Package: MaddisonData (via r-universe)

May 20, 2026

Title Maddison Project Data

Version 1.1.0

Date 2026-01-09

Description Relatively easy access is provided to 2023 version of the Maddison project data downloaded 2025-08-28. This project collates all the credible data on population and GDP for 169 countries, with some dating back to the year 1 of the current era. One function makes it easy to find the leaders for each year, allowing users to delete countries like OPEC with narrow economies to focus on technology leaders. Another function makes it easy to plot data for only selected countries or years. Another function makes it relatively easy to obtain references to the original sources, which must be cited per the copyright rules of the Maddison Project for different uses of their data.

License MIT + file LICENSE

URL https://github.com/sbgraves237/MaddisonData

BugReports https://github.com/sbgraves237/MaddisonData/issues

Depends R (>= 4.1)

Language en-US

Suggests egg, ggplot2, ipumsr, KFAS, knitr, lubridate, readxl, RefManageR, rmarkdown, testthat (>= 3.0.0), tibble, usethis

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Repository https://sbgraves237.r-universe.dev

Date/Publication 2026-03-21 02:36:18 UTC

RemoteUrl https://github.com/sbgraves237/maddisondata

RemoteRef HEAD

RemoteSha 2a19cadf2efdb2e815bd1b206c2645099a280d71

Contents

`getMaddisonSources` *Get Maddison sources*

Description

The Maddison project collates historical economic statistics from many sources.

They have a citation policy: CONDITIONS UNDER WHICH ALL ORIGINAL PAPERS MUST BE CITED:

a) If the data is shown in any graphical form b) If subsets of the full dataset that include less than a dozen (12) countries are used for statistical analysis or any other purposes

When neither a) or b) apply, then the MDP as a whole can be cited.

`getMaddisonSources` returns a `data.frame` of relevant sources for a particular application.

Usage

```
getMaddisonSources(
  ISO = NULL,
  plot = TRUE,
  sources = MaddisonData::MaddisonSources,
  years = MaddisonData::MaddisonYears
)
```

Arguments

ISO	either NULL to return all sources or a character vector of ISO codes for the countries included in the analysis or a data.frame with the first column being the ISO codes followed by yearBegin and optionally yearEnd.
plot	logical indicating whether the use does nor does not include plotting data. The Maddison project requires citing all relevant MaddisonSources if they are plotted, denoted here by plot = TRUE. If no data are plotted, then the Maddison project requires citing all sources only if less than a dozen are used, denoted here by plot = FALSE, in which case, the Maddison project requires a specific project-level citation. Default = TRUE.
sources	list of sources in the format of MaddisonSources; default is MaddisonSources.
years	data.frame in the format of MaddisonYears; default is MaddisonYears.

Value

a data.frame with 3 columns:

ISO 3-letter ISO code for country.

years character vector of years or year ranges for which source applies.

source character vector of sources.

in the format of MaddisonSources.

Examples

```
getMaddisonSources() # all
getMaddisonSources(plot=FALSE) # only MDP
GBR <- getMaddisonSources('GBR') # GBR

getMaddisonSources(names(MaddisonSources)[1:12], FALSE) # only MDP
getMaddisonSources(data.frame(ISO=c('GBR', 'USA'),
             yearBegin=rep(1500, 2)) ) #GBR, USA since 1500
getMaddisonSources('AUS') # AUS: no special sources for AUS.
```

---

ggplotPath ggplot *paths*

Description

ggplotPath plots y vs. x (typically year) with a separate line for each group with options for legend placement, horizontal and vertical lines and labels.

Usage

```
ggplotPath(
  x = "year",
  y,
  group,
  data,
  scaley = 1,
  logy = TRUE,
  ylab,
  legend.position,
  hlines,
  vlines,
  labels,
  fontsize = 10,
  color,
  linetype
)
```

