Package ‘campfin’

October 20, 2023

**Type**  Package

**Title**  Wrangle Campaign Finance Data

**Version**  1.0.11

**Description**  Explore and normalize American campaign finance data. Created by the Investigative Reporting Workshop to facilitate work on The Accountability Project, an effort to collect public data into a central, standard database that is more easily searched: [https://publicaccountability.org/](https://publicaccountability.org/).

**License**  CC BY 4.0

**URL**  [https://github.com/irworkshop/campfin](https://github.com/irworkshop/campfin), [https://irworkshop.github.io/campfin/](https://irworkshop.github.io/campfin/)

**BugReports**  [https://github.com/irworkshop/campfin/issues](https://github.com/irworkshop/campfin/issues)

**Depends**  R (>= 3.2)

**Imports**  dplyr (>= 0.8.3), fs (>= 1.3.1), ggplot2 (>= 3.2.1), glue (>= 1.3.1), htr (>= 1.4.1), lubridate (>= 1.7.4), magrittr (>= 1.5), purr (>= 0.3.2), readr (>= 1.3.1), rlang (>= 0.4.0), scales (>= 1.0.0), stringdist (>= 0.9.5.2), stringr (>= 1.4.0), tibble (>= 2.1.3)

**Suggests**  covr (>= 3.3.2), knitr (>= 1.23), rmarkdown (>= 1.14), spelling (>= 2.1), testthat (>= 2.1.0), usethis (>= 1.6.0)

**VignetteBuilder**  knitr

**Encoding**  UTF-8

**Language**  en-US

**LazyData**  true

**RoxygenNote**  7.2.3

**NeedsCompilation**  no

**Author**  Kiernan Nicholls [aut, cre, cph], Investigative Reporting Workshop [cph], Yanqi Xu [aut], Schuyler Erle [cph]
R topics documented:

Maintainer  Kiernan Nicholls <kiernann@protonmail.com>
Repository  CRAN
Date/Publication  2023-10-20 11:00:02 UTC

R topics documented:

abbrev_full  .......................................................... 3
abbrev_state .......................................................... 4
add_prop ................................................................. 5
all_files_new .......................................................... 6
check_city ............................................................... 6
col_date_mdy ............................................................ 7
col_stats ................................................................. 8
count_diff ............................................................... 9
count_in ................................................................. 10
count_na ................................................................. 11
count_out ............................................................... 11
dark2 .......................................................... 12
expand_abbrev .......................................................... 13
expand_state ............................................................. 14
explore_plot .............................................................. 14
extra_city ................................................................. 15
fetch_city ................................................................. 16
file_age ................................................................. 16
file_encoding ............................................................ 17
flag_dupes ................................................................. 17
flag_na ................................................................. 18
flush_memory ............................................................ 19
guess_delim ............................................................... 19
invalid_city .............................................................. 20
invert_named ............................................................. 20
is_abbrev ................................................................. 21
is_binary ................................................................. 21
is_even ................................................................. 22
keypad_convert .......................................................... 22
most_common ............................................................. 23
na_in ................................................................. 24
na_out ................................................................. 24
na_rep ................................................................. 25
non_ascii ................................................................. 26
normal_address .......................................................... 26
normal_city .............................................................. 27
normal_phone ........................................................... 28
normal_state ............................................................ 29
normal_zip ............................................................... 30
path.abbrev .............................................................. 31
progress_table ........................................................... 32
**Description**

Create or use a named vector (c("full" = "abb")) and pass it to `stringr::str_replace_all()`. The full argument is surrounded with `\b` to capture only isolated intended full versions. Note that the built-in `usps_street`, `usps_city`, and `usps_state` dataframes have the columns reversed from what this function needs (to work by default with the counterpart `expand_abbrev()`).

**Usage**

```r
abbrev_full(x, full = NULL, rep = NULL, end = FALSE)
```
Arguments

x  A vector containing full words.

full  One of three objects: (1) A dataframe with full strings in the first column and corresponding abbreviations in the second column; (2) a named vector, with full strings as names for their respective abbreviations (e.g., c("full" = "abb")); or (3) an unnamed vector of full words with an unnamed vector of abbreviations in the rep argument.

rep  If full is an unnamed vector, a vector of abbreviations strings for each full word in abb.

data  logical; if TRUE, then the $ regular expression will be used to only replace words at the end of a string (such as "ROAD" in a street address). If FALSE (default), then the \b regular expression will target all instances of full to be replaced with rep.

Value

The vector x with full words replaced with their abbreviations.

See Also

Other geographic normalization functions: abbrev_state(), check_city(), expand_abbrev(), expand_state(), fetch_city(), normal_address(), normal_city(), normal_state(), normal_zip(), str_normal()

Examples

abbrev_full("MOUNT VERNON", full = c("MOUNT" = "MT"))
abbrev_full("123 MOUNTAIN ROAD", full = usps_street)
abbrev_full("123 MOUNTAIN ROAD", full = usps_street, end = TRUE)
abbrev_full("Vermont", full = state.name, rep = state.abb)
Value

The 2-letter USPS abbreviation of for state names (e.g., "VT").

See Also

Other geographic normalization functions: abbrev_full(), check_city(), expand_abbrev(), expand_state(), fetch_city(), normal_address(), normal_city(), normal_state(), normal_zip(), str_normal()

Examples

abbrev_state(full = state.name)
abbrev_state(full = c("new mexico", "france"))

add_prop(.data, n, sum = FALSE)

Arguments

.data A data frame with a count column.
.n The column name with a count, usually n from dplyr::count().
.sum Should cumsum() be called on the new p column.

Details

mean(x %in% y)

Value

A data frame with the new column p.

Examples

add_prop(dplyr::count(ggplot2::diamonds, cut))
all_files_new  Check if all files in a directory are new

Description
Tests whether all the files in a given directory have a modification date equal to the system date. Useful when repeatedly running code with a lengthy download stage. Many state databases are updated daily, so new data can be helpful but not always necessary. Set this function in an if statement.

Usage
all_files_new(path, glob = NULL, ...)

Arguments
- **path**: The path to a directory to check.
- **glob**: A pattern to search for files (e.g., "*.csv").
- **...**: Additional arguments passed to `fs::dir_ls()`.

Value
logical; Whether all() files in the directory have a modification date equal to today.

