Package ‘VisitorCounts’

April 22, 2022

Type Package

Title Modeling and Forecasting Visitor Counts Using Social Media

Version 1.0.1

Date 2022-4-19

Author Russell Goebel [aut],
Robert Bowen [aut, cre],
Beth Ann Brackett [ctb],
Kimihiro Noguchi [aut],
Dylan Way [aut]

Maintainer Robert Bowen <robertbowen.bham@gmail.com>

Description Performs modeling and forecasting of park visitor counts
using social media data and (partial) on-site visitor counts.
Specifically, the model is built based on an automatic decomposition
of the trend and seasonal components of the social media-based park visitor counts,
from which short-term forecasts of the visitor counts and percent changes
in the visitor counts can be made. A reference for generating social media-based
visitor counts can be found at
Wood, Guerry, Silver, and Lacayo (2013) <doi:10.1038/srep02976>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.2

Imports Rssa, methods

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

Depends R (>= 2.10)

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2022-04-22 02:20:02 UTC

R topics documented:

auto_decompose 2
check_arguments 4
decompose_proxy 5
estimate_lag 8
estimate_parameters 10
fit_model 12
flickr_userdays 12
generate_proxy_trend_forecasts 13
new_decomposition 14
new_visitation_forecast 14
new_visitation_model 15
park_visitation 17
plot.decomposition 18
plot.visitation_forecast 19
plot.visitation_model 20
predict.decomposition 21
predict.visitation_model 22
print.decomposition 23
print.visitation_forecast 24
print.visitation_model 25
summary.decomposition 26
summary.visitation_forecast 27
summary.visitation_model 28
visitation_model 29

Index **34**

`auto_decompose` *Automatic Decomposition Function*

Description

Automatically decomposes a time series using singular spectrum analysis. See package Rssa for details on singular spectrum analysis.

Usage

```
auto_decompose(
  time_series,
  suspected_periods = c(12, 6, 4, 3),
  proportion_of_variance_type = c("leave_out_first", "total"),
  max_proportion_of_variance = 0.995,
  log_ratio_cutoff = 0.2,
  window_length = "auto",
  num_trend_components = 2
)
```

Arguments

`time_series`	A vector which stores the time series of interest in the log scale.
`suspected_periods`	A vector which stores the suspected periods in the descending order of importance. The default option is c(12,6,4,3), corresponding to 12, 6, 4, and 3 months.
`proportion_of_variance_type`	A character string specifying the option for choosing the maximum number of eigenvalues based on the proportion of total variance explained. If "leave_out_first" is chosen, then the contribution made by the first eigenvector is ignored; otherwise, if "total" is chosen, then the contribution made by all the eigenvectors is considered.
`max_proportion_of_variance`	A numeric specifying the proportion of total variance explained using the method specified in proportion_of_variance_type. The default option is 0.995.
`log_ratio_cutoff`	A numeric specifying the threshold for the deviation between the estimated period and candidate periods in suspected_periods. The default option is 0.2, which means that, if the absolute log ratio between the estimated and candidate period is within 0.2 (approximately a 20% difference), then the estimated period is deemed equal to the candidate period.
`window_length`	A character string or positive integer specifying the window length for the SSA estimation. If "auto" is chosen, then the algorithm automatically selects the window length by taking a multiple of 12 which does not exceed half the length of time_series. The default option is "auto".
`num_trend_components`	A positive integer specifying the number of eigenvectors to be chosen for describing the trend in SSA. The default option is 2.

Value

`reconstruction`	A list containing important information about the reconstructed time series. In particular, it contains the reconstructed main trend component, overall trend component, seasonal component for each period specified in suspected_periods, and overall seasonal component.
`grouping`	A matrix containing information about the locations of the eigenvalue groups for each period in suspected_periods and trend component. The locations are indicated by '1'.
`window_length`	A numeric indicating the window length.
`ts_ssa`	An ssa object storing the singular spectrum analysis decomposition.

Examples

```
data("park_visitation")

### Decompose national parks service visitor counts and flickr photo user-days

# parameters ---------------------------------------------
```

```
suspected_periods <- c(12,6,4,3)
proportion_of_variance_type = "leave_out_first"
max_proportion_of_variance <- 0.995
log_ratio_cutoff <- 0.2

# load data ----------------------------------------------

park <- "YELL" #for Yellowstone National Park

nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, freq = 12)
nps_ts <- log(nps_ts)

pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)

# decompose time series and plot decompositions -----------
decomp_pud <- auto_decompose(pud_ts,
                                   suspected_periods,
                             proportion_of_variance_type = proportion_of_variance_type,
                                   max_proportion_of_variance,
                                   log_ratio_cutoff)
plot(decomp_pud)

decomp_nps <- auto_decompose(nps_ts,suspected_periods,
                             proportion_of_variance_type = proportion_of_variance_type,
                                   max_proportion_of_variance,log_ratio_cutoff)

plot(decomp_nps)
```

check_arguments *Check Arguments*

Description

Check arguments.

Usage

```
check_arguments(
  log_scale,
  popularity_proxy,
  onsite_usage,
  ref_series,
  constant,
  omit_trend,
  ...
)
```

Arguments

`log_scale`	A Boolean specifying whether or not the results should be returned in the log scale.
`popularity_proxy`	A vector which stores a time series which may be used as a proxy for the social media time series in the log scale. The length of popularity_proxy must be the same as that of onsite_usage.
`onsite_usage`	A vector which stores on-site usage in the log scale for a particular social media platform and recreational site.
`ref_series`	A numeric vector specifying the original visitation series in the log scale. If such series is available, then its length must be the same as that of time_series.
`constant`	A numeric specifying the constant term in the model. This constant is understood as the mean of the trend-adjusted time_series. If ref_series is supplied, the constant is overwritten by the least squares estimate.
`omit_trend`	A Boolean specifying whether or not to consider the trend component to be 0.
`...`	Additional arguments.