Arguments

x	name of a numeric column in data to pass as x in aes(x=.data[[x]], ...); default = year.
y	name of column in data to pass as y in aes(y=.data[[y]], ...); must be supplied.
group	name of grouping variable, i.e., plot a separate line for each level of group using aes(group=.data[[group]], ...), unless group is missing or length(unique(data[, group])) = 1.
data	data.frame or tibble::tibble with columns x, y, and group.
scaley	number to divide y by for plotting. Default = 1, but for data in monetary terms, e.g., for MaddisonData, y = 'gdppc' is Gross domestic product (GDP) per capita in 2011 dollars at purchasing power parity (PPP), for which we typically want scaley = 1000.
logy	logical: if TRUE, y axis is on a log scale; default = TRUE.‘
ylab	y axis label. Default = if(scaley==1) y else paste(y, '/', scaley)
legend.position	argument passed to ggplot2::theme. If !missing(labels), default is no legend. Otherwise, default depends on nGps <- length(unique(data[, group]): If nGps = 1, there is no legend. If nGps < 10, legend.position = c(.15, .5). Else if nGps < 2-. legend.position = 'right'. Else legend.position = 'none'. To suppress the legend, use legend.position = 'none'. For other alternatives, see ggplot2::theme.
hlines	numeric vector of locations on the y axis for horizontal lines using ggplot2::geom_hline(yintercept = hlines, ...) with color='grey', lty='dotted' unless color or colour and / or lty are available as attr(x, ...).
vlines	numeric vector of locations on the x axis for vertical lines using ggplot2::geom_vline(xintercept = vlines, ...) with color='grey', lty='dotted' unless color or colour and / or lty are available as attr(x, ...).

labels = data.frame with columns x, y, label, and optionally srt, col, size, where x, y, srt, and size are are numeric, label is character, and col are acceptable values for color in with(labels, annotate('text', x=x, y=y, label = label, srt=srt, color=col, size=size)). Defaults for srt, col, and size are 0, 'black', and 4, respectively.

fontsize for legend and axes labels in theme(text=element_text(size=fontsize)); default = 10.

color for lines to pass to scale_color_manual(values = color) if present. If present, length(color) should equal length(unique(data[, group])).

linetype optional vector. Default if(missing(group)) 1 else rep(1:6, length=length(unique(data[, group
Else either

- an integer (0-6), a name (0 = blank, 1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash), or
- a mapping to a discrete variable, or
- a string of an even number (up to eight) of hexadecimal digits which give the lengths in consecutive positions in the string.

Value

an object of class ggplot2::ggplot, which can be subsequently edited, and whose print method produces the desired plot.

Examples

```
str(GBR_USA <- subset(MaddisonData::MaddisonData, ISO %in% c('GBR', 'USA')))
GBR_USA1 <- ggplotPath('year', 'gdppc', 'ISO', GBR_USA, 1000)
GBR_USA1a <- ggplotPath('year', 'gdppc', 'ISO', GBR_USA, 1000,
                        color=c('red', 'blue'))

GBR_USA1+ggplot2::coord_cartesian(xlim=c(1500, 1850)) # for only 1500-1850
GBR_USA1+ggplot2::coord_cartesian(xlim=c(1600, 1700), ylim=c(.9, 3))

# label the lines
ISOll <- data.frame(x=c(1500, 1800), y=c(2.5, 1.7), label=c('GBR', 'USA'),
              srt=c(0, 30), col=c('red', 'green'), size=c(2, 9))
GBR_USA2 <- ggplotPath('year', 'gdppc', 'ISO', GBR_USA, 1000,
                    labels=ISOll, fontsize = 20)

# h, vlines, manual legend only
Hlines <- c(1,3, 10, 30)
Vlines = c(1849, 1929, 1933, 1939, 1945)
(GBR_USA3 <- ggplotPath('year', 'gdppc', 'ISO', GBR_USA, 1000,
       ylab='GDP per capita (2011 PPP K$)',
       legend.position = NULL, hlines=Hlines, vlines=Vlines, labels=ISOll))

# do.call(ggplotPath, ...) with 1 line
list1 <- list(x='Time', y='lvl', data=data.frame(Time=1:4, lvl=sqrt(1:4)))
doCallPlot <- do.call(ggplotPath, list1)
```

ggplotPath2 ggplotPath *in multiple panels with no space between panels and a shared horizontal axis on the bottom.*

Description

ggplotPath2 accepts a variety of inputs. The most general is with object = a list of sublists to be fed individually to do.call(ggplotPath,..) and then assembled into the desired plot after eliminating the space between the individual plots. Optional \dots arguments give default values for the individual calls to ggplotPath.

