R packages

What is an R Package?

- In a nutshell, an R package is a collection of functions and datasets, with manuals.
- Remember one needs to “source” a user defined function into the workspace, or “load” a pre-saved dataset. But
  - It’s troublesome to source/load many functions/datasets.
  - Cannot provide function helps in R.
  - Functions written in other language (C, Fortran) depend on OS, so need to be recompiled.
  - More advanced issues: accessibility of functions (Namespace).

What is an R Package?

"*R Packages allow for easy, transparent and cross-platform extension of the R base system*."

"R package is a comfortable way to maintain collections of R functions and data sets. As an article distributes scientific ideas to others, a package distributes statistical methodology to others."

cran.r-project.org/doc/contrib/Leisch-CreatingPackages.pdf

Nowadays, it's typical (often expected) to have an R package with a new statistical method.

Terminologies for R Packages

Let's clarify some terms which sometimes get confused:

- **Package:** An extension of the R base system with code, data and documentation in standardized format.
- **Library**: A directory containing installed packages.
- **Repository**: A website providing packages for installation.
- **Package source:** The original version of a package with human-readable text and code.

- **Binary package:** A compiled version of a package with computer-readable text and code, may work only on a specific platform. So Each OS has its own version of the binary package.
- **Base packages:** Part of the R source tree, maintained by R Core team.
- **Recommended packages:** Part of every R installation, but not necessarily maintained by R Core.
- **Contributed packages:** All the rest. Contributed by developers and researchers. Many contributed packages on CRAN are written and maintained by R Core members.

All R functions are distributed with Packages

- Every single function in R belongs to a package.
- Most functions we've seen are in **base** or **stats** packages (two major base packages).
- Currently loaded package can be seen by calling `sessionInfo`:

```
> sessionInfo()
R version 3.2.1 (2015-06-18)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.10.4 (Yosemite)

locale:[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-
   8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base
```

All R functions are distributed with Packages

For a function, the package it belongs to can be seen in the help page:

```
> ?sum

sum                     package:base                 R Documentation

Sum of Vector Elements

Description:

    ‘sum’ returns the sum of all the values present in its arguments.

Usage:

     sum(..., na.rm = FALSE)
```

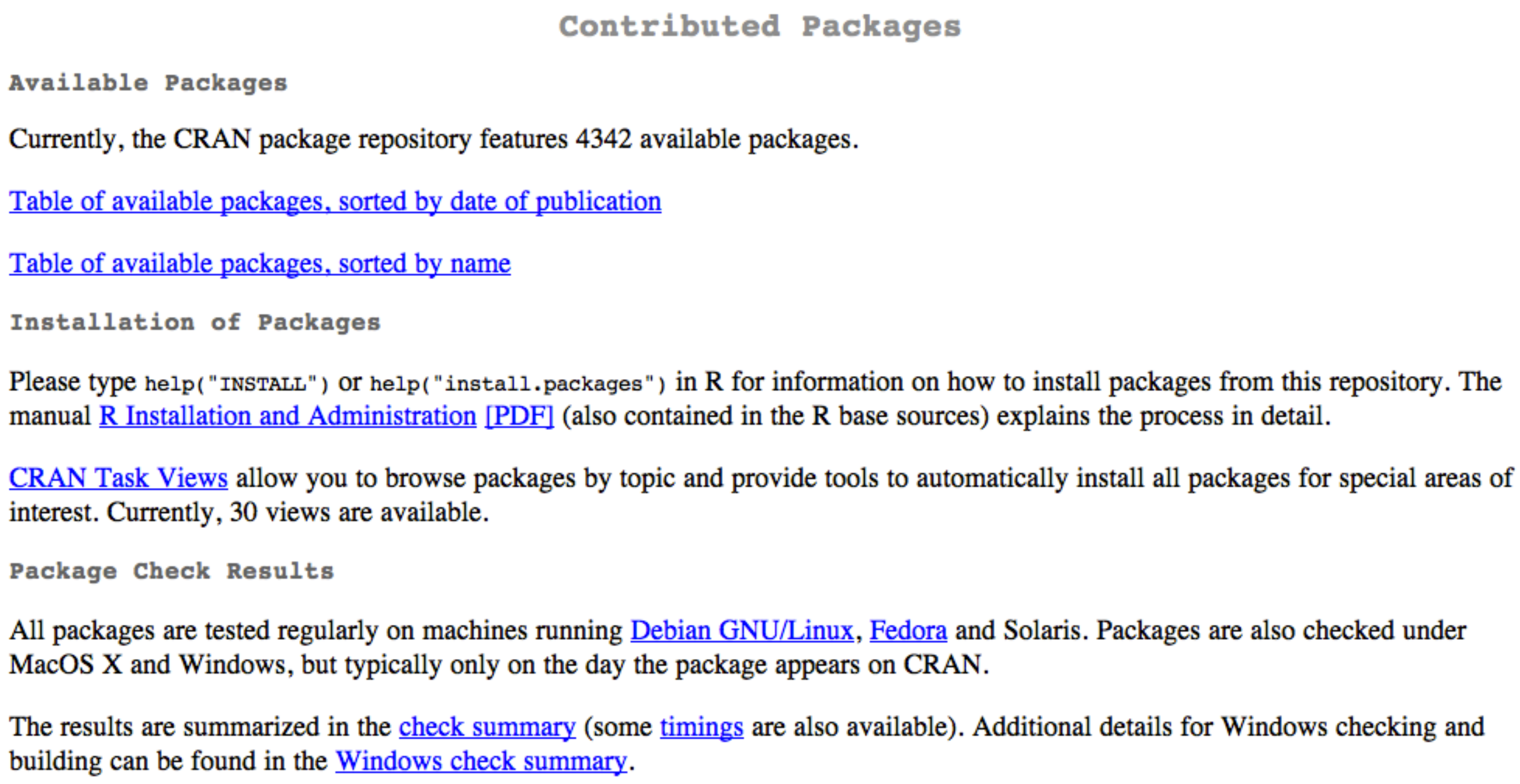
Finding and installing R packages

Where to obtain R packages

There are several Repositories where one can obtain R packages, including:

- CRAN: The Comprehensive R Archive Network. This is the “official” R package distribution center.
- Bioconductor: for bioinformatics related R packages.
- R-Forge
- Github.
- Researchers’ website.

CRAN: The Comprehensive R Archive Network



Contributed Packages

Available Packages

Currently, the CRAN package repository features 4342 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual R Installation and Administration [PDF] (also contained in the R base sources) explains the process in detail.