Value

No return value, called for extra information.

`decompose_proxy` *Decompose Popularity Proxy*

Description

Decomposes the popularity proxy time series into trend and seasonality components.

Usage

```
decompose_proxy(
  onsite_usage,
  popularity_proxy = NULL,
  suspected_periods = c(12, 6, 4, 3),
  proportion_of_variance_type = c("leave_out_first", "total"),
  max_proportion_of_variance = 0.995,
  log_ratio_cutoff = 0.2,
  window_length = "auto",
  num_trend_components = 2,
  criterion = c("cross-correlation", "MSE", "rank"),
  possible_lags = -36:36,
  leave_off = 6,
  estimated_change = 0,
  order_of_polynomial_approximation = 7,
  order_of_derivative = 1,
  ref_series = NULL,
  beta = "estimate",
  constant = 0,
  log_scale = TRUE,
  spline = FALSE,
  parameter_estimates = c("separate", "joint"),
  omit_trend = TRUE,
  onsite_usage_decomposition,
  ...
)
```

Arguments

`onsite_usage`	A vector which stores on-site usage in the log scale for a particular social media platform and recreational site.
`popularity_proxy`	A vector which stores a time series which may be used as a proxy for the social media time series in the log scale. The length of popularity_proxy must be the same as that of onsite_usage. The default option is NULL, in which case, no proxy needs to be supplied.
`suspected_periods`	A vector which stores the suspected periods in the descending order of importance. The default option is c(12,6,4,3), corresponding to 12, 6, 4, and 3 months if observations are monthly.
`proportion_of_variance_type`	A character string specifying the option for choosing the maximum number of eigenvalues based on the proportion of total variance explained. If "leave_out_first" is chosen, then the contribution made by the first eigenvector is ignored; otherwise, if "total" is chosen, then the contribution made by all the eigenvectors is considered.
`max_proportion_of_variance`	A numeric specifying the proportion of total variance explained using the method specified in proportion_of_variance_type. The default option is 0.995.
`log_ratio_cutoff`	A numeric specifying the threshold for the deviation between the estimated period and candidate periods in suspected_periods. The default option is 0.2, which means that if the absolute log ratio between the estimated and candidate period is within 0.2 (approximately a 20 percent difference), then the estimated period is deemed equal to the candidate period.
`window_length`	A character string or positive integer specifying the window length for the SSA estimation. If "auto" is chosen, then the algorithm automatically selects the window length by taking a multiple of 12 which does not exceed half the length of onsite_usage. The default option is "auto".
`num_trend_components`	A positive integer specifying the number of eigenvectors to be chosen for describing the trend in SSA. The default option is 2. This is relevant only when omit_trend is FALSE.

`criterion`	A character string specifying the criterion for estimating the lag in popularity_proxy. If "cross-correlation" is chosen, it chooses the lag that maximizes the correlation coefficient between lagged popularity_proxy and onsite_usage. If "MSE" is chosen, it does so by identifying the lagged popularity_proxy whose derivative is closest to that of onsite_usage by minimizing the mean squared error. If "rank" is chosen, it does so by firstly ranking the square errors of the derivatives and identifying the lag which would minimize the mean rank.
`possible_lags`	A numeric vector specifying all the candidate lags for popularity_proxy. The default option is -36:36. This is relevant only when omit_trend is FALSE.
`leave_off`	A positive integer specifying the number of observations to be left off when estimating the lag. The default option is 6. This is relevant only when omit_trend is FALSE.
`estimated_change`	A numeric specifying the estimated change in the visitation trend. The default option is 0, implying no change in the trend.
`order_of_polynomial_approximation`	A numeric specifying the order of the polynomial approximation of the difference between time series used in `estimate_lag`. The default option is 7, the seventh-degree polynomial. This is relevant only when omit_trend is FALSE.
`order_of_derivative`	A numeric specifying the order of derivative for the approximated difference between time_series1 and lagged time_series2. The default option is 1, the first derivative. This is relevant only when omit_trend is FALSE.
`ref_series`	A numeric vector specifying the original visitation series in the log scale. The default option is NULL, implying that no such series is available. If such series is available, then its length must be the same as that of time_series.
`beta`	A numeric or a character string specifying the seasonality adjustment factor. The default option is "estimate", in which case, it is estimated by using the Fisher's z-transformed lag-12 autocorrelation. Even if an actual value is supplied, if ref_series is supplied, it is overwritten by the least squares estimate.
`constant`	A numeric specifying the constant term in the model. This constant is understood as the mean of the trend-adjusted time_series. The default option is 0, implying that the time_series well represents the actual visitation counts, which is rarely the case. If ref_series is supplied, the constant is overwritten by the least squares estimate.
`log_scale`	A Boolean specifying whether or not the results should be returned in the log scale. The default option is TRUE, in which case, the results are returned in the log scale.
`spline`	A Boolean specifying whether or not to use a smoothing spline for the lag estimation. This is relevant only when omit_trend is FALSE.
`parameter_estimates`	A character string specifying how to estimate beta and constant parameters should a reference series be supplied. Both options use least squares estimates, but "separate" indicates that the differenced series should be used to estimate beta separately from the constant, while "joint" indicates to estimate both using non-differenced detrended series.