Usage

```
ggplotPath2(object, ...)

## S3 method for class 'list'
ggplotPath2(object, ...)

## S3 method for class 'mts'
ggplotPath2(object, Time, ...)

## S3 method for class 'KFS'
ggplotPath2(object, ...)

## Default S3 method:
ggplotPath2(
  object,
  Time,
  object2,
  scaley,
  logy,
  ylab,
  hlines,
  vlines = numeric(0),
  labels,
  fontsize = 10,
  color,
  linetype,
  ...
)
```

Arguments

object with an associated matrix to be plotted. The following classes of objects are supported:

- matrix, data.frame, tibble or mts of dim = c(n, k), e.g., k components of a state vector at n points in time. ggplotPath is called separately for

each of the k column. Each such call returns a ggplot object. Those ggplot objects are assembled as separate panels into a ggplot object with a shared horizontal axis with no space between the panels with at most one more row than length(Time) and the first row of object is dropped if needed so n == length(k).

- KFS (defined in package KFAS) calls ggplotPath.matrix(object$a, ...).
- list with a component a, assumed to a model produced from estimation in the KFAS package, calls ggplotPath2(object$a, ...).
- list with component model: If it has a component a, call ggplotPath2(object$model$a, ...). Else call ggplotPath2(KFAS::KFS(object), ...).

`...` optional arguments.

`Time` optional numeric vector with lenght being either the number of rows of the matrix obtained from object or one less and fed to ggplotPath to create the individual panels of the plot. Default depends on class(object):

- matrix: Default Time = rownames(object).
- mts: Default Time = time(object).
- KFS (defined in package KFAS): Default = rownames or time of object$model$y.
- model (defined in package KFAS): Default = rownames or time of object$y.

`object2` an optional vector or matrix-type object with the same number of rows as object or one less and k2 columns that provide separate reference lines to appear in the same panels as the first k2 columns of object. Obviously, k2 <= k. (Also, legend.position = 'none'.)

`scaley` = optional numeric vector of the length k fed individually as the scaley argument accompanying the different successive calls to ggplotPath. If the matrix obtained from object has two columns with names = c('level', 'slope') or c('level', 'growthRate'), then the default for scaley = c(1000, .0). Otherwise, default = 1.

`logy` optional character vector of length k to control the scales used in the individual panels of the plot:

- exp_log: Feed exp of column i of the matrix to plot to ggplotPath(..., logy=TRUE).
- log: Feed column i of the matrix to plot to ggplotPath(..., logy=TRUE).
- ": Feed column i of the matrix to plot to ggplotPath(..., logy=FALSE).

The default is 'exp_log' for the first panel and " for the rest.

`ylab` optional character vector of length k for y axis labels.

`hlines` optional list of at most k of numeric vectors (possibly with attributes), for horizontal lines in panel i, feeding hlines[[i]] to ggplotPath(..., hlines = hlines[[i]]) for panel i = 1:k. color='grey', lty='dotted' unless color or colour and / or lty are available as attr(hlines[[i]], ...). hlines outside the plot range are suppressed with a warning.

`vlines` optional numeric vector of locations on the x axis for vertical lines using ggplotPath(..., vlines=vlines) with color='grey', lty='dotted' unless color or colour and / or lty are available as attr(vlines, ...). vlines outside the plot range are suppressed with a warning.

labels	= optional data.frame with columns x, y, label, component, and optionally srt, col, size, where x, y, srt, and size are are numeric, label is character, and col are acceptable values for color in with(labels, annotate('text', x=x, y=y, label = label, srt=srt, color=col, size=size)). Defaults for srt, col, and size are 0, 'black', and 4, respectively. component is an integer in 1:k. labelsi = subset(labels, component==i) is used in ggplotPath(..., labels=labelsi) to label panel i = 1:k. labels outside the plot range are suppressed with a warning. labels near the plot boundaries may be clipped.
fontsize	optional number for legend and axes labels, used in 'ggplotPath(..., fontsize=fontsize); default = 10.
color	optional vector or list of length 'k of vectors to feed to ggplotPath(..., color=color[[i]]) for panel i=1:k. If present, length(color) should equal k. Default is c('black', 'red') for each panel with a reference line provided in object2 and black otherwise. See ggplotPath for options.
linetype	optional vector or list of length k of vectors to feed to ggplotPath(..., linetype=linetype[[i]]) for panel i=1:k. If present, length(linetype) should equal k. Default for each panel with a reference line in object2 = c(1, 3) = c('solid', 'dotted') else 1 = solid. See ggplotPath for options.

Details

Alternatively, the first object argument can be a matrix, data.frame, tibble, or multivariate time series (of class mts). An optional Time argument must be a numeric vector with length equal to the number of rows of object. If Time is provided, it overrides any values for the horizontal axis that could be inferred from rownames or time of object. A second optional object2 can be either a vector whose length matches the number of rows of object or a matrix or ts object similarly matching the number of rows of object.

