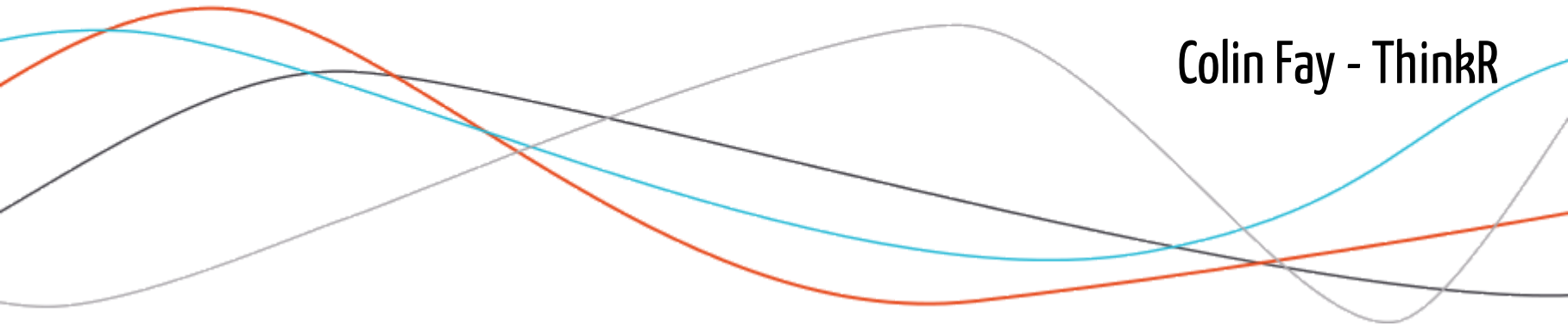
Hacking RStudio

useR! 2019

Colin Fay - ThinkR



What are we going to talk about today?

- 09h00 - 10h00: Addin & `{rstudioapi}`
- 10h00 - 10h30: Customising RStudio with CSS & Snippets

Coffee Break: 10h30 - 11h00

- 11h00 - 11h45: Building Templates
- 11h45 - 12h30: Connections

Tweet that!

- Hashtag: #useR2019
- @_ColinFay
- @thinkr_fr
- @UseR2019_Conf

Internet connexion:

- USER
- useR!2019

Find these slides

https://github.com/ColinFay/user2019workshop

$ whoami

Colin FAY

Data Scientist & R-Hacker at ThinkR, a french company focused on Data Science & R.

Hyperactive open source developer.

- https://thinkr.fr
- https://rtask.thinkr.fr
- https://colinfay.me
- https://twitter.com/_colinfay
- https://github.com/colinfay

ThinkR

Data Science engineering, focused on R.

- Training
- Software Engineering
- R in production
- Consulting

$whoarewe

Vincent Guyader

Codeur Fou, formateur et expert logiciel R

Diane Beldame

Dompteuse de ~~dragons~~ données, formatrice logiciel R

Colin Fay

Data scientist et R hacker

Sébastien Rochette

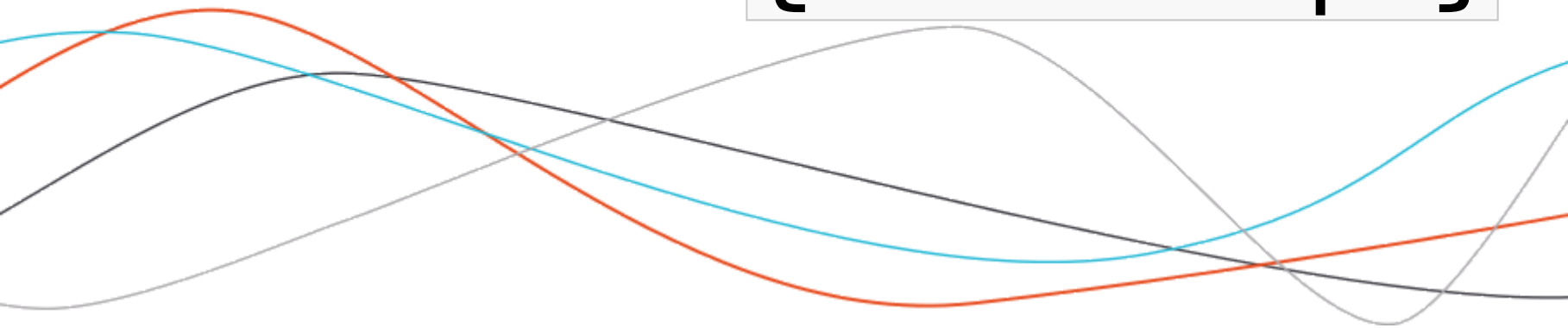
Modélisateur, Formateur R, Joueur de cartographies

Cervan Girard

Le nouveau

Hacking RStudio

Part 1 Addin & `{rstudioapi}`



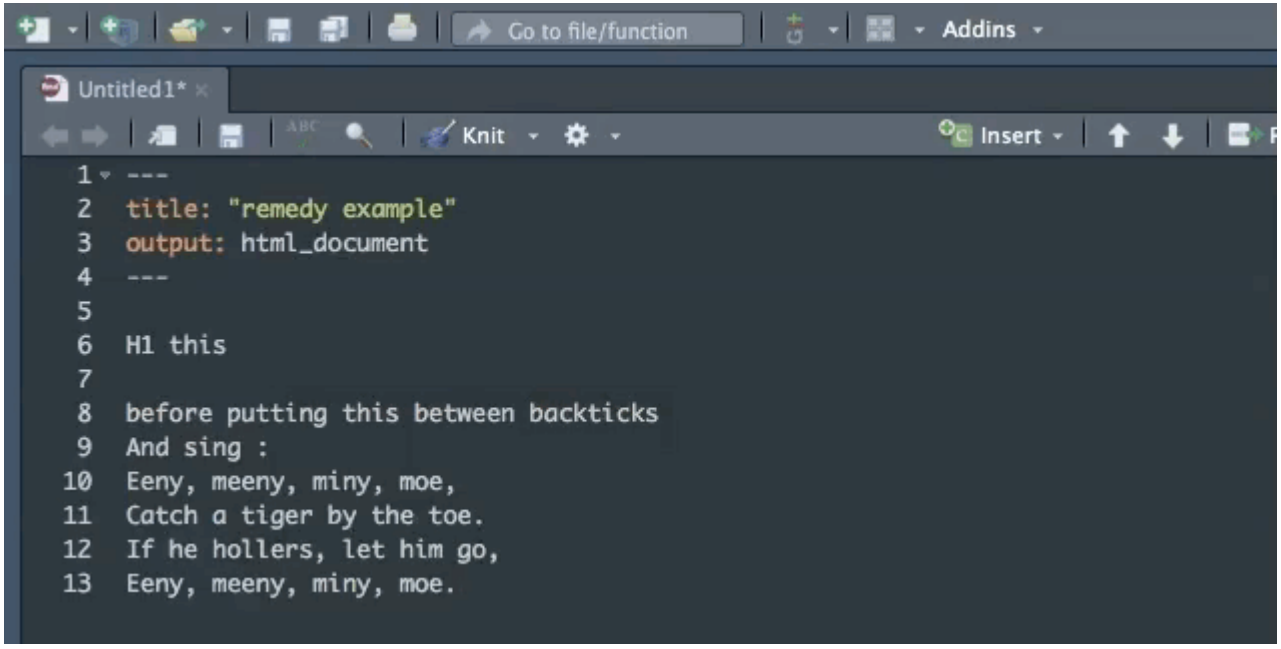
Why?

Improved workflow & Better user experience

Everything that can be (safely) automated should be automated

- Automate the boring stuff
- Avoid copying and pasting
- Create shortcuts for common behaviours
- Better user experience for a package

One example


Go to file/function
Addins
Untitled1*
Knit
Insert
1 ---
2 title: "remedy example"
3 output: html_document
4 ---
5
6 H1 this
7
8 before putting this between backticks
9 And sing :
10 Eeny, meeny, miny, moe,
11 Catch a tiger by the toe.
12 If he hollers, let him go,
13 Eeny, meeny, miny, moe.

One example

- `{remedy}` is a 📦 that tries to bring RMarkdown writing closer to a "word-processor experience"
- Mimics what you would find in things like Open Office (e.g: select a portion of text, and bold it with a keypress)
- This package uses the `{rstudioapi}` and RStudio addin template

Hacking RStudio

The `{rstudioapi}` 

{rstudioapi}

The {rstudioapi} 📦 is designed to manipulate RStudio (when available) through Command line. You can:

- Manipulate documents (edit, save, open...) and projects
- Generate dialog boxes
- Interact with RStudio terminals & the current R Session
- Launch jobs
- Open new tabs

{rstudioapi}

The versions used for this workshop is:

```
packageVersion("rstudioapi")
```

```
#> [1] '0.10'
```

```
rstudioapi::versionInfo()$version
```

```
[1] ‘1.2.1335’
```

Manipulate documents w/ `{rstudioapi}`

Creating elements

- `documentNew()` & `documentClose()`
- `initializeProject()`

Navigation

- `navigateToFile(path, line, column)`
- `openProject(path)`

Saving files

- `documentSave()`, `documentSaveAll()`

Manipulate documents w/ `{rstudioapi}`

- `document_range()` & `modifyRange()`

A range is a set of `document_position` in an RStudio document.

A position is defined by a row and a column in a document. A range is two positions, one for the beginning, one for the end of the range.

- `insertText(range, text, id)` & `setDocumentContents(text, id = NULL)`

`insertText()` adds a text at a specific range inside a given document (passed to the `id` argument. If `id` is NULL, the content will be passed to the currently open or last focused document).

`setDocumentContents()` takes a text and the id of a document, and set the content of this doc to `text`.

> If range == `Inf`, the content is added at the end of the document.

