R/Insurance Webinars
Jan 2024

For the R Consortium’s R/Adoption Series

Welcome

Welcome

1. From Excel to programming in R
2. From programming in R to putting R into production (today's topic)
3. R performance culture
4. High performance programming in R

Delivered on behalf of the R Consortium by Georgios Bakoloukas and Benedikt Schamberger, Actuarial Control, Group Risk Management, Swiss Re

Background

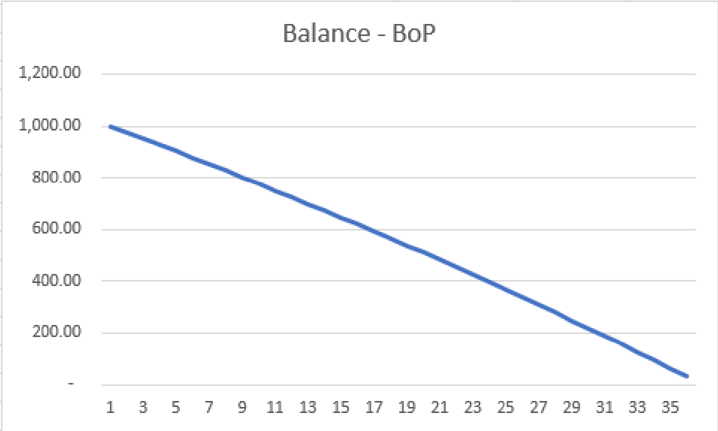
- Swiss Re internal R community sponsored by our Group Chief Actuary Philip Long (Atelier programme)
- 2000+ community with 500+ regular coders who also support each other
- The case we see today appeared in our Microsoft Teams community channel by an actuary in a high-growth market
- Views expressed belong solely to the speakers and not necessarily to the speaker's employer

Running example for webinars 1 & 2

- Insurer covers the remaining balance of loans in case of death/disability of the borrower
- Requires a quote for a portfolio of ca. 300,000 policies
- Has provided information on a) loan amount b) loan duration and c) interest rate for each policy
- Problem: The actuary needs to calculate the sum-insured profile for each policy as it amortises
- A solution in Excel and a potential solution in R
- Putting the solution ‘into production’ with R

A credit life insurance quote

Data input	
Loan Amount	1,000
Loan Term (in years)	3
Loan Term (in months)	36
Parameter input	
Interest Rate (Annual Percentage Rate)	10%
Monthly interest rate	0.83%
Modelling and Output	
Equivalent monthly payment	
EMI	32.27
Total Payments	1,161.62
Total Interest	161.62
Key	
Inputs	
Distinct calc in column	



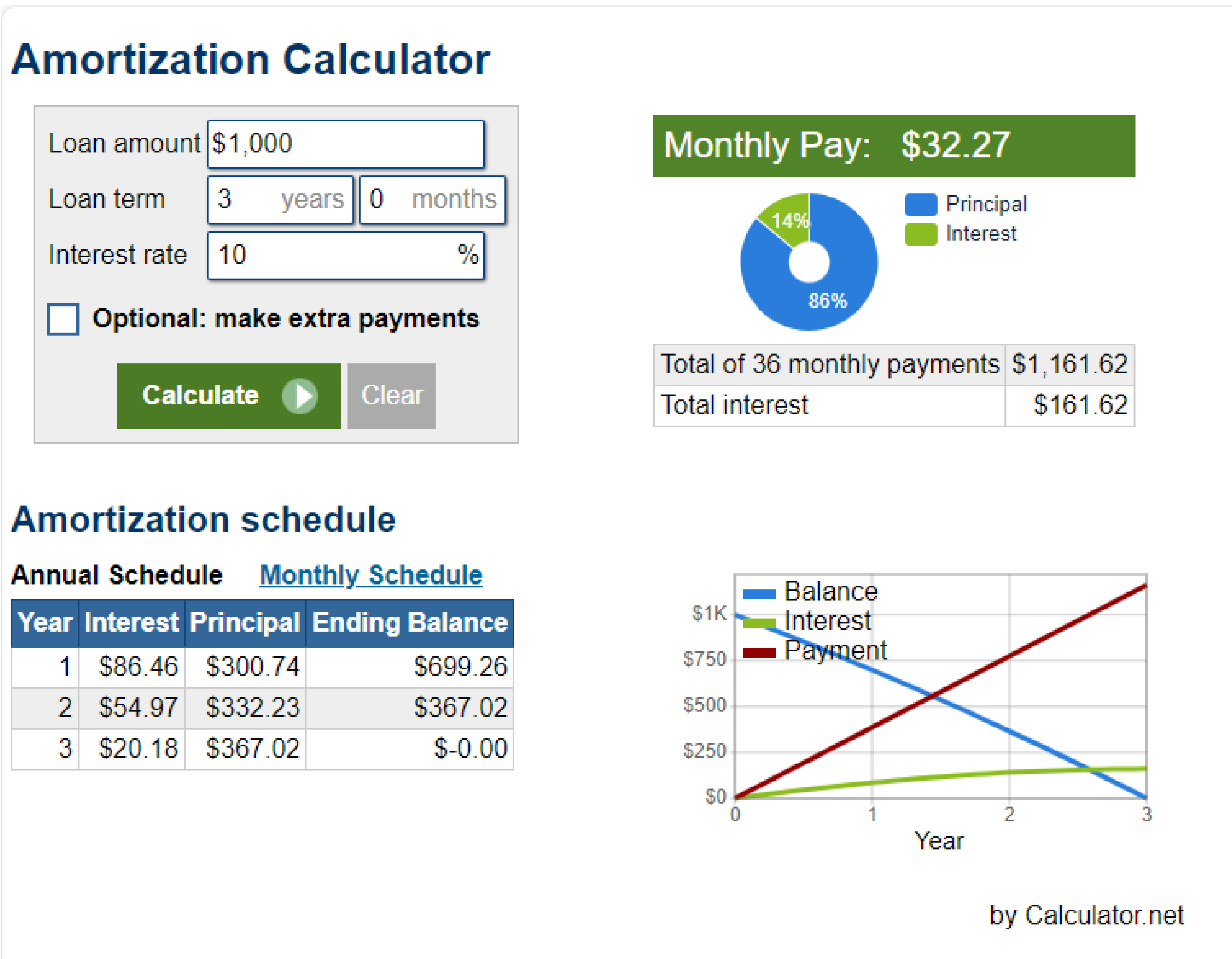
Modelling and output

Monthly cashflows

Time-months	Balance - BoP	Interest	Principal	Balance - EoP	Time-years
1	1,000.00	8.33	23.93	976.07	1
2	976.07	8.13	24.13	951.93	1
3	951.93	7.93	24.33	927.60	1
4	927.60	7.73	24.54	903.06	1
5	903.06	7.53	24.74	878.32	1
6	878.32	7.32	24.95	853.37	1
7	853.37	7.11	25.16	828.22	1
8	828.22	6.90	25.37	802.85	1
9	802.85	6.69	25.58	777.27	1
10	777.27	6.48	25.79	751.48	1
11	751.48	6.26	26.00	725.48	1
12	725.48	6.05	26.22	699.26	1
13	699.26	5.83	26.44	672.82	2
14	672.82	5.61	26.66	646.16	2
15	646.16	5.38	26.88	619.27	2
16	619.27	5.16	27.11	592.17	2
17	592.17	4.93	27.33	564.84	2
18	564.84	4.71	27.56	537.28	2
19	537.28	4.48	27.79	509.49	2
20	509.49	4.25	28.02	481.46	2
21	481.46	4.01	28.25	453.21	2
22	453.21	3.78	28.49	424.72	2
23	424.72	3.54	28.73	395.99	2
24	395.99	3.30	28.97	367.02	2
25	367.02	3.06	29.21	337.81	3
26	337.81	2.82	29.45	308.36	3
27	308.36	2.57	29.70	278.67	3
28	278.67	2.32	29.94	248.72	3
29	248.72	2.07	30.19	218.53	3
30	218.53	1.82	30.45	188.08	3
31	188.08	1.57	30.70	157.38	3
32	157.38	1.31	30.96	126.42	3
33	126.42	1.05	31.21	95.21	3
34	95.21	0.79	31.47	63.74	3
35	63.74	0.53	31.74	32.00	3
36	32.00	0.27	32.00	-	3

Graphical user interfaces available

eg https://www.calculator.net/amortization-calculator.html



Where we ended in webinar 1

```r
# Data and parameter input ---------------------------------
A <- 1000
n_yr <- 3
int_yr <- 0.1
# Intermediate calculations --------------------------------
n <- n_yr * 12
i <- int_yr / 12
emi <- (1 + i)^n / ((1 + i)^n - 1) * i * A
# Define amortisation function -----------------------------
amortise_one <- function(a, b) {a + a * i - emi}
# apply it successively to the loan amount -----------------
P <- purrr::accumulate(1:(n-1), amortise_one, .init = A)
P[1:6]                 # print first few results
```

```
[1] 1000.0000  976.0661  951.9328  927.5984  903.0612  878.3196
```