Value

an object of class ggplot2::ggplot, which can be subsequently edited, and whose print method produces the desired plot.

Examples

```
# matrix examples
Mat <- cbind(lvl=1:5, vel=rep(1:2, length=5), acc=sin(1:5))
Mat1 <- Mat
rownames(Mat1) <- 1951:1955
Mat2 <- as.data.frame(cbind(Mat, year=1951:1955))

# mts example
MTS <- ts(Mat, 1951)

# Do
Matp <- ggplotPath2(Mat)
# with object2 = vector
  Matp1 <- ggplotPath2(Mat, object2=sqrt(1:5))
# with object2= = 2 column matrix for first 2 panels.
```

```
Matp2 <- ggplotPath2(Mat, object2=2*Mat[, 2:1])
Mat1p <- ggplotPath2(Mat1)
Mat2p <- ggplotPath2(Mat2[, 1:3], Time=Mat2[, 'year'])
MTSp <- ggplotPath2(MTS)
MTSep <- ggplotPath2(MTS, logy=c('exp_log', 'log', ''))

# list example
List2 <- list(
    level=list('year', 'lvl'),
    slope=list('year', 'vel'),
    accel=list('year', 'acc'))
Mat2l <- ggplotPath2(List2, data=Mat2)

# State space / Kalman filtering model for GBR
GBR <- subset(MaddisonData, (ISO=='GBR') & !is.na(gdppc))
# model example
growthFormula <- (log(gdppc)~ -1 + SSMbespoke(growthModel(.04, GBR$gdppc) ))
library(KFAS)
GBR2m <-SSModel(growthFormula, GBR, H=matrix(NA) )

# NOTE: This call ignores Time
GBRgrowthFit1 <- fitSSM(GBR2m, inits=-6, method = "BFGS",
                        updatefn = growthUpdateFn)
# NOTE: This call currently also ignores Time; MUST BE FIXED
GBRgrowthFit1t <- fitSSM(GBR2m, inits=-6, method = "BFGS",
                         updatefn = growthUpdateFn, Time=GBR$year)
GBRfitp <- ggplotPath2(GBRgrowthFit1t)

#KFS example
GBR_KFS <- KFAS::KFS(GBRgrowthFit1$model)
GBR_KFSt <- KFAS::KFS(GBRgrowthFit1t$model)
GBR_KFSp0 <- ggplotPath2(GBR_KFS)
GBR_KFSp <- ggplotPath2(GBR_KFS$a)
GBR_KFStp <- ggplotPath2(GBR_KFSt$a)

# label the lines
ISOll1 <- data.frame(x=c(1500, 1800), y=c(2.5, 1.7),
                 label=c('GBR', 'Napoleon'), srt=c(0, 30),
                 col=c('red', 'green'), size=c(2, 9),
                 component=1)

GBR_KFSp1 <- ggplotPath2(GBR_KFS$a, labels=ISOll1)

GBR_KFSp <- ggplotPath2(GBR_KFS, labels=ISOll1)
ISOll2 <- ISOll1
ISOll2$component <- 1:2
GBR_KFSp2 <- ggplotPath2(GBR_KFS, labels=ISOll2)

# hlines, vlines
zero <- 0
attr(zero, 'color') <- 'red'
attr(zero, 'lty') <- 'dashed'
Hlines1 <- list(c(1,3, 10, 30), zero)
```

```
Vlines <- c(1649, 1929, 1933, 1945)

GBR_KFSp3 <- ggplotPath2(GBR_KFS, labels=ISOll2, hlines=Hlines1,
                         vlines=Vlines)
```

`growthModel` *Two-dimensional growth model for* `KFAS`

Description

`growthModel` returns a list returned by `KFAS::SSMcustom()` for a model with potentially irregularly spaced univariate observations with a 2-dimensional (`level`, `growthRate`) state.

