

# Package ‘priceR’

May 25, 2021

**Type** Package

**Title** Economics and Pricing Tools

**Version** 0.1.6

**Imports** dplyr, gsubfn, stringr, purrr, jsonlite, stats, lubridate,  
curl, stringi

**Author** Steve Condylis [aut, cre] (<<https://orcid.org/0000-0003-0599-844X>>)

**Maintainer** Steve Condylis <[steve.condylis@gmail.com](mailto:steve.condylis@gmail.com)>

**BugReports** <https://github.com/stevecondylis/priceR/issues>

**License** MIT + file LICENSE

**URL** <https://github.com/stevecondylis/priceR>

**Description** Functions to aid in micro and macro economic analysis and handling of price and currency data. Includes extraction of relevant inflation and exchange rate data from World Bank API, data cleaning/parsing, and standardisation. Inflation adjustment calculations as found in Principles of Macroeconomics by Gregory Mankiw et al (2014). Current and historical end of day exchange rates for 171 currencies from the European Central Bank Statistical Data Warehouse (2020) <<https://sdw.ecb.europa.eu/curConverter.do>>.

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**Suggests** testthat

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2021-05-25 06:40:05 UTC

## R topics documented:

adjust_for_inflation . . . . .	2
convert_to_iso2Code . . . . .	4
country_input_type . . . . .	5
currencies . . . . .	6
currency_characters . . . . .	6

currency_to_numeric . . . . .	7
display_api_info . . . . .	7
exchange_rate_latest . . . . .	8
extract_salary . . . . .	9
format_currency . . . . .	11
format_dollars . . . . .	12
historical_exchange_rates . . . . .	13
make_dates . . . . .	14
retrieve_historical_rates . . . . .	15
retrieve_inflation_data . . . . .	16
round_down_to_nearest . . . . .	17
round_to_nearest . . . . .	17
round_up_to_nearest . . . . .	18
show_countries . . . . .	19
url_all_results . . . . .	19

## Index 21

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).

**Value**

A vector of inflation-adjusted prices

**Examples**

```
## Not run:
# Assign these variables once
country <- "AU"
inflation_dataframe <- retrieve_inflation_data(country)
countries_dataframe <- show_countries()

# Convert $100 from 2005 into 2017 dollars

adjust_for_inflation(100, 2005, country, to_date = 2017,
inflation_dataframe = inflation_dataframe,
countries_dataframe = countries_dataframe)

# [1] 133.9861 # i.e. $100 in 2005 had the same purchasing power as $133.99 in 2017
```

```
## End(Not run)
```

---

```
convert_to_iso2Code Convert any country input into its iso2Code
```

---

## Description

‘convert\_to\_iso2Code’ accepts the type of country input and the country, and returns the relevant iso2Code

## Usage

```
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

## 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	<i>Determines whether country input is a country name or iso2Code</i>
--------------------	-----------------------------------------------------------------------

---

### 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

```
## 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)
```

---

currencies                      *Retrieve available currencies and their respective symbols/codes*

---

**Description**

Retrieve available currencies and their respective symbols/codes

**Usage**

```
currencies()
```

**Value**

A data.frame of available currencies and their respective symbols/codes

**Examples**

```
## Not run:
# Display available currencies and their respective symbols/codes
currencies()
#           description code
# 1   United Arab Emirates Dirham AED
# 2     Afghan Afghani           AFN
# 3   Albanian Lek             ALL
# 4   Armenian Dram           AMD
# 5 Netherlands Antillean Guilder ANG
# 6   Angolan Kwanza          AOA
# 7   Argentine Peso          ARS

## 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
```

---

```
display_api_info      Display link to further information
```

---

**Description**

Display link to further information

**Usage**

```
display_api_info()
```

**Examples**

```
## Not run:
# Display a message indicating where further documentation can be found

## End(Not run)
```

---

exchange\_rate\_latest *Retrieve the latest exchange rates between the provided currency code*

---

### Description

Retrieve the latest exchange rates between the provided currency code

### Usage

```
exchange_rate_latest(currency)
```

### Arguments

currency            A currency code (see currencies() for supported codes)

### Value

A data.frame containing the latest exchange rates between the provided currency code and each other available currency

### Examples

```
## Not run:

exchange_rate_latest("AUD")
# Daily AUD exchange rate as at end of day 2020-07-27 GMT
#   currency one_aud_is_equivalent_to
# 1      AED                2.61894
# 2      AFN                54.47724
# 3      ALL                75.51799
# 4      AMD               343.40193
# 5      ANG                 1.26829
# 6      AOA               400.54604

# Defaults to USD
exchange_rate_latest()
# Daily USD exchange rate as at end of day 2020-07-27 GMT
#   currency one_usd_is_equivalent_to
# 1      AED                 3.6730
# 2      AFN                76.4035
# 3      ALL               105.9129
# 4      AMD               481.6162
# 5      ANG                 1.7788
# 6      AOA               561.7599

## End(Not run)
```



---

extract_salary	<i>Extract numeric salary from text data</i>
----------------	----------------------------------------------

---

### 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 identified as Daily. Default is 8 hours.

**days\_per\_workweek**  
Set assumed number of days per workweek. Only affects annualisation of rates identified as Daily. Default is 5 days.

**working\_weeks\_per\_year**  
Set assumed number of working weeks in the year. Only affects annualisation of rates identified 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",
                    "$160000.00 - $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: <code>currency_characters()</code>
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))
# "¥2,423,562,534,234.0" "$20.0"
```