Today: Putting R into production

- Build functions to reuse logic and abstract away complexity
- Iterate over all data with functional programming approach
- Bundle functions into packages to share with others
- Expose functions into Shiny apps for non-programming use
- Expose functions into Web APIs for use by other apps

Functions

Abstracting complexity away

```r
calc_emi <- function(L, t, r) {
  emi <- (1 + r)^t / ((1 + r)^t - 1) * r * L
  emi
}
amort_helper_i <- function(x, y, r, emi_val) {x + x * r - emi_val}

amortise <- function(loan, term, rate) {
  term <- term * 12 # turn it into months
  rate <- rate / 12 # turn it to monthly effective rate
  emi <- calc_emi(L = loan, t = term, r = rate)
  amortised_loan <- purrr::accumulate(
    .x = c(loan, rep(0, term - 1)),   # c concatenates; rep repeats
    .f = ~ amort_helper_i(x = .x, r = rate, emi_val = emi)
  )
  amortised_loan
}
```

Try function

```
amortise(loan = 1000, term = 3, rate = 0.1)
```

```
 [1] 1000.00000  976.06615  951.93284  927.59843  903.06123  878.31955
 [7]  853.37170  828.21594  802.85055  777.27379  751.48388  725.47906
[13]  699.25753  672.81749  646.15711  619.27457  592.16800  564.83555
[19]  537.27533  509.48543  481.46396  453.20897  424.71852  395.99066
[25]  367.02339  337.81473  308.36267  278.66517  248.72019  218.52567
[31]  188.07953  157.37968  126.42399   95.21033   63.73657   32.00052
```

Automating with functions

- makes your code easier to understand
- update code in one place
- avoid copy and paste
- easier to reuse work

To learn more about functions in R, you may start at the Functions chapter from R for Data Science 2e by Wickham, Cetinkaya-Rundel and Grolemund which is freely available online

Iteration with functionals

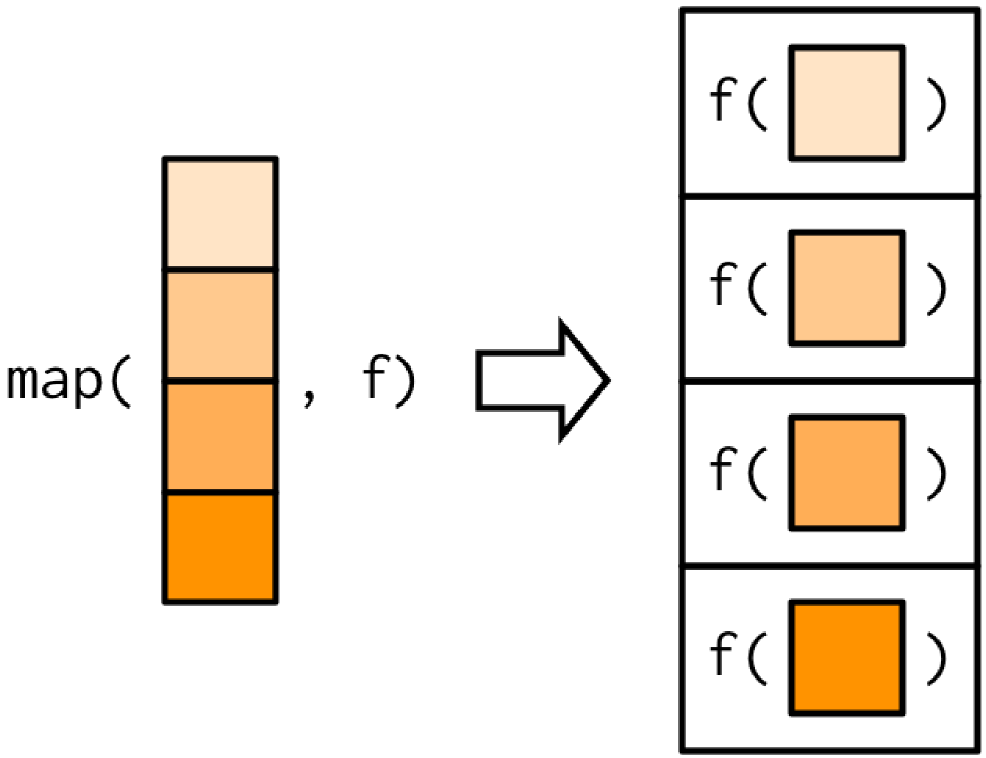
Create some data

```
z <- 1e3 # Number of customers (To Do: Find better name)
inforce <- tibble::tibble(
  customer_id = 1:z,
  loan_amount = pmax(100, round(rnorm(n = z, mean = 1000, sd = 100))),
  policy_term = sample(x = 2:30, size = z, replace = TRUE),
  interest_rate = sample(x = seq(8, 20, 0.25) / 100, size = z, replace = T)
)
inforce <- dplyr::bind_rows(
  tibble::tibble(
    customer_id = 0,
    loan_amount = 1000,
    policy_term = 3,
    interest_rate = 0.1),
  inforce
)
readr::write_csv(x = inforce, file = "data/client_data.csv")
```

Create some data

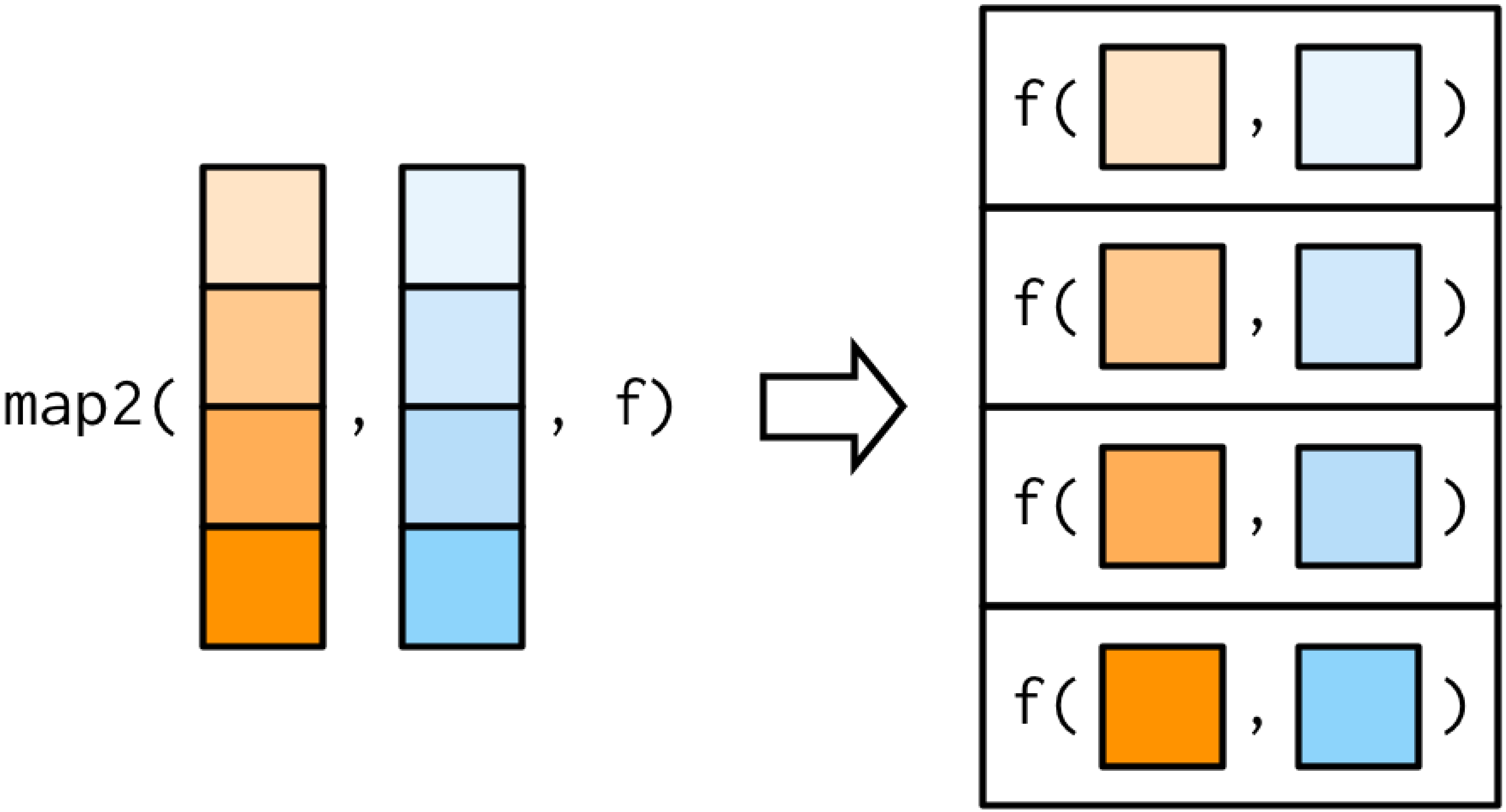
```
# A tibble: 1,001 × 4
   customer_id loan_amount policy_term interest_rate
         <dbl>       <dbl>       <dbl>         <dbl>
 1           0        1000           3         0.1  
 2           1        1018          22         0.155
 3           2         956          13         0.172
 4           3        1080          15         0.192
 5           4         899          11         0.155
 6           5         971           8         0.12 
 7           6        1241          21         0.135
 8           7        1010          21         0.185
 9           8         843          12         0.165
10           9         867          12         0.162
# ℹ 991 more rows
```