Manipulate documents w/ {rstudioapi}

```
enclose <- function(prefix, postfix = prefix) {
  # Get the context of the Editor
  a <- rstudioapi::getSourceEditorContext()
  # a$selection is a list refering to the selected text
  for (s in a$selection) {
    rstudioapi::insertText(
      location = s$range,
      text = sprintf(
        "%s%s%s",
        prefix,
        s$text,
        postfix
      )
    )
  }
}
italicsr <- function() enclose("_")
```

> https://github.com/ThinkR-open/remedy

Access RStudio interface elements

- `getActiveDocumentContext()`

```
# get console editor id
context <- rstudioapi::getActiveDocumentContext()
id <- context$id
id
```

```
#> [1] "45E5023C"
```

- `getActiveProject()`
- `getConsoleEditorContext()`
- `getSourceEditorContext()`

Manipulate R session(s) w/ `{rstudioapi}`

- `restartSession()`

This function restarts the current R Process.

- `sendToConsole(code, execute, echo, focus)`

```
sendToConsole("library('golem')")
```

Dialogs w/ {rstudioapi}

- `selectFile()` & `selectDirectory()`
- `askForPassword()` & `askForSecret()`
- `showDialog()`, `showPrompt()` & `showQuestion()`

```
con <- DBI::dbConnect(RMySQL::MySQL(),
  host = "mydb",
  user = "colin",
  password = rstudioapi::askForPassword("password")
)
```

> https://db.rstudio.com/dplyr/

Dialogs w/ {rstudioapi}

```
file_info <- function(){
  path <- selectFile()
  file.info(path)
}
```

```
completed <- function(...){
  res <- force(...)
  showDialog("Done !", "Code has completed")
  return(res)
}
```

Playing with the terminals

- `terminalCreate()` & `terminalActivate()`
- `terminalExecute()`
- `terminalList()`
- `terminalVisible()`, `terminalBusy()` & `terminalRunning()`
- `terminalExitCode(termId)`
- `terminalBuffer(termId)`
- `terminalKill()`

Jobs

- `jobRunScript()`

sourceMarkers

Allow to display a custom `sourceMarkers` pane.

```
rstudioapi::sourceMarkers(
  "export",
  df
)
```

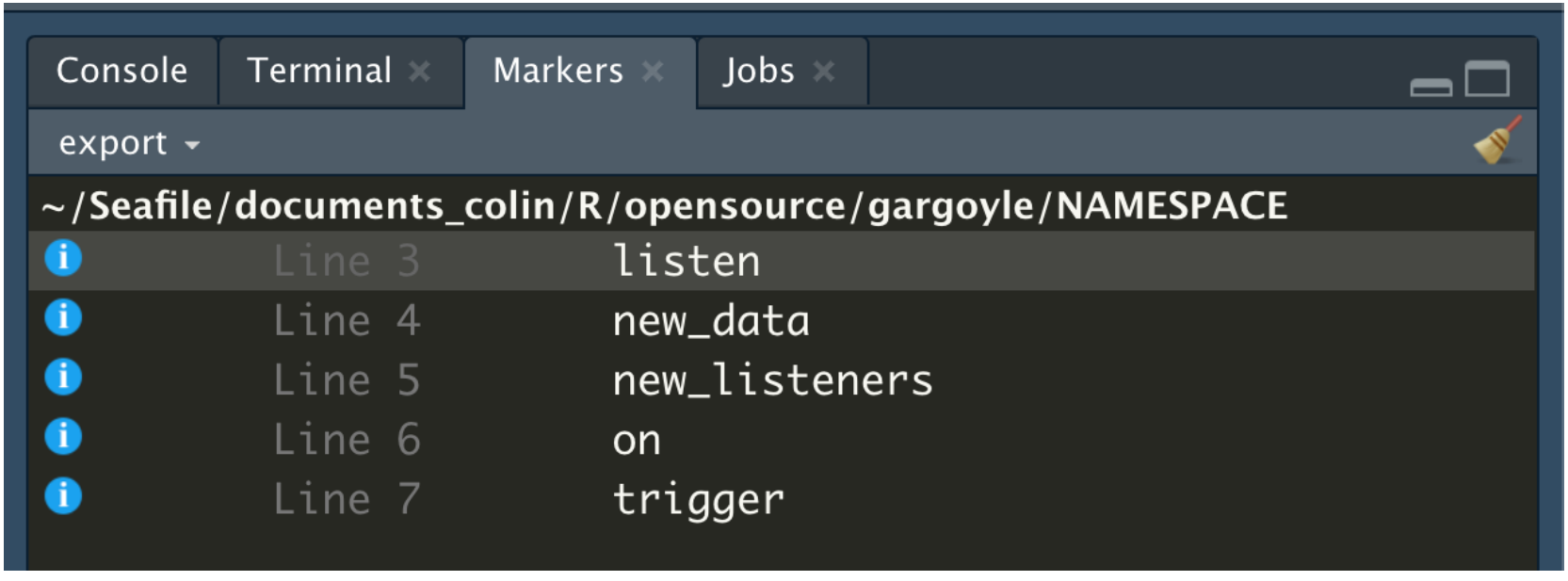
`df` must have:

1. `type` Marker type ("error", "warning", "info", "style", or "usage")
2. `file` Path to source file
3. `line` Line number witin source file
4. `column` Column number within line
5. `message` Short descriptive message

sourceMarkers

```
a <- readLines("NAMESPACE")
l <- grepl("export", a)

df <- data.frame(
  type = "info",
  file = "NAMESPACE",
  line = which(l),
  column = 1,
  message = gsub(".*\\((.*)\\)", "\\1", a[l]),
  stringsAsFactors = FALSE
)
rstudioapi::sourceMarkers("export",  df)
```

sourceMarkers



`{rstudioapi}` dev pattern

=> Before running any `{rstudioapi}`-based function, check if RStudio is available.

- `rstudioapi::isAvailable()` (returns a Boolean)

You can even check for a specific version:

- `rstudioapi::isAvailable(version_needed = "0.1.0")`

There is also `rstudioapi::verifyAvailable()`, which returns an error message if RStudio is not running (instead of a boolean).

```
rstudioapi::isAvailable()
```

```
#> [1] TRUE
```

{rstudioapi} dev pattern

=> Check if a function is available in the {rstudioapi} 📦

- rstudioapi::hasFun()

As {rstudioapi} relies on internal RStudio functions, the availability of {rstudioapi} is linked to the user version of RStudio.

For example, the askForPassword() function was added in version 0.99.853 of RStudio.

This function allows to run function only if they are available:

```
if (rstudioapi::hasFun("askForPassword")){
  rstudioapi::askForPassword()
}
```

Addins

- Execute R functions interactively from the IDE
- Can be used through the drop down menu
- Can be mapped to keyboard shortcuts

Two types

1. Text macros
2. Shiny Gadgets

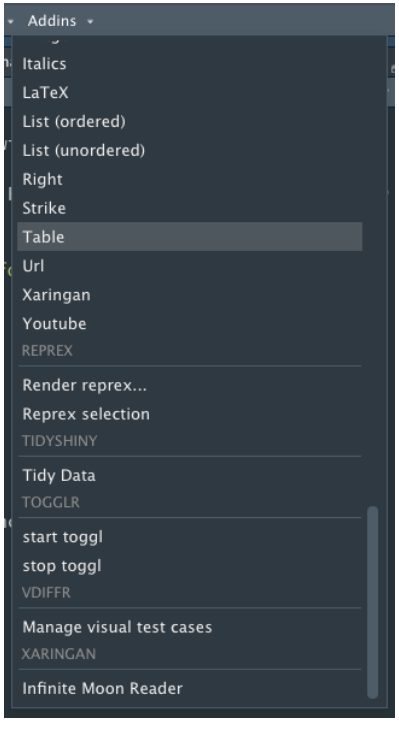
Addin examples

- `{datapasta}`: Reducing resistance associated with copying and pasting data to and from R
- `{giphyr}`: An R package for giphy API
- `{colourpicker}`: A colour picker tool for Shiny and for selecting colours in plots (in R)
- `{styler}`: Non-invasive pretty printing of R code
- `{esquisse}`: RStudio add-in to make plots with ggplot2
- `{todor}`: RStudio add-in for finding TODO, FIXME, CHANGED etc. comments in your code.

> See https://github.com/daattali/addinslist for more

Create an addin

- An addin is a package, so create a package
- Create the function that you want to be launched
- Run `usethis::use_addin()`
- Complete the `addins.dcf`
- Install the package
- And tadaa 🥳



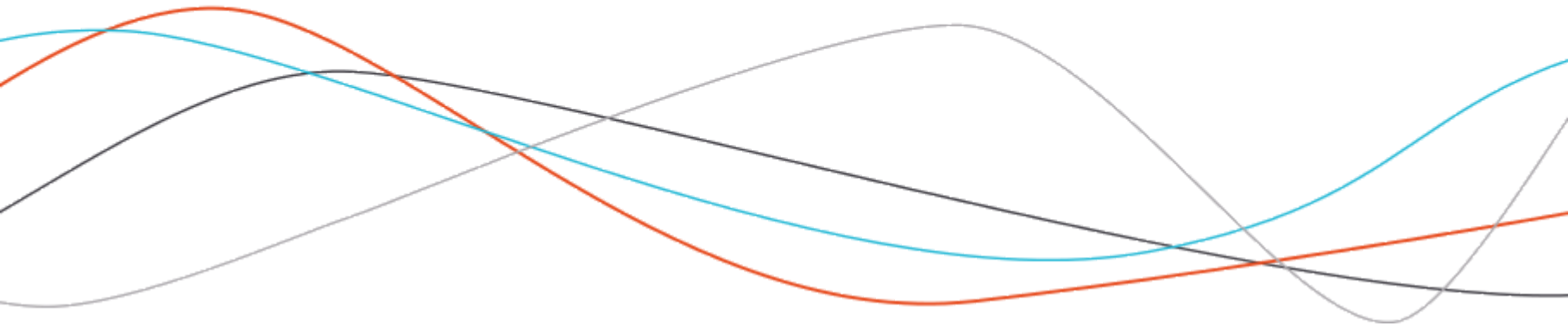
Create an addin

`addins.dcf`

```
Name: New Addin Name
Description: New Addin Description
Binding: new_addin
Interactive: false
```

- Name of the addin
- Its description
- The function to bind to the addin
- Is the addin interactive (i.e does it launch a Shiny app)?