`omit_trend`	A Boolean specifying whether or not to consider the trend component to be 0. The default option is TRUE, in which case, the trend component is 0.
`onsite_usage_decomposition`	A "decomposition" class object containing decomposition data for the onsite usage time series (outputs from ‘auto_decompose‘).
`...`	Additional arguments to be passed onto the smoothing spline (`smooth.spline`).

Value

`proxy_decomposition`	A "decomposition" object representing the automatic decomposition obtained from popularity_proxy (see auto_decompose())
`lagged_proxy_trend_and_forecasts_window`	A ‘ts‘ object storing the potentially lagged popularity proxy trend and any forecasts needed due to the lag
`ts_trend_window`	A ‘ts‘ object storing the trend component of the onsite social media usage. This trend component is potentially truncated to match available popularity proxy data.
`ts_seasonality_window`	A ‘ts‘ object storing the seasonality component of the onsite social media usage. This seasonality component is potentially truncated to match available popularity proxy data.
`latest_starttime`	A ‘tsp‘ attribute of a ‘ts‘ object representing the latest of the two start times of the potentially lagged populairty proxy and the onsite social media usage.
`endtime`	A ‘tsp‘ attribute of a ‘ts‘ object representing the time of the final onsite usage observation.
`forecasts_needed`	An integer representing the number of forecasts of popularity_proxy needed to obtain all fitted values. Negative values indicate extra observations which may be useful for predictions.
`lag_estimate`	A list storing both the MSE-based esitmate and Rank-based estimates for the lag.

`estimate_lag` *Estimate Lag Function*

Description

Uses polynomial approximation and derivatives for time series objects to estimate lag between series.

Usage

```
estimate_lag(
  time_series1,
  time_series2,
  possible_lags,
  method = c("cross-correlation", "MSE", "rank"),
  leave_off,
  estimated_change = 0,
  order_of_polynomial_approximation = 7,
  order_of_derivative = 1,
  spline = FALSE,
  ...
)
```

Arguments

`time_series1`	A numeric vector which stores the time series of interest in the log scale.
`time_series2`	A numeric vector which stores the trend proxy time series in the log scale. The length of trend_proxy must be the same as that of time_series1.
`possible_lags`	A numeric vector specifying all the candidate lags for trend_proxy. The default option is -36:36.
`method`	A character vector specifying the method used to obtain the lag estimate. "polynomial" uses polynomial approximation, while "cross-correlation" uses cross-correlation.
`leave_off`	A positive integer specifying the number of observations to be left off when estimating the lag.
`estimated_change`	A numeric specifying the estimated change in the visitation trend. The default option is 0, implying no change in the trend.
`order_of_polynomial_approximation`	A numeric specifying the order of the polynomial approximation of the difference between time series used in `estimate_lag`. The default option is 7, the seventh-degree polynomial.
`order_of_derivative`	A numeric specifying the order of derivative for the approximated difference between time_series1 and lagged time_series2. The default option is 1, the first derivative.
`spline`	A Boolean specifying whether or not to use a smoothing spline for the lag estimation.
`...`	Additional arguments to be passed onto the `smooth.spline` function, if method is "polynomial".

Value

`cc_lag`	A numeric indicating the estimated lag with the cross-correlation criterion.
`mse_criterion`	A numeric indicating the estimated lag with the MSE criterion.
`rank_criterion`	A numeric indicating the estimate lag with the rank criterion.

Examples

```
# Generate dataset with known lag and recover this lag --------------#'

lag <- 3
n <- 156
start_year <- 2005
frequency <- 12
trend_function <- function(x) x^2

x <- seq(-3,3, length.out = n)

y1 <- ts(trend_function(x),start = start_year, freq = frequency)
y2 <- stats::lag(y1, k = lag)


# Recover lag
estimate_lag(y1,y2, possible_lags = -36:36,
             method = "rank",leave_off = 0, spline = FALSE)
```

`estimate_parameters` *Estimate Parameters for Visitation Model*

Description

Estimate the two parameters (y-intercept and seasonality factor) for the visitation model.

Usage

```
estimate_parameters(
  popularity_proxy_decomposition_data = NULL,
  onsite_usage,
  onsite_usage_decomposition,
  omit_trend,
  ref_series,
  beta,
  constant,
  parameter_estimates,
  ...
)
```

Arguments

`popularity_proxy_decomposition_data`
: A "decomposition" class object containing decomposition data for the popularity proxy time series (outputs from ‘auto_decompose‘).

`onsite_usage`	A vector which stores on-site usage in the log scale for a particular social media platform and recreational site.
`onsite_usage_decomposition`	A "decomposition" class object containing decomposition data for the onsite usage time series (outputs from ‘auto_decompose‘).
`omit_trend`	A Boolean specifying whether or not to consider the trend component to be 0.
`ref_series`	A numeric vector specifying the original visitation series in the log scale.
`beta`	A numeric or a character string specifying the seasonality adjustment factor. The default option is "estimate", in which case, it is estimated by using the Fisher's z-transformed lag 12 autocorrelation. Even if an actual value is supplied, if ref_series is supplied, it is overwritten by the least squares estimate.
`constant`	A numeric specifying the constant term in the model. This constant is understood as the mean of the trend-adjusted time_series. If ref_series is supplied, the constant is overwritten by the least squares estimate.
`parameter_estimates`	A character string specifying how to estimate beta and constant parameters should a reference series be supplied. Both options use least squares estimates, but "separate" indicates that the differenced series should be used to estimate beta separately from the constant, while "joint" indicates to estimate both using non-differenced detrended series.
`...`	Additional arguments.

