Fundamentals of Creating R Package

Riaz Khan, MS, Statistics, SDSU

02/02/2020

Table of Contents

[System preparation 2](#_Toc31660989)

[Create library 2](#_Toc31660990)

[Package naming guidelines 3](#_Toc31660991)

[Package configuration 3](#_Toc31660992)

[Build and check package 4](#_Toc31660993)

[Package metadata 5](#_Toc31660994)

[Version 6](#_Toc31660995)

[Multiple authors 6](#_Toc31660996)

[License 7](#_Toc31660997)

[Dependencies 7](#_Toc31660998)

[Package documentation 8](#_Toc31660999)

[Writing own function 9](#_Toc31661000)

[straight line 101 10](#_Toc31661001)

[Function straighten 10](#_Toc31661002)

[Coding style: best practices 12](#_Toc31661003)

[S3 method: print for class stline 13](#_Toc31661004)

[External data 13](#_Toc31661005)

[Dependency 14](#_Toc31661006)

[Package vignette 15](#_Toc31661007)

[Package testing 16](#_Toc31661008)

[Test Exmaple 16](#_Toc31661009)

[Package release 18](#_Toc31661010)

[README file 18](#_Toc31661011)

[NEWS file 18](#_Toc31661012)

[Release 19](#_Toc31661013)

# System preparation

* Install the latest version of R and RStudio:
  + R (3.6.2): <https://cran.r-project.org/>
  + RStudio (1.2.5033): <https://rstudio.com/products/rstudio/download/#download>
* Install the following packages:
  + devtools
  + roxygen2
  + testthat
  + knitr

install.packages("devtools")  
install.packages("roxygen2")  
install.packages("testthat")  
install.packages("knitr")

* Check if the system is ready for package building:

devtools::has\_devel()

If following message is shown, it suggests that the system is ready for package development. The C compiler installation part can be skipped.

Your system is ready to build packages!

Depending on the system, a C compiler may or may not be required. If required:

* For windows, install Rtools <https://cran.r-project.org/bin/windows/Rtools/>
* For Mac, get Xcode (available for free in the App Store)