Let's practice !



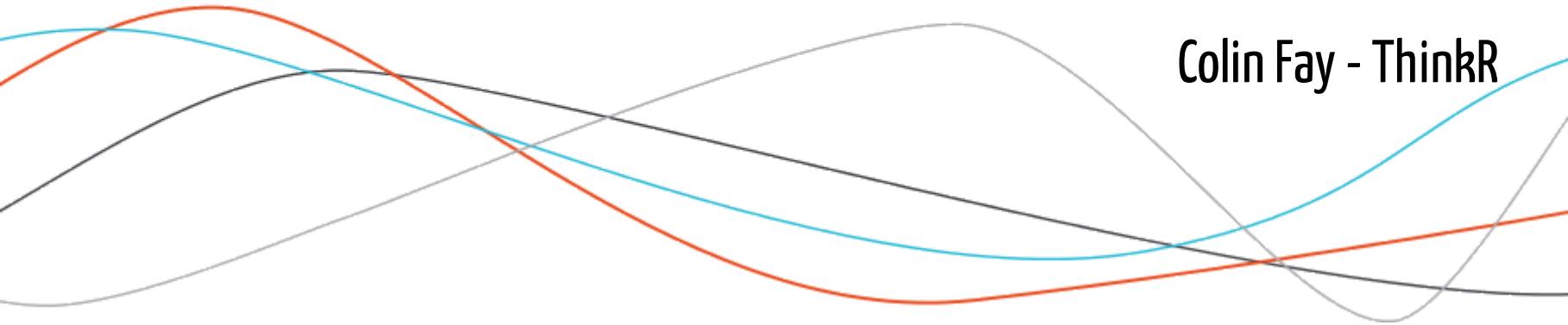
Now it's your turn to create an addin

Pick an idea (or choose your own)

- Takes a selected word, and look for it on Wikipedia.
- Inserts a random cat picture in a markdown.
- Takes a selected word, and allows to turn to lower & uppercase.
- Opens a dialog that take a password, and looks if this password is anywhere in the current project. Optional: opens a sourceMarkers pane with the results.
- Takes a selected function, and add it into a `R/` folder.

Hacking RStudio

Part 2: themes & snippets

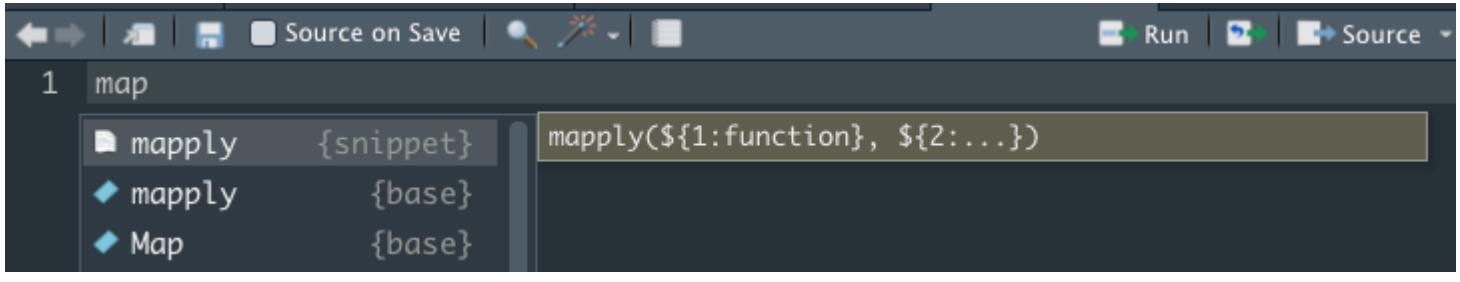


Colin Fay - ThinkR

Snippets

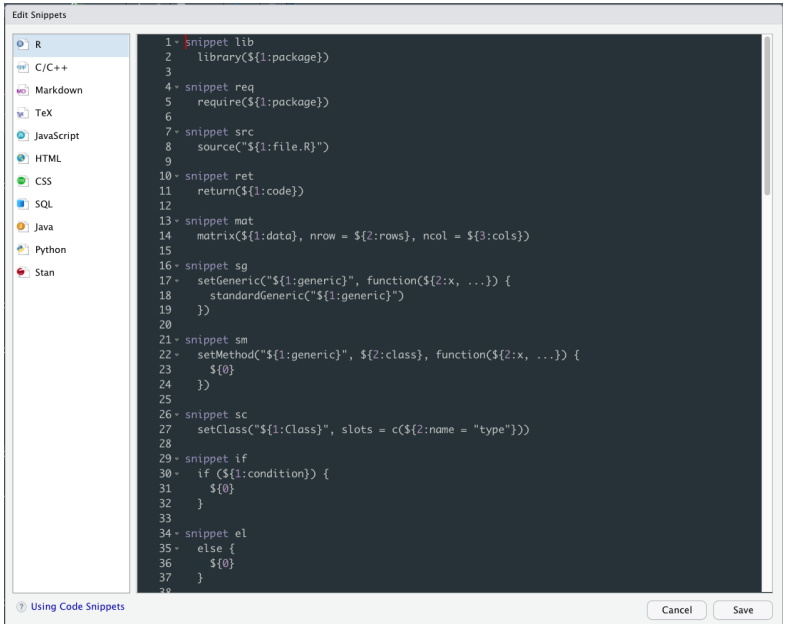
Snippets are text macros that can be used to quickly insert text into RStudio.

Can be written for R, C/C++, Java, Python, SQL, JavaScript, HTML and CSS.



Snippets

You can define them in Global Options > Code > Edit Snippets



Snippets

```
snippet switch
    switch (${1:object},
        ${2:case} = ${3:action}
    )
```

- Elements in mustachs will be suggested for autocompletion in order
- Users can switch from one element to another with tab

Snippets

Snippets can execute code:

```
83    rapply(${1:list}, ${2:function})
84
85  snippet ts
86    `r paste("#", date(), "------------------------------\n")`
87
88  snippet shinyapp
```

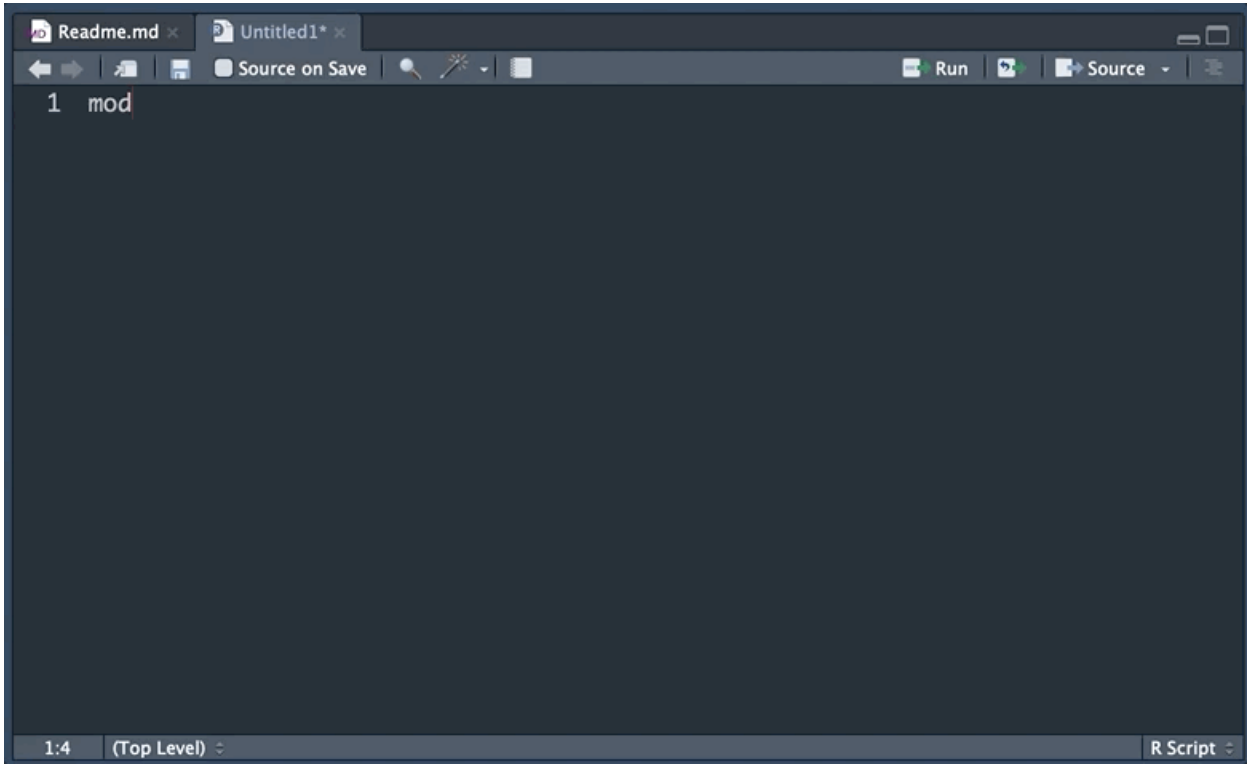
Example

`{shinysnippets}` (https://github.com/ThinkR-open/shinysnippets)

