How to Read, Write, and Google for Success in Coding

Christopher Skovron and Jon Atwell
Northwestern University

June 20, 2018

Learning to do computational research involves learning to write and read different things than you are used to

You probably hear a lot about “clean” code

Most of that is bullshit and preening

- unless you are working in industry in a setting where things need to highly scale
- remember the point from Day One about human scale

In a year or five years, you will look back on your past self's code and be horrified

Pinned Tweet

Sarah Bouchat @sarahbouchat · 27 Feb 2017

me to 6-months-ago me

2 5

Set yourself realistic goals in coding

- Acquire data
- Clean it in (tidy) forms ready for analysis
- Analyze
- Visualize and communicate

These goals animate today’s lesson, on how to teach yourself, and tomorrow’s lesson on “getting organized”

- These goals require lots of practice. Halfway between muscle memory and an art form

Reading and writing documents related to coding

- You will be reading new kinds of documentation, vignettes, examples, error messages
- These will seem confusing at first
- There are strategies, but nothing can beat practice

Your goals in writing code are more modest

- Does this do what I need it to do in a reasonably computationally efficient manner?
- Will my future self understand it?
- Will my collaborators and/or replicators understand it?
- Have I written in such a way that will prevent me from making mistakes? We will address each of these goals in order.

Code should be

- Easily understood by humans
- Relatively consistent
- Able to be adapted to new situations

Some resources to orient you to best practicies in writing code

- Read points 1-8 of this guide (specific to R but applies to other languages)
- Hadley Wickham's R style guide
- Another good resource on R best practices.

Pick good names

- You might be tempted to use abbreviations or non-words to save yourself from typing the same characters over and over
- With modern IDEs, tab completion means this is not a big issue
- Choose names that are legible, clear, and precise
- Name functions what they do, name variables what they are

Good and bad names

In R, dots are common, underscores are ok:

```
## a few bad variable names from my past work
incspendtran
V101
gender # if coded as one or zero, as a factor I'd accept this


## better names
increase.spending.transportation
respodent.id
female # coded 1 if female
```

Similar variables should have similar names

- For example, my candidates' perceptions of things always begin with `perc.`
- Next, geography, are they perceiving district, `perc.district`, or state, `perc.state`
- Sometimes they are perceiving opinion within a party: `perc.district.dem`
- Then the issue: `perc.district.dem.marriage` means their district-level perception of support for same-sex marriage. `perc.district.guns.banassault` is district-level support for banning assault weapons. There's also `guns.background.check`....
- This lets me use things like `grep()` to grab variables that are similar

Commenting

- Messages to all of your audiences – most importantly to yourself
- Say what you’re doing in plain words and why you’re doing it
- People differ a lot in their commenting preferences
- Find what’s comfortable for you, let it evolve
- A good rule of thumb - if it’s not obvious what’s going on from a quick skim of the code, comment it

Principles of good commenting

- Don’t comment things that are painfully obvious
- But most things won’t be painfully obvious
- Comment to introduce each new section of code or each major task
- If you write a new function, leave a comment explaining what it does

Introduce each new subsection of code with a comment

```r
# plot partisan estimates against one another
# voters only
g = ggplot(ncs, aes(voted.pid7.dem)) +
  geom_density(color = 'blue') +
  theme_classic() +
  theme(legend.position='none') +
  xlim(0,1) +
  theme(axis.title=element_text(size=8), plot.title = element_text(hjust=
  geom_density(aes(voted.pid7.rep), color = 'red') +
  ggtitle("Distribution of electorate partisanship estimates") +
  xlab("Percent of voters in district belonging to each party")
```

More examples of comments form my code

```r
# set up poststratification file and poststratify 2014 MRSP models

# level is upper or lower districts, pid.level is 3 or 7 point item
get.cell.predictions <- function(individual.model,
                                 pid.var, # e.g., 'pid3.dem'
                                 pstrat){
  # district random effects need zeros for missing dists
  district.ranefs <- ranef(individual.model)$modgeoid
  missing.districts <- setdiff(unique(pstrat$modgeoid), rownames(distric
  missing.districts.df <- data.frame(district = missing.districts)
  rownames(missing.districts.df) <- missing.districts.df$district
  missing.districts.df$`(Intercept)` <- 0
  missing.districts.df <- dplyr::select(missing.districts.df, -district)
  district.ranefs <- rbind.data.frame(district.ranefs, missing.districts

  # NOTE: make sure AK and HI is working
```

More comments from that R script

```r
# divide within proportions grouped by party
# and district to get dist-party estimates

pstrat.upper = pstrat %>%
  dplyr::filter(grepl("U", modgeoid)) %>%
  cbind(upper.cell.predictions) %>%
  dplyr::select(-X)

pstrat.lower = pstrat %>%
  dplyr::filter(grepl("L", modgeoid)) %>%
  cbind(lower.cell.predictions) %>%
  dplyr::select(-X)
```

More

```r
# transform so the columns are "the percent
# of the [PARTY] in the district that falls in each cell

# lower
make.party.proportions.for.level <- function(level) {
  if(level=="upper") pstrat = pstrat.upper
  if(level=="lower") pstrat = pstrat.lower

  party.proportions = data.frame(matrix(nrow = nrow(pstrat), ncol = lengt
  names(party.proportions) = party.id.vars

  for(i in 1:length(party.id.vars)){
    pid <- party.id.vars[i]

    # models is in same order as party.id.vars above
    pstrat[,pid] <- pstrat$proportion * pstrat[,pid]   }

    return(pstrat)
```

Commenting sounds easy in the abstract

- You’ll be cruising along writing code (no, really, eventually you will) and not feel like stopping to write comments
- Things will seem really obvious when you write them despite being not obvious to those who come after, including your future self
- Consistently remind yourself to leave comments

How to find help

- Three problems:
  - You don’t know where to start
  - You write code and it gives the wrong output
  - You write code and it throws an error

Error messages

- Can be helpful or inscrutable
- Often googling them will get you right to the solution or an explanation
- A guide on understanding the structure of R error messages

Expect to constantly be searching for help

- There’s way too much to remember, we offload that knowledge to the internet and elsewhere
- Messing up is the best way to learn when coding
- Get comfortable with spending a lot of time googling

Do due diligence before googling aimlessly

- You may be able to Google a solution, but that is sort of the being-given-a-fish way. You want to learn to fish.
- The best ways to teach yourself are to code a lot and read a lot of examples
- It’s a really slow process

Steal intelligently

- Figure out who’s doing similar tasks to you and look for tutorials online
- Steal people’s replication code (But be critical of it!)

How to read documentation

- Most documentation will show you a function and various *parameters*
- It’s important to understand different classes of data structures and variable classes in order to effectively use documentation
- In R, you’ll want to learn the differences between

Find vignettes and lessons written by others

- You’ll want to read package documentation, but more didactic examples are helpful too
- In R world these are called vignettes, you’ll also find “tutorials”
- Many github repositories have vignettes and usage examples attached to them.
- An example of a nice package vignette, for `broom`
- Another nice package vignette, for `dotwhisker`

Build up a library of cheats

- Bookmark useful help pages
- Make yourself a notebook of examples of code. Have a dropbox folder where you save snippets that do tasks you frequently do.
- I have things like group-level proportions that I frequently do for summary statistics
- Also have my favorite ggplot settings, etc
- Pillage old code for new projects (But be careful! You or your software have probably improved, so writing it anew could be a good learning experience and/or improve efficiency)

Someone else has had your problem before

- You will almost never need to post your own question to a mailing list or to StackOverflow.
- Someone has had a similar problem before, you just need to know how to find it
- People might have similar but slightly different problems that you can

Figuring out your google query

- Most of the time, you’re going to want to end up on StackOverflow. Often it’s good just to search within SO
- Look for tutorials if you find yourself frequently using a package: If you find yourself working with lots of strings, set aside an afternoon to look at everything written on `stringr`