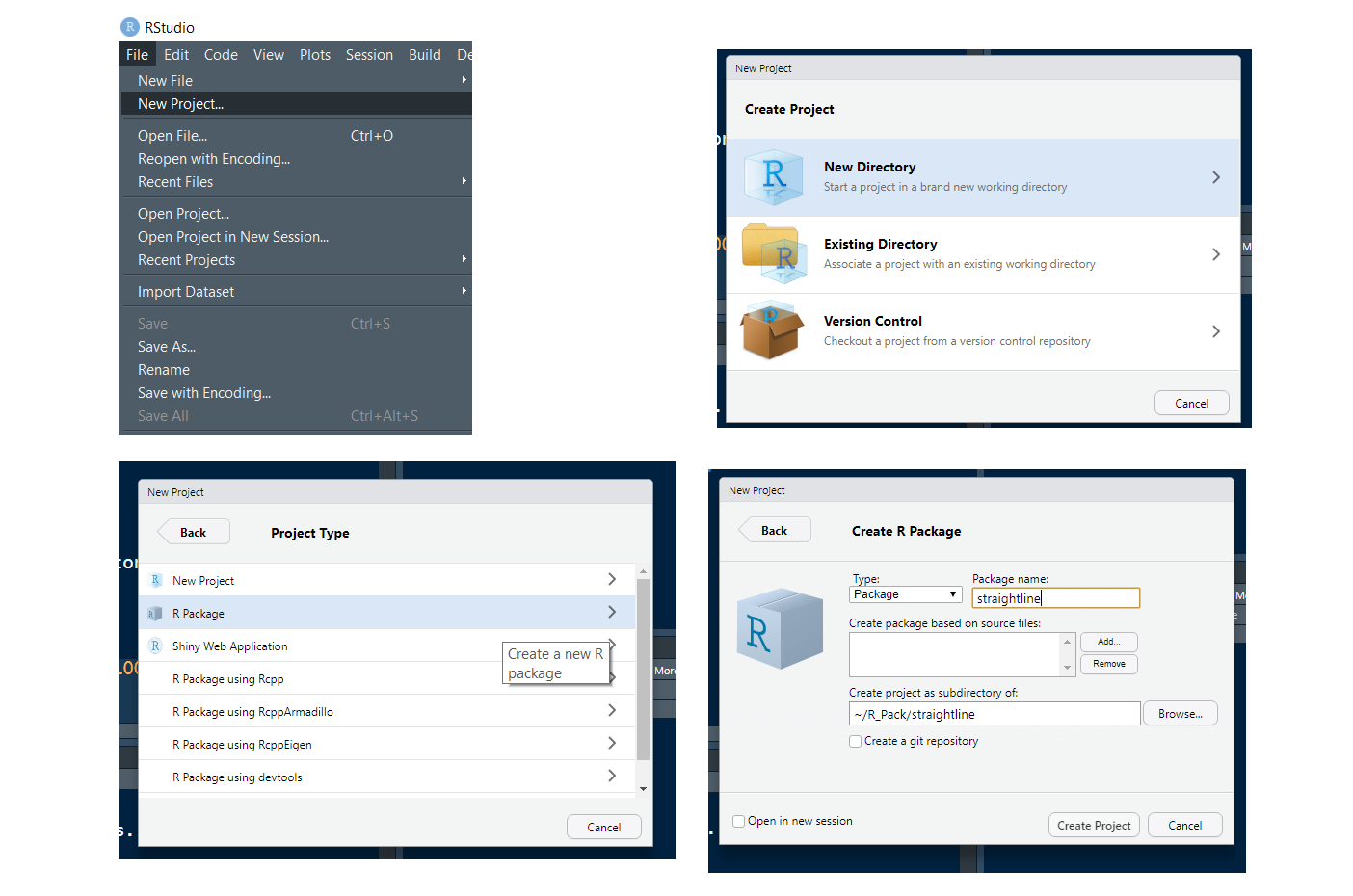
Optional: Get the updated version of devtools from GitHub by running devtools::install\_github("r-lib/devtools", force = TRUE)

# Create library

To create a new package, open RStudio

* File >> New Project >> New Directory >> R Package

Give the name of the package straightline

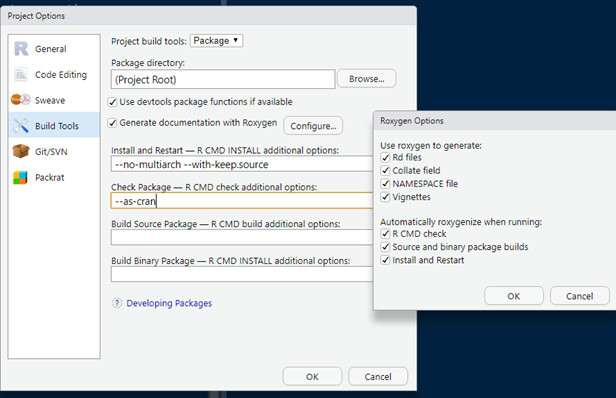


## Package naming guidelines

* Unique name
* Matcehs with the type of problem the library deals with
  + Easy to google
  + Example neuralnet, NLP
* Case consistency
  + Same case preferred
  + Mixing not uncommom (randomForest)
* It is not uncommon to see an extra r in the name
  + Example: stringr, knitr
  + straightliner? Instead of straightline
* Formal requirements for an R package:
  + Contains only alphabetic letter, number and period
  + Cannot end with a period
  + Can start with only letter

# Package configuration

To let roxygen to some heavylifting, go to Buid > Configure Build Tools and configure as follows:

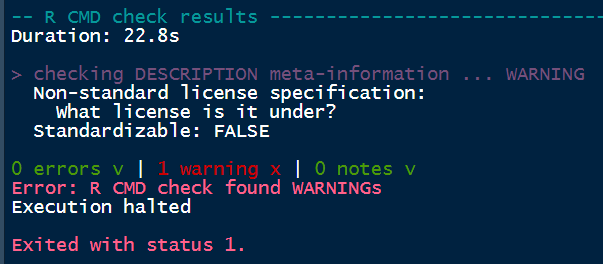


# Build and check package

RStudio includes a hello function with documentation in a default package when a package is created. To build the package, Build > Install and Restart (Cntl + Shift + B). This will install the package in the system and will be included in R session.