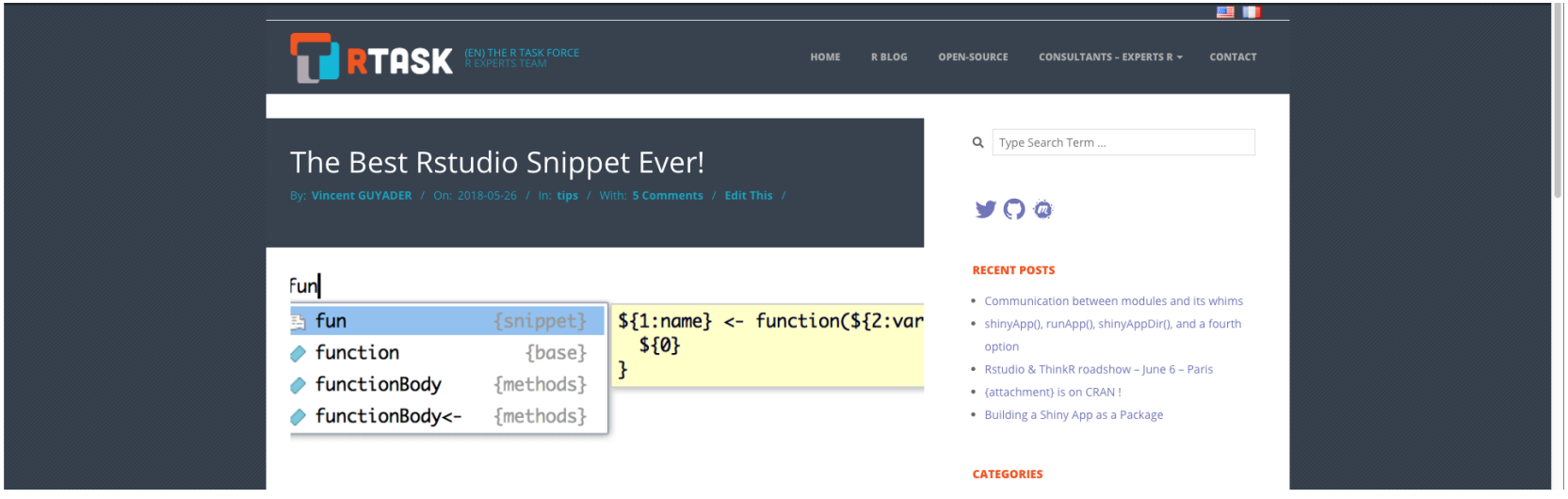
```
snippet module
    ${1:name}ui <- function(id){
        ns <- NS(id)
        tagList(

            )
        }

    ${1:name} <- function(input, output, session){
        ns <- session\$ns
    }

    # Copy in UI
    ${1:name}ui("${1:name}ui")

    # Copy in server
    callModule(${1:name}, "${1:name}ui")
```

Example

`{shinysnippets}` (https://github.com/ThinkR-open/shinysnippets)

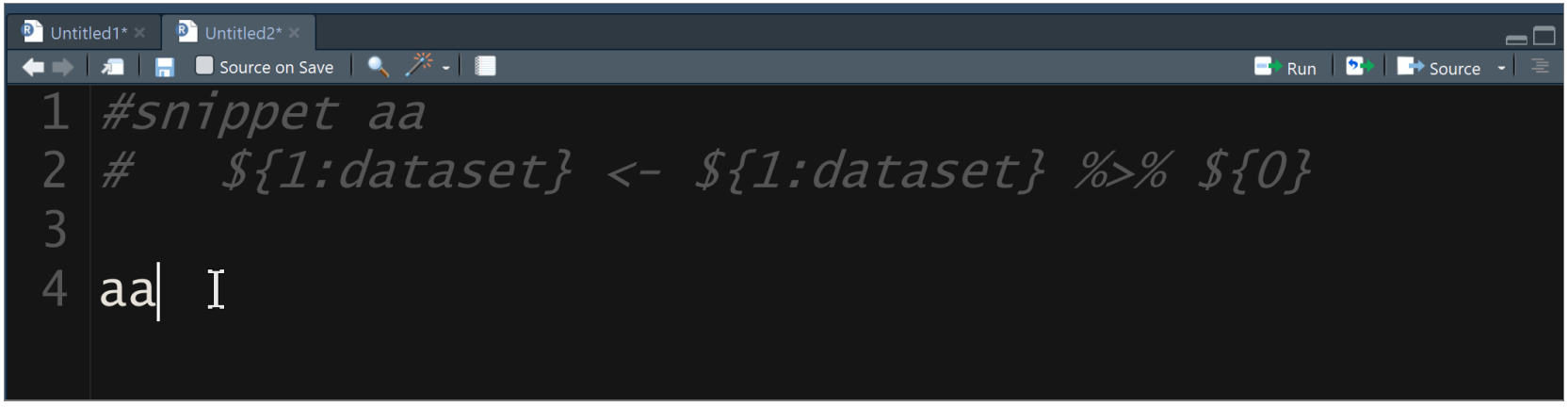


Example



> https://rtask.thinkr.fr/blog/the-best-rstudio-snippet-ever/

Example




```
snippet aa
    ${1:dataset} <- ${1:dataset} %>% ${0}
```

Snippets: another scenario

Snippets don't have to be "templates", but can be code you use a lot.

```
snippet dbi
    library(DBI)
    con <- DBI::dbConnect(RMySQL::MySQL(),
    host = "mydb",
    user = "colin",
    password = rstudioapi::askForPassword("password")
    )
```

Inject code in snippets

(RStudio >= 0.99.706)

system.r.snippets 184 Bytes

Edit | Web IDE

```
# On Shift+Tab, executes the text inserted after $$ via system
# The result is then pasted at cursor
# Use with care!
snippet $$
        `r eval(parse(text = "system('$$', intern = TRUE)"))`
```

> https://jozef.io/r906-rstudio-snippets/

RStudio themes

- Started with RStudio v1.2
- tmTheme or rstheme

tmTheme

- XML based
- introduced by the text editor TextMate
- Can be created with a tmTheme editor

rstheme

- CSS based
- Works specifically with RStudio

A web approach

=> On the web

- Find a theme at : https://tmtheme-editor.herokuapp.com
- Change the theme online
- Add the theme `rstudioapi::addTheme("Monokai.tmTheme")`

Do it all manually

```
file.create("mytheme.rstheme")
rstudioapi::navigateToFile("mytheme.rstheme")
rstudioapi::addTheme("mytheme.rstheme")
```

A pragmatic approach

=> On the web

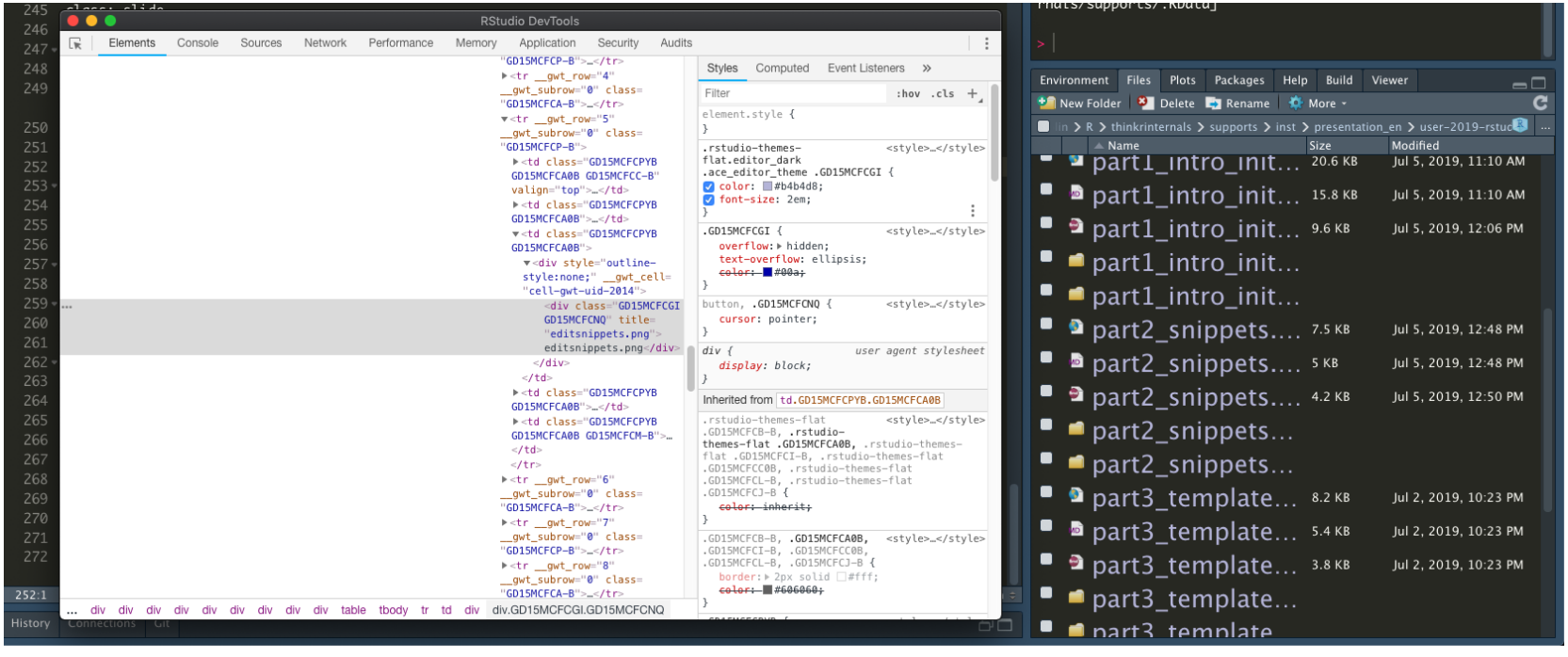
- Find a theme at : https://tmtheme-editor.herokuapp.com and download it
- Add the theme & modify it manually:

```
rstudioapi::addTheme("Monokai.tmTheme")

rstudioapi::navigateToFile("~/.R/rstudio/themes/Monokai.rstheme")
```

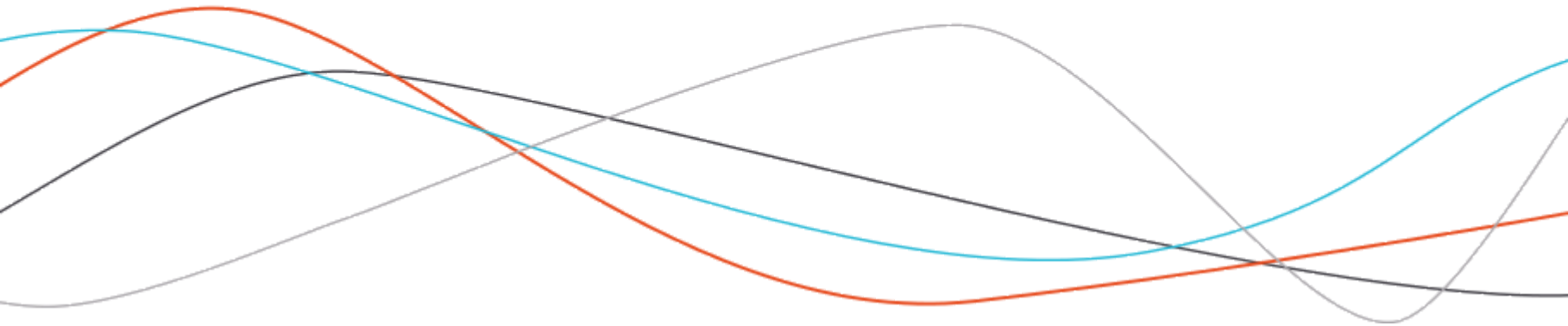
Some elements you can change

- `.ace_comment`, `.ace_constant`, `.ace_keyword`, `.ace_string`: color of the comment, constant, keyword, and string in the theme.
- `ace_content`: main frames of RStudio editors

And any other CSS tricks... because RStudio is a big webpage!