Functionals

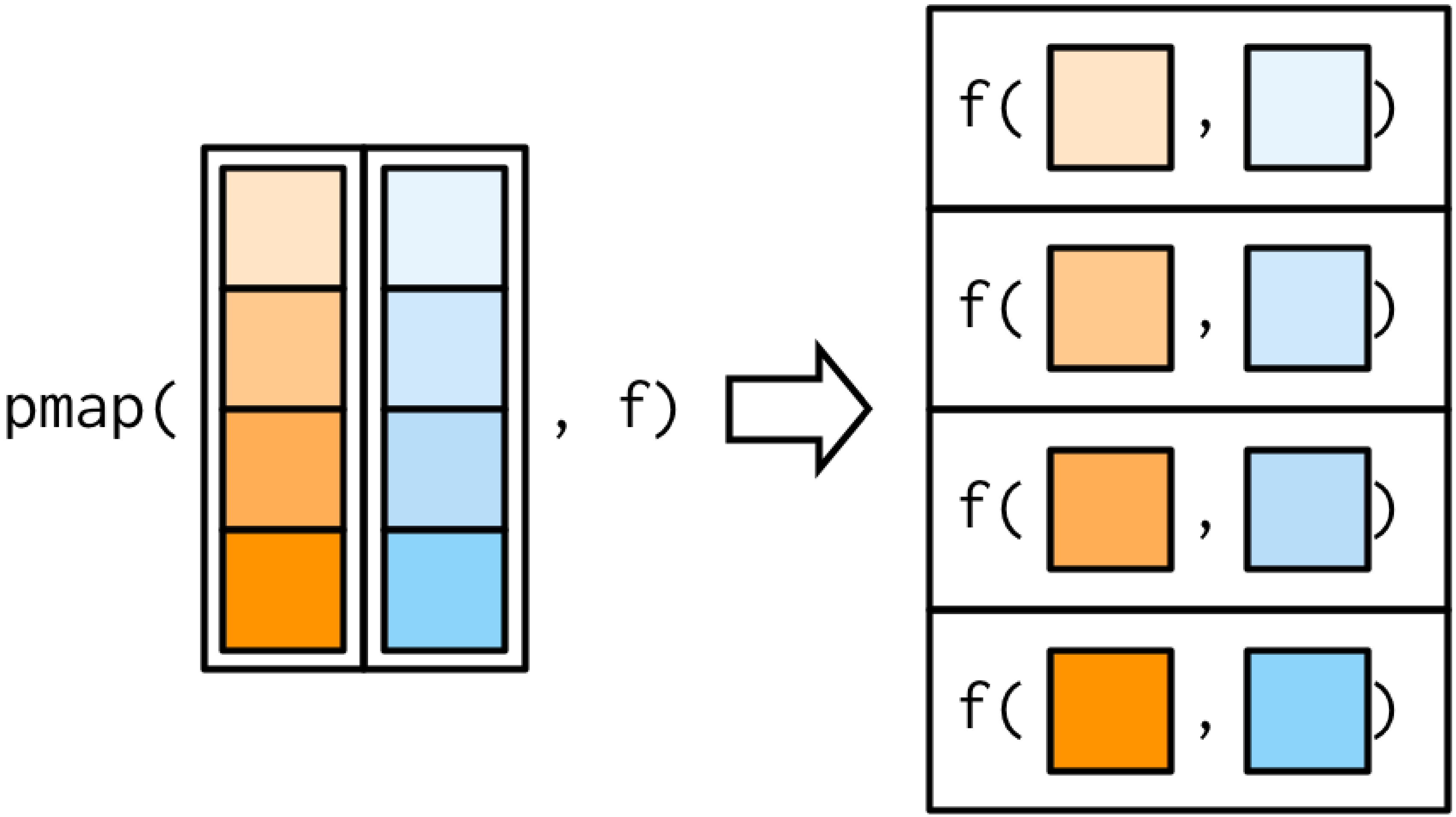


For more information about functionals please see the Iteration chapter from R for Data Science 2e

Functionals



Functionals



Single record: Pick the first record

```r
inforce |> slice(1)
```

```
# A tibble: 1 × 4
  customer_id loan_amount policy_term interest_rate
        <dbl>       <dbl>       <dbl>         <dbl>
1           0        1000           3           0.1
```

Single record: Apply pmap

```r
inforce |>
  dplyr::slice(1) |>                          # select first record
  dplyr::mutate(                              # create a new column
      amortised_loan = pmap(      # parallel mapping
        .l = list(..1 = loan_amount, ..2 = policy_term, ..3 = interest_rate
        .f = ~ amortise(loan = ..1, term = ..2, rate = ..3)
      )
  )
```

```
# A tibble: 1 × 5
  customer_id loan_amount policy_term interest_rate amortised_loan
        <dbl>       <dbl>       <dbl>         <dbl> <list>
1           0        1000           3           0.1 <dbl [36]>
```

Chapter 23 on Hierarchical data from R for Data Science talks more about list-columns and unnesting

Single record: Unnest the list-column

```r
inforce |>
  dplyr::slice(1) |>                         # select first record
  dplyr::mutate(                             # create a new column
    amortised_loan = purrr::pmap(      # parallel mapping
      .l = list(..1 = loan_amount, ..2 = policy_term, ..3 = interest_rate),
      .f = ~ amortise(loan = ..1, term = ..2, rate = ..3) |>
        tibble::enframe(name = "proj_month", value = "principal_bop")
      )
  ) |>
  tidyr::unnest(amortised_loan)
```

```
# A tibble: 36 × 6
  customer_id loan_amount policy_term interest_rate proj_month principal_bop
        <dbl>       <dbl>       <dbl>         <dbl>      <int>         <dbl>
1           0        1000           3           0.1          1         1000
2           0        1000           3           0.1          2          976.
3           0        1000           3           0.1          3          952.
4           0        1000           3           0.1          4          928.
5           0        1000           3           0.1          5          903.
6           0        1000           3           0.1          6          878.
7           0        1000           3           0.1          7          853.
8           0        1000           3           0.1          8          828.
9           0        1000           3           0.1          9          803.
```