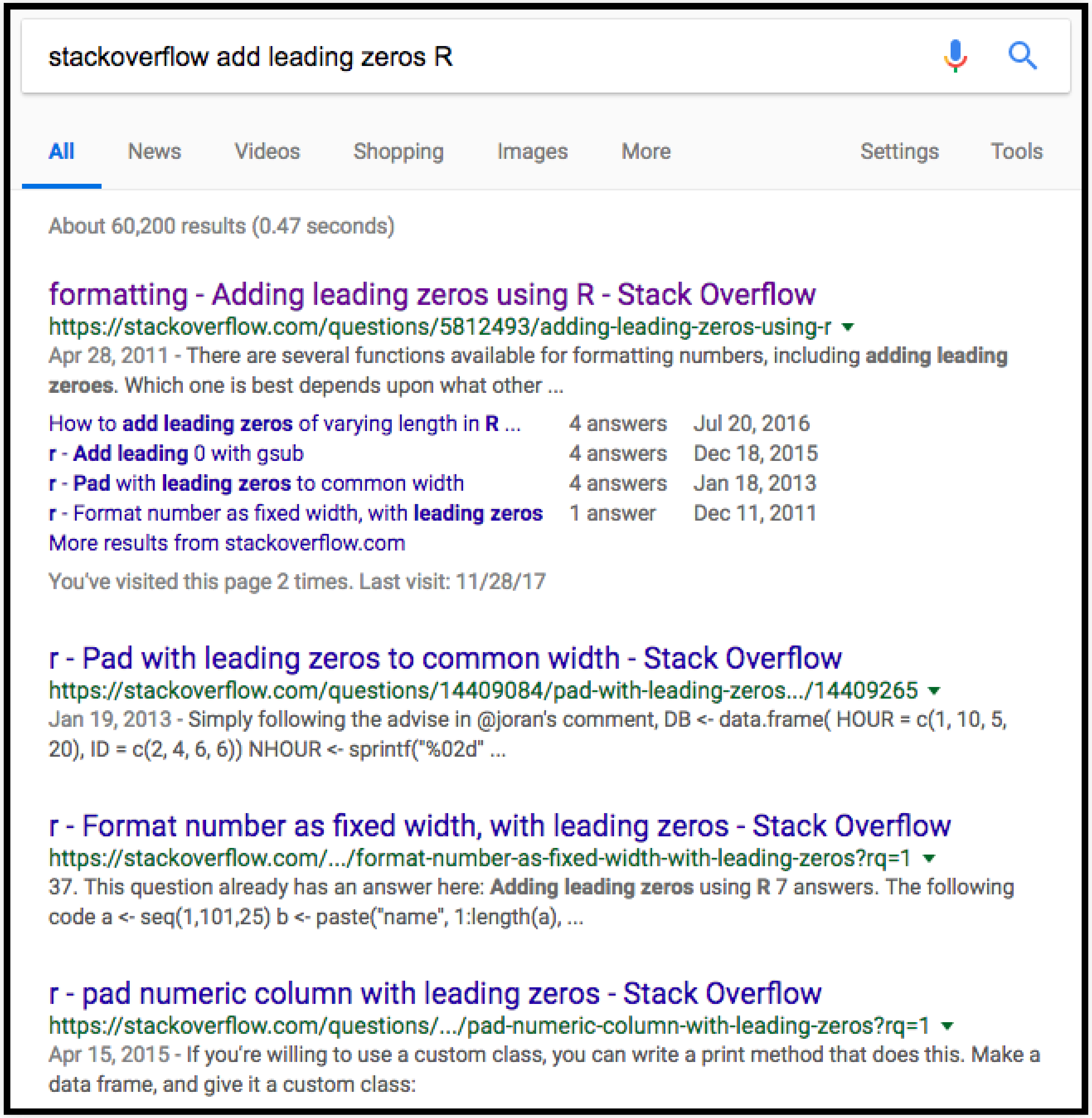
Effectively searching StackOverflow

- I’ve found the best queries are action verb type things:
  - Merge two data frames on multiple variables R
  - Rename columns with similar names R

Googling tips

- Include your language!
- StackOverflow will know the letter R means the language. In some google queries you might want to

What you’ll find on StackOverflow



stackoverflow add leading zeros R

About 60,200 results (0.47 seconds)

formatting - Adding leading zeros using R - Stack Overflow
https://stackoverflow.com/questions/5812493/adding-leading-zeros-using-r
Apr 28, 2011 - There are several functions available for formatting numbers, including **adding leading zeroes**. Which one is best depends upon what other ...

How to **add leading zeros** of varying length in **R** ...	4 answers	Jul 20, 2016
r - **Add leading** 0 with gsub	4 answers	Dec 18, 2015
r - **Pad** with **leading zeros** to common width	4 answers	Jan 18, 2013
r - Format number as fixed width, with **leading zeros**	1 answer	Dec 11, 2011

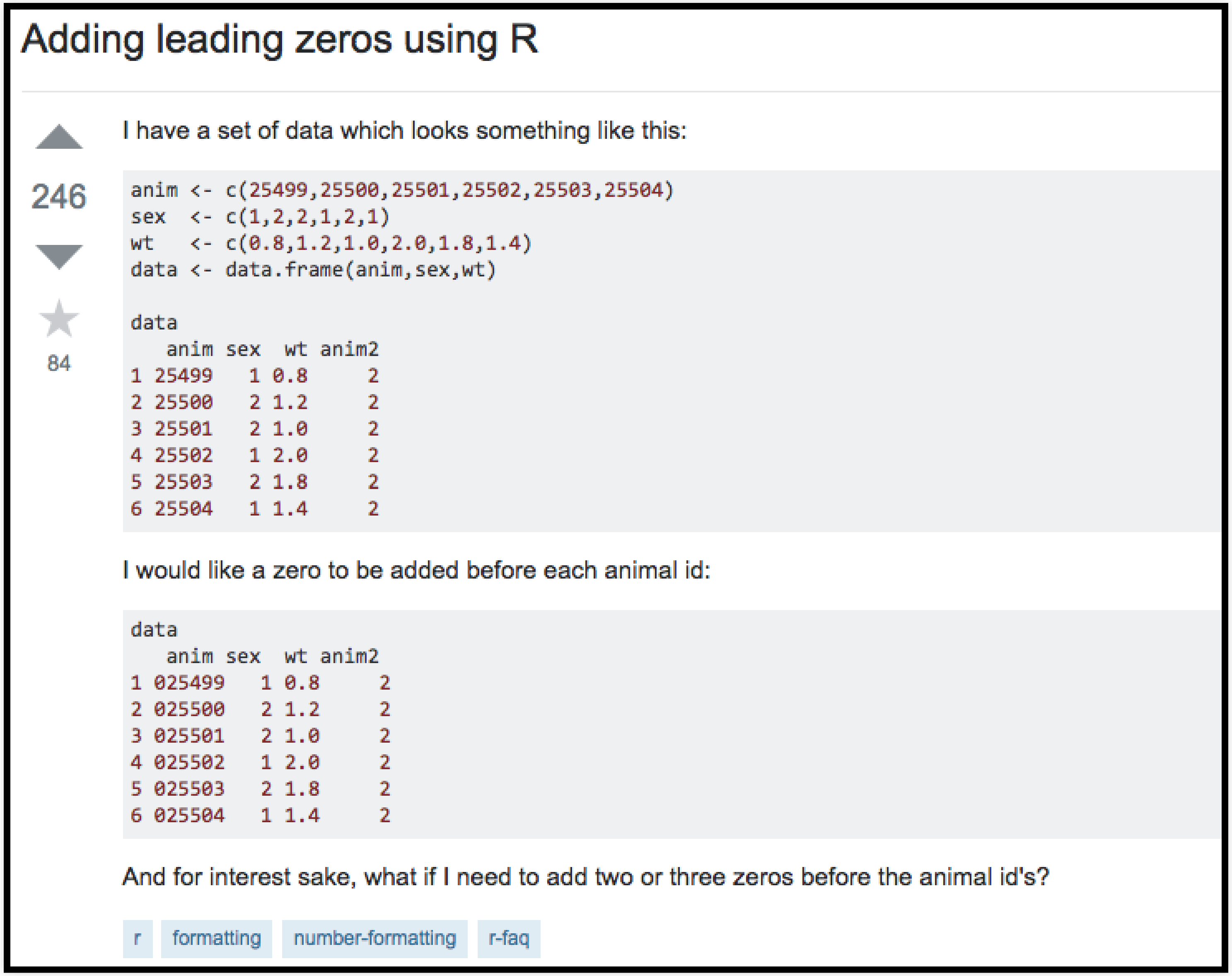
More results from stackoverflow.com
You've visited this page 2 times. Last visit: 11/28/17

r - Pad with leading zeros to common width - Stack Overflow
https://stackoverflow.com/questions/14409084/pad-with-leading-zeros.../14409265
Jan 19, 2013 - Simply following the advise in @joran's comment, DB <- data.frame(HOUR = c(1, 10, 5, 20), ID = c(2, 4, 6, 6)) NHOUR <- sprintf("%02d" ...