Usage

```
growthModel(
  sigma,
  y,
  a1,
  Time,
  stateNames = c("level", "growthRate"),
  Log = TRUE,
  ...
)
```

Arguments

`sigma`	a numeric vector, forced to length 2 by replacing it by `rep(sigma, length=2)`. The one-step transition variance for `level` is `Q[1,1] = sigma[1]^2` and for `growthRate` is `Q[2,2] = sigma[2]^2`. With `m` missing values, `Q[1,1] = (m+1)*sigma[1]^2`, `Q[2,2] = (m+1)*sigma[2]^2`, and `Q[1,2] = Q[2,1] = sqrt((m+1)*choose(m+1,2)*sigma[1]*sigma`
`y`	= optional numeric vector or `ts` object of length at least 2 used to estimate the `a1` parameter for `KFAS::SSMcustom()` if `a1` is not supplied and for `Time` if it is not supplied. `a1 <- (if(Log) log(c(y[1], y[2]/y[1])) else c(y[1], y[2]-y[1]))`.
`a1`	= optional numeric vector of length 2 to pass to `KFAS::SSMcustom`. If supplied, the numeric values of `y` are ignored.
`Time`	= optional integer vector of times at which non-missing observations are available. Default = `time(y)` if `y` is of class `ts` or or `names(y)` if `!is.null` or `1:length(y)` otherwise.
`stateNames`	= `c('level', 'growthRate')`
`Log`	default = TRUE.
`...`	optional arguments passed to `KFAS::SSMcustom()`.

Value

a list returned by KFAS::SSMcustom() with an additional Time component.

Examples

```
GBR <- subset(MaddisonData, (ISO=='GBR') & !is.na(gdppc))
growthMdl1 <- growthModel(.1, GBR$gdppc, Time=GBR$year)
GBRgdppc1 <- with(GBR[-1, ], ts(gdppc, year[1]))
growthMdl2 <- growthModel(c(.1, .2), GBRgdppc1)
growthMdl0 <- growthModel(.1, GBR$gdppc, a1=c(10, 1), Log=FALSE,
                  stateNames=c('lvl', 'vel'))
```

---

growthUpdateFn *Update function to estimating a two-dimensional growth model*

Description

growthUpdateFn

Usage

```
growthUpdateFn(pars, model, Time)
```

Arguments

pars	= log(variance) of noise for [1] observations, [2] level, and [3] growthRate. Forced to length 3 via c(pars, rep(tail(pars, 1), length=3-length(pars))). With no missing values, the one-step transition variance for level is Q[1,1] = exp(pars[2] and for growthRate is Q[2,2,] = exp(pars[3]). With m missing values, Q[1,1,] = (m+1)*exp(pars[2]), Q[2,2] = (m+1)*exp(pars[3]), and Q[1,2,] = Q[2,1] = sqrt((m+1)*ch
model	= list assumed to have components n = third dimension of the transition covariance array Q and state_names = c('level', 'growthRate') as returned by growthModel.
Time	= optional integer vector of times at which non-missing observations are available. Default is model$Time if available or time(y) if model has a component y or names(y) if !is.null or 1:n otherwise.

Details

returns a list returned by KFAS::SSMcustom() for a model with potentially irregularly spaced univariate observations with a 2-dimensional (level, growthRate) state.

Value

a model with components H and Q updated as described.

Examples

```
GBR <- subset(MaddisonData, (ISO=='GBR') & !is.na(gdppc))
growthMdl1 <- growthModel(.1, GBR$gdppc, Time=GBR$year)
growthMdl1v0 <- growthUpdateFn(.1, growthMdl1)
growthMdl1v <- growthUpdateFn(.1, growthMdl1, Time=GBR$year)
growthMdl1v3 <- growthUpdateFn(1:3, growthMdl1, Time=GBR$year)

growthFml <- (log(gdppc)~ -1 + SSMbespoke(growthModel(.1, GBR$gdppc) ))
library(KFAS)
growthSSmdl <-SSModel(growthFml, GBR, H=matrix(NA) )
growthSSmdl1 <- growthUpdateFn(.1, growthSSmdl)
```

---

logMaddison *Select countries and add logged variables*

Description

logMaddison returns a tibble::tibble of data on selected countries extracted from MaddisonData, appending columns lnGDPpc and lnPop = natural logarithms of gdppc and pop.

Usage

```
logMaddison(ISO = NULL)
```

Arguments

ISO	either NULL to select all the data in MaddisonData or a character vector of ISO codes used in the Maddison project.

Value

a tibble::tibble with 6 columns:

ISO 3-letter ISO code for countries selected

year numeric year in the current era.

gdppc Gross domestic product per capita adjusted for inflation to 2011 dollars at purchasing power parity.

pop Population, mid-year (thousands)

lnGDPpc log(gdppc)

lnPop log(pop)

Examples

```
logMaddison() # all
logMaddison(c('GBR', 'USA')) # GBR, USA
```

MadDateRanges *Convert a vector of date ranges into a data.frame*

Description

MadDateRanges returns a data.frame with 3 numeric columns: yearBegin, yearEnd, and sourceNum from the vector of dateRanges associated with different sources in MaddisonSources.