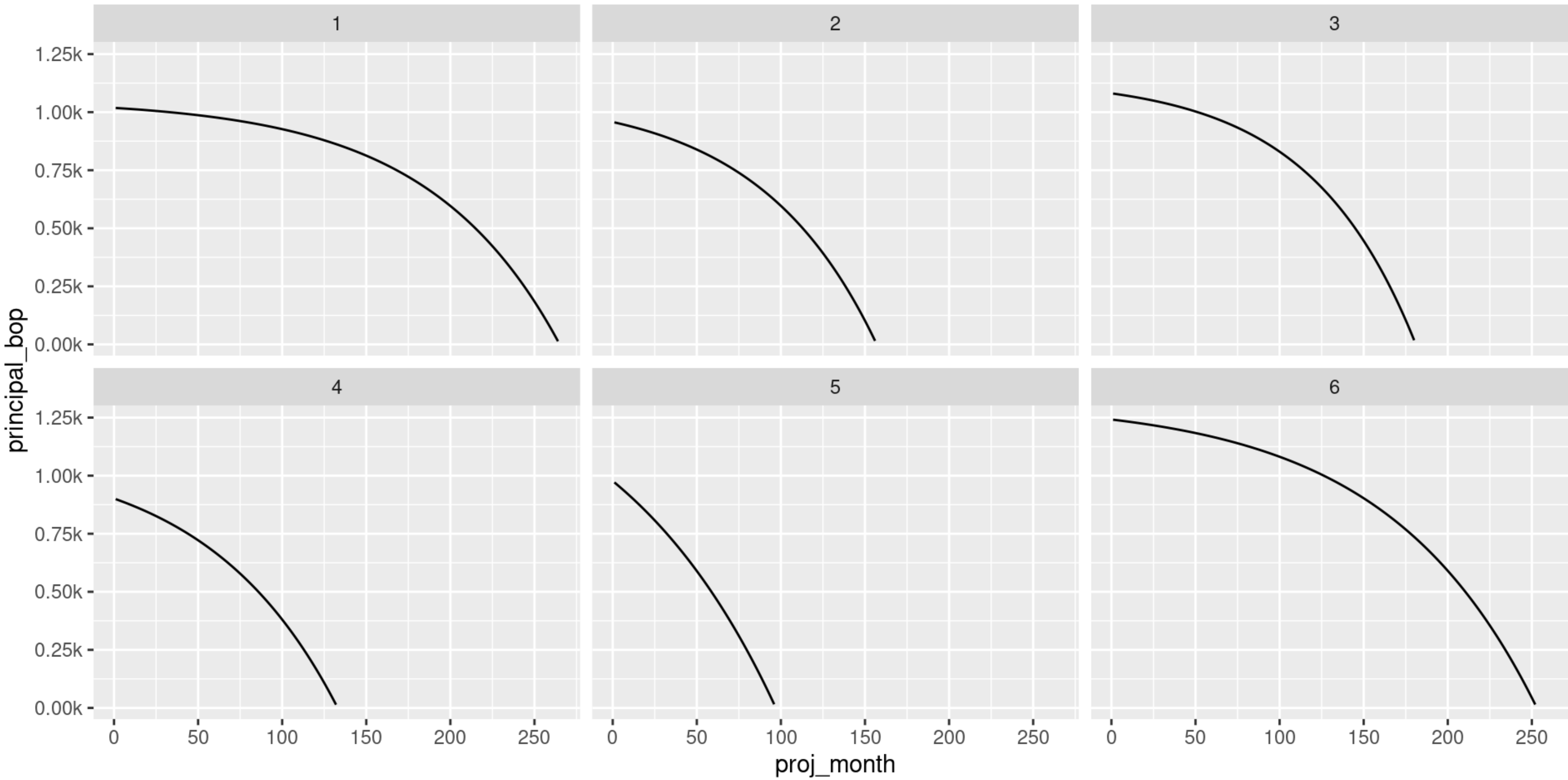
Run all records

```
result <-
  inforce |>
  mutate(                              # create a new column
      amortised_loan = pmap(      # parallel mapping
        .l = list(..1 = loan_amount, ..2 = policy_term, ..3 = interest_rate
        .f = ~ amortise(loan = ..1, term = ..2, rate = ..3) |>
        enframe(name = "proj_month", value = "principal_bop")
      )
  ) |>
  tidyr::unnest(amortised_loan)
```

Review the result as table

```
# A tibble: 192,036 × 6
   customer_id loan_amount policy_term interest_rate proj_month principal_bop
         <dbl>       <dbl>       <dbl>         <dbl>      <int>         <dbl>
 1           0        1000           3           0.1          1         1000 
 2           0        1000           3           0.1          2          976.
 3           0        1000           3           0.1          3          952.
 4           0        1000           3           0.1          4          928.
 5           0        1000           3           0.1          5          903.
 6           0        1000           3           0.1          6          878.
 7           0        1000           3           0.1          7          853.
 8           0        1000           3           0.1          8          828.
 9           0        1000           3           0.1          9          803.
10           0        1000           3           0.1         10          777.
# ℹ 192,026 more rows
```

Review sample records in a plot



Packages

The rationale

R packages are a familiar concept for sharing code with other R users.

In addition to sharing, there are other benefits that mean it can be a good idea even if you don’t plan on sharing your code widely.

These include ease of **documentation** and **testing**, and we will demonstrate some handy tools for managing these elements and others.

Packaging - getting started 1

A great place to start for anyone new to developing packages in R is the R Packages book freely available online .

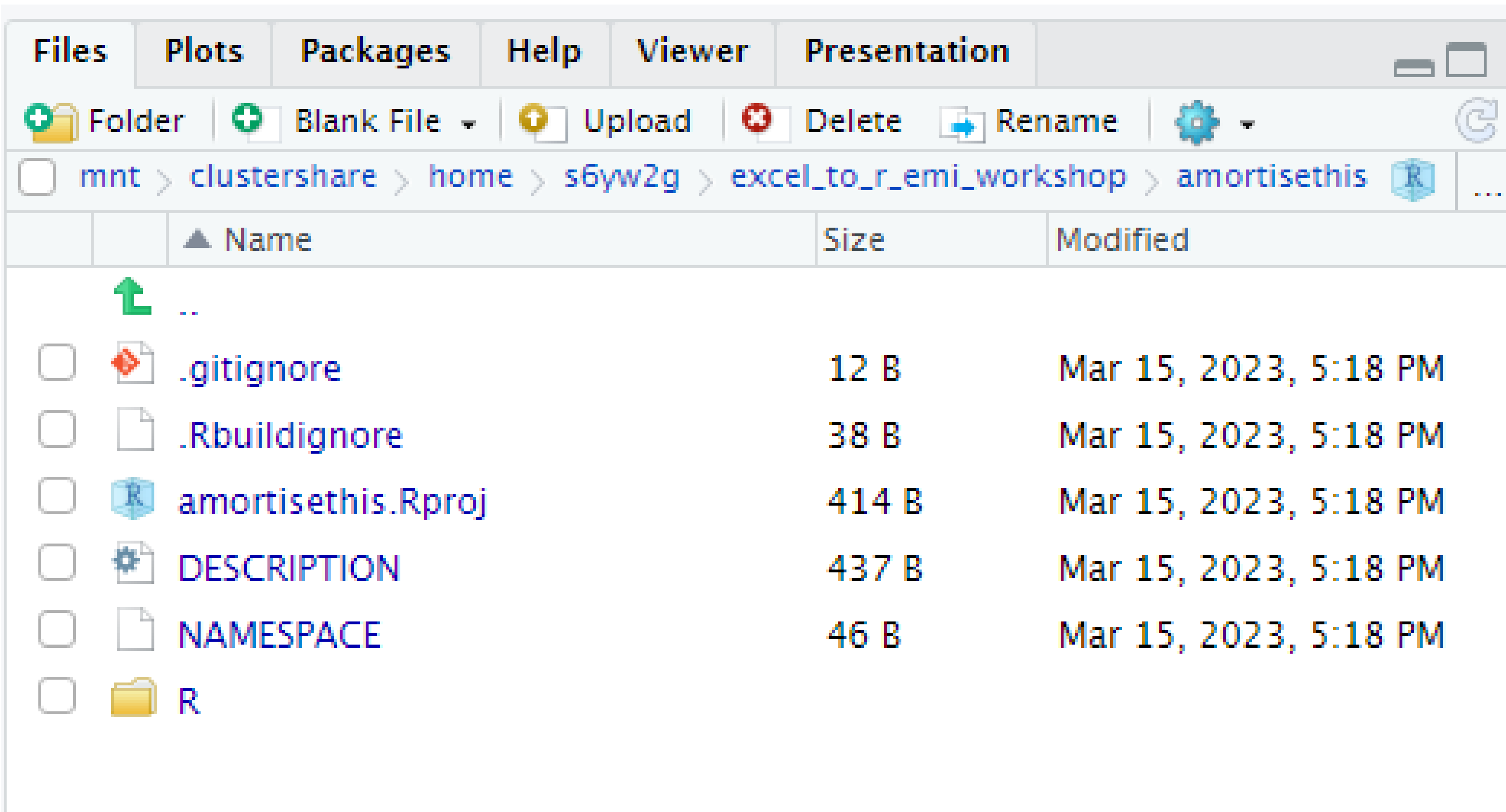
We'll use the `devtools` and `usethis` packages to help create and develop our package.

```
usethis::create_package("amortisethis")
```

```
✓ Creating 'amortisethis/'✓ Setting active project to '/mnt/clustershare/home/s6yw2g/excel_to_r_emi_workshop/amortisethis'✓ Creating 'R/'✓ Writing 'DESCRIPTION'Package: amortisethis
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R (parsed):
    * First Last <first.last@example.com> [aut, cre] (YOUR-ORCID-ID)
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends to
    pick a license
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
✓ Writing 'NAMESPACE'✓ Writing 'amortisethis.Rproj'✓ Adding '^amortisethis\\.Rproj$' to '.Rbuildignore'✓ Adding '.Rproj.user' to '.gitignore'✓ Adding '^\\.Rproj\\.user$' to '.Rbuildignore'✓ Opening '/mnt/clustershare/home/s6yw2g/excel_to_r_emi_workshop/amortisethis/' in new RStudio session✓ Setting active project to '<no active project>'
>
```

Packaging - getting started 2

In your new session you should see the following files



Packaging - metadata 1

The DESCRIPTION file contains fundamental package info - some of which we’ve populated here

