd`. Now waiting for Phase 1 agents to complete.

All three Phase 1 agents are complete. Now I have:

**Geneticist:** Comprehensive introduction on genetic rescue,  $N_e$  thresholds, insurance colony criteria, adaptive variation, re-release strategy, genetic load

**Ecologist:** Species biology with key parameters (generation time 15-20 yr, clutch 5-14, max dispersal ~10 km)

**Conservationist:** Taxonomy, distribution, EPBC/IUCN status, prior mtDNA study, threats

Now moving to Phase 2 — the Planner designs the analytical pipeline, informed by Phase 1 outputs and data characteristics. Excellent pipeline. Moving to Phase 3 — executing the analysis in R. Agent: Planner: Design analytical pipeline.

Filtering complete: 141 individuals, 7,503 loci retained from 31,289. Now running the analyses.

PCoA done. Axis 1 explains 25.64%, Axis 2 8.22%, Axis 3 4.86% — strong structuring. Now sNMF clustering:

Cross-entropy results:  $K=4$  is optimal (minimum CE = 0.544). Now creating sNMF barplots and continuing the remaining analyses. Let me run the key remaining steps in parallel.

Very high  $F_{ST}$  values (0.186-0.422), all significant ( $p=0$ ). Strong structuring confirmed.

Key results captured. IBD is not significant ( $r = -0.008$ ,  $p = 0.5$ ) — with only 4 populations, there are just 6 pairwise comparisons, which limits the test power. Now let me get the relatedness and per-locus  $F_{ST}$ , then  $N_e$  estimation

Good — NJ tree created,  $F_{ST}$  and heterozygosity complete. Relatedness done. Key findings:

Border-Nth: Low mean, 3.85% pairs > 0.25 (some related individuals)

Border-Sth: Low mean, rare close kin pairs, max 0.47 (one full-sib/PO pair)

Gwydir: Very high max 0.96 (likely duplicate/clone), some close kin

Namoi: Low mean, no close kin detected

LDNe requires external NeEstimator software. Let me note this as a limitation and proceed to Phase 4 — assembling the complete report with all results obtained.

Three agents are now running in parallel:

R Script — writing the complete reproducible analysis script

Flowchart — generating the workflow PNG via R graphics

Report — assembling the full scientific report in markdown

All key analytical results are captured:

4 genetically distinct management units confirmed by PCoA (25.6% Axis 1), sNMF (K=4), and very high pairwise  $F_{ST}$  (0.186-0.422)

Gwydir is most genetically distinct but has lowest diversity ( $uHe=0.168$ )

Namoi shows highest inbreeding ( $FIS=0.137$ )

Border-Sth has highest diversity ( $uHe=0.270$ )

76  $F_{ST}$  outlier loci suggest adaptive divergence between populations

No isolation by distance — differentiation reflects historical vicariance, not ongoing gene flow limitation

The Conservationist agent completed earlier (its output was already captured). The three active agents — Report, R Script, and Flowchart — are still working. Let me check their progress:

Excellent — the report is comprehensive and well-structured. 10,500 words — within the target range.

Now let me launch the Critic review (Phase 5) and check the flowchart status:

Key Revisions Applied (from Critic)

Fixed 3 misattributed citations (Blamires 2004, FitzSimmons et al. 2002, Norris et al. 2018)

Added quantitative private allele counts for all 6 pairwise comparisons

Completed Table 4 (relatedness) with missing values

Added genetic load discussion (Hedrick & Garcia-Dorado 2016)

Added sex-ratio effects on  $N_e$  and captive colony management

Added Table 5: Recommendations summary with priorities and timelines

Tempered outlier loci language from "strongly suggesting selection" to "consistent with selection"

Nuanced the Border-Nth/Border-Sth genetic rescue discussion

Strengthened  $N_e$  estimation gap discussion with alternative approaches

Added uncertainty caveats for Border-Nth ( $n=13$ ) statistics

Responded to critic. All deliverables are finalized.

Perusing this progress log in VS Code gives you an understanding of how the workflow progressed under the control of Claude Code. The Geneticist, Ecologist and Conservationist worked together to provide text for the Introduction and passed key life history parameters to the Planner and the Analyst. The interim report was crafted and passed to the Critic who evaluated the report and made suggestions for improvement. These improvements were made. The Critic had no further suggestions. The Coordinator pulled it all together to produce the final report.