r - Format number as fixed width, with leading zeros - Stack Overflow
https://stackoverflow.com/.../format-number-as-fixed-width-with-leading-zeros?rq=1
37. This question already has an answer here: **Adding leading zeros** using **R** 7 answers. The following code a <- seq(1,101,25) b <- paste("name", 1:length(a), ...

r - pad numeric column with leading zeros - Stack Overflow
https://stackoverflow.com/questions/.../pad-numeric-column-with-leading-zeros?rq=1
Apr 15, 2015 - If you're willing to use a custom class, you can write a print method that does this. Make a data frame, and give it a custom class:

What you'll find on StackOverflow



Adding leading zeros using R

246

84

I have a set of data which looks something like this:

```
anim <- c(25499,25500,25501,25502,25503,25504)
sex  <- c(1,2,2,1,2,1)
wt   <- c(0.8,1.2,1.0,2.0,1.8,1.4)
data <- data.frame(anim,sex,wt)

data
   anim sex  wt anim2
1 25499   1 0.8     2
2 25500   2 1.2     2
3 25501   2 1.0     2
4 25502   1 2.0     2
5 25503   2 1.8     2
6 25504   1 1.4     2
```

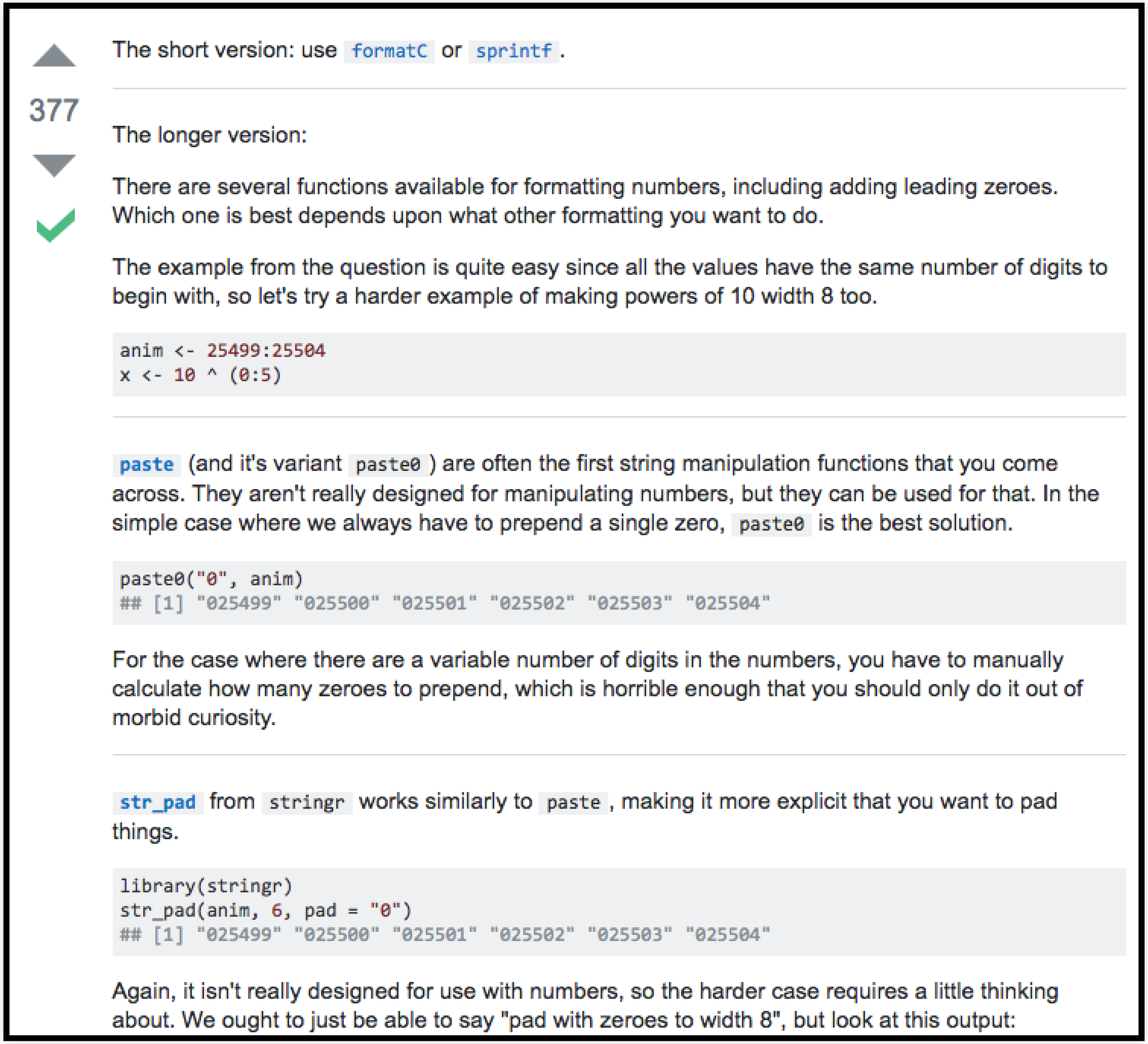
I would like a zero to be added before each animal id:

```
data
   anim sex  wt anim2
1 025499   1 0.8     2
2 025500   2 1.2     2
3 025501   2 1.0     2
4 025502   1 2.0     2
5 025503   2 1.8     2
6 025504   1 1.4     2
```

And for interest sake, what if I need to add two or three zeros before the animal id's?

r formatting number-formatting r-faq

What you’ll find on StackOverflow


377

✓

The short version: use `formatC` or `sprintf`.

The longer version:

There are several functions available for formatting numbers, including adding leading zeroes. Which one is best depends upon what other formatting you want to do.

The example from the question is quite easy since all the values have the same number of digits to begin with, so let's try a harder example of making powers of 10 width 8 too.

```
anim <- 25499:25504
x <- 10 ^ (0:5)
```

`paste` (and it's variant `paste0`) are often the first string manipulation functions that you come across. They aren't really designed for manipulating numbers, but they can be used for that. In the simple case where we always have to prepend a single zero, `paste0` is the best solution.

```
paste0("0", anim)
## [1] "025499" "025500" "025501" "025502" "025503" "025504"
```