Value

`lagged_proxy_trend_and_forecasts_window`	A ‘ts‘ object storing the potentially lagged popularity proxy trend and any forecasts needed due to the lag
`ts_trend_window`	A ‘ts‘ object storing the trend component of the onsite social media usage. This trend component is potentially truncated to match available popularity proxy data.
`ts_seasonality_window`	A ‘ts‘ object storing the seasonality component of the onsite social media usage. This seasonality component is potentially truncated to match available popularity proxy data.
`latest_starttime`	A ‘tsp‘ attribute of a ‘ts‘ object representing the latest of the two start times of the potentially lagged populairty proxy and the onsite social media usage.
`endtime`	A ‘tsp‘ attribute of a ‘ts‘ object representing the time of the final onsite usage observation.
`beta`	A numeric storing the estimated seasonality adjustment factor.
`constant`	A numeric storing estimated constant term used in the model.

`fit_model` *Fit Model*

Description

Fit the visitation model.

Usage

```
fit_model(
  parameter_estimates_and_time_series_windows,
  omit_trend,
  log_scale,
  ...
)
```

Arguments

`parameter_estimates_and_time_series_windows`	# a list storing the outputs of ‘estimate_parameters‘, including parameter estimates ‘beta‘ and ‘constant‘ as well as data pertaining to time series windows.
`omit_trend`	A Boolean specifying whether or not to consider the trend component to be 0.
`log_scale`	A Boolean specifying whether or not the results should be returned in the log scale.
`...`	Additional arguments

Value

`visitation_fit`	A vector storing fitted values of visitation model.

`flickr_userdays` *Popularity of Flickr, in User-Days*

Description

A time series representing the popularity of Flickr in the United States, as measured in user-days. Here, user-days count the number of unique users posting on Flickr on a given day.

Usage

```
flickr_userdays
```

Format

A time series object with 156 observations.

Source

Flickr. (2019). Retrieved October, 2019, from https://flickr.com/

`generate_proxy_trend_forecasts` *Generate Proxy Trend Forecasts*

Description

Generating proxy trend forecasts from objects of the class "visitation_model".

Usage

```
generate_proxy_trend_forecasts(
  object,
  n_ahead,
  starttime,
  endtime,
  proxy_trend_correction,
  ts_frequency
)
```

Arguments

`object`	A visitation model object.
`n_ahead`	The number of desired forecasts.
`starttime`	The start time of the desired forecasts.
`endtime`	The end time of the desired forecasts.
`proxy_trend_correction`	The lag correction needed on the proxy trend.
`ts_frequency`	Frequency of the time series to forecast.

Value

A time series object storing forecasts for the proxy trend.

`new_decomposition` *"decomposition" Constructor Function*

Description

Constructs objects of the "decomposition" class.

Usage

```
new_decomposition(reconstruction_list, grouping_matrix, window_length, ts_ssa)
```

Arguments

`reconstruction_list`	A list containing important information about the reconstructed time series. In particular, it contains the reconstructed main trend component, overall trend component, seasonal component for each period specified in suspected_periods, and overall seasonal component.
`grouping_matrix`	A matrix containing information about the locations of the eigenvalue groups for each period in suspected_periods and trend component. The locations are indicated by '1'.
`window_length`	A numeric indicating the window length.
`ts_ssa`	An object of the class "ssa".

Value

A list of the class "decomposition".

`new_visitation_forecast` *visitation_forecast Class*

Description

Class for visitation_model predictions (for use with predict.visitation_model()).

Usage

```
new_visitation_forecast(
  forecasts,
  n_ahead,
  proxy_forecasts,
  onsite_usage_forecasts,
  beta,
  constant,
  criterion,
  past_observations,
  lag_estimate
)
```

Arguments

`forecasts`	A time series of forecasts for the visitation model.
`n_ahead`	An integer describing the number of forecasts made.
`proxy_forecasts`	A time series of forecasts of the popularity proxy series.
`onsite_usage_forecasts`	A time series of forecasts of the original time series.
`beta`	A numeric or a character string specifying the seasonality adjustment factor. The default option is "estimate", in which case, it is estimated by using the Fisher's z-transformed lag-12 autocorrelation. Even if an actual value is supplied, if ref_series is supplied, it is overwritten by the least squares estimate.
`constant`	A numeric specifying the constant term in the model. This constant is understood as the mean of the trend-adjusted time_series. The default option is 0, implying that the time_series well represents the actual visitation counts, which is rarely the case. If ref_series is supplied, the constant is overwritten by the least squares estimate.
`criterion`	One of "MSE" or "Nonparametric", to specify the criterion used to select the lag.
`past_observations`	One of "none", "fitted", or "ref_series". If "fitted", past model fitted values are used. If "ref_series", the reference series in the visitation model object is used. Note that if difference = TRUE, one of these is needed to forecast the first difference.
`lag_estimate`	A numeric value specifying the estimated lag in the visitation model.

Value

Object of class "Visitation_forecast".

`new_visitation_model` *"visitation_model" Constructor Function*

Description

Constructs objects of the "visitation_model" class.

