Package ‘abjutils’

August 13, 2020

Type Package

Title Useful Tools for Jurimetical Analysis Used by the Brazilian Jurimetrics Association

Version 0.3.1

Date 2020-08-12

Description The Brazilian Jurimetrics Association (ABJ in Portuguese, see <https://abj.org.br/> for more information) is a non-profit organization which aims to investigate and promote the use of statistics and probability in the study of Law and its institutions. This package implements general purpose tools used by ABJ, such as functions for sampling and basic manipulation of Brazilian lawsuits identification number. It also implements functions for text cleaning, such as accentuation removal.

License MIT + file LICENSE

URL https://github.com/abjur/abjutils

Depends R (>= 4.0)

Imports dplyr, magrittr, purrr, rlang, rstudioapi, stringi, stringr, tidyrr

Suggests testthat

Encoding UTF-8

LazyData TRUE

RoxygenNote 7.1.1

NeedsCompilation no

Author Caio Lente [aut, cre], Julio Trecenti [aut] (<https://orcid.org/0000-0002-1680-6389>), Associação Brasileira de Jurimetria [cph, fnd]

Maintainer Caio Lente <clente@abj.org.br>

Repository CRAN

Date/Publication 2020-08-12 22:50:05 UTC
R topics documented:

- build_id .................................................. 2
- calc_dig ................................................... 3
- carf_build_id ........................................... 3
- carf_calc_dig ........................................... 4
- carf_check_dig ........................................... 4
- check_dig ................................................ 5
- check_dig_vet ........................................... 6
- chrome_to_body ......................................... 6
- clean_cnj ............................................... 7
- clean_id ............................................... 7
- escape_unicode .......................................... 7
- extract_parts ........................................... 8
- file_sans_ext ........................................... 8
- gather_subjects ......................................... 9
- lsos ..................................................... 9
- pattern_cnj ............................................. 10
- precision ............................................... 10
- reais ..................................................... 10
- rm_accenct ............................................. 11
- sample_cnj ............................................. 11
- separate_cnj ........................................... 13
- tabela ................................................... 13
- test_fun ............................................... 14
- verify_cnj ............................................... 15
- write_data ............................................. 15

Index 16

---

**build_id**  
*Add separators to lawsuit IDs*

**Description**

Add separators to lawsuit IDs

**Usage**

```
build_id(id)
```

**Arguments**

- **id**  
  One or more lawsuit IDs
**calc_dig**

*Calculate digits for Brazilian lawsuit identification numbers*

**Description**

Returns the check digit of a lawsuit numbers in the format unified by the Brazilian National Council of Justice.

**Usage**

```r
calc_dig(num, build = FALSE)
```

**Arguments**

- `num` Ordered digits of the lawsuit number (including 0's) excluding the check digit
- `build` Whether or not the function return the complete lawsuit number (or only the check digits)?

**Value**

The check digits or the complete identification number

**Examples**

```r
{  
calc_dig("001040620018260004", build = TRUE)  
calc_dig("001040620018260004", build = FALSE)  
}
```

**carf_build_id**

*Add separators to CARF lawsuits*

**Description**

Add separators to CARF lawsuits

**Usage**

```r
carf_build_id(id)
```

**Arguments**

- `id` One or more lawsuit ids
### carf_calc_dig

*Calculate check digit for CARF*

**Description**

Returns the check digit of a CARF number or full number with the check digit.

**Usage**

```r
carf_calc_dig(id, build = FALSE, verify = TRUE)
```

**Arguments**

- `id` - Lawsuit number (including trailing zeros), excluding the check digit.
- `build` - Whether or not the function return the complete number (or only the check digits)?
- `verify` - Verify if number is well formed (gives error if it's not)

**Value**

The check digits or the complete identification number

**Examples**

```r
{  
carf_calc_dig("10120.008427/2003", build = TRUE)
carf_calc_dig("15374.002430/99", build = FALSE)
carf_calc_dig(c("101200084272003", "1537400243099"))
}
```

### carf_check_dig

*Validate check digits for Brazilian lawsuits identification number*  

**Description**

Verifies if a check digit is correct

**Usage**

```r
carf_check_dig(id)
```

**Arguments**

- `id` - String containing the complete lawsuit number
check_dig

Value

Whether or not the check digit is well calculated

Examples

```
{  
carf_check_dig("10120.008427/2003-02")  
carf_check_dig(c("1012008427200302", "10766.000511/96-12"))  
}
```

Check digits for Brazilian lawsuits identification number

Description

Verifies if a check digit is correct

Usage

```
check_dig(num)  
```

Arguments

```
num String containing the complete lawsuit number  
```

Value

Whether or not the check digit is well calculated

Examples

```
{  
  check_dig("0005268-75.2013.8.26.0100")  
}
```
### check_dig_vet

Validate check digits for Brazilian lawsuits identification number on vectors.