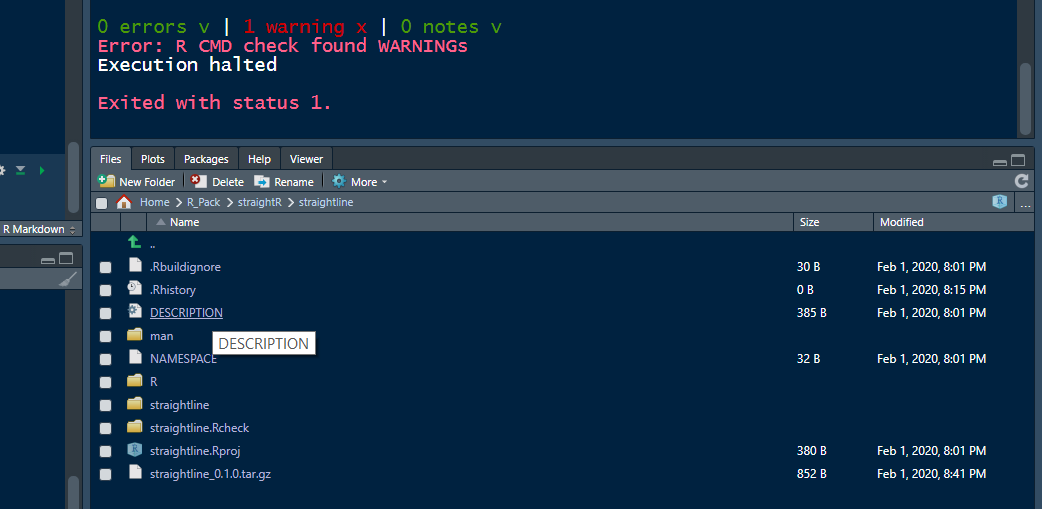
After building, check the package, Build > Check Package (Cntl + Shift + E). This command will check the package using best practices.

If everything left unaltered, the check should return with a warning for non-standard license.



# Package metadata

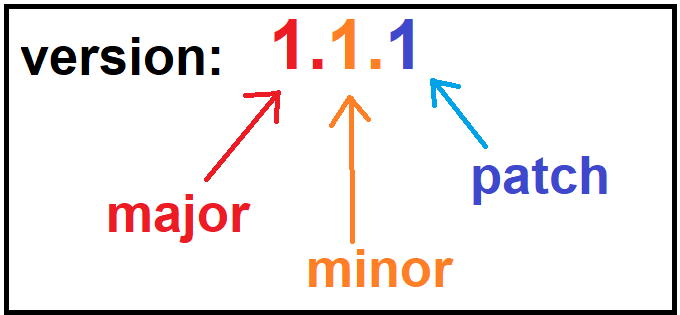
* Every package must have a description
* There will be an auto-generated DESCRIPTION file
* Using the file explorer-plot window of RStudio, go to the package folder and double click the DESCRIPTION file. This will open the file in RStudio editor window.



Edit Title, Version, Author, Maintainer, Description and License in the DESCRIPTION file. An example:

Package: straightline  
Type: Package  
Title: An Example R Package for Straight Line in 2d Coordinate System  
Version: 1.1.1  
Author: Riaz Khan  
Maintainer: Riaz Khan <rk@sdsu.net>  
Description: This package will do basic calculations of a straight   
 line in 2D coordinate system. This package will do basic   
 calculations of a straight line in 2D coordinate system. This   
 package will do basic calculations of a straight line in 2D   
 coordinate system.   
License: GPL-3  
Encoding: UTF-8  
LazyData: true  
RoxygenNote: 7.0.2

## Version



## Multiple authors

If there are multiple authors/contributors, then Authors@R field can be used:

Package: straightline  
Type: Package  
Title: An Example R Package for Straight Line in 2d Coordinate System  
Version: 1.1.1  
Authors@R: c(person("Riaz", "Khan",   
 email = "rk@sdsu.net",   
 role = c("aut", "cre")),  
 person("Second", "Author",  
 email = "se@sdsu.net",   
 role = "aut" ))  
Description: This package will do basic calculations of a straight   
 line in 2D coordinate system. This package will do basic   
 calculations of a straight line in 2D coordinate system. This   
 package will do basic calculations of a straight line in 2D   
 coordinate system.   
License: GPL-3  
Encoding: UTF-8  
LazyData: true  
RoxygenNote: 7.0.2

Following roles can be added in the Authors@R filed:

* cre: maintainer
* aut: author having significant contributions
* ctb: contributors with smaller contributions
* cph: copyright holder, in case where the copyright holder is other than the author (e.g employer of the author)

## License

There are three that can be used in an R package:

* MIT: Simple permissive. Anyone can do anything as long as the original copyright and licence notice included. See <https://tldrlegal.com/license/mit-license>
* GPL-2, GPL-3: If the code is distributed in a bundle, then the whole bundle must be in GPL license. In case of code modification, the source code must be available. GPL-3 is little stricter than GPL-2. See <https://tldrlegal.com/license/gnu-general-public-license-v2>, <https://tldrlegal.com/license/gnu-general-public-license-v3-(gpl-3)>

## Dependencies

Frequently, a new package uses the functionality of other packages. For this package dependency, Imports field must be used. If a package is used, but not required (likely for examples), then those should be declared as Suggests. For example:

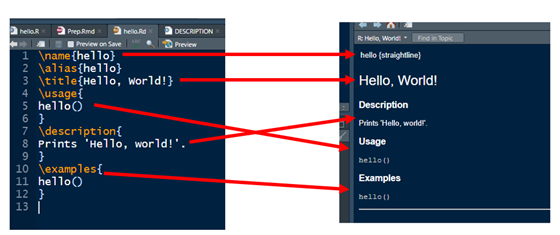
Package: straightline  
Type: Package  
Title: An Example R Package for Straight Line in 2d Coordinate System  
Version: 1.1.1  
Authors@R: c(person("Riaz", "Khan",   
 email = "rk@sdsu.net",   
 role = c("aut", "cre")),  
 person("Second", "Author",  
 email = "se@sdsu.net",   
 role = "aut" ))  
Description: This package will do basic calculations of a straight   
 line in 2D coordinate system. This package will do basic   
 calculations of a straight line in 2D coordinate system. This   
 package will do basic calculations of a straight line in 2D   
 coordinate system.   
Imports: ggplot2  
Suggests: MASS  
License: GPL-3  
Encoding: UTF-8  
LazyData: true  
RoxygenNote: 7.0.2

## Package documentation

So far, we have only one function in our package, called hello. Let us see the documentation of this function.

?hello

A description, usage and example is given in the function Help section. This documentation is creted by the default Rd file. Go to ~/man/hello.Rd to see the actual codes that generated this documentation.



**Note**:

* Documentation and R files live in separate folders
  + ~/man and ~/R folders respectively
* R and Rd documents need to be written separately
* Roxygen– codes and documentation simultaneously
* Roxygen auto generates the Rd files
  + When we configured at the beginning, we told roxygen to handle the Rd files along with other things