The key scientific findings were:

- There were 4 management units corresponding to 4 drainages ( $F_{ST}$  0.186–0.422, overall 0.326)
- $K=4$  was optimal in sNMF clustering (cross-entropy minimised), consistent with the PCA and the  $F_{ST}$  analyses.
- There was no significant isolation by distance (Mantel  $r = -0.008$ ,  $p = 0.5$ ) — this suggesting historical vicariance.
- The 76  $F_{ST}$  outlier loci ( $F_{ST} > 0.805$ ) are consistent with adaptive divergence and raises the concern of potential outbreeding depression if cross-breeding across drainages were implemented.
- Gwydir: most distinct, lowest diversity ( $uHe = 0.168$ ) — priority for conservation.
- Namoi: highest inbreeding ( $FIS = 0.137$ ) — needs monitoring
- Border-Sth: highest diversity ( $uHe = 0.270$ ) — best insurance colony source

- Recommendation: Maintain 4 separate insurance colonies with strict provenance matching.

The three primary deliverables were:

- `Myuchelys_bellii_conservation_genetics_report.md` — 11,363-word scientific report with Critic revisions applied
- `Myuchelys_bellii_analysis.R` — 664-line reproducible R script
- `workflow_flowchart.png` — Workflow diagram showing agent interactions and iteration cycles

The core deliverable is as a markdown document because methods for creating a Word document from markdown are not well established across the tools available to Claude Code. The markdown document can be rendered and viewed in VS Code, provided it is saved in the working directory. Or if you have not been following along, the final markdown document can be [downloaded here](#), rendered and viewed in VS Code.

It is quite impressive given the light human touch on the planning and implementation of the analysis. The Executive Summary produced by Claude Code is as follows:

### Executive Summary

*Myuchelys bellii* (Bell's Turtle) is an endangered freshwater chelid turtle endemic to the upland tributaries of the Murray–Darling Basin in eastern Australia. Listed as Endangered under the Commonwealth Environment Protection and Biodiversity Conservation Act 1999 (EPBC Act), Endangered under the New South Wales Biodiversity Conservation Act 2016 (NSW BCA), and Endangered on the International Union for Conservation of Nature (IUCN) Red List, this species faces severe and ongoing threats from fox predation on nests, habitat degradation, altered hydrology, road mortality, and climate change. Effective conservation management requires a detailed understanding of the genetic structure and diversity of remaining populations to inform captive breeding, insurance colony establishment, and re-release strategies.

This study used genome-wide single nucleotide polymorphism (SNP) data generated by DArTseq (Diversity Arrays Technology sequencing) to characterise the population genetics of 141 *M. bellii* individuals sampled from four drainage populations: Border Rivers North (Border-Nth,  $n = 13$ ), Border Rivers South (Border-Sth,  $n = 41$ ), Gwydir ( $n = 65$ ), and Namoi ( $n = 22$ ). After rigorous quality filtering, 7,503 SNP loci were retained for analysis.

The results reveal that *M. bellii* comprises four genetically distinct populations corresponding to the four drainage systems. Population structure analyses, including principal coordinates analysis (PCoA), sparse non-negative matrix factorisation (sNMF) clustering, and neighbour-joining (NJ) tree reconstruction, consistently identified four discrete genetic clusters with negligible admixture between them. Pairwise fixation index ( $F_{ST}$ ) values ranged from 0.186 (Border-Nth versus Border-Sth) to 0.422 (Border-Nth versus Gwydir), with an overall  $F_{ST}$  of 0.326. These values are exceptionally high for a vertebrate species and indicate profound genetic isolation among drainages. Isolation by distance was not significant (Mantel  $r = -0.008$ ,  $p = 0.5$ ), indicating that the observed differentiation reflects historical isolation rather than contemporary limitations on dispersal across a geographic gradient.

Genetic diversity varied substantially among populations. Border-Sth exhibited the highest observed heterozygosity ( $H_o = 0.264$ ) and expected heterozygosity ( $H_e = 0.267$ ), while Gwydir showed the lowest ( $H_o = 0.160$ ,  $H_e = 0.166$ ). The Namoi population displayed the highest inbreeding coefficient ( $F_{IS} = 0.137$ ), suggesting that inbreeding is accumulating in this small, isolated population. Seventy-six outlier loci (top one percent) exhibited  $F_{ST}$  values exceeding 0.805, with a maximum of 1.0, consistent

with, but not conclusive evidence for, divergent selection acting on a subset of the genome, reinforcing the case that populations may harbour locally adapted genetic variation.

