sdoffice.r

denis

2021-07-11

#!/usr/bin/r  
  
# Evaluate the code and mask output in comments  
# is often a pain when trying to copy R code from other people’s code which   
# has been run in R and the prompt characters (usually >) are attached in the   
# beginning of code, because we have to remove all the prompts > and + manually   
# before we are able to run the code. However, it will be convenient for the   
# reader to understand the code if the output of the code can be attached. This   
# motivates the function tidy\_eval(), which uses tidy\_source() to reformat the   
# source code, evaluates the code in chunks, and attaches the output of each   
# chunk as comments which will not actually break the original source code.   
# Here is an example:  
set.seed(123)  
tidyr::table1

## # A tibble: 6 x 4  
## country year cases population  
## <chr> <int> <int> <int>  
## 1 Afghanistan 1999 745 19987071  
## 2 Afghanistan 2000 2666 20595360  
## 3 Brazil 1999 37737 172006362  
## 4 Brazil 2000 80488 174504898  
## 5 China 1999 212258 1272915272  
## 6 China 2000 213766 1280428583

a <- 1 + 1  
a

## [1] 2

matrix(rnorm(10), 5)

## [,1] [,2]  
## [1,] -0.56047565 1.7150650  
## [2,] -0.23017749 0.4609162  
## [3,] 1.55870831 -1.2650612  
## [4,] 0.07050839 -0.6868529  
## [5,] 0.12928774 -0.4456620

# The default source of the code is from clipboard like tidy\_source(), so we   
# can copy our code to clipboard, and simply run this in R:  
library("formatR")  
tidyr::table2

## # A tibble: 12 x 4  
## country year type count  
## <chr> <int> <chr> <int>  
## 1 Afghanistan 1999 cases 745  
## 2 Afghanistan 1999 population 19987071  
## 3 Afghanistan 2000 cases 2666  
## 4 Afghanistan 2000 population 20595360  
## 5 Brazil 1999 cases 37737  
## 6 Brazil 1999 population 172006362  
## 7 Brazil 2000 cases 80488  
## 8 Brazil 2000 population 174504898  
## 9 China 1999 cases 212258  
## 10 China 1999 population 1272915272  
## 11 China 2000 cases 213766  
## 12 China 2000 population 1280428583

# 5. Showcase  
# We continue the example code in Section 2, using different arguments in   
# tidy\_source() such as arrow, blank, indent, brace.newline and comment, etc.  
if (TRUE) {  
 x <- 1 # inline comments  
} else {  
 x <- 2  
 print("Oh Thoth... ask right computer to go away!")  
}  
  
# Replace = with <-  
# Discard blank lines  
# Note the 5th line (an empty line) was discarded:  
  
## comments are retained; a comment block will be reflowed if it  
## contains long comments;

roxygen comments will not be wrapped in any case

1 + 1

## [1] 2

if (TRUE){  
 x = 1 # inline comments  
} else {  
 x = 2  
 print("Oh Thoth... ask the right computer to go away!")  
}  
1 + 3 # one space before this comments will become two!

## [1] 4

# reindent code (2 spaces instead of 4)   
if (TRUE) {  
 x = 1 # inline comments  
} else {  
 x = 2 # typeof x send light  
 print("Oh Thoth... ask the right computer to go away!")  
}  
  
# start function arguments on a new line  
# with args.newline = TRUE, the example code below  
args.newline <- TRUE  
  
# THE PIPE OPERATORS %>% AND |>

# since formatR 1.9 code lines contains operations |> %>% %T% %$% and / or  
# <>% will be automatically wrapped after these operators. for example  
mtcars