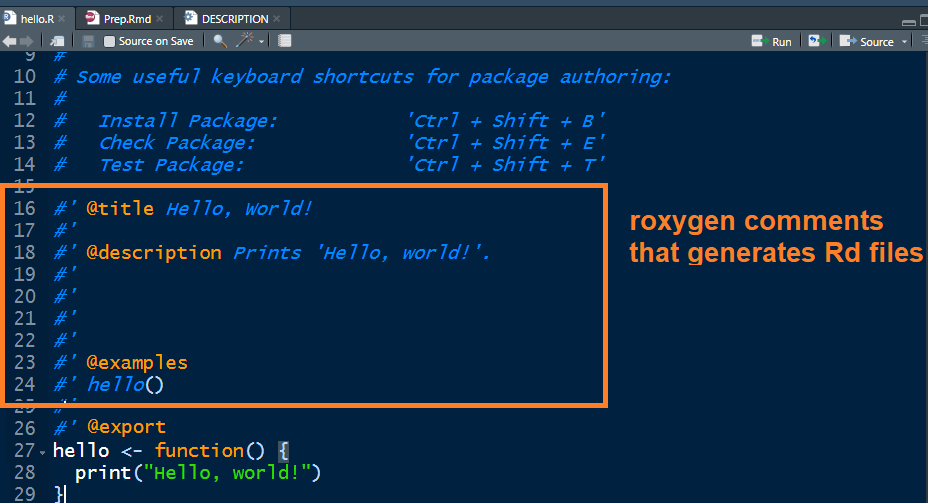
Go to the manual directory and delete the hello.Rd file. Then build the package (ctrl+Shift+B) and run the following again.

?hello

Now there is no documentation for the hello function. Now we will try to re-create the documentation.

* Go to the hello.R, put the cursor on hello where the function is defined. Then click Code > Insert Roxygen Skeleton. This will create some roxygen style field for function documentation. These fields can be used for function documentation.

**Note**: The title can be documented using the @title tag.



Now, build the package again. Examine the Rd file that is genrated by roxygen. Now, documentation will be available for the hello function.

**Note**: It is recommended that we check the package periodically (ctrl+Shift+E). If checking gives warning that the NAMESPACE will not be overwritten, then delete the NAMESPACE and build+check again. Roxygen generates the NAMESPACE and the Rd files each time we build package.

# Writing own function

Functions are main drivers of specific tasks that the package is intended for. Now we will make our own function and document it.

We will write a function (let us call it straighten) that takes three inputs , , and , which are the parameters of a straight line of form in 2D coordinate system. The function will throw error message if and both are zero. We want the function to return the followings:

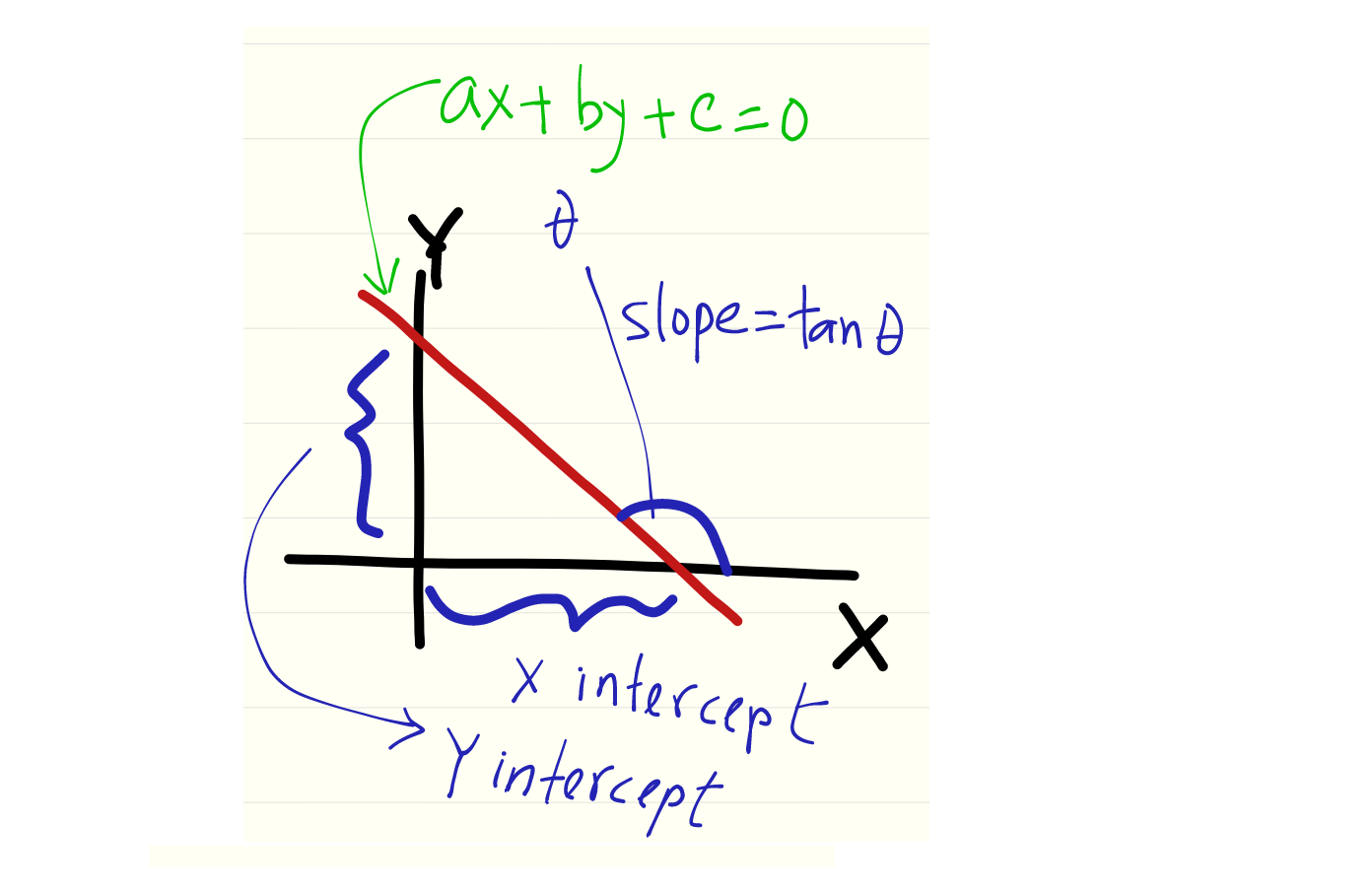
* the value of a
* the value of b
* the value of c
* the slope of the straight line
* x intercept of the straight line
* y intercept of the straight line
* the angle between x axis and the straight line in degrees