Usage

```
new_visitation_model(
  visitation_fit,
  differenced_fit,
  beta,
  constant,
  lag_estimate,
  proxy_decomposition,
  onsite_usage_decomposition,
  forecasts_needed,
  ref_series,
  criterion,
  omit_trend,
  call
)
```

Arguments

`visitation_fit`	A time series storing the fitted values of the visitation model.
`differenced_fit`	A time series storing the differenced fitted values of the visitation model.
`beta`	Seasonality adjustment factor.
`constant`	A double describing the constant term used in the model.
`lag_estimate`	An integer representing the lag parameter for the model fit.
`proxy_decomposition`	A decomposition class object representing the decomposition of a popularity measure (e.g., US Photo-User-Days).
`onsite_usage_decomposition`	A decomposition class object representing the decomposition of time series (e.g., park Photo-User-Days).
`forecasts_needed`	An integer describing how many forecasts for the proxy_decomposition are needed for the fit.
`ref_series`	A reference time series (or NULL) used in the model fit.
`criterion`	A character string specifying the criterion for estimating the lag in popularity_proxy. If "cross-correlation" is chosen, it chooses the lag that maximizes the correlation coefficient between lagged popularity_proxy and onsite_usage. If "MSE" is chosen, it does so by identifying the lagged popularity_proxy whose derivative is closest to that of onsite_usage by minimizing the mean squared error. If "rank" is chosen, it does so by firstly ranking the square errors of the derivatives and identifying the lag which would minimize the mean rank.
`omit_trend`	A Boolean specifying whether or not to consider the NPS trend to be zero.
`call`	A call for the visitation model.

Value

A list of the class "model_forecast".

`park_visitation` *National Park Visitation Counts and Associated Photo-User-Days Data.*

Description

A data frame storing monthly visitation counts by National Park Service (NPS) for 20 popular US national parks and associated Flickr photo-user-days (PUD). Here, photo-user-days (PUD) count the number of unique users posting a photo on Flickr on a given day from within the boundaries of a given National Park.

Usage

```
park_visitation
```

Format

A data frame with 3276 rows and 4 variables.

date Date of monthly observation, in year-month-day format.

park National Park alpha code identifying a National Park.

pud Flickr photo-user-days (PUD). Here, PUD count the number of unique users posting a photo on flickr on a given day from within the boundaries of a given National Park.

nps Visitation count for the corresponding park and month given by the National Park Service (NPS).

Source

National Park Service (2018). National park service visitor use statistics. Retrieved May 10, 2018 from https://irma.nps.gov/Stats/

Flickr (2019). Retrieved October, 2019, from https://flickr.com/

`plot.decomposition` *Decomposition Plot Methods*

Description

Methods for plotting objects of the class "decomposition".

Usage

```
## S3 method for class 'decomposition'
plot(x, type = c("full", "period", "classical"), legend = TRUE, ...)
```

Arguments

`x`	An object of class "decomposition".
`type`	A character string. One of "full","period", or "classical". If "full", the full reconstruction is plotted. If "period", the reconstruction of each period is plotted individually. If "classical", the trend and seasonality are plotted.
`legend`	A Boolean specifying whether a legend should be added when type is "full". The default option is TRUE.
`...`	Additional arguments.

Value

A plot of the reconstruction in the "decomposition" class object.

Examples

```
data("park_visitation")

park <- "YELL"
nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, frequency = 12)
nps_ts <- log(nps_ts)

pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, frequency = 12)
pud_ts <- log(pud_ts)
nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, frequency = 12)
nps_ts <- log(nps_ts)



decomposition_pud <- auto_decompose(pud_ts)
decomposition_nps <- auto_decompose(nps_ts)

plot(decomposition_pud,lwd = 2)
plot(decomposition_pud,type = "period")
plot(decomposition_pud,type = "classical")
plot(decomposition_nps,legend = TRUE)


plot(decomposition_nps,type = "period")
plot(decomposition_nps,type = "classical")
```

`plot.visitation_forecast` *visitation_forecast Plot Methods*

Description

Methods for plotting objects of the class "visitation_forecast".

Usage

```
## S3 method for class 'visitation_forecast'
plot(x, type = c("fitted"), difference = FALSE, ...)
```

Arguments

`x`	An object of class "decomposition".
`type`	A character string. One of "full","period", or "classical". If "full", the full reconstruction is plotted. If "period", the reconstruction of each period is plotted individually. If "classical", the trend and seasonality are plotted.
`difference`	A Boolean specifying whether to plot the original fit or differenced series. The default option is FALSE, in which case, the series is not differenced.
`...`	Additional arguments.

Value

No return value, called for plotting objects of the class "visitation_forecast".

Examples

```
#' #Example:

data("park_visitation")
data("flickr_userdays")

n_ahead <- 12
park <- "YELL"
pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)
trend_proxy <- log(flickr_userdays)
```

```
mf <- visitation_model(pud_ts,trend_proxy)
vf <- predict(mf,12, only_new = TRUE)
plot(vf)
```

plot.visitation_model *visitation_model Plot Methods*

Description

Methods for plotting objects of the class "decomposition".

Usage

```
## S3 method for class 'visitation_model'
plot(x, type = c("fitted"), difference = FALSE, ...)
```

Arguments

x	An object of class "decomposition".
type	A character string. One of "full","period", or "classical". If "full", the full reconstruction is plotted. If "period", the reconstruction of each period is plotted individually. If "classical", the trend and seasonality are plotted.
difference	A Boolean specifying whether to plot the original fit or differenced series. The default option is FALSE, in which case, the series is not differenced.
...	Additional arguments.

Value

No return value, called for plotting objects of the class "decomposition".