Let's practice !



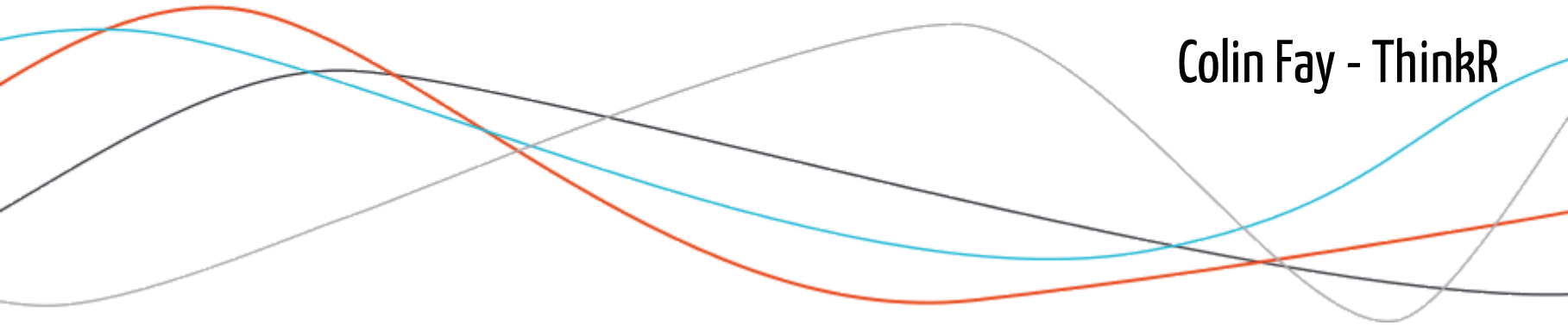
Now it's your turn to create a snippet / theme

Pick an idea (or choose your own):

- Create a FALSE snippet that inserts TRUE and vice-versa
- Create a ggplot skeleton snippet
- Add a snippet that inserts a list of `library()`
- Create a snippet that add a random `{cowsay}` to your script
- Add a background image to RStudio
- Change some colors from RStudio

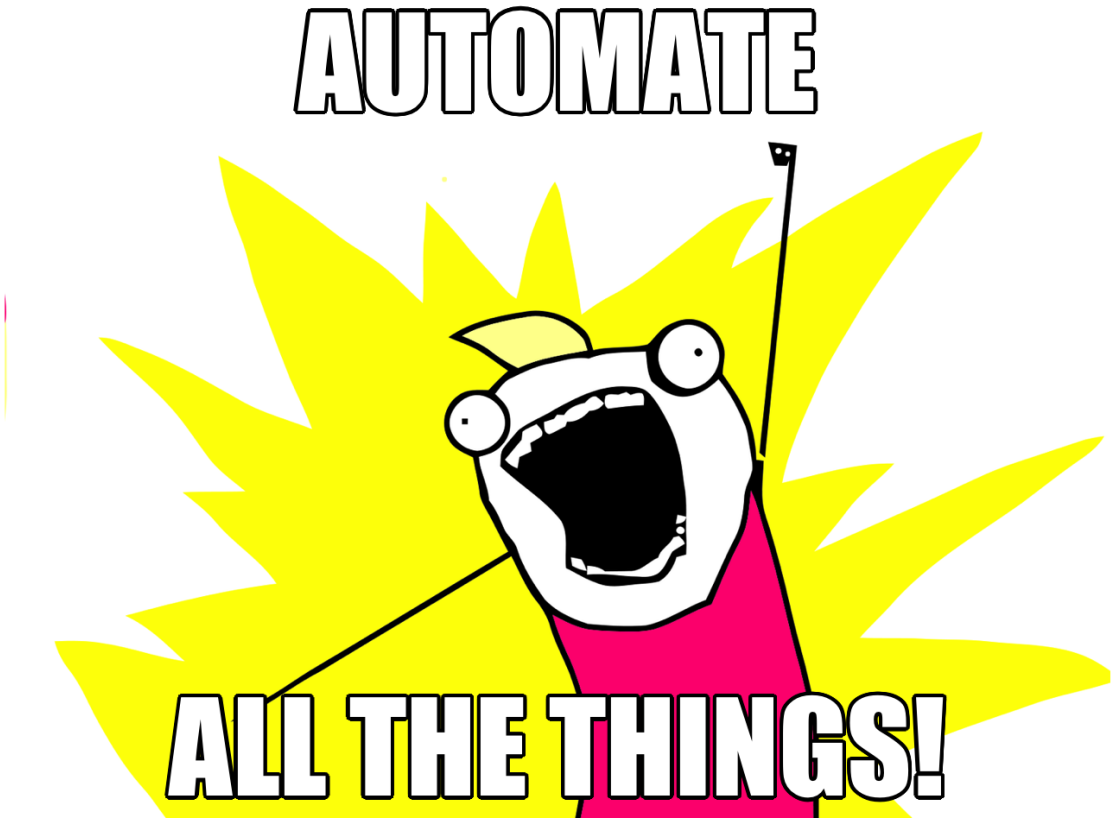
Hacking RStudio

Part 3: templates



Colin Fay - ThinkR

Templates



Templates - Why ?

- Automation of common markdown formats
- Sharing templates

> For example, ThinkR has an internal package to produce slides (theses slides are generated through a customized `{xaringan}` format)

Examples

- `{pagedown}` - Paginate the HTML Output of R Markdown with CSS for Print
- `{xaringan}` - Presentation Ninja 幻灯忍者 · 写轮眼
- `{rticles}` - LaTeX Journal Article Templates for R Markdown
- `{tufte}` - Tufte Styles for R Markdown Documents

Markdown template

- Create a package
- `usethis::use_rmarkdown_template("mymarkdown")`
- update `inst/rmarkdown/templates/mymarkdown/template.yaml` & `inst/rmarkdown/templates/mymarkdown/skeleton/skeleton.Rmd`
- Install the package
- Find your template File > New File > RMarkdown > From Template

template.yaml

```
name: mymarkdown
description: >
   A description of the template
create_dir: FALSE
```

skeleton.Rmd

````
---
title: "Template Title"
author: "Your Name"
date: "The Date"
output: output_format
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## Adding an RMarkdown Template

This file is what a user will see when they select your template. Make sure
that you update the fields in the yaml header. In particular you will want
to update the `output` field to whatever format your template requires.

This is a good place to demonstrate special features that your template
provides. Ideally it should knit out-of-the-box, or at least contain clear
instructions as to what needs changing.

Finally, be sure to remove this message!
````

Include external files along the Rmd

- Add files to the `skeleton` folder
- Turn `create_dir` to `TRUE` in the `yaml`

Project Templates

- Create a package
- Create a function to launch on project creation
- Define template metadata
- Input Widgets

Project Templates

```
create_golem <- function(path, ...) {

  dir.create(path, recursive = TRUE, showWarnings = FALSE)

  ll <- list.files(
    path = golem_sys("shinyexample"),
    full.names = TRUE, all.files = TRUE, no.. = TRUE
  )

  file.copy(
    from = ll,
    to = path,
    overwrite = TRUE,
    recursive = TRUE
  )

}
```

Project Templates

`inst/rstudio/templates/project/create_golem.dcf`

```
Binding: create_golem_gui
Title: Package for Shiny App using golem
OpenFiles: dev/01_start.R
Icon: golem.png
```

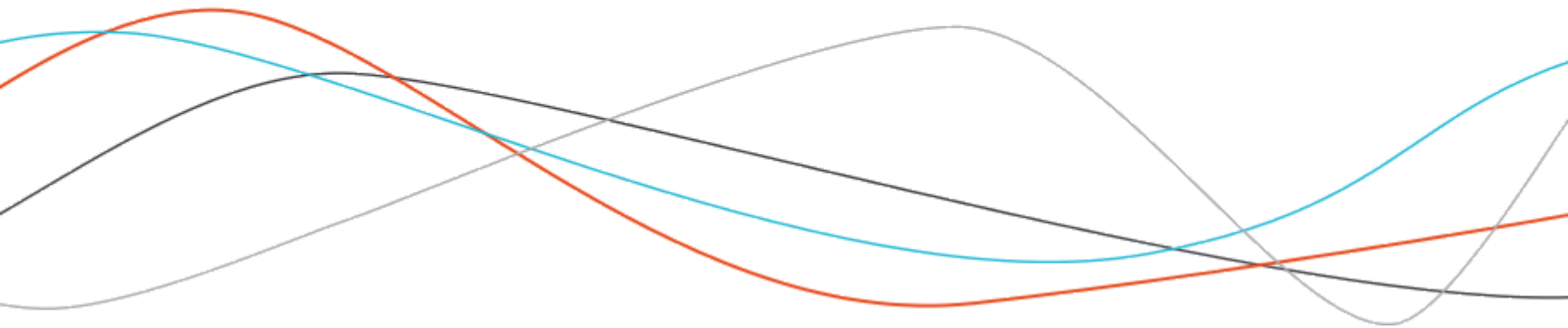
- Binding: the function to run on project creation
- Title: title of the template in the Gadget
- OpenFiles: which file to open at launch
- Icon: icon

Project Templates - input widgets

```
Parameter: check
Widget: CheckboxInput
Label: Checkbox Input
Default: On
Position: left
```

- Parameter: The name of the parameter that will be passed to the function that creates the project.
- Widget: The type of the widget (`CheckboxInput` / `SelectInput` / `TextInput` / `FileInput`)
- Label: label to display
- Default: Default value of the element
- Position: where to put this element in the widget

Let's practice !