Additionally, we want the return object of the function of its own class, stline

## straight line 101

In 2D coordinate system, if a straight line has the form of , then:

* line parallel to axis, if
* line parallel to axis, if
* passes through the origin, if



## Function straighten

* Function in ~/R folder
* file: straighten.R

straighten <- function(a, b, c){  
 if(a==0 & b==0){  
 # error message  
 stop("a and b both cannot be zero")  
 }  
 # calculations  
 slope <- - a / b  
 x\_intercept <- - c / a  
 y\_intercept <- - c / b  
 angle\_x <- atan(slope) \* (180 / pi)  
 # outputs in a list  
 value <- as.list(c(a = a, b = b, c = c,  
 slope = slope,  
 x\_intercept = x\_intercept,  
 y\_intercept = y\_intercept,  
 angle\_x = angle\_x))  
 # output class  
 class(value) <- "stline"  
 # return output  
 return(value)  
}

Please add roxygen comments to document the function.





## Coding style: best practices

* Lower case recommended for function and object, with efficient use of underscore (\_)
* There should be spaces before and after all infix operators (e.g: +, -, \*, /, =, ==, <-, ->)
  + Space after comma, not before
* Use of <- for assignment (not =)
* Spelling out TRUE and FALSE, as opposed to just using T and F
* Opening curly brace followed by a new line
* Closing curly brace gets its own line
  + Exception: use of else
* Limited number of characters per line
* Use of unique name to avoid conflicts
* Proper indentations for easy understanding

For indentation, select the code block, then Code > Reindent Lines

# S3 method: print for class stline

It is recommended that print method can be applied to an object of user defined class. Here we will define the print function applicable for stline object.