Examples
```r
tmp <- tempdir()
file.create(tempfile(pattern = as.character(1:5)))
all_files_new(tmp)
```

check_city  Check whether an input is a valid place with Google Maps API

Description
Check whether a place is a valid place or misspelling by matching against the Google Geocoding search result. Use the `httr::GET()` to send a request to the Google Maps API for geocoding information. The query will concatenate all the geographical information that is passed in into a long string. Then the function pulls the `formatted_address` endpoint of the API results and then identifies and extracts the long name field from the API `locality` result and compare it against the input to see if the input and output match up. Note that you will need to pass in your Google Maps Place API key to the key argument.

Usage
check_city(city = NULL, state = NULL, zip = NULL, key = NULL, guess = FALSE)
Arguments

- **city**: A string of city name to be submitted to the Geocode API.
- **state**: Optional. The state associated with the city.
- **zip**: Optional. Supply a string of ZIP code to increase precision.
- **key**: A character string to be passed into key. Save your key as "GEOCODE_KEY" using Sys.setenv() or by editing your .Renviron file.
- **guess**: logical; Should the function return a single row tibble containing the original data sent and the multiple components returned by the Geocode API.

Value

A logical value by default. If the city returned by the API comes back the same as the city input, the function will evaluate to TRUE, in all other circumstances (including API errors) FALSE is returned.

If the `guess` argument is set to TRUE, a tibble with 1 row and six columns is returned:

- **original_city**: The city value sent to the API.
- **original_state**: The state value sent to the API.
- **original_zip**: The zip value sent to the API.
- **check_city_flag**: logical; whether the guessed city matches.
- **guess_city**: The legal city guessed by the API.
- **guess_place**: The generic locality guessed by the API.

See Also

https://developers.google.com/maps/documentation/geocoding/overview?csw=1

Other geographic normalization functions: abbrev_full(), abbrev_state(), expand_abbrev(), expand_state(), fetch_city(), normal_address(), normal_city(), normal_state(), normal_zip(), str_normal()
Value

A POSIXct vector.

Examples

```r
col_stats(dplyr::storms, dplyr::n_distinct)
col_stats(dplyr::storms, campfin::count_na)
```

---

**col_stats**

*Apply a statistic function to all column vectors*

**Description**

Apply a counting summary function like `dplyr::n_distinct()` or `count_na()` to every column of a data frame and return the results along with a *percentage* of that value.

**Usage**

```r
col_stats(data, fun, print = TRUE)
glimpse_fun(data, fun, print = TRUE)
```

**Arguments**

- `data`  
  A data frame to glimpse.
- `fun`  
  A function to map to each column.
- `print`  
  logical; Should all columns be printed as rows?

**Value**

A tibble with a row for every column with the count and proportion.

**Examples**

```r
col_stats(dplyr::storms, dplyr::n_distinct)
col_stats(dplyr::storms, campfin::count_na)
```
count_diff

Count set difference

Description

Find the length of the set of difference between x and y vectors.

Usage

count_diff(x, y, ignore.case = FALSE)

Arguments

x A vector to check.

y A vector to compare against.

ignore.case logical; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

Details

sum(x %out% y)

Value

The number of unique values of x not in y.

See Also

Other counting wrappers: count_in(), count_na(), count_out(), na_in(), na_out(), na_rep(), prop_distinct(), prop_in(), prop_na(), prop_out(), what_in(), what_out()

Examples

# only unique values are checked
count_diff(c("VT", "NH", "ZZ", "ZZ", "ME"), state.abb)
count_in

Description

Count the total values of x that are %in% the vector y.

Usage

count_in(x, y, na.rm = TRUE, ignore.case = FALSE)

Arguments

x  A vector to check.
y  A vector to compare against.
na.rm  logical; Should NA be ignored?
ignore.case  logical; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

Details

sum(x %out% y)

Value

The sum of x present in y.

See Also

Other counting wrappers: count_diff(), count_na(), count_out(), na_in(), na_out(), na_rep(), prop_distinct(), prop_in(), prop_na(), prop_out(), what_in(), what_out()

Examples

count_in(c("VT", "NH", "ZZ", "ME"), state.abb)
count_na  Count missing

Description
Count the total values of x that are NA.

Usage
count_na(x)

Arguments
x A vector to check.

Details
sum(is.na(x))

Value
The sum of x that are NA

See Also
Other counting wrappers: count_diff(), count_in(), count_out(), na_in(), na_out(), na_rep(), prop_distinct(), prop_in(), prop_na(), prop_out(), what_in(), what_out()

Examples
count_na(c("VT", "NH", NA, "ME"))

count_out  Count out

Description
Count the total values of x that are %out% of the vector y.

Usage
count_out(x, y, na.rm = TRUE, ignore.case = FALSE)
Arguments

- **x**: A vector to check.
- **y**: A vector to compare against.
- **na.rm**: logical; Should NA be ignored?
- **ignore.case**: logical; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

Details

\[ \text{sum}(x \ %out\% y) \]

Value

The sum of \( x \) absent in \( y \).

See Also

Other counting wrappers: `count_diff()`, `count_in()`, `count_na()`, `na_in()`, `na_out()`, `na_rep()`, `prop_distinct()`, `prop_in()`, `prop_na()`, `prop_out()`, `what_in()`, `what_out()`

Examples

`count_out(c("VT", "NH", "ZZ", "ME"), state.abb)`

---

dark2

**Dark Color Palette**

Description

The Dark2 brewer color palette

Usage

`dark2`

Format

A named character vector of hex color codes (length 8).
**Description**

Create or use a named vector (c("abb" = "rep")) and pass it to stringr::str_replace_all(). The `abb` argument is surrounded with `\b` to capture only isolated abbreviations. To be used inside `normal_address()` and `normal_city()` with `usps_street` and `usps_city`, respectively.

**Usage**

```r
expand_abbrev(x, abb = NULL, rep = NULL)
```

**Arguments**

- `x`: A vector containing abbreviations.
- `abb`: One of three objects: (1) A dataframe with abbreviations in the first column and corresponding replacement strings in the second column; (2) a named vector, with abbreviations as names for their respective replacements (e.g., c("abb" = "rep")); or (3) an unnamed vector of abbreviations with an unnamed vector of replacements in the `rep` argument.
- `rep`: If `abb` is an unnamed vector, a vector of replacement strings for each abbreviation in `abb`.

**Value**

The vector `x` with abbreviation replaced with their full version.