**Description**

Verifies if a check digit is correct

**Usage**

```r
check_dig_vet(num)
```

**Arguments**

- `num` A vector containing strings with the complete lawsuit number

**Value**

Whether or not the check digit is well calculated

**Examples**

```r
{  
}
```

### chrome_to_body

Convert Chrome’s Query String Parameters to a list

**Description**

To use this function, simply copy the Query String Parameters returned by Chrome when analyzing the network flow of a web page. Paste these QSPs into an R string with double quotes (as you would to create any string) and pass it to `chrome_to_body()`; the function will print to the console a formatted command that creates a list with the QSPs. This list works perfectly with `httr::GET()` and `httr::POST()` so that you can easily reproduce a website’s behavior.

**Usage**

```r
chrome_to_body(x)
```

**Arguments**

- `x` A string with Chrome’s Query String Parameters

**See Also**

- `httr::GET()`, `httr::POST()`
clean_cnj

Clean a cnj number.

Description
Remove all non-numeric character from a string

Usage
clean_cnj(x)

Arguments
x A string (cnj)

clean_id
Remove separators from lawsuit IDs

Description
Remove separators from lawsuit IDs

Usage
clean_id(id)

Arguments
id One or more lawsuit IDs

escape_unicode
Escape accented characters in a document

Description
This function is used by the "Escape Unicode" add-in and removes all accented characters from the current file, replacing them by their equivalent Unicode-escaped values.

Usage
escape_unicode()
extract_parts  Extract different parts from lawsuit ID

Description
Given one or more lawsuit IDs, this function extracts one or more parts of the IDs given the following correspondence:

- "N": number
- "D": verification digits
- "A": year
- "J": segment
- "T": court
- "O": origin
- ": all of the above

Usage
extract_parts(id, parts = ")

Arguments
id  One or more lawsuit IDs
parts  String or string vector with desired parts (see description)

Examples
## Not run:
extract_parts("001040620018260004", "N")
extract_parts("001040620018260004", c("N", "A", "O"))

## End(Not run)

file_sans_ext  Extract file name without extension

Description
Extract file name without extension

Usage
file_sans_ext(x)

Arguments
x  Character vector of file paths
**gather_subjects**

Gather subjects from `esaj::cjsg_table("subjects")`

**Description**

Once you run `esaj::cjsg_table("subjects")`, you can use this function to gather the subjects automatically. Download esaj by running `devtools::install_github("courtsbr/esaj")`.

**Usage**

```r
gather_subjects(subjects)
```

**Arguments**

- `subjects` Table returned by `esaj::cjsg_table("subjects")`

---

**lsos**

Improved list of objects

**Description**

Elegantly list objects in a R session.

**Usage**

```r
lsos(
  pos = 1,
  pattern,
  order.by = "Size",
  decreasing = TRUE,
  head = TRUE,
  n = 10
)
```

**Arguments**

- `pos` Where to look for the object (see "Details" in `base::get()`'s documentation)
- `pattern` An optional regular expression to match names (`utils::glob2rx()` can be used to convert wildcard patterns to regular expressions)
- `order.by` Sort by "Size" (default), "Type", "Rows" or "Columns"
- `decreasing` Should the sorting be decreasing?
- `head` Should `utils::head()` function be used for printing?
- `n` How many lines `utils::head()` function should show?

**References**

**pattern_cnj**  
*Regex pattern for finding lawsuit numbers*

**Description**  
Regex pattern for finding lawsuit numbers

**Usage**  
`pattern_cnj()`

**precision**  
*Mirror of scales:::precision()*

**Description**  
Mirror of scales:::precision()

**Usage**  
`precision(x)`

**Arguments**

- `x`  
  See scales:::precision()

**reais**  
*Convert Brazilian currency values (text) to numeric*

**Description**  
Convert Brazilian currency values (text) to numeric

**Usage**  
`reais(x)`

**Arguments**

- `x`  
  A currency vector. Ex: `c("R$ 10.000,00", "R$ 123,00")`
### rm_accent

**Remove accentuation**

**Description**
Remove accented characters from strings converting them to ASCII.

**Usage**

```r
rm_accent(x)
```

**Arguments**

- `x`: A string vector

**Value**
A version of `x` without non-ASCII characters

### sample_cnj

**Generate sample Brazilian lawsuit identification numbers**

**Description**
Returns a data frame containing a random sample of lawsuit numbers distributed according to some regional and jurisdictional parameters. The implementation supports both vector and scalar parameters, depending whether or not the function should uniformly sample from a scope of lawsuit numbers or one should define the parameters for each sample unit.