Usage

```
MadDateRanges(dateRanges)
```

Arguments

dateRanges	character vector of date ranges, each associated with a different source.

Value

a data.frame with 3 columns

yearBegin, yearEnd numeric years

sourceNum 1, 2, 3, ... for the location in dateRanges

Examples

```
MadDateRanges(c('1', '700 - 1500', '1252-1700 (England)',
      '1915-1919 & 1949', '1820, 1870, 1913, 1950'))
# equal
data.frame(
yearBegin=c(1,  700, 1252, 1820, 1870, 1913, 1950),
yearEnd  =c(1, 1500, 1700, 1820, 1870, 1913, 1950),
sourceNum=c(1, 2, 3, rep(4, 4)))
```

MaddisonCountries *Maddison Project data*

Description

The Maddison project collates historical economic statistics from many sources. MaddisonCountries is a data.frame of all (countrycode, country, region) combinations in those data.

Usage

```
MaddisonCountries
```

Format

`MaddisonCountries`:
A data frame with 3 columns:

ISO 3-letter ISO country code

country Country name used by the Maddison project

region Geographic region including `country`

Its `rownames = ISO`.

Source

https://www.rug.nl/ggdc/historicaldevelopment/maddison/releases/maddison-project-database-2020?lang=en"Groningen Growth and Development Centre"

Examples

```
# Get the country for a countrycode (IS)
subset(MaddisonCountries, ISO=='GBR', country)
# Or
MaddisonCountries['GBR', 'country']
# Find Yugoslavia
subset(MaddisonCountries, grepl('Yugo', country), 1:3)
# number of countries by region
table(MaddisonCountries$region)
# What are "Western Offshoots"?
subset(MaddisonCountries, grepl('Of', region), c(country, ISO))
```

`MaddisonData` *Maddison Project data*

Description

The Maddison project collates historical economic statistics from many sources. `MaddisonCountries` is a `data.frame` of all (`countrycode`, `country`, `region`) combinations in those data. This object provides easy access to the 2023 version of the Maddison project data downloaded 2025-08-28.

Usage

```
MaddisonData
```

Format

`MaddisonData`:
A data frame with 4 columns:

ISO 3-letter ISO country code

year numeric year starting with year 1 CE

gdppc Gross domestic product (GDP) per capita in 2011 dollars at purchasing power parity (PPP)

pop Population, mid-year (thousands)

Source

https://www.rug.nl/ggdc/historicaldevelopment/maddison/releases/maddison-project-database-2020?lang=en"Groningen Growth and Development Centre"

Examples

```
# Get the countrycode for a country
subset(MaddisonCountries, country=='United Kingdom', ISO)
# Select
str(GBR <- MaddisonData[MaddisonData$ISO=='GBR', ])
```

`MaddisonLeaders` *Identify leading countries*

Description

`MaddisonLeaders` computes the countries with the highest gdppc for each `year`.

Usage

```
MaddisonLeaders(
  except = character(0),
  y = "gdppc",
  group = "ISO",
  data = MaddisonData::MaddisonData,
  x = "year"
)
```

Arguments

except	either NULL to select all the data in `MaddisonData` or a character vector of `group` codes to EXCLUDE, e.g., so the result reflects apparent technology leaders, excluding countries whose high gdppc may be due to a dominant position in a single commodity.
y	name of column in `data` to consider. Default = gdppc.
group	name of column in `data` as the grouping variable. Default = `ISO`.
data	`data.frame` or `tibble::tibble` with first two columns being `ISO` and `year` and y being the name of another column.
x	time variable. Default = `year`.