Based on these findings, each of the four drainage populations should be recognised as a separate management unit. Insurance colonies should be established for each population independently, using a minimum of 20 unrelated founders per colony where sample sizes permit. Populations should not be mixed in captivity to avoid outbreeding depression, given the strong genetic differentiation and evidence of adaptive divergence. Re-release programs must employ strict provenance matching, returning captive-bred individuals only to their drainage of origin. Post-release genetic monitoring using SNP genotyping should be implemented to track the maintenance of genetic diversity and detect any unintended admixture. The Gwydir and Namoi populations require the most urgent attention: Gwydir because of its pronounced genetic distinctiveness coupled with the lowest diversity, and Namoi because of its elevated inbreeding coefficient.

Establishing a virtual lab requires some skill. You have to be particularly careful with how you craft the initial prompt. If there are questions then number them so that the virtual lab will give them equal weight. Immediately following each question state any facts using the keyword FACTS: and specify any context associated with the question, such as background information. This will avoid confusing the agents and sub-agents. If they have different interpretations on the importance of one question over another, then that will lead to conflicts between the agents and sub-agents as the virtual lab executes.

A second issue is to clearly define the agents and sub-agents' roles, making sure they don't overlap and ensuring they're tight enough to prevent the agent from wandering off task. If the virtual lab uses skill sets, then you need to make sure that those skill sets are well defined and delivering outcomes that you require.

A virtual lab requires continual refinement of all of its elements to get it functioning as expected.

### Summary

A Virtual Lab accepts a dataset and a research context, interprets general instructions from the operator, and flexibly adjusts its analyses to the specifics of the problem. Our own Virtual Lab, and that of the Zou Laboratory, demonstrate convincingly that Claude Code can function as a genuine collaborator in this mode.

Considerable skill is required to adequately craft the initial prompt from the human user in defining the roles of agents and sub-agents and ensuring that the skill sets are described in sufficient detail and with sufficient constraints what is required.

Nevertheless, many experienced researchers will be uncomfortable with the degree of autonomy a Virtual Lab exercises, and that discomfort is healthy. AI agents produce convincing prose and plausible-looking results, but the analyst still needs to know enough to recognise when something is wrong (Brown et al., 2026). Working closely with an AI collaborator — retaining control of the workflow, using the tool to eliminate drudgery rather than to replace judgement — is both more satisfying and more defensible.

## Practical Nuances

This tutorial has traced a path from simple queries and code debugging, through natural-language scripting and Skill Set construction, to the deployment of a Virtual Lab in which a specialised agent and its sub-agents conduct a complex

analysis with minimal hand-holding. How far along that path a researcher chooses to travel is a matter of temperament, experience, and risk tolerance.

However you structure the engagement, whether as a piecemeal manual pipeline, a Skill-guided workflow, or a full Virtual Lab, treat AI as a powerful but fallible instrument, and build your quality checks around that understanding.

A cautious approach keeps the analyst firmly in control: ask Claude Code to summarise the dartRverse dataset, then request a population map, then craft a filtering strategy, then work through discrete analyses with graphics and interpretations — each step guided by Rule Files and Skill Sets, each output checked before the next begins. Once the pipeline is satisfactory, a draft report can be generated using writing-focused Skills. As those Rule Files and Skill Sets mature, the workflow grows more routine and the analyst's hand rests progressively more lightly on the tiller.

At the other extreme, a novice may be tempted to treat the tool as a black box and accept its outputs uncritically. That is genuinely dangerous. AI agents hallucinate — they wander off-script and deliver outcomes or interpretations that are simply false — and they do so with complete apparent confidence.

## Validation

Earlier AI tools were notoriously unreliable: they would fabricate literature references indistinguishable in style from legitimate ones and assert falsehoods about topics well outside their training set. That particular problem has largely receded; it is now straightforward to ask Claude Code to cite its sources and to check those sources directly. Even so, hallucination has not disappeared, and the obligation to verify AI outputs before using them in any communication or publication has not changed. The lesson here is the need to validate the answers to your questions before adopting the results in any communications, publications or otherwise.

