Package ‘priceR’

March 23, 2020

Type Package
Title Microeconomics and Pricing Tools
Version 0.1.1
Imports dplyr, gsubfn, stringr, purrr, jsonlite, stats, lubridate,
curl, stringi
Author Steve Condylios [aut, cre] (<https://orcid.org/0000-0003-0599-844X>)
Maintainer Steve Condylios <steve.condylios@gmail.com>
BugReports https://github.com/stevecondylios/priceR/issues
License MIT + file LICENSE
URL https://github.com/stevecondylios/priceR
Description Functions to aid in microeconomic analysis and handling of price and currency data. This includes extraction of relevant data (e.g. from World Bank API, and other sources), data cleaning/parsing, and standardisation.
Encoding UTF-8
LazyData true
RoxygenNote 7.0.2
Suggests testthat
NeedsCompilation no
Repository CRAN
Date/Publication 2020-03-23 11:10:06 UTC

R topics documented:

priceR-package ............................................. 2
adjust_for_inflation ..................................... 2
convert_to_iso2Code .................................... 4
country_input_type ..................................... 5
currency_characters ..................................... 6
currency_to_numeric .................................... 7
PriceR-package

PriceR: Economics and Pricing Tools

Description

PriceR: Economics and Pricing Tools

Details

It has the goal of providing a quick and practical way of extracting numeric price and currency data from text fields, thus allowing faster and easier analysis of ostensibly numeric data.

Author(s)

Steve Condylios <steve.condylios@gmail.com>

Adjust for Inflation

Convert nominal prices into real prices

Description

Inflate/deflate prices from any year to any year, using World Bank inflation data and assumptions only where necessary. Typically used for converting past (nominal) values into current (real) values. This uses World Bank inflation data where available, but allows for both historical and future assumptions in extrapolation.
Usage

adjust_for_inflation(price, from_date, country, to_date, inflation_dataframe, countries_dataframe, extrapolate_future_method, future_averaging_period, future_rate, extrapolate_past_method, past_averaging_period, past_rate)

afi(
    price,
    from_date,
    country,
    to_date,
    inflation_dataframe,
    countries_dataframe,
    extrapolate_future_method,
    future_averaging_period,
    future_rate,
    extrapolate_past_method,
    past_averaging_period,
    past_rate
)

Arguments

price          A price (or prices).
from_date      A date(s) from which the prices will be converted.
country        A country or region in whose currency the prices are denominated.
to_date        A date(s) to which the prices will be converted.
inflation_dataframe
    The R object (list) representing the JSON retrieved by calling retrieve_inflation_data().
countries_dataframe
    The R object (data.frame) representing the JSON retrieved by calling show_countries().
extrapolate_future_method
    The extrapolation method that shall be used if extrapolation into the future is required. Options are ‘average’ or ‘rate’.
future_averaging_period
    The number of recent periods to average in order to extrapolate forward (if ‘average’ is method being used).
future_rate    An assumed rate of inflation to use for extrapolating forward (if ‘rate’ is method being used).
extrapolate_past_method
    The extrapolation method that shall be used if extrapolation from the earliest available data to some even earlier period is required.
past_averaging_period
    The number of periods back from the earliest available inflation data for a given country to average in order to extrapolate into the past (if ‘average’ is method being used).
past_rate      An assumed rate of inflation to use for extrapolating from the earliest available data to some even earlier period (if ‘rate’ is method being used).
**convert_to_iso2Code**

**Description**

`convert_to_iso2Code` accepts the type of country input and the country, and returns the relevant iso2Code

**Usage**

```r
convert_to_iso2Code(country_input_type_string, country, countries_dataframe)
```

**Arguments**

- `country_input_type_string`  
  Either "country_name" or "iso2Code" - use `country_input_type(country, countries_dataframe)` to determine or assign manually.

- `country`  
  A country/region name or iso2Code.