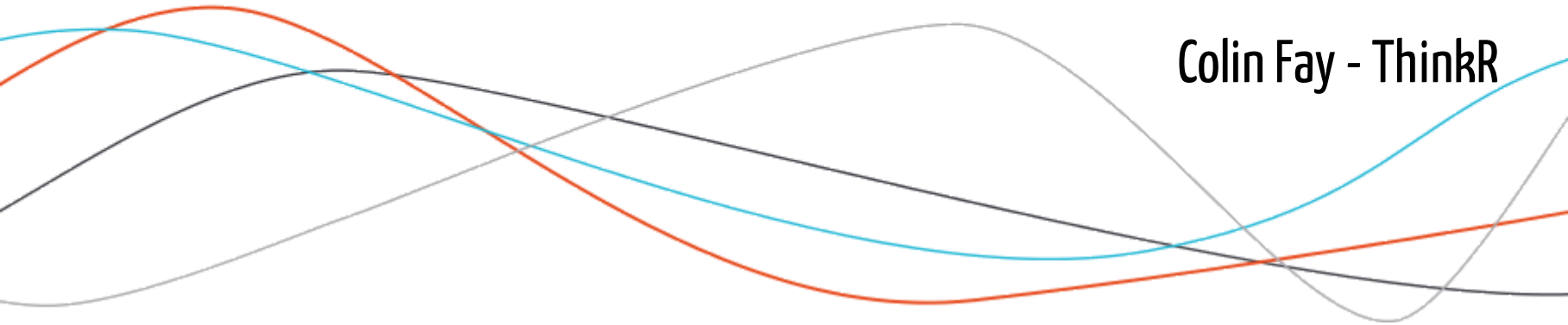
Now it's your turn to create an addin

Pick an idea (or choose your own)

- Create a template for a data analysis (markdown or project)
- Create a Markdown template with a custom CSS
- Create a template to connect to a database
- Create a template for launching a twitter data scraping

Hacking RStudio

Part 4: Connections



Colin Fay - ThinkR

What are RStudio Connections?

Connections are a way to extend RStudio connection features, so that you can connect to external data sources easily.

Connections can be handled through:

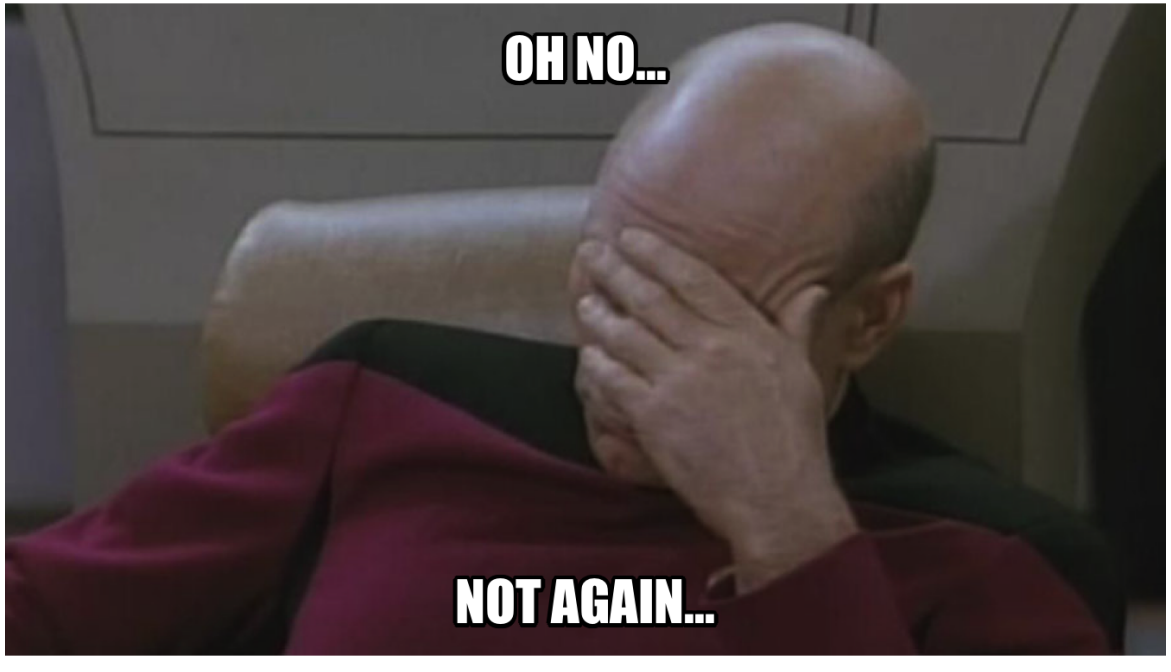
- Snippet Files, which are read with the Connection Pane
- Packages that can implement either Snippet Files or Shiny Applications
- Package can use Connection Contracts

RStudio Connections example

- `{sparklyr}` - R interface for Apache Spark
- `{odbc}` - Connect to ODBC databases (using the DBI interface)
- `{neo4r}` - A Modern and Flexible Neo4J Driver
- `{fryingpane}` - Serve datasets from a package inside the RStudio Connection Pane.

RStudio Connections

Step 1... Create a package!



Creating a snippet

```
dir.create(path = "inst/rstudio/connections", recursive = TRUE)
file.create(path = "inst/rstudio/connections.dcf")
file.create(path = "inst/rstudio/connections/snippet.R")
rstudioapi::navigateToFile("inst/rstudio/connections.dcf")
rstudioapi::navigateToFile("inst/rstudio/connections/snippet.R")
```

`connections.dcf`

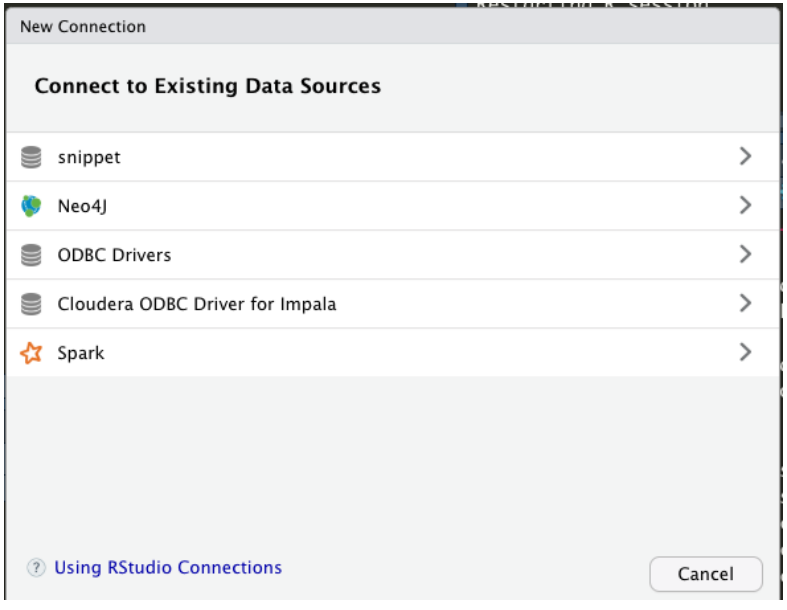
A `dcf` containing a series of `Name` refering to the `.R` files in the `connections/` folder.

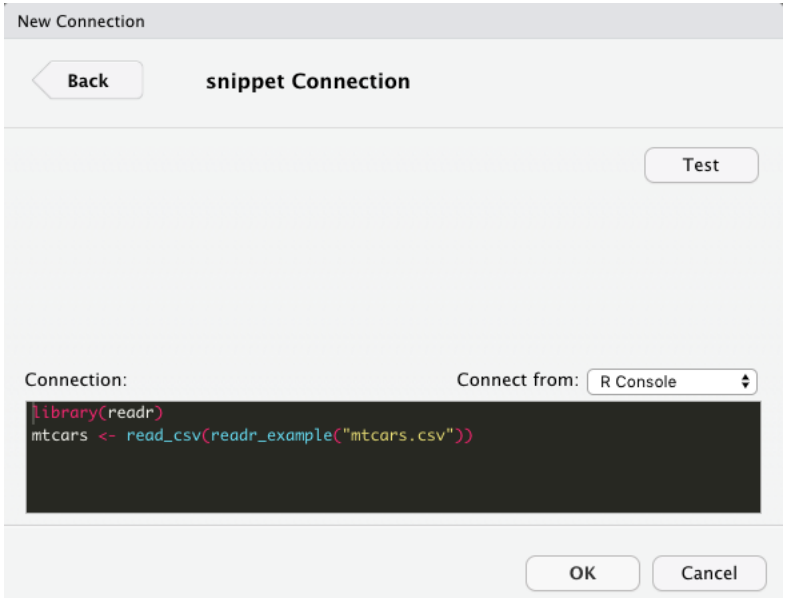
```
Name: snippet
```

snippet.R

```
library(readr)
mtcars <- read_csv(readr_example("mtcars.csv"))
```

