



A blog about data analytics, R and eoda

Professioneller R Code – einfach eigene Pakete erstellen und dokumentieren

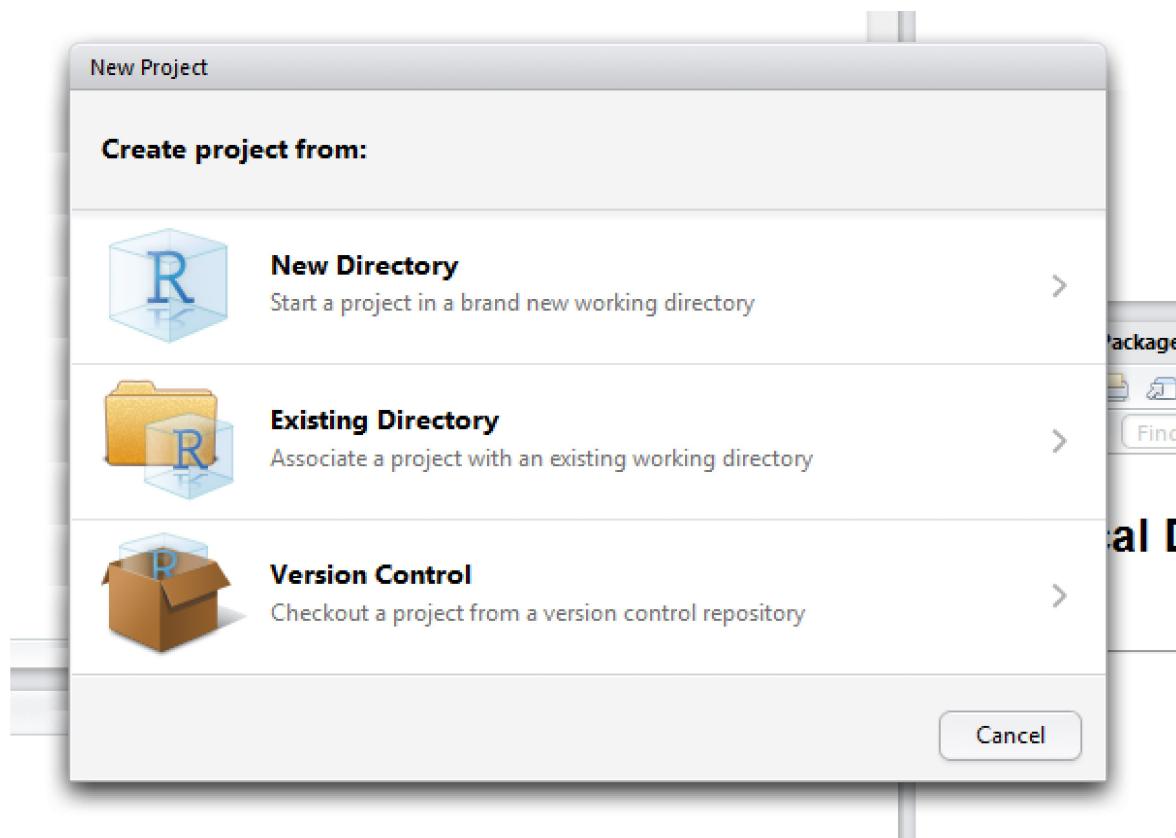
Selbst in kleineren R-Projekten fällt schnell eine Vielzahl selbst geschriebener Funktionen an. Der Ordnung halber möchte man deren Definition und Beschreibung möglichst aus dem eigentlichen Arbeitsskript heraushalten. Eine einfache Lösung hierfür ist es, eine Funktion in einem eigenen Skript auszulagern und dieses dann mit der `source()` Funktion im Arbeitsskript aufzurufen. Die so geladenen Funktionen tauchen nun als Objekte im aktuellen Workspace auf. Hat man eine Vielzahl von Funktionen definiert leidet jedoch schnell die Übersichtlichkeit.

Ein eleganterer und besserer Weg sind eigene Pakete. Die Funktionsdefinitionen werden in eine eigene Umgebung geladen und erscheinen nicht direkt im Workspace. Pakete bieten zudem die Möglichkeit einer einheitlichen Dokumentation, welche besonders hilfreich ist wenn man die entwickelten Funktionen weitergeben oder sie selbst nach einem größeren Zeitabstand wiederverwenden möchte. Jeder R Nutzer ist an die typische R-Hilfe gewöhnt und weiß genau wo er nach den für ihn relevanten Informationen suchen muss.