- `countries_dataframe`  
  The output of `show_countries()`

**Value**

A character vector containing a valid iso2Code
country_input_type

Examples

## Not run:

# Assign so as to save on API calls (recommended)
countries_dataframe <- show_countries()
country <- "Australia"
country_input_type_string <- country_input_type(country, countries_dataframe)
convert_to_iso2Code(country_input_type_string, country, countries_dataframe)
# [1] "AU"

country <- "AU"
country_input_type_string <- country_input_type(country, countries_dataframe)
convert_to_iso2Code(country_input_type_string, country, countries_dataframe)
# [1] "AU"

## End(Not run)

country_input_type

Determines whether country input is a country name or iso2Code

Description

Determines whether a string is a country name, an iso2Code, or invalid (not a World Bank API country/region)

Usage

country_input_type(country_input, countries_dataframe)

Arguments

country_input A country/region the user wishes to validate (string) E.g. "Australia".
countries_dataframe A dataframe containing available iso2Code and country_name (see show_countries()).

Value

A character vector
Examples

```r
## Not run:
# Assign so as to save on API calls - recommended
countries_dataframe <- show_countries()

country <- "Australia"
country_input_type(country, countries_dataframe)
# [1] "country_name"

country <- "AU"
country_input_type(country, countries_dataframe)
# [1] "iso2Code"

country <- "something other than a valid country name or iso2Code"
country_input_type(country, countries_dataframe)
# [1] "invalid"

## End(Not run)
```

---

currency_characters

Provide currency characters

Description

Provide currency characters

Usage

currency_characters()

Value

A character vector of currency symbols

Examples

currency_characters()
currency_to_numeric  Convert human readable currencies into numeric data

Description
Convert human readable currencies into numeric data

Usage
currency_to_numeric(currency_text)

Arguments
currency_text  Price or vector of prices

Value
A numeric vector

Examples
library(dplyr)
c($"134,345.05", "£22", "¥30000") %>% currency_to_numeric()
# [1] 134345 22 30000

extract_salary  Extract numeric salary from text data

Description
Extract numeric salary from text data. 'extract_salary' automatically converts weekly and hourly rates to amounts per annum.

Usage
extract_salary(salary_text, exclude_below, exclude_above, salary_range_handling, include_periodicity, hours_per_workday, days_per_workweek, working_weeks_per_year)

Arguments

salary_text  A character string, or vector of character strings.
exclude_below  A lower bound. Anything lower than this number will be replaced with NA.
exclude_above  An upper bound. Anything above this number will be replaced with NA.
salary_range_handling  A method of handling salary ranges. Defaults to returning an average of the range; can also be set to "max" or "min".
include_periodicity
Set to TRUE to return an additional column stating the detected periodicity in the character string. Periodicity is assumed to be 'Annual' unless evidence is found to the contrary.

hours_per_workday
Set assumed number of hours in the workday. Only affects annualisation of rates indentified as Daily. Default is 8 hours.

days_per_workweek
Set assumed number of days per workweek. Only affects annualisation of rates indentified as Daily. Default is 5 days.

working_weeks_per_year
Set assumed number of working weeks in the year. Only affects annualisation of rates indentified as Daily or Weekly. Default is 50 weeks.

Value
A data.frame of 1 column, or 2 columns if include_periodicity is set to TRUE

Examples

# Provide a salary string and 'extract_salary' and will extract the salary and return it
extract_salary("$160,000 per annum")
# 160000

# If a range is present, the average will be taken by default
extract_salary("$160,000 - $180000.00 per annum")
# 170000

# Take the 'min' or 'max' of a salary range by setting salary_range_handling parameter accordingly
extract_salary("$160,000 - $180000.00 per annum", salary_range_handling = "min")
# 160000