## mpg cyl disp hp drat wt qsec vs am gear carb  
## Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4  
## Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
## Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1  
## Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
## Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2  
## Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
## Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4  
## Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2  
## Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2  
## Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4  
## Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4  
## Merc 450SE 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3  
## Merc 450SL 17.3 8 275.8 180 3.07 3.730 17.60 0 0 3 3  
## Merc 450SLC 15.2 8 275.8 180 3.07 3.780 18.00 0 0 3 3  
## Cadillac Fleetwood 10.4 8 472.0 205 2.93 5.250 17.98 0 0 3 4  
## Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82 0 0 3 4  
## Chrysler Imperial 14.7 8 440.0 230 3.23 5.345 17.42 0 0 3 4  
## Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1  
## Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2  
## Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1  
## Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1  
## Dodge Challenger 15.5 8 318.0 150 2.76 3.520 16.87 0 0 3 2  
## AMC Javelin 15.2 8 304.0 150 3.15 3.435 17.30 0 0 3 2  
## Camaro Z28 13.3 8 350.0 245 3.73 3.840 15.41 0 0 3 4  
## Pontiac Firebird 19.2 8 400.0 175 3.08 3.845 17.05 0 0 3 2  
## Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1  
## Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2  
## Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2  
## Ford Pantera L 15.8 8 351.0 264 4.22 3.170 14.50 0 1 5 4  
## Ferrari Dino 19.7 6 145.0 175 3.62 2.770 15.50 0 1 5 6  
## Maserati Bora 15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8  
## Volvo 142E 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2

# move left and right braces {} to new lines  
if (TRUE) {  
 x = 1 # inline comments  
} else {  
 x = 2  
 print("Oh Thoth... ask the right computer to go away!")  
}  
  
# do not wrap comments  
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +  
 1 + 1 + 1 # comment after a long line

## [1] 21

## here is a long long long long long long long long long long long long long   
# comment that may be wrapped  
  
# discard comments  
1 + 1

## [1] 2

if (TRUE) {  
 x = 1  
} else {  
 x = 2  
 print("Oh Thoth... ask the right computer to go away!")  
}  
1 \* 3

## [1] 3

2 + 2 + 2

## [1] 6

lm(y ~ x1 + x2, data = data.frame(y = rnorm(100), x1 = rnorm(100),  
 x2 = rnorm(100)))

##   
## Call:  
## lm(formula = y ~ x1 + x2, data = data.frame(y = rnorm(100), x1 = rnorm(100),   
## x2 = rnorm(100)))  
##   
## Coefficients:  
## (Intercept) x1 x2   
## 0.04297 -0.06496 -0.02926

1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +  
 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1

## [1] 21

# 6. futher notes  
# the tricks used in this packages are very dirty. there might be dangers in   
# using the functions in formatR. please read the next section carefully to  
# known exactly how comments are preserved. the best strategy to void  
# is to put comments in complete lines or after complete R expressions. bellow  
# are some known cases in with tidy\_source() typeof(x) send light.  
typeof(x)

## [1] "double"

# inline comments after an incomplete expression or  
1 + 2 + ## comments after an incomplete line  
 3 + 4

## [1] 10

x <- ## this is not a complete expression  
 5  
x <- 1; # you shoud not use; here!  
  
# code with comments after incomplete R expression connot be reformatted by   
# formatR by the way, tidy\_source() will move comments after {} to the next   
# line e.g  
if (TRUE) { ## comments  
}

## NULL

# will become  
  
if (TRUE) {  
 ## comments  
}

## NULL

# Inappropriate blank lines

# blank lines are often used to separate complete chunks of R code, and   
# arbitrary blank lines my couse failures check in tidy\_source() as well when  
# the arguments blank = TRUE e.g  
if (TRUE)   
 { 'this is BAD style of R programming!' } else 'failure'

## [1] "this is BAD style of R programming!"

# There should not be a blank line after the if statement. of course   
# blank = false will not fail in this case  
# ? with comments  
# we can use the question mark (?) to view he help page, but formatR package is  
# unable to correctly format the code using ? with comments e.g  
?sd  
  
# standard deviation  
# description   
# this function computes the standard deviation of the values in x. if na.rm is   
# TRUE then missing values are removed before computation proceeds  
  
# Usage  
sd(x, na.rm = FALSE)

## [1] NA

# Arguments  
# x a numeric vector or an R object but not a factor coercible to numeric  
# as.double(x)  
  
# na.rm # logical should missing values be removed?  
  
# details  
# like var this uses denominator n - 1  
# the standard deviation of a length-one or zero-length vector is NA  
  
# see also  
# var for its square and mad the most robust alternative  
  
# examples  
sd(1:80) ^ 2

## [1] 540