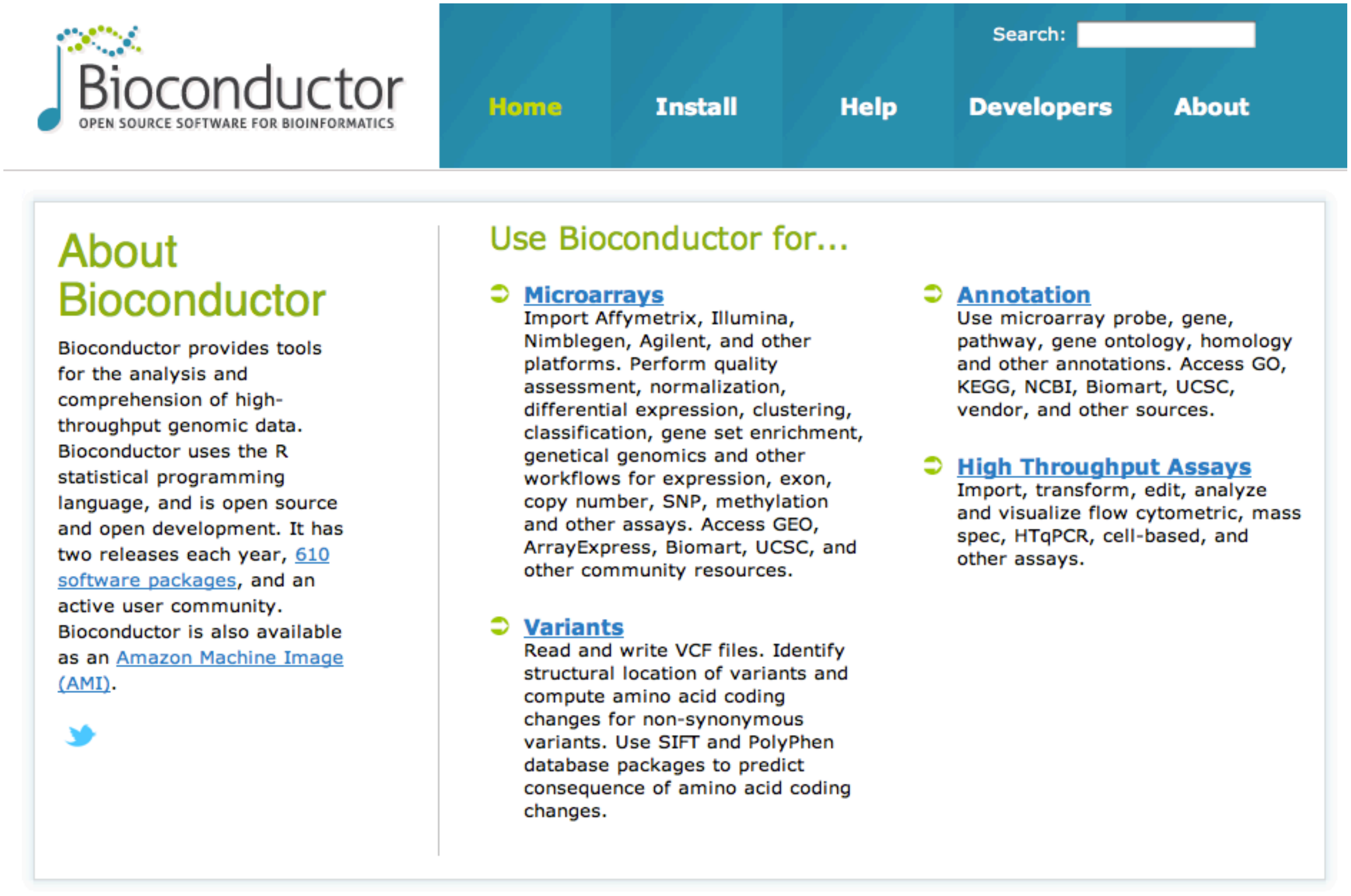
CRAN Task Views allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 30 views are available.

Package Check Results

All packages are tested regularly on machines running Debian GNU/Linux, Fedora and Solaris. Packages are also checked under MacOS X and Windows, but typically only on the day the package appears on CRAN.

The results are summarized in the check summary (some timings are also available). Additional details for Windows checking and building can be found in the Windows check summary.

Bioconductor



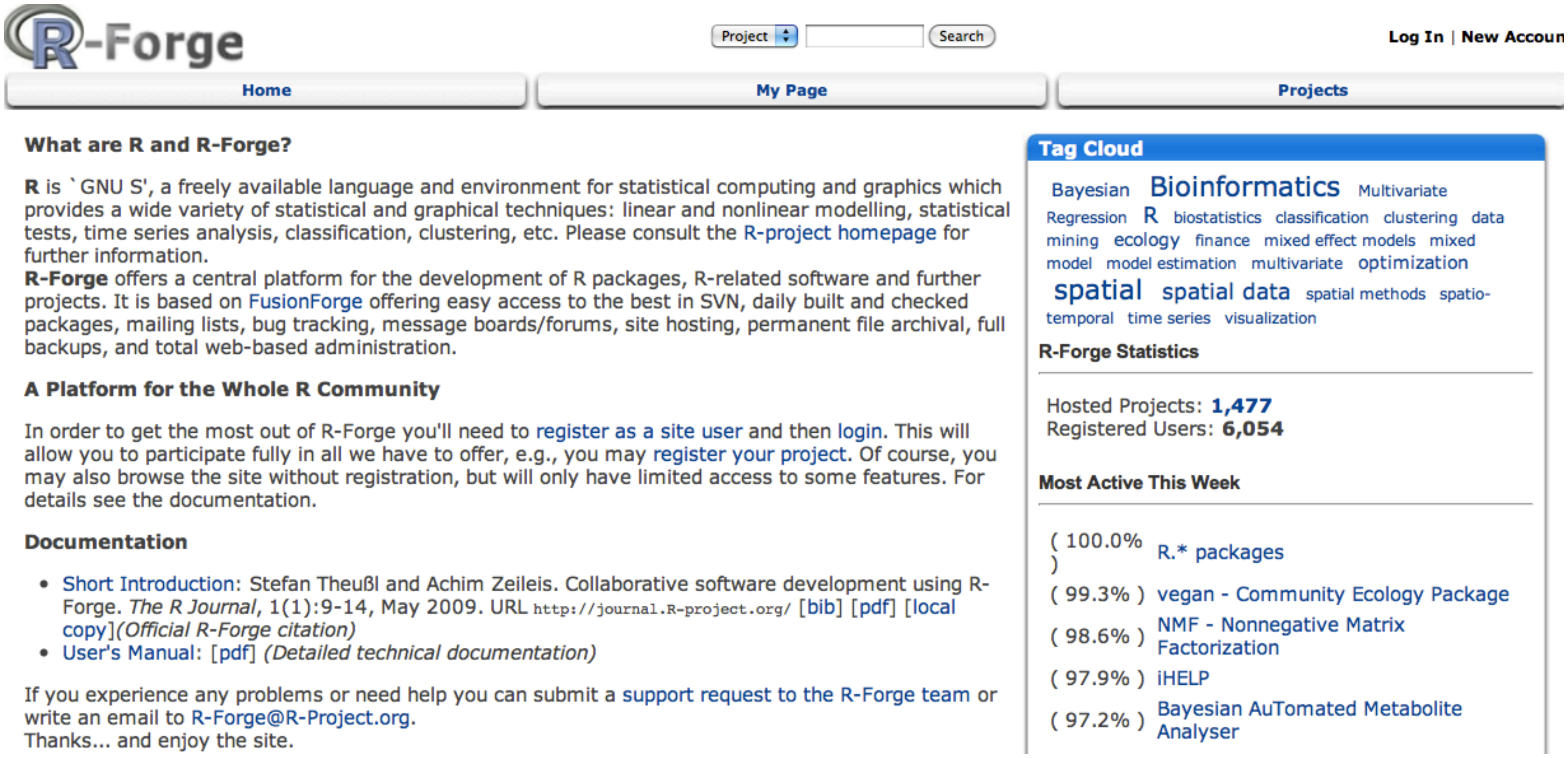
About Bioconductor

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor uses the R statistical programming language, and is open source and open development. It has two releases each year, 610 software packages, and an active user community. Bioconductor is also available as an Amazon Machine Image (AMI).

Use Bioconductor for...