**See Also**

Other geographic normalization functions: `abbrev_full()`, `abbrev_state()`, `check_city()`, `expand_state()`, `fetch_city()`, `normal_address()`, `normal_city()`, `normal_state()`, `normal_zip()`, `str_normal()`

**Examples**

```r
expand_abbrev(x = "MT VERNON", abb = c("MT" = "MOUNT"))
expand_abbrev(x = "VT", abb = state.abb, rep = state.name)
expand_abbrev(x = "Low FE Level", abb = tibble::tibble(x = "FE", y = "Iron"))
```
**expand_state**  
*Expand US state names*

**Description**
This function is used to first normalize an abb and then call `expand_abbrev()` using `valid_state` and `valid_name` as the abb and rep arguments.

**Usage**
```r
expand_state(abb)
```

**Arguments**
- `abb`  
  A abb US state name character vector (e.g., "Vermont").

**Value**
The 2-letter USPS abbreviation of state names (e.g., "VT").

**See Also**
Other geographic normalization functions: `abbrev_full()`, `abbrev_state()`, `check_city()`, `expand_abbrev()`, `fetch_city()`, `normal_address()`, `normal_city()`, `normal_state()`, `normal_zip()`, `str_normal()`

**Examples**
```r
expand_state(abb = state.abb)
exapand_state(abb = c("nm", "fr"))
```

---

**explore_plot**  
*Create Basic Barplots*

**Description**
This function simply wraps around `ggplot2::geom_col()` to take a dataframe and categorical variable to return a custom barplot ggplot object. The bars are arranged in descending order and are limited to the 8 most frequent values.

**Usage**
```r
explore_plot(data, var, nbar = 8, palette = "Dark2", na.rm = TRUE)
```
Arguments

data  The data frame to explore.
var   A variable to plot.
nbar  The number of bars to plot. Always shows most common values.
palette The color palette passed to \texttt{ggplot2::scale_fill_brewer}.
na.rm logical: Should \texttt{NA} values of \texttt{var} be removed?

Value

A \texttt{ggplot} barplot object. Can then be combined with other \texttt{ggplot} layers with \texttt{+} to customize.

Examples

\begin{verbatim}
explore_plot(iris, Species)
\end{verbatim}

\begin{verbatim}

\end{verbatim}
fetch_city  Return Closest Match Result of Cities from Google Maps API

Description

Use the `httr::GET()` to send a request to the Google Maps API for geocoding information. The query will concatenate all the geographical information that is passed in into a single string. Then the function pulls the `formatted_address` endpoint of the API results and extracts the first field of the result. Note that you will need to pass in your Google Maps Place API key with the `key` argument.

Usage

```r
fetch_city(address = NULL, key = NULL)
```

Arguments

- `address`: A vector of street addresses. Sent to the API as one string.
- `key`: A character containing your alphanumeric Google Maps API key.

Value

A character vector of formatted address endpoints from Google. This will include all the fields from street address, city, state/province, zipcode/postal code to country/regions. `NA_character_` is returned for all errored API calls.

See Also

- https://developers.google.com/maps/documentation/geocoding/overview?csw=1
- Other geographic normalization functions: `abbrev_full()`, `abbrev_state()`, `check_city()`, `expand_abbrev()`, `expand_state()`, `normal_address()`, `normal_city()`, `normal_state()`, `normal_zip()`, `str_normal()`

file_age  File modification date age

Description

The period of time since a system file was modified.

Usage

```r
file_age(...)```
Arguments

Arguments passed to `file.info()`, namely character vectors containing file paths. Tilde-expansion is done: see `path.expand()`.

Value

A Period class object.

Examples

```r
file_age(system.file("README.md", package = "campfin"))
```

---

### file_encoding

**File Encoding**

**Description**

Call the `file` command line tool with option `-i`.

**Usage**

```r
file_encoding(path)
```

**Arguments**

- **path**
  
  A local file path or glob to check.

**Value**

A tibble of file encoding.

---

### flag_dupes

**Flag Duplicate Rows With New Column**

**Description**

This function uses `dplyr::mutate()` to create a new `dupe_flag` logical variable with TRUE values for any record duplicated more than once.

**Usage**

```r
flag_dupes(data, ..., .check = TRUE, .both = TRUE)
```
flag_na

**Arguments**

- `data` A data frame to flag.
- `...` Arguments passed to `dplyr::select()` (needs to be at least `dplyr::everything()`).
- `.check` Whether the resulting column should be summed and removed if empty.
- `.both` Whether to flag both duplicates or just subsequent.

**Value**

A data frame with a new `dupe_flag` logical variable.

**Examples**

```r
flag_duplicates(iris, dplyr::everything())
flag_duplicates(iris, dplyr::everything(), .both = FALSE)
```

---

**flag_na**

*Flag Missing Values With New Column*

**Description**

This function uses `dplyr::mutate()` to create a new `na_flag` logical variable with TRUE values for any record missing *any* value in the selected columns.

**Usage**

```r
flag_na(data, ...)
```

**Arguments**

- `data` A data frame to flag.
- `...` Arguments passed to `dplyr::select()` (needs to be at least `dplyr::everything()`).

**Value**

A data frame with a new `na_flag` logical variable.

**Examples**

```r
flag_na(dplyr::starwars, hair_color)
```
flush_memory  

**Flush Garbage Memory**

**Description**
Run a full `gc()` a number of times.

**Usage**

```r
flush_memory(n = 1)
```

**Arguments**

- `n` The number of times to run `gc()`.

---

guess_delim  

**Guess the delimiter of a text file**

**Description**
Taken from code used in `vroom::vroom()` with automatic reading.

**Usage**

```r
guess_delim(file, delims = c(",", "\t", "|", ";"), string = FALSE)
```

**Arguments**

- `file` Either a path to a file or character string (with at least one newline character).
- `delims` The vector of single characters to guess from. Defaults to: comma, tab, pipe, or semicolon.
- `string` Should the file be treated as a string regardless of newline.

**Value**

The single character guessed as a delimiter.

**Source**

[https://github.com/tidyverse/vroom/blob/85143f7a417376eaf0e2037ca9575f637e4346c2/R/vroom.R#L288](https://github.com/tidyverse/vroom/blob/85143f7a417376eaf0e2037ca9575f637e4346c2/R/vroom.R#L288)
Examples

```r
guess_delim(system.file("extdata", "vt_contribs.csv", package = "campfin"))
guess_delim("ID;FirstName;MI;LastName;JobTitle", string = TRUE)
guess_delim("a|b|c
1|2|3
")
```

---

### invalid_city

#### Invalid City Names

A custom vector containing common invalid city names.

### Usage