**Usage**

```r
sample_cnj(
  n, foros, anos, orgao, tr, first_dig = "0", sample_pars = TRUE, return_df = TRUE
)
```
Arguments

- **n**: A non-negative integer giving the number of codes to generate.
- **foros**: One or more strings with 4 characters indicating the juridical forum for the sampled codes.
- **anos**: One or more strings with 4 characters indicating the distribution years of the generated codes.
- **orgao**: One or more strings with 1 character indicating the jurisdiction of the sampled codes.
- **tr**: One or more strings with 1 character indicating the court of the generated codes.
- **first_dig**: The first digit of the lawsuit code ("0" by default and sampled if ")
- **sample_pars**: Whether or not the parameters define the characteristics of the codes.
- **return_df**: Whether or not the function should return a data frame.

Value

A data frame or a vector containing a random sample of lawsuits IDs.

Examples

```r
# sampling the parameters
sample_cnj(3,
   foros = "0000",
   anos = "2015", orgao = 8, tr = 26,
   first_dig = "0", sample_pars = TRUE, return_df = FALSE)

sample_cnj(10,
   foros = c("0000", "0001"),
   anos = c("2014", "2015"), orgao = 8, tr = 26,
   first_dig = "0", sample_pars = TRUE, return_df = FALSE)
# not sampling the parameters

sample_cnj(3,
   foros = c("0000", "0001", "0002"),
   anos = c("2014", "2015", "2016"), orgao = rep(8, 3), tr = rep(26, 3),
   first_dig = "0", sample_pars = FALSE, return_df = FALSE)
```

separate_cnj

Separate a lawsuit ID column into its parts

Description

Wrapper around `tidyr::separate()` that splits a column with lawsuit IDs into 6 columns with its parts (see `extract_parts()`). Note that the IDs must be built (see `build_id()`).

Usage

```r
separate_cnj(data, col, ...)
```

Arguments

- `data` A data frame
- `col` Column name or position (see `tidyr::separate()`)  
- `...` Other arguments passed on to `tidyr::separate()`

`tabela` Produce frequency and relative frequency tables

Description

Produces a contingency table of the elements of a vector calculating relative frequencies as well.

Usage

```r
tabela(x, label = "variavel")
```

Arguments

- `x` A vector
- `label` Quoted name of the column to create in output

Value

A data frame containing frequency and relative frequencies for the levels of `x`
test_fun

Tests a function by checking if its arguments are declared

Description

This function verifies whether all of the arguments of another function already have assigned values. If an argument has a default value but there isn’t a corresponding variable, it creates that variable.

Usage

test_fun(f, force_default = FALSE)

Arguments

f          A function
force_default Whether or not to assign the default value to arguments that already have assigned values

Examples

## Not run:
f <- function(a, b = 3) {
  a * b
}

test_fun(f)
a
b

b <- 5
test_fun(f)
a
b

test_fun(f, TRUE)
a
b

a <- 2
test_fun(f)
a
b

## End(Not run)
**verify_cnj**

**Description**

Verifies if a Brazilian lawsuit identification is a cnj number.

**Usage**

```r
verify_cnj(cnj)
```

**Arguments**

- `cnj` A vector containing strings with the complete lawsuit number

**Value**

Whether or not the check digit is well calculated

---

**write_data**

*Shortcut to write file to "data/" directory from a pipe*

**Description**

Shortcut to write file to "data/" directory from a pipe

**Usage**

```r
write_data(x, name, dir = "data/")
```

**Arguments**

- `x` Object to write
- `name` Name of the object (important when loading)
- `dir` Directory where to save file
Index

base::get(), 9
build_id, 2
build_id(), 13

calc_dig, 3
carf_build_id, 3
carf_calc_dig, 4
carf_check_dig, 4
check_dig, 5
check_dig_vet, 6
chrome_to_body, 6
clean_cnj, 7
clean_id, 7

escape_unicode, 7
extract_parts, 8
extract_parts(), 13

file_sans_ext, 8
gather_subjects, 9

httr::GET(), 6
httr::POST(), 6

lsos, 9

pattern_cnj, 10
precision, 10

reais, 10
rm_accent, 11

sample_cnj, 11
separate_cnj, 13

tabela, 13
test_fun, 14
tidyr::separate(), 13

utils::glob2rx(), 9
utils::head(), 9

verify_cnj, 15
write_data, 15