#'   
#' @title Print \code{"stline"} Object  
#'  
#' @param object An object of class \code{"stline"}  
#'  
#' @param ... NULL; supplied for S3 method consistency  
#'  
#' @examples  
#' line <- straighten(1, 0, 5)  
#' print(line)  
#'  
#' line2 <- straighten(7, -5, 0)  
#' print(line2)  
#'  
#' @method print stline  
#'  
#' @export  
print.stline <- function(x, ... = NULL){  
 char\_a <- paste0("(", as.character(round(x$a, 3)), ")", "\*x")  
 char\_b <- paste0("(", as.character(round(x$b, 3)), ")", "\*y")  
 char\_c <- paste0("(", as.character(round(x$c, 3)), ")")  
 Equation <- paste(char\_a, "+", char\_b, "+", char\_c)  
 message("Straight Line Equation : ", Equation)  
 message("Slope : ", x$slope)  
 message("x intercept : ", x$x\_intercept)  
 message("y intercept : ", x$y\_intercept)  
 message("Angle with x axis (degrees) : ", x$angle\_x)  
 if(x$a == 0){  
 message("Line parallel to x axis")  
 }  
 if(x$b == 0){  
 message("Line parallel to y axis")  
 }  
 if(x$c == 0){  
 message("Line passes through origin")  
 }  
}

# External data

* External data lives in the ~/data folder, with extension .RData
* Data can be saved in .RData format using save
* Example

set.seed(10)  
a <- runif(100, 0, 100)  
set.seed(20)  
b <- runif(100, 0, 100)  
set.seed(30)  
c <- runif(100, 0, 100)  
example\_data <- data.frame(cbind(a = a, b = b, c = c))  
save(example\_data, file = "example\_data.RData")

* Create a folder called data and keep a copy of the .RData file in it
* Create documentation of the data

#' @title \code{"example\_data"}: An Example Data  
#'  
#' @description A data containing information about 900 borrowers. It is a  
#' modified version of publicly available real data.  
#'  
#'  
#'  
#' @format A data frame with 100 rows and 3 variables:  
#' \describe{  
#' \item{a}{A series of parameter \eqn{a}}  
#' \item{b}{A series of parameter \eqn{b}}  
#' \item{c}{A series of parameter \eqn{c}}  
#' }  
#'  
#' @examples  
#' data("example\_data")  
#' plot(example\_data$a~example\_data$b)  
#' grid()  
#'  
#'  
#' @source \url{https://github.com/riazakhan94/straightline}  
"example\_data"

**Note**: Adding .RData may require to add dependency. If so, edits need to be maded in the DESCRIPTION file (Depends: R (>= 3.5.0)).

# Dependency

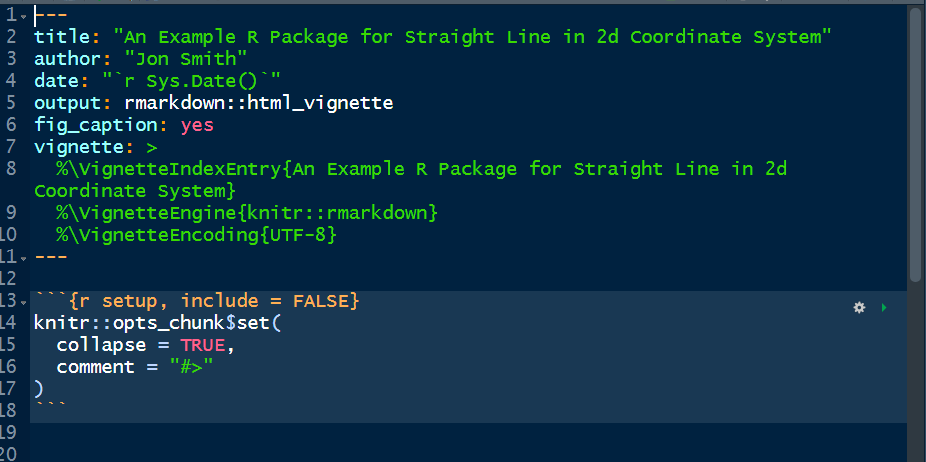
#' @title Dependency Example  
#'  
#' @param x A numeric data series  
#'  
#' @description This is an exmple to show the use of dependencies  
#'  
#'  
#' @examples  
#' df <- MASS::Boston  
#' age <- df$age  
#' example\_dependency(age)  
#'  
#'  
#'  
#' @export  
example\_dependency <- function(x){  
 ggplot2::qplot(x, geom = "histogram", binwidth = 10)  
}