```r
invalid_city
```

### Format

A vector of length 54.

---

### invert_named

#### Invert a named vector

Invert the names and elements of a vector, useful when using named vectors as the abbreviation arguments both of `expand_abbrev()` and `abbrev_full()` (or their parent normalization functions like `normal_address()`)

### Usage

```r
invert_named(x)
```

### Arguments

- **x**
  
  A named vector.

### Value

A named vector with names in place of elements and *vice versa*.

### Examples

```r
invert_named(x = c("name" = "element"))
```
is_abbrev

**Check if abbreviation**

**Description**

To return a value of TRUE, (1) the first letter of `abb` must match the first letter of `full`, (2) all letters of `abb` must exist in `full`, and (3) those letters of `abb` must be in the same order as they appear in `full`.

**Usage**

```r
is_abbrev(abb, full)
```

**Arguments**

- `abb` A suspected abbreviation
- `full` A long form string to test against

**Value**

logical; whether `abb` is a potential abbreviation of `full`

**Examples**

```r
is_abbrev(abb = "BRX", full = "BRONX")
is_abbrev(abb = state.abb, full = state.name)
is_abbrev(abb = "NOLA", full = "New Orleans")
is_abbrev(abb = "FE", full = "Iron")
```

is_binary

**Check if Binary**

**Description**

Uses `dplyr::n_distinct()` to check if there are only two unique values.

**Usage**

```r
is_binary(x, na.rm = TRUE)
```

**Arguments**

- `x` A vector.
- `na.rm` logical; Should NA be ignored, TRUE by default.
**Value**

TRUE if only 2 unique values.

**Examples**

```r
if (is_binary(x <- c("Yes", "No"))) x == "Yes"
```

---

**is_even**

*Check if even*

---

**Description**

Check if even

**Usage**

```r
is_even(x)
```

**Arguments**

- `x` A numeric vector.

**Value**

logical; Whether the integer is even or odd.

**Examples**

```r
is_even(1:10)
is_even(10L)
```

---

**keypad_convert**

*Convert letters or numbers to their keypad counterpart*

---

**Description**

This function works best when converting numbers to letters, as each number only has a single possible letter. For each letter, there are 3 or 4 possible letters, resulting in a number of possible conversions. This function was intended to convert phonetic telephone numbers to their valid numeric equivalent; when used in this manner, each letter in a string can be lazily replaced without changing the rest of the string.

**Usage**

```r
keypad_convert(x, ext = FALSE)
```
Arguments

- **x**: A vector of characters or letters.
- **ext**: logical; Should extension text be converted to numbers. Defaults to `FALSE` and matches `x`, `ext`, and extension followed by a space or number.

Details

When replacing letters, this function relies on the feature of `stringr::str_replace_all()` to work with named vectors (`c("A" = "2")`).

Value

If a character vector is supplied, a vector of each elements numeric counterpart is returned. If a numeric vector (or a completely coercible character vector) is supplied, then a list is returned, each element of which contacts a vector of letters for each number.

Examples

```r
keypad_convert("1-800-CASH-NOW ext123")
keypad_convert(c("abc", "123"))
keypad_convert(letters)
```

---

**most_common**  
Find most common values

Description

From a character vector, which values are most common?

Usage

most_common(x, n = 6)

Arguments

- **x**: A vector.
- **n**: Number of values to return.

Value

Sorted vector of `n` most common values.

Examples

```r
most_common(iris$Species, n = 1)
```
na_in

Remove in

Description
Set NA for the values of x that are %in% the vector y.

Usage
na_in(x, y, ignore.case = FALSE)

Arguments
- x: A vector to check.
- y: A vector to compare against.
- ignore.case: logical; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

Value
The vector x missing any values in y.

See Also
Other counting wrappers: count_diff(), count_in(), count_na(), count_out(), na_out(), na_rep(), prop_distinct(), prop_in(), prop_na(), prop_out(), what_in(), what_out()

Examples
na_in(c("VT", "NH", "ZZ", "ME"), state.abb)
na_in(1:10, seq(1, 10, 2))

na_out

Remove out

Description
Set NA for the values of x that are %out% of the vector y.

Usage
na_out(x, y, ignore.case = FALSE)
### Argument

- **x**: A vector to check.
- **y**: A vector to compare against.
- **ignore.case**: logical; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

### Value

The vector `x` missing any values not in `y`.

### See Also

Other counting wrappers: `count_diff()`, `count_in()`, `count_na()`, `count_out()`, `na_in()`, `na_rep()`, `prop_distinct()`, `prop_in()`, `prop_na()`, `prop_out()`, `what_in()`, `what_out()`

### Examples

```r
na_out(c("VT", "NH", "ZZ", "ME"), state.abb)
na_out(1:10, seq(1, 10, 2))
```

---

### Description

Set `NA` for the values of `x` that contain a single repeating character and no other characters.

### Usage

```r
na_rep(x, n = 0)
```

### Arguments

- **x**: A vector to check.
- **n**: The minimum number times a character must repeat. If 0, the default, then any string of one character will be replaced with `NA`. If greater than 0, the string must contain greater than `n` number of repetitions.

### Details

Uses the regular expression `"^\(.\)\1+$"`.

### Value

The vector `x` with `NA` replacing repeating character values.
See Also

Other counting wrappers: `count_diff()`, `count_in()`, `count_na()`, `count_out()`, `na_in()`, `na_out()`, `prop_distinct()`, `prop_in()`, `prop_na()`, `prop_out()`, `what_in()`, `what_out()`

Examples

```r
na_rep(c("VT", "NH", "ZZ", "ME"))
```

---

### non_ascii

**Show non-ASCII lines of file**

**Description**

Show non-ASCII lines of file

**Usage**

```r
non_ascii(path, highlight = FALSE)
```

**Arguments**

- `path`: The path to a text file to check.
- `highlight`: A function used to add ANSI escapes to highlight bytes.

**Value**

Tibble of line locations.

**Examples**