Install and Restart RStudio





connections.dcf

```
Name: snippet

Name: more_snippet
```

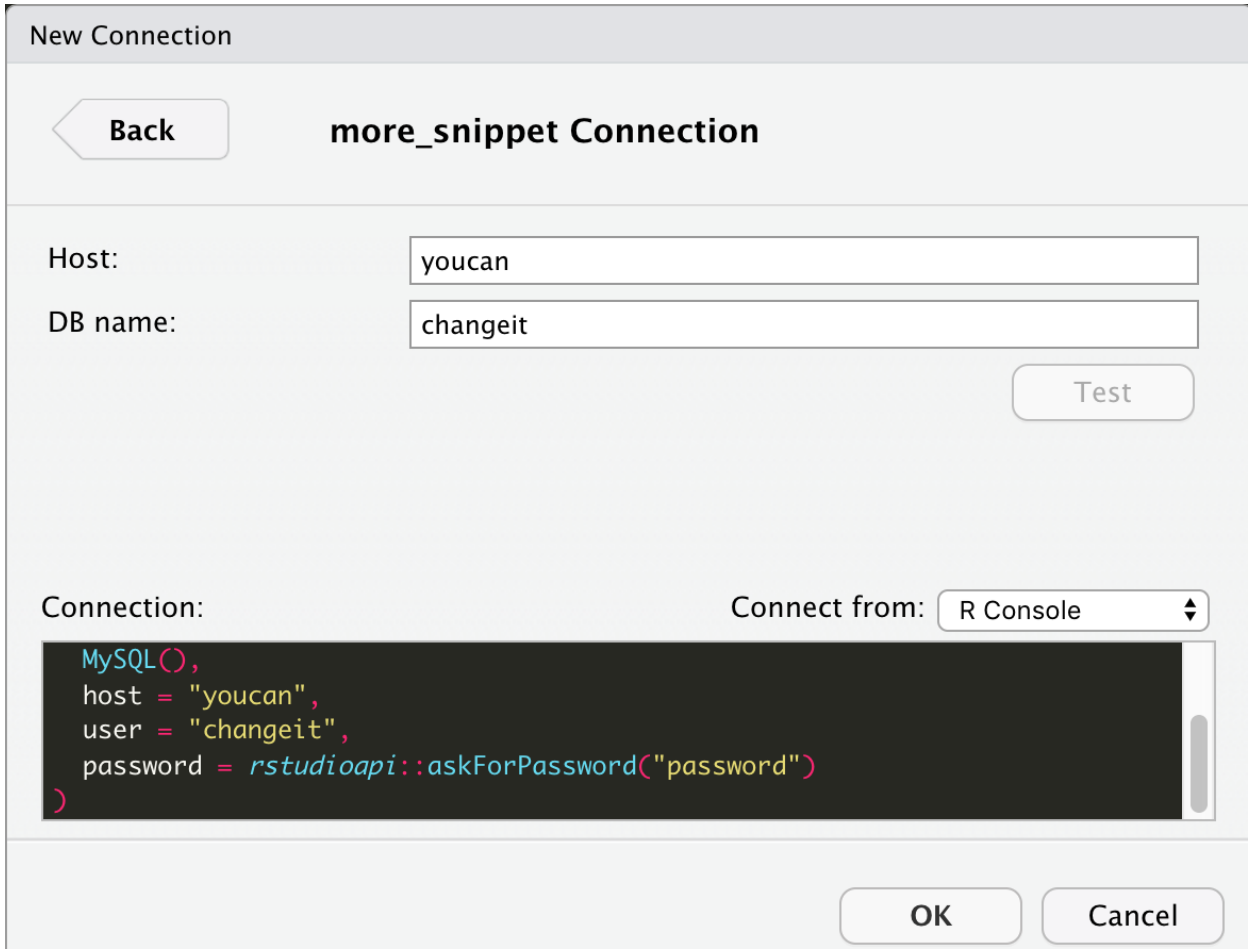
```r
file.create(path = "inst/rstudio/connections/more_snippet.R")
rstudioapi::navigateToFile("inst/rstudio/connections/more_snippet.R")
```

more_snippet.R

> Can pass parameters with `${Position:Label=Default}`

```
library(DBI)
library(RMySQL)
con <- dbConnect(
MySQL(),
host = ${0:Host="mydb"},
user = ${1:DB name="colin"},
password = rstudioapi::askForPassword("password")
)
```

Install and Restart RStudio



Connection contract

Connection contracts are a way to interact with data sources and display an interface in the Connection Pane.

The Connection Pane is handled with the `connectionObserver` from RStudio.

You can get it with:

```
observer <- getOption("connectionObserver")
```

The observer has three methods, to be called on creation, update, and closing of the Connection with the data source.

- `connectionOpened()`
- `connectionClosed()`
- `connectionUpdated()`

`connectionOpened()`

This method takes several arguments:

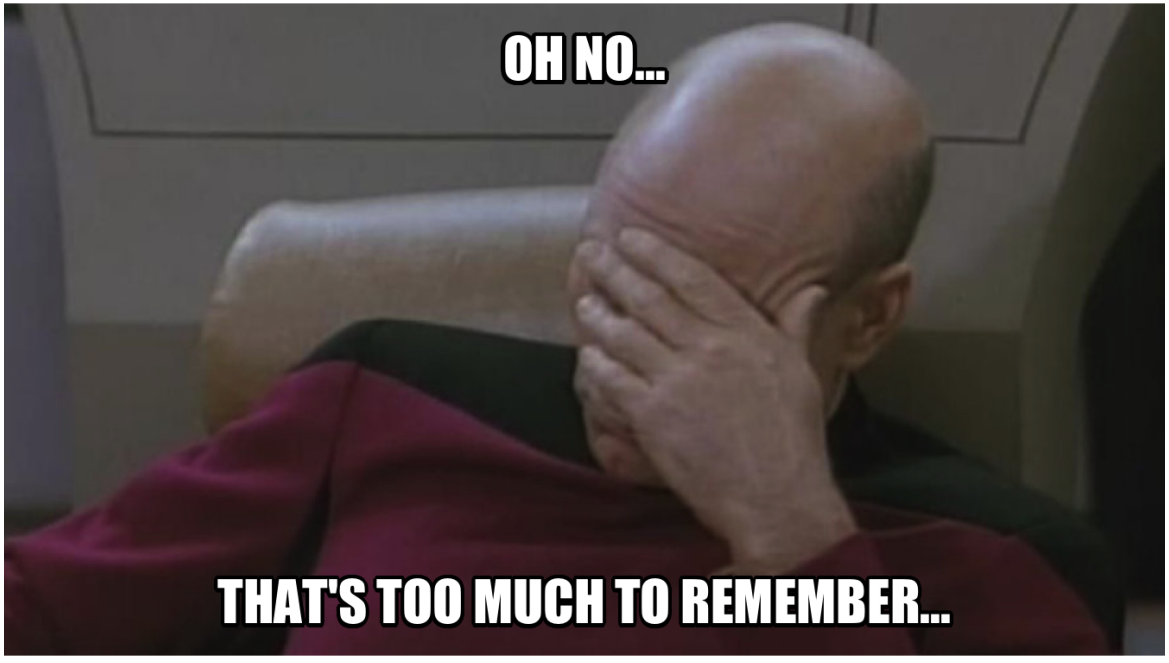
- `type` & `displayName`, texts describing the connection
- `host`, the address of the host we are connected to
- `icon`, the path to an icon for the connection pane
- `connectCode`, a snippet of R code to be reused in the connection Pane
- `disconnect`, a *function* used to close the connection from the connection Pane
- `listObjectTypes`, a *function* that returns a nested list from the connection
- `listObjects`, a *function* that list top level objects from the data source when called without arg, or the object to retreive. Returned object is a dataframe with `name` and `type` column
- `listColumns`, a *function* listing the columns of a data object. Returned object is a dataframe with `name` and `type` column

`connectionOpened()`

This method takes several arguments:

- `previewObject`, a *function* accepting row limit and an object, returns the specified number of rows from the object
- `actions`, a list of things to add to the connection pane as buttons. Lists with `icon` and `callback`
- `connectionObject`, the raw connection object

Too much?



Step 1: define a `on_connection_opened` fun

```
on_connection_opened <- function(pkg_name = "pkg") {
  data_list  <- data(package = pkg_name)
  data_results <- as.data.frame(data_list$results)
  observer <- getOption("connectionObserver")
  if(!is.null(observer)){
    observer$connectionOpened()
  }
}
```

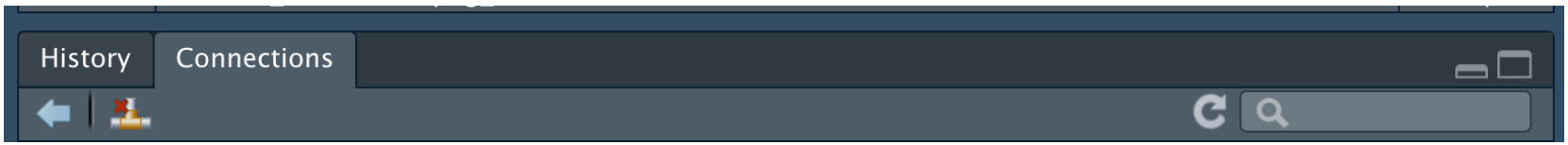
Step 2: define metadata

```
observer$connectionOpened(
  type = "Data sets",
  host = pkg_name,
  displayName = pkg_name,
  icon = system.file("img","package.png", package = "fryingpane"),
  connectCode = glue('library(fryingpane)\ncook("{pkg_name}")')
)
```

Step 3 function to call on connection closed

```
observer$connectionOpened(
  disconnect = function() {
    close_connection(pkg_name)
  }
)
```

```
close_connection <- function(pkg_name) {
  # Deconnection logic
  print("Connection closed")
}
```



Step 4 List object types

```
observer$connectionOpened(
  listObjectTypes = function() {
    list_objects_types()
  }
)
```