- **Microarrays**
  Import Affymetrix, Illumina, Nimblegen, Agilent, and other platforms. Perform quality assessment, normalization, differential expression, clustering, classification, gene set enrichment, genetical genomics and other workflows for expression, exon, copy number, SNP, methylation and other assays. Access GEO, ArrayExpress, Biomart, UCSC, and other community resources.
- **Variants**
  Read and write VCF files. Identify structural location of variants and compute amino acid coding changes for non-synonymous variants. Use SIFT and PolyPhen database packages to predict consequence of amino acid coding changes.
- **Annotation**
  Use microarray probe, gene, pathway, gene ontology, homology and other annotations. Access GO, KEGG, NCBI, Biomart, UCSC, vendor, and other sources.
- **High Throughput Assays**
  Import, transform, edit, analyze and visualize flow cytometric, mass spec, HTqPCR, cell-based, and other assays.

R-Forge



What are R and R-Forge?

R is `GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R-project homepage for further information.
R-Forge offers a central platform for the development of R packages, R-related software and further projects. It is based on FusionForge offering easy access to the best in SVN, daily built and checked packages, mailing lists, bug tracking, message boards/forums, site hosting, permanent file archival, full backups, and total web-based administration.

A Platform for the Whole R Community

In order to get the most out of R-Forge you'll need to register as a site user and then login. This will allow you to participate fully in all we have to offer, e.g., you may register your project. Of course, you may also browse the site without registration, but will only have limited access to some features. For details see the documentation.

Documentation

- Short Introduction: Stefan Theußl and Achim Zeileis. Collaborative software development using R-Forge. *The R Journal*, 1(1):9-14, May 2009. URL `http://journal.R-project.org/` [bib] [pdf] [local copy]*(Official R-Forge citation)*
- User's Manual: [pdf] *(Detailed technical documentation)*

If you experience any problems or need help you can submit a support request to the R-Forge team or write an email to R-Forge@R-Project.org.
Thanks... and enjoy the site.

Tag Cloud

Bayesian Bioinformatics Multivariate Regression R biostatistics classification clustering data mining ecology finance mixed effect models mixed model model estimation multivariate optimization spatial spatial data spatial methods spatio-temporal time series visualization

R-Forge Statistics

Hosted Projects: **1,477**
Registered Users: **6,054**

Most Active This Week

(100.0%)	R.* packages
(99.3%)	vegan - Community Ecology Package
(98.6%)	NMF - Nonnegative Matrix Factorization
(97.9%)	iHELP
(97.2%)	Bayesian AuTomated Metabolite Analyser

A few remarks for R packages

- There's no guarantee for bug free (as for all free software).
- The major packages (base, stat, util, etc.) should work fine.
- There are a lot of badly written packages.
- Packages are updated often (especially Bioconductor packages), and backward compatibility is not guaranteed. This means sometimes your old codes stop working with the new package.
- To get help, use mailing list (see http://www.r-project.org/mail.html), or ask the developer directly.

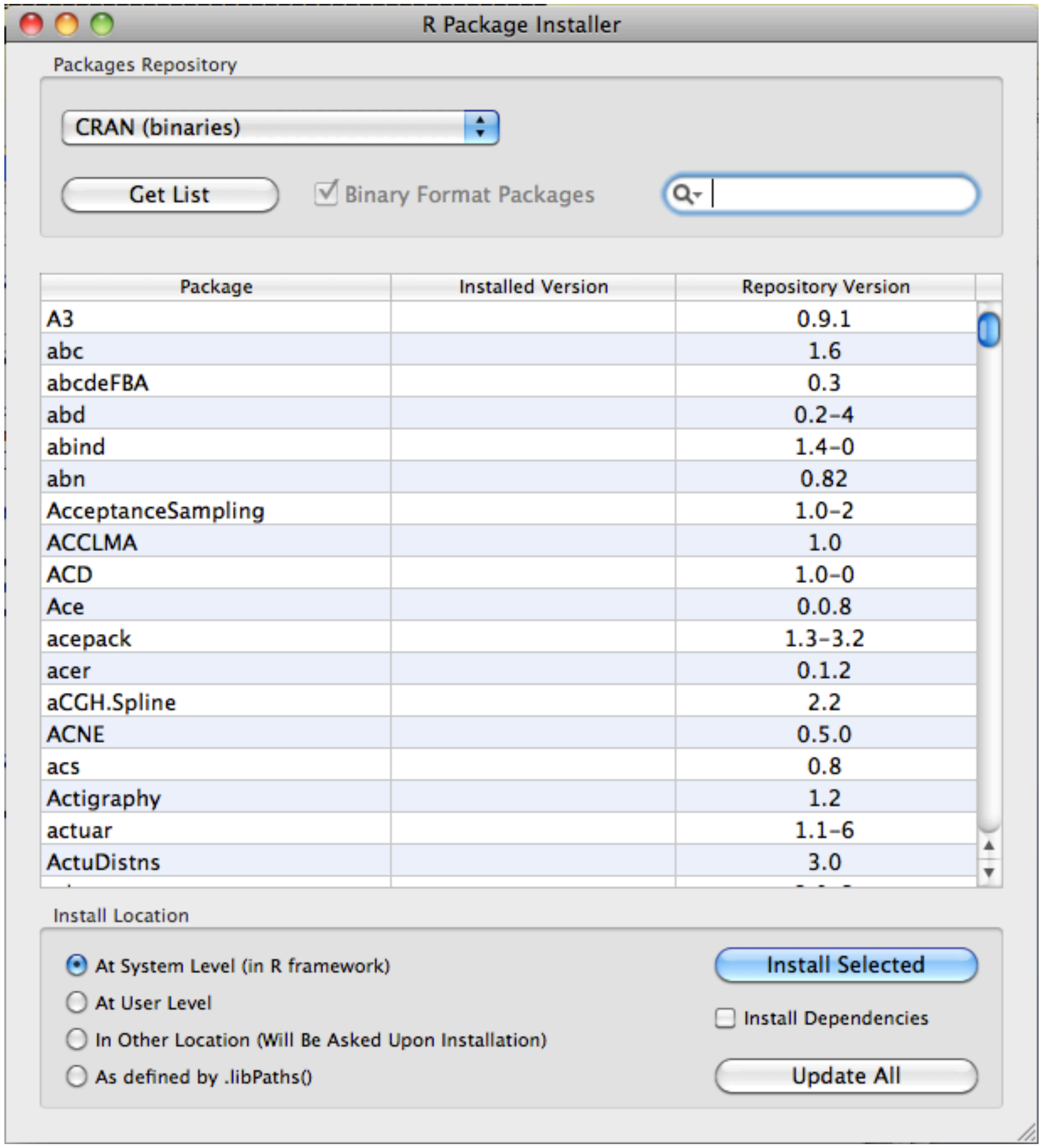
R package installation

- Can be done in different ways:
  - Use GUI.
  - In R console.
  - Use command line.
- Using the GUI or R console is easier. It will determine the correct platform and R version, also install dependence packages.
- It's tricky to use command line to install from **package source**. But sometimes one has to.
- Packages will be installed to the **Library**.

Install a new R package - using the GUI

On Mac:

- Under “Packages & Data” menu, click “Package Installer” and get a dialog window.
- Choose a repository and specify the location, then install.