```r
non_ascii(system.file("README.md", package = "campfin"))
```

---

### normal_address

**Normalize street addresses**

**Description**

Return consistent version of a US Street Address using `stringr::str_*()` functions. Letters are capitalized, punctuation is removed or replaced, and excess whitespace is trimmed and squished. Optionally, street suffix abbreviations ("AVE") can be replaced with their long form ("AVENUE"). Invalid addresses from a vector can be removed (possibly using `invalid_city`) as well as single (repeating) character strings ("XXXXXX").
normal_address

Usage

normal_address(
  address,
  abbs = NULL,
  na = c("", "NA"),
  punct = "",
  na_rep = FALSE,
  abb_end = TRUE
)

Arguments

address A vector of street addresses (ideally without city, state, or postal code).
abbs A named vector or two-column data frame (like usps_street) passed to expand_abbrev().
  See ?expand_abbrev for the type of object structure needed.
na A character vector of values to make NA (like invalid_city).
punct A character value with which to replace all punctuation.
na_rep logical; If TRUE, replace all single digit (repeating) strings with NA.
abb_end logical; Should only the last word the string be abbreviated with the abbs argument?
  Passed to the end argument of str_normal().

Value

A vector of normalized street addresses.

See Also

Other geographic normalization functions: abbrev_full(), abbrev_state(), check_city(), expand_abbrev(),
  expand_state(), fetch_city(), normal_city(), normal_state(), normal_zip(), str_normal()

Examples

normal_address("P.O. #123, C/O John Smith", abbs = usps_street)
normal_address("12east 2nd street, #209", abbs = usps_street, abb_end = FALSE)

normal_city

Normalize city names

Description

Return consistent version of a city names using stringr::str_*() functions. Letters are capitalized,
hyphens and underscores are replaced with whitespace, other punctuation is removed, numbers
are removed, and excess whitespace is trimmed and squished. Optionally, geographic abbreviations
("MT") can be replaced with their long form ("MOUNT"). Invalid addresses from a vector can be
removed (possibly using invalid_city) as well as single (repeating) character strings ("XXXXXX").
normal_phone

Normalize phone number

Description

Take US phone numbers in any number of formats and try to convert them to a standard format.

Usage

```r
normal_phone(
  number,
  format = "(%a) %e-%l",
  na_bad = FALSE,
  convert = FALSE,
  rm_ext = FALSE
)
```
normal_state

Arguments

number A vector of phone number in any format.
format The desired output format, with %a representing the 3-digit area code, %e representing the 3-digit exchange, and %l representing the 4-digit line number. The punctuation between each part of the format is used in the normalized number (e.g., "(%a) %e-%l" or "%a-%e-%l").
na_bad logical; Should invalid numbers be replaced with NA.
convert logical; Should keypad_convert() be invoked to replace numbers with their keypad equivalent.
rm_ext logical; Should extensions be removed from the end of a number.

Value
A normalized telephone number.

Examples

normal_phone(number = c("916-225-5887"))

normal_state Normalize US State Abbreviations

Description
Return consistent version of a state abbreviations using stringr::str_*() functions. Letters are capitalized, all non-letters characters are removed, and excess whitespace is trimmed and squished, and then abbrev_full() is called with usps_state.

Usage

normal_state(
state,
abbreviate = TRUE,
na = c("" , "NA" ),
na_rep = FALSE,
valid = NULL
)

Arguments

state A vector of US state names or abbreviations.
abbreviate If TRUE (default), replace state names with the 2-digit abbreviation using the built-in state.abb and state.name vectors.
na A vector of values to make NA.
na_rep logical; If TRUE, make all single digit repeating strings NA (removes valid "AA" code for "American Armed Forces").
valid A vector of valid abbreviations to compare to and remove those not shared.
normal_zip

Value
A vector of normalized 2-digit state abbreviations.

See Also
Other geographic normalization functions: abbrev_full(), abbrev_state(), check_city(), expand_abbrev(), expand_state(), fetch_city(), normal_address(), normal_city(), normal_zip(), str_normal()

Examples

```r
normal_state(
  state = c("VT", "N/A", "Vermont", "XX", "ZA"),
  abbreviate = TRUE,
  na = c("", "NA"),
  na_rep = TRUE,
  valid = NULL
)
```

---

define normal_zip

 Normalize ZIP codes

Description
Return consistent version US ZIP codes using stringr::str_*() functions. Non-number characters are removed, strings are padded with zeroes on the left, and ZIP+4 suffixes are removed. Invalid ZIP codes from a vector can be removed as well as single (repeating) character strings.

Usage

```r
normal_zip(zip, na = c("", "NA"), na_rep = FALSE, pad = FALSE)
```

Arguments

- **zip**: A vector of US ZIP codes.
- **na**: A vector of values to pass to `na_in()`.
- **na_rep**: logical: If TRUE, `na_rep()` will be called. Please note that 22222, 44444, and 55555 valid ZIP codes that will not be removed.
- **pad**: logical: Should ZIP codes less than five digits be padded with a leading zero? Leading zeros (as are found in New England ZIP codes) are often dropped by programs like Microsoft Excel when parsed as numeric values.

Value
A character vector of normalized 5-digit ZIP codes.
See Also

Other geographic normalization functions: `abbrev_full()`, `abbrev_state()`, `check_city()`, `expand_abbrev()`, `expand_state()`, `fetch_city()`, `normal_address()`, `normal_city()`, `normal_state()`, `str_normal()`

Examples

```r
normal_zip(
  zip = c("05672-5563", "N/A", "05401", "5819", "00000"),
  na = c("", "NA"),
  na_rep = TRUE,
  pad = TRUE
)
```

---

**path.abbrev**

Abbreviate a file path

Description

This is an inverse of `path.expand()`, which replaces the home directory or project directory with a tilde.

Usage

```r
path.abbrev(path, dir = fs::path_wd())
```

Arguments

- **path**  
  Character vector containing one or more full paths.
- **dir**  
  The directory to replace with ~. Defaults to `fs::path_wd()`.

Value

Abbreviated file paths.

Examples

```r
print(fs::path_wd("test"))
path.abbrev(fs::path_wd("test"))
```
progress_table  
*Create a progress table*

**Description**
Create a tibble with rows for each stage of normalization and columns for the various statistics most useful in assessing the progress of each stage.

**Usage**

```r
progress_table(..., compare)
```

**Arguments**
- `...` Any number of vectors to check.
- `compare` A vector to compare each of `...` against. Useful with `valid_zip`, `valid_state` (`valid_name`), or `valid_city`.

**Value**
A table with a row for each vector in `...`.

**Examples**

```r
progress_table(state.name, toupper(state.name), compare = valid_name)
```

---

prop_distinct  
*Proportion missing*

**Description**
Find the proportion of values of `x` that are distinct.

**Usage**

