Package ‘abjutils’

February 7, 2019

Type Package

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

Version 0.2.3

Date 2019-02-04

Maintainer Caio Lente <caio.lente@usp.br>

Description The Brazilian Jurimetrics Association (ABJ in Portuguese, see <http://www.abjur.org.br/en/> 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 (>= 3.4)

Imports devtools, dplyr, furrr, future, glue, httr, magrittr, progress, purrr, readr, rlang, rstudioapi, scales, stringi, stringr, tibble, tidyr

Suggests testthat

LazyData TRUE

RoxygenNote 6.1.1

NeedsCompilation no

Author Julio Trecenti [aut], Athos Damiani [ctb], Fernando Correa [aut], Caio Lente [aut, cre], Brazilian Jurimetrics Association [cph]

Repository CRAN

Date/Publication 2019-02-07 21:43:35 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
- pvec ............................................ 10
- reais ........................................... 11
- rm_accent ....................................... 12
- sample_cnj ...................................... 12
- separate_cnj .................................... 13
- tabela ........................................... 14
- test_fun ........................................ 14
- use_pipe ....................................... 15
- verify_cnj ...................................... 16
- write_data ...................................... 16

Index 17

---

| 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**
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**

```
{  
calc_dig("001040620018260004", build = TRUE)
calc_dig("001040620018260004", build = FALSE)
# Not run:
calc_dig("0010406200182600", build = TRUE)

# End(Not run)
}
```

---

**carf_build_id**  
*Add separators to CARF lawsuits*

**Description**
Add separators to CARF lawsuits

**Usage**
carf_build_id(id)

**Arguments**
- **id**: One or more lawsuit ids
carf_check_dig

**Description**

Verifies if a check digit is correct

**Usage**

```r
carf_check_digit(id)
```

**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("15374.002439/99", build = FALSE)
  carf_calc_dig(c("101200084272003", "1537400243099"))
  carf_calc_dig("10766.000511/96-12")
}
```
check\_dig

Arguments

\begin{itemize}
\item \texttt{id} String containing the complete lawsuit number
\end{itemize}

Value

Whether or not the check digit is well calculated

Examples

\begin{verbatim}
{
  carf_check_dig("1P1RPNPP8TR7ORPPS-PR")
  carf_check_dig(c("1P766NPPPU11O96-1R", "10766.000511/96-12"))
}
\end{verbatim}

---

\begin{tabular}{ll}
\textbf{check\_dig} & \textit{Validate check digits for Brazilian lawsuits identification number} \\
\end{tabular}

Description

Verifies if a check digit is correct

Usage

\begin{verbatim}
check\_dig(num)
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{num} String containing the complete lawsuit number
\end{itemize}

Value

Whether or not the check digit is well calculated

Examples

\begin{verbatim}
{
  check\_dig("0005268-75.2013.8.26.0100")

  \# Not run:
  check\_dig("0005268-75.2013.8.26.100", build = TRUE)

  \# End(Not run)
}
\end{verbatim}
check_dig_vet  Validate check digits for Brazilian lawsuits identification number on vectorial scale.

Description
Verifies if a check digit is correct

Usage
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
{
}

crime_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 analysing 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 crime_to_body(); the function will print to the console a formatted command that creates a list with the QSPs. This list works perfectly with \texttt{httr::GET()} and \texttt{httr::POST()} so that you can easily reproduce a website’s behavior.

Usage
crime_to_body(x)

Arguments
x  A string with Chrome’s Query String Parameters
## clean_cnj

### Description
Remove all non-numeric character from a string

### Usage
```r
clean_cnj(x)
```

### Arguments
- **x** A string (cnj)

## clean_id

### Description
Remove separators from lawsuit IDs

### Usage
```r
clean_id(id)
```

### Arguments
- **id** One or more lawsuit IDs

## escape_unicode

### 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
```r
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

{  
  extract_parts("001040620018260004", "N")  
  extract_parts("001040620018260004", c("N", "A", "O"))  
}

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

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

`lsos(pos = 1L, patternL, orderNby = "size"L, decreasing = TRUEL, head = TRUEL, n = 1P)`

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()

pvec  
*Verbose, parallel, and safe map-like*

**Description**

Using the same argument notation as `purrr::map()`, this function iterates over a list of inputs `.x`, applying `.f` to each element. It returns a tibble with the id, whether the function returned an error and the output.

**Usage**

```
pvec(.x, .f, ..., .cores = get_cores(), .progress = TRUE, .flatten = FALSE, .options = future_options())
```
reais

**Arguments**

- `.x` A list or atomic vector
- `.f` A function, formula, or atomic vector (see `purrr::map()`)
- `...` Other parameters passed on to `.f`
- `.cores` Number of cores to use when multiprocessing
- `.progress` Whether or not to display progress
- `.flatten` If TRUE, the errors are filtered from the output, and the returned object is flattened (a vector, a list, or a tibble)
- `.options` Options passed on to `furrr::future_map()` (furrr::future_options() by default)

**Value**

A tibble with 3 columns: input, return, and output

**See Also**

- `purrr::map()`, `furrr::future_map()`, `furrr::future_options()`

---

**Description**

Convert brazilian currency values (text) to numeric

**Usage**

```r
code("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**
```
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**
```
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A data frame</td>
</tr>
<tr>
<td>col</td>
<td>Column name or position (see <code>tidyr::separate()</code>)</td>
</tr>
<tr>
<td>...</td>
<td>Other arguments passed on to <code>tidyr::separate()</code></td>
</tr>
</tbody>
</table>
tabela | Produce frequency and relative frequency tables

Description

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

Usage

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
use_pipe

Examples

```r
## 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)
```

---

**use_pipe**  
*Add pipe template*

---

**Description**

Adds pipe template to package documentation.

**Usage**

```r
use_pipe(pkg = ".")
```

**Arguments**

- pkg  
  Package description (can be path or package name)
**verify_cnj**

Validate Brazilian lawsuits identification number on vectorial scale.

**Description**

Verifies if a brazilian lawsuit identification is a cnj number.

**Usage**

`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**

`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
furrr::future_map(), 11
furrr::future_options(), 11

gather_subjects, 9

httr::GET(), 6, 7
httr::POST(), 6, 7

lsos, 9

pattern_cnj, 10
precision, 10
purrr::map(), 10, 11
pvec, 10

reais, 11
rm_accent, 12

sample_cnj, 12
separate_cnj, 13

tabela, 14

test_fun, 14
tidyr::separate(), 13

use_pipe, 15
utils::glob2rx(), 9
utils::head(), 9
verify_cnj, 16
write_data, 16