R Package Installer

Packages Repository

CRAN (binaries)

Get List | [✓ Binary Format Packages]

Package	Installed Version	Repository Version
A3		0.9.1
abc		1.6
abcdeFBA		0.3
abd		0.2-4
abind		1.4-0
abn		0.82
AcceptanceSampling		1.0-2
ACCLMA		1.0
ACD		1.0-0
Ace		0.0.8
acepack		1.3-3.2
acer		0.1.2
aCGH.Spline		2.2
ACNE		0.5.0
acs		0.8
Actigraphy		1.2
actuar		1.1-6
ActuDistns		3.0

Install Location

[✓ At System Level (in R framework)]
At User Level
In Other Location (Will Be Asked Upon Installation)
As defined by .libPaths()

Install Selected

Install Dependencies

Update All

Install a new R package - using the GUI

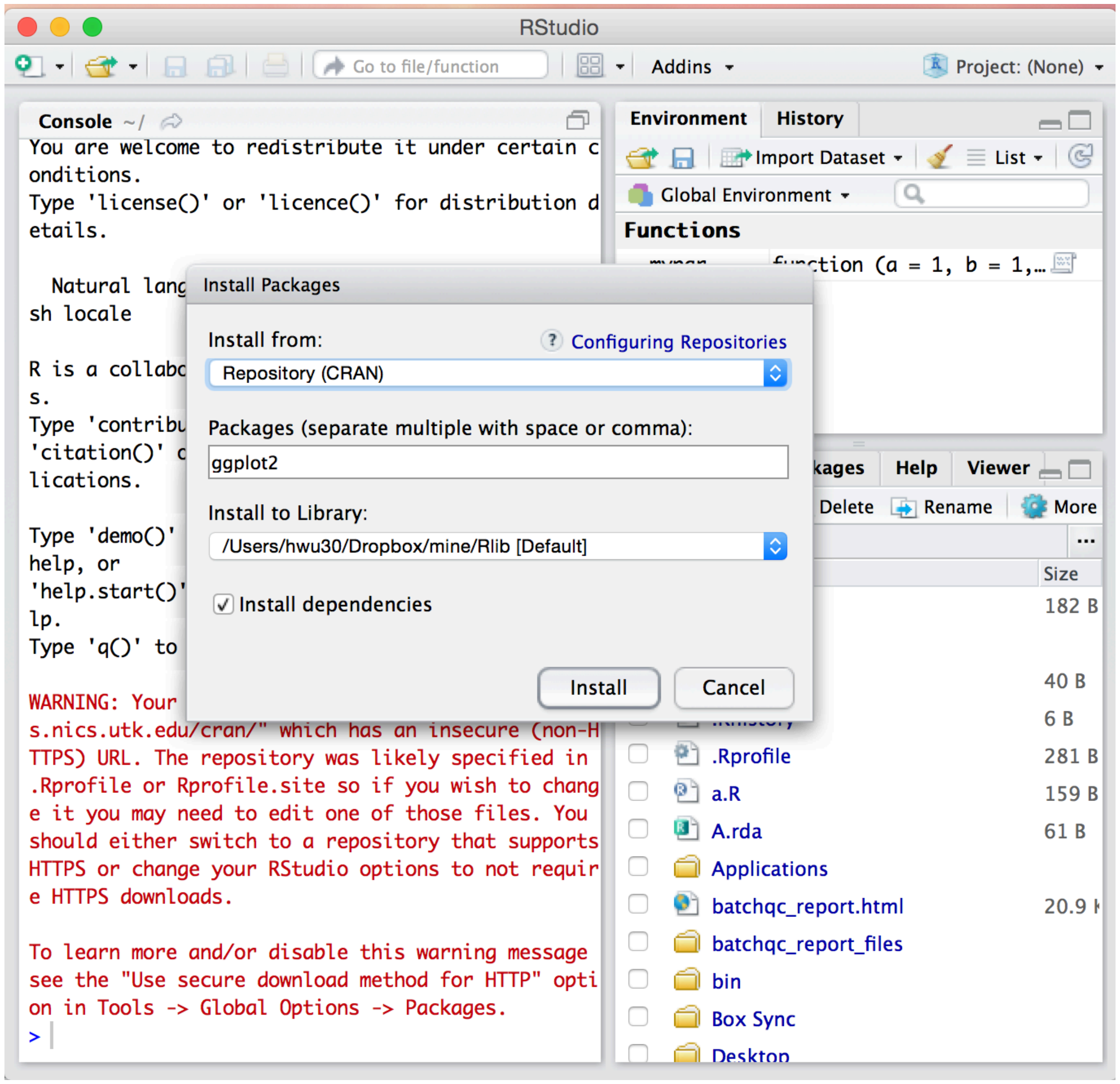
On Windows:

For CRAN packages:

- Click “Packages” menu, then select “Install package(s) …”. If you do this the first time, you will be asked to choose a CRAN mirror set. Select “USA(TN)”.
- Click the package you want to install, then click OK. Again if it’s the first time, you’ll be asked to create a personal library, just click OK.
- For packages not on CRAN. You will have to have the package source as a zip or gz file.
- Click “Packages” menu, then select “Install package(s) from local zip files …”.

Install a new R package - RStudio

- Under "Tools" menu, click "Install Packages" and get a dialog window.
- Choose a repository (CRAN or package file) and specify the location, then install.



```
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural lang
sh locale

R is a collabo
s.
Type 'contribu
'citation()' o
lications.

Type 'demo()'
help, or
'help.start()'
lp.
Type 'q()' to

WARNING: Your
s.nics.utk.edu/cran/" which has an insecure (non-HTTPS) URL. The repository was likely specified in .Rprofile or Rprofile.site so if you wish to change it you may need to edit one of those files. You should either switch to a repository that supports HTTPS or change your RStudio options to not require HTTPS downloads.

To learn more and/or disable this warning message see the "Use secure download method for HTTP" option in Tools -> Global Options -> Packages.
>
```

Install Packages

Install from: Repository (CRAN) — Configuring Repositories

Packages (separate multiple with space or comma): ggplot2

Install to Library: /Users/hwu30/Dropbox/mine/Rlib [Default]

☑ Install dependencies

Install | Cancel

	Name	Size
		182 B
		40 B
		6 B
☐	.Rprofile	281 B
☐	a.R	159 B
☐	A.rda	61 B
☐	Applications	
☐	batchqc_report.html	20.9
☐	batchqc_report_files	
☐	bin	
☐	Box Sync	
☐	Desktop	

Install a new R package - in R console

- For packages from CRAN, use the "**`install.packages`**" function to install.

```
> install.packages("ggplot2")
also installing the dependencies ‘digest’, ‘gtable’, ‘reshape2’, ‘proto’

trying URL 'http://cran.r-
   project.org/bin/macosx/leopard/contrib/2.15/digest_0.6.3.tgz'
Content type 'application/x-gzip' length 161113 bytes (157 Kb)
opened URL
==================================================
downloaded 157 Kb

trying URL 'http://cran.r-
   project.org/bin/macosx/leopard/contrib/2.15/gtable_0.1.2.tgz'
Content type 'application/x-gzip' length 60865 bytes (59 Kb)
opened URL
==================================================
```

Install a new R package - in R console

- For packages from Bioconductor, use the “**`biocLite`**” function to install.

```
> source("http://bioconductor.org/biocLite.R")
Bioconductor version 2.11 (BiocInstaller 1.8.3), ?biocLite for help

> biocLite("GenomicFeatures")
BioC_mirror: http://bioconductor.org
Using Bioconductor version 2.11 (BiocInstaller 1.8.3), R version 2.15.
Installing package(s) 'GenomicFeatures'
also installing the dependencies ‘IRanges’, ‘GenomicRanges’

trying URL
   'http://bioconductor.org/packages/2.11/bioc/bin/macosx/leopard/contrib/2.1
   5/IRanges_1.16.5.tgz'
Content type 'application/x-gzip' length 2052325 bytes (2.0 Mb)
opened URL
==================================================
downloaded 2.0 Mb
```

Install a new R package - in R console

- For packages from github, use the "**`install_github`**" function to install.
- **`devtools`** package needs to be installed first.

```
> library(devtools)
> install_github("username/packagename")
```

Install a new R package - in command line

- If you are familiar with Linux style command line, you can obtain the package code (usually in a .tar.gz format), and issue the following command to install:

```
R CMD INSTALL  pkg_1.0.tar.gz
```

- Note that sometimes you have to do this, for example, when obtaining a source package from a researcher’s website.