```r
prop_distinct(x)
```

**Arguments**
- `x` A vector to check.

**Details**

```r
length(unique(x))/length(x)
```
prop_in

Value

The ratio of distinct values \( x \) to total values of \( x \).

See Also

Other counting wrappers: `count_diff()`, `count_in()`, `count_na()`, `count_out()`, `na_in()`, `na_out()`, `na_rep()`, `prop_in()`, `prop_na()`, `prop_out()`, `what_in()`, `what_out()`

Examples

```r
prop_in(c("VT", "VT", NA, "ME"))
```

---

Description

Find the proportion of values of \( x \) that are \%in\% the vector \( y \).

Usage

```r
prop_in(x, y, na.rm = TRUE, ignore.case = FALSE)
```

Arguments

- `x` A vector to check.
- `y` A vector to compare against.
- `na.rm` logical; Should NA be ignored?
- `ignore.case` logical; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

Details

```r
mean(x %in% y)
```

Value

The proportion of \( x \) present in \( y \).

See Also

Other counting wrappers: `count_diff()`, `count_in()`, `count_na()`, `count_out()`, `na_in()`, `na_out()`, `na_rep()`, `prop_in()`, `prop_distinct()`, `prop_na()`, `prop_out()`, `what_in()`, `what_out()`

Examples

```r
prop_in(c("VT", "NH", "ZZ", "ME"), state.abb)
```
prop_na

Proportion missing

Description
Find the proportion of values of x that are NA.

Usage
prop_na(x)

Arguments
x  A vector to check.

Details
mean(is.na(x))

Value
The proportion of values of x that are NA.

See Also
Other counting wrappers: `count_diff()`, `count_in()`, `count_na()`, `count_out()`, `na_in()`, `na_out()`, `na_rep()`, `prop_distinct()`, `prop_in()`, `prop_out()`, `what_in()`, `what_out()`

Examples
prop_na(c("VT", "NH", NA, "ME"))

prop_out

Proportion out

Description
Find the proportion of values of x that are %out% of the vector y.

Usage
prop_out(x, y, na.rm = TRUE, ignore.case = FALSE)
Arguments

- **x**: A vector to check.
- **y**: A vector to compare against.
- **na.rm**: logical; Should NA be ignored?
- **ignore.case**: logical; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

Details

\[
\text{mean}(x \%out\% y)
\]

Value

The proportion of \(x\) absent in \(y\).

See Also

Other counting wrappers: `count_diff()`, `count_in()`, `count_na()`, `count_out()`, `na_in()`, `na_out()`, `na_rep()`, `prop_distinct()`, `prop_in()`, `prop_na()`, `what_in()`, `what_out()`

Examples

```r
prop_out(c("VT", "NH", "ZZ", "ME"), state.abb)
```

---

**Description**

Read the first line of a delimited file as vector.

**Usage**

```r
read_names(file, delim = guess_delim(file))
```

**Arguments**

- **file**: Path to text file.
- **delim**: Character separating column names.

**Value**

Character vector of column names.

**Examples**

```r
read_names("date,lgl\n11/09/2016,TRUE")
```
rename_prefix  
**Convert data frame name suffixes to prefixes**

**Description**

When performing a `dplyr::left_join()`, the suffix argument allows the user to replace the default `.x` and `.y` that are appended to column names shared between the two data frames. This function allows a user to convert those suffixes to *prefixes*.

**Usage**

```r
rename_prefix(df, suffix = c(".x", ".y"), punct = TRUE)
```

**Arguments**

- **df**
  - A joined data frame.
- **suffix**
  - If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2. Will be converted to prefixes.
- **punct**
  - logical; Should punctuation at the start of the suffix be detected and placed at the end of the new prefix? TRUE by default.

**Value**

A data frame with new column names.

**Examples**

```r
a <- data.frame(x = letters[1:3], y = 1:3)
b <- data.frame(x = letters[1:3], y = 4:6)
df <- dplyr::left_join(a, b, by = "x", suffix = c(".a", ".b"))
rename_prefix(df, suffix = c(".a", ".b"), punct = TRUE)
```

---

**rx_break**  
**Form a word break regex pattern**

**Description**

Wrap a word in word boundary (\b) characters. Useful when combined with `stringr::str_which()` and `stringr::str_detect()` to match only entire words and not that word inside another word (e.g., "sting" and "testing").

**Usage**

```r
rx_break(pattern)
```
**Arguments**

- **pattern**: A regex pattern (a word) to wrap in `\b`.

**Value**

The a glue vector of `pattern` wrapped in `\b`.

**Examples**

- `rx_break("test")`
- `rx_break(state.abb[1:5])`

---

**rx_phone**  
*Phone number regex*

**Description**

The regex string to match US phone numbers in a variety of common formats.

**Usage**

- `rx_phone`

**Format**

- A character string (length 1).

---

**rx_state**  
*State regex*

**Description**

The regex string to extract state string preceding ZIP code.

**Usage**

- `rx_state`

**Format**

- A character string (length 1).
### rx_url

**URL regex**

**Description**

The regex string to match valid URLs.

**Usage**

```r
rx_url
```

**Format**

A character string (length 1).

---

### rx_zip

**ZIP code regex**

**Description**

The regex string to extract ZIP code from the end of address.

**Usage**

```r
rx_zip
```

**Format**

A character string (length 1).

---

### scale_x_truncate

**Truncate and wrap x-axis labels**

**Description**

Truncate the labels of a plot’s discrete x-axis labels so that the text does not overflow and collide with other bars.

**Usage**

```r
scale_x_truncate(n = 15, ...)
scale_x_wrap(width = 15, ...)
```
str_dist

Arguments

n                      The maximum width of string. Passed to `stringr::str_trunc()`.
...                    Additional arguments passed to `ggplot2::scale_x_discrete()`.
width                  Positive integer giving target line width in characters. A width less than or equal to 1 will put each word on its own line. Passed to `stringr::str_wrap()`.

Description

This function wraps around `stringdist::stringdist()`.

Usage

```r
str_dist(a, b, method = "osa", ...)
```

Arguments

a                      R object (target); will be converted by `base::as.character()`.
b                      R object (source); will be converted by `base::as.character()`.
method                 Method for distance calculation. The default is "osa."
...                    Other arguments passed to `stringdist::stringdist()`.

Value

The distance between string a and string b.

Examples