Value

an object of class `c('MaddisonLeaders', 'data.frame')`, with columns

- `paste0(x, 'Begin)`,
- `paste0(x, 'End')`,

- paste0(y, '0'),
- paste0(y, '1'), and
- {{group}}
- paste0('d', x, '0') = paste0(x, 'End') - paste0(x, 'Begin') + min(dx), where dx = min(diff(sort(unique(data[, x]))))
- paste0('d', x, '1') = c(tail(paste0(x, 'Begin'), -1) - head(paste0(x, 'End'), -1), NA) (defaults: dy0 = yearEnd - yearBegin +1 and dy1 = c(tail(yearBegin, -1) - head(yearEnd, -1), NA))

(defaults:

- yearBegin,
- yearEnd,
- gdppc0,
- gdppc1, and
- ISO, plus
- dyear0 = yearEnd - yearBegin + 1 and
- dyear1 = c(tail(yearBegin, -1) - head(yearEnd, -1), NA)

with an attribute LeaderByYear = a data.frame with columns, {{x}}, paste0('max', y), and {{group}} (defaults: year, maxgdppc, ISO).

Examples

```
Leaders0 <- MaddisonLeaders() # max GDPpc for each year.

# Presumed technology leaders without commodity leaders with narrow
# economies
Leaders1 <- MaddisonLeaders(c('ARE', 'KWT', 'QAT'))

# since 1600
MadDat1600 <- subset(MaddisonData, year>1600)
Leaders1600 <- MaddisonLeaders(c('ARE', 'KWT', 'QAT'), data=MadDat1600)

# max pop by region within percentiles of gdppc
noGDP <- is.na(MaddisonData$gdppc)
MadDat <-MaddisonData[!noGDP, ]
gdpPcts <- quantile(MadDat$gdppc, seq(0, 1, .01), na.rm=TRUE)
gdpPct <- unique(as.numeric(gdpPcts[-1]))
gdpPc <-c(gdpPct[-100], tail(gdpPct, 1)*(1+sqrt(.Machine$double.eps)))
gdp100 <- MadDat$gdppc
nObs <- nrow(MadDat)
for(i in 1:nObs){gdp100[i] <- min(gdpPc[MadDat$gdppc[i]<gdpPc])}
MadDat$gdp100 <- gdp100
MadDat$region <- MaddisonCountries[MadDat$ISO, 'region', drop=TRUE]
MadPopRgnGDP<-MaddisonLeaders(y='pop',group='region',data=MadDat,x='gdp100')
```

MaddisonSources *Maddison Project data*

Description

The Maddison project collates historical economic statistics from many sources. MaddisonSources is a list of tibble::tibbles with ISO names giving the sources of GDP per capita for different years for the said country.

MaddisonYears is a data.frame giving yearBegin and yearEnd and the number of each source in MaddisonSpources for each ISO.

Usage

```
MaddisonSources

MaddisonYears
```

Format

MaddisonSources:

A named list of tibble::tibbles, one for each country, named with the ISO country codes. Each tibble has one row for each source for the indicated ISO and two columns:

years character variable of year(s) for this source starting with year 1 CE.

source character variable giving the source for the years described.

In addition, MaddisonSources has an attribute since2008, which says, "gdppc since 2008: Total Economy Database (TED) from the Conference Board for all countries included in TED and UN national accounts statistics for all others."

MaddisonYears:

A data.frames with 4 columns:

ISO 3-letter country code.

yearBegin, yearEnd Integer year begin and end for each source.

sourceNum Integer of the source within MaddisonSources[[ISO]].

An object of class data.frame with 133 rows and 4 columns.

Source

https://www.rug.nl/ggdc/historicaldevelopment/maddison/releases/maddison-project-database-2020?lang=en"Groningen Growth and Development Centre"

Examples

```
MaddisonSources[['GBR']]
MaddisonSources[['GBR']][, 1, drop=TRUE]
# = c('1', '1252-1700 (England)', '1700-1870')
# for data from the year 1
# and for England only between 1252 and 1700, etc.

MaddisonSources[['IRN']][, 1, drop=TRUE]
# = '1820, 1870, 1913, 1950'
# for those 4 years only.

MaddisonSources[c('GBR', 'USA')]

MaddisonSources[['GBR']][, 1, drop=TRUE]
# = c('1', '1252-1700 (England)', '1700-1870')

MaddisonYears[MaddisonYears$ISO=='GBR', ] =
data.frame(
ISO=rep('GBR', 3),
yearBegin=c(1, 1252, 1700),
yearEnd  =c(1, 1700, 1870),
sourceNum=1:3
)

MaddisonSources[['EGY']][, 1, drop=TRUE]
# = c('1', '700 - 1500', '1820, 1870, 1913, 1950')

MaddisonYears[MaddisonYears$ISO=='EGY', ] =
data.frame(
ISO=rep('EGY', 6),
yearBegin=c(1,  700, 1820, 1870, 1913, 1950),
yearEnd  =c(1, 1500, 1820, 1870, 1913, 1950),
sourceNum=c(1,    2, rep(3, 4))
)
```

`path_package2` *Construct a path to a location within an installed or development package*

Description

`path_package2` returns a character vector of matches to `target`. It differs from `system.file()` in that it supports searching for a `target` file or folder possibly in `subdirs` of the working directory or in `nparents` of its parents.

