Customizing and extending Emacs Speaks Statistics

Frédéric Santos[1]

January 18, 2021

[1] frederic.santos@u-bordeaux.fr

A short tutorial for customizing ESS

Target audience

This tutorial is for R-users beginning to use ESS. We only assume that you have mild familiarity with Emacs, and that you know how to customize Emacs by filling your `.emacs` or `init.el` file.

Goals

- Helping newcomers and beginners to have a more friendly and efficient setup
- Helping users migrating from other IDEs to keep behaviours and habits they are used to
- Showcasing some other Emacs packages that play well with ESS and may improve your coding experience

These slides are a companion for the video tutorial available at . . .

An opinionated tutorial?

Some parts of this tutorial (e.g., replacing `Flymake` by `Flycheck` for syntax checking, or using `company` instead of other solutions for auto-completion) might be seen as quite opinionated. However, when several options or approaches were available for a given task, the most widespread and efficient (I hope!) was presented.

If you prefer other approaches, the ESS manual remains the canonical documentation: `http://ess.r-project.org/Manual/ess.html`

A word about use-package

In this document, several pieces of Emacs Lisp code will be proposed so that you can use them in your init file.

It is assumed that you use use-package for your init file: the Emacs Lisp code can be adapted in a straightforward manner if you do not use it.

As a reminder, this is the minimal code to add in your init file so as to use use-package, once it has been installed:

```
;; Make sure that use-package is installed:
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))
;; Load use-package:
(eval-when-compile
  (require 'use-package))
```

Seeing matching parentheses

Directly taken from the ESS manual (`http://ess.r-project.org/Manual/ess.html#Parens`):

> *"Emacs has facilities for highlighting the parenthesis matching the parenthesis at point. This feature is very useful when trying to examine which parentheses match each other. This highlighting also indicates when parentheses are not matching."*

To activate parenthesis matching in ESS[R] (source) buffers, add this to your init file:

```
;; Activate global mode for parenthesis matching:
(show-paren-mode)
```

Navigating through matching parentheses

Here are some convenient tricks for navigating through parenthetical groups (this can be useful when dealing with large paren groups, e.g. when developing a shiny UI):

Shortcut	`Elisp function` (Docstring)
`C-M-p`	`backward-list` (Move backward across one balanced paren group)
`C-M-n`	`forward-list` (Move forward across one balanced paren group)
`C-M-SPC`	`mark-sexp` (Set mark at the end of the paren group)
`C-M-k`	`kill-sexp` (Kill from point to end of paren group)

Table 1: Some useful shortcuts for dealing with parenthetical groups.

For instance, when the point is over a closing parenthesis, `C-M-p` brings you to the matching opening parenthesis. Then, `C-M-k` kills to whole paren group.

On-the-fly syntax checking I

ESS has facilities for on-the-fly syntax checking. Instead of using `Flymake`, which is the default choice, using `Flycheck` appears to be a better and more stable option. To switch from Flymake to Flycheck, you can add the following in your init file:

```
;; Remove Flymake support:
(setq ess-use-flymake nil)
;; Replace it (globally) by Flycheck:
(use-package flycheck
  :ensure t
  :init
  (global-flycheck-mode t))
```

Completion I

As mentioned in the ESS manual, there are several completion frameworks for writing R code with ESS. The Emacs package `company` is an elegant solution, which also supports many other programming languages.

Here is a minimal piece of Elisp code to add in your init file to install and load `company`:

```
(use-package company
  :ensure t
  :config
  ;; Turn on company-mode globally:
  (add-hook 'after-init-hook 'global-company-mode)
  ;; Only activate company in R scripts, not in R console:
  (setq ess-use-company 'script-only))
```