DESCRIPTION

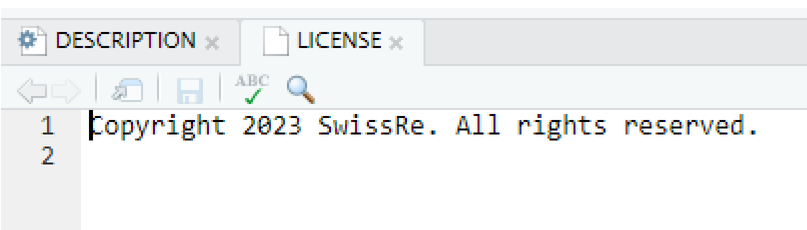
```
Package: amortisethis
Title: Calculate amortisation schedule for loans
Version: 0.0.0.9000
Authors@R:
    person("Tom", "Bowling", , "Tom.Bolwing@SwissRe.com", role = c("aut")),
    person("Georgios", "Bakoloukas", , "Georgios_Bakoloukas@swissre.com", role = c("aut", "cre"))
Description: Calculates amortisation schedules for loans based on the loan amount, the interest rate and the loan term.
License: `use_mit_license()`, `use_gpl3_license()` or friends to pick a
    license
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3

```

Packaging - metadata 2

The License field details how the package can be shared. We can use a `usethis` helper function to populate this for us

```
> usethis::use_proprietary_license("SwissRe")
✓ Setting License field in DESCRIPTION to 'file LICENSE'
✓ Writing 'LICENSE'
```

which creates this file



Packaging - metadata 3

The License field now looks like this

DESCRIPTION × LICENSE ×

```
Package: amortisethis
Title: Calculate amortisation schedule for loans
Version: 0.0.0.9000
Authors@R:
    person("Tom", "Bowling", , "Tom.Bolwing@SwissRe.com", role = c("aut")),
    person("Georgios", "Bakoloukas", , "Georgios_Bakoloukas@swissre.com", role = c("aut", "cre"))
Description: Calculates amortisation schedules for loans based on the loan amount, the interest rate and the loan term.
License: file LICENSE
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
```

Packaging - metadata 4

If we want to use any other packages inside our package, we must import them. Again, `usethis` has a helper function

```
> usethis::use_package("purrr")
✓ Adding 'purrr' to Imports field in DESCRIPTION
• Refer to functions with `purrr::fun()`
```

which adds this line to the DESCRIPTION file

```
Package: amortisethis
Title: Calculate amortisation schedule for loans
Version: 0.0.0.9000
Authors@R:
    person("Tom", "Bowling", , "Tom.Bolwing@SwissRe.com", role = c("aut")),
    person("Georgios", "Bakoloukas", , "Georgios_Bakoloukas@swissre.com", role = c("aut", "cre"))
Description: Calculates amortisation schedules for loans based on the loan amount, the interest rate and the loan term.
License: file LICENSE
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
Imports:
    purrr
```

Packaging - where to store your functions

In an R package, functions are stored in the R/ folder. We can again leverage `usethis`

```
> usethis::use_r("calc_emi")
• Modify 'R/calc_emi.R'
• Call `use_test()` to create a matching test file
```

which creates and opens a blank file, in to which we can enter our function

```r
calc_emi <- function(L, t, r) {
  emi <- (1 + r)^t / ((1 + r)^t - 1) * r * L
  return(emi)
}
```

Packaging - development workflow

Once all functions are added, we can load the package by running `devtools::load_all()`.

Now we can interactively test and use our new functions.

We can check on the status of our package using `devtools::check()`

This forms our general developer workflow:

1. Add/change some code
2. Load the changes and do some basic testing
3. Run devtools check to see that the package is still in good shape.

Documentation - the rationale

When we write functions, we generally expect them to be used again in the future, either by ourselves or by others

Well documented functions are easier to pick up and use than poorly documented ones

Good documentation reduces the amount of questions you receive as the author of the function, and allows users to be more efficient as they spend less time working out how to use it

Documentation - roxygen headers 1

R uses the roxygen framework, which enables you to document your functions in what are called headers. These take the form of metadata stored above the function definition. They can be inserted by pressing `ctrl+shift+alt+r` with your cursor inside the function

Documentation - roxygen headers 2

We fill in the details with as much info as we can/think will be helpful for other users

```r
#' Calculate EMI
#'
#' Calculate the equated monthly installment (EMI) for a loan
#' given the loan value, term and interest rate.
#'
#'
#' @param L {numeric} - the loan value
#' @param t {integer} - the loan term in months
#' @param r {double} - the interest rate as a decimal - e.g. 0.05
#'
#' @return a numeric vector of length 1
#' @export
#'
#' @examples
#' emi(L = 1000, t = 12, r = 0.05)
calc_emi <- function(L, t, r) {
  emi <- (1 + r)^t / ((1 + r)^t - 1) * r * L
  return(emi)
}

```

Documentation - rendering docs

To render the docs we'll use `devtools::document()`

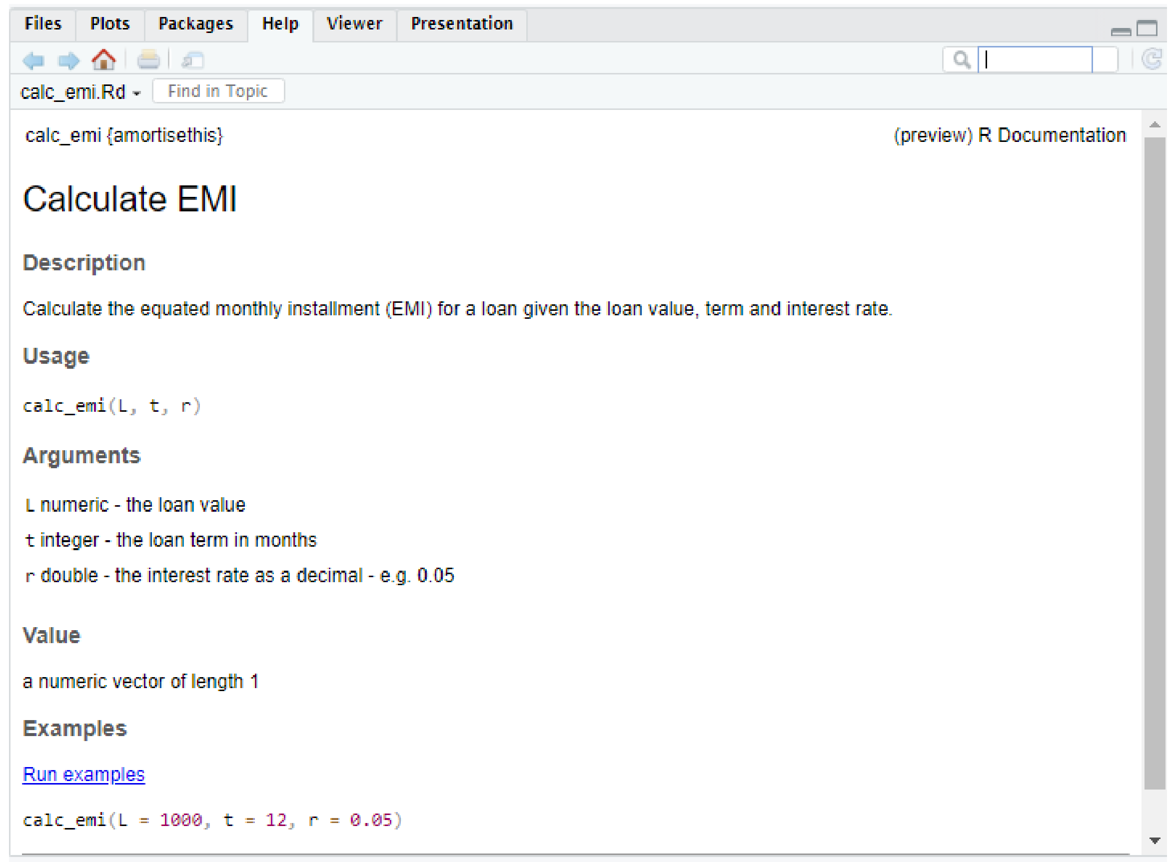
```
> devtools::document()
i Updating amortisethis documentation
i Loading amortisethis
Writing NAMESPACE
Writing calc_emi.Rd
```

Documentation - viewing docs

Users can access the help for functions in our package just like any other, either with

```
?calc_emi
```

or by pressing F1 with the function name highlighted, or by searching in the help pane



Testing - the rationale

Unit testing is a way of confirming that all functions are working as expected

Automating these tests reduces the amount of time that a developer spends checking outputs when they make changes to code

Running `devtools::check()` runs any tests in your package, so embedding that step in your developer workflow means you're more likely to catch any bugs before they get to your end users