# Package vignette

* Inclusion of package vignette recommended.
* Long guide of the package
  + What is does
  + How to use
* Markdown file
* In the vignettes folder
* To create:
  + create a folder called vignette and open an Rmd file (for example, stline.Rmd)
  + add knitr and rmarkdown to the Suggests field in DESCRIPTION
  + add VignetteBuilder field in the DESCRIPTION

Suggests: MASS, knitr, rmarkdown  
VignetteBuilder: knitr

Add the following at the top of the vignette, before starting the document:



[R Markdown Cheat Sheet](https://rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf)

# Package testing

Authomated tests are useful for robust code, provides all the tests in an organized way. To set up:

* Make a ~/tests/testthat folder
* Add testthat to the Suggests field of DESCRIPTION
* Make a ~/tests/testthat.R file

A test lives in the ~/tests/testthat directory. All files must start with test. Tests in ~/tests/testthat.R file are checked during the package check.

## Test Exmaple

* Create ~/tests/testthat/test\_classtype.R file:

# file: test\_classtype.R  
context("Class of function straighten")  
library(straightline)  
  
  
myline <- straighten(pi, exp(1), log(7))  
  
test\_that("Class of the function return is stline", {  
 expect\_equal(class(myline), "stline")  
})  
  
  
# direct access  
test\_that("Class of the function return is stline", {  
 expect\_equal(class(straighten(1,1,0)), "stline")  
 expect\_equal(class(straighten(2,15,7)), "stline")  
 expect\_equal(class(straighten(0,1,5)), "stline")  
})

* Create ~/tests/testthat/test\_anlge.R file:

# file: test\_anlge.R  
context("Angle of a straight line")  
library(straightline)  
  
test\_that("Angle of a straight line 45/-45", {  
 expect\_equal(straighten(1, -1, 0)$angle\_x, 45)  
 expect\_equal(straighten(1, 1, 0)$angle\_x, -45)  
})  
  
  
  
test\_that("Angle of a straight line 60/-60", {  
 expect\_equal(straighten(1, 1/sqrt(3), 0)$angle\_x, -60)  
 expect\_equal(straighten(1, -1/sqrt(3), 0)$angle\_x, 60)  
})  
  
  
test\_that("Angle of a straight line 30/-30", {  
 expect\_equal(straighten(1, sqrt(3), 0)$angle\_x, -30)  
 expect\_equal(straighten(1, -sqrt(3), 0)$angle\_x, 30)  
})

* Create ~/tests/testthat/test\_error.R file:

# file: test\_error.R  
  
context("Error message if both a = b = 0")  
library(straightline)  
  
test\_that("a, b both cannot be zero", {  
 expect\_error(straighten(0, 0, 3), "a and b both cannot be zero")  
})

* More test functions:
  + expect\_length
  + expect\_message
  + expect\_match
  + expect\_null
  + See [testthat documentation](https://cran.r-project.org/web/packages/testthat/testthat.pdf)
* Add all the tests in ~/tests/testthat.R file:
  + the library needs to added once for all the tests

After writing all the tests in ~/tests/testthat directory, run all the tests by running devtools::test(), or use Ctrl+Shit+T.

Check the package (Ctrl+Shift+E)

# Package release

If checking returns 0 errors v | 0 warnings v | 0 notes then we are ready for submission

## README file

* README is usually aimed for the new users.
* README should contain some high level information about the package and how to use the package.
* README has extension .md.
* Possible to use R Markdown (.Rmd) to create the .md file.
* Example using .Rmd
  + Copy the contents from the [README.Rmd](https://github.com/riazakhan94/straightline) file and run. It will create the README.md file. Include the README.md file in the package directory.

## NEWS file

* Aimed for the old users.
* Brief description of changes made in the new version
* An example NEWS.md for a new package

straightline 1.1.1  
=====  
  
Author  
=======  
  
Riaz Khan `rk@sdsu.net`  
Second Autor `se@sdsu.net`  
  
  
Maintainer  
===============  
Riaz Khan `rk@sdsu.net`

## Release

* Run devtools::release()
* Iterative process

Go through all the process.

Congratulations! :)

*Reference:* [R packages by Hadley Wickham](http://r-pkgs.had.co.nz/)