---

format\_dollars

*Make numeric currency values human readable*


---

**Description**

Make numeric currency values human readable

**Usage**

```
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**

```
# 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)
# "$2,423,562,534,234.00"
```

---

historical\_exchange\_rates

*Retrieve historical exchange rates*

---

## Description

Retrieves historical exchange rates between a currency pair

## Usage

```
historical_exchange_rates(from, to, start_date, end_date)
```

## Arguments

from	A currency code (see <code>currencies()</code> for supported codes)
to	A currency code
start_date	A start date (of the form "2010-01-01")
end_date	An end date

## Value

A data.frame containing exchange rate data for select currency pair

## Examples

```
## Not run:
# Note date range >365 days', and returns 912 rows (as expected)
historical_exchange_rates("USD", to = "AUD", start_date = "2018-01-01", end_date = "2020-06-30")

historical_exchange_rates("USD", to = "AUD",
                           start_date = "2020-01-01", end_date = "2020-06-30")

historical_exchange_rates("AUD", to = "USD",
                           start_date = "2010-01-01", end_date = "2020-06-30")

historical_exchange_rates("AUD", to = "USD",
                           start_date = "2010-01-01", end_date = "2020-06-30")
```

```
## End(Not run)
```

---

make_dates	<i>Creates date ranges so as to batch up large API calls into many smaller ones</i>
------------	-------------------------------------------------------------------------------------

---

### Description

Creates date ranges so as to batch up large API calls into many smaller ones

### Usage

```
make_dates(start_date, end_date, n_days)
```

### Arguments

start_date	A start date (of the form "2010-01-01")
end_date	An end date
n_days	The maximum number of days in each period

### Value

A data.frame containing start and end dates for periods of length no longer than n\_days

### Examples

```
# Simple test
start_date = "2010-01-01"
end_date = "2020-06-30"
n_days = 365
priceR::make_dates(start_date, end_date, n_days)

# With lots of periods
start_date = "2010-01-01"
end_date = "2020-06-30"
n_days = 20
priceR::make_dates(start_date, end_date, n_days)

# Less than one period
start_date = "2020-01-01"
end_date = "2020-06-30"
n_days = 365
priceR::make_dates(start_date, end_date, n_days)

# 366 days (note 2020 was a leap year)
start_date = "2019-07-30"
```



```
## End(Not run)
```

---

```
retrieve_inflation_data
      Retrieve historical inflation data
```

---

### Description

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

### Usage

```
retrieve_inflation_data(country, countries_dataframe)
```

### Arguments

country            A country\_name or iso2code (see show\_countries() for complete list of available inputs).

countries\_dataframe    The output from show\_countries(). It is optional, but if not provided, it will be retrieved via the API.

### Value

A data.frame containing inflation data from World Bank API for specified country

### Examples

```
## Not run:
# Retrieve inflation data for any country (or iso2Code)
country <- "AU"
inflation_dataframe <- retrieve_inflation_data(country)

country <- "Australia"
countries_dataframe <- show_countries()
inflation_dataframe <- retrieve_inflation_data(country, countries_dataframe)

## End(Not run)
# inflation_dataframe
#   indicator.id           indicator.value country.id country.value   value
# FP.CPI.TOTL.ZG Inflation, consumer prices (annual %)      AU   Australia   <NA>
# FP.CPI.TOTL.ZG Inflation, consumer prices (annual %)      AU   Australia  1.94864
# FP.CPI.TOTL.ZG Inflation, consumer prices (annual %)      AU   Australia  1.27699
# FP.CPI.TOTL.ZG Inflation, consumer prices (annual %)      AU   Australia  1.50836
# Etc
```



---

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

**Examples**

```
# 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)
```

---

round\_to\_nearest *Round prices to the nearest specified increment*

---

**Description**

Round prices to the nearest specified increment

**Usage**

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

**Arguments**

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

## Examples

```
# 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

```
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

```
# 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)

# 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	<i>Show available country codes</i>
----------------	-------------------------------------

---

**Description**

'show\_countries' calls the World Bank API and retrieves a list of available countries and regions

**Usage**

```
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	<i>Generate a World Bank API URL that will return all results for a given indicator in JSON format</i>
-----------------	--------------------------------------------------------------------------------------------------------

---

**Description**

results and returns JSON format

**Usage**

```
url_all_results(original_url)
```

**Arguments**

original\_url A World Bank API URL. E.g. "http://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 <- "http://api.worldbank.org/v2/country" # Note: no ?format=json on url
url_all_results(original_url)
# "http://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  
[currencies](#), 6  
[currency\\_characters](#), 6  
[currency\\_to\\_numeric](#), 7

[display\\_api\\_info](#), 7

[exchange\\_rate\\_latest](#), 8  
[extract\\_salary](#), 9

[format\\_currency](#), 11  
[format\\_dollars](#), 12

[historical\\_exchange\\_rates](#), 13

[make\\_dates](#), 14

[retrieve\\_historical\\_rates](#), 15  
[retrieve\\_inflation\\_data](#), 16  
[round\\_down\\_to\\_nearest](#), 17  
[round\\_to\\_nearest](#), 17  
[round\\_up\\_to\\_nearest](#), 18

[show\\_countries](#), 19

[url\\_all\\_results](#), 19