```r
str_dist(a = "BRULINGTN", b = "BURLINGTON")
```

str_normal

Normalize a character string

Description

The generic normalization that underpins functions like `normal_city()` and `normal_address()`. This function simply chains together three `stringr::str_*()` functions:

1. Convert to uppercase.
2. Replace punctuation with whitespaces.
3. Trim and squish excess whitespace.
Usage

str_normal(x, case = TRUE, punct = "", quote = TRUE, squish = TRUE)

Arguments

x A character string to normalize.
case logical; whether stringr::str_to_upper() should be called.
punct character; A character string to replace most punctuation with.
quote logical; whether stringr::str_replace_all() should be called on double quotes.
squish logical; whether stringr::str_squish() should be called.

Value

A normalized vector of the same length.

See Also

Other geographic normalization functions: abbrev_full(), abbrev_state(), check_city(), expand_abbrev(), expand_state(), fetch_city(), normal_address(), normal_city(), normal_state(), normal_zip()

Examples

str_normal(" TestING 123 example_test.String ")

Description

This function tests whether a single file has a modification date equal to the system date. Useful when repeatedly running code with a lengthy download stage. Many state databases are updated daily, so new data can be helpful but not always necessary. Set this function in an if statement.

Usage

this_file_new(path)

Arguments

path The path to a file to check.

Value

logical; Whether the file has a modification date equal to today.
Examples

tmp <- tempfile()
this_file_new(tmp)

url2path(url, dir)

Arguments

url The URL of a file to download.
dir The directory where the file will be downloaded.

Details

Useful in the destfile argument to download.file() to save a file with the same name as the URL's file name.

Value

The desired file path to a URL file.

Examples

url2path("https://floridalobbyist.gov/reports/llob.txt", tempdir())

url_file_size(url)

Description

Call http::HEAD() and return the number of bytes in the file to be downloaded.
Arguments
    url   The URL of the file to query.

Value
    The size of a file to be downloaded.

use_diary Create a new template data diary

Description
    Take the arguments supplied and put them into the appropriate places in a new template diary. Write
    the new template diary in the supplied directory.

Usage
    use_diary(
        st,
        type,
        author,
        path = "state/{st}/{type}/docs/{st}_{type}_diary.Rmd",
        auto = FALSE
    )

Arguments
    st   The USPS state abbreviation. State data only, no federal agencies.
    type The type of data, one of "contribs", "expends", "lobby", "contracts", "salary", or
          "voters".
    author The author name of the new diary.
    path The file path, relative to your working directory, where the diary file will be
          created. If you use NA, then the lines of the diary will be returned as a character
          vector. If you specify a character string file path that contains directories that do
          not exist then they will be created. By default, the path creates the diary in a
          directory that is expected by the Accountability Project GitHub repository.
    auto Must be set to TRUE for the diary to be created and opened.

Value
    The file path of new diary, invisibly.

Examples
    use_diary("VT", "contribs", "Kiernan Nicholls", NA, auto = FALSE)
    use_diary("DC", "expends", "Kiernan Nicholls", tempfile(), auto = FALSE)
usps_city

**USPS City Abbreviations**

**Description**
A curated and edited subset of `usps_street` containing the USPS abbreviations found in city names. Useful as the `geo_abbs` argument of `normal_city()`.

**Usage**
`usps_city`

**Format**
A tibble with 154 rows of 2 variables:
- `full` Primary Street Suffix
- `abb` Commonly Used Street Suffix or Abbreviation...

**Source**
USPS Appendix C1, *Street Abbreviations*

---

usps_state

**USPS State Abbreviations**

**Description**
A tibble containing the USPS.

**Usage**
`usps_state`

**Format**
A tibble with 62 rows of 2 variables:
- `full` Primary Street Suffix
- `abb` Commonly Used Street Suffix or Abbreviation...

**Source**
USPS Appendix B, *Two-Letter State Abbreviations*
usps_street  USPS Street Abbreviations

Description
A tibble containing common street suffixes or suffix abbreviations and their full equivalent. Useful as the add_abbs argument of normal_address().

Usage
usps_street

Format
A tibble with 325 rows of 3 variables:

full  Primary Street Suffix.

abb  Commonly Used Street Suffix or Abbreviation. ...

Source
USPS Appendix C1 Street Abbreviations.

valid_abb  US State Abbreviations

Description
The abb column of the usps_state tibble.

Usage
valid_abb

Format
A vector of 2-digit abbreviations (length 62).
### valid_city

**US City Names**

**Description**

The city column of the zipcodes tibble.

**Usage**

`valid_city`

**Format**

A sorted vector of unique city names (length 19,083).

---

### valid_name

**US State Names**

**Description**

The state column of the usps_state tibble.

**Usage**

`valid_name`

**Format**

A vector of state names (length 62).

**Details**

Contains 12 more names than `datasets::state.name`.

---

### valid_state

**US State Abbreviations**

**Description**

The abb column of the usps_state tibble.

**Usage**

`valid_state`

**Format**

A vector of 2-digit abbreviations (length 62).
### valid_zip

*Almost all of the valid USA ZIP Codes*

#### Description

The zip column of the geo tibble.

#### Usage

```r
valid_zip
```

#### Format

A sorted vector of 5-digit ZIP codes (length 44334).

### what_in

*Which in*

#### Description

Return the values of x that are `%in%` of the vector y.

#### Usage

```r
what_in(x, y, ignore.case = FALSE)
```

#### Arguments

- **x**: A vector to check.
- **y**: A vector to compare against.
- **ignore.case**: logical; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

#### Details

```r
x[which(x %in% y)]
```

#### Value

The elements of x that are `%in%` y.

#### See Also

Other counting wrappers: `count_diff()`, `count_in()`, `count_na()`, `count_out()`, `na_in()`, `na_out()`, `na_rep()`, `prop_distinct()`, `prop_in()`, `prop_na()`, `prop_out()`, `what_out()`
what_out

Examples

what_in(c("VT", "DC", NA), state.abb)

---

what_out Which out

Description

Return the values of x that are %out% of the vector y.

Usage

what_out(x, y, na.rm = TRUE, ignore.case = FALSE)

Arguments

x A vector to check.
y A vector to compare against.
na.rm logical; Should NA be ignored?
ignore.case logical; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

Details

x[which(x %out% y)]

Value

The elements of x that are %out% y.

See Also

Other counting wrappers: count_diff(), count_in(), count_na(), count_out(), na_in(), na_out(), na_rep(), prop_distinct(), prop_in(), prop_na(), prop_out(), what_in()

Examples

what_out(c("VT", "DC", NA), state.abb)
Description

This tibble is the third version of a popular zipcodes database. The original CivicSpace US ZIP Code Database was created by Schuyler Erle using ZIP code gazetteers from the US Census Bureau from 1999 and 2000, augmented with additional ZIP code information from the Census Bureau’s TIGER/Line 2003 data set. The second version was published as the zipcode::zipcode dataframe object. This version has dropped the latitude and longitude, reorganized columns, and normalize the city values with normal_city().

Usage