Eigene R Pakete mit RStudio

Gerade in [RStudio](#) ist der Prozess der Paketerstellung dank der implementierten Unterstützung zur Paket Entwicklung überraschend einfach umzusetzen.

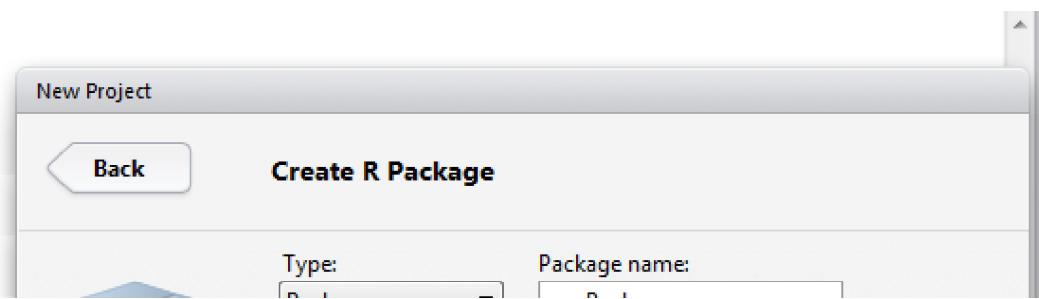
Hierzu wählt man unter File „New Project“ und in den darauf folgenden Pop-Ups „New Directory“ und „R Package“.

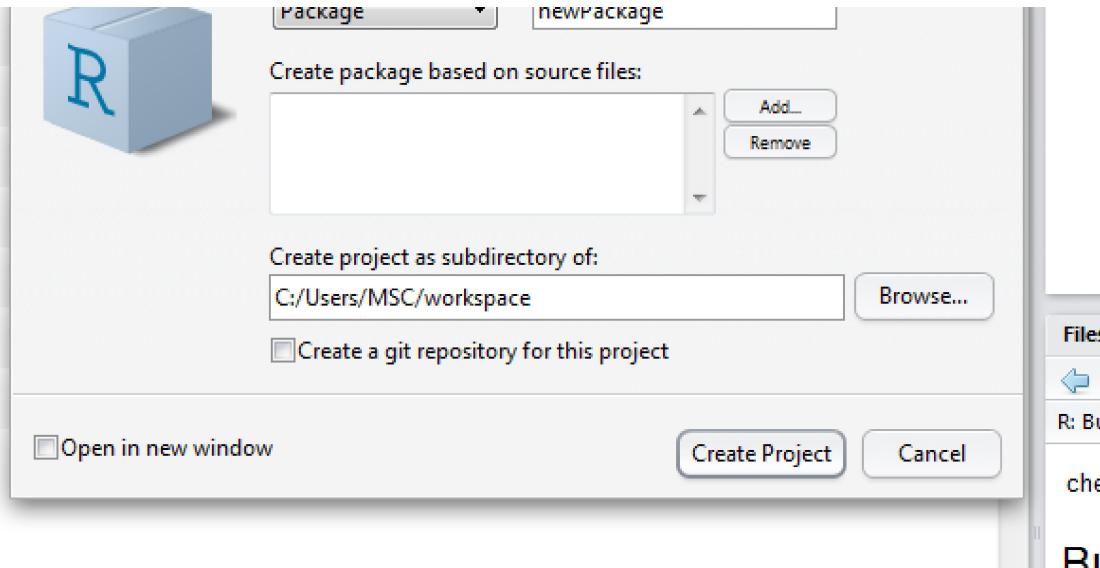


Einstieg in die Paketerstellung mit RStudio

Im nun folgenden Fenster vergibt man schließlich den Paketnamen, bindet eventuell schon vorhandene Skripte ein und gibt den Dateipfad an, in dem das Paket abgelegt werden soll.

Der Vorgang wird mit **Create Project** abgeschlossen.





Abschluss des Vorgangs der R-Paketerstellung

RStudio legt nun automatisch eine Verzeichnisstruktur an. Mit den Ordnern **man** für die Dokumentationsdateien und **R** für die verwendeten R-Skripte. Des Weiteren u.a. die Dateien **DESCRIPTION**, **NAMESPACE** und **newPackage.Rproj**.

man	19.11.2013 13:43	Dateiordner
R	19.11.2013 13:35	Dateiordner
.Rbuildignore	19.11.2013 11:28	RBUILDIGNORE-D... 1 KB
DESCRIPTION	20.11.2013 14:44	Datei 1 KB
NAMESPACE	20.11.2013 16:46	Datei 1 KB
newPackage.Rproj	19.11.2013 11:28	R Project 1 KB
Read-and-delete-me	19.11.2013 11:28	Datei 1 KB