Setting up personal R library

This is important if you are using a Linux machine, or a Mac but you don’t have root access.

- The R “root library” is not accessible to you (you have no write permission, so can’t install package).
- You can create a personal R library by doing:

- Create a directory somewhere you have access, say, C:\Rlib or ~/Rlib.

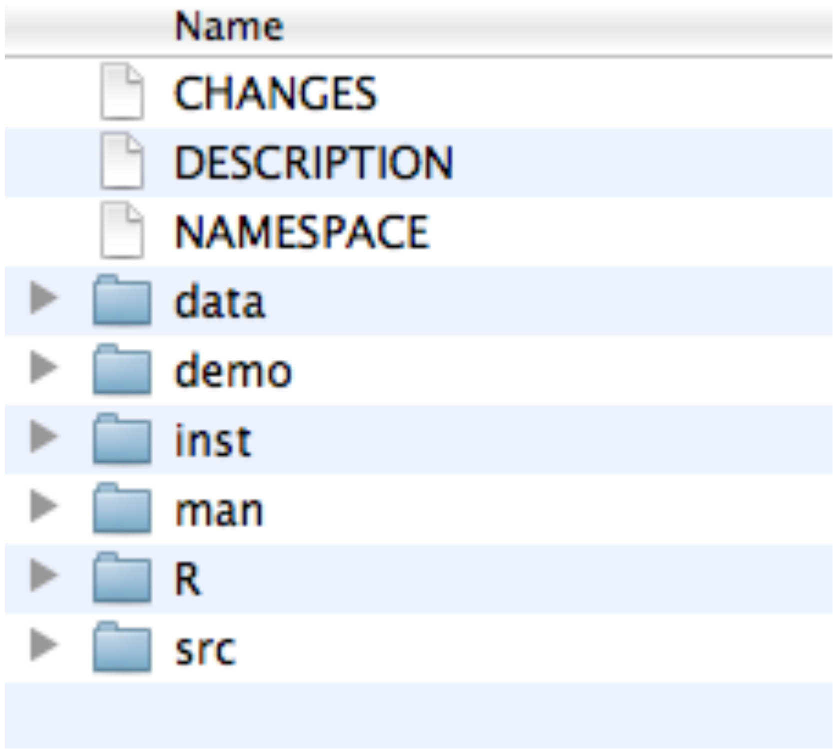
- Create a file call “**`.Renviron`**” (start with a dot) containing following line: `R_LIBS=~/Rlib`.

- Every time R start, it’ll read **`.Renviron`** and know that you have a personal library at `~/Rlib`.

Creating R packages

The structure of an R package

- The source of an R package is saved in a directory with following files and directories:



Name
CHANGES
DESCRIPTION
NAMESPACE
data
demo
inst
man
R
src

The structure of an R package

- Three simple text files:
  - **DESCRIPTION** (required): description of the package.
  - **CHANGES** (optional): a log of changes in the codes.
  - **NAMESPACE** (optional): define the package “namespace”.
- Directories:
  - **R** (required): all R source codes.
  - **man** (optional): help files (in Rd).
  - **src** (optional): source codes in other languages (C/Fortran).
  - **data** (optional): data (in RData) distributed with the package.
  - **inst** (optional): additional instruction files (like software manual).
  - **demo** (optional): demo scripts.

Create an R package

- In RStudio, this can be done by creating a new project: from menu: File -> New Project -> New Directory -> R package. This will creates the “skeleton” of the package, e.g., empty directories and files.
- Alternatively from R console, one can use “**`package.skeleton`**” function.

```
> package.skeleton("coolpkg")
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.

Further steps are described in './coolpkg/Read-and-delete-me'.
```

Steps after having the empty package folder

1. Modify DESCRIPTION file to include information about the package.
2. Modify NAMESPACE file for package namespace *.
3. Put R codes into the R directory.
4. Put C/Fortran codes (if any) into the src directory *.
5. Go to man directory and create help files.
6. Create the package (as a single zip file).

* if necessary.

Package files - DESCRIPTION

The **DESCRIPTION** file contains basic information about the package in the following format:

```
Package: coolpkg
Type: Package
Title: What the package does (short line)
Version: 1.0
Date: 2016-02-04
Author: Who wrote it
Maintainer: Who to complain to <yourfault@somewhere.net>
Description: More about what it does
License: What license is it under?
```

Packages files - DESCRIPTION

```
Package: coolpkg
```

The mandatory ‘Package’ field gives the name of the package. This should contain only letters, numbers or a dot, have at least two characters and start with a letter and not end in a dot.

```
Version: 1.0
```

The mandatory ‘Version’ field gives the version of the package. This is a sequence of at least *two* (and usually three) non-negative integers separated by single ‘.’ or ‘-’ characters.

Packages files - DESCRIPTION

```
License: GPL-2
```

The mandatory 'License' field should specify the license of the package in a standardized form. Alternatives are indicated *via* vertical bars. Individual specifications must be one of the "standard" short specifications: GPL-2 GPL-3 LGPL-2 LGPL-2.1 LGPL-3 AGPL-3 Artistic-1.0 Artistic-2.0 . Some other examples:

```
License: GPL (>= 2) | BSD
License: LGPL (>= 2.0, < 3)
```

Packages files - DESCRIPTION

```
Description: It does great things
```

The mandatory ‘Description’ field should give a comprehensive description of what the package does. One can use several (complete) sentences, but only one paragraph.

```
Title: CoolPkg
```

The mandatory ‘Title’ field should give a short description of the package. Some package listings may truncate the title to 65 characters. It should be capitalized, not use any markup, not have any continuation lines, and not end in a period.

Packages files - DESCRIPTION

`Author: John Doe`

The mandatory ‘Author’ field describes who wrote *the package*. It is a plain text field intended for human readers

`Maintainer: <youremail@somewhere.net>`

The mandatory ‘Maintainer’ field should give a *single* name with a *valid* (RFC 2822) email address in angle brackets (for sending bug reports etc.). It should not end in a period or comma.

Packages files - DESCRIPTION

```
Date: 2014-01-17
```

The ‘Date’ field gives the release date of the current version of the package. It is strongly recommended to use the yyyy-mm-dd format conforming to the ISO 8601 standard.

```
Depends: R (>= 2.11.0)
```

The optional ‘Depends’ field gives a comma-separated list of package names which this package depends on. The package name may be optionally followed by a comment in parentheses.

Other optional contents in an R package

- Function helps
  - Need to create .Rd files and save under /man directory.
- R data
  - Need Need to save .RData files and save under /data directory.
- R vignette
  - Software manual, often created using Sweave.
- We will cover these later.

Building the Package - RStudio

- RStudio provide convenient interface to check and build packages.
- Build -> Check Package: run a comprehensive checking of the package.
    - If there are errors, they need to be corrected. Warnings are okay. However submitting to CRAN requires no warning from checking.
- Build -> Build Package (source or binary): build the package source or binary.
    - This create a single file (usually .tar.gz) for the package, which can be shared or submitted to CRAN.

Building the Package – command line

- Checking package: **`R CMD check coolpkg`**
- Building source package: **`R CMD build coolpkg`**

Congratulations ! You now have a package that you can distribute to someone else or post on CRAN for sharing with the world.

