JOHANNES KEPLER
UNIVERSITY LINZ

Extending R with Packages

Bettina Grün

R Ladies 2019

About me

- Associate Professor at the Department of Applied Statistics, Johannes Kepler University Linz.
- MSc and PhD in Applied Mathematics, Vienna University of Technology.
- Ordinay R Foundation member.
- Editor-in-Chief of the Journal of Statistical Software.

Why packages?

- An R package represents a standardized form to combine code, datasets and documentation and share them.
- The success of the R project is strongly linked to the possibility to extend the software and share this with others.
- Distribution of packages in particular via repositories:
  - Comprehensive R Archive Network (CRAN)
  - Bioconductor
- Most R users have experience with packages:
  - Packages are installed from CRAN using `install.packages()`.
  - Packages are loaded using `library()`, before they can be used.

Why packages? / 2

- Repository CRAN:
  - Available at `https://CRAN.R-project.org` with several mirrors.
  - Packages are distributed as source and binary packages.
  - In addition basic information on the package is provided as well as archived versions and the dependency structure with other packages.

  See, e.g., `https://CRAN.R-project.org/package=movMF`.

Packages in R

- R is built in a modular way via packages.
- There are different types of packages, depending on priority:

  base: These packages are part of the standard distribution of R and are developed by the R Core Team. They provide basic functinality as well as some statistical one.

  recommended: These packages are for example on Windows part of the standard distribution. They provide in particular statistical functionality.

  contributed: All other packages.

Documentation

- The manual “Writing R Extensions”, which is part of the standard installation of R, represents the official reference: `https://CRAN.R-project.org/doc/manuals/r-release/R-exts.html`
- The book “R Packages: Organize, Test, Document and Share Your Code” by Hadley Wickham: `http://r-pkgs.had.co.nz/`
- Both documents are freely available in the internet.

Package structure

- The sources of an R package consist of a folder containing the following elements:
  - File `DESCRIPTION`
  - File `NAMESPACE`
  - Folder `R`
  - Folder `man`
  - Folder `data`
  - Eventually also the folders: `demo`, `exec`, `inst`, `po`, `src`, `tests`, `tools`, `vignettes`
  - Eventually also the files: `INDEX`, `configure`, `cleanup`, `LICENSE`, `LICENCE`, `NEWS`

The `DESCRIPTION` file

- Contains the basic information about the package in “Debian Control File” format.
- The following fields are required:
  - Package
  - Version
  - License
  - Title
  - Description
  - Author, Maintainer or Authors@R
- Further fields are, for example, the dependency to other packages: Depends, Imports, . . .
- Base and recommended packages also have a field Priority.

Package dependencies

- `Depends`:
  - Dependency on a specific R version.
    E.g., `R (>= 3.0.0)`.
  - Package names (plus version), which need to be loaded before this package is loaded.
- `Imports`:
  - Package names (plus version), where the namespace needs to be imported.
- `Suggests`:
  - Package names (plus version), which are only needed for examples, tests and vignettes, or which are loaded inside of functions in the package.
- `Enhances`:
  - Indicates which packages are enhanced by this package.

The `NAMESPACE` file

- R has a namespace management system for code in packages.
- The large number of packages have made it necessary to define which packages and which functions from packages are used in a package. This avoids that functions with the same name are confused.
- This allows to explicitly define, which functions / variables in a package are **exported** and which functions / variables from other packages are **imported**.
- Within R one can access explicitly a function from a package using `::`, e.g., `movMF::movMF`
- Functions which are not exported are not part of the API and should in general not be used. One can nevertheless access these functions using `:::` `movMF:::A`
- All namespaces import the namespace of the **base** package. Other packages with priority base still need to be imported.

The folder `R`

- This folder contains R code.
- The files can have arbitrary names and should be sourceable.
- The R code can be distributed in an arbitrary way. In general reasonable file names help.
- The file names by default decide in which order files are sourced. One can use Collate in the `DESCRIPTION` file to specify an order.

The folder `data`

- This folder contains data sets.
- The data sets can either be lazy-loaded (field LazyLoad in the `DESCRIPTION` file) or explicitly loaded using `data()`.
- Files containing data can have 3 formats:
  - R code.
  - Tables in text format via .tab, etc. (see `?data`).
  - `save()` images.

  The data is stored with different efficiency / compression.

Documentation

- The folder `man` contains documentation in **R documentation** (Rd) format.
- Only user-visible functions and objects (which are exported) need to be documented. Selectively exporting functions reduces documentation work!
- The Rd format is a simple mark-up language, which is similar LaTeX and can be mapped to different other formats such as LaTeX, HTML, and plain text.

Creating a package

- There are tools available in the base distribution of R which help to create packages, once R cde, data, etc. are available.
  - `package.skeleton()`
  - `prompt()`, `promptClass()`, `promptMethods()`, `...`
- These files need to still be edited.
- Good documentation **cannot** be automatically generated. Only templates are created which define the structure and ensure that for example the indicated R usage is correct.

A first package

- Simple package which contains:
  - A function to fit a decision stump.
  - S3 methods for `print` and `fitted` for the returned object.
- In addition one requires:
  - `DESCRIPTION`.
  - `NAMESPACE`.
  - Documentation in folder `man`.

Further folders

- Folder `src`: For external code in C, C++, FORTRAN.
- Folder `vignettes`: Additional form of documentation combining text with R code with the final form available in PDF or HTML.
- Folder `tests`: Automatic testing of code.
- Folder `inst`: Additional files recursively copied into the installation folder.
- Folder `demo`: Contains code for function `demo()`.
- Folder `exec`: Contains executable scripts, e.g., shell or Perl scripts.
- Folder `po`: Contains files for internationalization.
- Folder `tools`: Is the preferred place for auxiliary files needed during configuration.

Checking packages

- `R CMD check` controls if an R source package is “correct”.
- A number of checks are performed, including:
  - The package is installed.
  - The file `DESCRIPTION` is checked.
  - R files are checked for syntax errors.
  - Documentation files Rd are checked.
  - Completeness and consistency of documentation is checked.
  - The examples in the documentation are run. If there is a file `tests/Examples/pkg-Ex.Rout.save`, it checks if results are unchanged.
  - If there is a folder `tests`, these are executed.
  - The code in vignettes is run and the PDF is created. The code output is also checked against a file `.Rout.save` if provided.
  - The PDF manual is generated.

Checking packages / 2

- `R CMD check` returns three types of messages:
  - `ERROR`s: Problems which need to be corrected.
  - `WARNING`s: Indicates that there is probably a problem. For submission to CRAN these need to be corrected.
  - `NOTE`s: Indicates a potential problem. Should be avoided as much as possible for submission to CRAN.

Building packages

- Packages can either be distributed as tar-balls (`.tar.gz` files) in form of source packages or as binary packages.
- `R CMD build` creates these files.
- If there is a file `.Rbuildignore` in the root directory of the package, this will be read to exclude files and folders from the built package.
- When building new folders are created and external code is compiled for binary versions.
- Built packages are intended for distribution.

Summary

- Starting point for creating a package is R code and data which should be packaged.
- In addition documentation needs to be created.
- Tests ensure that the package works.
- Namespaces ensure that the correct functions from other packages are used.
- Different tools for creating packages are available within base R and extension packages.