Testing - testthat framework 1

One of the most commonly used testing frameworks in R is `testthat`. We can use this framework in our package with `usethis`

```
> usethis::use_testthat()
✓ Setting active project to '/mnt/clustershare/home/s6yw2g/excel_to_r_emi_workshop/amortisethis'
✓ Adding 'testthat' to Suggests field in DESCRIPTION
✓ Setting Config/testthat/edition field in DESCRIPTION to '3'
✓ Creating 'tests/testthat/'
✓ Writing 'tests/testthat.R'
• Call `use_test()` to initialize a basic test file and open it for editing.
```

which creates the test folder in our package

Testing - testthat framework 2

The testthat.R script contains set-up code that is run before the tests, for now it is pretty basic

```r
# This file is part of the standard setup for testthat.
# It is recommended that you do not modify it.
#
# Where should you do additional test configuration?
# Learn more about the roles of various files in:
# * https://r-pkgs.org/tests.html
# * https://testthat.r-lib.org/reference/test_package.html#special-files

library(testthat)
library(amortisethis)

test_check("amortisethis")
```

Testing - a basic test 1

To set up our first test, again we turn to `usethis`

```
> usethis::use_test("calc_emi")
✓ Writing 'tests/testthat/test-calc_emi.R'
• Modify 'tests/testthat/test-calc_emi.R'
```

which creates a dummy test for us in the tests/testthat folder

calc_emi.R | testthat.R | test-calc_emi.R

```
test_that("multiplication works", {
  expect_equal(2 * 2, 4)
})
```

Testing - a basic test 2

We might test such things as the type of the output, the size of the output, and the value.

```r
test_that("multiplication works", {

  result <- calc_emi(1000, 12, 0.05)

  expect_equal(is.numeric(result), TRUE)
  expect_length(result, 1)
  expect_equal(result, 112.8254)

})
```

we can execute our tests using

```
> devtools::test()
i Testing amortisethis
✓ | F W S  OK | Context
✓ |          3 | calc_emi

== Results ==============================
Duration: 0.2 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 3 ]
>
```

Testing - a testing checklist

When first starting out, it can be hard to know what to test.

A basic checklist would cover

- expected inputs -> expected outputs
- unexpected inputs -> expected error handling

Testing - defensive programming 1

The second item on our testing checklist leads us to defensive programming

Consider the following, are either of them desirable

```
> calc_emi("$1000", 12, 0.05)
Error in (1 + r)^t/((1 + r)^t - 1) * r * L :
  non-numeric argument to binary operator
>
```

```
> calc_emi(1000, -12, 0.05)
[1] -62.82541
```

Defensive programming enables us to mitigate for these sorts of situations

Testing - defensive programming 2

So we might do something like this

```r
#' Calculate EMI
#'
#' Calculate the equated monthly installment (EMI) for a loan
#' given the loan value, term and interest rate.
#'
#'
#' @param L {numeric} - the loan value
#' @param t {integer} - the loan term in months
#' @param r {double} - the interest rate as a decimal - e.g. 0.05
#'
#' @return a numeric vector of length 1
#' @export
#'
#' @examples
#' emi(L = 1000, t = 12, r = 0.05)
calc_emi <- function(L, t, r) {

  if (!is.numeric(L) | L <= 0) stop("L must be numeric and positive")
  if (!is.numeric(t) | t <= 0) stop("t must be numeric and positive")
  if (!is.numeric(r) | r <= 0) stop("r must be numeric and positive")

  if (r >= 1) warning("r should be a decimal representation e.g. for 5% r should be 0.05 - a value of 1 relates to a rate of 100%")

  if (t %% 1 != 0) {
    warning("t must be a whole number, t will be rounded to the nearest value")
    t <- as.integer(t)
  }

  emi <- (1 + r)^t / ((1 + r)^t - 1) * r * L
  return(emi)
}
```

Testing - defensive programming 3

which would then return the following in practice

```
> calc_emi("$100", 12, 0.05)
Error in calc_emi("$100", 12, 0.05) : L must be numeric and positive
> calc_emi(100, 12, 5)
[1] 500
Warning message:
In calc_emi(100, 12, 5) :
  r should be a decimal representation e.g. for 5% r should be 0.05 - a value of 1 relates to a rate of 100%
```

Testing - defensive programming 4

Our tests for the unexpected inputs could look like this

```
test_that("calc_emi works", {

  result <- calc_emi(1000, 12, 0.05)

  expect_equal(is.numeric(result), TRUE)
  expect_length(result, 1)
  expect_equal(result, 112.82541)


  expect_error(calc_emi("$1000", 12, 0.05), 'L must be numeric and positive')
  expect_error(calc_emi(-1000, 12, 0.05), 'L must be numeric and positive')
  expect_error(calc_emi(1000, "12 months", 0.05), 't must be numeric and positive')
  expect_error(calc_emi(1000, -12, 0.05), 't must be numeric and positive')
  expect_warning(calc_emi(1000, 12.3, 0.05), 't must be a whole number, t will be rounded to the nearest value')
  expect_error(calc_emi(1000, 12, "5%"), 'r must be numeric and positive')
  expect_error(calc_emi(1000, 12, -5), 'r must be numeric and positive')
  expect_warning(calc_emi(1000, 12, 5), 'r should be a decimal representation e.g. for 5% r should be 0.05 - a value of 1 relates to a rate of 100%')
})
```

Testing - coverage

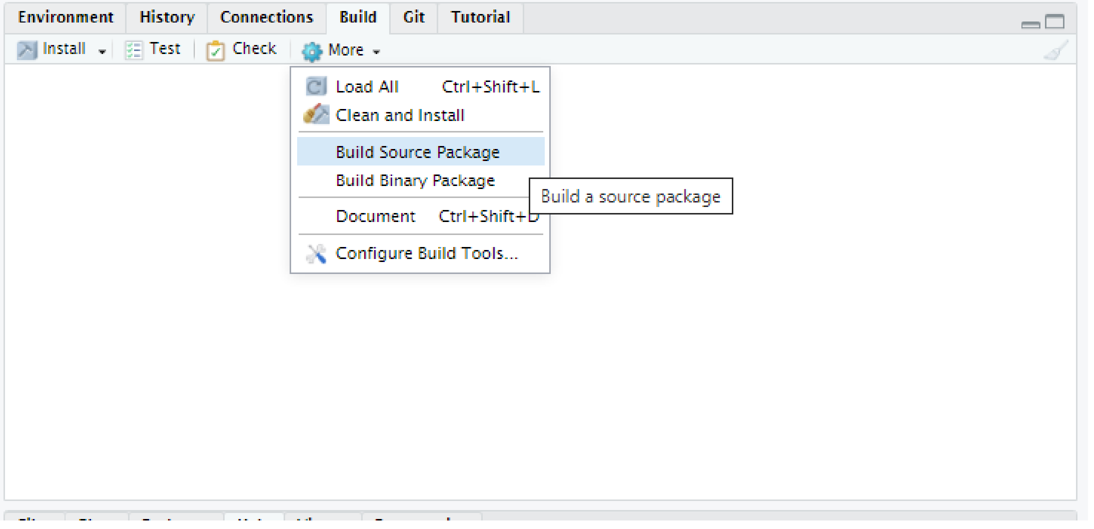
Test coverage looks at what % of lines of our code are run as part of our unit tests

The R package covr provides a nice way to look at this

```
> covr::package_coverage()
amortisethis Coverage: 56.25%
R/amortise.R: 0.00%
R/calc_emi.R: 100.00%
```

Sharing packages within our organisation

To add our package to our designated package manager (eg an internal to the organisation Posit Package Manager installation), we must build our package and upload it to our server.



Web apps (Shiny)

Sharing: Web apps

If we wanted to share the calculation with a user who had no familiarity with R, we could use R's `shiny` framework to build a simple web app

Sharing: Web apps - full app.R file

```r
library(shiny)
library(amortisethis)
library(ggplot2)
library(tidyr)
library(dplyr)

# Define UI for application
ui <- fluidPage(

  titlePanel("Basic Amortisation App"),
  # Sidebar with inputs
  sidebarLayout(
    sidebarPanel(
      numericInput("loan_val",
                   "Loan Value:",
                   value = 1000,
                   min = 1
      ),
      numericInput("loan_term",
                   "Loan Term:",
                   value = 12,
                   min = 1
      ),
      numericInput("rate",
                   "Interest Rate:",
                   value = 0.05,
                   min = 0
      ),
      actionButton("calc",
                   "Calculate")
    ),
    # Show a plot of the generated schedule
    mainPanel(
      plotOutput("Results_plot"),
      tableOutput("Results_table")
    )
  )
)

# Define server logic required to draw a histogram
server <- function(input, output) {

  result <- reactiveValues()

  observeEvent(input$calc, {

    balance <- amortisethis:::amortise(loan = input$loan_val,
                                       term = input$loan_term,
                                       rate = input$rate)
    interest <- input$rate * balance
    principal <- amortisethis::calc_emi(input$loan_val, input$loan_term, input$rate) - interest

    result$schedule <- data.frame(period = 1:input$loan_term,
                                  opening_balance = balance,
                                  interest = interest,
                                  principal = principal,
                                  closing_balance = dplyr::lead(balance, n = 1, default = 0))

  })

  output$Results_plot <- renderPlot({
    req(result$schedule)
    result$schedule |>
      pivot_longer(!period, names_to = "type", values_to = "amount") |>
      filter(type != "opening_balance") |>
    ggplot(aes(x = .data$period, y = .data$amount, color = .data$type)) +
      scale_color_manual(values = c(srColors::sr_bougainvillea, srColors::sr_blue_sky, srColors::sr_lake)) +
      geom_line() +
      labs(x = "Term", y = "Amount", title = "Amortisation Schedule")

  })

  output$Results_table <- renderTable({
    req(result$schedule)
    result$schedule
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Sharing: Web apps - UI code