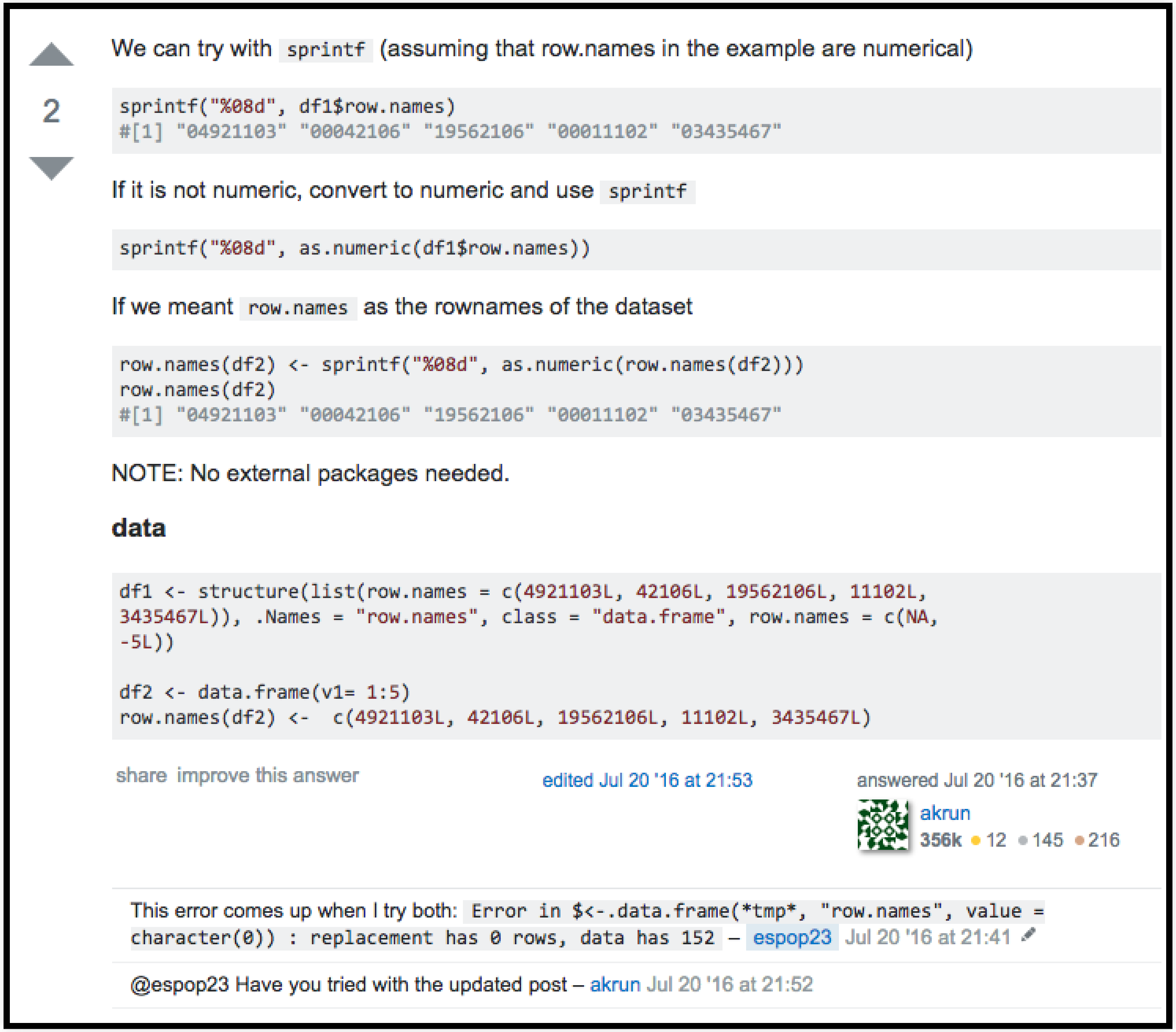
For the case where there are a variable number of digits in the numbers, you have to manually calculate how many zeroes to prepend, which is horrible enough that you should only do it out of morbid curiosity.

`str_pad` from `stringr` works similarly to `paste`, making it more explicit that you want to pad things.

```
library(stringr)
str_pad(anim, 6, pad = "0")
## [1] "025499" "025500" "025501" "025502" "025503" "025504"
```

Again, it isn't really designed for use with numbers, so the harder case requires a little thinking about. We ought to just be able to say "pad with zeroes to width 8", but look at this output:

What you’ll find on StackOverflow



2

We can try with `sprintf` (assuming that row.names in the example are numerical)

```
sprintf("%08d", df1$row.names)
#[1] "04921103" "00042106" "19562106" "00011102" "03435467"
```

If it is not numeric, convert to numeric and use `sprintf`

```
sprintf("%08d", as.numeric(df1$row.names))
```

If we meant `row.names` as the rownames of the dataset

```
row.names(df2) <- sprintf("%08d", as.numeric(row.names(df2)))
row.names(df2)
#[1] "04921103" "00042106" "19562106" "00011102" "03435467"
```

NOTE: No external packages needed.

data

```
df1 <- structure(list(row.names = c(4921103L, 42106L, 19562106L, 11102L,
3435467L)), .Names = "row.names", class = "data.frame", row.names = c(NA,
-5L))

df2 <- data.frame(v1= 1:5)
row.names(df2) <-  c(4921103L, 42106L, 19562106L, 11102L, 3435467L)
```

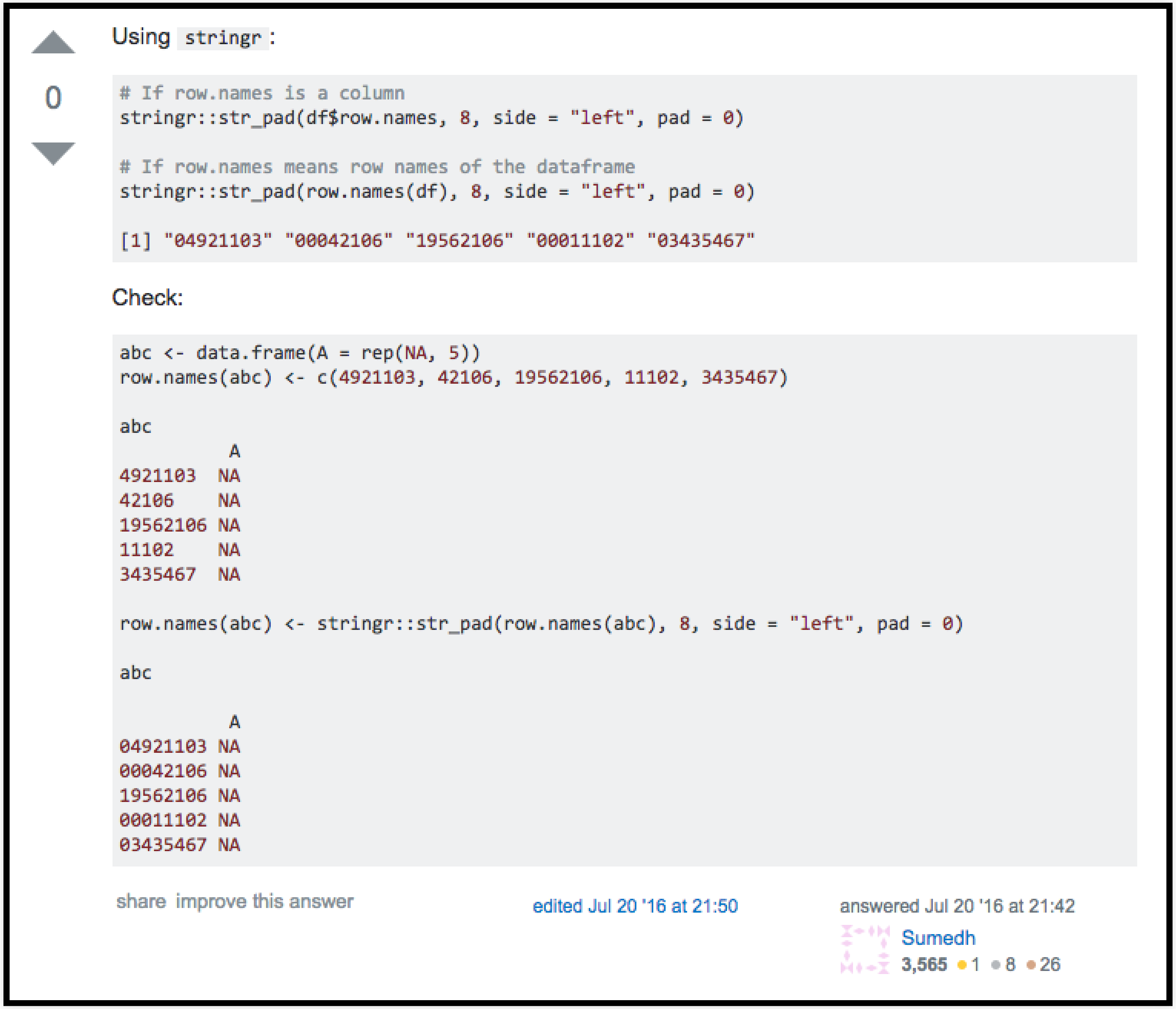
share improve this answer edited Jul 20 '16 at 21:53 answered Jul 20 '16 at 21:37

akrun
356k ● 12 ● 145 ● 216

This error comes up when I try both: `Error in $<-.data.frame(*tmp*, "row.names", value = character(0)) : replacement has 0 rows, data has 152` – espop23 Jul 20 '16 at 21:41

@espop23 Have you tried with the updated post – akrun Jul 20 '16 at 21:52

What you'll find on StackOverflow



0

Using `stringr`:

```
# If row.names is a column
stringr::str_pad(df$row.names, 8, side = "left", pad = 0)

# If row.names means row names of the dataframe
stringr::str_pad(row.names(df), 8, side = "left", pad = 0)

[1] "04921103" "00042106" "19562106" "00011102" "03435467"
```