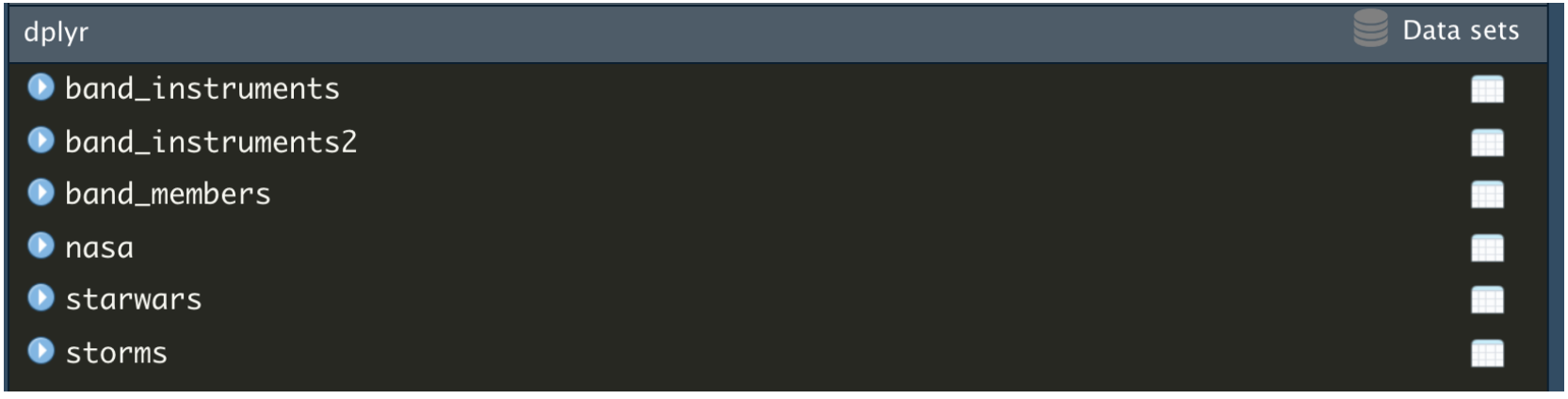
```
list_objects_types <- function() {
  return(
    list(
      table = list(contains = "data")
    )
  )
}
```

Step 5 How are object displayed?

```
observer$connectionOpened(
  listObjects = function( type = "table") {
    list_objects(
      includeType = TRUE,
      data_names = as.character(data_results$Item),
      data_type = "dataset"
    )
  }
)
```

Step 5 How are object displayed?

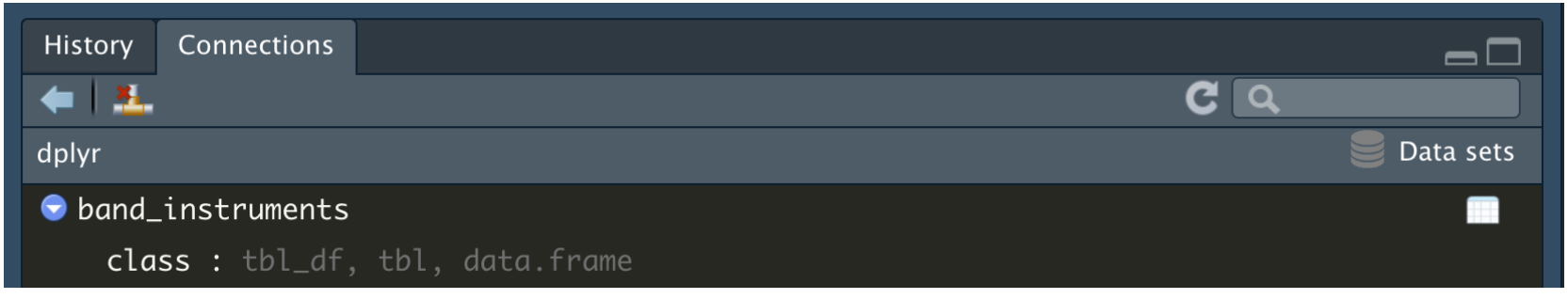
```
list_objects <- function(includeType, data_names, data_type) {
  if (includeType) {
    data.frame(
      name = data_names,
      type = rep_len("table", length(data_names)),
      stringsAsFactors = FALSE
    )
  }
}
```



Step 6 List Colums

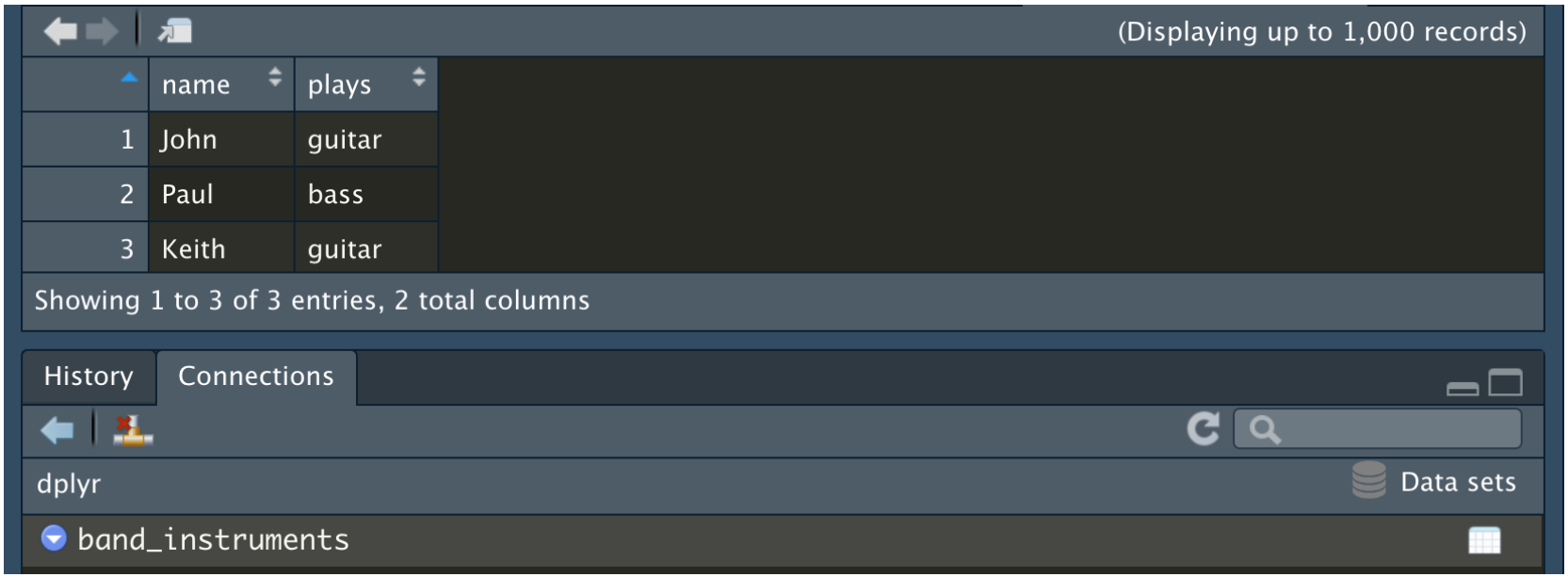
```
observer$connectionOpened(
  listColumns = function(table) {
    list_columns(table, pkg_name = pkg_name)
  }
)
```

```
list_columns <- function(table, pkg_name) {
  res <- get(data(list = table, package = pkg_name))
  tibble(
    name = "class",
    type = paste(class(res), collapse = ", "))
}
```



Step 7 Preview objects

```
observer$connectionOpened(
  previewObject = function(rowLimit, table) {
    preview_object(pkg_name = pkg_name, table, rowLimit)
  }
)
```



Step 8 Actions

```
observer$connectionOpened(
  actions = pkg_actions(pkg_name)
)
```

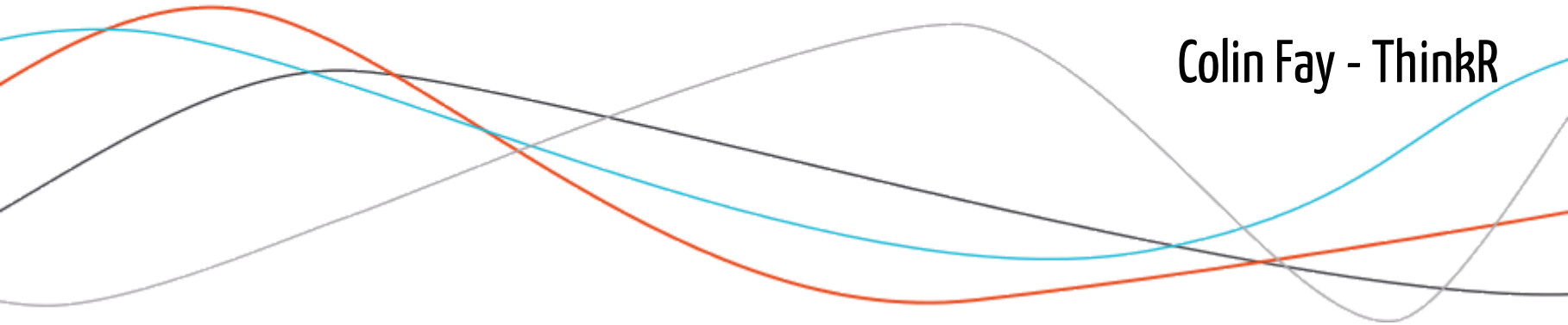
```
pkg_actions <- function(pkg_name){
  list(
    help = list(
      icon = system.file("icons","github.png", package = "neo4r"),
      callback = function() {
        help(pkg_name, try.all.packages = TRUE, help_type = "text")
      }
    )
  )
}
```

Finally, wrap this in a function !

```
cook <- function(pkg_name){
  test_if_exists(pkg_name)
  on_connection_opened(pkg_name)
}

test_if_exists <- function(pkg_name){
  stop_if(find.package(pkg_name, quiet = TRUE),
          ~ length(.x) == 0,
          glue("{pkg_name} wasn't found on the machine."))
}
```

Hacking RStudio

Part 5: Conclusion



Colin Fay - ThinkR

Why enhancing RStudio?

To be more productive!

- Automate things
- Add Keyboard shortcuts
- Manipulate documents
- Don't lose time typing things over and over

What can we do?

- Manipulate the RStudio API (document manipulation, navigation, file creation...)
- Add addins and map them to keyboard shortcuts
- Make things appear fast with snippets
- Template all the things
- Manipulate your databases connections within RStudio

Thx! Questions?

Colin Fay

colin@thinkr.fr

http://twitter.com/_colinfay

http://twitter.com/thinkr_fr

https://github.com/ColinFay

https://thinkr.fr/

https://rtask.thinkr.fr/

https://colinfay.me/