Writing R function helps

- In package, we have defined some functions (contained in R files under the /R directory).
- We want to provide helps for these functions that can be accessed by typing "`?function`" in R.
- The function helps are written in simple text files with extension **Rd**. These files are saved under **/man** directory.
- You don't have to have help files for making a package, e.g., `man` directory can be empty. But it's better to have function helps.
- If you were to submit package to CRAN/Bioconductor, function helps are required.

Creating an Rd file

- The simplest way is to use "**prompt**" function. It takes the name of a function and generates a template of the Rd file. You can then modify the Rd file to put in more information.
- For example, I have following function in my R workspace:

```
myRowSums=function(x) {
  n0 = nrow(x)
  result = rep(0, n0)
  for(i in 1:n0) {
    result[i]=sum(x[i,])
  }
  return(result)
}
```

- I can use `prompt` to create a help file (saved under current directory) for this function:

```
> prompt(myRowSums)
Created file named ‘myRowSums.Rd’.
Edit the file and move it to the appropriate directory.
```

myRowSums.Rd

```
\name{myRowSums}
\alias{myRowSums}
%- Also NEED an '\alias' for EACH other topic documented here.
\title{
%%  ~~function to do ... ~~
}
\description{
%%  ~~ A concise (1-5 lines) description of what the function does. ~~
}
\usage{
   myRowSums(x)
}
%- maybe also 'usage' for other objects documented here.
\arguments{
  \item{x}{
%%     ~~Describe \code{x} here~~
  }
}
```

```
\details{
%%  ~~ If necessary, more details than the description  above ~~
}
\value{
%%  ~Describe  the value returned
%%  If it is a LIST, use
%%  \item{comp1  }{Description  of 'comp1'}
%%  \item{comp2  }{Description  of 'comp2'}
%% ...
}
\references{
%% ~put references  to the literature/web  site here ~
}
\author{
%%  ~~who you are~~
}
\note{
%%  ~~further  notes~~
}
```

```
\seealso{
%% ~~objects to See Also as \code{\link{help}}, ~~~
}

\examples{
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--     or do  help(data=index)   for the standard data sets.
}

% Add one or more standard keywords, see file 'KEYWORDS' in the
% R documentation  directory.
\keyword{ ~kwd1 }
\keyword{ ~kwd2 }% __ONLY ONE__ keyword per line
```

Rd file format

- Similar to a LaTex file format.
- Multiple sections. Section title starts with \. Section is within curly braces {}. Pay attention to the pairing of curly braces.
- Required fields: `\name, \title, \description, \usage, \arguments, \value`
- Others are optional: `\alias, \details, \references, \author, \note, \seealso, \examples, \keyword`.
- The `arguments` field, use \item to create bulleted list.

Edited myRowSums.Rd

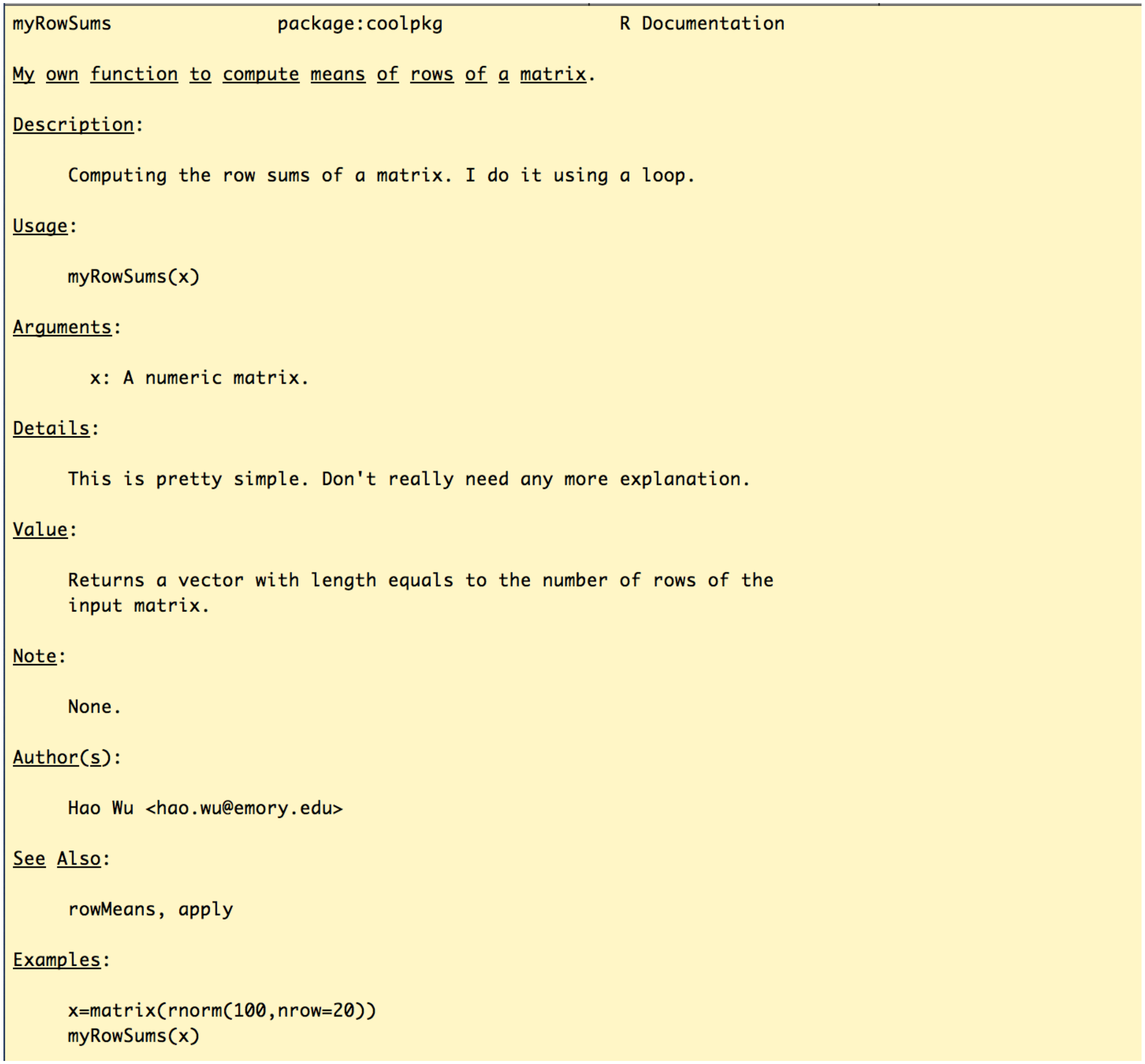
```
\name{myRowSums}
\alias{myRowSums}
\title{
  My own function to compute means of rows of a matrix.
}
\description{
  Computing the row sums of a matrix. I do it using a loop.
}
\usage{
myRowSums(x)
}
\arguments{
  \item{x}{A numeric matrix.}
}
\details{
  This is pretty simple. Don't really need any more explanation.
}
```

```
\value{
  Returns a vector with length equals to the number of rows of the input
matrix.
}
\author{
  Hao Wu <hao.wu@emory.edu>
}
\note{
  None.
}

\seealso{
  rowMeans, apply
}
\examples{
x=matrix(rnorm(100,nrow=20))
myRowSums(x)
}

\keyword{test}
```

Build/install package with function help

- We can use the same procedure to build and install the package.
- This time the function helps will be available, and can be accessed by “?function”.