Check:

```
abc <- data.frame(A = rep(NA, 5))
row.names(abc) <- c(4921103, 42106, 19562106, 11102, 3435467)

abc
          A
4921103  NA
42106    NA
19562106 NA
11102    NA
3435467  NA

row.names(abc) <- stringr::str_pad(row.names(abc), 8, side = "left", pad = 0)

abc

          A
04921103 NA
00042106 NA
19562106 NA
00011102 NA
03435467 NA
```

share improve this answer

edited Jul 20 '16 at 21:50

answered Jul 20 '16 at 21:42
Sumedh
3,565 ● 1 ● 8 ● 26

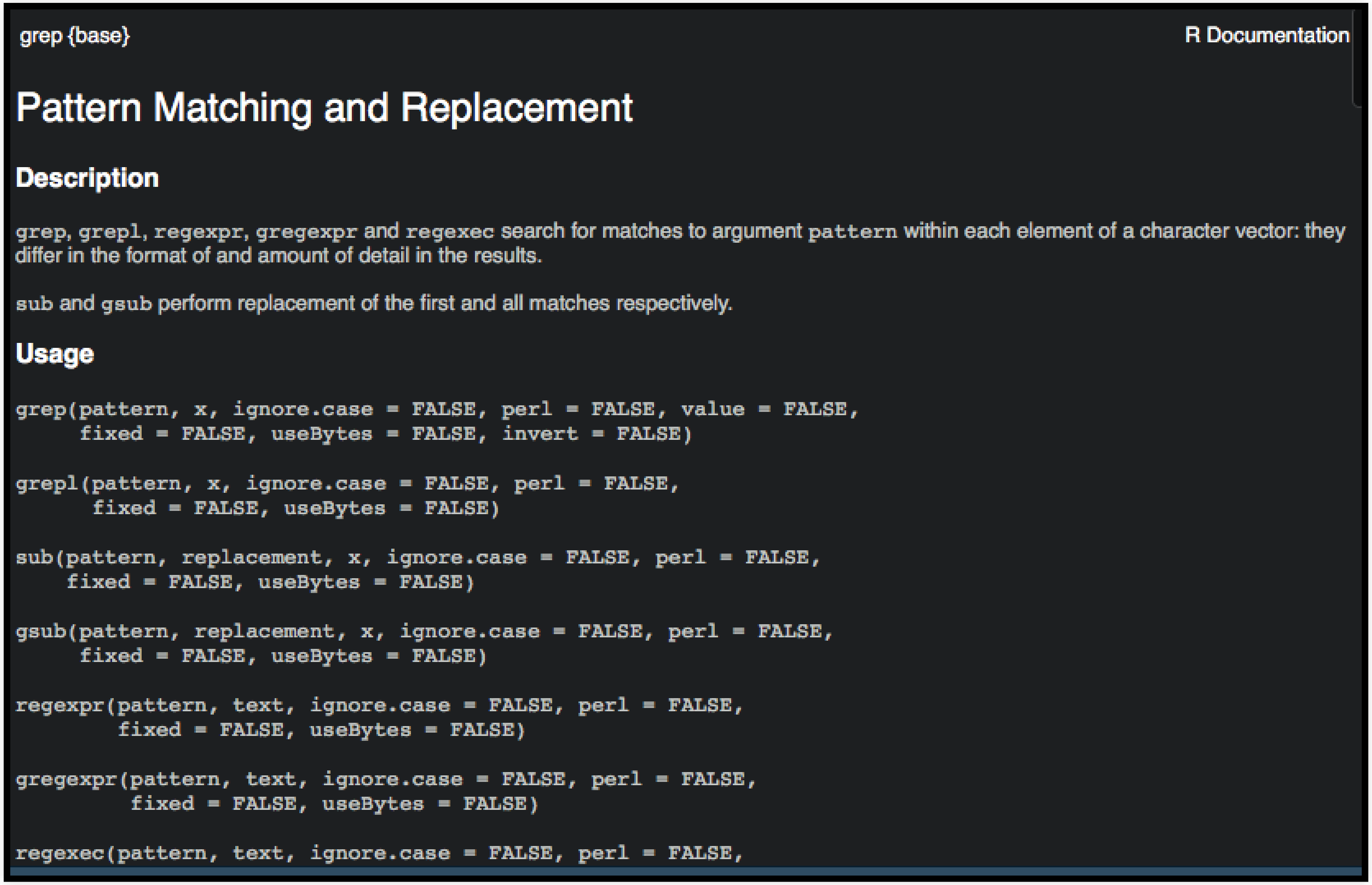
Use built-in help

```r
?lm
?read.csv
?dplyr::rename
```

Built-in help will

- Tell you how functions work, to varying degrees of helpfulness
- Often provide some examples of how functions work
- Often still leave you clueless and needing to Google around some more

Not-so-useful help



grep {base}　　　R Documentation

Pattern Matching and Replacement

Description

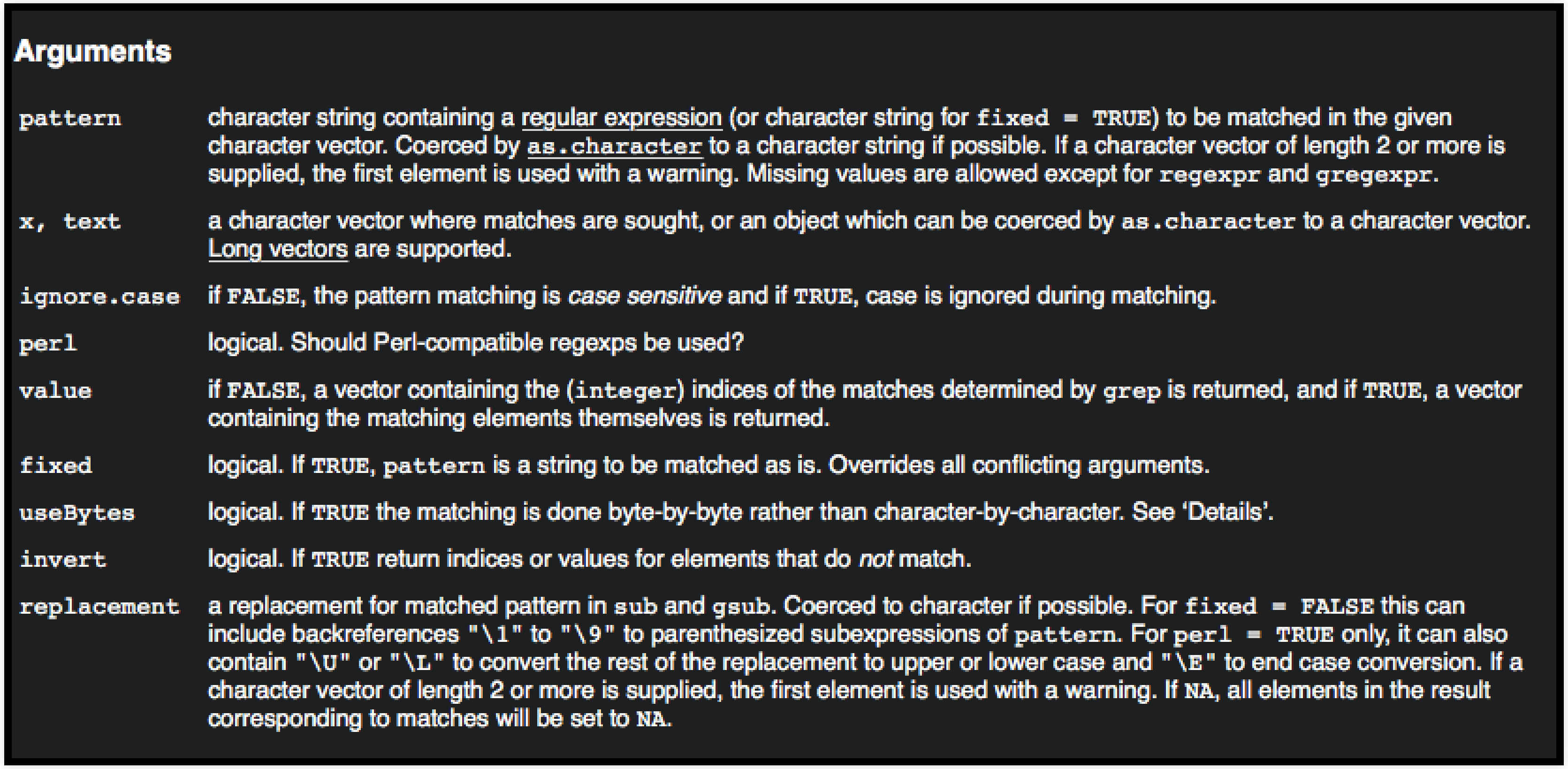
grep, grepl, regexpr, gregexpr and regexec search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.

sub and gsub perform replacement of the first and all matches respectively.