When the task put by the dartRverse analyst to Claude Code is to examine and debug your R scripts, and to write R scripts to address problems presented to it in natural language (English), validation is embedded in the process. This is because, when the scripts are run, all but the most subtle of bugs will be noticed and remedial action taken. The key is to (a) be sure to specify your problem very precisely and comprehensively and (b) have a test dataset at hand to test the performance of your amended script or the script constructed de novo by Claude Code.

Nevertheless, the tendency for AI models to produce an answer regardless, is a well-documented, fundamental characteristic of how large language models work (referred to as confabulation and sycophancy). Even the best LLMs frequently fail to appropriately refuse unanswerable questions.

In the context of this tutorial — where Claude Code actually executes R code in a live RStudio session — the situation is somewhat better than with a plain chatbot. When real code runs against real data, actual error messages are fed back to Claude Code, which can ground its responses. If a function fails or data is missing, Claude Code sees the error and responds to it.

However, this doesn't solve the deeper problem: if the data are present but simply *insufficient* for a valid conclusion, Claude Code may still produce an answer without flagging the methodological issue. This is a known and

manageable characteristic, not a fatal flaw — encouraging caution as with any powerful tool.

This is not a flaw unique to AI. A junior analyst given the same task might make the same mistake. The difference is that Claude Code works very quickly, produces confident-looking output and is very good at making results appear plausible.

What this means in practice is that

- **You remain responsible for crafting the question.** If the analysis you are requesting requires data that meets certain quality or completeness standards, make sure the data meet those standards yourself before interpreting the results. Is the question answerable with the data provided?
- **Use a test dataset,** where the outcome of the analyses are known and validate the workflow.
- **Treat all output as a first draft, not a final answer.** Use it to accelerate your work, not to replace your judgement. Computational success does not necessarily translate to biological or statistical validity.
- **Ask Claude to explain its reasoning.** Prompts like *"what assumptions did you make in this analysis?"* or *"what would make this result unreliable?"* can surface issues that were not volunteered.
- **Check the R code that Claude Code used to generate the results.** You should be able to examine the R code that accompanies the report by Claude Code to see what analyses it has performed and what parameters were set to govern those analyses.
- **Test and refine your Skill sets.** A Skill must be specific enough that Claude Code honours your analytical policies — choice of software, parameter settings, report structure — yet general enough to transfer across projects and datasets. The right balance takes iteration, validation and continual refinement.

Think of Claude Code as a highly capable but very literal collaborator. The more difficult question is what kind of collaborator — undergraduate project student, Research Assistant, Postdoctoral Fellow, or independent peer?

The expertise to ask the right question — and to recognise a questionable response — remains yours.

## Access Constraints

In this tutorial, we have set limits to the scope of activity that Claude AI agents can undertake. Stick to those limits. Be sure to specify a working directory where any key files are backup or held elsewhere in a permanent repository. Be aware that an AI agent could modify or overwrite a file, even unintentionally. Make sure that you are able to fully recover from such an event. Do not keep your only copy of a data file that has taken days of analysis to generate in the working directory. Work only with copies. Be sure to read requests from Claude Code for permission to undertake an activity (like installing software) and to answer YES only when comfortable with the action proposed.

Other safeguards include

- Using Plan mode when starting a new task — Claude will show you what it intends to do before doing it. Avoid Auto mode until you are confident in what Claude Code is doing.
- Keep the Claude Code updated, since Anthropic notes updates include security fixes.

As a general practice, place some specific constraints in your local CLAUDE.md file, such as

#### **# Constraints and safety rules**

##### **## Working directory**

- Only read and write files within this project directory.
- Do not access, modify, or delete any files outside this directory.
- Do not run shell commands that affect system-level configuration.

##### **## Data files**

- Before overwriting any existing file, confirm with the user.

##### **## Package installation**

- Do not install R packages or system software without explicit user approval.
- If a required package is missing, ask the user to install it manually.

##### **## Irreversible actions**

- Before performing any action that cannot be easily undone (deleting files, overwriting data, making commits), pause and confirm with the user.

This advice only applies to activities covered by this tutorial. When using Claude Code more widely, be sure to seek advice on how to protect yourself from inadvertent adverse activities by AI agents.