What we get



```
myRowSums               package:coolpkg               R Documentation

My own function to compute means of rows of a matrix.

Description:

     Computing the row sums of a matrix. I do it using a loop.

Usage:

     myRowSums(x)

Arguments:

       x: A numeric matrix.

Details:

     This is pretty simple. Don't really need any more explanation.

Value:

     Returns a vector with length equals to the number of rows of the
     input matrix.

Note:

     None.

Author(s):

     Hao Wu <hao.wu@emory.edu>

See Also:

     rowMeans, apply

Examples:

     x=matrix(rnorm(100,nrow=20))
     myRowSums(x)
```

Data in R packages

- If one wants to distribute data with an R package, the saved data files (with extension .RData or .rda) need to be put under the `data` directory.
- The data then can be loaded using “data” function. For example, if there’s **mydata.RData** under **`\data`** directory for my package, doing “**data(mydata)**” will load that data into R.
- “prompt” function can be used to create Rd file for the help page for the data.

Other (more advanced) package contents

- In addition to the Rd files, one can provide additional software manual or user guide (often under `/inst/doc` or `/vignette`). These can be written using Sweave (discuss a little later).
- A `CITATION` file can be used to include references to literature connected to the package, the contents of the file can be accessed from within R using:

  `citation("coolpkg")`.
- If there are C/Fortran functions, the complied binary need to be loaded when loading the library. In that case a "processing function" (traditionally called `zzz.R`) is needed.

Sweave and R vignette

What is Sweave

- Sweave is a **system** that enables integration of R code into LaTeX documents. The purpose is "*to create dynamic reports, which can be updated automatically if data or analysis change*".
- Sweave is an also an **R function** (?Sweave) that implements the system.
- The name of Sweave (S-weave) is from Cweave, which is a way to generate documentations for C.

The basic ideas of Sweave

- When writing a document such as data analysis report, one can:
  - perform all analyses and save the figures/tables, then create a document with these; or
  - write an Sweave file, which is a mix of LaTex and R codes. At compilation, the R codes will run and result figures/tables will be included in the document automatically.

The steps of Sweave

The steps of using Sweave to create a document are:

1.Create a text file with extension .Rnw. The text file is a mixture of LaTex and R codes. Assume the file is called coolpkg.Rnw.

2.Process the file using Sweave function:

```
> Sweave("coolpkg.Rnw")
Writing to file coolpkg.tex
Processing code chunks with options …
You can now run (pdf)latex on ‘coolpkg.tex’
```

This generates a .tex file under current directory. What it did is to run the R codes embedded in Rnw, and replace those with proper tex commands.

3.Compile the tex file using your LaTex compiler (latex, pdflatex, etc.)

A (tiny) bit of LaTex

- LaTex is a high quality typesetting system (like MS Office on PC or iWork on Mac).
- Supports multiple types documents: article, slides.
- Widely used in academia, especially when there are a lot of equations. So this is important for statisticians.
- The idea is to write the source in a "markup language" as a text file (with extension .tex), then compile and generate the document in postscript or pdf.

Compared to other WYSIWYG software

- Compared to other “what you see is what you get” typesetting software (MS Office, Google docs, iWorks, etc.), LaTex has following advantages:
  - Free.
  - Better, more professional looking.
  - Better control of formatting, if you know it.
  - Faster and smaller, because it’s a simple text file.
  - Most importantly, faster to type equations once you know it (note that you can type LaTex in Word to generate equations now).
- The disadvantages are:
  - A little more difficult to learn compared to, say, MS Word.
  - All formatting needs to be set manually. This could be troublesome sometimes.
  - Lack of some useful features (e.g., no track changes.).

A simple LaTex example

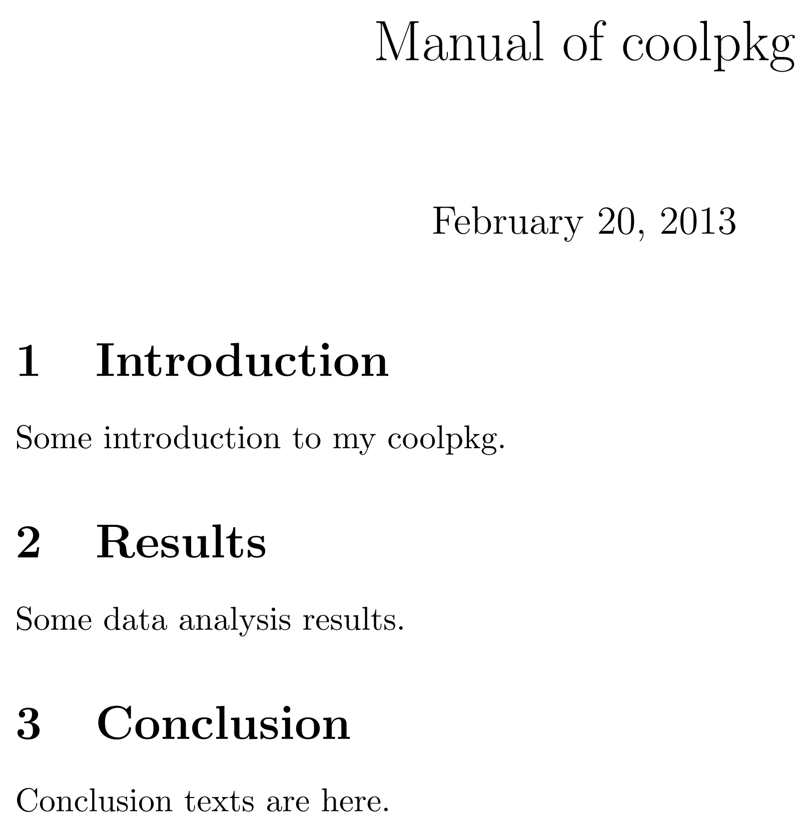
```
\documentclass{article}
\title{Manual of coolpkg}

\begin{document}
\maketitle

\section{Introduction}
Some introduction to my coolpkg.

\section{Results}
Some data analysis results.

\section{Conclusion}
Conclusion texts are here.

\end{document}
```

Compiling LaTex

- You will need some software to compile LaTex:
  - On Linux, the “latex” and “pdflatex” command are distributed with the OS, so no additional installation needed.
  - On Mac, one needs to install MacTex, and can use TexShop as the GUI.
  - On PC, use MikTex and WinEdt.
  - There are other options. Google them.

What I get

The tex file is saved as coolpkg.tex. After compilation (running “`pdflatex coolpkg.tex`” at command window), I get the following cookpkg.pdf.



Manual of coolpkg

February 20, 2013

1 Introduction

Some introduction to my coolpkg.

2 Results

Some data analysis results.

3 Conclusion

Conclusion texts are here.

Now back to Sweave

- A Sweave file (.Rnw) is a mix of LaTex and R codes. The R codes are within special tags: start by **<<>>=** , and end by **@**.
- Control commands can be specified within <<>>.
- For example, to include some R codes and results, one can add following in the tex:

```
\section{Results}
Some data analysis results. Below is example of table function.
<<echo=TRUE, eval=TRUE>>=
table(mtcars$cyl)
@
```