```
# Define UI for application
ui <- fluidPage(

  titlePanel("Basic Amortisation App"),

  # Sidebar with inputs
  sidebarLayout(
    sidebarPanel(
      numericInput("loan_val",
                   "Loan Value:",
                   value = 1000,
                   min = 1
      ),
      numericInput("loan_term",
                   "Loan Term:",
                   value = 12,
                   min = 1
      ),
      numericInput("rate",
                   "Interest Rate:",
                   value = 0.05,
                   min = 0
      ),
      actionButton("calc",
                   "Calculate")
    ),
    # Show a plot of the generated schedule
    mainPanel(
      plotOutput("Results_plot"),
      tableOutput("Results_table")
    )
  )
)
```

Sharing: Web apps - Server code

```r
# Define server logic required to draw a histogram
server <- function(input, output) {

  result <- reactiveValues()

  observeEvent(input$calc, {

    balance <- amortisethis:::amortise(loan = input$loan_val,
                                       term = input$loan_term,
                                       rate = input$rate)

    interest <- input$rate * balance
    principal <- amortisethis::calc_emi(input$loan_val, input$loan_term, input$rate) - interest

    result$schedule <- data.frame(period = 1:input$loan_term,
                                  opening_balance = balance,
                                  interest = interest,
                                  principal = principal,
                                  closing_balance = dplyr::lead(balance, n = 1, default = 0))

  })

  output$Results_plot <- renderPlot({
    req(result$schedule)
    browser()
    result$schedule |>
      pivot_longer(!period, names_to = "type", values_to = "amount") |>
      filter(type != "opening_balance") |>
    ggplot(aes(x = .data$period, y = .data$amount, color = .data$type)) +
      scale_color_manual(values = c(srColors::sr_bougainvillea, srColors::sr_blue_sky, srColors::sr_lake)) +
      geom_line() +
      labs(x = "Term", y = "Amount", title = "Amortisation Schedule")

  })

  output$Results_table <- renderTable({
    req(result$schedule)
    result$schedule
  })
}
```

Sharing: Web apps - Run locally

Basic Amortisation App

Loan Value:

1000

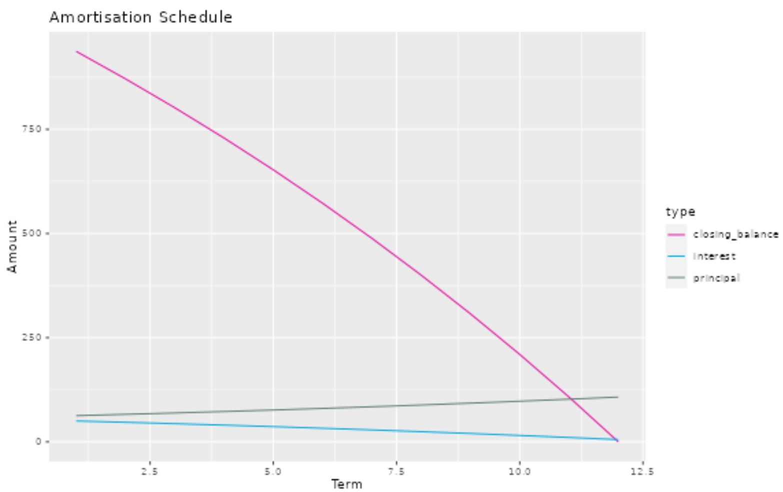
Loan Term:

12

Interest Rate:

0.05

Calculate



period	opening_balance	interest	principal	closing_balance
1	1000.00	50.00	62.83	937.17
2	937.17	46.86	65.97	871.21
3	871.21	43.56	69.27	801.94
4	801.94	40.10	72.73	729.21
5	729.21	36.46	76.36	652.85
6	652.85	32.64	80.18	572.67
7	572.67	28.63	84.19	488.47
8	488.47	24.42	88.40	400.07
9	400.07	20.00	92.82	307.25
10	307.25	15.36	97.46	209.79
11	209.79	10.49	102.34	107.45
12	107.45	5.37	107.45	0.00

Sharing: Web apps - Publish application

Sharing: Web apps - Deployed Application

Link for demonstration only: not available outside Swiss Re

https://rstudioconnect.atelier.swissre.com/amortise_app_test/

Web APIs

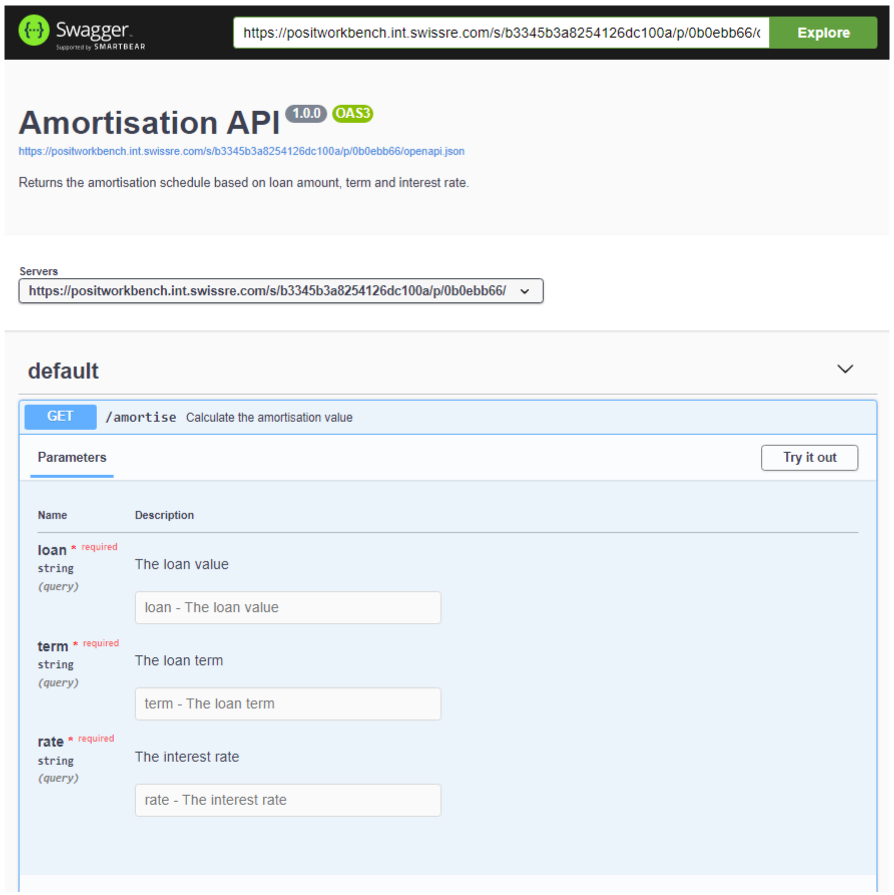
Sharing: Web API

If we want other systems to interact with our functions, we can use R’s `plumber` framework to deploy our functions as an API.

```r
#
# This is a Plumber API. You can run the API by clicking
# the 'Run API' button above.
#
# Find out more about building APIs with Plumber here:
#
#    https://www.rplumber.io/
#

library(plumber)
library(amortisethis)

#* @apiTitle Amortisation API
#* @apiDescription Returns the amortisation schedule based on loan amount, term and interest rate.