```
zipcodes
```

Format

A tibble with 44,336 rows of 3 variables:

- `city` Normalized city name.
- `state` Two letter state abbreviation.
- `zip` Five-digit ZIP Code. ...

Source


%out%

Inverted match

Description

%out% is an inverted version of the infix %in% operator.

Usage

```
x %out% table
```

Arguments

- `x` vector: the values to be matched. Long vectors are supported.
- `table` vector or NULL: the values to be matched against.
Details
%out% is currently defined as "%out%" <- function(x, table) match(x, table, nomatch = 0) == 0

Value
logical; if x is not present in table

Examples
c("A", "B", "3") %out% LETTERS
Index

* Simple Counting Wrappers
  progress_table, 32
* counting wrappers
  count_diff, 9
  count_in, 10
  count_na, 11
  count_out, 11
  na_in, 24
  na_out, 24
  na_rep, 25
  prop_distinct, 32
  prop_in, 33
  prop_na, 34
  prop_out, 34
  what_in, 46
  what_out, 47
* datasets
  dark2, 12
  extra_city, 15
  invalid_city, 20
  rx_phone, 37
  rx_state, 37
  rx_url, 38
  rx_zip, 38
  usps_city, 43
  usps_state, 43
  usps_street, 44
  valid_abb, 44
  valid_city, 45
  valid_name, 45
  valid_state, 45
  valid_zip, 46
  zipcodes, 48
* geographic normalization functions
  abbrev_full, 3
  abbrev_state, 4
  check_city, 6
  expand_abbrev, 13
  expand_state, 14
  fetch_city, 16
  normal_address, 26
  normal_city, 27
  normal_state, 29
  normal_zip, 30
  str_normal, 39
  %out%, 48
  abbrev_full, 3, 5, 7, 13, 14, 16, 27, 28, 30, 31, 40
  abbrev_full(), 4, 20, 29
  abbrev_state, 4, 4, 7, 13, 14, 16, 27, 28, 30, 31, 40
  add_prop, 5
  all(), 6
  all_files_new, 6
  base::as.character(), 39
  basename(), 41
  check_city, 4, 5, 6, 13, 14, 16, 27, 28, 30, 31, 40
  check_city(), 15
  col_date_mdy, 7
  col_date_usa(col_date_mdy), 7
  col_stats, 8
  count_diff, 9, 10–12, 24–26, 33–35, 46, 47
  count_in, 9, 10, 11, 12, 24–26, 33–35, 46, 47
  count_na, 9, 10, 11, 12, 24–26, 33–35, 46, 47
  count_na(), 8
  count_out, 9–11, 11, 24–26, 33–35, 46, 47
  cumsum(), 5
  dark2, 12
  datasets::state.name, 45
  download.file(), 41
dplyr::count(), 41
dplyr::everything(), 18
dplyr::left_join(), 36
dplyr::mutate(), 17, 18
dplyr::n_distinct(), 8, 21
dplyr::select(), 18
expand_abbrev, 4, 5, 7, 13, 14, 16, 27, 28, 30, 31, 40
expand_abbrev(), 3, 14, 20, 27
expand_state, 4, 5, 7, 13, 14, 16, 27, 28, 30, 31, 40
explore_plot, 14
extra_city, 15
fetch_city, 4, 5, 7, 13, 14, 16, 27, 28, 30, 31, 40
file.info(), 17
file_age, 16
file_encoding, 17
flag_dupes, 17
flag_na, 18
flush_memory, 19
fs::dir_ls(), 6
fs::path_wd(), 31
gc(), 19
ggplot2::geom_col(), 14
ggplot2::scale_x_discrete(), 39
glimpse_fun(col_stats), 8
guess_delim, 19
http::GET(), 6, 16
http::HEAD(), 41
invalid_city, 20, 26–28
invert_named, 20
is_abbrev, 21
is_binary, 21
is_even, 22
keypad_convert, 22
keypad_convert(), 29
most_common, 23
na_in, 9–12, 24, 25, 26, 33–35, 46, 47
na_in(), 30
na_out, 9–12, 24, 25, 26, 33–35, 46, 47
na_rep, 9–12, 24, 25, 26, 33–35, 46, 47
na_rep(), 30
non_ascii, 26
normal_address(), 13, 20, 39, 44
normal_city, 4, 5, 7, 13, 14, 16, 27, 28, 30, 31, 40
normal_city(), 13, 39, 43, 48
normal_phone, 28
normal_state, 4, 5, 7, 13, 14, 16, 27, 28, 29, 31, 40
normal_zip, 4, 5, 7, 13, 14, 16, 27, 28, 30, 30, 40
path.abbrev, 31
path.expand(), 17, 31
progress_table, 32
prop.table(), 5
prop_distinct, 9–12, 24–26, 32, 33–35, 46, 47
prop_in, 9–12, 24–26, 33, 34, 35, 46, 47
prop_na, 9–12, 24–26, 33, 34, 35, 46, 47
prop_out, 9–12, 24–26, 33, 34, 34, 34, 34, 46, 47
read_names, 35
readr::col_date(), 7
rename_prefix, 36
rx_break, 36
rx_phone, 37
rx_state, 37
rx_url, 38
rx_zip, 38
scale_x_truncate, 38
scale_x_wrap(scale_x_truncate), 38
str_dist, 39
str_normal, 4, 5, 7, 13, 14, 16, 27, 28, 30, 31, 39
str_normal(), 27
stringdist::stringdist(), 39
stringr::str_detect(), 36
stringr::str_replace_all(), 3, 13, 23, 40
stringr::str_squish(), 40
stringr::str_to_upper(), 40
stringr::str_trunc(), 39
stringr::str_which(), 36
stringr::str_wrap(), 39
this_file_new, 40
url2path, 41
url_file_size, 41
use_diary, 42
usps_city, 3, 13, 28, 43
usps_state, 3, 29, 43
usps_street, 3, 13, 27, 43, 44

valid_abb, 44
valid_city, 15, 32, 45
valid_name, 4, 14, 32, 45
valid_state, 4, 14, 32, 45
valid_zip, 32, 46

what_in, 9–12, 24–26, 33–35, 46, 47
what_out, 9–12, 24–26, 33–35, 46, 47

zipcodes, 48