Profiling and Optimising R code

An introduction to `{profvis}` and `{bench}`

About this talk

Based on the Performance Profiling & Optimisation (Python) course made by Robert Chisholm

- One-day course available through myDevelopment

This is a very condensed version of that course translated to R

Structure

- The why’s, what’s, and when’s of Profiling
- Profiling in R
- Profiling Demo
- Optimisation and benchmarking
- Optimisation Demo

Why use Profiling

Quantifying code performance

- Execution time
- Memory usage

(We'll be focussing on execution time here)

For simple code something like system.time() may be enough, less so for complex code (or something which runs for hours)

Where is time being spent, and should/can I do anything about it?

"To make slow code faster, we need accurate information about what is making our code slow"

What is going on when I'm profiling code?

Generally the profiler will be tracking which code is being executed, when, and for how long for.

How it does this depends:

- Deterministic profiler: Keeps track of all code during execution
- Sampling profiler: Pauses the code at intervals to see what code is being executed

R only uses sampling profiling (which has pros and cons)

When should I profile my code?

Should be a quick process that allows identification of bottlenecks

Best for code that:

- Will be used repeatedly and takes more than a few mins to run.
- Is working and you are considering deploying

Types of profiling

- Manual
  - DIY profiling (e.g. system.time())
- Function-level
  - Focus on functions: How many times is each function called? How long is spent in a function overall and on average?
- Line-level
  - More granular than function-level profiling: How much time is spent on each line of code?
- Timeline
  - Subset of function-level profiling: Presents the execution as a timeline indicating order of function execution, child functions, and time spent in each function.

What should I profile?

Goldilocks problem:

- Too small and results may not be useful
- Too big and the results may be overwhelming (will also slow down execution)

Test case should be "*both representative of a typical workload and small enough that it can be quickly iterated*"

- Ideally profiling should take minutes

Profiling tools in R

Rprof() & summaryRprof()

- Tools built into base R, but output can be difficult to parse

`{profvis}`

- Package developed by RStudio and integrated with IDE
- Uses Rprof() but provides a more useful user interface

General Limitations

- Will only work on R code
  - Compiled (C/C++/Fortran) code needs to be profiled separately
- Using lots of anonymous functions may make it difficult to work out which functions are being called
  - Name the functions
- Likewise, R's lazy evaluation can complicate the call stack and make interpreting the output difficult
  - Can be solved with force() function

Quick demo of Rprof()

Introducing {profvis}

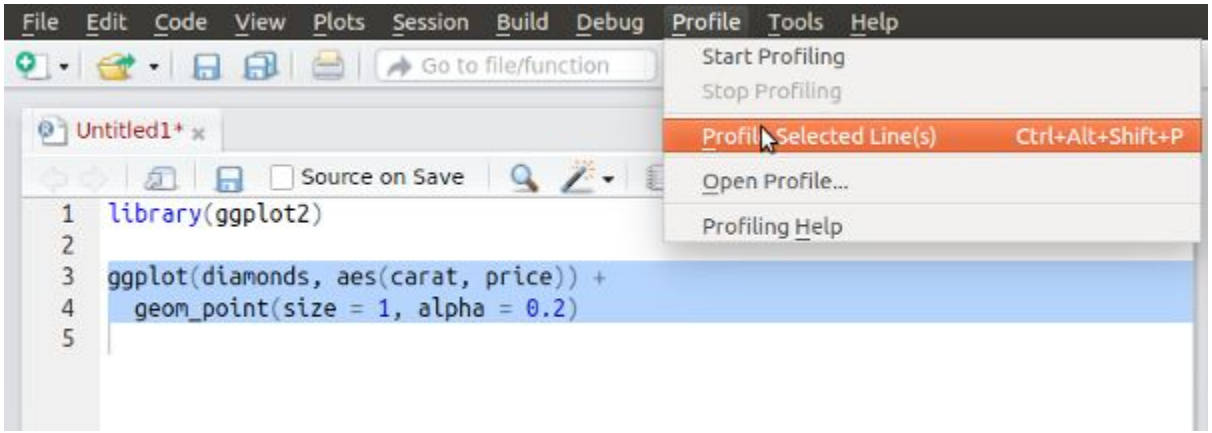
May need to be installed:

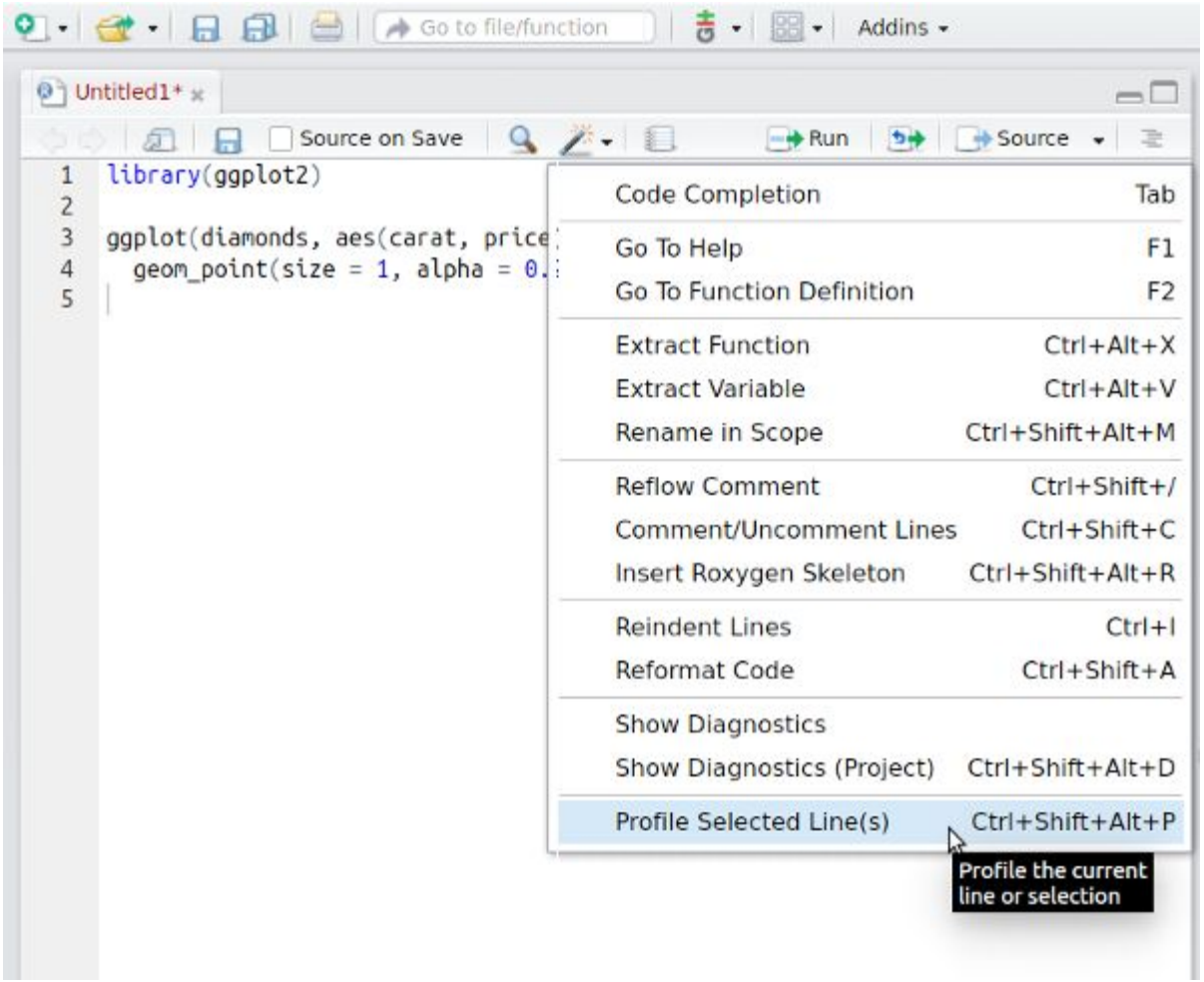
```
install.packages("profvis")
```

Can be run either as code:

```
profvis::profvis({ your code here })
```

Or through the Rstudio IDE


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Start Profiling
Stop Profiling
Profile Selected Line(s) Ctrl+Alt+Shift+P
Open Profile...
Profiling Help
Untitled1*
Source on Save
1 library(ggplot2)
2
3 ggplot(diamonds, aes(carat, price)) +
4 geom_point(size = 1, alpha = 0.2)
5


Go to file/function
Addins
Untitled1*
Source on Save
Run
Source
1 library(ggplot2)
2
3 ggplot(diamonds, aes(carat, price
4 geom_point(size = 1, alpha = 0.
5
Code Completion Tab
Go To Help F1
Go To Function Definition F2
Extract Function Ctrl+Alt+X
Extract Variable Ctrl+Alt+V
Rename in Scope Ctrl+Shift+Alt+M
Reflow Comment Ctrl+Shift+/
Comment/Uncomment Lines Ctrl+Shift+C
Insert Roxygen Skeleton Ctrl+Shift+Alt+R
Reindent Lines Ctrl+I
Reformat Code Ctrl+Shift+A
Show Diagnostics
Show Diagnostics (Project) Ctrl+Shift+Alt+D
Profile Selected Line(s) Ctrl+Shift+Alt+P
Profile the current line or selection

Profvis Demo 1
Tour of Profvis

Profvis Demo 2
More realistic example

Next step

Now you have an idea of where time is being spent in your code you can start thinking about optimising it.

This is typically an iterative process:

- Pick the worst performing part
- Try and improve it
- Repeat

Some rules of thumb for where you can speed things up

Optimising your code

- Vectorisation is fastest, then *apply (usually), then for-loops
- Matrices are faster than data frames
- Don’t grow data frames in a loop (e.g. using cbind, rbind, etc)
- Use different packages/functions (e.g. `data.table` is faster than `dplyr`, `dtplyr` sits between the two)
- Inbuilt pipe (|>) vs magrittr pipe (%>%)

Most important rule is to test your new code against the original:

- Could adjust and re-profile, but benchmarking may be more useful

Benchmarking

Measures the total runtime of a piece of code

Typically code is run multiple times to get a median execution time

Differs from profiling as it doesn't provide more granular timing breakdown

Two benchmarking packages under active development:

- `{bench}`
- `{microbenchmark}`

The {bench} package

Benchmarking tool from RStudio/Posit

`bench::mark()` takes lines of code doing the same thing and compares execution time for them

`bench::press()` is similar, but allows a grid of parameters to be supplied.

Things to be aware of with {bench}

Default is that the results of the code being being compared should be equal.

- Will throw an error if they are not
- Can be turned off using the `check` argument

Statistics generated by running each line of code repeatedly, but defaults mean that longer running code may only be run a few times.

- Samples may not be of comparable size
- Min sample can be specified with the `min_iterations` argument

Bench demo 1
Toy example

Bench demo 2
Optimising previous code

Resources

Advanced R by Hadley Wickham

- Chapter 23: Measuring Performance
- Chapter 24: Improving Performance

Efficient R programming by Colin Gillespie & Robin Lovelace

- Chapter 7: Efficient optimisation

`{profvis}` documentation

`{bench}` documentation