Überblick über die von RStudio angelegte Verzeichnisstruktur

In der **DESCRIPTION** Datei werden Meta-Informationen über das Paket abgetragen. Wie z.B der Autor des Paketes, der „Maintainer“ (für die Pflege des Paketes zuständige Person) einschließlich seiner Email Adresse, einer allgemeinen Beschreibung des Paketes und unter welcher Lizenz das Paket veröffentlicht wird. Für eine Veröffentlichung des Paketes über CRAN muss diese Datei entsprechend ausgefüllt werden. Es ist zu beachten, dass der Name des Paketes nur ASCII Buchstaben, Zahlen und Punkte enthalten darf. Darüber hinaus muss ein Paketname mindestens 2 Zeichen lang sein, muss mit einem Buchstaben beginnen

und darf nicht mit einem Punkt enden. Die genauen Regeln findet man in dem „[Writing R Extensions](#)“ Manual von CRAN.

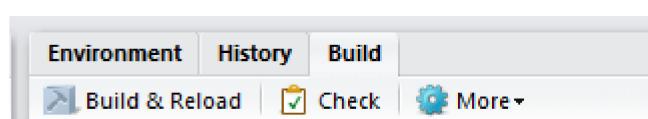
Unter **NAMESPACE** wird der Export von Funktionen geregelt. Der Export legt fest welche Funktionen des neuen Paketes von den Nutzern selbst benutzt werden können und welche Funktionen nur implizit durch andere Funktionen aufgerufen werden. Unter Import kann darüber hinaus definiert werden welche Funktionen aus anderen Paketen verwendet werden. Die **NAMESPACE** Datei stellt zudem sicher, dass im Falle von doppelt vergebenen Funktionsnamen in unterschiedlichen Paketen die richtigen Funktionen aufgerufen werden. Die Datei muss nicht manuell ausgefüllt werden, sondern lässt sich mit Hilfe der Pakete [roxygen2](#) und [devtools](#) direkt aus R erstellen.

Die **.Rproj** Datei ist eine von RStudio zum Projekt Management verwendete Datei.

Außerhalb von RStudio kann man diese Basis Grundstruktur natürlich auch manuell erstellen oder man nutzt die `create()` Funktion aus dem `devtools` Package von Hadley Wickham. Eventuell muss noch die **NAMESPACE** Datei manuell hinzugefügt werden und der Autor in der **DESCRIPTION** Datei angepasst.

Funktionen können nun wie gewohnt entwickelt werden. Die entstandenen .R Dateien speichert man dann unter dem R Ordner. Nach individueller Vorliebe kann man die Funktionen des Paketes alle in einer Datei definieren oder, die empfehlenswertere Variante, die Funktionen auf mehrere Skripte aufteilen.

Die fertigen Funktionen werden dann in einem Paket gebündelt. RStudio hat dafür eine Funktion direkt in seiner GUI implementiert, die sich unter **Build: Build & Reload** betätigen lässt.



==> **Rcmd.exe INSTALL --no-multiarch --w**

Bündelung der Funktionen in einem Paket mit der
RStudio Funktion Build: Build & Reload

Das Paket wird nun zusammengestellt und zu den bereits vorhandenen Paketen in der Bibliothek hinzugefügt. Hieraus kann es wie gewohnt mit *library()* aufgerufen werden.

Außerhalb von RStudio würde man das Paket mit *build()* aus *devtools* erstellen. Es wird eine tar.gz Datei erstellt und diese dann mit *install()* installiert.

Merkmal eines guten R-Paketes ist eine aussagekräftige Dokumentation. Mit Hilfe des *roxygen2* Pakets lässt sich diese nahezu automatisch erstellen

Paket Dokumentation mit roxygen2

Möchte man ein selbst erstelltes R Paket anderen Personen zur Verfügung stellen oder es auch selbst über einen längeren Zeitraum nutzen, ist eine gute Dokumentation essentiell. Im Folgenden wird beschrieben wie sich mit Hilfe des *roxygen2* Paketes aus R heraus bequem eine Dokumentation erstellen lässt.

Die Paketdokumentation in R ist in **Rd** Dateien („R documentation“) angelegt, wobei jede Funktion ihre eigene **Rd** Datei erhält. Diese Dateien können entweder manuell in einem Texteditor oder automatisiert mit dem *roxygen2* Paket erstellt werden. Für eine Veröffentlichung auf CRAN sind dabei entsprechende Vorschriften zu beachten.