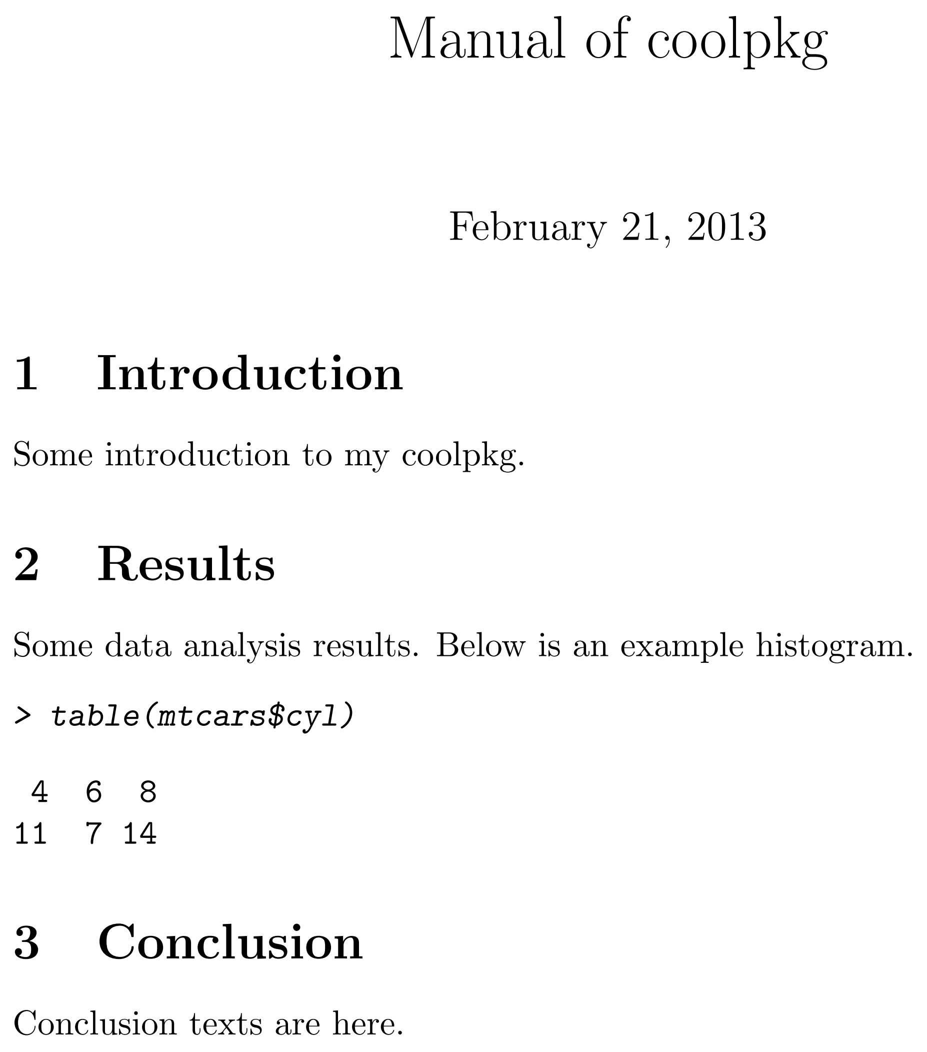
After Sweave

After running Sweave(“coolpkg.Rnw”), I got coolpkg.tex like this:

```
\documentclass{article}
\usepackage{Sweave}
......

\section{Results}
Some data analysis results. Below is an example histogram.
\begin{Schunk}
\begin{Sinput}
> table(mtcars$cyl)
\end{Sinput}
\begin{Soutput}
 4  6  8
11  7 14
\end{Soutput}
\end{Schunk}
```

The document

After compiling coolpkg.tex (Sweavy.sty is needed!), the pdf looks like:



Manual of coolpkg

February 21, 2013

1 Introduction

Some introduction to my coolpkg.

2 Results

Some data analysis results. Below is an example histogram.

```
> table(mtcars$cyl)

 4  6  8
11  7 14
```

3 Conclusion

Conclusion texts are here.

Include figures

- To include figure, one can do:

```
\section{Results}
Some data analysis results. Below is example of table function.
<<echo=TRUE, eval=TRUE>>=
table(mtcars$cyl)
@

Below is a histogram:
<<fig=TRUE>>=
  hist(rnorm(100))
@
```

After Sweave

After running Sweave(“coolpkg.Rnw”), now the coolpkg.tex is:

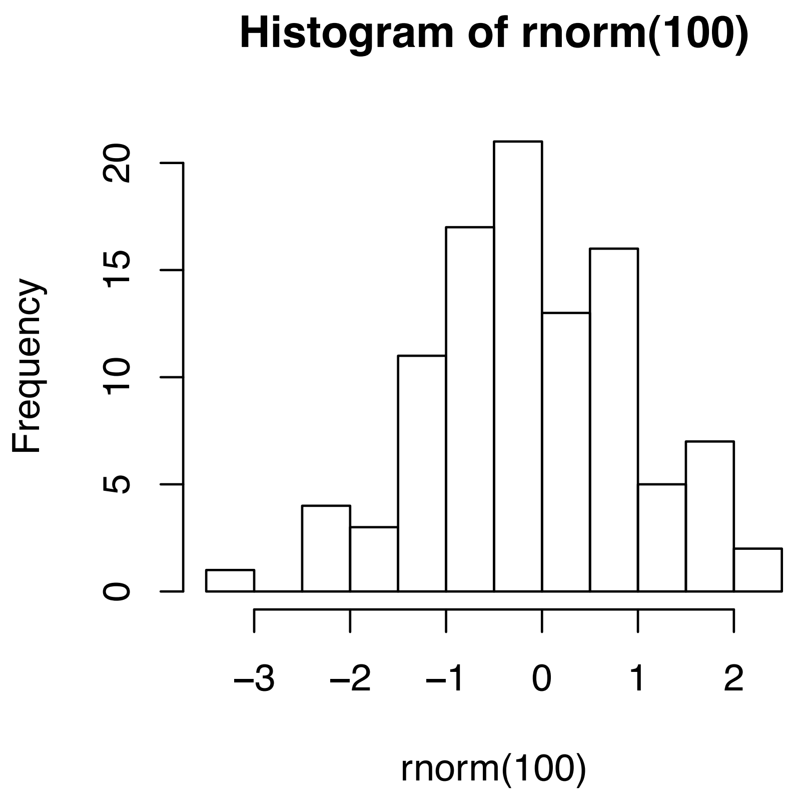
```
\documentclass{article}
\usepackage{Sweave}
......

\section{Results}
......
Below is a histogram:
\begin{Schunk}
\begin{Sinput}
>   hist(rnorm(100))
\end{Sinput}
\end{Schunk}
\includegraphics{coolpkg-002}
```

The document with the figure

After compiling the tex, the figure is there now:

Below is a histogram:

```
>  hist(rnorm(100))
```



The Sweave commands

Now you’ve learned it! You can start to write your homework using Sweave.

The only other things you need to know are the commands (within <<>>), and there aren’t many. Here are the most common ones:

- **echo**: logical (default is TRUE). Indicating whether to Include R code in the output file.
- **eval**: logical (default is TRUE). Indicating whether to evaluate (run) the code and include the results in the output.
- **fig**: logical (default is FALSE), indicating whether to include figures produced from the code.

For others, read manual.

What is an R vignette

- A vignette is an optional, supplemental documentation for an R package. Note that the R reference manual is merely a list of function helps.
- Not all packages have vignette. Bioconductor packages are required to have vignette.
  - `vignette(all = TRUE)` lists of functions with vignette.
- For those packages with vignette, the vignette can be accessed by vignette function:
  - `vignette("Sweave")` opens the vignette (a pdf) in a separate window for package “Sweave”.

Build an R package with a vignette

- Vignette is usually created using Sweave.
- The Sweave file (.Rnw) is distributed with the package under `/vignettes`.
- When building R package (e.g., running `R CMD build`), the Rnw file will be processed, and a pdf vignette will be created.
- The vignette can be accessed in R using “vignette” function.

Build an R package with a vignette in RStudio

- In RStudio, Build & Reload a package doesn’t compile the vignette.
- You have to Build Source Package, and then install it from source.
- We will practice this in the lab.

Review

We have covered R package today, including:

- Install package from existing repositories or source.
- Build a package.
- Write function helps for package.
- Write package manual (vignette) using Sweave.