Examples

```
data("park_visitation")
data("flickr_userdays")

park <- "YELL"
pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)

nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, freq = 12)
nps_ts <- log(nps_ts)

nps_decomp <- auto_decompose(nps_ts)

trend_proxy <- log(flickr_userdays)

vm <- visitation_model(pud_ts,trend_proxy,ref_series = nps_ts)
plot(vm)
```

`predict.decomposition` *Predict Decomposition*

Description

Methods for generating predictions from objects of the class "decomposition".

Usage

```
## S3 method for class 'decomposition'
predict(object, n_ahead, only_new = TRUE, ...)
```

Arguments

`object`	An object of class "decomposition".
`n_ahead`	An integer describing the number of forecasts to make.
`only_new`	A boolean describing whether or not to include past values.
`...`	Additional arguments.

Value

`forecasts`	A vector with overall forecast values.
`trend_forecasts`	A vector with trend forecast values.
`seasonality_forecasts`	A vector with seasonality forecast values.

Examples

```
data("park_visitation")
suspected_periods <- c(12,6,4,3)
proportion_of_variance_type = "leave_out_first"
max_proportion_of_variance <- 0.995
log_ratio_cutoff <- 0.2

park <- "DEVA"

nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, freq = 12)
nps_ts <- log(nps_ts)

pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)

nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, freq = 12)
nps_ts <- log(nps_ts)

decomp_pud <- auto_decompose(pud_ts,
                                      suspected_periods,
                                proportion_of_variance_type = proportion_of_variance_type,
                                      max_proportion_of_variance,
                                     log_ratio_cutoff)
n_ahead = 36
pud_predictions <- predict(decomp_pud,n_ahead = n_ahead, only_new = FALSE)
```

---

predict.visitation_model

Predict Visitation Model

Description

Methods for generating predictions from objects of the class "visitation_model".

Usage

```
## S3 method for class 'visitation_model'
predict(
  object,
  n_ahead,
  only_new = TRUE,
  difference = FALSE,
  past_observations = c("fitted", "reference"),
  ...
)
```

Arguments

`object`	An object of class "visitation_model".
`n_ahead`	An integer indicating how many observations to forecast.
`only_new`	A Boolean specifying whether to include only the forecasts (if TRUE) or the full reconstruction (if FALSE). The default option is TRUE.
`difference`	A Boolean specifying whether to forecast differences (if TRUE) or the original series (if FALSE). The default option is FALSE.
`past_observations`	A character string; one of "fitted" or "reference". Here, "fitted" uses the fitted values of the visitation model, while "reference" uses values supplied in ‘ref_series’.
`...`	Additional arguments.

Value

A predictions for the automatic decomposition.

`forecasts`	A vector with forecast values.
`n_ahead`	A numeric that shows the number of steps ahead.
`proxy_forecasts`	A vector for the proxy of trend forecasts.
`onsite_usage_forecasts`	A vector for the visitation forecasts.
`beta`	A numeric for the seasonality adjustment factor.
`constant`	A numeric for the value of the constant in the model.
`criterion`	A string which specifies the method used to select the appropriate lag. Only applicable if the trend component is part of the forecasts.
`past_observations`	A vector which specifies the fitted values for the past observations.
`lag_estimate`	A numeric for the estimated lag. Only applicable if the trend component is part of the forecasts.

Examples

```
data("park_visitation")
data("flickr_userdays")

n_ahead <- 36
park <- "ROMO"
pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, frequency = 12)
pud_ts <- log(pud_ts)

nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, frequency = 12)
nps_ts <- log(nps_ts)
popularity_proxy <- log(flickr_userdays)

vm <- visitation_model(pud_ts,popularity_proxy, ref_series = nps_ts,omit_trend = TRUE)
predict_vm <- predict(vm,n_ahead, difference = TRUE,
                      only_new = FALSE, past_observations = "reference")
plot(predict_vm, difference = FALSE)
predict_vm2 <- predict(vm,n_ahead, difference = TRUE,
                       only_new = FALSE, past_observations = "reference")
plot(predict_vm2, difference = FALSE)
```

`print.decomposition` *Decomposition Summary Method*

Description

S3 method for summarizing objects of the class "decomposition".

Usage

```
## S3 method for class 'decomposition'
print(x, ...)
```

Arguments

`x`	An object of class "decomposition".
`...`	Additional arguments.

Value

A "decomposition" class object.

Examples

```
 data("park_visitation")

park <- "YELL"
nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, freq = 12)
nps_ts <- log(nps_ts)

pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)
nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, freq = 12)
nps_ts <- log(nps_ts)


decomposition_pud <- auto_decompose(pud_ts)
decomposition_nps <- auto_decompose(nps_ts)
summary(decomposition_pud)
summary(decomposition_nps)
```

---

print.visitation_forecast *visitation_forecast Summary Method*

Description

Methods for summarizing objects of the class "decomposition".

Usage

```
## S3 method for class 'visitation_forecast'
print(x, ...)
```

Arguments

`x`	An object of class "decomposition".
`...`	Additional arguments.

Value

A "decomposition" class object.

Examples

```
#Example:

data("park_visitation")
data("flickr_userdays")

n_ahead <- 12
park <- "YELL"
pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)
trend_proxy <- log(flickr_userdays)

mf <- visitation_model(pud_ts,trend_proxy)
vf <- predict(mf,12, only_new = FALSE)
summary(vf)
```

`print.visitation_model` *visitation_model Summary Method*

Description

Methods for summarizing objects of the class "decomposition".