#* Calculate the amortisation value
#* @param loan The loan value
#* @param term The loan term
#* @param rate The interest rate
#* @get /amortise
function(loan, term, rate) {

    amortisethis:::amortise(as.numeric(loan),
                            as.numeric(term),
                            as.numeric(rate))
}

```

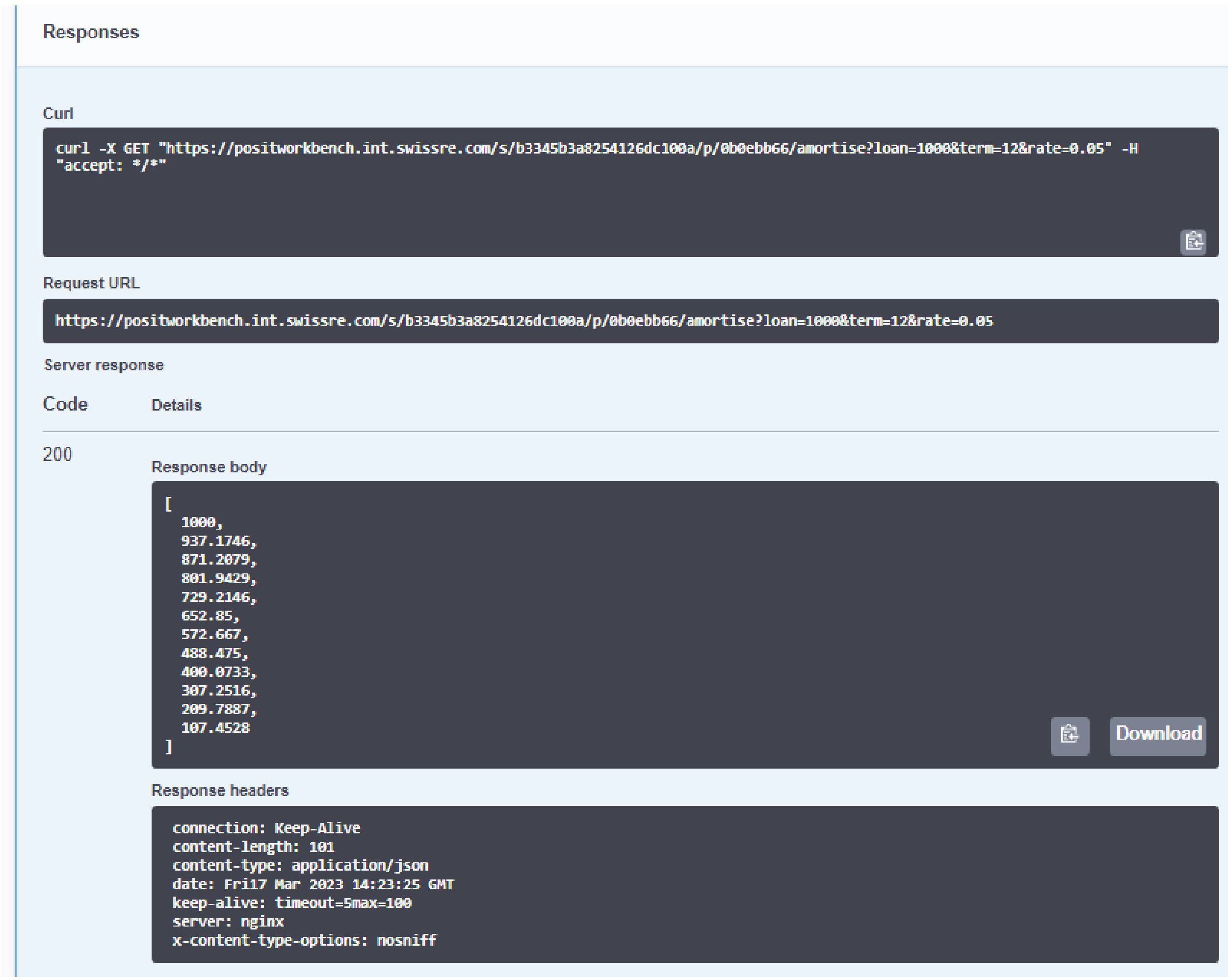
Sharing: Web API

We can test locally by hitting run API, it generates a test interface for us



Sharing: Web API

If we fill in the values and hit execute, we can see the output

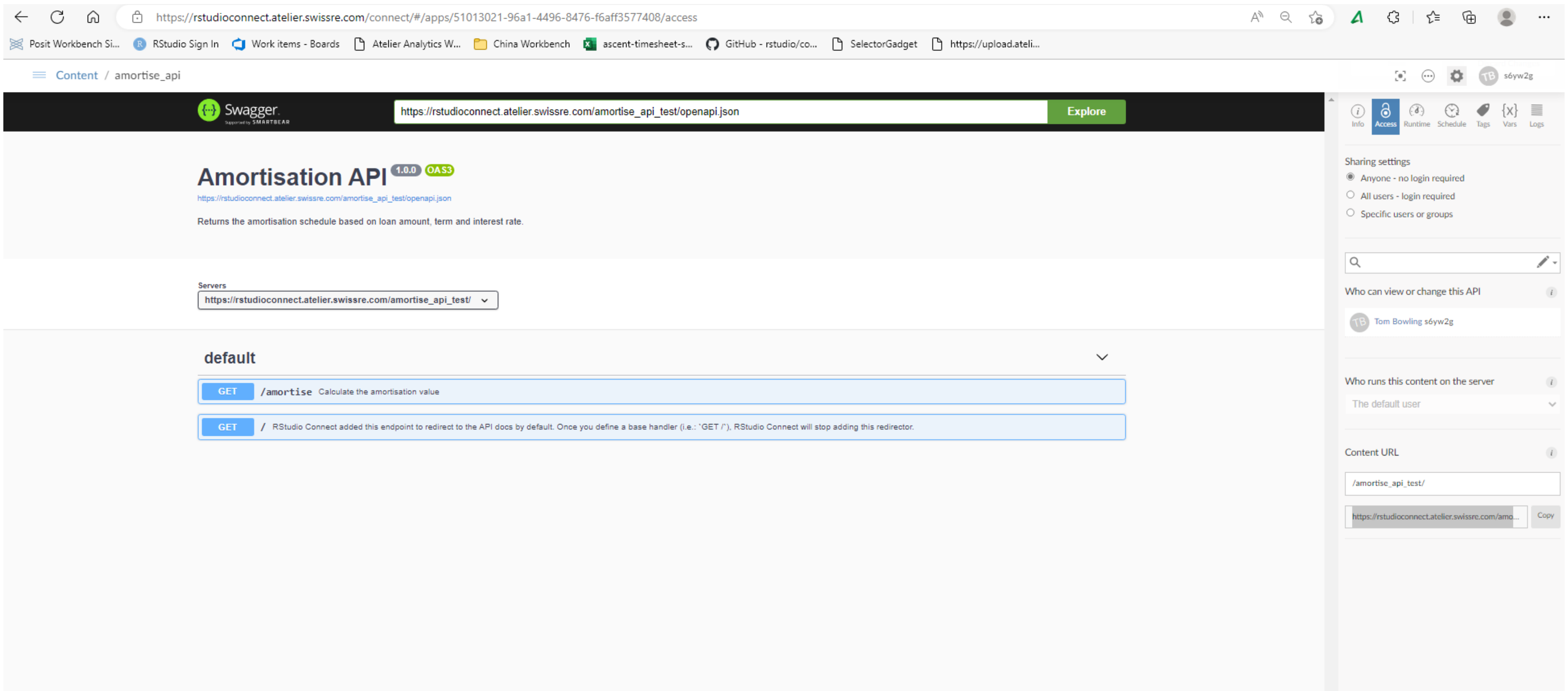


Sharing: Web API

As with the shiny app, we can publish our API to Rstudio Connect

Sharing: Web API

Once the content is published we can edit the access settings



Sharing: Web API

We can test the API from the terminal (i.e. not using R) like so

Console | Terminal | Deploy | Background Jobs | Workbench Jobs

Terminal 1 s6yw2g@scl000111696: /mnt/clustershare/home/s6yw2g/excel_to_r_emi_workshop

```
s6yw2g@scl000111696:/mnt/clustershare/home/s6yw2g/excel_to_r_emi_workshop$ curl -X GET "https://rstudioconnect.atelier.swissre.com/amortise_api_test/amortise?loan=1000&term=12&rate=0.05" -H  "accept: */*"
[1000,937.1746,871.2079,801.9429,729.2146,652.85,572.667,488.475,400.0733,307.2516,209.7887,107.4528]s6yw2g@scl000111696:/mnt/clustershare/home/s6yw2g/excel_to_r_emi_workshop$
```

Sharing: Web API

Link for demonstration only: not available outside Swiss Re

https://rstudioconnect.atelier.swissre.com/amortise_api_test/

Summary

In the first session we showed how we can take a process from Excel, move it in to R.

Today we have shown how we can

- structure our code as functions to abstract complexity away
- iterate using functionals to avoid writing explicit loops
- Package our code to improve robustness of our solution
- Demonstrated further ways we may productionalise our work via Web apps and Web APIs

Join the R Consortium

R Consortium Impact

- R Consortium Community **Grants** and Sponsorships Over **USD $1.4 Million**
- Organize large scale **collaborative projects**
    - R Validation Hub
    - R-Ladies
    - Diversity and Inclusion Working Group
- Co-host multidisciplinary **data science forums**
    - Stanford Data Institute
- Direct support for key **R events**
    - R/Medicine, R/Pharma, useR!, LatinR, more
- Direct support for **R User Groups**

Organizations Can Become a Member Today!

Email Joseph Rickert at **director@r-consortium.org** to set up first call