Completion II

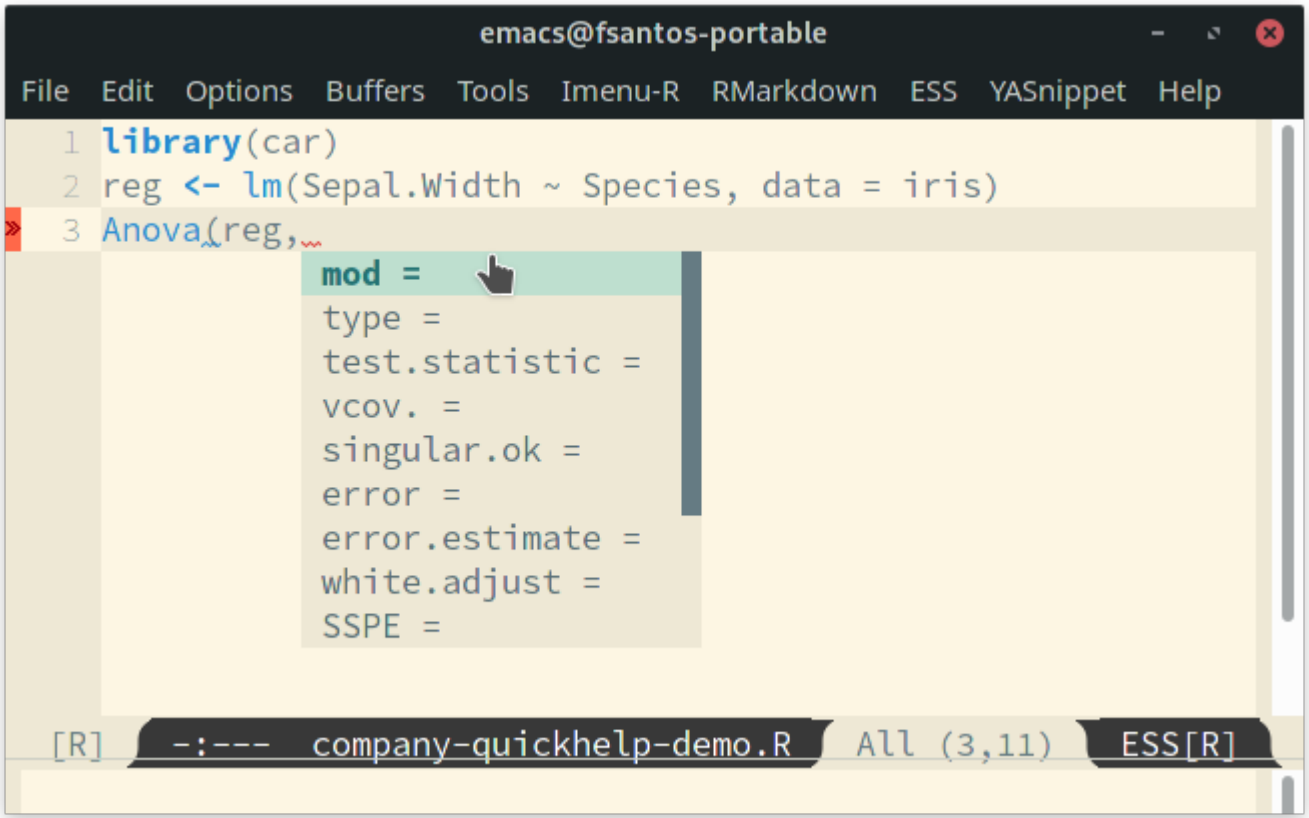

Figure 1: An example of code completion with `company`: various candidates are proposed for the arguments of the function `car::Anova()`.

Completion III

`company` offers completion candidates in various contexts: function name, argument name within a function call (as in Fig. 1), object name.

It may seem preferable to adopt a non-intrusive workflow. For functions or objects names, completion starts automatically after you type a few letters. For arguments names within a function call, it is suggested that you trigger manually the completion only when you need it. This can be done with `M-x company-complete`, or more conveniently, by binding this function to a convenient shortcut. For example, to bind it to `F12`, add the following to your init file:

```
;; Use F12 to trigger manually completion on R function args:
(add-hook 'ess-r-mode-hook
          '(lambda ()
             (local-set-key (kbd "<f12>") #'company-R-args)))
```

Completion IV

Of course, further customization of `company` can be done in your init file. For instance:

```
;; More customization options for company:
(setq company-selection-wrap-around t
      ;; Align annotations to the right tooltip border:
      company-tooltip-align-annotations t
      ;; Idle delay in seconds until completion starts automatica
      company-idle-delay 0.45
      ;; Completion will start after typing two letters:
      company-minimum-prefix-length 2
      ;; Maximum number of candidates in the tooltip:
      company-tooltip-limit 10)
```

Documentation popups I

`company-quickhelp` allows for documentation popups, e.g. to further describe function arguments.