Usage

```
## S3 method for class 'visitation_model'
print(x, ...)
```

Arguments

`x`	An object of class "decomposition".
`...`	Additional arguments.

Value

A "decomposition" class object.

Examples

```
#Example:

data("park_visitation")
data("flickr_userdays")

n_ahead <- 12
park <- "YELL"
pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)
trend_proxy <- log(flickr_userdays)

vm <- visitation_model(pud_ts,trend_proxy)
summary(vm)
```

`summary.decomposition` *Decomposition Summary Method*

Description

S3 method for summarizing objects of the class "decomposition".

Usage

```
## S3 method for class 'decomposition'
summary(object, ...)
```

Arguments

`object`	An object of class "decomposition".
`...`	Additional arguments.

Value

A "decomposition" class object.

Examples

```
 data("park_visitation")

park <- "YELL"
nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, freq = 12)
nps_ts <- log(nps_ts)

pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)
nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, freq = 12)
nps_ts <- log(nps_ts)#'
```

```
decomposition_pud <- auto_decompose(pud_ts)
decomposition_nps <- auto_decompose(nps_ts)
summary(decomposition_pud)
summary(decomposition_nps)
```

---

summary.visitation_forecast

visitation_forecast Summary Method

Description

Methods for summarizing objects of the class "decomposition".

Usage

```
## S3 method for class 'visitation_forecast'
summary(object, ...)
```

Arguments

`object`	An object of class "decomposition".
`...`	Additional arguments.

Value

A "decomposition" class object.

Examples

```
#Example:

data("park_visitation")
data("flickr_userdays")

n_ahead <- 12
park <- "YELL"
pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)
trend_proxy <- log(flickr_userdays)

mf <- visitation_model(pud_ts,trend_proxy)
vf <- predict(mf,12, only_new = FALSE)
summary(vf)
```

summary.visitation_model

visitation_model Summary Method

Description

Methods for summarizing objects of the class "decomposition".

Usage

```
## S3 method for class 'visitation_model'
summary(object, ...)
```

Arguments

`object`	An object of class "decomposition".
`...`	Additional arguments.

Value

A "decomposition" class object.

Examples

```

#Example:

data("park_visitation")
data("flickr_userdays")

n_ahead <- 12
park <- "YELL"
pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, freq = 12)
pud_ts <- log(pud_ts)
trend_proxy <- log(flickr_userdays)

vm <- visitation_model(pud_ts,trend_proxy)
summary(vm)
```

`visitation_model` *Visitation Model*

Description

Fits a time series model that uses social media posts and popularity of the social media to model visitation to recreational sites.

Usage

```
visitation_model(
  onsite_usage,
  popularity_proxy = NULL,
  suspected_periods = c(12, 6, 4, 3),
  proportion_of_variance_type = c("leave_out_first", "total"),
  max_proportion_of_variance = 0.995,
  log_ratio_cutoff = 0.2,
  window_length = "auto",
  num_trend_components = 2,
  criterion = c("cross-correlation", "MSE", "rank"),
  possible_lags = -36:36,
  leave_off = 6,
  estimated_change = 0,
  order_of_polynomial_approximation = 7,
  order_of_derivative = 1,
  ref_series = NULL,
  beta = "estimate",
  constant = 0,
  log_scale = TRUE,
  spline = FALSE,
  parameter_estimates = c("separate", "joint"),
  omit_trend = TRUE,
  ...
)
```

Arguments

`onsite_usage`	A vector which stores on-site usage in the log scale for a particular social media platform and recreational site.
`popularity_proxy`	A vector which stores a time series which may be used as a proxy for the social media time series in the log scale. The length of popularity_proxy must be the same as that of onsite_usage. The default option is NULL, in which case, no proxy needs to be supplied.

`suspected_periods`	A vector which stores the suspected periods in the descending order of importance. The default option is c(12,6,4,3), corresponding to 12, 6, 4, and 3 months if observations are monthly.
`proportion_of_variance_type`	A character string specifying the option for choosing the maximum number of eigenvalues based on the proportion of total variance explained. If "leave_out_first" is chosen, then the contribution made by the first eigenvector is ignored; otherwise, if "total" is chosen, then the contribution made by all the eigenvectors is considered.
`max_proportion_of_variance`	A numeric specifying the proportion of total variance explained using the method specified in proportion_of_variance_type. The default option is 0.995.
`log_ratio_cutoff`	A numeric specifying the threshold for the deviation between the estimated period and candidate periods in suspected_periods. The default option is 0.2, which means that if the absolute log ratio between the estimated and candidate period is within 0.2 (approximately a 20 percent difference), then the estimated period is deemed equal to the candidate period.
`window_length`	A character string or positive integer specifying the window length for the SSA estimation. If "auto" is chosen, then the algorithm automatically selects the window length by taking a multiple of 12 which does not exceed half the length of onsite_usage. The default option is "auto".
`num_trend_components`	A positive integer specifying the number of eigenvectors to be chosen for describing the trend in SSA. The default option is 2. This is relevant only when omit_trend is FALSE.
`criterion`	A character string specifying the criterion for estimating the lag in popularity_proxy. If "cross-correlation" is chosen, it chooses the lag that maximizes the correlation coefficient between lagged popularity_proxy and onsite_usage. If "MSE" is chosen, it does so by identifying the lagged popularity_proxy whose derivative is closest to that of onsite_usage by minimizing the mean squared error. If "rank" is chosen, it does so by firstly ranking the square errors of the derivatives and identifying the lag which would minimize the mean rank.
`possible_lags`	A numeric vector specifying all the candidate lags for popularity_proxy. The default option is -36:36. This is relevant only when omit_trend is FALSE.
`leave_off`	A positive integer specifying the number of observations to be left off when estimating the lag. The default option is 6. This is relevant only when omit_trend is FALSE.
`estimated_change`	A numeric specifying the estimated change in the visitation trend. The default option is 0, implying no change in the trend.
`order_of_polynomial_approximation`	A numeric specifying the order of the polynomial approximation of the difference between time series used in `estimate_lag`. The default option is 7, the seventh-degree polynomial. This is relevant only when omit_trend is FALSE.