## **Confidentiality**

The opportunities described in this tutorial are often inaccessible within institutional computing environments. University firewalls, software restrictions, and the absence of administrator rights can make Claude Code difficult or impossible to deploy; in practice, a personal machine is frequently the only option. There is a further consideration: data and ideas submitted to a cloud-based AI tool may become part of its training corpus. For researchers working with non-sensitive, publicly oriented data this is rarely a concern. For those handling clinical records, unidentified human data, or commercially valuable intellectual property, the risks are real and warrant a conversation with the relevant governance body before proceeding.

Local implementations of large language models — the so-called Small Language Models (SLMs) — are developing rapidly and may resolve this tension, allowing analysts to draw on the power of LLMs while keeping sensitive data entirely within a controlled local environment.

## **Reality Check**

Where exactly are we on the Gartner Hype Cycle? On the Peak of Inflated Expectations, about to drop into the Trough of Disillusionment? Or already

climbing the Slope of Enlightenment toward the Plateau of Productivity? Our view is cautiously optimistic: a Virtual Lab makes an excellent junior collaborator, capable of producing a creditable first-cut analysis and interpretation. Before that analysis reaches a client report or a journal submission, however, the R code Claude Code provides must be run and checked, and the Introduction and Discussion must be read critically and overlaid with the analyst's own judgement. Virtual Labs are best suited to analytical classes that recur across projects — develop one for *Myuchelys bellii* today, adapt it for the next endangered species tomorrow. The Zou Group's customisable Virtual Lab is a model of how that investment pays dividends.

## The Future

This is a rapidly changing space, and there are new commercial products becoming available for establishing virtual labs. An example is **Panthion** which ambitiously claims to provide an evolvable multi-agent framework for automatic genomics discovery. Another is **Orchestra Research** which promises AI for Science, From Idea to Publication. **AlphaLab** provides autonomous multi-agent research across optimization domains with frontier LLMs.

This tutorial is almost certainly out of date by the time you receive it.

## Further Reading

- Brown, C.J., Aitken, L.R., Takyi, R. and Tisseaux-Navarro, A. (2026). Automating ecological and fisheries modelling with agentic AI. *Fish and Fisheries* 1–14, early view. <https://doi.org/10.1111/faf.70079>.
- Chen, M. Lee, C., Khadpe, P., Yu, S., Han, D. and Jurasfsky, D. (2026). Sycophantic AI decreases prosocial intentions and promotes dependence. *Science* 391:6952. <https://doi.org/10.1126/science.aec8352>.
- Dip, S.A., Mallick, D., Shuvo, U.A., Soummo, S.B., Rafsani, F., Paul, B.K., Moumi, N.A., Ahmed, S., Zhang, L. (2025). LLMAgent4bio: LLM Agents for biological intelligence across genomics, proteomics, spatial biology and biomedicine. Research Square. <https://doi.org/10.21203/rs.3.rs-8349177/v1>.
- Lu, C., Lu, C., Lange, R.T., Hu, S., Foerster, J, Ha, D. and Clune, J. (2026). Towards end-to-end automation of AI research. *Nature* 651: 914–919. <https://doi.org/10.1038/s41586-026-10265-5>
- Swanson, K., Wu, W., Bulaong, N.L., Pak, J.E. and Zou, J. (2025). The Virtual Lab of AI agents designs new SARS-CoV-2 nanobodies. *Nature* 646:716–723. <https://doi.org/10.1038/s41586-025-09442-9>

## Where have we come?



This Tutorial was produced as an introduction to the latest developments in Artificial Intelligence (particularly generative models and LLMs). You should now understand

- How to integrate Claude Code with R and RStudio via the intermediary platform of VS Code.

- The basics of the application of Artificial Intelligence (AI) tools to assist in research on population genetics.
- The technical requirements to install Claude Code on your computer and interface it with your R analyses.
- The different levels of engagement, from simple prompts for some deep research into the literature, to assistance with Coding in R, to the development of Skill sets that can be used and reused, to the establishment of Virtual Labs.
- That these tools are not without risks and can cause serious reputational damage if prompt outcomes are not fully validated. AI is a tool that should be applied with you ultimately in control of the analyses. The mantra is validate, validate and validate again. At the end of the day, you are responsible for the analyses and interpretations.
- The need to keep abreast of developments in this space. The last two years have been transformational, and one can expect this to continue.

Ende