# Extract salaries from character string(s)
annual_salaries <- c("$160,000 - $180000.00 per annum",
"$145000 - $155000.00 per annum",
"$70000.00 - $90000 per annum",
"$70000.00 - $90000.00 per annum plus 15.4% super",
"$80000.00 per annum plus 15.4% super",
"60,000 - 80,000",
"$78,686 to $89,463 pa, plus 15.4% superannuation",
"80k - 100k")

extract_salary(annual_salaries)
# 170000 170000 150000 80000 53338 40008 70000 56055 90000
# Note the fifth, sixth, and eighth elements are averages including '15' (undesirable)
# Using exclude_below parameter avoids this (see below)
# Automatically detect, extract, and annualise daily rates
daily_rates <- c("$200 daily", "$400 - $600 per day", "Day rate negotiable dependent on experience")
extract_salary(daily_rates)
# 48000 120000 NA

# Automatically detect, extract, and annualise hourly rates
hourly_rates <- c("$80 - $100+ per hour", "APS6/EL1 hourly rate contract")
extract_salary(hourly_rates)
# 172800 6720
# Note 6720 is undesirable. Setting the exclude_below and exclude_above sensibly avoids this

salaries <- c(annual_salaries, daily_rates, hourly_rates)

# Setting lower and upper bounds provides a catch-all to remove unrealistic results
# Out of bounds values will be converted to NA
extract_salary(salaries, exclude_below = 20000, exclude_above = 600000)
# 170000 170000 150000 80000 80000 80000 70000 84074 90000 48000 120000 NA 172800 NA

# extract_salary automatically annualises hourly and daily rates
# It does so by making assumptions about the number of working weeks in a year,
# days per workweek, and hours per workday
# And the assumed number of hours per workday can be changed from the default (8)
# The assumed number of workdays per workweek can be changed from the default (5)
# The assumed number of working weeks in year can be changed from the default (50)
# E.g.
extract_salary(salaries, hours_per_workday = 7, days_per_workweek = 4,
# working_weeks_per_year = 46, exclude_below = 20000)
# 170000 170000 150000 80000 53338 40008 70000 56055 90000 36800 92000 NA 115920 NA

# To see which salaries were detected as hourly or weekly, set include_periodicity to TRUE
extract_salary(salaries, include_periodicity = TRUE, exclude_below = 20000)

# salary periodicity
# 1 170000 Annual
# 2 170000 Annual
# 3 150000 Annual
# 4 80000 Annual
# 5 80000 Annual
# 6 80000 Annual
# 7 70000 Annual
# 8 84074 Annual
# 9 90000 Annual
# 10 48000 Daily
# 11 120000 Daily
# 12 NA Daily
# 13 172800 Hourly
# 14 NA Hourly
format_currency

Make numeric currency values human readable

Description

Make numeric currency values human readable

Usage

format_currency(amount, symbol, digits)

Arguments

amount  Price or vector of prices (character, numeric, or integer)
symbol  Symbol to prepend to amount (e.g. $) see: currency_characters()
digits  The number of decimal places. Set equal to 2 to include cents (defaults to 0 i.e. whole major currency units)

Value

A character vector

Examples

format_currency("2423562534234", "," )
# "$2,423,562,534,234"

format_currency("2423562534234.876", ",", 0)
# "$2,423,562,534,234.88"

format_currency("2423562534234.876", ",", 2)
# "$2,423,562,534,234.88"

format_currency("2423562534234", "¥", 2)
# "¥2,423,562,534,234.00"

# format_currency() is vectorized and can accept vector arguments
format_currency(c("2423562534234", "20"), c("¥", "$"), c(1, 2))
# "¥2,423,562,534.0" "$20.0"
### format_dollars

**Make numeric currency values human readable**

**Description**

Make numeric currency values human readable

**Usage**

```r
format_dollars(amount, digits)
```

**Arguments**

- `amount`: Price or vector of prices (character, numeric, or integer)
- `digits`: The number of decimal places. Set equal to 2 to include cents (defaults to 0 i.e. whole dollars)