`order_of_derivative`	A numeric specifying the order of derivative for the approximated difference between time_series1 and lagged time_series2. The default option is 1, the first derivative. This is relevant only when omit_trend is FALSE.
`ref_series`	A numeric vector specifying the original visitation series in the log scale. The default option is NULL, implying that no such series is available. If such series is available, then its length must be the same as that of time_series.
`beta`	A numeric or a character string specifying the seasonality adjustment factor. The default option is "estimate", in which case, it is estimated by using the Fisher's z-transformed lag-12 autocorrelation. Even if an actual value is supplied, if ref_series is supplied, it is overwritten by the least squares estimate.
`constant`	A numeric specifying the constant term in the model. This constant is understood as the mean of the trend-adjusted time_series. The default option is 0, implying that the time_series well represents the actual visitation counts, which is rarely the case. If ref_series is supplied, the constant is overwritten by the least squares estimate.
`log_scale`	A Boolean specifying whether or not the results should be returned in the log scale. The default option is TRUE, in which case, the results are returned in the log scale.
`spline`	A Boolean specifying whether or not to use a smoothing spline for the lag estimation. This is relevant only when omit_trend is FALSE.
`parameter_estimates`	A character string specifying how to estimate beta and constant parameters should a reference series be supplied. Both options use least squares estimates, but "separate" indicates that the differenced series should be used to estimate beta separately from the constant, while "joint" indicates to estimate both using non-differenced detrended series.
`omit_trend`	A Boolean specifying whether or not to consider the trend component to be 0. The default option is TRUE, in which case, the trend component is 0.
`...`	Additional arguments to be passed onto the smoothing spline (`smooth.spline`).

Value

`visitation_fit`	A vector storing fitted values of visitation model.
`differenced_fit`	A vector storing differenced fitted values of visitation model. (Equal to `diff(visitation_fit)`.)
`beta`	A numeric storing the estimated seasonality adjustment factor.
`constant`	A numeric storing estimated constant term used in the model.
`proxy_decomposition`	A "decomposition" object representing the automatic decomposition obtained from popularity_proxy (see auto_decompose())
`time_series_decomposition`	A "decomposition" object representing the automatic decomposition obtained from time_series (see auto_decompose())

forecasts_needed	An integer representing the number of forecasts of popularity_proxy needed to obtain all fitted values. Negative values indicate extra observations which may be useful for predictions.
lag_estimate	A list storing both the MSE-based estimate and Rank-based estimates for the lag.
criterion	A string; one of "cross-correlation", "MSE", or "rank", specifying the method used to select the appropriate lag.
ref_series	The reference series, if one was supplied.
omit_trend	Whether or not trend was considered 0 in the model.
call	The model call.

See Also

See `predict.visitation_model` for forecast methods, `estimate_lag` for details on the lag estimation, and `auto_decompose` for details on the automatic decomposition of time series using SSA. See the package Rssa for details regarding singular spectrum analysis.

Examples

```
### load data --------------------

data("park_visitation")
data("flickr_userdays")

park <- "YELL" #Yellowstone National Park
pud_ts <- ts(park_visitation[park_visitation$park == park,]$pud, start = 2005, frequency = 12)
pud_ts <- log(pud_ts)

nps_ts <- ts(park_visitation[park_visitation$park == park,]$nps, start = 2005, frequency = 12)
nps_ts <- log(nps_ts)
popularity_proxy <- log(flickr_userdays)


### fit two models ---------------

vm_pud_only <- visitation_model(pud_ts,popularity_proxy = popularity_proxy, omit_trend = FALSE)
vm_ref_series <- visitation_model(pud_ts,
                                  popularity_proxy,
                                  ref_series = nps_ts,
                                  parameter_estimates = "separate",
                                  possible_lags = -36:36,
                                  omit_trend = TRUE)


### visualize fit ------------------

plot(vm_pud_only, ylim = c(-3,3), difference = TRUE)
lines(diff(nps_ts), col = "red")
```

```
plot(vm_ref_series, ylim = c(-3,3), difference = TRUE)
lines(diff(nps_ts), col = "red")
```

Index

∗ **datasets**
    flickr_userdays, 12
    park_visitation, 17

auto_decompose, 2, *32*

check_arguments, 4

decompose_proxy, 5

estimate_lag, 8, *32*
estimate_parameters, 10

fit_model, 12
flickr_userdays, 12

generate_proxy_trend_forecasts, 13

new_decomposition, 14
new_visitation_forecast, 14
new_visitation_model, 15

park_visitation, 17
plot.decomposition, 18
plot.visitation_forecast, 19
plot.visitation_model, 20
predict.decomposition, 21
predict.visitation_model, 22, *32*
print.decomposition, 23
print.visitation_forecast, 24
print.visitation_model, 25

Rssa, *2*, *32*

summary.decomposition, 26
summary.visitation_forecast, 27
summary.visitation_model, 28

visitation_model, 29