Usage

```
grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
     fixed = FALSE, useBytes = FALSE, invert = FALSE)

grepl(pattern, x, ignore.case = FALSE, perl = FALSE,
      fixed = FALSE, useBytes = FALSE)

sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
    fixed = FALSE, useBytes = FALSE)

gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
     fixed = FALSE, useBytes = FALSE)

regexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
        fixed = FALSE, useBytes = FALSE)

gregexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
         fixed = FALSE, useBytes = FALSE)

regexec(pattern, text, ignore.case = FALSE, perl = FALSE,
```

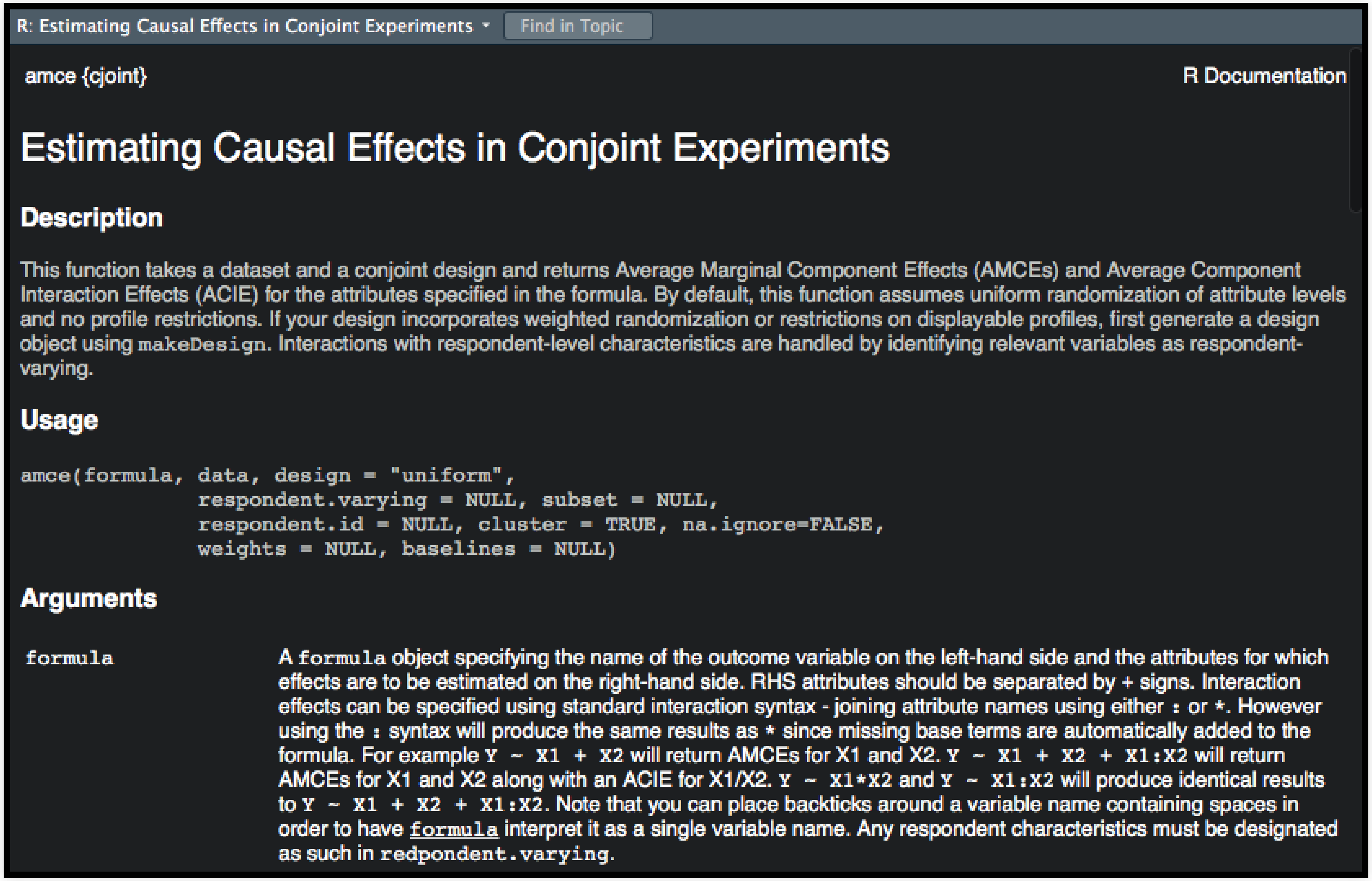
Not-so-useful help



Arguments

`pattern`	character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector. Coerced by `as.character` to a character string if possible. If a character vector of length 2 or more is supplied, the first element is used with a warning. Missing values are allowed except for `regexpr` and `gregexpr`.
`x, text`	a character vector where matches are sought, or an object which can be coerced by `as.character` to a character vector. Long vectors are supported.
`ignore.case`	if `FALSE`, the pattern matching is *case sensitive* and if `TRUE`, case is ignored during matching.
`perl`	logical. Should Perl-compatible regexps be used?
`value`	if `FALSE`, a vector containing the (`integer`) indices of the matches determined by `grep` is returned, and if `TRUE`, a vector containing the matching elements themselves is returned.
`fixed`	logical. If `TRUE`, `pattern` is a string to be matched as is. Overrides all conflicting arguments.
`useBytes`	logical. If `TRUE` the matching is done byte-by-byte rather than character-by-character. See 'Details'.
`invert`	logical. If `TRUE` return indices or values for elements that do *not* match.
`replacement`	a replacement for matched pattern in `sub` and `gsub`. Coerced to character if possible. For `fixed = FALSE` this can include backreferences `"\1"` to `"\9"` to parenthesized subexpressions of `pattern`. For `perl = TRUE` only, it can also contain `"\U"` or `"\L"` to convert the rest of the replacement to upper or lower case and `"\E"` to end case conversion. If a character vector of length 2 or more is supplied, the first element is used with a warning. If `NA`, all elements in the result corresponding to matches will be set to `NA`.

Some builtin help is more useful



amce {cjoint} R Documentation

Estimating Causal Effects in Conjoint Experiments

Description

This function takes a dataset and a conjoint design and returns Average Marginal Component Effects (AMCEs) and Average Component Interaction Effects (ACIE) for the attributes specified in the formula. By default, this function assumes uniform randomization of attribute levels and no profile restrictions. If your design incorporates weighted randomization or restrictions on displayable profiles, first generate a design object using `makeDesign`. Interactions with respondent-level characteristics are handled by identifying relevant variables as respondent-varying.