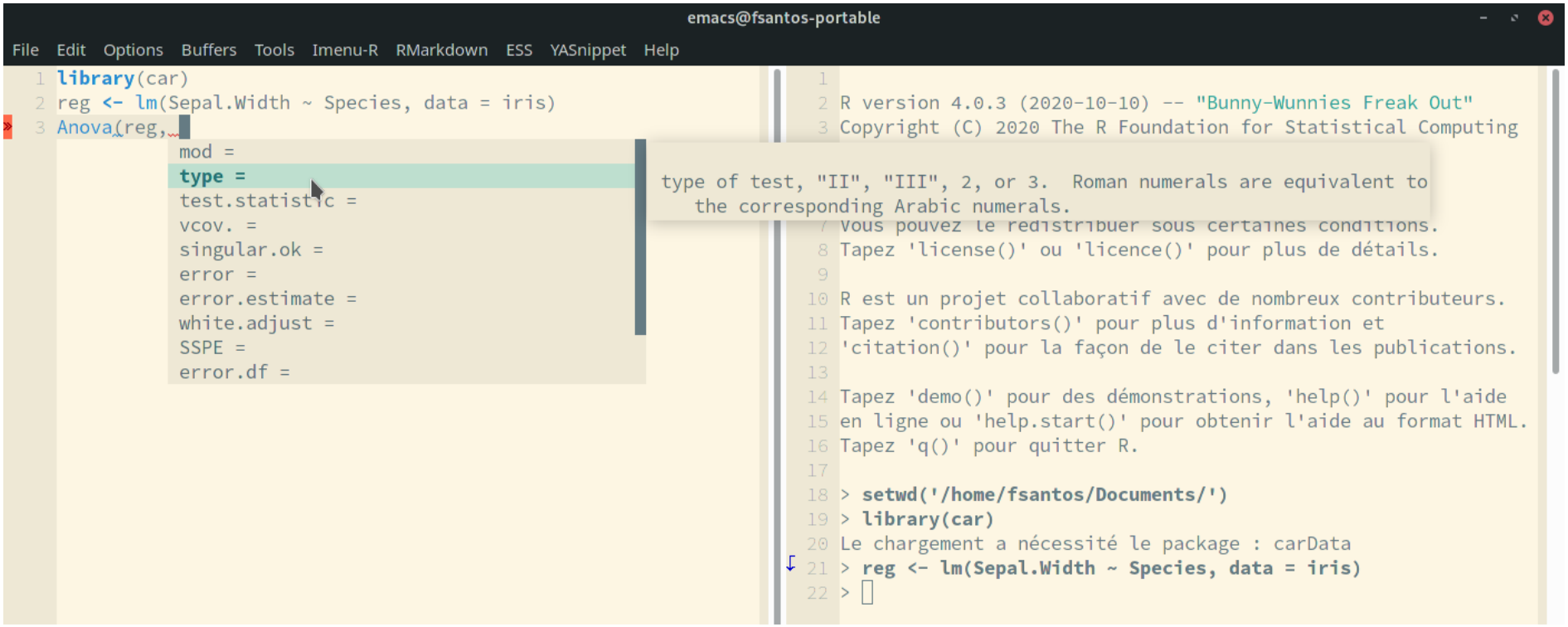

Figure 2: Documentation popups with `company-quickhelp`.

Documentation popups II

The minimal elisp code to add to your init file is straightforward:

```
(use-package company-quickhelp
  :ensure t
  :config
  ;; Load company-quickhelp globally:
  (company-quickhelp-mode)
  ;; Time before display of documentation popup:
  (setq company-quickhelp-delay 0.3))
```

By default, the documentation popup is shown automatically. You can adjust the time before the popup shows up by customizing the variable `company-quickhelp-delay`.

Code snippets

`yasnippet` is an Emacs package allowing for the expansion of whole pieces of code you often use (*snippets*) from one given abbreviation.

Key features of `yasnippet`

- All code snippets are stored as plain-text files in one given directory, so that they are easy to share with other people, and can be easily version controlled.
- As a corollary, it is also easy to retrieve and use large collection of snippets already available online. For instance, Andrea Crotti maintains a great collection available at `https://github.com/AndreaCrotti/yasnippet-snippets`.
- Although we only demonstrate its use within ESS and R here, note that `yasnippet` is not an R-specific solution, and that you can use it for any other programming language.

Setting up yasnippet I

To set up yasnippet, proceed through the following steps:

1. Create a directory snippets/ at some convenient location, and add a subfolder ess-r-mode/ in this directory.
2. Add the minimal following code in your init file:

```
(use-package yasnippet
  :ensure t
  :config
  ;; Indicate the directory containing your snippets:
  (setq yas-snippet-dirs '("path/to/your/snippets"))
  ;; Load your snippets on startup:
  (yas-reload-all)
  ;; Turn on yasnippet (minor) mode when editing R files:
  (add-hook 'ess-r-mode-hook #'yas-minor-mode))
```

Setting up yasnippet II

3. You can now fill your `snippets/ess-r-mode/` directory with your own snippets. For instance, create a file `function` (without any extension) in this directory, with the following contents:

```
#name : function
#key : fun
# --
${1:name} <- function(${2:args}) {
    ${3:body}
}
```

Each snippet has a unique `name`, and can be triggered by typing a given `key` (followed by `TAB`). As we will see later on, the present snippet allows for the expansion of a template for defining new R functions more easily. The `yasnippet` manual gives more details about the expected syntax to define your own code snippets: `http://joaotavora.github.io/yasnippet/`.

Setting up yasnippet III

4. Now your `snippets` directory should look like:

```
└── snippets
    └── ess-r-mode
        └── function
```

Feel free to add or retrieve (a lot!) more snippets, i.e. to add more template files within the `ess-r-mode` sub-directory.

Using `yasnippet` in an ESS[R] buffer

While you are editing an R source file with ESS, each snippet can be triggered by typing its `key` and then pressing `TAB`. You can then navigate through the placeholders of the expanded template by pressing `TAB` again.

For instance, with our previously defined snippet, typing `fun` followed by `TAB` will expand the full `function` template; you will then be able to specify easily a value for each of the three placeholders (the function's `name`, its `args` and `body`).

Note that `yasnippet` has a short video tutorial, available at `https://www.youtube.com/watch?v=ZCGmZK4V7Sg`.