Usage

```
path_package2(
```

```
  target,
  package = NULL,
  nparents = 1,
  subdirs = c("extdata", paste("inst", "extdata", sep = .Platform$file.sep))
)
```

Arguments

target	A regular expression describing the file of folder desired.
package	Name of the package to in which to search. If NULL, search in the working directory. Otherwise search in system.file(package).
nparents	integer indicate the number of parents of the working directory in which to search; default = 1.
subdirs	= c('extdata', paste('inst','extdata', sep=.Platform$file.sep))

Details

This works in a vignette searching for a target that could be in the vignettes directory of its parent package or in the package directory or in, e.g., one of subdirs = c('extdata', paste('inst', 'extdata', sep=.Platform$file.sep)).

Returns the full path to match(s) if found and a character vector of length 0 if no matches are found. The returned object also has a searched attribute being a character vector of the directories searched.

This was inspired by a desire to share with others a vignette describing how to create data objects from a file that could not itself be shared on CRAN. This is not easy, because the working director available to code in a vignette changes depending on how that code is run.

path_package2 allows the user to store the target locally, e.g., in inst/extdata but include it in .gitignore to prevent it from leaving the local computer. The vignette then decides what to do after calling path_package2() based on the length of the the object returned.

Value

a character vector with an attribute searched giving the full paths of all directories searched for target.

Examples

```
# search for a file matching a regular expression
path_package2('^mpd.*xlsx$')
# search only in the working directory
path_package2('^mpd.*xlsx$', nparents=0, subdirs=character(0))
```

summary.MaddisonLeaders *Summary method for an object of class* MaddisonLeaders

Description

summary.MaddisonLeaders returns a data.frame with columns ISO, paste0(x, 'Begin), paste0(x, 'End'), n, and p.

Usage

```
## S3 method for class 'MaddisonLeaders'
summary(object, sortBy = "ISO", decreasing = FALSE, ...)
```

Arguments

object	= object of class MaddisonLeaders.
sortBy	= column of output used for sorting; default = ISO
decreasing	default = FALSE
...	= optional arguments for summary (not used)

Value

a data.frame with columns

- ISO = One row for each level of ISO in unique(object[, 'ISO'])
- paste0(x, 'Begin) = earliest object[, paste0(x, 'Begin')] for ISO
- paste0(x, 'End'), last object[, paste0(x, 'End')] for ISO
- n = sum of (paste0(x, 'End') - paste0(x, 'Begin') + 1 for ISO.
- p = n/(paste0(x, 'End') - paste0(x, 'Begin') + 1).

)

(defaults:

- ISO = One row for each level of ISO in unique(object[, 'ISO'])
- yearBegin = earliest object[, 'yearBegin')] for ISO
- yearEnd = last object[, 'yearEnd')] for ISO
- n = sum of ('yearEnd' - 'yearBegin' + 1) for ISO.
- p = n/(yearEnd - yearBegin + 1).

[, 'yearBegin')]: R:,%20'yearBegin') [, 'yearEnd')]: R:,%20'yearEnd')

Examples

```
Leaders0 <- MaddisonLeaders() # max GDPpc for each year.
summary(Leaders0)
```

yr *year with fraction‘*

Description

yr converts a Date to a year and fraction. For example, 2025-01-01 becomes 2025.00000, while 2025-01-02 becomes 2025.00234, because (2-1)/365 is 0.00234 to 5 significant digits. However, 2024-01-02 becomes 2024.0233, because (2-1)/366 is only 0.00233 to 5 significant digits.

Usage

```
yr(x, ...)
```

Arguments

x	quantity that can be converted to a Date object using as.Date(x).
...	arguments passed to lubridate::ymd().

Value

a number (numeric vector).

See Also

lubridate::decimal_date(), lubridate::ymd()

Examples

```
Jan2_24_25 <- c('2024-01-02', '2025-01-02')
J2yr <- yr(Jan2_24_25)
J2y <- yr(as.POSIXct(Jan2_24_25))
all.equal(J2yr, J2y)
```

Index