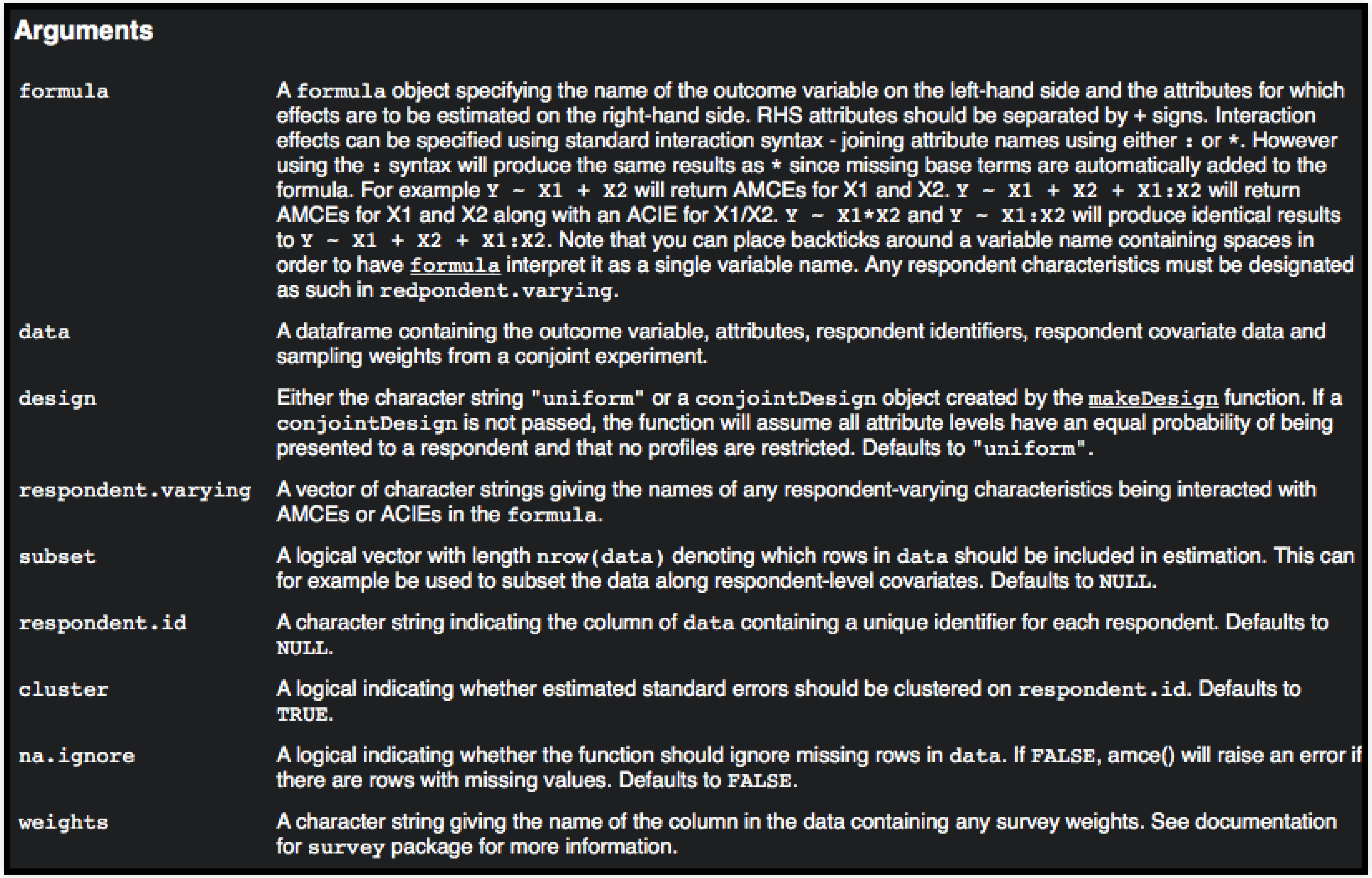
Usage

```
amce(formula, data, design = "uniform",
            respondent.varying = NULL, subset = NULL,
            respondent.id = NULL, cluster = TRUE, na.ignore=FALSE,
            weights = NULL, baselines = NULL)
```

Arguments

`formula`	A `formula` object specifying the name of the outcome variable on the left-hand side and the attributes for which effects are to be estimated on the right-hand side. RHS attributes should be separated by + signs. Interaction effects can be specified using standard interaction syntax - joining attribute names using either `:` or `*`. However using the `:` syntax will produce the same results as `*` since missing base terms are automatically added to the formula. For example `Y ~ X1 + X2` will return AMCEs for X1 and X2. `Y ~ X1 + X2 + X1:X2` will return AMCEs for X1 and X2 along with an ACIE for X1/X2. `Y ~ X1*X2` and `Y ~ X1:X2` will produce identical results to `Y ~ X1 + X2 + X1:X2`. Note that you can place backticks around a variable name containing spaces in order to have `formula` interpret it as a single variable name. Any respondent characteristics must be designated as such in `redpondent.varying`.