**Value**

A character vector

**Examples**

```r
format_dollars("2423562534234")
# "$2,423,562,534,234"

format_dollars("2423562534234.876", 0)
# "$2,423,562,534,234"

format_dollars("2423562534234.876", 2)
# "$2,423,562,534.234.88"

format_dollars("2423562534234", 2)
# "$2,423,562,534.00"
```

### retrieve_inflation_data

**Retrieve historical inflation data**

**Description**

Retrieve inflation data for any country/region (using iso2Code or country_name)

**Usage**

```r
retrieve_inflation_data(country, countries_dataframe)
```
Argument round_down_to_nearest

Round prices down to the nearest specified increment

Description

Round prices down to the nearest specified increment

Usage

round_down_to_nearest(amount, to_nearest)

Arguments

amount Price to be rounded
to_nearest Increment to which price is to be rounded down to
**round_to_nearest**

**Examples**

```r
# Round down to nearest 0.05 (5c)
library(dplyr)
prices <- c(4.45, 5.22, 0.16, 27.88, 112.19)
prices %>% round_down_to_nearest(0.05)

# Round down to nearest $10
prices <- c(4.45, 5.22, 0.16, 27.88, 112.19)
prices %>% round_down_to_nearest(10)
```

**Description**

Round prices to the nearest specified increment

**Usage**

```r
round_to_nearest(amount, to_nearest)
```

**Arguments**

- `amount` Price to be rounded
- `to_nearest` Increment to which price is to be rounded

**Examples**

```r
# Round to nearest 0.05 (5c)
library(dplyr)
prices <- c(4.45, 5.22, 0.16, 27.88, 112.19)
prices %>% round_to_nearest(0.05)

# Round to nearest $10
prices <- c(4.45, 5.22, 0.16, 27.88, 112.19)
prices %>% round_to_nearest(10)
```
### `round_up_to_nearest`  
*Round prices up to the nearest specified increment*

**Description**

Round prices up to the nearest specified increment

**Usage**

```r
round_up_to_nearest(amount, to_nearest)
```

**Arguments**

- `amount`  
  Price to be rounded
- `to_nearest`  
  Increment to which price is to be rounded up to

**Examples**

```r
# Round up to nearest 0.05 (5c)
library(dplyr)
prices <- c(4.45, 5.22, 0.16, 27.88, 112.19)
prices %>% round_up_to_nearest(0.05)
```

```r
# Round up to nearest $10
prices <- c(4.45, 5.22, 0.16, 27.88, 112.19)
prices %>% round_up_to_nearest(10)
```

### `show_countries`  
*Show available country codes*

**Description**

`show_countries` calls the World Bank API and retrieves a list of available countries and regions

**Usage**

```r
show_countries()
```

**Value**

A data.frame of countries available to query using the World Bank API
Examples

# Simply call show_countries() to receive a dataframe of all countries (and regions) and their
# iso2Code

# show_countries()
# iso2Code country_name
# 1 AW Aruba
# 2 AF Afghanistan
# 3 A9 Africa
# 4 AO Angola
# Etc

url_all_results  Generate a World Bank API URL that will return all results for a given
indicator in JSON format

Description
results and returns JSON format

Usage

url_all_results(original_url)

Arguments

original_url A World Bank API URL. E.g. "https://api.worldbank.org/v2/country".

Value

A character vector

Examples

# Provide a World Bank API URL and `url_all_results` will convert it into one with all results
# for that indicator
original_url <- "https://api.worldbank.org/v2/country" # Note: no ?format=json on url
url_all_results(original_url)
# "https://api.worldbank.org/v2/country?format=json&per_page=304"
Index

adjust_for_inflation, 2
afi (adjust_for_inflation), 2

convert_to_iso2Code, 4
country_input_type, 5
currency_characters, 6
currency_to_numeric, 7

extract_salary, 7

format_currency, 10
format_dollars, 11

priceR-package, 2

retrieve_inflation_data, 11
round_down_to_nearest, 12
round_to_nearest, 13
round_up_to_nearest, 14

show_countries, 14

url_all_results, 15