Das *roxygen2* Paket orientiert sich an diesen Vorschriften. Die grundlegende Funktionsweise ist es in einem R-Skript vor der zu dokumentierenden Funktion besonders formatierte Kommentare zu schreiben. Mit dem Aufruf der *document()* Funktion aus dem *devtools* Paket parst *roxygen2* diese Kommentare und legt entsprechende **Rd**

Dateien an. Ein *roxygen* Kommentar wird stets mit `#‘` eingeleitet. In der ersten Zeile wird der Titel der Dokumentation eingetragen. Dies sollte nicht der Name der Funktion sein sondern vielmehr eine knappe Beschreibung der Funktionalität. Nach einer Zeile mit einem simplen *roxygen* Kommentar Zeichen `#‘`, folgt eine genauere Beschreibung der Funktionsaufgabe. Über die Möglichkeit hinaus einem vorgebendem Ablauf zu Folgen gibt es auch die Möglichkeit die einzelnen Dokumentationselemente zu definieren. Eine solche Definition leitet man mit einem `@` ein, gefolgt von dem entsprechendem Element.

Wichtig: Die von *roxygen* verwendeten Element Bezeichnungen sind nicht zwingend Deckungsgleich mit ihrem Gegenpart in der eigentlichen R Dokumentation. So wird beispielsweise die Dokumentation von einem Argument nicht mit `@Arguments` eingeleitet, sondern mit `@param`.

Diese Elemente werden als **roclets** bezeichnet. *roxygen* unterscheidet dabei zwischen **Rd Roclets** (Sie werden zur Erstellung der Dokumentationsdateien verwendet), **Namespace Roclets** wie z.B. `#‘@export` (Mit ihrer Hilfe wird automatisiert die Namespace Datei erstellt) und **Collate Roclets** (Sie ermöglichen die Erstellung des Collate Feldes in der Description Datei, welches die Reihenfolge definiert in der die Paket Funktionen geladen werden).

Eine Argument Definition wird also mit `@param` eingeleitet, darauf folgen der Name und eine Beschreibung des Argumentes. Diese sollte das Argument erklären und die an sie gestellten Anforderungen beschreiben.

Möchte man seine Dokumentation abschließen ruft man die `document()` Funktion auf. Diese erhält als Input den Ober-Pfad des R Paketes. Die Funktion erstellt dann neue bzw. aktualisierte **Rd** Dateien. In RStudio lässt sich nun wieder mit **Build & Reload** das Paket neu zusammenstellen. Die neu erstellten Hilfefunktionen können dabei wie gewohnt aus R heraus aufgerufen werden.

roxygen Code

```
#' Multiplies two values
#'
#' This Functions takes 2 values and multiplies them with each other
#'
#' @param x a numeric value
#' @param y a numeric value
#'
#' @details
#' This is a function only created as an Example for the Process of Package Development
#'
#' @export
#'
#' @examples
#' newFunction(2,4)
#'
newFunction <- function(x,y){
  return(x*y)
}
```

|

roxygen Code

Resultierende Hilfeseite

The screenshot shows a web-based R documentation page. At the top, there's a header bar with a dropdown menu "R: Multiplies two values" and a "Find in Topic" search bar. The main content area has a title "newFunction {newPackage}" and a "R Documentation" link. Below the title is the function name "Multiplies two values". The "Description" section contains the text "This Functions takes 2 Values and multiplies them with each other". The "Usage" section shows the code "newFunction(x, y)". The "Arguments" section lists "x a numeric value" and "y a numeric value". The "Details" section contains the note "This is a function only created as an Example for the Process of Package Development". The "Examples" section shows the example "newFunction(2, 4)". At the bottom of the page is a footer with the text "[Package newPackage version 1.0 Index]".

Resultierende Hilfeseite

Für eine eventuelle Veröffentlichung des Paketes auf CRAN bietet devtools die Funktion `check()`, welche testet ob das Paket die CRAN Anforderungen erfüllt.

Weiterführende Lesetipps:

[Advanced R programming](#) von Hadley Wickham

[Writing R Extensions](#) von dem R Core Team

[Developing Packages with RStudio](#) von dem RStudio Team

Die R-Akademie bietet zu diesem Thema einen eigenen Kurs an :

[Paketerstellung in R](#)



Martin Schneider / 11. Dezember 2013 / eoda news / Big Data, R-Pakete

Ein Gedanke zu „Professioneller R Code – einfach eigene Pakete erstellen und dokumentieren“

Pingback: [Neuigkeiten zu eoda, R und Datenanalyse | eoda, R und Datenanalyse](#)

/ Stolz präsentiert von WordPress