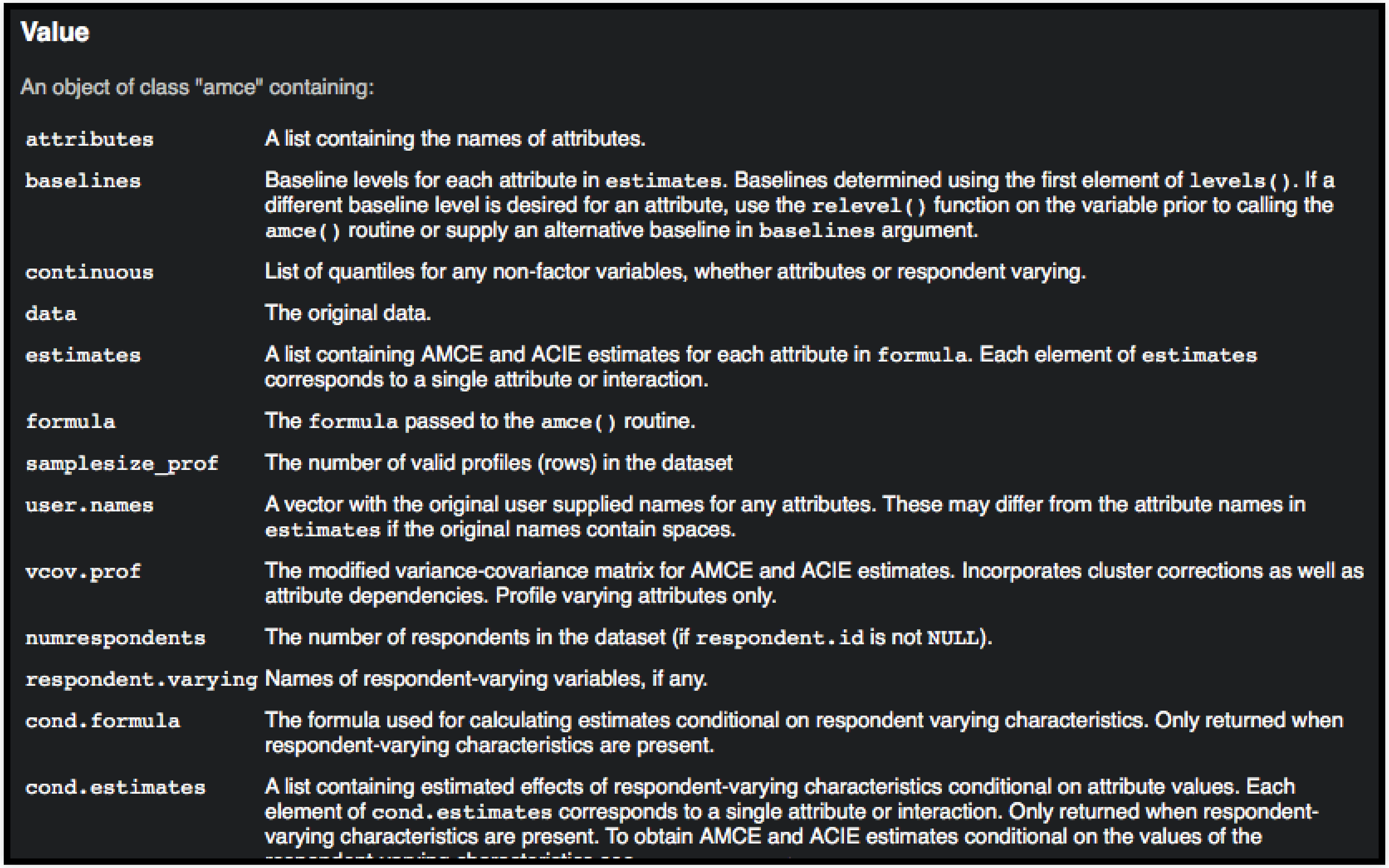
Some builtin help is more useful



Arguments

`formula`	A `formula` object specifying the name of the outcome variable on the left-hand side and the attributes for which effects are to be estimated on the right-hand side. RHS attributes should be separated by + signs. Interaction effects can be specified using standard interaction syntax - joining attribute names using either `:` or `*`. However using the `:` syntax will produce the same results as `*` since missing base terms are automatically added to the formula. For example `Y ~ X1 + X2` will return AMCEs for X1 and X2. `Y ~ X1 + X2 + X1:X2` will return AMCEs for X1 and X2 along with an ACIE for X1/X2. `Y ~ X1*X2` and `Y ~ X1:X2` will produce identical results to `Y ~ X1 + X2 + X1:X2`. Note that you can place backticks around a variable name containing spaces in order to have `formula` interpret it as a single variable name. Any respondent characteristics must be designated as such in `redpondent.varying`.
`data`	A dataframe containing the outcome variable, attributes, respondent identifiers, respondent covariate data and sampling weights from a conjoint experiment.
`design`	Either the character string `"uniform"` or a `conjointDesign` object created by the `makeDesign` function. If a `conjointDesign` is not passed, the function will assume all attribute levels have an equal probability of being presented to a respondent and that no profiles are restricted. Defaults to `"uniform"`.
`respondent.varying`	A vector of character strings giving the names of any respondent-varying characteristics being interacted with AMCEs or ACIEs in the `formula`.
`subset`	A logical vector with length `nrow(data)` denoting which rows in `data` should be included in estimation. This can for example be used to subset the data along respondent-level covariates. Defaults to `NULL`.
`respondent.id`	A character string indicating the column of `data` containing a unique identifier for each respondent. Defaults to `NULL`.
`cluster`	A logical indicating whether estimated standard errors should be clustered on `respondent.id`. Defaults to `TRUE`.
`na.ignore`	A logical indicating whether the function should ignore missing rows in `data`. If `FALSE`, amce() will raise an error if there are rows with missing values. Defaults to `FALSE`.
`weights`	A character string giving the name of the column in the data containing any survey weights. See documentation for `survey` package for more information.

Some builtin help is more useful

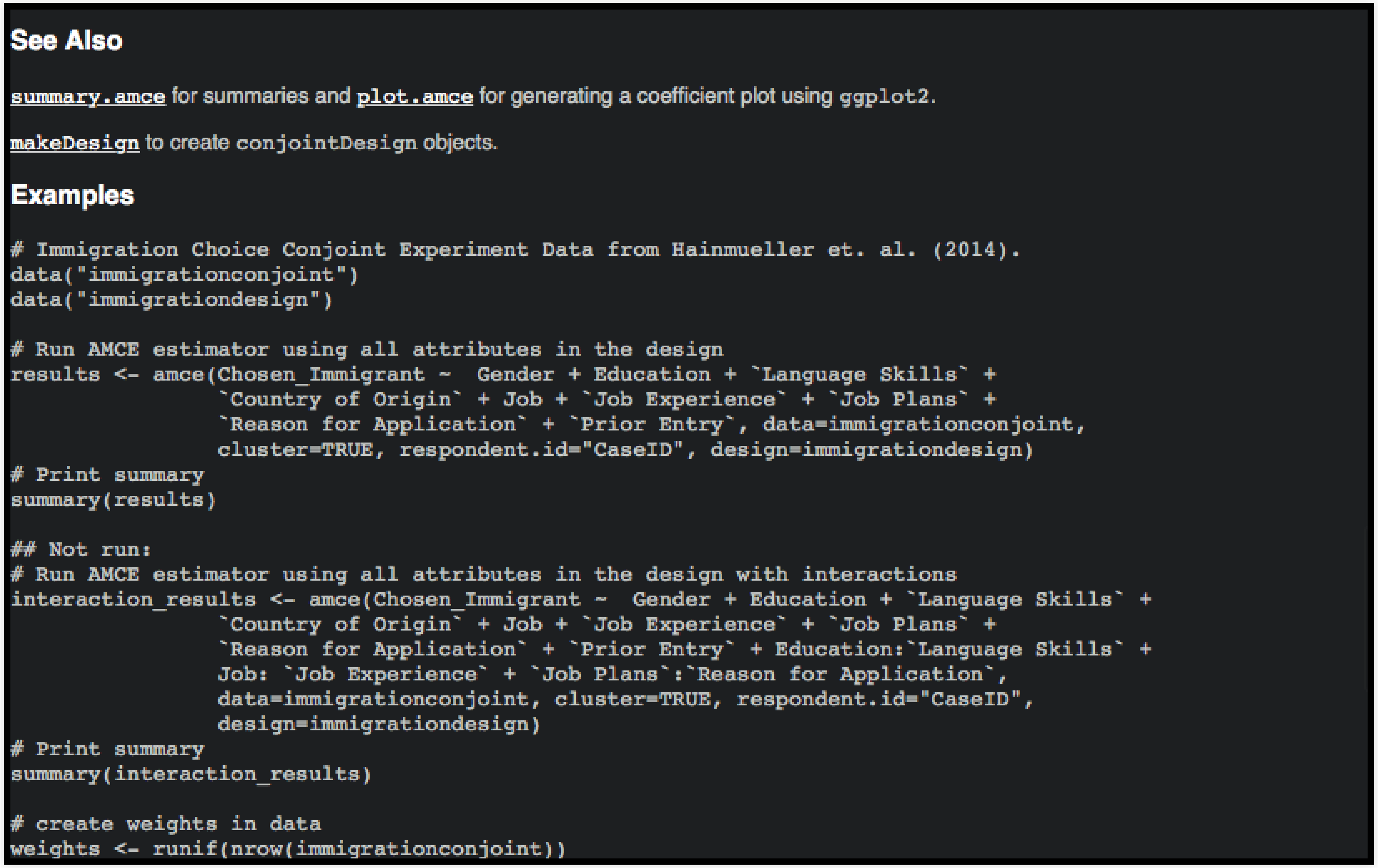


Value

An object of class "amce" containing:

`attributes`	A list containing the names of attributes.
`baselines`	Baseline levels for each attribute in `estimates`. Baselines determined using the first element of `levels()`. If a different baseline level is desired for an attribute, use the `relevel()` function on the variable prior to calling the `amce()` routine or supply an alternative baseline in `baselines` argument.
`continuous`	List of quantiles for any non-factor variables, whether attributes or respondent varying.
`data`	The original data.
`estimates`	A list containing AMCE and ACIE estimates for each attribute in `formula`. Each element of `estimates` corresponds to a single attribute or interaction.
`formula`	The `formula` passed to the `amce()` routine.
`samplesize_prof`	The number of valid profiles (rows) in the dataset
`user.names`	A vector with the original user supplied names for any attributes. These may differ from the attribute names in `estimates` if the original names contain spaces.
`vcov.prof`	The modified variance-covariance matrix for AMCE and ACIE estimates. Incorporates cluster corrections as well as attribute dependencies. Profile varying attributes only.
`numrespondents`	The number of respondents in the dataset (if `respondent.id` is not `NULL`).
`respondent.varying`	Names of respondent-varying variables, if any.
`cond.formula`	The formula used for calculating estimates conditional on respondent varying characteristics. Only returned when respondent-varying characteristics are present.
`cond.estimates`	A list containing estimated effects of respondent-varying characteristics conditional on attribute values. Each element of `cond.estimates` corresponds to a single attribute or interaction. Only returned when respondent-varying characteristics are present. To obtain AMCE and ACIE estimates conditional on the values of the

Some builtin help is more useful



See Also

summary.amce for summaries and **plot.amce** for generating a coefficient plot using `ggplot2`.

makeDesign to create `conjointDesign` objects.

Examples

```
# Immigration Choice Conjoint Experiment Data from Hainmueller et. al. (2014).
data("immigrationconjoint")
data("immigrationdesign")

# Run AMCE estimator using all attributes in the design
results <- amce(Chosen_Immigrant ~  Gender + Education + `Language Skills` +
                `Country of Origin` + Job + `Job Experience` + `Job Plans` +
                `Reason for Application` + `Prior Entry`, data=immigrationconjoint,
                cluster=TRUE, respondent.id="CaseID", design=immigrationdesign)
# Print summary
summary(results)

## Not run:
# Run AMCE estimator using all attributes in the design with interactions
interaction_results <- amce(Chosen_Immigrant ~  Gender + Education + `Language Skills` +
                `Country of Origin` + Job + `Job Experience` + `Job Plans` +
                `Reason for Application` + `Prior Entry` + Education:`Language Skills` +
                Job: `Job Experience` + `Job Plans`:`Reason for Application`,
                data=immigrationconjoint, cluster=TRUE, respondent.id="CaseID",
                design=immigrationdesign)
# Print summary
summary(interaction_results)

# create weights in data
weights <- runif(nrow(immigrationconjoint))
```

Asking for help

- You will want a “minimal working example”
- Here’s a good guide for writing one in R from Jared Knowles
- The most efficient way to ask for help, in exponentially decreasing order: Ask Google, ask people you know, ask strangers online

IT GETS BETTER